目录

第4章 总线和接口

一个完整的计算机硬件系统中，不同部件间，如从硬盘到处理器、从 CPU 到内存，从内存到显卡，都需要进行数据传输。这些部件两两之间都用单独的通道连接是不现实的。总线提供了一种可行思路，即，不同计算机部件均接到一个公共的通道上，通过某种预先约定的方式协调大家对通道的占用。这种协调多个不同部件通过公共通道交换数据的机制就是总线。直观地看，总线是一组导线（铜线、光纤等），计算机内外不同功能单元通过这组导线连接到了一起。这组导线中，一些用于传输数据，一些用于传输控制信号（如时钟、握手信号等），还有一些用来传输地址信号。而从功能上看，总线又可视为一个通信系统，提供了计算机不同部件之间、甚至是不同计算机之间相互传输数据的能力。这种通信能力不仅仅依赖于直观上的一组导线，还需要相关电路模块、相关软件模块的支持。关于总线的基本知识点在 4.1 小节中讨论，4.2 和 4.3 小节分别介绍用于芯片内部和芯片外部的典型总线。

更进一步看，当处理器与外设发生数据交换时，并不只是简单地把不同电路单元的信号线对接在一起。外设的功能是多种多样的，有些外设是输入设备，有些外设是输出设备，有些外设既作输出设备又作输入设备，还有些外设作为检测设备或控制设备。外设的输入信号形式也是多种多样的，如数字式的、模拟式的，有的仅一根信号线、有的有数十根信号线。且不同种类外设的工作速度也不同，如文档编辑的时键盘每秒钟产生多个字符输入，而鼠标数秒钟才有一次点击。简言之，外设种类繁多，不同外设在速度、信号形式等方面存在巨大的差异。因此，要保证 CPU 与外设之间可靠地传输数据，必须在 CPU 与外设之间加入合适的中间环节，来协调 CPU 与外设的差异，这个中间环节就是 I/O（Input/Output，输入/输出）接口。此部分知识点在 4.4 小节讨论。

总线与接口的区别和联系：①接口包含电路单元之间的硬件电路，该电路的作用是连接不同设备，其功能需要按照约定规则设计。而总线不仅包括分时共享传输线路和相关电路，也包括传输和管理信息的规则（即协议）。注意，挂接在总线上的电路单元一定要有相应的接口，例如，USB 总线上的设备必须支持 USB 接口。②从适用场景看，只要两个电路单元连接，一定需要接口电路。如果仅有两个电路单元，或者线路上只有一个电路单元会发送数据时，可以不需要总线定义的共享通道规则，而由简单的逻辑电路协调两个电路单元。而如果是多个电路单元基于一套传输线路两两形成连接通道，那么每个电路单元不仅要有接口电路，还需要按照总线协议来规范多个电路单元共享传输线路的方式。③广义的接口还包含接口电路相应的驱动程序，不同外设接口电路和驱动程序是不同的，同一种外设连接到不同的计算机时接口电路和驱动程序也不同。鉴于上述总线和接口的密切联系，在有些资料中并不严格区分总线和接口的概念，但仍作为两个概念予以表述。在学习的过程中，要注意通过案例分析接口和总线技术的适用范围，比较两类技术之间的差异和联系。

简言之，本章讨论计算机系统中微处理器与其他外部功能单元之间、或者不同外部功能单元之间实现互联的原理、方法与相关实现技术。

4.1 总线技术

本节介绍计算机系统总线的分类、结构，阐述多个模块争用共享线路资源的方法，分析

不同模块通过总线进行信息交换的基本过程。

4.1.1 总线技术概述

总线是计算机系统中的公共信息传输通道，由计算机系统内各个部件所共享。总线包含一组公用信号传输线，服务于模块与模块之间、部件与部件之间、设备与设备之间的信息传递。总线的基本特点在于其共享性，即总线可以同时挂接多个模块、部件或设备。单独服务某两个模块、部件或设备之间的专用信号连接线，不能称为总线。依据计算机系统内部不同部件的功能、位置，往往在不同场合有模块、部件、设备等不同称呼，为方便陈述，本章后续内容统称其为模块。

为什么计算机系统要用总线连接各个模块呢？考虑一个具有 n 个电路模块的计算机系统，若各模块采用两两直接连接的方式，则实现所有模块互联需要 $n\times(n\text{-}1)/2$ 组连接线。但采用总线连接后，就只需要一组连接线，所有的模块都直接连接到总线上。故计算机系统都采用总线结构，以降低计算机系统内部电路连接的复杂度。

1. 总线的概念

如果说计算机的主板（Mother Board）是一座城市，那么总线就像是城市里的公共汽车（bus），能按照固定行车路线，来回不停地运输比特。每条线路在一个时间单元内仅能传输一个比特，采用多条线路才能传送更多数据。总线可同时传输的数据位数就称为总线宽度（Width），以比特为单位，总线宽度越大，传输速率就越高。当然，受到电路特性、电路板运行的周边环境的影响，总线宽度不能无限制地大。通常，用总线带宽来说明总线所能达到的最高传输能力。总线带宽（即单位时间内可以传输的总数据量）定义为：总线带宽=总线时钟频率×总线宽度÷8（Bytes/sec）。

从物理上看，总线是一种共享的传输媒介。一方面，对连接到同一套总线上的多个模块而言，任何一个模块所传输的信号可以被其他模块所接收。另一方面，若连接在同一总线上的两个或多个模块在同一时间内都输出各自的信号，不同的信号重叠在一起就会引起混淆。因此，在任何时刻，连接到总线上的多个模块中只能有一个模块输出信号，而其他模块只能处于接收状态。可以利用“输出高阻态”的方式来隔离单个模块对总线的影响，即，物理上连接，但电气上是断开的。当总线空闲时，所有模块都以“输出高阻态”的方式连接在总线上。当某个模块要与其他模块通信时，发起通信的模块驱动总线，发出地址和数据。其他以“输出高阻态”方式连接在总线上的模块如果收到与自己相符的地址信息后，就开始接收总线上的数据。发送模块完成发送后，再将总线让出（输出变为高阻态）。

那么，如何避免不同模块在同一时间向总线输出信号呢？这就需要总线控制器来协调线路资源的使用。总线控制器的功能包括：指定哪个模块可在总线上发送数据，指定数据的传送方向，指定数据传送的源和目的。在总线控制器的协调下，各个不同功能模块进行有序的数据传送。常把包含总线控制器功能的模块称作主控模块，在主控模块协调下被动工作的模块被称作从属模块。实际中，很多模块既可以工作在主控模块方式，也可以工作在从属模块方式。

2. 总线的分类

可以从不同的角度进行总线分类。例如，按照总线在计算机系统中所处的位置，可分为

片内总线、片间总线、系统总线、系统外总线。按照总线的功能，可分为地址总线、数据总线、控制总线。按照数据传送格式，可分为并行总线和串行总线。按照总线上是否采用共有时钟，可分为同步总线和异步总线。按照是否复用，可分为时分复用总线和非复用总线。

1） 按总线位置分类

从位置在芯片内或芯片外看，可分为片内总线和片间总线。片内总线（in chip bus）是指芯片内部用于连接各功能单元电路的信息通路。如微处理器芯片内部连接 ALU、寄存器、控制器等部件的总线。片间总线（chip bus），也称为芯片总线（component-level bus）或者局部总线（local bus），连接微处理器芯片和外围芯片。

从位置在计算机系统内或系统外看，可分为内总线和外总线。内总线（internal bus），又称为板级总线（board-level bus）或系统总线（system bus），用于系统内部各高速模块之间的互连。例如 ISA、EISA、PCI、PCI Express 等。外总线（external bus），又称 I/O 总线或通信总线（communication bus），用于计算机之间、或计算机与外设之间的互连。例如 SCSI、USB 等。

近年来，随着 SoC（System On Chip，片上系统）的迅速发展，在单颗芯片内不仅集成了 CPU，也集成了存储器，甚至集成了过去以外设形式存在的 GPS、蓝牙模块等。上述片内总线、片上总线、内总线、外总线的区分方式在基于 SoC 的嵌入式系统中逐渐不再使用，而采用术语“片上总线（on-chip bus）”来描述芯片内部（事实上涵盖了传统内总线及外总线的定义）使用的总线。ARM 处理器采用的 AMBA 总线就是一种典型的片上总线。

在同一计算机系统内，往往片内总线、系统总线、系统外总线同时存在，大部分计算机系统中大都采用了总线多级分层结构，如图 4.1 所示。图 4.1 所示微处理器芯片的片内总线连接了 CPU 核心与高速缓存（cache）。高速缓存控制器一方面连接了微处理器的片内总线，还与连接主存储器的系统总线相连。高速缓存有利于减少微处理器对主存储器的访问次数，可以隔离超高速的片内总线和相对较慢的系统总线。这样的电路结构设计中，微处理器的片内总线应对 CPU 核心与高速缓存之间的数据传输；系统总线则提供了高速缓存和主存储器之间的数据传输通道，有利于提高计算机系统的整体性能。

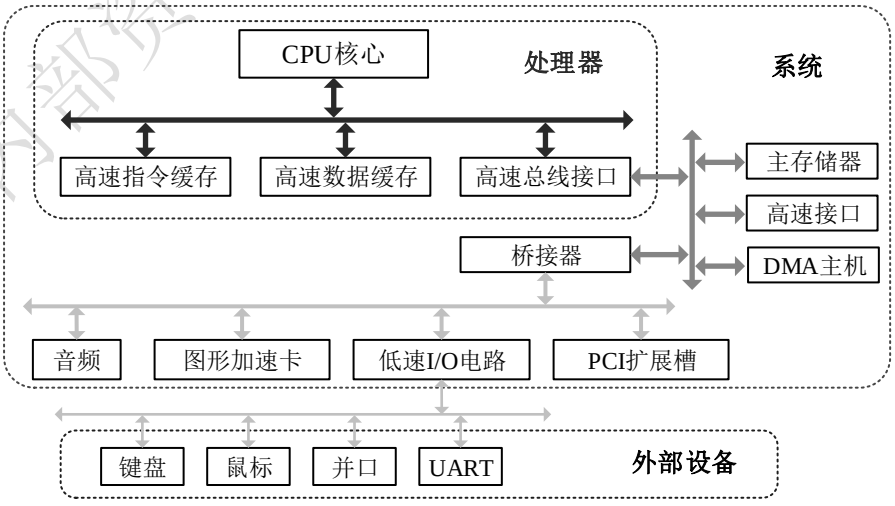


图 4.1 总线的层次结构

图 4.1 中系统与外部设备之间的总线即为外部总线。由于外设种类繁多，速率不一致，在很多计算机系统中外部总线也采用层次化的结构。如，图 4.2 显示了 CPU 与存储器之间，CPU/存储器与高速外设之间，以及它们与低速外设之间三种不同层次的总线。分层外部总线的基本设计思路是将速率分等级，让速率等级一致的单元挂接到同一总线上，不同层次之间通过特定的接口实现总线间的互联。

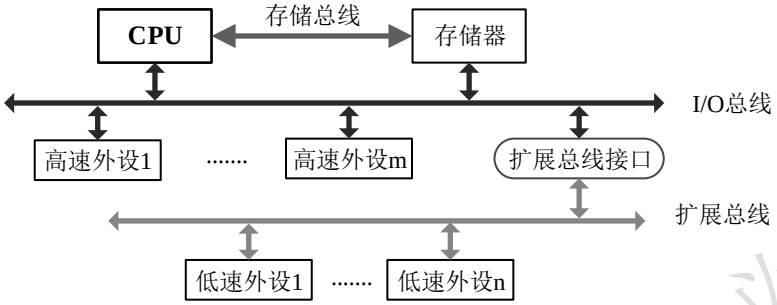


图 4.2　总线的多级分层结构

2） 按总线功能分类

从总线的功能来看，总线一般分为三大类，即地址总线（AB，Address Bus）、数据总线（DB，Data Bus）和控制总线（CB，Control Bus），如第 2 章图 2.2 所示。这三类总线分工明确，“地址总线 AB”专门用来传送地址；“控制总线 CB”用来传送控制信号和时序信号；“数据总线 DB”用于传送数据信息。计算机系统的各种操作，通常需要地址总线、数据总线和控制总线协同配合才能实现。

“地址总线 AB”主要完成系统地址信号的传送，一般为单向传送总线，信号通常从 CPU 发出，送往总线上所连接的各个模块。地址信号线用来指定数据总线上数据的去向与来源。例如，如果 CPU 希望从存储器中读取一个字的数据，便将所期望的字地址放到地址总线信号线上。地址总线的宽度决定了系统最大存储器空间寻址范围。如，20 位宽度的地址总线可寻址空间为 2^{20}=1M，可寻址 1M 个存储单元。若数据总线为 8 位宽，此时可寻址 1M 字节即 1MB；而若数据总线为 16 位宽，此时可寻址 2MB。当然，地址总线也可用于 I/O 端口的寻址。通常地址总线的高位部分通过译码电路后，用于选择一个特定的模块（或 I/O 的端口位置），而低位部分用于选择模块内的存储单元。关于 I/O 端口地址，详情可参阅 4.4.1 5 小节。

“数据总线 DB”用于传送数据信息。数据总线通常为双向三态形式的总线，既可以把 CPU 的数据传送到存储器或 I/O 接口等其它部件，也可以将其它部件的数据传送到 CPU。数据总线的位数（宽度）是计算机系统的一个重要指标，通常与微处理的字长相一致。例如 Intel 8086 微处理器字长 16 位，其数据总线宽度也是 16 位。需要指出的是，数据的含义是广义的，它可以是真正的数据，也可以是指令代码或状态信息，有时甚至是控制信息。因此计算机工作过程中，数据总线上传送的并不一定仅仅是真正意义上的数据。如果数据总线的宽度是 8 位，而每条指令码是 16 位，那么通过该数据总线获取一条指令 CPU 需要访问存储器 2 次。

“控制总线 CB”用于传输完成各项操作所需要的控制信号，包括辅助数据传送、进行

总线仲裁、执行中断控制等。控制信号中，有的是 CPU 送往存储器和 I/O 接口电路的，如读/写信号，片选信号、中断响应信号等；也有是其它部件反馈给 CPU 的，如中断请求信号、复位信号、总线请求信号、设备就绪信号等。换言之，控制总线用于控制数据和地址总线的控制。由于连接到数据总线和地址总线上的多个模块共享使用总线资源，因此必须对它们的使用进行有效管理。控制信号负责总线上所连各个模块之间的时序传输和联络控制，使数据总线和地址总线的使用有序化。由于涉及到不同的电路单元，有些控制信号仅连接了一个外设模块，有些控制信号同时连接多个模块，有些控制信号线为单向传送，有些控制信号线为双向传送，有些控制信号线甚至在不同的时间段有不同的功能定义。一般来说，总线信号线中，除电源线、地线、数据线和地址线外的所有信号线都归纳为控制总线。

3） 按数据传输方式分类

按照传输数据的方式，总线可以分为串行总线（serial bus）和并行总线（parallel bus）。并行总线有多条数据传输线，可以同时并行传送多个二进制位，一般为一字节或多字节，其位数为该总线的宽度，如图 4.3 (a)所示。过去很长一段时间内，内总线一般为并行总线；但近十几年串行的系统内总线也得到了广泛应用。

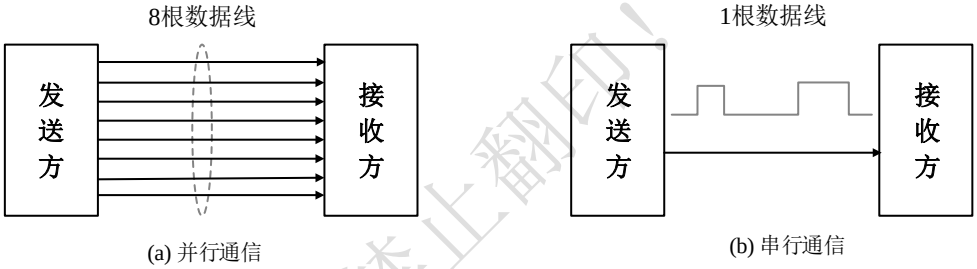


图 4.3 并行数据传送和串行数据传送示意图

串行总线只用一根信号线（如果传输的是差分信号则是两根信号线）来传输数据，故只能串行地逐位传送数据。如图 4.3(b)所示。系统外总线多采用串行总线，由于传输信号线数量好，可节省线路成本，便于实现远距离传输。有些串行总线有两条、甚至四条数据线，分别实现两个方向的数据传输，称为双工通信。但从每次传输来看，数据仍然是逐位传输的。

4） 按时序控制方式分类

根据数据传输时收发双方时钟信号是否独立，总线可分为同步总线和异步总线。同步总线需要独立的信号线用来传输时钟，而异步总线的时钟信号是从本地电路或从数据中提取出来的。例如，嵌入式系统常用的串行总线中，SPI、I^2C 是同步总线，而 RS232 为异步总线。

在同步总线上传送数据时，需要依据收发双方公用的时钟信号来定时。一般设置有同步定时信号，如时钟同步、读写信号等。而异步总线中则没有公用的时钟信号，传输数据的方式也与同步总线有较大差异。关于同步传输和异步传输的区别，将在 4.4.4 小节中进一步分析。

5） 按时分复用方式分类

有的系统中，数据总线和地址总线是分开的。而有的系统中两者是复用的，即总线在某些时刻出现的信号表示数据而另一些时刻表示地址。往往一些功能简单的单片机的地址总线和数据总线是复用的，而一般 PC 中的总线则是分开的。故从总线是否时分复用看，把总线上传输的信号分为时分复用信号和非复用信号。

非复用信号被定义为一种永久不变的功能，如物理上独立的数据总线和地址总线，数据总线上传输的始终是数据信号，地址总线上则始终传输地址信号。中断请求信号线上传输的则是中断源向处理器发出的中断申请。采用非复用信号可以使计算机获得较高的传输速率。

复用信号是指在一条信号线上，分时传输不同功能的信号。如何区分信号线上到底传输的是什么功能的信号呢？通常，需要定义专用的控制信号线，控制信号线上的信号用来指示当前总线工作于哪种模式。

在地址总线和数据总线复用的系统中，常用一条（甚至多条）控制信号指示复用信号线上传输的信号是地址信号还是控制信号。在总线开始传输时，复用总线上传输的是地址信号，同时地址有效控制信号为有效电平，与该总线相连的所有模块在规定的时间内都接收该地址，并通过译码电路判断是否为被寻址的模块。然后，复用总线上的信息发生切换，由地址信息变为数据信息，同时地址有效控制信号变为无效电平，完成地址传输的操作，也意味着数据传输操作的开始。如图 4.4 所示。这种时分复用方案的优点是，可以用较少的信号线完成数据传输的功能，有利于节省布线空间，降低成本。其缺点是电路复杂，每个模块都需要复接电路，且传输性能下降。

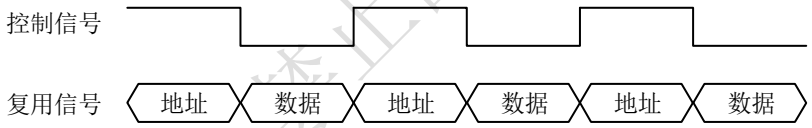


图 4.4 时分复用信号示意图

3. 总线的结构

从计算机体系结构来看，有两类总线结构，即单总线结构和多总线结构。在多总线结构中，又以双总线结构为主。

1） 单总线结构

在单总线结构中，CPU 与主存之间、CPU 与 I/O 设备之间、I/O 设备与主存之间、各种设备之间都通过同一套总线交换信息。单总线结构的优点是控制简单，扩充方便。由于所有设备部件均挂在单一总线上，单总线结构只能分时工作，即同一时刻只能在两个模块之间传送数据。这种限制导致系统总体数据传输效率受到制约，这是单总线结构的缺点。典型单总线结构如图 4.5 所示，图中 CPU、存储器、I/O 接口均挂接在相同的总线上。

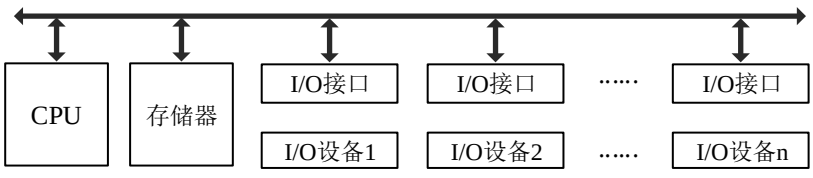


图 4.5　经典的单总线结构

2） 双总线结构

双总线结构又分为面向 CPU 的双总线结构和面向存储器的双总线结构。面向 CPU 的双总线结构如图 4.6 所示。其中一组总线是 CPU 与主存储器之间进行信息交换的公共通路，称为存储总线。另一组总线是 CPU 与 I/O 设备之间进行信息交换的公共通路，称为输入/输出总线（或称 I/O 总线）。外部设备通过连接在 I/O 总线上的接口电路与 CPU 交换信息。

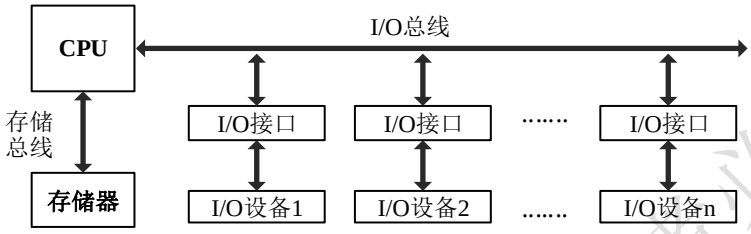


图 4.6　面向 CPU 的的双总线结构

由于在 CPU 与主存储器之间、CPU 与 I/O 设备之间分别设置了总线，从而提高了微机系统信息传送的速率和效率。但外部设备与主存储器之间没有直接的通路，它们之间的信息交换必须通过 CPU 才能进行中转，会降低了 CPU 的工作效率，这是面向 CPU 的双总线结构的主要缺点。一般来说，外设工作时要求 CPU 干预越少，外设进行数据传输时 CPU 占用率就越低，有利于系统整体性能的提升。

面向存储器的双总线结构保留了单总线结构的优点，即所有设备和部件均可通过总线交换信息。与单总线结构不同的是在 CPU 与存储器之间，专门设置了一条高速存储总线，使 CPU 可以通过它直接与存储器交换信息。面向存储器的双总线结构信息传送效率较高，这是它的主要优点。但 CPU 与 I/O 接口都要访问存储器时，仍会产生冲突。这种总线结构如图 4.7 所示。

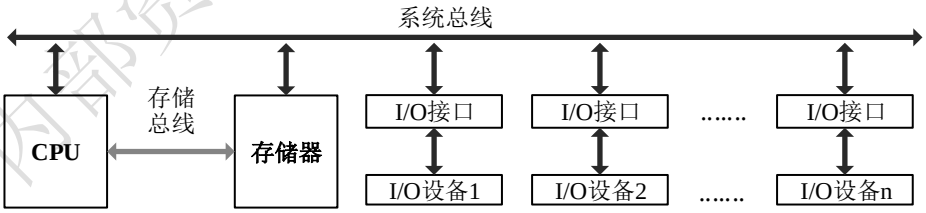


图 4.7　面向存储器的的双总线结构

3） 三总线/四总线结构

典型的三总线结构计算机系统如图 4.8 所示。这个结构为高速 I/O 设备建立了一个到达主存的高速通道，称为 DMA（Direct Memory Access，直接存储访问）总线。这个结构中任意时刻只能有一套总线在使用，主存总线与 DMA 总线不能同时访问主存，I/O 总线只有在 CPU 执行 I/O 指令时才能用到。

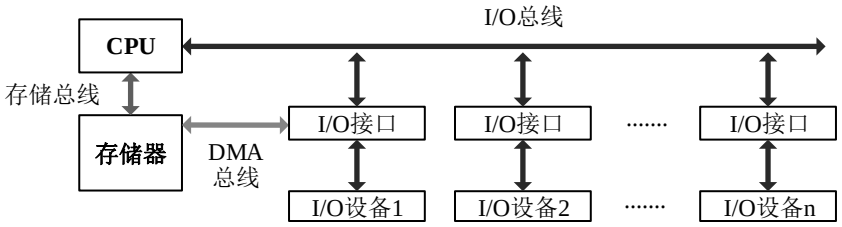


图 4.8　典型三总线结构

如图 4.9 所示是另外一种总线形式：处理器与高速缓存之间有一个局部总线，它将 CPU 和高速缓存或者更多的局部设备连接起来。高速缓存不仅连接到局部总线上，还直接连接到系统总线上，这样缓存就可以通过系统总线和主存交互。与此同时，I/O 设备与主存之间也可以通过系统总线交互，而不必通过 CPU。还有一条扩展总线，它将网络、硬盘接口、串行接口等接口连接起来，并且通过这些接口又可以与各类 I/O 设备连接，因此可以支持很多的外接设备。事实上，图 4.2 的总线分层结构就是图 4.9 所示的三总线结构。

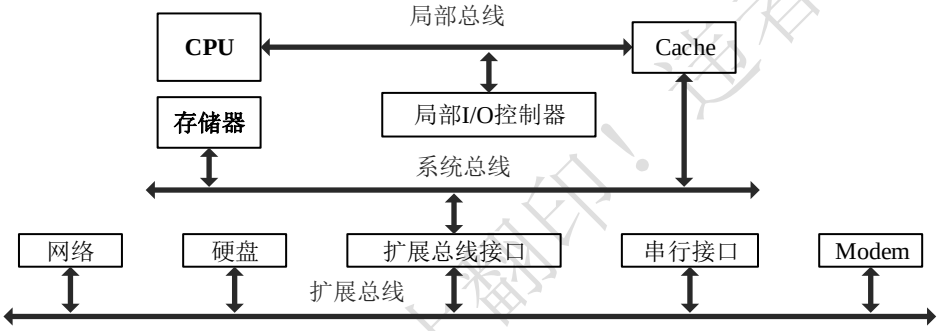


图 4.9　带缓存控制的三总线结构

为了进一步提高 I/O 设备的性能，使其更快地响应命令，又出现了四总线的结构，典型的四总线结构如图 4.10 所示。相比三总线结构，又增加了一条与系统相连的高速总线。高速总线上挂了很多高速的 I/O 设备，它们通过高速总线桥或者高速缓冲器与系统总线和局部总线连接。而一些较低速度的设备则放在了扩展总线上。这种结构对于高速设备而言，它们又比扩展总线上的设备更加靠近 CPU，有利于提升高性能设备与 CPU 的效率。并且，局部高速总线和扩展总线的速度以及各自的信号线的定义可以不同，有利于扩展不同类型的外设。

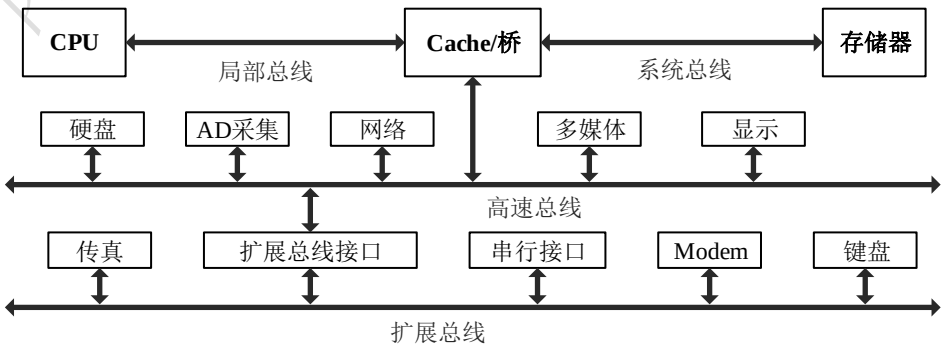


图 4.10　典型的四总线结构

4.1.2 总线仲裁

总线上的设备一般分为总线主设备（主控模块）和总线从设备（从属模块）。总线主设备是指具有控制总线能力的模块，通常是 CPU 或以 CPU 为中心的逻辑模块。总线主设备在获得总线控制权之后能启动数据的传输；与之相对应的总线从设备，能够对总线上的数据请求做出响应，但本身不具备总线控制能力。在一些文献中，常把总线主设备（主控模块）称作总线主机，把总线从设备称作总线从机。

在早期的计算机系统中，一条总线上只有一个主设备，总线一直由它占用，其实现电路较为简单。随着计算机技术及相关应用的发展，多个主设备共享总线的情况越来越普遍，这对总线技术提出了新的要求。在这类系统中，需要解决各个主设备之间资源争用等问题，使得总线电路的复杂性大为增加。

总线仲裁（bus arbitration）就是在多总线主设备的环境中提出来的。总线仲裁又称总线判决，其目的是合理地控制和管理系统中多个主设备的总线请求，以避免冲突。当多个主设备同时提出总线请求时，仲裁机构按预定义的优先级算法来确定由谁获得总线控制权。硬件实现的总线分配逻辑电路称为总线仲裁器（bus arbiter），其电路功能是响应不同主设备的总线请求，通过对分配过程的控制来协调总线资源使用。

依据总线控制单元设置方式的不同，总线仲裁分为两种方式：集中式（主从式）控制和分布式（对等式）控制。集中式控制是指采用专门的总线控制器或仲裁器来分配总线时间，总线控制器或仲裁器可以是独立的模块或器件，也可以集成在 CPU 芯片中。采用集中式控制方式的总线协议简单而有效，但总体系统性能较低。分布式控制指总线控制逻辑分散在连接于总线的各个模块中。采用分布式控制方式的总线协议较为复杂，电路成本较高，但有利于提高 CPU 和总线的利用率。

1. 集中式仲裁

按仲裁机制的不同，集中式控制又可分为串行仲裁、并行仲裁和混合仲裁。

（1）串行仲裁

串行仲裁又称为“菊花链”（daisy chain）仲裁。如图 4.11 所示，常用的三线“菊花链”仲裁电路中的信号线包括：BR（Bus Request，总线请求）信号、BG（Bus Grant，总线允许）信号和 BB（Bus Busy，总线忙）信号。

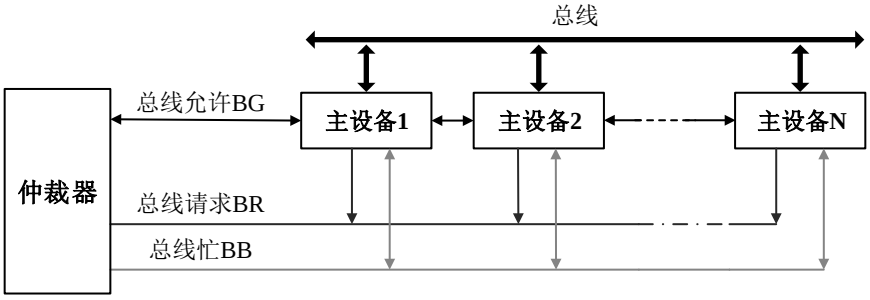


图 4.11 串行仲裁

在串行仲裁方式下，主设备在提出总线请求前，首先检测总线是否忙（BB 信号是否处于无效状态），只有在 BB 信号线无效（总线空闲）时，主设备才能提出总线请求。系统中各主设备的 BR 信号用“线与[1]”方式接到仲裁器的请求输入端。仲裁器在接到 BR 信号后输出 BG 信号，BG 信号通过主设备链向后传递，直到提出总线请求的那个设备为止。请求总线的主设备收到 BG 号后，获得总线的控制权，将 BB 信号置为有效，以通知其他设备总线已被占用，同时使总线仲裁器撤销 BG 信号。主设备的总线操作结束后，撤销 BB 信号，从而允许其他设备重新申请总线。

串行裁的特点是设备使用总线的优先级由它到总线仲裁器的距离决定，最近的优先级最高，最远的优先级最低。串行仲裁优点是：信号线数与设备数目无关，电路实现简单；其缺点是：速度慢，连接方式确定后优先级就固定了，不易更改。

（2）并行仲裁

如图 4.12 所示，在并行仲裁方式中，每个主设备都有独立的 BR 和 BG 信号线，并分别接到仲裁器上。任一主设备使用总线都要通过 BR 信号向仲裁器发出请求，仲裁器按规定的优先级算法选中一个主设备，并把 BG 信号送给该设备。被选中的设备撤销 BR 信号，并输出有效的 BB 信号，通知其他设备“总线已经被占用”。主设备的总线操作结束后，撤销 BB 信号，同时仲裁器在撤销 BG 信号后，根据各请求输入的情况重新分配总线控制权。并行仲的优点是仲裁速度快，优先级设置灵活。但是连接到总线上的设备数量受到限制。

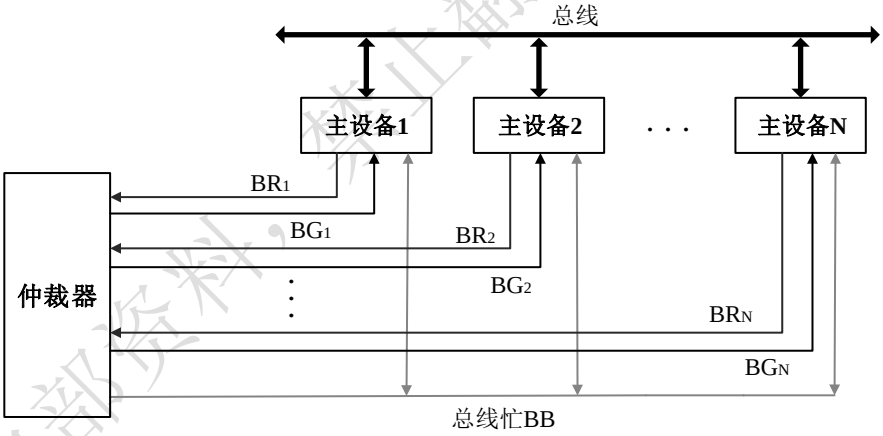


图 4.12　并行仲裁

（3）混合仲裁

混合仲裁又称为多级仲裁，结合了并行仲裁和串行仲裁两种仲裁方法的思路，更为灵活。如图 4.13 所示，图中总线判决器按照并行方式连接着 BR1 和 BR2；所有主设备的总线请求

[1]“线或”和“线与”就是把“或”和“与”的逻辑功能体现到电子线路中。以“二根输入线一根输出线”的电路为例，当二根输入线中任意一根为高时输出线就为高，这就是“线或”；如若二根输入线都为高时输出线才为高，就是线与。

BR 要么连接在 BR1，要么连接在 BR2 上；BG1 按照串行方式连接着一部分主设备，BG2 也按照串行方式连接着另一部分主设备。

所有主设备的总线请求首先经过总线判决器（BR1 或 BR2），总线判决器决定是 BR1 所连主设备还是 BR2 所连主设备获得总线控制权。然后再按串行方式来决定 BR1 上的设备是主设备 2 还是主设备 4、BR2 上的设备是主设备 1 还是主设备 3 应该获得总线控制权。并行请求线 BR1 和 BR2 的优先级由总线判决器内部逻辑确定，同一链路上各设备的优先级则由该设备与总线判决器的远近程度确定。

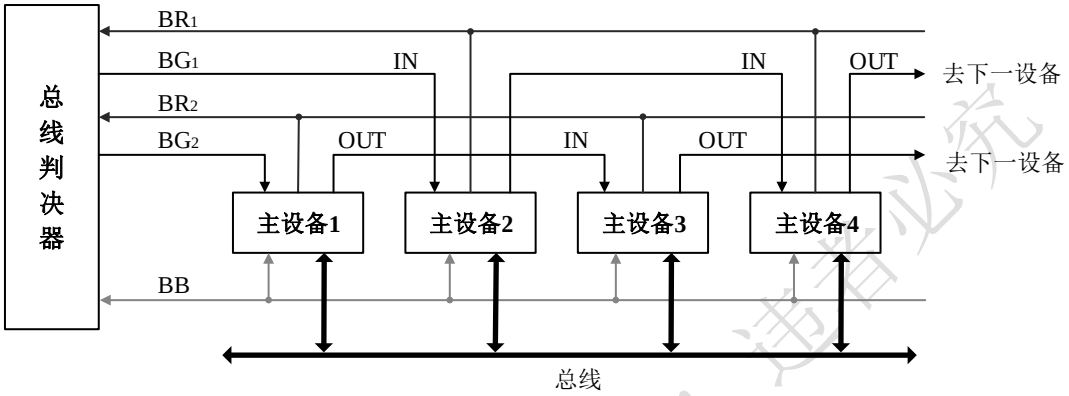


图 4.13　两级混合式仲裁

混合式总线判决系统兼具有串行仲裁和并行仲裁的优点，既有较好的灵活性、可扩充性又可容纳较多的设备而结构也不会过于复杂，且有较快的响应速度，这对于那些含有很多主设备的大型计算机系统来说，是一种很好的折中方案。

2. 分布式仲裁

分布式（竞争式）仲裁方式中，每个设备模块都包含总线访问的控制逻辑，这些设备模块共同作用，分享总线。总裁过程分为申请期和裁决期。在申请期需要请求总线控制权的设备在各自对应的总线请求线上送出请求信号。在裁决期，每个设备将有关请求线上的合成信号取回分析，以确定自己能否拥有总线控制权。每个设备通过取回的合成信息能够检测出其它设备是否发出了总线请求。如果检测到其它优先级更高的设备也请求使用总线，则本设备暂时不能使用总线；否则，本设备就可立即使用总线。

图 4.14 所示为一种自举分布式的总线仲裁实现，图中并没有中心仲裁器，使用多根请求线，每个设备独立地决定自己是否是最高优先级请求者。主设备 1～主设备 4 通过 BR_1～BR_4 进行自举分布式仲裁。其中 BR_1 为总线忙信号，正在使用总线的模块应将 BR_1 置为有效；BR_i 为主设备 i 的总线请求信号线，只有在 BR_1 无效时才能发总线请求。

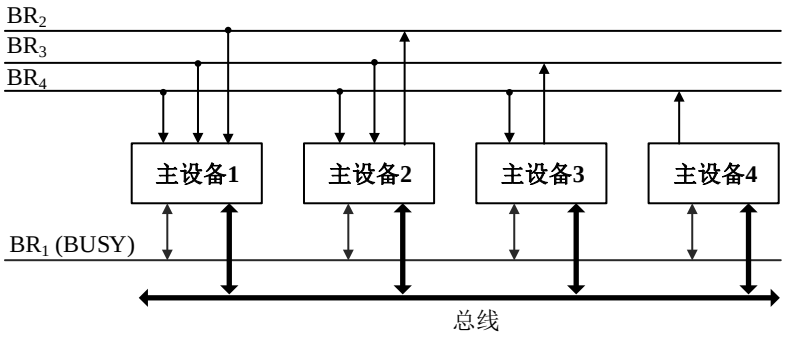


图 4.14　自举分布式仲裁

4.1.3　总线操作与时序

计算机系统中，通过总线进行信息交换的过程称为总线操作。总线设备完成一次完整信息交换的时间称为总线周期（或总线传输周期），如读/写存储器周期、读/写 I/O 端口周期、DMA 周期、中断周期等。在含有多个主控制器的总线系统中，一个总线操作周期一般分为如下四个阶段。

①请求及仲裁（request and arbitration）阶段。拟使用总线的主模块提出请求，由总线仲裁机构决定把下一个总线传输周期使用权分配给哪一个请求源。

②寻址（addressing）阶段。取得总线使用权的主模块通过总线发出本次要访问的从模块（存储器地址或 I/O 端口）地址及有关命令，通知参与传输的从模块开始启动。

③数据传输（data transfering）阶段。主模块和从模块进行数据传输，数据由源模块发出，经数据总线到达目的模块。

④结束阶段。主模块、从模块的有关信息均从总线上撤销，让出总线，以便下一个总线传输周期其他模块能够使用总线。

若计算机系统中仅含一个主设备，则总线始终归该设备所有。此情形不存在总线的请求、分配和撤销的问题，总线传输周期只有寻址和数据传输两个阶段。

总线时序是指总线操作过程中总线上各信号在时间顺序上的配合关系。即主设备和从设备如何在时间上协调和配合，以实现可靠的寻址和数据传送。总线时序分为四种：同步总线时序、半同步总线时序、异步总线时序和周期分裂式总线时序。

1.　同步总线时序

同步总线意味着收发双方的电路模块按照统一的时钟工作，所有信号线上传输的信号都受控于该时钟。一般来说，这种每根信号线遵循相同节拍的同步总线速度较快，且逻辑控制较为容易。但存在两个缺点：一是由于总线偏离[2]问题，总线的物理长度一般很短；二是总线虽可连接不同速度的设备，但工作速度取决于速度最慢的设备。

[2] 总线偏离（bus skew），是指总线中不同信号线的传输速度之间的差别。

CPU 与主存储器之间的总线一般采用同步总线。图 4.15 所示是一种典型的同步总线读存储器操作（参考 PROLOG 公司的 STD 总线[3]）的时序图。需要注意的是，图 4.15 中时序波形中上升沿或下降沿没有画成垂直的，因为信号不能在零时间内将其电平从高电平变为低电平，或从低电平变为高电平，故一般的时序图中将波形的上升沿和下降沿画成呈一定角度的斜线。

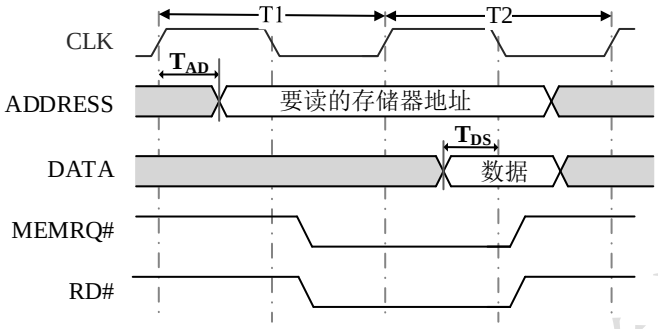


图 4.15　一种典型的同步总线读时序

图 4.15 中各信号线的定义如下。

- CLK，时钟，时钟频率选取 CPU 和存储器中速率低的。
- ADDRESS，地址总线，由 CPU 驱动该信号，存储器读取到地址后选中对应的存储单元；如果是读数据则将该存储单元的数据驱动到数据总线上，如果是写数据则准备将数据总线上的数据写入该存储单元。
- DATA，数据总线，如果是 CPU 读存储器，则存储器将特定存储单元内容驱动到数据总线上；CPU 往存储器写，则存储器将数据总线上的数据写入特定存储单元。
- MEMRQ#，CPU 当前需要读存储器（而非 I/O），低电平有效。
- RD#，CPU 当前执行的是读操作，低电平有效。

T1 周期起始于时钟信号的上升沿。在 T1 周期中，CPU 在地址线（ADDRESS）上给出了它所要读数据的主存储器字地址。地址信号与单个时钟信号不同，地址信号对应多根信号线，因此不能只用一条线来表示，而是用两条且在地址变化时刻由相互交叉的线来示。这表明多条的地址线中，有些地址线是低电平的，而有些地址线是高电平的，将其重叠在一起，则如图 4.15 所示，阴影部分表示这部分的值没有意义。图 4.15 中数据线上的内容直到进入 T2 周期后才有效。

当地址信号稳定后，CPU 发出 MEMRQ#和 RD#，MEMRQ#信号为低电平表示 CPU 要访问的是主存储器（若为高电平则表明访问的是 I/O 端口），RD#信号为低电平表示 CPU 要执行读操作（若为高电平则表明是写操作）。由于主存储器芯片要在地址建立后一定时间后才能输出数据，所以设计在 T2 时钟周期内存储器将 CPU 要读的数据输出。

[3] STD 总线（Standard Data Bus）由美国 PROLOG 公司发布，1987 年初，IEEE 将 STD 总线定为 IEEE-P961 标准总线。STD 总线是工业控制领域最常用的标准总线之一。

在 T2 周期的前半部分，主存储器将读出的数据放到数据线上，然后在 T2 的下降沿，CPU 选通（读）数据信号线，将读出的数据锁存到 CPU 内部的寄存器中。读完数据后，CPU 再将 MEMRQ#和 RD#信号置为无效。如果需要那么 CPU 可以在时钟的下一个时钟周期的上升沿启动另一个总线周期。

时序逻辑电路中，常需要在时钟信号的上升沿或者下降沿触发一些动作（输出其他信号）。图 4.15 中，T1 周期时钟的上升沿、下降沿，以及 T2 周期时钟的上升沿、下降沿，这四个关键的时间点都有一些相应的动作触发。这些被触发的电路动作必须保证先后关系，在总线标准中通常会规定一些参数来描述不同事件间的时间关系，这些参数称之为时序参数。

例如，图 4.15 中 T1 周期的时钟上升沿，CPU 驱动地址到地址总线上。需要经过一小段时间这个输出的地址才可以稳定，这一小段的时间我们记为 T_{AD}，即 T_{AD} 是从 T1 的上升沿开始到地址建立好时的地址建立时间。由于 T1 的下降沿开始要输出 MEMRQ#和 RD#信号，假如 T_{AD} 大于半个时钟周期，那么就会影响到 MEMRQ#和 RD#信号的输出，故而 T_{AD} 不能超过规定的最大值。再如，T2 周期的时钟上升沿，存储器把特定地址的存储单元内容驱动到数据总线（DATA）上。T_{DS} 是 T2 周期时钟下降沿之前的数据已稳定的时间。如果数据稳定需要的时间如果超过半个时钟周期，会导致 $T_{DS} \leqslant 0$，那么原定在 T2 周期时钟下降沿执行的操作就无法按期执行。故而 T_{DS} 不能小于规定的最小值。

注意，图 4.15 中的时序关系是一个实际时序的高度简化版本；此处也仅介绍了 T_{AD} 和 T_{DS} 的物理意义，没有对该时序中其他参数给出描述。在实际系统中，还有许多其他的时间限制条件。通常在电路设计的时候需要仔细阅读所用微处理器芯片与存储器芯片的时序特性，充分评估该器件挂接到特定总线时是否能满足总线标准的时序要求。

2. 异步总线时序

同步总线时序中，主从模块按照统一的时钟执行相应的动作，模块间的配合简单一致。但是，若一条同步总线上连接有多个工作速度有快有慢的设备，那么总线周期必须按照最慢设备的速度来设计，导致速度快的设备不能满负荷地运行。

为了解决上述同步时序带来的效率问题，又提出了异步总线的设计。异步总线依靠传送双方互相约定的握手（handshake）信号来实现定时控制。在这种方式下，系统没有公用的时钟，收发设备之间采用一问一答的方式进行联络和协调工作。根据问答信号之间的关系，异步总线时序可分成不互锁方式、半互锁方式和全互锁方式。

图 4.16 左半部分所示为一种问答式的主设备读取从设备数据的过程：①处表示主设备（master，简称 M）已在地址线上发出拟访问数据的地址，即已准备好接收数据，发出请求信号；②处表示从设备（slave，简称 S）已将数据驱动到数据总线上，发出应答信号。图 4.16 右半部分所示为主设备向从设备写入数据的过程：①处表示主设备已在地址线上发出拟写入的单元地址，同时在数据总线上发出拟写入的数据，即已准备好发送数据，发出请求信号；②处表示从设备已从数据总线读取数据，发出应答信号。无论是主设备的请求信号，还是从设备的应答信号，均在持续固定的时间间隔后撤销。这种方式称作不互锁的异步时序。

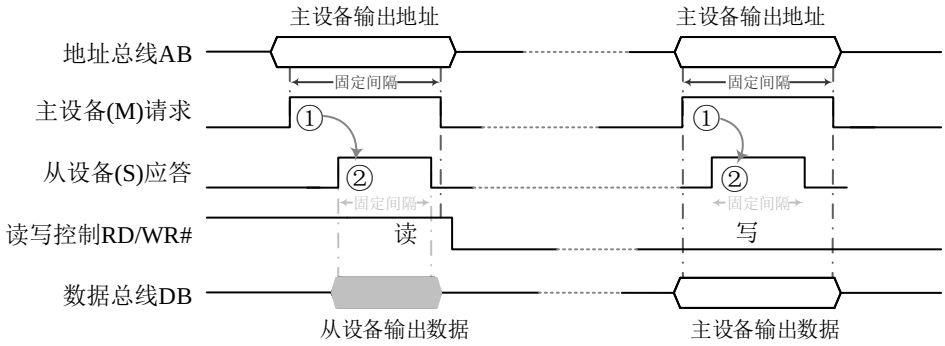


图 4.16　异步总线时序（不互锁方式）

图 4.17 所示的数据访问过程是半互锁方式异步时序。主设备读取从设备数据的基本流程为：①处主设备发请求信号并等待从设备的应答；②处从设备发出应答；③处主设备收到从设备的应答后才撤销请求信号。和图 4.16 所示不互锁方式不同的是：主设备发请求信号后，必须等待从设备的应答，只有收到应答才撤销请求；从设备发出应答后，不等待主设备响应，在间隔固定时间后，撤销应答信号。

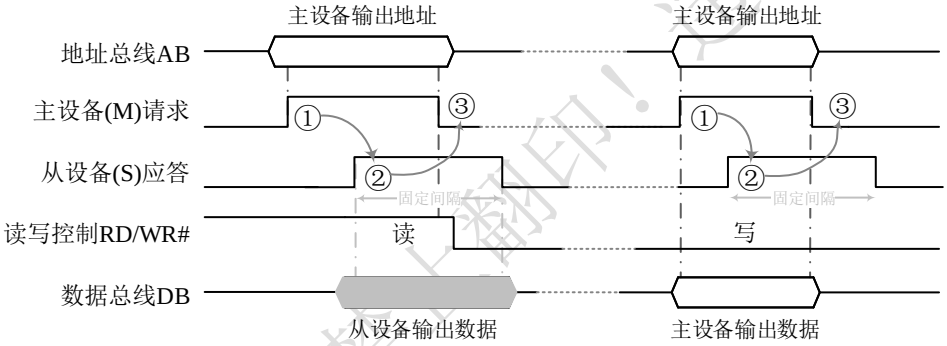


图 4.17　异步总线时序（半互锁方式）

图 4.18 所示则为全互锁方式的时序。图 4.18 左半部分是主设备读取从设备数据的过程：①处表示主设备已准备好接收数据，请求从设备将数据送出；②处表示从设备已将数据送出，请求接收；③处表示主设备已收到数据；④处表示从设备已将数据放弃，并且主设备和从设备都准备好进入下一个异步周期。与不互锁方式、半互锁方式对比，全互锁方式时序中，主设备撤销请求信号等待从设备的应答信号，从设备撤销应答信号也必须在主设备撤销请求信号后。主从设备相互等待，传输可靠性最高。图 4.18 右半部分所示为主设备向从设备写入数据的过程，请读者自行分析。

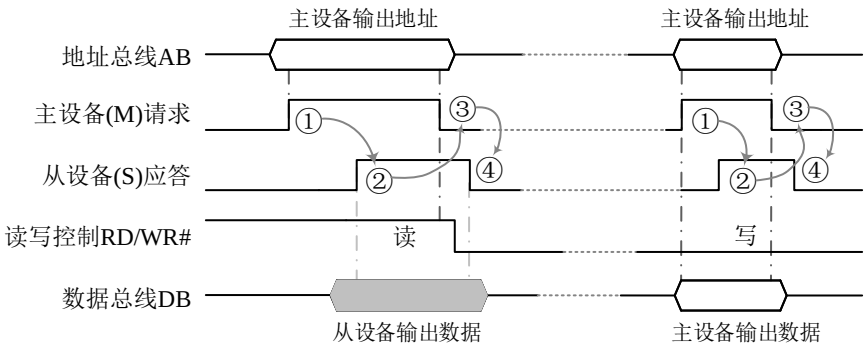


图 4.18　异步总线时序（全互锁方式）

异步总线通过预定义的问答协议可以保证两个工作速度差异很大的模块或设备之间可靠地进行信息交换；但由于握手需要额外的总线开销，因此效率比同步总线低。异步总线适用于设备类型多且距离较远的系统。很多实际系统中的 I/O 总线都采用异步总线。

3.　半同步总线时序

在上述同步时序中，如果主存储器的速度太慢，如，从地址总线读取地址到准备好数据之间的时延超过一个时钟周期，此时就不能如图 4.15 所示在 T2 时钟周期读数据。可以通过插入一个时钟周期的办法让 CPU 等待存储器准备数据。这种具有同步时钟而又能插入等待周期的时序称为半同步时序，如图 4.19 所示。

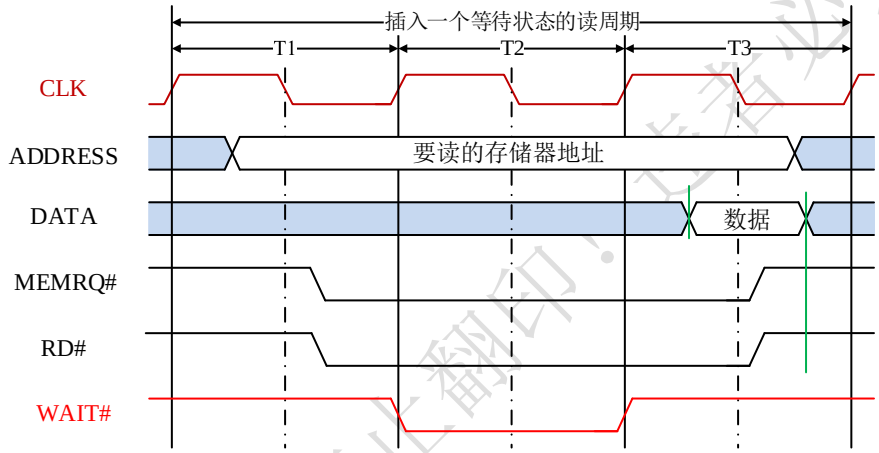


图 4.19　半同步总线时序

图 4.19 中 WAIT#信号表示需要进行等待，WAIT#信号由 CPU 发出，低电平有效。相比于图 4.15，图 4.19 所示时序 CPU 读到数据的时间滞后了一个时钟周期（WAIT#信号持续一个时钟周期）。插入等待时钟周期的数目可以根据存储器访问速度来确定，如果存储器访问速度低于两个时钟周期，那么就需要插入两个等待时钟周期（WAIT#信号持续两个时钟周期）。

4.　周期分裂式总线时序

前述三种总线时序方式，从主模块发出地址和读写命令开始，直到数据传输结束，整个传输周期中，总线一直处于被占用的状态。但实际上，从主模块发送了地址和读/写命令，到从模块提供好数据，这之间有一定的时间间隔（即从模块执行读/写命令的时间）。在这段时间内，系统总线上并没有实质性的信息传输，是空闲的。

为了充分利用这段空闲时间，可以将一个读周期分解为两个分离的子周期：寻址子周期、数据传送子周期。在寻址子周期，主模块发送地址和命令及有关信息，经总线传输，由有关从模块接收下来后，立即和总线断开。随即总线可以被其他主模块使用。待从模块准备好数据后，启动数据传输子周期，从模块申请总线，请求主模块接收数据。在一些资料中，也把上述寻址子周期称作地址阶段，把数据传送子周期称作数据阶段。

两个子周期均按同步方式传送，在占用总线时才进行高速信息传送。这样，就可以把两个独立子周期之间的空闲时间给系统中其他主模块使用，从而大大提高了总线的利用率，使系统的整体性能增强。这种分离式传输非常适合于有多个主模块的系统。

上述分裂式操作的技巧是高性能总线设计中的一种重要思路。事实上，将一个耗时长的操作分成几个耗时短的操作也是流水线设计的基本思想。一些资料中，常把周期分裂式操作称作“分离式操作”、“流水线分离”等，我们将在 4.2.2 小节中，以 AMBA 总线中 AHB 子系列为例进一步分析这种周期分裂式操作的具体实现。

4.2 片内总线 AMBA

4.2.1 AMBA 总线概述

AMBA（Advanced Microcontroller Bus Architecture）总线是 ARM 公司研发的一种片上总线标准。AMBA 的设计独立于处理器和芯片制造工艺技术，旨在为芯片内部不同电路模块之间互联提供了一种全球统一的规范。AMBA 总线规范是一个开放标准，可免费从 ARM 获得。目前，AMBA 拥有众多第三方支持，被 ARM 公司 90%以上的合作伙伴采用，在基于 ARM 处理器内核的 SoC 设计中，已获得广泛的支持。

随着 ARM 处理器的不断更新换代，AMBA 总线演进出多个版本，各个版本的功能差异情况参阅 4.2.6 小节。从学习的角度看，我们最关注的是 ARM 公司在片上总线设计方面的基本原则和思路。AMBA2 已具备相对完整的功能；AMBA3 侧重引入高性能的功能支持；AMBA4、AMBA5 等较晚推出的版本中增加的功能则主要针对多 CPU 核环境做相应的优化。故本小节中我们主要结合 AMBA2 和 AMBA3 版本学习 AMBA 总线的设计思想。

AMBA2 定义了三种不同的总线，分别是 AHB（Advanced High-performance Bus，高级高性能总线)、ASB（Advanced System Bus，高级系统总线）和 APB（Advanced Peripheral Bus，高级外设总线）。三类不同总线的特征如表 4.1 所示。与此同时，AMBA 规范还定义了一套测试方法（test methodology），用于测试和诊断。

表 4.1 AMBA2 中 AHB、ASB、APB 特性对比

AMBA AHB	AMBA ASB	AMBA APB
❒ 高性能 ❒ 流水线（pipelined）操作 ❒ 多总线主机（multiple bus masters） ❒ 突发传输（burst transfers） ❒ 分裂式操作（split transactions）	❒ 高性能 ❒ 流水线操作 ❒ 多总线主机	❒ 低功耗 ❒ 地址锁存和控制 ❒ 接口简单 ❒ 适合大多数外设

基于 AMBA2 总线的典型互联结构中，包括一个高性能系统中枢总线（AHB 或 ASB），能够支持 CPU、片上存储器、外部存储器和其他直接数据存取（DMA）设备相互之间的数据传输。这条高性能总线上也有一个桥接器以连接低带宽的 APB，而在 APB 上连接着大多数的系统外设，图 4.20 为基于 AMBA2 总线的典型微处理器系统。

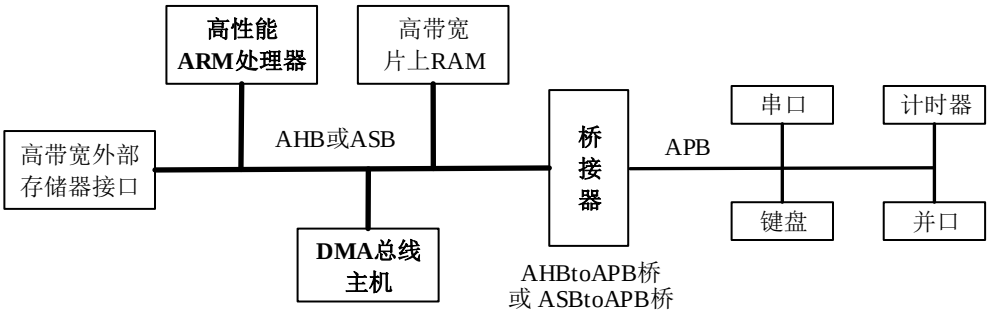


图 4.20　典型 AMBA2 总线的微控制器系统

AMBA 总线是片内总线，是一种与工艺无关的片上协议，不提供任何与电气特征有关的信息，电气特性取决于设计时选择的生产工艺。AMBA 总线在周期级别上定义了多种信号的行为，但准确的时序则取决于所使用的处理工艺和操作频率。

AMBA 中定义了信号名称的前缀规则。所有 AMBA 信号的命名都用名称的第一个字母来指示信号和哪个总线相关联，信号若为低电平有效，则信号名称的最后加小写字母 n，测试信号有前缀 T（与总线类型无关），所有信号名均为大写字母。H、A、B、D 等前缀分别代表了不同总线上的信号，如表 4.2 所示。

表 4.2　AMBA2 中定义的信号前缀

前缀	含义及示例
T	测试信号（与总线类型无关）。
H	AHB 信号，如 HREADY 是表示 AHB 的数据传输完毕的信号，高电平有效。
A	ASB 信号，主机与仲裁器之间的单向信号。
B	ASB 信号，如 BRESn 为 ASB 复位信号，低电平有效。
D	ASB 信号，单向的 ASB 译码信号。
P	APB 信号，如 PCLK 表示 APB 使用的主时钟。

4.2.2　AHB 总线

1.　AHB 系统的构成

在 AMBA2 中，AHB 担当高性能系统的中枢总线。AHB 支持处理器、片上存储器、片外存储器，以及低功耗外设单元之间的有效连接。AHB 总线是支持多主机的总线，能支持高带宽、高时钟频率的数据传输操作。一个典型的 AHB 系统包括如下部分。

① AHB 主机（Master），即总线的主控模块。总线主机能够通过提供地址和控制信息发起读写操作。任何时候只允许一个总线主机处于有效状态并能使用总线。

② AHB 从机（Slave），即总线的从属模块。总线从机在给定的地址空间范围内响应主机发起的读写操作。总线从机将成功、失败或者等待数据传输的信号返回给有效的主机。

③ AHB 仲裁器（Arbiter）。总线仲裁器确保每次只有一个总线主机被允许发起数据传输。AHB 要求即便在单总线主机系统中也要实现仲裁器，且所有系统中只有一个仲裁器。

④ AHB 译码器（Decoder）。AHB 译码器用来对每次传输进行地址译码，从而为从机提供

选择信号。AHB 要求实现一个中央译码器（centralized decoder）。

AHB 总线有如下特点：

- 支持突发传输（burst transfer），即启动一次 AHB 传输可以连续传输一个数据块。
- 支持分裂式操作（split transactions），及寻址和数据传输子周期分离。
- 单周期总线主机移交（single cycle bus master handover），主机在时钟上升沿发出总线使用请求，仲裁器在时钟下降沿采样获得该请求，随即在下一个时钟上升沿更新总线允许信号。即，仲裁器在每个时钟周期都更新总线允许信号。
- 单一时钟沿操作（single clock edge operation），所有信号变化均发生在时钟上升沿，这利于芯片内集成更多的多电路单元；
- 非三态的实现（non-tristate implementation），AMBA 允许信号线以三态或非三态的形式实现，如，数据线为三态实现时可为双向传输线，非三态实现的时候读数据和写数据是两套独立的传输线。
- 可扩展至更宽的数据总线架构（64 位或 128 位）等。

AHB 总线通过一组多路选择器实现主、从设备的互连，如图 4.21 所示，数据传输操作所需要的地址和控制信号由总线主机产生。地址和控制选择器负责将主机发出的地址和控制信号连接至从机，同时数据选择器负责主机和从机之间的数据信号连接。仲裁器在不同的总线主机之间进行总线使用权仲裁，并决定当前得到授权的主机能将它的地址和控制信号连通到哪个从机。译码器控制读数据选择器，读数据选择器负责将从机信号连接到对应的主机上。

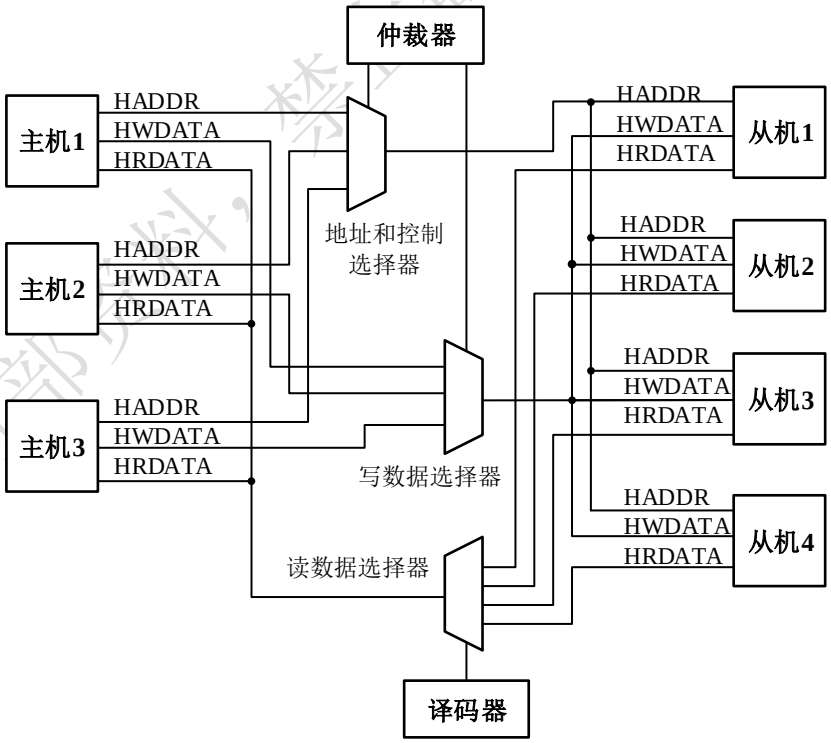


图 4.21　基于多路选择器的 AHB 总线互联

2. AHB 信号定义

AHB 信号定义如表 4.3 所示。

表 4.3 AMBA2 中 AHB 的信号

信号名称	信号来源	用途说明
HCLK 总线时钟	时钟源	为所有总线传输提供时基，所有信号时序都与 HCLK 的上升沿或下降沿相关。
HRESTn 复位	复位控制器	总线复位信号，用于复位系统和总线，低电平有效。
HADDR[31:0] 地址总线	主机	32 位系统地址总线。
HTRANS[1:0] 传输类型	主机	表示当前传输的类型，可以是不连续（NONSEQUENTIAL）、连续（SEQUENTIAL）、空闲（IDLE）和忙（BUSY）。
HWRITE 传输方向	主机	高电平时表示一个写传输，低电平时表示读传输。
HSIZE[2:0] 传输大小	主机	表示传输的大小，可以是字节（8 位）、半字（16 位）或字（32 位）等，协议允许的最大传输大小是 1024 位。
HBURST[2:0] 突发类型	主机	表示传输是否组成了突发的一部分。支持突发长度为 4、8 或 16 的突发传输。
HPROT[3:0] 保护控制	主机	总线访问的附加信息，指示当前传输的安全保护级别。
HWDATA[31:0] 写数据总线	主机	用来在写操作期间，从主机到从机传输数据。建议最小数据宽度为 32 位，在高带宽运行时可扩展。
HSELx 从机选择	译码器	每个 AHB 从机都有自己独立的从机选择信号，并且用该信号来表示当前传输是否打算送给选中的从机。该信号是地址总线的组合译码。
HRDATA[31:0] 读数据总线	从机	用来在读操作期间，由从机向主机传输数据。建议最小数据宽度为32位，在高带宽运行时可扩展。协议允许的数据宽度为：8、16、32、64、128、256、512、1024位。
HREADY 传输完成	从机	当该信号为高电平时，表示总线上的传输已经完成。在扩展传输时该信号可能会被拉低。
HRESP[1:0] 传输响应	从机	给传输状态提供了附加信息，提供四种不同的响应：OKAY、ERROR、RETRY 和 SPLIT。

为了持多主机操作，AMBA AHB 中定义了许多仲裁信号，如表 4.4 所示。其中有些仲裁信号用于点对点连接，信号后缀 x 表示模块 x。例如，HBUSREQx 可能分别表示 HBUSREQarm 或 HBUSREQdma。

表 4.4 AMBA2 中 AHB 的仲裁信号

信号名称	信号来源	用途说明
HBUSREQx 总线请求	主机	从总线主机 x 传向总线仲裁器，表示该主机请求使用总线的信号。每个总线主机都有一个 HBUSREQx 信号，最多允许 16 个总线主机。
HLOCKx 锁定的传输	主机	当该信号为高时，表示主机请求锁定对总线的访问，并且在该信号为低之前，其他主机不应该被允许授予总线。
HGRANTx	仲裁器	该信号用来表示总线主机 x 目前是优先级最高的主机。当

总线授予		HREADY 为高电平时，传输结束，地址控制信号的所有权发生改变。所以主机应在 HREADY 和 HGRANTx 都为高电平时，获得对总线的访问。
HMASTER[3: 0] 主机号	仲裁器	表示哪个总线主机正在执行传输，被支持突发传输的从机用来确定哪个主机正在尝试一次访问。
HMASTLOCK 锁定顺序	仲裁器	表示当前主机正在执行一个锁定序列（sequence）的传输。该信号与 HMASTER 信号有相同时序。
HSPLITx[15:0] 分离式传输请求	从机（支持分裂式操作）	指示仲裁器，总线主机被允许重试一个分裂式操作（split transaction）。每位对应一个总线主机。

3. AHB 的数据传输过程及“流水线”

在一次 AHB 传输开始之前，总线主机必须先获得总线访问的授权。这个授权过程由总线主机向仲裁器发出一个请求信号而发起，随后仲裁器指示主机何时被授权。获得授权的主机通过提供关于地址信号，传输方向、传输宽度和传输类型等控制信号发起一次 AHB 传输。

1） 单个数据简单传输

一个 AHB 传输分成两个阶段：地址阶段（address phase）、数据阶段（data phase）。地址阶段仅持续一个时钟周期，用来传输地址和控制信息；而数据阶段可以持续一个或者多个时钟周期，用来传输有效数据。如图 4.22 所示为地址阶段和数据阶段都持续一个时钟周期的简单 AHB 传输。注意，图 4.22 中并未画出 HWRITE 信号，即图中不对主机写传输和主机读传输做区分。换言之，如果是主机写传输应关注 HADDR[31:0]和 HWDATA[31:0]，反之在主机读传输时应关注 HADDR[31:0]和 HRDATA[31:0]。

图 4.22 中的第一个时钟周期的上升沿，主机把地址、控制等信息驱动到总线上（HADDR[31:0]和其他控制信号线）。第二个时钟周期的上升沿，从机采样获得这些地址（图示为 A）和控制信息，并据此（地址 A）把相应的数据驱动到数据总线 HRDATA[31:0]上。第三个时钟周期的上升沿，主机在 HRDATA[31:0]上采样即可获得从机响应的数据。

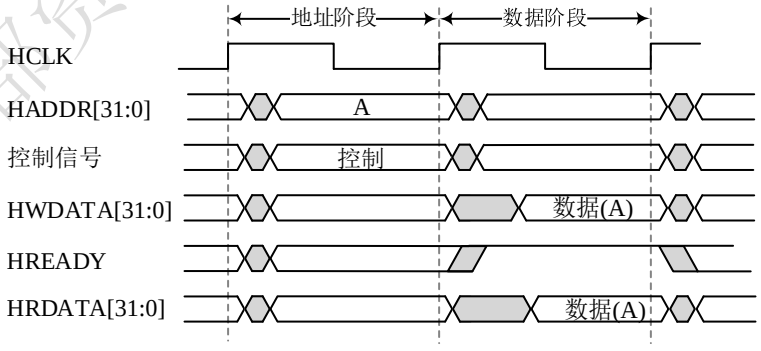


图 4.22 简单传输（无等待状态）时的 AHB 传输时序图

传输过程中，①主机在地址阶段把地址信息 A 驱动到了地址总线上，②而从机时下一个时钟周期（图 4.22 中数据阶段的第 1 个时钟周期）时钟的上升沿通过采样获取这些地址和控制信息，③随后的下一个时钟周期（图 4.22 中数据阶段结束的下一个时钟周期）时钟的上升沿通过采样数据总线 HWDATA[31:0]获取数据。整个过程中对地址信息的更新和对该

地址数据的更新在节拍上是错开的，在当前 AHB 传输地址阶段的地址信息实际上是上一次 AHB 传输最后一个时钟周期就已经驱动到 HADDR[31:0]上了，而本此 AHB 传输的数据更新至 HRDATA[31:0]之后，只能在下一次的 AHB 传输开始第一个时钟周期才能被读取。这种地址信息和数据信息交叠（overlapping）的操作方式，被称作流水线（pipeline）机制。流水线机制有利于提高性能，在其他类型的总线如用于外设的 APB 中，为了进一步简化电路则不提供这种流水线机制。

2） 单个数据简单传输中插入等待状态

如果数据阶段持续一个时钟周期不足以完成数据的传输，从机可以通过 HREADY 信号扩展数据周期（即插入等待周期）。

图 4.23 是数据阶段被扩展（即插入了等待周期）的 AHB 传输时序。主机在 HCLK 上升沿之后将地址和控制信号驱动到总线上；在时钟的下一个上升沿，从机采样地址和控制信号；随后进入数据阶段从机开始驱动适当的响应。若从机在数据阶段的第一个时钟周期没能准备好，则需要把 HREADY 拉为低电平，从而插入了等待周期（图 4.23 中从机插入了两个时钟周期的等待）。从机准备好后，再将 HREADY 重新置为高电平。总线主机在随后的下一个时钟（第五个时钟）上升沿采样 HRDATA[31:0]获得从机响应的数据。需要说明的是：对写操作而言，总线主机必须保持数据在整个扩展期中稳定；而在读传输中，从机没必要提供有效数据直至传输结束，只需要在相应周期提供数据即可。

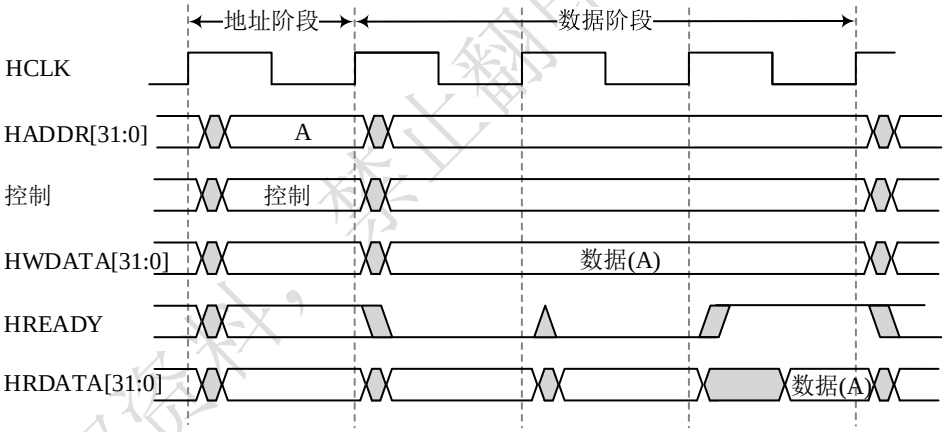


图 4.23 有等待状态的 AHB 传输时序图

需要注意的是，在流水线机制的作用下，如果数据阶段被扩展（即插入了等待周期），相应地，下一个 AHB 传输的地址阶段也需要被扩展，如图 4.23 中 HADDR[31:0]所示。

3） 多个数据的传输

图 4.24 所示为另外一个示例，三次 AHB 传输分别需要传输的是地址 A、地址 B 和地址 C 对应的数据。

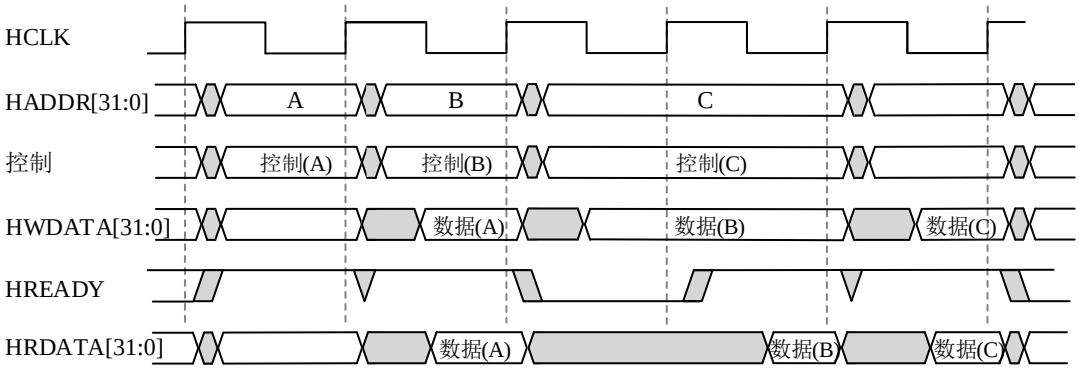


图 4.24　多个 AHB 传输的时序

图 4.24 中，传输地址 A 和地址 C 数据都没有插入等待周期，但是传输地址 B 数据的时候插入了一个等待周期。地址 B 对应数据在总线上停留的时间被扩展为 2 个周期，对应地，关于地址 C 对应数据传输的时候，在地址总线 HADDR[31:0]上地址 C 也被扩展了，其停留时间也是 2 个周期。

4. AHB 的突发传输

图 4.22、图 4.23、图 4.24 所示的三个例子基本说明了主机获取总线使用权后进行数据传输的基本过程。为了支持高效率的数据传输，在 AMBA2 的 AHB 协议中，还定义了突发传输。简单说，突发传输就是一次传输过程传输一个数据块而不是单个数据。既然是传输一个数据块，就涉及到数据块的长度，数据块中数据地址的递增方式等。表 4.5 中定义了突发式传输不同方式下的数据块长度和数据地址的递增方式。

表 4.5　AMBA2 中突发传输类型信号 HBURST[2:0]含义

HBURST[2:0]	类型	类型的描述
000	SINGLE	单次传输 Single transfer
001	INCR	未标识长度的地址递增式传输 Incrementing burst of unspecified length
010	WRAP4	突发长度为 4 的地址循环递增式传输 4-beat wrapping burst
011	INCR4	突发长度为 4 的地址顺序递增式传输 4-beat incrementing burst
100	WRAP8	突发长度为 8 的地址循环递增式传输 8-beat wrapping burst
101	INCR8	突发长度为 8 的地址顺序递增式传输 8-beat incrementing burst
110	WRAP16	突发长度为 16 的地址循环递增式传输 16-beat wrapping burst
111	INCR16	突发长度为 16 的地址顺序递增式传输 16-beat incrementing burst

为了能够向从机指示突发传输过程中的不同状态，定义了传输状态指示信号（HTRANS[1:0]，如表 4.6），从机可以从 HTRANS[1:0]指示信号中获知自身下一步的操作

提示。除此之外，协议中还定义了控制信息的细节（如传输方向、传输块大小、保护方式等）；定义了从机相应信息格式（HREADY 及 HRESP[1:0]参数组合形式）。

表 4.6 AMBA2 中传输状态指示信号 HTRANS[1:0]含义

HTRANS[1:0]	状态	状态的描述
00	IDLE	指示当前周期没有数据需要传输，当主机被授权使用总线，但是不需要传输时使用该状态。
01	BUSY	BUSY 传输类型指示主机正在进行突发传输，但是下一次数据传输不会马上发生。地址和控制信号线上的信号对应下一次即将发生的传输。
10	NONSEQ	NONSEQUENTIAL，指示当前传输是一次突发传输过程或单次传输的第一次传输。地址和控制信号线上的信号与前一次传输无关。
11	SEQ	SEQUENTIAL，一次突发传输过程中非第一次的传输。地址和信号线上的信号对应上一个刚完成的传输。

下面通过两个突发传输的例子说明一下表 4.5 定义的突发传输类型和表 4.6 定义的传输状态指示信号的作用。

图 4.25 所示为突发类型 WRAP4 的突发传输过程。整个传输过程共进行了 4 次数据传输，插入了 1 个等待周期（T2）。这四次传输中，第 1 个拟传输数据对应的地址为 0x38，该地址在 T1 周期被推送到地址总线 HADDR[31:0]上，T1 周期传输状态指示信号 HTRANS[1:0]为 NONSEQ，指示拟传输的数据是突发传输中的第 1 次。T2 周期，第 2 个拟传输数据的地址 0x3C 被推送到地址总线上，同时传输状态指示为 SEQ。T3 周期，第 1 个拟传输数据被推送到数据总线 HWDATA[31:0]上。T4 周期，第 3 个拟传输数据的地址 0x30 被推送到地址总线上，同时，第 2 个拟传输数据被推送到数据总线上。T5 周期，第 4 个拟传输数据的地址 0x34 被推送到地址总线上，同时，第 3 个拟传输数据被推送到数据总线上。T6 周期，第 4 个拟传输数据被推送到数据总线上。

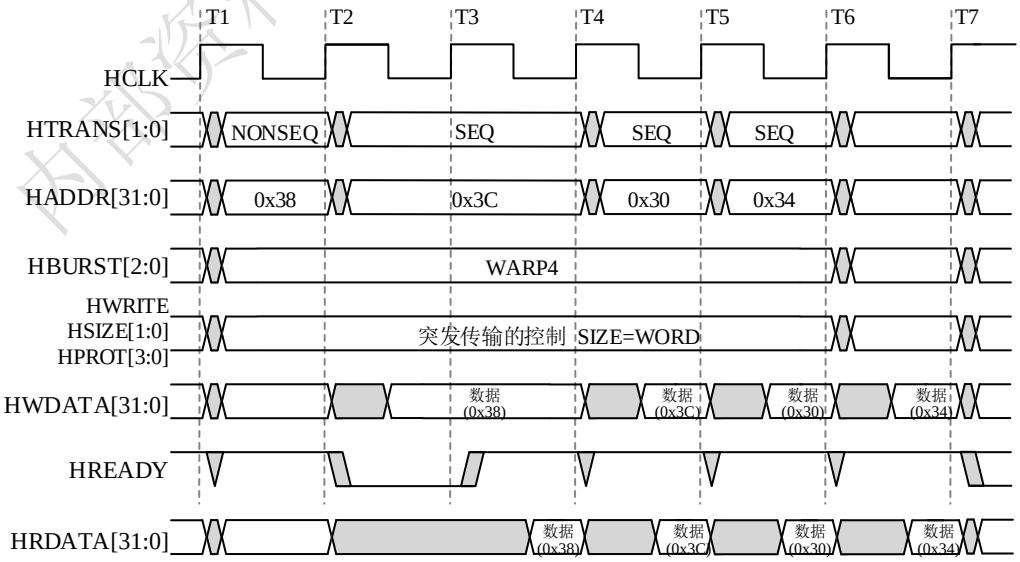


图 4.25 WRAP4（Four-beat wrapping burst）突发传输时序

由于突发类型是 WRAP4，故而 4 个传输数据的地址变化为：0x38、0x3C、0x30、0x34。拟传输数据的地址在 16 字节的边界处发生回转（即地址总线的低四位置零，通常我们把这种地址在边界处置零的操作称为循环式地址递增，对应数字逻辑电路中四位计数器的递增操作）。与图 4.25 的 WRAP4 类型不同，在图 4.26 所示的突发传输过程中，突发传输的类型为 INCR4，4 个传输数据的地址变化是递增的，没有在边界处发生回转。

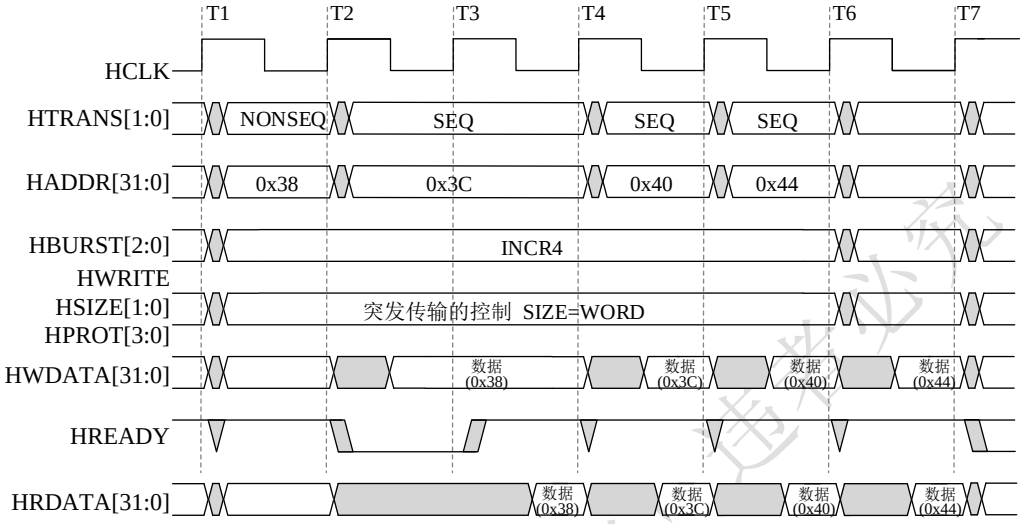


图 4.26 INCR4（Four-beat incrementing burst）**突发传输时序**

需要注意的是，在 AMBA 规范中定义了，突发传输过程中拟传输的数据块不能跨越 1kB 的边界。

5. AHB 的译码

如图 4.21 所示，AHB 总线通过一组多路选择器实现主、从设备的互连。

从机选择信号 HSELx：AHB 总线上的所有从机使用一个中央地址译码器提供从机选择信号 HSELx。选择信号是高位地址信号的组合译码结果。从机 x 在 HSELx 和 HREADY 有效的情况下，对地址和控制信号线进行采样，从而获得地址和相关的控制信息。

能够分配给单个从机的最小地址空间是 1KB。所有总线主机不允许执行超过 1KB 的地址边界的增量传输，因此确保了一个突发传输不会超过地址译码的边界。从机选择信号如图 4.27 所示。

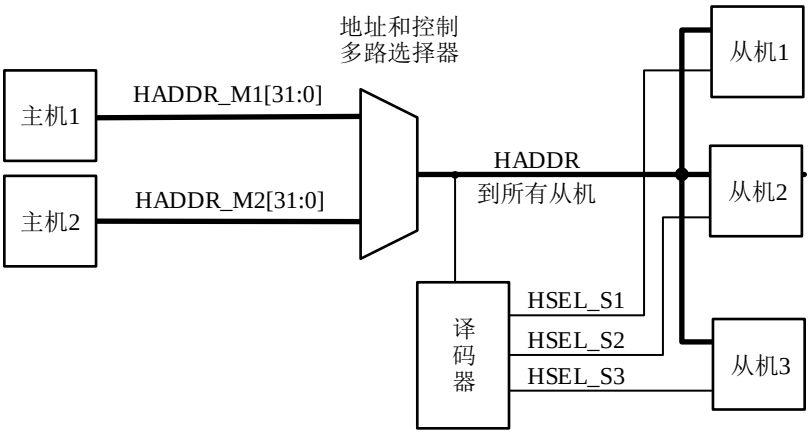


图 4.27　地址译码和从机选择信号

主机允许信号 HGRANTx：由于采用了一个集中式的译码器，每个主机可以在需要的时候随即驱动自身的地址信号，而无须等待总线允许信号 HGRANTx。对于主机而言，HGRANTx 信号是自身获得总线控制权的指示，同时，通过 HGRANTx 可知自己所驱动的地址信号是否已经被从机采样。如图 4.28 所示，集中式的译码器根据各个主机的总线使用请求产生总线允许信号 HGRANTx，并在仲裁器的控制下生成主机号 HMASTER[3:0]，HMASTER[3:0]指示地址和控制多路选择器把主机的地址总线与对应从机的地址总线连接。

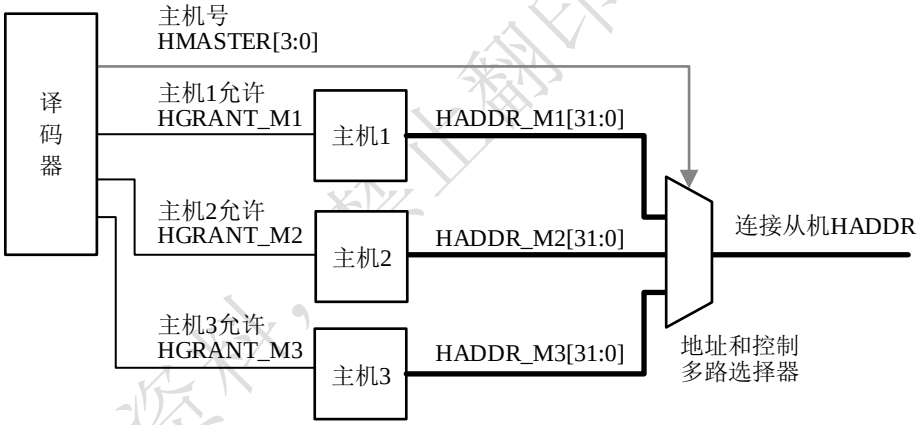


图 4.28　总线允许信号与主机号的生成

6. AHB 的仲裁

如前述表 4.4 所示，AMBA2 的 AHB 中定义了一些专门的信号用于仲裁。信号后缀 x 表示模块 x，如 HBUSREQx 可能代表了不同主机的总线使用请求，可能分别表示了 HBUSREQarm 或 HBUSREQdma 等。系统中每个总线主机都有一个 HBUSREQx 信号，最多支持 16 个总线主机。AHB 中仲裁器的接口信号如图 4.29 所示。

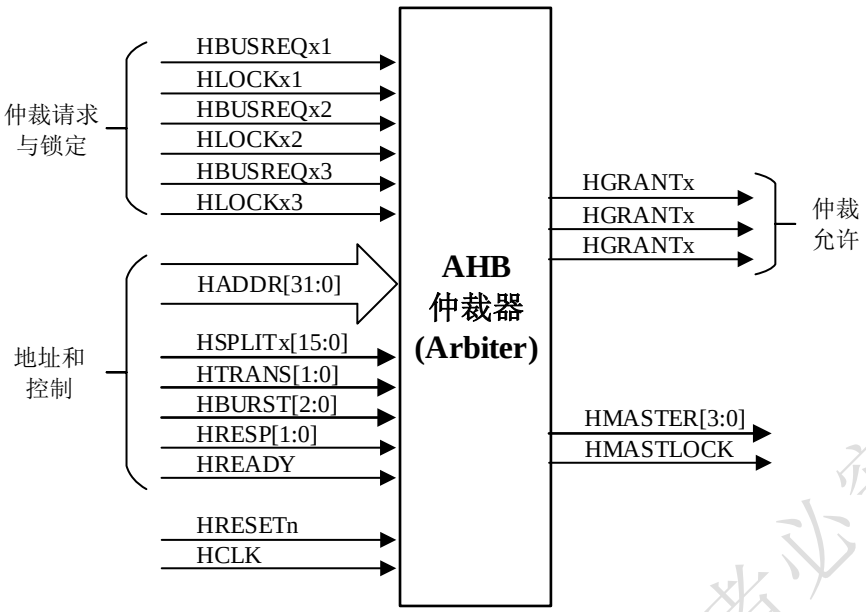


图 4.29　AMBA2 中 AHB 的仲裁器接口示意图

仲裁机制是为了保证在同一时刻，只有一个主机（master）能控制总线。仲裁器通过检测请求的优先级等信息确定哪个主机能够获取总线控制权。同时，仲裁器也响应从机关于分离式（SPLIT）传输的请求。

1） 仲裁授予过程

仲裁授予过程的基本步骤包括：①主机通过 HBUSREQx 信号请求对总线的使用需求，如果主机 x 希望使用总线的时候能够锁定总线资源，则同时需要发出 HLOCKx 信号。②仲裁器通过 HGRANTx 信号指示主机 x 获得了总线的使用权。③如果当前 HREADY 有效，则不需要等待，仲裁器改变 HMASTER[3:0]以提示当前获得总线使用权的主机号。

图 4.30 所示即为 HREADY 有效的前提下的仲裁授予过程。在 T3 周期的时钟上升沿，HGRANTx 有效；在下一个周期即 T4 的时钟上升沿，仲裁器开始驱动 HMASTER[3:0]指示了当前获得总线使用权的主机号，同时，获取了总线控制权的主机将地址（A）驱动到地址总线 HADDR[31:0]上；再后续的周期即 T5 的时钟上升沿地址（A)对应的数据被驱动到数据总线 HWDATA[31:0]。

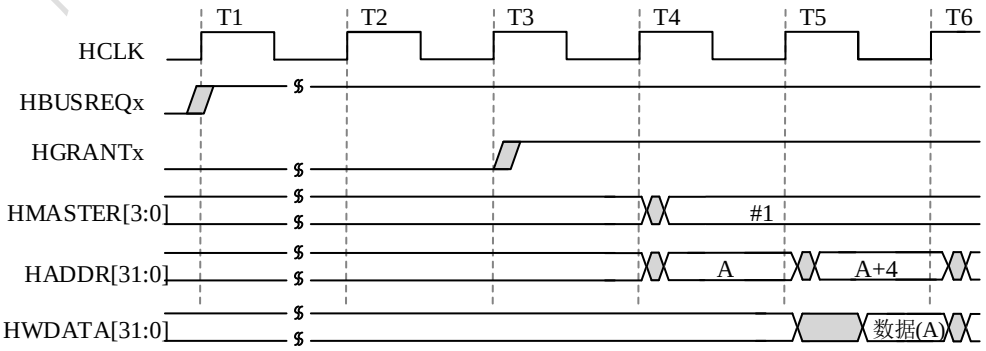


图 4.30　无等待状态的仲裁授予

如果 HGRANTx 有效时，HREADY 为无效状态（低电平），则需要插入等待周期，如图 4.31 所示，T4 周期时钟上升沿由于 HREADY 无效，插入了一个等待周期；T5 周期时钟上升沿时候，可检测到 HREADY 有效，仲裁器开始驱动 HMASTER[3:0]指示了当前获得总线使用权的主机号，同时，获取了总线控制权的主机将地址(A)驱动到地址总线 HADDR[31:0]上。T 周期时钟上升沿的时候，HREADY 无效，故获得总线使用权的主机又等待了一个周期，在 T7 周期时钟上升沿的时候，才驱动数据到 HWDATA[31:0]上。

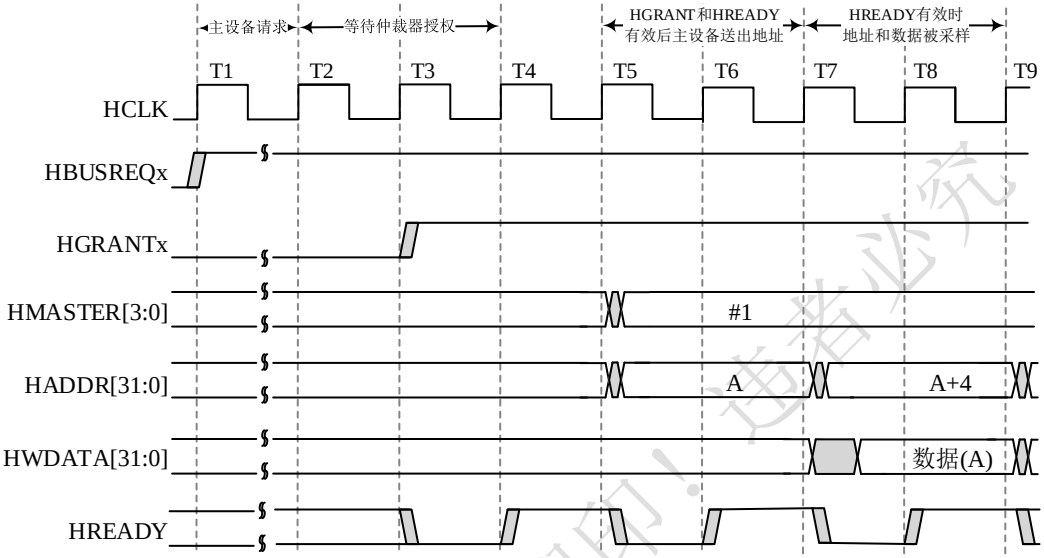


图 4.31　有等待状态的仲裁授予

2）　总线控制权的移交

总线的控制权包括地址总线控制权和数据总线控制权，数据总线的控制权滞后于地址总线。当一次传输完成后（通过 HREADY 信号高电平予以指示），随后拥有地址总线控制权的主机接管数据总线。图 4.32 所示为总线控制权在两个主机间的移交（Handover）过程，从图中可看出主机 1（Master 1）对数据总线的控制权是滞后于对地址总线控制权一个时钟周期的。

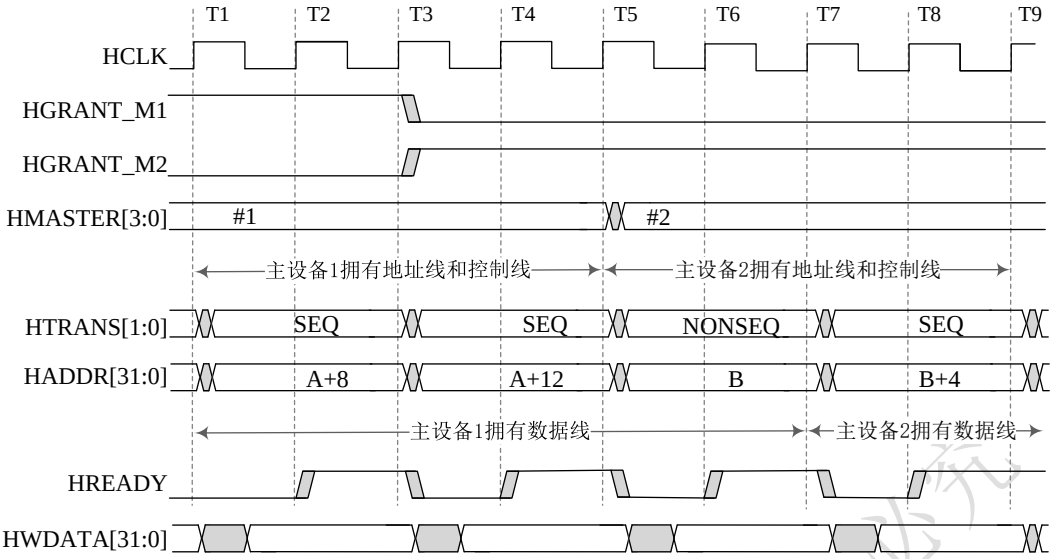


图 4.32　总线控制权在两个主机间的移交

图 4.33 所示为总线控制权移交发生在一次突发传输的末尾。仲裁器在倒数第二个地址被采样后改变总线允许信号 HGRANTx。新的总线允许信号 HGRANTx 将与最后一个地址在相同的时刻被采样。

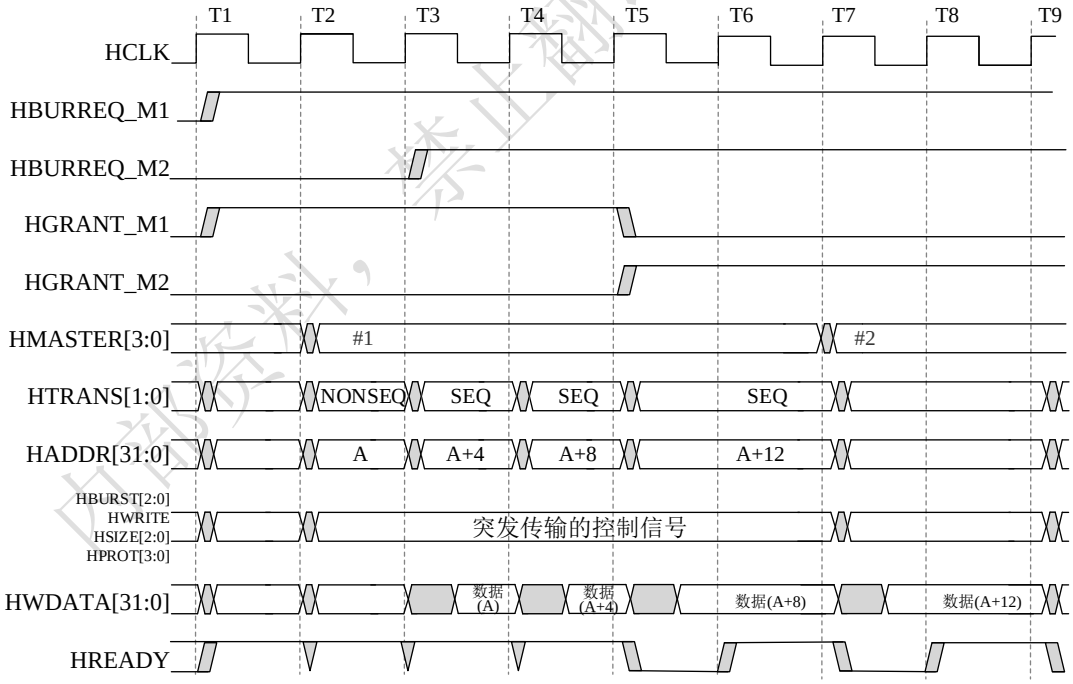


图 4.33　突发传输后的总线移交

7. AHB 主机接口

图 4.34 所示为 AHB 总线主机接口框图。图 4.35 所示为 AHB 总线从机接口框图。

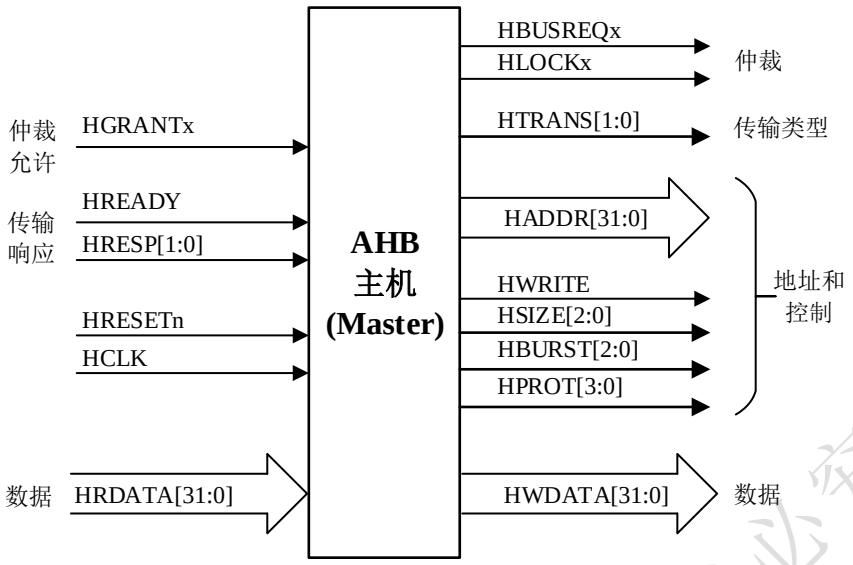


图 4.34　AHB 主机接口框图

复位信号 HRESETn 是 AHB 中定义的唯一一个低电平有效的信号。该信号将引起所有总线上电路单元进行复位操作。复位信号 HRESETn 的产生（asserted）可以与时钟信号 HCLK 异步，但是复位信号的撤除（deasserted）是与 HCLK 同步的，需要与 HCLK 上升沿对齐。复位过程中，总线上所有的主机要确保地址总线和控制总线处于有效的状态且 HTRANS[1:0]指示处于 IDLE 状态（HTRANS[1:0]定义参见表 4.6））。

8. AHB 从机接口及流水线“分离”

在图 4.35 所示 AHB 总线从机接口框图中，HMASTER[3:0]、HMASTLOCK、HSPLIT[15:0]用于支持名为“SPLIT”的传输模式（在 ARM 公司的资料中，被称作分裂式操作，Split transactions；其他一些资料中也称作流水线分离）。

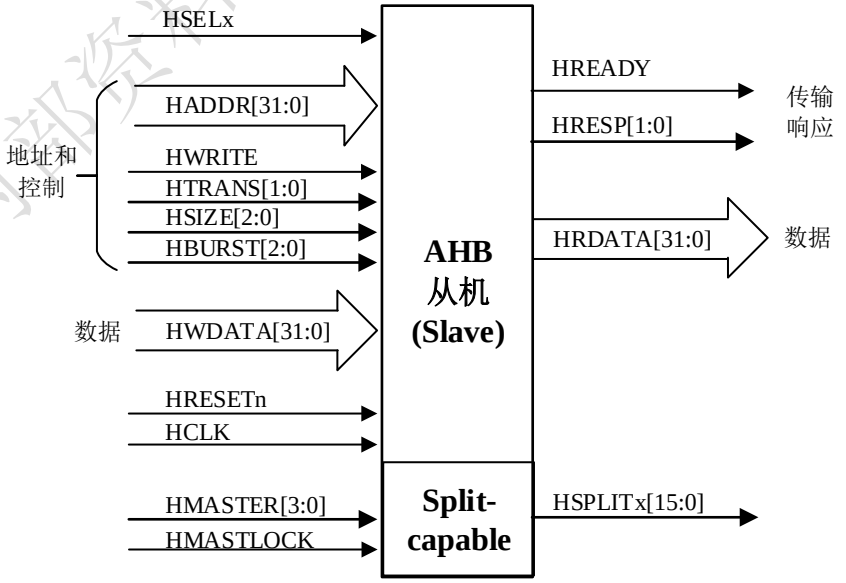


图 4.35　AHB 从机接口框图

SPLIT 传输，指前述 AHB 传输的两个阶段（地址阶段和数据阶段可以被分离，参见 4.1.3 4 周期分裂式时序）。根据前面介绍的“流水线”机制可知，前一次 AHB 传输的数据实际上在下一次 AHB 传输的一开始才被读取的，第 *n* 次 AHB 传输的地址则是在第 *n-1* 次 AHB 传输的时候就已经被驱动到了地址总线上。这样“驱动地址”和“驱动数据”两个操作构成了两级流水线操作。

这种两级的流水线操作要求从机能够及时地响应主机，在主机驱动地址到地址总线 HADDR[31:0]后，能够及时被从机读取；主机驱动数据到数据总线 HWDATA[31:0]上后，也要能够及时被从机读取。如果从机因为某种原因不能及时响应，这个流水线就会被打断，影响到总体性能。SPLIT 方式的数据传输就是为了应对从机不能及时响应的情形。

从机通过驱动 HSPLITx[15:0]发出启动 SPLIT 传输的信号，当从机发出请求后，并不需要记录当前主机已经驱动到地址总线上的地址以及控制信息。仲裁器检测到 HSPLITx 后，知道从机当前不进行传输，则可以把总线的使用权出让给其他主机。当从机做好接收数据准备后，通过 HSPLITx[15:0]发出重新启动传输的信号，仲裁器根据挂起操作主机的优先级决定何时再次分配总线使用权。当主机获得总线使用权后，重新发送地址、控制等信息，继续刚才挂起的传输操作。

在传输的地址阶段，仲裁器产生一个标识号（主机号）HMASTER[3:0]，用来指示哪个主机正在使用总线。任何一个从机如果要发出 SPLIT 信号，需要依据 HMASTER[3:0]信号决定 HSPLITx[15:0]中哪个位需要置 1。HSPLITx 有 16 根信号线，意味着最多支持 16 个不同主机来源的传输采用 SPLIT 分离机制，不同从机产生的 HSPLITx 位以“或”的方式合成在一起，供仲裁器进行仲裁判断。并且，AHB 中仅允许一个主机存在一个 SPLIT 传输，即 SPLIT 传输是不可以“嵌套”的。

4.2.3 ASB 总线

ASB 是另外一种高性能的总线，支持处理器片上存储器、片外存储器，以及低功耗外设宏功能单元之间的有效连接。如图 4.20 所示，ASB 可以作为微控制器芯片中的主总线。其系统结构与 AHB 类似，包括如下几部分：

- ASB 主机（Master），即总线的主控模块。总线主机能够通过提供地址和控制信息发起读写操作，任何时候只允许一个总线主机处于有效状态并使用总线。
- ASB 从机（Slave），即总线的从属模块。总线从机在给定的地址空间范围内响应总线读写操作。总线从机将成功、失败或者等待数据传输的信号返回给有效的主机。
- ASB 译码器。总线译码器将传输地址译码并且选择合适的从机。总线译码器也确保总线在没有总线传输时也保持在工作状态。ASB 要求仅有一个中央译码器。
- ASB 仲裁器。总线仲裁器确保每次只有一个总线主机被允许发起数据传输。即使仲裁协议已经固定，任何一种仲裁算法都能够根据应用要求而得到执行。ASB 必须只包含一个仲裁器。

图 4.36、图 4.37 所示分别为 AMBA2 中 ASB 的主机、从机的接口框图。与图 4.34、图 4.35 对比可知 ASB 信号的定义与 AHB 信号的定义是类似的，但信号名称前缀不同（前缀含义表参见表 4.2）。

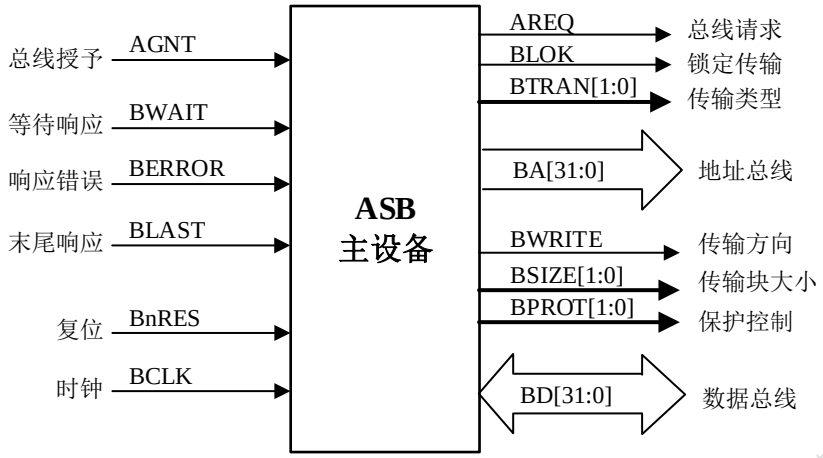


图 4.36 ASB 主机接口框图

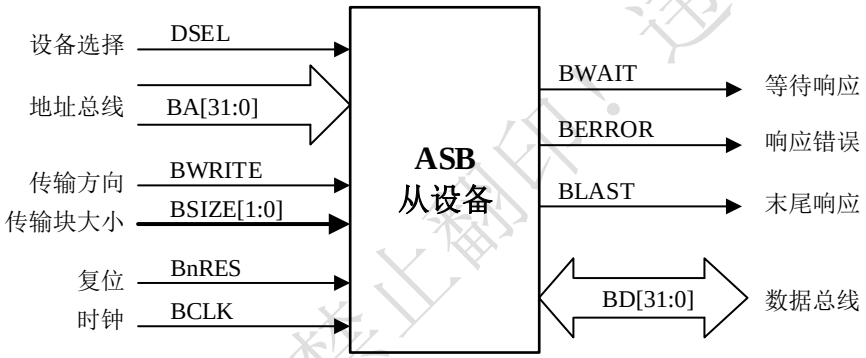


图 4.37 ASB 从机接口框图

表 4.7 所示为 AMBA2 中定义的 ASB 信号列表。

表 4.7 AMBA2 中 ASB 信号列表

信号名称	信号用途描述
AGNTx（总线授予）	从仲裁器到主机 x，用来指示总线主机是否被授权，系统中每个总线主机都有一个 AGNTx 信号。
AREQx（总线请求）	从主机 x 到仲裁器，表示总线主机请求使用总线，系统中每个总线主机都有一个 AREQx 信号。
BA[31:0]（地址总线）	系统地址总线，由有效的总线主机驱动。
BCLK（总线时钟）	时钟信号，为所有总线传输提供时基。
BD[31:0]（数据总线）	系统数据总线，在写传输时，由当前的总线主机驱动，在读传输时，由被选中的总线从机驱动。
BERROR（响应错误）	高电平时，表示被选中的总线从机发生一个传输错误。
BLAST	由被选中的从机驱动，指示当前传输是否是一次突发的最

信号名称	信号用途描述
（末尾指示）	后一次传输。
BLOK （锁定传输）	高电平时，表示当前传输和下一个传输是独占的，其他总线主机不应该被授权访问总线。
BRESn （复位）	低电平时，复位系统和总线。
BPORT[1:0] （保护信息）	总线访问的附加信息，指示当前传输的安全保护级别。
BSIZE[1:0] （传输大小）	指示传输的大小，可以是字节，半字或字。
BTRAN[1:0] （传输类型）	这些信号表示下一次传输的类型，可以是仅地址、不连续（NONSEQUENTIAL）或连续（SEQUENTIAL）。
BWAIT （等待指示）	由被选中的从机驱动，低电平表示当前传输完成，高电平时表示需要插入等待周期。
BWRITE （传输方向）	高电平时表示写传输，低电平时表示读传输。
DSELx （从机选择）	从总线译码器到从机 x，指示该从机被选中，每个 ASB 总线从机都有一个 DSELx 信号。

4.2.4 APB 总线

APB 设计目标是为低传输速率的外设提供一个低成本的接口。APB 为了降低功率消耗和接口电路复杂度，不支持 AHB 中的流水线功能。不同版本 APB 演进如表 4.8 所示。从 APB 设计思想的角度看，在 AMBA2 版本中定义的 APB2 已经具备的最核心的设计思路，故本小节的学习以 AMBA2 版本为主。版本间的差异信息会进行补充说明。

表 4.8 APB 的版本演进（截止 2019 年）

简称	全称	最后版本	更新时间
APB	APB Specification Rev E	AMBA1	1998
APB2	AMBA 2 APB Specification	AMBA2	1999
APB v1.0	AMBA 3 APB Protocol Specification v1.0	AMBA3	2004
APB v2.0	AMBA APB Protocol Specification v2.0	AMBA4	2010

APB 表现为一个局部二级总线，可封装为 AHB 或 ASB 的一个外设。APB 在 AHB 和 ASB 信号的基础上直接为系统总线提供了低功耗的延伸。图 4.20 所示为 AMBA2 版本中 APB 与 AHB 对接的示意图。APB 的最新版本定义在 AMBA4 版本中，可以和更多类型的总线对接。

表 4.9 APB 可对接的总线（截止 2019 年）

可对接的总线名称	最后更新
AMBA Advanced High-performance Bus (AHB)	AMBA2
AMBA Advanced High-performance Bus Lite (AHB-Lite)	AMBA2
AMBA Advanced Extensible Interface (AXI)	AMBA4
AMBA Advanced Extensible Interface Lite (AXI4-Lite)	AMBA4

APB 应该作为任何低带宽和不需要通道总线接口的高性能外设接口。APB 规定所有信号的传输只和时钟的上升沿相关，因此 APB 具有以下优点：

- ❒ 易于实现较高频率的操作；
- ❒ 性能和时钟占空比无关；
- ❒ 通过使用单时钟沿来简化静态时序分析等。

APB 桥（APB bridge）作为一个从模块，处理总线握手并且从局部外设总线的角度控制信号的时序。APB 桥用来将 AHB 或 ASB 传输转变成适合于从设备的形式。APB 桥提供所有地址、数据和控制信号的锁存，也提供一个二级译码，以产生 APB 外设的从机选择信号。所有 APB 模块均是 APB 从机。

表 4.10 所示为 APB 信号列表。此处不仅列出了 AMBA2 版本中的信号定义，也列出 APB 最新版本（APB v2.0）在 AMBA4 新增的信号定义。

表 4.10 APB 信号列表

名称	发起源	用途说明	备注
PCLK （总线时钟）	时钟源	上升沿作为所有 APB 传输的时基	AMBA2
PRESENTn （APB 复位）	系统总线	低电平时，复位总线和连接到总线上的设备。	AMBA2
PADDR[31:0] （APB 地址总线）	APB 桥	地址信号，由外设总桥接单元驱动。	AMBA2
PSELx （APB 选择）	APB 桥	表示从机被选中，每个从机都有一个 PSELY 信号。	AMBA2
PENABLE （APB 选通）	APB 桥	指示 APB 传输的第二个周期。	AMBA2
PWRITE （APB 传输方向）	APB 桥	高/低电平分别表示 APB 写/读。	AMBA2
PRDATA （APB 读数据总线）	从机	数据信号，由被选中的从机驱动。	AMBA2
PWDATA （APB 写数据总线）	APB 桥	数据信号，由外设总线桥接单元驱动。	AMBA2
PPROT （APB 传输的保护类型）	APB 桥	总线访问的附加安全信息。	AMBA4
PSTRB （APB 写选通）	从机	指示要进行数据更新。	AMBA4
PREADY （APB 准备好）	从机	指示从机拟拓展至下一个周期的传输。	AMBA4
PSLVERR （APB 传输错误）	从机	指示发生了传输错误。	AMBA4

图 4.38、图 4.39 所示分别为 APB 的写传输总线时序和读传输总线时序。写传输时，PWRITE 为高电平，数据传输总线为 PWDATA；读传输时，PWRITE 为低电平，数据传输总线为 PRDATA。

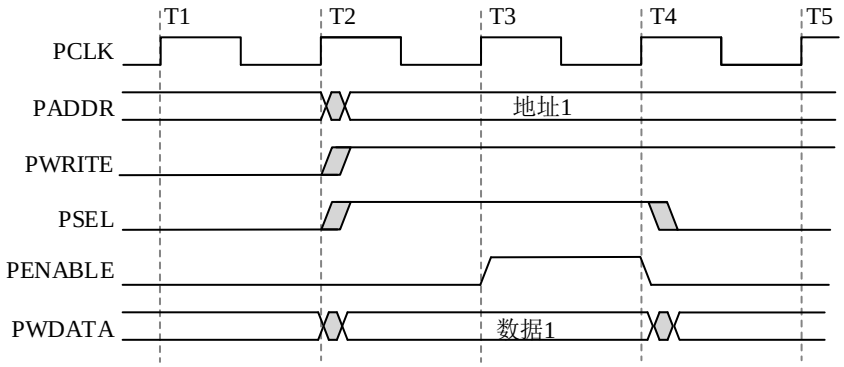


图 4.38　APB 总总线的写传输总线时序

APB 的时序定义较为简单。可以用三个操作状态来描述总线上的不同阶段：空闲周期（IDLE，可持续多个周期）、设置周期（SETUP，持续单个周期）、使能周期（ENABLE，持续单个周期）。

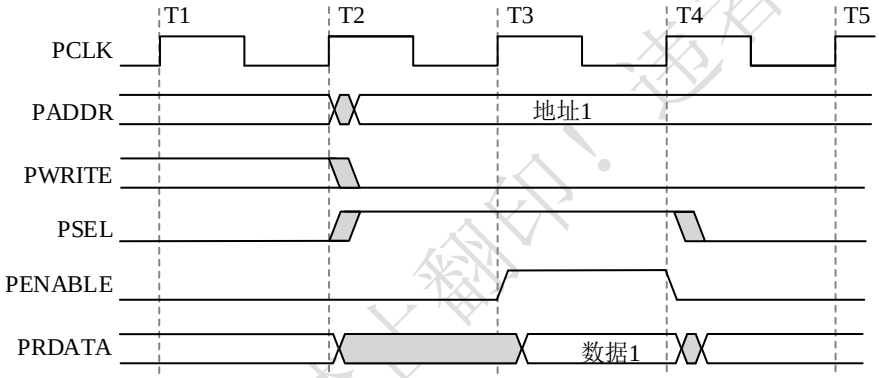


图 4.39　APB 总线的读传输总线时序

在时钟上升沿到来后，地址信号 PADDR、选通信号 PENABLE、选择信号 PSEL 和读写数据方向 PWRITE 信号等全部改变，读写传输的开始。传输的第一个时钟周期称为 SETUP 周期，在下一个时钟上升沿使能信号 PENABLE 生效，表明进入 ENABLE 周期。地址、数据和控制信号全都在整个 ENABLE 周期保持有效，传输在这个周期结束时完成。使能信号 PENABLE 在传输结束时失效；选择信号也将变成低电平，除非当前传输之后紧跟着另一个到该外设的传输。为了降低功耗，地址信号和写信号将在传输之后不再改变，直到下一个传输发生为止，即进入了空闲阶段（IDLE）。

APB 桥是 AMBA APB 中的唯一总线主机，在 APB 桥所连接的 ASB 总线中，又表现为异构从机。图 4.40 为 APB 桥的接口信号。APB 桥接单元将 ASB 传输转换成 APB 传输，并实现下列功能：

- ❒ 锁存地址并使之在整个传输期间有效。
- ❒ 译码地址并产生外设选择信号 PSELx，在一个传输周期只会有一个选择信号有效；
- ❒ 对于写传输，驱动数据到 APB 上；
- ❒ 对于读传输，驱动 APB 数据到系统总线上；
- ❒ 为传输产生一个选通信号 PENABLE。

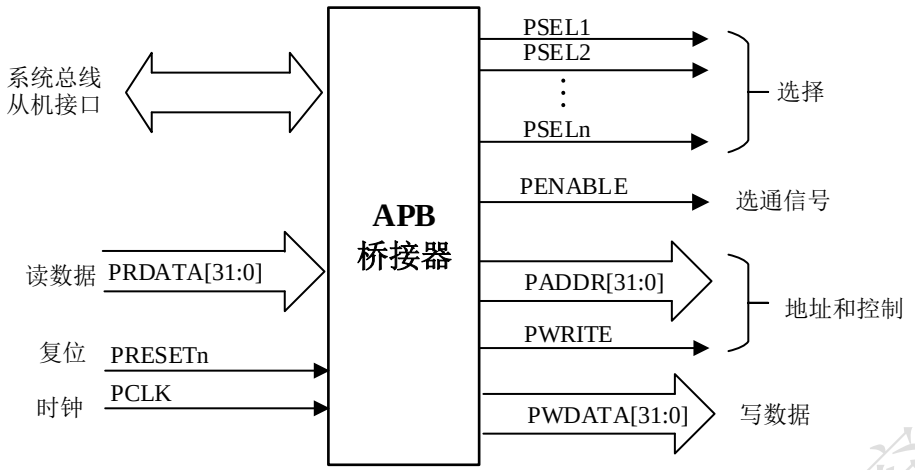


图 4.40　APB 桥接口框图

图 4.41 为 APB 外设的信号接口。对于写传输，下列两种情况数据被锁存：①在任意一个 PCLK 的上升沿，当 PSEL 为高电平时；②在 PENABLE 的上升沿，当 PSEL 为高电平时。通过选择信号 PSELx、地址信号 PADDR 和写信号 PWRITE 来决定哪个寄存器由写操作来更新。而对于读传输数据，数据在 PWRITE 为低电平且 PSELx 和 PENABLE 都为高电平时被驱动到数据总线上，此时 PADDR 用来决定该读取哪个寄存器。

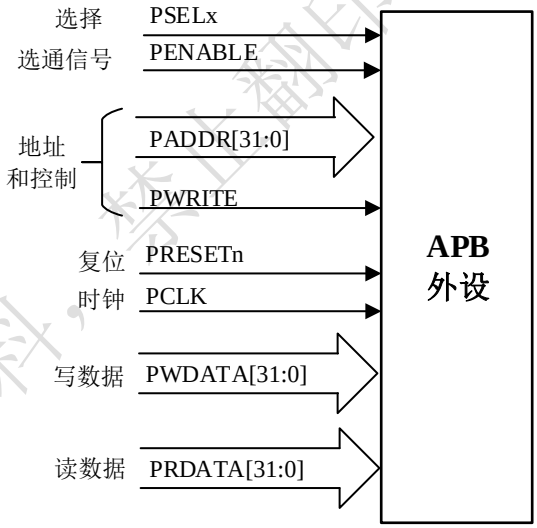


图 4.41　APB 外设（从机）接口框图

4.2.5　AXI 总线

2001 年，ARM 发布了 AMBA 3 总线规范，引入了 AXI（Advanced eXtensible Interface）总线。AXI 总线的引入，使 AMBA 总线迈向新的台阶。AHB 总线分离了一个总线周期的地址阶段和数据阶段，便于实现在现代总线中常用的流水线（pipelining）和分裂（split）式操作。AXI 总线则进一步分离了总线的通道，将 AHB 的单通道分解为五个独立的通道：读地址通道（read address）、读数据通道（read data）、写地址（write address）通道、写数据（write data）通道、写响应（write response）通道，进一步提升了对存储器的读写访问效率。表 4.11 显示了 AXI 与 AHB 及 APB 特性的对比。

表 4.11　AXI、AHB、APB 对比

总线	AXI	AHB	APB
总线宽度	8、16、32、64、128、256、512、1024	32、64、128、256	8、16、32
地址宽度	32	32	32
通道特性	✓读写地址通道 ✓读写数据通道均独立	✓读写地址通道共用 ✓读写数据通道独立	✓读写地址通道共用 ✓读写数据通道独立 ✓不支持读写并行操作
体系结构	✓多主/从设备 ✓仲裁机制	✓多主/从设备 ✓仲裁机制	✓单主设备（桥）/多从设备 ✓无仲裁
数据协议	✓支持流水线 ✓支持分裂式操作(split) ✓支持突发传输(burst) ✓支持乱序访问 ✓字节/半字/字 ✓大小端对齐 ✓非对齐操作	✓支持流水线 ✓支持分裂式操作(split) ✓支持突发传输(burst) ✓支持乱序访问 ✓字节/半字/字 ✓大小端对齐 ✓不支持非对齐操作	✓一次读/写传输占两个时钟周期 ✓不支持突发传输
传输方式	支持读写并行操作	不支持读写并行操作	不支持读写并行操作
时序	同步	同步	同步
互联	多路	多路	无定义

随后，在 2010 年发布的 AMBA 4 总线规范中，进一步增强了多层结构，将 AXI 总线细分为 AXI4、AXI4-Lite 和 AXI4-Stream（AXI4-stream 是 ARM 公司和 Xilinx 公司一起提出，主要用在 FPGA 进行以数据为主导的大量数据的传输应用）。目前 AXI 总线已取得了广泛的应用和工业界的支持。Xilinx（赛灵思）公司同 ARM 公司合作，共同为基于 FPGA 的高性能系统和设计定义了 AXI4 规范。赛灵思公司的 Xilinx Vivado Design Suite 软件和 ISE Design Suite 软件凭借 AXI4 标准进一步扩展了赛灵思平台的设计方法。作为 AXI4 推广工作的一部分，赛灵思采用 AXI4 作为 UltraScale、Zynq-7000、Spartan-6、Virtex-6 及未来产品系列的互连标准。

4.2.6　AMBA 的不同版本

AMBA 总线 V1.0 于 1995 年正式发布，用于 SoC 内部各个模块间的互联，支持多个主设备，支持芯片级别测试。在 AMBA V1.0 中定义了 ASB 和 APB 两条总线。也定义了一个连接存储器的外部接口，这个外部接口可以用做测试。ASB 总线支持多个主设备与从设备，主要作用是连接 CPU，DMA 引擎，内部存储器和一些快速外部设备。APB 总线连接一些慢速设备，APB 是 ASB 的局部二级总线。

如今来看，AMBA V1.0 总线十分简陋。1999 年 AMBA 总线更新到 V2.0，增加了一个新的总线 AHB。AHB 总线逐渐取代了 ASB 在系统中的位置，AHB 分离了总线操作的地址阶段和数据阶段，有利于实现 Pipelining 和 Split 技术，提高了存储器读的效率，并且将总线宽度拓展至 128 位。

2001 年，ARM 发布了 AMBA V3.0 总线规范，引入 ATB（Advacned Trace Bus）和 AXI 总线。AXI 总线的引入，使 AMBA 总线迈向新的台阶。AXI 在 AHB 基础上进一步分离了

总线通路，将 AHB 的单通路分解五个独立通路，进一步加速了对存储器的读写访问。

AMBA 总线阵营规范了嵌入式领域的平台总线，日益壮大。2010 年 3 月 8 日，ARM 正式推出 AMBA V4.0 总线，引入了 QoS 机制，进一步增强了多层结构，将 AXI 总线细分为 AXI4、AXI-Lite 和 AXI-Stream。在 AMBA4 总线中最值得注意的是 CoreLink CCI（Cache Cohenrent Interconnect）架构。CoreLink CCI 架构有利于多个 SMP（Symmetric Multi-Processing）系统之间实现缓存一致性。Cortex A15 也借此超越了用于嵌入式领域的 PowerPC 和其他多核 MIPS。处理器总线技术中，类似的互联技术还有 IBM 的 CoreConnect 和 Intel 的 QPI。

2013 年，为了适应高性能异构计算环境， AMBA5 版本中引入了 CHI（Coherent Hub Interconnect）。CHI 可视为 AXI 的重新设计版本。基于 AXI 协议的信号被替换为基于数据包的层次化的协议。这种全新的设计为集成不同类型的计算单元如 GPU、DSP、FPGA、内存控制器和不同的 I/O 子系统提供了便利。

目前，ARM® AMBA® 协议已成为连接和管理 SoC 中功能模块的开放标准和片上互连规范。截至目前（2019 年），AMBA 定义了 CHI™、ACE™、AXI™、AHB™、APB™ 和 ATB™ 等一系列规范，为 SoC 模块定义了共同的框架结构，有助于设计的重复使用。不同 AMBA 版本定义的主要区别如图 4.42 所示。

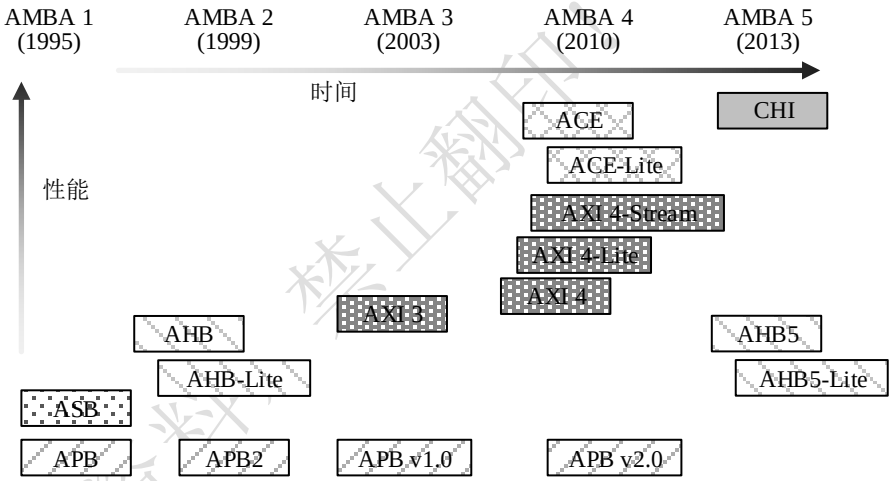


图 4.42 AMBA 版本演进图

这些不同系列的规范具有不同的特点，适用于不同的场景。例如，AMBA2 中 AHB/ASB/APB 被广泛用于微控制器芯片；AMBA3 中 AXI、ACE 被广泛应用于智能手机的微处理器芯片中；CHI 则用在服务器与网络应用程序所需要的高可扩展性的片上系统。这些不同子系列的全称及定义时间如表 4.12 所示。

表 4.12 AMBA 不同版本定义的子系列汇总

缩写	英文全称	用途简述	首次引入	最后更新
APB	Advanced Peripheral Bus	连接低速率的外设	AMBA 1	AMBA 4
ASB	Advanced System Bus	高级系统总线	AMBA 1	AMBA 2
AHB	Advanced High-performance Bus	连接高传输速率的功能单元	AMBA 2	AMBA 2

AHB-lite		仅支持单主机的简化版 AHB，主要用于 Cortex-M	AMBA 3	AMBA 3
AXI	Advanced Extensible interface	在复杂的片上系统中至多连接 100 个主、从模块	AMBA 3	AMBA 4
ATB	Advanced Trace Bus	用于在芯片间传输 trace 数据	AMBA 3	AMBA 4
AXI-lite		不支持突发传输的简化 AXI	AMBA 4	AMBA 4
AXI-stream		支持主机到从机的流式数据传输	AMBA 4	AMBA 4
ACE	AXI extension	多核 CPU 情形支持缓存一致性	AMBA 4	AMBA 4
ACE-Lite		支持无缓存代理的简化 ACE	AMBA 4	AMBA 4
CHI	Coherent Hub Interface	为支持数目众多的异构处理器核心而改造的 ACE	AMBA 5	AMBA 5

4.3 系统总线/外部总线

在计算机技术发展的过程中，出现过很多总线标准。例如，扩展总线 ISA、EISA、AGP、VESA、PCI、PCIe 等；用于外部存储设备的总线 ATA、IDE、SCSI、PCMCIA、SATA 等；外部总线 LPT、USB、IrDA、PS/2、RS232C、IEEE1394 等。部分典型总线名称如表 4.13 所示。其中大部分已经不再使用了，有些（USB、PCIe）则自诞生来不断演进。本小节仅简单介绍一下 PCI、PCIe 和 USB。

PCI 总线是一种外部设备互连总线，早期在微机系统中广泛使用，目前在工业控制领域仍被广泛支持。PCI Express 是 PCI 升级后的新一代，目前不仅用于微型计算机系统内不同部件与 CPU 的连接，也被用于开发新的外部总线。异步串行总线标准中的 USB 则是目前最为普遍的外部总线。

表 4.13 常见总线的简称与全名

总线类别	名称	英文全称
扩展总线	ISA	Industry Standard Architecture
扩展总线	PCI	Personal Component Interconnect
扩展总线	EISA	Extended ISA
扩展总线	SIMM	Single Inline Memory Module
扩展总线	DIMM	Dual Inline Memory Module
扩展总线	MCA	Micro-Channel Architecture
扩展总线	AGP	Accelerated Graphics Port
扩展总线	VESA	Video Electronics Standards Association
存储总线	ATA	Advanced Technology Attachment
存储总线	IDE	Integrated Drive Electronics (same as ATA)
存储总线	SCSI	Small Computer Systems Interface
存储总线	ESDI	Enhanced Small Device Interface (mid-80s, obsolete)
存储总线	PCMCIA	Personal Computer Memory Card International Association
存储总线	SATA	Serial Advanced Technology Attachment
外部总线	Parallel	sometimes called LPT (“line printer”)
外部总线	Serial	typically RS232C (sometimes RS422)
外部总线	PS/2	Personal System 2
外部总线	USB	Universal Serial Bus

外部总线	IrDA	Infrared Device Attachment
外部总线	FireWire	new, very high speed, developed by IEEE
外部总线	IEEE-488(GPIB)	General Purpose Interface Bus

4.3.1 PCI

PCI（Peripheral Component Interconnect，外部设备互连）总线，是一种高性能的局部总线。PCI 的相关工作始于 1900 年，最早由 Intel 的 IAL 实验室（Intel's Architecture Development Lab）提出，并率先在 IBM 的兼容 PC 上得到应用，后来又得到 Compaq、HP 和 DEC 等多家计算机公司的响应，并成立了 PCI-SIG（PCI Special Interest Group，PCI 特别兴趣工作组）。

采用 PCI 总线是为了解决微机总线的低速度和微处理器的高速度而造成的数据传输瓶颈问题，同时，PCI 的引入也用于高速外设的 I/O 接口和主机相连。最初的 PCI 标准中总线频率为 33MHz，数据线宽度为 32 位，可扩充到 64 位，故数据传输率可达 132MB/s～264MB/s。后期的 PCI 版本 64 位并行数据传送，总线时钟支持 33/66MHz，能够达到最高 528MB/s 的数据传输速率。

PCI 总线在高速处理器与其他低速设备之间架起了一座“桥梁”。使得计算机结构升级为基于 PCI 总线的三级总线结构，如图 4.43 所示。PCI 总线是一种树型结构，并且独立于 CPU 总线，可以和 CPU 总线并行操作。PCI 总线上可以挂接 PCI 设备和 PCI 桥片，PCI 总线上只允许有一个 PCI 主设备，其他的均为 PCI 从设备，而且读写操作只能在主从设备之间进行，从设备之间的数据交换需要通过主设备中转。

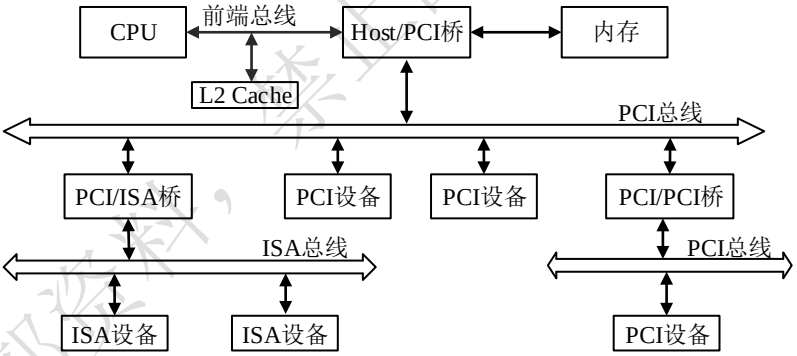


图 4.43 PCI 总线结构图

4.3.2 PCI Express

PCI Express（Peripheral Component Interconnect Express）是新一代的总线标准，简称 PCI-E 或 PCIe。早在 2001 年的春季，英特尔公司就提出了要用新一代的技术取代 PCI 总线和多种芯片的内部连接，并称之为第三代 I/O 总线技术。随后在 2001 年底，英特尔、AMD、DELL、IBM 在内的 20 多家业界主导公司开始起草新技术的规范，并在 2002 年完成，对其正式命名为 PCI Express。标准由 PCI-SIG 负责维护和升级。

相比于 PCI，PCIe 最大的改变是由并行改为串行，通过使用差分信号传输（differential transmission）。采用了目前业内流行的点对点串行连接，比起 PCI 以及更早期的计算机总线

的共享并行架构，每个设备都有自己的专用连接，不需要向整个总线请求带宽，而且可以把数据传输率提高到一个很高的频率，达到 PCI 所不能提供的高带宽。

PCIe 总线是一种完全不同于过去 PCI 总线的一种全新总线规范，与 PCI 总线共享并行架构相比，PCIe 使用了一种点对点串行连接的设备连接方式。点对点意味着每一个 PCIe 设备都拥有自己独立的数据连接，各个设备之间并发的数据传输互不影响。而传统 PCI 共享总线方式下，总线上只能有一个设备进行通信，一旦总线上挂接的设备增多，每个设备的实际传输速率就会下降，性能得不到保证。PCIe 以点对点的方式处理通信，每个设备在要求传输数据的时候各自建立自己的传输通道，对于其他设备这个通道是封闭的，从而保证了通道的专有性，避免其他设备的干扰。图 4.44 显示了这种点对点串行连接拓扑结构。

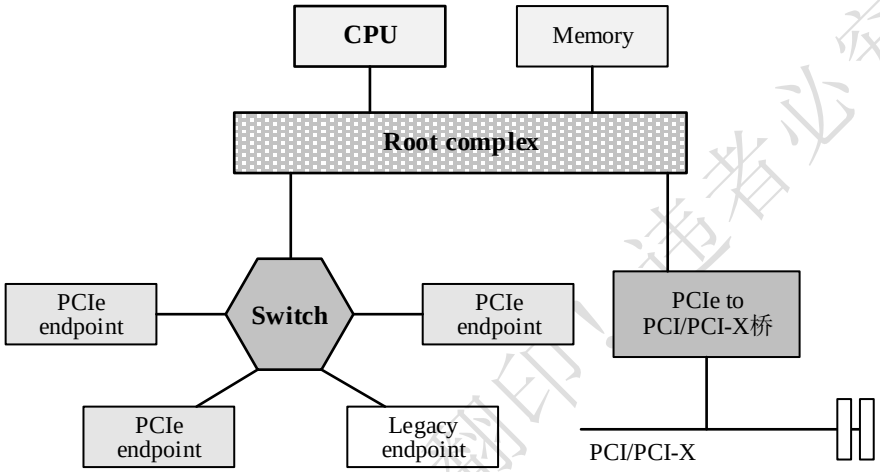


图 4.44 PCIe 架构示意图

图 4.44 所示 PCIe 系统中包括 Root complex、Switch、PCIe brige、Endpoint 四大类设备。①Root complex（根复合体）是处理器与 PCIe 总线的接口，通常由 CPU 集成。②Switch（交换器），允许多个设备连接到一个 PCIe 端口，用于扩展 PCIe 总线，具有路由功能。③PCIe 桥，负责 PCIe 和其他总线转换连接，如连接 PCI、PCI-X 甚至是另外一条 PCIe 总线。④Endpoint（端点设备），即 PCIe 接口的各种设备，分为标准 PCIe 端点和传统端点（Legacy Endpoint）。

两个设备之间的一条 PCIe 链路（link）可以包含 1 至 32 个通道（lane）。习惯上用 X1、X4、X8、X16、X32 等方式表示链路所包含的通道数目，也称为 PCIe 的“宽度”或“位宽”。单个通道包含两对差分传输信号线，如图 4.45 所示，其中一对差分信号线用于接收数据，另外一对用于发送数据。故每个通道共四根信号线（wire）。从逻辑概念上看，每个通道就是一条双向的位流传输通路。

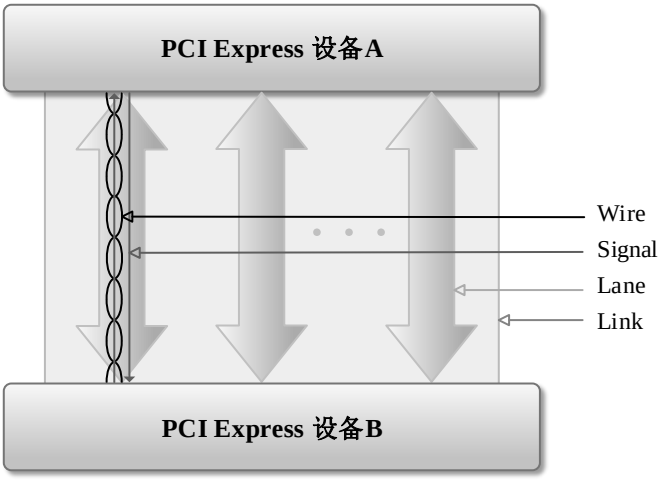


图 4.45 PCIe 链路及其包含的通道

由于链路的传输速率等于通道数目乘以单个通道的传输速率，PCI-E 可通过增加通道数目来提高整个链路的传输速率。一个单通道的 PCI-E 扩展卡（X1）的传输速度为 250MB/s，而 X16 就是等于 16 倍于 X1 的速度，即是 4GB/s。并且，单通道的 PCI-E 扩展卡（X1）可以插入多通道的插槽（如 X16、X8 等）。

PCIe 规格从 1 条通道连接到 32 条通道连接，有非常强的伸缩性，以满足不同系统设备对数据传输带宽不同的需求。例如，PCI Express X1 规格支持双向数据传输，每向数据传输带宽 250MB/s，PCI Express X1 已经可以满足主流声效芯片、网卡芯片和存储设备对数据传输带宽的需求，但是远远无法满足图形芯片对数据传输带宽的需求。因此，如果要取代传统用于显卡连接的 AGP 总线，至少需要采用 PCI Express X16，即 16 条点对点数据传输通道。PCI Express X16 也支持双向数据传输，每向数据传输带宽高达 4GB/s，双向数据传输带宽有 8GB/s 之多。

近年来，PCI-E 的应用范围越来越广，标准及相应技术也在不断发展和完善。截止到 2019 年 5 月，PCI-Express 5.0 规范已正式发布。表 4.14 显示了不同版本的发布时间及基本参数。

表 4.14 PCI-E 不同版本的传输速率

PCI-E 版本	发布时间	编码	传输速率	吞吐量				
				×1	×2	×4	×8	×16
1.0	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 GB/s	15.8 GB/s
4.0	2017	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.5 GB/s
5.0	2019	128b/130b	32.0 GT/s	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.0 GB/s

备注：表中“8b/10b 编码”意为将 8 比特需要传输的比特编码为 10 比特。表中“GT/s”表示“gigatransfers per second”，故 2.5 GT/s 对应每秒钟传输 2500M 比特，若采用 8b/10b 编码，实际每秒能传输的业务比特是 2500M*8/10=2000M (bits)=250M (Bytes)。

4.3.3 USB

USB（Universal Serial Bus，通用串行总线）是一种外部总线标准。用于规范电脑与外部设备的连接和通讯，是应用在 PC 领域的接口技术。USB 接口支持设备的即插即用和热插拔功能。USB 是在 1994 年底由由 Compaq、 Digital、BM、 Intel， Microsoft、NEC 和 Nothern Telecom 等七家公司联合提出的。1995 年 11 月，USB 0.9 规范正式提出，1998 年发布 USB 1.1，2004 年 4 月推出 USB 2.0，2008 年 11 月推出 USB 3.0。USB 采用差分方式传输，如图 4.46 所示，在 USB1.0/USB2.0 中，“D+”和“D-”组成一对差分信号线用于数据传输，VBUS 和 GND 对应 5V 电源和地。在 USB3.0 版本后，又增加了两对差分信号线以提供更高速的数据传输。

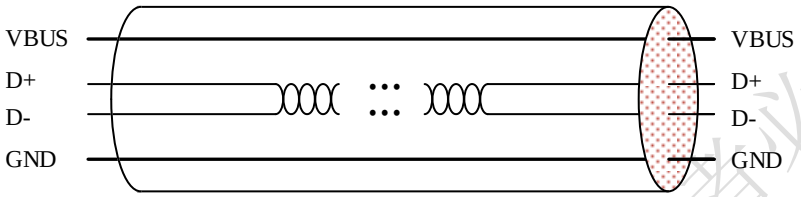


图 4.46 USB1.0/2.0 连接线缆的信号线定义

2013 年 7 月份，USB 3.1 发布，速度翻番至 10Gbps，但这个时候 USB-IF (USB Implementers Forum)把 USB 3.0 改名为 USB 3.1 Gen 1，新的 USB 3.1 则叫做 USB 3.1 Gen 2。2017 年 9 月份，USB 3.2 发布，虽然版本号依然变化不大，但速度再次翻番为 20Gbps。表 3.8 为 USB1.0、USB1.1、USB2.0、USB3.0、USB3.1、USB3.2 的主要特性比较。

根据最新公布的规范，USB 3.0、USB 3.1 的版本命名都将彻底消失，统一被划入 USB 3.2 的序列，三者分别再次改名叫做 USB 3.2 Gen 1、USB 3.2 Gen 2、USB 3.2 Gen 2x2。三者还各自有一个市场推广命名，分别是 SuperSpeed USB、SuperSpeed USB 10Gbps、SuperSpeed USB 20Gbps。2019 年 3 月 USB-IF 宣布开始 USB4 的计划，USB4 将在兼容 USB3.2 的基础上增加对 Thunderbolt 的支持，USB4 开发的时间表截止 2019 年 6 月尚未公布。

表 4.15 USB 的不同版本

USB 版本	理论最大传输速率	速率称号	最大输出电流	推出时间
USB1.0	1.5Mbps(192KB/s)	低速(Low-Speed)	5V/500mA	1996 年 1 月
USB1.1	12Mbps(1.5MB/s)	全速(Full-Speed)	5V/500mA	1998 年 9 月
USB2.0	480Mbps(60MB/s)	高速(High-Speed)	5V/500mA	2000 年 4 月
USB3.0	5Gbps(500MB/s)	超高速(Super-Speed)	5V/900mA	2008 年 11 月
USB 3.1	10Gbps(1280MB/s)	超高速+(Super-speed+)	20V/5A	2013 年 12 月
USB 3.2	20 Gbps (2.5 GB/s)	SuperSpeed USB 20Gbps	20V/5A	2017 年 7 月

4.3.4 典型的计算机总线系统

1. 以 8051 为核心的嵌入式控制器的总线

8051 是一种八位单芯片微控制器，属于 MCS-51 单芯片的一种，由英特尔公司于 1981 年设计。微控制器是 70 年代中期发展起来的一种超大规模集成电路芯片，把 CPU、RAM、

ROM、I/O 接口和中断系统集成于同一芯片内部。8051 系列微控制器采用典型的单总线结构如图 4.47 所示。

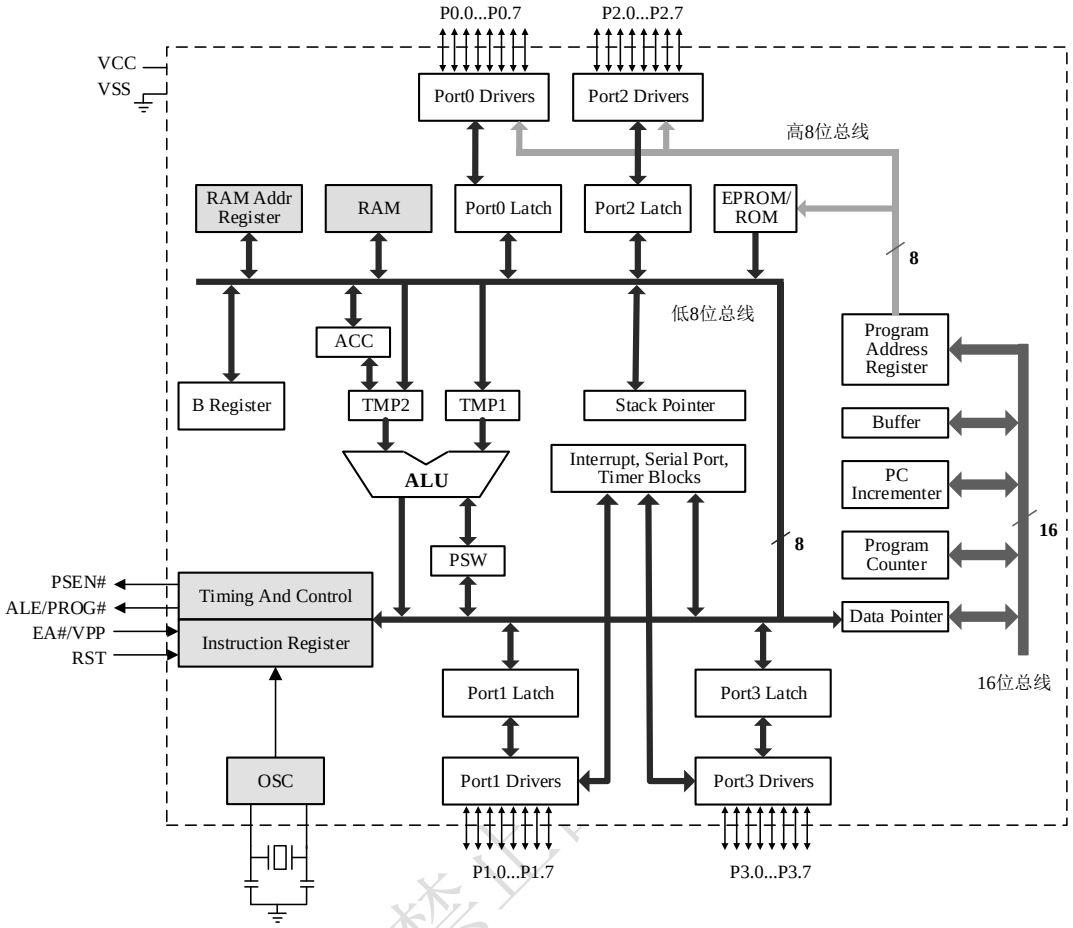


图 4.47　8051 系列微控制器的总线系统

2. 以 Cortex-M3 为核心的嵌入式控制器的总线

意法半导体的 STM 系列微控制器是基于 ARM 处理器核设计的。下面以 STM32F0 系列的微控制器为例分析其内部的多总线设计。如图 4.48 所示，Cortex™-M3 内核通过 DCode 总线访问数据存储器 SRAM，通过 ICode 总线访问代码存储器 Flash，同时挂接到系统总线上；两个 DMA 控制器通过 DMA 总线连接系统总线；外设模块则 APB 总线挂接，AHB 到 APB 的桥连接所有的 APB 设备。所有的总线通过一个多级的 AHB 总线构架相互连接的。关于 AHB 总线、APB 总线的细节信息参见 4.2 小节。

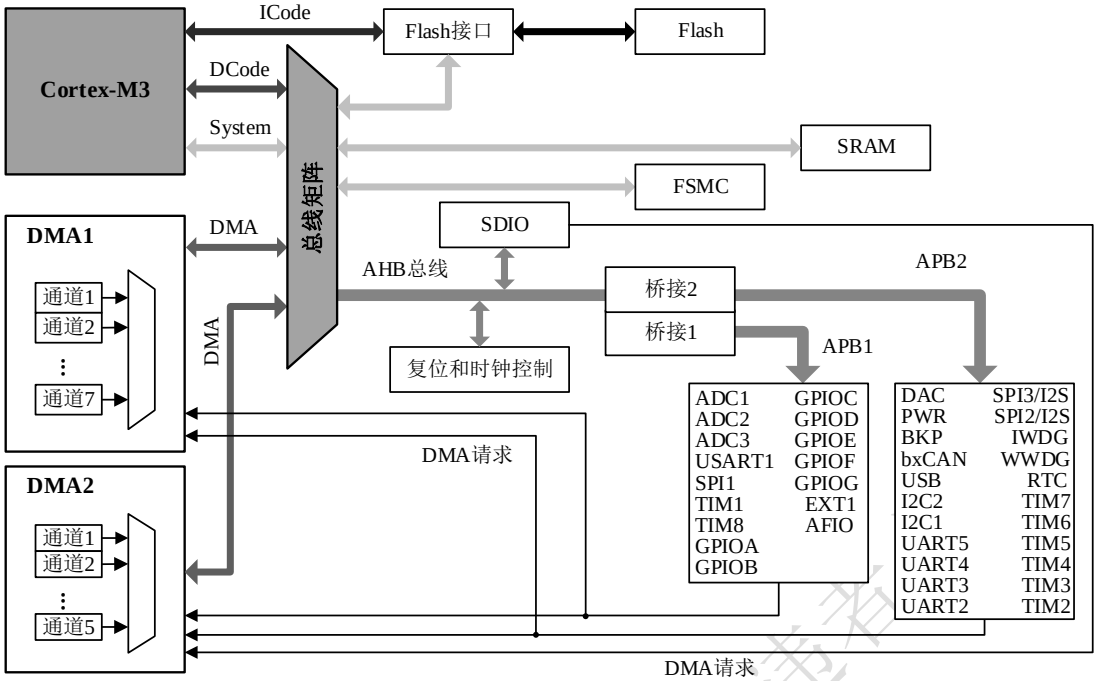


图 4.48　STM32F0 系列嵌入式控制器的多总线系统

3. 基于 8086 的初期 PC 系统的总线

英特尔在二十世纪 70 年代设计了 8086 微处理器，是 x86 架构处理器的鼻祖。8086 有 16 根数据线和 20 根地址线（其中 16 根是与数据线复用的），如图 4.49 所示，为基于 8086 微处理器构建的最大模式计算机系统。图 4.49 中 8284A 为时钟发生器，8288 为总线控制器，8259A 为中断控制器，8286 数据收发（缓冲）器。该系统为简单的单总线结构。

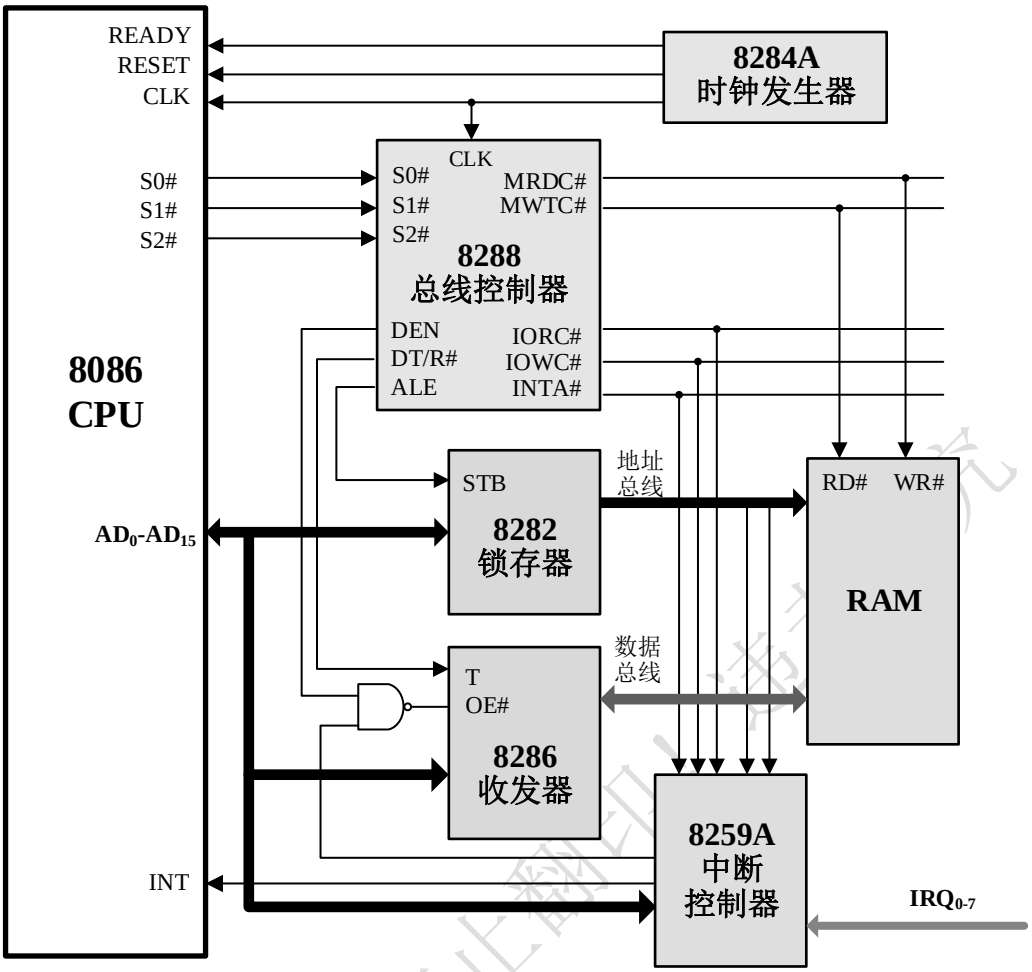


图 4.49 8086 系统最大模式的总线系统

4. x86 架构 PC 早期基于前端总线的多总线结构

早期基于 x86 架构的计算机系统，CPU 通过 FSB（Front Side Bus，前端总线）与存储器和外设进行数据交互，如图 4.50 所示。计算机主板上的北桥（north bridge）芯片负责联系内存、显卡等数据吞吐量最大的部件，并和南桥芯片（south bridge）连接。CPU 就是通过 FSB 连接到北桥芯片，进而通过北桥芯片和内存、显卡交换数据。

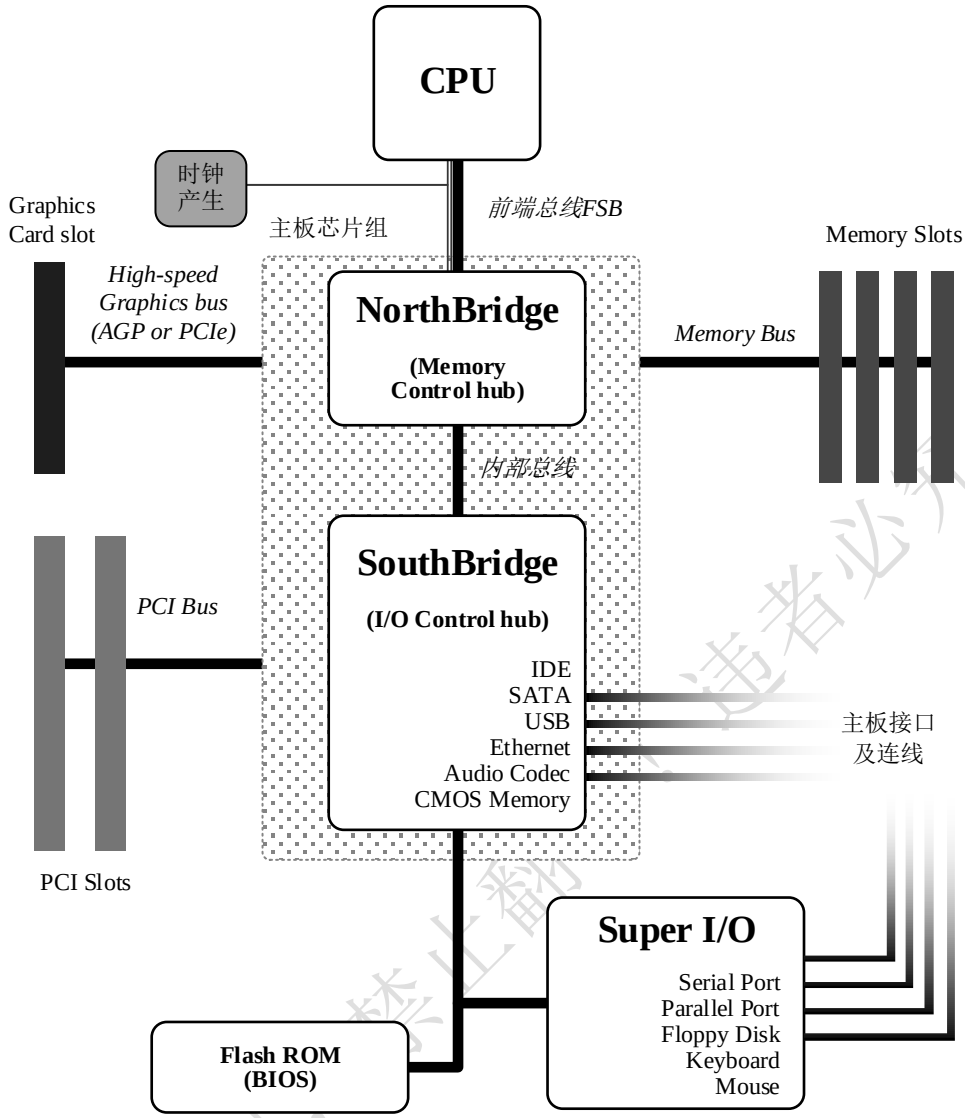


图 4.50 x86 架构基于前端总线的多总线系统

从逻辑功能角度来看，传统型的北桥芯片主要包括内存控制器、图形接口控制器与前端总线控制器、南北桥总线控制器等四个逻辑组成，分别负责同内存、显卡、CPU 和南桥芯片通信。在整合芯片组概念诞生之后，图形核心也被集成于北桥芯片内。与之对应，南桥芯片则侧重于功能性，诸如 PCI 总线、ATA 总线、USB、IEEE 1394、音频、网络等周边系统外设与处理器的通信都经由南桥芯片。早期的南桥功能简单，只包括 ISA、PCI、ATA 总线和键盘鼠标 PS/2 接口控制，音频、网络功能都通过额外的 ISA/PCI 扩展卡实现，在 2000 年左右，音频和网络都被南桥所集成，此后南桥的功能不断强化，除了 IEEE1394 接口和无线网络功能未集成外，南桥芯片已经可以提供其它所有的扩展功能。

从图 4.50 中可以看出，FSB 是 CPU 和外界交换数据的最主要通道，因此 FSB 传输能力对计算机整体性能作用很大。过去 PC 机 FSB 频率规格有 266MHz、333MHz、400MHz、533MHz、800MHz 等。FSB 频率高，代表着 CPU 与北桥芯片之间的数据传输能力越大，更能充分发挥出 CPU 的功能，反之较低的 FSB 频率会限制 CPU 性能的发挥。

5. x86 架构 PC 的 PCH 控制总线结构

随着主板芯片集成度的提高，同时，也为了提高 CPU 与存储器间数据交换的速度，CPU 集成了内存控制器，北桥剩余的功能与南桥功能逐渐融合在了同一枚芯片中。这种设计的优点在于：CPU 内核可以同内存系统直接通信，有利于提升 CPU 的指令效能。并且，每个 CPU 都拥有自己的内存系统，不必再与其他 CPU 分享，在多路服务器系统中可表现出巨大的性能优势。

2009 年，英特尔推出了单芯片设计的南北桥芯片整合方案。这颗主控芯片既不叫北桥，也不叫南桥，而是称作 PCH（Platform Controller Hub）芯片。PCH 芯片负责 PCIe 和 I/O 设备的管理，它实际上仅提供南桥的功能（此时处理器已经整合北桥功能，独立的北桥芯片实际上已不复存在）。这套设计标志着主流英特尔平台芯片组步入了单芯片时代，并沿用至今（2019 年）。注意，在一些高端服务器中并未采取这类整合手段，强大的 PCIe 图形控制器依然是独立的。

如图 4.51 所示，DMI（Direct Media Interface，直接媒体接口）是英特尔开发的用于连接主板南北桥的总线。DMI 采用点对点的连接方式，基于 PCI-Express 总线，跟随 PCIe 总线的换代而换代。图 4.51 中所示的 DMI 3.0，单通道传输速率达到 8GT/s。

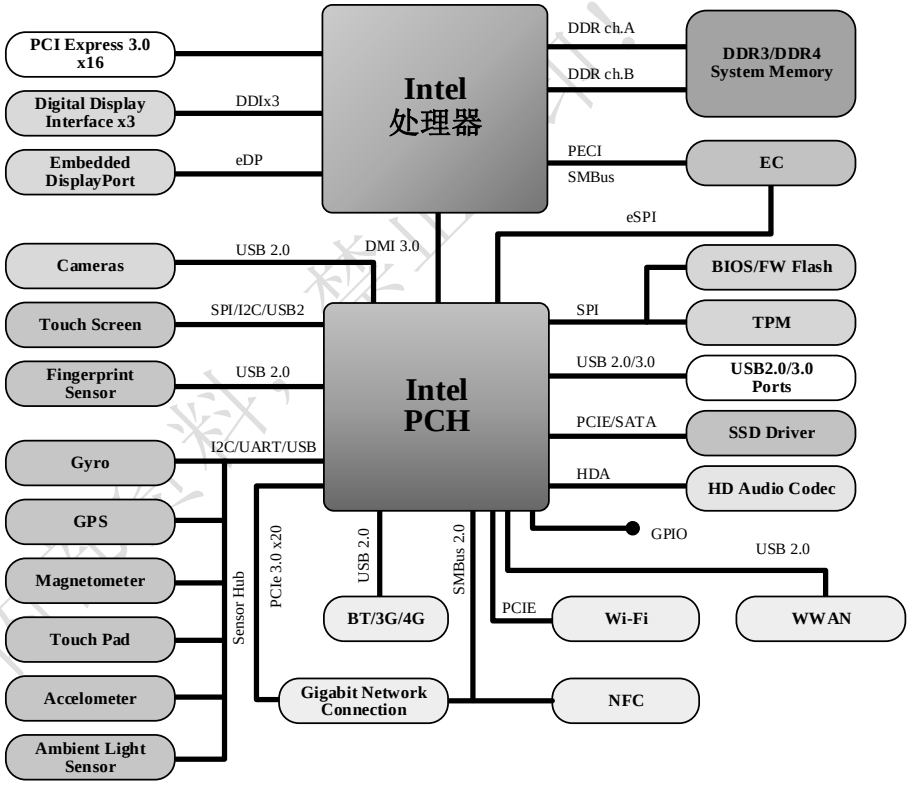


图 4.51　x86 架构单 PCH 芯片的多总线系统

2019 年 Intel 发布的第十代智能英特尔® 酷睿™ 移动式处理器，添加了一些新的接口标准，如雷电接口、USB3.1、SATA3.0，其总线系统如图 4.52 所示。值得注意的是，图 4.52 中 PCH 已经不再是一颗单独封装的芯片，而是和 CPU 一起被封装在一起。CPU 和 PCH 虽然是两个独立设计的电路模块，但是通过封装（package）放在了同一颗芯片的内部，CPU 和

PCH 通过 OPI（On Package DMI interconnect Interface）进行连接。

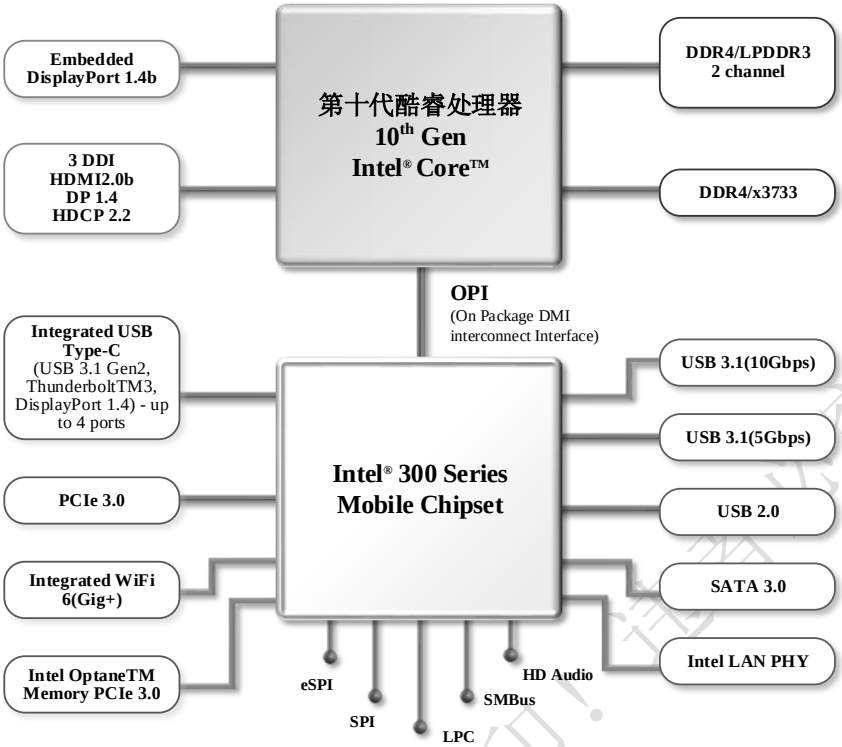


图 4.52　第十代智能英特尔®酷睿处理器的片上多总线系统

4.4 输入/输出接口

与各种外部设备进行数据传输是计算机的一个基本特性。这种通信能力使人机通信得以实现，例如，在 PC 上可以使用键盘和显示器来进行文本及图形图像的处理。更广泛地，在计算机网络中，利用不同计算机之间的通信能力来进行数据交互。在嵌入式应用中，计算机往往集成在特定的设施中，如家用电器、制造设备、车辆系统、移动电话、自动售货机、自动取款机等。在这些嵌入式应用场景中，触摸屏、传感器、数码相机、麦克风等往往是计算机的输入设备。与此同时，显示屏、扬声器、电机、机械臂等则作为计算机的输出设备。

计算机应该具备与各种输入或输出设备进行数据传输的能力。在多数应用场景下，计算机的处理器全程参与这些交互过程。而还有一些场合下，数据的传输直接发生在不同的输入输出设备之间（如硬盘和内存之间）。换言之，不同的应用场景中，处理器参与数据传输的程度有深有浅。本小节和后续小节将阐述上述计算机与外部设备进行数据传输的基本方式、典型的接口电路。

输入/输出接口（简称 I/O 接口），也往往被称为 I/O 接口电路或 I/O 控制器。完整的 I/O 接口不仅包括外部设备与 CPU 或计算机之间的硬件电路，也包括相应的驱动程序。通常情况下，不同外部设备所需要的接口电路和驱动程序是不同的。同一种外部设备连接到不同的计算机时所需要的接口电路和驱动程序也不同。本章所讨论的 I/O 接口限定在硬件方面，即，本章关注的是 I/O 接口电路。

4.4.1 输入/输出接口概述

1. I/O 接口的功能

外部设备的功能是多种多样的。有些外部设备作为输入设备，有些外部设备作为输出设备，也有些外部设备既作为输出设备又作为输入设备，还有些外部设备作为检测设备或控制设备。每一种外部设备的工作原理不同，它们与微处理器之间交换数据的时候需要用不同的电路单元进行适配。

通常外设的输入信号形式也是多种多样的。对于一个特定的外部输入设备来说，其信号形式可能是数字的，也可能是模拟的。模拟信号需要经过 A/D 接口电路转换为微处理器可以识别的数字信号。即便外部设备提供了数字信号，也不一定能被微处理器所接受。例如，外部设备输入的串行数据信息需要进行串/并转换后，才能送入 CPU。与此同时，微处理器向外部设备输出信号的时候，也要考虑外部设备能接收的信号形式。如，来自 CPU 的并行数据需要转换为串行数据才能为串行外部设备所使用；再如，CPU 的数字信号需要经过 D/A 转换才可以送给支持模拟信号的外部设备。因此，为了适配外部设备与微处理器不同的信号形式，往往接口电路需要提供 A/D 或 D/A 转换能力，或者并/串和串/并转换能力。

另外，外设的工作速度通常比 CPU 的工作速度慢，且不同种类外设的工作速度也不同。例如，使用键盘人工输入的时候，每个字符输入的间隔时间可达数秒钟；电脑游戏中鼠标的输入则可能每秒钟内有数次点击；数字摄像头每秒钟需要传输几十副图像；普通家用的打印机往往一分钟内只能处理不到十页的文档；超市收银中常用的针式打印机一秒钟只能打印数个字符。也有一些外设工作速度高于普通微机或嵌入式计算机系统，如高速示波器每秒钟可以产生数百万个样本数据；高清晰度的摄像机每秒钟可以产生数千兆比特的数据。故 I/O 接口电路还需要完成微处理器与外设之间的速率匹配。

为了协调上述微处理器与外设之间的不一致，针对不同的外设，引入专门的接口电路，微处理器通过接口电路接收外部设备送来的信息或将信息发送给外部设备。对于输入设备而言，接口通常起转换和缓冲作用。转换的含义包括模拟量到数字量的转换、串行数据到并行数据的转换、数据格式的转换和电平的转换等。对于输出设备来说，需要类似的的转换和缓冲机制。往往接口要将微处理器输出的并行数据放到缓冲器中，并将它转换为外部设备所需的形式，这种形式可能是并行的、串行的或模拟信号等。

2. I/O 接口的分类

接口种类的划分方式有多种。根据接口工作的方式，及其在整个系统中的工作性质和作用，通常有如下不同的分类方式。

（1）按照数据传输方式分类

按照数据传输方式可分为串行接口、并行接口。并行接口是指微处理器与 I/O 接口之间、I/O 接口与外部设备之间均以多个位（比特）的并行方式送数据。串行方式是指接口与外设之间采用单个位串行方式传送数据。并行接口适用于传输距离较近、传输速度较高的场合，接口电路相对简单。串行接口则适用于传输距离较远、传输速度相对较低的场合，传输线路成本较低，接口电路相对前者更复杂。一般来说，在电路的时钟频率不太高的数字电路系统中，高效率的传输多采用并行接口。随着技术的发展，近些年也出现了一些近距离的高速串

行传输技术，如 PCIe、USB 等。

（2）按时序控制方式分类

按时序控制方式可分为同步接口、异步接口。同步接口是指接口上的数据传输由统一的时钟信号同步控制，这个时钟信号是接口上的发送方和接收方共有的。异步接口上的数据传送则采用异步应答的方式进行，不依赖于统一的时钟。

（3）按主机访问 I/O 设备的控制方式分类

按主机访问 I/O 设备的控制方式可分为程序查询接口、中断接口、DMA 接口。程序查询方式是指 CPU 通过程序来查询 I/O 设备的状态（状态信息通常保存在状态寄存器中），并执行相应接口的数据访问操作。如果需要传送的数据总是准备好的，那么不需要任何状态信息或联络信号，CPU 直接执行输入输出指令即可实现读取 I/O 设备数据或输出数据至 I/O 设备的操作，这种方式常被称为无条件传送方式。中断接口是指 I/O 设备与 CPU 之间采用中断方式进行联络，即 I/O 设备向 CPU 提出中断请求，CPU 响应中断请求后运行中断服务程序与 I/O 设备进行信息交换，因此接口中包含中断控制逻辑。DMA 接口是指 I/O 设备与主存间采用直接内存访问的方式传递数据，这种交换方式一旦建立后，不需要 CPU 参与即可实现存储器与 I/O 设备间的数据传送。

3. I/O 接口规范的常规内容

如前所述，接口设计的目的是为了协调上述微处理器与外设之间的不一致，用来实现速率匹配、缓冲、数据格式的转换和电平的转换等功能。依据不同外设的特点，形成了很多典型的接口设计，这些全球通用的接口设计方案往往以工业界公认的行业标准（standard）或行业规范（specification）形式出现。

输入/输出接口电路的功能隶属于通常意义的物理层功能。物理层是国际标准化组织 ISO（International Organization for Standardization）定义的开放系统互联 OSI（Open System Interconnect）模型中最低的一层。物理层规定了为传输数据所需要的物理链路创建、维持、拆除，而提供具有机械的、电子的、功能的和规范的特性。概括来说，在输入/输出接口的标准中，需要规定四方面的特性：机械特性、规程特性、功能特性、电气特性。这四方面的特性需要约定的内容如下。

- 机械特性，指明通信实体间硬件连接接口的机械特点。如接口所用接线器的形状和尺寸、引线数目和排列、固定装置等。
- 电气特性，规定了在物理连接上，导线的电气连接及有关电路的特性。一般包括接口电缆的各条信号线上出现的电压的范围、信号的识别、最大传输速率的说明、发送器的输出阻抗、接收器的输入阻抗等电气参数。
- 功能特性，指明物理接口各条信号线的用途，如某条信号线上出现的高低电压表示什么含义。如接口信号线分为数据线、控制线、地址线、时钟线和接地线等。
- 规程特性，规定接口传输比特流的全过程，及不同传输事件发生的先后顺序，规定了物理连接建立、维持和交换信息时，收发双方在各自电路上的动作序列。

需要注意的是，在一些输入/输出接口的标准定义中，除了上述物理层功能，也纳入了数据块格式、数据块检验方式等链路层功能（OSI 模型的第二层）。例如，接口电路通常采

取奇/偶校验方式检错并通过重发方式纠错。例如，图 4.53 所示为 IBM 的 BSC（Binary Synchronous Communications）协议定义的传输数据格式。BSC 是一种典型的面向字符的同步通信协议，每个数据块（信息帧）由三个部分组成。

- 同步字符作为一个数据块（信息帧）的起始标志。
- 多个连续传送的数据。
- 循环冗余校验码（CRC)。

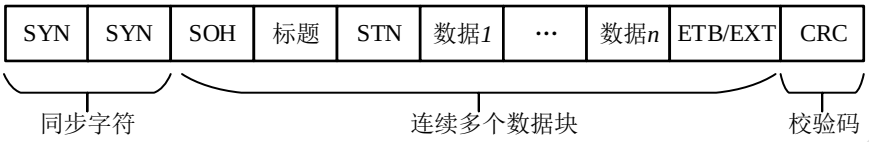


图 4.53　面向字符的同步通信协议定义的信息帧

BSC 协议规定了十个特殊字符（称为控制字符）作为信息传输的标志。图 4.53 中，SYN（SYNchronous character）为同步字符，每帧可加一个（单同步）或两个（双同步）同步字符。SOH（Start Of Header）为标题开始。标题（Header）包含源地址（发送方地址）、目的地址（接收方地址）、路由指示。STX（Start of Text）是正文开始。数据块即正文（Text）由多个字符组成。ETB（End of Transmission Block）为块传输结束，标识本数据块结束。ETX（End Of Text）为全文结束，全文分为若干块传输。校验码对从 SOH 开始，直到 ETB/ETX 字段计算 CRC 检验码。

4. I/O 接口的结构

简单来说，输入/输出接口电路的基本功能就是，通过主机与外设之间以下三类信息的交互，来完成数据的控制传送。

- 外设的状态信息（输入）：编码方式的二进制数据，用于指示外部设备的状态情况。主机通过了解外设的当前状态，来决定是否可以进行数据传送。
- 数据信息（输入/输出）：要传送的二进制数据。
- 控制信息（输出）：编码方式的二进制数据，控制外设的工作模式与操作方式等。

通常 I/O 接口电路中包含一组寄存器，分别用于存储这三类信息，这些寄存器被称为端口寄存器或 I/O 端口（I/O Port），简称为端口。事实上，计算机系统中所有外设都可以抽象成为一组端口寄存器，CPU 通过对各个端口寄存器的访问实现与外设的数据交换。对应上述三种信息，外设接口电路中的端口寄存器分为三类：

- 数据端口（寄存器）：CPU 和外设之间用来暂存传送数据的寄存器，在输入/输出过程中，对数据起锁存和缓冲的作用。
- 状态端口（寄存器）：用来存放外设或者接口电路单元本身状态的寄存器。常见的状态位有准备就绪位（Ready）忙碌位（Busy）、错误位（Error）等。
- 控制端口（寄存器）：用来存放 CPU 发往外设的控制命令的寄存器。常见的命令信息有启动、停止、允许中断等。

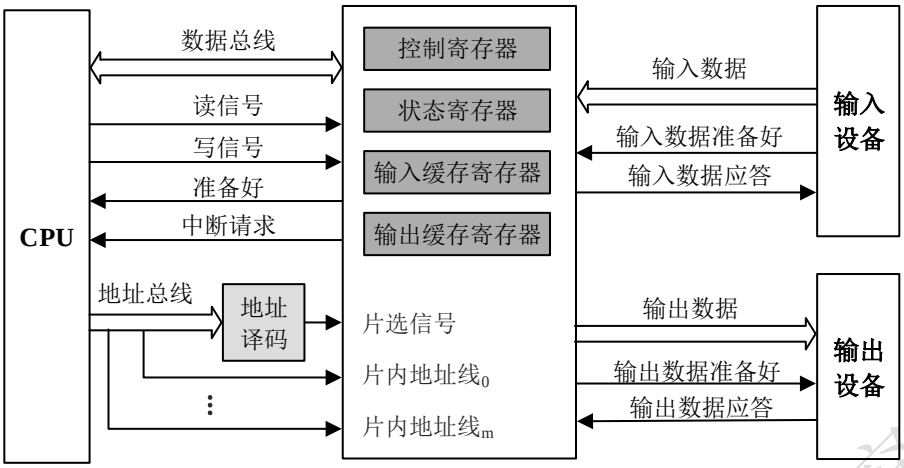


图 4.54 I/O **接口电路的典型结构**

如图 4.54 所示为 I/O 接口的典型结构。该接口电路包括一个控制寄存器、一个状态寄存器、一个输入数据缓存寄存器和一个输出缓存寄存器。I/O 接口与 CPU 的数据传送通过数据总线进行。读信号用以指示数据从 I/O 传输到 CPU，写信号用以指示数据从 CPU 传输到 I/O 接口。高位地址生成片选信号，选择当前接口；低位地址连接外设片内地址线，选择内部端口。

图 4.54 中左边给出的是与 CPU 的接口信号，右边给出的是与外部设备的接口信号。CPU 对接口电路的输入端口/输出端口的读/写操作可实现微处理器与外设的数据传送。CPU 对 I/O 接口的控制寄存器的写操作可实现对外设的控制操作。而外设的状态信息可由 CPU 读取接口电路的状态寄存器获得。

以打印机为例，数据端口是存放 CPU 输出给打印机的的数据信息。状态端口存放反映打印机的工作状态信息，如打印机是否准备好、是否忙碌、是否缺纸、是否卡纸等。CPU 可以通过状态信息来确保与打印机的正确连接。控制端口存放 CPU 送给打印机的控制命令，如启动打印、中止打印、打印机的走纸换行等，CPU 对外设的控制操作都通过控制端口进行。一般来说，CPU 对数据端口是可读且可写的，而对状态端口只能读，对控制端口只能写。

5. I/O 端口编址

在计算机硬件系统中可以含有多个接口以含有多个 I/O 接口，每个 I/O 接口可以含有多个 I/O 端口。为了保证能访问到每个 I/O 接口的各个 I/O 端口，这些端口寄存器在计算机系统中像存储器单元一样需要编址，称之为端口地址，处理器通过端口地址可对各个端口寻址访问。I/O 端口有两种编址方式：独立编址和统一编址。两种编址方式如图 4.55 所示。

1） I/O 端口统一编址

I/O 端口统一编址也称内存映像编址（memory mapped I/O addressing）方式。将外设接口中的 I/O 寄存器（即 I/O 端口）视为主存的存储单元，每个端口占用一个存储单元的地址，把主存地址的一部分划出来用作 I/O 地址空间。如摩托罗拉公司的 68 系列、英特尔公司的 51 系列 ARM 处理器，Power PC 等处理器，把这些寄存器看作内存的一部分，寄存器参与

内存统一编址。统一编址也称为“I/O 内存”方式，外设寄存器位于“内存空间”（很多外设有自己的内存、缓冲区，外设的寄存器和内存统称“I/O 空间”）。

I/O 端口统一编址时，访问寄存器就通过访问一般的内存指令进行，故这种情况下 CPU 没有专门用于 I/O 设备的读写指令。编程的时候可以使用存储器的访问指令来访问 I/O 端口，因此可以充分利用存储器指令的强大功能，编写程序方便。这种方式的缺点是，减小了有效的存储器空间，同时程序员必须知道 I/O 端口的地址在存储器地址空间中的分配情况。

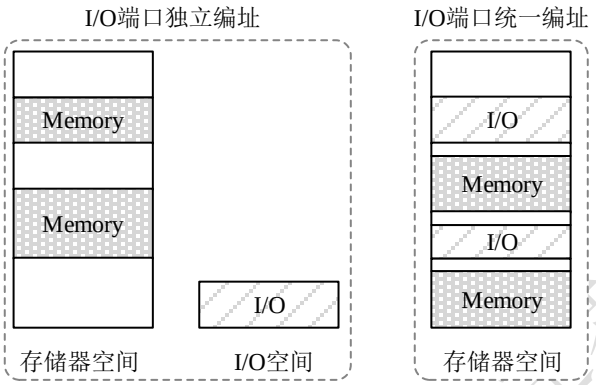


图 4.55 I/O 端口两种编址方式下的地址空间示意图

2） I/O 端口独立编址

I/O 端口独立编址）又称为分离编址（Isolated I/O Addressing。存储器和 IO 端口分开编址，即存储器和 I/O 端口的地址空间相互独立。独立编址方式与前述统一编址方式相反，所有端口地址单独构成一个地址空间，即 I/O 编址空间与存储器空间互不相干。

这种方式下的 I/O 地址空间与存器地址空间可以重叠，CPU 通过不同指令来区分操作对象是 I/O 端口还是存储器。这种编址方式的好处是程序员无须同时兼顾考虑 I/O 端口的地址空间和存储器的地址空间分配情况。缺点是需要专门的 I/O 访问指令（例如输入指令 IN、输出指令 OUT），不能利用灵活的存储器访问指令访问 IO 端口。同时，I/O 端口专有操作指令执行时系统总线上须有相应的控制信号指明当前指令的寻址空间是存储器或 I/O 空间。80x86 系列微处理器就采用独立的 I/O 编址方式。CPU 使用地址总线中的 A0~A15 来寻址 I/O 端口，连续两个 8 bits 的端口可以组成一个 16 bits 的端口，连续四个组成一个 32 bits 的端口，最大 I/O 空间是 64K 个字节端口（或 32K 个字端口，或 16K 个双字端口）。

4.4.2 输入/输出接口的数据传送方式

如前所述，外设的多样性使外设及其接口电路的差异极大，故而需要多种 CPU 与外设接口交换信息的方式。常用的数据传送方式包括：无条件数据访问方式、状态查询数据访问方式、中断数据访问方式和 DMA 方式等。

1. 无条件传送方式

有一些外设，处理器在访问时不必关心其状态，这些设备永远处于“准备好”的状态，如按键、继电器、数码管、发光二极管等。CPU 访问这类设备时，输入设备总是随时准备好向 CPU 提供数据，而输出设备也总是随时准备好接收 CPU 送来的数据。此类外设称为无条

件外设。CPU 对此类外设进行输入输出操作时不需要考虑外设的状态，故此类外设的访问方式称为无条件传送方式。访问这类 I/O 设备简单使用指令直接读/写数据端口即可。

采用无条件传送方式的接口电路很简单。如无条件输入设备的计算机接入，考虑到外设数据保持时间相对于 CPU 的处理时间要长得多，输入数据不能影响系统总线的正常使用因而外设数据要经过三态缓冲器和 CPU 数据总线相连。如图 4.56 所示。

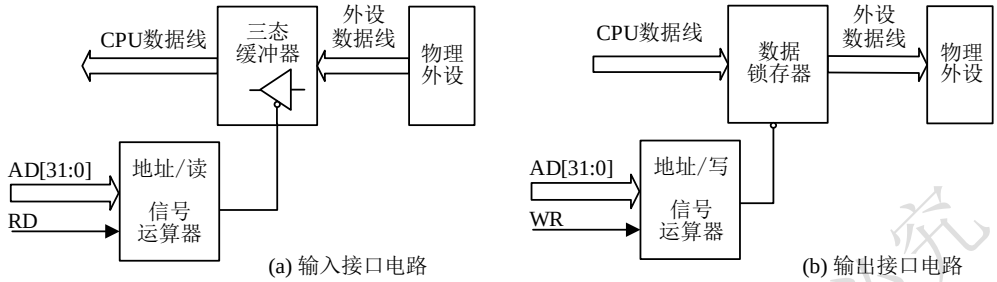


图 4.56 无条件数据访问方式 I/O 接口电路示意

三态缓冲器（three-state buffer），又称为三态门、三态驱动器，其三态输出受使能输出端的控制，当使能输出有效时，器件实现正常逻辑状态输出，当使能输入无效时，输出处于高阻状态，即等效于与所连的电路断开。由于外设的数据总是准备好的，随时可通过三态缓冲器上传到系统总线。当 CPU 执行该外设输入的指令（即读 I/O 端口）时，三态缓冲器的输出使能，从而物理外设中早已准备好的输入数据加载到数据总线送达 CPU。而三态缓冲器未被选通时，外设数据被高阻隔离。

同理，无条件输出的外设，由于 CPU 速度很快，而物理外设的速度比较慢，这就要求电路的输出端保持，因而一般需要一个锁存器，如图 4.56 所示。CPU 执行输出指令时，产生数据锁存器的选中信号，从而 CPU 输出的数据经过数据总线送入输出数据锁存器。输出锁存器保持这个数据，供外设使用，直至下次新数据写入。

2. 查询传送方式

有一些外设，处理器访问的时候需要关心外设的状态，只有状态许可方可访问外设。例如对 A/D 转换器的访问，只有模/数转换结束后，CPU 才能读取转换的结果。又如，在使用串口外设发送数据时，只有串口发送缓冲区有空位置了，CPU 才可以写入数据；同理，CPU 接收数据的时候，也需要在接收缓冲区不空（有接收数据）时进行读取。此类须满足一定条件进行数据访问的外设称为条件访问外设。由于 CPU 对此类外设进行输入或输出操作时需要考虑外设的状态，故此类外设的访问方式称为条件传送方式。计算机中大多数外设都是条件访问方式。

实现这种有条件的 I/O 的访问有两种可能的方法：程序查询外设状态方式（简称查询方式，或状态查询方式，或程序查询方式）和中断控制方式。

状态查询方式的原理和过程为：CPU 在执行数据端口访问指令前，先查询该设备的状态，当设备处于“准备好”状态时，CPU 才执行对设备数据端口的输入/输出指令，与外设交换信息。为了实现状态的查询，接口电路中既要有数据端口，又要有状态端口。状态查询方式 I/O 接口电路原理如图 4.57 所示。

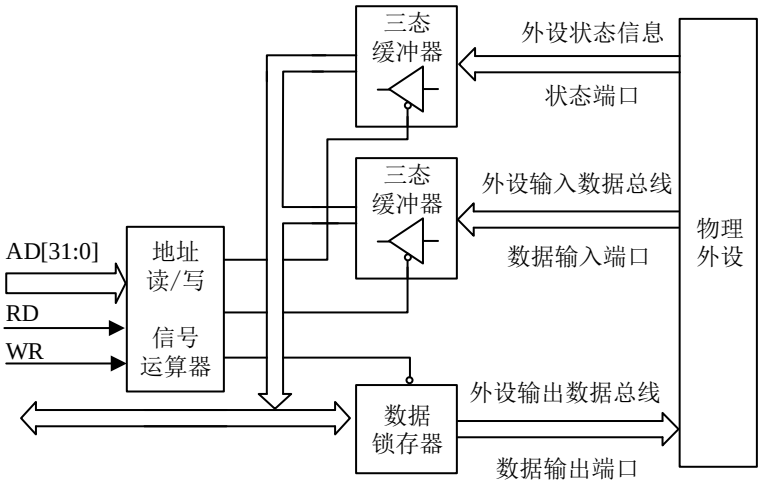


图 4.57　状态查询方式 I/O 接口电路示意

状态查询方式进行一次数据传送的步骤和过程如下，流程图如图 4.58 所示。其基本步骤为：①CPU 从状态端口中读取状态字。②CPU 检测相应的状态位是否满足数据访问的就绪条件。③如果不满足，则重复步骤①和②，若外设已处于就绪状态，则访问数据端口，完成数据收/发。

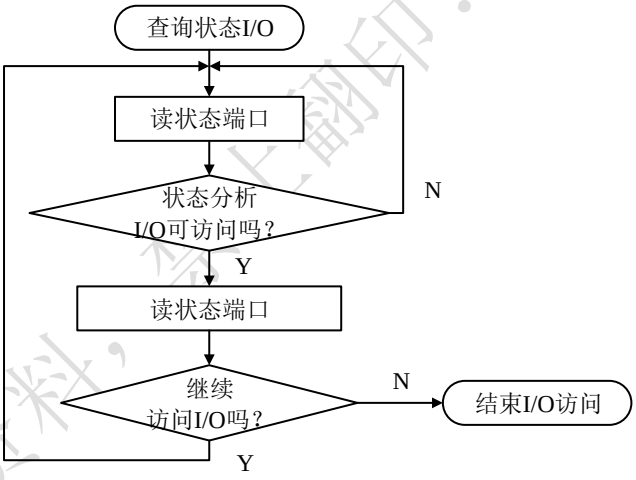


图 4.58　状态查询方式 I/O 控制流程

从上述状态查询方式数据传输的过程可知，如果外设的速度远远低于 CPU 的速度，则 CPU 会在步骤①和②等待，这会导致大量的 CPU 时间片消耗在等待环节，进而影响到计算机系统的整体性能。故而状态查询方式中，CPU 的利用率不高。如果 CPU 查询外设的周期太短，则长期处于等待传输状态，等待期间不能对其他事件做出响应，CPU 效率降低。反之，如果 CPU 查询外设的周期太长，就不能对外设状态的改变及时做出响应，可能导致有用信息丢失。可能的改进方式为：将 CPU 等待外设就绪变成外设主动将就绪的状态信息告知 CPU，这样就能够有效地节约 CPU 的时间。这种思路就是在后续小节将要讨论的中断传输方式。

3. 中断传送方式

在程序查询方式中，由于 CPU 对外部设备会执行大量状态查询的指令，CPU 的利用率不高。如果 CPU 采取不断查询的方法，则长期处于“等待”状态，不能进行别的处理，也不能对其他事件做出响应。即使采取周期性查询的方法，也不能完全克服上述缺点。解决这一问题的一种可行方法是使用程序中断传输方式。

在程序中断传输方式下，I/O 端口状态查询工作交给中断控制器去完成。当外部设备需要传输数据时，外设触发中断控制器的中断请求信号并由中断控制器把该信号送给 CPU，CPU 收到中断请求信号后，暂停当前程序的运行，而转去执行读/写接口数据的中断服务程序，实现 CPU 与外设的数据传输。执行完中断服务程序后，CPU 再回去继续执行被中断的程序。故而程序中断传输方式能克服程序查询传输方式的缺点。图 4.59 所示为程序中断传输方式的接口电路示意图。

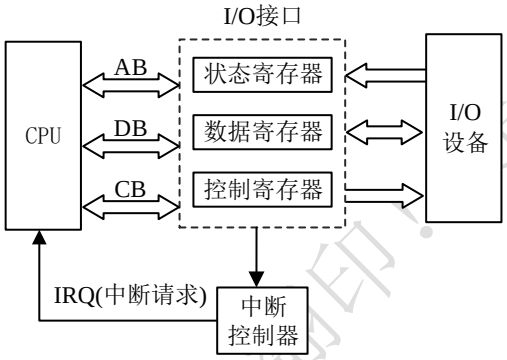


图 4.59　中断方式 I/O 接口电路示意

程序中断传输方式和前述的程序状态查询方式相比，都是有条件输入/输出控制的具体实现手段。中断方式有利于提高了 CPU 使用率，便于 CPU 和外设并行工作，因而成为最常用的输入/输出方式。目前，绝大多数的微处理器都具有中断功能，可以实时高效地执行输入/输出访问，提高整体系统的性能。同时，中断技术也不局限于 I/O 接口的处理，在其他许多方面也有重要的应用，例如实时控制、故障处理等。

中断是计算机系统，尤其是嵌入式系统中较为重要的一项技术。本小节后续针对中断进行较为详细的分析。

1） 中断的概念

“中断”是一种信号，中断信号通知 CPU 已发生了某种事件，需要 CPU 去处理或为其服务。如用户使用键盘时，每次击键会发出一个中断信号，告诉 CPU 有“键盘输入”事件发生，要求 CPU 读入键盘当前的输入值。中断不仅能由计算机系统的外设产生，也可以由 CPU 内部事件引起，如 CPU 的逻辑运算单元发生除零错误时，就会产生“除零”的中断。

为了便于区分源自 CPU 的内、外的中断源，早期的资料习惯把来自 CPU 外部的外中断称作中断（interrupt），而把来自 CPU 内部的内中断称作陷阱（trap）。如今的微处理器芯片则习惯用异常（exception）来描述所有能打断 CPU 正常执行的事件。例如，ARM 公司的 Cortex-M 处理器的异常处理（exception handling）机制与此处描述的中断机制原理上是一致的。在本小节中，对术语“中断”和“异常”不作严格区分。

图 4.60 所示为某事件通过中断向 CPU 请求服务的过程示意图。CPU 在执行当前已运行程序的过程中，可以接受和检测中断请求信号；在系统允许处理中断的条件下，一旦发生中断，CPU 会暂停现行程序的运行，转去执行中断服务子程序，为中断请求者服务；服务完成后，CPU 将返回原来的程序继续向下执行。

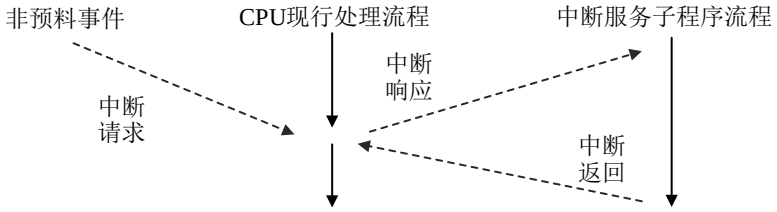


图 4.60　中断方式示意图

下面介绍中断技术涉及的几个概念。

(1)中断源

中断源指引起中断的事件或发出中断请求的来源，如电压跌落、数据校验出错等异常事件，或键盘、磁盘、鼠标、网络接口等外设的数据传输请求等，如图 4.61 所示（注：图 4.61 中 NMI，INTR 引脚的差异稍后在中断屏蔽知识点部分予以说明）。ARM 公司的 Cortex-M3/M4 定义的异常来源则包括不可屏蔽中断（NMI）、中断、内存管理错误（memory mange fault）、总线错误（bus fault）、非法指令（usage fault）、系统定时器（SysTick）错误等。

通常计算机系统的中断处理模块可以管理多个中断源并处理多种不同类型的中断。为了能够识别不同的中断源，中断处理模块会对中断源进行命名或编号，这个编号被称为中断类型码或中断类型号。

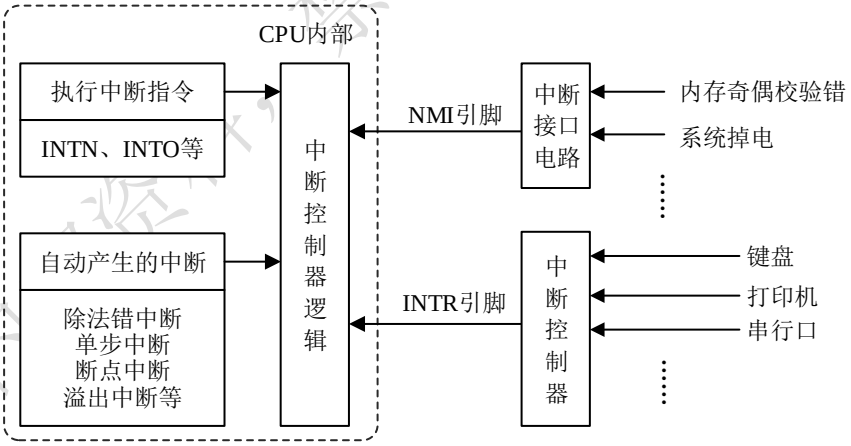


图 4.61　典型计算机系统的中断源

(2)中断向量

8086 和 ARM 等处理器中，中断向量是中断服务子程序的入口地址。通常的处理器设计中，所有的中断服务子程序入口地址会集中存放在存储器的特定区域（如 8086 系统中是内

存的最低 1KB），这个特定的区域称为 IVT（Interrupt Vector Table，中断向量表[4]）。当发生中断的时候，系统根据中断类型码到中断向量表中查询中断向量。

(3)断点和现场

断点指被中断的主程序下一条待执行指令在存储器中的地址，也就是中断返回时的程序地址。为了保证在中断服务子程序执行完后能正确返回到原来的程序，中断系统必须能在中断发生时自动保存断点，并在中断返回时自动恢复断点。

现场指中断发生时原主程序的运行状态，一般指系统标志（状态）寄存器和其他相关寄存器中的内容。为了保证在中断返回后能继续正确地执行原来的程序，中断系统必须在中断发生时对现场进行保护（即保存状态信息），并在中断返回时恢复现场（即恢复状态信息）。

(4)中断优先级和中断嵌套

如果多个中断源同时提出中断请求，就需要按预定义的规则依次进行服务。最常用的规则是中断优先级，就是给每个中断源指定一个优先级，根据优先级高低协调多个中断服务程序的先后顺序。当系统中同时发生多个中断的时候，中断系统就按优先级对中断源的中断请求进行排队，根据预先定义好的顺序优先处理重要的（优先级高的）中断请求。

在实际系统中，CPU 正在为某个中断请求服务时，可能会再次接收到其他中断源的中断请求。如果后发生的中断请求的优先级比正在处理的中断请求的优先级高，则中断系统应该能再次中断正在执行的中断服务子程序，转去处理新的、优先级更高的中断请求，这就是中断嵌套。很多处理器的中断系统设计中对中断嵌套的层数做出了一定的限制（如最多嵌套的层数不超过三层）。图 4.62 是两级中断嵌套的示意图，图中阿拉伯数字表示各程序片段的执行顺序。

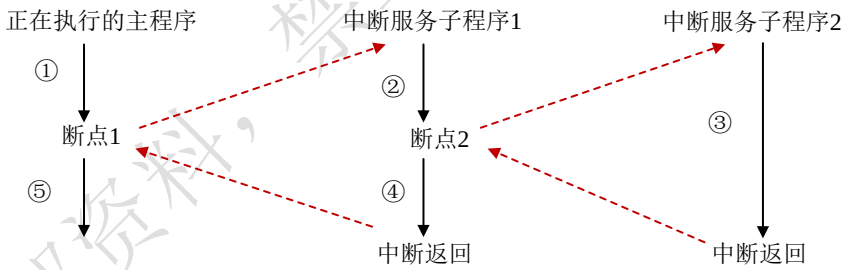


图 4.62　中断嵌套示意图

(5)中断屏蔽

在一些情况下，CPU 可能暂时不对中断请求信号做出响应或处理，这种情况称为中断屏蔽。中断屏蔽可能在两种情况下发生：一种情况是中断系统允许设置中断屏蔽标志（或中断允许标志），以屏蔽某些中断源的请求；另一种情况是当系统在处理优先级别较高的中断请求时不会响应后到的级别较低的中断请求，也就是中断系统会自动屏蔽优先级低的中断，不允许其产生中断嵌套。

[4] 在其他一些处理器中，用中断描述符表 IDT（Interrupt Descriptor Table）来保存中断源与对应中断服务子程序入口地址的映射关系。中断描述符表原理与此处阐述的中断向量表类似，但具体格式有差异。

并非所有的中断都可以被屏蔽。时钟故障、电源电压跌落故障、存储器校验错误等特殊情况的中断请求是不可屏蔽的。这里的“不可屏蔽”，是指该中断不受中断屏蔽标志位影响，即只要当前指令执行完毕，CPU 就会检测并响应这些中断请求。按照是否可以被屏蔽，可将中断分为两大类：NMI（Non Maskable Interrupt，不可屏蔽中断）和 INTR（Interrupt Require，可屏蔽中断）。不可屏蔽中断源一旦提出请求，CPU 必须无条件响应，而对可屏蔽中断源的请求，CPU 可以响应，也可以不响应。

2） 中断处理过程

在计算机系统中，中断的整个过程分为：中断响应、中断处理和中断返回等三个阶段。

(1) 中断响应

中断响应指 CPU 从确定响应目标到跳转至中断服务子程序入口的过程。中断请求可能在任意时刻产生，但 CPU 只有在当前指令执行完后才能响应。如果有多个中断源同时发出中断请求，CPU 还需要进行优先级排队，在允许中断的条件下，响应和处理优先权最高的中断请求。中断请求检测与中断优先级判断均由 CPU 内部的硬件电路自动完成的。

图 4.63 中上半部分虚线框内描述了 8086 CPU 的中断响应过程。大虚线框内即为 CPU 的中断响应过程。图 4.63 中开始部分的查询分支就是 8086 CPU 对中断优先级别的判断过程，其中 TF（Trap Flag）标志位和 IF（Interrupt Flag）标志位是 CPU 内部中断状态寄存器（或称中断标志寄存器）中的位。*TF*=1 表示产生了陷阱；*IF*=1 表示产生了（外）中断。

不同类型的处理器中，依据其中断控制器电路设计中断优先级别的判断过程会有细节差异，但都需要在确定需要响应中断之后获取中断类型码，然后根据中断类型码去查找中断向量表中对应的表项（中断服务子程序的入口地址）。一般的计算机系统中，不同来源中断的类型码由 CPU 设计开发人员确定，发生中断时，电路自动产生对应的类型码。有些系统中，外部中断源的中断类型码可以通过总线从外设获取。

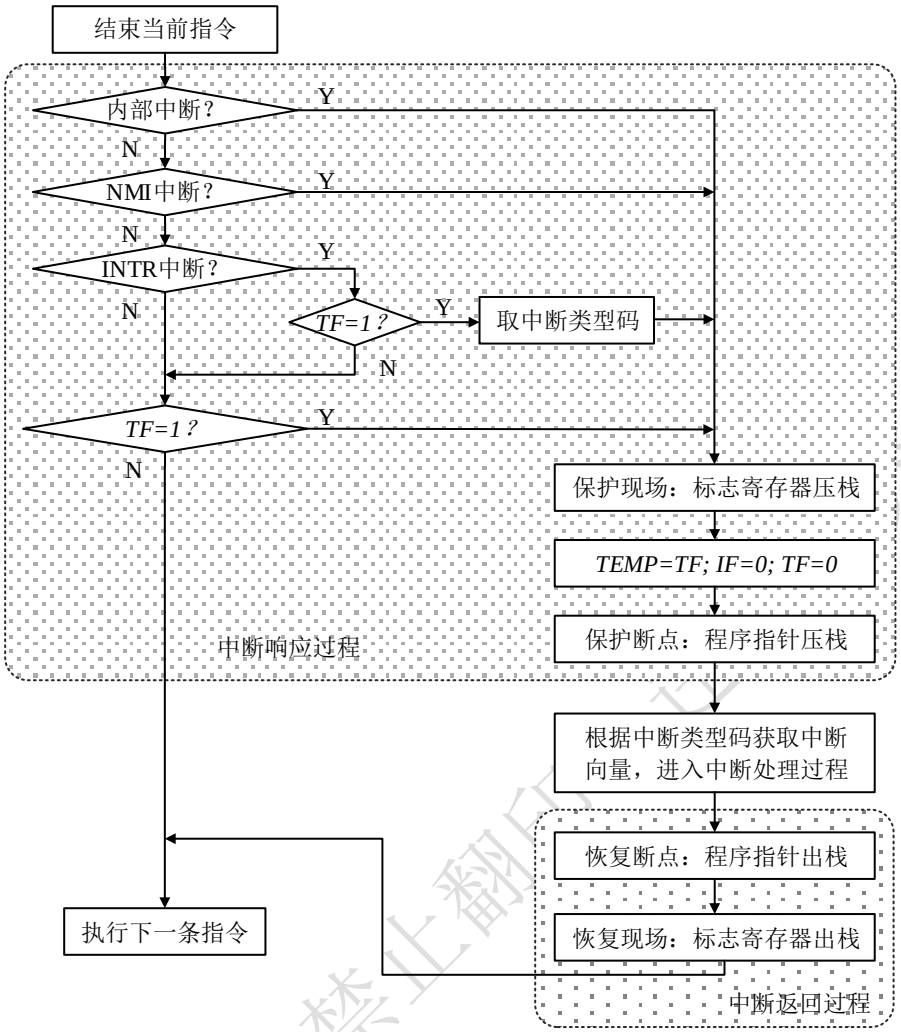


图 4.63　8086 CPU **的中断过程**

当 CPU 根据中断类型码获取到中断向量后，并不是马上转到中断服务子程序的入口，而是先保护现场和断点。为了在中断处理结束后能正确返回和继续执行被中断的程序，系统自动将现场数据（标志寄存器内容）和断点地址压入堆栈保护。随后，中断向量被装入 PC 寄存器，在下一个指令周期即进入中断服务子程序。

(2)中断处理

中断处理过程就是执行中断服务子程序的过程，如图 4.64 中虚线框内所示。中断服务子程序由用户编写，根据中断源所对应外设的功能进行相应的操作，如读取外设数据、向外设输出数据等。

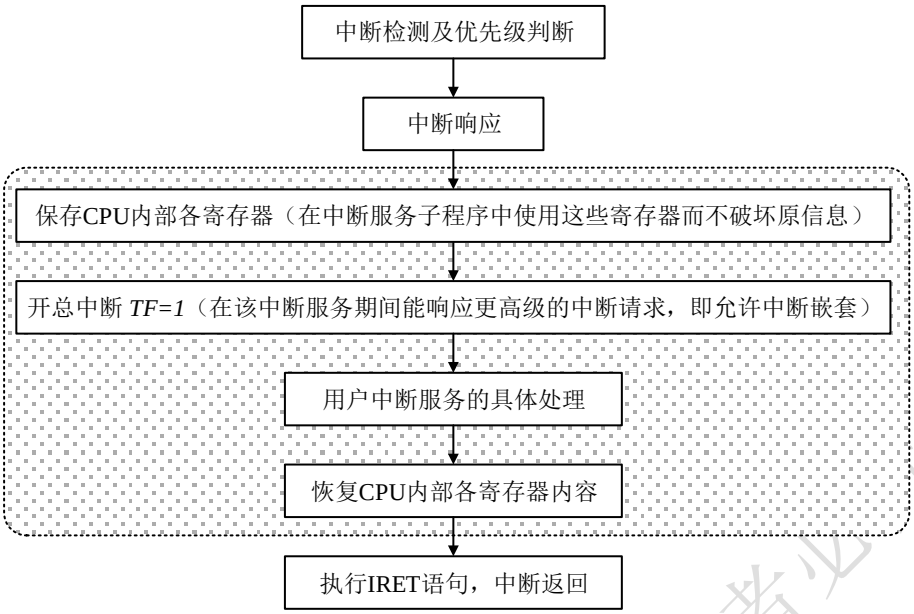


图 4.64 CPU 的中断处理过程

(3)中断返回

任何中断服务子程序的最后一条执行指令都是中断返回指令（图 4.64 中所示为 8086 处理器的 IRET 指令），该指令告诉 CPU 结束中断过程。中断返回过程包括断点恢复和现场恢复，如图 4.63 下部虚线框内所示。该过程也由 CPU 内部硬件电路自动完成的。

3） 中断优先级

对外设优先级的判断一般可以采用软件查询、硬件排序或采用专用芯片管理等方法来实现。

(1)软件查询

软件查询的方法是实现中断优先级别判定的最简单方式。软件查询也需要使用少量硬件电路，如图 4.65 所示，当系统中有多种外部设备，将这些设备的中断请求信号相“或”，从而产生一个总的中断请求信号 INTR 发给 CPU。CPU 读取中断请求寄存器后可获知有哪些中断，之后程序员所选择的处理顺序就体现了优先权。

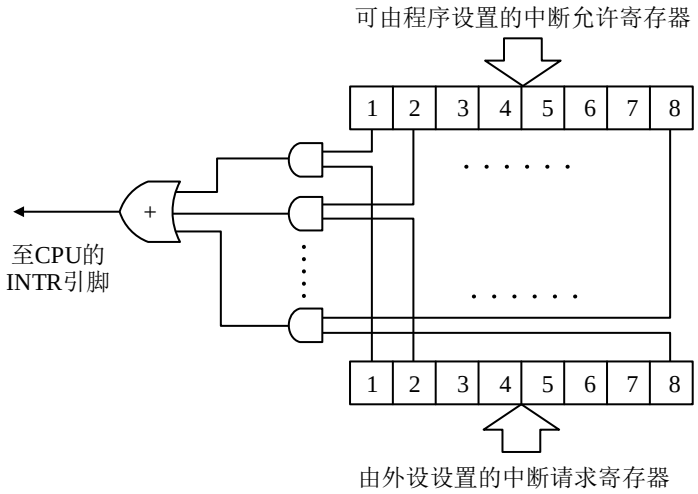


图 4.65　软件查询判优系统的接口电路

软件查询方法的查询流程如图 4.66 所示。由于读到中断状态字节之后，需要再判断应该先为谁服务，故软件查询法的缺点是速度略慢。

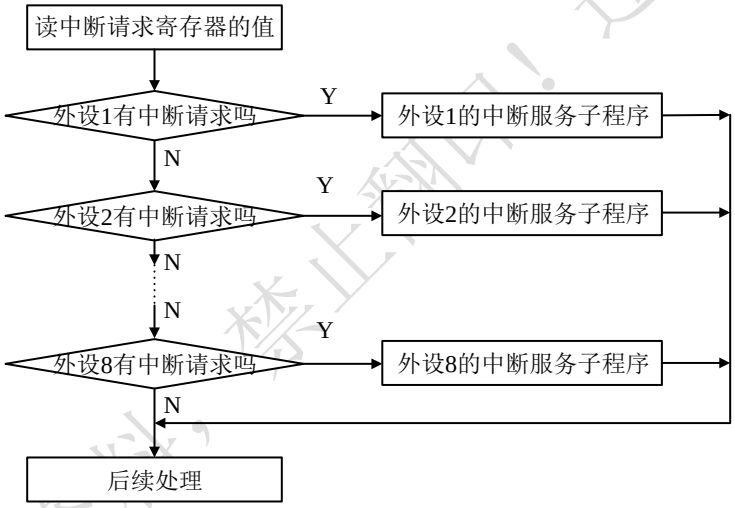


图 4.66　软件查询判优流程

(2)硬件排序

硬件排序判断优先级的实现方法很多，常用的有中断优先权编码电路和菊花链式（daisy-chain）优先权排队电路。中断优先权编码电路如图 4.67 所示，系统中八个中断源均来自外设，任一外设对应的中断请求位和中断允许位同时为“1”时，即可产生中断请求信号。在两种情况下该中断请求信号能传送至 CPU 的 INTR 引脚：比较器输出为“1”时，中断请求信号可通过与门 1 送出；“优先权失效”信号为“1”时，中断请求信号可通过与门 2 送出。

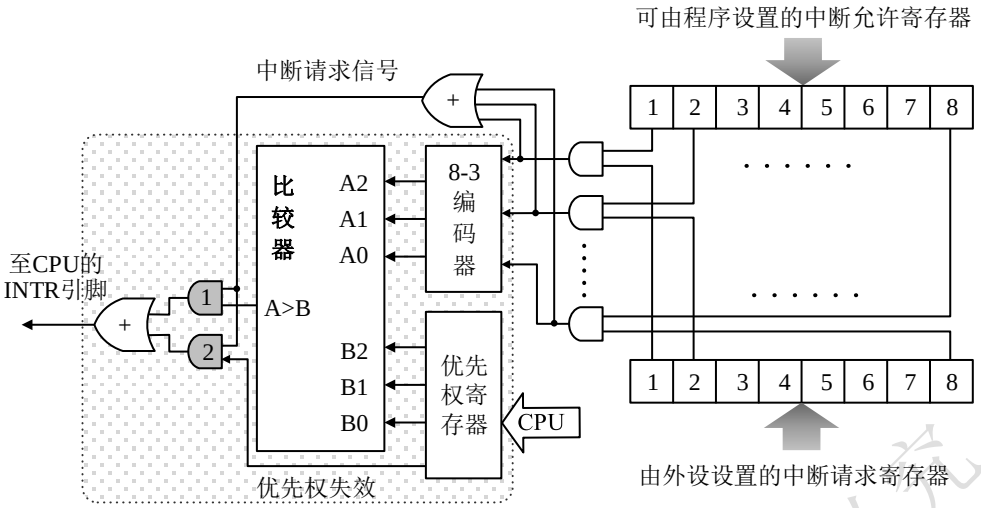


图 4.67　用编码器和比较器实现的中断优先权排队电路

在图 4.67 中，编码器每位输入线的有效电平都对应一个编码，在多位输入线有效时编码器只输出优先权最高的编码；优先权寄存器中记录的是 CPU 当前正在处理的中断的编码；比较器输出为“1”（即 A>B）说明可以产生中断嵌套；“优先权失效”信号说明当前 CPU 没有处理任何中断，不需要再判断中断能否嵌套。

菊花链是 CPU 管理外部中断优先级的另一种简单硬件方法。在每个中断源的接口电路中设置一个逻辑电路，这些逻辑电路组成一个链，叫菊花链，由它来控制中断响应信号的传递通道。如图 4.68 所示，任一个外设的中断请求都可以通过或门直接送到 CPU 的 INTR 引脚，若 CPU 允许处理，则发中断响应信号给外设。菊花链电路被用来确定中断响应信号到底送到哪一个外设。

在图 4.68 中，假设中断请求信号和中断响应信号都是高电平有效的。CPU 送出中断响应信号后，若设备 1 无中断请求，则与门 A1 关门，A2 开门，中断响应信号继续向下一级传送；若设备 2 有中断请求，则与门 B1 开门，中断响应信号送至设备 2，同时 B2 关门，中断响应信号不再向下传送（无论后面的设备是否有中断请求）。

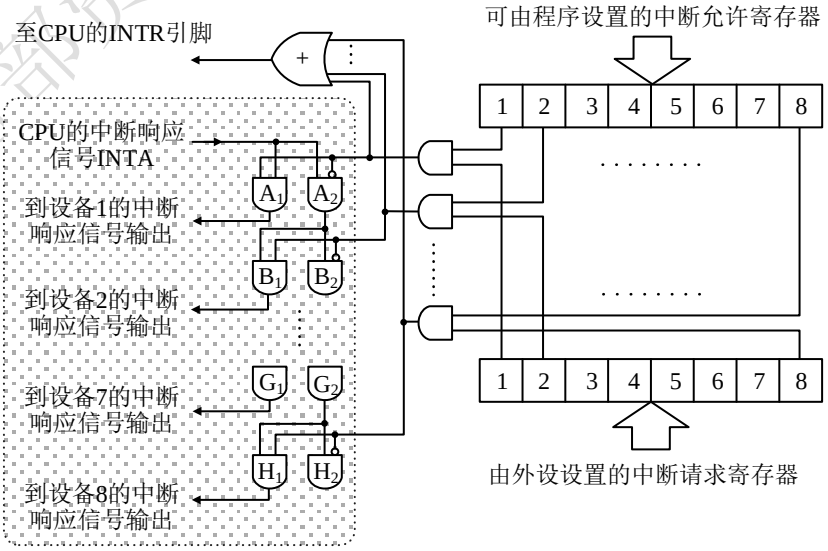


图 4.68　菊花链式优先权排队电路

依据上述分析，设备接入菊花链的顺序就确定了设备的中断优先级别：越靠近链前端（靠近 CPU）的设备优先级越高。

除了上述中断优先权编码电路和菊花链式优先权排队电路之外，中断优先级的判断和处理还可以使用独立的中断控制器芯片。一般来说中断控制器的功能应包括：检测后判断外部送来的中断请求信号是否有效、是否被屏蔽；对多个同时送来的中断请求信号进行优先级判断，确定是否将中断请求信号送给 CPU；在中断请求被响应后负责将中断类型码（或中断向量）送给 CPU；在 CPU 处理中断的过程中持续监测外部中断请求。

4） 中断传输过程

中断传输方式与前述 CPU 主动读取外设状态的方式不同，由外设在需要传输数据时向 CPU 提出请求，CPU 是被动响应外设的。如果 CPU 接受了外设的服务请求，就中断当前执行的主程序，转入外设的服务子程序中，处理完后再返回断点处继续执行。

中断控制方式的接口电路如图 6-22 所示。由图可知，当外设准备好输入数据后发出选通信号，数据进入锁存器中，并同时将中断请求触发器置“1”。此时 CPU 是否能够收到中断请求，取决于中断屏蔽触发器。如果中断屏蔽触发器允许中断（Q1 输出为“1”），与门被打开，因而可以产生中断请求信号 INT；但如果中断屏蔽触发器关闭，则无论外设是否准备好，CPU 都无法收到有效的 INT 申请。在正常情况下，CPU 收到中断请求后，如果内部允许中断，就在当前指令执行完后响应中断。

CPU 进入中断服务子程序后，从输入设备对应的数据端口读取输入数据。中断服务子程序执行完后，CPU 返回断点处继续执行原来的程序。中断控制方式中，没有 CPU 等待外设而浪费的时间，故有利于提高 CPU 的工作效率。有了中断机制后，CPU 与多个外设可以同时工作，也提高了整体系统的效率。

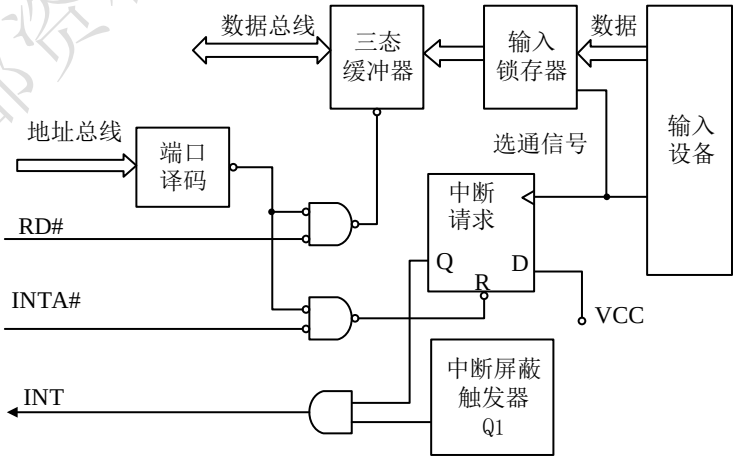


图 4.69　中断控制方式的接口电路

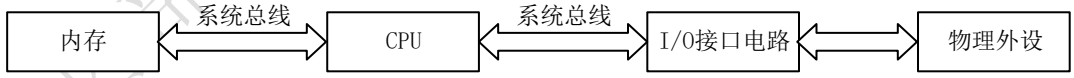
4. 直接存储器访问（DMA）方式

相比查询传送方式，中断传送方式减少了 CPU 查询外设工作状态的时间。在中断传送方式下，CPU 不再轮流查询每个外设进行查询，一方面有利于 CPU 与外设并行工作，另一方面，由于外设具有申请 CPU 的主动权，可以很好地匹配外设对 CPU 提出输入/输出数据传输要求的随机性，具备了实时性。

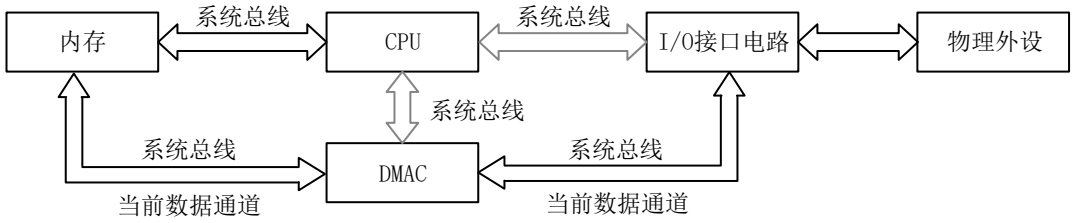
但中断方式仍存在可改进之处。分析中断传输方式的过程，整个中断处理过程中都需要 CPU 参与，即使外设所需传输的数据不需要 CPU 进行处理（如某内存区域数据保存到硬盘）也要经过 CPU。并且，在中断传输方式下，CPU 在两段任务代码间切换的时候，需要进行相应的现场保护和现场恢复（即一些寄存器数据的保存和恢复）操作，这些操作本身对数据传输本身来说没有实际的意义。常把这些仅是为了 CPU 能分别处理多个任务而进行的现场保护和现场恢复任务称之为系统开销。每传送一次数据都产生一次中断申请，进行现场保存、现场恢复等过程。在需要进行大量数据传输并且外设数据传输率比较高的情况下，中断频率也会随之增高，系统开销大增，此时会降低计算机的系统性能。另外，对于流水线处理器结构的计算机，中断产生的程序代码跳转会引起 CPU 指令流水线的清空与重建，这也会影响 CPU 的指令执行效率。

直接存储器访问 DMA 方式的思路为：引入一个专用电路，该电路的功能是与 CPU 协商获得系统总线的控制权，然后在高速外设和存储器之间建立一条可直接传输数据的临时通道。该电路控制并完成数据块在内存与高速外设之间高速地传输，这个专用电路称为 DMAC（Direct Memory Access Controller，DMA 控制器）。DMAC 控制外围设备与主存储器之间传送数据时不需要执行程序，数据传输过程中也不用 CPU 参与，因此，也就不需要 CPU 做现场的保存与恢复工作。

更进一步地，数据在主存中地址的递增/递减，数据块长度寄存器递减等操作也由 DMAC 硬件实现，外设与内存间的数据直接通过总线传送而不经过 CPU 中转，从而使得 DMA 方式比中断等其他方式的数据传送速度大大提高。在 DMA 方式中，只有在 DMAC 初始化时，需要 CPU 对其编程以设置参数，而整个数据的传送过程不需要 CPU 的干预，适合于大数据量的高速传送。图 4.70 对比了 CPU 控制下的 I/O 设备到存储器的数据传送，DMAC 控制下的 I/O 设备到存储器的数据传送。



(a) CPU控制下的I/O数据传输通道



(b) DMA方式的I/O数据传输通道

图 4.70　DMA 方式 I/O 接口电路示意

DMA 方式下，DMAC 和 CPU 以分时方式共享系统总线，CPU 通过暂时让出系统总线控制权达到了提升传输效率的目的。DMA 方式适用于数据块的高速传送，其特点是：①外设可直接访问存储器；②大批量数据可高速访问。DMA 期间，CPU 让出总线控制权，而处于“挂起”的暂停状态。

1） DMA 传送过程

DMA 传送的典型场景包括：从主存储器传送数据到 I/O 端口；从 I/O 端口传送数据到主存储器；存储器到存储器间的传送。DMA 传送系统连接如图 4.71 所示。

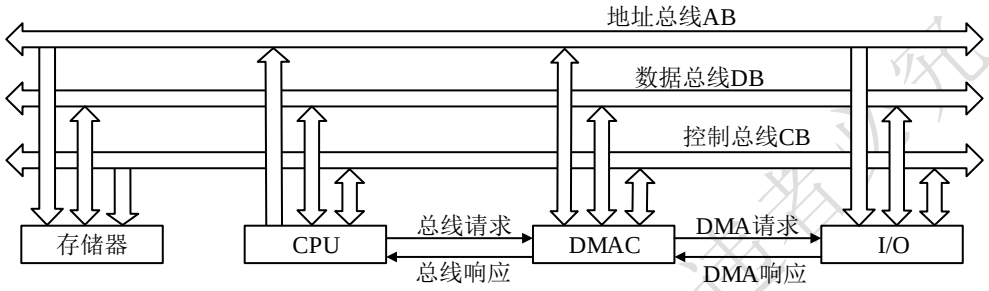


图 4.71 DMA 传送系统示意图

从图 4.71 中可以看出，系统总线分别受到 CPU 和 DMAC 控制，即 CPU 可以向地址总线、数据总线和控制总线发送信息，DMAC 也可以向地址总线、数据总线和控制总线发送信息。但是，同一时刻系统总线只能受一个主控器件控制。当 CPU 控制系统总线时，DMAC 与一般的 I/O 器件一样，是总线上的从属设备，受到 CPU 的控制；而当 DMAC 控制总线时，CPU 必须与总线脱离，即处于高阻态。

DMA 传送的工作过程如下：

① I/O 设备通过端口向 DMAC 发出 DMA 请求，请求传送数据。

② DMAC 收到 I/O 端口的 DMA 请求后，向 CPU 发出总线请求信号。

③ CPU 在执行完当前指令的当前总线周期（不是指令周期）后，向 DMAC 发出总线响应信号。随即 CPU 与系统的控制总线、数据总线和地址总线脱离，即处于高阻态。这时 DMAC 接管地址总线、数据总线和控制总线的控制权。

④ DMAC 向 I/O 端口发出 DMA 应答信号。

⑤ DMAC 把当前 DMA 传送涉及的 RAM 地址送到系统地址总线上；如果进行 I/O 端口到主存储器的传送，则 DMAC 向 I/O 端口发出读命令，向主存储器发出存储器写命令；如果进行主存储器到 I/O 端口的传送，则 DMAC 向主存储器发出存储器读命令，向 I/O 端口发出写命令，从而完成一次总线传送。

⑥ 当预先设定的字节数传送完成后，DMA 传送过程结束。也可由来自外部的终止信号迫使传送结束。当 DMA 传送结束后，DMAC 就将总线请求信号置为无效，并放弃对系统总线的控制，CPU 检测到总线请求信号无效后，也将总线响应信号变为无效，于是 CPU 重新控制三套总线，继续执行被打断的当前指令的其他总线周期。

2） DMA 的传输方式

在 I/O 设备和主存储器之间进行 DMA 传输的时候，既可以每次传输一个数据块，也可以仅传输一个数据，还可以以间歇的方式传输多个数据。通常可区分为如下三种不同的传输方式。

①字节传送（单次传送），其特点是：每完成一次传送，无论任务是否结束，DMAC 都主动放弃总线。如果任务没有完成，DMAC 继续申请总线使用权。单次传送的处理流程如图 4.72 所示。

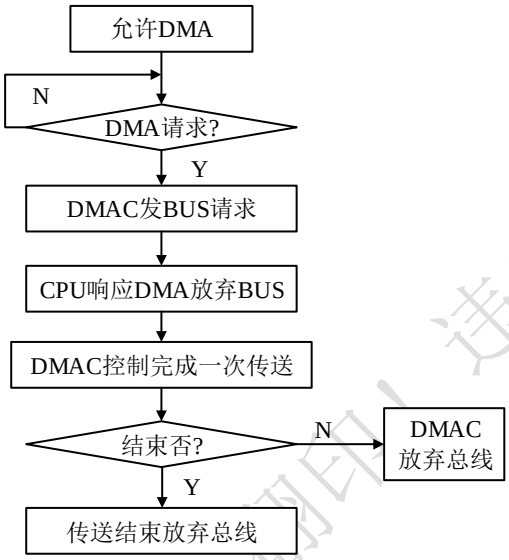


图 4.72 DMA 单次传送处理流程

②数据块传送（成组传送），其特点是：获得总线使用权之后，DMAC 控制总线连续传送，直到传送任务全部完成。但是，如果在传送期间遇到了外部输入的 EOP（End Of Process）信号，传送任务将被强行终止。成组传送的处理流程如图 4.73 所示。

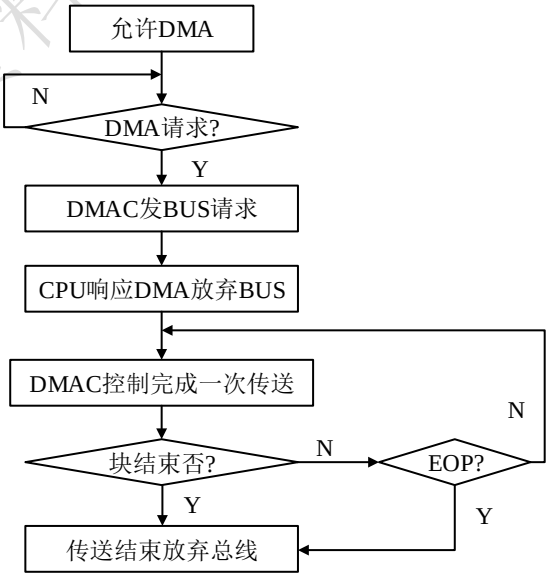


图 4.73 DMA 成组传送处理流程

③询问传送（请求传送），此方式与与成组传送方式的区别在于，传送期间，若 DMA 请求信号无效，DMAC 暂停传送并放弃总线。当 DMA 请求信号重新有效，DMAC 再次申请总线使用权，获准之后从断点处开始续传。请求传送的处理流程如图 4.74 所示。

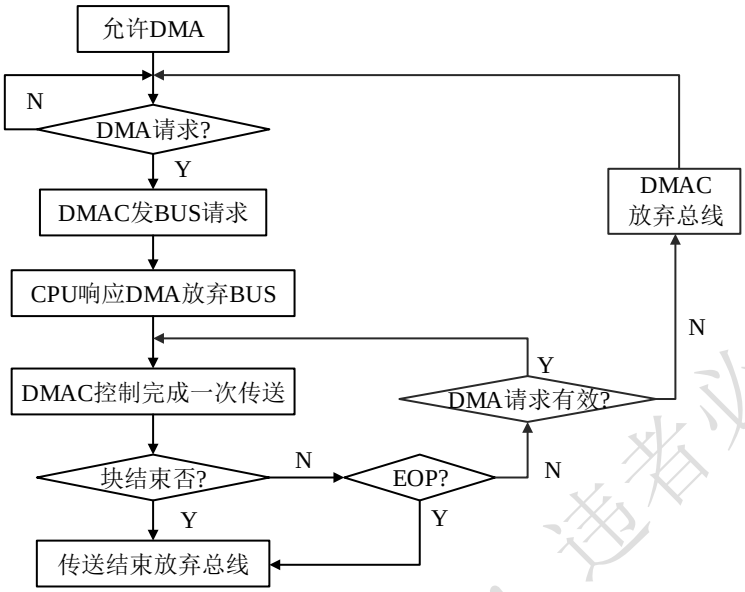


图 4.74　DMA 请求传送处理流程

通常高速的 I/O 设备与主存储器进行数据传输时采用数据块传送方式。在一些资料中，也把数据块传送方式为突发传输（burst transfer）方式。反之，如果是慢速设备，传输完一个数据字后外设还没有准备好下一个数据字，常采用请求传送方式。

DMA 传送方式主要是直接依靠硬件实现数据传送的，它不运行程序，不能处理较复杂的事件。故 DMA 方式并不能完全取代中断方式，当某事件处理不只是单纯的数据传送时，还是需要采用中断方式。事实上，在以 DMA 方式传送完一批数据后，往往利用中断通知通知 CPU 结束处理。

DMAC 本身也是接口芯片，在使用之前需要进行初始化配置。DMAC 有个缺陷，无法对接口进行寻址，当多个外设需要采用 DMA 传送方式时，DMAC 一般会设置多个通道，为多个外设提供 DMA 传送服务。每个通道都有自己的一套寄存器和 DMA 请求响应信号线。多个通道还需要通过优先级电路加以排队，决定先响应哪个通道的 DMA 请求。

4.4.3　并行接口

如果一组数据的各数据位在多条线上同时被传输，这种传输方式称为并行通信。并行通信时数据的各个位同时传送，常以字或字节为单位进行。并行通信速度快，但信号线多，存在线间干扰，故不宜进行远距离通信。串行通信是指使用一条数据线，将数据一位一位地依次传输，每一位数据占据一个固定的时间长度。其只需要少数几条线就可以在系统间交换信息，特别适用于计算机与计算机、计算机与外设之间的远距离通信。串行接口我们将在后续小节讨论。

并行通信传输中有多个数据位，同时在两个设备之间传输。发送设备将这些数据位通过

对应的数据线传送给接收设备，还可以附加一位数据校验位。接收设备可同时接收到这些数据，不需要做任何变换就可直接使用。并行方式主要用于近距离通信。

并行接口，指采用并行通信方式来传输数据的接口标准。从最简单的一个并行数据寄存器，或专用接口集成电路芯片如 8255，乃至较复杂的 IDE 或 SCSI 并行接口，种类有数十种。并行接口的接口传输性能可用两个参数予以刻画：（1）数据通道的宽度，即接口上同时传输的位数；（2）接口时钟频率。数据的宽度可以从 1～128 位或者更宽，最常用的是八位，可通过接口一次传送八个数据位。二十世纪在计算机领域最常用的并行接口是通常所说的 LPT（Line Print Terminal）接口，但目前已淘汰不用。本小节以示例的方式讲解并行接口的工作方式和原理。

1. 无握手信号的并行接口

采用无握手信号接口连接的外设一般是功能简单的电路模块，如按键，数码显示管等。CPU 与这些无握手信号的外设接口传输数据时，总是假定外设总是处于准备好的状态，因此不需要握手信号的并行接口电路框图如图 4.75 所示。

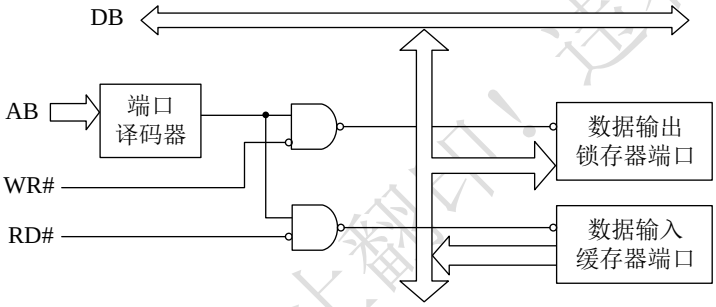


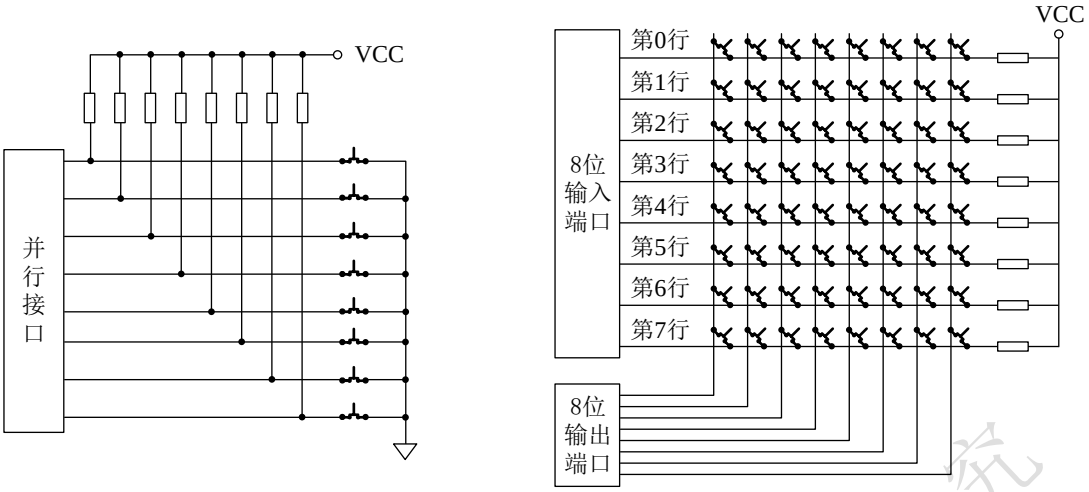
图 4.75　无握手信号并行接口电路示意图

1） 输入接口示例：键盘

键盘是计算机系统中常用的输入设备，通过键盘可实现信息的输入进而控制计算机系统的运行。下面我们以键盘接口为例分析无握手信号的并行接口。

依据按键状态获取方式的不同，可以将键盘分为线性键盘和矩阵键盘，矩阵键盘又可分为非编码键盘和编码键盘。线性键盘的每一个按键需要占用 I/O 端口的一根口线如图 4.76(a)所示。程序通过读取口线的电平状态来判断按键是否被按下。通常，程序需要实现按键状态的判断、按键的软件去抖动和按键状态的定义。

矩阵键盘是将按键按照行、列方式排列起来的矩阵开关，图 4.76(b)所示是 8×8 矩阵键盘，行线与八位并行输入端口相接，而列线与另外八位并行输出端口相接。矩阵键盘比线性键盘节约了更多的口线，换言之，口线数目相同时，矩阵键盘的按键数目比线性键盘的按键数更多。假设口线的数目为 M 和 N，则矩阵键盘的按键数为 M×N，而线性键盘的按键数为 M＋N。



(a) 线性键盘结构　　(b) 矩阵键盘结构

图 4.76　线性键盘与矩阵键盘结构示意图

矩阵键盘需要通过软件对按键状态进行判断和定义。识别矩阵键盘的按键是否处于按下的状态，可以使用行扫描法、行列反转扫描法或状态机法。此处仅介绍原理最常用的行扫描法。

行扫描法的基本过程是首先快速判断是否有键按下，先使输出端口的各位都为低电平的零状态，相当于各列都接地，再从输入端口读取数据，如果读取的数据是 1111 1111B，则说明当前所有行线处于高电平状态，没有键被按下，程序应该在循环中等待。如果并行输入端口读取的数据不是 1111 1111B，则说明必有行线处于低电平，也就是说肯定有键被按下。为了消除键的抖动，经过一定的延迟后，进入下一步，确定到底哪个键被按下。

行扫描法的程序流程如图 4.77 所示。基本思路是：逐列检查是否有按键被按下，发现有按键被按下后再确定行（此即行扫描，对应图 4.77 中阴影区域部分）。

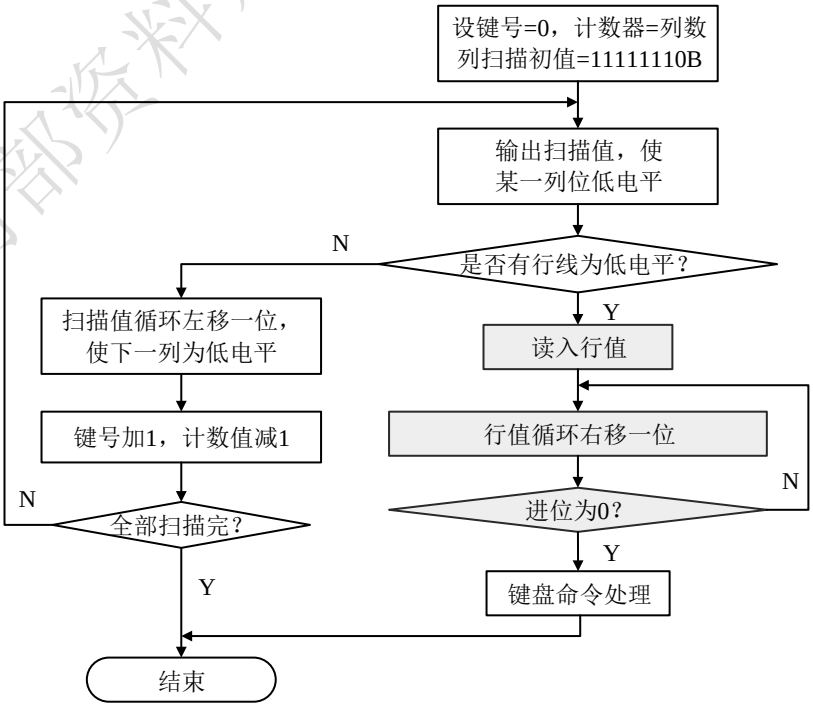


图 4.77　行扫描法的程序流程

程序先将键号寄存器置零，将计数值设为键盘列的数目，然后再设置扫描初值。首先设置扫描初值为 1111 1110B，即使第 0 列为低电平，而其他列为高电平。输出扫描初值后，马上读取行线的值，看是否有行线处于低电平。如果没有表示该列没有按键按下，将扫描初值循环左移一位，变成 1111 1101B，即使第 1 列为低电平，其他列为高电平。这样进入第 1 列的扫描，使键号为 8（第 1 列的第 0 号键），计数值减 1。如此循环，直到计数值为 0 结束。

如果在此过程中，查到有行线为低电平，则将行线数据保留，然后通过循环右移操作后检查进位的方法确定具体键号。具体过程为：行线数据循环右移一位，此时若进位位为 0 则表明第 0 行线的 0 号键闭合；否则继续循环右移，直至找出闭合键为止。由于已经确定了此列上有键闭合，所以一定可以在此列线中查出某行处于低电平。

2） 输出接口示例：LED 显示

数码显示管是一种半导体发光器件，常简称数码管，其基本单元是 LED（Light EmittingDiode，发光二极管）。数码管可分为七段数码管和八段数码管，区别在于八段数码管比七段数码管多一个用于显示 DP（decimal point，小数点）的发光二极管单元。

数码管分为共阳极和共阴极两种结构。共阳极数码管的正极（或阳极）是所有发光二极管的共有正极，其他节点为独立发光二极管的负极（或阴极），使用时把正极接电源，不同的负极受控接地就能显示不同的数字。共阴极数码管与共阳极数码管的接驳方法相反。图 4.78 所示为八段数码管的器件引脚示意图。习惯上数码管中的八个发光二极管命名为“a、b、c、d、e、f、g”，小数点命名为“dp”，公共端命名为“com”。如果是共阳极结构，则“com”接正电源 VCC，若为共阴极结构则“com”接地 GND。

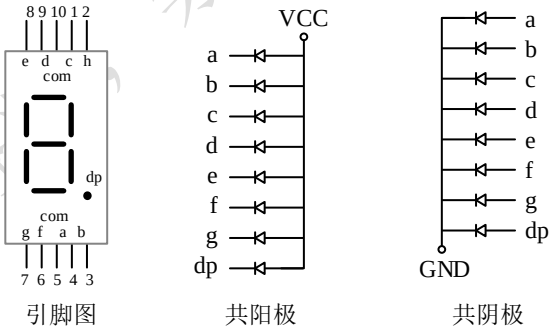


图 4.78　八段数码管器件引脚示意图

以八段数码管为例，显示一个四位二进制数（即十六进制的数字符号或 BCD 码的数字符号）时，需要将四位二进制数转换为表 4.16 所示的八段显示码。可以用专门的码制转换芯片（译码器）完成这种代码转换，也可以用软件的方法编程实现。

表 4.16　八段 LED 数码管段代码编码表（16 进制）

数字	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**黑**
共阳	C0	F9	A4	B0	99	92	82	F8	80	90	FF
共阴	3F	06	5B	4F	66	6D	7D	07	7F	6F	00

实际系统中常需要多个数码管来同时显示多个数字，此时显示接口电路有静态显示和动态显示两种接口方式。静态显示方式是，每个数码管相应的段（发光二极管）恒定地导通或截止，直到下一次更换信息。此时，共阴极结构数码管的公共端直接接地（或共阳极结构数码管的公共端直接接正电源），各个数码管的输入控制端相互独立。若需要显示 N 个数字，那么数码管个数为 N 时，需要的接口口线数目为 8×N。

为了能用更少的信号线控制显示，又出现了另外一种称之为动态显示的方式。动态显示方式的基本思路是让各个数码管轮流显示，每位数码管的点亮时间约 1～2ms，由于人的视觉暂留现象及发光二极管的余辉效应，尽管各个数码管并非同时点亮，但给人的印象是所有数码管同时显示。在动态显示方式下，各个数码管的对应段输入控制端并连在一起，因此无论数码管的个数是多少，需要的口线数目都只需要 8 条，该端口称为段选口。各个数码管的公共端分别连接一根口线，该口线称为位选口。当数码管的个数为 N 时，则需要的位选口口线数目为 N，因此动态显示方式总共需要的口线数为 8＋N。

图 4.79 所示为显示五位十进制数的共阴极 LED 数码管接口电路。八位段选码由一个 I/O 端口控制，在每次输出段码之后，五个数码管同时获得了该段码值。要在每个数码管显示不同的字符，就必须采用扫描方法，轮流点亮各个 LED 数码管。首先，在显示第一个十进制数字的时刻，段选控制端口输出相应字符的显示码；位选控制端口输出的位选码中，对应第一个数码管的位为“0”（低电平，因数码管为共阴极结构），而其他位为“1”（高电平）。

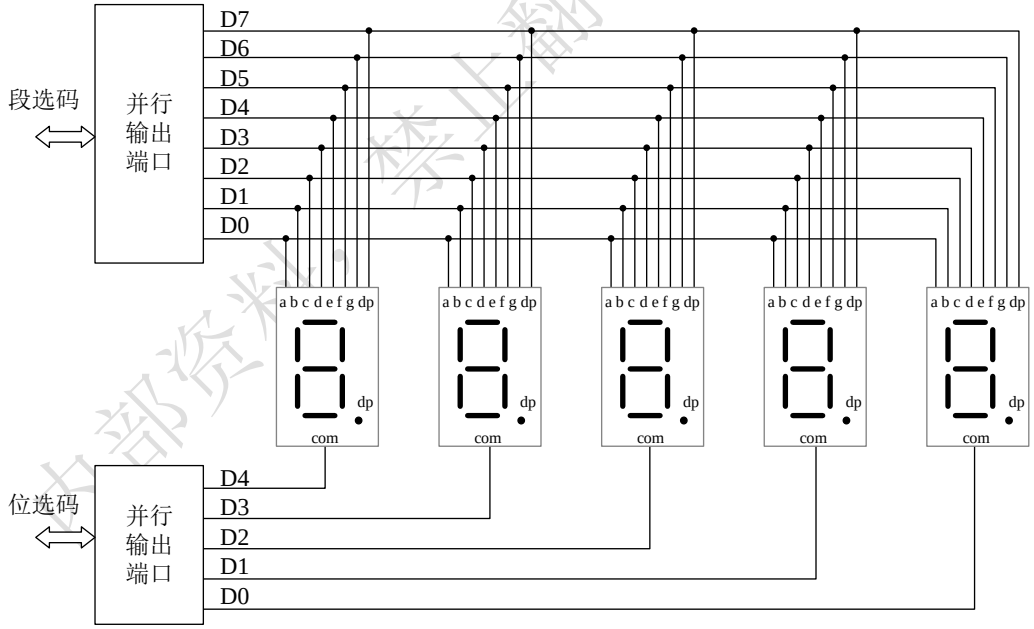


图 4.79　多个数码管的显示接口电路

随后显示下一个数字的时刻，段选控制端口输出另一字符的显示码；位选控制端口输出的位选码中，第二个数码管的位选码为“0”，其他位都为“1”，这样就完成了第二个数码管上的显示。以此类推，将各个显示码依次送到数码管，并将位选码依次循环移位后送出。当扫描速度较快时，由于数码管的余辉和肉眼视觉的暂停特性，就能同时看到各个数码管显示出不同的字符来。

2. 带握手信号的并行接口

为了保证数据传输的可靠性，一些外设需要通过握手联络（handshake）采用查询方式或者中断方式来实现数据交换，如打印机、模数转换模块等。此类带握手信号的接口电路除了数据端口之外，通常还有状态端口和控制端口，典型输入接口电路结构如图 4.80 所示，输出接口电路结构如图 4.81 所示。

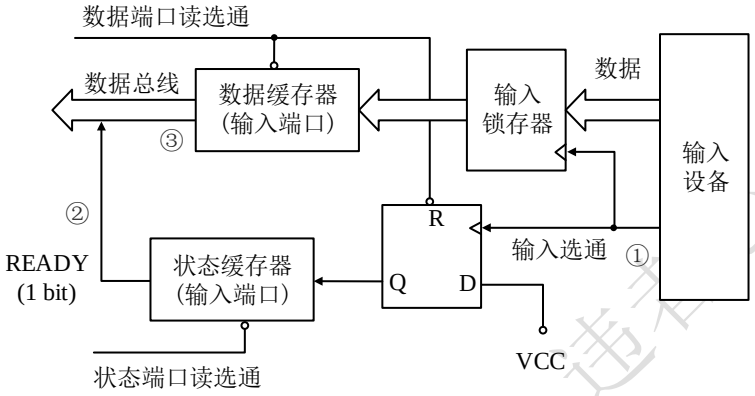


图 4.80　带握手信号的输入接口电路示意图

图 4.80 所示带握手信号的输入接口电路上数据输入的过程和步骤如下。

① 输入设备发出选通信号，一方面将准备好的数据送到接口电路的数据锁存器中，同时使接口电路中的 D 触发器置“1”并将该信号送到状态缓存寄存器中等待 CPU 查询。

② CPU 读取接口中的状态寄存器，检查状态信息以确定外设数据是否已经准备好。

③ 若 READY 为“1”，说明外设已将数据送到数据缓冲寄存器中；CPU 读数据端口，同时数据端口的读信号将接口中的 D 触发器清零，即令 READY 为“0”，完成本次数据传送。

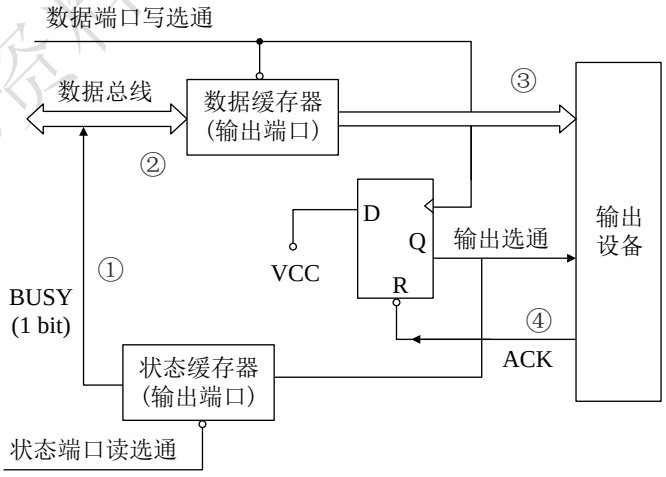


图 4.81　带握手信号的输出接口电路示意图

图 4.81 所示带握手信号的输出接口电路上数据输入的过程和步骤如下。

① CPU 读接口中的状态寄存器，检查状态信息以确定外设是否可以接收数据。

② 若 BUSY 为“0”，说明数据锁存器空，CPU 向数据端口写入需发送的数据，同时数据端口的写信号将接口中的 D 触发器置“1”，即令 BUSY 为“1”，该信号一方面用于通知输出设备数据已准备好，另一方面被送到状态寄存器以备 CPU 查询。

③ 输出设备从接口的数据锁存器中读出数据。

④ 输出设备发出响应信号 ACK 将接口中的 D 触发器清零，即令 BUSY 为“0”，完成本次数据传送。

3. 可编程并行接口（GPIO）

可编程通用并行接口是计算机及其他数字电路系统中最常使用的一种简单外部设备接口电路。其结构简单、应用途广泛，且可以通过编程控制字来实现控制，故得名“通用可编程 I/O 接口”，即 GPIO（General-Purpose IO ports）。用户可以写入控制字改变 GPIO 的工作方式，因而可适应多种应用场合的需求。例如，GPIO 可以仅使用一个引脚来连接只需要开/关两种状态的简单设备，如控制灯的亮灭；也可以使用多个引脚同时输出多个控制信号，如控制 LCD（Liquid Crystal Display）显示屏。

图 4.82 为典型的 GPIO 引脚电路结构图，该引脚可以作为普通的输入/输出信号引脚，也可以作为多功能引脚与其他输出信号复用（图中复用选择位）。通常编程的时候可以设置的信号（寄存器位）包括：中断允许控制位、复用选择位、三态输出使能位、上拉电阻使能位等。

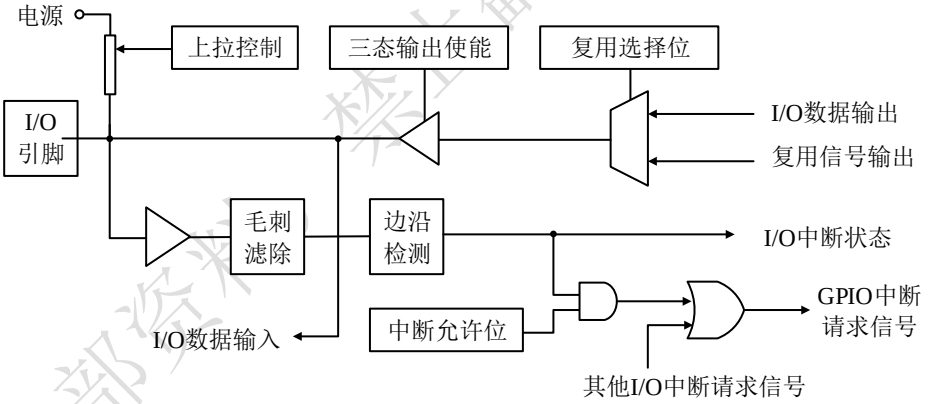


图 4.82 GPIO 典型接口电路示意图

早期的计算机系统中使用单独的芯片来控制这些 I/O 引脚，如英特尔的 8255 芯片是一款非常经典的可编程并行 I/O 接口芯片。8255 有三个八位并行 I/O 口，具有三个通道、三种工作方式。其各个端口的功能可由软件编程控制，使用灵活，通用性强，可作为计算机或单片机与多种外设连接时的中间接口电路。目前（2019 年）该芯片已经很少再使用，但常作为教科书中讲授并行可编程接口的参考范本。图 4.83 为 Intersil 公司近期生产的 8255 系列芯片手册中关于该芯片内部的电路框图。此处对图 4.83 内部结构不再做诠释，后续将以一些常用芯片中 GPIO 接口为例进一步分析。

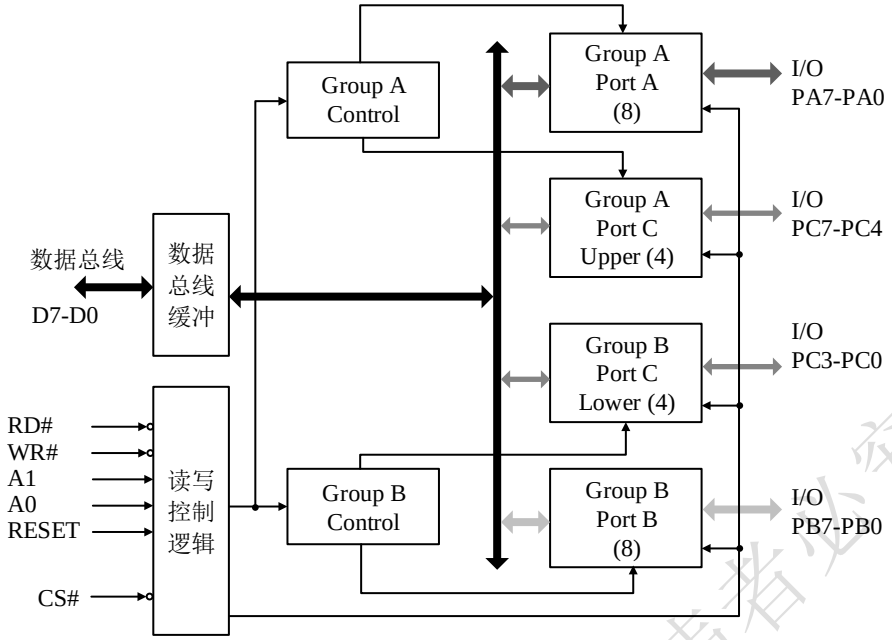


图 4.83　8255 芯片内部电路框图

目前（2019 年）在不同行业广泛使用的微控制器芯片（单片机）往往会集成 GPIO 接口。一般来说，GPIO 接口至少有两个寄存器，即“通用 I/O 控制寄存器”与“通用 I/O 数据寄存器”。数据寄存器的各位都直接引到芯片外部，为双向引脚，其信号传输方向可通过控制寄存器相应位的设置进行配置。图 4.84 所示电路中，对 I/O 引脚的读/写操作被简化为读取输入数据寄存器位/写入输出数据寄存器位；该 I/O 的输入/输出控制则通过写控制寄存器（控制寄存器未在图中画出）来实现。图 4.84 中，I/O 引脚复用了多种功能：普通的数字输入、普通的数字输出、模拟输入、复用的其他输入方式。这些输入/输出方式的选择也是通过写控制寄存器来实现。

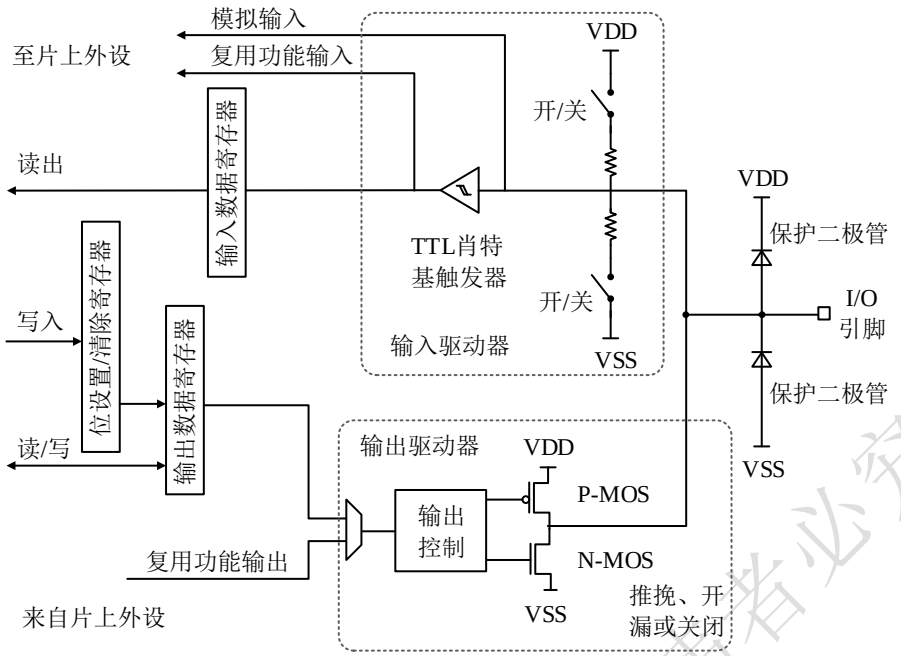


图 4.84　意法半导体 STM32 系列（Cortex-M3）I/O 端口位的基本结构

图 4.84 所示为意法半导体 STM32 系列嵌入式控制器中 I/O 端口中一位的接口电路结构。该芯片的 GPIO 支持浮空输入（IN_FLOATING），带上拉输入（IPU），带下拉输入（IPD），模拟输入（AIN），开漏输出（OUT_OD），推挽输出（OUT_PP），开漏复用输出（AF_OD），推挽复用输出（AF_PP）等八种工作模式，可以满足诸多不同需求的场景，可以参阅意法半导体 STM32 系列的用户手册获取关于不同工作模式的详细介绍。

4.4.4　串行接口

前述并行接口支持的是并行数据传输，使用多条信号线同时传输多位数据。串行数据传输时，数据是一位一位顺序地在通信线上传输的。实现串行数据传输的接口电路称为串行接口。由于 CPU 内部数据总线为并行总线，当数据从 CPU 送出至串行接口前，并行数据需要经并/串转换电路转换成串行方式，再逐位经传输线送达接收端。接收端则需要把数据从串行方式重新转换成并行方式，再推送到内部并行的数据总线上。如果采用相同的接口时钟频率，串行数据传输的速度要比并行传输慢。

串行数据传输的过程往往被称作串行通信。由于使用的信号线少，在远距离数据传输的时候，常优先考虑串行通信方式。如计算机网络中常见的以太网就是以双绞线作为传输介质的典型串行通信系统。本小节首先阐述早期在计算机系统中使用广泛的异步串行通信方式及其接口电路和协议，然后介绍在电路板上芯片间互联较为常见的 I^2C 和 SPI 通信方式及其接口电路和协议。

1.　串行通信概述

串行通信常用于需要远距离传输的情形，然而受限于传输线的特性，数字信号无法在传输线上直接进行远距离传输。一般发送方需要使用调制器（Modulator），把要传送的数字信号调制为适合在线路上传输的信号；接收方则使用解调器（Demodulator），将从线路上接收到的调制信号进行解调，还原成数字信号。调制器和解调器两者通常集成在一起作为一个设

备，称为调制解调器（Modem）。在数据通信系统中，调制解调器被称为 DCE（Data Communication Equipment，数据通信设备）。而计算机或其它数据终端被称为 DTE（Data Terminal Equipment，数据终端设备）。图 4.85 是两台计算机通过电话线和调制解调器实现远程数据通信的示意图。

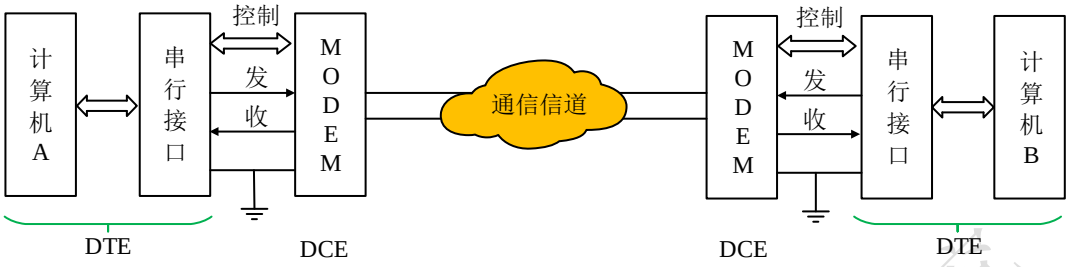


图 4.85　通过 Modem 实现远程数据通信

从数据传送方向来看，串行通信分为单工（Simplex）、半双工（Half Duplex）、全双工（Full Duplex）方式，称为传送制式。单工的制式是指发送方与接收方只有一条数据线路，而且这条数据线路永远只能进行一个方向的数据传送。半双工是指发送方与接收方也只有一条数据线路，但这条数据线路可以在不同时刻进行两个方向的传输。全双工是指发送方与接收方有两条数据线路，同一时刻可以利用这两条数据线路进行不同方向的数据传输。

以半双工为例，仅用单条传输线来传送数据，这条传输线上，有时候是发送方向接收方传送数据，有时候是接收方反向传递应答数据。并且，我们还希望这样的传输是可靠的。这样就需要用合适的方式告知对方传送的有效数据什么时候开始，什么时候结束，以便对方进行相应的操作。在实际系统中，为了实现上述传输线的有效控制，收发双方必须有严格的约定。这些双方共同遵循的共同约定，称为通信协议。

在通信协议中，为确保通信各方能准确的发送、接收、识别、理解和利用信息，需要在信号电平、编码类型、数据格式、差错控制、流量控制、信息封装、连接控制等诸多方面约定规则。为了简化协议设计，使通信系统结构更加清晰，通信协议采用分层模型。单纯的接口电路，无论是并口还是串口，电路功能隶属于 OSI 模型中物理层。但一般通信协议中约定差错控制、流量控制、连接控制方式等则隶属于 OSI 模型的链路层。

1）　串行通信的性能参数

数据传输速率是串行通信最基本的性能参数。衡量数据传输速率有两个单位：①比特率，即单位时间内传送的二进制码元的个数，单位是 bps（bit per second）。由于 1 个二进制码元代表了 1 bit 的信息，因此比特率也称为传信率。②波特率：单位时间内传输的符号个数，单位是波特（Baud 或 Bd）。计算机普遍采用二进制，一个“符号”仅有高、低两种电平，分别代表逻辑值“1”和“0”，所以每个符号的信息量为一位（一比特），此时波特率等于比特率。但在其它一些应用场合，一个“符号”的信息含量可能超过一位，此时波特率小于比特率。

数据通过信号线传输的过程中，会因外部干扰或其他信号线的影响发生传输错误。故而串行通信系统常通过检错码来发现错误，常用的校验方法有奇偶校验、CRC 循环冗余校验等。另外，有些系统在检测到传输错误后，会重传发生错误的数据；还有些系统中引入了更为复杂的纠错码，可以在一定程度上纠正错误。这些措施在通信协议中被称作差错控制。

2） 串行通信的同步方式

同步技术指能够检测和识别所传送的数据单元（位、字符或字节、帧、数据块等）的起止的技术。根据同步方式，串行通信又分同步通信方式（Synchronous）和异步通信方式（Asynchronous）。与之相应的串行总线可分为同步串行总线和异步串行总线。“同步”和“异步”是以数据位之间的时间相关性来分类的。

同步串行接口传输信息的时候，发送方和接收方电路在同一个时钟下工作，因而所传输信息的字节与字节之间、位与位之间均与时钟有严格的时间关系。但是异步串行总线传输信息时，发送方和接收方的电路模块按照各自的时钟工作，故所传输的信息虽然同一字节内的各位之间有相对的时间关系，但字节与字节之间无任何时间关系。图 4.86 所示为同步传输协议下发送串行数据的示意图，图中包含了八个数据位。很多同步协议首先发送 MSB（Most Significant Bit），而很多异步协议首先发送 LSB（Least Significant Bit）。

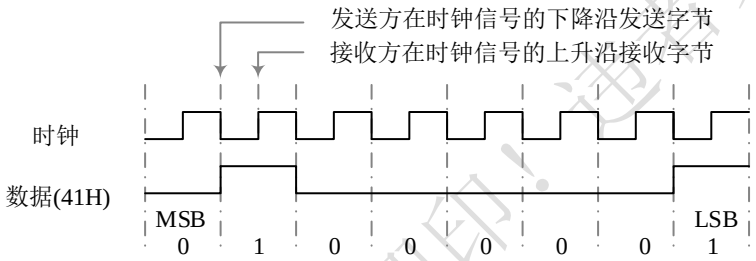


图 4.86 典型的同步传输（很多同步协议首先发送 MSB）

异步串行接口与同步串行接口最大的区别在于是否使用公共的时钟。收发双方使用异步串行接口传输数据时，只使用数据线，而不像同步串行接口那样还需要时钟信号。异步串行接口常用于计算机系统之间的远程数据传输，可以实现远程通信。图 4.87 所示为异步传输协议下发送串行数据的示意图，图中包含了七个数据位、一个起始位和一个停止位，关于起始位和停止位在后续小节中还会深入讨论。

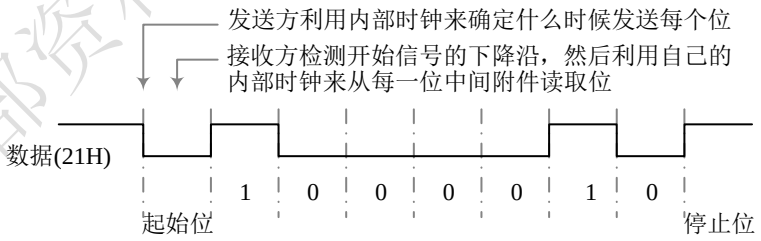


图 4.87 典型的异步传输（异步协议首先发送 LSB）

2. 异步串行接口

一般而言，术语“异步串行接口”包括了两方面的内容：①串行通信双方进行数据传输时的接口电路、②收发双方约定的通信协议。下面首先介绍常用的数据帧格式及相应的波形检测方法，然后阐述典型的接口电路和串行接口协议。

1） 异步串行数据帧格式

异步通信传输数据的过程中，拟传输的数据以字符（一个字符通常含五至八位）为单位进行传送。考虑到传送发生的相对时间是随机的，为了确保整个通信过程的正确性，需要找一种合适的方法，使发送方和接收方在所传送的字符与字符间实现同步。图 4.87 所示为一种常用的方法：在字符数据格式中设置起始位和停止位。图 4.87 中，发送端在一个字符正式发送前，先发送一个起始位（一位的持续时间是本地发送时钟的一个周期），而在该字符发送完成之后，再发送一个（或多个）停止位。接收方一旦检测到起始位，就获知字符已到达，应开始接收字符；而检测到停止位后，就获知字符传送已结束。

按照上述规则，一种可能的串行异步通信格式如图 4.88 所示。每个字符传输包括：一个起始位（低电平，逻辑“0”）、五至八位有效数据位、一位奇偶校验位（也可以没有校验位）、一位（或 1.5 位，或 2 位）停止位，停止位之后是不定长度的空闲位。停止位和空闲位都约定为高电平（逻辑“1”）。

图 4.88 异步串行通信的数据格式

注意，图 4.88 所示字符传输格式规定了不同传输事件发生的先后顺序，这隶属于异步串行接口标准物理层特性中的规程特性。但是图 4.88 所示仅为数据线上的时序特征，在一个实际的异步串行接口中，通常还定义有一些控制信号线，此处并未展开讨论，故而这里只提供了部分规程特性的描述。

2） 异步传输信号波形的检测

异步通信的收发双方采用各自独立的时钟。虽然时钟的频率可以设计为一样，但具体工程实现的时候难以保证收发双方时钟的频率完全相等，且时钟的相位也无法保证对齐。假设约定的波特率为 N（二进制情形下即时钟频率为 N），为了成功检测出信号的波形，接收方采用频率为 M 的时钟对接收信号进行检测，而 $M=K\times N$，K 称为波特率因子，$K\geq 1$。有些芯片 K 可以编程设置，有些则固定，如 K=16。

下面以起始位的检测为例来说明接收端检测波形的过程。如图 4.89 所示，假设 K=16，当接收端检测到一个低电平时，还必须进一步证实是否的确是起始位。不同的接口芯片可能采用不同的方法，例如：隔八个再检测一次，若仍然为低电平则确认是起始位。或者连续检测八个，若有五次以上是低电平则确认是起始位而不是干扰。

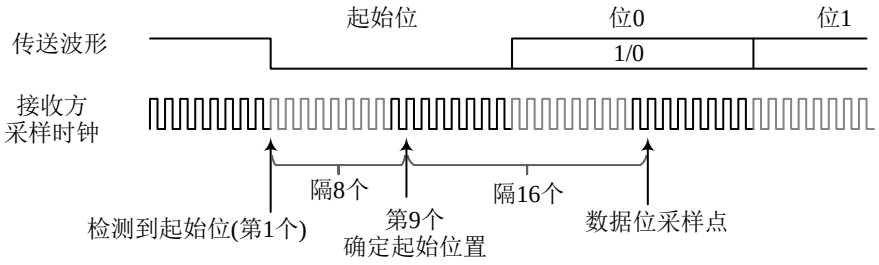


图 4.89　16 倍波特率时钟采样

若确认是起始位，则从第九个检测脉冲（起始位中间位置）开始，接收端每隔 16 个脉冲采样一次输入信号，顺序接收各个数据位。

3） 异步串行接口电路

为了基于图 4.88 所示的数据格式进行可靠的串行通信，早期的计算机系统中使用 UART（Universal Asynchronous Receiver/Transmitter，通用异步收发传输器）来规范接口电路。UART 可表现为独立的模块化芯片（如 Maxim 公司的 MAX232），或作为集成于微处理器中的周边电路。现在的计算机以及各类嵌入式系统往往在 UART 上追加同步方式，被称为 USART（Universal Synchronous Asynchronous Receiver Transmitter）。这里我们不针对具体芯片进行讨论，仅分析异步串行接口电路的一般性功能，即讨论异步串行接口标准中功能特性相关部分。

一般而言，异步串行接口电路需要完成的基本功能包括：①数据的串/并、并/串转换。②串行数据的格式化（如自动加入起始位、校验位或同步字符等）。③校验码的生成。④接口间控制信号的解析。⑤调制与解调。

典型串行通信接口电路组成如图 4.90 所示。一般包括发送模块、接收模块、数据缓存、控制寄存器、状态寄存模块、调制解调、中断控制和波特率发生器等。这个电路模块的功能组合在一起，可以实现图 4.88 所示帧格式的数据帧产生和接收。图 4.90 中 RXD 对应接收信号、TXD 对应发送信号，DTR、DSR、RTS、CTS、DCD、BELL 等控制信号含义参见表 4.17。

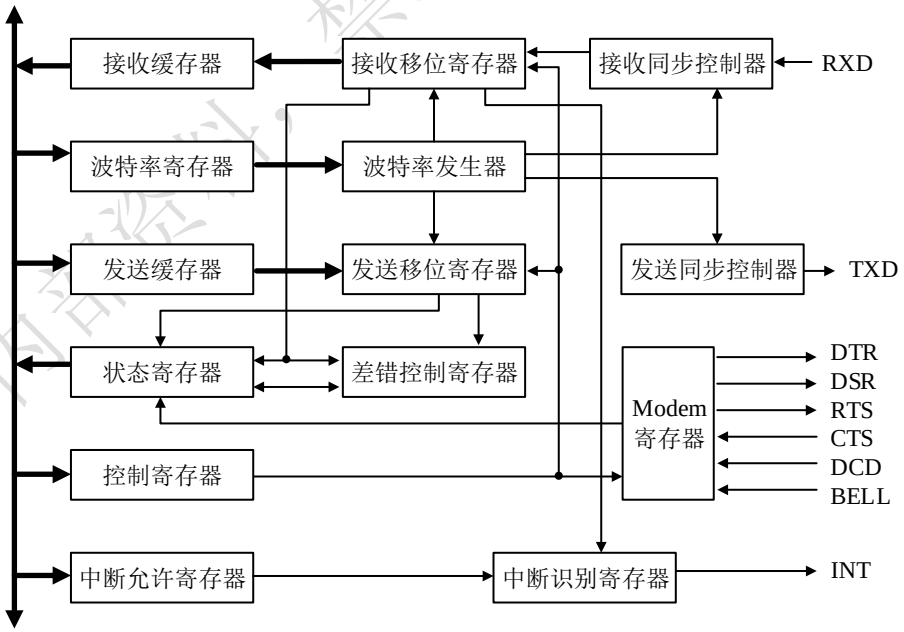


图 4.90　异步串行接口框图

❒ 发送过程包括：从数据总线上接收来自 CPU 的并行（8bits）数据。基于移位寄存器进行并/串转换，插入起始位、停止位等操作以实现如图 4.88 所示的帧结构。按照用户设

置的波特率由 TXD 引脚发送。

- 接收的过程包括：接收来自 RXD 引脚输入的数据，在本地时钟和波特率发生器的控制下，经同步控制器送给移位寄存器，经过串/并转换后进入接收缓存器。与此同时进行错误检测、帧解析并把相关信息写入状态寄存器，然后通过中断告知 CPU 接收完毕，进而 CPU 可读取状态寄存器获知传输状态、通过接收缓存器读取数据。
- 波特率发生器是异步串行接口中一个重要模块。通常利用本地电路时钟（注：图 4.90 中并未画出时钟信号）按照波特率设置值进行合适的分频、锁相和同步产生数据接收和移位所需的时钟。
- 控制寄存器可由用户设置，其参数可定义帧结构、停止位、校验位等。这些参数用于控制发送过程和接收过程。

对于用户而言，了解上述接口电路的基本工作过程是非常有必要的。软件编程时，在使用串口前需要先设置好波特率、数据位数目、校验位数目、停止位数目，以及是否进行流控制等。所谓的流控制指利用 RTS/CTS 或者 DTR/DSR 进行收发双发传输节奏的控制。如果收发双方仅使用 TXD/RXD，是可以进行数据收发的，但是不能进行流控制。若要使能流控，则应该连接双方 RTS/CTS/DTR/DSR 等信号（具体信号含义参见表 4.17）。

4） 异步串行接口协议

接口协议（标准）是通信的收发双方共同遵循的传输数据帧结构、传输速率、检错与纠错、数据控制信息类型等相关约定。以下仅讨论异步串行通信协议，如 RS-232、RS-422、RS423 和 RS-485 等。在任何一个串行通信协议中，都会对接口物理层的机械特性、电气特性、规程特性和功能特性进行规范。RS-232、RS-422、RS423 和 RS-485 等标准中主要的差异在电气特性。考虑到前述关于“异步串行数据帧格式”和“异步串行接口电路”已经部分讨论了规程特性和功能特性，此处我们把重点放在电气特性和机械特性方面。

- RS-232C 标准

RS-232 是由 EIA（Electronic Industry Association，美国电子工业协会）1970 年制定的串行二进制数据交换接口技术标准。先后出现了多个版本，其中应用最广的是修订版 C，即 RS-232C。RS-232C 标准最初是为远程通信连接 DTE 与 DCE 而制定的。虽然该标准并未考虑计算机系统的要求，但后来它被广泛应用于计算机系统。如图 4.85 所示，在计算机系统中，主机端和接口电路对应 DTE；通过 MODEM 连接其他设备的时候，MODEM 对应 DCE。

RS-232 标准中所提到的“发送”和“接收”，都是从 DTE 角度而言。标准规定了 22 条控制信号线，用 25 芯 DB（Distribution Board）插座连接，主机端为插头，而电缆端为插座。所有信号线可分为主信道组、辅信道组这两组，大多数微机通信仅使用主信道组，且并非所有主信道组的信号都要连接。简化后只有八条信号，因而很多系统采用如表 4.17 所示的九芯的 DB9 插座。

表 4.17 RS232C 信号定义

9 针串口（DB9）	25 针串口（DB25）

针号	功能说明	缩写	针号	功能说明	缩写
1	数据载波检测	DCD	8	数据载波检测	DCD
2	接收数据	RXD	3	接收数据	RXD
3	发送数据	TXD	2	发送数据	TXD
4	数据终端就绪	DTR	20	数据终端就绪	DTR
5	信号地	GND	7	信号地	GND
6	数据设置就绪	DSR	6	数据设置就绪	DSR
7	请求发送	RTS	4	请求发送	RTS
8	清除发送	CTS	5	清除发送	CTS
9	振铃指示	BELL	22	振铃指示	BELL

RS-232C 传输电缆的长度与负载的电容值有关。标准规定被驱动电路（终端）的电容，包括电缆的等效电容必须小于 2500pF。对于多芯电缆，每英尺（0.305 米）电容为 40~50pF，所以满足电容特性要求的电缆长度最长为 50 英尺（约 15 米）。

RS232C 采用负逻辑，它的电平与 TTL 电平不同。RS-232C 要求逻辑“0”对应于电平+3~+15V，逻辑“1”对应于电平-15~-3V。因此，计算机与外设的数据通信必须经过相应的电平转换，能完成这种电平转换的芯片有很多，常见的如 MAX232 等。

❒ RS-422/423 标准

RS-232 接口是一种基于单端非对称电路的接口，即一条信号线与一条地线，这种结构容易受到干扰，传输距离受限。为此，EIA 又制定了 RS-422 和 RS-423 等标准。

RS-422 标准采用了平衡差分传输技术，即每路信号都使用一对以地为参考的正负信号线。这种平衡差分结构的抗噪声能力较强，传输速率与距离都明显提高。最高传输速率为 10Mbps，最远传输距离为 4000 英尺（1218 米）。

RS422 标准有点对点全双工与广播两种通信方式。广播方式下只允许一个发送驱动器工作，而接收器可以多达十个。RS-423 则是 RS-232 与 RS422 之间的一个过渡标准，因而兼有两者的特点，此处不再赘述。

❒ RS-485 标准

RS-485 是 RS-422 标准的改进增强版本，该标准兼容 RS-422，提供了多点通信能力，很好地支持了联网功能，在仪器仪表和工业控制等领域得到了广泛的应用。

表 4.18 给出了 RS-232、RS-422 和 RS-485 的电气特性对比。从某种角度说，RS-485 提供的多点通信能力的支持，已经把 RS-232 定义的异步串行接口改造为了一种总线。多个主机系统可以同时挂接到线缆上，RS-485 实际上提供了“总线仲裁”的能力。

表 4.18 RS232/RS485/RS422 **特点对比**

	RS232	**RS422**	**RS485**
线缆连接方式 **Cabling**	单点 single ended	单点/多点 single ended/multi-drop	多点 multi-drop
最大设备数目 **Number of Devices**	1 发+1 收 1 transmit 1 receive	5 发+10 收 5 transmitters 10 receivers	32 发+32 收 32 transmitters 32 receivers
双工方式	全双工	全双工/半双工	半双工

	RS232	RS422	RS485
Communication Mode	full duplex	full duplex/half duplex	half duplex
最大传输距离 **Max. Distance**	50 英尺 50 feet @ 19.2 Kbps	4000 英尺 4000 feet @ 100 Kbps	4000 英尺 4000 feet @ 100 Kbps
最高数据速率 **Max. Data Rate**	50 英尺时 19.2 Kbps 19.2 Kbps for 50 feet	50 英尺时 10 Mpbs 10 Mpbs for 50 feet	50 英尺时 10Mbps 10 Mpbs for 50 feet
信号驱动方式 **Signaling**	非平衡方式 unbalanced	平衡差分传输 balanced	平衡差分传输 balanced
逻辑“1” **Mark (data 1)**	电平范围-15~-5V -5V min. -15V max.	电平范围-6~-2V 2V min. (B>A) 6V max. (B>A)	电平范围-5~-1.5V 1.5V min. (B>A) 5V max. (B>A)
逻辑“0” **Space (data 0)**	电平范围 5~15V 5V min. 15V max.	电平范围 2~6V 2V min. (A>B) 6V max. (A>B)	电平范围 1.5~5V 1.5V min. (A>B) 5V max. (A>B)
最大输出电流 **Output Current**	500 mA	150 mA	250 mA

3. I²C 接口及总线

I²C（Inter Integrated Circuit）总线是一种广泛使用的总线标准，可以用来实现处理器与外设之间、或者不同外设模块之间的串行接口。这些外设可以是串行 EEPROM、显示驱动器、A/D 转换器、LED 灯、LCD 显示器等。I²C 也常用于同一块电路板上两块芯片之间的连接，如图 4.91 所示为微芯（Microchip）公司微控制器芯片 dsPIC30F 与存储芯片 24LC256 通过 I²C 实现连接的电路示意图。

I²C 总线是 Philips 公司 1982 年开发的一种简单、双向二线制同步串行总线。它只需要两根线即可实现多个器件之间传送信息，这两根信号线分别是 SCL（时钟信号线）与 SDA（数据线）。因为 SDA 和 SCL 是双向的，驱动 SDA 和 SCL 的器件其输出级必须漏极开路（OD 门），以执行总线的“线与”功能。另外使用了外部上拉电阻，以确保当没有器件将信号拉低时能保持高电平。

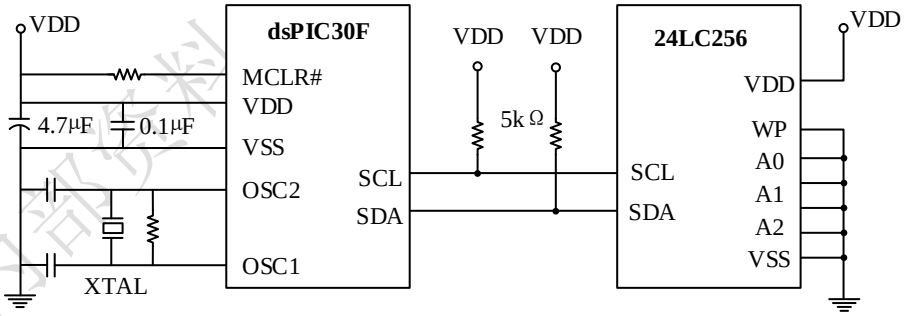


图 4.91 通过 I²C 连接两块不同的芯片

通常 I²C 总线接口被集成在芯片内部，使用时只需要连接两根信号线。片上接口电路往往还集成了滤波器，可以滤去总线数据上的毛刺。因此使用 I²C 总线有利于简化硬件电路 PCB（Printed Circuit Board，印刷电路板）布线，降低系统成本。I²C 芯片需要的信号线少，使用 I²C 作为接口的电路模块，可以很容易形成标准化和模块化，便于重复利用。

1） I²C 接口典型电路

不同厂家实现的 I²C 的接口电路一般会包括数据寄存器、控制寄存器、状态寄存器、地

址寄存器，有些还包括时钟控制有关的分频寄存器。下面我们以意法半导体的 STM32 系列微控制器中集成的 I²C 的接口电路为例进行分析，其接口电路如图 4.92 所示。

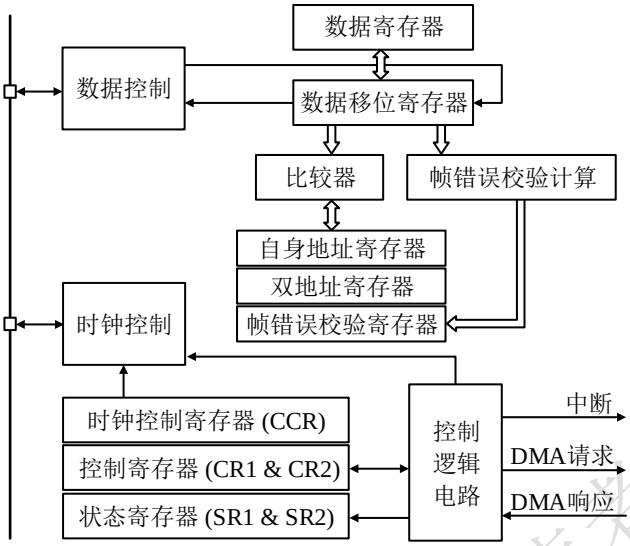


图 4.92　意法半导体 STM32 系列处理器的 I²C 功能示意图

发送数据时，CPU 通过内部总线将需要发送的数据写入数据寄存器。由于 I²C 是串行通信，故数据寄存器的并行数据需要经过移位寄存器转为串行数据。图 4.92 中有两个地址寄存器（地址的定义可参阅下一小节），自身地址寄存器保存的是本 I²C 设备的地址，而双地址寄存器是该系列芯片独特的设计，使芯片可保存额外的一个地址，从而可响应两个从地址，大部分厂家的 I²C 接口电路中仅有一个地址寄存器。接收数据时，比较器用于比较从 SDA 接收到的地址是否和本设备的地址一致。对接收到的数据还需要检查是否发生错误，由帧错误校验计算电路提供校验功能，此功能也并非所有厂家实现 I²C 接口电路都有。

整个接口电路的控制模块则包括时钟控制寄存器、控制寄存器和状态寄存器等，可以实现对 I²C 接口电路状态的调整和监控。并且，该芯片还支持中断方式和 DMA 方式的数据访问，这两种模式的差异可参阅 4.4.2 小节。

2） I²C 仲裁机制

I²C 采用的是主从式通信方式。为便于描述，I²C 定义了“主器件”和“从器件”的概念。主器件（主机，或称为主设备）用于启动总线传送数据，并产生时钟，此时任何被寻址的器件均被认为是从器件（从机，或称为从设备）。在不至于混淆的情况下，本节内容不对主器件、从器件，主设备、从设备，主机、从机作区分。

从如果主器件要发送数据给从器件，则主器件首先寻址从器件，然后主动发送数据至从器件，最后由主器件终止数据传送。反之，如果主器件要接收从器件的数据，首先由主器件寻址从器件。然后主器件接收从器件发送的数据，最后由主器件终止接收过程。在这种情况下，主器件负责产生定时时钟和终止数据传送。

I²C 总线是一个多主机总线，如图 4.93 所示，在两根信号线上，可以同时挂接多个器件（单片机、存储器、键盘、LED、时钟模块、ADC 等）。如果两个或多个主机同时初始化数

据传输，可以通过冲突检测和仲裁防止数据破坏。每个连接到总线上的器件都有唯一的地址，任何器件既可以作为主机也可以作为从机，但同一时刻只允许有一个主机。数据传输和地址设定由软件设定，非常灵活。总线上的器件增加和删除不影响其他器件正常工作。

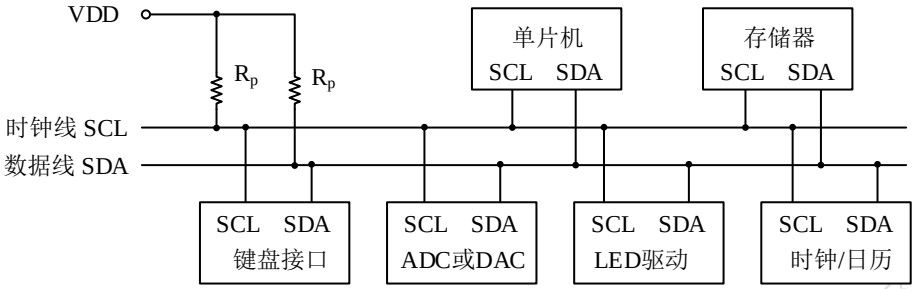


图 4.93 连接了多个 I²C 器件的电路示意图

I²C 总线是一个多主机总线，通过仲裁机制予以协调。I²C 总线对发生在 SDA 信号线上的总线竞争进行仲裁。其基本原理为：在检测到总线空闲（SCL 和 SDA 均为高电平）后，拟使用总线的主机向 SDA 信号线发送数据（高电平或低电平），每一位数据发送后随即检测 SDA 信号线电平是否与自身发送电平一致，若电平不符则竞争失败，自动关闭其输出。

每个 I²C 设备都有一个地址，I²C 总线标准中地址有七位和十位两种定义。I²C 总线标准对同类型的器件规定了一个固有的地址和可编程的地址，采用软件寻址方式，实现对每一个器件的访问。部分器件的地址定义如表 4.19 所示。

表 4.19 典型器件的 I²C 地址

器件型号	器件类型	地址格式
2402	EEPROM	1010XXXB
SAA1064	LED 驱动器	0111XXXB
LM75A	数字温度传感器	1001XXXB
PCF8591	模数转换器	1001XXXB
MAX518	数模转换器	10110XXB

3） I²C 总线协议定义的状态

I²C 总线协议规定：（1）只有在总线不忙时才可以启动数据传输。（2）在数据传输时，只要 SCL 时钟线为高电平，数据线就必须保持稳定。当 SCL 时钟线为高电平时数据线发生变化，将被解释为“启动”或“停止”条件。与此对应，定义的总线条件（状态）如图 4.94 所示。

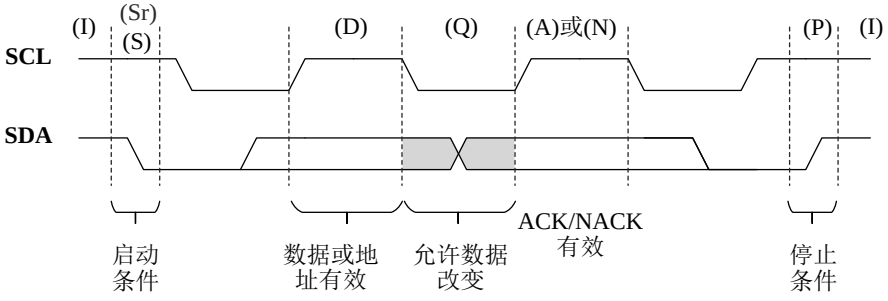


图 4.94 I²C 协议定义的状态

以下分别解释图 4.94 所示的各个状态。

❒ 总线空闲（I）

I²C 总线的 SDA 和 SCL 两条信号线同时处于高电平时，规定为总线的空闲状态，如图 4.94 中状态 I 所示。此时各个器件的输出级场效应管均处于截止状态，即释放总线，由两条信号线各自的上拉电阻把电平拉高。

❒ 启动数据传输（S）

当时钟线 SCL 为高电平时，数据线 SDA 由高电平变为低电平（即负跳变）会产生“启动”条件，如图 4.94 中状态 S 所示。所有数据传输前必须有一个“启动”条件。启动信号是一种电平跳变时序信号，而不是一个电平信号，它标志着一次数据传输的开始。启动信号是由主机主动建立的，在建立该信号之前 I²C 总线必须处于空闲状态。

❒ 停止数据传输（P）

当时钟线 SCL 处于高电平时，数据线 SDA 由低电平变为高电平（即正跳变）会产生“停止”条件，如图 4.94 中状态 P 所示。所有的数据传输必须以“停止”条件结束。停止信号也是一种电平跳变时序信号，而不是一个电平信号，，它标志着一次数据传输的终止。停止信号也是由主机主动建立的，建立该信号之后，I²C 总线将返回空闲状态。

❒ 重新启动（Sr）

在“等待”状态后，当时钟线 SCL 为高电平时，数据线 SDA 由高电平变为低电平会产生“重新启动”条件。如图 4.94 中状态 Sr 所示。重复启动可以让主机在不失去总线控制的情况下改变总线方向。在主机控制总线完成了一次数据通信（发送或接收）之后，如果想继续占用总线再进行一次数据通信（发送或接收），而又不释放总线，就需要利用重启动 Sr 信号时序。重启动信号 Sr 既作为前一次数据传输的结束，又作为后一次数据传输的开始。利用重启动信号的优点是，在前后两次通信之间主机不需要释放总线，这样就不会丢失总线的控制权，即不让其他主机抢占总线。

❒ 数据有效（D）

在启动条件之后，如果数据线 SDA 在时钟信号的高电平期间保持稳定，则数据线 SDA 的状态代表有效数据。数据位的传输过程：在 I²C 总线上传送的每一位数据都和一个时钟脉冲相对应，即在 SCL 串行时钟的配合下，在 SDA 上逐位地串行传送每一位数据。进行数据传送时，在 SCL 呈现高电平期间，SDA 上的电平必须保持稳定，低电平为数据“0”，高电平为数据“1”。如图 4.94 中状态 D 所示。

❒ 等待/数据无效（Q）

在时钟线 SCL 的低电平期间，必须修改线上数据。通过将时钟线 SCL 拉低，器件可以延长时钟低电平时间，导致总线的“等待”状态。只有在 SCL 为低电平期间，才允许 SDA 上的电平改变状态。如图 4.94 中状态 Q 所示。

❒ 应答（A）或不应答（N）

所有的数据字节传输必须由接收方应答（ACK）或不应答（NACK）。接收方会将数据线 SDA 拉低发出 ACK 或释放 SDA 发出 NACK。应答或不应答信号使用一个 SCL 时钟周期。I²C 总线上的所有数据都以八位（一个字节）为单位传送，发送方每发送一个字节，就

在随后的时钟周期释放数据线，由接收方反馈一个应答信号。应答信号为低电平时，规定为有效应答位（ACK），表示接收方已经成功地接收了该字节；应答信号为高电平时，规定为非应答位（NACK），表示接收方没有成功接收该字节。如果接收方是主机，则在它收到最后一个字节后，发送一个 NACK 信号，以通知从机结束数据发送，并释放 SDA 线，以便主控接收器发送一个停止信号。

4） I^2C 总线数据传输过程

I^2C 总线上主设备向从设备发送信息时，写时序步骤如下。这些步骤分别对应图 4.95 中各个阶段。

① 主设备发送开始信号，对应图 4.95 中 S 状态；

② 主设备发送七位的从设备地址，对应图 4.95 中发送地址阶段；

③ 主设备发送写命令（低电平），对应图 4.95 中 R/W#；

④ 接着从设备应答，对应图 4.95 中 ACK，ACK 表示有这个设备；

⑤ 随后主设备发送八位数据；

⑥ 接着从设备应答，ACK 为应答成功；

⑦ 如果从设备应答成功则主设备继续发送数据；

⑧ 若应答不成功，NACK，则主设备发送停止信号，对应图 4.95 中 P 状态。

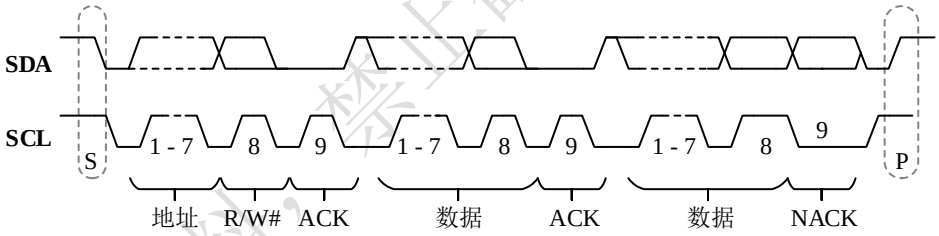


图 4.95 I^2C 总线上的数据传输过程的时序

与之相对，主设备读取从设备信息时，首先要知道将要所读取设备的地址告诉从设备，从设备才能将数据放到 SDA 上供主设备读取。读时序步骤如下。

① 主设备发送开始信号，对应图 4.95 中 S 状态；

② 主设备发送七位的从设备地址，对应图 4.95 中发送地址阶段；

③ 主设备发送读命令（高电平），对应图 4.95 中 R/W#；

④ 接着从设备应答，ACK 为应答成功，表示有这个设备；

⑤ 从设备发送八位字节数据，主设备从 SDA 上读取数据；

⑥ 接着主设备应答，ACK 为应答成功；

⑦ 如果主设备应答成功 ACK，从设备继续往 SDA 上送数据；

⑧ 若数据已传输结束，主设备在发送 NACK 后发送停止信号，对应图 4.95 中 P 状态。

为了更清晰地描述，我们将图 4.95 所示的时序表示为 SDA 信号线上的位顺序图，如图 4.96 所示。图 4.96 中，深色底色的部分为主设备上 SDA 信号线发送的位，白色底色部分为从设备向 SDA 信号线上发送的位。S 为开始信号，P 为停止信号，ACK/NACK 为应答，R/W# 为读/写命令（高电平为读 R，低电平为写 W#）。SDA 信号线的驱动方在图 4.95 中无法显示，而图 4.96 中(a)和(b)则把主设备读和主设备写两种操作的差异显示出来了。

S (1bit)	从设备地址 (7bits)	R (1bit)	ACK (1bit)	数据 (8bits)	ACK (1bit)	数据 (8bits)	NACK (1bit)	P (1bit)

(a) 主设备读取从设备信息时SDA信号线的比特顺序

S (1bit)	从设备地址 (7bits)	W# (1bit)	ACK (1bit)	数据 (8bits)	ACK (1bit)	数据 (8bits)	NACK (1bit)	P (1bit)

(b) 主设备向从设备写信息时SDA信号线的比特顺序

图 4.96 I²C 的 SDA 信号线上的位顺序

每次发送地址/数据后加上应答信号构成的一个周期我们称之为帧（frame），则 I²C 总线上在开始 S 状态和停止 P 状态之间是以帧为单位进行信息传递的。例如，图 4.96 中(a)，S 状态后先发送了读命令，此后第一帧传递的是从设备的地址，第二帧传递的是从设备发出的数据。

往往 I²C 设备内部有一些具有存储能力的电路单元，如 EEPROM、RAM，或者用于芯片配置的多个寄存器等。根据在存储器章节中学习到的知识，访问这些设备中不同存储单元的内容时，首先要提供需访问数据单元的地址，然后才可以得到相应地址上的数据。如图 4.97(a)所示，如果从设备的存储单元地址约定为八位，那么传递的第一帧信息为拟访问存储单元的地址，第二帧和第三帧才是得到的数据。同理，图 4.97(b)中主设备向一个从设备写入信息，由于该从设备内部存储单元的地址是 16 位，所以第一帧和第二帧均为拟访问存储单元的地址。

				第1帧		第2帧		第3帧		
S	从设备地址 (7bits)	R	ACK (1bit)	地址 (8bits)	ACK (1bit)	数据 (8bits)	ACK (1bit)	数据 (8bits)	NACK (1bit)	P

(a) 主设备读取从设备信息（存储单元地址为8比特）的比特顺序

				第1帧		第2帧		第3帧		第4帧		
S	从设备地址 (7bits)	W#	ACK (1bit)	地址 (8bits)	ACK (1bit)	地址 (8bits)	ACK (1bit)	数据 (8bits)	ACK (1bit)	数据 (8bits)	NACK (1bit)	P

(b) 主设备向从设备写信息（存储单元地址为16比特）的比特顺序

图 4.97 通过 I²C 访问带存储单元的器件时位顺序示意图

5） I²C 电气特性

I²C 总线允许利用不同制造工艺生产的器件以及使用不同电源电压的器件进行通信。具有固定输入电平的 I²C 总线器件，可以分别单独连接适合自己的电源电压，但是 SDA 和 SCL 必须通过上拉电阻（如图 4.98 中 R_P）连接到 5V（1±10%）的公共电源上。单个 IO 引脚

的负载电容不能超过 10pF，所有 IO 引脚的总负载电容不能超过 400pF。为了拓展 I²C 总线的传输距离和可挂接器件数目，可以将 I²C 总线分段。分段时可以使用缓存（buffers）、多路复用器（multiplexers）、交换开关（switches）等不同的芯片隔离不同的段，每段总的负载电容不能超过 400pF。

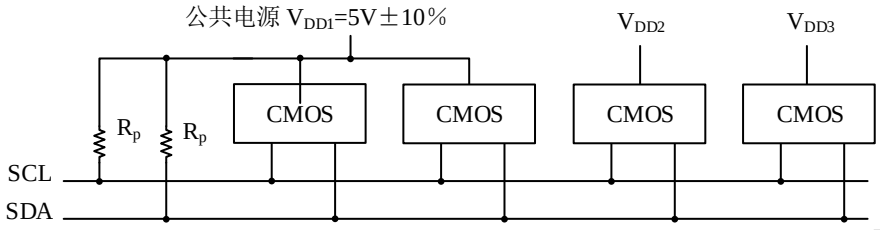


图 4.98　不同供电电压的设备挂接到同一 I²C 总线

当总线空闲的时候，所有的连接线均保持高电平。设备需要向总线输出的时候，必须是开集（open-collector）或者开漏（open-drain）的方式，以保证能够实现线与（AND）逻辑。

备注：Open-collector/Open-drain 是用于多个设备间使用一个连接线来双向（Bi-directionally）通信的技术。集电极引脚（Collector）没有和任何其他设备连接称之为开集（Open Collector）。BJT（双极结型晶体管，Bipolar Junction Transistor）的情况叫做 Open-Collector，MOSFET（金属氧化物半导体场效晶体管，Metal-Oxide-Semiconductor Field-Effect Transistor）的情况叫做开漏（Open-drain）。理论上 BJT 和 MOSFET 一样。

6） 示例：通过 I²C 访问 EEPROM

图 4.91 所示为微芯公司微控制器 dsPIC30F 与 24LC256 通过 I²C 实现连接的电路示意图。24LC256 是微芯公司生产的 256Kb（32K×8bits）串行 EEPROM（Serial Electrically Erasable PROM）。处理器访问该 EEPROM 芯片时，遵循 I²C 协议的数据传输过程。

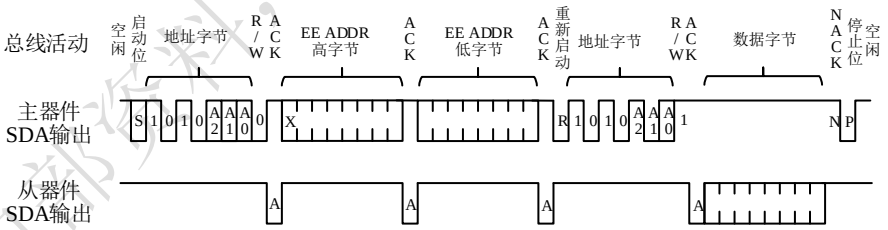


图 4.99　通过 I²C 访问 EEPROM 的报文示意

图 4.99 所示为典型的访问过程，在此示例中，dsPIC30F 作为主器件，24LC256 器件作为从器件，dsPIC30F 从 24LC256 读取指定的字节。由于访问 EEPROM 存储器芯片需要先提供拟读/写的存储单元地址，然后再传输数据，故图 4.99 所示过程与图 4.95 所示基本流程有些不同。可以将处理器芯片访问 EEPROM 的过程分成四个阶段。①主器件（处理器芯片）输出 24LC256 芯片的地址（I²C 地址），并发写指令；从器件应答。②主器件输出拟访问 24LC256 内存储单元的地址，16 位，从器件应答。③重复启动改变总线方向，输出 EEPROM 芯片的地址（I²C 地址），并发读指令，从器件应答。④从器件（24LC256）发出所指定地址存储单元的内容，主器件应答。

4. SPI 接口及总线

SPI（Serial Peripheral Interface，串行外设接口）是一种高速、全双工、同步的通信总线，通信过程使用四根信号线。SPI 在 19 世纪 80 年代推出，由 Motorola 首先在其 MC68HCXX 系列处理器上定义，目前是一种全球通用的标准。

1） SPI 概述

SPI 是一个同步串行接口，可用于与其他外设或者单片机进行通信。这些外设可以是串行 EEPROM、移位寄存器、显示驱动器和 A/D 转换器等。SPI 以主从方式工作，这种模式通常有一个主设备和一个或多个从设备，需要至少四根信号线，事实上三根也可以（单向传输时）。这四根信号线分别是 MISO、MOSI、SCLK 和 CS，含义如下。

- MISO（Master Input Slave Output，主设备数据输入/从设备数据输出）；
- MOSI（Master Output Slave Input，主设备数据输出/从设备数据输入）；
- SCLK（Serial Clock，时钟信号），由主设备产生；
- CS（Chip Select，从设备使能信号），由主设备控制。

注意，在很多描述 SPI 的资料中，往往也把 MOSI 称作 SDO，把 MISO 称作 SDI，把 SCLK 称作 SCK，把 CS 称作 SS 或 NSS。阅读不同厂家芯片数据手册时候，要注意区分信号线命名方式的差异。

如图 4.100 所示，在 SPI 上可以挂载一个主设备和多个从设备。任何时刻，一个主设备只与一个从设备进行通信，通信的从设备 CS 为低电平（有效）。在不使用 CS 信号时，SPI 上只能有一个主设备与一个从设备。SPI 的缺点是没有应答机制，传输过程全都由主设备进行控制，数据传输成功与否没法直接验证。

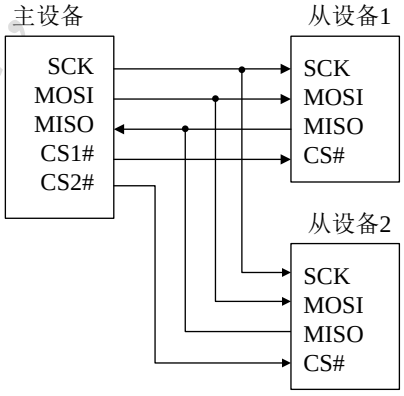


图 4.100 SPI 中主设备与从设备的典型连接

SCK、MISO 和 MOSI 三根信号线配合完成通信功能。由于 SPI 是串行通讯协议，也就是说数据是一位一位的传输，每个时钟周期传输一位。SCK 提供时钟脉冲，MISO、MOSI 则基于此脉冲完成数据传输。数据输出通过 MOSI 线，数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被读取。完成一位数据传输，输入也使用同样原理。因此，至少需要八次时钟信号的改变（上升沿加下降沿为一次），才能完成八位数据的传输。

SCK 时钟信号线只能由主设备控制，从设备不能控制该信号线。SPI 总线上，至少要有一个主控设备。SPI 的传输过程允许暂停，主控设备控制的 SCK 时钟线上没有时钟跳变时，从设备不采集或传送数据。也就是说，主设备通过对 SCK 时钟线的控制可以完成对通信过程的控制。SPI 的数据输入信号线和输出信号线独立，所以允许同时完成数据的输入和输出。不同的 SPI 设备的实现方式不尽相同，主要是数据改变和采集的时间不同，在时钟信号上沿或下沿采集有不同定义，具体需要查阅器件的数据手册。

2） SPI 典型接口电路

各个不同的厂家实现的 SPI 的接口电路一般包括数据发送寄存器、数据接收寄存器、控制寄存器、状态寄存器，时钟有关的寄存器等。图 4.101 所示为意法半导体的 STM32 系列微控制器中集成的 SPI 的接口电路。该芯片定义的四个引脚名称为：MISO，主设备输入/从设备输出引脚；MOSI，主设备输出/从设备输入引脚；SCK，时钟引脚；NSS：从设备选择引脚。

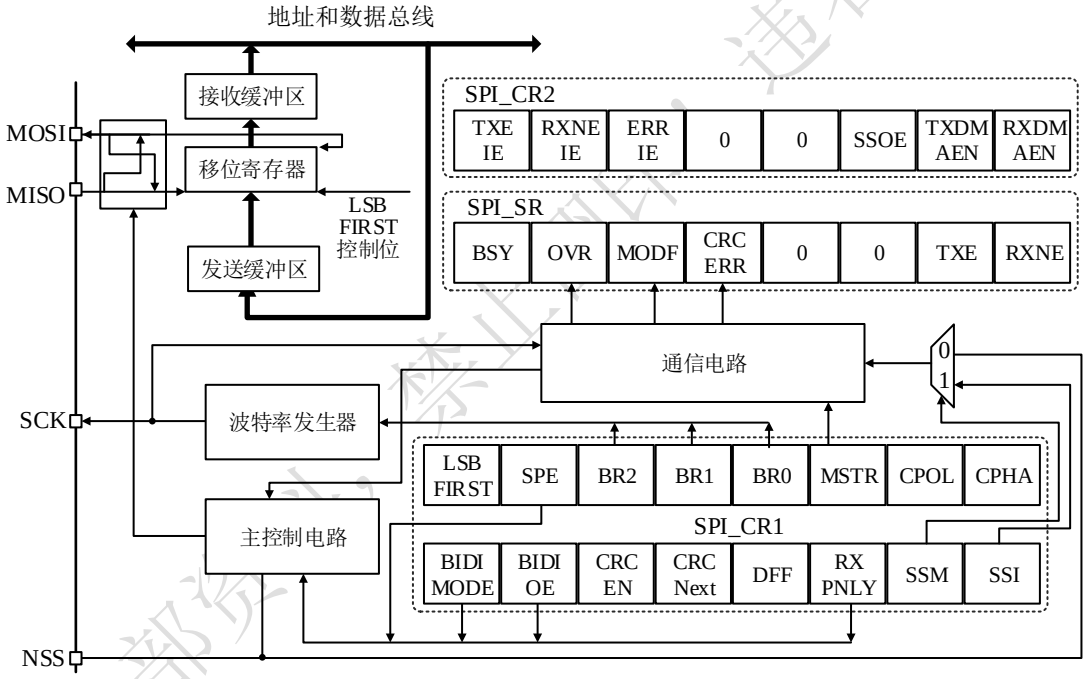


图 4.101 意法半导体 STM32 系列处理器的 SPI 接口电路

图 4.101 中发送缓冲区、移位寄存器、接收缓冲区的设计在一般串行通信接口电路中大体一致。状态寄存器 SPI_SR、控制寄存器 SPI_CR1、SPI_CR2、波特率控制寄存器 BR 虽然在不同厂家的 SPI 接口电路中也都有，但是寄存器位定义往往略有差异。

3） SPI 数据传输过程

SPI 是一种简单的主从通信协议。整个通信过程由主设备发起，从设备参与。当一个主设备需要向从设备发送数据，或者希望读取从设备数据的时候，主设备通过拉低对应从设备的 CS#来告知从设备。对于主设备来说，发送数据就是把位逐个放到 MOSI 信号线上，而读取数据就是在 MISO 信号线上进行采样。如图 4.102 所示。

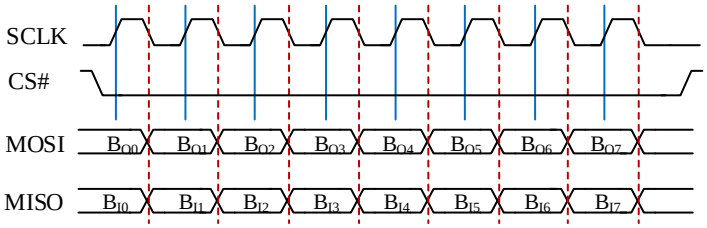


图 4.102　SPI 的一次通信过程（时钟上升沿采样输入/下降沿输出位）

SPI 工作状态极为简单，只有工作和空闲两个状态。根据空闲状态对应时钟的高电平还是低电平，以及时钟上升沿和下降沿的动作，可以形成不同的工作方式。SPI 总线有四种工作方式（SPI0、SPI1、SPI2、SPI3），其中使用的最为广泛的是 SPI0 和 SPI3 方式。这四种工作方式是根据 *CPHA*（clock phase，时钟相位）和 *CPOL*（clock polarity，时钟极性）来区分的。*CPOL* 为“0”时 SPI 总线空闲为低电平，为“1”时 SPI 总线空闲为高电平；*CPHA* 为“0”时在 SCK 第一个跳变沿采样，为“1”时在 SCK 第二个跳变沿采样。*CPOL* 与 *CPHA* 不同取值可形成四种组合：*CPHA*=0、*CPOL*=0，*CPHA*=0、*CPOL*=1，*CPHA*=1、*CPOL*=0，*CPHA*=1、*CPOL*=1，依次对应 SPI0、SPI1、SPI2、SPI3 四种工作方式。

4.5 习题

4.1　计算机系统为什么需要采用总线结构？

4.2　举例说明何为总线复用？

4.3　计算机总线有哪些类型？

4.4　某计算机系统的地址总线宽度是 13 位，其数据总线宽度是 11 位，再不采用总线分时复用的情形下，请计算该计算机的最大存储器空间寻址范围。

4.5　计算机系统中总线层次化结构是怎样的？

4.6　对比陈述面向 CPU 的双总线结构和面向存储器的双总线结构的优缺点。

4.7　总线性能的评价指标有哪些？

4.8　计算机系统什么情况下需要总线仲裁（arbitration）？

4.9　总线仲裁方式有哪几种？各有什么特点？

4.10　“线与”用什么样的电路可以实现？

4.11 总线周期分为哪些阶段？

4.12 同步总线传输对收发模块有什么要求？什么情况下应该采用异步传输方式，为什么？

4.13 异步总线有哪些可能的握手方式？

4.14 半同步总线相比同步总线和异步总线有哪些优点？适用于什么样的场景？

4.15 周期分裂式总线操作时序有哪些特点？适用于什么样的场景？

4.16 AMBA2 总线定义了哪三种总线？他们各有什么特点？

4.17 AMBA AHB 总线的特点是什么？总线仲裁器的作用是什么？

4.18 APB 桥接器的功能是什么？

4.19 为什么 AMBA 总线中没有定义电气特性和机械特性？

4.20 AHB 中为什么要定义地址阶段（Address Phase）和数据阶段（Data Phase）？

4.21 简述 AHB 总线的流水线机制。

4.22 简析 AHB 中 SPLIT 操作的优点。

4.23 解释图 4.23 中 HREADY 信号的作用。

4.24 AHB 突发传输有什么特点？

4.25 AHB 中突发传输定义了哪些类型？各自有什么特点？

4.26 画出 AHB 中采用突发传输的“INCR4”类型主机接收从机数据的时序。

4.27 画出 AHB 中采用突发传输的“WRAP8”类型主机向从机发送数据的时序。

4.28 PCI 系统总线有什么样的特点？

4.29 PCIe X32 中 X32 的含义是什么？

4.30 PCIe 5.0 版本中 X16 的吞吐量 63.0 GB/s 是如何计算得到的？

4.31 解释 PCIe 中通道（lane）和信号线（wire）的概念。

4.32 串行传输的特点是什么？

4.33 什么是串行传输的全双工和半双工方式？

4.34 发送时钟和接收时钟与波特率有什么关系？

4.35 异步串行通信中的起始位和停止位有什么作用？

4.36 简述 RS-232C 的规程特性。

4.37 简述 I/O 接口的功能和作用。

4.38 什么是 I/O 端口？一般接口电路中有哪些端口？

4.39 CPU 对 I/O 端口的编址方式有哪几种？各有什么特点？

4.40 接口电路的输入需要用缓冲器，而输出需要用锁存器。为什么？

4.41 CPU 与 I/O 设备之间的数据传送有哪几种方式？每种工作方式的特点是什么？各适用于什么场合？

4.42 简述中断处理的流程。

4.43 分析图 4.67 所示电路，解释该电路如何保证在多个中断同时发生时仅把优先权最高的中断信号送给 CPU。

4.44 分析图 4.68 所示菊花链电路，某计算机系统有 4 个中断源，设计基于菊花链的优先级排队电路（画出电路示意图），并指出优先级最高的是哪个中断源。

4.45 名词解释：中断向量表。

4.46 数据块传送方式的 DMA 适用于什么场景？

4.47 常用的中断优先级的管理方式有哪几种？分别有哪些优缺点？

4.48 在微机与外设的几种输入输出方式中，便于 CPU 处理随机事件和提高工作效率的 I/O 方式各是哪一种？数据传输速率最快的是哪一种？

4.49 什么是并行接口？什么是串行接口？各有什么特点。

4.50 简述线性键盘与矩阵键盘的区别如何消除键盘的抖动？

4.51 什么是矩阵键盘的行扫描法？

4.52 简述 LED 数码管的静态显示原理。

4.53 简述 LED 数码管的动态显示原理。

4.54 串行通信双方为什么要约定通信协议？异步串行通信协议包括哪些内容？

4.55 远距离的串行通信系统为何需要调制解调器（Modem）？

4.56 异步串行通信中收发双方时钟难以保持一致，接收端如何确保正确的信号波形检测？

4.57 采用异步串行通信时，接收器如何确定起始位？

4.58 异步串行通信系统中，采样数据时为什么要在数据位的中间？

4.59 有哪些措施有利于提高串行通信系统的最大通信距离？

4.60 SPI 标准中有片选信号 CS，而 I^2C 总线中没有定义片选，I^2C 总线采用了什么方法实现片选信号的功能？

4.61 描述 I^2C 总线协议中的状态，并画出状态转移图。