

# 零死角玩转STM32

与野火同行 乐意惬意无边



原创教程，完全开源。



由浅入深，结合实操。



通俗易懂，详尽解读。



配套板子，全面玩转。



强强联合，不断更新。



野火团队 Wild Fire Team

## 0、友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬意无边。

另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！

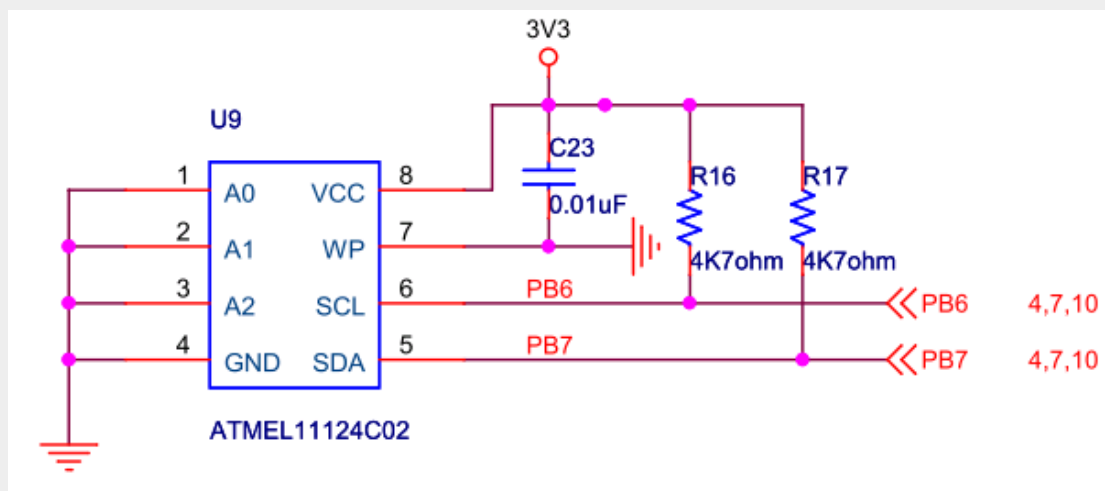


## 5、IIC（EEPROM）

### 5.1 实验描述及工程文件清单

实验描述	向 EEPROM 写入数据，再读取出来，进行校验，通过串口打印写入与读取出来的数据，并输出校验结果。
硬件连接	PB6-I2C1_SCL, PB7-I2C1_SDA
用到的库文件	startup/start_stm32f10x_hd.c CMSIS/core_cm3.c CMSIS/system_stm32f10x.c FWlib/stm32f10x_gpio.c FWlib/stm32f10x_rcc.c FWlib/stm32f10x_usart.c FWlib/stm32f10x_i2c.c
用户编写的文件	USER/main.c USER/stm32f10x_it.c USER/usart1.c USER/i2c_ee.c

野火 STM32 开发板 I2C-EEPROM 硬件原理图：



## 5.2 I2C 简介

I2C(芯片间)总线接口连接微控制器和串行 I2C 总线。它提供多主机功能，控制所有 I2C 总线特定的时序、协议、仲裁和定时。支持标准和快速两种模式，stm32 的 I2C 可以使用 DMA 方式操作。

野火 STM32 开发板用的是 [STM32F103VET6](#)。它有 2 个 I2C 接口。I/O 口定义为 [PB6-I2C1\\_SCL](#), [PB7-I2C1\\_SDA](#); [PB10-I2C2\\_SCL](#), [PB11-I2C2\\_SDA](#)。本实验使用 [I2C1](#)，对应地连接到 EERPOM（型号：[AT24C02](#)）的 [SCL](#) 和 [SDA](#) 线。实现 I2C 通讯，对 EERPOM 进行读写。

本实验采用主模式，分别用作主发送器和主接收器。通过查询事件的方式来确保正常通讯。

## 5.3 代码分析

首先要添加用的库文件，在工程文件夹下 **Fwlib** 下我们需添加以下库文件：

```
1. stm32f10x_gpio.c
2. stm32f10x_rcc.c
3. stm32f10x_usart.c
4. stm32f10x_i2c.c
```

还要在 [stm32f10x\\_conf.h](#) 中把相应的头文件添加进来：

```
1. /* Uncomment the line below to enable peripheral header file inclusion
   */
2. /* #include "stm32f10x_adc.h" */
3. /* #include "stm32f10x_bkp.h" */
4. /* #include "stm32f10x_can.h" */
5. /* #include "stm32f10x_crc.h" */
6. /* #include "stm32f10x_dac.h" */
7. /* #include "stm32f10x_dbgmcu.h" */
8. /* #include "stm32f10x_dma.h" */
9. /* #include "stm32f10x_exti.h" */
10. /* #include "stm32f10x_flash.h" */
11. /* #include "stm32f10x_fsmc.h" */
12. #include "stm32f10x_gpio.h"
13. #include "stm32f10x_i2c.h"
14. /* #include "stm32f10x_iwdg.h" */
15. /* #include "stm32f10x_pwr.h" */
16. #include "stm32f10x_rcc.h"
17. /* #include "stm32f10x_rtc.h" */
18. /* #include "stm32f10x_sdio.h" */
```



```
19. /* #include "stm32f10x_spi.h" */
20. /* #include "stm32f10x_tim.h" */
21. #include "stm32f10x_usart.h"
22. /* #include "stm32f10x_wwdg.h" */
23. /*#include "misc.h"*/ /* High level functions for NVIC and SysTick (a
    dd-on to CMSIS functions) */
```

配置好所需的库文件之后，我们就从 `main` 函数开始分析：

```
1. /*
2.  * 函数名: main
3.  * 描述   : 主函数
4.  * 输入   : 无
5.  * 输出   : 无
6.  * 返回   : 无
7.  */
8. int main(void)
9. {
10.    /* 配置系统时钟为 72M */
11.    SystemInit();
12.
13.    /* 串口1 初始化 */
14.    USART1_Config();
15.
16.    /* I2C 外设初(AT24C02)始化 */
17.    I2C_EE_Init();
18.
19.    USART1_printf(USART1, "\r\n 这是一个 I2C 外设(AT24C02)读写测试例
    程 \r\n");
20.    USART1_printf(USART1, "\r\n (" __DATE__ " -
    " __TIME__ ") \r\n");
21.
22.    I2C_Test();
23.
24.    while (1)
25.    {
26.    }
27. }
```

系统库函数 `SystemInit()`；将系统时钟设置为 72M，`USART1_Config()`；配置串口，关于这两个函数的具体讲解可以参考前面的教程，这里不再详述。/\*

```
1.  * 函数名: I2C_EE_Init
2.  * 描述   : I2C 外设(EEPROM)初始化
3.  * 输入   : 无
4.  * 输出   : 无
5.  * 调用   : 外部调用
6.  */
7. void I2C_EE_Init(void)
8. {
9.
10.    I2C_GPIO_Config();
11.
12.    I2C_Mode_Config();
13.
14.    /* 根据头文件 i2c_ee.h 中的定义来选择 EEPROM 要写入的地址 */
15.    #ifdef EEPROM_Block0_ADDRESS
16.    /* 选择 EEPROM Block0 来写入 */
17.    EEPROM_ADDRESS = EEPROM_Block0_ADDRESS;
18.    #endif
```



```
19.
20. #ifndef EEPROM_Block1_ADDRESS
21.     /* 选择 EEPROM Block1 来写入 */
22.     EEPROM_ADDRESS = EEPROM_Block1_ADDRESS;
23. #endif
24.
25. #ifndef EEPROM_Block2_ADDRESS
26.     /* 选择 EEPROM Block2 来写入 */
27.     EEPROM_ADDRESS = EEPROM_Block2_ADDRESS;
28. #endif
29.
30. #ifndef EEPROM_Block3_ADDRESS
31.     /* 选择 EEPROM Block3 来写入 */
32.     EEPROM_ADDRESS = EEPROM_Block3_ADDRESS;
33. #endif }
```

`I2C_EE_Init()`; 是用户编写的函数, 其中调用了 `I2C_GPIO_Config()`; 配置好 I2C 所用的 I/O 端口, 调用 `I2C_Mode_Configu()`; 设置 I2C 的工作模式。并使能相关外设的时钟。其中的条件编译确定了 EEPROM 的器件地址, 按我们的硬件设置方式, 地址为 `0xA0`;

```
1. /*
2.  * 函数名: I2C_EE_Test
3.  * 描述   : I2C(AT24C02)读写测试。
4.  * 输入   : 无
5.  * 输出   : 无
6.  * 返回   : 无
7.  */
8. void I2C_Test(void)
9. {
10.     u16 i;
11.
12.     printf("写入的数据\n\r");
13.
14.     for ( i=0; i<=255; i++ ) //填充缓冲
15.     {
16.         I2c_Buf_Write[i] = i;
17.
18.         printf("0x%02X ", I2c_Buf_Write[i]);
19.         if(i%16 == 15)
20.             printf("\n\r");
21.     }
22.
23.     //将 I2c_Buf_Write 中顺序递增的数据写入 EEPROM 中
24.     I2C_EE_BufferWrite( I2c_Buf_Write, EEP_Firstpage, 256);
25.
26.     printf("\n\r 读出的数据\n\r");
27.     //将 EEPROM 读出数据顺序保持到 I2c_Buf_Read 中
28.     I2C_EE_BufferRead(I2c_Buf_Read, EEP_Firstpage, 256);
29.
30.     //将 I2c_Buf_Read 中的数据通过串口打印
31.     for (i=0; i<256; i++)
32.     {
33.         if(I2c_Buf_Read[i] != I2c_Buf_Write[i])
34.         {
35.             printf("0x%02X ", I2c_Buf_Read[i]);
36.             printf("错误:I2C EEPROM 写入与读出的数据不一致\n\r");
37.             return;
38.         }
39.         printf("0x%02X ", I2c_Buf_Read[i]);
```



```
40.     if(i%16 == 15)
41.         printf("\n\r");
42.
43.     }
44.     printf("I2C(AT24C02) 读写测试成功\n\r");
45. }
```

I2C\_Test(void)是这个例程中最主要的部分，把 0~255 按顺序填入缓冲区并通过串口打印到端口，接着把缓冲区的数据通过调用 I2C\_EE\_BufferWrite() 函数写入 EEPROM。

```
1.  /*
2.  * 函数名: I2C_EE_BufferWrite
3.  * 描述   : 将缓冲区中的数据写到 I2C EEPROM 中
4.  * 输入   : -pBuffer 缓冲区指针
5.  *          -WriteAddr 接收数据的 EEPROM 的地址
6.  *          -NumByteToWrite 要写入 EEPROM 的字节数
7.  * 输出   : 无
8.  * 返回   : 无
9.  * 调用   : 外部调用
10. */
11. void I2C_EE_BufferWrite(u8* pBuffer, u8 WriteAddr, u16 NumByteToWrite)
12. {
13.     u8 NumOfPage = 0, NumOfSingle = 0, Addr = 0, count = 0;
14.
15.     Addr = WriteAddr % I2C_PageSize;
16.     count = I2C_PageSize - Addr;
17.     NumOfPage = NumByteToWrite / I2C_PageSize;
18.     NumOfSingle = NumByteToWrite % I2C_PageSize;
19.
20.     /* If WriteAddr is I2C_PageSize aligned */
21.     if(Addr == 0)
22.     {
23.         /* If NumByteToWrite < I2C_PageSize */
24.         if(NumOfPage == 0)
25.         {
26.             I2C_EE_PageWrite(pBuffer, WriteAddr, NumOfSingle);
27.             I2C_EE_WaitEepromStandbyState();
28.         }
29.         /* If NumByteToWrite > I2C_PageSize */
30.         else
31.         {
32.             while (NumOfPage--)
33.             {
34.                 I2C_EE_PageWrite(pBuffer, WriteAddr, I2C_PageSize);
35.                 I2C_EE_WaitEepromStandbyState();
36.                 WriteAddr += I2C_PageSize;
37.                 pBuffer += I2C_PageSize;
38.             }
39.
40.             if(NumOfSingle!=0)
41.             {
42.                 I2C_EE_PageWrite(pBuffer, WriteAddr, NumOfSingle);
43.                 I2C_EE_WaitEepromStandbyState();
44.             }
45.         }
46.     }
47.     /* If WriteAddr is not I2C_PageSize aligned */
48.     else
49.     {
50.         /* If NumByteToWrite < I2C_PageSize */
```

```
51.     if(NumOfPage== 0)
52.     {
53.         I2C_EE_PageWrite(pBuffer, WriteAddr, NumOfSingle);
54.         I2C_EE_WaitEepromStandbyState();
55.     }
56.     /* If NumByteToWrite > I2C_PageSize */
57.     else
58.     {
59.         NumByteToWrite -= count;
60.         NumOfPage =  NumByteToWrite / I2C_PageSize;
61.         NumOfSingle = NumByteToWrite % I2C_PageSize;
62.
63.         if(count != 0)
64.         {
65.             I2C_EE_PageWrite(pBuffer, WriteAddr, count);
66.             I2C_EE_WaitEepromStandbyState();
67.             WriteAddr += count;
68.             pBuffer += count;
69.         }
70.
71.         while (NumOfPage--)
72.         {
73.             I2C_EE_PageWrite(pBuffer, WriteAddr, I2C_PageSize);
74.             I2C_EE_WaitEepromStandbyState();
75.             WriteAddr += I2C_PageSize;
76.             pBuffer += I2C_PageSize;
77.         }
78.         if(NumOfSingle != 0)
79.         {
80.             I2C_EE_PageWrite(pBuffer, WriteAddr, NumOfSingle);
81.             I2C_EE_WaitEepromStandbyState();
82.         }
83.     }
84. }
85. }
```

因为 AT24C02 型号的 EEPROM 按页写入方式中每页最大字节数为 8 字节，若超过 8 字节则会在该页的起始地址覆盖数据，因此需要

I2C\_EE\_BufferWrite() 函数处理写入位置和缓冲区的地址。把处理好的地址交给 I2C\_EE\_PageWrite() 函数，这个函数是与 EEPROM 进行 I2C 通讯的最底层函数，以下我们通过分析 I2C\_EE\_PageWrite() 来了解 stm32 的 I2C 通讯方法。

```
1.  /*
2.  * 函数名: I2C_EE_PageWrite
3.  * 描述   : 在 EEPROM 的一个写循环中可以写多个字节，但一次写入的字节数
4.  *          不能超过 EEPROM 页的大小。AT24C02 每页有 8 个字节。
5.  * 输入    : -pBuffer 缓冲区指针
6.  *          -WriteAddr 接收数据的 EEPROM 的地址
7.  *          -NumByteToWrite 要写入 EEPROM 的字节数
8.  * 输出    : 无
9.  * 返回    : 无
10. * 调用    : 外部调用
11. */
12. void I2C_EE_PageWrite(u8* pBuffer, u8 WriteAddr, u8 NumByteToWrite)
13. {
14.     while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY)); // Added by Najoua
15.     27/08/2008
16.     /* Send START condition */
```

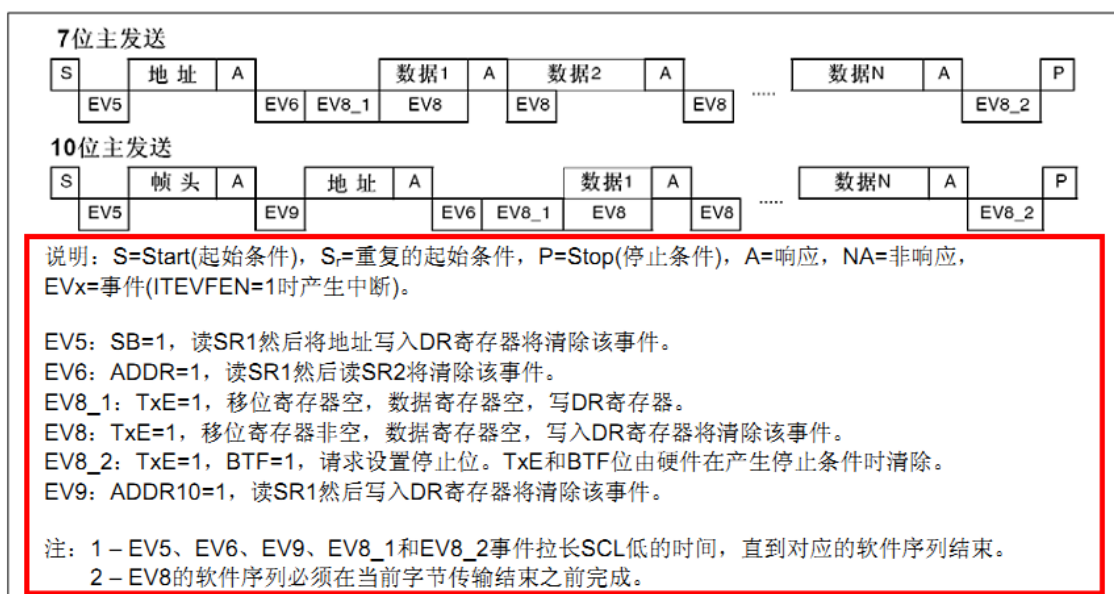


```
17. I2C_GenerateSTART(I2C1, ENABLE);
18.
19. /* Test on EV5 and clear it */
20. while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
21.
22. /* Send EEPROM address for write */
23. I2C_Send7bitAddress(I2C1, EEPROM_ADDRESS, I2C_Direction_Transmitter)
    ;
24.
25. /* Test on EV6 and clear it */
26. while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECT
    ED));
27.
28. /* Send the EEPROM's internal address to write to */
29. I2C_SendData(I2C1, WriteAddr);
30.
31. /* Test on EV8 and clear it */
32. while(! I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
33.
34. /* While there is data to be written */
35. while(NumByteToWrite--)
36. {
37.     /* Send the current byte */
38.     I2C_SendData(I2C1, *pBuffer);
39.
40.     /* Point to the next byte to be written */
41.     pBuffer++;
42.
43.     /* Test on EV8 and clear it */
44.     while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
45. }
46. /* Send STOP condition */
47. I2C_GenerateSTOP(I2C1, ENABLE);
48. }
```

从 stm32 参考手册的序列图可以看到，在 I2C 的通讯过程中，会产生一系列的事件，出现事件后在相应的寄存器中会产生标志位。

截图来自《STM32 参考手册中文》。

图245 主发送器传送序列图



我们在做出 I2C 通讯操作时，可以通过循环调用库函数 `I2C_CheckEvent()` 进行查询，以确保上一操作完成后才发出下一个

I2C 通讯信号。如：在确定 SDA 总线空闲的之后，作为主发送器的 stm32 发出起始信号，若成功，这时会产生“事件 5”（EV5），我们调用

`while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));` 来检测这个事件，确保检测到之后再执行下一操作。“`I2C_EVENT_MASTER_MODE_SELECT`”在固件函数库中可以查到这就是“EV5”的宏，后面的相应操作类似。

现在我们回到 `I2C_EE_BufferWrite()` 这个函数，在每次调用完 `I2C_EE_PageWrite()` 后，都调用了一个 `I2C_EE_WaitEepromStandbyState()` 函数。

```
1.  /*
2.  * 函数名: I2C_EE_WaitEepromStandbyState
3.  * 描述   : Wait for EEPROM Standby state
4.  * 输入   : 无
5.  * 输出   : 无
6.  * 返回   : 无
7.  * 调用   :
8.  */
9. void I2C_EE_WaitEepromStandbyState(void)
10. {
11.     vu16 SR1_Tmp = 0;
12.
13.     do
14.     {
15.         /* Send START condition */
16.         I2C_GenerateSTART(I2C1, ENABLE);
17.         /* Read I2C1 SR1 register */
18.         SR1_Tmp = I2C_ReadRegister(I2C1, I2C_Register_SR1);
19.         /* Send EEPROM address for write */
20.         I2C_Send7bitAddress(I2C1, EEPROM_ADDRESS, I2C_Direction_Transmitter);
21.     } while (!(I2C_ReadRegister(I2C1, I2C_Register_SR1) & 0x0002));
22.
23.     /* Clear AF flag */
24.     I2C_ClearFlag(I2C1, I2C_FLAG_AF);
25.     /* STOP condition */
26.     I2C_GenerateSTOP(I2C1, ENABLE); // Added by Najoua 27/08/2008
27. }
```

这是利用了 EEPROM 在接收完数据后，启动内部周期写入数据的时间内不会对主机的请求作出应答的特性。所以这个函数循环发送起始信号，若检测到 EEPROM 的应答，则说明 EEPROM 已经完成上一步的数据写入，进入 Standby 状态，可以进行下一步的操作了。

回到 `I2C_Test()` 这个函数，再分析一下它调用的读 EEPROM 函数

`I2C_EE_BufferRead()`。

```
1.  /*
2.  * 函数名: I2C_EE_BufferRead
3.  * 描述   : 从EEPROM里面读取一块数据。
4.  * 输入   : -pBuffer 存放从EEPROM读取的数据的缓冲区指针。
5.  *          -WriteAddr 接收数据的EEPROM的地址。
6.  *          -NumByteToWrite 要从EEPROM读取的字节数。
7.  * 输出   : 无
8.  * 返回   : 无
9.  * 调用   : 外部调用
10. */
11. void I2C_EE_BufferRead(u8* pBuffer, u8 ReadAddr, u16 NumByteToRead)
12. {
13.     /*((u8 *)0x4001080c) |= 0x80;
14.     while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY)); // Added by Najoua
15.     27/08/2008
16.
17.     /* Send START condition */
18.     I2C_GenerateSTART(I2C1, ENABLE);
19.     /*((u8 *)0x4001080c) &= ~0x80;
20.
21.     /* Test on EV5 and clear it */
22.     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
23.
24.     /* Send EEPROM address for write */
25.     I2C_Send7bitAddress(I2C1, EEPROM_ADDRESS, I2C_Direction_Transmitter)
26.     ;
27.     /* Test on EV6 and clear it */
28.     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECT
29.     ED));
30.     /* Clear EV6 by setting again the PE bit */
31.     I2C_Cmd(I2C1, ENABLE);
32.
33.     /* Send the EEPROM's internal address to write to */
34.     I2C_SendData(I2C1, ReadAddr);
35.
36.     /* Test on EV8 and clear it */
37.     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
38.
39.     /* Send STRAT condition a second time */
40.     I2C_GenerateSTART(I2C1, ENABLE);
41.
42.     /* Test on EV5 and clear it */
43.     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
44.
45.     /* Send EEPROM address for read */
46.     I2C_Send7bitAddress(I2C1, EEPROM_ADDRESS, I2C_Direction_Receiver);
47.
48.     /* Test on EV6 and clear it */
49.     while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)
50.     );
51.     /* While there is data to be read */
52.     while(NumByteToRead)
53.     {
54.         if(NumByteToRead == 1)
55.         {
56.             /* Disable Acknowledgement */
57.             I2C_AcknowledgeConfig(I2C1, DISABLE);
58.
59.             /* Send STOP Condition */
60.             I2C_GenerateSTOP(I2C1, ENABLE);
61.         }
62.     }
```



```
63.     /* Test on EV7 and clear it */
64.     if(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED))
65.     {
66.         /* Read a byte from the EEPROM */
67.         *pBuffer = I2C_ReceiveData(I2C1);
68.
69.         /* Point to the next location where the byte read will be saved
        */
70.         pBuffer++;
71.
72.         /* Decrement the read bytes counter */
73.         NumByteToRead--;
74.     }
75. }
76.
77. /* Enable Acknowledgement to be ready for another reception */
78. I2C_AcknowledgeConfig(I2C1, ENABLE);
79. }
```

这个读 EEPROM 函数与写的类似，也是利用 `I2C_CheckEvent()` 来确保通讯正常进行的，要注意一下的是读取数据时遵循 I2C 的标准，主发送器 stm32 要发出两次起始 I2C 讯号才能建立通讯。

最后，总结一下在 stm32 如何建立与 EEPROM 的通讯。

1、 配置 I/O 端口，确定并配置 I2C 的模式，使能 GPIO 和 I2C 时钟。

2、 写：

检测 SDA 是否空闲；

->按 I2C 协议发出起始讯号；

->发出 7 位器件地址和写模式；

->要写入的存储区首地址；

->用页写入方式或字节写入方式写入数据；

每个操作之后要检测“事件”确定是否成功。写完后检测 EEPROM 是否进入 standby 状态。

3、 读：

检测 SDA 是否空闲；

->按 I2C 协议发出起始讯号；

->发出 7 位器件地址和写模式（伪写）；

->发出要读取的存储区首地址；

->重发起始讯号;

->发出 7 位器件地址和读模式;

->接收数据;

类似写操作，每个操作之后要检测“事件”确定是否成功。

## 5.4 实验现象

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，即可看到超级终端打印出如下信息：

写入的数据：

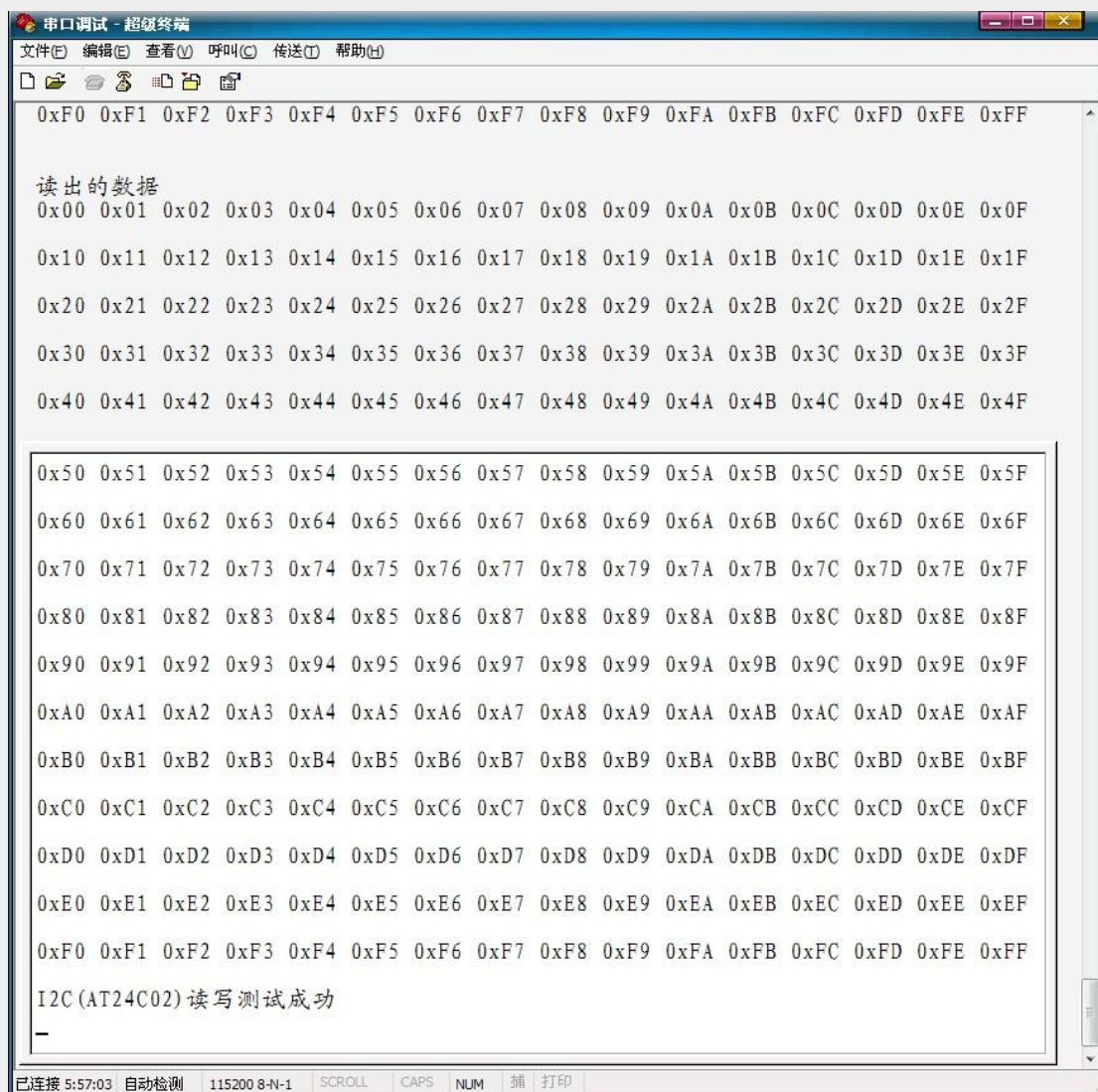
```
0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0xFF

这是一个I2C外设(AT24C02)读写测试例程

(Oct 27 2011 - 20:00:11)
写入的数据
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1A 0x1B 0x1C 0x1D 0x1E 0x1F
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2A 0x2B 0x2C 0x2D 0x2E 0x2F
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3A 0x3B 0x3C 0x3D 0x3E 0x3F
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4A 0x4B 0x4C 0x4D 0x4E 0x4F
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5A 0x5B 0x5C 0x5D 0x5E 0x5F
0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F
0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7A 0x7B 0x7C 0x7D 0x7E 0x7F
0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8A 0x8B 0x8C 0x8D 0x8E 0x8F
0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9A 0x9B 0x9C 0x9D 0x9E 0x9F
0xA0 0xA1 0xA2 0xA3 0xA4 0xA5 0xA6 0xA7 0xA8 0xA9 0xAA 0xAB 0xAC 0xAD 0xAE 0xAF
0xB0 0xB1 0xB2 0xB3 0xB4 0xB5 0xB6 0xB7 0xB8 0xB9 0xBA 0xBB 0xBC 0xBD 0xBE 0xBF
0xC0 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7 0xC8 0xC9 0xCA 0xCB 0xCC 0xCD 0xCE 0xCF
0xD0 0xD1 0xD2 0xD3 0xD4 0xD5 0xD6 0xD7 0xD8 0xD9 0xDA 0xDB 0xDC 0xDD 0xDE 0xDF
0xE0 0xE1 0xE2 0xE3 0xE4 0xE5 0xE6 0xE7 0xE8 0xE9 0xEA 0xEB 0xEC 0xED 0xEE 0xEF
0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0xFF
```



读出的数据：校验结果，读取出的数据与写入的一致，实验成功！



串口调试 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0xFF

读出的数据

0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F

0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1A 0x1B 0x1C 0x1D 0x1E 0x1F

0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2A 0x2B 0x2C 0x2D 0x2E 0x2F

0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3A 0x3B 0x3C 0x3D 0x3E 0x3F

0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4A 0x4B 0x4C 0x4D 0x4E 0x4F

0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5A 0x5B 0x5C 0x5D 0x5E 0x5F

0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F

0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7A 0x7B 0x7C 0x7D 0x7E 0x7F

0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8A 0x8B 0x8C 0x8D 0x8E 0x8F

0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9A 0x9B 0x9C 0x9D 0x9E 0x9F

0xA0 0xA1 0xA2 0xA3 0xA4 0xA5 0xA6 0xA7 0xA8 0xA9 0xAA 0xAB 0xAC 0xAD 0xAE 0xAF

0xB0 0xB1 0xB2 0xB3 0xB4 0xB5 0xB6 0xB7 0xB8 0xB9 0xBA 0xBB 0xBC 0xBD 0xBE 0xBF

0xC0 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7 0xC8 0xC9 0xCA 0xCB 0xCC 0xCD 0xCE 0xCF

0xD0 0xD1 0xD2 0xD3 0xD4 0xD5 0xD6 0xD7 0xD8 0xD9 0xDA 0xDB 0xDC 0xDD 0xDE 0xDF

0xE0 0xE1 0xE2 0xE3 0xE4 0xE5 0xE6 0xE7 0xE8 0xE9 0xEA 0xEB 0xEC 0xED 0xEE 0xEF

0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0xFF

I2C(AT24C02)读写测试成功

已连接 5:57:03 自动检测 115200 8-N-1 SCROLL CAPS NUM 捕 打印

