AN2204

应用笔记

UM2001A TWI 使用注意事项

版本：V1.3

UNICMICRO

广芯微电子

广芯微电子（广州）股份有限公司

http://www.unicmicro.com/

条款协议

目录

1 摘要

本篇应用笔记介绍 TWI 使用注意事项及配置流程。TWI（Two-Wire Interface）作为 UM2001A 的串行通信接口，在 UM2001A 芯片中有着特定的使用规范和操作流程，对确保芯片与外部设备之间稳定、高效的通信至关重要。

本篇应用笔记主要包括：

- TWI操作说明
- TWI打开操作流程
- 参考示例
- 总结

2 TWI 操作说明

TWI 在线配置模式可对以下 2 种情况的芯片进行在线配置，分别为未烧录参数的芯片和已烧录默认参数的芯片。本应用中的 OTP 未烧录参数和 OTP 已烧录参数，仅考虑寄存器 Reg1A/Reg1B/Reg1C/Reg1D 是否已被烧录参数。

2.1 未烧录参数芯片

未烧录参数是指芯片 OTP 寄存器 Reg1A=0x00，Reg1B=0x00，Reg1C=0x00，Reg1D=0x00，芯片唤醒后，等待芯片上电稳定后，芯片会一直处于 TWI 窗口时间等待 TWI 打开时序。

2.2 烧录默认参数

烧录默认参数是指芯片 OTP 寄存器 Reg1A=0x01，Reg1B=0x00，Reg1C=0x79，Reg1D=0x3F，芯片唤醒后，等待芯片上电稳定后，芯片会处于极低速率发射状态，发射完成后芯片会重新进入睡眠状态，所以 TWI 时序需要在上电稳定后且在进入睡眠前发送，默认参数是一个非常低的速率，使有充足的时间接收 TWI 打开时序，进入 TWI 模式。

2.2.1 烧录默认参数的原因

在低功耗应用中，如果芯片没有烧录默认参数，芯片可能会因 CLK 或 DATA 脚的异常抖动导致误唤醒，而 MCU 又不会对芯片进行 TWI-OFF 操作。这种情况下，芯片会一直处于 IDLE 状态（耗电约 1mA）直到 MCU 完成下一次的正常唤醒/休眠操作。若芯片写入默认参数，芯片误唤醒后仍然会重新进入睡眠状态。

2.3 操作 Reg3F 命令说明

在读写 Reg3F 命令寄存器时，需要在操作 Reg3F 前先复位 TWI，即发送连续 50 个 TWI CLK 时钟，TWI Data 引脚保持低电平后再对 Reg3F 进行操作。

3 TWI 打开操作流程

TWI 打开操作流程分为 3 步，分别为唤醒（上电）、进入 TWI 模式、测试 TWI。

3.1 唤醒（上电）

通过拉低 TWI 的 CLK 引脚唤醒芯片，低电平时间需要保持至少 1.5ms 时间后拉高。芯片唤醒稳定后进入 TWI 窗口。

3.2 进入 TWI 模式

芯片唤醒后，发送 TWI_ON 时序进入 TWI 配置模式，即发送 50 个连续的 0（共 50 个时钟的数据，数据引脚为低电平），可用于复位 TWI 电路同时也进入 TWI 编程模式，以便在遇到未知错误的时候恢复 TWI 功能。

3.3 测试 TWI

测试 TWI 接口是否可以正常读取寄存器值，需要先进行 TWI 复位，即发送 50 个 TWI CLK 时钟，TWI 的 DATA 引脚保持低电平；然后读取寄存器 Reg3F，判断寄存器 Reg3F 的值是否为 0x17，若是则说明芯片已进入 TWI 配置状态，可以正常配置芯片，否则发送 TWI OFF 时序，然后重复以上步骤。

4 参考示例

4.1 定义

```
#define TWI_CLK_DIR_OUTPUT      **   /* 设置为输出 */
#define TWI_CLK_LEVEL_LOW       **   /* 输出低电平 */
#define TWI_CLK_LEVEL_HIGH      **   /* 输出高电平 */
#define TWI_DATA_DIR_INPUT      **   /* 设置为输入 */
#define TWI_DATA_DIR_OUTPUT     **   /* 设置为输出 */
#define TWI_DATA_LEVEL_LOW      **   /* 输出低电平 */
#define TWI_DATA_LEVEL_HIGH     **   /* 输出高电平 */
#define TWI_DATA_LEVEL_READ     **   /* 读取电平 */
#define TWI_DELAY                    /* TWI 延迟，如果 TWI 速率过快，需加延迟 */
```

4.2 TWI 读写字节

```
/*************************************************************
 * Function       : twi_write_byte
 * Description    : TWI 写一个字节
 * Input          : uint8_t byte
 * Output         : none
 * Return         : none
 ************************************************************/
void twi_write_byte(uint8_t byte)
{
    TWI_DATA_DIR_OUTPUT;                    /* data 脚设置为输出 */
    for(uint8_t i=0;i<8;i++)
    {
        if(byte&0x80)                       /* 高位如果为 1 则拉高电平 */
        {
            TWI_DATA_LEVEL_HIGH;
        }
        else
        {
            TWI_DATA_LEVEL_LOW;             /* 为 0 则拉低电平 */
        }
```

```
            TWI_CLK_LEVEL_HIGH;                    /* 拉高电平 */
            TWI_DELAY;
            TWI_CLK_LEVEL_LOW;                     /* 拉低电平 */
            TWI_DELAY;
            byte <<= 1;                            /* 左移 1 位 */
        }
        TWI_CLK_LEVEL_HIGH;
        TWI_DATA_LEVEL_HIGH;
    }

    /*********************************************************
     * Function       : twi_read_byte
     * Description    : twi 读一个字节
     * Input          : none
     * Output         : none
     * Return         : uint8_t
     *********************************************************/
    uint8_t twi_read_byte(void)
    {
        TWI_DATA_DIR_INPUT;
        uint8_t value = 0;
        for(uint8_t i=0;i<8;i++)
        {
            value <<= 1;
            TWI_CLK_LEVEL_HIGH;
            TWI_DELAY;
            TWI_CLK_LEVEL_LOW;
            TWI_DELAY;
            if(TWI_DATA_LEVEL_READ == 1)
            {
                value++;
            }
        }
        TWI_DATA_DIR_OUTPUT;                       /* data 脚设置为输出 */
        TWI_DATA_LEVEL_HIGH;
        TWI_CLK_LEVEL_HIGH;                        /* 拉高 clk 脚电平 */
        return value;
    }
```

```
/*********************************************************
 * Function      : twi_ reset
 * Description   : TWI 复位
 * Input         : none
 * Output        : none
 * Return        : none
 *********************************************************/
void twi_reset(void)
{
    TWI_DATA_LEVEL_LOW;
    for(uint8_t i=0;i<50;i++)
    {
        TWI_CLK_LEVEL_LOW;                    /* 拉低电平 */
        TWI_DELAY;
        TWI_CLK_LEVEL_HIGH;                   /* 拉高电平 */
        TWI_DELAY;
    }
}
```

4.3 TWI 操作

```
/*********************************************************
 * Function      : twi_on
 * Description   : 打开 TWI
 * Input         : none
 * Output        : none
 * Return        : uint8_t
 *********************************************************/
uint8_t twi_on(void)
{
    uint8_t count = 10;
    do
    {
        /* 唤醒 */
        TWI_DATA_LEVEL_LOW;
        TWI_CLK_LEVEL_LOW;
```

```
            delay_us(1500);
            TWI_CLK_LEVEL_HIGH;

            /* twi on */
            TWI_DATA_LEVEL_LOW;
            for(uint8_t i=0;i<50;i++)
            {
                TWI_CLK_LEVEL_LOW;
                TWI_DELAY;
                TWI_CLK_LEVEL _HIGH;
                TWI_DELAY;
            }

            /*  读 Reg3F 确认  */
            twi_write_byte(0x3F|0x80);
            if(twi_read_byte() == 0x17)
            {
                return SUCCESS;
            }
            twi_off();
        }while(count--);
        return FAILED;
    }

    /*********************************************************
     * Function        : twi_off
     * Description     :  关闭 twi
     * Input           : none
     * Output          : none
     * Return          : none
     *********************************************************/
    void twi_off(void)
    {
        twi_reset();
        twi_write_byte(0xFF);
        twi_write_byte(0x02);
        delay_ms(10);                                   /*   等待芯片完全关闭  */
    }
```

```
/************************************************************
 * Function        : init
 * Description     : 初始化
 * Input           : none
 * Output          : none
 * Return          : none
 ************************************************************/
em_ret_t init(void)
{
    em_ret_t ret = FAILED;

    twi_init();                                          /* 初始化 TWI */
    ret = twi_on();
    if(ret   == SUCCESS)
    {
        static uint8_t flag = 0;
        if(flag == 0)                                    /* 仅上电调用一次即可 */
        {
            flag = 1;
            ret = FAILED;
            twi_write_reg(0x3F,0x20);
            delay_ms(1);

            uint8_t reg1A = twi_read_reg(0x1A);
            uint8_t reg1B = twi_read_reg(0x1B);
            uint8_t reg1C = twi_read_reg(0x1C);
            uint8_t reg1D = twi_read_reg(0x1D);

            static uint8_t rf_cfg[4] = {0x01,0x00,0x79,0x3F};
            if(reg1A == 0x00 && reg1B == 0x00 && reg1C == 0x00 && reg1D == 0x00)
            {
                twi_write_reg(0x1A,rf_cfg[0]);
                twi_write_reg(0x1B,rf_cfg[1]);
                twi_write_reg(0x1C,rf_cfg[2]);
                twi_write_reg(0x1D,rf_cfg[3]);
                twi_write_reg(0x3F,0x40);
                delay_ms(200);
```

```
                    ret = SUCCESS;
                }
                else
                {
                    if(reg1A == rf_cfg[0] && reg1B == rf_cfg[1] && reg1C == rf_cfg[2] && reg1D
== rf_cfg[3])
                    {
                        ret = SUCCESS;
                    }
                    else
                    {
                        ret = FAILED;
                    }
                }
            }

            uint8_t len = sizeof(rf_config)/sizeof(rf_config[0]);
            for(uint8_t i=0;i<len;i++)
            {
                if(rf_config[i][0] != 0xFF)
                {
                    twi_write_reg(rf_config[i][0],rf_config[i][1]);
                }
            }
        }
        return ret;
    }
```

5 总结

1. 操作 Reg3F 前需要发射时序复位 TWI 接口。
2. TWI 在线配置未烧录参数或已烧录默认参数的芯片。
3. 芯片唤醒后，需等待芯片上电稳定后，发送 TWI 打开时序，使芯片进入 TWI 模式。
4. TWI 打开需要读取 Reg3F 是否为 0x17 来判断是否可正常读写寄存器。

6 版本修订

版本	日期	描述
V1.0	2025.03.25	初始版
V1.1	2025.04.11	更新 TWI 操作代码。
V1.2	2025.09.27	更新 TWI 操作时序。
V1.3	2025.10.28	更新 TWI 操作代码。