

R X 3 2 x 6 x x 应 用 笔 记

低优先级 I2C 通信应用指导

文档编号：AN00019

版本：V1.0

目录

1 简介	5
2 I2C 通信基础概念	6
3 I2C 通信基础流程	7
3.1 向从机写入	7
3.2 向从机读取	7
4 I2C 通信主机处理接收的字节	8
4.1 当接收的传输字节数量 $N > 2$	8
4.2 当接收的传输字节数量 $N = 2$	10
4.3 当接收的传输字节数量 $N = 1$	12
5 版本历史	14

表目录

表 2.1 部分 I2C 状态位	6
表 2.2 部分 I2C 操作.....	6
表 5.1 版本历史.....	14

图目录

图 4.1 当接收字节数量 $N > 2$ 时主接收器传输时序图	8
图 4.2 当接收字节数量 $N = 2$ 时主接收器传输时序图	10
图 4.3 当接收字节数量 $N = 1$ 时主接收器传输时序图	12

1 简介

I2C (Inter-Integrated Circuit) 是一种两线制串行通信协议，包含 SCL (时钟线) 和 SDA (数据线)，支持主从式多设备通信。

I2C 广泛用于微控制器和外围设备之间的通信。在通信数据需要处理时，通常涉及到时序和状态位的判断，逻辑相对繁琐。而当 I2C 处于低优先级任务进行通信处理时，对软件控制时序有更加严格的需求。本文以 RX32x6xx (64KB Flash / 8KB SRAM) 系列芯片为例，试编写常见的 I2C 通信逻辑和数据传输接收的处理，以便于用户参考。

2 I2C 通信基础概念

I2C 广泛用于微控制器和外围设备之间的通信。例如当 RX32x6xx 作为主机时，主机负责控制整个通信过程，产生时钟信号，包括设定相应的时钟频率信号，从而控制数据传输速率和保证数据的准确性。

RX32x6xx 系列的 I2C 模块典型速率包括：

- 标准模式：100 kbps
- 快速模式：400 kbps

从机会根据主机发出的时钟信号来同步操作，负责响应主机发送的指令（ACK），并在主机的时钟信号控制下返回相应的数据。

主机在处理通信任务时，通常需要根据 I2C 的状态（I2C_SR1/SR2 寄存器）来处理通信逻辑。

在一个 I2C 基础的主从通信的流程中，常见的部分状态位判断如下：

表 2.1 部分 I2C 状态位

状态位	释义
BUSY	总线是否正在进行数据通信？
SB	起始条件是否已发送？
ADDR	从机地址是否已发送？
BTF	字节是否发送结束？
RxNE	数据寄存器是否为空？

当逻辑标志位置位时，则操控 I2C（I2C_CR1 寄存器）来完成数据的传输控制，开始与停止。

通过获取对应的状态标志位，按逻辑操作 I2C 进行数据传输控制：

表 2.2 部分 I2C 操作

操作	释义
START	生成或重复生成起始条件信号
STOP	生成停止条件
NACK	配置为非应答模式
ACK	配置为应答模式
POS	控制应答位置

3 I2C 通信基础流程

当 I2C 在应用中未使用最高优先级中断或 I2C 采用轮询方式使用的情况，应利用状态位操作 I2C。

举例说明：当 RX32x6xx 作为主机，eeprom 作为从机一个基础的读/写流程如下：

3.1 向从机写入

基本通信流程：
● 通信前确保应答使能，ACK=1 ● 确保总线正常，判断总线是否忙（BUSY）或存在其他总线错误（ERROR） ● 生成起始条件（START），等待起始条件生成完毕（SB=1） ● 发送从机地址和写操作 ● 等待地址发送完毕（ADDR=1），清除地址发送结束标志位（ADDR=0） ● 向从机发送（寄存器地址），等待字节发送结束（BTF=1）
● 向从机发送（通信数据），等待字节发送结束（BTF=1） ● 处理完所有数据，主机生成停止条件（STOP）结束通信

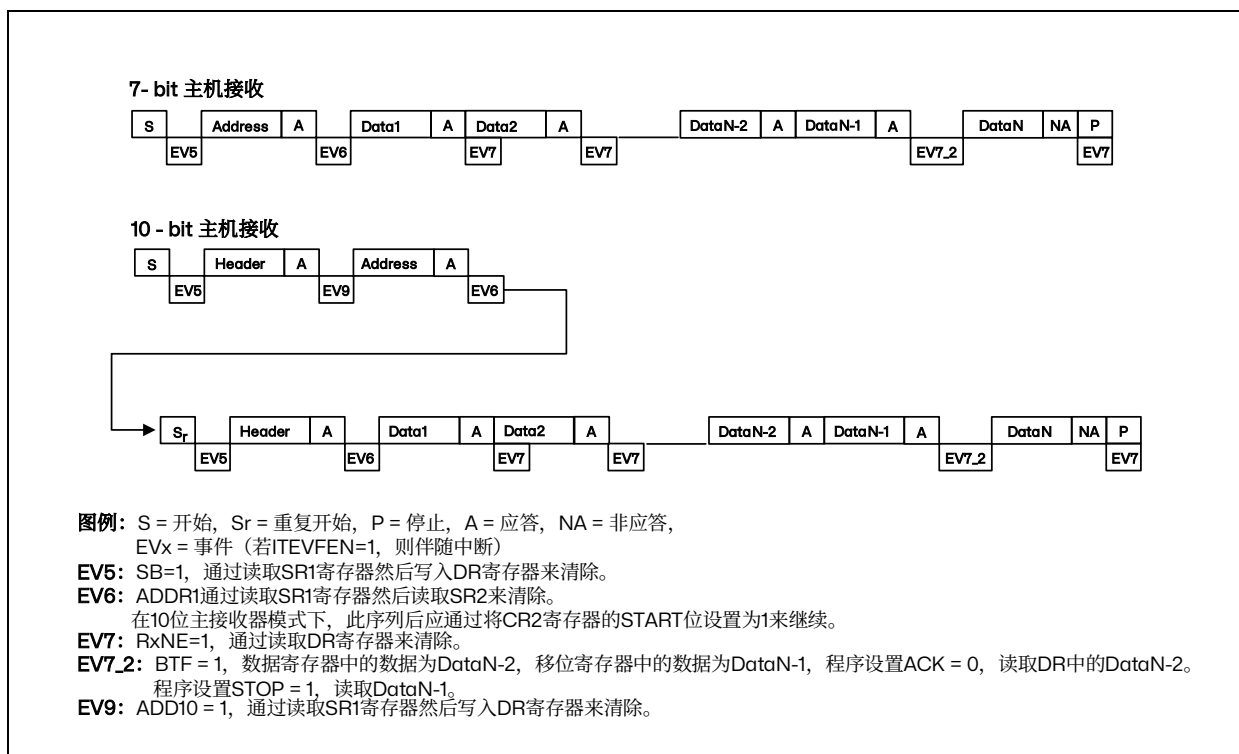
3.2 向从机读取

基本通信流程：
● 通信前确保应答使能，ACK=1 ● 确保总线正常，判断总线是否忙（BUSY）或存在其他总线错误（ERROR） ● 生成起始条件（START），等待起始条件生成完毕（SB=1） ● 发送从机地址和写操作 ● 等待地址发送完毕（ADDR=1），清除地址发送结束标志位（ADDR=0） ● 向从机发送（寄存器地址），等待字节发送结束（BTF=1）
● 重新生成起始条件（START），并重新等待起始条件生成完毕（SB=1） ● * 视传输字节数量而定接收流程

注：数据接收时根据字节数量不同需要采取不同的软件时序处理方式，详见下一章节：I2C 通信主机处理接收的字节。

4 I2C 通信主机处理接收的字节

4.1 当接收的传输字节数量 $N > 2$



1. EV5、EV6 和 EV9 事件会将 SCL 拉低, 直到相应的软件序列结束。
2. EV7 软件序列必须在当前字节传输结束之前完成。如果无法在当前字节传输结束之前管理 EV7 软件序列, 建议使用 BTF 而非 RXNE, 但缺点是会降低通信速度。

图 4.1 当接收字节数量 $N > 2$ 时主接收器传输时序图

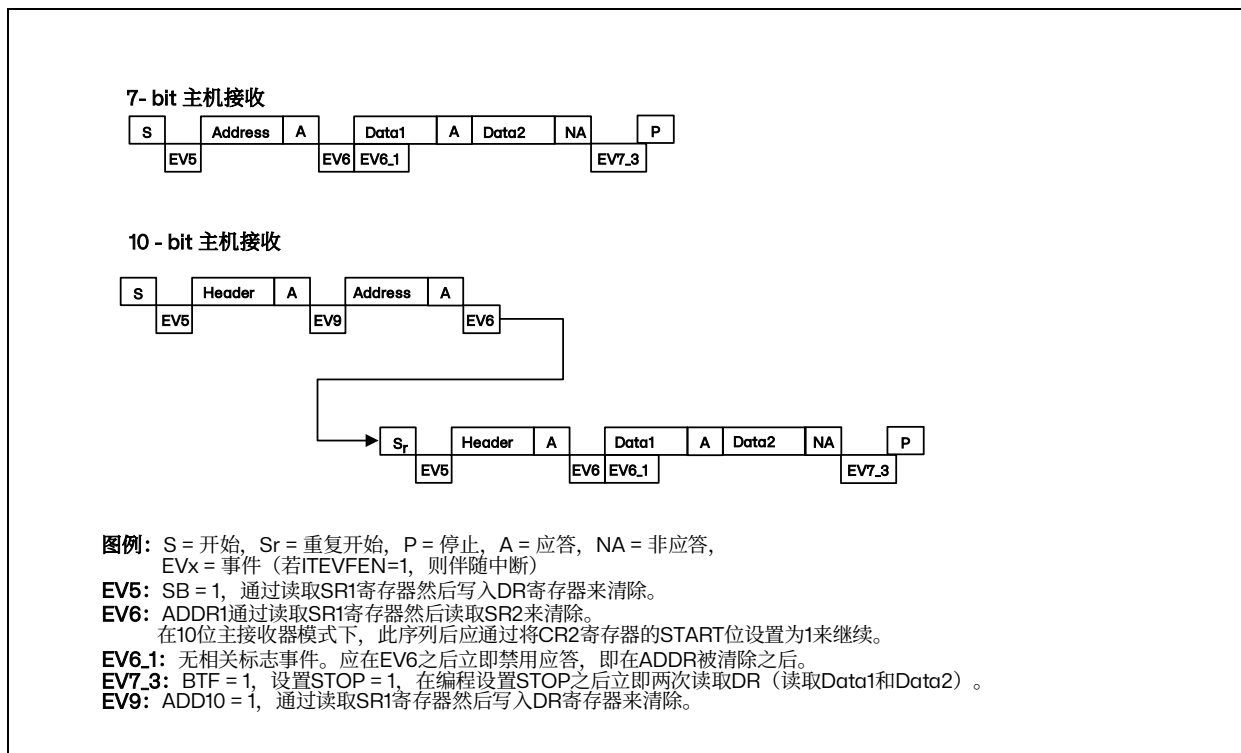
当接收大于 2 个字节数据, 传输时最后还剩 3 个数据时:

- RxNE = 1 => 无操作 (DataN-2 尚未读取)。
- 已接收 DataN-1。
- BTF = 1, 因为移位寄存器和数据寄存器均已满: DR (数据寄存器) 中有 DataN-2, 移位寄存器中有 DataN-1 => SCL 被拉低: 总线上不会接收其他数据。
- 清除 ACK 位。
- 从 DR 中读取 DataN-2 => 这将启动移位寄存器中的 DataN 接收。
- 已接收 DataN (伴随 NACK)。
- 生成 STOP 信号。
- 读取 DataN-1。
- RxNE = 1
- 读取 DataN。

对应的代码流程示例:

```
//产生起始条件
I2C_Generate_StartCondition(I2C1);
//等待起始条件生成完毕
while (!I2C_Get_Flag(I2C1, I2C_FLAG_SB));
//发送地址和读操作
I2C_Transmit_Data8(I2C1, slave_addr_8 | I2C_REQUEST_READ);
while (!I2C_Get_Flag(I2C1, I2C_FLAG_ADDR));
//清除地址标志位
I2C_Clear_Flag(I2C1, I2C_FLAG_ADDR);
//处理通信数据
for (uint16_t i=0; i<dataLen; i++)
{
    if (i < dataLen-3)
    {
        while (!I2C_Get_Flag(I2C1, I2C_FLAG_RXNE));
        data[i] = I2C_Receive_Data8(I2C1);
    }
    //读取 DataN-2
    if (i == dataLen-3)
    {
        while (!I2C_Get_Flag(I2C1, I2C_FLAG_BTF));
        I2C_Set_AcknowledgeNextData(I2C1, I2C_CR1_NACK);
        data[i] = I2C_Receive_Data8(I2C1);
    }
    //读取 DataN-1
    if (i == dataLen-2)
    {
        I2C_Generate_StopCondition(I2C1);
        data[i] = I2C_Receive_Data8(I2C1);
    }
    //读取 DataN
    if (i == dataLen-1)
    {
        while (!I2C_Get_Flag(I2C1, I2C_FLAG_RXNE));
        data[i] = I2C_Receive_Data8(I2C1);
    }
}
```

4.2 当接收的传输字节数量 N=2



1. EV5、EV6 和 EV9 事件会将 SCL 拉低, 直到相应的软件序列结束。
2. EV6.1 软件序列必须在当前字节传输的 ACK 脉冲之前完成。

图 4.2 当接收字节数量 N=2 时主接收器传输时序图

当接收 2 个字节数据时:

- 设置 POS 和 ACK 位。
- 等待 ADDR 标志置位。
- 清除 ADDR。
- 清除 ACK 位。
- 等待 BTF 标志置位。
- 生成 STOP 信号。
- 两次读取 DR (数据寄存器)。

对应的代码流程示例:

```
//产生起始条件
I2C_Generate_StartCondition(I2C1);
//等待起始条件生成完毕
while (!I2C_Get_Flag(I2C1, I2C_FLAG_SB));
//发送地址和读操作
I2C_Transmit_Data8(I2C1, slave_addr_8 | I2C_REQUEST_READ);
//设置 POS 和 ACK 位
I2C_Enable_POSBit(I2C1);
I2C_Set_AcknowledgeNextData(I2C1, I2C_CR1_ACK);
//等待 ADDR 标志置位
while (!I2C_Get_Flag(I2C1, I2C_FLAG_ADDR));
//清除 ADDR
I2C_Clear_Flag(I2C1, I2C_FLAG_ADDR);
//清除 ACK
I2C_Set_AcknowledgeNextData(I2C1, I2C_CR1_NACK);
//等待 BTF 标志置位
while (!I2C_Get_Flag(I2C1, I2C_FLAG_BTF));
//生成 STOP 信号
I2C_Generate_StopCondition(I2C1);
//处理通信数据
for (uint16_t i=0; i<dataLen; i++)
{
    //读取 DataN-1
    if (i == dataLen-2)
    {
        data[i] = I2C_Receive_Data8(I2C1);
    }
    //读取 DataN
    if (i == dataLen-1)
    {
        data[i] = I2C_Receive_Data8(I2C1);
    }
}
```

7-bit 主机接收

10-bit 主机接收

图例: S = 开始, Sr = 重复开始, P = 停止, A = 应答, NA = 非应答,
EVx = 事件 (若ITEVFEN=1, 则伴随中断)

EV5: SB = 1, 通过读取SR1寄存器然后写入DR寄存器来清除。

EV6: ADDR = 1, 通过读取SR1寄存器然后读取SR2寄存器来清除。

EV6.3: ADDR = 1, 设置ACK = 0, 通过读取SR1寄存器然后读取SR2寄存器来清除ADDR, 在ADDR被清除后立即设置STOP = 1。

注: EV6.3软件序列必须在当前字节传输结束之前完成。

EV7: RxNE = 1, 通过读取DR寄存器来清除。

EV9: ADD10 = 1, 通过读取SR1寄存器然后写入DR寄存器来清除。

1. EV5、EV6 和 EV9 事件会将 SCL 拉低，直到相应的软件序列结束。

图 4.3 当接收字节数量 $N=1$ 时主接收器传输时序图

当接收 1 个字节数据时:

- 在 ADDR（地址匹配）事件中，清除 ACK 位。
- 清除 ADDR 标志。
- 生成 STOP 信号。
- 在 RxNE 标志置位后读取数据。

对应的代码流程示例:

```
//产生起始条件
I2C_Generate_StartCondition(I2C1);
//等待起始条件生成完毕
while (!I2C_Get_Flag(I2C1, I2C_FLAG_SB));
//发送地址和读操作
I2C_Transmit_Data8(I2C1, slave_addr_8 | I2C_REQUEST_READ);
//等待 ADDR 标志置位
while (!I2C_Get_Flag(I2C1, I2C_FLAG_ADDR));
//清除 ACK 位
I2C_Set_AcknowledgeNextData(I2C1, I2C_CR1_NACK);
//清除 ADDR 标志
I2C_Clear_Flag(I2C1, I2C_FLAG_ADDR);
//生成 STOP 信号
I2C_Generate_StopCondition(I2C1);
//处理通信数据
while (!I2C_Get_Flag(I2C1, I2C_FLAG_RXNE));
p_data[0] = I2C_Receive_Data8(I2C1);
```

5 版本历史

表 5.1 版本历史

日期	版本	更改内容
2025 年 03 月 28 日	V1.0	初版