嵌入式系统原理及实验

顾　震

信息科学与工程学院自动化系

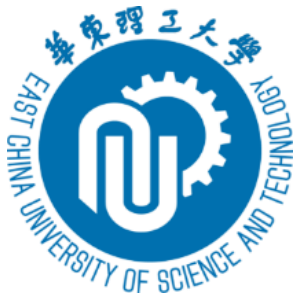
华东理工大学

Email: guzhen@ecust.edu.cn

课程大纲

1. 嵌入式系统导论
2. Cortex-M3微处理器
3. STM32最小系统及开发环境
4. 嵌入式C语言
5. 通用输入输出GPIO模块
6. 中断
7. 定时器原理与应用
8. USART通信原理及实现
9. DMA控制器
10. SPI与I2C通信原理及实现
11. 模数转换原理及实现
12. 人工智能辅助的嵌入式项目开发
13. 嵌入式应用前沿

	UART	SPI
同步通信	否	是
通信方向	全双工	全双工
串行通信	是	是
引脚数量	2	4（多从机增加）
速度	低速	高速
抗干扰	弱	高
拓扑结构	点对点	一主多从
传输距离	短	极短



10. SPI与I2C通信原理及实现

本章知识与能力要求

- 理解和掌握I2C通信协议基本原理；
- 理解通信协议的时序分析方法；
- 熟悉使用HAL库中GPIO软件控制实现I2C通信方法。

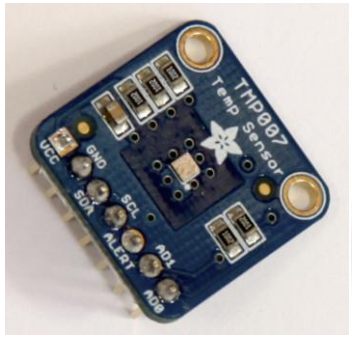
10. I2C通信

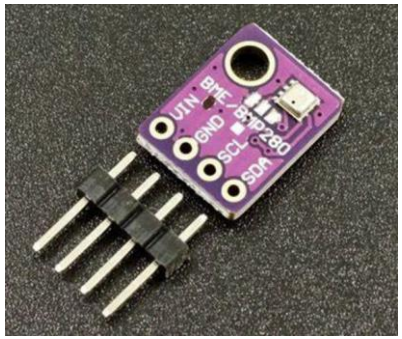
10.4 I2C总线基本概念

- I2C总线是Philips公司在八十年代初推出的一种**串行、半双工的总线**，主要用于**近距离、低速**芯片之间的通信；
- I2C总线有两根双向的信号线：

  SDA：用于收发数据

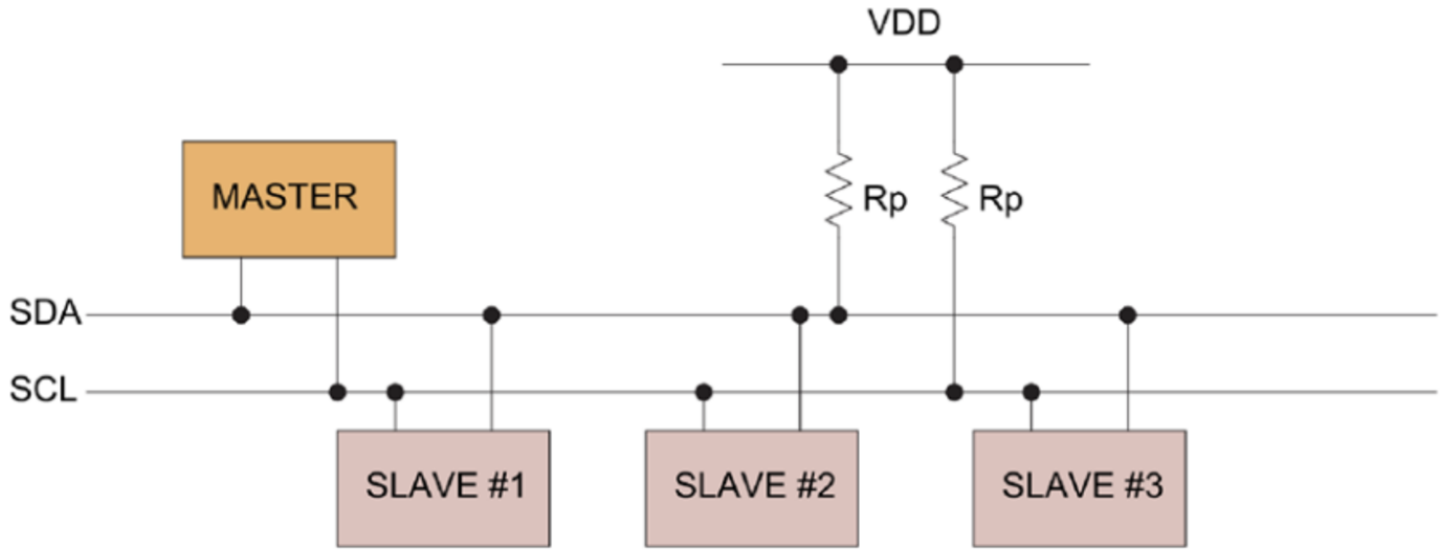
  SCL：用于通信双方时钟的同步
- I2C总线硬件结构简单，简化了PCB布线，降低了系统成本，提高了系统可靠性，因此在各个领域得到了广泛应用。





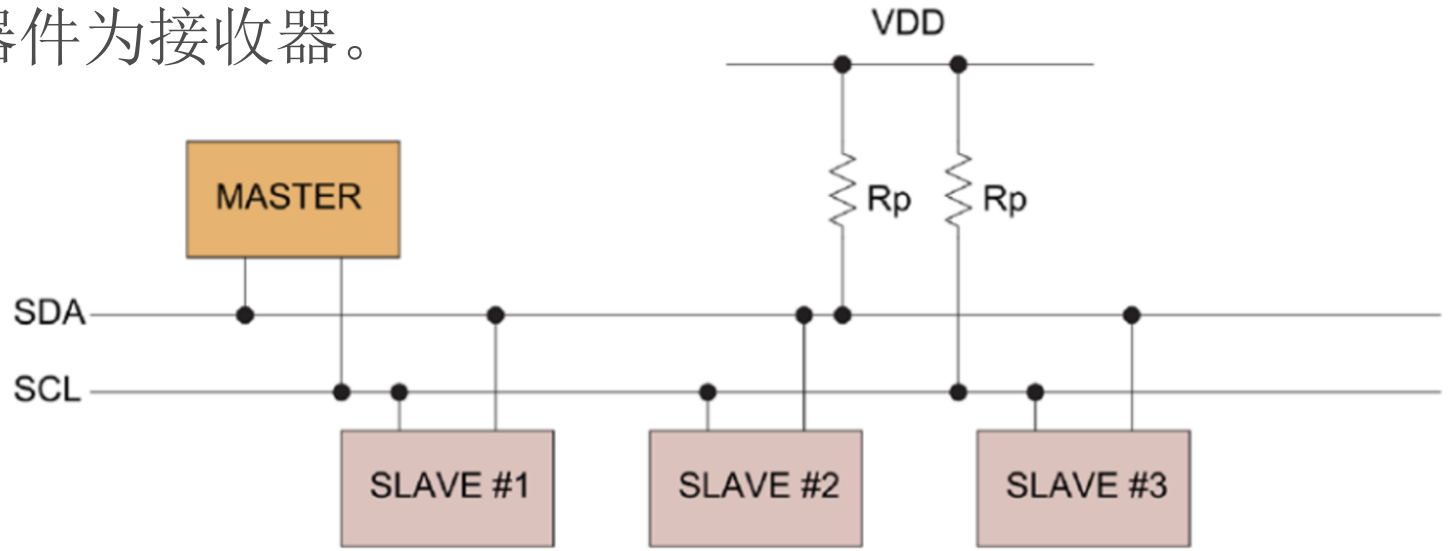
10.4 I2C总线基本概念

- I2C总线是一种**多主机总线**，连接在 I2C总线上的器件分为主机和从机。
- 主机有权发起和结束一次通信，从机只能被动呼叫；
- 当总线上有多个主机同时启用总线时，I2C也具备冲突检测和仲裁的功能来防止错误产生；



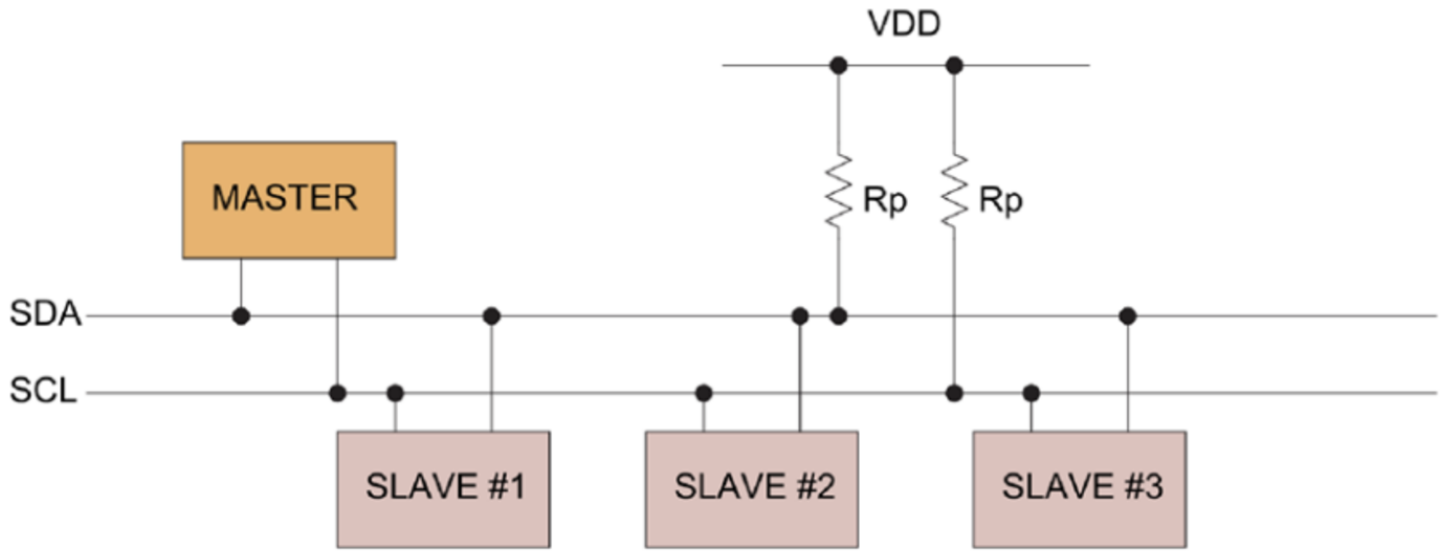
10.4 I2C总线基本概念

- 每个连接到I2C总线上的器件都有一个**唯一的地址（7bit）**，且每个器件都可以作为主机也可以作为从机（但同一时刻只能有一个主机）；
- 总线上的器件增加和删除不影响其他器件正常工作；
- I2C总线在通信时总线上发送数据的器件为发送器，接收数据的器件为接收器。

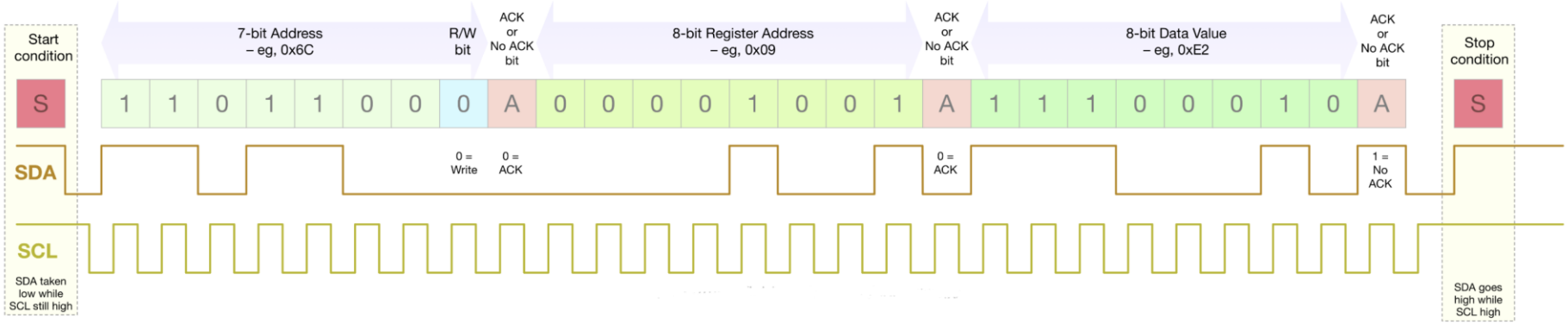


10.4 I2C总线基本概念

- 串行的8位双向数据传输速率在标准模式下可达100Kbit/s，快速模式下可达400Kbit/s，高速模式下可达3.4Mbit/s。
- 总线具有极低的电流消耗，抗噪声干扰能力强，增加总线驱动器可以使总线电容扩大10倍，传输距离达到15m；兼容不同电压等级的器件，工作温度范围宽。

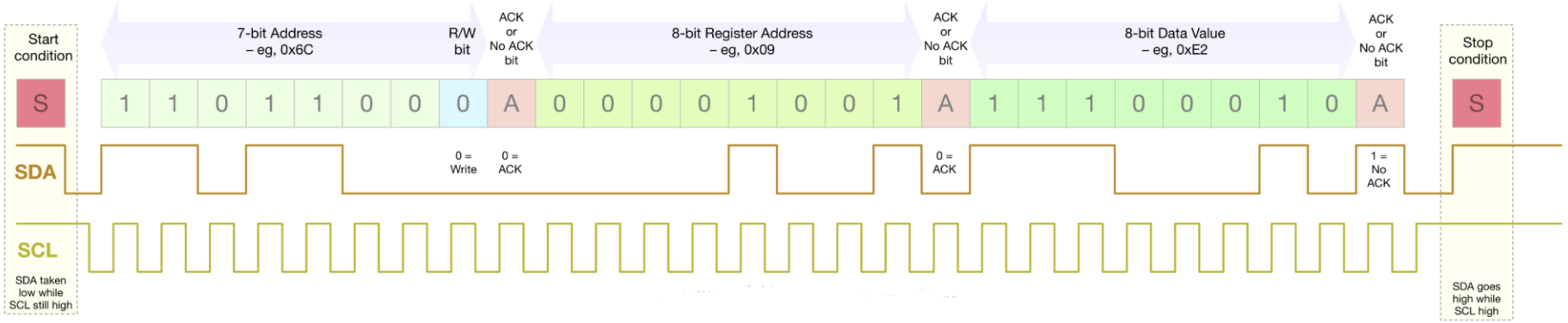


10.4.1 I2C通信过程



1. 主机发送起始信号启用总线
2. 主机发送一个字节数据指明从机地址和后续字节的传送方向
3. 被寻址的从机发送应答信号回应主机
4. 发送器发送一个字节数据
5. 接收器发送应答信号回应发送器
6. ..... （循环步骤4、5）
7. 通信完成后主机发送停止信号释放总线

10.4.3 I2C起始信号与停止信号



- **起始信号**：SCL为高电平时，SDA由高变低；
- **停止信号**：SCL为高电平时，SDA由低变高；
- 起始信号和停止信号都是由主机发出，起始信号产生后总线处于占用状态，停止信号产生后总线被释放，处于空闲状态。
- 空闲时，SCL与SDA都是高电平。

两种停止情况：

1. 主机不想发了，就发送停止信号；
2. 从机不想接了，不应答，主机就发送停止信号结束此次通信。

10.4.2 I2C寻址方式

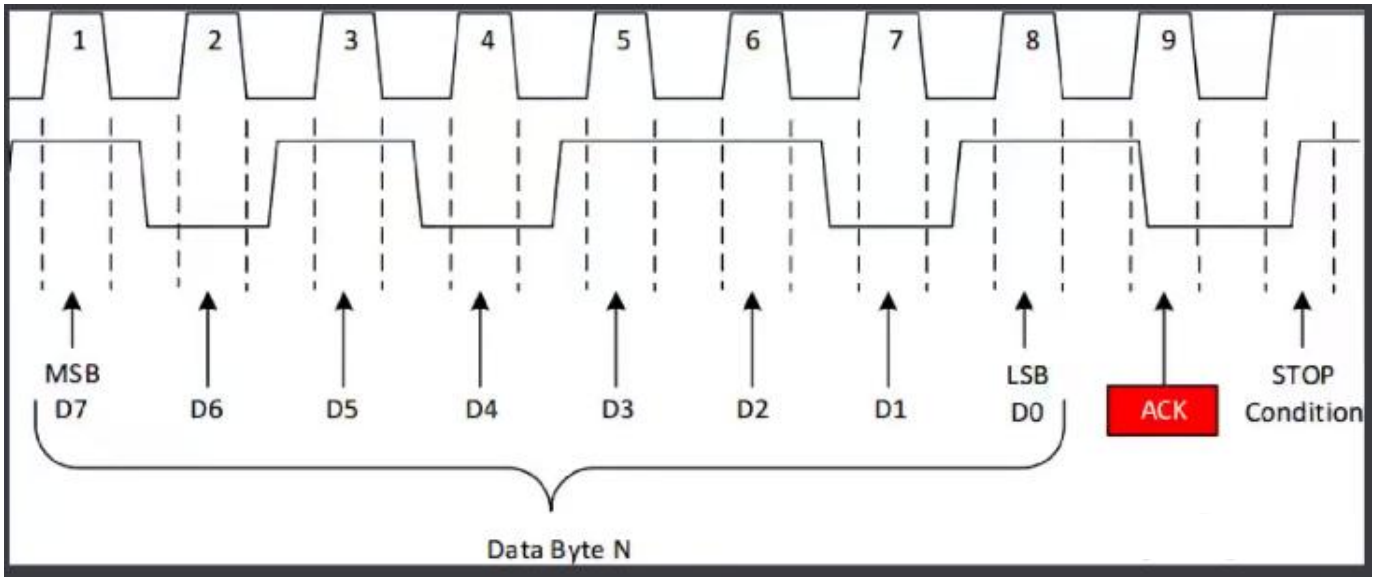
- I2C总线上传送的数据是广义的，既包括地址，又包括真正的数据。
- 主机在发送起始信号后必须先发送一个字节的数据，该数据的**高7位为从机地址**，最低位表示后续字节的传送方向:

  ‘0’表示主机发送数据给从机

  ‘1’表示从机发送数据给主机
- 总线上所有的从机接收到该字节数据后都将这7位地址与自己的地址进行比较，如果相同，则认为自己被主机寻址，然后再根据第8位将自己定为发送器或接收器。

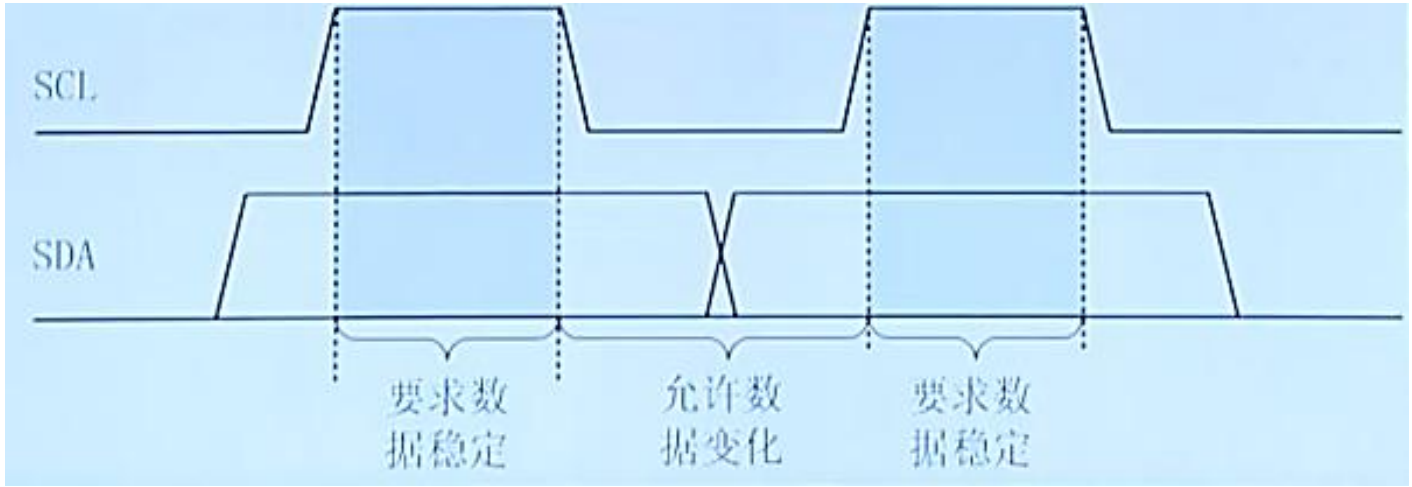
10.4.4 字节传送与应答

- I2C总线通信时每个字节为8位长度，数据传送时，先传送最高位，后传送低位，发送器发送完一个字节数据后接收器必须发送1位应答位来回应发送器，即一帧共有9位。
- I2C每次发送数据必须是8位。
- 最高有效位MSB固定，先发高位，再发低位。



10.4.5 同步数据信号

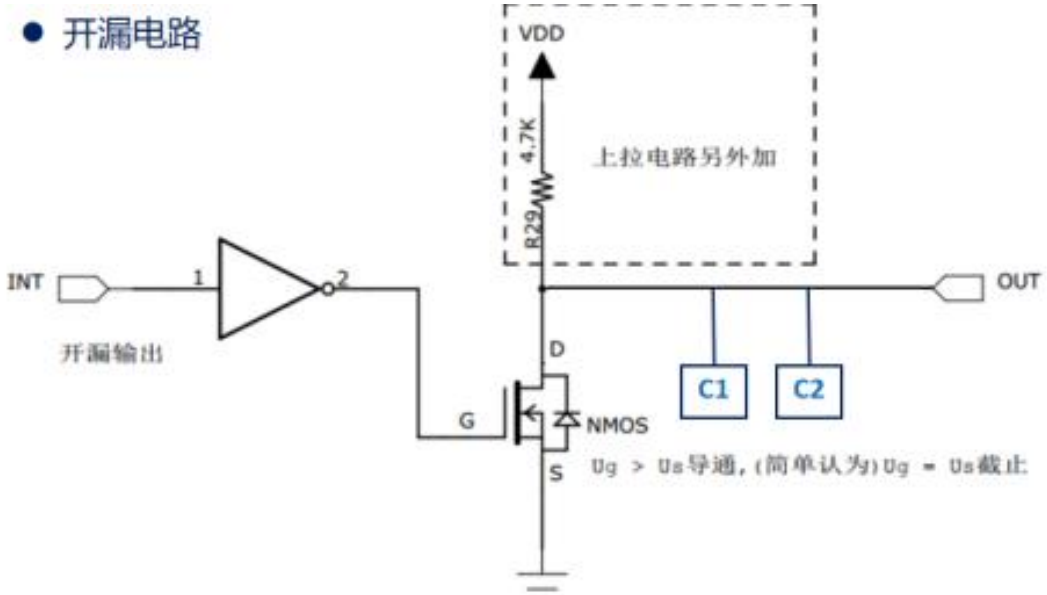
- I2C总线在进行数据传送时，时钟线SCL为低电平期间发送器向数据线上发送一位数据，在此期间数据线上的信号允许发生变化
- 时钟线SCL为高电平期间**接收器从数据线上读取一位数据**，在此期间**数据线上的信号不允许发生变化**，必须保持稳定。



10.4.6 时钟同步与仲裁

（1）时钟同步

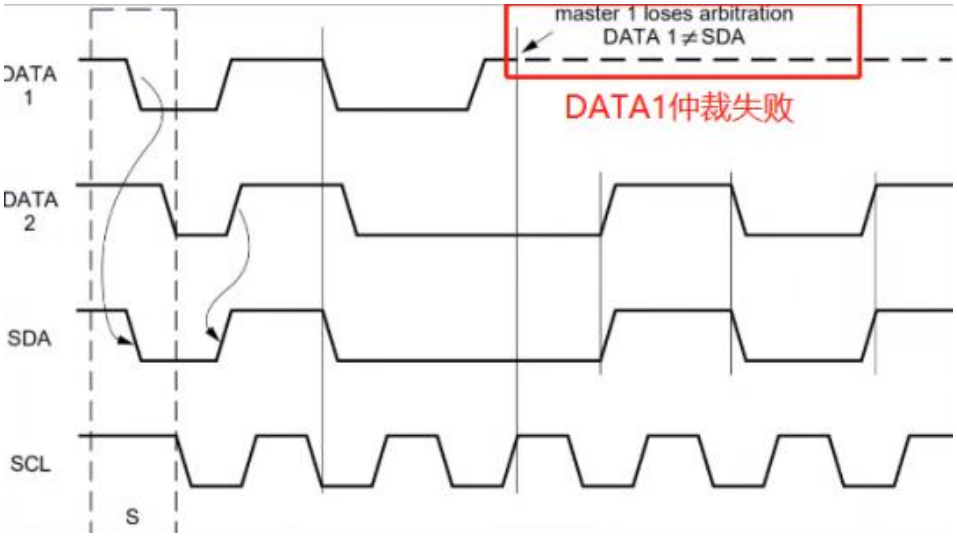
- 时钟同步是通过I2C总线上的SCL之间的线“与”（wire-AND）来完成的，即如果有多个主机同时产生时钟，那么只有所有master都发送高电平时，SCL上才表现为高电平，否则SCL都表现为低电平。



10.4.6 时钟同步与仲裁

（2）仲裁

- 总线仲裁与时钟同步类似，当所有主机在SDA上都写1时，SDA的数据才是1，只要有一个主机写0，那此时SDA上的数据就是0。
- 一个主机每发送一个bit数据，在SCL为高电平时，就检查SDA的电平是否和发送的数据一致，如果不一致，这个主机便知道自己输掉了仲裁，然后停止向SDA写数据。



10.5 典型I2C时序

（1）主机向从机发送数据

阴影：由主机向从机传送数据
无阴影：从机向主机发送数据。

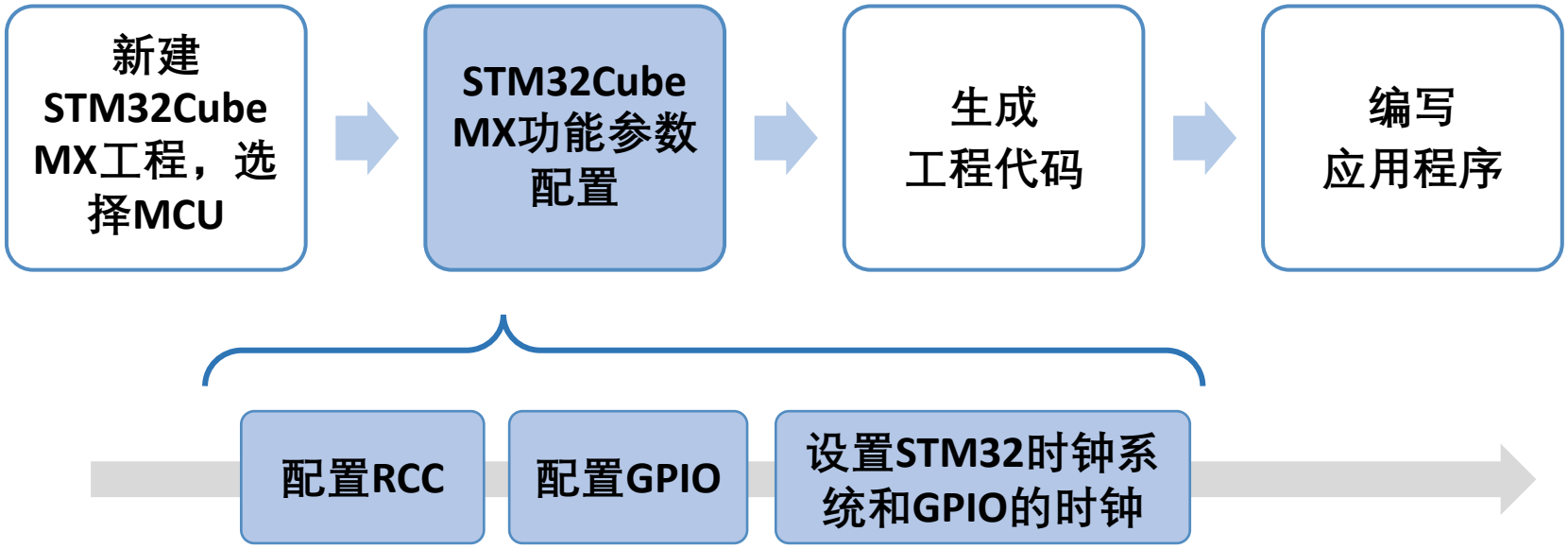
（2）从机向主机发送数据

（3）主机先向从机发送数据，从机再向主机发送数据

S：起始信号
P：终止信号

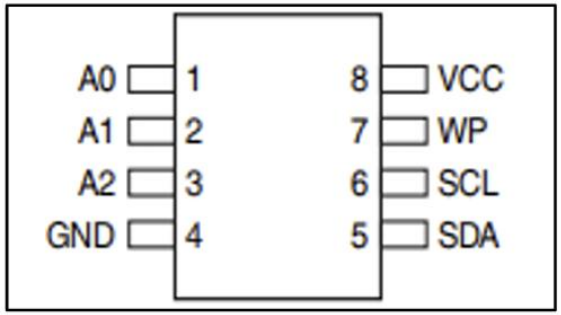
A：应答信号，表示成功接收数据
$\bar{A}$：非应答信号，表示接收失败或不再接收数据

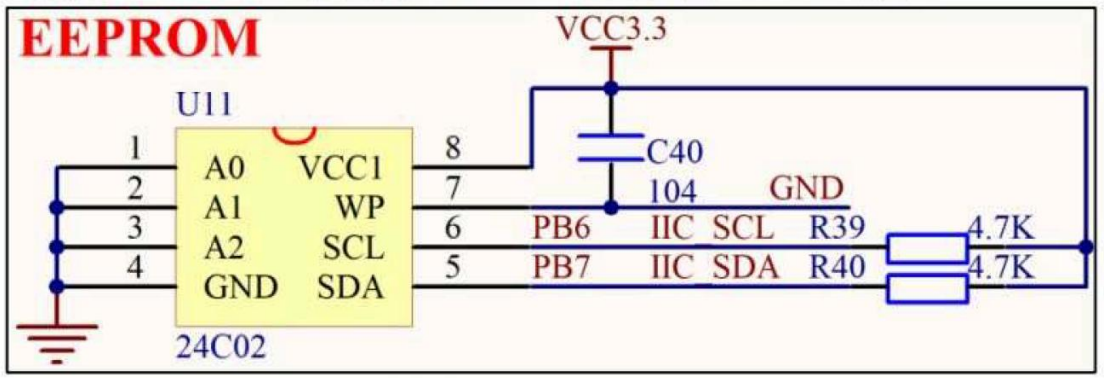
10.6 基于HAL的I2C开发



10.6 基于HAL的I2C开发

24C02 是一个2K bit 的串行EEPROM存储器，内部含有256个字节。在24C02里面还有一个8字节的页写缓冲器。该设备的通信方式I2C，通过其SCL和SDA与其他设备通信，A0-A2为地址选择引脚，用于多颗芯片同时使用时的片选，WP的写保护引脚。





10.6 基于HAL的I2C开发

软件模拟I2C通信

GPIO引脚控制宏定义

```
#define SDA_ON HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET)
#define SDA_OFF  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET)
#define SDA_Read  HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_7)
#define SCK_ON HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET)
#define SCK_OFF  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET)
```

I2C函数定义

```
void i2c_start(void);
void i2c_stop(void);
void i2c_ack(void);
void i2c_nack(void);
void i2c_send_byte(uint8_t data);
uint8_t iic_read_byte(uint8_t ack);
```

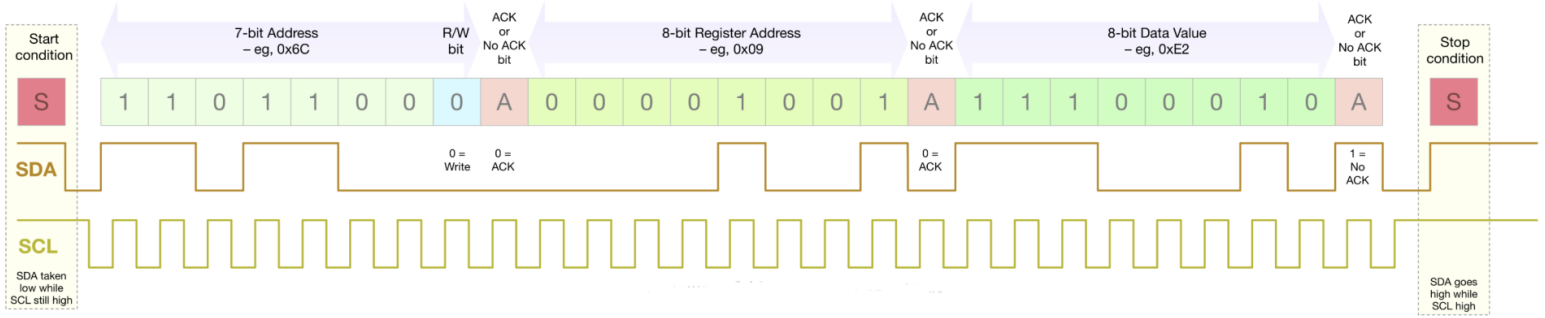
10.6 基于HAL的I2C开发

软件模拟I2C通信 "i2c.c"

```
void i2c_start(void)
{
        SDA_ON;
        SCK_ON;
        HAL_Delay(1);
        SDA_OFF;
        HAL_Delay(1);
        SCK_OFF;
        HAL_Delay(1);
}
```

```
void i2c_stop(void)
{
        SDA_OFF;
        HAL_Delay(1);
        SCK_ON;
        HAL_Delay(1);
        SDA_ON;
        HAL_Delay(1);
}
```

```
void i2c_ack(void)
{
        SDA_OFF;
        HAL_Delay(1);
        SCK_ON;
        HAL_Delay(1);
        SCK_OFF;
        HAL_Delay(1);
        SCK_ON;
        HAL_Delay(1);
}
```



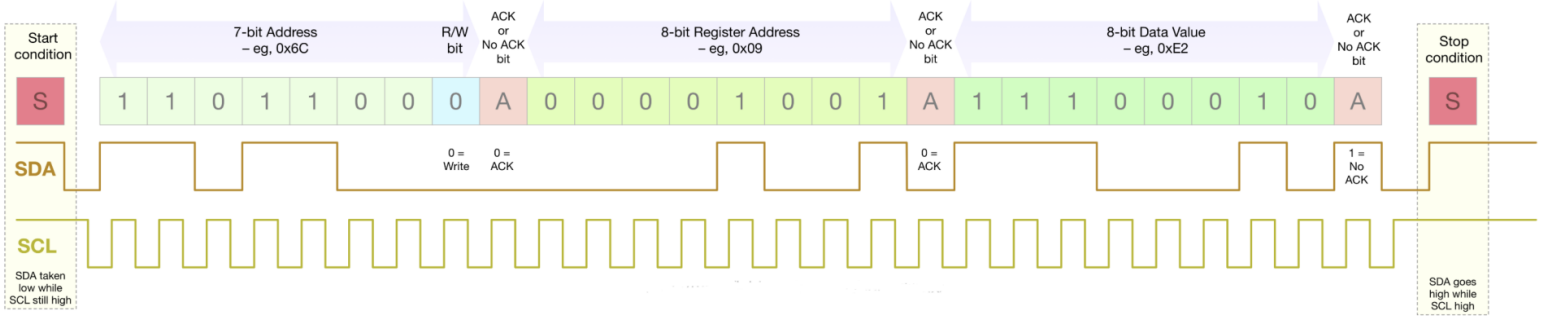
10.6 基于HAL的I2C开发

软件模拟I2C通信

"i2c.c"

```
void i2c_nack(void)
{
        SCK_ON;
        HAL_Delay(1);
        SCK_ON;
        HAL_Delay(1);
        SCK_OFF;
        HAL_Delay(1);
}
```

```
void i2c_send_byte(uint8_t data)
{
        uint8_t t;
        for (t = 0; t < 8; t++)
        {
            if((data & 0x80) >> 7)
            {
                SDA_ON;
            }
            else {
                SDA_OFF;
            }
                                                    HAL_Delay(1);
                                                    SCK_ON;
                                                    HAL_Delay(1);
                                                    SCK_OFF;
                                                    data <<= 1;
                                                }
                                                SDA_ON;
```



10.6 基于HAL的I2C开发

软件模拟I2C通信　"i2c.c"

```
uint8_t i2c_read_byte(uint8_t ack)
{
    uint8_t i, receive = 0;
    for (i = 0; i < 8; i++ )
    {
        receive <<= 1;
        SCK_ON;
        HAL_Delay(1);
        if (SDA_Read)
        {
            receive++;
        }
        SCK_OFF;
        HAL_Delay(1);
    }
    if (!ack)
    {
    i2c_nack();
    }
    else
    {
        i2c_ack();
    }
    return receive;
}
```

10.6 基于HAL的I2C开发

软件模拟I2C通信 "24c02.c"

```
uint8_t AT24c02_check(void)
{
        uint8_t temp;
        temp = AT24c02_read_one_byte(255);
        if (temp == 0X55)
        {
        return 0;
        }
        else
        {
        AT24c02_write_one_byte(255, 0X55);
        temp = AT24c02_read_one_byte(255);
        if (temp == 0X55)return 0;
        }
        return 1;
}
```

```
void AT24c02_read(uint16_t addr, uint8_t *pbuf, uint16_t datalen)
{
        while (datalen--)
        {
        *pbuf++ = AT24c02_read_one_byte(addr++);
        }
}
void AT24c02_write(uint16_t addr, uint8_t *pbuf, uint16_t datalen)
{
        while (datalen--)
        {
        AT24cxx_write_one_byte(addr, *pbuf);
        addr++;
        pbuf++;
        }
}
```

10.6 基于HAL的I2C开发

软件模拟I2C通信 "24c02.c"

```
void AT24c02_write_one_byte(uint16_t addr, uint8_t data)
{
        i2c_start();
        i2c_send_byte(0XA0 + ((addr >> 8) << 1));
        i2c_wait_ack();
        i2c_send_byte(addr % 256);
        i2c_wait_ack();
        i2c_send_byte(data);
        i2c_wait_ack();
        i2c_stop();
        HAL_Delay(10);
}
```

```
uint8_t AT24c02_read_one_byte(uint16_t addr)
{
        uint8_t temp = 0;
        i2c_start();
        i2c_send_byte(0XA0 + ((addr >> 8) << 1));
        i2c_wait_ack();
        i2c_send_byte(addr % 256);
        i2c_wait_ack();
        i2c_start();
        i2c_send_byte(0XA1);
        i2c_wait_ack();
        temp = i2c_read_byte(0);
        i2c_stop();
        return temp;
}
```

10.6 基于HAL的I2C开发

- 软件模拟I2C通信　"main.c"

```
const uint8_t g_text_buf[] = {"ECUST I2C TEST"};
#define TEXT_SIZE sizeof(g_text_buf)
uint8_t g_read_buf[15];
```

Main函数中初始化检测设备

```
while (AT24c02_check()) /* 检测不到24c02 */
{
  HAL_Delay(500);
  HAL_GPIO_TogglePin(); /* 红灯闪烁 */
}
```

写入及读出数据

```
AT24c02_write(0, (uint8_t *)g_text_buf, TEXT_SIZE);
AT24c02_read(0, g_read_buf, TEXT_SIZE);
```

本章小结

10.4 I2C总线基本概念

10.4.1 I2C通信过程

10.4.2 I2C寻址方式

10.4.3 I2C起始信号与停止信号

10.4.4 字节传送与应答

10.4.5 同步数据信号

10.4.6 时钟同步与仲裁

10.5 典型I2C时序

10.6 基于HAL的I2C开发