例程十五 AT24C02 基于 I2C 总线的读写

这个例程是讲解一下 I2C，相信大家以前学 51 的的时候想驱动 AT24C02 的时候就是用到模拟 I2C 总线吧。在 STM8S207RB 中，是以硬件来实现 I2C 总线的，如果大家说哪一种方式来驱动 AT24C02 的话，我只能说是各有千秋。用 IO 模拟的 I2C 总线的话就简单点，但速度不够快，如果是用片内的硬件 I2C 的话，程序编写有点复杂，但速度快很多。下面我主要是介绍 STM8S207RB 的硬件 I2C.

21.3 I²C简介

I²C模块不仅可以接收和发送数据，还可以在接收时将数据从串行转换成并行数据，在发送时将数据从并行转换成串行数据。可以开启或禁止中断。接口通过数据引脚(SDA)和时钟引脚(SCL)连接到I²C总线。允许连接到标准(最高100kHz)或快速(最高400kHz)的I²C总线。

模式选择

接口可以下述4种模式中的一种运行：

- 从设备发送模式
- 从设备接收模式
- 主设备发送模式
- 主设备接收模式

默认条件下，该模块工作于从模式。接口在产生起始条件后自动地从从模式切换到主模式；当仲裁失败或发送STOP信号时，则从主模式切换到从模式。允许多主机功能。

通信过程

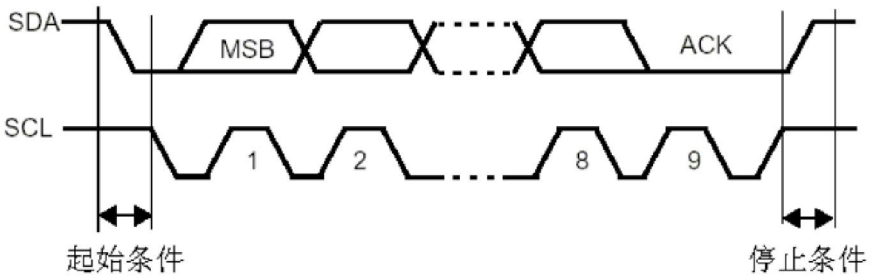
主模式时，I²C接口启动数据传输并产生时钟信号。串行数据传输总是以起始条件开始并以停止条件结束。起始条件和停止条件都是在主模式下由软件控制产生。

从模式时，I²C接口能识别它自己的地址(7位或10位)和广播呼叫地址。软件能够控制开启或禁止广播呼叫地址的识别。

数据和地址按8位/字节进行传输，高位在前。跟在起始条件后的1或2个字节是地址(7位模式为1个字节，10位模式为2个字节)。地址只在主模式发送。

在一个字节传输的8个时钟后的第9个时钟期间，接收器必须回送一个应答位(ACK)给发送器。参考下图。

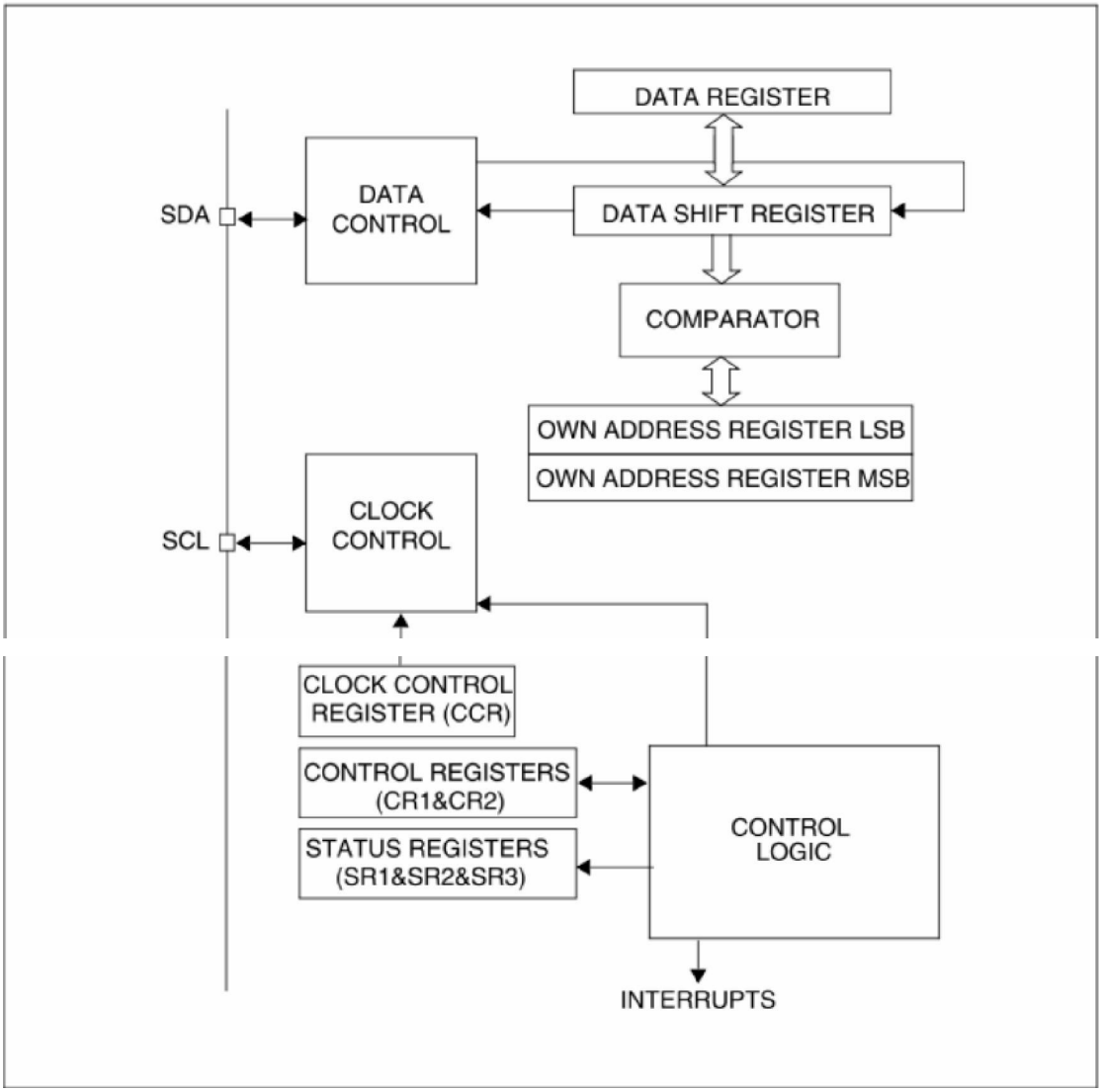
图92 I²C总线协议



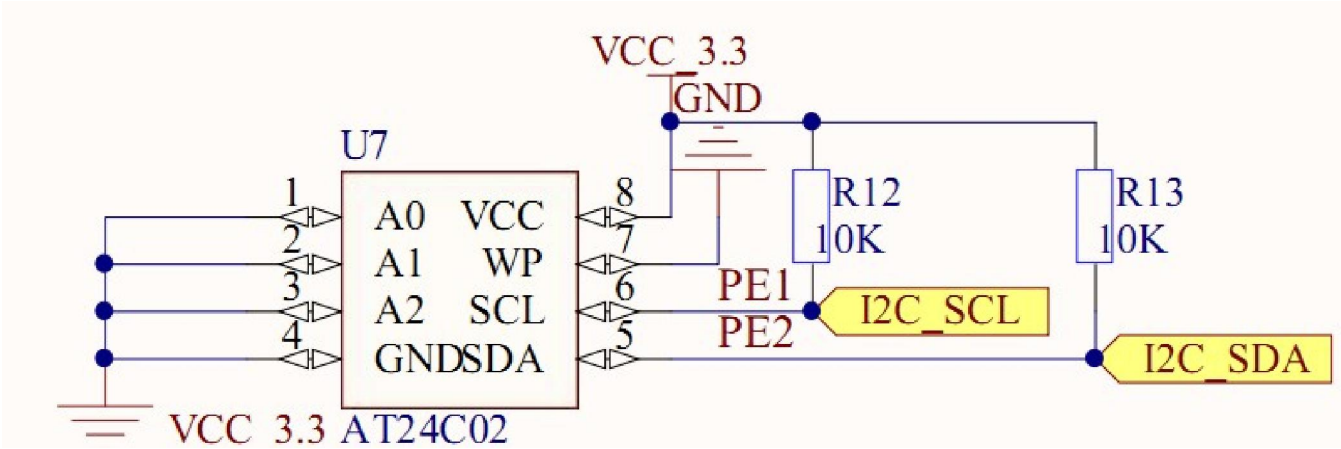
软件可以开启或禁止应答(ACK)，并可以设置I²C接口的地址(7位、10位地址或广播呼叫地址)。

I²C接口的功能框图示于*图93*。

图93 I²C的功能框图



大家可以参考 STM8 寄存器.pdf 文档中的第 246 页至 248 页。
大家在看看我们风驰电子 STM8 开发板的硬件连接



下面我们从主函数看起

```
int main(void)
{
    /* Infinite loop */
  /*设置内部时钟16M为主时钟*/
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
  /*!<Set High speed internal clock  */
    I2C_DeInit();
    Uart_Init();
    /* Initialize the I2C */
    I2C_EEInit();
    I2C_EE_PageWrite(Tx1_Buffer, BASE_ADDRESS, BufferSize1 );
    Delay(2000); /* To let eeprom the time to finish the write operation */
    I2C_EE_BufferRead(Rx1_Buffer, BASE_ADDRESS, BufferSize1 );
    UART1_SendString("What is writed into a page of AT24C02 EEPROM:",\
                     sizeof("What is writed into a page of AT24C02 EEPROM:"));
    Delay(0xfff);
    UART1_SendString(Tx1_Buffer,BufferSize1);
    Delay(0xffff);
    UART1_SendString("What is read from a page of AT24C02 EEPROM:",\
           sizeof("What is read from a page of AT24C02 EEPROM:"));
    Delay(0xffff);
    UART1_SendString(Rx1_Buffer,BufferSize1);
    Delay(0xffff);

   while (1)
   {

      UART1_SendString("What is read from a page of AT24C02 EEPROM:",\
             sizeof("What is read from a page of AT24C02 EEPROM:"));
      Delay(0xffff);
      UART1_SendString(Rx1_Buffer,BufferSize1);
      Delay(0xffff);

   }
}
```

其他的初始化就不多说了，现在前面的例程都说得很清楚了，先主要是说下与 I2C 相关的初始化和读写

I2C_DeInit()　I2C 的复位，其实设和没设都一样，因为单片机复位后它也跟着复位了。

I2C_EEInit()　I2C 的初始化

函数原型：

```
void I2C_EEInit(void)
{
   u8 Input_Clock = 0x0;
  /* Get system clock frequency */
  Input_Clock = CLK_GetClockFreq()/1000000;
  /* I2C Peripheral Enable */
  I2C_Cmd(ENABLE);
  /* Apply I2C configuration after enabling it */
  I2C_Init(I2C_Speed, I2C1_SLAVE_ADDRESS7, I2C_DUTYCYCLE_2,\
           I2C_ACK_CURR, I2C_ADDMODE_7BIT, Input_Clock);
}
```

I2C_Init(I2C_Speed, I2C1_SLAVE_ADDRESS7, I2C_DUTYCYCLE_2,\
I2C_ACK_CURR, I2C_ADDMODE_7BIT, Input_Clock)重点看下这个函数，这个就是 I2C 总线的初始化。设置 I2C 的速度为标准的速度 100K，从地址为 0xA0，快速模式 Fast mode Tlow/THigh = 2 ，当前数据位应答，7 位从地址，时钟为系统时钟。

函数原型：void I2C_Init(u32 OutputClockFrequencyHz, u16 OwnAddress, I2C_DutyCycle_TypeDef DutyCycle, I2C_Ack_TypeDef Ack, I2C_AddMode_TypeDef AddMode, u8 InputClockFrequencyMHz)

```
/**
  * @brief Initializes the I2C according to the specified parameters in standard or fast mode
  * @param[in] OutputClockFrequencyHz : Specifies the output clock frequency in Hz.
  * @param[in] OwnAddress : Specifies the own address.
  * @param[in] DutyCycle : Specifies the duty cycle to apply.
  * This parameter can be any of the  @ref I2C_DutyCycle_TypeDef enumeration.
  * @param[in] Ack : Specifies the acknowledge mode to apply.
  * This parameter can be any of the  @ref I2C_Ack_TypeDef enumeration.
  * @param[in] AddMode : Specifies the addressing mode to apply.
  * This parameter can be any of the  @ref I2C_AddMode_TypeDef enumeration.
  * @param[in] InputClockFrequencyMHz : Specifies the input clock frequency in MHz.
  * @retval None
  */
```

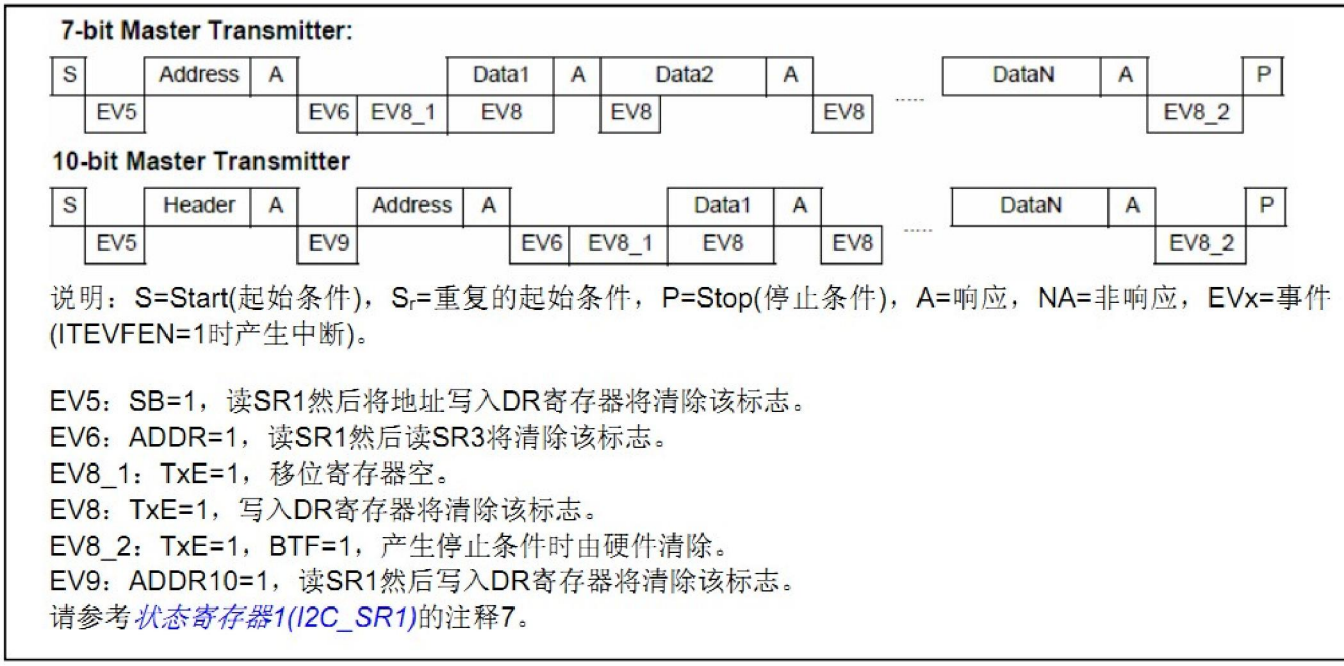
代码太长了，看工程代码就可以了。

初始化弄好了，那怎样往从地址写一个字节呢？这个很重要
我们的选择模式是主设备方式读写

```
/*******************************************************************************
* Function Name  : I2C_EE_ByteWrite
* Description    : Writes one byte to the I2C EEPROM.
* Input          : - pBuffer : pointer to the buffer  containing the data to be
*                    written to the EEPROM.
*                  - WriteAddr : EEPROM's internal address to write to.
* Output         : None
* Return         : None
*******************************************************************************/
void I2C_EE_ByteWrite(u8* pBuffer, u16 WriteAddr)
{
  /* Send STRAT condition */
  I2C_GenerateSTART(ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_START_SENT));

  /* Send EEPROM address for write */
  I2C_Send7bitAddress(EEPROM_ADDRESS, I2C_DIRECTION_TX);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_ADDRESS_ACKED));

  /* Send Address (on 2 bytes) of first byte to be written & wait event detection *
  I2C_SendData((u8)(WriteAddr >> 8)); /* MSB */
  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  I2C_SendData((u8)(WriteAddr)); /* LSB */
  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));

  /* Send the byte to be written */
  I2C_SendData(*pBuffer);

  /* Test on EV8 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  /* Send STOP condition */
  I2C_GenerateSTOP(ENABLE);
}
```

这个函数主要是根据 I2C 的时序来操作的，下面介绍一下 I2C 的时序

图96　主设备发送模式发送序列图



来源于 STM8 寄存器.pdf 中第 251 页

```
/*******************************************************************************
* Function Name  : I2C_EE_PageWrite
* Description    : Writes more than one byte to the EEPROM with a single WRITE
*                  cycle. The number of byte can't exceed the EEPROM page size.
* Input          : - pBuffer : pointer to the buffer containing the data to be
*                    written to the EEPROM.
*                  - WriteAddr : EEPROM's internal address to write to.
*                  - NumByteToWrite : number of bytes to write to the EEPROM.
* Output         : None
* Return         : None
*******************************************************************************/
void I2C_EE_PageWrite(u8* pBuffer, u16 WriteAddr, u8 NumByteToWrite)
{
  /* While the bus is busy */
  while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

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

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2C_EVENT_MASTER_START_SENT));

  /* Send EEPROM address for write */
  I2C_Send7bitAddress(EEPROM_ADDRESS, I2C_DIRECTION_TX);
```

```
/* Test on EV6 and clear it */
while(!I2C_CheckEvent(I2C_EVENT_MASTER_ADDRESS_ACKED));
I2C_ClearFlag(I2C_FLAG_ADDRESSSENTMATCHED);

/* Send Address (on 2 bytes) of first byte to be written & wait event detection *
I2C_SendData((u8)(WriteAddr >> 8)); /* MSB */
/* Test on EV8 and clear it */
while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));
I2C_SendData((u8)(WriteAddr)); /* LSB */
/* Test on EV8 and clear it */
while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));


/* While there is data to be written */
while(NumByteToWrite--)
{
  /* Send the current byte */
  I2C_SendData(*pBuffer);

  /* Point to the next byte to be written */
  pBuffer++;

  /* Test on EV8 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));
}

 /* Send STOP condition */
 I2C_GenerateSTOP(ENABLE);
}
```

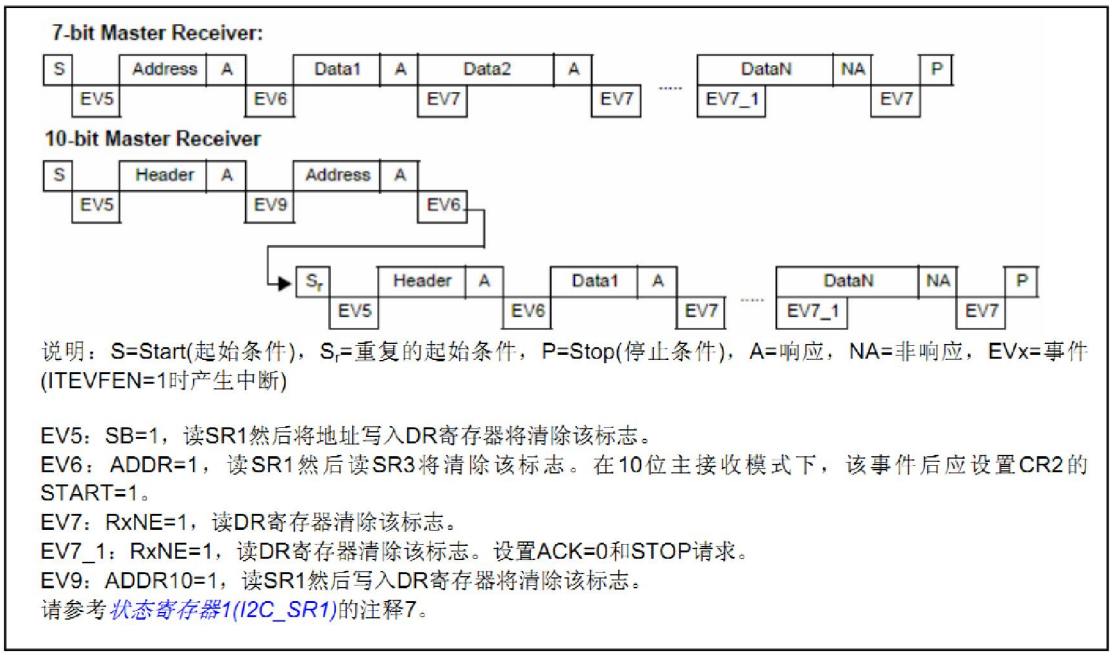
这个函数是往 AT24C02 写一页数据，每页最多能写 8Byte。

```
/*******************************************************************************
* Function Name  : I2C_EE_BufferRead
* Description    : Reads a block of data from the EEPROM.
* Input          : - pBuffer : pointer to the buffer that receives the data read
*                    from the EEPROM.
*                  - ReadAddr : EEPROM's internal address to read from.
*                  - NumByteToRead : number of bytes to read from the EEPROM.
* Output         : None
* Return         : None
*******************************************************************************/
void I2C_EE_BufferRead(u8* pBuffer, u16 ReadAddr, u8 NumByteToRead)
{
    /* While the bus is busy */
  while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

  /* Generate start & wait event detection */
    I2C_GenerateSTART(ENABLE);
  /* Test on EV5 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_START_SENT));

  /* Send slave Address in write direction & wait detection event */
    I2C_Send7bitAddress(EEPROM_ADDRESS, I2C_DIRECTION_TX);
   /* Test on EV6 and clear it */
    while (!I2C_CheckEvent(I2C_EVENT_MASTER_ADDRESS_ACKED));
    I2C_ClearFlag(I2C_FLAG_ADDRESSSENTMATCHED);

  /* Send Address of first byte to be read & wait event detection */
   I2C_SendData((u8)(ReadAddr >> 8)); /* MSB */
   /* Test on EV8 and clear it */
   while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));
   I2C_SendData((u8)(ReadAddr)); /* LSB */
/* Test on EV8 and clear it */
   while (!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

/* Send STRAT condition a second time */
I2C_GenerateSTART(ENABLE);
   /* Test on EV5 and clear it */
  while (!I2C_CheckEvent(I2C_EVENT_MASTER_START_SENT));

/* Send slave Address in read direction & wait event */
   I2C_Send7bitAddress(EEPROM_ADDRESS, I2C_DIRECTION_RX);
  /* Test on EV6 and clear it */
   while (!I2C_CheckEvent(I2C_EVENT_MASTER_ADDRESS_ACKED));
   I2C_ClearFlag(I2C_FLAG_ADDRESSSENTMATCHED);

/* While there is data to be read */
while(NumByteToRead)
{
   if(NumByteToRead == 1)
   {
     /* Disable Acknowledgement */
     I2C_AcknowledgeConfig(I2C_ACK_NONE);
```

```
      /* Disable Acknowledgement */
      I2C_AcknowledgeConfig(I2C_ACK_NONE);

      /* Send STOP Condition */
      I2C_GenerateSTOP(ENABLE);
    }

    /* Test on EV7 and clear it */
    if(I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_RECEIVED))
    {
      /* Read a byte from the EEPROM */
      *pBuffer = I2C_ReceiveData();

      /* Point to the next location where the byte read will be saved */
      pBuffer++;

      /* Decrement the read bytes counter */
      NumByteToRead--;
    }
  }

  /* Enable Acknowledgement to be ready for another reception */
  I2C_AcknowledgeConfig(I2C_ACK_CURR);
}
```

这个函数是从 AT24C02 任何地址读多个字节的（可以超过 8 个字节）回来放在内存 pBuffer 中。

图97　主设备接收模式接收序列图



参考 STM8 寄存器.pdf 文档第 252 页。

实验效果：

风驰电子祝您学习愉快～～～!!!!!