

PCI 局部总线结构

第一章 总论

一、什么是 PCI 局部总线

所谓 PCI, 是 Peripheral Component Interconnect 的缩写。而 PCI 局部总线标准的制定主要目的是为了实现在一种将周边设备与处理器高速结合起来的总线结构, 以便适应用户对于数据率越来越高的要求。使用 PCI 总线结构的设备, 可以达到理论上峰值为 132Mbytes/s 的数据率, 虽然这个数字会因为总线的繁忙和设备自身的原因而和实际值有些出入, 但是达到 60Mbytes/s 的平均传送数据率还是有可能的。

二、为什么要使用 PCI 局部总线

在用户具有越来越多的数据传送需求的情况之下, 提高设备和主机之间的速度显然是大势所趋。而且使用 PCI 总线一个突出的有点就是 CPU 占用率极低。因为它和存储器之间的交互基本上通过 DMA 方式, 所以即使是低速设备比如声卡等等, 也开始淘汰 ISA 总线。

三、PCI 局部总线的应用范围

如二所述, 需要高数据率和低耗的场合, 都可以使用 PCI 总线设备。

四、主要的 PCI 接口芯片

纯粹的接口芯片主要有 PLX (90xx 系列), AMCC (59xx 系列), 还有 Altera 的 MegaCore 中的 PCI 功能系列以及 Xilinx 的 LogiCore 等等。

第二章 PCI 局部总线信号

为处理数据、寻址、接口控制、仲裁及系统功能, PCI 接口要求作为目标设备的设备至少有 47 条引脚, 作为总线主设备的设备至少有 49 条引脚, 图 2.1 为按功能分类的引脚, 必要的引脚在左边, 任选的引脚在右边。信号的方向说明是针对总线主设备/目标设备组合设备而言的。

2.1 信号类型定义

in	input(输入)是一种只用于输入的标准信号。	— —INPUT
out	output(图腾柱输出)是一种标准的有效驱动器。	— —OUT
t/s	Tri-state(三态)是一种双向、三态输入/输出引脚, 无效时是高阻态。	— —BIDIR
s/t/s	Sustained Tri-state(持续三态)是一种每次由且只由一个单元拥有并驱动的低有效双向、三态信号。驱动一个 s/t/s 信号到低的单元在释放该信号浮空之前必须将它驱动到高至少一个周期(以给总线预充电)。在前一个拥有者使其三态(高阻)之后一个周期内, 一个新的单元不能开始驱动 s/t/s 信号。为在下一个单元来驱动该信号之前维持无效状态, 要求有一个提拉电阻, 并且必须由中央资源(系统板)提供。	
o/d	Open Drain(漏极开路)允许多器件共享, 可作线或。	

2.2 引脚功能组

在 PCI 协议中, 中央资源用来表示由主系统所支持的总线支持功能, 特别是 PCI 所用的桥路和标准芯片集, 这些功能包括中央仲裁; 在复位期间驱动 REQ64#; 在系统配置操作时产生有效的 IDSEL 信号给每个设备; 反向解码; 提拉电阻或称保持器。PCI 控制信号常常要求提拉电阻以保证在无单元有效地驱动总线时它们保持稳定值。这些信号包括: FRAME#, TRDY#, IRDY#, DEVSEL#, STOP#, PERR#, SERR#, 在用到时也包括 LOCK#, REQ64#, ACK64#。点到点及共享的 32 位信号不要求提拉电阻, 总线放置保证它们稳定。

64 位数据通道的扩展信号 AD[63..32]、C/BE[7..4]#和 PAR64 在接上时, 也要求提拉电

阻，如果它们未连接上，器件必须自己处理这些浮空输入。

在图 2.1 中，PCI 引脚定义按功能组组织。在信号名之后的一个“#”标志说明该信号是低电平有效的，当无“#”标志时，信号是高电平有效的。每条引脚上的信号类型跟在信号名之后。

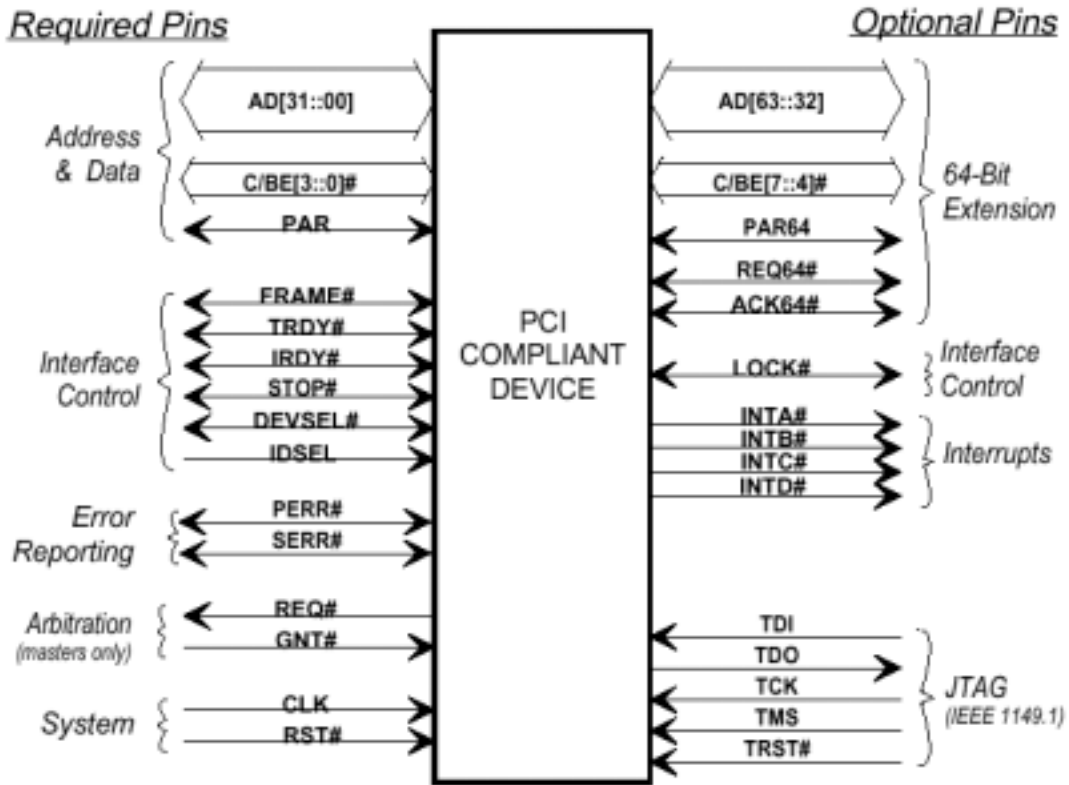


图 2.1 PCI 引脚列表

2.2.1 系统引脚

CLK in 系统时钟。

为所有 PCI 上的传输及总线仲裁提供时序。除 RST# 及四个中断引脚外，其它的 PCI 信号都在 CLK 信号的上升沿采样，所有别的时间参数都是基于这个上升沿而定义的。CLK 最小频率是直流(0Hz)，最高可达 33Mhz。

RST# in 异步复位。

用于使 PCI 确定的寄存器、配置寄存器、顺序发生器和信号置于一个固定的状态。无论何时，在 RST# 有效期间，所有 PCI 信号必须驱动到它们的起始状态。通常情况下，这意味这它们必须是三态(高阻态)。SERR# 被浮空。REQ# 和 GNT# 都必须是三态(在复位期间，它们不能是高也不能是低)。如果不能提供三态输出的话，SBO# 和 SDONE# 可以选择驱动到低。为防止 AD、C/BE# 和 PAR 信号在复位期间被浮空，中央设备可以在复位期间驱动这些线，但是只能驱动到逻辑低——它们不可驱动到高。REQ64# 在复位结束时有意义。这些基本都是由中央设备驱动的。

2.2.2 地址和数据引脚

一个 PCI 总线传输由一个地址段及相随的一个或多个数据段组成。

AD[31..00]	t/s	地址数据复用引脚。 FRAME#开始变为有效的那个时钟周期为地址段 AD[31..00]包含有一个物理地址。对于配置空间和存储器空间，这是一个双字地址，对于 I/O 空间，这是一个字节地址。在数据段，AD[7..0]包含最低字节数据，而 AD[31..24]包含最高字节数据。
C/BE[3..0]#	t/s	总线命令和字节允许复用引脚。在地址段，C[3..0]上定义了总线命令。在数据段期间，BE[3..0]#用作字节允许，表示哪些通道上的数据有意义，BE0#对应于最低字节而 BE3#对应于最高字节。
PAR	t/s	AD[31..00]和 C/BE[3..0]#上的数据偶校验。通常 PCI 单元都要求奇偶校验。PAR 与 AD[31..00]有相同的时序，但延迟一个时钟，在地址段后一个时钟，PAR 稳定并有效；对于数据段，在写传输中，PAR 在 IRDY#有效后一个时钟稳定并有效，而在读传输中，PAR 在 TRDY#有效后一个时钟稳定并有效。一旦 PAR 有效，它必须保持有效直到当前数据段完成后一个时钟。在地址段和写数据段，总线主设备驱动 PAR，在读数据段时，目标设备驱动 PAR。

2.2.3 接口控制引脚

FRAME#	s/t/s	周期构成(cycle Frame)。由当前总线主设备驱动，以说明一个操作的开始和延续。FRAME#有效，说明总线传输开始。当 FRAME#维持有效时，说明总线传输继续进行，当 FRAME#是无效状态时(高电平)，说明传送的最后一个字节正在进行。
IRDY#	s/t/s	启动者准备好(Initiator Ready)。说明传输的启动者完成当前数据传输的能力。在读操作中，IRDY#有效说明总线主设备已准备好接收数据。在写操作中，它说明 AD[31..00]上已有有效数据。在 IRDY#和 TRDY#都有效的时钟期间完成数据传输。在 IRDY#和 TRDY#都有效之前，需要插入等待状态。
TRDY#	s/t/s	目标设备准备就绪(Target Ready)。说明传输的目标设备完成当前数据传输的能力，在写操作中，TRDY#有效说明目标设备已准备好接收数据。在读操作中，它说明 AD[31..00]上已有有效数据。在 IRDY#和 TRDY#都有效的时钟期间完成数据传输。在 IRDY#和 TRDY#都有效之前，需插入等待状态。
STOP#	s/t/s	停止。说明当前的目标设备要求总线主设备停止当前传输。
LOCK#	s/t/s	锁定信号。 说明一种需要多个传输完成的原子级操作。当 LOCK#有效时，非独占传输可以对当前非锁定的地址进行。当只有一个总线主设备拥有 LOCK#时，不同的单元也可以使用 PCI 总线。PCI 总线传输开始并不能保证对 LOCK#的控制。对 LOCK#的控制必须由 LOCK#的拥有协议与 GNT#来完成。如果一个设备采用了可执行的存储器，它就必须使用 LOCK#，并且保证完成存储器中的锁定操作。连接系统存储器的主桥也必须使用 LOCK#。支持 LOCK#操作的目标设备，必须能至少提供 16 字节(顺序)的锁定。
IDSEL	in	初始化设备选择(Initialization Device Select)。在配置空间读写操作中，用作片选。
DEVSEL#	s/t/s	设备选择。 当有效驱动时，说明驱动它的设备已将其地址解码为当前操作的目标设

备。作为输入信号，DEVSEL#说明了总线上是否有目标设备被选中。

2.2.4 仲裁引脚(只对总线主设备)

REQ# t/s 申请。向仲裁器说明该单元想使用总线。这是一个点一点信号。每个总线主控都有自己的 REQ#。

GNT# t/s 允许。仲裁器向申请单元说明其对总线的操作已被允许。这是一个点一点信号，每个总线主设备都有自己的 GNT#。

2.2.5 错误反馈引脚(所有设备都要求有错误反馈引脚)

PERR# s/t/s 奇偶校验错误(Parity Error)。该引脚只用于反馈在除特殊周期外的其它传送过程中的数据奇偶校验错误。PERR#维持三态，并在检测到传送数据中的奇偶错误后，在数据结束后两个时钟，由接收数据的单元驱动 PERR#有效，并至少持续一个时钟周期。和所有的 s/t/s 信号一样，在被释放到三态之前，PERR#必须驱动到高电平一个时钟周期，对数据奇偶错误信息被丢失或错误反馈被延迟没有特殊条件。只有发出 DEVSEL#的单元才能发出 PERR#。

SERR# o/d 系统错误。用于反馈地址奇偶错误、特殊周期命令中的数据奇偶错误和将引起重大事故的其它灾难性的系统错误。如果一个单元不想产生不可屏蔽中断(NMI) 则可用 SERR#反馈给系统。SERR#是单纯的漏极开路信号，由反馈错误的单元驱动，在一个 PCI 时钟脉冲内有效。SERR#与时钟同步，并满足所有总线信号的建立与保持时间。然而，SERR#释放到无效状态，则是由系统设计者所提供的一个小提拉电阻实现的(同 s/t/s 信号)，不是由特定单元或中
央资源来实现。该提拉电阻完成恢复 SERR#可能花费 2—3 个时钟周期。支持 SERR#的单元在采样到 SERR#有效时。就向操作系统报告系统错误。

2.2.6 中断引脚

中断引脚是“电平触发”，低有效，用漏极开路输出驱动器驱动，与时钟异步的。

PCI 为每一个单一功能设备定义一根中断线。

INTA# o/d 中断 A，用于单一功能设备请求一次中断。

INTB# o/d 中断 B，用于多功能设备请求一次中断。

INTC# o/d 中断 C，用于多功能设备请求一次中断。

INTD# o/d 中断 D，用于多功能设备请求一次中断。

多功能设备的任何一种功能都能连到任何一条中断线上。中断引脚寄存器决定该功能用哪一条中断线去请求中断。如果一个设备只用了一条中断线，则这条中断线就被称为 INTA#，如果该设备用了两条中断线，那么它们就被称为 INTA#和 INTB#，依次类推。对于多功能设备，可以是所有功能用一条中断线，也可以是每种功能有自己的一条中断线(最多 4 种功能)，还可以是上述两种情况的综合。一个单功能设备不能用一条以上的中断线去请求中断。

系统商可以将 PCI 联接器上任何组合方式将中断线连接到中断控制器上，可以是线或方式，也可以是程序控制的电子开关切换方式，或是别的任何组合方法。这意味着设备驱动器对共用中断不能作任何假定。所有设备驱动器必须能与任何别的逻辑设备共用中断，包括同一多功能封装中不同设备之间的情况。

2.2.7 高速缓存(cache)支持引脚(可选用)

一个能高速缓存的 PCI 存储器必须利用这两条高速缓存支持引脚作为输入，以支持写通(write-through)和回写(write-back)。如果可高速缓存的存储器是位于 PCI 上，则连接回写高速缓存到 PCI 的桥路必须利用两条引脚，且作为输出。连接写通高速缓存的桥路可以只使用一条引脚 SDONE。

SBO# in/out 监视补偿(Snoop Backoff)。当其有效时，说明对某条变化线的一次命中。

当 SBO#无效而 SDONE 有效时,说明了一次“干净”的监视结果。

SDONE in/out 监视进行(Snoop Done)。表明对当前操作的监视状态。当其无效时,说明监视结果仍未定;当有效时,说明监视已有结果。

2.2.864 位总线扩充引脚(可选用)

AD[63..32]	t/s	地址数据复用引脚提供 32 个附加位。在一个地址段(用 DAC 指令且 REQ64#已有效),传送 64 位地址的高 32 位;如无高 32 位地址,这些引脚就被保留,其上数据是稳定的,但值是不确定的。在数据段期间,当 REQ64#和 ACK64#都有效时,传送 64 位数据中的高 32 位。
C/BE[7..4]#	t/s	总线命令和字节允许复用引脚。在一个地址段(用 DAC 指令且 REQ64#已有效),在 C/BE[7..4]上传送有效总线命令;否则,这些引脚被保留且其值不定。在数据段期间,当 REQ64#和 ACK64#都有效时,C/BE[7..4]#是字节允许,说明那些字节通道上含有有数据。C/BE4#相应于第四字节而 C/BE7#相应于第七字节。
REQ64#	/t/s	请求 64 位传输。当其被当前总线主设备有效地驱动时,说明该总线主设备想作 64 位的传送。REQ64#与 FRAME#有相同的时序。在复位结束后,若 REQ64#有效,该设备就已连到 64 位通道上,否则,就没有。
ACK64#	s/t/s	应答 64 位传送。在当前操作所寻址的目标设备有效驱动该信号时,说明该目标设备能够进行 64 位传送,ACK64#和 DEVSEL#有相同的时序。
PAR64	t/s	高双字偶校验。是 AD[63..32]和 C/BE[7..4]#的偶校验位。当 REQ64#有效且 C/BE[7..4]#上有 DAC 命令时,第一个地址段后一个时钟周期 PAR64 有效。DAC 命令的第二地址段后的哪一个时钟周期,PAR64 也有效。对于数据段,当 REQ64#及 ACK64#均有效时,读操作中,TRDY#有效后,PAR64 是稳定且有效的,写操作中,IRDY#有效后,PAR64 是稳定且有效的,一旦 PAR64 有效,必须保持有效直到数据段完成后一个时钟(PAR64 时序与 AD[63..32]相同但延迟一个时钟)。在地址段和写数据段,总线主设备驱动 PAR64,在读数据段,则由目标设备驱动 PAR64。

在总线主设备和目标设备之间,64 位传送是动态协调的(每个地址段一次)。而且,只有存储器命令支持 64 位传送。总线主设备使 REQ64#有效,目标设备则通过使 ACK64#加以应答。REQ64#和 ACK64#是外部上拉的,以保证 64 位和 32 位单元混用。一旦 64 位传送建立,就一直保持到这次传送结束。

2.2.9 JTAG/边缘扫描引脚(任选)

IEEE 标准 1149.1,测试存取口及边缘扫描结构(Test Access Port and Boundary scan Architecture)。在一个设备中包含测试存取口(TAP),使得在测试该设备或装有该设备的板时可以使用边缘扫描。TAP 由 4 至 5 条引脚组成,它们用于和 PCI 设备中的 TAP 控制器作串行接口。

TCK in 测试时钟。在 TAP 操作期间记录状态信息和测试设备的输入输出数据。

TDI in 测试数据输入。用来在 TAP 操作期间将测试数据和测试指令串行移入设备中。

TDO out 测试输出。用来在 TAP 操作期间将测试数据和测试指令串行移出设备中。

TMS in 测试模式选择。用来控制设备中的 TAP 控制器的状态。

TRST# in 测试复位。这个可选引脚给 TAP 控制器提供了一个异步初始化。

PCI 规范支持带有包含 1149.1 边缘扫描信号的联接器的扩展板。扩展板上的设备要连接到主板上的 1149.1 环。为了不中断设备间的串行链码,不支持 IEEE1149.1 标准接口的

扩展板在硬件上要将其 TDI 引脚接到其 TDO 引脚上。

第三章 PCI 设备配置空间

一、简介

除了主/PCI 桥之外，其他的 PCI 设备都应该实现 PCI 设备配置空间。该配置空间包括一系列的 PCI 配置寄存器。配置空间实现的具体位置可以在 PCI 配置空间当中（这是最有可能的情况），或者 IO 空间（但是一般来说一个 PCI 设备占用的 IO 空间为 256 个字节，配置空间的大小就是 64 个双字），也可以直接在我们申请的 Memory 空间中来实现。

一般情况下，我们不是自己完全实现一个 PCI 接口芯片的功能，而是使用厂商提供的芯片，这些芯片的某些配置寄存器是可以设置，而另外一些是不可设置的。我们本章将对一些会直接影响设备特性的配置寄存器进行介绍，而关于对于这些寄存器的读写操作将会在第六章中进行介绍。

会直接影响 PCI 设备特性的配置寄存器集中在 PCI 配置空间的前 16 个双字里。该区域成为 PCI 配置头。目前 PCI 2.2 规范定义了三种配置头的格式，分别是类型 0、1 和 2。

类型 0：除类型 1 和类型 2 以外所有的 PCI 设备；

类型 1：PCI—PCI 桥设备，用于将两条 PCI 总线进行连接；

类型 2：PCI—CardBus（主要用于笔记本的插卡式总线）桥，在 PC Card 规范中进行定义。

我们这里只对类型 0 的配置头进行介绍。

二、必须实现的配置寄存器

以下描述的配置寄存器在所有的 PCI 设备中都应当实现，包括桥设备。

一般情况下，操作系统使用以下配置寄存器的内容来决定为该 PCI 设备加载何种驱动程序。

供应商 ID；

设备 ID；

版本号；

类别代码；

子系统供应商 ID；

子系统 ID；

1. 供应商 ID 配置寄存器：

该 16 位寄存器代表 PCI 设备的制造商。本寄存器只读，其中烧入的内容是一个由 PCI SIG 分配给该制造商的一个编号。如果系统读取该寄存器返回值不是

FFFFh, 那么就证明在当前查找的 PCI 插槽上存在一个 PCI 设备, 否则就视为不存在, 因此 FFFFh 的值不可以被用来标示设备。

2. 设备 ID 配置寄存器:

这一 16 位值由设备制造商自行定义, 表示设备的用途。它与供应商 ID 配置寄存器一道来向系统提供一个确定设备所需要驱动程序的途径。

3. 子系统供应商 ID 寄存器和子系统 ID 寄存器

同样的, 子系统供应商 ID 也是由 PCI SIG 进行管理, 厂商从它那里获得一个唯一的标识。而子系统 ID 则是由厂商自定的。

这一对寄存器的用途是显而易见的, 如果我们使用了相同的 PCI 接口芯片, 且供应商将供应商 ID 和设备 ID 硬连线, 那么两个不同功能的设备必须有一种被区分开来的方法。利用这样两个寄存器, 操作系统可以区分使用同样接口芯片的不同设备。

4. 版本 ID 寄存器

这个寄存器共 8 位, 表示设备的版本号。

5. 类别代码寄存器

这是一个 24 位的只读寄存器, 共分为三个独立的 8 位单元: 基类型字节、子类型字节和编程接口字节。它们分别代表设备的基本功能 (例如大容量存储控制器)、细化的设备子类型 (例如 IDE 大容量存储控制器) 以及在一些情况下寄存器指定的编程接口 (例如 IDE 寄存器组的指定格式)。

这 24 位当中最高 8 位代表基类型, 中间 8 位代表子类型, 最低 8 位代表编程接口。

当然, 对于许多的基类型和子类型的组合而言, 其编程接口一般都硬连线成 0 (即没有意义可言); 对于一些特定的类型, 比如 VGA 兼容设备和 IDE 控制器, 编程接口字节还是有意义的。

类别代码寄存器的作用体现在当查找新硬件的时候, 就会根据类别代码来自动判断该设备属于何种类型。然后, Windows 或其他的支持 PnP 的操作系统会自动根据这样的类别码建议使用一种通用设备驱动程序 (如果可能的话), 最少在这样的驱动程序支持下设备可以完成本类别设备的基本功能, 至于其他功能, 要靠自行设计的硬件来实现了。当然, 我们可以随便给出类别代码, 只要能够保证设备可以使用就行。我可以将设备由多媒体视频设备改成音频设备, 对于系统的功能没有丝毫的影响 (当然, 我还没有尝试过将设备设置成 IDE 控制器同时使用编程接口所带来的后果)。所以, 请记住, PCI 设备是按照驱动程序的指引在工作, 而不是类别寄存器, 这一寄存器存在的目的是为了设备更加规范化而已。

由于类别代码实在太多, 有兴趣的同学可以自行参阅 PCI 规范中的相应部分, 这里就不耗费篇幅了。

6. 命令寄存器

该寄存器提供了控制设备对于 PCI 访问的响应以及执行的能力。这是一个 16 位的寄存器，而只有低端的 10 位有意义，高 6 位目前保留。下面逐一简要介绍各位的意义。

- (0) IO 空间。为 1 表示 PCI 总线上出现的地址将被译码成 IO 地址，0 则禁止。缺省为 0。
- (1) 存储器空间。为 1 表示 PCI 总线上出现的地址被译码成 PCI 设备上的存储器地址，0 表示禁止（显见，0 位和 1 位不可同时为 1，同时，在系统上电的过程中，这两位都要被初始化成 0，以免发生尚未配置成功设备时就出现设备的方位情况）。缺省为 0。
- (2) 总线主设备。为 1 表示该设备可以作为主设备（如果该设备具有成为主设备的能力），0 表示禁止。缺省为 0。
- (3) 特殊周期。为 1 表示设备将会监视 PCI 总线的特殊周期（设备具有监视这种周期的能力）。0 将会使设备忽略特殊周期。缺省为 0。
- (4) 存储器写和使失效使能。为 1 表示设备将会产生存储器写和使失效命令。当设置为 0 时，设备使用存储器写命令来替代。
- (5) VGA 调色板监测。为 1 将会使 VGA 兼容设备检测所有对于 VGA 颜色调色板寄存器的写操作（本位只用于显示设备）。对于非 VGA 图形设备，复位将会将该位置 1；而 VGA 兼容的显示控制器复位将会将该位置 0。
- (6) 奇偶校验错响应。为 1 时设备将会通过将 PERR#信号置 0 来向系统报告奇偶校验错。为 0 的时候，奇偶校验错将不会在 PERR#信号上显示出来。不过，无论如何，状态寄存器的奇偶校验错位还是需要置的。缺省为 0。
- (7) 步进控制。该位控制设备是否可以使用地址/数据步进。不使用步进功能的设备必须将该为硬连线为 0。总是使用的设备可以将该位置为 1；能够同时处于这样两种状态的设备应该将该位设定成可以读写的，同时在复位过后将该位置为 1。（步进是实现猝发操作的关键，使用 PCI 于数据率大的应用环境时应选择步进）。
- (8) SERR#使能。为 1 表示设备能够驱动 SERR#信号线，0 表示不驱动。
- (9) 快速接续使能。如果总线主设备可以与不同的目标设备设备在相邻的两次处理中执行快速接续传输，那么这一位就用于使能或者禁止这样的功能。当然，这样一项功能是需要主设备和从设备都支持才可以得到执行的，所以，如果主设备所在的 PCI 总线上所有目标设备设备都支持快速接续能力，那么我们就应当将这一位置为 1，因为我们可以不必担心两次处理是否针对同一个目标设备设备。缺省为 0。

6. 状态寄存器

该寄存器用于记录一个 PCI 设备目前的状态。表示具有某种功能的方法就是将相应的功能位在硬件中实现。对于这个寄存器可以进行读取操作。而写入的时候则是对于置为 1 的位的写入将会将该位置为 0，同时写入操作不可以将目前为 0 的位置为 1。

下面描述这些状态寄存器位

3:0 只读 保留

硬连线为 0

- 4 只读 能力列表
如果该位为 1，那么表示该设备实现了新能力列表。
- 5 只读 66MHz 支持
1 表示设备工作于 66MHz
0 表示设备工作于 33MHz
- 6 只读 保留
- 7 只读 快速接续能力
1 表示支持快速接续能力
0 表示不支持快速接续能力
- 8 读写 主设备奇偶校验错哦
该位只有总线主设备实现，并且仅当符合下列条件的时候设置：
 - a. 总线主设备是处理的发起者
 - b. 通过自行设置 PERR#（在读操作的时候）或者依靠从设备设置（在写操作的时候）。
 - c. 命令寄存器中奇偶校验响应位已经置 1。
- 10:9 只读 设备选择定时
这些位是为目标设备设备（进行配置访问除外）定义了 DEVSEL#最慢

定时

- 00b=快速
- 01b=中速
- 10b=慢速
- 11b=保留

- 11 读写 发出目标设备设备终止
一旦目标设备设备在处理中终止处理，那么设置这个位。
- 12 读写 收到目标设备设备终止
总线主设备收到目标设备设备终止信号。
- 13 读写 收到主设备终止
- 14 读写 发出系统错误
- 15 读写 发现奇偶校验错

7. 首部类型寄存器

这个单字节寄存器的 6:0 位定义了首部寄存器第 4 到第 15 个双字的格式。位 7 定义了该设备是否属于多功能设备（位 7=1）。

三、其他的配置寄存器

下面将要描述的配置寄存器都属于首部类型为 0 的 PCI 设备。这些寄存器有可能是可选的，也有可能是必须实现的，这都依赖于设备的类型。

1. Cache 行容量寄存器

对于使用存储器写和使失效命令的主设备而言，这个寄存器是必须实现的。对于支持 Cache 行换行寻址的存储器而言，也是必须得。

这个可读写配置寄存器指明系统 Cache 行容量的双字递增数目。总线主设备为了保证在 Cache 行边界开始一次处理，必需要知道 Cache 行的容量。如果这个寄存器为 0 时，总线主设备可以不使用存储器写与使失效命令，那么这个时候设

备可以使用存储器写命令。设备可以限定其所支持的 Cache 行容量的范围，如果软件将一个并不支持的数据值写入，将会被看作是写了一个 0。

2. 延迟定时器

对于需要执行猝发操作的总线主设备而言，这个寄存器是必须的。延迟定时器确定了一个时间段，它表明当总线主设备启动一个处理过程，那么他将在这个延迟定时器计时超时以前一直保有总线的控制权。在启动处理过程之后，延迟定时器将会在每个 PCI 时钟的上升沿自动减 1。

主设备在定时器计时期间持续进行处理知道以下任一情况发生：

- a. 主设备已经完成了该次处理
- b. 目标设备发出 STOP#信号，提前终止处理
- c. 计时器超时，同时已经有其他的 PCI 主设备抢占了总线

如果总线主设备进行处理的时候需要执行多于两个数据周期的猝发，那么延迟定时器就一定要实现。如果主设备从不执行超过两个数据周期的猝发操作，那么它可以将该寄存器硬连线为 0。如果时间片为 0 表示主设备不具有时间片，即当第一个数据周期过去之后，主设备立刻失去 GNT#，将之让给其他的 PCI 主设备。只作目标设备使用的 PCI 设备不需要这个寄存器。

可编程延迟定时器中，低 3 位硬连线为 0，高 5 位可以任一设置。这样用户可以 8 个 PCI 周期为单位确定其需要的时间片。如果该寄存器可以编程，那么复位信号将会清除这个寄存器。

3. 基地址寄存器

基地址寄存器在 PCI 设备功能实现上是相当重要的。基地址寄存器在配置空间中的位置从第四个双字一直到第九个。它们被用来存放 PCI 设备映射的内存地址或者使用的 IO 空间的首地址。

在这里，PCI 规范设计者提供了一种机制，使得 IO 和 Memory 分开，即在基地址寄存器的最低位上，如果是 0，表明这个基地址寄存器指向的是一个存储器空间，而如果是 1，那么就是指向一个 IO 空间。

在我们要为存储器或者 IO 空间的分配空间的时候，我们还需要了解这个地址空间的大小。在这里，可以通过向一个基地址寄存器写全 1，然后读回这个寄存器的值，没有被改变的位（一般来说都是低位是只读的）的长度就是我们需要向系统申请的地址空间大小。

经过以上的步骤，我们就可以通过驱动程序向系统注册这样一个设备，然后将获得的系统认可的地址范围写回到基地址寄存器，将来可以通过对基地址寄存器的读取来判断映射的 IO 和存储器地址。

4. 中断引脚寄存器

对于 PCI 设备来说，它总共有四条可以选择的中断请求引脚，他们分别是 INTA#、INTB#、INTC#和 INTD#。这个寄存器的值从 01h 到 04h 分别表示设备上边的四个引脚。返回值 0 表示没有使用中断，而其他值都是保留的。

5. 中断线寄存器

可读写的中断线寄存器用来表示 PCI 设备的中断是连接到主机的中断控制器的哪一个引脚上的。在 PC 机中，这个值是从 00h 到 0fh 的。而 ffh 表示这个设备

具有未知的中断线。其他的值都被保留。一般来说，在 RST#信号有效的时候，我们应该把这个寄存器的值初始化成 ffh。

我们知道了这个寄存器的内容，那么在编写驱动的时候，就可以向系统注册这个中断的处理函数，以便将来在设备发生中断地时候从系统手里接管设备。

另外要注意的是，PCI 属于可共享中断设备，那么在中断处理函数中一定要判断是否是自己控制的设备发生的中断，如果是，那么进行中断处理，如果不是，那么将这个中断传递下去。

6. Min_Gnt 寄存器

这个寄存器是主设备专用的，而且同时也是可选的。其中存储的值的意义是主设备在启动一次数据传送任务之后需要保持多久的总线所有权，进行计数的单位是四分之一毫秒，0 表示对这一指标没有额外的要求。

7. Max_Lat 寄存器

这个寄存器是主设备专用的，而且同时也是可选的。这个寄存器中存储的值表示设备要求以多快的速度访问 PCI 总线，计数单位也是四分之一毫秒。

其他的配置空间寄存器请大家参阅 PCI 2.2 规范。

第四章 PCI 局部总线仲裁协议

一、仲裁器

在任意时刻，一个或者多个 PCI 总线主设备可能要求使用 PCI 总线，执行数据传送工作到另外一个 PCI 设备。每个发出请求的主设备使其 REQ#输出有效，通知总线仲裁器，它正在请求使用 PCI 总线。每个主设备都是通过一对独立的 REQ#/GNT#信号连接到仲裁器。虽然仲裁器是一个独立器件，但是它一般都集成到 PCI 芯片组当中，特别是南桥/北桥芯片组当中。

二、仲裁算法

Pci 规范并没有定义当多个主设备同时请求总线使用权的时候，PCI 总线仲裁器应该如何来确定竞争优胜者的机理。仲裁器可以使用任何一种设计者认为正确的机理，例如基于固定或者循环优先级的或者将二者结合起来使用。规范中要求仲裁器的算法要足够公平以便避免死锁。

而公平就意味着：必须授权每个潜在的总线主设备，独立于其他请求地访问总线。公平是这样一种策略，它可以保证高优先级的主设备连续访问总线的时候，不会一直强占总线，不让较低优先级的主设备访问。但是这并不意味着要求所有的设备平均的访问总线。这里就需要用到总线主设备的最大延迟寄存器 (Max_Lat) 以便确定分配到每个总线主设备的优先级。总线主设备的设计者可以硬连线该寄存器，表明主设备要求以多快的频率访问总线以便达到应有的性能。

为了使某个总线主设备获得 PCI 总线的访问权，PCI 仲裁器将该设备的 GNT#置有效，这样将主设备可以在此次处理中占有总线。如果主设备发出请求，然后仲裁器授予它总线，但是在随后的 16 个 PCI 时钟周期内没有启动数据传送处理过程 (即 FRAME#信号无效)，总线仲裁器可以认为主设备的功能出现异常，在这样的情况之下，仲裁器采取的行动是由相应的系统来决定的。

三、公平的仲裁例子

系统将会把 PCI 总线上的全部总线设备分成两个部分：

1. 为了达到较高性能，要求快速访问总线的总线主设备。例如显示适配器，ATM 网络接口或者 FDDI 网络接口。
2. 不要求快速访问总线以便达到较高性能的设备，例如 SCSI 适配器。

在例子中，仲裁器将 REQ#/GNT#分成两组，其中一组具有更大的优先权。假设总线主设备 A 与 B 在要求快速访问的组中，主设备 X、Y 和 Z 在另外一组当中，对于仲裁器的编程或者设计，将每个组内的设备之间都设计成循环优先权，而两组之间也是循环优先权。

假设下列情况：

- 1、在第一组当中下一个接受总线的设备是主设备 A
- 2、在第二组当中下一个接受总线的设备是主设备 X
- 3、在第一组当中下一个接受总线的设备是某个主设备
- 4、所有主设备都置 REQ#有效以便执行多次处理过程

那么，获得总线的主设备的顺序依次是：

1. 主设备 A
2. 主设备 B
3. 主设备 X
4. 主设备 A
5. 主设备 B
6. 主设备 Y
7. 主设备 A
8. 主设备 B
9. 主设备 X
10. 主设备 A
11. 主设备 B
12. 主设备 X 依次类推

而且可以看出第一组当中的主设备被授予更多的访问 PCI 总线的次数。

四、主设备要求进行多次处理

如果主设备希望在当前处理结束之后马上开始下一次处理，那么在它使 FRAME#信号有效的同时应该保持使得 REQ#信号有效。通过这样一种方式，主设备通知总线仲裁器它希望完成当前处理之后继续拥有 PCI 总线。而这一请求的满足与否取决于仲裁器对于总线申请的仲裁结果。如果当前主设备在当前处理结束之后没有能够保持住总线控制权，那么主设备应该保持 REQ#有效，知道它又取得总线控制权为止。

在任意时刻，只有一个主设备可以拥有 PCI 总线，这就意味着在任何一个 PCI 总线周期之内，总线仲裁器只会使一条 GNT#信号线有效。

五、隐式总线仲裁

当前主设备正在进行数据传送的时候，PCI 机理允许总线仲裁发生。如果总线仲裁器决定将 PCI 总线的控制权赋予其他的主设备，而非当前的主设备。那么当前主设备获得的 GNT#信号将会失效，同时仲裁器决定的下次处理的主设备将会得到 GNT#有效。当然，当前主设备的数据传送不会被这一仲裁过程中断，知道当前主设备完成数据传送并使得总线处于空闲状态，GNT#有效的那个主设备才会获得总线控制权。这一方式使得在总线仲裁周期的时间里没有中断数据传送过程，相对于数据传送过程

这样的仲裁是不可见的，所以这样的仲裁过程被称为隐式仲裁。

六、总线停靠

总线主设备如果目前有对于总线的需求，那么就应该使其 REQ#输出有效，那么，从反面来看，主设备就不应该在不需要的时候将 REQ#置有效，而使得总线一直停靠在该主设备上。作为系统设计者，如果实现了总线停

七、请求/确认时序

当总线仲裁器确定主设备可以使用总线的时候，它会令 GNT#线有效。总线仲裁器可以在任何一个时钟周期令 GNT#失效。作为总线主设备，必须在其发起一次总线数据传送操作前获得 GNT#有效的指示，否则处理将不被接受。一旦仲裁器有效 GNT#，那么 GNT#可以在以下的情况下无效：

1. 如果 GNT#无效同时 FRAME#有效，那么传送有效而且连续。仲裁器收回 GNT#信号表明，主设备在完成当前的处理之后将不会重新获得总线；当前处理在进行过程当中，主设备保持 FRAME#有效，在它准备结束这样一次操作的时候，也就是已经达到最后需要传送的数据的时候，撤销 FRAME#信号。
2. 如果总线并不是处于空闲状态，那么一个主设备的 GNT#无效，另外一个主设备的 GNT#将会马上有效。空闲状态定义为 FRAME#和 IRDY#同时无效的时钟周期。如果总线空闲，那么正在失去 GNT#的主设备可能已经使用步进法驱动总线。伴随另外一个主设备的 GNT#有效，此主设备的 GNT#同时无效，肯定会导致了 AD 总线上的冲突（因为可能上一个总线主设备的数据还在总线上没有去除驱动）。另外一个主设备又可能会马上开始自己的数据传送过程。通过在一个主设备失去总线一个周期之后再授予其他主设备 GNT#信号的方法，可以防止这样一个问题的出现。我们下面将会列出 FRAME#和 IRDY#的组合代表的总线状态。

FRAME#	IRDY#	描述
无效	无效	总线空闲
无效	有效	数据传送在进行当中而且主设备还没有准备结束
有效	无效	主设备准备完成最后的数据传送
有效	有效	出力在进行中，同时主设备准备完成当前数据传送

3. 在数据传送过程的最后一个时钟周期内 GNT#可以无效（这个时候 FRAME#也无效），这可以反映出 REQ#处于无效状态。

八、其他需要注意的事项

当 RST#有效的时候，所有的主设备都应该使自己的 REQ#输出为高阻态，同时忽略 GNT#信号的变化。

如果在总线仲裁器发出 GNT#到达总设备开始的 16 个时钟周期内系统总线空闲，同时主设备没有使 FRAME#信号有效，那么仲裁器就可以假定这个 PCI 设备是已经损坏的。这样一来，以后来自这个设备的 REQ#可以被忽略，同时仲裁器会向系统报告这个设备已经损坏这样的事实。

第五章 PCI 局部总线命令与操作

5. 1. 命令编码

总线命令对目标设备说明当前总线主设备正在请求的传输类型。总线命令在地址段期间 C/BE[3..0]#上，其编码如表 3.1 所示

PCI 总线命令

C/BE[3..0]#	总线命令
0000	中断应答(interrupt Acknowledge)命令是一个寻址系统中断控制器的隐性读。在地址段，地址位逻辑上无关紧要，字节允许说明返回矢量的大小。
0001	特殊周期(Special Cycle)命令提供一种简单的、广播式的信息传播机制。
0010	I/O 读命令用于从一个映射于 I/O 地址空间的单元中读取数据。AD[31..00]提供某个字节的地址，全部 32 位都必须解码。字节允许说明传送的大小且必须与字节地址段一致。
0011	I/O 写命令用于写数据到一个映射于 I/O 地址空间的单元中去。全部 32 位都必须解码。字节允许说明传送的大小且必须与字节地址段一致。
0100 0101 1000 1001	保留命令。这些命令编码留作将来使用。PCI 目标设备不能将保留编码与其它编码混同。目标设备不能对保留编码作出应答。如果接口上用了保留编码，操作时总线主设备将用总线主设备失败终止此操作。
0110	存储器读命令用于从一个映射于存储器地址空间的单元中读取数据。只要目标设备保证预取没有副作用，便可以用该命令预取。进而，目标设备必须保证在 PCI 传送完成之后，保留在暂时缓冲区中的数据的一致性(包括顺序)。在任何同步事件通过该通道之前，这种缓存必须初始化(清空)。
0111	存储器写命令用于写数据到一个映射于存储器地址空间的单元中去。当目标设备返回“准备好”时，它表明能正确接收对象数据。实现这一点，或者以一种完全同步的方式实现这条命令。或者保证在任何同步事件(更新 I/O 状态寄存器或存储器标志)通过这种操作通道之前，任何软件透明中继缓存器都被刷新。这表明总线主设备在执行这条命令之后可立即处理同步事件。
1010	配置读命令用于配置空间的读操作。当 IDSEL 信号有效且 AD[1..0]是 00 时，就选中了一个单元。在配置周期的地址段期间，AD[7..2]寻址每个设备配置空间 64 个双字寄存器之一，字节允许寻址每个双字中的字节，且 AD[31..11]上的逻辑是不必关心的，AD[10..8]说明寻址多功能单元的哪个设备。
1011	配置写命令用于传送数据到配置空间。当 IDSEL 信号有效且 AD[1..0]是 00 时，就选中了一个单元。在配置周期的地址段期间，AD[7..2]寻址每个设备配置空间 64 个双字寄存器之一，字节允许寻址每个双字中的字节，且 AD[31..11]上的逻辑是不必关心的，AD[10..8]说明寻址多功能单元的哪个设备。
1100	存储器重复读(memory Read multiple)命令除说明总线主设备在解除连接前打算读取一条以上高速缓存线的数据外，其它与存储器读命令相同。只要 FRAME#有效，存储控制器就将维持其流水线存储器请求。该命令为大块数据传送而设计的，此时，如果一个软件透明缓冲器能用作暂时存储，超前顺序读取额外的一条高速缓存线上的数据可以提高存储器系统的性能。
1101	双地址周期(DAC)命令用于传送 64 位地址给支持 64 位寻址的设备。只支持 32 位寻址的目标设备把该命令当作保留对待而对当前传送不作任何反应。
1110	高速缓存线存储器读(memory Read Line)命令说明总线主设备打算完成两个以上的 32 位数据段外，其余与存储器读类似。该命令为大块数据传送而设计。作为对传送申请的响应，一次读取一条行缓存线范围内的所有数据，而不是单个存储器读周期，从而提高存储器系统的性能。和存储器读命令一样，在任何同步事件提供这种操作路径之前，预取缓存器必须初始化为空。
1111	高速缓存写(Memory Write and Invalidate)命令除保证最小传输为一个完整的高速缓存线数据外，其余与存储器写命令相同。如果总线主设备想在一次 PCI 传送中，写完被寻址的高速缓存线上的全部字节，就可用该命令说明。

如果总线主设备想将下一条高速缓存线也全部传送完，那么传送就可以跨越高速缓存线边界。该命令要求在总线主设备中有一个配置寄存器说明高速缓存线的范围，该命令可以通过不要求有效回写周期而将一条“脏”线初始化到回写高速缓存，从而缩短操作时间而使存储器性能优化。

5. 2. 命令使用规则

所有 PCI 设备对配置读/写命令而言，都是目标设备，都必须作出应答，对别的命令则有选择余地。I/O 读/写命令是可选的，命令执行规则保证 I/O 读写命令的执行。有重定位功能或寄存器的目标设备要求，能通过配置寄存器而映射到存储器空间，并响应基本的存储器读/写命令，这就为没有 I/O 空间的设备提供了一种选择。当这种映射实现时，无论设备映射到 I/O 空间还是存储器空间，命令执行规则都由系统设计者来保证。对一个被映射设备的存储器读和写都构成“存储器映射 I/O 口”。

总线主设备可以根据需要使用任选命令。目标设备也可根据需要而选用指令，但如果它选用了基本存储器命令，它就必须支持所有存储器命令，包括高速缓存写命令，高速缓存线存储器读命令和存储器重复读命令。如果不能全部使用，这些性能已优化的命令必须转化为基本存储器命令。例如，一个目标设备可以不用高速缓存线存储器读命令，但它必须接收这种申请并把它当成存储器读命令。同样，一个目标设备可以不用高速缓存写命令，但它必须接收这种申请并把它当成存储器写命令。

对于进/出系统存储器的块数据传输，对能支持高速缓存写和高速缓存线存储器读的总线主设备，建议采样这两条命令。如果由于某些原因，总线主设备不能使用性能已优化的命令，那么就用存储器读和存储器写命令。

对于使用存储器读命令的总线主设备，对所有命令都可作任意长度的操作，但最优方法如下所列。只有高速缓存写命令要求实现高速缓存线范围寄存器，建议存储器读命令也用它。所有情况下，桥路保证任何隐含数据的正确性。

使用高速缓存线范围寄存器时的最优方法：

- 存储器读命令： 当猝发传送少于半条高速缓存线数据时使用。
- 高速缓存线存储器读命令： 当猝发传送半条到三条高速缓存线数据时使用。
- 存储器重复读命令： 当猝发传送三条以上高速缓存线数据时使用。

未用高速缓存线范围寄存器时的最优方法：

- 存储器读命令： 当猝发传送两个或更少的数据时使用。
- 高速缓存线存储器读命令： 当猝发传送 3 到 12 个数据时使用。
- 存储器重复读命令： 做长猝发时使用。

5. 3. PCI 协议的主要内容

PCI 总线的基本传送机制是猝发传送。一个猝发传送由一个地址段和一个或多个数据段组成。它要求目标设备和总线主设备都必须能理解隐含寻址，PCI 支持对存储器和 I/O 地址空间的猝发。主桥路在无副作用的情况下可以将多个存储器写操作合并为一个猝发传送。设备通过设置基本地址寄存器中的预取位来表明没有副作用。桥路可以通过初始化期间配置软件提供的地址范围来判断哪里允许合并、哪里不允许。当接下来的是一个不可预取的读或写时，合并数据到缓存器必须停止(且缓存器被刷新)。如果在可预取范围，跟在上述两事件之后的写传送可以与后续的写合并，但不合并前面的数据。

主桥路通常可以将处理器产生的连续双字按原顺序组合到一个猝发传送中，例如，当处理器写的顺序是双字 0、双字 2、双字 3 时，桥路可以产生一次猝发传送。该 PCI 猝发顺序可以是双字 0、双字 1(无字节允许)、双字 2、双字 3。组合任何时候都使得后一个双

字的地址比前一个双字的地址更有意义。当读猝发对被寻址的目标设备无副作用时，桥路可以将处理器的单个存储器读请求转换为读猝发。

因为从处理器中发出的 I/O 操作不能被组合，所以这种操作将按正常情况只有一个数据段。目前尚没有已知处理器和总线主设备能对 I/O 空间产生猝发操作。但当 I/O 猝发成为现实时，目标设备和总线主设备都必须能理解隐含寻址。不能进行多数据段处理的 PCI 设备必须在第一个数据段后脱离操作。所有 I/O 操作必须正确出现在 PCI 上犹如处理器产生它那样(如果 I/O 操作中有目标设备被选中，但字节允许却说明传送字节数大于该目标设备所支持的字节数，则目标设备用目标设备失败终止传送)。

5. 4. 操作规则:

5. 4. 1. 何时信号稳定:

在 PCI 总线信号中，除了 RST#、INTA#、INTB#、INTC#和 INTD#外，其余信号的都在时钟信号的上升沿采样。每个信号对应于时钟上升沿都有建立和保持时间的限制。在这样的时间范围内，不允许传送。在该时间范围以外，信号值或传送就没有意义了。

1. 一旦复位完成，下列信号要保证在每个 CLK 的上升沿是稳定的: LOCK#, IRDY#，

TRDY#， FRAME#， DEVSEL#， STOP#， REQ#， GNT#， REQ64#， ACK64#， SBO#， SDONE， SERR#和 PERR#。

2. 在下列特定的时钟沿，地址数据段必须是稳定的:

- a. 地址——在采样到 FRAME #有效后的第一个时钟，AD[31..00]应是稳定的，无论某些线是否在逻辑上有意义。
- b. 地址——在采样到 REQ64 #有效后的第一个时钟，AD[63..32]应是稳定的，无论某些线是否在逻辑上有意义。
- c. 数据——在读操作中一旦 TRDY#有效或写操作中一旦 IRDY#有效时，AD[31..00]稳定并有效，无论哪些字节通道参与了这个传送。在别的时间它们可以是不定的。在写传送中一旦 IRDY#有效或读传送中一旦 TRDY#有效，AD 线就不能改变直到当前数据段完成。
- d. 数据——在 ACK64#有效及读操作中 TRDY#有效或写操作中 IRDY#有效时，AD[63..32]稳定并有效，无论哪些字节通道参与了这个传送。在别的时间它们可以是不定的。
- e. 数据——特殊周期命令，当 IRDY#有效时，AD[31..00]稳定并有效，无论哪些字节通道参与了这个传送。
- f. 在读传送中 TRDY#有效后或写传送中 IRDY#有效后，不要直接选通异步数据到 PCI 上。

3. 命令/字节允许在下列特定的时钟沿要保证达到稳定:

- a. 命令——C/BE[3..0]#和 C/BE[7..4]#在第一次采样到 FRAME#和 REQ64#有效时稳定并有效，并且包含有命令编码。
- b. 字节允许——在地址段后那个时钟或(和)数据段中每个时钟沿，无论是否插入等待状态，C/BE[3..0]#和 C/BE[7..4]#是稳定并有效的。在猝发传送期间，从每个数据段完成(IRDY#和 TRDY#都有效)的那个时钟起，总线主设备修改字节允许。在下一个时钟，修改后的值有效。字节允许在数据段之间可以自由改变，但在每个数据段开始的那个时钟沿必须是有效的，并在每个数据段期间保持有效。
- c. C/BE#输出缓冲器在从数据段的第一个时钟到传送结束这段时间内都必须保持输出允许。这样就能确保 C/BE#不长时间浮空。——C/BE#.oe=!IRDY# # !TRDY#

4. PAR 在 AD[31..00]有效后的那个时钟沿稳定并有效。PAR64 在 AD[63..32]有效后的那个时钟沿稳定并有效。

5. 当操作是一个配置命令时，只有在采样到 FRAME#有效的第一个时钟 IDSEL 稳定并有效。在任何别的时间 IDSEL 都是不定的。
 6. RST#、INTA#、INTB#、INTC#和 INTD#无限制或不同步。
5. 4. 2. 控制信号:
7. 在 FRAME#和 IRDY#无效而 GNT#有效时，该单元可能启动一次操作。
 8. 当第一次采样 FRAME#有效时，一个传送开始。
 9. 在所有 PCI 传送中都有下列对 FRAME#的约束：
 - a. FRAME#及其相应的 IRDY#决定了总线的忙/闲状态，当两者之一有效时，总线忙，当二者都无效时总线闲。
 - b. 一旦 FRAME#无效，在同一次传送中，它就不能再有效。
 - c. 在 IRDY#有效前，FRAME#不能无效，在 FRAME#无效后，IRDY#应维持有效至少一个时钟周期，甚至在该传送由总线主设备失败来终止时也一样。
 - d. 一旦总线主设备发出 IRDY#有效，无论 TRDY#状态如何，它必须在当前数据段完成之后才能改变 IRDY#和 FRAME#。
 10. 最后数据段完成，当：
 - a. FRAME#无效而 IRDY#有效(正常终止)。在目标设备说明最后数据传送后(TRDY#有效)，接口恢复到 IDLE 状态。
 - b. FRAME#无效而 STOP#有效(目标设备终止)。
 - c. FRAME#无效而设备选择计时器溢出(总线主设备失败)。
 - d. DEVSEL#无效而 STOP#有效(目标设备失败)。
 11. 当 FRAME#和 IRDY#均无效时，传送结束。
 12. 在所有 PCI 传送中，FRAME#，IRDY#，TRDY#和 STOP#都有下述规则：
 - a. 无论何时，只要 STOP#有效，按 FRAME#无效规则，应尽快使 FRAME#无效(IRDY#必须有效)。使 FRAME#无效应在 STOP#有效后尽快进行，可能是 2~3 个时钟。目标设备不能假定任何 STOP#有效和 FRAME#无效之间的时间关系，但必须保持 STOP#有效直到 FRAME#无效。当总线主设备采样到 STOP#有效，它必须在 IRDY#有效后第一个时钟使 FRAME#无效，此后 IRDY#仍有效。IRDY#有效可以作为总线主设备的 IRDY#正常操作之结果(当前传送未被目标设备终止)，并根据总线主设备何时做好完成数据传送的准备而被延迟零个或多个时钟周期。相应地，如果 TRDY#无效时，总线主设备可立即使 IRDY#有效，说明再也没有数据传送了。
 - b. 一旦 STOP#有效，它必须保持有效直到 FRAME#无效。在 FRAME#无效的下一个时钟，STOP#必须无效。
 - c. 在传送的最后一个数据段期间(FRAME#无效而 IRDY#有效)，STOP#或 TRDY#之一有效的任何时钟沿就成为本次传送的最后时钟，且在下一个时钟沿 IRDY#无效(从此开始一个 IDLE 周期，并规定了传送的结束)。
 - d. 一旦目标设备已使 TRDY#或 STOP#有效，在当前数据段完成之前，它就不能再改变 DEVSEL#，TRDY#或 STOP#。一旦总线主设备或目标设备开始了数据传送，它就不能改变状态。
 13. 总线主设备和目标设备之间每个 IRDY#和 TRDY#都有效的时钟沿都在传送数据。总线主设备和目标设备分别可用 IRDY#无效或 TRDY#无效将等待周期插入到数据段中。
 14. 当数据有效时，数据源应无条件地使其 XRDY#有效(写传送中 IRDY#，读传送中 TRDY#)。接收单元应按其选择而有效的驱动它的 XRDY#。
 15. 当前主设备由目标设备终止时(STOP#有效)，总线主设备必须至少在两个时钟内时其

REQ#无效，一个是总线回到空闲状态，另一个在它之前或之后。如果总线主设备想完成操作，它必须用下一个未传送数据的地址在晚些来重试被目标设备终止的传送。

16. 单元通过使其 DEVSEL#有效来确认作为操作的目标设备。
17. 只有 DEVSEL#有效，目标设备才能驱动 TRDY#。
18. 一旦 DEVSEL#有效，就不能无效，直到最后数据段完成。除非发出目标设备失败信号。

5. 4. 3. 锁定操作

19. LOCK#只能由一个单元拥有并驱动，在总线释放时可能保持不变。
20. 支持 PCI 上的 LOCK#的目标设备必须遵守下列规则：
 - a. 当 LOCK#在地址段期间无效时，锁定操作的目标设备自我锁定。
 - b. 一旦锁定建立，该目标设备保持锁定直到它采样到 FRAME#及 LOCK#均无效或它发出目标设备失败信号。
 - c. 保证给 LOCK#的拥有者(一旦锁定建立)至少 16 字节的资源。对多口设备，这包括不在 PCI 上进行的操作。
21. LOCK#由当前的总线主设备驱动，在 PCI 上使用 LOCK#的总线主设备必须遵守下列规则：
 - a. 在锁定操作期间，一个总线主设备只能操作一个资源。
 - b. 锁定不能跨越设备的界限。
 - c. 16 字节(线性排列)是总线主设备在锁定操作期间能计数的最大资源范围。对这 16 个字节的任一部分操作都要锁定全部 16 个字节。
 - d. 锁定操作的第一个传送必须是读传送。
 - e. LOCK#必须在地址段后那个时钟有效，并保持有效以维持控制。
 - f. 如果在数据段完成和锁定未建立之前发重试信号，LOCK#必须释放。
 - g. 无论何时，操作被目标设备失败或总线主设备失败终止时 LOCK#必须释放。
 - h. 在连续锁定操作之间，LOCK#必须释放至少一个空闲周期。

5. 4. 4 仲裁:

22. 总线仲裁器可以在任何周期使某单元的 GNT#无效。任何单元必须保证在开始传送的时钟沿 GNT#是有效的。如果 GNT#无效，传送就不能进行。
23. 一旦 GNT#有效，根据下列规则可使其无效：
 - a. 如果 GNT#无效而 FRAME#有效，总线传送有效且继续进行。
 - b. 如果总线不是处于空闲状态，一个 GNT#能在另一个 GNT#有效的同时无效。否则，使某个 GNT#无效和使下一个 GNT#有效之间必须有一个时钟的延迟，不然在 AD 线和 PAR 线上就可能会有冲突。
 - c. FRAME#无效时，GNT#可以在任何时间无效。以便为高优先级的总线主设备服务，作为对相应 REQ#无效的应答。
25. 若仲裁器使某个单元的 GNT#有效而总线处于空闲状态，该单元就必须在 8 个 PCI 时钟内(要求值)开放其 AD[31..00]、C/BE[3..0]#及 PAR 输出缓存，推荐值是 2~3 个时钟。

5. 5. 5. 奇偶校验:

25. 在每个地址段和数据段,所有 AD 线(如总线主设备支持 64 位数据通道,还包 AD[63..32])

都必须驱动到稳定的值。甚至未参与当前数据传送的字节通道读应有稳定的数据在其上，以便进行奇偶校验。奇偶校验根据下列规则产生：

- a. 无论传送类型、格式，所有 PCI 传送的奇偶校验的计算是相同的。
- b. AD[31..00]、C/BE[3..0]#和 PAR 上“1”的数量是偶数。
- c. AD[63..32]、C/BE[7..4]#和 PAR64 上“1”的数量是偶数。
- d. 产生奇偶校验是不可选择的，所有 PCI 兼容设备都必须做奇偶校验。

5.6 寻址

PCI 上地址解码是分散的，即每个单元负责自己的地址解码，这样避免了采用中央解码逻辑及在使用配置的设备之外的设备选择信号。PCI 支持两种类型的设备地址解码：正解码和反解码。正解码的设备只在分配给它的地址范围内进行解码，解码速度相对较快。反解码只能被总线上的一个设备使用，因为它接收所有不被其它单元解码的操作，这种解码对于诸如相应于高端地址空间的标准扩展总线这样的单元是很有效的，该单元常常就是与一个标准总线相连的桥路。能完成正的或反的解码的目标设备，对保留总线命令编码不能作出反应(驱动 DEVSEL#有效)。

包含于低两位地址(AD[1..0])上的信息随地址空间而改变。在 I/O 地址空间，所有 32 位都用来提供完整的字节地址。这样就使得要求地址分解到字节一级的单元能完成地址解码，不必因为等待字节允许而增加等待状态的周期(这会将反解码推迟一个额外时钟周期)。AD[1..0]仅用于地址解码并说明参与当前传送的最低字节。AD[1..0]的编码组合如下表所示，任何不在表中的组合都是非法的，并由目标设备失败来终止。

AD1	AD0	C/BE3#	C/BE2#	C/BE1#	C/BE0#
0	0	×	×	×	0
0	1	×	×	0	1
1	0	×	0	1	1
1	1	0	1	1	1

一旦一个目标设备确认了一个 I/O 操作，那么它就要决定它能否完成如字节允许所说明的全部操作。如果所选中的字节都不在选中的目标设备地址范围内，整个操作不能完成，此时，目标设备不传送任何数据，用目标设备失败来终止传送。

在存储器命令期间，AD[1..0]决定了猝发顺序，有如下意义：

AD1	AD0	猝发顺序
0	0	线性增加
0	1	高速缓存线触发器模式
1	×	保留

在存储器命令传送期间，所有目标设备都应检查 AD[1..0]，并且提供期上所要求的猝发顺序，或在第一个数据段后让目标设备脱离总线。所有支持猝发的设备都要求线性触发顺序的采用。对采用高速缓存线触发器则不要求。在存储器地址空间，是对由 AD[31..02]解码得到的双字地址进行操作的。在线性增加模式下，在每个数据段后，认为地址增加 4 个字节，直到传送终止。

当目前的读传送是针对可高速缓存的时候，无论字节允许情况如何，都必须返回全部字节的数据。这就要求决定可高速缓存性能的单元保证目标设备返回全部字节的数据。如果可高速缓存性由提出传送要求的一方决定，那么它必须使所有字节允许都有效(低)，以使目标设备能返回全部所要求的数据。如果可高速缓存性由目标设备决定，它就必须忽略字节允许而返回全部双字。可高速缓存的目标设备要么就返回整条高速缓存线上的数据，要么只返回所要求数据的第一个。


如果目标设备不支持高速缓存但支持预取，它也必须不管字节允许情况而返回全部字

节数据。如果没有别的影响，目标设备可以只工作于这种模式。

在配置地址空间，是对由 AD[7..2]解码得到的双字地址进行操作的。当一条配置命令被解码，IDSEL 有效且 AD[1..0]是 00 时，某个单元就判断出它是本次操作的目标设备。否则该单元不理睬当前传送。通过解码配置命令及桥路号，且 AD[1..0]是 01，桥路可以判别出某个操作是针对挂在它边上的设备的。

PCI 允许任何连续或非连续的字节允许的组合。如果没有字节允许有效(全为 1)，目标设备必须用 TRDY#有效来完成这次传送并在读请求时提供奇偶校验。无字节允许有效的传送的目标设备必须在无任何永久性改变的情况下结束当前数据段。在读传送中，这意味着数据和状态都没有改变。如果完成该操作对数据和状态都没有影响，那么目标设备可以在该操作中提供数据，也可以不提供。在读操作中，无论字节允许的状态怎样，目标设备都必须提供 AD[31..00]及 C/BE[3..0]#上所有数据的奇偶校验。在写传送中，数据不存储且 PAR 有效。

5.7 总线传送

由一个以上单元驱动的信号必须有一个转换周期。该转换周期用以避免在一个单元停止驱动该信号而另一个单元开始驱动它时可能发生的冲突。在时序图上这种转换用  来表示。对不同的信号，转换周期发生的时间不一样。例如：IRDY#、TRDY#、DEVSEL#、STOP#和 ACK64#用地址段作为它们的转换周期。FRAME#、REQ64#、C/BE[3..0]#、C/BE[7..4]#、AD[31..00]及 AD[63..32]用传送之间的 IDLE 周期作为它们的转换周期。LOCK#的转换周期是当前拥有者释放它之后的那个时钟。PERR#的转换周期是最后的数据段之后的第四个时钟，此时 AD 线转换完成已有三个时钟。

下列时序图中示出参与 32 位传送的各种重要信号之间的关系。实线表示正被当前总线主设备或目标设备驱动的信号。虚线表示没有单元驱动的信号，然而，如果虚线为高，则应假定它仍保持稳定值。当虚线为高、低之间，说明三态信号是不确定值(例如 AD 线和 C/BE#线)。实线变为点划线，说明信号被驱动，现在是三态。当一条实线由低变高，然后变为点划线，这表明信号被驱动到高，以给总线预充电(s/t/s 信号)，然后变为三态。

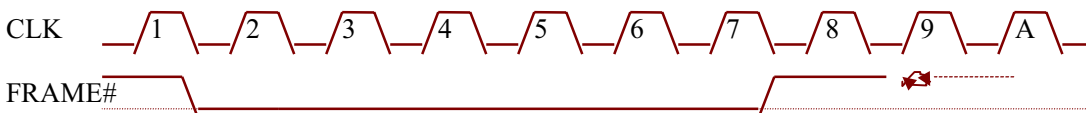
5.7.1 读传送

图 5.1 的读传送发生在 clock2，由 FRAME#第一次有效时的地址段开始，在地址段期间，AD[31..00]包含一个有效的地址，C/BE[3..0]#包含一个有效的总线命令。

第一个数据段的第一个时钟是 clock3。在数据段期间，C/BE#说明了哪些字节通道参与了当前数据段的传送。C/BE#输出缓冲器在从数据段的第一个时钟到传送结束这段时间内都必须保持输出允许。这样就能确保 C/BE#不长时间浮空。

读传送的第一个数据段要求有一个转换周期(由目标设备通过 TRDY#设置)。在这种情况下，地址在 clock2 有效后，总线主设备停止驱动 AD 线。目标设备可以提供有效数据的最早时间是 clock4。当 DEVSEL#有效时，在转换周期后，目标设备必须驱动 AD 线。一旦允许输出，输出缓冲器必须保持输出允许直到传送结束。

图中，在 clock3、5、7，IRDY#和 TRDY#其中之一无效，就插入等待周期。在 clock4、6、8，IRDY#和 TRDY#都有效时，完成数据传送。只有 DEVSEL#有效，才能驱动 TRDY#。在最后一个数据段，只有 IRDY#有效时，FRAME#才能无效。



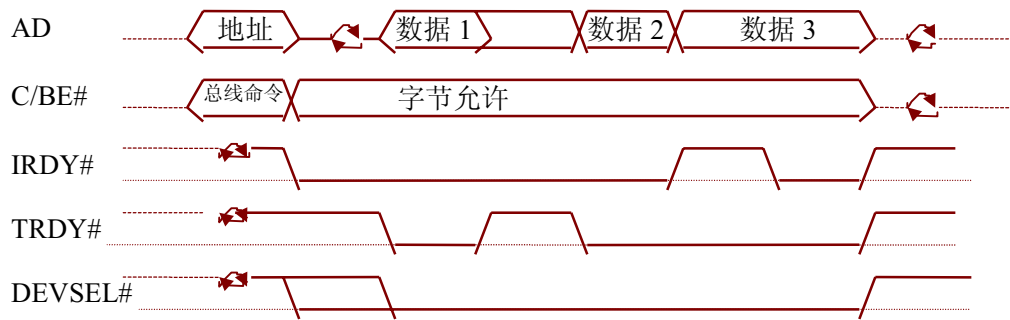


图 5.1 基本的读操作

5.7.2 写传送

图 5.6 所示为写传送，地址和数据都由总线主设备提供，因而在地址段之后不要求有转换周期。第一、第二个数据段是按零等待周期完成的。在 clock5，因 IRDY#无效，传送由总线主设备延迟。虽然这样做允许总线主设备延迟数据，但不允许延迟字节允许。

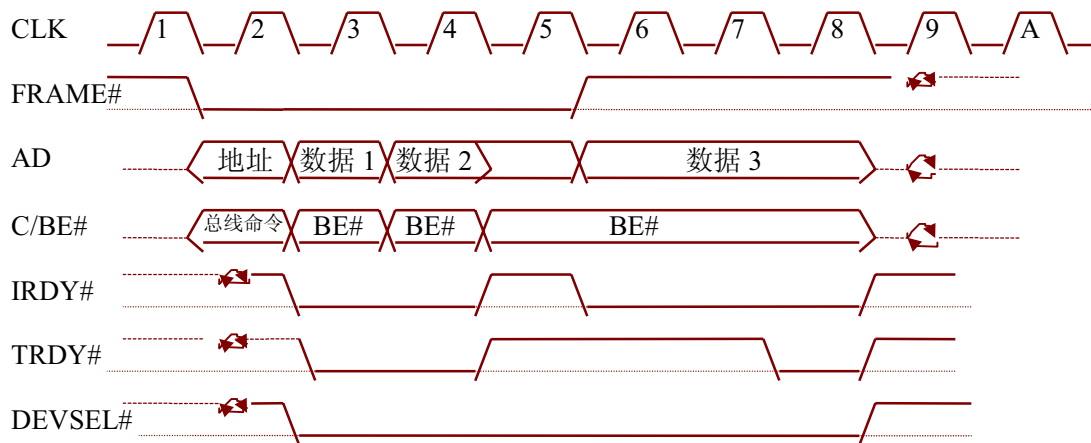


图 5.6 基本写操作

5.7.3 传送终止

总线主设备和目标设备都可以终止 PCI 传送。在总线主设备和目标设备都不能单独有效地终止传送时，总线主设备保持主要的控制，无论终止是由什么引起的，都将带给传送一个有效的、系统的结论。

5.7.3.1 总线主设备引起的终止

总线主设备用来引起终止的机制是当 IRDY#有效时，FRAME#无效。这就对目标设备表明最后一个数据段正在进行。当 IRDY#和 TRDY#都有效时，最后一个数据传送发生。当 FRAME#及 IRDY#都无效时，传送完成，总线回到 IDLE 状态。

总线主设备可以因下列两种原因之一而用这种机制去终止传送：

- 完成： 相应于总线主设备已完成它想要进行的传送。这是终止的最常见原因。
- 时间溢出： 总线主设备的 GNT#已无效且其内部延迟计时器已满(目标设备引起操作延迟或是要进行的操作太长)。高速缓存写传送不被延迟计时器所控制，它只能在高速缓存边界上被停止。用高速缓存写命令启动传送的总线主设备的在写满一条高速缓存线之前不会理会延迟计时器，当传送到一条高速缓存线的边界且延迟计时器已计满(并且 GNT#无效)

时，总线主设备必须终止这次传送。如果是目标设备来终止高速缓存写传送，则总线主设备用存储器写命令尽快完成这个传送(因为不再存在高速缓存写的条件)。也应该注意到，除非 GNT#无效，否则在计时器计满后也不需终止传送。

图 5.7 是两个正常完成的例子，在 clock3，FRAME#无效，IRDY#有效向目标设备表明是最后一个数据传送，主设备保持 IRDY#有效，待目标设备发出 TRDY#有效，最后一个数据传送完成，总线回到 IDLE 状态。

图 5.7 的两种情况也可能是由时间溢出引起正常终止。左边的情况，因计时器满，FRAME#在 clock3 无效，GNT#无效，且总线主设备已准备好传送最后一个数据(IRDY#有效)。因为在计时器满时 GNT#已无效，故不允许继续使用总线，除非是使用高速缓存写命令，因为它只能在高速缓存线边界上被停止。右边的情况，在 clock1 计时器满。IRDY#在 clock2 无效，总线主设备未准备好传送数据，这就要求 FRAME#仍维持有效。在 clock3，总线主设备已准备好完成这次传送(IRDY#有效)，故 FRAME#无效。这种终止延迟最多不能超过 2—3 个时钟周期。

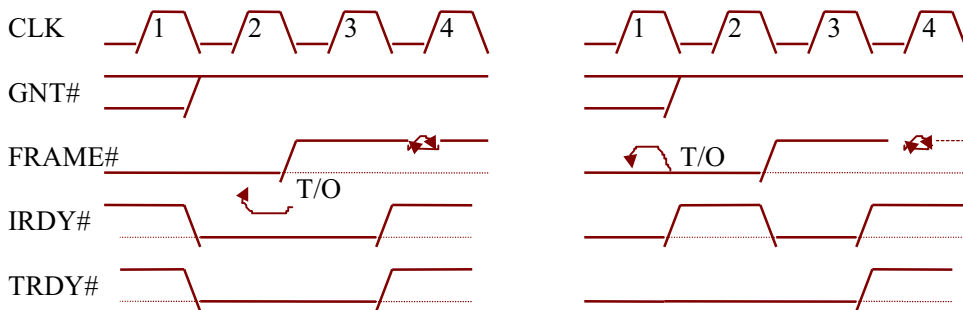


图 5.7 总线主设备引起的终止

在这种机制的改进型中，允许在没有目标设备应答的情况下总线主设备终止传送。这种异常终止称为**总线主设备失败**。尽管这样可能会在要求这次传送的应用中引起严重错误，但传送能很好的完成，以保护别的单元的正常 PCI 操作。

由总线主设备失败引起终止的例子如图 5.8 所示。这是由总线主设备终止的一种不正常情况(除配置和特殊命令周期外)。如果 DEVSEL#直到 clock6 仍无效，则总线主设备可判断这次传送没有应答。总线主设备假定这次操作的目标设备没有能力完成所要求的传送或地址错误，在 clock7 使 FRAME#无效，在 clock8 使 IRDY#无效。总线主设备用总线主设备失败来终止一次传送的最快情况是在第一次采样到 FRAME#有效后五个周期，总线主设备作一个单数据传送时情况就是这样，如果 DEVSEL#在 clock3、4、5、或 6 已有效，这就说明传送请求已由一个单元获知，就不再允许总线主设备失败。

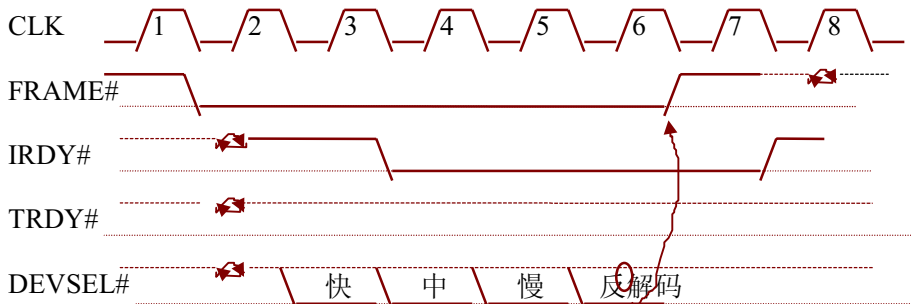


图 5.8 总线主设备失败终止时序

在 PC 兼容系统中,主总线桥路在用总线主设备失败终止时,读传送必须全部返回“1”,写传送必须放弃数据。桥路必须在状态寄存器中设立总线主设备失败判断位。在总线主设备不能通过其设备驱动器反映错误时,别的总线主设备设备可以将这一位看作错误标志而发出 SERR#。一个 PCI 对 PCI 桥路必须支持 PC 兼容性,就象主总线桥路那样。当这个 PCI 对 PCI 桥路在别的系统中应用时,桥路必须象别的总线主设备一样工作。桥路所作的读数据的预取对系统必须是透明的。这意味着当一个预取传送由总线主设备失败终止时,桥路必须简单地停止传送并继续进行无响应的正常操作。当传送不是由目标设备申请时,这种情况发生。

5.7.3.2 目标设备终止

目标设备终止中所用的机制是 STOP#信号。目标设备发出 STOP#信号去要求总线主设备终止传送,一旦发出,STOP#就保持有效直到 FRAME#无效。总线主设备用下一个未传送数据的地址在晚些时候重新开始由重试或解除连接终止的传送。当当前传送由目标设备终止时,总线主设备必须使其 REQ#信号无效。总线主设备至少应使 REQ#保持两个 PCI 时钟无效,一个是在总线主设备回到 IDLE 状态时,另一个是在 IDLE 状态之前或之后。如果总线主设备想完成传送,在其使 REQ#无效两个时钟之后或某些可能的不能进行传送的状态发生之后立即重使 REQ#有效。否则,单元只是在它重新需要使用接口时使 REQ#有效。

目标设备可由下列两个原因之一而启动使用这种机制的终止:

重试(Retry): 相应于目标设备正处于不能处理传送的状态的终止。这种情形包括死锁的可能性、某些非 PCI 资源忙状态或一种锁定操作的锁定状态。重试意味着目标设备终止该传送且没有传送数据。

解除连接(Disconnect): 相应于目标设备在 PCI 导线延迟时间内不能作出应答而要求终止(PCI 导线延迟时间是 8 个 PCI 时钟)。注意在第一个数据段时这种情况通常不会发生。解除连接意味着目标设备在数据传送时或在数据已被传送之后终止传送。

可高速缓存的目标设备除非到了一条高速缓存线的边界,否则不能解除与高速缓存线存储器的连接,无论高速缓存当前是否被允许。所以当操作是针对一个可高速缓存的存储器范围时,“监视”单元常假设一条高速缓存写命令将完成而不被解除连接。

在这种机制的改进型中,目标设备可终止有重大错误的传送,或是不能作出应答的传送。这种异常终止称为**目标设备失败**。尽管这样可能会在要求这次传送的应用中引起严重错误,但传送能较好完成,以保护别的单元的正常 PCI 操作。

许多目标设备将至少被要求具有重试能力,但目标设备也可以选择其它的目标设备启动终止的版本。总线主设备对它们全部都要能准确处理。象下面这些简单的目标设备,重试能力也是可选用的:(1)不支持锁定操作;(2)不能判断死锁和活锁的可能性;(3)不进入一种可能需要的已拒绝操作的状态。

图 5.5 示出三种解除连接的例子,STOP#与 FRAME#之间有如下关系:

- 当 STOP#有效时给出解除连接信号(这时 DEVSEL#必须有效,否则就说明目标设备失败),一旦 STOP#有效,它必须保持有效直到 FRAME#无效。

- 在 STOP#有效后,FRAME#应立即无效。例 C 中多花了一个时钟周期是因为在 STOP#有效后 IRDY#不能立即有效。

- 在 FRAME#无效后的下一个时钟 STOP#立即无效。

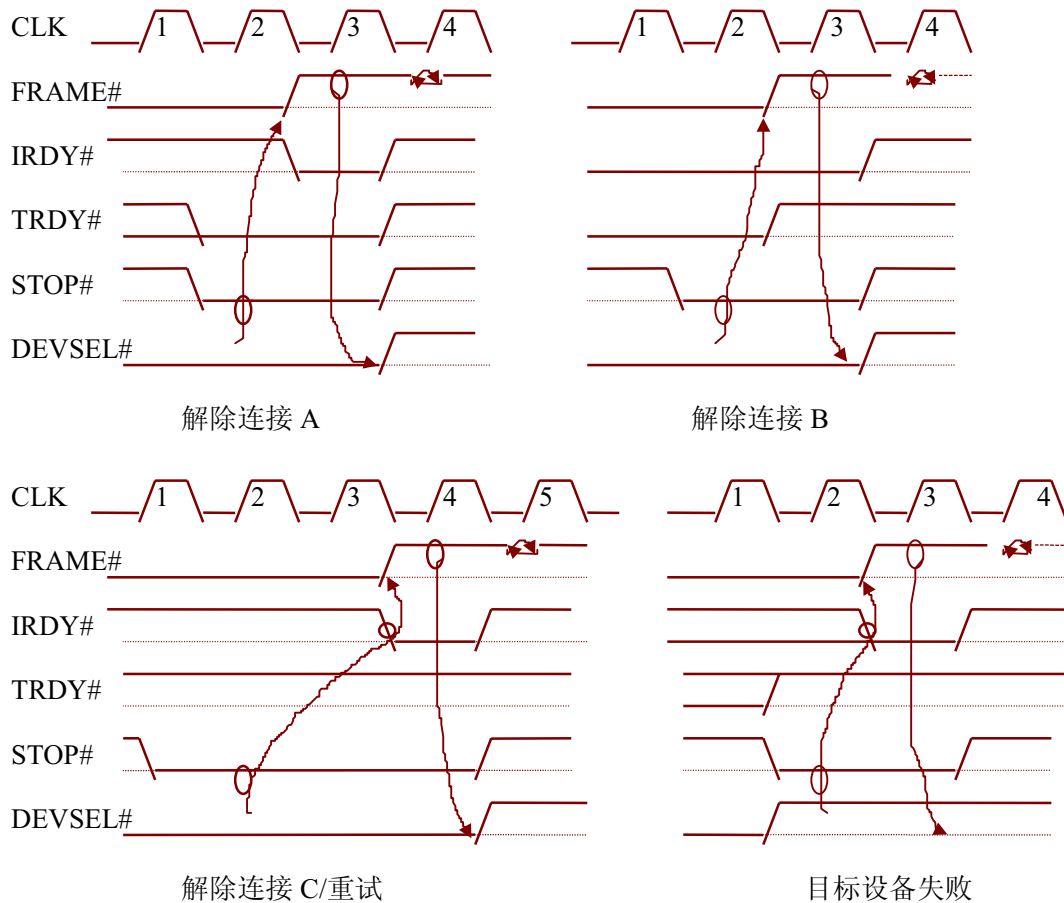


图 5.5 目标设备终止

IRDY#与 TRDY#之间的关系独立于 STOP#和 FRAME#之间的关系。所以，在目标设备要求终止期间，数据可以被传送，也可以不被传送，这仅取决于 IRDY#和 TRDY#的状态。然而，当 STOP#有效且 TRDY#无效，说明目标设备将不再传送数据，且总线主设备也不再象在一个完整传送中那样去等待最后的数据传送。

在例 A 和 B 中，目标设备想在 STOP#有效后，还传送一个数据，然后停止，因而它在 STOP#有效时使 TRDY#有效，但只能进行一个数据传送，完成后 TRDY#必须无效。例 A 中，因总线主设备未准备好，数据在 FRAME#无效后传送。例 B 中数据在 FRAME#无效前传送。如果目标设备在最后数据段中要求一个等待周期，它必须延迟 STOP#的有效直到它准备好完成这次传送。

例 C 是解除连接的一种全无数据传送的特殊情况(TRDY#无效因而在 STOP#有效后无数据传送)，注意，FRAME#的无效延迟到 IRDY#有效后才进行。这也是一个重试的例子。重试的一个普通的例子是当目标设备因进行锁定操作而被另一个总线主设备锁定时。另外的例子是目标设备在允许传送进行之前，先要对别的非 PCI 资源进行操作。

图 5.5 右下角是一个当 STOP#有效时 DEVSEL#无效时目标设备终止的例子。这说明目标设备要求终止传送且不再重做该传送。此外，如果在当前传送中已有数据被传送，它有可能不可靠。在发出目标设备失败信号前，DEVSEL#必须有效一个或多个时钟周期有效，

且 TRDY#必须无效。

5.8 仲裁

为了减少操作延迟，PCI 仲裁更近于基于操作而不是基于时间槽。就是说总线主设备必须就它在总线上完成的每一个操作作出仲裁。PCI 使用了一个中央仲裁电路，在其中每个总线主设备单元都有一个唯一的请求(REQ#)和允许(GNT#)信号。这种简单的请求—允许联络用来获得对总线的操作权。仲裁是“隐蔽”的，意为它发生在上一个操作期间，所以不会因仲裁而花费 PCI 周期，除非总线处于 IDLE 状态。

中央仲裁采用了特别的仲裁算法，例如循环、优先、公正等等。必须定义一种仲裁算法作为产生错误情况延时的基础。然而，因为仲裁算法基本上不属于总线规范的范围，系统设计者可以改变它，但必须提供它们所选用的 I/O 控制器和外插卡的延迟要求。总线允许同一单元的背对背(back to back)传送，也允许仲裁对优先的和重要的请求的灵活性。在任何周期，只要 GNT#信号有效，仲裁就提供给某一单元总线操作权。

某一单元通过使其 REQ#有效而请求总线。单元只能使用 REQ#来发出真正的总线使用需求信号。单元绝不能用 REQ#将自己“停放(park)”在总线上，如果应用了总线停放，正是仲裁器指派了约定的拥有者。当仲裁器判断某一单元可以使用总线时，它使该单元的 GNT#有效。

仲裁信号协议详见 5.1 操作规则，图 5.6 说明了基本仲裁，用了两个单元来说明仲裁器如何交换总线操作。REQ#—a 在 clock1 之前或 clock1 时有效，以申请使用接口。因为 GNT#—a 在 clock2 有效，所以单元 A 可以操作总线。单元 A 可以在 clock2 开始传送，因为此时 FRAME#和 IRDY#无效而 GNT#—a 有效。在 clock3，当 FRAME#有效时，单元 A 开始传送。因为单元 A 想作另一次传送，所以保持 REQ#—a 一直有效。当 FRAME#在 clock3 有效时，仲裁器决定单元 B 是下一个总线使用者，且在 clock4 使 GNT#—b 有效，GNT#—b 无效。

在 clock4，当单元 A 完成它的传送时，放弃总线。当 FRAME#和 IRDY#都无效时，所有 PCI 单元都能判断当前传送的结束。在 clock5，单元 B 拥有总线操作权(因 FRAME#和 IRDY#均无效)，并在时钟 clock7 完成其传送。

注意在 clock6，REQ#—b 无效且 FRAME#有效使，说明单元 B 只要求一次传送。因为 REQ#—a 仍然有效，故仲裁器同意单元 A 作下一个总线操作。

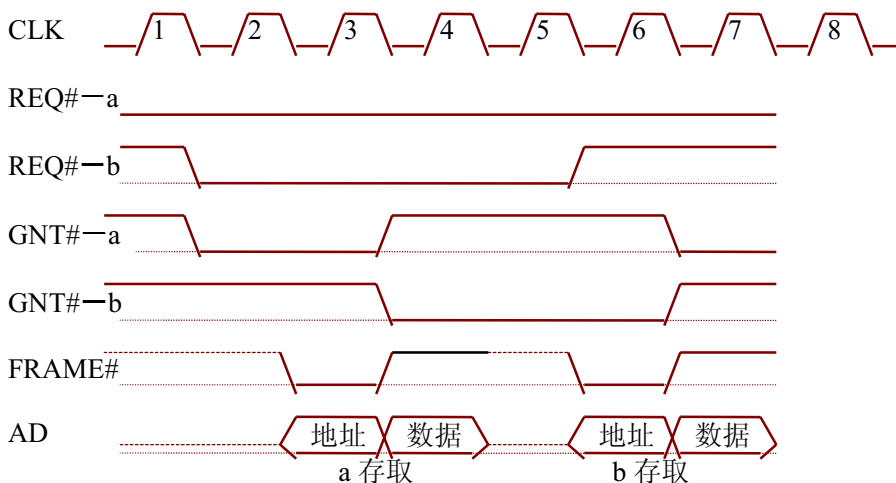


图 5.6 基本仲裁

当前总线拥有者在它要求额外(外加)传送时，应保持 REQ#有效。如果没有别的总线请求有效，或当前总线主设备具有最高优先级，仲裁器让当前总线主设备继续操作总线。

GNT#让单元作一次总线传送，如果某一单元只想作一次传送，它将在 FRAME#有效的同一个时钟使其 REQ#无效，如果一个单元想作另一个操作，它将继续使其 REQ#有效。单元可在任何时候使其 REQ#无效，但仲裁器可以将其理解为该单元不再延长它对总线的请求而且可以使其 GNT#无效。

当目标设备终止某一传送使(STOP#有效)，总线主设备必须使其 REQ#至少在两个 PCI 时钟内无效，一个是总线回到 IDLE 状态，另一个在此之前或之后。如果总线主设备想完成该传送，它必须在 REQ#无效或一个潜在的需求条件可能发生之后重新使其 REQ#有效。如果总线主设备不想完成该传送(因为它正在预取或一个高优先级的内部需求要求服务)，那么该单元在下次它又需要使用接口时使 REQ#有效。这样，当上一个目标设备准备下一次传送时，允许别的单元使用接口。

如果当前总线主设备的 GNT#已有效后它没有开始操作(其 REQ#也有效)，并且有 16 个 PCI 时钟总线处于 IDLE 状态，则仲裁器可以假定当前总线主设备“断路”。然而，仲裁器可以在任何时候改变 GNT#而为更高优先级的单元服务。

5.9 仲裁放置(PARKING)

“放置”指的是当前无单元使用总线或申请总线时，允许仲裁器使某一被选定单元的 GNT#信号有效。仲裁器可以按其所愿方式选择缺省总线拥有者或选择完全不放置。当仲裁器使某一被选定单元的 GNT#信号有效而总线处于 IDLE 状态时，该单元必须在 8 个 PCI 时钟(要求这样)内使其 AD[31..00]、C/BE[3..0]#和(一个时钟后)PAR 的输出缓冲器，推荐值是 2—3 个时钟。不强迫该单元在一个时钟内开放所有缓存器。这种要求保证仲裁器能安全地将总线放置给某些单元并知道总线将不会浮空。如果仲裁器不放置总线，嵌有仲裁器的中央资源设备就驱动总线。

在所有情况下，如果总线处于 IDLE 状态而仲裁器使一个单元的 GNT#无效，该单元就失去了对总线操作的权力。唯一的例外情况是仲裁器使 GNT#无效的同时，该单元使 FRAME#有效，这时总线主设备将继续进行传送。否则，该单元就必须使 AD[31..00]，C/BE[3..0]#和(一个时钟后)PAR 处于三态以脱离总线。与上述情况不同，该单元必须在一个时钟关闭所有缓存器以避免和下一个总线拥有者发生冲突。

如上所述，从总线 IDLE 状态起到实现总线仲裁的最小延迟如下：

- 放置： 已放置单元零个时钟周期，其它单元两个时钟周期。
- 不放置： 每个单元都是一个时钟周期。

当总线已被放置到某一单元上时，可以允许该单元不用 REQ#有效就开始传送。当总线处于 IDLE 状态且 GNT#有效，总线主设备就能开始一次传送。当总线主设备只要求一次传送时，它不能使 REQ#有效；否则，在它不要求使用总线时，仲裁器也可以继续使其 GNT#有效。当该单元需要作多个传送时，它应使 REQ#有效以告知仲裁器它想作多个传送。

5.10 延迟

PCI 是一种低延迟、高通过率的 I/O 总线。本节叙述帮助预测和控制错误情况延迟的 PCI 机制。给出这些机制，对单一 PCI 环境和高效主存储器接口的等待时间能很准确地预测出来。扩展标准总线(ISA、EISA 或 MC)的加入将使延迟预算更加困难。在带有扩展总线的情况下，错误延迟方案可以由扩展总线或表现不佳的接口属性而不是 PCI 总线决定。

5.10.1 PCI 上的延迟

操作延迟由三部分组成：

- **仲裁延迟**——从总线主设备发出 REQ#有效到收到 GNT#有效的的时间。它由仲裁算法、设

备的优先级要求和系统使用情况决定。对高优先级设备，该时间的典型值为 2 个 PCI 时钟。低优先级设备的延迟则根据高优先级的设备的数量和设备使用情况来决定。在没有别的总线主设备请求时，仲裁延迟的典型值是 2 个时钟

• **总线获取延迟**——从总线主设备收到 GNT#有效到它使 FRAME#有效的的时间，是设备等待总线变空共等待的时间。它要么是系统中第一个数据段的最大延迟时间(从 FRAME#有效到 TRDY#有效的的时间)。要么就是总线主设备延迟计时器的值(如果所有第一个数据段延迟都较低的话)。当总线处于 IDLE 状态时总线获取延迟是零周期。否则要等待当前总线主设备完成其数据传送或其延迟计时器满。

• **目标设备延迟**——从总线主设备使 FRAME#有效到目标设备为第一次数据传送使 TRDY#有效所花费的时间。其典型值正好是第一个数据的延迟。但作为缓存的 PCI 桥路设备能使目标设备延迟更长。例如，如果一个设备正在通过 CPU 桥路缓存对主存储器继续操作，该桥路在允许操作完成前可能不得不刷新其缓存器。该桥路将发出重试信号给提出要求的设备，刷新缓存器，然后准备好应答请求。根据要刷新什么地方和缓存器的大小，这样做会大大增加目标设备延迟时间。

在无别的单元请求总线的情况下，PCI 总线主设备能猝发传送目标设备所能接收长度的数据。然而，PCI 规定了两种机制，在有别的请求存在时，覆盖总线主设备的总线占有状态。这样，可预计的总线获取延迟就能实现。这两种机制定义如下：

总线主设备延迟计时器(LT)：在总线主设备未使 FRAME#有效时，每个总线主设备的 LT 都被清除且被挂起。当总线主设备使 FRAME#有效时，它就允许其 LT 计数。如果该总线主设备在计数计满之前使 FRAME#无效，则 LT 无意义。否则，一旦计数满(计数 T 时钟，时间溢出)，如果总线主设备之 GNT#改变，它必须立即放弃总线占用(目标设备准备好后就开始有效的终止)。大致上，T 代表了分配给总线主设备的最小时间片段(单位是 PCI 时钟)，它是一种通过量(高值)和延迟(低值)之间的折衷。例如，假设第一个数据段要 8 个时钟周期的延迟，T=40 时在总线主设备和目标设备读能做零等待猝发的情况下提供了一个 32 个数据段的猝发，将 T 减少到 20 则每次猝发都只传送 12~14 个数据段，但最大传送就限制在 28 个时钟内。

目标设备开始的终止(特别解除连接)：如果到数据段“N+1”所要增加的延迟超过 8 个时钟，目标设备就要根据数据段 N(此时 N=1, 2, 3...)的完成按结束传送方式处理 TRDY#和 STOP#。例如假设一个 PCI 总线主设备从一个扩展总线读取数据要花费至少 15 个时钟。根据这一规则，N=1，到数据段 2 的延迟是 15 个时钟周期，所以目标设备必须在数据段 1 完成时终止传送。

注意，这两种机制都有限制第一个数据段的延迟。例如，假设在一特定系统中，没有目标设备慢到完成第一个数据段要延迟 TRDY#超过 T+8 个以上的时钟。根据给定假设和上述机制，总线主设备将做的最长的传送是 T+8 个时钟(假设总线主设备之 GNT#在其 LT 计满之前变化)。

不同系统所用实际操作延迟往往有所不同，这是由于各系统所用设备的延迟特性引起的。但 PCI 设备制造商必须做一些典型操作延迟的测量，所以它们能提供足够的片内缓存来减少超载和欠载。

5.10.2 延迟指导原则

在许多的 PCI 系统中，操作延迟的典型值都较短(大约 2us)内并且易于测量，然而，错误情况延迟不仅会增大，而且在一些情况下，很难预测。例如，通过桥路的扩展标准总线适配器(ISA/EISA/MC)的延迟通常是适配器的性能，而不是 PCI 的性能。作为补偿，要求保证错误情况下操作延迟的总线主设备必须为 30ms 而提供足够的缓存。如果某些错误是

可以忽略的，并能够判断缓存器的超载/欠载，以起动数据包的传送与重发，可将错误延迟时间做到 10ms 左右以降低成本。

某些应用(如嵌入控制，多媒体等)可能会要求比典型主系统(通用目的)的原始 PCI 所提供的延迟更加严格的延迟控制。这种类型的应用，要求目标设备具有快速、可预计、有限延迟等特性。建议采用如下原则来设计目标设备接口。其中“单层”意指对选定资源进行立刻操作的 PCI 目标设备，例如单口 DRAM 和帧缓存 VRAM。“多层”相应于这样的目标设备：为给所选定资源提供操作而必须取得一个独立的仲裁资源。如 PCI 到总线的桥路和双口 DRAM。

(1) 单层目标设备将第一个数据段的延迟限制在 16 个时钟。如果一个暂时的内部状态(如 DRAM 刷新)将要操作，则该操作的延迟会超过 16 个时钟，并要立刻重试。

(2) 多层目标设备将重试与忙资源发生冲突的操作。例如，对 EISA 的从属设备进行操作，而此时，EISA 被别的总线主设备所占用，该操作就要立刻重试。同样，一个对正在扫描刷新的帧缓存 DRAM 的操作也要重试。

(3) 多层目标设备(尤其是总线对总线的桥路)更加注重写缓存的方案。写缓存使得限制延迟更加困难，因为它所要求的较强的顺序性常常是在允许别的方式的操作进行之前要完成所有的序列。特别要推荐的是，在主 CPU 到 PCI 的桥路能分解到固定的 PCI 帧缓存写的同时，也支持分解到 ISA/EISA/MC 目标设备的写。

最后，为了易于做到可靠的系统级折衷，设备销售商应该清楚地标明设备的延迟特性。总线主设备必须明确标明延迟的预计值及违法延迟回出现的结果。目标设备则规定错误状态的反应，最好是所有可能引起重试和解除连接的事情都有说明，如果目标设备的延迟由外部因素而决定，也要清楚地表明。

5.11 快速背对背传送

快速背对背传送是总线传送之间没有 IDLE 状态的传送。当 IRDY#和 TRDY#都有效，FRAME#无效时，完成最后的数据段。在上一个传送的最后数据传送的同一个时钟，当前总线主设备不经过 IDLE 状态而直接开始下一个传送。当避开 TRDY#、DEVSEL#和 STOP#上的争用时，在 PCI 上就可作快速背对背传送。快速背对背传送有两种类型：

第一种类型的快速背对背传送支持由总线主设备承担避免争用的任务。当总线主设备能保证无争用发生时，它可以取消传送之间的 IDLE 周期。如果总线主设备的第一个操作是写操作而第二个操作的目标设备与第一个相同时，这种情况就可能实现。这种类型的快速背对背传送要求总线主设备能知道潜在的目标设备地址范围，否则争用就可能发生。这种类型的快速背对背传送对总线主设备是任选的，但必须能被目标设备解码。

第二种类型的快速背对背传送支持由所有可能的目标设备共同承担无争用的负担。状态寄存器中的快速背对背能力位，当且仅当该设备作为总线的目标设备且满足下面条件时，可以被硬件置为逻辑“1”(高电平)。

(1) 目标设备在传送前没有 IDLE 状态的情况下开始传送，不能错过总线主设备的开始信号和地址。换句话说，就是在连续的时钟周期中，目标设备必须能跟上总线状态直接由最后数据传送(FRAME#高，IRDY#低)转换到地址段(FRAME#低，IRDY#高)。注意在这两个传送中，或在其中之一中，目标设备可能被选中，也可以不被选中，但必须要无遗漏地跟踪总线状态。

(2) 目标设备必须避免 TRDY#、DEVSEL#和 STOP#信号发生冲突。如果目标设备不是使用可能的最快 DEVSEL#有效时间，基本上就可避免这种冲突发生。对于做零等待解码的目标设备，除下列两种情形之一外，该目标设备必须延迟这三个信号有效一个时钟周期。

- a. 当前总线传送前有一个总线 IDLE 状态。这就是说该传送不是一个快速背对背传送。
- b. 在上一次总线传送时，当前目标设备已驱动 DEVSEL#有效。这就是说该传送是一个与上次传送目标设备相同的快速背对背传送。

如果某目标设备不能提供上述两种保证，它就不能使用这一位，且在读状态寄存器时，自动返回零。

对于想完成由目标设备机制支持的快速背对背传送的总线主设备，在其配置空间的命令寄存器中有一个快速背对背允许位(该可读/写位只对作为总线主设备的设备有意义，而且是可选的)。当该位设置为“1”(高)，如果当前总线主设备所作的上一次传送是写传送，总线主设备可以无视哪个目标设备被寻址而开始快速背对背传送。如果该位被设置成“0”(低)或没用它，只有当总线主设备能保证新的传送的目标设备与上一次传送的目标设备相同时，它才能做快速背对背传送。在确认同一总线上所有目标设备的快速背对背传送置位后，由系统配置程序来设置这一位。注意，基于总线主设备的快速背对背传送机制不能象基于目标设备的机制那样，对不同的目标设备进行快速周期操作。

在所有其它条件下，总线主设备必须至少插入一个 IDLE 总线状态。(在不同总线主设备所进行的传送之间也总有至少一个 IDLE 总线状态)。注意，多口目标设备在快速背对背传送期间，当其被真正锁住时，将只锁它们自己。

注意，未加入快速背对背传送的单元不能仅用 IRDY#和 FRAME#就立刻区分出传送的界限(无 IDLE 状态)，在快速背对背传送期间，只有参与传送的目标设备和总线主设备才需要区分这些界限。当最后传送结束时，所有的单元都能观察到一个 IDLE 周期。然而，支持基于目标设备的机制的单元必须能区分所有 PCI 传送的完成并能识别所有的地址段。

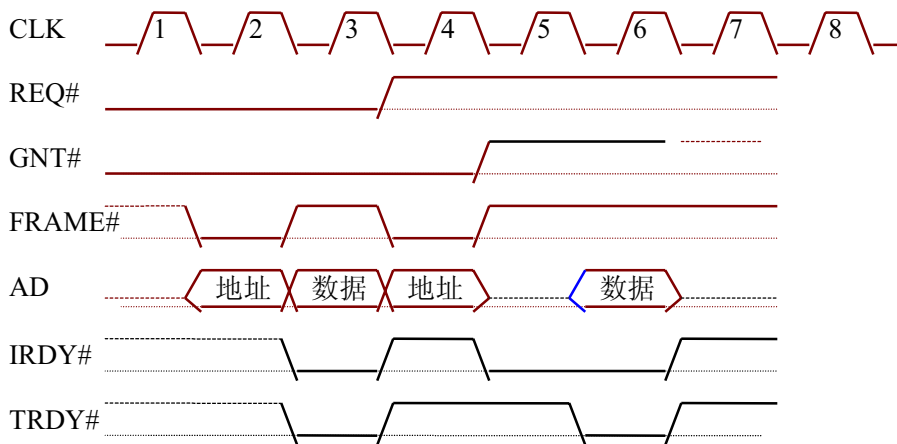


图 5.7 快速背对背操作的仲裁

在图 5.7 中，总线主设备在 clock3 完成第一次写操作，在 clock4 产生下一个传送的地址段。目标设备必须在当前数据传送完成后的那个时钟开始采用 FRAME#。在总线主设备可以选择支持这种功能时，目标设备必须能解码背对背操作。在使 DEVSEL#有效而确认对总线的拥有关系后，目标设备可以自由地重试这种要求。

提供该特性的最重要的好处就是对使用 PCI 总线进行的“低端”系统配置具有很好的运行性能而代价很小。建议所有新的目标设备或这种配置中可能使用这种性能的目标设备都使用这一性能。*** 在调试过程中在目标设备中设置，要仔细分析

5.12 锁定操作

PCI 提供一种锁定操作，这种锁定操作相应于**资源锁定**。当使用资源锁定时，即使有别的总线主设备拥有 LOCK#，也能对进行非锁定操作的单元进行处理，即它允许在锁定状态有非锁定操作的存在。这就允许将来的处理器作一种贯穿几个操作的、对非锁定操作无干扰的硬件锁定实时的传送诸如视频信号这样的数据。这种机制基于只锁定作为初始锁定操作的目标设备的 PCI 资源。该机制与现存用于锁定的软件完全兼容。

在资源锁定中，一个操作的锁定性由操作的目标设备来保证，而不是把其它单元从正在操作的总线上排斥掉。锁定的量被定义成对齐的 16 字节。对这个 16 字节块中的任何一个字节的锁定操作都将要锁定这个 16 字节块。总线主设备不能依赖锁定的 16 字节之外的任何地址。对目标设备的所定，最小是 16 字节，最大到整个资源。每个提供系统存储器的设备都必须有 LOCK#信号。特别是，如果一个设备采用了可锁定的存储器，那么它就必须用 LOCK#信号，并且保证完成存储器中的锁定操作。连接系统存储器的主桥也要用 LOCK#。

LOCK#信号说明一个锁定操作正在进行。GNT#的有效并不能保证对 LOCK#的控制。LOCK#的控制只能由它自己的协议与 GNT#相互配合后才能获得。对于兼容指令，仲裁器能有选择地将资源锁定转换成“总线所定”，其方式是允许拥有 LOCK#的单元对总线进行锁定操作，直到 LOCK#释放。

LOCK#的规则对总线主设备和目标设备都适用，PCI 上支持 LOCK#的目标设备必须遵循以下规则：

1. 所有支持锁定操作的 PCI 目标设备必须按地址去采样 LOCK#信号。当 LOCK#在地址段期间无效时，一个操作的目标设备锁定它自己。当 LOCK#在地址段期间有效时，当前被锁定的目标设备对所有传送都应之以 STOP#有效而 TRDY#无效。

2. 一旦锁定建立，该目标设备保持锁定直到它采样到 FRAME#及 LOCK#均无效或它发出目标设备失败信号时，就不再锁定自己了。

3. (一旦锁定建立) LOCK#的拥有者就要保证至少锁定资源中的 16 字节。对多口设备，这还要包括不在 PCI 上进行的操作。

如果一个操作的目标设备只能进行中速和低速解码，它就必须在地地址段期间锁住 LOCK#信号，以便在解码完成之后再决定该操作是否是锁定操作。如果目标设备采样 LOCK#直到它使 DEVSEL#有效，它就不能确定当前操作是一个锁定操作还是一个和锁定操作同时发生的操作。一个单元可以用存储“状态”来确定操作是锁定的，但这就要求在连续时钟中去锁住 LOCK#并加以比较才能决定是否是否锁定操作。

为允许对多口设备的别的操作，目标设备可以在地址段后的那个时钟再采样 LOCK#以确定设备是否真被锁定。如果 LOCK#在地址段期间无效而在地址段后的那个时钟又有效，那么该多口设备已被锁定并要对 PCI 总线主设备确保锁定。如果在地址段及其后的那个时钟，LOCK#都无效，该目标设备就可以对别的要求作出应答，且不锁定。当 LOCK#在地址段期间无效时，当前被锁定的目标设备才能接受请求。

注意，当 LOCK#有效时，仲裁器必须处于“公正”规则的几种状态下，否则就有可能发生活锁(Livelock)。

现存的不支持 PCI 锁定使用规则的软件有不能正常工作的可能性。推荐对支持 LOCK#并且能向后兼容现存软件的 PCI 驻留存储器(原来的系统存储器)实现资源锁定。

LOCK#由总线主设备驱动，在 PCI 上使用 LOCK#的总线主设备必须遵循以下规则：

1. 在一个锁定操作期间，一个总线主设备只能锁定一个唯一的资源。
2. 锁定不能跨越设备界限。
3. 在一个锁定操作期间，(对齐的)16 个字节是一个总线主设备所能计数的作为锁定的最大资源范围。对这 16 个字节中的任何部分的操作都必须锁定全部 16 字节。

4. 锁定操作的第一个传送必须是读传送。
5. LOCK#在地址段后的那个时钟必须有效并维持有效以保持控制。
6. 如果在数据段完成之前且锁定尚未建立，重试信号已发出，则 LOCK#必须释放。
7. 无论何时，如果传送由目标设备失败或由总线主设备失败终止，LOCK#必须释放。
8. 在连续锁定操作之间的至少一个 IDLE 状态中，LOCK#必须释放。

5.12.1 开始锁定操作

无论何时，LOCK#有效则总线主设备就标注 LOCK#忙；LOCK#和 FRAME#都无效时就标注 LOCK#不忙。

当某一单元需要锁定操作时，在发出 REQ#之前，它要检查内部 LOCK#的跟踪状态。如果 LOCK#忙，该单元延迟使 REQ#有效，直到 LOCK#不忙。在等待 GNT#时，总线主设备继续监视 LOCK#，如果 LOCK#又变成忙，总线主设备就使 REQ#无效，因为别的总线主设备已获得 LOCK#的控制权。

当允许总线主设备操作总线而 LOCK#又不忙时，对 LOCK#的拥有关系就成立。在当前传送已完成，且在总线上只有一个单元能驱动 LOCK#的情况下，该总线主设备就能顺利实现一个锁定操作。甚至当它们是当前总线主设备时，所有别的单元也不能驱动 LOCK#。

图 5.8 说明了一个锁定操作的开始。在地址段 LOCK#无效，以要求一个由读命令为开始。在地址段后的那个时钟，LOCK#必须有效即 clock3 以使目标设备保持在被锁定状态，而这种状态又允许当前的总线主设备保留 LOCK#的拥有关系直到当前传送结束。

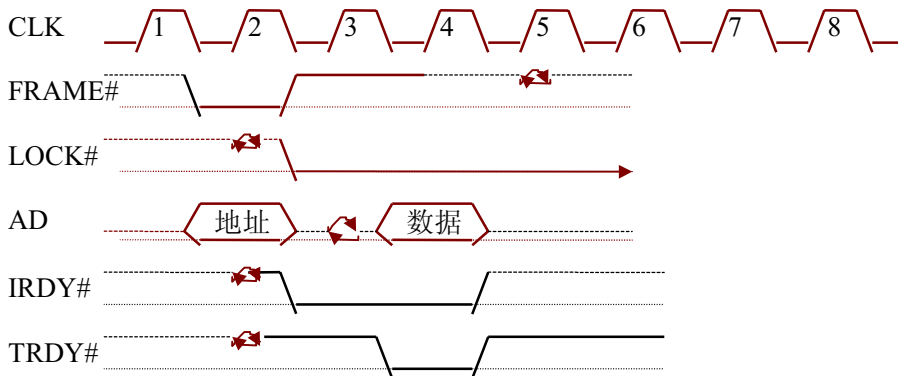


图 5.8 锁定操作开始

如果目标设备在没有完成第一个数据段时重试该传送，总线主设备不仅必须终止该传送，而且它还必须释放 LOCK#。只有在第一个传送的第一个数据段完成之后(读传送，IRDY#TRDY#有效)，一个锁定操作才在总线上建立。并且总线主设备要保持 LOCK#有效，直到锁定操作完成或一个错误(总线主设备或目标设备失败)引起的一个提前终止。即使在锁定操作已建立的情况下，目标设备重试或解除连接也属于正常终止操作。如果总线主设备被目标设备解除连接或重试终止，则说明目标设备正处于忙状态，没有能力完成所要求的数据段。在目标设备处于不忙状态且继续通过拒绝别的操作来遵守锁定规则时，它将接受该操作。总线主设备继续控制 LOCK#。当 LOCK#有效时，对 PCI 上非锁定目标设备的非锁定操作也是可以进行的。当锁定操作完成后，LOCK#就变成无效，别的总线主设备就可以争取该信号的拥有关系。

5.12.2 继续锁定操作

图 5.13 为总线主设备继续一个锁定操作，然而，这个锁定操作可能完成所示锁定操作，也可能完不成。当该总线主设备获得对总线的操作权后，它对预先锁定的目标设备开始作

另一个锁定操作。LOCK#在地址段期间无效以重建锁定。已被所定的设备接受并应答这个请求。在 clock3, LOCK#有效以保持目标设备处于锁定状态并允许当前的总线主设备对 LOCK#的拥有关系直到当前传送结束。

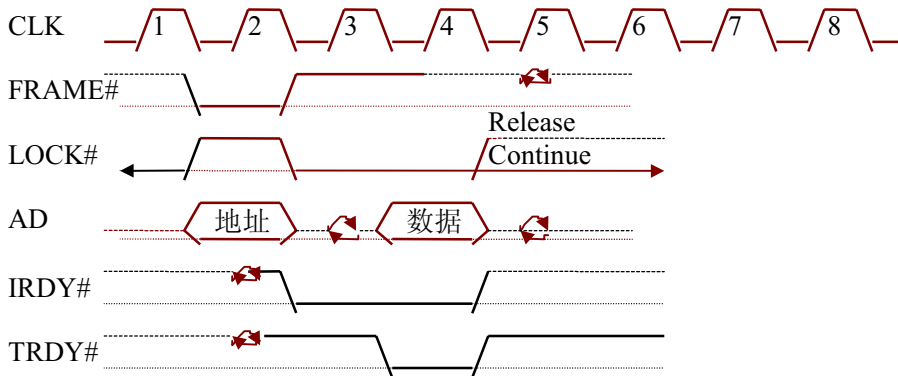


图 5.13 继续锁定操作

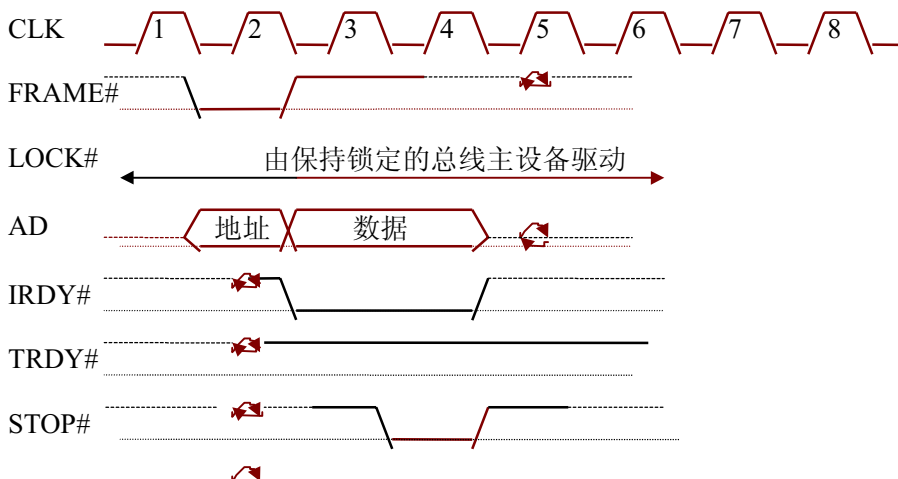
当总线主设备继续作锁定操作时，它维持使 LOCK#有效。当该总线主设备完成该锁定操作时，在最后一个数据段之后(clock5)，它使 LOCK#无效。

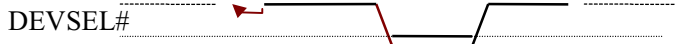
5.12.3 对锁定操作进行非锁定操作

图 5.10 示出一个总线主设备试图对锁定目标设备作非锁定操作。当 LOCK#在地址段有效，且目标设备被锁定，这就说明了当前被锁定的目标设备对所有传送都应之以 STOP#有效而 TRDY#无效，因而对锁定目标设备作非锁定操作没有结果。一个被锁定的目标设备在决定它是否作出应答时忽略 LOCK#。由于 LOCK#和 FRAME#在地址期间有效，未锁定的目标设备不会进入锁定状态。

5.12.4 完成锁定操作

在锁定操作的最后传送期间，LOCK#无效，于是目标设备将接受请求，然后又重新有效直到锁定操作成功地终止。在锁定操作完成后，总线主设备可以在任何时候使 LOCK#无效。然而，建议(但不是要求)在锁定操作的最后数据段完成后，LOCK#和 IRDY#一起无效。在别的时候释放 LOCK#可能会引起一个顺序传送被不必要的重试终止。只要 LOCK#和 FRAME#都无效，已锁定的单元就不会再锁定自己。如果某一总线主设备想在总线上作两个独立的锁定操作，那么它必须确保在 FRAME#和 LOCK#均无效的两个操作之间，至少有一个时钟。这样就能保证任何被第一个锁定的目标设备在第二个操作开始之前释放。





DEVSEL#

图 5.10 对锁定单元的非锁定操作

5.12.5 对 LOCK#和高速缓存器的连续回写的支持

如前所述的资源锁定在使用高速缓存器的连续回写时有发生死锁的可能性。在支持高速缓存回写并且使用了一个已完成锁定的资源时，如果软件允许锁定跨越高速缓存线界限，则死锁就可能要发生。这种可能性的一个例子是锁定跨越高速缓存线 n 和 $n+1$ 时。缓存线 $n+1$ 已被高速缓存器改变。某一总线主设备通过读高速缓存线 n 而建立了一个锁定。该锁定操作通过读高速缓存线 $n+1$ 而继续。对 $n+1$ 的监视导致了 HITM，它说明已监测到一条被改变的高速缓存线。因该目标设备只能接受从 LOCK#拥有者处来的操作，改变后的高速缓存线的回写就要失败。这就会导致死锁，因为只有改变后的高速缓存线之回写完成之后才能进行读，而只有 LOCK#释放后才能进行回写。

这种死锁可以通过要求支持可高速缓存回写存储器的目标设备在被锁定后仍允许回写而得以避免。

目标设备可以通过地址段期间(或在地址排队情况下 SDONE 有效后的那个时钟)SDONE 和 SBO#的状态来区分回写和别的写传送。在地址段期间，如已说明 CLEAN，则当前传送要么是 CLEAN，要么就是回写。在地址段期间，从 STANDBY 到 CLEAN 的变化说明某条高速缓存线的复原。在地址段期间，从 HITM 到 CLEAN 的变化说明回写由监视所引起，由监视引起的回写的目标设备就算已被锁定，也要接受该回写。该目标设备可以有选择地接受高速缓存线的复原，但如它已被锁定，则不要求这样做。当目标设备锁定后，所有别的操作都被它用重试终止(主意，由监视引起的回写的目标设备不能终止这种传送，直到高速缓存线已传送，这意味着对监视引起的回写不能作重试和解除连接)。

5.12.6 完整的总线锁定

通过在 LOCK#有效时，仲裁器不允许别的单元操作总线，可将 PCI 资源锁定转换成完整的总线锁定。在完整的总线锁定情况下，当一个锁定操作正在进行时，别的总线锁定不能进行。若锁定顺序中第一个操作要重试，那么总线主设备就必须使它的 REQ#和 LOCK#无效。若第一个操作正常完成，那么完整的总线锁定就建立了，并且仲裁器就不会再允许别的单元操作总线。如果完整的总线锁定建立后，仲裁器又允许别的单元操作总线，那么仲裁器就必须改变别的单元以便近似获得完整总线锁定。完整的总线锁定对系统性能有很大影响，尤其是视频子系统。

与完整的资源锁定及回写高速缓存类似，完整的总线锁定也存在着死锁的可能性。支持完整的总线锁定的仲裁器，在一个锁定已进行后，必须允许高速缓存器对总线进行操作，以完成由于对一改变的高速缓存线的监视而引起的回写(要求目标设备在锁定后接受回写，因为它不能说出正在使用的是完整的总线锁定还是资源锁定)。

5.13 PCI 协议对 Cache 的支持

5.13.1 Cache 的作用

近年来，由于 VLSI、RSIC 和高速逻辑设计技术的飞速发展，微处理器(MPU)的工作主频和运算速度成倍增长。工作主频处于 100~400MHz(时钟周期为 10~2.5ns)，指令执行速度高达 12 亿次。而 DRAM 存储器读/写周期约 60ns，如果微处理器直接访问 DRAM，由于两者速度差异较大，降低微处理器的速度。解决的办法是在微处理器的寄存器与 DRAM 存储器间设置高速缓存 Cache，以形成三个层次。三部分的速度水平大致如下：MPU 10~2.5ns；Cache 10ns 左右；DRAM 60ns 左右。Cache 由 SRAM 组成，可在 MPU 内部或片外

实现。PCI 高速缓存支持能力在 PCI 存储器单元和桥路(或高速缓存单元)之间提供了一种标准接口, 这种接口允许使用对高速缓存连接的监视机制。它改进了性能, 同时也引入了下述问题: 1).MPU 读/写数据的源或目的是在 Cache 还是在 DRAM 中? 2).访问共享存储器时, 其中的内容与 Cache 中的内容是否一致, 即 Cache Coherence 问题。PCI 协议通过提供对 Cache 的支持解决了上述问题。

Cache 数据一致性的基本含义是: 在 MPU—Cache—Memory(简记为 Mem, 下同)的系统中, Cache 将保留 Mem 的部分副本, 并依据某种策略, 如 Write Back, 对副本进行修改。由于 Mem 是共享的, 必须保证 Cache 以外的任何访问源 R/W 共享 Cacheable 存储器时, 能读到 Cache 与 Mem 两者中最新的数据; 在写操作时, 能使两者的数据相同或只保留 Mem 一个副本。

在 X86 微机系统中, 一级 Cache(L1—Cache)在 MPU 片内, CACHE/BRIDGE 是二级 Cache(L2—Cache)和系统桥路控制器的组合体。Mem 可分为两类: 一类为系统 Mem, 它不直接连在 PCI 总线上, 总线主设备经过桥路访问它。在访问过程中, 桥路自动完成对 Cache 数据一致性的检查与处理, 其过程对总线主设备透明。简要过程如下: 桥路接收到总线主设备的 Read Mem 命令后, 同时启动 Mem Read 和检查 MPU 内部的 L1—Cache(假定 L2—Cache 不存在)是否有相同的副本。如果命中且该副本已被修改, 则将 Cache 的副本回写到 Mem 中, 同时改变 Cache 副本的状态为非修改状态。桥路从回写的数据中取出所需的部分送给总线主设备, 到此读命令完成。对写命令, 桥路将数据直接写入 Mem 同时作废或修改 Cache 中的相应副本。第二类为 PCI—Mem 它独立地直接连在 PCI 总线上, 可被 MPU 和一个或多个总线主设备共享。当 PCI—Mem 的内容是可被高速缓存时, 称其为 Cacheable, 否则为 non—Cacheable。。

5.13.2 Cache 的组织和访问

Cache 的组织如图 5.11 所示;

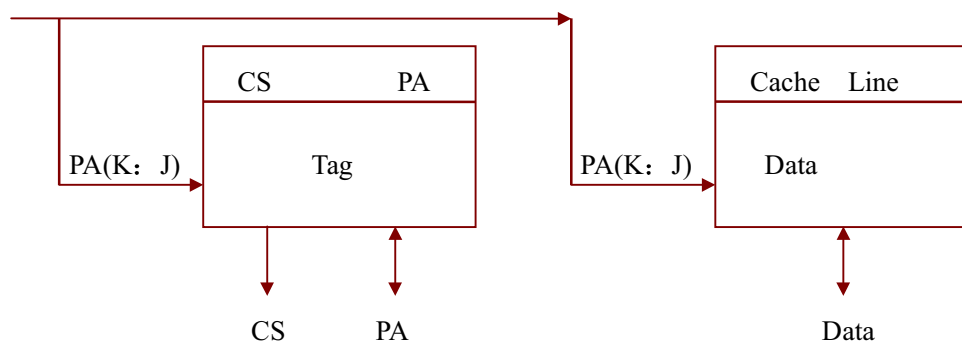


图 5.11 Cache 的组织

Cache 以 Cache Line 为单位进行组织。每个 Line 通常有 $8 \times n\text{Byte}(n=1,2,\dots)$ 。除数据存储单元外, 还配有相应的 Tag, 它由此 Line 对应的物理地址(PA)和状态(CS)组成。地址位 K 由 Cache 容量确定, 如 2^K , J 由 Line 的大小决定, 例如 $J=4$, 则 Line 为 16 字节。每个 Line 或者是有效的, 或者是无效的, 不存在部分有效的 Cache Line, 它有如下状态:

- Invalid: 数据无效;
- Clean: 数据有效但未被修改;
- Dirty: 数据有效, 但已被修改, 与 Mem 中相应的数据不一致。

对 Cache 的访问是先用物理地址 PA(K: J)(或虚地址)访问 Data 和 Tag。然后, 比较 PA

和检查 CS，以确定所需的 Cache Line 是否在 Cache 中。如果 CS 为 Invalid 或 PA 不匹配则称为未命中 Miss，反之，为命中 Hit。

由于 Cache 中的内容是 Mem 内容的副本，依据 MPU 对 Cache 写操作是否修改相应 Mem 中的副本，提出了两个广泛使用的 Cache 访问策略：

1. 通写(Write-through)

MPU 写 Cache 时，同时修改相应的 Mem 单元，使两者相对应的 Line 数据相同。当这样以来，写 Cache 时间与写 Mem 时间相同，无加速作用。

2 回写(Write-back)

MPU 写 Cache 时，仅修改 Cache，此时 Cache Line 的内容与 Mem 中相应 Line 的不一致。这时，读/写 Cache 的时间都比读/写 Mem 的时间短，有较高的加速作用。

两种策略下，MPU、PCI 总线主设备访问共享 Cacheable PCI-Mem 的操作比较如表 5.1 所示：

表 5.1

操作	Write-through	Write-back
MPU-Write-Mem	Hit: 修改 Cache 和 Mem Miss: 修改 Mem	Hit: 修改 Cache Miss: 修改 Mem
MPU-Read-Mem	Hit: 从 Cache 中读数据 Miss: 从 Mem 中读一个 Cache Line, 写入 Cache, 同时将 MPU 所需数据送回	Hit: 从 Cache 中读数据 Miss: 从 Mem 中读一个 Cache Line, 写入 Cache, 同时将 MPU 所需数据送回
PCI 总线主设备 Write-Mem	写 Mem, 检查 Cache, 若 Hit, 则该 Cache Line 将被更新	桥路检查 Cache 若 Miss 则直接写 Mem; 若 Hit-Clean 则写 Mem 并失效 Cache Line; 若 Hit-Dirty, 桥路强制 PCI-Mem 终止此次写操作, 然后回写已修改的 Line 到 Mem, 失效此 Line, 回写完成后, 允许 PCI 总线主设备重新启动被终止的写操作。
PCI 总线主设备 Read-Mem	直接从 Mem 读数。	桥路检查 Cache 若 Miss 或 Hit-Clean 则直接从 Mem 读数; 若 Hit-Dirty, 桥路强制 PCI-Mem 终止此次读操作, 然后回写已修改的 Line 到 Mem, 改变此 Line 的状态为 Clean, 回写完成后, 允许 PCI 总线主设备重新启动被终止的读操作。

可见，在 Write-through 策略下，PCI 总线主设备访问共享 Cacheable PCI-Mem 时，按正常的存储器操作完成，对 Cache 的处理由桥路完成，无需总线主设备感知，为了使总线主设备对存储器的操作和桥路对高速缓存的处理同步完成，桥路需要提供一条信号线通知存储器，桥路何时完成 Cache 操作，以便作为目标设备的 PCI 存储器在 Cache 操作完成后执行存储器读写/操作。

而在 Write-back 策略下，由于在 Hit-Dirty 条件下，需将 Cache 中被修改的 Cache Line 回写到 PCI 存储器中，所以，除了有与 Write-through 中相同的指示信号外，还需增加指示 Cache 回写的信号线，这便是 PCI 协议 Cache 的两个信号线 SDONE 和 SBO#。

5.13.3 PCI 协议下 Cache 的状态

SDONE 为 Cache 检查完成标志，SBO#为是否命中了已被修改的 Line 标志。它们在桥

路/高速缓存和所要求的存储器目标设备之间传递高速缓存状态信息。任何支持可高速缓存的 PCI 目标设备都必须监视高速缓存支持引脚并作出适当的应答。不可高速缓存的目标设备可以不理睬 SDONE 和 SBO#, 这可以节省一点操作延迟。

因为 PCI 允许长度不定的猝发传送, 作这种操作的可高速缓存的目标设备, 在对超过一条高速缓存线界线的存储器范围进行操作时, 必须解除连接。这意味这任何可高速缓存的目标设备必须知道高速缓存线范围, 或是利用高速缓存线寄存器, 或是由硬件设置此参数。如果允许作跨越高速缓存线界线的猝发传送, 高速缓存相关性就可能“崩溃”(相应地, 桥路/高速缓存能监视该传送并为监视产生下一个高速缓存线地址)。

在此定义的 Cache 状态是桥路依据 PCI 总线主设备启动的存储器读/写操作, 对 Cache 进行 Hit 及状态检查后在 PCI 总线上报告的检查结果(只作为输出)。

SDONE	SBO#	状 态
0	×	STANDBY
1	1	CLEAN
1	0	HITM

在 Write-back 策略下, 各状态的具体含义如下:

STANDBY 表示下述三个状态之一:

- (1) 高速缓存器当前未监视地址但已做好监视;
- (2) 正对接收(锁存)的第一个 Mem 访问地址作 Cache 检查。同时, 可接收第二个地址;
- (3) 已接收到两个地址, 正对第一个地址作 Cache 检查, 完成后, 如果第二个地址仍有效, 则对第二个地址作 Cache 检查。

CLEAN 说明没有高速缓存冲突且存储器操作可正常完成:

- (1) 读/写操作没有命中;
- (2) 读/写操作命中了未修改的 Line;
- (3) Memory Write and Invalidate 命令操作命中了已修改的 Line。

由高速缓存写命令或存储器写命令所引起的对一条未变化的高速缓存线的回写是不要求的, 当高速缓存是当前总线主设备且正在回写一条变化的高速缓存线时, 它将在地址段期间发出 CLEAN 信号。说明监视碰到了一条变化了的高速缓存线, 且要求高速缓存把回写这条变化的高速缓存线作为它的下一个操作。高速缓存将保持这种状态直到回写发生。当 HITM 在总线上出现时, 所有其它可高速缓存的操作都由存储控制器以重试来终止这次传送, 允许回写产生, 然后由重试终止的那个单元重新请求传送。在对变化的高速缓存线的回写过程中, 高速缓存在地址段由 HITM 转到 CLEAN。

为能有效地使用 PCI 总线, 可高速缓存控制器(相应于存储器控制器)和高速缓存/桥路都要跟踪总线操作。为减少高速缓存传送的重试终止, 参与可高速缓存传送的单元要能锁存两个地址, 连在 PCI 总线上的 Mem 都不知道其它 Mem 是不是可高速缓存的, 这便要求每个 Mem 控制器实时监控 SDONE 和 SBO#信号, 以掌握 Cache 检查的结果, 特别是自身已被 PCI 总线主设备访问时, 确定本次读写操作是终止(HITM)还是继续完成。由于不可高速缓存的存储控制器不关心 SDONE 和 SBO#, 一旦被启动, 便义无反顾地执行到完成。另外, 若只在桥路和可高速缓存的存储控制器中只设一个用于 Cache 检查(Bridge)和监视(Mem)的地址锁存, 则在不可高速缓存的存储器和可高速缓存的存储器访问交替进行使, 将对可高速缓存的存储器的访问无法完成, 其基本过程如下:

首先, 一个不可高速缓存的存储器读/写被启动, 桥路锁存此次读/写的地址, 并开始对 Cache 的检查。由于可不高高速缓存的存储器对 Cache 检查结果不关心, 在结果出现前, 完成了读/写操作, 释放 PCI 总线。随后, 一个可高速缓存的存储器读/写被启动, 因只能

锁存单个地址，且桥路正对前一个地址作 Cache 检查，所以刚启动的可高速缓存的存储器读/写被强行终止。假定在终止过程中，检查结果报出，地址锁存器控。若此时，PCI 仲裁器准许启动另一个不可高速缓存的存储器操作，则将重复上述过程。最终将使可高速缓存的存储器读/写操作无法完成。解决的办法是可高速缓存的存储器控制器和桥路可分别锁存两个访问地址。

5.13.4 Cache 检查状态的转换关系：

1. Write-back 策略下的状态转换

(1) STANDBY→CLEAN→[CLEAN]→STANDY

这是正常情况，高速缓存保持 STANDBY 状态，直到监视完成，然后发出 CLEAN 信号说明传送将正常完成。当对第一个传送的监视已完成而第二个地址尚未到达时，高速缓存转换到 STANDBY 状态。如果第二个地址已锁存且高速缓存是第二个传送的总线主设备时，高速缓存将在两个连续时钟内发出 CLEAN 信号，条件是传送是对高速缓存线的回写，或知道监视是 CLEAN，否则高速缓存转换到 STANDBY。

(2) STANDBY→HITM→CLEAN→[CLEAN]→STANDY

这是在监视中监测到了一条变化了的高速缓存线的情况，一旦高速缓存发出 HITM 信号，它将保持这种状态直到变化的高速缓存线被回写。高速缓存变换到 CLEAN 说明它正在进行回写。在 CLEAN 之后，高速缓存将发出 STANDBY 信号，说明它已准备好监视一个新的地址。如果高速缓存是第二次传送的总线主设备，而这次传送又是对高速缓存的回写，或是知道监视是 CLEAN 时，它将继续发出 CLEAN 信号，否则，它转换到 STANDBY。

2. Write-through 策略下的状态转换

STANDBY→CLEAN→[CLEAN]→STANDY

这与回写高速缓存大致一样，只是不用 SBO#信号。存储器控制器监视总线，每当 SDONE 有效，存储器就能允许别的可高速缓存的传送完成。如果高速缓存是第二次传送的总线主设备，而这次传送又是对高速缓存的回写，或是知道监视是 CLEAN 时，高速缓存将继续发出 CLEAN 信号，否则它转换到 STANDBY。建议能高速缓存的目标设备采用 SBO#和 SDONE。

对于每次在总线上出现的 FRAME#，高速缓存在它已监视地址后，使 SDONE 有效。若 FRAME#有效两次而 SDONE 无一次有效，如果第二个传送是可高速缓存的，它就不能完成。如果第二个操作是可高速缓存的存储器控制器必须插入等待状态直到上一个监视完成(SDONE 有效)。如果第二个操作是不可高速缓存的，该操作能完成并且高速缓存将不监视地址。这时，只有一个地址未完成。

5.13.5 时序关系说明

可高速缓存的存储器读/写被启动后，目标设备总是发出 TRDY#无效，等待桥路发出 Cache 检查结果。据此来决定是终止还是完成此次操作。而不可高速缓存的存储器读/写操作则不管 SDONE 和 SBO#，按正常时序完成操作。以下各图中假定时钟 1 时总线从 IDLE 状态开始。

碰到变化的高速缓存线并随后进行回写的时序关系如图 5.12 所示

— 在时钟 2 地址锁存，开始第一个传送，目标设备保持 TRDY#(插入等待状态)直到监视完成，高速缓存在时钟 4 使 SDONE 和 SBO#都有效，说明监视碰到了变化的高速缓存线(一旦 SBO#有效，就必须保持有效直到 SDONE 有效)。因为传送的目标设备是可高速缓存的，所以在时钟 5，它使 STOP#有效，从而终止传送。这样就允许发出 HITM 信号的高速缓存

回写变化的高速缓存线到存储器。在读传送中，如 HITM 出现，缓存控制器必须使 AD 线三态。当总线上出现 HITM 信号时，所有对可高速缓存的目标设备的传送都要由重试终止。

虚线说明自监视在第一次传送中被标志以来，不可高速缓存传送有可能完成，可高速缓存传送也可能会开始，但因 HITM 信号已发出，故必须用重试来终止。回写传送发生在时钟 A，在地址段高速缓存由 HITM 到 CLEAN 的转变向存储器表明监视回写已开始，且要求它接收这条高速缓存线上的所有数据(如果存储控制器不能完成这次传送，它必须插入等待状态直到它能完成。这种情况只有在可高速缓存的目标设备有内部冲突时才会发生)。在回写期间，高速缓存和存储控制器都可以插入等待状态。在时钟 B 高速缓存由 CLEAN 到 STANDBY 的转变表明高速缓存可接收下一个地址。回写完成后，总线返回到正常操作，可高速缓存传送将继续进行。

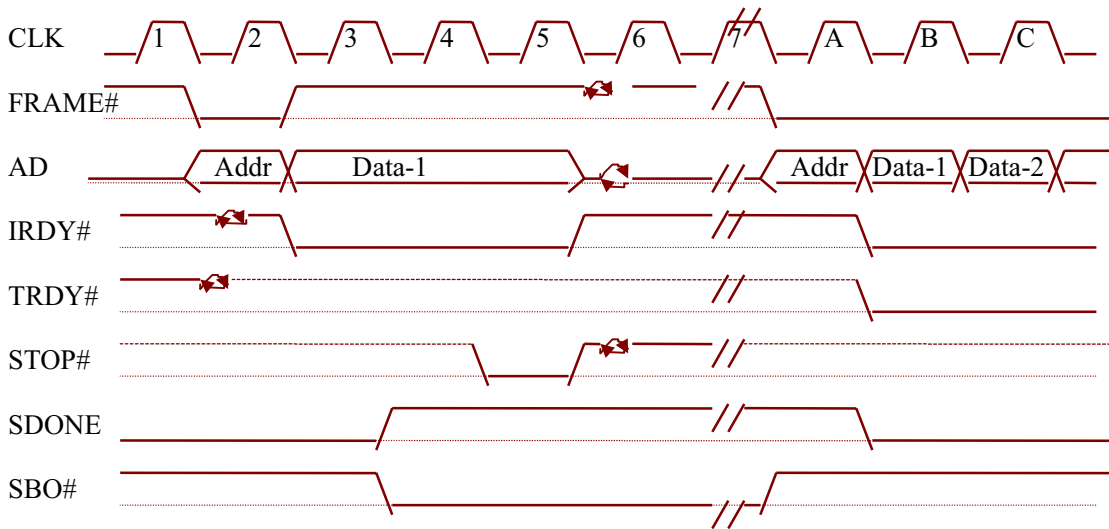


图-5.12 紧随回写操作的时序关系

对于高速缓存写(Memory Write and Invalidate)命令，高速缓存在处理该命令上有几种选择。因为总线主设备保证高速缓存线上的每个字节都将要改变，Mem 不等检查结果，按正常写操作时序完成。桥路失效相应 Cache Line，高速缓存只是简单地发出 CLEAN 信号，甚至在碰到变化的高速缓存线时也是这样。

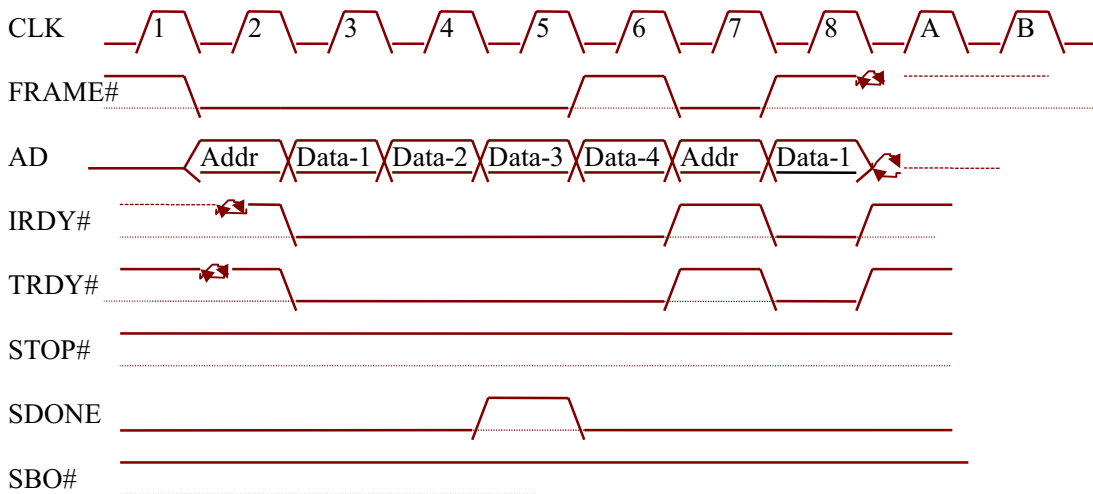
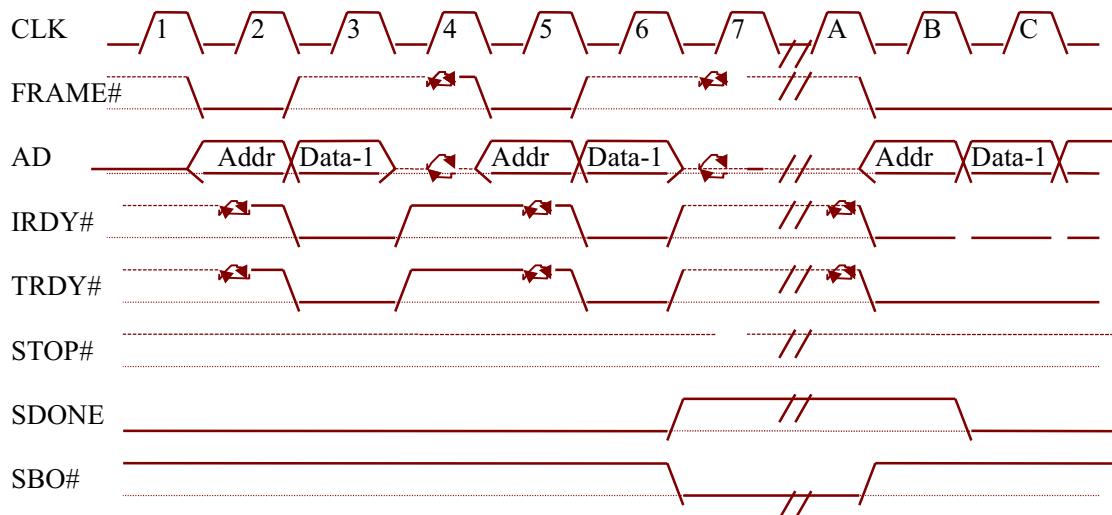


图-5.13 高速缓存写命令

在图 5.13 中，高速缓存在时钟 5 发出 CLEAN 信号，说明要么是监视错过或碰到一条变化的高速缓存线。一旦高速缓存表明了 CLEAN，它就要准备监视出现在总线上的下一个地址。如果 SBO#在时钟 5 有效，就表明监视碰到了一条变化的高速缓存线并将回写。高速缓存可以象对待特别的命令一样对待高速缓存写命令，并允许总线上出现 HITM 条件(这个回写在总线上引起了一个没要求的额外传送)。高速缓存会在说明监视结果之前花费固定数量的时钟去对待 TRDY#有效。如果在结果出现之前 TRDY#有效，建议高速缓存放弃这条高速缓存线并发出 CLEAN 信号。如果 TRDY#仍未有效，高速缓存就继续提供监视结果。然而，等待 TRDY#的时间必须是恒定的，因为存储控制器在继续传送之前，通常要等 SDONE 有效。



图—5.14 数据传送—碰到变化的高速缓存线并随即回写

在图—5.14 中，第一个传送开始于时钟 2 并且结束于时钟 3。当对第一个传送的监视正在进行时，另一个传送在时钟 5 开始于时钟 6 结束。如果第二个传送完成时对第一个传送的监视仍在进行，那么第二个传送就是不可高速缓存的传送。在时钟 7，第一个传送监视完成。一旦 FRAME#有效，且一个监视正在进行，SDONE 和 SBO#的状态只对第一个地址有意义，直到 SDONE 有效。一旦 SDONE 有效，那么下一次再有效时就只适应于第二个传送了，如果在上图中 SDONE 是在时钟 5 有效而不是在时钟 7 有效，那么监视的结果对第二次传送没有影响，即便它在第二次传送期间发出。

为减少可高速缓存的存储器写操作等待 Cache 检查结果的时间，改善性能，允许短数据写(如单个 32 位数据)在结果出现前完成。这具有“赌”的性质，要求存储控制器能保留刚完成的写操作的地址和数据，以便在 HITM 时，对回写的数据依据刚完成的写操作的数据进行修改。时序关系略。

当总线上呈现 HITM 时，仲裁器就要遵循一定的算法顺序，否则就会发生活锁，当两个高优先级的单元正在操作可高速缓存的存储器，而使具有变化的高速缓存线的高速缓存无法完成回写时，就会发生活锁。当总线上呈现 HITM 时，所有可高速缓存的传送都要由重试来终止。

建议当系统中有高速缓存时，仲裁器可选择将高速缓存的 REQ#接到固定输入，以便当总线呈现 HITM 时，它的优先级能得到提高。这样就确保回写时，被重试终止的可高速缓存的传送保持最小且延迟也会缩小。

当系统中使用了高速缓存时(尤其是回写高速缓存)，有目标设备所造成的延迟必须增加到它对变化的高速缓存线所作的回写的时间中去。这个增加的值依赖何时回写能操作总

线的仲裁算法。

5.14 其它总线锁定

5.14.1 设备选择

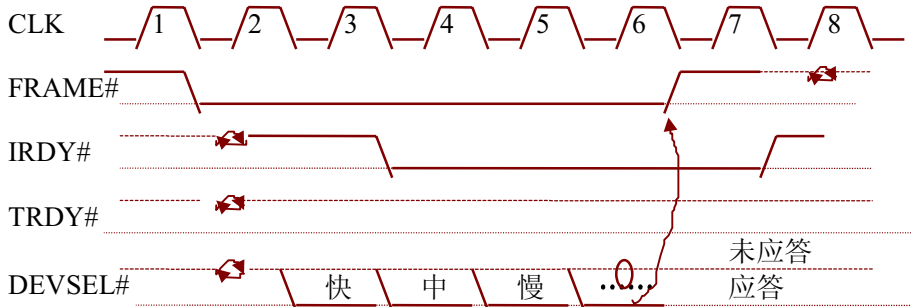


图 5.15 DEVSSEL#有效

如图 5.15, DEVSSEL#由当前传送的目标设备驱动。DEVSSEL#可以在地址段后 1、2、3 个时钟内驱动, 并且要和 PCI 配置空间状态寄存器中的说明一致。如果在 FRAME#有效后 3 个时钟内没有单元驱动 DEVSSEL#有效, 则做反解码的单元就响应请求并使其 DEVSSEL#有效。如果该系统没有反解码单元, 总线主设备就不能知道有效的 DEVSSEL#, 也无法利用总线主设备失败机制去终止这次传送。目标设备最好能在 FRAME#有效后 1 到 2 个时钟内完成解码并使 DEVSSEL#有效。

目标设备必须在开放其 TRDY#、STOP#或数据(读)时或在此之前使 DEVSSEL#有效。除目标设备失败外, 在所有其它情况下, 一旦 DEVSSEL#有效, 它必须保持 DEVSSEL#有效直到 FRAME#无效(IRDY#无效)且完成最后的数据段。对正常的总线主设备终止, DEVSSEL#必须与 TRDY#同时有效。

目标设备在驱动/有效 DEVSSEL#或别的任何目标设备应答信号之前, 必须做完全解码。在完全解码之前驱动 DEVSSEL#, 然后再在别的总线周期去解码是错误的。在非配置命令中, 目标设备在 DEVSSEL#有效之前必须用 FRAME#去开放 AD 线。在配置命令中, 目标设备在 DEVSSEL#有效之前, 必须用 FRAME#去开放 IDSEL 和 AD[1..0]。

如果第一个操作映射到某一目标设备地址范围, 它就使 DEVSSEL#有效以确认该操作。但如果总线主设备想作跨越资源界限的猝发, 该目标设备就要申请解除连接。

当某一目标设备确认一个 I/O 操作, 并且字节允许表明有一个或几个要操作的字节位于该目标设备的地址范围之外, 它就必须发出目标设备失败信号。为了解决这类 I/O 问题, 反解码设备(扩展总线桥)应作下列事情之一:

- 对不同设备公用的公共双字的地址做正极性解码, 并用字节允许去检测此问题, 然后发出目标设备失败信号。
- 将全部操作移到扩展总线上, 放弃在该总线上不能进行的那部分操作(这种情况仅发生在第一个寻址的目标设备在扩展总线上而其它则在 PCI 上)。

5.14.2 特殊周期

特殊周期命令在 PCI 总线上提供了一种简单的信息传播机制, 它可以利用 PCI 单元之间的逻辑边带信号传送, 条件是这样的信号传送不要求物理信号的精确时间和同步。

特殊周期命令有时可包含可选的、基于数据的信息, PCI 定序器(Sequencer)本身并不对它进行译码, 只是在必要时让它通过到与 PCI 定序器有联络的硬件应用环境中去。

特殊周期命令也和和其它别的命令一样包含地址段和数据段, 地址段以 FRAME#有效来

起始, 在 FRAME#和 IRDY#都无效时地址段完成。地址段除命令域 C/BE[3..0]#=0001(特殊周期) 外, 不包含别的信息。没有明确的目标设备地址, 但 AD[31..00]都要驱动到稳定电平并保证奇偶校验正确, PCI 单元将不使 DEVSEL#有效来应答它, 因而要传播到所有单元。每个接收单元都要检测这些信息对它是否可用。这意味着在这种传送中没有任何形式的目标设备联络, 并且反解码桥路不得将这种总线操作传到它的二级总线上去。特殊周期命令将不会通过桥路。

在数据段期间, C/BE[3..0]#有效, AD[15..00]包含信息编码, AD[31..16]为可选的数据域编码, 特殊周期命令的总线主设备能插入等待状态但目标设备不能(因为没有目标设备)。信息和其所依赖的数据仅在 IRDY#有效的第一个时钟有效。其中包含的信息和顺序数据段的时间与信息有关。

PCI 总线定序器象别的命令一样起始这种命令, 并用总线主设备失败来终止它。硬件应用提供象其它任何别的命令一样的全部信息并启动总线定序器。当定序器反映该操作已由总线主设备失败终止时, 硬件应用环境就知道该操作已完成。在这种情况下, 配置状态寄存器中“收到总线主设备失败”位不能被置位。特殊周期命令最快可在 5 个时钟能完成, 在下一个操作前, 要求增加一个转换周期。所以, 从一个特殊周期开始到别的周期开始共需要 6 个 PCI 时钟。

当前的特殊周期信息编码如下表所示:

信息编码(AD[15..00])	信息类型	含 义
0000H	SHUTDOWN	是一种广播信息, 说明该处理器正处于关闭模式
0001H	HALT	是从处理器来的关闭信息, 说明它已执行了一条“停机”指令
0002H	X86 特殊配置	是一种 X86 处理器和芯片集所用的通用编码。 AD[31..16]规定特殊周期信息的特别意义, 这些特别意义有 Intel 公司定义并可在产品特别文件中找到。
0003H~FFFFH	保留	

5.14.3 地址/数据分步

某单元使适当的信号在几个时钟内分别有效的能力称为分步, 这种概念允许带有“弱”输出缓存的单元驱动一组信号, 在几个时钟内到有效状态, 由此而减少由每个缓存器所引起的地电流负载。另一种方法允许带有“强”输出缓存的单元在几个时钟的每个时钟沿驱动它的子系统直到这些附属部分全部都被驱动, 以此来减少必须同时切换的信号的数量。

任一应用都允许某一单元因价格而综合考虑性能(减少电源/地引脚)。当使用连续分步方法时, 要注意避免在每个时钟沿都要采样的关键控制信号和在每个时钟沿可能传送的分步信号之间的相互耦合。对性能要求苛刻的外设, 应小心地使用这种“承诺”。

分步只允许在 AD[31..00]、AD[63..32]、PAR、PAR64#和 IDSEL 引脚上进行, 因为它们总是由控制信号选中, 这些信号只有在选中的时钟沿才被认为有效。在地址段, AD 线有 FRAME#使其选中, 在数据段有 IRDY#或 TRDY#(根据数据被传送的方向)。PAR 在 AD 选中后每个时钟沿都隐含地选中, IDSEL 由 FRAME#和配置命令解码的组合去选中。

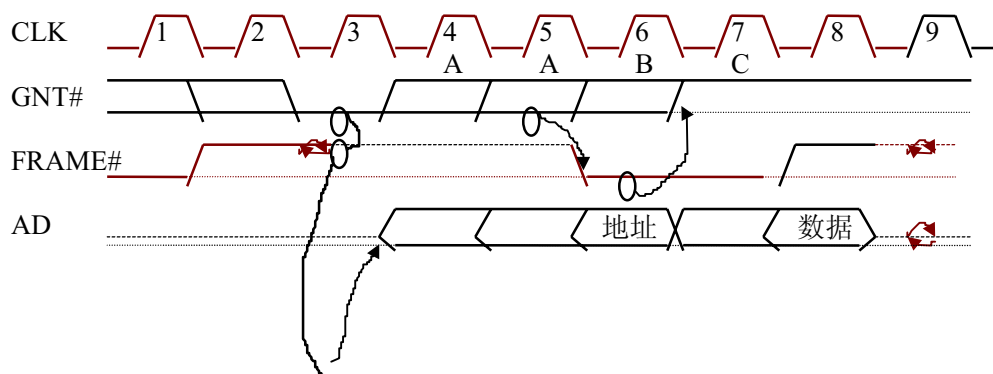




图 5.16 地址分步

图 5.16 所示为总线主设备延迟发出 FRAME#直到它成功地驱动了全部 AD 线，一旦总线主设备拥有了总线，而且此时总线处于 IDLE 状态，它就能且应该驱动 AD 线和 C/BE# 线。但它也可以在使 FRAME#有效之前花费多个时钟去驱动有效地址。然而，延迟发出 FRAME#有效总线主设备有失去总线拥有权的危险。如果在注有 A 的那个时钟沿 GNT#无效，该总线主设备就要立刻使它的信号处于三态，因为仲裁器已允许别的单元操作总线(这个新的总线主设备应该是优先级较高的)。如果在标有 B 或 C 的那些时钟沿 GNT#无效，FRAME#将有效而且传送继续进行。

在配置地址空间操作时，要求额外作设备选择解码，并通过 IDSEL 引脚发出信号到 PCI 设备，IDSEL 引脚如何准确地驱动由主/存储器桥路或系统设计者实现，设计上已允许该信号连接到在配置操作中无别的用途的高 21 位地址线的任何一条上，就能选中 21 个不同的设备，这种方法增加了 AD 线上的额外负担，可以通过适当的阻性耦合来减轻它，这又使得 IDSEL 线上的转换率非常慢，因而需要在 FRAME#之前预驱动地址总线几个时钟。以保证 IDSEL 被采用时它稳定。

5.14.4 中断应答

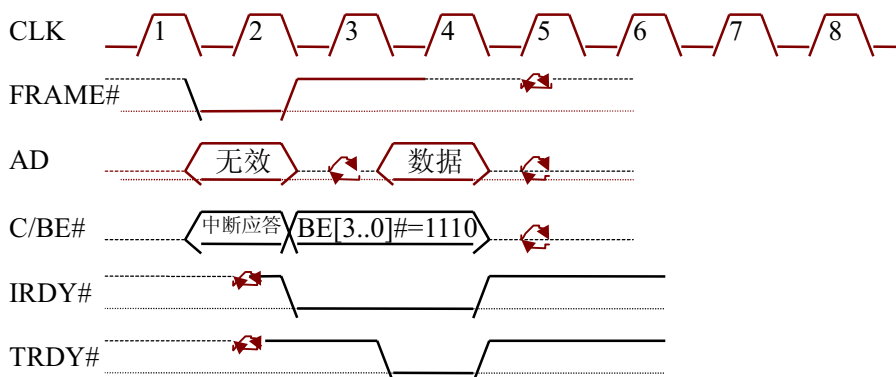


图 5.17 中断应答

PCI 总线支持一种如图 5.16 所示的中断应答周期。此图所示是 PCI 上的一次 X86 中断应答周期的例子。在地址段，AD[31..00]不包含有有效的地址，但必须用有效数据驱动，PAR 有效，并且奇偶性可以检测，在数据段，只有一个字节允许有效。只有一个单元能对中断应答作出反应。做此事的设备必须使其 DEVSEL#有效，否则此周期将被负解码。当 TRDY#有效时，必须返回中断向量。和别的周期一样，中断应答周期也能插入等待状态，并且这种请求可以被目标设备终止。与 8259 的双周期应答不一样，PCI 执行的是一种单周期应答。通过放弃处理器第一个中断应答请求，桥路将处理器的双周期格式就轻易地转换成 PCI 单周期格式。

5.14.5 错误功能

PCI 提供奇偶校验和其它系统错误的检测并发出有错信号。PCI 错误服务区可以包括从对错误(尤其是奇偶校验错误)不感兴趣的设备到检测、发生信号，并修复错误单元。这样就可以使出错的单元得以修复而不致影响到未出错的单元。为达到如此灵活性，所有单

元对所有传送都要作奇偶校验。

5.14.5.1 奇偶校验

PCI 上的奇偶校验提供了一种机制，以逐个检查总线主设备是否成功地寻址所希望的目标设备，或它们之间的数据传送是否正确。在地址段和数据段期间，无论是否所有 AD[31..00]及 C/BE[3..0]#线都带有有意义的信息，它们都要参与奇偶校验。未传送数据的字节通道也要求驱动到稳定并包括在奇偶校验中。在配置周期，特殊周期，或中断应答命令中，一些(或全部)地址线未定义，但都要求驱动到稳定数据并包括到奇偶校验计算中去。

所有总线主设备和目标设备都要对从 PCI 总线上获得的地址和数据做奇偶校验并报告奇偶校验错误，为进行奇偶校验，设备要执行下面功能：

1. 设备内部通过锁存 AD[31..00 和 C/BE[3..0]#并进行异或产生偶校验。
2. 在下一个时钟脉冲，设备将它的内部计算结果和产生偶校验设备的 PAR 进行异或。
3. 如果这两个值一致，偶校验没错，否则，在下一个时钟脉冲设备或通过使 PERR#有效报告数据错误，或通过使 SERR#有效报告地址错误。

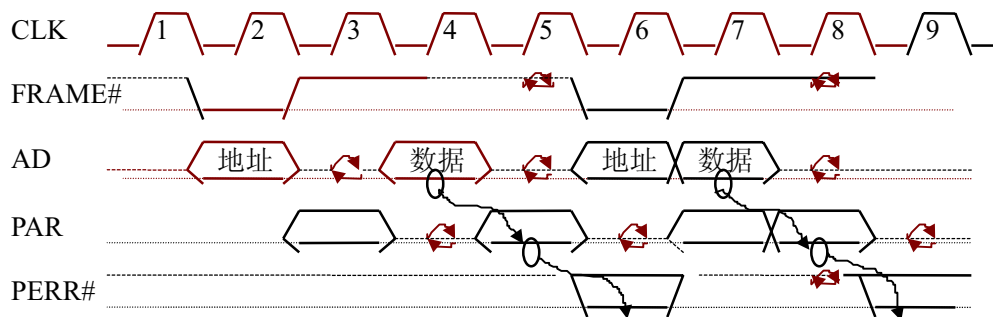


图 5.18 奇偶校验操作

在任何一个给定的总线段，驱动 AD[31..00]的设备也必须驱动 PAR 信号，且比相应的地址或数据段滞后一个时钟。图 5.18 说明了一个带有奇偶校验的读和写传送，在时钟 3 和 7，总线主设备驱动地址段的 PAR。读传送中目标设备驱动数据段的 PAR(时钟 5)，写传送中总线主设备驱动数据段的 PAR(时钟 8)。注意，除了一个时钟的滞后外，PAR 和 AD[31..00 一样也要包括等待状态和转换周期。

在发生了错误的数据传送之后，单元必须在两个时钟内使 PERR#有效，正在接收数据的单元在检测到奇偶错误之后可以不受约束地使 PERR#有效(可能在数据传送开始之前就发生了)。一旦 PERR#有效，它就要保持有效直到实际传送完之后两个时钟。只要 PERR#有效，总线主设备就知道发生了数据奇偶错误，但只有在传送之后两个时钟才知道传送出了错误。

在未插入等待状态的多数据传送情况下，PERR#将在多个连续时钟内有意义，并可能在其中部分或全部时钟内有效。因为 PERR#是一个连续三态信号，所以在每个有意义的时钟沿它都应该驱动到正确的电压值。为使它在每次总线操作结束后返回标称状态，它必须在 AD 线转换周期之后两个时钟驱动到高，历时一个时钟周期(图 5.14 的时钟 7)，之后一个时钟是 PERR#的转换时钟周期(图 5.14 的时钟 8)。PERR#在当前周期不被驱动直到地址段后至少三个时钟周期。

5.14.5.2 错误反应

当 PCI 检测到奇偶校验或其它系统错误时，可以预计，只要可能，无论何时奇偶错误

都会通过这种操作和设备驱动程序链而被返回。从目标设备到总线主设备、设备驱动程序、设备管理器及操作系统的错误反应链都是为了在任何一级上都允许错误弥补。因为通常情况下不可能用特殊错误链来消除系统错误，所以它们就直接反应到系统一级。

在 PCI 错误反应设计中，使用了两个信号 PERR#和 SERR#(引脚)，PERR#专用于反应除特殊周期命令外的所有传送中的数据奇偶错误。它是一个持续三态信号并连到索引 PCI 单元中。总线协议保证 PERR#不会同时被多个总线设备驱动，并且适当的信号转换时间也避免了任何驱动器争用。只有总线主设备才能反映读数据奇偶错误，只有选中的目标设备才能反应写数据的奇偶错误。

必须要检测奇偶性以确定总线主设备是否成功地寻址了所希望的目标设备，数据传送是否正确。在所有情况下，支持奇偶性检测的单元在检测到奇偶错误时，必须设置配置空间状态寄存器的奇偶错误检测到位(Parity Error Detected)。对奇偶校验错误的信号发生和应答都由奇偶错误应答(Parity Error Response)位来控制。除了后面两种专用设备外，所有设备都要用到这一位。如果此位清除，该单元就忽略所有奇偶错误，并认为奇偶正确而完成传送。如果这一位设置，则该单元在检测到奇偶错误时，要使 PERR#有效，其它错误应答根据设备而定。

当一个总线主设备检测到数据奇偶错误并使 PERR#有效(在读传送中)或采样到 PERR#有效(在写传送中)时，它必须设置数据奇偶反应位(状态寄存器的第 8 位)，并可继续该传送或终止它。一个检测到奇偶错误的传送的目标设备可以继续该操作或通过目标设备终止来结束它。目标设备绝不能设置数据奇偶错误反应位。当 PERR#有效时，建议让总线主设备和目标设备完成这次传送，对目标设备而言，PERR#只是一个输出信号，而对总线主设备，它可以是输入和输出信号。

当一个操作的总线主设备发现在它的传送中发生了奇偶校验错误，它就应该向系统说明。建议采样中断的方法让总线主设备通知其设备驱动程序(或调整状态寄存器，或标志)，如果这些方法都无效，作为最后弥补，使 SERR#有效，以便可靠地将错误信息送给操作系统。注意，系统设计人员可能会在中央资源中将所有 PERR#错误系统转换成 SERR#错误信号以便将奇偶错误信息传送给操作系统。

SERR#用于发出所有地址奇偶校验错误和特殊周期命令中的数据奇偶校验错误，并且可能有选择地用于别的非奇偶性或系统错误中。它是漏极开路输出，并可以与所有 PCI 单元之 SERR#线或所以可能同时被多个单元驱动。因为漏极开路信号在每个时钟沿不能产生稳定的信号，一旦 SERR#有效，它的逻辑值必须看成是未定的，直到该信号至少在两个连续的时钟沿被采样到无效。

任何单元，无论是总线主设备或是目标设备，都可以在 SERR#上检查到或发出地址奇偶错误。无论是何种类型的错误，只有命令寄存器中的 SERR#允许位设置成逻辑 1 时，SERR#才可能有效。无论是何种类型的错误，只要某一单元使 SERR#有效时，该单元都要设置配置空间状态寄存器中的系统错误信号位。此外，如果错误类型是奇偶错误时(如地址奇偶错误)，在各种情况下，奇偶错误检测到这一位必须设置，但在 SERR#上的信息反应则由命令寄存器中的奇偶错误应答位来制约。

检测到地址奇偶错误的被选中单元将做下列事情之一：

确认周期并象地址正常一样终止；确认周期并用目标设备失败终止；或是不确认周期，任由传送被总线主设备失败终止。该目标设备不允许用重试或解除连接终止传送，因为已检测到地址奇偶错误。

SERR#与任何 PCI 传送没有时序上的联系(CLK 之外)。然而，错误将尽快地传送出去，最好就在检测的两个时钟内。只有将低脉冲转换成给处理器的信号的中央资源才会在意 SERR#(作为输入)。中央资源如何给处理器发信号要根据系统而定，但应包括产生 NMI，

高优先级中断，设置状态位或标志。然而，使 **SERR#**有效的单元必须使中央资源产生一个 **NMI**，否则，错误信息就会通过别的机制发出(即中断状态寄存器或标志)。

当奇偶错误应答位处于允许状态时，而且 **SERR#**允许位也允许时，在下列条件之下单元将使 **SERR#**有效：

- 地址奇偶错误或特殊周期中的数据奇偶错误被检测到。
- 奇偶错误的检测结果未通过别的机制传送出去(仅对总线主设备)。

当 **SERR#**允许位允许时，在下列条件下单元将使 **SERR#**有效：

- 总线主设备(无驱动程序)参与了被异常终止的传送。
- 重大错误使得该单元丧失正常操作的能力。

注意，所有单元都要求产生奇偶校验(对此要求无例外情况)。用于非奇偶校验错误的 **SERR#**信号是可选的。然而，必须注意到 **SERR#**上发出信号将产生 **NMI**，所以，在使用 **SERR#**时要很小心。

对于配置周期命令和特殊周期命令，总线主设备失败对于桥路来说不属于异常条件。在这种情况下或是能正常弥补的情况下，不能用 **SERR#**。目标设备失败通常是目标设备异常终止，并且在总线主设备不能通过自己的设备驱动程序反应错误时，可能将它当作一种错误而由 **SERR#**来反应出去(近由总线主设备)。

因为在 **PCI** 上要求发出奇偶校验错误信号，故需要 **PERR#**和 **SERR#**引脚，然而对以下这两类设备可以不考虑这种要求：

1. 为主板或平面设计的设备，如芯片集。因为它不能用于扩展板，系统商要控制这类设备的使用。
2. 不涉及、不包含、不处理任何代表固定的或有后遗症的系统或应用状态的数据的设备。例如人机界面和视频/音频设备。这类设备只涉及固有的或有后效的系统或应用状态的暂时出现的数据(如象素点)，所以，即使不检测错误也不会产生系统综合性问题。

第六章 **PCI** 接口状态机及其实现

本章描述 **PCI** 总线主设备和目标设备设备状态机及其实现，这些状态机仅用于描述 **PCI** 协议的目的。实际实现时不能直接用这些状态机，这些状态机在原理上是正确的，但如果和协议有冲突，当然要以协议为准。

这些状态机用了三种类型的变量：状态机的状态=**STATE**(大写)，**PCI** 信号=**SIGNAL**(黑体)和中间信号=**Signal**(小写)。

状态机假设从进入一个状态到信号产生及可供状态机使用没有延迟，所有 **PCI** 信号在 **CLK** 信号的上升沿锁存。

这些状态机支持在 **PCI** 协议中讨论过的一些选项(但不是全部)，对每一个状态及选项的描述在每一个状态定义的后面。

总线接口包括两部分，第一部分是执行实际总线操作的总线定序器，第二部分是终端设备或硬件应用。在一个总线主设备中，终端设备产生传送并提供地址、数据、命令、字节允许和传送的长度。当一个传送被重试时它还要负责提供地址。在一个目标设备中，终端设备决定什么时候传送终止。定序器按照要求执行总线操作并保证不会违反 **PCI** 协议，注意目标设备实现了一个资源锁定。

在状态机方程中，“+”代表逻辑或(**OR**)，“*”代表逻辑与(**AND**)并优先于逻辑或，括号的优先级又高于它们俩，“!”代表逻辑非(**NOT**)。在状态机方程中，**PCI SIGNALs** 代表 **PCI** 总线上信号的实际状态，低有效信号当它们以 **!SIGNAL#**出现时是真的或有效的，当它们以 **SIGNAL#**出现时是假的或无效的；高有效信号当它们以 **SIGNAL** 出现时是真的或有效的，

当它们以!**SIGNSL** 出现时是假的或无效的，中间信号当它们以 **Signal** 出现时是真的，当它们以!**Signal** 出现时是假的，一些输出允许方程用“=”信号表示前一个状态，例如

PAR.oe==[**S_DATA** *!**TRDY#** * (cmd=read)]

这表示 **PAR** 信号的输出缓冲器在一个读传送中当前一个状态是 **S_DATA** 且 **TRDY#**有效时使能。第一个状态机针对目标设备，第二个状态机真对主设备，需要特别小心的是当一个单元既是主设备又是目标设备时，每一个都要有自己的能独立于另一个进行操作以避免死锁的状态机。虽然它们有相似的状态，但它们不能在一个状态机中。

6.1 目标设备状态机

下面定义的状态机中间输入信号在目标设备总线定序器和终端设备之间，它们表明总线定序器是怎样响应当前总线操作：

Hit =地址解码命中
Dt =Devsel 定时器超时而 **DEVSEL#**还没有有效
T_abort =目标设备处于错误状态并要求当前传送停止
Term =终止传送(内部冲突或大于 N 个等待状态)
Ready =准备好传送数据
L_lock# =在地址段期间锁存的 **LOCK#**信号
Tar_dly =仅用于零等待状态解码中的回转延迟
R_perr =报告奇偶校验错误是一个 PCI 时钟的脉冲

目标设备定序器状态机：

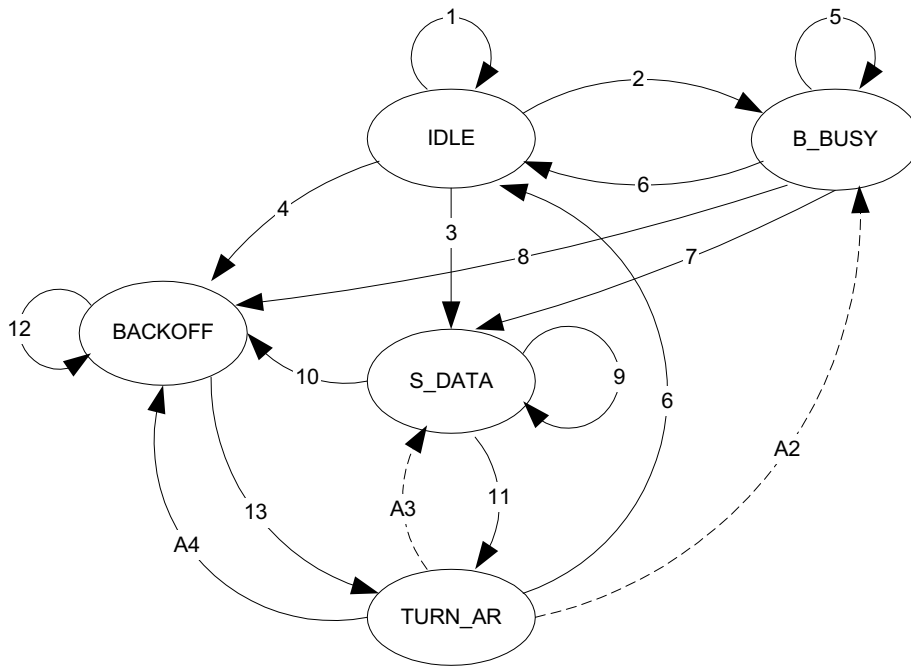
IDLE 或 TURN_AR	IDLE 条件或总线传送已完成
1 goto IDLE	if FRAME#
2 goto B_BUSY	if! FRAME# *! Hit
3 goto S_DATA	if! FRAME# * Hit * (! Term + Term * Ready) * (FREE + LOCKED * LOCK#)
4 goto BACKOFF	if! FRAME# * Hit * (Term *! Ready + LOCKED *! LOCK#)
B_BUSY	不包含于当前传送中
5 goto B_BUSY	if (! FRAME# +! IRDY#) *! Hit
6 goto IDLE	if FRAME#
7 goto S_DATA	if (! FRAME# +! IRDY#) * Hit * (! Term + Term * Ready) * (FREE + LOCKED * L_lock#)
8 goto BACKOFF	if (! FRAME# +! IRDY#) * Hit * (Term *! Ready + LOCKED *! L_lock#)
S_DATA	单元已接受请求并将响应
9 goto S_DATA	if! FRAME# *! STOP# *! TRDY# * IRDY# +! FRAME# * STOP# + FRAME# * TRDY# * STOP#
10 goto BACKOFF	if! FRAME# *! STOP# * (TRDY# +! IRDY#)
11 goto TURN_AR	if FRAME# * (! TRDY# +! STOP#)
BACKOFF	此时单元忙，不能响应
12 goto BACKOFF	if! FRAME#
13 goto TURN_AR	if FRAME#

目标设备 LOCK 状态机:

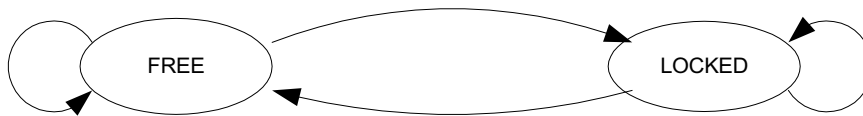
```
FREE          单元可以自由地响应所有传送
  goto LOCKED  if !FRAME# * LOCK# * Hit * (IDLE + TURN_AR)
                + L_lock# * Hit * B_BUSY)
  goto FREE    if ELSE
LOCKED        除非 LOCK#在地址段无效, 否则单元将不予响应
  goto FREE    if FRAME# * LOCK#
  goto LOCKED  if ELSE
```

传送的目标设备负责驱动以下信号:

```
AD[31..00].oe = S_DATA * Tar_dly * (cmd=read)
TRDY#.oe      = BACKOFF + S_DATA + TURN_AR
STOP#.oe      = BACKOFF + S_DATA + TURN_AR
DEVSEL#.oe   = BACKOFF + S_DATA + TURN_AR
PAR.oe        == S_DATA * !TRDY# * (cmd = read)    (AD[31..00].oe 延迟一个时钟)
PERR.oe       = (cmd = write) * !IRDY# * (delay by two clocks)
!TRDY#        = Ready * !T_abort * S_DATA * (cmd = write + cmd = read * Tar_dly)
!STOP#        = BACKOFF + S_DATA * (T_ abort + Term)
                * (cmd = write + cmd = read * Tar_dly)
!DEVSEL#     = (BACKOFF + S_DATA) * !T_abort
PAR          = AD[31..00]和 C/BE[3..0]#线上的偶校验
PERR#        = R_perr
```



目标定序器状态机



目标LOCK状态机

图 6.1 目标设备状态机

下面段落讨论每一个状态并叙述如果某些 PCI 可选项没有实现时哪些方程可以去掉。**IDLE** 和 **TURN_AR** 在状态机中是两个分开的状态，但是因为从这两个状态开始的状态变化是一样的，这里将它们合并在一起，它们之所以做成分开的状态是因为有效信号在目标设备将它置为三态之前先要使它们无效。

如果一个目标设备不能做单周期地址解码，从 **IDLE** 到 **S_DATA** 的通道(3)可以去掉，从 **TURN_AR** 到 **S_DATA** 和 **B_BUSY** 的通道(A2、A3)是目标设备用于快速背对背传送操作的，目标设备必须能解码背对背传送操作。

B_BUSY 是单元等待当前传送完成而让总线回到 **IDLE** 状态的状态，这个状态对于做慢速地址解码或反解码的设备很有用处，如果目标设备没有做这两项，从 **B_BUSY** 到 **S_DATA**

和 **BACKOFF** 的通道(7、8)可以去掉。“!Hit”也可以从 **B_BUSY**(5、7)方程中去掉，这可以减少等待当前总线传送完成的状态。

S_DATA 是目标设备传送数据的状态，没有可选项方程。

BACKOFF 是目标设备使 **STOP#**有效并等待总线主设备使 **FRAME#**无效时状态机所处的状态。

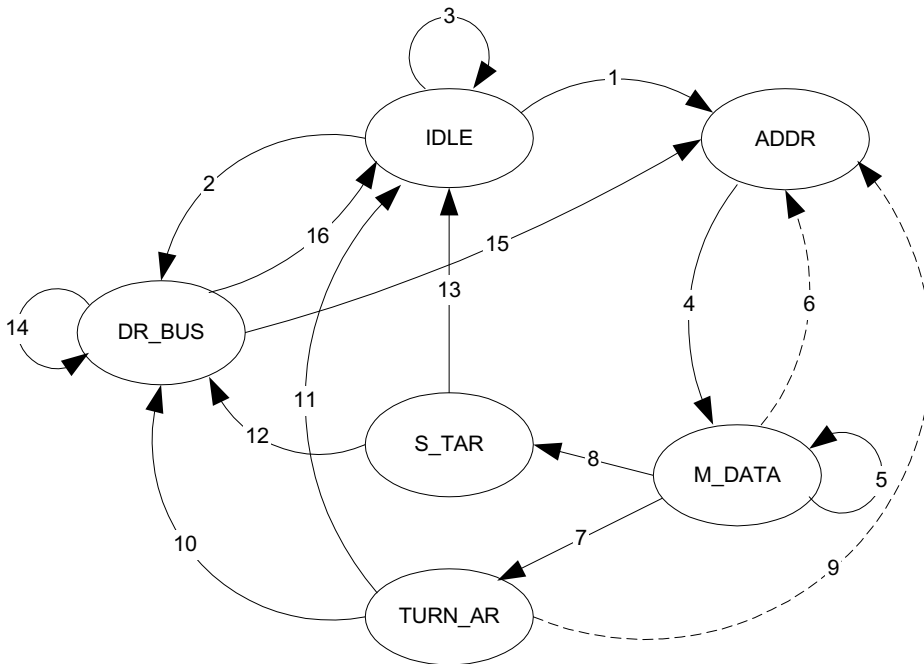
FREE 和 **LOCKED** 是目标设备相应于锁定操作的状态，如果目标设备没有实现 **LOCK#**，这些状态就不需要，**FREE** 表明当单元是目标设备时合时可以接受任何请求。如果 **LOCKED**

目标设备将 重试任何请求，除非 **LOCK#**在地址段无效。当单元是传送的目标设备且 **LOCK#**在地址段 期间无效标志着它被锁定。目标设备在没有被锁定的传送中锁定自己是有点混乱，但是从实现的观点看这是一个用组合逻辑并总能工作的简单的机制，在传送结束时当设备检测到 **FRAME#**和 **LOCK#**都无效时它将开锁。

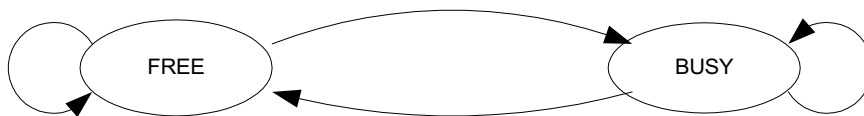
当做快速解码时，从 **FREE** 状态到 **LOCKED** 的第二个方程可以去掉。当做中速或慢速解码时第一个方程可以去掉。

L_lock#是在地址段锁存的 **LOCK#**信号并在单元解码完成时用到。

6.2 总线主设备状态机



总线主控定序器状态机



总线主控**LOCK**状态机

图 6.2 总线主设备状态机

下面定义的这些信号在总线定序器和终端设备之间，它们向总线定序器提供何时执行一个传送信息，并向终端设备提供传送怎样进行的信息，当一个周期被重试，终端设备会修改受影响的寄存器并向定序器表明执行另一个传送，总线定序器不去记忆一个传送是被重试还是被中止，但要接受来自终端设备的请求并执行 PCI 传送。

Master_abort = 传送被总线主设备中止(没有设备选通)

Target_abort = 传送被目标设备中止

Step = 单元使用地址分步(在该状态等待直到!Step)

Request	=请求挂起(pending)
Comp	=当前传送处于最后一个数据段
L_cycle	=最后一个周期是写周期
To	=总线主设备定时器溢出
Dev_to	=设备选通定时器溢出 DEVSEL# 无效
Sa	=下一个传送和上一个传送针对同一个单元
Lock_a	=请求是一个锁定操作
Ready	=准备好传送数据
Sp_cyc	=特殊周期命令
Own_lock	=该单元当前拥有 LOCK#
Ldt	=数据在锁定操作期间被传送
R_perr	=报告奇偶校验错误是一个 PCI 时钟的脉冲

总线主设备定序器状态机

IDLE 总线空闲条件

```

1  goto ADDR      if Request * !GNT# * FRAME# * IRDY# * !Step
2  goto DR_BUS   if (!Request * !GNT# + Request * !GNT# * Step)
                    * FRAME# * IRDY#
3  goto IDLE     if ELSE
ADDR          总线主设备开始一个传送
4  goto M_DATA   在下一个 CLK 上升沿
M_DATA       总线主设备传送数据
5  goto M_DATA  if !FRAME# + FRAME# * TRDY# * STOP#
                    * !Dev_to * !(cmd=0001 * Comp)
6  goto ADDR     if FRAME# * !Step * !TRDY# * STOP# * !(cmd=0001)
                    * (Sa * L_cycle * Request * !GNT#)
7  goto TURN_AR if FRAME# * !TRDY# * STOP#
                    * !(Sa * L_cycle * Request * !GNT#)
                    + !(cmd=0001) * Comp
8  goto S_TAR   if FRAME# * !STOP# + FRAME# * Dev_to
TURN_AR      传送完成, 做内部事务管理
9  goto ADDR    if Request * !GNT# * !Step
10 goto DR_BUS  if !Request * !GNT# + Request * !GNT# * Step
11 goto IDLE    if GNT#
S_TAR       STOP 有效, 做回转周期
12 goto DR_BUS  if !GNT#
13 goto IDLE    if GNT#
DR_BUS      总线停驻在该单元上或该单元用地址分步
14 goto DR_BUS  if Request * !GNT# * Step + !Request * !GNT#
15 goto ADDR    if Request * !GNT# * !Step
16 goto IDLE    if GNT#

```

总线主设备 LOCK 状态机

FREE LOCK#信号没有用(没有拥有)

```

goto FREE      if LOCK# + !LOCK# * Qwn_lock
goto BUSY      if !LOCK# * !Own_lock

```

BUSY 当前 LOCK#正被使用(拥有)

```

goto FREE      if LOCK# * FRAME#
goto BUSY      if !LOCK + !FRAME#

```

传送的总线主设备负责驱动下列信号:

输出缓冲使能:

```

FRAME#.oe      = ADDR + M_DATA
C/BE[3..0]#.oe = ADDR + M_DATA + DR_BUS
                I  if ADDR 驱动命令
                   if M_DATA 驱动字节允许
                   if DR_BUS if (Step * Request) 驱动命令 else 驱动总线到正确状态
AD[31..00].oe = ADDR + M_DATA * (cmd=write) + DR_BUS
                   if ADDR 驱动地址
                   if M_DATA 驱动数据
                   if DR_BUS if (Step * Request) 驱动地址 else 驱动总线到正确状态
LOCK#.oe       = Own_lock * M_DATA + LOCK#.oe * (!FRAME# + !LOCK#)
IRDY#.oe       = M_DATA + ADDR
PAR.oe         = ADDR + M_DATA * !IRDY# * (cmd=write)
PERR.oe        = (cmd=read) * !TRDY# * (delay by two clocks)

```

下列信号由状态产生并采样总线信号:

```

!FRAME#        = ADDR + M_DATA * !Dev_to
                  * ((!Comp * (!To + !GNT#) * !STOP#) + !Ready)
!IRDY#         = M_DATA * (Ready + Dev_to)
!REQ#          = (Request * !Lock_a + Request * Lock_a * FREE)
                  * (!S_TAR + Last State not S_TAR)
LOCK#          = Own_lock * ADDR + Target_abort
                  + Master_abort + M_DATA * !STOP# * TRDY# * !Ldt
                  + Own_lock * !Lock_a * Comp * M_DATA * FRAME# * !TRDY#
PAR            = AD[31..00]和 C/BE[3.0]#上的偶校验
PERR#          = R_perr

Master_abort     = (M_DATA * Dev_to)
Target_abort     = (!STOP# * DEVSEL# * M_DATA * FRAME# * !IRDY#)
Own_lock         = LOCK# * FRAME# * IRDY# * Request * !GNT# * Lock_a
                  + Own_lock * (!FRAME# + !LOCK#)

```

总线主设备状态机包括许多对一些实现没有意义的可选项。在讨论每一个状态时会明确某个选项对方程的影响。

IDLE 是总线主设备等待请求总线操作的状态, 这个状态唯一的可选项是“Step”, 如果不支持地址分步, 它便可以从方程(1、2)中去掉。所有的通道都必须实现, 到 DR_BUS

的通道(2)是必须的以保证总线不会长期浮空。 如果其请求无效，GNT#有效的总线主控必须驱动总线。

ADDR 用于驱动地址和命令到总线上，它没有可选项。

M_DATA 数据被传送，如果总线主设备不支持快速背对背传送，到 ADDR 状态的通道(6)就不需要。

从协议的角度看这些方程是正确的，但当编译器检查所有可能的组合时可能给出错误，例如，因为协议，当 FRAME#无效时 Comp 不能有效，Comp 表明总线主设备处于最后一个数据段，FRAME#必须无效才对，所以在到 M_DATA 方程(5)的第一条和到 TURN_AR 方程(7)的最后一条会引起问题。修复的办法是将 **FRAME# * TRDY# * STOP#** 加到(cmd=0001 * Comp)上，如果只加上 **FRAME#**，另一个方程会受到类似的影响。同样原因，在到 S_TAR 方程(8)的最后一条中要加上 TRDY#。

TURN_AR 是总线主设备使信号无效以三态它们的状态，如果总线主设备不做背对背传送，到 ADDR 的通道(9)可以去掉。

S_TAR 可以以好几种方法实现，选择这个状态来区分目标设备使 **STOP#**有效的状态。

DR_BUS 用在 GNT#已经有效但总线主设备或者没有准备好开始传送(地址分步)，或者还没有挂起(has none pending)，如果地址分步没有做，则在到 DR_BUS 的有“Step”的方程可以去掉，到 ADDR 的方程也可以去掉“Step”。

如果总线主设备不支持 LOCK#，FREE 和 BUSY 状态可以去掉，这两个状态用在总线主设备希望作锁定传送时知道 LOCK#的状态。状态机简单地检查 LOCK#是否有效。一旦有效，它处于 BUSY 状态直到 FRAME#和 LOCK#都无效标志着 LOCK#已被释放。

6.3 举例

6.3.1 ALTERA PCI 开发包的总线主设备和目标设备宏功能的特点如下表所示：

总线主设备	目标设备
执行单数据周期读和写，可以做成猝发数据传送	响应单数据周期读和写，可以做成猝发数据传送
进行存储器或 I/O 空间寻址	支持 32 位 I/O 空间
启动配置空间操作	响应配置空间操作
解除对配置空间的猝发传送	解除配置空间的猝发传送，并执行 I/O 空间的猝发传送
总线主设备定时器溢出终止，识别目标设备终止	产生目标设备终止
识别重试	产生重试
产生并检查奇偶校验	产生并检查奇偶校验
响应系统复位	响应系统复位
产生奇偶校验错误和系统错误，并向设备驱动器报告系统错误	产生奇偶校验错误和系统错误

6.3.2 ALTERA PCI 开发包的总线主设备和目标设备宏功能状态机如下表所示：

状态	主 控	目 标
IDLE	总线主设备锁存来自总线的信号以确定另外一个总线主设备试图对其配置空间进行操作，并检查来自终端设备的数据请求，这是缺省状态	目标设备锁存来自总线主设备的信号以确定总线主设备是否试图对其进行操作，这是缺省状态。
M_ADDR1	总线主设备在下一个 CLK 脉冲	--

	的开始使 FRAME#无效，这个状态用于准备传送的地址段	
状态	主 控	目 标
M_ADDR2	总线主设备驱动传送的地址段到 PCI 总线上，然后，FRAME#有效，AD 线上驱动以地址，C/BE[3..0]#驱动以总线命令。	--
CMP_ADDR	--	当一个总线传送开始后，总线上的地址要和终端设备的地址空间进行比较，如果地址 匹配，接口进入 S_DATA 状态以开始传送，否则，接口转入 B_BUSY 状态。
M_DATA	总线主设备驱动传送的数据段：总线主设备在地址段之后驱动地址奇偶校验及在写传送的数据段之后驱动数据奇偶校验到 PCI 总线上	
S_DATA	总线主设备驱动配置操作数据传送，一个传送完成后，总线主设备要么因不做猝发传送而进入 TURN_AR 状态，要么试图一个猝发传送而进入 BACKOFF 状态	目标设备处理所有的数据传送，当所有数据传送完成之后，接口或者在非猝发传送无误情况下进入 TURN_AR 状态，或者在检测到数据错误或拒绝猝发传送的企图而进入 BACKOFF 状态。
BACKOFF	总线主设备在解除猝发配置操作后，在该状态等待到 FRAME#无效后进入 TURN_AR 状态	目标设备解除来自总线主设备的猝发传送企图，或表明目标设备终止或重试，目标设备在该状态等待到主设备使 FRAME# 无效后进入 TURN_AR 状态
TURN_AR	总线主设备执行下面功能：在数据传送完成后主动使 PCI 信号无效以释放总线，检查(读)或产生(写)最后一个数据段的奇偶校验，并在目标设备发出重试或终止信号时向终端设备报告错误	目标设备主动使 PCI 信号无效以释放总线，目标设备从这个状态自动返回到 IDLE 状态
DR_BUS	不是允许设备在不主动或浪费能量期间三态信号，总线仲裁器将总线停放在仲裁主设备上，导致主设备驱动无效的数据到总线上，除了在 IDLE 状态总线主设备不驱动数据到总线上外，这个状态类似于 IDLE 状态，	
B_BUSY	总线主设备在传送的中间不能判断是不是一个配置操作，IDLE 信号是从 AD 线解码而来的，所以，即使不是一个配置操作，在一个传送的数据段期间它也可以有效，目标设备只在 FRAME#有效后 CLK 脉冲第一次有效时(地址段)检测到一个配置操作但它不是这次配置操作的目标设备时进入 DROP_REQ 状态，总线主设备在 FRAME#无效时进入 IDLE 状态，同时，在终端设备使 DEV_REQ#有效时，总线主设备继续请求总线	目标设备监视不包括终端设备的总线传送，只要另一个总线传送正在进行(FRAME#有效)，接口就一直处于该状态，另一个传送完成后(FRAME#无效)，目标设备返回到 IDLE 状态并等待下一个传送的地址段
DROP_REQ	终端设备停止驱动所有请求信号并准备一个配置操作	
READ_BAR		目标设备读配置空间的基地址寄存器

WRITE_BAR	目标设备写配置空间的基地址寄存器
-----------	------------------

6.4 接口状态机的实现

6.5.1 简介

PCI 总线是为多处理器系统和高性能外围设备设计的，包括音频和视频系统，网络适配器，图形加速板和数据存储控制器。PCI 兼容设备需要很多引脚，很多驱动器，较高的集成度，和符合时间特性。PCI 兼容设备的设计不需要实现 PCI 局部总线规范所描述的所以功能，一般只用到这些规范的一个子集。

在 PCI 总线规范中，总线主设备控制进出目标设备的数据传送，要发起一个数据传送，总线主设备必须使必要的信号有效并保持有效直到它们被目标设备认可。要表明传送完成，目标设备必须提供获取周期，可能包括错误或重试信号，PCI 接口自动插入必要的等待状态以防止主设备时间溢出，以保证高速的 PCI 传送能在速度相对较低的终端设备上进行。

一个 PCI 总线主设备必须能够执行猝发读和写，对存储器、I/O 和配置空间进行操作寻址，响应系统复位，产生奇偶校验，报告奇偶校验错误，识别目标设备终止，识别重试和在时间溢出时主设备终止。一个目标设备必须能够进行地址解码，处理配置操作，响应系统复位，产生奇偶校验，报告奇偶校验错误和系统错误，产生重试和目标设备终止。

一个 PCI 终端设备被定义为使用 PCI 总线存储、发送和重新获得信息的设备(例如视频卡，存储器卡，磁盘驱动器，多媒体卡)，习惯上，认为终端设备就是一个目标设备。目标设备接口逻辑和存储器既可以包装在控制设备中，也可以部分或全部集成在终端设备中，例如在配置请求期间让目标设备能够操作终端设备上的存储器可以在控制设备上得到更多的自由空间。

终端设备在物理上远离处理器局部总线，终端设备通过桥路和处理器通信，桥路担任在处理器和终端设备之间的管理层，从而使数据传送流畅。接口支持总线主设备，从而允许智能设备直接操作主存储器。

PCI 总线规范支持猝发传送，一个猝发传送由一个地址段和相随的一个或多个数据段组成，最大的猝发数据段是 256 个，猝发传送对终端设备和配置空间都是一样的。

表 6.5.1 PCI 总线信号

信号	类型	主设备	目标设备	描述
CLK	in	in	in	系统时钟，为所有传送及总线主设备提供时序
RST#	in	in	in	系统复位，使 PCI 定序器，总线及配置寄存器处于特定的状态
AD[31..00]	t/s	bidir	bidir	地址/数据复用，地址段及写传送由总线主设备驱动，读传送有选定的目标设备驱动
C/BE[3..0]#	t/s	bidir	in	总线命令(地址段)/字节允许(数据段)复用
PAR	t/s	bidir	bidir	AD[31..00]和 C/BE[3..0]#上的偶校验
FRAME#	s/t/s	bidir	in	由当前总线主设备驱动，表明传送的开始和进行
IDSEL	in	in	in	在操作配置空间时用作片选
DEVSEL#	s/t/s	in	out	由目标设备驱动的地址解码选通，如果在 6 个 CLK 周期内仍然无效，当前主设备用总线主设备失败终止这次传送
IRDY#	s/t/s	bidir	in	发起传送者准备好数据(写)或已收到数据(读)
TRDY#	s/t/s	bidir	out	读传送中目标设备数据已有效或写传送中目标设备已收到数据
STOP#	s/t/s	bidir	out	目标设备请求总线主设备中止这次传送

LOCK#	s/t/s	bidir	in	资源锁定，要求多个周期完成的操作
REQ#	t/s	out	—	总线主设备向仲裁器申请单独使用总线
GNT#	t/s	in	—	仲裁器批准申请总线的主设备使用总线
PERR#	s/t/s	bidir	out	数据奇偶校验错误
SERR#	o/d	out	out	地址奇偶错误或特殊周期中的数据奇偶错误，由配置空间命令寄存器 COMMAND.8 位控制输出允许
INTX#	o/d	out	out	INTA#、INTB#、INTC#、INTD#是连接到系统中断控制器的集电极开路信号
SDONE	bidir	—	in	当 Cache 完成对一个 PCI 存储器操作的监视时由 Cache 使之有效
SBO#	bidir	—	in	当 SDONE 有效时，SBO#有效表明正在进行的 PCI 存储器操作打算读或更新存储器得信息
REQ64#	s/t/s	bidir	in	由当前总线主设备驱动，表明一个 64 位传送
ACK64#	s/t/s	in	out	由选定的目标设备驱动，回答总线主设备的 REQ64#信号
AD[63..32]	t/s	bidir	bidir	高 32 位地址/数据复用
C/BE[7..4]#	t/s	out	in	附加的命令和字节允许信号
PAR64	t/s	bidir	bidir	AD[63..32]和 C/BE[7..4]#上的偶校验
TCK	in	in	in	JTAG 操作同步时钟
TDI	in	in	in	所有 JTAG 移位寄存器的输入引脚
TDO	out	out	out	来自 JTAG 寄存器的扫描输出引脚
TMS	in	in	in	控制测试存取口状态机(TAP)的状态
TRST#	in	in	in	测试存取口状态机(TAP)复位

所有位于终端设备和目标设备之间的信号都和 PCI 总线的 CLK 信号同步。终端设备 I/O 信号如表 7.2 所示

表 6.5.6 终端设备 I/O 信号

信号	功能	描述
ADDR[31..00]	地址总线	载荷地址信息
APAR	地址偶校验	在地址信息正确时驱动来自终端设备的偶校验到 ADDR[31..00]上
BE[3..0]#	字节允许	在双字操作中识别字节允许
CNFG	配置操作	为高时表明是配置操作，为低时表明是 I/O 操作或告诉终端设备停止驱动总线准备让主设备做配置操作
DATA[31..00]	数据总线	载荷数据
DEV_REQ#	设备请求	告诉目标设备终端设备已经完成到目标设备的数据传送，或告诉总线主设备终端设备有数据要向主设备传送。
DEV_ACK#	设备应答	告诉目标设备总线主设备有一个到终端设备的数据请求，或告诉终端设备到总线主设备的数据传送已经完成
DPAR	数据偶校验	在数据正确时驱动来自终端设备的偶校验到 DATA[31..00]上
M_ABORT#	主设备终止	当其有效时主设备告诉终端设备没有目标设备响应它的请求
MEM_IO	MEM/ I/O	为高表明是存储器传送，为低表明是 I/O 传送

RETRY#	重试	由于目标设备当时不能响应，告诉总线主设备以后重试
RD_WR	读/写	为高表明是读传送，为低表明是写传送
T_ABORT#	目标设备终止	表明目标设备发生了致命错误

6.5 PCI 接口控制器

PCI 控制器将一个总线的地址空间映射到另一个总线并作为两个同步系统的弹性缓冲区，在终端设备支持 32 位数据及相关的控制和地址信号时，PCI 控制器将超过 100 个引脚。逻辑将超过 10000 门。

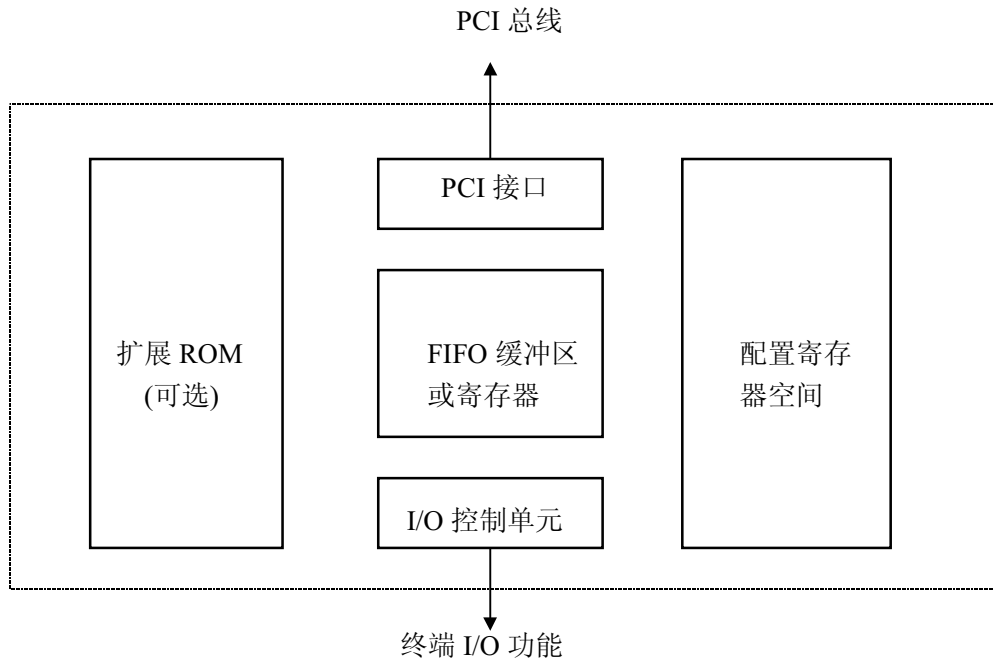
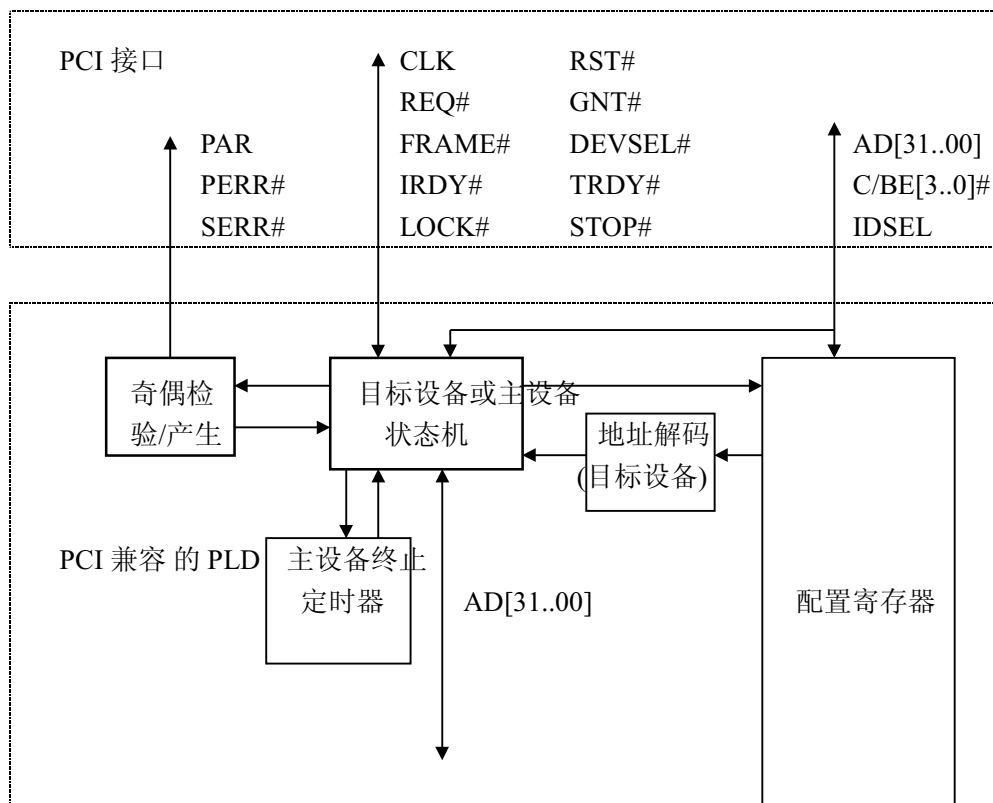
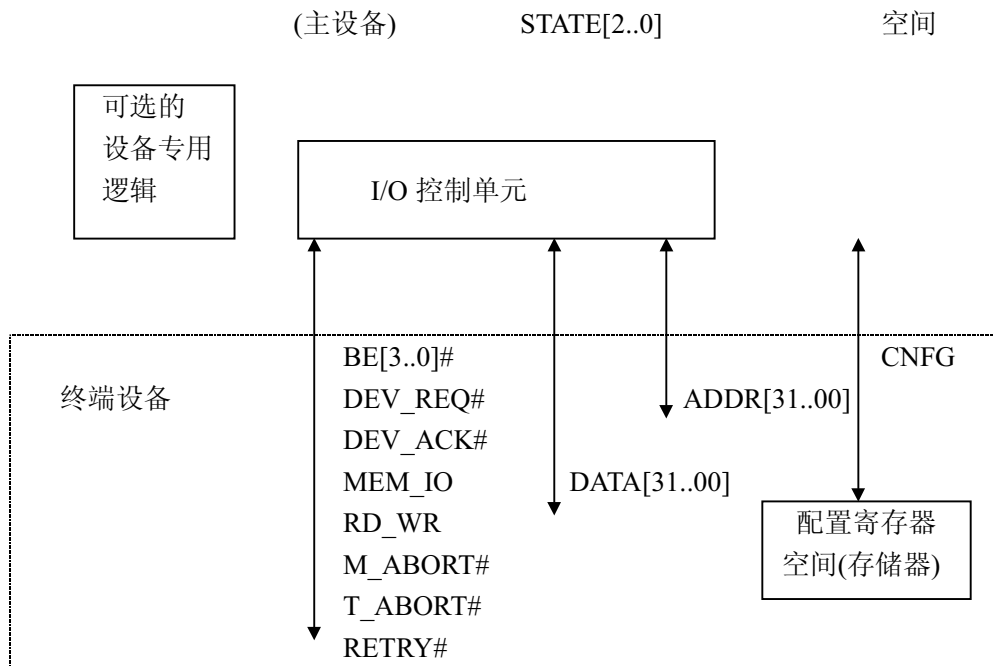


图 6.2 PCI 控制的结构

总线主设备/目标设备的结构如图 6.2 所示





第七章 PCI 总线在具体操作系统中 (Linux) 的应用

一、简介

之所以选择 Linux 作为我们介绍 PCI 在具体操作系统中应用的问题，是因为它的源码公开性，同时我们可以很容易地找到它。看了头文件以后，如果有兴趣，当然可以看这些函数的具体实现，当然，我们看到的是 C 语言，而不必像在 Windows 里边那样去看汇编代码.....

二、必要的头文件、宏以及函数

下面的这些头文件、宏以及函数都被 PCI 设备的驱动程序用来查询它们自己的硬件设备是否存在。

```
#include <linux/config.h>
```

设备驱动程序需要获得内核是否支持 PCI 函数。通过包含这样的头文件，驱动程序可以使用 CONFIG_XXX 宏，包括了 CONFIG_PCI 宏。在 1.3.73 内核以后，这个头文件包含在 <linux/fs.h> 中，如果保持向后兼容性，那么我们应该显式地包含这个头文件。

CONFIG_PCI

如果内核中包含了对于 PCI BIOS 的支持，那么这个宏将会被定义。当然，在编译内核的时候如果发现自己的计算机上根本就没有 PCI 总线，那么完全可以在编译选项当中去掉 PCI 选项，这样一来系统占据的内存就会减少。如果 CONFIG_PCI 没有定义，那么我们下面介绍的这些函数都将不可以使用。驱动程序在使用预编译指令当中应该将所有对于 PCI 支持函数的调用放在 #ifdef CONFIG_PCI 和 #endif 这样的预编译条件当中，否则在你加载驱动程序的时候一定会出现 "unresolved symbol" 这样的错误。

```
#include <linux/bios32.h>
```

这一头文件将会包含我们所介绍的所有函数的原型。同时还会定义函数返回的错误代码的符号值。

```
Int pcibios_present(void)
```

本函数的返回值将会告诉调用它的程序到底计算机是否支持 PCI。如果 BIOS 支持 PCI，那么返回值将会是真。要知道即使系统支持 PCI，对于内核而言，PCI 功能仍然是运行时可调的。因此，建议在调用我们下面将要介绍的函数之前，先检查一下这个函数的返回值。

```
#include <linux/pci.h>
```

这个头文件定义了下面函数使用的所有数值的助记符号名。当然，并不是所有的设备的 ID 都在这个文件中列出了（例如我自己做的设备，在我没有改动这个头文件以前，☺）。

```
Int pcibios_find_device (unsigned short vendor,unsigned short id,unsigned short index,unsigned char *bus,unsigned char *function);
```

如果定义了 CONFIG_PCI，同时 pcibios_present 也为真，那么可以使用这个函数从 BIOS 获取关于特定设备的信息。Vendor（即制造商 ID）和 Id（即设备 ID）用于确定设备。Index 用于支持具有相同的 vendor 的设备。对于这个函数调用将会返回设备在总线上的位置以及一些函数指针。如果返回值为 0 表示成功，否则表示失败。

```
Int pcibios_find_class (unsigned int class_code, unsigned short index, unsigned char *bus, unsigned char *function);
```

本函数使用方法和上函数类似，它用于寻找特定类别的 PCI 设备。参数 class_code 的传递形式为：16 位的类别寄存器左移 8 位，这与 BIOS 接口使用类别寄存器的方式有关。返回值为 0 表示成功。

```
Char *pcibios_strerror(int error);
```

这个函数用来解析 PCI 错误代码（例如 pcibios_find_device 返回的），其返回值是一个字符串。因为我们可能需要在返回值既不是 PCIBIOS_SUCCESSFUL(0)，也不是 PCIBIOS_DEVICE_NOT_FOUND 的时候（这是所有的设备都被查找过以后我们应当得到的返回错误值），将具体的错误信息打印出来。

我们下面将会提供一段驱动程序在加载时（insmod）检测设备所使用的典型代码。如上所述，检测过程可以基于制造商 ID 或者基于设备的类别。不过不管是何种情况，驱动程序不可以存储 bus 或者 function 的值，因为他们在后边确定设备的时候将会用到。Function 的前 5 位确定设备，后 3 位确定功能。

下面代码中，每个与特定设备相关的符号都加上前缀 temp_。

如果驱动程序可以使用唯一的 vendor 和 id 的组合来确定设备，那么下面的循环调用可以用来初始化驱动程序。

```
#ifdef CONFIG_PCI
```

```

        if (pcibios_present()) {
            unsigned char bus,function;
            int index, result;

            for (index=0; index < TEMP_MAX_DEV; index++) {
                result = pcibios_find_device(TEMP_VENDOR, TEMP_ID, index, &bus,
&function);
                if (result != PCIBIOS_SUCCESSFUL)
                    break;
                temp_init_dev (bus,function);
            }

            if (result != PCIBIOS_DEVICE_NOT_FOUND)
                printk (KERN_WARNING "temp: pci error: %s\n",
pcibios_strerror(result));
        }
        if (index == 0)
            return -ENODEV;
    #else
        return -ENODEV;
    #endif;

```

在很多情况下，驱动程序需要同时处理 PCI 和 ISA 设备，在这种时候，驱动程序仅在没有检测到 PCI 设备或者 CONFIG_PCIBIOS 返回值为失败的时候才去检测 ISA 设备。

当使用 pcibios_find_class 的时候，temp_init_dev 要比上边提到的函数完成更多的功能，因为在一个地方简化了，在另外的地方当然要补足。

三、访问配置空间

在驱动程序检测到设备之后，它一般要对三个地址空间进行访问：内存空间、IO 空间和配置空间。其中，对配置空间的访问对驱动程序来说最为重要，它是发现设备被映射到内存空间或 IO 空间的地址的为一方法（因为我们要根据 BAR 的值来确定）。

对于驱动程序来说，配置空间可以通过 8 位、16 位或者 32 位的读写指令来访问。相关函数的原型存放在<linux/bios32.h>当中：

```

int pcibios_read_config_byte (unsigned char bus, unsigned char function, unsigned char
where, unsigned char *ptr);
int pcibios_read_config_word (unsigned char bus, unsigned char function, unsigned
char where, unsigned char *ptr);
int pcibios_read_config_dword (unsigned char bus, unsigned char function, unsigned
char where, unsigned char *ptr);

```

从由 bus 和 function 确定的设备配置空间读取 1、2 或者 4 个字节。参数 where 表示是从配置空间开始处的字节偏移。从配置空间读取的值通过 ptr 返回。这些函数返

回值都是错误代码。字和双字函数将以 little-endian 方式读取得数据转换成处理器本身认可的字节序。所以不需要调用转换字节序的函数。

```
int pcibios_write_config_byte (unsigned char bus, unsigned char function, unsigned char where, unsigned char val);
```

```
int pcibios_read_config_word (unsigned char bus, unsigned char function, unsigned char where, unsigned short val);
```

```
int pcibios_read_config_dword (unsigned char bus, unsigned char function, unsigned char where, unsigned int val);
```

向配置空间写 1、2 或者 4 个字节。设备仍然由 bus 和 function 来确定，需要写入的值由 val 传递。字和双字函数在向外设写出之前将数值转换成 little-endian 字节序。

访问配置变量我们推荐的方法是使用在<linux/pci.h>中定义的符号名。例如，下面的两行程序通过给 pcibios_read_config_byte 的 where 传递一个符号常量来获取设备的 revision_id。

```
Unsigned char * temp_get_revision ( unsigned char bus, unsigned char fn)
{
    unsigned char *revision;
    pcibios_read_config_byte(bus, fn, PCI_REVISION_ID, &revision);
    return revision;
}
```

四、访问 IO 空间和内存空间

一个 PCI 设备可能会实现六个基地址寄存器。这些基地址寄存器中有用的可能一个都没有。每个寄存器所指定的地址可能是内存地址也可能是 IO 空间。大多数设备用一个内存区域代替 IO 空间，因为有些处理器根本没有 IO 空间的概念（例如 Alpha 处理器本身就是以内存空间的一段代表 IO 空间），同时还因为 PC 上的 IO 空间也是非常有限的资源。内存和 IO 空间的区别通过基地址寄存器的最低位来实现。

检测一个 PCI 区段空间（不管是内存空间还是 IO 空间）可以通过在<linux/pci.h>中的位掩码来简化：是内存空间的时候 PCI_BASE_ADDRESS_SPACE 将会被置位；PCI_BASE_ADDRESS_MEM_MASK 为内存区域掩码掉配置位；PCI_BASE_ADDRESS_IO_MASK 为内存区域掩码掉配置位。同时 pci 规范还强调地址区域必须按照顺序分配，这就意味着如果前面有基地址寄存器没有使用的话，后边的基地址寄存器一定是没有意义的。

检测 PCI 区域当前位置和大小的典型代码如下：

```
static u32 addresses[] = {
    PCI_BASE_ADDRESS_0,
    PCI_BASE_ADDRESS_1,
    PCI_BASE_ADDRESS_2,
    PCI_BASE_ADDRESS_3,
    PCI_BASE_ADDRESS_4,
    PCI_BASE_ADDRESS_5,
```

```

    0
};

int pciregions_read_proc (char *buf, char **start, off_t offset, int len, int unused)
{
    #define PRINTF(fmt,args...) sprintf (buf+len, fmt, ##args)
    len =0;

    for ( I=0; addresses[I]; I++) {
        u32 curr, mask;

        pcibios_read_config_dword (bus, fun, addresses[I], &curr);
        cli();
        pcibios_write_config_dword (bus, fun, addresses[I], ~0);
        pcibios_read_config_dword (bus, fun, addresses[I], &mask);
        pcibios_write_config_dword (bus, fun, addresses[I], curr);
        sti();

        len += PRINTF ("\tregion %I: mask 0x%08lx, now at 0x%08lx\n", I
(unsigned long) mask, (unsigned long) curr);

        if (!mask) {
            len += PRINTF ("\tregion %I not existent\n",I);
            break;
        }

        if (mask & PCI_BASE_ADDRESS_SPACE) {
            type = "I/O"; mask &= PCI_BASE_ADDRESS_IO_MASK;
        } else {
            type = "mem"; mask &= PCI_BASE_ADDRESS_MEM_MASK;
        }

        len += PRINTF ("\tregion %I : type %s, size %I \n", I, type , ~mask+1);
    }
    return len;
}

```

五、PCI 设备的中断

至于中断，PCI 的是比较容易处理的。计算机的硬件已经为每个设备分配了一个唯一的 interrupt 号，驱动程序只需去使用这个 interrupt 号就可以了。interrupt 号存放在配置寄存器 60 当中 (PCI_INTERRUPT_LINE)，它的大小是一个字节。这允许最多 256 条 interrupt 线，但是实际上那就要依赖于我们使用的 CPU 支持的 interrupt 数目。

如果设备不支持中断，寄存器 61 (PCI_INTERRUPT_PIN) 为 0；否则就会非 0。不过由于驱动程序一般情况下都是在已经知道设备是否支持中断的时候才进行编写的，所以没有必要去检查这个寄存器。

因此，处理 PCI 的中断只需要读取配置寄存器的第 60 字节以获得中断号，如下面的代码。

```
Result = pcibios_read_config_byte ( bus, fnct, PCI_INTERRUPT_LINE, &my_irq);  
If (result)  
.....
```

到这里为止，我们已经对于具体的操作系统如何处理 PCI 设备有了一个大致地了解。实际上，驱动程序完成的工作就是帮助 PCI 设备获得它所需要的所有的资源，然后由操作系统，硬件和 PCI 设备完成工作。

其他还有一些没有涉及到的问题，比如 PCI 总线的电气特性，66MHz PCI 总线，64bit PCI 总线扩展等等，请大家参考 PCI 2.2 规范。