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 到内存以及从内存到显卡等，都需要进行数据传输。在计算机发展的早期，这些部件两两之间采用专用通道进行连接，致使计算机内部连线十分复杂，不仅布线十分困难，并且可扩展性差。总线技术出现后简化了系统结构，极大改善了部件之间的协调性和系统的可扩展性。直观地看，总线是一组导线，不同功能单元通过这组导线彼此互连。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传输数据的能力。这种通信能力不仅依赖于直观上的一组导线，还需要相关硬件电路和软件的支持。本章 4.1 节将介绍总线基本知识，4.2 节和 4.3 节分别介绍用于芯片内部和芯片外部的典型总线。

当处理器与外设进行数据交换时，并不只是简单地把不同电路单元的信号线接在一起。一台计算机主机可能同时连接多台外设，而外设的功能多种多样，有些是输入设备，有些是输出设备，有些既作输出设备又作输入设备。不同外设的输入信号形式也是多种多样，如模拟信号、数字信号和开关信号[1]，有些输入信号仅有一根信号线，有些则多达数十根，有些采用串行通信方式，有些则采用并行通信方式。不同外设的工作速度也不同，例如，键盘和鼠标每秒钟产生的输入事件不超过几十次；而主机与外部存储系统的数据传输速度现已达 16Gpbs。总而言之，外设种类繁多，不同外设在速度、信号形式和电平等方面存在巨大差异。因此，要保证可靠的数据传输，必须在处理器与外设之间加入合适的中间环节，来协调处理器与外设的差异，这个中间环节就是 I/O（Input/Output，输入/输出）接口。广义的接口还包含接口电路相应的驱动程序，不同外设的接口电路和驱动程序是不同的，同一种外设连接到不同计算机时，接口电路和驱动程序也不同。本章 4.4 节将对此进行讨论。

总线与接口之间既有联系又有区别，这些联系与区别包括：

- 接口包含模块之间互连的硬件电路，由于需要连接不同类型的设备，其功能应该按照约定的规则进行设计；而总线不仅包括传输线路及相关电路，也包括传输和管理信息的规则（即协议）。
- 从适用场景看，如果仅仅是两个模块之间进行互连，无须定义共享规则，只需简单的逻辑电路进行协调即可；但多个模块基于一套传输线路进行数据交换时，则每个模块不仅要有接口电路，还需要按照一定的协议来规范多个模块共享传输线路的方式。

简而言之，本章主要介绍计算机系统中处理器与其他功能模块之间、主机与外部设备之间、或者不同功能模块之间实现互连的原理、方法与实现技术。

4.1 总线技术

本节介绍计算机系统总线的分类和结构，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。

[1] 开关信号可以看作一位数字信号，它只有两个状态，用以表示事件发生与否、开关的开或闭以及灯的亮与灭等。

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，为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。单独服务于某两个模块、部件或设备之间的专用信号连接线，不能称为总线。依据计算机系统部件的功能、位置，在不同场合往往有模块、部件和设备等不同称呼，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，则实现所有模块互连需要 $n\times(n\text{-}1)/2$ 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

总线英文名称是 bus，bus 为世人熟知的含义是公共汽车。如果将计算机的主板（Main board 或者 Mother Board）比作一座城市，那么总线就是城市里的公共汽车，为整座城市提供公共交通运输服务。总线中的每条线路在一个总线时钟周期内仅能传输一个比特，采用多条线路或者使用更多的时间周期才能传送更多数据。使用多条线路同时传送数据称为并行总线，并行总线所使用的线路条数称为总线宽度（Width）。理论上，并行总线的宽度越大，总线时钟频率越高，数据传输速率就越高。但是受电路特性以及电路板运行周边环境的影响，总线宽度和总线时钟频率都存在上限。通常用总线带宽（Band Width）来描述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（字节/秒）

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，造成总线冲突。因此，在任何时刻，连接到总线上的多个模块只能有一个模块输出信号，而其他模块处于接收状态。

那么，如何避免总线冲突呢？这就需要使用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，并指定总线驱动器的传送方向。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。通常，包含总线控制器功能的模块被称作主控模块（Master），在主控模块协调下被动工作的模块被称作从属模块（Slave）。实际中，很多模块在不同的时刻扮演不同的角色，在某些时刻工作在主控模块方式，在另外一些时刻工作在从属模块方式。

通过以上介绍可以看出，总线有两个基本特性，其一是共享。多个模块可以连接在同一组总线上，彼此之间交换的信息都可以通过这组总线传送。其二是分时。所谓分时是指任意时刻只能有一个模块向总线发送信息，但是多个模块可同时从总线接受相同信息。分时特性制约了系统性能。

2. 总线的分类

可以从不同的角度对总线分类进行。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、系统总线和系统外总线。按照总线的功能，可分为地址总线、数据总线和控制总线。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）指芯片内部用于连接各单元电路的信息通路，如微处理器芯片内部连接 ALU、寄存器和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），用于系统内部各模块互连。例如 ISA、PCI 和 PCI Express 等。外总线（external bus），又称 I/O 总线或通信总线（communication bus），用于计算机之间或者计算机与外设之间的互连，例如 SCSI 总线和 USB 总线等。

近年来，随着 SoC（System On Chip，片上系统）的迅速发展，单颗芯片内不仅集成了 CPU，也集成了存储器，甚至集成了过去作为外设的 GPS 和蓝牙模块等。上述片内总线、片上总线、内总线和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“片上总线（on-chip bus）”来描述芯片内部使用的总线。事实上，片上总线涵盖了传统的内总线及外总线。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同时存在片内总线、系统总线和外总线，大部分计算机系统采用多级分层总线结构，如图 4.1 所示。在图 4.1 中，处理器的片内总线连接了 CPU 核心与高速缓存（cache）和高速缓存控制器。高速缓存控制器也是片内总线与高速系统总线之间的桥梁和纽带，所以也称为桥接器。这样的电路结构中，微处理器片内总线提供 CPU 核心与高速缓存间的高速传输通道；系统总线则提供了高速缓存和主存储器以及其它高速部件之间的数据传输通道，有利于提高计算机系统的整体性能。

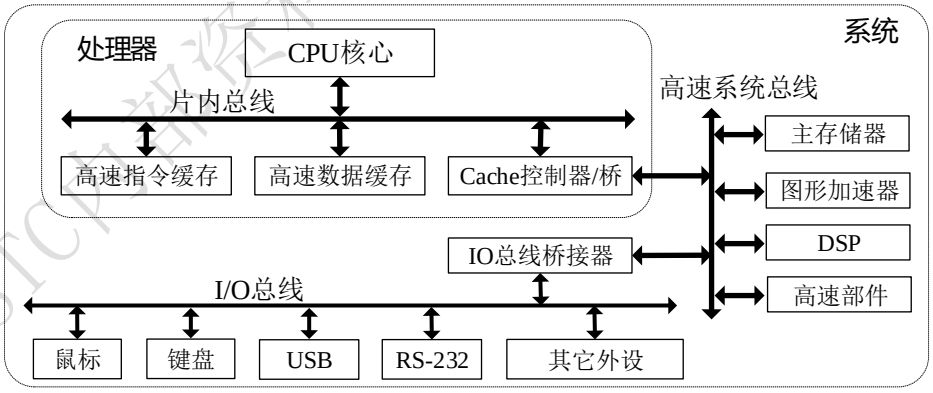


图 4.1　总线的层次结构

图 4.1 中 I/O 总线也称为外部总线，用于连接外部设备。由于外设种类繁多，数据传输速率不一致，很多计算机系统的外部总线也采用层次化结构。例如，图 4.2 显示了另外一种分层总线结构。图中，CPU 与存储器、高速外设以及低速外设之间使用了三种不同层次的总线。外部总线采用层次化结构的基本思想是将各个部件按照速率进行等级划分，让速率差距相对不大的单元挂接到同一条总线上，不同层次的总线通过特定接口（称为总线桥）实现彼此之间的互连。

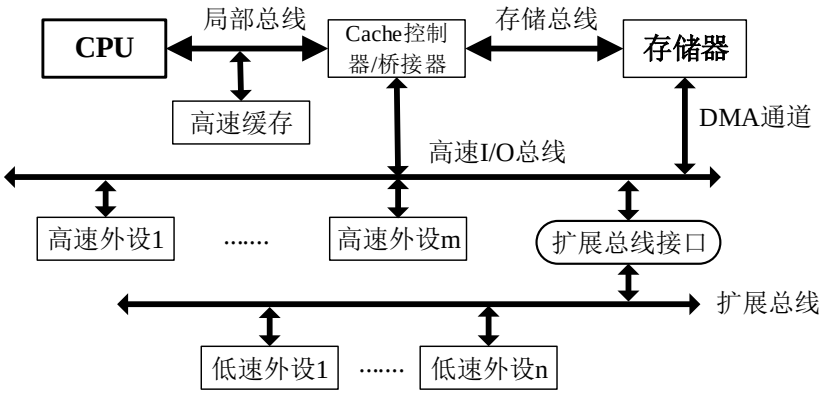


图 4.2　外部总线的分层结构

2） 按总线功能分类

从总线功能角度，总线分为地址总线 AB、数据总线 DB 和控制总线 CB 三大类，如第 2 章图 2.2 所示。三类总线分工明确，地址总线 AB 专门传送地址；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，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，送往总线上各个模块。地址信号指示数据总线上数据的来源与去向。例如，CPU 希望从存储器中读取一个字的数据，需要将字所在的地址信息送到地址总线上。地址总线的宽度决定了系统存储器空间的最大寻址范围。例如，20 位宽度的地址总线可寻址空间为 2^{20}=1M，即可寻址 1M 个存储单元。若数据总线为 8 位宽，此时可寻址 1MB（即 1M 字节）；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 端口地址方面的内容可参阅 4.4 节。

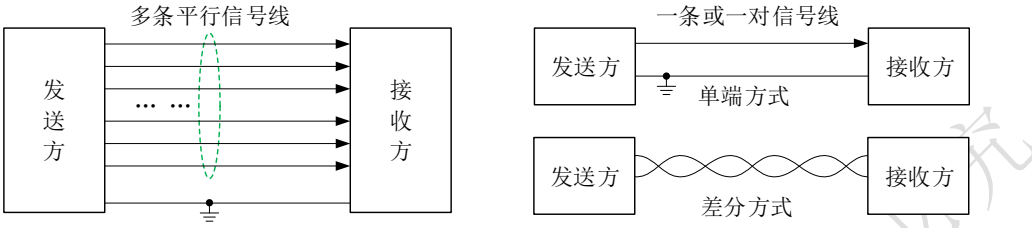
数据总线 DB 传送数据信息。数据总线通常为双向三态形式的总线，既可以把 CPU 的数据传送到存储器或 I/O 接口等部件，也可以将各部件的数据传送到 CPU。数据总线的位数（宽度）是计算机系统的一个重要指标，通常与微处理器的字长相一致。例如 Intel 8086 微处理器字长 16 位，其数据总线宽度也是 16 位。需要指出的是，数据的含义是广义的，可以是真正的数据，也可以是指令代码或状态信息，有时甚至是控制信息。因此计算机工作过程中，数据总线上传送的并不仅仅是真正意义上的数据。

控制总线 CB 传输各种控制信号。控制信号中，有的是 CPU 送往存储器和 I/O 接口电路的，如读/写使能信号、片选信号和中断响应信号等；也有其它部件反馈给 CPU 的，如中断请求信号、复位信号、总线请求信号和设备就绪信号等。换言之，控制总线用于控制数据总线和地址总线。控制信号负责总线上各模块间的联络控制，让数据总线和地址总线的使用有序化。有些控制信号仅连接一个外设模块，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、数据线和地址线以外的所有信号线都归为控制总线。

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，可以同时传送多个二进制位，其二进制位的个数为该总线的宽度，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。并行总线需要使用多条平行的信号线，其总线时钟频率不能太高，否则平行信号线之间会出现较强的线间干扰，将严重影响数据传输的正常进行。为了进一步提高内总线的数据传输速率，近十几年来，基于串行方式的系统内总线得到了广泛应用。例如，在目前各种类型的 PC 机中，采用串行方式的 PCIe 总线已经全面取代了基于并行方式的 PCI 总线。



(a) 并行通信　　(b) 串行通信

图 4.3　并行数据传送和串行数据传送示意图

串行总线的通信方式分为单端（single ended）和差分（differential）两种。单端方式只需使用一根信号线以及一根信号参考地线，差分方式需要使用一对差分信号线，如图 4.3(b)所示。其中差分传输方式具有较强的抗干扰能力，传输距离远并且可以大幅度提高总线时钟频率。因此，虽然串行总线只能逐位传送数据，但是如果采用差分传输方式，可以通过提高总线时钟频率使其达到较高的数据传输速率。

系统外总线多采用串行总线，由于传输信号线的数量少，可节省线路成本，同时便于实现远距离传输。

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的关系，总线可分为同步总线、异步总线和半同步总线三类。

在同步总线中，收发双方的任何动作都基于相同的时钟基准，所有操作均按照规定的时钟周期进行，并要求在规定的时间内完成规定的操作。一般来说，同步总线的传输速率快，但是对总线所连接模块的速度有较严格要求，对模块的适应性差，可靠性也不高。CPU 与高速缓存控制器以及与系统主存之间普遍使用同步总线，以期提高系统性能。

而在异步总线中，收发双方的时钟信号相互独立，彼此之间通过某种协调机制了解对方当前的动作或者行为，然后再确定自身应该执行何种操作。收发双方的协调又称为应答或者握手。由于增加了协调机制，总线上速度快慢不同的模块之间均能进行可靠的数据传输。但是彼此之间过多的协调和握手将影响数据传输效率。历史上著名的 VME 总线，以及广泛应用于智能仪器的 GPIB（IEEE-488/IEC-625）总线是异步总线的典型代表。

如果在同步总线的基础上，收发双方之间增加一种协议机制，平时按照同步总线方式进行传输。如果有一方“跟不上节奏”，可通过协调机制要求对方延长一个或者多个时钟周期（称为插入等待），保证双方可靠地进行数据传输。这样即兼顾了数据传输速度，也提高了数据传输的可靠性。采用这种方式的总线即是半同步总线，例如，在早期 PC 机中所使用过的从 I/O Channel、ISA、EISA 和 PCI 总线等都属于半同步总线。

5） 按时分复用方式分类

在许多系统中，为了减少物理信号线或者芯片引脚数量，往往将不同的信号共用一条或者一套物理信号线或者芯片引脚。这些物理信号线或者引脚在不同的时刻体现不同的功能和用途。例如，在 Intel 8086 芯片上，20 位地址总线的低 16 位与 16 位数据总线共用物理引脚。在总线周期的最初时钟周期，这些物理引脚上出现的信号表示地址，在接下来的时钟周期这些引脚又用来传输数据。这种方式称作时分复用。在系统总线上，如果需要，可以使用锁存器将分时复用的信号进行分离。总线中也有一些信号无需分离，在不同的时钟周期表示不同的信号。例如，PCI 总线中的$C/\overline{BE}[3:0]$在传输地址周期表示总线命令；在传输数据周期，这 4 条信号线用于字节使能（指明 32 位总线上哪些字节有效）。

一般来说，采用非时分复用总线有利于提高数据传输速率，而采用时分复用总线有利于减少芯片引脚或者物理信号线的数量，设计师需要在两者之间寻求平衡。

3. 总线的结构

从计算机组成结构来看，有两类总线结构：单总线结构和多总线结构。多总线结构又有双总线、三总线和四总线之分。

1） 单总线结构

单总线结构中，计算机系统的所有部件都通过同一套总线交换信息。其优点是控制简单，扩充方便。典型单总线结构如图 4.4 所示，图中 CPU、存储器、I/O 接口均挂接在相同的总线上。显然，速度快慢不同的模块均使用同一条总线，同一时刻只有一个模块可以驱动总线，之间传送数据，总线的数据传输速率必定受限。

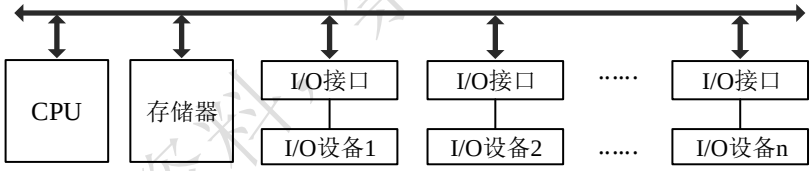


图 4.4　经典的单总线结构

2） 双总线结构

双总线结构分为面向 CPU 的双总线和面向存储器的双总线。面向 CPU 的双总线结构如图 4.5 所示。其中一组总线是 CPU 与主存储器间交换信息的公共通路，称为存储总线。另一组总线是 CPU 与 I/O 设备交换信息的公共通路，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。由于外部设备与主存储器之间没有直接的通路，彼此之间的信息交换须经过 CPU，这会降低 CPU 的工作效率。

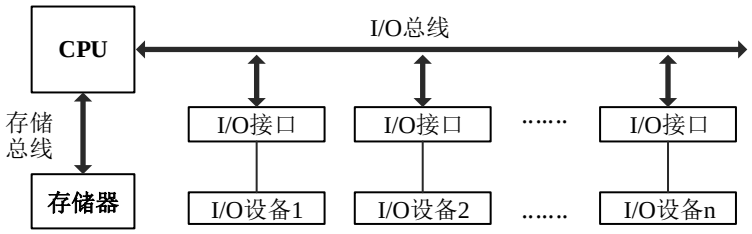


图 4.5　面向 CPU 的的双总线结构

面向存储器的双总线结构如图 4.6 所示。这种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。但是与单总线结构不同的是，CPU 与存储器之间专门设置了一条高速存储总线，使 CPU 与存储器之间可以高速交换信息，提高了系统的整体性能。但是 I/O 接口与存储器交换信息时，仍然需要 CPU 负责操作。

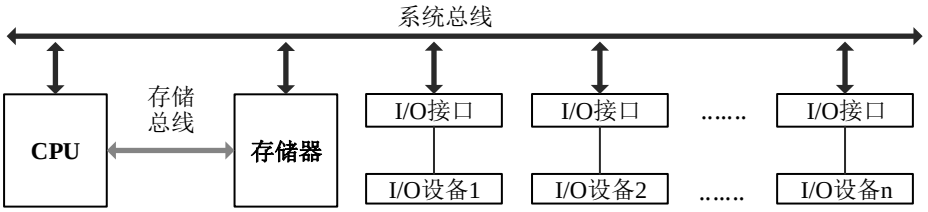


图 4.6　面向存储器的的双总线结构

3) 三总线结构

典型的三总线结构计算机系统如图 4.7 所示。这种结构在高速 I/O 接口与主存之间建立了一条高速的直达通道，该通道称为 DMA（Direct Memory Access，直接存储访问）总线。高速 I/O 接口与主存之间的数据交换由专门的设备进行控制，该设备称为 DMA 控制器或者 DMAC（图 4.7 中没有画出）。关于 DMA 传送方式更详细的介绍请参见本章 4.4.2 小节第 4 部分。

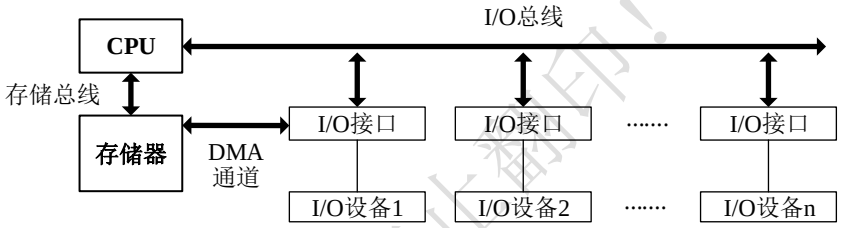


图 4.7　典型三总线结构

4) 四总线结构

为了进一步提高 I/O 设备的性能，使其更快地响应命令，又出现了四总线结构，典型的四总线结构如图 4.8 所示。图 4.8 条总线分别是局部总线、系统总线、高速 IO 总线和扩展总线。高速 IO 总线专门用于连接高速 I/O 设备，如网络适配器、图形图像处理器和硬盘接口等。而慢速设备（如键盘、鼠标、FAX 和 Modem 等）则与扩展总线相连。PC 机中一度广泛使用的 PCI 总线就属于 4 总线结构。在 PCI 总线结构中，按照地图上的方位，图 4.8 上方的 Cache/桥又被称为北桥，而下方的扩展总线接口被称为南桥。

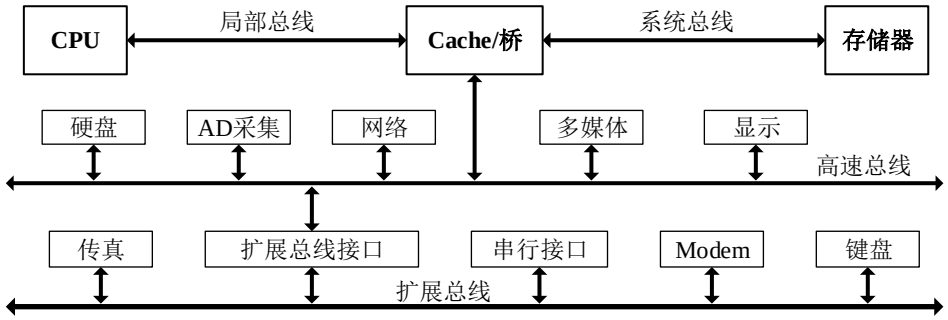


图 4.8　典型的四总线结构

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）。总线主设备有控制总线数据传输的能力，可以是 CPU 或其他逻辑控制模块。总线主设备在获得总线控制权之后能启动并控制数据的传输；与之相对应的总线从设备，可以对总线上的数据请求做出响应，但不能控制总线。常把总线主设备称作总线主机，把总线从设备称作总线从机。

早期计算机系统中，一条总线上只有一个主设备，由该主设备控制总线传输。随着计算机技术的发展，多个主设备共享一条总线的情况越来越普遍，此时需要协调各个主设备对总线资源的争用。

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使用请求，以避免冲突。多个主设备同时提出总线使用请求时，需要按预定义的算法确定哪个主设备获得总线控制权。实现总线资源分配功能的逻辑电路称为总线仲裁器（bus arbiter），简称仲裁器。

总线仲裁有两大类实现方式：集中式和分布式。集中式控制由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，其总线协议较简单，但总体性能不易提高。分布式控制将总线控制逻辑分散在各个模块中，其总线协议略显复杂，电路成本较高，但有利于提高总线利用率。

1. 集中式仲裁

依据仲裁机制，集中式控制分为串行仲裁、并行仲裁和混合仲裁。

（1）串行仲裁

串行仲裁最典型的当属“雏菊链”（daisy chain）或者链式仲裁方式。图 4.9 所示为常用的三线链式仲裁电路示意图，使用到的信号线包括：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。

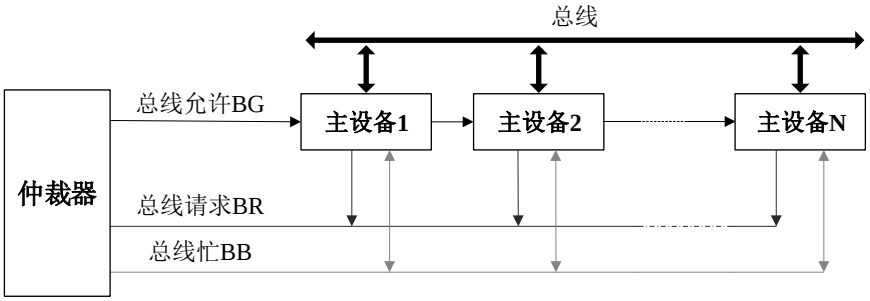


图 4.9　串行仲裁

在图 4.9 所示电路中，主设备在提出总线使用请求前，要先检测 BB 信号，当 BB 信号无效（总线空闲）时，主设备才能提出总线请求。各主设备的 BR 信号用“线与[2]”方式接到仲裁器的请求信号输入端。仲裁器收到 BR 信号后输出 BG 信号，BG 信号通过主设备链向后传递，直至提出总线请求的那个设备。当请求总线的主设备收到 BG 号后，获得总线控制权，并且不再向后传递 BG 信号，同时将 BB 信号置为有效，以通知其他设备总线已被占用。当主设备的总线操作结束后，撤销 BB 信号，随后其他设备可以继续申请总线使用权。

链式仲裁特点包括：①设备使用总线的优先级由它与仲裁器之间距离决定，离仲裁器越近优先级越高；②信号线数量与设备数目无关，电路实现简单；③BG 信号传递必须在一个总线周期内完成，由于存在电路时延，总线时钟频率不能过高，总线性能受限，并且主设备链上的设备个数一般不超过 3 个；④主设备链的连接方式确定之后，优先级就固定了，不易更改。

（2）并行仲裁

并行仲裁方式下，每个主设备都有独立的 BR 和 BG 信号线，分别接到仲裁器上，如图 4.10 所示。需要使用总线的主设备通过 BR 信号向仲裁器发出请求，仲裁器按预先约定的优先级算法选中某个主设备，并把 BG 信号送给该设备。被选中的主设备撤销 BR 信号，并输出有效的 BB 信号，以通知其他设备总线已被占用。主设备的总线操作结束后，撤销 BB 信号，同时仲裁器撤销对应 BG 信号，并根据整体请求情况重新分配总线控制权。并行仲裁的优点是速度快，优先级设置灵活，但每增加一个主设备就需要增加一对 BR 和 BG 信号线，系统中允许接入的主设备数量受仲裁器电路规模的限制。

[2]“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。

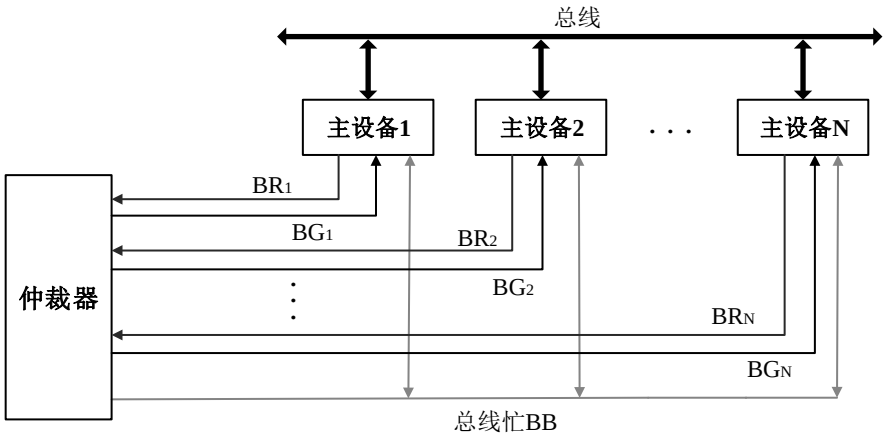


图 4.10 并行仲裁

（3）混合仲裁

混合仲裁又称多级仲裁，它结合了并行仲裁和串行仲裁两种思路，更为灵活。

图 4.11 所示为一种两级混合仲裁，图中的 BR1 和 BR2 以并行方式连接到仲裁器，各主设备的总线请求 BR 要么连接在 BR1，要么连接在 BR2 上。BG1 以串行方式连接一部分主设备，BG2 以串行方式连接另一部分主设备。

所有主设备的总线请求经 BR1 或 BR2 送达仲裁器，仲裁器决定把总线控制权交给 BR1（所连主设备）或 BR2（所连主设备）。若总线控制权交给 BR1（所连主设备），则接下来 BG1 串行传递，主设备 1 优先级高于主设备 3。若总线控制权交给 BR2（所连主设备），则接下来按串行方式决定获得总线控制权的是主设备 2 还是主设备 4。并行请求 BR1 和 BR2 的优先级由仲裁器预先约定的优先级算法确定，同一链上各设备优先级则决定于该设备与仲裁器的距离。

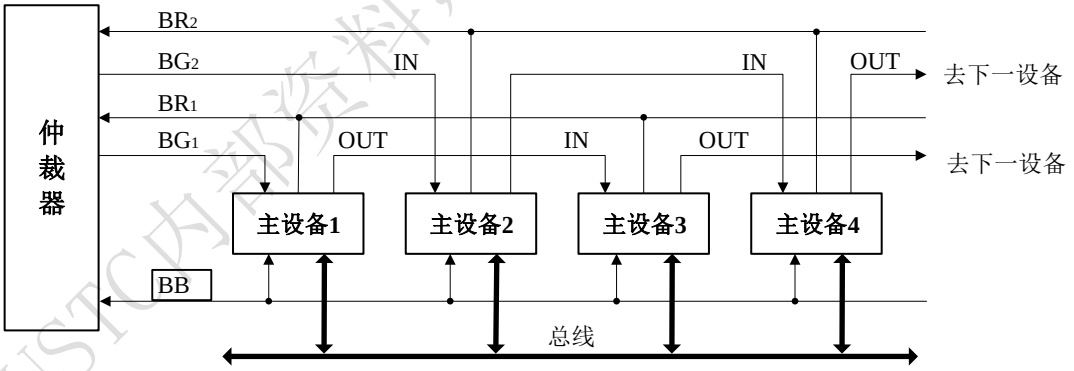


图 4.11 混合式仲裁

混合式仲裁方式兼具串行仲裁和并行仲裁的优点，灵活并且易于扩充，较好地兼顾了电路结构和响应速度两方面的要求。

2. 分布式仲裁

分布式仲裁方式没有中心仲裁器，而是把总线仲裁逻辑分散到每个主设备上，主设备之间通过协商自主确定总线使用权。图 4.12 所示是一种自举式仲裁逻辑的原理图，属于分布仲裁方式中的一种。图中，每个模块都带有总线控制逻辑，并使用了多根请求线。总线使用权的分

配过程分为申请期和裁决期两个阶段。在申请期，需要使用总线的设备在各自的总线请求线上送出请求信号；在裁决期，每个设备将请求线上的信号取回分析，以确定自己能否拥有总线使用权。对于发出总线申请的设备而言，如果在取回的合成信息中检测到有优先级更高的设备发出了总线请求，则本设备暂不能使用总线；反之，本设备可立即使用总线。

在图 4.12 中，主设备 4 如果需要使用总线，在申请期通过 BR_4 发出总线请求，在裁决期无需理会其他设备是否有请求，因此主设备 4 拥有至高无上的最高优先级。而主设备 3 在申请期发出了请求，在裁决期对取回的合成信息进行分析，只要 BR_4 无效（主设备 4 没有请求），主设备 3 认为自己具有当前最高优先级，可以使用总线，依次类推。至于主设备 1，其在申请期发出的 BR1 无人理会，可以移除。只有其他设备都没有总线申请时，主设备 1 才有机会使用总线，因此只有最低的优先权。图 4.12 中 BUSY 为总线忙信号，正在使用总线的模块应将 BR_1 置为有效，其他主设备只有在 BUSY 无效时才能在申请期发出请求。

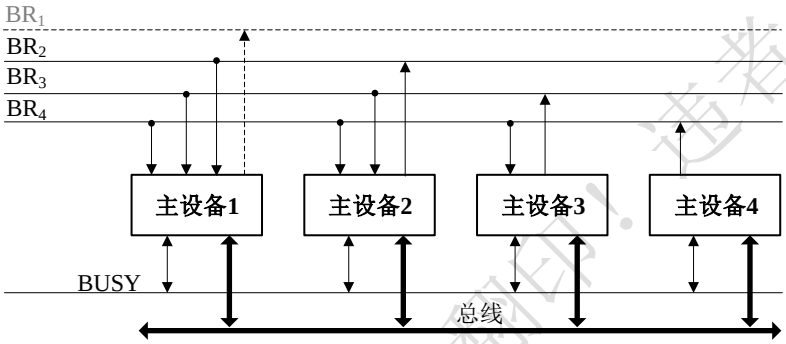


图 4.12　自举分布式仲裁

4.1.3 总线操作与时序

计算机不同部件通过总线交换信息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），如读存储器周期、写存储器周期、读 I/O 端口周期和写 I/O 端口周期等。多主设备的总线系统中，一个总线周期分为四个阶段：请求及仲裁阶段、寻址阶段、数据传输阶段和结束阶段。若系统中仅有一个主设备，不需要总线请求和仲裁，也不需要结束阶段，总线传输周期只有寻址和数据传输两个阶段。

在请求及仲裁阶段，拟使用总线的主模块提出请求，由总线仲裁器确定哪一个模块获得下一个总线传输周期的使用权。在寻址阶段，取得总线使用权的主模块发出拟访问的从模块（存储器地址或 I/O 端口）地址以及操作命令。在数据传输阶段，主模块和从模块间进行数据传输。在结束阶段，主从模块均从总线上撤销相关信号，让出总线。

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。根据主、从设备在时间上配合方式的不同，总线时序常被分为四种：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，每根信号线按照相同节拍工作。一般来说，同步总线速度较快，且逻辑控制较为简单。但

同步总线存在总线偏离[3]问题，总线速度受制于最慢的设备。

CPU 与主存储器间的总线一般采用同步总线。图 4.13 所示是一种典型的同步总线对存储器进行读写操作（参考 PROLOG 公司的 STD 总线[4]）的时序。图 4.13 中波形的上升沿或下降沿没有画成垂直的，这是因为信号不可能在零时间内从高电平变为低电平，或从低电平变为高电平。一般时序图中都会将波形的上升沿和下降沿画成呈一定角度的斜线。

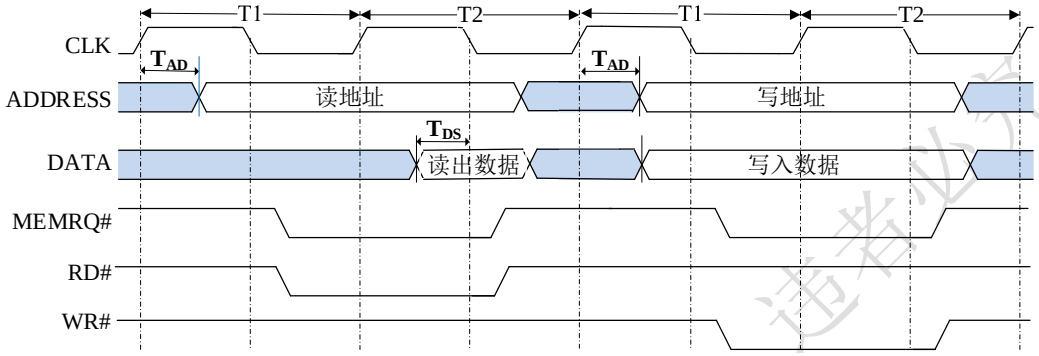


图 4.13　一种典型的同步总线时序

以下以 CPU 对存储器的读写操作为例，介绍图 4.13 中各信号线的定义和作用。

- CLK，时钟信号，总线标准中需要约定时钟频率的范围；
- ADDRESS，地址总线，由主模块中的 CPU 驱动，存储器根据地址对存储单元进行寻址；
- MEMRQ#，CPU 当前要读的是存储器（而非 I/O），低电平有效；
- RD#，低电平有效，有效时表明 CPU 当前执行的是读操作；
- WR#，低电平有效，有效时表明 CPU 当前执行的是写操作；
- DATA，数据总线，若 CPU 执行读存储器操作，则存储器将读出的数据送到数据总线上；若 CPU 向存储器进行写操作，CPU 将需要写入的数据送到数据总线上，并由写信号将数据写入被寻址的存储单元。

图 4.13 所示的 T1 周期，CPU 把需要读取的主存储器单元地址驱动到地址线上。地址信号对应多根信号线，习惯上用两条平行的线绘制，意为多条信号线中有些是低电平，有些是高电平。并且，地址信号变化的时刻用交叉的斜线表示，如图 4.13 所示，其中阴影部分表示该时段这部分信号无意义。

地址信号稳定后，CPU 发出 MEMRQ#和 RD#。MEMRQ#信号有效（低电平）表示 CPU 要访问的是主存储器（若为高电平表明访问 I/O 端口），RD#信号有效（低电平）表示 CPU 要执行读操作。由于被访问的存储器件收到地址信号之后，需要一段时间进行准备才能输出数据，所以约定存储器在 T2 时钟周期输出数据。

在 T2 周期，存储器输出数据到数据总线上，CPU 在 RD#信号后沿（上升沿）将总线上的数据存入内部寄存器，完成本次读操作，然后撤销地址信号和 MEMRQ#信号。

[3] 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。

[4] 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。

在时序逻辑电路中，经常需要在时钟信号的上升沿或者下降沿触发一些动作（如输出某些信号或读取某些信号）。如图 4.13 中 T1 周期时钟的上升沿和下降沿，以及 T2 周期时钟的上升沿和下降沿，这四个关键的时间点都有一些相应的动作被触发。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 4.13 中 T1 周期的时钟上升沿，CPU 输出地址到地址总线上，需要经过一小段时间地址信号才能稳定，这一小段的时间记为 T_{AD}，即 T_{AD} 是从 T1 的上升沿开始到地址建立好的时间，称作地址建立时间。由于在 T1 的下降沿需要输出 MEMRQ#和 RD#信号，假如 T_{AD} 大于半个时钟周期，就会影响到 MEMRQ#和 RD#信号的输出，所以对 T_{AD} 的最大值有一定约束。再如，在读操作的 T2 周期前半周期，存储器读出的内容送到数据总线上，T_{DS} 是数据稳定需要的时间。如果 T_{DS} 过长，那么在 T2 周期后半周期的读操作就会出现错误，所以对 T_{DS} 的长度也有限制。

需要说明的是，图 4.13 中的时序关系是一种简化版，用于说明总线操作的基本过程。在时序参数方面，仅介绍了 T_{AD} 和 T_{DS} 的物理意义，没有描述其他所需的时序参数。通常在电路设计时要仔细分析所用微处理器芯片与存储器芯片的时序特性，充分评估器件挂接到特定总线时是否能满足时序要求。

2. 异步总线时序

异步总线依靠收发双方约定的握手信号对传送过程进行控制。异步总线没有公共时钟，收发设备之间采用一问一答的（握手）方式进行协调。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 4.14 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主设备将拟访问的地址信号送上地址总线，同时发出主设备请求信号，并且延迟 D_1 后发出固定宽度的读命令；②从设备收到主设备请求之后，将所需数据读出并送往数据总线，同时发出从设备应答信号；③主设备在读信号后沿，将从设备输出的数据读入主设备内部寄存器。

图 4.14 右半部分所示为主设备向从设备写入数据的过程：①主设备将拟写入的单元地址以及需要写入的数据驱动至相应的总线，同时发出主设备请求信号，并且延迟 D_2 后发出固定宽度的写命令；②从设备收到主设备请求后准备接收数据，并在完成准备工作之后发出应答信号；③从设备利用主设备的写信号后沿，将总线上的数据锁入内部寄存器。

在图 4.14 所示的传送过程中，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自行撤销；主设备总是认为从模块收到请求后，一定会在规定的时间间隔内做出响应，主模块只需延迟 D_1 或 D_2 后发出读写命令即可。这种异步时序称作不互锁方式。显然，不互锁方式要求主设备和从设备的速度差异应在一定范围之内，否则将会导致数据传送错误。

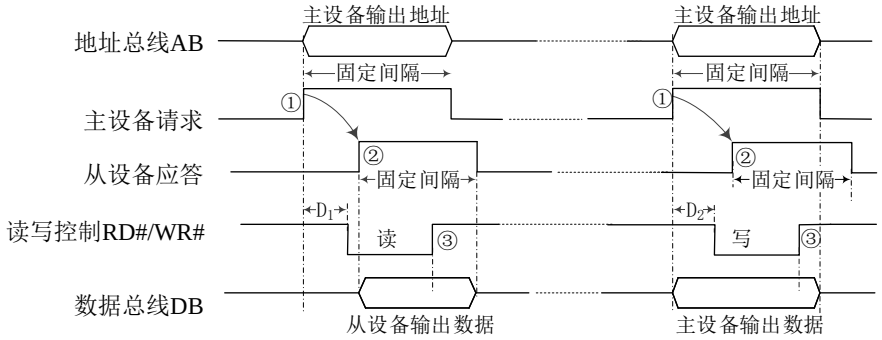


图 4.14　异步总线时序（不互锁方式）

图 4.15 所示为半互锁方式异步时序，为简单起见，图中没有画出读写信号线和数据总线。以主设备读取从设备数据为例，其基本流程为：①主设备发请求信号并等待从设备的应答；②从设备收到请求后输出数据，然后作出应答；③主设备收到从设备的应答后开始读数据，数据接收完毕才撤销请求信号。半互锁方式与不互锁方式的主要区别在于：主设备发出请求信号之后，必须等待从设备应答后才启动读数据操作，只有收到数据后才撤销请求。但是从设备输出数据和发出应答后，不关心主设备数据是否接收完毕，在间隔固定时间后，自行撤销数据和应答信号。

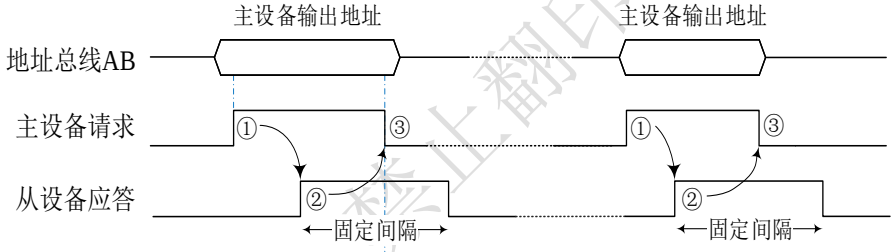


图 4.15　异步总线时序（半互锁方式）

图 4.16 所示是一种全互锁方式时序。图 4.16 左半部分是主设备读取从设备数据的过程：①主设备发出读请求；②从设备送出数据并发出应答；③主设备收到从设备的应答后开始读数据，并在数据读取结束之后撤销读请求；④从设备得知主设备撤销请求后，才停止驱动数据总线并撤销应答。与不互锁方式和半互锁方式对比，全互锁方式的主从双方都在确认对方状态之后才开始下一步操作，各个动作之间环环相扣，传输可靠性最高。图 4.16 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。

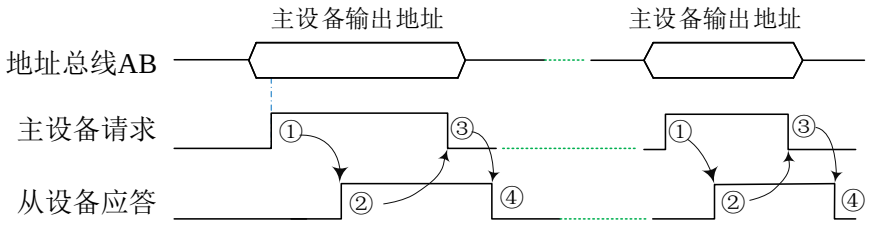


图 4.16　异步总线时序（全互锁方式）

异步总线通过预先约定的应答协议，可以保证两个速度差异很大的模块可靠地进行信息交换。但是握手过程需要额外的时间开销，异步总线的传输效率要低于同步总线。显然，握手过程越多，虽然传输愈加可靠，但是总线效率越低。异步总线常用于设备类型多或对系统可靠

性有较高要求的场合。

3. 半同步总线时序

在同步时序中，当主模块读取某个从模块数据时，如果从模块的速度较慢，从收到地址信息到准备好数据之间的时延超过一个时钟周期，就无法在图 4.13 所示的 T2 周期完成读操作。为此，可以在主模块和从模块之间增加一条信号线 READY/ $\overline{\text{WAIT}}$ /，当出现上述情形时，从模块通过 READY/ $\overline{\text{WAIT}}$ 向主模块提出请求，表明从模块尚未准备好，请主模块延长一个或者数个时钟周期，以便从模块能够跟上节奏，保证正确完成读数据操作。这种使用同步时钟，又能插入等待周期的时序称为半同步时序，如图 4.17 所示。

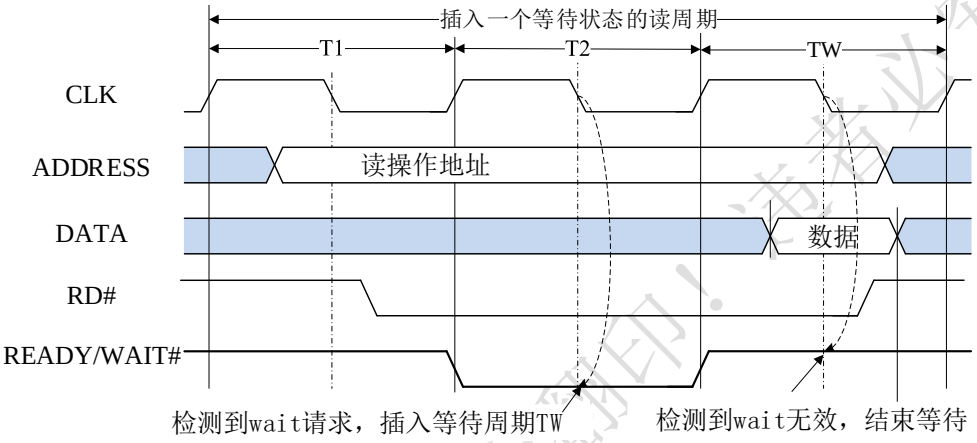


图 4.17 半同步总线时序

图 4.17 中，CPU 在 T2 周期的时钟下降沿检测 READY/ $\overline{\text{WAIT}}$ 信号线，如果发现是低电平，表明从模块没有准备好，CPU 则在 T2 周期之后插入一个等待周期 TW，把读操作延长一个节拍。CPU 在 TW 周期的时钟下降沿再次检测 READY/ $\overline{\text{WAIT}}$ 信号，如果是高电平，表明从模块已经准备好，无需继续等待，CPU 结束本次读操作（RD#信号回到高电平）。相比于图 4.13 所示同步时序，图 4.17 所示时序中 CPU 读到数据的时间滞后了一个时钟周期（READY/ $\overline{\text{WAIT}}$ 信号持续一个时钟周期）。在半同步总线时序中，CPU 插入等待时钟周期的数目根据 READY/ $\overline{\text{WAIT}}$ 信号的持续时间而定。

4. 周期分裂式总线时序

以上三种总线时序方式，在整个传输过程中总线一直处于被占用的状态。实际系统中，主模块发出地址和读/写命令后，从模块往往需要一些时间来准备数据。从模块准备数据期间，系统总线上并没有数据传输，处于空闲状态。

为了提高系统的总体性能，出现了周期分裂式总线时序。以读存储器操作为例，整个总线周期可以分解为两个子周期：寻址子周期和数据传送子周期。在寻址子周期，主模块发送地址、读命令和主模块识别码（主模块 ID 编号），由被寻址的从模块接收。随后，主模块释放总线，以便总线可以被其他主模块使用。待从模块准备好数据之后，由从模块发起总线使用申请，获准后启动数据传输子周期，从模块输出数据和主模块识别码，相关主模块接收数据。这种将总线周期进行分解的模式，减少了总线资源的无效占用，从而提高了总线的利用率。

在一些资料中，上述寻址子周期也称作地址阶段，数据传送子周期称作数据阶段，周期分

裂式操作又被称作“分离式操作”或“流水线分离”等。分裂式操作是高性能总线设计的一种重要思路，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，本书将在 4.2.2 小节以 AHB 总线为例，进一步分析这种周期分裂式操作的具体实现方式。

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芯片制造工艺，未定义电气特征，电气特性和准确的时序参数取决于芯片生产时采用的工艺和操作频率。AMBA 规范是一个开放标准，可免费获得。目前，AMBA 拥有众多第三方支持，在基于 ARM 处理器内核的 SoC 设计中，已获得广泛的应用。

随着 ARM 处理器的不断更新换代，AMBA 总线演进出多个版本，各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了针对多 CPU 核的优化。故本小节中主要结合 AMBA2 和 AMBA3 版本分析 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。

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） ☑ 多主机（multiple bus masters） ☑ 突发传输（burst transfers） ☑ 分裂式操作（split transactions）	☑ 高性能 ☑ 流水线 ☑ 多主机	☑ 低功耗 ☑ 地址锁存和控制 ☑ 接口简单

AMBA2 总线的典型互连结构如图 4.18 所示。AMBA2 总线基于一条高性能系统中枢总线（可采用 AHB 或 ASB），以实现 CPU、片上存储器、外部存储器及 DMA 设备之间的高速数据传输。这条高性能总线有一个桥接器，用来连接低功耗低速率的 APB，并由 APB 连接外设。

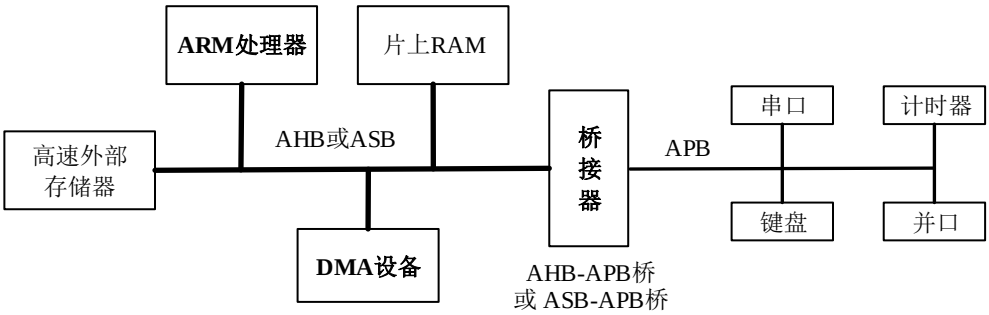


图 4.18　典型 AMBA2 总线的微控制器系统

为便于描述总线行为，AMBA 定义了信号名称的命名规则。这些规则包括：信号名称为大写字母；信号名称的第一个字母指示信号和哪个总线相关联；低电平有效的信号，其名称以小写字母 n 做后缀；测试信号加前缀 T；而 H、A、B、D、P 等前缀分别代表不同总线上的信号，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 HRESETn 表示 AHB 复位信号，低电平有效
A	ASB 信号，主机与仲裁器之间的单向信号
B	ASB 信号，如 BWRITE 表示 ASB 写操作指示，高电平有效
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

AHB 是支持多主机的总线，一个系统中最多允许有 16 个总线主机。AHB 总线通过一组多路选择器实现主、从设备的互连，其典型结构如图 4.19 所示。AHB 总线系统主要构成包括：主机（Master）、从机（Slave）、仲裁器（Arbiter）和译码器（Decoder）。

AHB 总线的数据传输由主机发起，数据传输操作所需要的地址和控制信号由主机产生。从机响应主机发起的操作，并将数据传输的状态信息反馈给主机。仲裁器确保任何时刻只有一个主机有效（即拥有总线使用权），并为地址和控制选择器以及写数据选择器提供主机选择信号。译码器通过解析地址信号为读数据选择器提供从机选择信号。

图 4.19 中，地址和控制选择器把当前有效主机的地址线连接至从机 HADDR；写数据选择器把当前有效主机的写数据总线连接至从机 HWDATA；读数据选择器把被选中从机的读数据总线连接至主机 HRDATA。

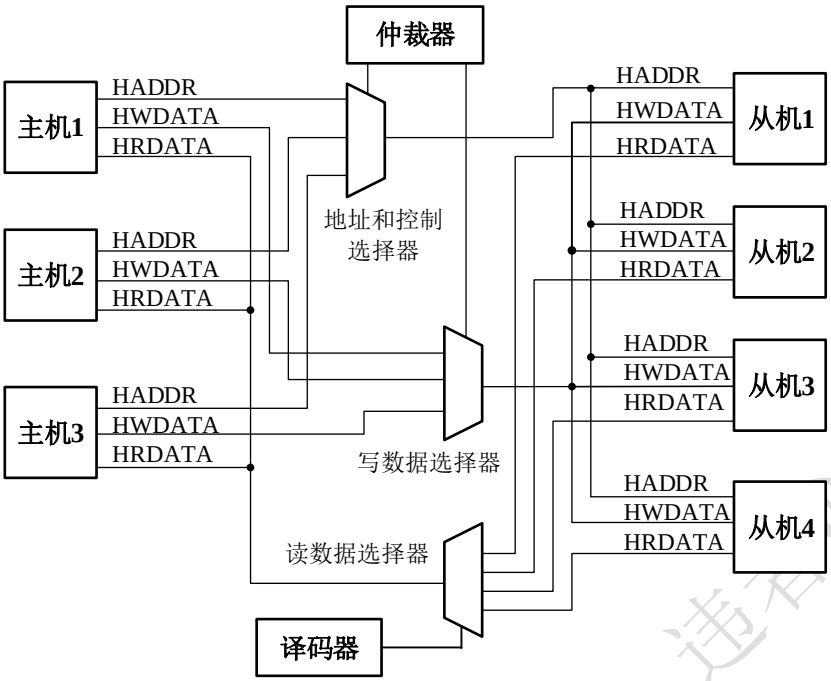


图 4.19　基于多路选择器的 AHB 总线互连结构

2. AHB 信号定义

AHB 信号定义如表 4.3 所示。为便于阅读，表 4.3 仅给出各信号的简要说明，详细信息可参阅 ARM 公司 AMBA2 标准。

表 4.3　AMBA2 中 AHB 的信号

信号名称	信号来源	用途说明
HCLK	时钟源	时钟，各信号时序与 HCLK 上升沿或下降沿相关
HRESTn	复位控制器	总线复位信号，低电平有效
HADDR[31:0]	主机	32 位系统地址总线
HTRANS[1:0]	主机	表示突发传输的类型，可以是不连续（NONSEQUENTIAL）、连续（SEQUENTIAL）、空闲（IDLE）或忙（BUSY）
HWRITE	主机	高电平表示是写传输，低电平表示是读传输。
HSIZE[2:0]	主机	传输宽度，可以是字节、半字或字等，允许最大传输 1024 位
HBURST[2:0]	主机	突发类型，表示传输是突发的一部分，支持突发长度为 4、8 或 16
HPROT[3:0]	主机	指示当前传输的安全保护级别
HWDATA[31:0]	主机	写数据总线，可扩展至更高位宽
HSELx	译码器	从机选择信号，HSELx 有效表示从机 x 被选中
HRDATA[31:0]	从机	读数据总线，可扩展至更高位宽
HREADY	从机	高电平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]	从机	传输响应，有四种可能：OKAY、ERROR、RETRY 或 SPLIT。

为支持多主机操作，AMBA AHB 中定义了多个仲裁相关的信号，如表 4.4 所示。信号后缀 x 表示模块 x，如，HBUSREQx 可能表示 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	主机	总线请求，从主机 x 送往仲裁器
HLOCKx	主机	表示主机 x 请求锁定对总线的访问

HGRANTx	仲裁器	指示主机 x 是当前优先级最高的主机
HMASTER[3:0]	仲裁器	主机号，用来区分不同的主机
HMASTLOCK	仲裁器	当前主机正在执行一个锁定序列（sequence）的传输
HSPLITx[15:0]	从机	分离式传输请求，指明由哪个主机重试一个分裂式传输操作

3. AHB 的数据传输过程及“流水线”

AHB 传输开始前，主机要先获得总线访问的授权。授权过程为：主机 x 通过 HBUSREQx 向仲裁器发出总线使用请求，随后仲裁器通过 HGRANTx 指示主机被授权。主机获得授权后开始驱动地址信号和控制信号（传输方向、传输宽度和传输类型等）。

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；而数据阶段可持续一个或者多个时钟周期，用来传输有效数据。图 4.20 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 4.20 中并未画出 HWRITE 信号，即图中未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]。

首先分析主机读取从机数据的传输过程。图 4.20 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，从机采样获得地址（图示为 A）和控制信息，并把此地址（A）对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]上采样获得从机响应的数据。

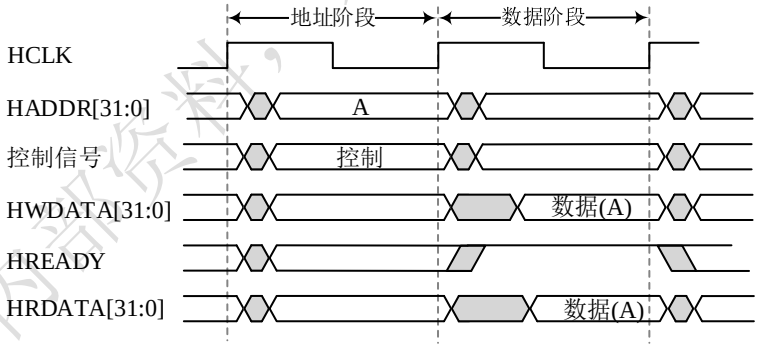


图 4.20　简单传输（无等待状态）时的 AHB 传输时序

写传输过程与读传输过程不同。①主机在地址阶段把地址信息 A 驱动到了地址总线上，②而从机在下一个时钟周期（图 4.20 中的数据阶段）时钟的上升沿采样获取地址和控制信息，③随后的下一个时钟周期（图 4.20 中数据阶段结束的下一个时钟周期）时钟的上升沿从机采样数据总线 HWDATA[31:0]获取数据。

实际上，传输过程中地址信息的更新和数据的更新在节拍上是错开的，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被驱动到 HADDR[31:0]上了，而本次 AHB 传输的数据更新至 HRDATA[31:0]（读传输）之后，在下一次 AHB 传输开始的第一个时钟周期才被读取。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。在稍后将要讨论的图 4.22 中，可以更清晰地观察到这种交叠。

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，为了简化电路不支持这种流水线机制。

2） 在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 4.21 是数据阶段被扩展（即插入了等待周期）的 AHB 传输时序。

以图 4.21 中读传输为例，主机在 HCLK 上升沿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。因从机在数据阶段的第一个时钟周期没能准备好，故把 HREADY 拉为低电平，从而插入了等待周期（图 4.21 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，并将数据驱动至 HRDATA[31:0]。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。读传输和写传输过程是有差异的：在写操作过程中，主机必须确保数据在整个扩展期内稳定；而在读传输过程中，从机不必一直确保数据有效，只需要在相应周期提供数据即可。

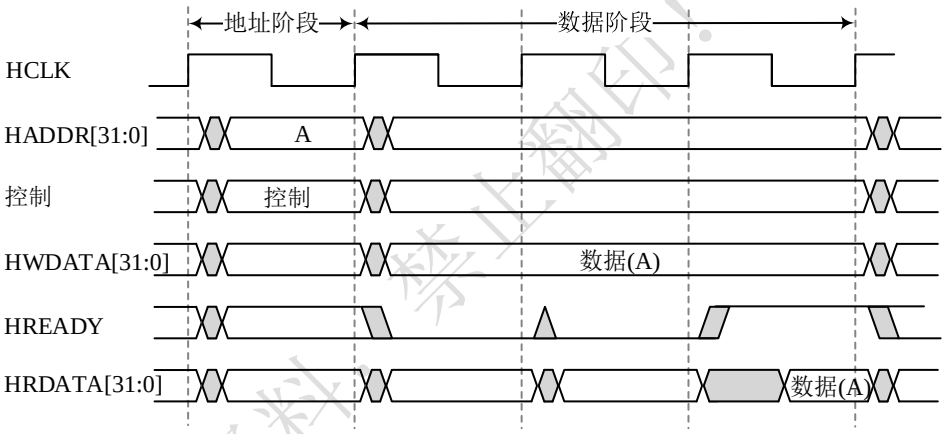


图 4.21　有等待状态的 AHB 传输时序

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 传输的地址阶段也被扩展，如图 4.21 中 HADDR[31:0]所示。

3） 多个数据的传输

图 4.22 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。

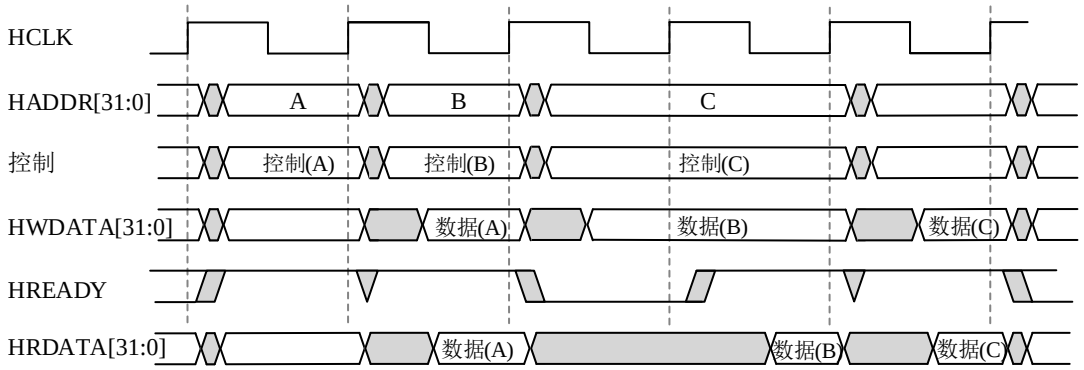


图 4.22　多个数据的 AHB 传输时序

图 4.22 中，传输地址 A 和地址 C 数据都没有插入等待周期，但传输地址 B 数据时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） 流水线“分离”

表 4.4 中 HMASTER[3:0]、HMASTLOCK、HSPLIT[15:0]信号用于支持名为“SPLIT”的传输模式。这种传输模式在 ARM 公司的资料中，被称作分裂式操作（Split transactions）；其它一些资料中也称作流水线分离。

SPLIT 传输，指 AHB 传输的两个阶段分离（地址阶段和数据阶段可以被分离，参见 4.1.4 小节）。根据前述 AHB“流水线”机制可知，当前 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，能够被从机及时读取；主机驱动数据到数据总线 HWDATA[31:0]上后，也要能够被从机及时读取。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。为了应对这种从机不能及时响应的情形，出现了流水线分离设计。

从机接收了主机发出的地址和控制信息后，如果不能在下一个时钟周期响应，可通过 HRESP[1:0]发出启动 SPLIT 传输的响应。仲裁器检测 HRESP[1:0]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]发出重新启动传输信号，仲裁器根据挂起操作主机的优先级，适时重新分配总线使用权。当主机再次获得总线使用权后，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。从机发出 SPLIT 响应时，需要记录当前的主机号。从机准备好数据后，依据所记录的主机号决定 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。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. AHB 的突发传输

图 4.20、图 4.21 和图 4.22 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。简单地说，突发传输就是在一次传输过程连续传输一个数据块。既然是连续传输一个数据块，就涉及到数据块的长度和连续传送时的地址改变方式等。表 4.5 中定义了不同突发传输类型下的突发长度和地址改变方式。数据块长度根据突发长度和传输宽度（HSIZE[2:0]）计算，如，HSIZE[2:0]==010B 表示传输宽度是 32 比特，此时 WRAP4 类型的突发传输的数据块大小是 128 比特（16 字节）。

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]获知下一步的操作提示。除此之外，协议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。

下面通过两个例子，说明表 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 4.23 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，指示传输的数据是突发传输中的第 1 次。T2 周期，第 2 个传输数据的地址 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 个传输数据被推送到数据总线 HWDATA[31:0]上。T4 周期，第 3 个传输数据的地址 0x30 被推送到地址总线上，同时，第 2 个传输数据被推送到数据总线上。T5 周期，第 4 个传输数据的地址 0x34 被推送到地址总线上，同时，第 3 个传输数据被推送到数据总线上。T6 周期，第 4 个传输数据被推送到数据总线上。

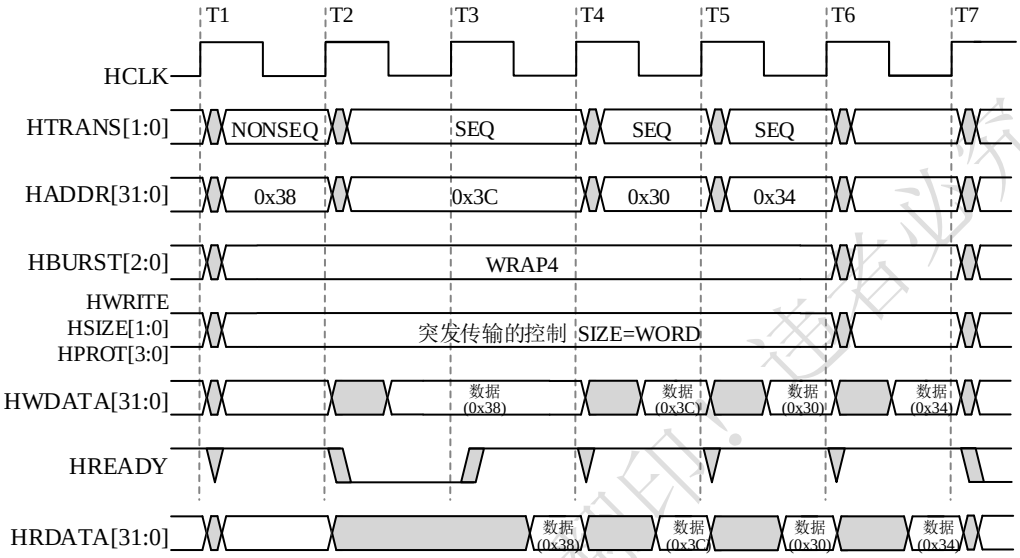


图 4.23　WRAP4（Four-beat wrapping burst）突发传输时序

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 4 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 4 位计数器的递增操作。与图 4.23 的 WRAP4 类型不同，在图 4.24 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。

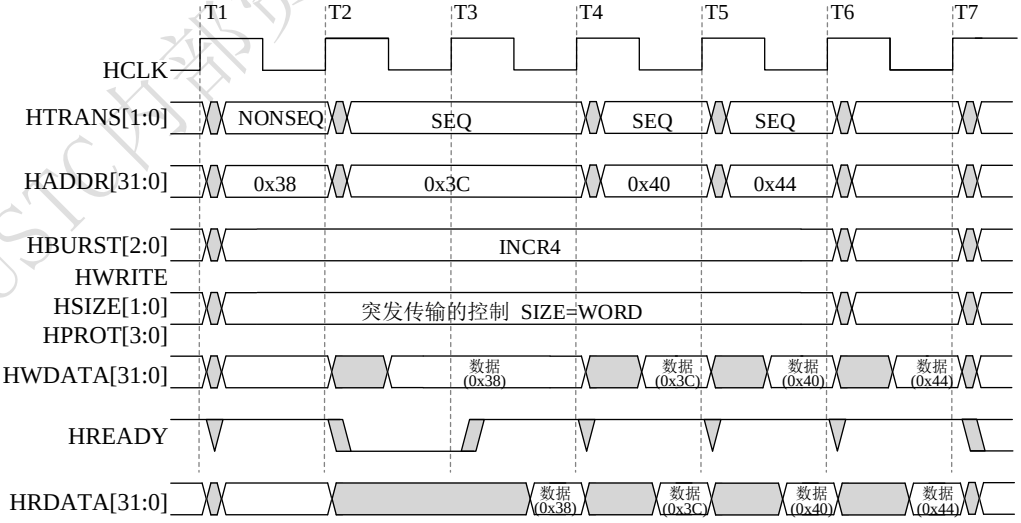


图 4.24　INCR4（Four-beat incrementing burst）突发传输时序

需要指出的是，AMBA 规范中规定突发传输时传输数据块不能跨越 1kB 的边界。

5. AHB 译码器

图 4.19 中，AHB 总线通过一组多路选择器实现主、从设备的互连，采用一个集中式译码器，译码器可以生成从机选择信号 HSELx。从机选择信号 HSELx 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，从而获得地址和控制信息。地址译码和从机选择信号如图 4.25 所示。

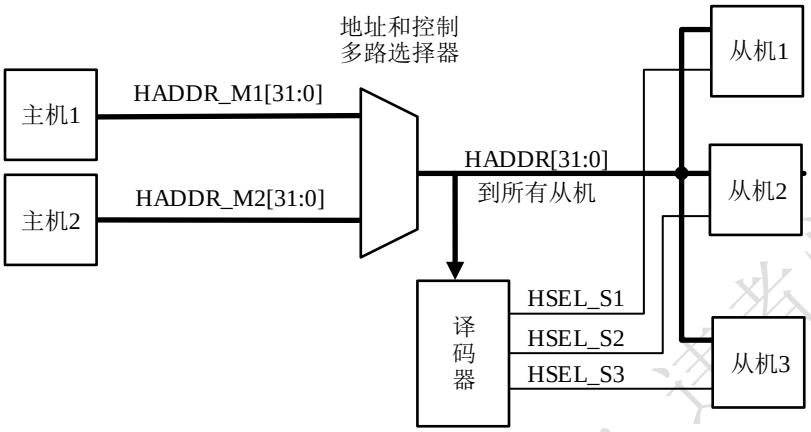


图 4.25　地址译码和从机选择信号

由于采用了集中式的译码器，每个主机可在需要时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。如图 4.26 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]，地址和控制多路选择器把主机号对应主机的地址总线与从机的地址总线连接。

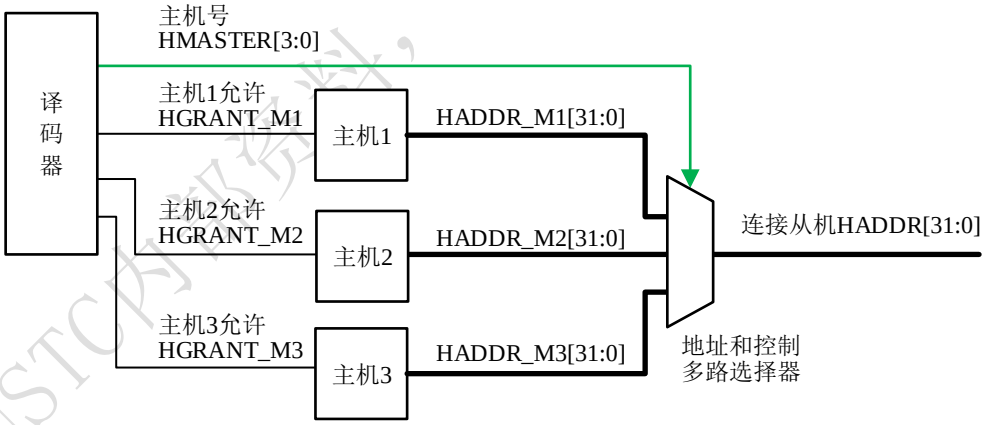


图 4.26　总线允许信号与主机号的生成

6. AHB 仲裁器

表 4.4 中，AMBA2 的 AHB 中定义了一些专门的信号用于仲裁。仲裁机制是为了保证在同一时刻，只有一个主机控制总线。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，则需要同时发出 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]以指示当前获得总线使用权的主机号。

图 4.27 所示即为 HREADY 有效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；再下一个周期即 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]。

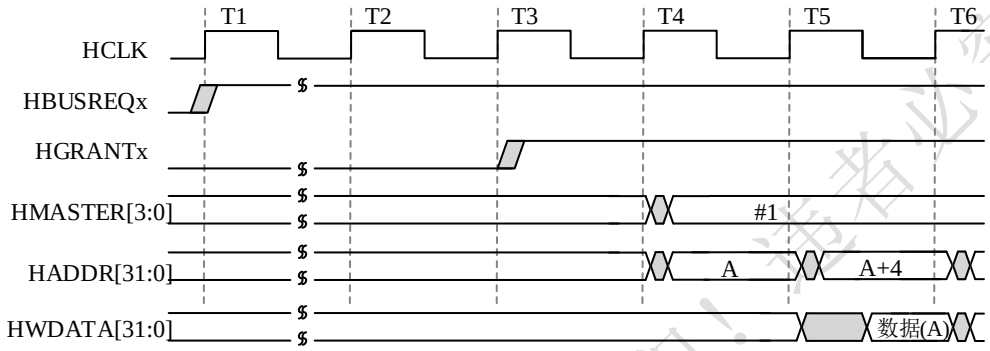


图 4.27　无等待状态的仲裁授予

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 4.28 所示，由于 T3 周期 HREADY 无效，T4 周期时钟上升沿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。T6 周期时钟上升沿的时候，检测到 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。

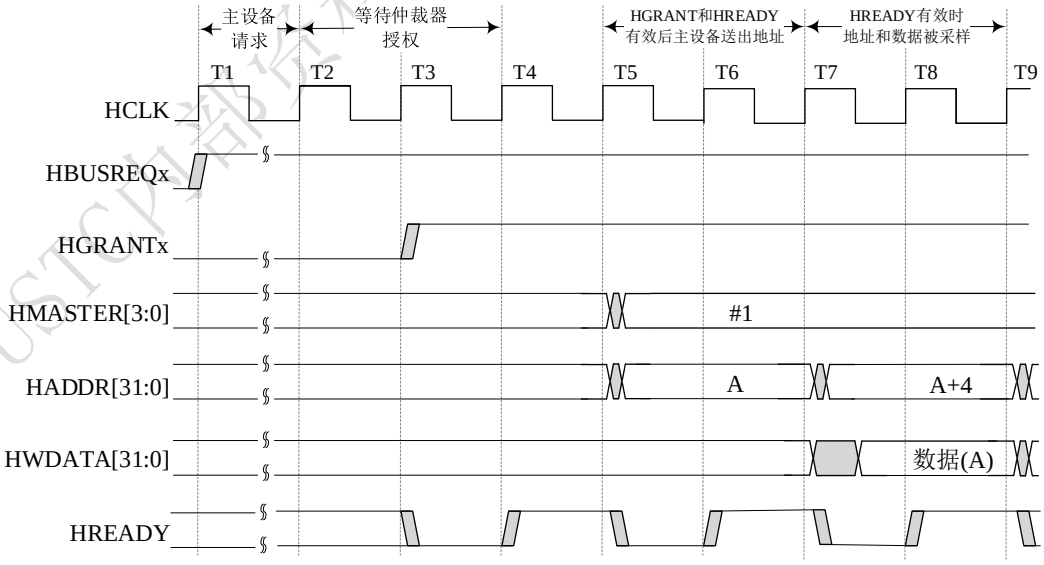


图 4.28　有等待状态的总线仲裁和授予

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。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 4.29 所示为总线控制权在两个主机间的移交过程，从图中可看出主机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。

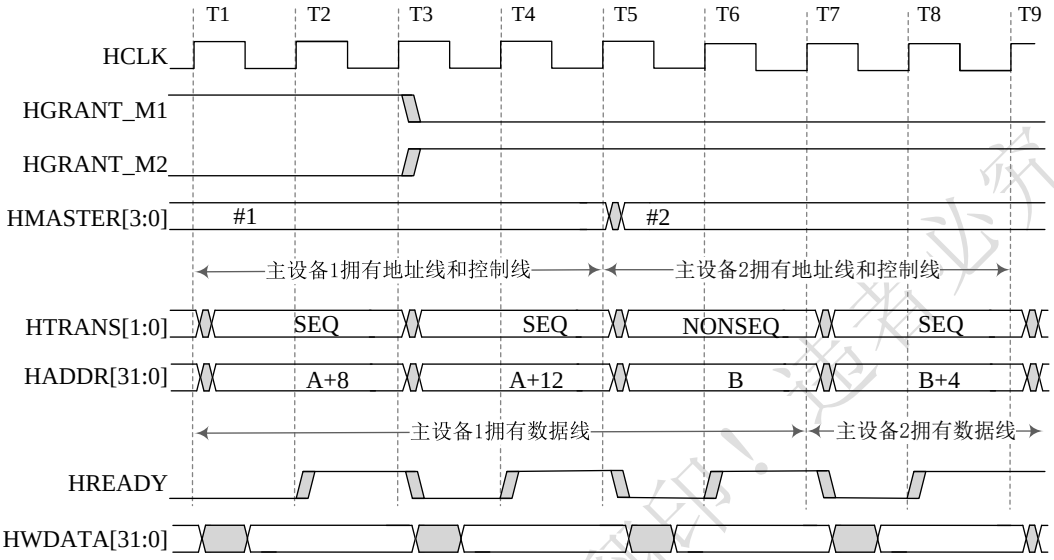


图 4.29　总线控制权在两个主机间的移交

图 4.30 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（HGRANT_M1 失效，HGRANT_M2 有效）。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。

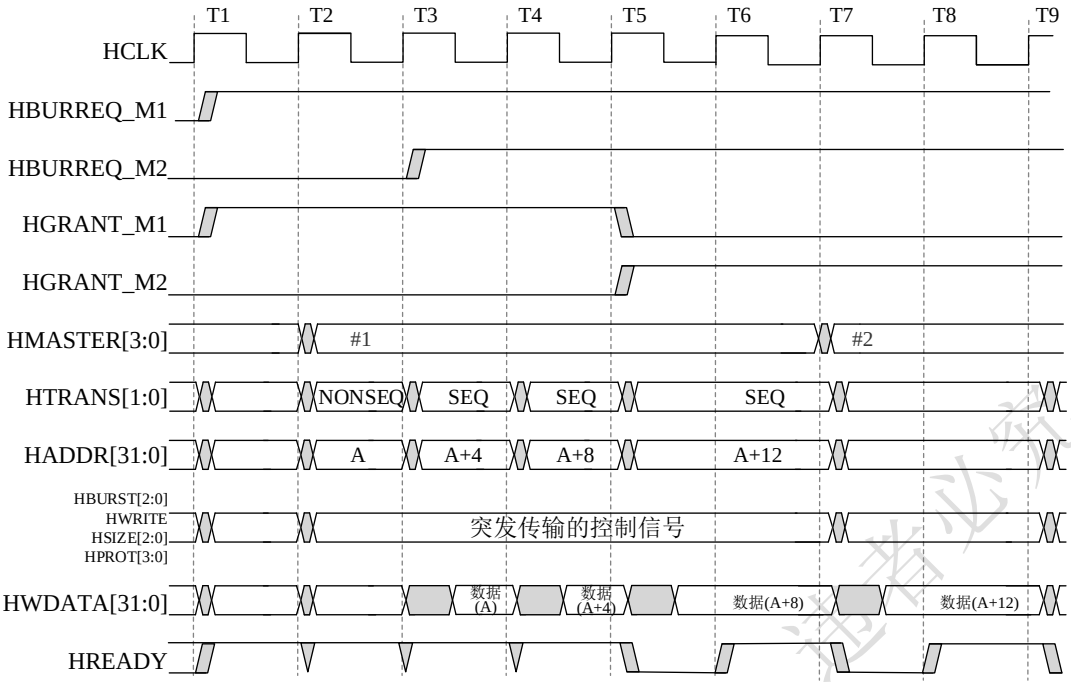


图 4.30　突发传输后的总线移交

4.2.3 ASB 总线

图 4.18 中的 ASB 也可以作为微控制器芯片中的主总线，其系统结构与工作原理均与 AHB 类似，但不支持突发传输功能（如表 4.1 所示）。AHB 系统构成包括：ASB 主机、ASB 从机、ASB 译码器以及 ASB 仲裁器。

表 4.7 所示为 AMBA2 定义的 ASB 信号列表。由于 ASB 功能可视为 AHB 功能的子集，故本小节不再对 ASB 的传输过程做解析。

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，AGNTx 有效表示主机 x 被授权使用总线
AREQx	总线请求，AREQx 有效表示主机 x 请求使用总线
BA[31:0]	系统地址总线，由总线主机驱动
BCLK	时钟信号，为总线传输提供时基
BD[31:0]	数据总线，写传输时由主机驱动，读传输时由从机驱动
BERROR	指示发生传输错误
BLAST	末尾指示，表示当前是突发传输的最后一个数据
BLOK	指示当前传输和下一个传输不可分割，总线是独占的
BnRES	低电平时，复位系统和总线
BPROT[1:0]	指示当前传输的安全保护级别
BSIZE[1:0]	指示传输的大小（字节、半字或字）
BTRAN[1:0]	指示下一次传输的类型
BWAIT	等待指示，高电平表示需要插入等待周期
BWRITE	高电平表示写传输，低电平表示读传输
DSELx	DSELx 有效表示从机 x 被选中

4.2.4 APB 总线

APB 是为低速率外设提供的一个低成本接口。为了降低功耗和接口复杂度，APB 不支持流水线功能。不同版本 APB 演进如表 4.8 所示。从了解设计思想角度看，在 AMBA2 版本中定义的 APB2 已经具备核心设计思路，故本小节的学习以 AMBA2 的 APB2 版本为主。

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，行为类似 AHB 或 ASB 的外设，通过 APB 桥接器（APB bridge）连接 AHB 或 ASB。图 4.18 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，如表 4.9 所示。

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

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	时钟源	总线时钟，仅使用上升沿作为 APB 时基	AMBA2
PRESENTn	系统总线	复位信号，低电平有效	AMBA2
PADDR[31:0]	APB 桥	地址总线	AMBA2
PSELx	APB 桥	PSELx 有效指示从机 x 被选中	AMBA2
PENABLE	APB 桥	使能信号，指示 APB 传输的第二个周期	AMBA2
PWRITE	APB 桥	高/低电平分别表示 APB 写/读	AMBA2
PRDATA	从机	读数据总线	AMBA2
PWDATA	APB 桥	写数据总线	AMBA2
PPROT	APB 桥	总线访问的附加安全信息	AMBA4
PSTRB	从机	写选通信号，指示要进行数据更新	AMBA4
PREADY	从机	就绪信号，低电平表示从机拟扩展传输	AMBA4
PSLVERR	从机	错误指示	AMBA4

图 4.31 所示为 APB 写传输总线时序，PWRITE 为高电平，使用 PWDATA 传输数据。图 4.32 所示为 APB 读传输总线时序，PWRITE 为低电平，使用 PRDATA 传输数据。

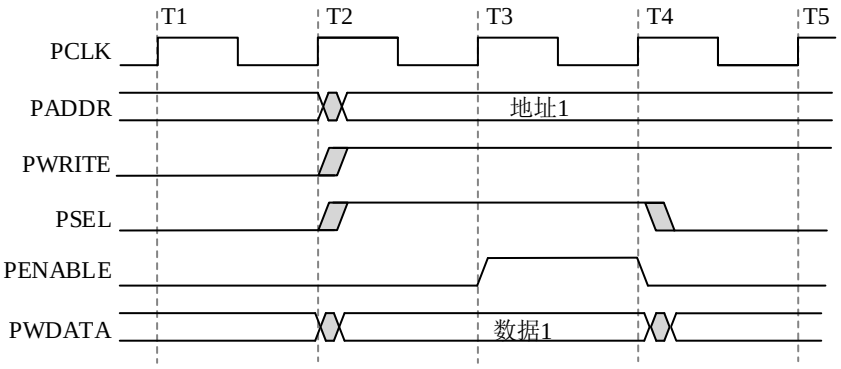


图 4.31　APB 写传输时序

APB 时序非常简单。可以用三个操作状态来描述总线上的不同阶段：空闲阶段（IDLE，可持续多个周期）、设置阶段（SETUP，持续单个周期）和使能阶段（ENABLE，持续单个周期）。

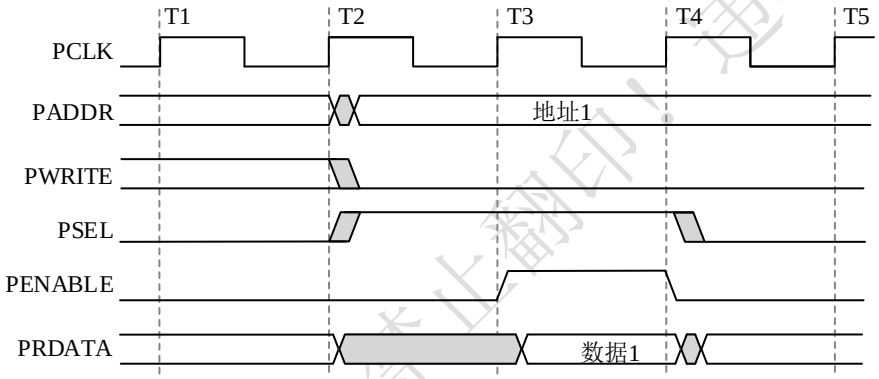


图 4.32　APB 读传输时序

时钟上升沿到来后，地址信号 PADDR、使能信号 PENABLE、选择信号 PSEL 和读写数据方向信号 PWRITE 全部改变，读写传输的开始。传输的第一个时钟周期处于设置阶段，在下一个时钟上升沿 PENABLE 生效，进入使能阶段。地址、数据和控制信号在整个使能阶段保持有效。传输结束后进入空闲阶段，此时 PENABLE、PSEL 均变成低电平，为了降低功耗，PADDR 和 PWRITE 在传输之后不再改变，直到下一次传输开始。

4.2.5 AXI 总线

2001 年，ARM 发布了 AMBA3 规范，引入了 AXI（Advanced eXtensible Interface）总线。AHB 总线分离了一个总线周期的地址阶段和数据阶段，便于实现流水线和分裂式操作。同时，AHB 中地址、读数据和写数据各是独立的总线，即有三个独立通道。AXI 总线进一步分离了总线的通道，形成五个独立的通道：读地址（read address）通道、读数据（read data）通道、写地址（write address）通道、写数据（write data）通道和写响应（write response）通道，进一步提升了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	读写地址通道独立 读写数据通道独立 写响应通道	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 支持突发传输 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 支持突发传输 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

随后，在 2010 年发布的 AMBA4 总线规范中，进一步强化了多层结构，将 AXI 总线细分为 AXI4、AXI4-Lite 和 AXI4-Stream。其中 AXI4-stream 由 ARM 公司和 Xilinx 公司共同提出，主要用于 FPGA 器件的高速数据传输。目前 AXI 总线已取得广泛应用。Xilinx（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连，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，这个外部接口可用于测试。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。2001 年，AMBA V3.0 发布，引入 ATB（Advacned Trace Bus）和 AXI 总线。

随着多核处理器的普及，又给总线技术提出了很多新的要求。2010 年，ARM 推出的 AMBA4 中，引入了 QoS 机制，进一步增强了多层结构。AMBA4 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）间实现缓存一致性。Cortex A15 正是藉此超越了嵌入式领域的其他处理器。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，采用了基于数据包的层次化协议。这种全新的设计为集成不同类型的计算单元（如 GPU、DSP、FPGA 等）和不同的 I/O 子系统提供了便利。

目前，ARM® AMBA® 协议已成为 SoC 功能模块互连的开放标准，极大地推动了片上互连规范的发展。截至目前（2019 年），AMBA 定义了 CHI™、ACE™、AXI™、AHB™、APB™ 和 ATB™ 等一系列规范。不同 AMBA 版本演进过程如图 4.33 所示。

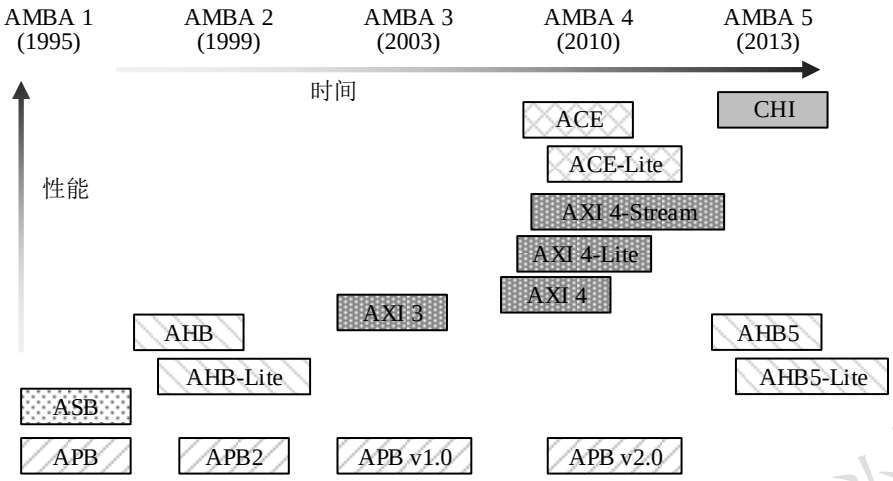


图 4.33　AMBA 版本演进图

这些不同系列的规范特点各异，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 则用于服务器处理器等高性能场景。不同子系列的全称及发布时间如表 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	高传输速率的片上互连	AMBA 2	AMBA 2
AHB-lite		仅支持单主机的简化版 AHB	AMBA 3	AMBA 3
AXI	Advanced Extensible interface	高带宽低延时的片上互连	AMBA 3	AMBA 4
ATB	Advanced Trace Bus	用于芯片间传输调试跟踪数据	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 系统总线和外部总线

在计算机技术发展的过程中，出现过很多总线标准。例如，系统总线有 ISA、EISA、VL、PCI 和 PCIe 等；用于外部存储设备的总线有 IDE、ATA、SCSI、PCMCIA、SATA 和 SAS 等；外部总线有 LPT、USB、I^2C、RS-232、RS-422、RS-485、IEEE-488 和 IEEE-1394 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 PCI、PCIe 和 USB。

PCI 是一种外部设备互连总线，在微机系统中的使用历史曾达 20 年之久，直至目前，其紧凑型版本 Compact PCI（CPCI）在工业控制领域仍有广泛应用。PCI Express 是 PCI 升级后的新一代总线标准，目前不仅用作计算机系统的内总线，也被用于开发新的外部总线。异步串行总线 USB 则是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
系统总线	ISA	Industry Standard Architecture
系统总线	PCI	Personal Component Interconnect
系统总线	EISA	Extended ISA
系统总线	MCA	Micro-Channel Architecture
系统总线	AGP	Accelerated Graphics Port
系统总线	VL	Video Electronics Standards Association Local Bus
存储总线	ATA	Advanced Technology Attachment
存储总线	IDE	Integrated Drive Electronics (ATA)
存储总线	SCSI	Small Computer Systems Interface
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	LPT (Line Print Terminal)
外部总线	Serial	RS-232、RS-422/RS-423、RS-485
外部总线	USB	Universal Serial Bus
外部总线	IEEE-488	General Purpose Interface Bus (GPIB)

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 1991 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，成立了 PCI-SIG（PCI Special Interest Group，PCI 特别兴趣工作组）。

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 版本为 64 位，总线时钟支持 33/66MHz，能够达到最高 528MB/s 的数据传输速率。过去有很长一段时间 PC 采用 PCI 作系统总线，基于 PCI 总线的系统结构如图 4.34 所示。图中，PCI 总线是一个以 HOST 主桥为根的树型结构，并且独立于 CPU 总线，可以和 CPU 总线并行操作。

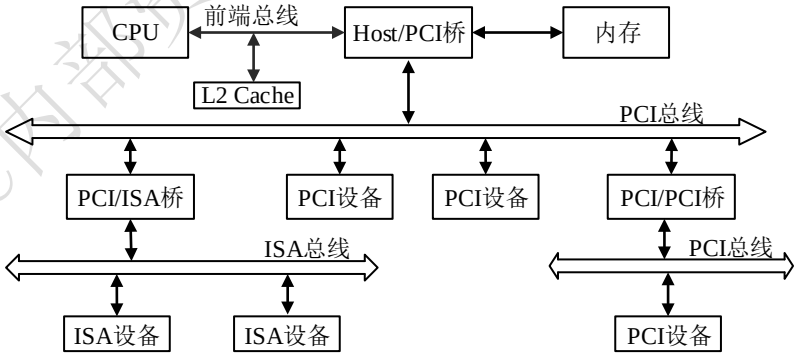


图 4.34　PCI 总线结构图

PCI 总线中有三类设备：PCI 主设备、PCI 从设备和桥设备。PCI 从设备只能被动地接收来自 HOST 主桥或 PCI 主设备的读写请求。PCI 主设备可以通过总线仲裁获得 PCI 总线的使用权，主动地向其他 PCI 设备发起读写请求。桥设备的主要作用是管理下游的 PCI 总线，并转发上下游总线之间的总线事务。

4.3.2 PCI Express

2001 年春，Intel 公司提出了要用新一代技术取代 PCI 总线，并称之为第三代 I/O 总线技术。随后 Intel 联合了 AMD、DELL 和 IBM 等 20 多家公司，在 2001 年底开始起草新的技术规范，并于 2002 年完成。新的总线技术被命名为 PCI Express，简称 PCI-E 或 PCIe。PCIe 标准由 PCI-SIG 组织负责维护和升级。

相比于 PCI，PCIe 最大的改变是将并行通信方式改为串行，并使用差分信号传输和点对点连接。点对点连接意味将基于总线的结构改为面向交换器的结构，每一个 PCIe 设备都拥有自己独立的数据连接，通过交换器实现设备之间的两两互连，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，总线上只能有一对设备通信，随着设备增多，每个设备的实际传输速率就会下降。图 4.35 显示了 PCIe 点对点串行连接拓扑结构。

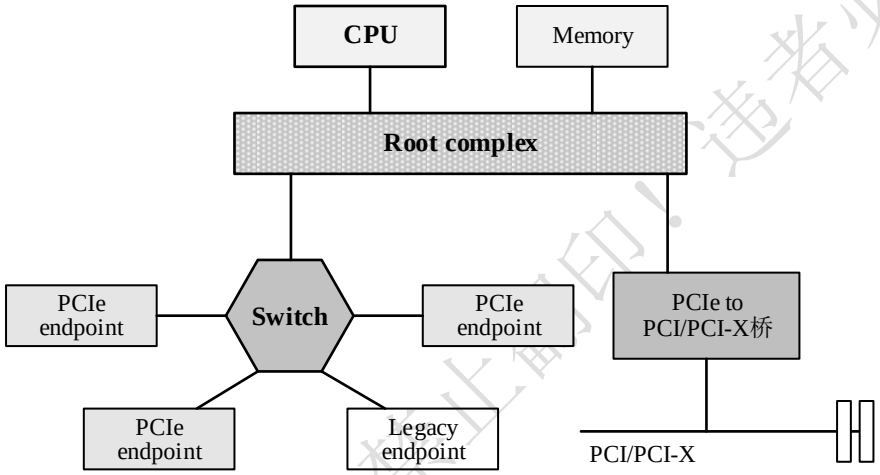


图 4.35　PCIe 架构示意图

图 4.35 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，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 4.36 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道。

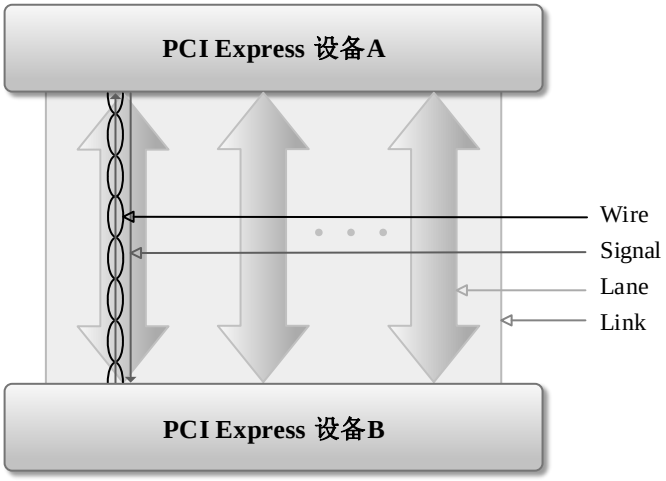


图 4.36　PCIe 链路及其包含的通道

链路的传输速率等于通道数目乘以单个通道的传输速率，PCIe 可通过增加通道来提高整个链路的传输速率。在 PCIe 1.0 规范中，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 PCIe 扩展卡（X1）可以插入多通道的插槽（如 X16、X8 等）。

近年来，PCIe 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	编码方式	传输速率				
			×1	×2	×4	×8	×16
1.0	2003	8b/10b	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	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	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.5 GB/s
5.0	2019	128b/130b	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 比特需要传输的数据编码为 10 比特。

4.3.3 USB

USB（Universal Serial Bus，通用串行总线）是一种外部总线标准。USB 最初在 1994 年底由 Compaq、DEC、IBM、Intel，Microsoft、NEC 和 Nothern Telecom 等七家公司联合提出的。USB 采用差分方式传输，如图 4.37 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实现更高速的数据传输。

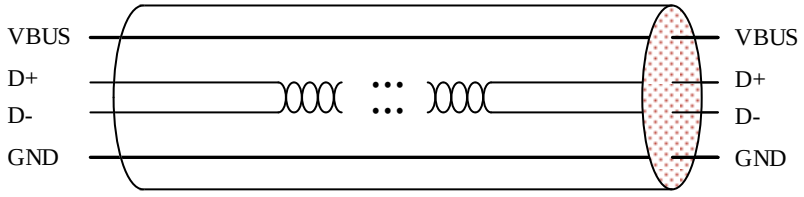


图 4.37　USB1.0/2.0 连接线缆的信号线定义

2013 年 7 月份，USB 3.1 发布，速度翻番至 10Gbps。但此时 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 4.15 为 USB1.0 ~ USB3.2 几个主要版本的特性比较。

根据最新公布的规范，USB 3.0 和 USB 3.1 的版本名称将彻底消失，统一被划入 USB 3.2 的序列，三者分别再次改名为 USB 3.2 Gen 1、USB 3.2 Gen 2 和 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 宣布了 USB4 的开发计划，USB4 将在兼容 USB3.2 的基础上增加对 Thunderbolt 的支持。截止 2019 年 6 月，USB4 的开发时间表尚未公布。

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简介

1. 以 8051 为核心的嵌入式控制器的总线

8051 是八位微控制器，属于 MCS-51 单片机的一种，由 Intel 公司于 1981 年设计。8051 采用典型的单总线结构，如图 4.38 所示。

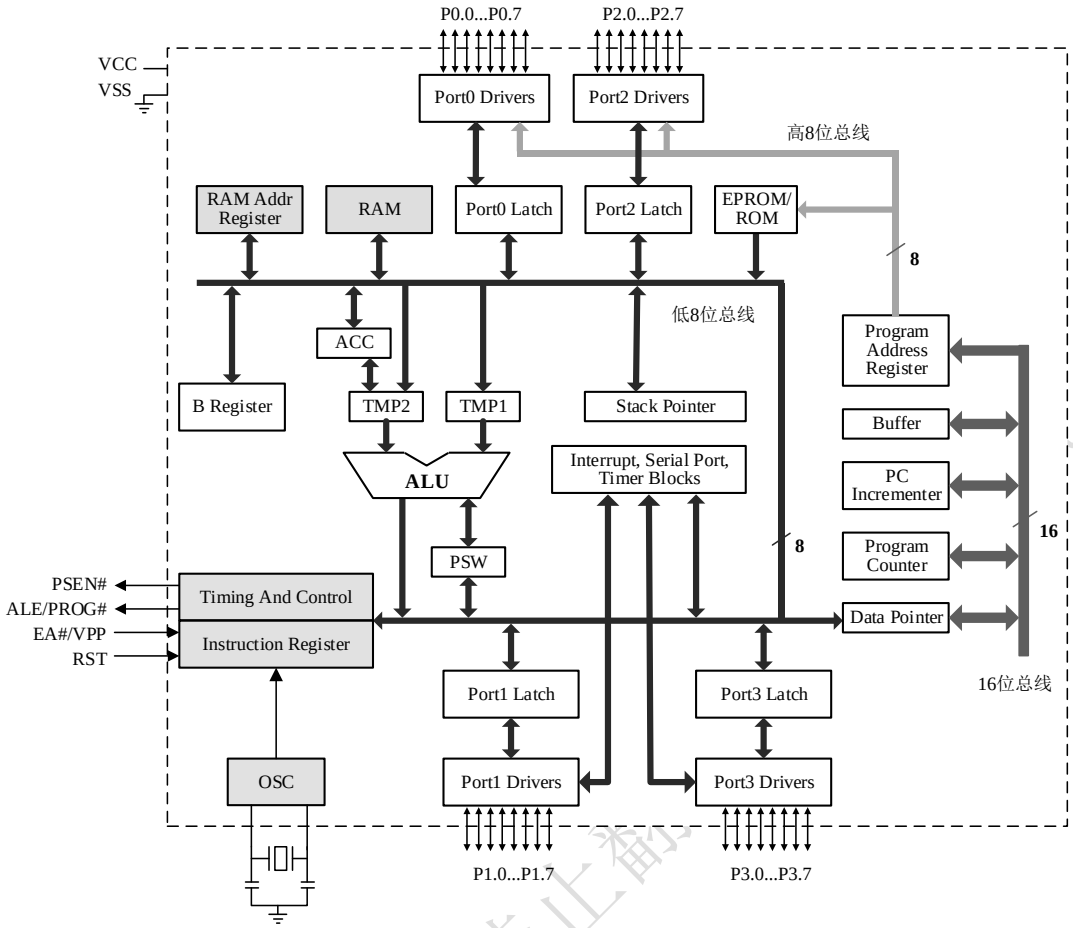


图 4.38　8051 系列微控制器的总线系统

2. 基于 Cortex-M3 内核的嵌入式控制器总线

意法半导体的 STM 系列微控制器基于 ARM 处理器内核设计。以下以 STM32F10 系列微控制器为例，分析其内部的多总线结构。图 4.39 所示的 Cortex-M3 内核通过 D-Code 总线访问数据存储器 SRAM，通过 I-Code 总线访问代码存储器 Flash，同时还拥有一条系统总线。图 4.39 两个 DMA 控制器通过 DMA 总线以及总线矩阵与系统总线相连。外设模块挂接在两条 APB 总线上，APB 总线通过桥接器连接至 AHB。关于 AHB 总线、APB 总线的细节信息参见 4.2 节。

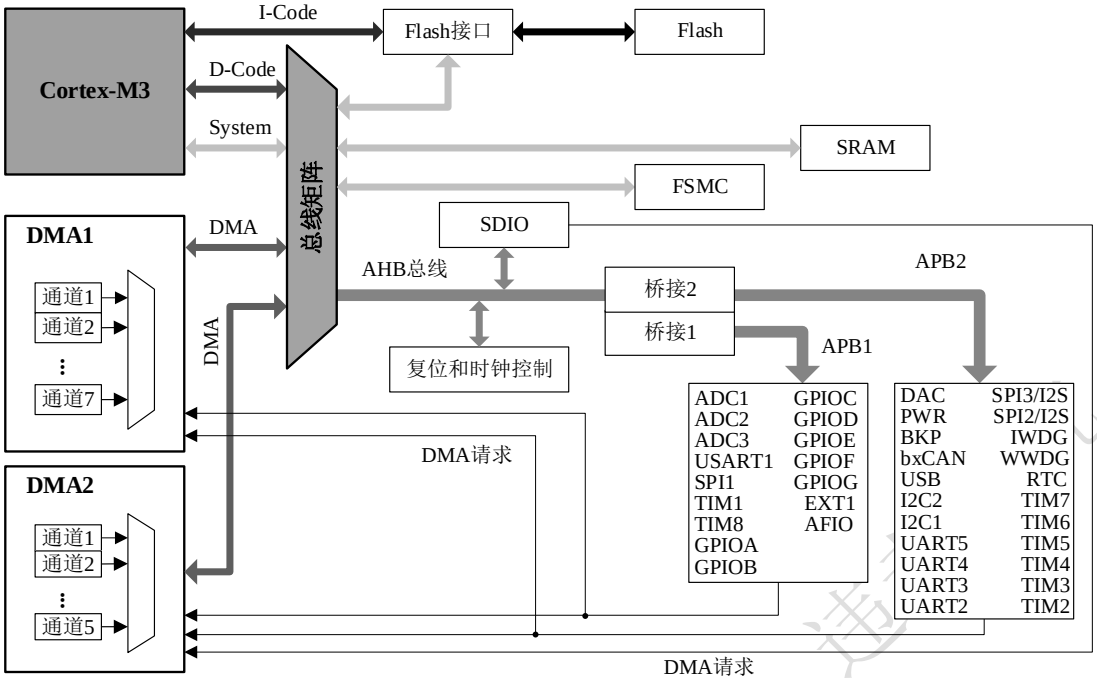


图 4.39　STM32F10 系列嵌入式控制器的多总线系统

3. 基于 Intel 8086 的 PC 机 IO Channel

Intel 在二十世纪 70 年代设计的 8086 微处理器，是 x86 架构处理器的鼻祖，其结构请参见本书 2.6.1 小节。Intel 8086 有 16 根数据线和 20 根地址线，属于 16 位微处理器。稍后 IBM 以 Intel 8088（Intel 8086 的准 16 位版）作为 CPU，设计和生产了第一代 IBM PC 机。Intel 8086 和 Intel 8088 均有最大和最小两种工作模式。所谓最小模式是所有的总线命令和总线控制信号均由 CPU 产生；而最大模式是 CPU 只输出表示下一阶段总线操作类型的状态信号，由外部独立的总线控制器根据状态信号产生相应的总线命令和总线控制信号。

基于 8086 微处理器最大模式构建的计算机系统如图 4.40 所示。图 4.40 中的 Intel 8288 就是为最大模式配套的总线控制器，8284A 为时钟发生器，8259A 为中断控制器，8286 为 8 位数据总线双向驱动器（等同于 74xx245），8288 为 8D 锁存器（等同于 74xx373）。该系统属于一种简单的单总线结构。

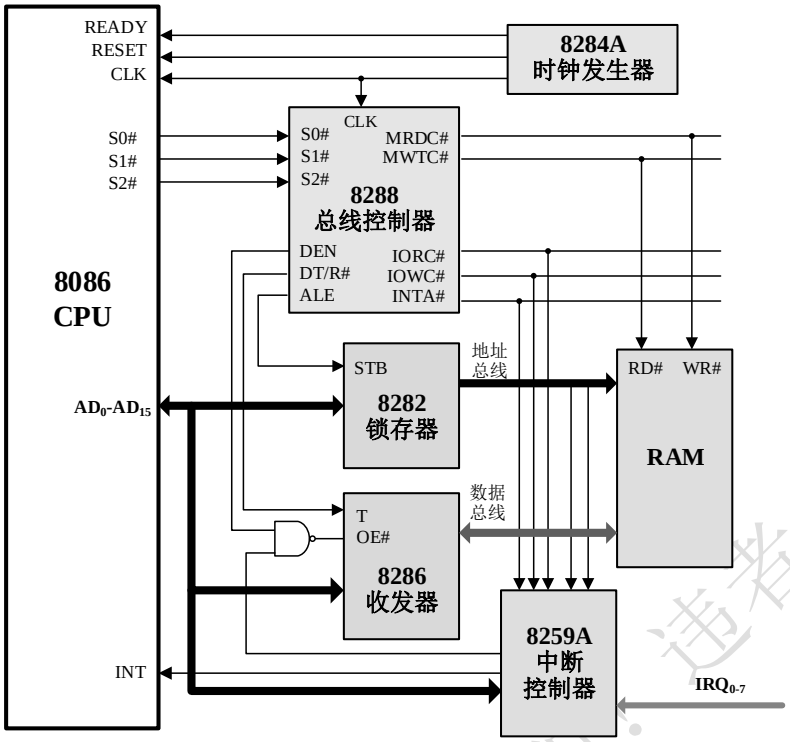


图 4.40　8086 系统最大模式的总线系统

4. 基于 x86 和前端总线的多总线结构

基于 x86 和前端总线的计算机系统中，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 4.41 所示。图中的北桥（north bridge）芯片负责连接内存和显卡等数据吞吐量大的部件。北桥芯片主要包括内存控制器、图形接口控制器、FSB 控制器和南北桥总线控制器。南桥芯片负责连接外设，早期只包括 ISA、PCI、ATA、键盘、鼠标等接口，后来又集成了音频和网络。

从图 4.41 中可以看出，FSB 是 CPU 和外界交换数据的最主要通道，因此 FSB 传输能力对计算机性能影响很大。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。

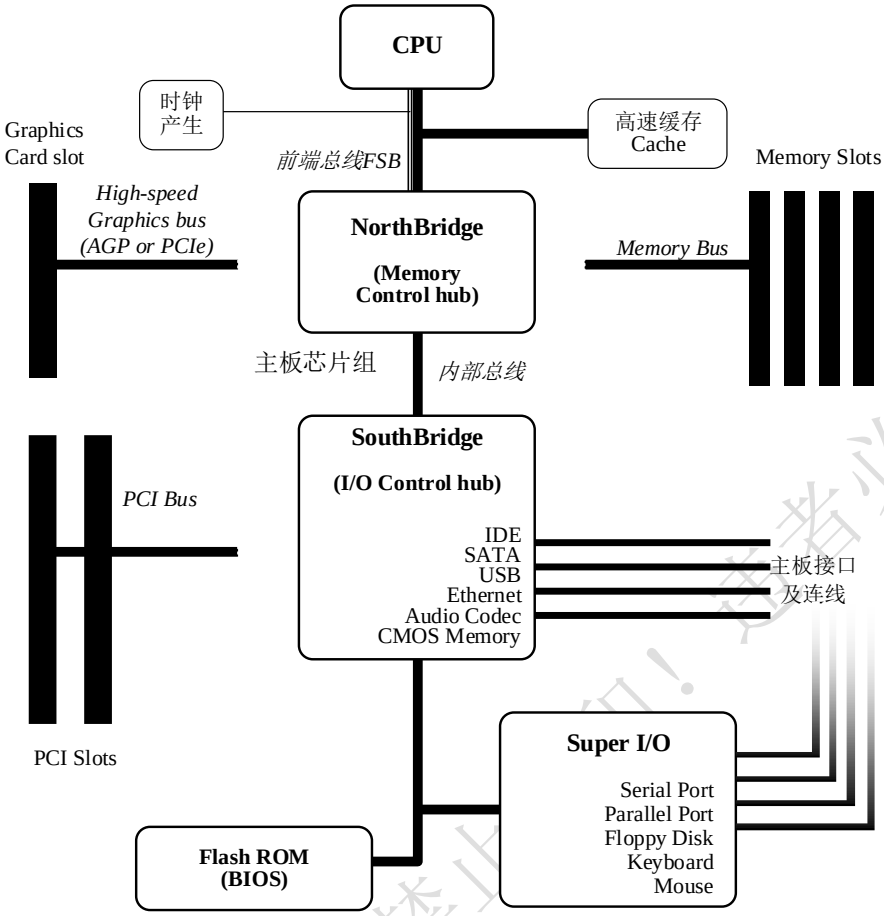


图 4.41　x86 架构基于前端总线的多总线系统

5. 基于 x86 的 PCH 控制总线结构

随着芯片集成度的不断增加，同时也为了提高 CPU 与存储器间数据交换的速度，2009 年，Intel 公司把原先位于北桥的内存控制器等功能集成到 CPU 内部，并将北桥剩余的功能与南桥功能融合在同一枚芯片中，推出了单芯片设计的南北桥芯片整合方案，并命名为 PCH（Platform Controller Hub）芯片。至此，由 PCH 芯片负责 PCIe 和 I/O 设备的管理，独立的北桥芯片已不复存在。这套设计标志着以往的 Intel 平台芯片组步入了单芯片时代，并沿用至今（2019 年）。

图 4.42 中的 DMI（Direct Media Interface，直接媒体接口）是 Intel 公司开发的用于连接 CPU 与 PCH 的总线。DMI 采用点对点的连接方式，基于 PCI-Express 总线，跟随 PCIe 总线的升级而换代。图 4.42 中所示的 DMI 3.0，单通道传输速率达到 8GT/s（Giga transaction per second）。

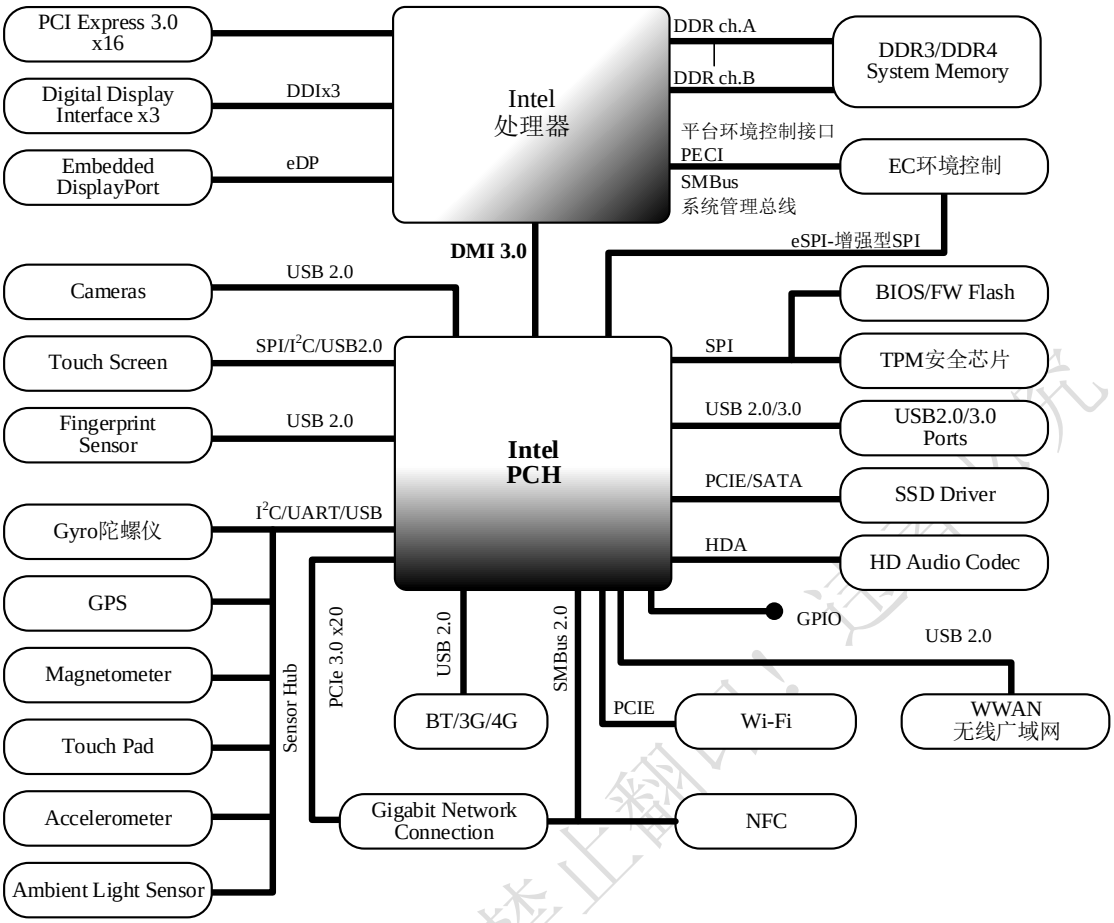


图 4.42　x86 架构单 PCH 芯片的多总线系统

2019 年 Intel 发布了第十代智能 Intel® 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 和 SATA3.0 等，其总线系统如图 4.43 所示。值得注意的是，图 4.43 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。

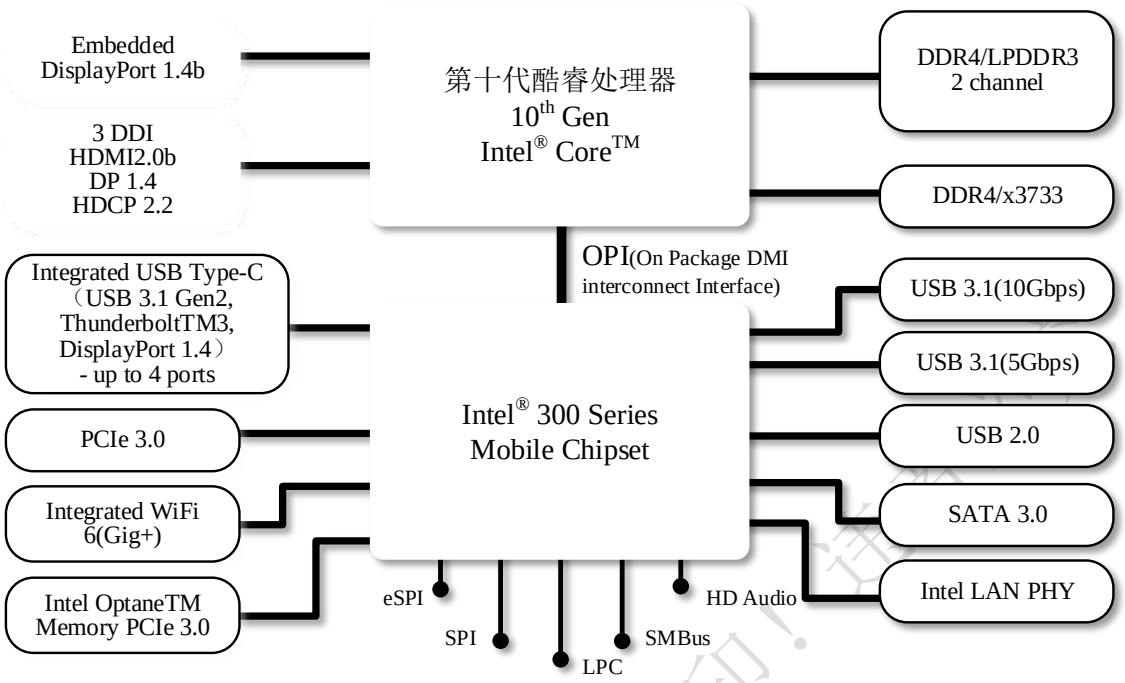


图 4.43　第十代智能 Intel®酷睿处理器的片上多总线系统

4.4 输入/输出接口

计算机的一个最重要也是最主要的基本特性就是能够与各种外部设备进行数据交换。例如，微型计算机上可以使用键盘和显示器进行文本及图形图像的处理。在嵌入式应用领域，计算机常集成在特定的设施中，如家用电器、车辆、移动电话和自动售货机等。在这些嵌入式系统中，显示器、触摸屏、摄像机、麦克风、扬声器、电机和各类传感器等都是常见的输入和输出设备。

计算机需要通过输入/输出接口（简称 I/O 接口）连接各种输入或输出设备。完整的 I/O 接口不仅包括外部设备与 CPU 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。本节所讨论的 I/O 接口限定在硬件方面，即，本章关注的是 I/O 接口电路。后续小节将陆续阐述 CPU 与外设进行数据交互的基本方法，分析典型 I/O 接口电路的特性，介绍一些通用的 I/O 接口标准。

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。有些外设用作输入设备，有些外设用作输出设备，也有些外设既作输出又作输入，还有些外设作为检测设备或控制设备。每一种外设的工作原理不同，它们与 CPU 交换数据时需要用不同的电路单元进行适配。

外设输入信号形式多种多样。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。模拟信号需要经过模拟/数字（A/D）转换电路转换成计算机可以识别和处理的数字信号，再通过接口电路输入到计算机中。对于数字信号，其输入方式既有串行也有并行，信号编码方式和信号电平五花八门，也必须经过接口电路进行转换后才能送入到计算机中。

计算机向外部设备输出信息时，也要考虑外部设备能接收的信号形式。例如，对于采用 RS-232 串行接口的外设，来自 CPU 的并行数据需要并/串转换和电平变换后才能输出；对于需要模拟信号驱动的外设，CPU 输出的数字信号需要经过数字/模拟（D/A）转换后才能送给外设。

另外，大部分外设的工作速度比 CPU 慢，且不同种类外设的工作速度差异较大。例如，使用键盘输入时，键盘的输入速度每秒钟最多只有几十次；电子竞技选手使用鼠标的点击速度，每秒钟不超过 10 次；普通办公用打印机一分钟内只能处理十页左右的文档。也有一些外设属于高速设备，例如在高速数据采集系统中，目前高速 A/D 转换芯片的采样速率已高达几个 Gsps（sample per second）；计算机主机连接存储域网络（SAN，Storage Area Network）所使用的光纤通道卡（HBA，Host Bus Adapter），目前最新规格为 16Gbps。

综上所述，为了保证计算机能够与形形色色的外部设备之间进行数据交换，接口电路必须实现计算机与外设之间的速度适配，必须提供信号类型和数据格式的转换功能，这些转换功能包括模拟量与数字量之间的转换、串行数据与并行数据之间的转换、数据格式的转换以及不同电平之间的转换。

2. I/O 接口的分类

接口种类的划分方式有多种，常见有如下几种分类方式。

（1）按照数据传输方式分类

按数据传输方式，I/O 接口可分为串行接口和并行接口。并行接口是指 CPU 与 I/O 接口之间以及 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。串行方式是指接口采用逐位串行方式传送数据。并行接口适用于传输距离较近的场合，接口电路相对简单。串行接口则适用于传输距离较远的场合，接口电路相对于前者略显复杂。一般来说，在电路时钟频率不太高的数字电路系统中，高速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、SATA、SAS 和 USB 等。

（2）按时序控制方式分类

按时序控制方式分，I/O 接口可分为同步接口和异步接口。同步接口指接口上的数据传送由统一的时钟信号控制，这个时钟信号是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 设备的控制方式分，I/O 接口可分为程序查询接口、中断接口和 DMA 接口等。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），并根据这些状态确定应该执行何种操作。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，在中断服务程序中与 I/O 设备交换信息，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 设备间的高速数据传送。

3. I/O 接口规范的常规内容

如前所述，接口设计之目的是为了协调处理器与外设之间的不一致，以实现彼此之间的速率匹配、数据格式转换和电平变换等功能。依据不同外设的特点，已经形成了很多典型的通用接口设计，这些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 ISO （International Organization for Standardization，国际标准化组织）定义的 OSI（Open System Interconnect，开放系统互连）查看考模型的最低层。概括来说，输入/输出接口标准需要规定四方面的特性：机械特性、规程特性、功能特性和电气特性。这四方面的特性需约定的内容如下。

- ❒ 机械特性，规定硬件接口的机械特征，如接线器的形状、尺寸和引脚排列规则等；
- ❒ 电气特性，规定信号线的电气特性，如电压范围、传输速率、发送器输出阻抗和接收器输入阻抗等；
- ❒ 功能特性，规定各信号线的用途，如高低电压的含义、用来传输数据还是地址、是时钟线还是接地线等；
- ❒ 规程特性，规定在传输全过程中各传输事件在时间上的先后顺序等。

需要注意的是，在一些输入/输出接口的标准中，除了物理层功能，还纳入了通信协议有关的功能。例如，在稍后将要介绍的串行接口中，就包含数据格式和检验方式等数据链路层功能（OSI 参考模型的第二层）。

4. I/O 接口的结构

简单来说，输入/输出接口电路需要实现处理器与外设间的信息交互。需要交互的信息包括三类：①外设状态信息（输入）：用于指示外设的状态；②数据信息（输入/输出）：要传送的二进制数据；③控制信息（输出）：用于控制外设的工作模式。

通常 I/O 接口电路用一组寄存器来存储这三类信息，CPU 通过地址对这些寄存器进行访问。这些寄存器称作端口寄存器或 I/O 端口，简称端口（Port）。CPU 通过读/写端口寄存器实现与外设的数据交换。对应上述三类信息，接口电路中端口寄存器也分为三类：①数据端口（寄存器）：用来存放 CPU 和外设间交换的数据，具有输入和输出数据缓冲功能。②状态端口（寄存器）：用来存放外设状态，如准备就绪（Ready）、忙碌（Busy）和错误（Error）等。③控制端口（寄存器）：用来存放 CPU 发往外设的控制命令，如启动、停止和复位等。

图 4.44 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数据缓存寄存器。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。

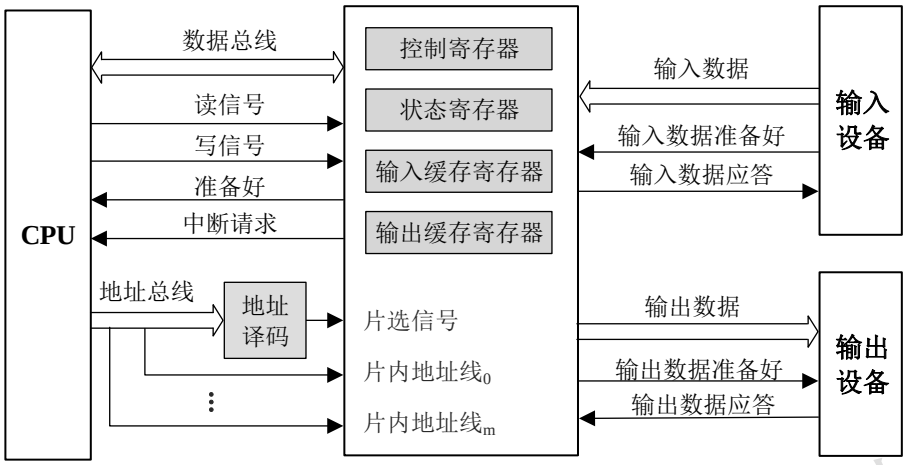


图 4.44　I/O 接口电路的典型结构

图 4.44 左边给出的是 I/O 接口与 CPU 的连接信号，右边给出的是 I/O 接口与外部设备的连接信号。CPU 对图中输入端口/输出端口的读/写操作可实现与外设的数据传送，对控制端口的写操作可实现对接口以及外设的控制，对状态端口的读操作可了解接口以及外设的状态信息。

以打印机为例，数据端口存放 CPU 输出给打印机的数据信息；状态端口存放打印机工作状态信息，如打印机是否准备好、是否忙碌、是否缺纸以及是否卡纸等；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、打印机的走纸和换行等；CPU 对外设的控制操作都通过控制端口进行。一般来说，CPU 对数据端口可读且可写，而对状态端口只能读，对控制端口只能写。

5. I/O 端口编址

计算机系统中可能有多个 I/O 接口，每个 I/O 接口可以有多个 I/O 端口。为了能访问到每个 I/O 接口的各个 I/O 端口，所有端口寄存器需要像存储器单元一样进行编址。编址结果称作端口地址，处理器通过端口地址可访问各个端口。I/O 端口有两种编址方式：内存映像编址与 I/O 端口独立编址，如图 4.45 所示。

1） 内存映像编址

内存映像编址（memory mapped I/O addressing）也称作 I/O 端口统一编址。这种编址方式将 I/O 接口中的每个端口视为主存的存储单元，每个端口占用一个存储单元地址，把主存地址空间的一部分划出来用作 I/O 地址空间。如摩托罗拉公司 68 系列、Intel 公司 51 系列、ARM 和 Power PC 等处理器，就采用 I/O 端口统一编址。内存映像编址也称为“I/O 内存”方式，I/O 寄存器位于“内存空间”。

内存映像编址时，访问 I/O 寄存器与访问内存使用相同的指令，由于访问存储器的指令功能相对较强，编程时使用访问存储器指令读写 I/O 端口，操作更加灵活，程序更加简洁。这种方式的缺点是占用了存储器空间；此外在阅读程序时，只能依据地址来区分是访问内存还是访问 I/O，程序可读性不好。

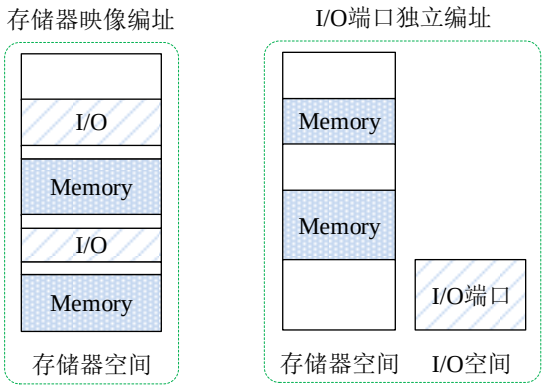


图 4.45　I/O 端口两种编址方式下的地址空间示意图

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），即存储器和 I/O 端口位于两个相互独立的地址空间，存储器和 I/O 端口独立编址。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，I/O 地址空间与存储器地址空间互不相干。例如，80x86 系列微处理器就采用 I/O 端口独立编址方式。

这种编址方式需要设计单独的 I/O 指令，专门用于访问 I/O 端口，还需要使用单独的信号线，指示当前的总线周期是访问内存还是访问 I/O。

4.4.2 I/O 接口数据传送方式概述

如前所述，外设的多样性使 I/O 接口电路的差异很大，CPU 与外设接口之间数据传送方式也有多种多样，常用的有无条件传送方式、状态查询传送方式、中断传送方式和 DMA 传送方式等。

1. 无条件传送方式

某些外设工作时始终处于“准备好”的状态，如按键和 LED 字符显示器等。这些外设随时能够接受 CPU 的访问，处理器不必关心其状态，故此类外设常称为无条件外设。CPU 对无条件外设进行输入输出操作时不需要考虑其状态，此类访问被称为无条件传送方式。

图 4.46(a)所示是无条件输入时的接口电路示意图。图中，外设数据输出接口始终有数据输出，并经三态缓冲器和系统数据总线相连，CPU 对端口进行读操作时，端口地址信号 AD[31:0]和读信号 RD#经地址/读信号控制电路生成三态缓冲器（three-state buffer）的输出使能信号。

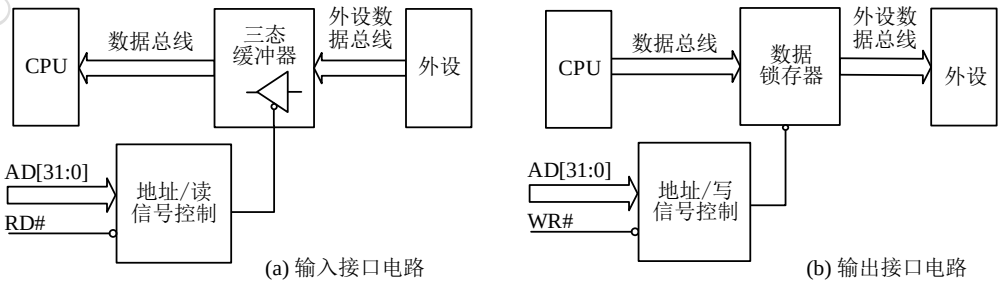


图 4.46　无条件数据访问方式 I/O 接口电路示意

三态缓冲器又称三态门，其工作状态受输出使能信号控制。输出使能信号有效时，器件实

现正常逻辑状态输出，称为三态门打开；输出使能信号无效时，其输出处于高阻状态，等效于与所连的电路断开，也称作三态门关闭。当图 4.46(a)中的 CPU 读 I/O 端口时，AD[31:0]送出 I/O 端口地址，RD#有效，地址/读信号控制电路输出的三态缓冲器输出使能信号有效，三态门打开，外设输出的数据通过三态缓冲器加载到系统数据总线上并送达 CPU。

无条件输入接口电路如图 4.46(b)所示，由于 CPU 速度高于外设速度，一般使用锁存器来保持 CPU 输出数据，供外设读取。CPU 输出数据时对输出 I/O 端口进行写操作，AD[31:0]送出 I/O 端口地址，WR#有效，地址/写信号控制电路生成数据锁存器的锁存信号，将 CPU 输出的数据锁存到输出数据锁存器。在下一次锁存信号到来之前，输出锁存器一直保持这个数据，供外设使用。

2. 查询传送方式

事实上，很多外设在对其进行操作之前需要先确认其工作状态，只有状态许可时才可以访问。例如，A/D 转换器转换结束后才能读转换结果；CPU 向打印机输出数据时，必须打印机准备好之后方可进行。此类须满足一定条件才能访问的外设称为条件访问外设，绝大多数计算机外设都属于条件访问外设。CPU 对此类外设进行输入或输出操作时需要先查询外设的状态，对此类外设的访问方式称为条件 I/O 传送方式。

条件 I/O 传送有两种主要实现方式：查询传送方式和中断控制方式。

查询传送方式又称为程序查询方式，或简称查询方式。其原理和过程为：CPU 访问数据端口前，先查询该设备的状态；设备“准备好”时，CPU 才对数据端口进行输入/输出。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 4.47 所示。

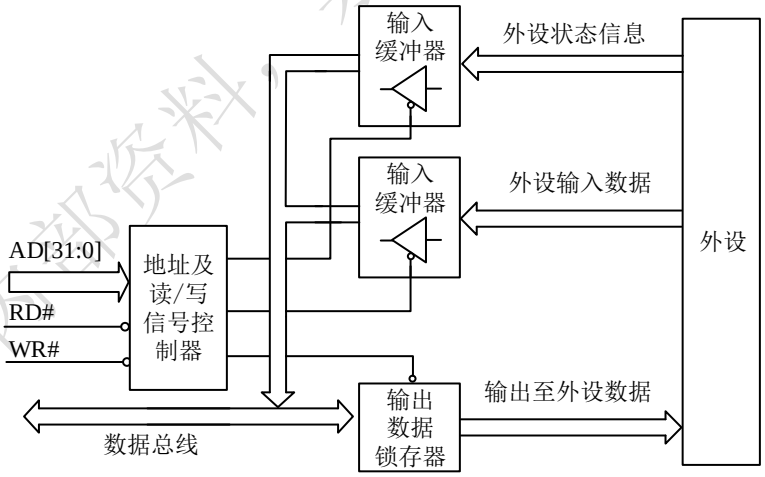


图 4.47　状态查询方式 I/O 接口电路示意

状态查询方式的数据传送流程如图 4.48 所示。其基本步骤为：①CPU 从状态端口读取状态信息。②CPU 检测状态信息是否满足数据访问要求。③若不满足，重复步骤①和②；若外设已就绪，则访问数据端口进行数据传输。

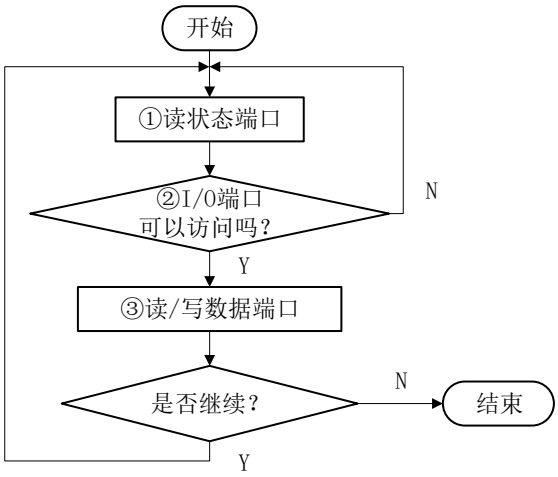


图 4.48　状态查询方式 I/O 控制流程

从上述状态查询方式数据传输过程可知，如果外设没有准备好，CPU 需要反复执行步骤①和②，在此期间 CPU 无法进行其它操作，导致宝贵的 CPU 资源出现大量浪费。另外一方面，如果 CPU 查询外设状态的周期设置过长，系统就不能对外设状态的改变及时做出响应，易导致数据丢失。针对上述问题的一种改进思路是采用接下来将要介绍的中断传输方式，把 CPU 等待外设就绪变成外设主动汇报状态信息，从而节约 CPU 的时间。

3. 中断传送方式

状态查询方式中，CPU 需要反复执行大量外设状态查询指令，故而 CPU 利用率不高。在中断传输方式中，平时 CPU 执行正常的处理任务，当外设状态改变或者需要与 CPU 进行数据交换时，由 I/O 接口主动向 CPU 发出数据传送请求，CPU 中断当前的任务执行，转而执行相应的中断服务程序，为外设（I/O 接口）进行服务。

中断传送方式通常会引入专门的中断控制器，对多个 I/O 接口的中断请求进行管理，中断控制器可以是独立电路单元（例如 Intel 8259A 可编程中断控制器），也可以与 CPU 集成在一个芯片上（如 ARM Cortex-M3/M4 处理器中的 NVIC）。中断传送的大致过程可以图 4.49 所示中断传输方式接口电路为例，示意图。

当 I/O 接口需要对其进行服务时，I/O 接口向中断控制器发出中断请求 IRQ（Interrupt Request）；中断控制器收到某个 IRQ 之后，对其中断条件以及中断优先级进行检查，如果符合条件，中断控制器向 CPU 发出外部中断请求信号 INT（Interrupt）。CPU 收到 INT 后暂停当前的程序执行，转去执行中断服务程序，在中断服务程序中为 I/O 接口进行服务（如读/写数据等）；中断服务程序执行完毕之后，CPU 返回原来被中断的程序断点，继续执行被中断的程序。

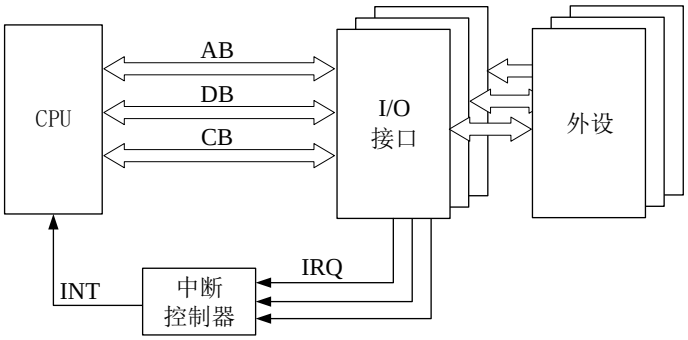


图 4.49　中断方式 I/O 接口电路示意

中断传输方式与前述状态查询方式，都是有条件传送方式的具体实现手段。中断方式有利于提高 CPU 使用率，已成为最常用的输入/输出方式。鉴于中断是计算机系统，尤其是嵌入式系统中一项较为重要的技术。本章将中断传输方式专门作为一小节，对其做更详细的介绍。

4.4.3 中断传输方式

1. 中断的基本概念

中断是一种信号，用来通知 CPU 发生了特定事件，需要 CPU 进行某种处理。如，用户敲击计算机键盘按键时会产生一个中断信号，告诉 CPU 发生了“键盘输入”，要求 CPU 读入键盘输入值。

不仅外设可以产生中断信号，CPU 内部的逻辑电路单元也可以产生中断信号。如 CPU 的 ALU 发生除零错误时，就会产生“除零”中断。为了便于区分源自 CPU 内部和外部的中断源，早期的资料习惯把来自 CPU 外部中断称作中断（interrupt），而把来自 CPU 内部的中断称作陷阱（trap）。如今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 4.50 所示为某随机事件通过中断向 CPU 请求服务的过程。在允许中断的条件下，一旦发生中断，CPU 暂停现行程序的运行，转去执行中断服务程序；中断服务完成后，CPU 再返回原来的程序。

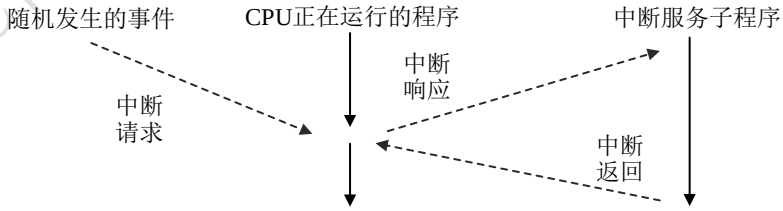


图 4.50　中断方式示意图

以下分别介绍中断技术中常用的一些术语，包括中断源、中断向量、断点、现场、中断优先级、中断嵌套和中断屏蔽等。

1) 中断源

中断源指引起中断事件的来源，如电压跌落、存储器校验错误等异常事件，或键盘、鼠标

等外设的数据传输请求，如图 4.51 所示（注：图 4.51 中 NMI 和 IRQ 的差异稍后在介绍中断屏蔽时予以说明）。例如，ARM 公司的 Cortex-M3/M4 定义的异常来源包括外部中断、内存校验错误、总线错误、用法错误和系统定时器中断等。为了能够识别并管理多个不同的中断源，需要对中断源编号，这个编号被称为中断类型码（或中断类型号）。

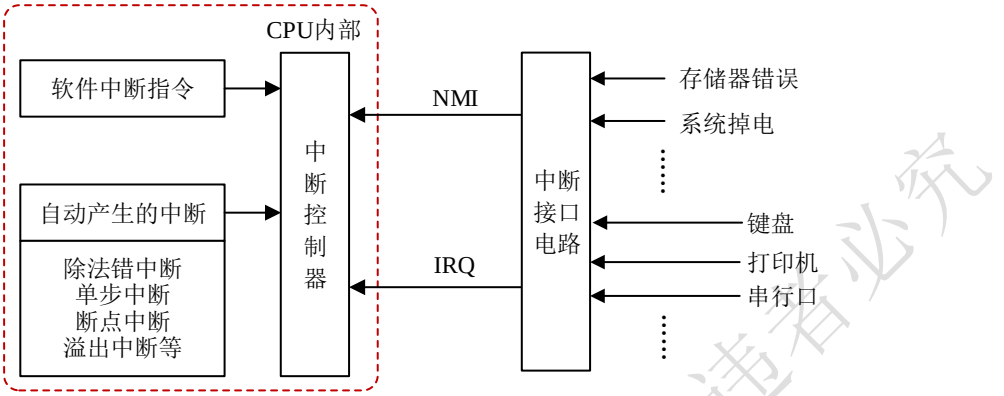


图 4.51　典型计算机系统的中断源

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。通常在处理器设计中，把所有中断服务子程序入口地址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[5]）。中断发生时，CPU 根据中断类型码到 IVT 进行查询，找到对应的中断服务程序入口地址。

3） 断点和现场

断点是指被中断的主程序中下一条待执行指令的地址，以及发生中断时 CPU 状态（标志）寄存器的内容。为确保中断服务子程序执行完后能正确返回原来的程序，还原 CPU 的工作状态，中断系统需要在进入中断服务程序之前保存断点，在中断返回时恢复断点。

现场是指中断发生时被中断程序的运行环境，主要是 CPU 内部各类寄存器的内容。由于中断服务时可能会使用到部分或者全部寄存器，中断服务之后这些寄存器的内容将发生改变（现场遭到破坏），为确保中断返回后能够恢复 CPU 的运行环境，中断系统还需要在执行中断服务程序之前对现场进行保护，在中断返回时恢复现场。

例如，在 Intel x86 系统中，保护断点是在中断响应过程时，由 CPU 自动完成，而保护现场应在中断服务程序的最开始处，由程序实现，两者的动作执行主体不同，执行时间也不同。保护断点时被保护的内容是确定的，而保护现场时，需要保护的对象应根据实际情况（中断服务程序需要使用哪些寄存器）由指令确定。恢复断点是在中断返回时完成的，而恢复现场应在中断服务结束之后以及中断返回之前完成。

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，需要按预先定义的规则依次服务。最常用的规则是中

[5] 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。

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。

实际系统中，CPU 为某个中断源服务的过程中，也可能会接收到其它中断请求。如果后到的中断请求优先级高于正在处理的中断请求，可以再次中断正在执行的中断服务子程序，转而去服务新的、优先级更高的中断请求，这种机制称为中断嵌套。图 4.52 描述了一种中断嵌套过程，其中中断源 B 的优先级高于中断源 A，图中数字表示各事件的发生顺序。

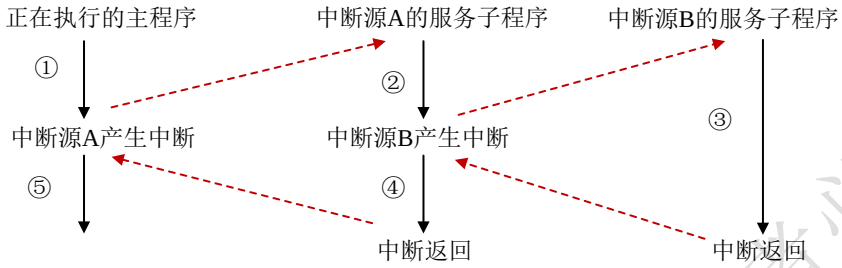


图 4.52　中断嵌套示意图

5） 中断屏蔽

CPU 在执行某些操作时，如果不希望被打断，可以对中断请求进行屏蔽不予响应。中断屏蔽有两种实现方式，一种是对 CPU 内部相关寄存器中的控制标志位进行设置，例如在 Intel x86 处理器中，如果状态寄存器中中断允许标志位“I”被复位为 0，则 CPU 对于来自外部的中断请求不予理会，这种方式常被称为“关中断”；另外一种方式是在中断控制器中设置中断屏蔽寄存器，对某些中断源进行屏蔽管理。通常所说的中断屏蔽一般是指后一种方式。此外，如果优先级较高的中断请求正在被服务，此时自动屏蔽优先级低的中断。在这种情形下，优先级较低而未被响应从而处于等待响应状态的中断也被称为挂起（Pending）中断。

并非所有的中断都可以被屏蔽，例如复位、硬件故障、电源电压跌落和存储器校验错误等事件必须立即处理，不允许被屏蔽。因此，这类中断应不受中断屏蔽标志位影响，即 CPU 总是会检测并响应此类中断请求。按照是否可以被屏蔽，中断可分为图 4.51 所示的两大类，不可屏蔽中断 NMI（Non Maskable Interrupt）和可屏蔽中断 IRQ（Interrupt Request）。

2. 中断处理过程

计算机系统处理中断的过程可分为中断响应、中断处理和中断返回三个阶段。①中断响应指 CPU 响应中断请求，暂时中断当前执行的程序并跳转到中断服务子程序的过程。当 CPU 确定需要响应中断后，首先获取中断类型码，然后根据中断类型码在中断向量表中查找到对应的表项，进而得到中断服务子程序的入口地址，随后保护现场，在下一个指令周期跳转至中断服务子程序入口，进入中断处理阶段。在现代计算机的中断系统中，上述过程都是自动完成的。②中断处理就是执行中断服务子程序。中断服务子程序由用户编写，根据中断服务要求进行相应的操作，如读取外设数据、向外设输出数据等。在中断服务程序最开始处，应根据实际需要，安排响应的保护现场操作指令。③中断服务子程序执行完毕，需要恢复现场，然后执行最后一条中断返回指令。中断返回指令执行后，恢复断点，程序返回被中断的程序继续执行。

3. 中断优先级

中断源优先级的判断既可以用软件查询方法实现，也可以用硬件排序电路实现。软件查询仅需要少量硬件电路。例如，在图 4.53 所示的电路中，系统共有 8 个外部中断源，某个中断源对应的中断请求位和中断允许位（作用与中断屏蔽位相同）同时为“1”时，对应的中断请求信号就为“1”；8 个中断源的中断请求信号经过“或”门，输出总的中断请求信号 IRQ 发往 CPU。CPU 收到 IRQ 之后读取中断请求寄存器，便可获知有哪些中断源有中断请求。之后，程序员所选择的处理顺序就体现了中断的优先级。由于需要读取中断请求寄存器后再判断应该先为谁服务，软件查询法速度略慢。

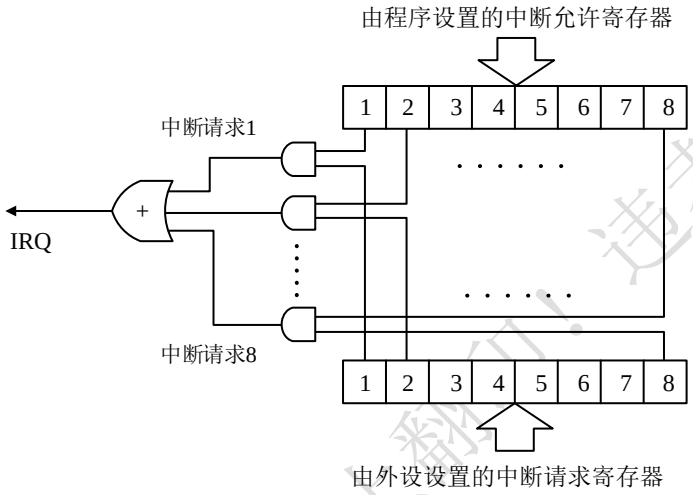


图 4.53　软件查询判优系统的接口电路

与软件查询法不同，硬件排序方法使用特定结构的电路实现中断优先级（优先权）的判定。硬件排序判断优先级的电路可以有多种实现方法，常用的是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 4.54 所示，系统中有 8 个中断源，一个或多个中断源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。在两种情况下中断请求信号可产生有效 IRQ：①比较器输出为“1”时，中断请求通过与门 1 送出；②优先权失效信号为“1”时，中断请求通过与门 2 送出。

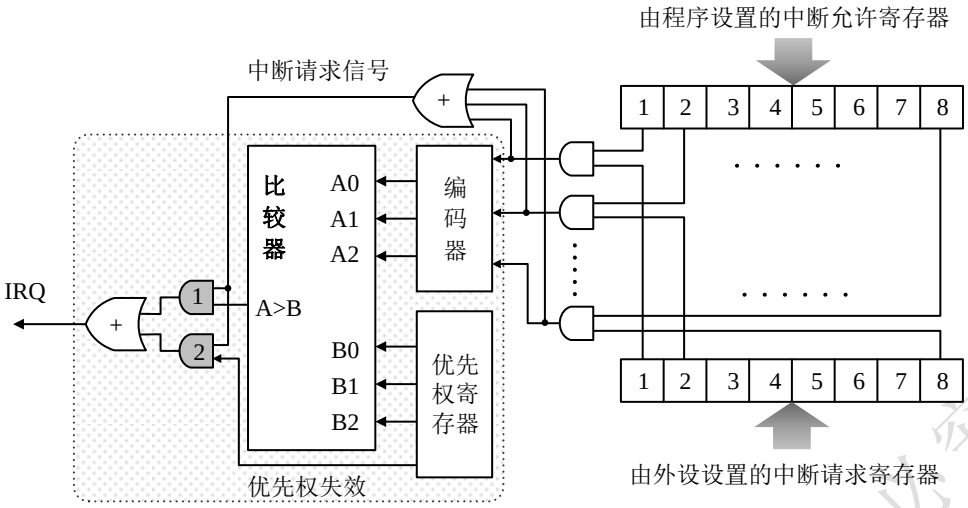


图 4.54　用编码器和比较器实现的中断优先权排队电路

图 4.54 中编码器按照预定义的优先级规则把不同来源的中断请求转换为三位二进制编码[A2,A1,A0]，在多个中断源有请求时输出优先权最高中断源对应的编码。优先权寄存器中保存 CPU 当前正在处理中断的编码[B2,B1,B0]。比较器输出为“1”（即 A>B）说明中断请求的优先级高于 CPU 正在服务中断的优先级，可以产生中断嵌套。图 4.54 中优先权失效信号指示当前 CPU 未处理中断，无须判断中断能否嵌套。

菊花链是管理中断优先级的另一种硬件方法。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过程。在图 4.55 所示电路中，任一个外设的中断请求都可以通过或门直接送到 CPU 的 IRQ 引脚，若 CPU 允许处理，则发中断响应信号 INTA（Interrupt Acknowledge）给外设。菊花链电路被用来确定中断响应信号究竟送到哪一个设备。

在图 4.55 中，假设中断请求信号和中断响应信号 INTA 都是高电平有效。CPU 送出 INTA 后，若设备 1 有中断请求，则与门 A1 开，与门 A2 关，中断响应信号送至设备 1 而不再向下一级传送。反之，若设备 1 无中断请求，则与门 A1 关，与门 A2 开，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，中断响应信号送至设备 2，同时与门 B2 关，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）。依据上述分析，设备接入菊花链的顺序决定了设备的中断优先级：越靠近链前端（靠近 CPU）的设备优先级越高。

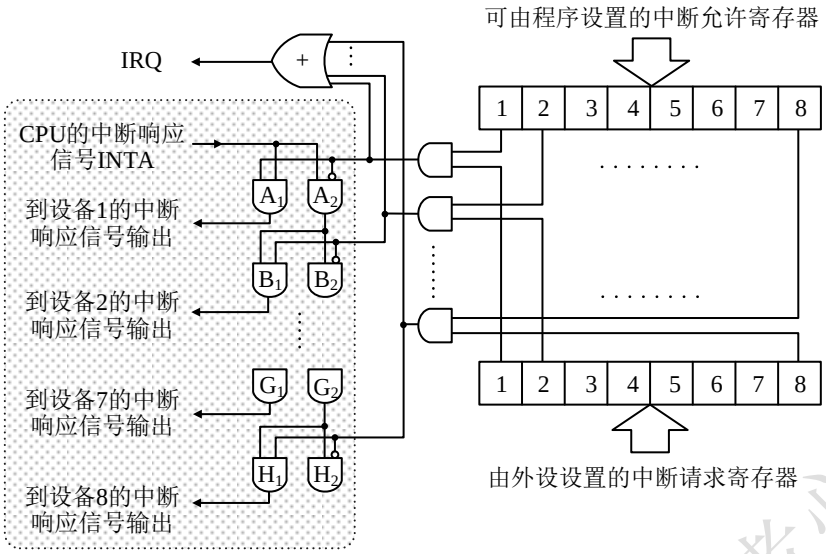


图 4.55　菊花链式优先权排队电路

4. 中断传输方式的接口电路

图 4.56 是一种中断控制方式输入接口电路。外设准备好数据后，发出选通信号，数据进入锁存器；同时选通信号将中断请求触发器置“1”。如果该中断源被允许（中断屏蔽位是“0”），则中断请求信号通过与门 A 产生 IRQ；但如果中断被屏蔽，则无论外设是否准备好，都无法产生 IRQ。CPU 收到 IRQ 后响应中断，产生中断响应信号 INTA，INTA 通过与门 B 后复位中断请求触发器。进入中断之后，中断服务程序对数据端口进行读操作，地址信号经过译码后和读信号 RD#通过与门 C 使能三态缓冲器输出，数据传送至数据总线。

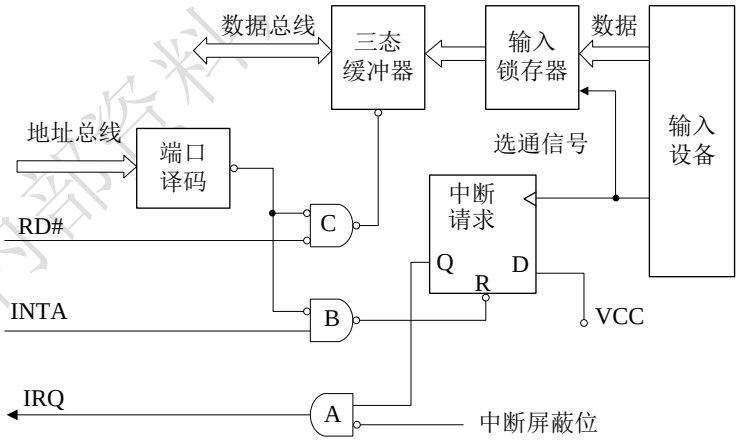


图 4.56　中断控制方式的输入接口电路

4.4.4 DMA 传送方式

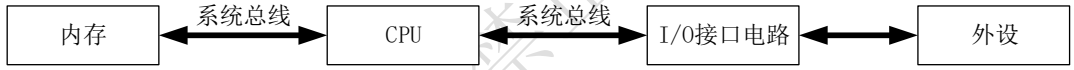
1. DMA 传送方式概述

虽然中断传送方式比查询传送方式效率高，但中断方式仍存在不足之处。分析中断传输方

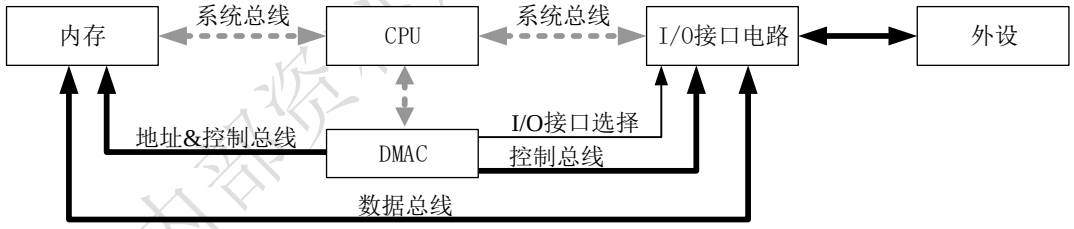
式的数据传输过程，整个中断处理过程中都需要 CPU 参与，数据的输入和输出都是由 CPU 执行指令完成的。即便所传输的数据无须 CPU 处理，也要经过 CPU 进行中转。例如，从 I/O 端口读取数据到内存某个存储单元，数据需要经过 I/O 端口到 CPU 内部寄存器、以及 CPU 内部寄存器到内存单元两个阶段，需要两次总线操作。此外，在进入中断服务时，无论中断服务内容简单还是复杂，都需要保护断点和保护现场；中断返回时还需要恢复现场和恢复断点，CPU 在两段任务代码间切换时有系统开销。当中断频度较高时，系统开销增加，会影响计算机系统性能。

DMA 方式顾名思义是在内存与 I/O 接口之间直接进行传输，无需 CPU 干预。为此，DMA 传送方式引入了一个专用电路单元，其功能是与 CPU 协商并获得系统总线控制权之后，在 I/O 接口和存储器之间建立一条可直接传输数据的临时通道，通过地址总线和专用引脚信号同时选中需要进行数据传送的内存单元和 I/O 端口，在一个总线周期内先后发读信号和写信号，将需要传送的数据读出至数据总线并写入目的单元，在一个总线周期内完成一次数据传送。这个专用电路单元被称为 DMAC（Direct Memory Access Controller，DMA 控制器）。

DMAC 本身没有指令系统，平时作为一个可编程接口控制芯片，接受 CPU 的管理和控制。这些管理包括设定 DMA 传送方式、数据块传送时的数据块大小和地址增长方向等，这个过程称为 DMAC 的初始化设置。进入 DMA 传送之后，DMAC 按照初始化设置的参数，控制系统总线完成高速数据传送。DMA 特别适合内存与 I/O 接口之间的大数据量传送。图 4.57 是 CPU 控制下的 I/O 接口与存储器之间数据传送过程，图 4.57（b）是 DMAC 控制下的 I/O 设备与存储器之间的数据传送，读者可自行对比两者的差别。



(a) CPU控制的I/O接口与内存之间的数据传输



(b) DMA传送时的I/O数据传输通道

图 4.57　DMA 方式 I/O 接口电路示意

2. DMA 传送过程

DMA 传送的典型场景有主存储器到 I/O 端口的（M→I/O）传送和 I/O 端口到主存储器的（I/O→M）传送。根据对存储器的访问类型，前者被称作 DMA 读操作，后者被称作 DMA 写操作。此外，某些 DMAC 还支持存储器到存储器（M↔M）的数据传送，与 CPU 控制下的传送相比，M↔M 测试并没有什么优点，所以很少采用，本书也不再介绍。上述传送场景可以用图 4.58 表示。

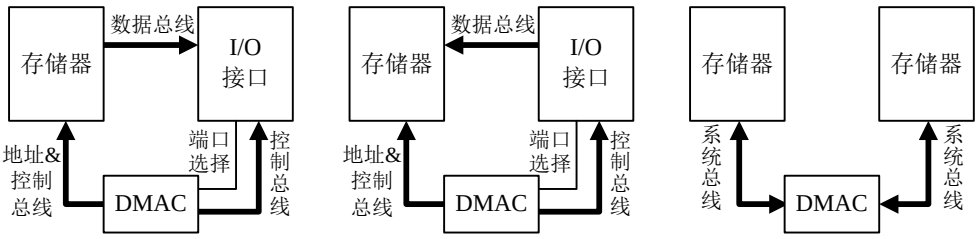


图 4.58　DMA 传送系统示意图

DMAC 与 CPU 的连接关系可以用图 4.59 加以说明。从图中可以看出，系统总线平时总是由 CPU 控制，当外设需要进行数据传送时，通过 I/O 接口向 DMAC 发出 DMA 传送请求 DRQ（DMA Request）。如果 DMAC 认为 DRQ 符合条件，则向总线仲裁逻辑发出总线使用申请 BR。总线仲裁逻辑收到 BR 信号后，若满足条件则同意 DMAC 使用总线。DMAC 驱动总线切换控制信号，断开 CPU 一侧与系统总线的连接，将系统总线的使用权交给 DMAC，然后用 BG 信号通知 DMAC，可以开始 DAM 传送。

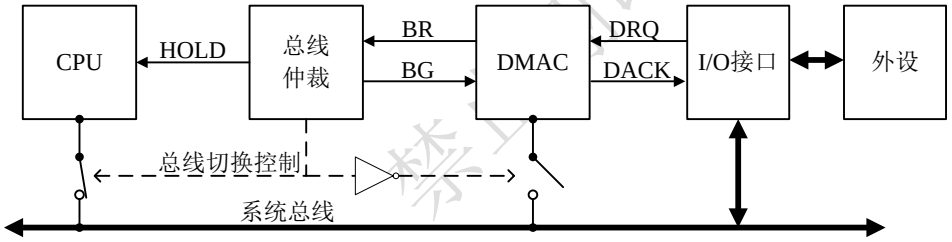


图 4.59 CPU 与 DMAC 之间的连接原理图

上述 DMA 传送的基本过程可以归纳为如下九个步骤：

① I/O 接口需要 DMA 传送时，向 DMAC 发 DMA 传送请求 DRQ；

② DMAC 收到 DRQ 后，若符合条件[6]则向总线仲裁逻辑发出总线使用请求信号 BR；

③ DMAC 收到 BR 后，如果符合条件，适时[7]发出总线切换控制命令，切断 CPU 与系统总线之间的连接，然后[8]再将系统总线控制权交给 DMAC；

④ DMAC 置 HOLD 信号有效，通知 CPU 暂时停止使用总线；

⑤ DMAC 置 BG 信号有效，通知 DMAC 总线切换已经完成，可以进入 DMA 传送；

⑥ DMAC 收到 BG 后置 DACK[9]（DMA Acknowledge）信号有效，通知 I/O 接口开始对其进行 DMA 传送服务；

⑦ 进入 DMA 传送之后，DMAC 按照初始化设置参数，把当前 DMA 传送涉及的存储器地址送到地址总线，如果是 I/O→M 传送，DMAC 发 I/O 端口读命令和存储器写命令，如果是

[6] 条件包括 DMA 请求未被屏蔽并具有当前最高优先级。

[7] DMA 传送的总线切换发生在两个总线周期之间，而中断和返回发生在两条指令之间。

[8] 总线切换操作有严格的时序要求，必须符合“先断后合”的原则，以避免出现总线冲突。

[9] DACK 也是 I/O 端口选择信号。由于仅有一套地址总线，DMA 传送期间被用于内存单元寻址，DMAC 只能采用硬连线方式，通过 DACK 选择 I/O 接口。DMAC 一般配置了多个通道，每个通道都有一对 DRQ 和 DACK 信号，可为多个 I/O 接口提供 DMA 传送服务。

M→I/O 传送，DMAC 发存储器读命令和 I/O 端口写命令，把数据读出到数据总线并写到目的单元，在一个总线周期内完成数据传送；

⑧ 如果是数据块传送，完成一次 DMA 传送后，DMAC 修改存储器地址，修改待传送次数寄存器内容（减 1），再重复上一步骤；

⑨ 当预先设定传送的次数（任务）全部完成，DMA 传送结束，DMAC 撤销总线请求，总线仲裁逻辑对总线连接进行切换，并将 HOLD 信号置为无效，重新恢复 CPU 对系统总线的控制。

3. DMA 的传输方式

I/O 设备和主存储器间进行 DMA 传输时，既可以每次传输一个数据块，也可以仅传输一个数据，还能按照间歇的方式传输多个数据。通常分为如下三种不同的传输方式。

1) 单次传送

单次传送的流程如图 4.60 所示，其特点是每完成一次传送，无论预定的传送任务是否完成，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，获准后继续执行单次传送，直至全部任务完成。

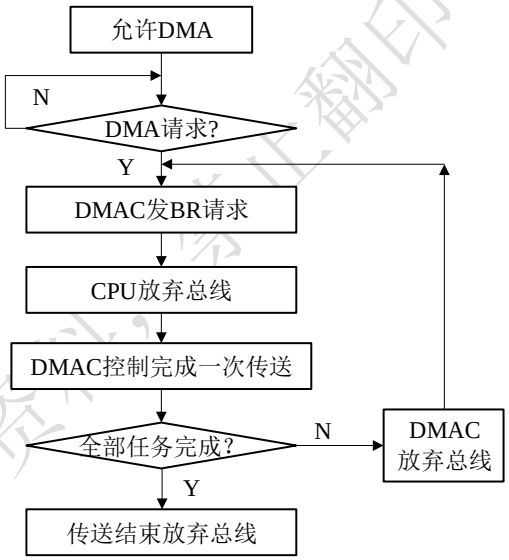


图 4.60 DMA 单次传送处理流程

2) 数据块传送

数据块传送也称为成组传送，其特点是 DMAC 在获得总线使用权之后，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 4.61 所示。

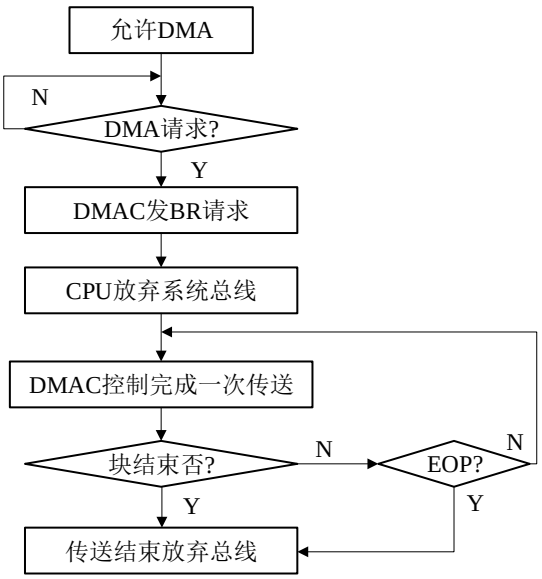


图 4.61　DMA 成组传送处理流程

3）询问传送

询问方式又称为请求传送方式。该方式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 4.62 所示。

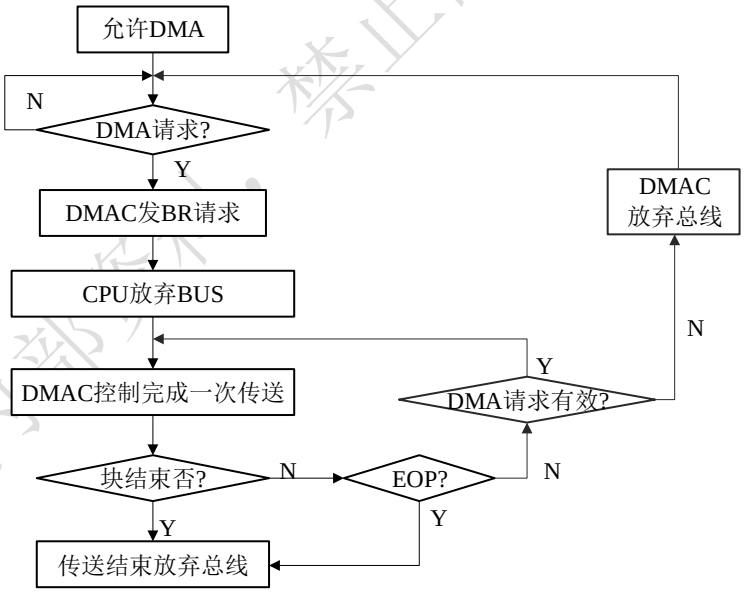


图 4.62　DMA 请求传送处理流程

上述成组传送方式适合大数据量的快速传送，但是 DMAC 长时间占用总线将使得 CPU 无法处理某些紧急事务，因此成组传送方式只能在特定条件下使用。询问传送方式非常适合与数据通信接口芯片之间进行数据交换。例如，目前广泛使用的带有 FIFO（Firs Input First Output，先进先出）数据缓冲器的串行通信接口芯片中，都带有接收缓冲器满（RxRDY）和发送缓冲器空（TxRDY）信号，当接收 FIFO 中接收字节数超过一定数值，或者发送 FIFO 中待发送字节少于一定数值，相应的信号线有效，接口芯片利用这两条信号线与 DMAC 协调，分别通过

两个 DMA 通道实现高效数据收发操作。

DMA 传送方式完全依靠硬件实现数据传送，没有专门的指令和传送程序，不能处理较复杂的事件，所以 DMA 方式不能完全取代中断传送方式。事实上，在很多应用场景中，当某次 DMA 传送结束后，往往还利用中断信号通知 CPU 进行下一步处理。

注意，DMAC 本身也是接口芯片，在使用之前需要通过 CPU 对其进行初始化配置。DMA 方式存在一个固有的缺陷，无法对 I/O 接口进行寻址。为满足多个外设的 DMA 传送需求，DMAC 一般会设置多个通道，为多个外设提供 DMA 传送服务。DMAC 每个通道都拥有一套独立的寄存器组以及一对 DMA 请求和响应信号线。多个通道的 DMA 请求信号还需要通过优先级电路进行排队，当同时出现多个通道的 DMA 请求时，以便确定应该先为哪个通道提供传送服务。此外，DMA 传送也不能嵌套，这一点也有别于中断传送方式。

4.4.5 并行接口

并行接口，指采用并行通信方式来传输数据的接口标准，多个数据位可以同时在两个设备之间传输。并行接口种类有数十种之多，从简单的寄存器接口，或专用的可编程并行接口芯片（如 Intel 8255），乃至复杂的 SCSI 接口，实现方式和功能各有所不同。并行接口的传输性能可用两个参数予以刻画：①数据通道的宽度，即接口上可以同时传输的位数；②接口的时钟频率。数据的宽度可以是 2～128 位或者更宽。二十世纪在计算机领域最常用的并行接口是 LPT（Line Print Terminal）接口，目前已被更为简单的 USB 接口所取代。本小节通过一些示例结束并行接口的原理和工作方式。

1. 无握手信号的并行接口

无握手信号接口常用于功能简单的外设，如按键，数码显示管等，CPU 与外设间传输数据时，外设总是处于准备好的状态，属于 4.4.2 小节图 4.46 所示的无条件传送方式。将图 4.46 (a)和图 4.46 (b)的输入和输出电路合并在一起，可以得到典型的无握手信号并行接口电路框图，如图 4.63 所示。

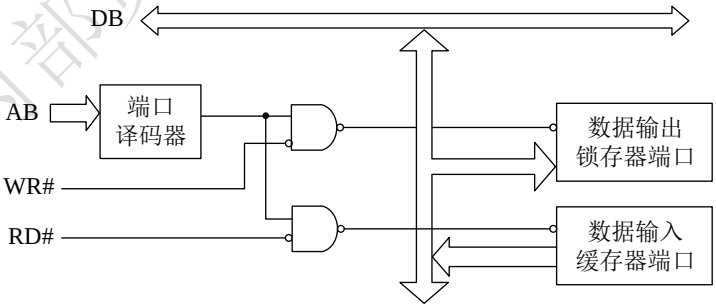
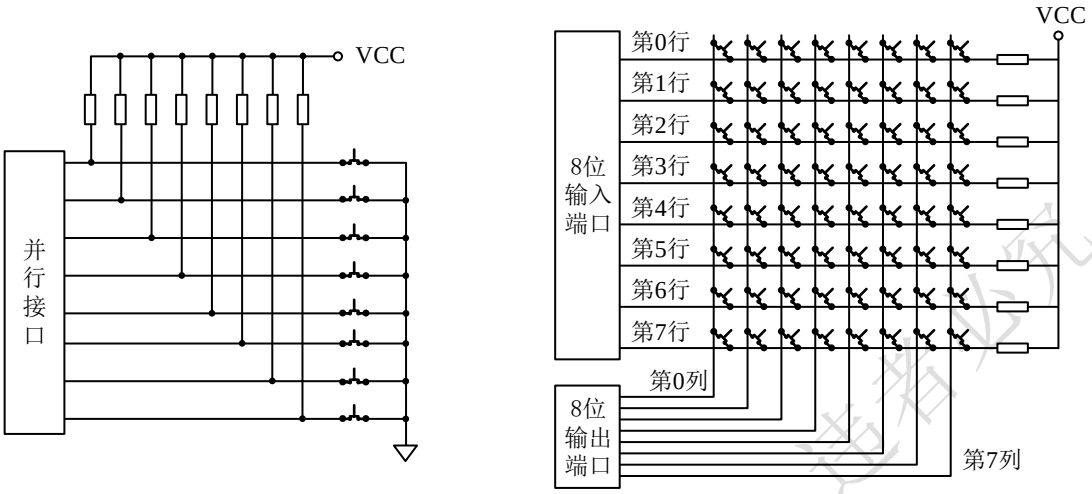


图 4.63　无握手信号并行接口电路示意图

1) 输入接口示例：键盘

键盘是一种常用的输入设备。依据按键状态获取方式的不同，可以分为线性键盘和矩阵键盘。线性键盘的每一个按键占用 I/O 端口的一根口线（I/O 端口的一根线）如图 4.64(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，除了按键状态的判断、程序还需要实现按键的软件去抖动。

矩阵键盘将按键按照行、列方式排列起来形成矩阵开关，图 4.64(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。假设用两个端口，口线数目分别为 M 和 N，考虑可支持的按键数，矩阵键盘为 M×N，而线性键盘只有 M＋N。



(a) 线性键盘结构　　(b) 矩阵键盘结构

图 4.64　线性键盘与矩阵键盘结构示意图

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、行列反转扫描法和行列扫描法等。此处仅以图 4.64(b)所示 8×8 矩阵键盘为例介绍行扫描法的原理。

行扫描法首先判断是否有键按下。方法是先在输出端口的各位输出“0”（即低电平），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。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。

在确定有按键被按下后，行扫描法按照图 4.65 所示的流程确定哪个键被按下。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 4.65 中阴影区域部分）。为方便描述，定义图 4.64 (b)所示 8×8 矩阵键盘中按键的键号为（列号×8+行号）。

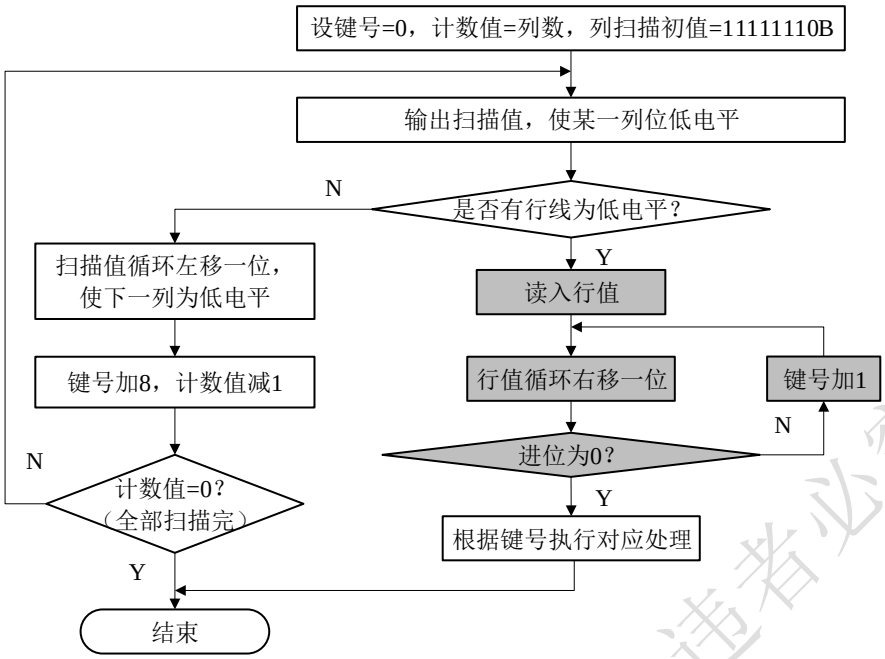


图 4.65　行扫描法的程序流程

逐列检查过程：程序先将键号置零，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。

如果在逐列检查过程中，发现有行线为低电平，则表示此列有按键被按下。例如，读入的行值为 1111 1011B 表示按下的按键在第 2 行（定义输入端口最低比特对应第 0 行）。此时可根据行值确定按键所在行的行号，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。

2）输出接口示例：LED 段式显示器

LED 段式显示器是一种半导体发光器件，常简称数码管，其基本单元是 LED（Light Emitting Diode，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 LED。

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 LED 的共有正极，各 LED 的负极（或阴极）独立，使用时把正极接电源，不同的负极受控接地。共阴极数码管与共阳极数码管的接驳方法相反。图 4.66 所示为八段数码管的结构示意图。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”和“dp”，公共端为“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。

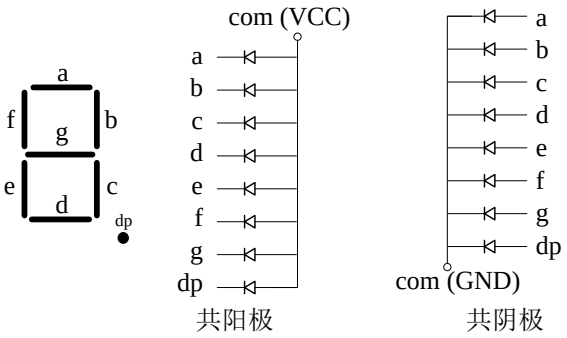


图 4.66　八段数码管结构

以八段数码管为例，显示一个十六进制的数字，需要将表示该数字的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，也可以软件编程实现。

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

实际系统往往需要使用多个数码管同时显示多个数字，此时有静态显示和动态显示两种方式。静态显示方式是各个数码管的输入控制端相互独立，每个数码管相应的段（LED）恒定地导通或截止，直到下一次更新信息。若需要显示 N 个数字，N 个数码管需要的接口口线数目为 8×N。

动态显示方式也称为扫描显示方式，可以使用较少的信号线实现显示控制，动态显示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，但给人的感觉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的输入控制端并连在一起，因此无论数码管的个数是多少，需要的口线数目都是 8，该端口称为段选口。各个数码管的公共端各自连接一根口线，该口线称为位选口。当数码管的个数为 N 时，则需要 N 根位选口口线，因此动态显示方式总共需要的口线数量为 8＋N。

图 4.67 所示为五位十进制数的共阴极 LED 数码管接口电路。五个数码管要显示不同字符，需要使用扫描方式，五个数码管轮流点亮。点亮第一个数码管时，段选码输出端口输出第一个数字的显示码；位选码输出端口输出的位选码中，对应第一个数码管的位为“0”（低电平，共阴极结构），而其他位为“1”（高电平）。

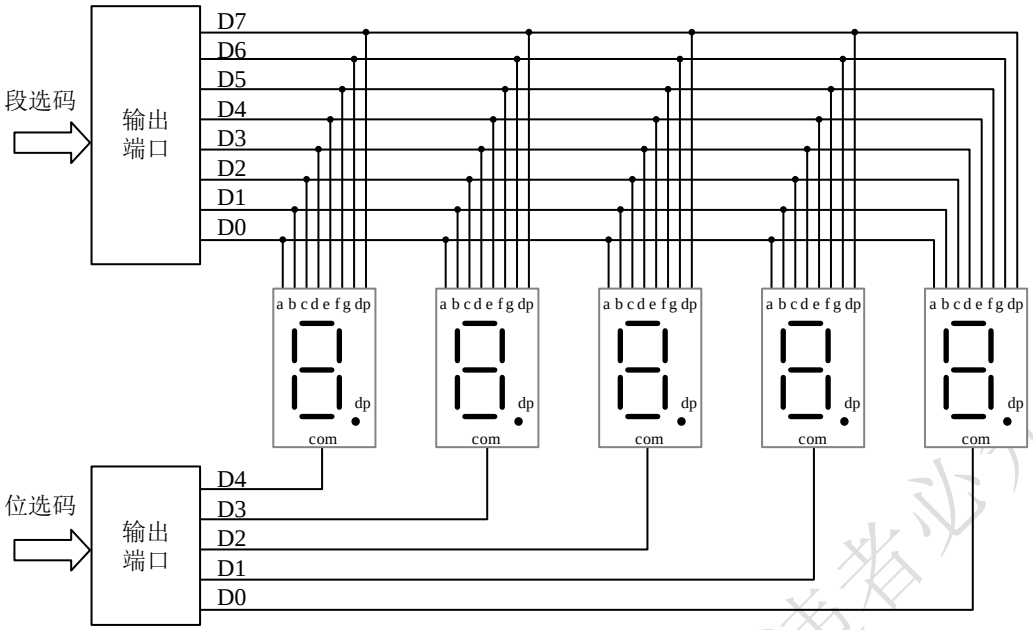


图 4.67　多个数码管的显示接口电路

随后点亮第二个数码管，段选码输出端口输出第二个数字的显示码；位选码输出端口输出的位选码中，对应第二个数码管的位为“0”，而其他位为“1”，这样就完成了第二个数码管的显示。以此类推，依次输出各个数字的段选码和位选码，即可实现五位十进制数的动态显示。

2. 带握手信号的并行接口

为了保证数据传输的可靠性，一些外设通过握手联络（handshake）实现数据交换。此类带握手信号的接口属于 4.4.2 小节所述查询传送方式，其工作原理如图 4.47 所示。带有握手信号的并行接口电路中，除了数据端口之外，通常还有状态端口和控制端口。在图 4.47 所示电路中增加必要的握手信号后，其输入部分构成了如图 4.68 所示的典型输入接口电路，其输出部分构成了如图 4.69 所示的典型输出接口电路。

图 4.68 所示输入接口电路中，数据选通信号 STB（Strobe）和输入缓冲器满（Input Buffer Full）是一对握手信号线。该电路的数据输入过程按照如下步骤进行：① 输入设备准备好数据后，检查 IBF；若 IBF 无效，表明 I/O 接口的输入缓冲器无数据，或者上次输入的数据已被 CPU 读取；输入设备输出数据并发出选通信号 STB；在 STB 信号的作用下，数据进入输入缓冲器/锁存器；同时选通信号将 D 触发器置“1”，即置 READY 为“1”、置 IBF 有效，IBF 有效表示 CPU 尚未读取数据。② CPU 读取状态缓存寄存器，状态信息位 READY 为“1”表示外设已经将数据送至数据缓冲寄存器。③ CPU 读数据端口，同时数据端口读选通信号将 D 触发器清零，使 READY 为“0”并将 IBF 置为无效，完成本次数据传送。

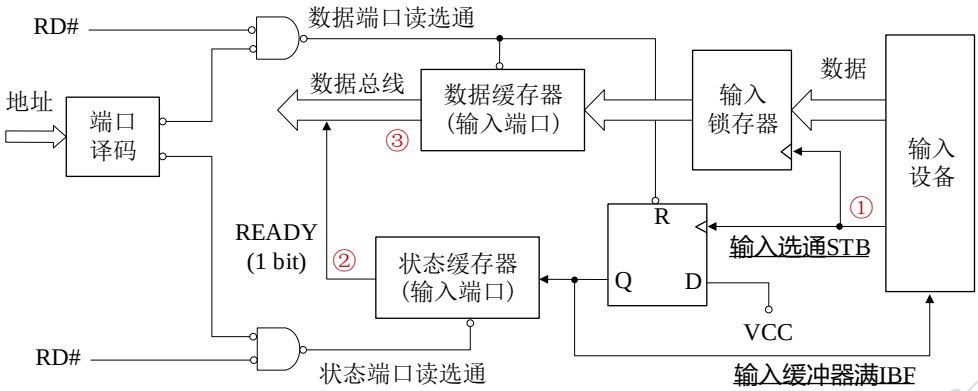


图 4.68　带握手信号的输入接口电路示意图

图 4.69 所示输出接口电路中，输出缓冲器满 OBF（Output Buffer Full）和输出设备应答 ACK（Acknowledge）是一对握手信号线。该电路的数据输出过程按照如下步骤进行。① CPU 读状态缓存寄存器，若状态信息位 BUSY 为“0”表明可以输出。② CPU 向数据端口写数据，写选通信号同时将 D 触发器置“1”（等同于置 BUSY 为“1”），该信号同时作为 OBF 信号告知输出设备，需要输出的数据已写入输出缓冲器。③ 输出设备根据 OBF 获知输出数据有效，读取数据锁存器中数据。④ 输出设备发出响应信号 ACK，将 D 触发器清零（置 BUSY 为“0”），，同时置 OBF 无效，完成本次数据传送。

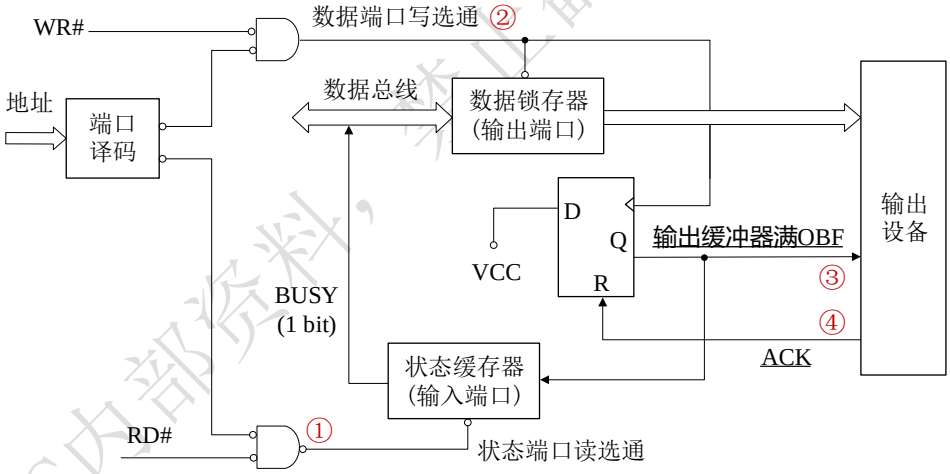


图 4.69　带握手信号的输出接口电路示意图

3. 可编程并行接口（GPIO）

可编程通用并行接口是一种简单外部设备接口电路。其结构简单、应用广泛，并且可以通过编程控制字实现灵活的控制，故得名“通用可编程 I/O 接口”，即 GPIO（General-Purpose IO ports）。用户可以通过写入控制字改变 GPIO 的工作方式，能够满足多种应用场景的实际需求。例如，GPIO 可以仅用一个引脚连接只需要开/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 LCD（Liquid Crystal Display，液晶显示屏）等需要多个控制信号的设备。

图 4.70 为典型的单个 GPIO 引脚电路结构图，该引脚可用于普通的输入/输出信号，也可与其它输出信号复用（图中复用选择位）。通常编程时可配置的寄存器位包括中断允许控制位、复用选择位、三态输出使能位和上拉电阻使能位等。

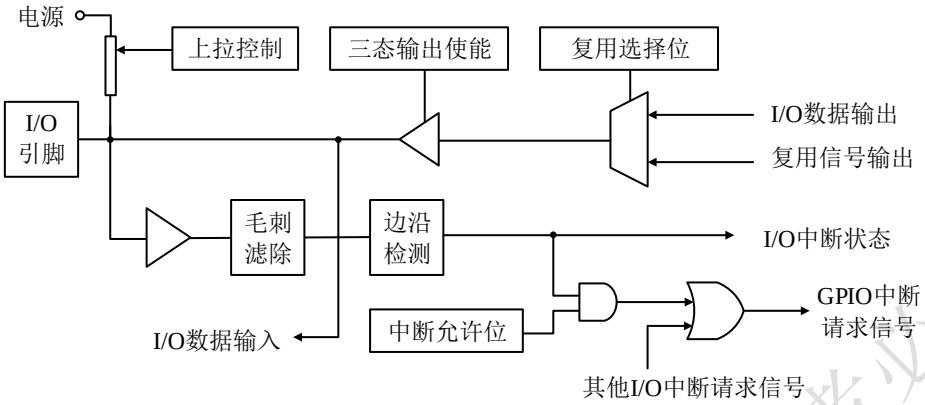


图 4.70　GPIO 典型接口电路示意图

早期计算机系统中使用单独的芯片来控制这些 I/O 引脚，如 Intel 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，有三种工作方式可选，其各个端口的功能可由软件编程控制，使用灵活，可作为计算机与多种外设连接时的接口电路。目前（2020 年）该芯片已经很少使用，但其仍然出现在各类教科书中，作为并行可编程接口的典型示例予以介绍。图 4.71 为 Intersil 公司近期生产的 8255 系列芯片手册中给出的电路框图。此处对图 4.71 内部结构不再做进一步诠释。

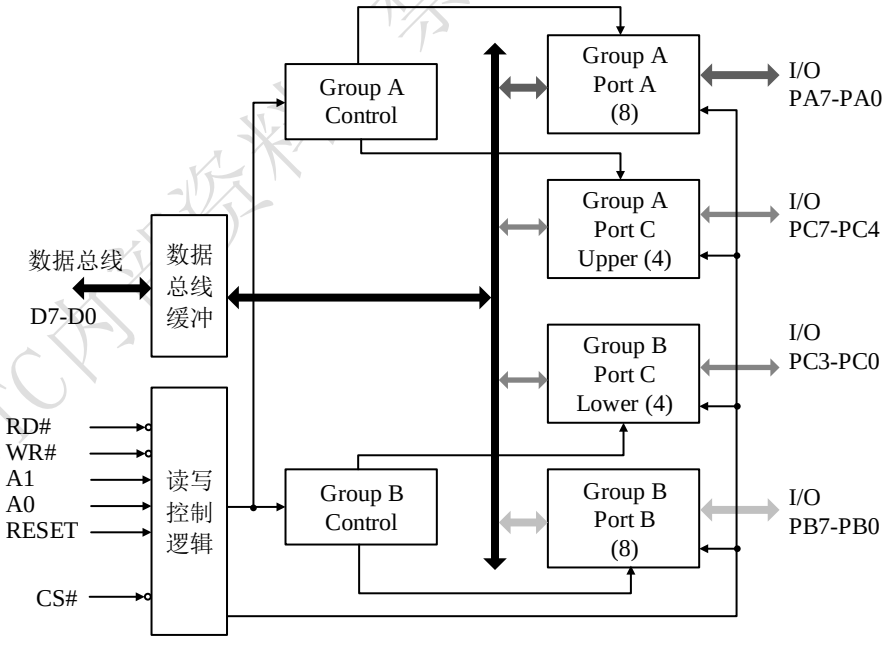


图 4.71　8255 芯片内部电路框图

目前，不同行业广泛使用的微控制器芯片往往都集成 GPIO 接口。一般来说，GPIO 接口包括控制寄存器、输入数据寄存器和输出数据寄存器等。图 4.72 所示为意法半导体 STM32 系列嵌入式控制器一个 I/O 端口位的结构。图中输入数据寄存器的位、输出数据寄存器的位连接

到外部引脚；引脚的信号传输方向、信号类型和是否复用等配置都是通过写控制寄存器（图中没有画出）来实现。

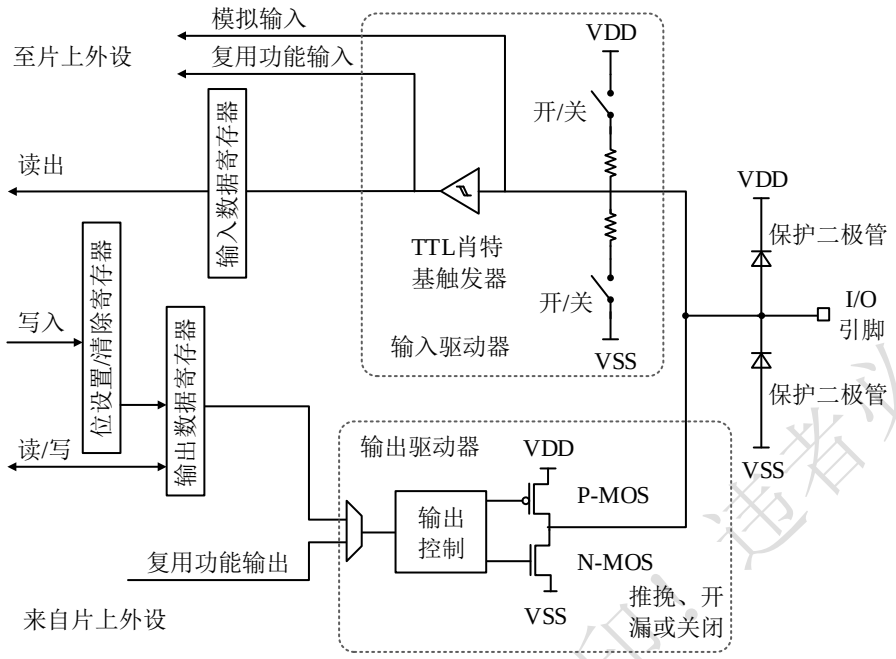


图 4.72　意法半导体 STM32 系列（Cortex-M3）I/O 端口位的基本结构

STM32 系列嵌入式控制器芯片的 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）和推挽复用输出（AF_PP）共八种工作模式，可以满足多种不同场景的应用需求，本书第八章将对 GPIO 部分工作模式做进一步介绍，关于 GPIO 更多工作模式的详细介绍可参阅意法半导体 STM32 用户手册。

4.4.6 串行接口

前述并行接口使用多条信号线同时传输多位数据。而串行数据传输时，数据是一位一位顺序地在信号线上传输。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，总线上的并行数据需要经并/串转换电路转换成串行方式，再逐位送至传输线。接收端则需要把数据从串行方式重新转换回并行方式，才能传送到内部并行数据总线上。如果采用相同的接口时钟频率，串行数据传输的速度显然要低于并行传输。

串行数据传输的过程往往被称作串行通信。如计算机网络中常见的以太网就是以双绞线作传输介质的串行通信系统。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连较为常见的 I^2C 和 SPI 接口。

1. 串行通信概述

串行通信常用于需要远距离传输的情形。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。一般来说，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），把从线路上接收到的调制信号解调还原成数字信号。调制器和解调器两者通常集成在一个设备中，称为调制解调器

（Modem）。在数据通信系统中，调制解调器等通信设备被称为 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 4.73 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。

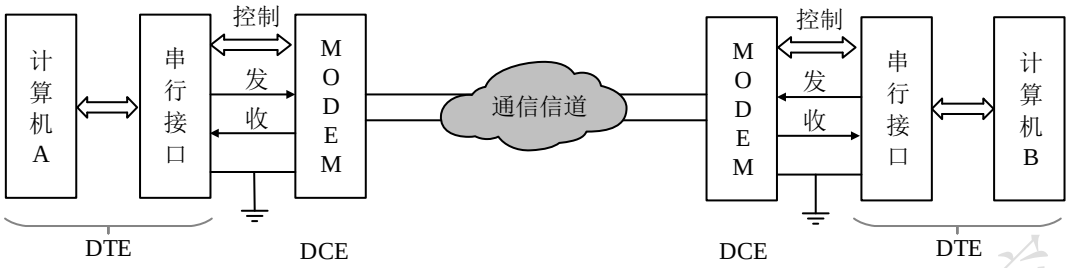


图 4.73　通过 Modem 实现远程数据通信

根据传送方向，串行通信可分为单工（Simplex）、半双工（Half Duplex）和全双工（Full Duplex）三种方式，也称为传送制式。单工指发送方与接收方之间只有一条通信通道（简称信道），而且这条信道永远只能进行一个方向的信息传送（例如无线电广播）。半双工指发送方与接收方之间也只有一条信道，但这条信道在不同时刻能够进行两个方向的传输（例如出租车上的车载电台）。全双工指发送方与接收方之间有两条信道，在同一时刻可以进行双向信息传输（如电话）。

以半双工数据通信为例，仅用单条数字信道传送数据，有时是发送方发送数据给接收方，有时候是接收方反向传送应答信息，并且还希望彼此之间能够可靠通信，这就需要以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，为了实现上述目的，收发双方必须有严格的约定。这些双方共同遵循的约定被称为通信协议。

在通信协议中，为确保通信各方能够准确地发送、接收、识别、理解和利用信息，需要在信号电平、数据格式、差错控制和流量控制等诸多方面做出约定。为了简化协议设计，使通信系统结构更加清晰，通信协议普遍采用分层模型。单纯的接口电路，无论是并口还是串口，电路功能隶属于 OSI 参考模型的最底层——物理层。但一般通信协议中约定的差错控制、流量控制和连接控制方式等则隶属于 OSI 参考模型的链路层。

1） 串行通信的传输速率

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，1 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 1 位（1 bit），此时波特率等于比特率。但在其它一些应用场合，1 个“符号”的信息含量可能超过 1 位，此时波特率小于比特率。

2） 串行通信的同步方式

同步是指能够检测和区分所传送数据单元（位、字符或字节、帧和数据块等）的起和止。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）。与之相应的串行通信总线可分为同步串行总线和异步串行总线。

同步串行总线传输信息时，发送方和接收方在同一个时钟[10]下工作，因而所传输信息的位与位之间、字节与字节之间均与时钟有严格的对应关系。图 4.74 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 8 个数据位。很多同步协议首先发送最高有效位 MSB（Most Significant Bit），而很多异步协议先发送最低有效位 LSB（Least Significant Bit）。图中，发送方在时钟下降沿发送数据，接收方在时钟上升沿接收（采样）数据，PCI 总线采用的就是这种方式，称为“下降沿同步、上升沿采样”，此处“同步”是指发送或者更新数据。

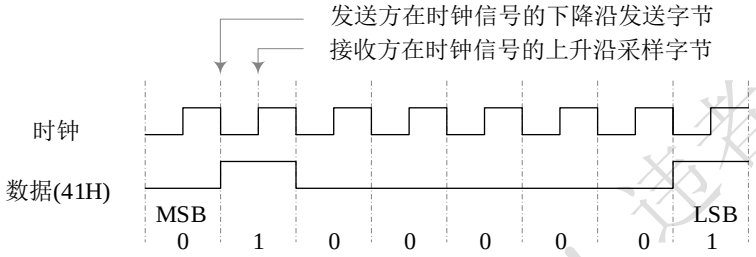


图 4.74　典型的同步传输

异步串行总线与同步串行总线的最大区别在于是否使用公共时钟。异步串行总线的收发双方使用各自独立的时钟，主要依赖波形编码来区分和识别数据的起始和终止，无需使用额外的时钟同步信号。异步串行通信要求在同一字节内各位之间的相对时间关系相对一致，对字节与字节间的时间关系没有要求。因此，异步串行通信收发双方的时钟频率只需基本相同，彼此之间可以存在一定的偏差，但是偏差应在一定范围之内。异步串行通信时，收发双方根据事先约定的波形编码规则和数据格式，按照各自的时钟发送和接收数据。图 4.75 所示是一种串行通信线路上的数据波形编码方案，该方案也称为起止式同步方式。目前在异步串行通信中应用最为广泛的 TIA/EIA RS-232[11]（简称 RS-232）标准采用的就是这种线路波形编码方案。图 4.75 中，“Tb”为发送方传送一位数据的时间长度，也称为码元宽度或码元间隔；EIA 电平是 EIA 颁布的标准中所规定的线路电平。RS-232 标准采用负逻辑，即高电平表示“0”，低电平表示“1”。在 RS-232 标准的子集 RS-232-C 中，高电平范围为+3 ～ +15v，低电平范围为－3 ～－5v。有关起止式同步的具体实现方式稍后讨论。

[10]同步通信所需时钟信号可使用单独的信道传送，这种同步方式称为外同步；也可以采用特定的编码方式，由接收方在接收的码流中提取和恢复时钟，无需使用单独的时钟信道，这种同步方式称为内同步。铜缆以太网采用的就是内同步方式。

[11]EIA（Electronic Industries Association），美国电子工业协会，其推荐的标准都冠以 RS（Recommended Standard），如 RS-232 和 RS422 等。1984 年 EIA 的电信与信息技术组与美国电信供应商协会（USTSA）合并，成立了美国电信工业协会 TIA（Telecommunications Industry Association）。现在 EIA 标准又称为 TIA/EIA 标准。

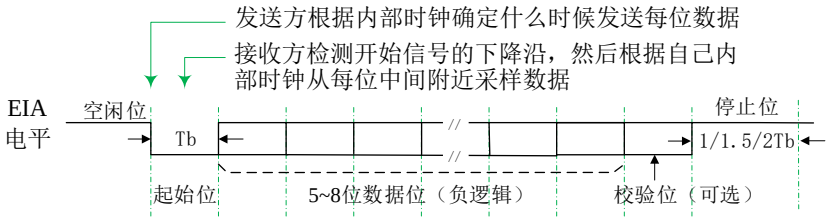


图 4.75　起止式异步串行传输线路波形

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

异步串行通信方式在传输数据时，待传输的数据以字符（一个字符通常含五至八位）为单位进行传送，传输字符也常被称为数据帧。为了能够区分数据帧的起和止，发送方在每个需要传送字符之前增加一个起始位，在每个字符之后增加一个停止位。此外，为了提高传输可靠性，字符和停止位之间可按照收发双方事先的约定（是否需要以及采用奇校验还是偶校验），插入一个奇偶校验位，如图 4.75 所示。图中，发送方在线路空闲（没有数据发送）时，输出高电平。如果有字符需要发送，发送方先发送一个 Tb 宽度的负脉冲作为起始位，然后再以 Tb 为周期，按照先低位后高位的顺序，依次发送字符中的每一位数据以及可能存在奇偶校验位，最后发送停止位。停止位的长度可以是 1 或者 1.5 或者 2 个 Tb（也称为 1 位停止位或者 1.5 位停止位或者 2 位停止位）。接收方串行接口电路一直在检测线路波形，一旦检测到并确认是起始位后，便按照事先约定的格式接收字符以及或有的校验位，而检测到停止位后，就获知一个字符传送已结束。

如果收发双方约定数据帧格式采用七位字符长度，一位偶校验位和一位停止位（简称 7E1 格式），串行通信线路波形如图 4.76 所示。

图 4.76　异步串行通信的数据格式

需要说明的是，图 4.76 所示字符传输格式规定了不同传输事件发生的先后顺序，这属于异步串行接口标准中物理层的规程特性。但是图 4.76 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和时序特征，此处并未展开讨论。

2） 异步传输信号波形的检测

异步通信收发双方采用各自独立的时钟。虽然时钟频率可以设计为一样，但在具体工程实现时，难以保证收发双方时钟的频率完全相等，并且时钟的相位也无法一致。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 为大于等于 1 的整数。有些芯片的 K 可以编程设置，有些则固定，如 K=16。

以下以 K=16 为例，说明接收端的波形检测过程。在图 4.77 中，线路空闲时，接收端检测的都是高电平。当接收端检测到一个低电平时，为慎重起见，还需要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。

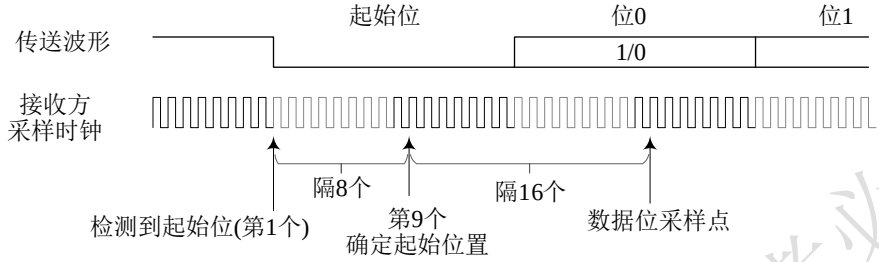


图 4.77　16 倍波特率时钟情形的起始位检测

若确认是起始位，则从第 9 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。

如果收发双方的时钟有偏差，并假设接收方时钟频率高于发送方，则接收方的 Tb 小于发送方的 Tb。在这种情形下，从起始位中点开始，接收方隔 16 个检测脉冲对第一位数据波形进行采样，采样时刻要比理论上的位置略微提前，出现偏差。在接下来的检测过程中，这种偏差逐渐积累，采样时刻不断提前，当检测到最后一个停止位时，采样时刻误差达到最大值。如果最大误差没有导致采样时刻超出停止位的范围，则本次数据接收结果不受时钟偏差的影响，并在检测下一个起始位时，偏差被“清零”。但是如果误差过大，采样时刻提前到停止位的前面一位（最后一位数据或者是校验位），本次字符接收过程就会出现严重错误。因为停止位始终是高电平，但是之前一位有时是高电平，有时是低电平，如果在本应是高电平之处检测到低电平，接口电路就认为出现错误。这种错误被归类为“数据格式错”，属于串行异步接口电路三大线路错误之一。

3） 异步串行接口电路

异步串行接口电路主要由 UART（Universal Asynchronous Receiver/Transmitter，通用异步收发传输器）和线路收发驱动器（Line Driver）构成。早期 PC 机中普遍使用的 UART 是 INS 8250，8250 的特点是每收发一个字符都需要 CPU 为其进行服务。后来 NS（National Semiconductor）公司推出了新的 UART 产品 PC 16550，在 INS 8250 的基础上增加了数据收发 FIFO 寄存器，并在芯片引脚以及内部寄存器等方面与 INS8250 保持兼容。PC16550 可在发送或者接收一组字符之后才需要 CPU 或者 DMAC 为其服务一次。之后又有一些公司对 PC16550 做了进一步的改进，改进之处包括扩大 FIFO 容量、节能设计和增加流量控制功能等，并且与 16550 在芯片引脚和内部寄存器等方面保持兼容。这些产品被命名为 xx16650、xx16750、xx16850 和 xx16950 等，因此，这类 UART 被统称为 16x50。

UART 芯片大都采用 TTL 电平，并且输入输出都是正逻辑，为了能够在串行通信线路上进行远距离传输，需要使用线路收发驱动器在正负逻辑以及 TTL 电平与 EIA 电平之间进行转换。在早期PC机中，串行通信接口所使用的线路驱动器（如摩托罗拉公司的MC1441和MC1442）

需要±12V 电源供电，增加了 PC 机电源系统的复杂性。现在广泛使用集成式线路驱动器，只需使用单一＋5V 电源，如 Maxim 公司的 MAX232 和 MAX233 等。

以下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。

一般而言，异步串行接口电路需要完成的基本功能包括：数据的串/并、并/串转换；串行数据的格式化（如加入起始位、校验位或同步字符等）；控制信号解析等。

典型串行通信接口电路如图 4.78 所示。图 4.78 中的 RxD 信号是串行数据接收，TxD 是串行数据发送，IRQ 是中断请求。图中 RTS、CTS、DTR、DSR、DCD 和 BELL 是异步串行接口与 Modem 之间的 6 条联络信号线，其含义可参见表 4.17。

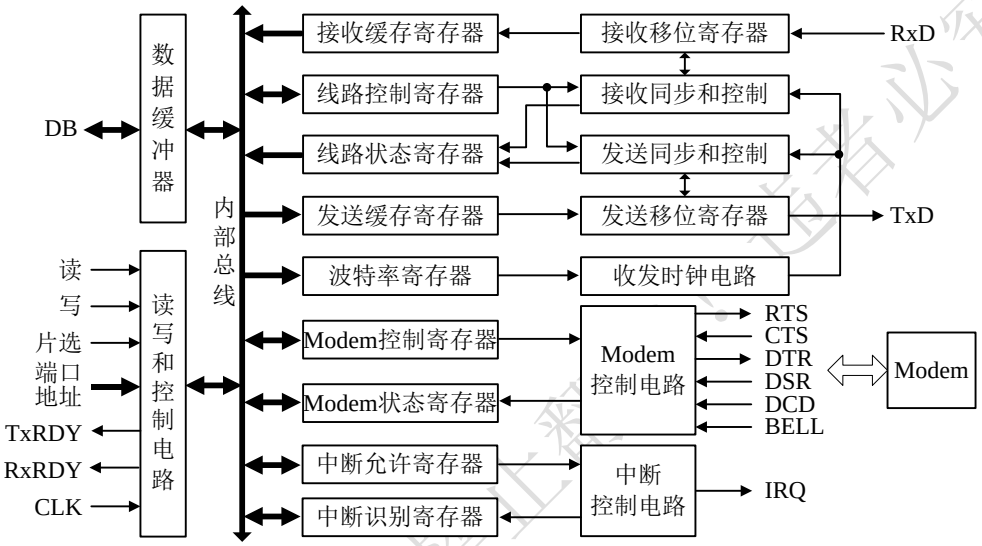


图 4.78　异步串行接口框图

以下以 CPU 通过图 4.78 所示串行接口进行数据收发为例，简介数据收发过程以及各个主要模块和信号线的作用。

- 数据发送过程：发送移位寄存器按照预先设置的波特率，在发送同步和控制电路的作用下，将来自发送缓存寄存器的并行数据，一位一位串行移出到 TxD 上，同时插入起始位、奇偶校验位和停止位等。由 TXD 引脚发送当最后一位数据移出之后，发送缓存寄存器将下一个待发送数据再传送到发送移位寄存器，继续发送。当发送缓存寄存器将数据输出到发送移位寄存器之后，以中断方式或者 TxRDY 引脚有效的形式，通知 CPU 将新的数据写入发送缓存寄存器，写入后撤销中断请求或 TxRDY 变为无效。在上述过程中，如果发送移位寄存器最后一位数据移出后，发送缓存寄存器中是空的，没有待发送数据，就会产生另外一种名为“发送缓冲器空（TxEmpty）”的中断，请求 CPU 进行处理。关于 TxRDY 与 TxEmpty 的区别请读者自行分析。
- 数据接收过程：在接收同步和控制电路的作用下，接收移位寄存器逐位检测和接收来自 RxD 引脚的数据，同时进行错误检测和帧解析并把相关信息写入状态寄存器，然后把接收到的数据写入接收缓存寄存器，再通过中断或者 RxRDY 引脚有效的形式，通知 CPU 可以读取已经接收的数据，读出后撤销中断请求或者 RxRDY 变为无效。在接收过程中，如果 CPU 还未将上一次接收的数据读出，接收缓存寄存器又被写入新接收的数据，就会产生中断请求，向 CPU 报告接收过程出现了错误。此外，接收电路在对输入数据进行检

测时，如果发现奇偶校验错、数据格式错以及数据接收过程中出现长时间停顿，都会产生中断，请求 CPU 予以处理。不同类型的中断都记录在中断识别寄存器中，CPU 收到中断请求后，在中断服务程序中读取中断识别寄存器，在根据具体中断原因跳转到相应的处理程序进行处理。

- ❒ 波特率发生器根据用户设置的参数，将输入时钟信号进行分频，产生数据接收和移位所需的时钟。
- ❒ 线路控制寄存器用于设置传输字符结构、停止位和校验位等。
- ❒ Modem 控制电路用于主机串行接口（DTE）与 Modem（DCE）设备之间的协调，需要协调的内容有 DTE 和 DCE 是否就绪？是否有呼叫（振铃）到达？数据链路是否已经建立？以及 DTE 和 DCE 之间的流量控制（Traffic Control）。实现上述协调工作所需的信号线参见表 4.17。

对于用户而言，了解上述接口电路的基本工作过程是非常必要的。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。收发双方仅使用 TXD/RXD 是可以进行数据收发的，但不能进行流控制。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）串行接口标准（协议）

接口标准（协议）是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约定。任何一个串行接口协议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。常用的串行接口标准有 TIA/EIA 推荐的 RS-232、RS-449、RS-422、RS-423 和 RS-485 等。这些标准在物理层的差异主要体现在机械特性和电气特性。由于之前已对串行通信主要的规程特性和功能特性做了概述，以下仅对这些标准的电气特性和机械特性进行简要介绍。

- ❒ RS-232-C 标准

RS-232 是由 EIA 于 1962 年制定的串行二进制数据交换接口技术标准[12]。RS-232 先后有多个版本，其中应用最广的是修订版 C，即 RS-232-C。完整的 RS-232 标准定义了 22 条信号线，用 25 芯 DB（Distribution Board）插座连接，主机端为插头，而电缆端为插座。22 条信号线包括一个主信道组和一个辅助信道组，但是绝大多数情况下仅使用主信道组。因此，RS-232 标准可以简化为只需八条信号线和一根地线，并采用体积较小的九芯 DB9 插座。DB9 和 DB25 连接器引脚针号的对应关系以及所涉及到的信号线名称和主要含义如表 4.17 所示。表中的输出和输入方向都是针对 DTE 而言。

表 4.17　RS-232C 信号定义

DB9 针号	DB25 针号	代号	方向	功能说明
1	8	DCD	IN	Data Carry Detected，数据载波检测（数据链路已建立）
2	3	RxD	IN	接收数据
3	2	TxD	OUT	发送数据
4	20	DTR	OUT	DTE Ready，数据终端就绪
5	7	GND	——	Signal Grand，信号地

[12] 与 RS-232 对应的是 ITU（International Telecommunication Union，国际电信联盟）颁布的 V.24 标准。欧洲各国主要采用 V.24 标准。

6	6	DSR	IN	DCE Ready，数据通信设备就绪
7	4	RTS	OUT	Request To Send[13]，请求发送，全双工模式下用于流量控制
8	5	CTS	IN	Clear To Send，清除发送，全双工模式下用于流量控制
9	22	BELL	IN	Ring Indicator，振铃指示

RS-232-C 的电气特性参见表 4.18，其有效传输距离与负载的等效电容值有关。RS-232-C 标准规定，终端负载等效电容，包括传输电缆电容必须小于 2500pF。对于多芯电缆，每英尺（约 0.305 米）等效电容为 40~50pF，所以满足要求的电缆长度最长为 50 英尺（约 15 米）。如果使用特制低电容屏蔽电缆，RS-232-C 的最大传输距离可以延长到 1500 英尺（150 米）。RS-232-C 的设计目标是点对点通信，其负载电阻为 3KΩ~7KΩ，适用于本地设备间的短距离低速通信。早期 RS-232-C 的接口速率（波特）标准只有 150、300、600、1200、2400、4800、9600、14400 和 19200 几个等级，后来扩展到 28800、33600、57600 和 115200。

❒ RS-449/RS-422/RS-423 标准

由于 RS-232-C 采用的是单端非平衡传输方式，存在共地噪声并且无法抑制共模干扰，因此传输距离短并且传输速率慢。为了弥补 RS-232 的缺点，1977 年 EIA 制定了 RS-449 标准（在 ITU 标准体系中对应标准为 V.35）。RS-449/V.35 标准除了保留与 RS-232-C/V.24 兼容的特点外，还在提高传输速率，增加传输距离及改进电气特性等方面作了很大努力。RS-449/V.35 在 RS-232/V.24 的基础上增加了 10 个控制信号，连接器也相应地改为 DB37 规格。RS-449/V.35 标准广泛应用于各种电信通信设备接口中。

与 RS-449 同时推出的还有 RS-422 和 RS-423，它们都属于 RS-449 标准的子集。RS-422 标准采用平衡驱动差分接收电路，提高了数据传输速率，增加了传输距离。RS-422 标准属于一种单机发送、多机接收的单向平衡传输规范，允许在一条平衡总线上使用一个发送器，最多挂载 10 个接收器，但是接收器之间彼此不能通信。如果两点之间需要实现全双工通信，必须另外增加一对逆向的收发驱动器及相应的平衡传输线路。

RS-422 最大传输距离与数据传输速率互相制约，实际能够达到的最大传输距离也与所使用的信号线品质有关。当数据传输速率小于 100Kbps 时，如果所使用的双绞线扭距（双绞线两扭之间的距离）小、线径粗、线路损耗小，RS-422 最大传输距离可达 4000 英尺（约为 1220 米）；如果进一步降低传输速率并且使用更粗线径的金属铠装屏蔽电缆，上述距离可以扩展到 6000 英尺（约为 1830 米）.如果传输距离在 50 英尺（约为 15 米）以内，RS-422 的最大传输速率可达 10Mbps。

RS-422 所使用的传输电缆特性阻抗为 120Ω±15Ω，当传输距离超过 300 米时，在传输线远端应该加装 120Ω电阻进行终端阻抗匹配，防止出现信号端点反射影响信号传输。RS-422 标准的主要电气特性传输参见表 4.18。

RS-423 标准的设计目标是为了解决与 RS-232 标准的兼容问题。RS-423 采用单端输出驱动和双端差分接收方式，RS-232 接口的单端输出信号可以连接双端差分接收的 RS-422 接口，从而实现新老两种体制接口之间的互连。一条 RS-423 总线最多可以挂载 10 个接收器，也只能进行单向传输。RS-423 的传输距离和最大传输速率均低于 RS-422，其最大传输速率为 300Kbps，最远传输距离可达 2000 英尺（600 米）。由于 RS-423 应用较少，所以不再赘述。

[13] 在半双工通信模式下，Modem（DSE）平时处于接收状态（与出租车车载电台的工作模式相同）。当 DTE 设备需要发送数据时，使用 RTS 通知 DSE 转换到发送模式，DSE 转换完成后使用 CLS 告诉 DCE。

❒ RS-485 标准

EIA 于 1983 年推出的 RS-485 标准可以认为是 RS-422 的增强版，也是采用平衡发送和差分接收技术，以提高总线的抗干扰能力。与 RS-422 不同的是，RS-485 采用半双工双向通信，一条总线上最多可以连接 32 个发送器，使其具有多点通信能力，能够在多个节点之间实现网络互连。由于同一时刻总线上最多只能有一个节点处于发送状态，因此，RS-485 还提供了总线仲裁能力，并且总线上所有发送器都具备三态功能。

RS-485 最大传输距离和最大传输速率与 RS-422 相同，如果需要进一步增加传输距离，可以使用 RS-485 中继器扩展传输距离。RS-485 电缆的特性阻抗也是 120Ω±15Ω，也需要在距离最远的两端，各加装一个终端阻抗匹配电阻（120Ω）以防止出现端点反射。

由于 RS-485 标准所具有的优良特性，面世之后很快就在仪器仪表、自动化、工业过程控制和建筑智能化等众多领域得到了广泛的应用。RS-485 标准主要电气特性参见表 4.18。

表 4.18　RS-232/RS-485/RS-422 特点对比

	RS-232-C	RS-422	RS-485
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 个发送器+1 个接收器 1 transmitter + 1 receiver	5 个发送器+10 个发送器 5 transmitters 10 receivers	32 个发送器+32 个发送器 32 transmitters 32 receivers
双工方式 Communication Mode	全双工 full duplex	全双工/半双工 full duplex / half duplex	半双工 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 10 Mbps 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	电平范围-6~-2V	电平范围-5~-1.5V
逻辑“0” Space (data 0)	电平范围 5~15V	电平范围 2~6V	电平范围 1.5~5V
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）是 Philips 公司 1982 年开发一种同步串行总线。广泛用于处理器与外设之间、或不同外设模块之间的连接，这些外设可以是存储芯片、A/D 芯片、LED、LCD 等。例如，图 4.79 所示为微芯（Microchip）公司微控制器芯片 dsPIC30F 与存储芯片 24LC256 通过 I²C 连接的电路示意图。

I²C 使用两根线实现多个器件之间的信息传送，这两根信号线分别是 SCL（时钟信号线）与 SDA（数据线）。SDA 和 SCL 允许被多个器件驱动，故驱动 SDA 和 SCL 的器件其输出级必须采用漏极开路（OD 门）结构，以实现总线的“线与”功能。另外，需要使用了外部上拉电阻，以确保没有器件驱动（将信号拉低）时信号线能保持在高电平。

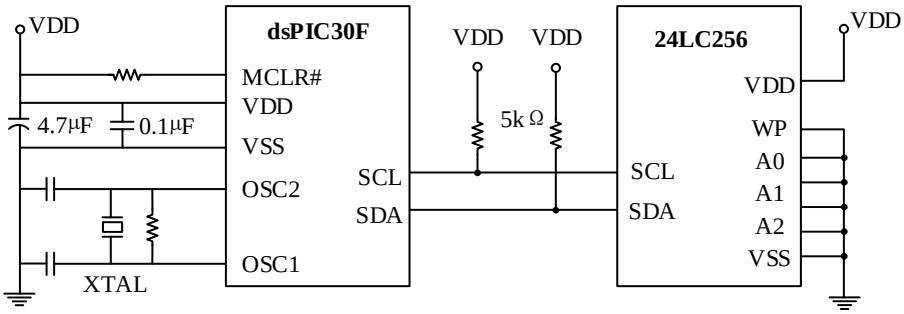


图 4.79　通过 I²C 连接两块不同的芯片

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，可以滤去信号线上的毛刺。因此使用 I²C 总线有利于简化 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，用 I²C 作为接口的电路模块，很容易标准化和模块化，便于重复利用。

1) I2C 仲裁机制

I²C 采用主从式通信方式。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）启动总线传送，并产生时钟；被寻址的器件为从器件（从机，或称为从设备）。

在图 4.80 所示的两根信号线上，可以同时挂接多个器件（如单片机、存储器、键盘、LED、时钟模块和 ADC 等）。为了能区别挂接在总线上的不同器件，每个 I²C 设备都有一个地址，地址有七位和十位两种定义。任何器件既可以作为主机也可以作为从机，但同一时刻 I²C 总线上只允许有一个主机。

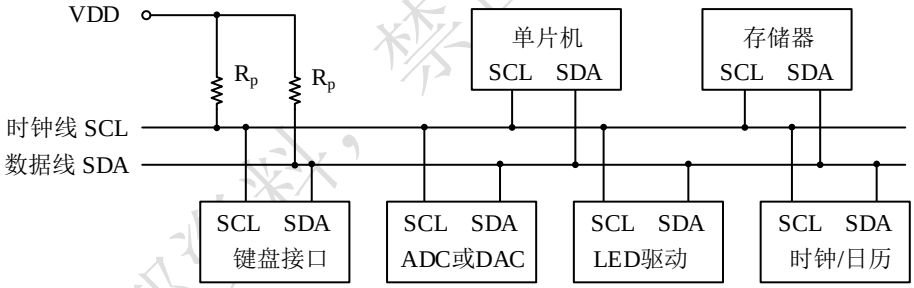


图 4.80 连接了多个 I²C 器件的电路示意图

挂接在 I²C 总线上的多个主机竞争使用 SCL 和 SDA。采用“线与”逻辑互连时，任一器件输出低电平，信号线就呈现为低电平；只有所有器件都输出高电平时，信号线才呈现为高电平。对于时钟信号线 SCL，被多个主机驱动时，SCL 呈现为多个时钟合成后的统一时钟。而对于数据线 SDA，假设主机 P 需要使用总线，当主机 P 检测到总线处于空闲状态（SCL 和 SDA 均为高电平，稍后将要介绍）后，主机 P 便向 SDA 信号线发送数据（高电平或低电平），每一位数据发送之后主机 P 检测 SDA 信号线电平，如果发现与自身发送的电平不一致，说明有其他主机同时也在发送数据，主机 P 则关闭其输出转为从机。

2) I2C 定义的状态

SCL 与 SDA 信号线上高电平、低电平、电平变化的不同组合具有特定含义。如 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。预定义的总线条件（状态）如图 4.81 所示，以下对其中各个状态进行简要说明。

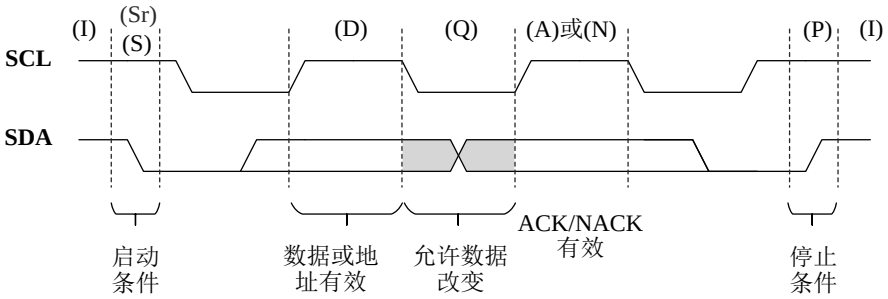


图 4.81 I²C 协议定义的状态

❒ 总线空闲（I）

SDA 和 SCL 两条信号线同时为高电平的状态，是空闲状态，如图 4.81 中状态 I。

❒ 启动数据传输（S）

SCL 为高电平时， SDA 由高电平变为低电平（即负跳变）会产生“启动”条件，如图 4.81 中状态 S。启动信号时序标志着一次数据传输的开始，由主机建立，主机须确保建立该信号时序前 I²C 总线处于空闲状态。

❒ 停止数据传输（P）

SCL 为高电平时，SDA 由低电平变为高电平（即正跳变）会产生“停止”条件，如图 4.81 中状态 P。停止信号时序标志着一次数据传输的终止，由主机建立，建立该信号时序后，总线进入空闲状态。

❒ 等待/数据无效（Q）

在启动条件之后，SCL 低电平期间，总线处于“等待”状态，允许 SDA 电平改变（修改线上数据），如图 4.81 中状态 Q。

❒ 重新启动（Sr）

在“等待”状态后，SCL 为高电平时，SDA 由高电平变为低电平会产生“重新启动”条件。如图 4.81 中状态 Sr。重新启动能让主机在不失去总线控制的情况下改变数据传输方向。主机控制总线完成了一次数据传输后，如果想继续占用总线再进行一次数据传输，就需要使用重新启动信号时序。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。

❒ 数据有效（D）

在启动条件之后，SCL 高电平期间，SDA 所呈现的电平代表了有效数据。一个数据位的传输需要一个时钟周期，每个时钟周期内 SCL 高电平期间，SDA 上的电平必须保持稳定，SDA 低电平表示数据“0”、高电平表示数据“1”。如图 4.81 中状态 D。

❒ 应答（A）或非应答（N）

所有的传输数据须由接收方应答（ACK）或非应答（NACK）。接收方将 SDA 驱动为低电平发出 ACK，表示接收成功可继续发送；或释放 SDA（呈现为高电平）发出 NACK，表示不要再继续发送了。应答或非应答信号占用一个时钟周期。传输数据以八位（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 SDA，由接收方反馈一个应答信号。

3） I2C 总线数据传输过程

图 4.82 是典型的 I²C 总线数据传输过程。下面以主设备向从设备发送信息为例，介绍图

4.82 的各个阶段和写时序的具体步骤。

① 主设备发送开始信号，对应图 4.82 中 S 状态；

② 主设备发送七位的从设备地址，对应图 4.82 中发送地址阶段；

③ 主设备发送写命令（W#，低电平），对应图 4.82 中 R/W#；

④ 从设备应答，ACK 表示有这个设备，设备就绪；

⑤ 主设备发送八位数据；

⑥ 从设备应答，ACK 表示成功接收，可以继续发送；

⑦ 如果从设备应答 ACK，主设备继续发送数据；

⑧ 如果从设备应答 NACK，主设备发送停止信号，对应图 4.82 中 P 状态。

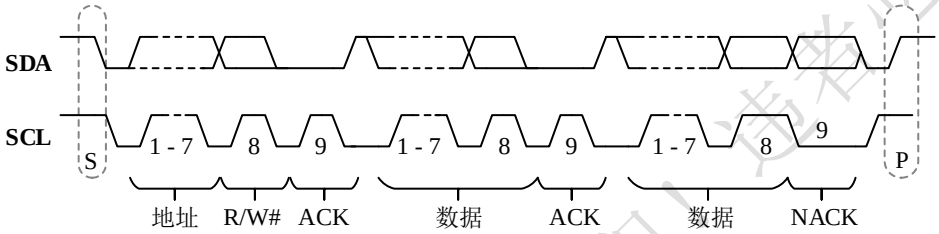


图 4.82 I²C 总线上的数据传输过程的时序

主设备读取从设备信息的时序与上述写时序步骤类似，此处不再赘述。

为了更清晰地描述，一些资料将图 4.82 所示的时序表示为 SDA 信号线上的位顺序图，如图 4.83 所示。图 4.83 中，深色部分为主设备上 SDA 信号线发送的位，浅色部分为从设备向 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 4.82 中无法体现，而图 4.83(a)和(b)显示了主设备写和主设备读两种操作的差异。

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

(a) 主设备向从设备写信息时SDA信号线的位顺序

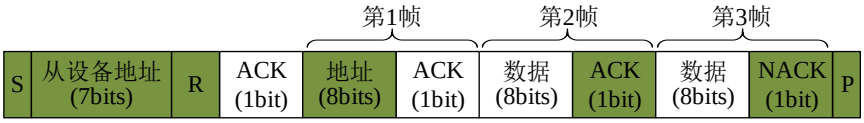
S (1bit)	从设备地址 (7bits)	R (1bit)	ACK (1bit)	数据 (8bits)	ACK (1bit)	数据 (8bits)	NACK (1bit)	P (1bit)

(b) 主设备读取从设备信息时SDA信号线的位顺序

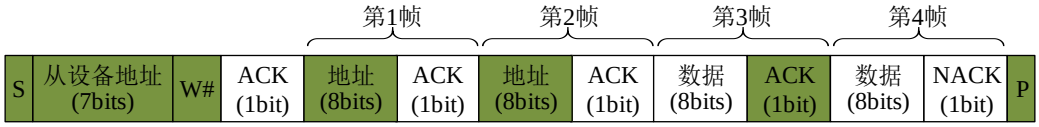
图 4.83 I²C 的 SDA 信号线上的位顺序

每次发送地址/数据到收到应答信号所构成的一个周期称为帧（frame），I²C 总线上从开始 S 状态和停止 P 状态之间以帧为单位传递信息。例如，图 4.83 中(a)，S 状态后先发送了写命令 W#，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。

I²C 总线上的设备内部往往有存储单元，如 EEPROM、RAM，或者用于芯片配置的多个寄存器等。访问这些设备中不同存储单元时，要先提供拟访问数据单元的地址，然后才可以得到相应存储单元内的数据。如图 4.84(a)所示，如果从设备的存储单元地址是八位，那么传递的第一帧是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 4.84(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。



(a) 主设备读取从设备信息（存储单元地址为8比特）的位顺序



(b) 主设备向从设备写信息（存储单元地址为16比特）的位顺序

图 4.84　通过 I²C 访问存储器类器件时的位顺序

4) I2C 接口典型电路

不同厂家实现的 I²C 接口电路都会包括数据寄存器、控制寄存器、状态寄存器和地址寄存器，有些还内置了用于时钟控制的分频寄存器。以下以意法半导体的 STM32 系列微控制器为例，其中所集成的 I²C 接口电路如图 4.85 所示。

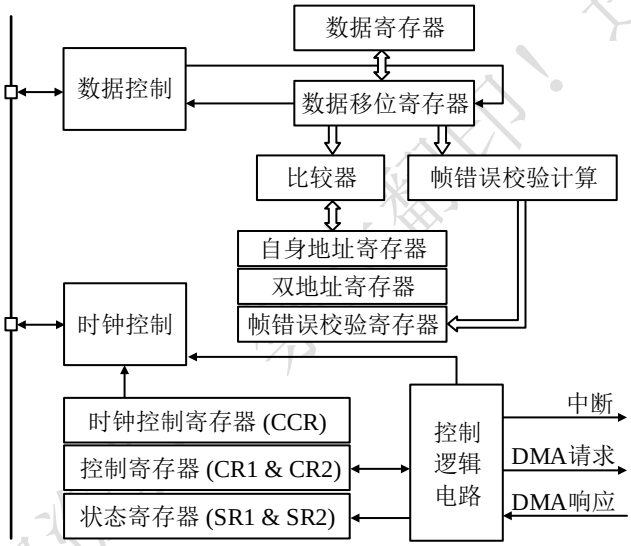


图 4.85　意法半导体 STM32 系列处理器的 I²C 功能示意图

发送数据时，CPU 通过内部总线将需要发送的数据写入 I²C 接口中数据寄存器。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 4.85 中有两个地址寄存器，自身地址寄存器保存的是本设备的地址，而双地址寄存器是该系列芯片独特的设计，使芯片可保存另外一个地址，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负责实现，此功能也并非所有厂家的产品都有。

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。该芯片还支持中断方式和 DMA 方式的数据访问，有关这两种模式的具体内容可参阅本章 4.4.2 小节。

5） 示例：通过 I2C 访问 EEPROM

图 4.79 给出了 dsPIC30F 与 24LC256 通过 I²C 连接的电路示意图。dsPIC30F 是微芯公司生产的微控制器，24LC256 是该公司生产的 256Kb（32K×8bits）串行 EEPROM。微控制器从 EEPROM 中读取数据的过程如图 4.86 所示。

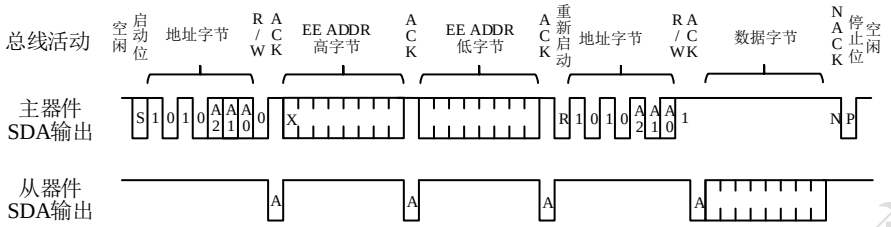


图 4.86 通过 I²C 访问 EEPROM 的报文示意

图 4.86 中，dsPIC30F 作主器件，24LC256 作从器件，需要读取的是 24LC256 内指定存储单元中的字节。由于访问存储器需要先发送存储单元地址，然后再传输数据，可以把 dsPIC30F 访问 24LC256 的过程分成四个阶段：①主器件 dsPIC30F 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 dsPIC30F 输出拟访问的存储单元地址（16 位），从器件 24LC256 应答。③重复启动改变传输方向，主器件 dsPIC30F 输出 24LC256 芯片的地址（I²C 地址），并发读指令，从器件 24LC256 应答。④从器件 24LC256 发出所指定存储单元的内容，主器件应答。

4. SPI 接口及总线

SPI（Serial Peripheral Interface，串行外设接口）是一种全双工的同步通信总线，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。广泛用于微控制器、存储芯片、显示模块、A/D 转换器等芯片间互连。

1） SPI 概述

SPI 使用四根信号线，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。

有些文献中，有时 MOSI 也称作 SDO，MISO 也称作 SDI，SCLK 也称作 SCK，CS 称作 SS 或 NSS，这些只是名称不同而已。

SPI 上可以挂载一个主设备和多个从设备，如图 4.87 所示。任何时刻，一个主设备只能与一个从设备通信。主设备利用 CS#信号选择需要与之通信的从设备。如果 SPI 上只有一个主设备和一个从设备，可以不使用 CS#信号，从设备的 CS#引脚接低电平即可。

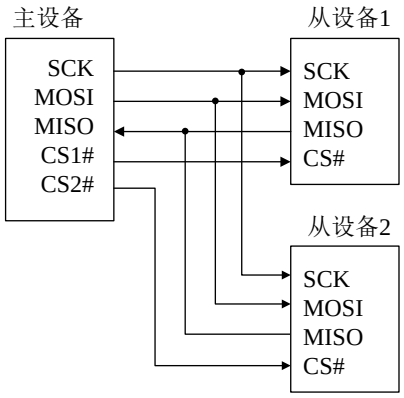


图 4.87　SPI 中主设备与从设备的典型连接

2） 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。当主设备需要向某个从设备发送数据时（简记为 Master→Slave），或者需要读取某个从设备输出的数据时（简记为 Slave→Master），主设备先将对应从设备的 CS#置为低电平，通知其参与接下来的数据传送。如果是 Master → Slave 传送，主设备在时钟信号的控制下，将数据逐位发送到 MOSI 信号线上，从设备采样和接收 MOSI 上的数据；如果是 Slave → Master 传送，被 CS# 选中的从设备在时钟信号的控制下，将数据逐位发送到 MISO 信号线上，主设备采样和接收 MISO 上的数据。

SPI 协议规定，传输时高位在前，低位在后。发送方可以选择在时钟下降沿同步（发送下一位数据），此时接收方应该在下一个时钟上升沿采样数据，如图 4.88 所示。发送方也可以选择在时钟上升沿同步，接收方则必须在时钟下降沿采样。

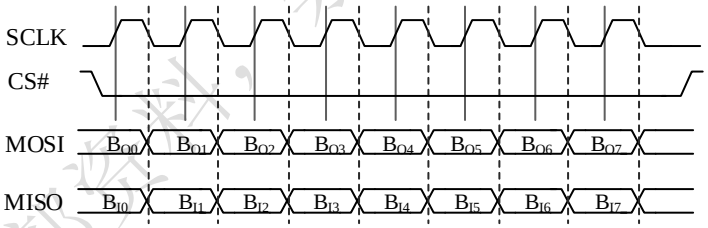


图 4.88　SPI 的一次通信过程（上升沿采样/下降沿同步）

3） SPI 典型接口电路

SPI 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、状态寄存器和时钟配置寄存器等。图 4.89 所示为意法半导体 STM32 系列 MCU 中集成的 SPI 接口电路。该电路定义的四个引脚名称分别为 MISO、MOSI、SCK 和 NSS（即 CS#）。

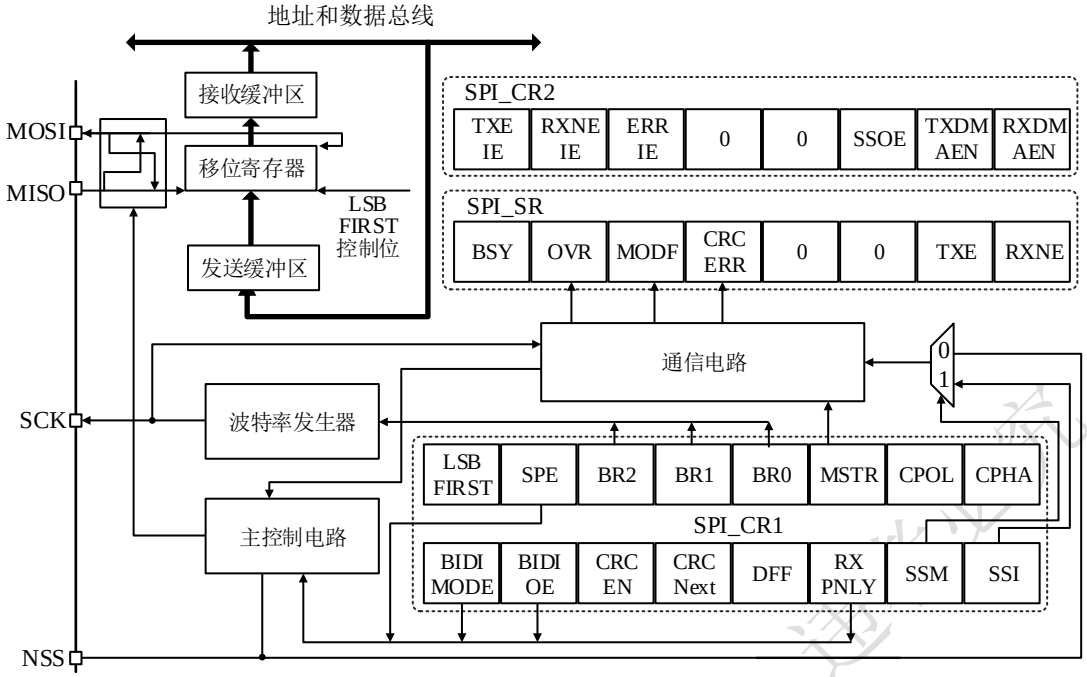


图 4.89　意法半导体 STM32 系列处理器 SPI 接口电路

图 4.89 中发送缓冲区、移位寄存器和接收缓冲区与一般串行通信接口电路大体一致。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 在不同厂家的 SPI 接口电路中都有，但寄存器名称和某些控制位的定义往往略有差异。关于 STM32 系列处理器 SPI 接口电路更多内容请参见本书第八章。

4.1　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，不采用总线分时复用。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　对比面向 CPU 的双总线结构与面向存储器的双总线结构，阐述优缺点。

4.7　总线性能的评价指标有哪些？

4.8　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？它们各有什么特点？

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 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 4.21 中 HREADY 信号的作用。

4.24 AHB 突发传输有什么特点？

4.25 AHB 突发传输定义了哪些类型？各自有什么特点？

4.26 考虑主机接收从机数据，画出 AHB 中“INCR4”类型突发传输的时序。

4.27 考虑主机向从机发送数据，画出 AHB 中“WRAP8”类型突发传输的时序。

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？

4.42 简述中断处理的流程。

4.43 分析图 4.54 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 4.55 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 什么是中断向量表？

4.46 数据块传送方式的 DMA 适用于什么场景？

4.47 常用的中断优先级管理方式有哪几种？分别有哪些优缺点？

4.48 微机与外设的几种输入输出方式中，哪种便于 CPU 处理随机事件？哪种有利于提高 CPU 效率？哪种数据传输速率最快？

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。

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检测信号波形？

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 有哪些措施可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。