



第六章 单片机输入输出 接口及系统扩展设计

6.5 并行输入输出接口的扩展

概述

- 标准的MCS-51单片机具有四个并行I/O端口，对于一般的应用而言已基本够用；
- 对于需要大量I/O端口的复杂应用，或进行了系统总线扩展的单片机(P0、P2和P3.6、P3.7被占用，作为数据总线、地址总线和读写控制线)，则单片机I/O端口的数量将会比较紧张；
- 此时单片机系统就需要进行I/O端口的扩展；
- 本节主要介绍两种常用的I/O端口扩展的方法。



6.5.1 通过串行口扩展并行输入输出接口

1、通过串行口扩展并行输出接口

● 设计原理

- 设置串行口工作在方式0（同步移位寄存器），此时串行口的TxD输出移位时钟，RxD输出数据；
- 外接一个串行-并行移位寄存器74LS164实现串-并变换，可扩展出一个8位的并行输出端口；
- 74LS164可级联扩展更大规模的并行输出接口。

单片机原理及系统设计

6.5.1 通过串行口扩展并行输入输出接口

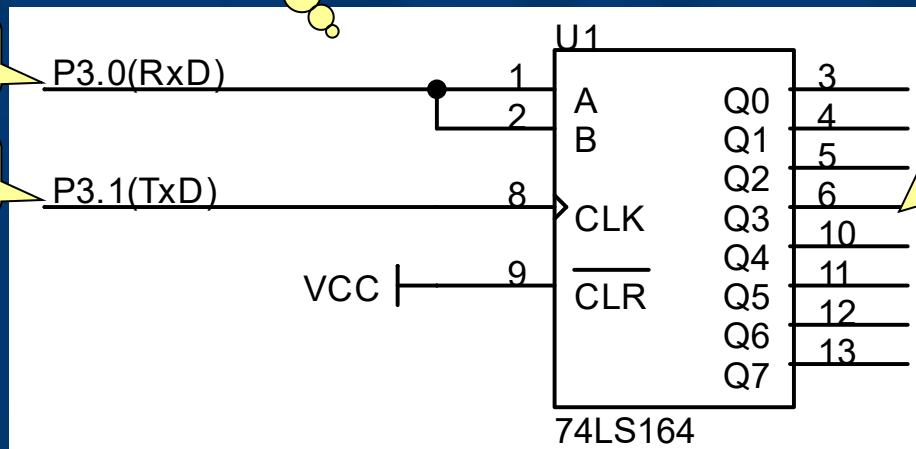
1、通过串行口扩展并行输出接口

● 单片机串行口扩展并行输出接口电路图

单片机串行口
工作于方式0

RxD输出数据

TxD输出移位脉冲



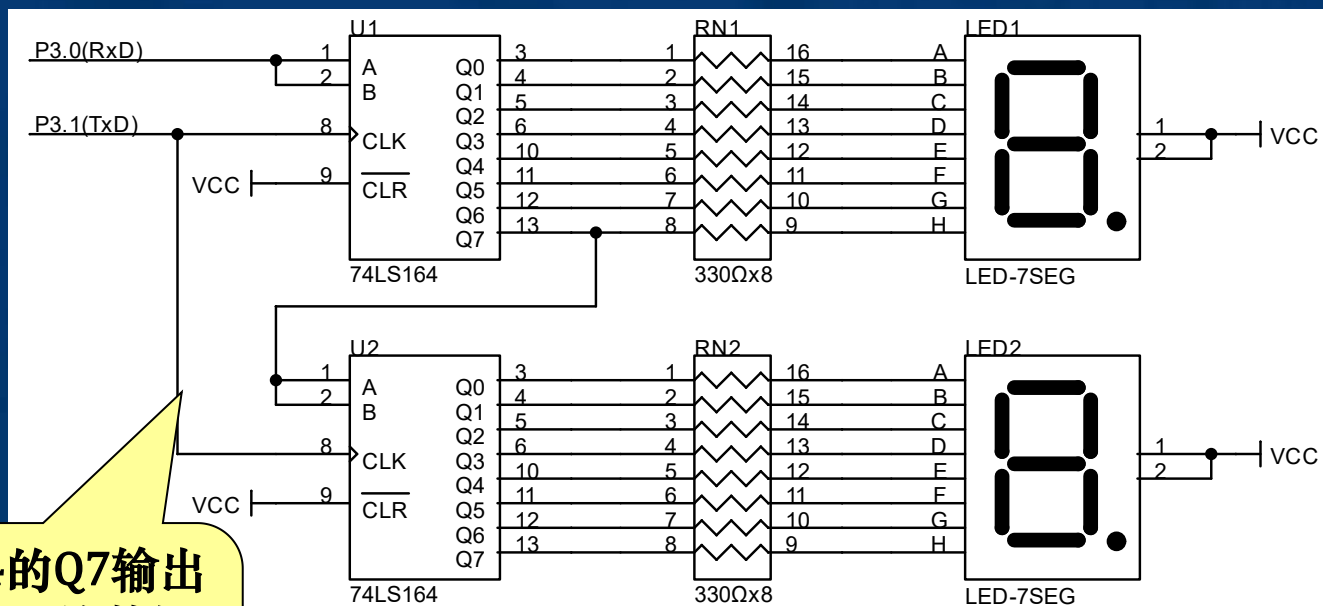
单片机通过串行口输出一个字节，经8次移位脉冲后输出到Q0~Q7；由于串口先输出D0，因此Q0~Q7字节序倒置，对应D7~D0

单片机原理及系统设计

6.5.1 通过串行口扩展并行输入输出接口

1、通过串行口扩展并行输出接口

- 单片机串行口级联扩展并行输出接口电路图



第一个164的Q7输出到第二个164的数据输入，移位时钟共用，可实现级联

6.5.1 通过串行口扩展并行输入输出接口

2、通过串行口扩展并行输入接口

● 设计原理

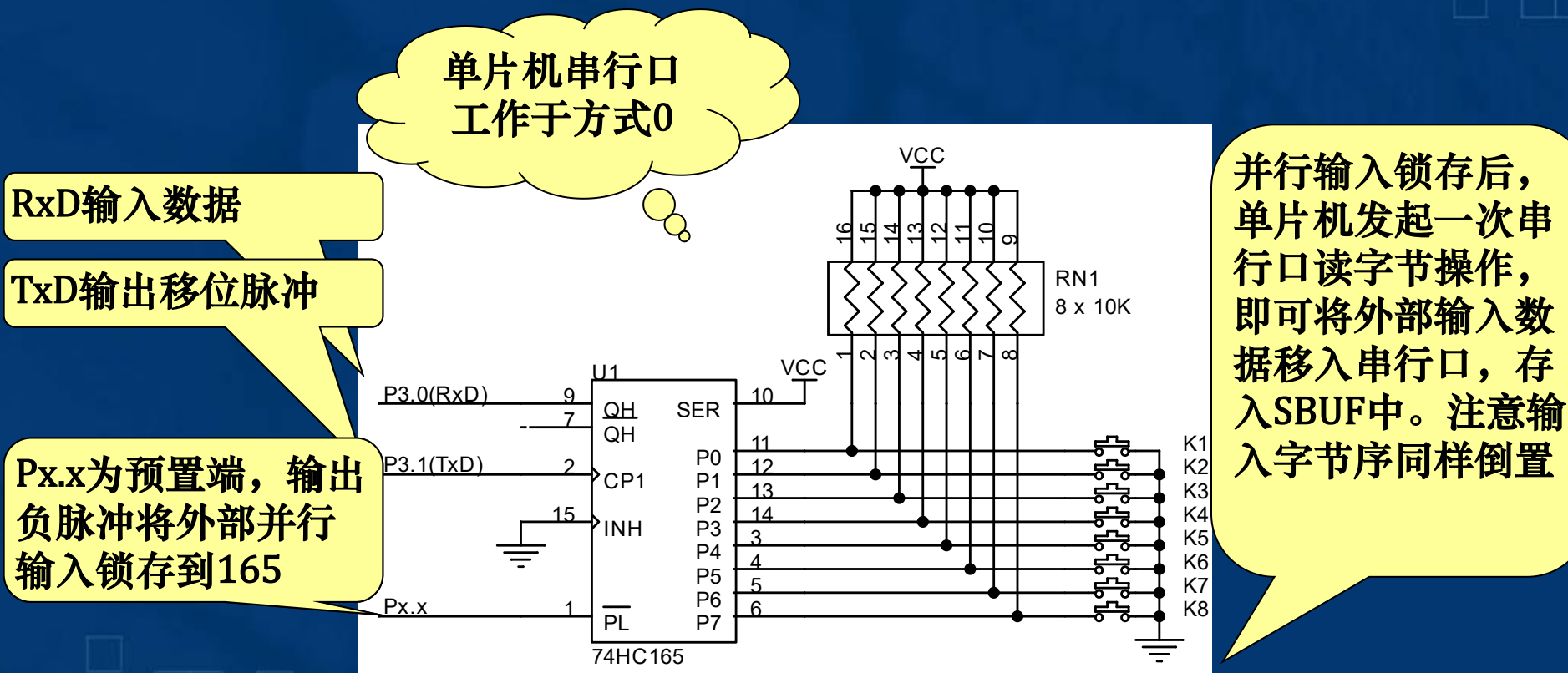
- 设置串行口工作在方式0（同步移位寄存器），此时串行口的TxD输出移位时钟，RxD输出数据；
- 外接一个并行-串行移位寄存器74LS165实现并-串变换，可扩展出一个8位的并行输入端口；
- 74LS165可级联扩展更大规模的并行输入接口。

单片机原理及系统设计

6.5.1 通过串行口扩展并行输入输出接口

2、通过串行口扩展并行输入接口

● 单片机串行口扩展并行输入接口电路图



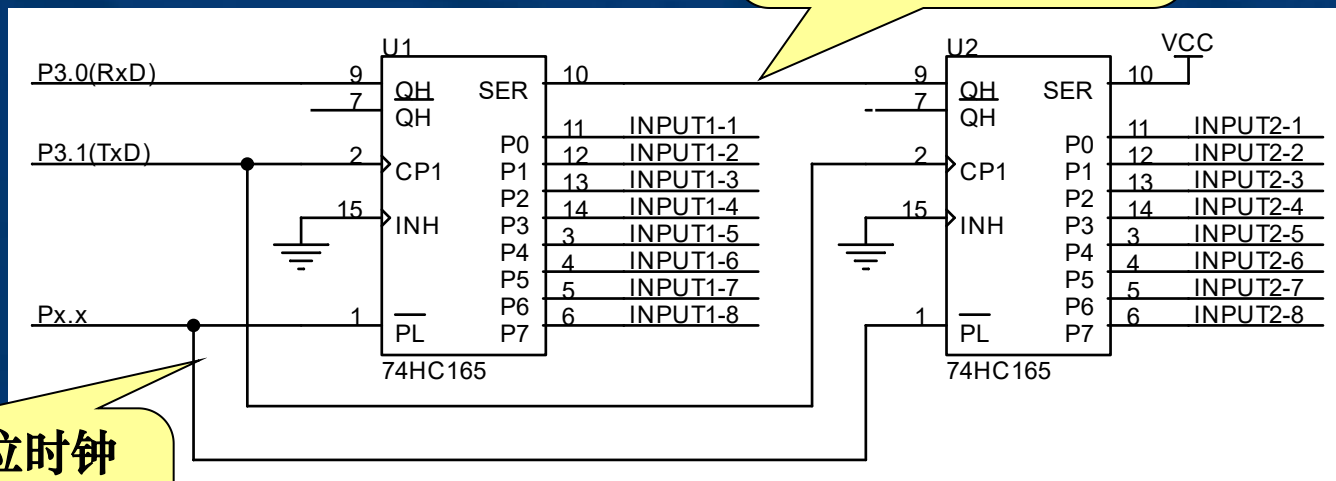
单片机原理及系统设计

6.5.1 通过串行口扩展并行输入输出接口

2、通过串行口扩展并行输入接口

● 单片机串行口级联扩展并行输入接口电路图

远端165的串行数据输出接到近端165的串行数据输入端



预置端及移位时钟端分别复接，即实现输入级联

6.5.2 通过系统总线扩展并行输入输出接口

- 设计原理

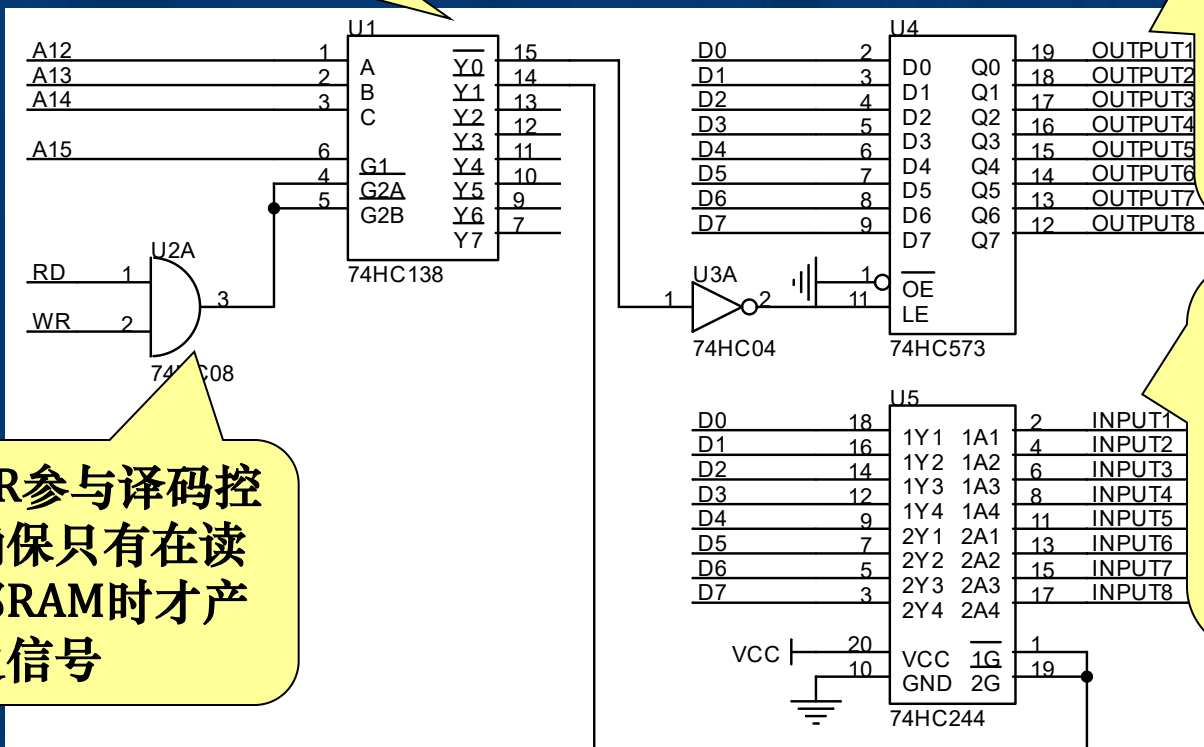
- 将锁存器/缓冲器连接在单片机数据总线上;
- 使用译码电路, 保证当单片机使用MOVX指令读写外部存储器时产生片选信号, 将此片选信号连接到锁存控制端及缓冲打开端;
- 此时单片机将挂接在扩展总线上的锁存器及缓冲器映像为外部RAM, 在执行读写外部RAM的MOVX指令时控制锁存及打开三态缓冲器, 实现CPU和扩展端口之间的数据交换。

单片机原理及系统设计

6.5.2 通过系统总线扩展并行输入输出接口

● 单片机系统总线扩展并行输入输出接口电路图

138实现地址译码



RD/WR参与译码控制，确保只有在读写外部RAM时才产生选通信号

输出端口，地址为0x8000~0x8FFF，
MOVX @DPTR,A指令执行后A中的数据输出到Q0~Q7

输入端口，地址为0x9000~0x9FFF，
MOVX A,@DPTR指令执行后
INPUT1~8的内容经数据总线读入A



第六章 单片机输入输出 接口及系统扩展设计

6.6 单片机I/O端口模拟时序 操作扩展设备

概述

- 单片机和外扩器件之间的接口通常分为并行接口和串行接口两种；
- 单片机不可能包括所有种类的接口控制逻辑，对于不能直接支持的器件接口，通常使用I/O端口，通过程序控制引脚模拟出符合接口操作要求的时序来完成功能；
- 串行接口占用I/O资源少、扩展方便、灵活，有利于减小系统体积，所以很多芯片使用串行总线和CPU连接；
- 除了常见的异步串行通信接口之外，串行总线还包括I²C总线、1-Wire总线、SPI总线及串行移位寄存器等等；
- 本节介绍单片机模拟I²C串行总线接口时序的设计原理。



6.6.1 I²C串行总线的基本特点

- I²C (Inter-Integrated Circuit) 是Philips公司推出的串行总线技术，它是在器件之间实现同步串行数据传输的技术，是一种采用两线制（数据线和时钟线）通信的标准总线。



6.6.1 I²C串行总线的基本特点

- I²C总线主要有以下几个特征：
 - 数据传输只需要两条通信线：串行数据线（SDA）和串行时钟线（SCL）；
 - 总线模式包括主发送模式、主接收模式、从发送模式、从接收模式；
 - 每个连接到I²C总线上的器件都必须有唯一的器件地址，通过这个地址，主器件可以对从器件进行寻址；
 - 存在冲突检测和仲裁机制以保证数据传输的完整性和稳定性。

6.6.1 I²C串行总线的基本特点

- I²C总线主要有以下几个特征：
 - 传输速率高，标准模式下数据传输速率可达100Kbps,快速模式下可达400Kbps，高速模式下可达3.4Mbps；
 - 由于总线接口引脚内部采用了漏极开路工艺，所以总线上要接上拉电阻。连接到相同总线上的IC数量只受到总线最大电容负载400pF的限制。
- I²C总线可极大简化子系统之间的连接，总线接口驱动集成在器件中，可最大限度地简化结构；
- 电路的简化可减少电路板面积，提高可靠性，降低成本；
- 使用I²C总线有利于促进系统的模块化、标准化设计。



6.6.2 I²C串行总线通信协议

1、I²C总线传输接口的特性

- 总线由两根双向的数据线和时钟线构成，使用时必须分别通过上拉电阻接VCC；
- 接入总线的所有器件的输出都必须是漏极开路方式；
- I²C总线是一个半双工、多主器件总线，即总线上可有多多个主控制器；当器件发送数据时即为主器件，接收数据时则为从器件；
- I²C总线进行数据传送时，每一位数据都与时钟脉冲相对应：时钟线为高时，数据线上必须保持稳定的逻辑电平；只有在时钟线为低时，才允许数据线的电平变化。

6.6.2 I²C串行总线通信协议

2、I²C总线的时序

- 一次完整的I²C总线时序过程由起始信号、从器件地址信号、应答信号ACK、字节数据信号和停止信号等组成;

(1) 起始和停止信号

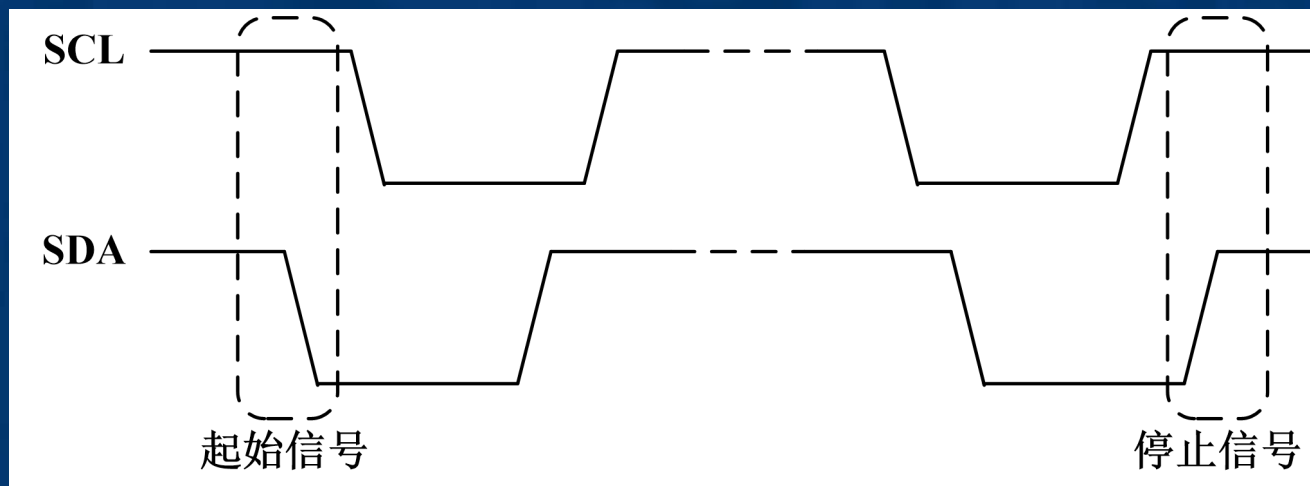
- 在I²C总线协议中, 起始信号 (S) 和停止信号 (P) 都是由主器件产生的。起始信号表明一次I²C总线传送的开始, 停止信号则表明I²C总线通信结束;
- 当SCL (时钟线) 为高电平时, SDA (数据线) 由高电平到低电平的负跳变被定义为起始信号; SDA (数据线) 由低电平到高电平的正跳变为停止信号。

6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(1) 起始和停止信号

- I²C总线起始信号和停止信号的时序



6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(2) 器件地址

- 接入I²C总线的每个器件都要有唯一的地址。每次发送器发出起始信号后，必须接着发出一个字节的地址信息，以选取连接在总线上的某一从器件；
- 地址字节用“从器件地址+R/ \bar{W} ”表示。该字节的高7位（D7~D1）为器件地址；D0为读写控制位，为1表示主器件后续将读从器件；为0则表示主器件将写从器件；
- 7位的从器件地址进一步分为固定部分和可编程部分。固定部分为器件标识，在出厂时设置。可编程部分为器件的物理地址，以区分连接在同一I²C总线上的同类器件。



6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(3) 应答信号ACK

- I²C总线上的发送器发送完地址字节和每一个字节数据后，被寻址的接收器都必须产生一个应答信号；产生该信号的接收器在第9个时钟周期时将SDA线拉低，表示已收到一个8位数据；
- 与应答信号相对应的第9个时钟由发送器产生。发送器必须在输出该时钟时释放数据线SDA，使其处于高阻状态，以便接收接收器件送出的应答信号；
- 若此后接收器在SDA线上输出了低电平应答信号（ACK），表示接收正常，后续将继续接收数据；如果接收器输出高电平的非应答信号（NO ACK），则表示结束本次通信。

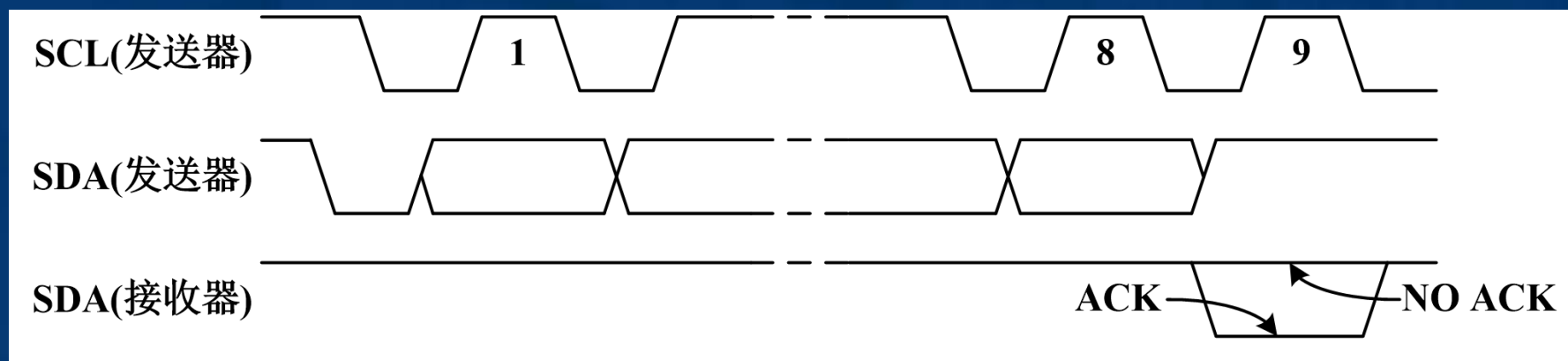
单片机原理及系统设计

6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(3) 应答信号ACK

- I²C总线应答时序如下图所示：



6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(4) 数据字节信号

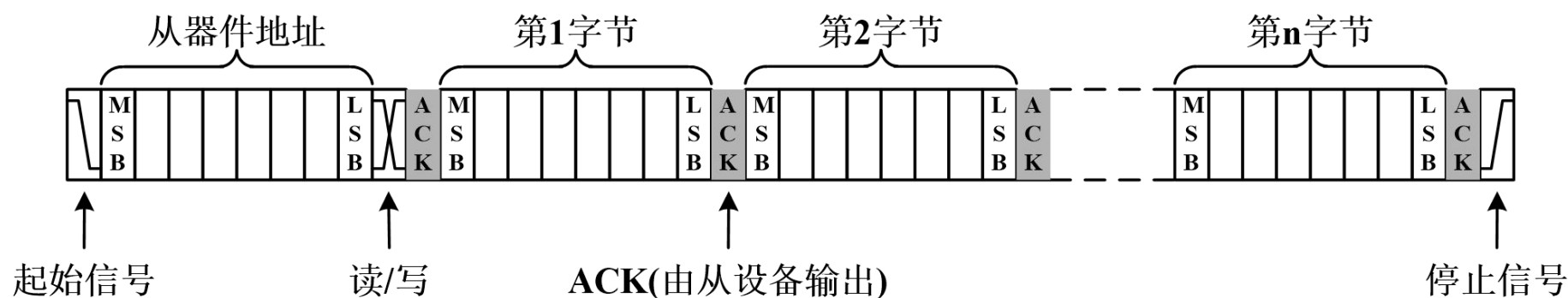
- I²C总线在器件之间传送数据的字节数是没有限制的；
- 传送的每字节必须是8位，且首先发送数据的最高位；
- 每个字节数据发完后发送器都必须发送一位应答脉冲，并在该脉冲有效（高）时接收由接收器发回的应答信号；
- 如果接收器回送ACK，则发送器继续发送后续字节；如果接收器回送NACK，则发送器发出停止信号P后结束数据传送；
- 数据传送过程中，如果接收器将SCL线拉成低电平，将迫使发送器处于等待状态，接收器释放SCL线后再继续发送过程。

6.6.2 I²C串行总线通信协议

2、I²C总线的时序

(4) 数据字节信号

- I²C总线连续发送多个数据字节的时序如下图所示：



单片机原理及系统设计

☺ The End ☺