ARC

I2C

支持	www.armrunc.com http://armrunc.taobao.com

Change History

版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-3-6

目录

第 12 章 I2C.....3
12.1 I2C 简介.....3
12.2 I2C 应用实例 ----- 读写 EEPROM.....6
12.2.1 实例描述.....6
12.2.2 硬件设计.....6
12.2.3 软件设计.....6

第12章 I2C

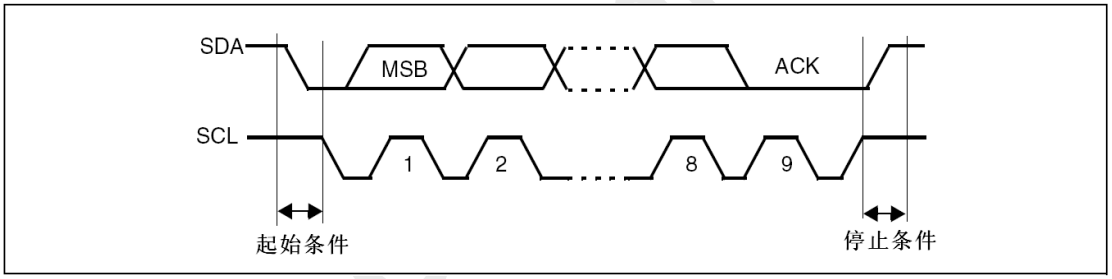
12.1 I2C 简介

I2C（Inter－Integrated Circuit）总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。

它的主要特性如下：

- 只要求两条总线线路：一条串行数据线 SDA，一条串行时钟线 SCL；
- 每个连接到总线的器件都可以通过唯一的地址和一直存在的简单的主机/从机关系软件设定地址，主机可以作为主机发送器或主机接收器；
- 它是一个真正的多主机总线，如果两个或更多主机同时初始化，数据传输可以通过冲突检测和仲裁防止数据被破坏；
- 串行的 8 位双向数据传输位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；
- 连接到相同总线的 IC 数量只受到总线的最大电容 400pF 限制。

数据和地址按 8 位/字节进行传输的波形图如下，高位在前。以下波形包含了起始位，停止位和应答位和数据。



接口可以下述4种模式中的一种运行：

- 从发送器模式
- 从接收器模式
- 主发送器模式
- 主接收器模式

以下重点介绍主发送模式和主接收，因为下面的实例就是用这两种模式，写入和读取 E2PROM 内容。

在主模式时，I2C 接口启动数据传输并产生时钟信号。串行数据传输总是以起始条件开始并以停止条件结束。当通过 START 位在总线上产生了起始条件，设备就进入了主模式。

通过以下操作顺序就可以进入主模式：

1. 在 I2C_CR2 寄存器中设定该模块的输入时钟以产生正确的时序
2. 配置时钟控制寄存器
3. 配置上升时间寄存器
4. 编程 I2C_CR1 寄存器启动外设
5. 置 I2C_CR1 寄存器中的 START 位为 1，产生起始条件

I2C 模块的频率必须至少是:

- 标准模式下为：2MHz
- 快速模式下为：4MHz

当 BUSY=0 时，设置 START=1，I2C 接口将产生一个开始条件并切换至主模式(M/SL 位置位)。

一旦发出开始条件，SB 位被硬件置位，如果设置了 ITEVFEN 位，则会产生一个中断。然后主设备等待读 SR1 寄存器，紧跟着将从地址写入 DR 寄存器。

从地址的发送

从地址通过内部移位寄存器被送到 SDA 线上。

- 在 7 位地址模式时，只需送出一个地址字节。一旦该地址字节被送出，ADDR 位被硬件置位，如果设置了 ITEVFEN 位，则产生一个中断。随后主设备等待一次读 SR1 寄存器，跟着读 SR2 寄存器。根据送出从地址的最低位，主设备决定进入发送器模式还是进入接收器模式。
- 在 7 位地址模式时，

— 要进入发送器模式，主设备发送从地址时置最低位为’0’。
— 要进入接收器模式，主设备发送从地址时置最低位为’1’。

主发送器

在发送了地址和清除了 ADDR 位后，主设备通过内部移位寄存器将字节从 DR 寄存器发送到 SDA 线上。主设备等待，直到 TxE 被清除。

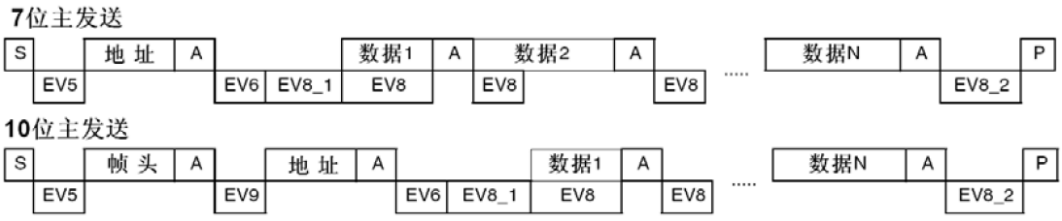
当收到应答脉冲时:

- TxE 位被硬件置位，如果设置了 INEVFEN 和 ITBUFEN 位，则产生一个中断。如果 TxE 被置位并且在上一次数据发送结束之前没有写新的数据字节到 DR 寄存器，则 BTF 被硬件置位，在清除 BTF 之前 I2C 接口将保持 SCL 为低电平；读出 I2C_SR1 之后再写入 I2C_DR 寄存器将清除 BTF 位。

关闭通信

在 DR 寄存器中写入最后一个字节后，通过设置 STOP 位产生一个停止条件然后 I2C 接口将自动回到从模式。

主发送器传送序列图如下：



说明：S=Start(起始条件)，S_r=重复的起始条件，P=Stop(停止条件)，A=响应，NA=非响应，
EVx=事件(ITEVFEN=1时产生中断)。

EV5：SB=1，读SR1然后将地址写入DR寄存器将清除该事件。
EV6：ADDR=1，读SR1然后读SR2将清除该事件。
EV8_1：TxE=1，移位寄存器空，数据寄存器空，写DR寄存器。
EV8：TxE=1，移位寄存器非空，数据寄存器空，写入DR寄存器将清除该事件。
EV8_2：TxE=1，BTF=1，请求设置停止位。TxE和BTF位由硬件在产生停止条件时清除。
EV9：ADDR10=1，读SR1然后写入DR寄存器将清除该事件。

注：1 – EV5、EV6、EV9、EV8_1和EV8_2事件拉长SCL低的时间，直到对应的软件序列结束。
2 – EV8的软件序列必须在当前字节传输结束之前完成。

主接收器

在发送地址和清除 ADDR 之后，I2C 接口进入主接收器模式。在此模式下，I2C 接口从 SDA 线接收数据字节，并通过内部移位寄存器送至 DR 寄存器。在每个字节后，I2C 接口依次执行以下操作：

- 如果 ACK 位被置位，发出一个应答脉冲。
- 硬件设置 RxNE=1，如果设置了 INEVFEN 和 ITBUFEN 位，则会产生一个中断(EV7)。

如果 RxNE 位被置位，并且在接收新数据结束前，DR 寄存器中的数据没有被读走，硬件将设置 BTF=1，在清除 BTF 之前 I2C 接口将保持 SCL 为低电平；读出 I2C_SR1 之后再读出 I2C_DR 寄存器将清除 BTF 位。

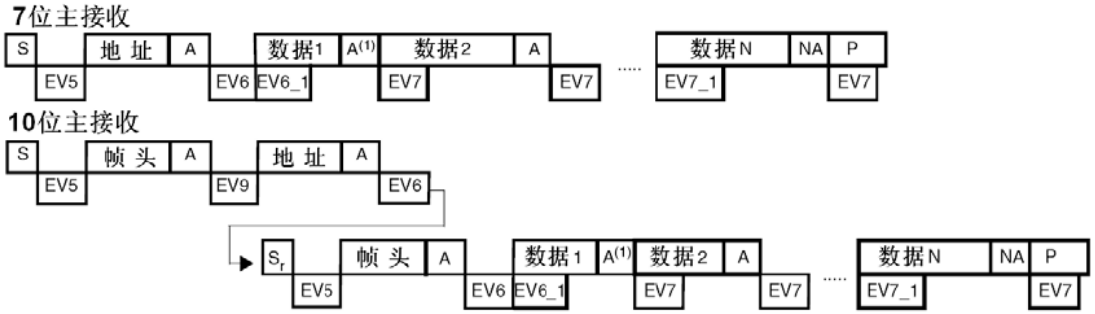
关闭通信

主设备在从从设备接收到最后一个字节后发送一个 NACK。接收到 NACK 后，从设备释放对 SCL 和 SDA 线的控制；主设备就可以发送一个停止/重起始条件。

- 为了在收到最后一个字节后产生一个 NACK 脉冲，在读倒数第二个数据字节之后(在倒数第二个 RxNE 事件之后)必须清除 ACK 位。
- 为了产生一个停止/重起始条件，软件必须在读倒数第二个数据字节之后(在倒数第二个 RxNE 事件之后)设置 STOP/START 位。
- 只接收一个字节时，刚好在 EV6 之后(EV6_1 时，清除 ADDR 之后)要关闭应答和停止条件的产生位。

在产生了停止条件后，I2C 接口自动回到从模式(M/SL 位被清除)。

主接收器传送序列图如下：



说明：S=Start(起始条件)，S_r=重复的起始条件，P=Stop(停止条件)，A=响应，NA=非响应，EVx=事件(ITEVFEN=1时产生中断)

EV5：SB=1，读SR1然后将地址写入DR寄存器将清除该事件。
EV6：ADDR=1，读SR1然后读SR2将清除该事件。在10位主接收模式下，该事件后应设置CR2的START=1。
EV6_1：没有对应的事件标志，只适于接收1个字节的情况。恰好在EV6之后(即清除了ADDR之后)，要清除响应和停止条件的产生位。
EV7：RxNE=1，读DR寄存器清除该事件。
EV7_1：RxNE=1，读DR寄存器清除该事件。设置ACK=0和STOP请求。
EV9：ADDR10=1，读SR1然后写入DR寄存器将清除该事件。

1. 如果收到一个单独的字节，则是 NA。
2. EV5、EV6 和 EV9 事件拉长 SCL 低电平，直到对应的软件序列结束。
3. EV7 的软件序列必须在当前字节传输结束前完成。
4. EV6_1 或 EV7_1 的软件序列 必须在当前传输字节的 ACK 脉冲之前完成。

12.2 I2C 应用实例 ----- 读写 EEPROM

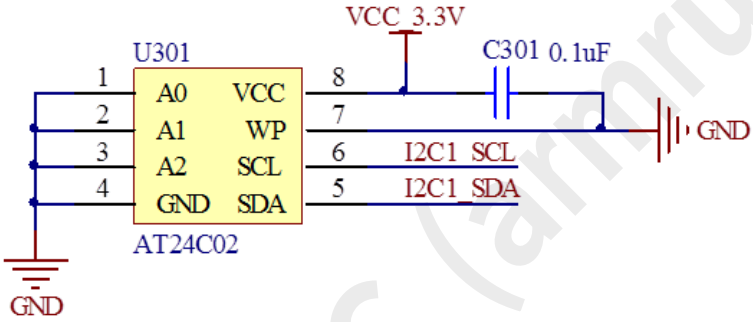
12.2.1 实例描述

本实例首先向 EEPROM 地址 0 写入一串字符串，然后从 EEPROM 地址 0 读出，读者可以对比该读出的字符串是否和写入的一致，来判定 EEPROM I2C 访问是否正确。I2C 使用 STM32 的 I2C 硬件，使用 I2C 中断模式。

12.2.2 硬件设计

该实例用到 PB6 和 PB7。GPIO 分配和硬件图如下：

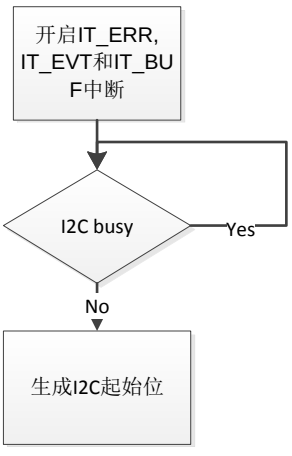
I2C1_SCL	PB6
I2C1_SDA	PB7



12.2.3 软件设计

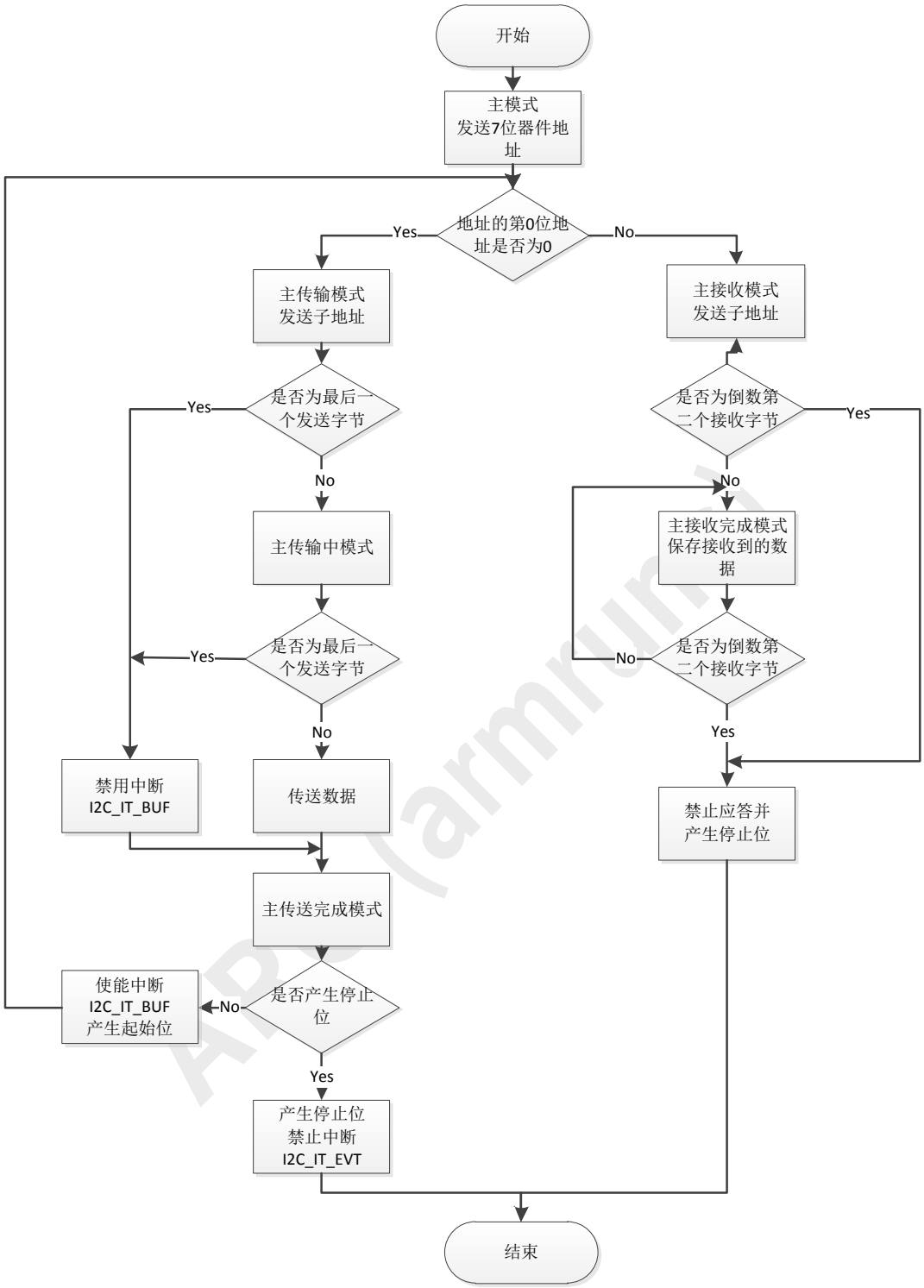
本实例首先初始化 SysTick 用于延迟，初始化串口用来调试输出。然后初始化 I2C，I2C 被配置为 7 位地址模式，时钟频率为 100kHz。然后使能 I2C 事件中断，注意，请将该中断函数的优先级设置为最高。

I2C 写入函数，该函数首先开启 IT_ERR, IT_EVT 和 IT_BUF 中断，然后生成一个 I2C 起始位。流程图如右图。



通过以上的配置，在起始位产生后会产生一个 EVT 中断，在此中断处理函数内，首先得到该中断函数的触发事件，根据不同的触发事件，进行相应的处理，产生起始位后，首先触发中断的事件是主模式。

主模式又分为主发送模式和主接收模式。他的处理流程图如下：



该实例的主要代码如下：

文件 I2C_main.c：

```
/**
  * @brief   Main program, write, read EEPROM example.
  * @param   None
  * @retval None
  */
```

```
int main(void)
{
    uint8_t TxBuffer[] = "ARC STM32, I2C example.";
    uint8_t RxBuffer[100] = "EEPROM not present\n";
    uint16_t Buffer_Size = sizeof(TxBuffer) / sizeof(*(TxBuffer));


    ARC_SysTick_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);
    ARC_I2C_Init();
    I2C_Cmd(I2C1, ENABLE);

    ARC_EEPROM_Write(TxBuffer, 0, Buffer_Size);

    ARC_EEPROM_Read(RxBuffer, 0, Buffer_Size);

    while(1)
    {
        printf("TX: %s\n", TxBuffer);
        printf("RX: %s\n", RxBuffer);
        ARC_SysTick_Delay(1000);
    }
}
```

文件 ARC_RCC.c

```
/**
  * @brief   Configures I2C clocks.
  * @param   None
  * @retval None
  */
void ARC_I2C_RCC_Init(void)
{
    /* Enable I2C and I2C_PORT & Alternate Function clocks */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB |
RCC_APB2Periph_AFIO, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
}
```

文件 ARC_GPIO.c

```
/**
  * @brief   Configures I2C GPIO ports.
  * @param   None
  * @retval None
  */
```

```
/*
            -------------
            | SCL | PB6 |
            -------------
            | SDA | PB7 |
            -------------
*/
void ARC_I2C_GPIO_Init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /* I2C SCL pin configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    /* I2C SDA pin configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

文件 ARC_I2C.c：

```
/**
  * @brief   get the I2C Device parameters.
  * @param   None
  * @retval the I2C device parameters
  */
I2C_param_struct __IO *ARC_get_I2C_param()
{
    return &I2C_Device;
}

/**
  * @brief   Configure the I2C parameters.
  * @param   None
  * @retval None
  */
void ARC_I2C_PARAM_Init()
{
    I2C_InitTypeDef I2C1_InitStructure;

    /* IOE_I2C configuration */
```

```
	I2C1_InitStructure.I2C_Mode = I2C_Mode_I2C;
	I2C1_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
	I2C1_InitStructure.I2C_OwnAddress1 = 0x00;
	I2C1_InitStructure.I2C_Ack = I2C_Ack_Enable;
	I2C1_InitStructure.I2C_AcknowledgedAddress =
I2C_AcknowledgedAddress_7bit;
	I2C1_InitStructure.I2C_ClockSpeed = 100000;

	I2C_Init(I2C1, &I2C1_InitStructure);
}

/**
  * @brief  Configure the I2C Peripheral.
  * @param  None
  * @retval None
  */
void ARC_I2C_Init()
{
	ARC_I2C_RCC_Init();
	ARC_I2C_GPIO_Init();
	ARC_I2C_PARAM_Init();
	ARC_I2C_NVIC_Init();
}

/**
  * @brief  @brief I2C write function.
  * @param  *I2Cx: I2C device ID.
  * @param  *I2C_param: I2C parameters.
  * @retval None
  */
void ARC_I2C_Read(I2C_TypeDef *I2Cx, I2C_param_struct __IO
*I2C_param)
{
	/* Enable I2C errors interrupts */
	I2Cx->CR2 |= I2C_IT_ERR;
	/* Enable EVT IT*/
	I2Cx->CR2 |= I2C_IT_EVT;
	/* Enable BUF IT */
	I2Cx->CR2 |= I2C_IT_BUF;

	I2C_GenerateSTART(I2Cx, ENABLE);

	/* Wait until BUSY flag is reset (until a STOP is generated) */
	while (I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));
```

```
    /* Enable Acknowledgement to be ready for another reception */
    I2C_AcknowledgeConfig(I2Cx, ENABLE);
}

/**
  * @brief   @brief I2C write function.
  * @param   *I2Cx: I2C device ID.
  * @param   *I2C_param: I2C parameters.
  * @retval None
  */
void ARC_I2C_Write(I2C_TypeDef *I2Cx, I2C_param_struct __IO
*I2C_param)
{
    /* Enable Error IT */
    I2C_ITConfig(I2Cx, I2C_IT_ERR, ENABLE);
    /* Enable EVT IT*/
    I2C_ITConfig(I2Cx, I2C_IT_EVT, ENABLE);
    /* Enable BUF IT */
    I2C_ITConfig(I2Cx, I2C_IT_BUF, ENABLE);

    /* Wait until BUSY flag is reset: a STOP has been generated on the bus
signaling the end
    of transmission */
    while (I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

    /* Send START condition */
    I2C_GenerateSTART(I2Cx, ENABLE);

    /* Wait until BUSY flag is reset: a STOP has been generated on the bus
signaling the end
    of transmission */
    while (I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));
    if(I2C_param->TX_Generate_stop == 0)
        I2C_AcknowledgeConfig(I2C1, ENABLE);
}
```

文件 ARC_NVIC_API.c：

```
/**
  * @brief   Initialize NVIC of I2C.
  * @param   None
  * @retval None
  */
void ARC_I2C_NVIC_Init(void)
{
```

```
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_InitStructure.NVIC_IRQChannel = I2C1_EV_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    NVIC_InitStructure.NVIC_IRQChannel = I2C1_ER_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

文件 ARC_EEPROM.c：

```
/**
  * @brief   EEPROM write api.
  * @param   *EEPBuff: the address of the buffer which will be written to
EEPROM
  * @param   SubAdd: the address of the EEPROM which stores the content
of the buffer
  * @param   EEPSize: the buffer size
  * @retval None
  */
void ARC_EEPROM_Write(uint8_t *EEPBuff, uint32_t SubAdd, uint32_t
EEPSize)
{
    uint8_t i = 0;
    for ( i = 0; i < EEPSize; i++)
    {
        I2C_param_struct __IO *pI2C_param;

        pI2C_param = ARC_get_I2C_param();

        pI2C_param->I2C_DIRECTION = ARC_I2C_DIRECTION_TX;
        pI2C_param->DeviceAddr = ARC_EEPROM_ADDR;

        pI2C_param->SubAddr = SubAdd + i;
        pI2C_param->TxData = EEPBuff + i;
        pI2C_param->TxNumOfBytes = 1;
        pI2C_param->TX_I2C_Index = 0;
        pI2C_param->TX_Generate_stop = 1;
```

```
            ARC_I2C_Write(I2C1, pI2C_param);
            ARC_SysTick_Delay(10);
        }
}

/**
   * @brief    EEPROM read api.
   * @param    *RXEEPBuff: the address of the buffer which stores EEPROM
read content.
   * @param    SubAdd: the address of the EEPROM read from
   * @param    EEPSize: the buffer size
   * @retval None
   */
void ARC_EEPROM_Read(uint8_t *RXEEPBuff, uint32_t SubAdd, uint32_t
EEPSize)
{
      I2C_param_struct __IO *pI2C_param;

      pI2C_param = ARC_get_I2C_param();

      pI2C_param->I2C_DIRECTION = ARC_I2C_DIRECTION_TX;
      pI2C_param->DeviceAddr = ARC_EEPROM_ADDR;

      pI2C_param->SubAddr = SubAdd;
      pI2C_param->TxNumOfBytes = 0;
      pI2C_param->TX_I2C_Index = 0;
      pI2C_param->TX_Generate_stop = 0;

      pI2C_param->RxData = RXEEPBuff;
      pI2C_param->RxNumOfBytes = EEPSize;
      pI2C_param->RX_I2C_Index = 0;

      ARC_I2C_Write(I2C1, pI2C_param);
}
```

文件 stm32f10x_it.c：

```
/**
   * @brief    This function handles I2C1 Event interrupt request, tx, rx
   *             buffer and number of bytes will be changed.
   * @param    None
   * @retval None
   */
void I2C1_EV_IRQHandler(void)
{
```

```
#ifdef ARC_I2C_IRQ
    uint32_t i2cEvent;
    I2C_param_struct __IO *pI2C_param;

    pI2C_param = ARC_get_I2C_param();
    i2cEvent = I2C_GetLastEvent(I2C1);

    switch (i2cEvent)
    {
        case I2C_EVENT_MASTER_MODE_SELECT:/* EV5 */
            if(pI2C_param->I2C_DIRECTION ==
ARC_I2C_DIRECTION_TX)
            {
                I2C_Send7bitAddress(I2C1, pI2C_param->DeviceAddr,
I2C_Direction_Transmitter);
            }
            else
            {
                I2C_Send7bitAddress(I2C1, pI2C_param->DeviceAddr,
I2C_Direction_Receiver);
            }
            break;

        /* Master Transmitter ----------------------------------------------------*/
        case I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED:
            I2C_SendData(I2C1, pI2C_param->SubAddr);
            if(pI2C_param->TxNumOfBytes == 0)
                I2C_ITConfig(I2C1, I2C_IT_BUF, DISABLE);
            break;

        case I2C_EVENT_MASTER_BYTE_TRANSMITTING:    /* Without
BTF, EV8 */
            if(pI2C_param->TX_I2C_Index < pI2C_param->TxNumOfBytes)
            {
                I2C_SendData(I2C1,
pI2C_param->TxData[pI2C_param->TX_I2C_Index++]);
            }
            else
            {
                I2C_ITConfig(I2C1, I2C_IT_BUF, DISABLE);
            }
            break;
```

```
            case I2C_EVENT_MASTER_BYTE_TRANSMITTED: /* With BTF
EV8-2 */
                if(pI2C_param->TX_Generate_stop == 1)
                {
                  I2C_GenerateSTOP(I2C1, ENABLE);
                  I2C_ITConfig(I2C1, I2C_IT_EVT, DISABLE);
                }
                else
                {
                  pI2C_param->I2C_DIRECTION = ARC_I2C_DIRECTION_RX;
                  I2C_ITConfig(I2C1, I2C_IT_BUF, ENABLE);
                  I2C_GenerateSTART(I2C1, ENABLE);
                }

            /* Master Receiver -----------------------------------------------------*/
            case I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED:
                  if(pI2C_param->RX_I2C_Index ==
(pI2C_param->RxNumOfBytes - 1))
                  {
                        I2C_AcknowledgeConfig(I2C1, DISABLE);
                        I2C_GenerateSTOP(I2C1, ENABLE);
                  }
                  break;

            case I2C_EVENT_MASTER_BYTE_RECEIVED:
                  pI2C_param->RxData[pI2C_param->RX_I2C_Index++] =
I2C_ReceiveData (I2C1);
                  if(pI2C_param->RX_I2C_Index ==
(pI2C_param->RxNumOfBytes - 1))
                  {
                        I2C_AcknowledgeConfig(I2C1, DISABLE);
                        I2C_GenerateSTOP(I2C1, ENABLE);
                  }
                  break;

            default:
                  break;
      }

#endif
}

/**
   * @brief    This function handles I2C1 Error interrupt request.
```

```
  * @param    None
  * @retval None
  */
void I2C1_ER_IRQHandler(void)
{
#ifdef ARC_I2C_IRQ
    uint32_t __IO SR1Register =0;

    /* Read the I2C1 status register */
    SR1Register = I2C1->SR1;
    /* If AF = 1 */
    if ((SR1Register & 0x0400) == 0x0400)
    {
        I2C1->SR1 &= 0xFBFF;
        SR1Register = 0;
    }
    /* If ARLO = 1 */
    if ((SR1Register & 0x0200) == 0x0200)
    {
        I2C1->SR1 &= 0xFBFF;
        SR1Register = 0;
    }
    /* If BERR = 1 */
    if ((SR1Register & 0x0100) == 0x0100)
    {
        I2C1->SR1 &= 0xFEFF;
        SR1Register = 0;
    }
    /* If OVR = 1 */
    if ((SR1Register & 0x0800) == 0x0800)
    {
        I2C1->SR1 &= 0xF7FF;
        SR1Register = 0;
    }
#endif
}
```