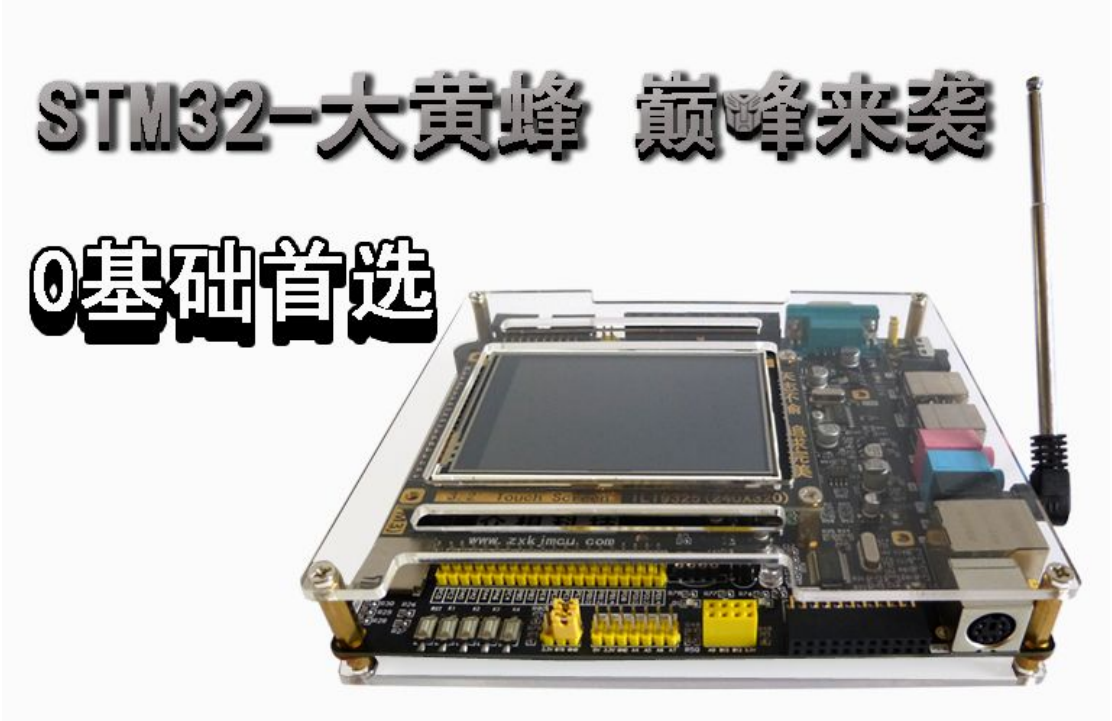
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.22 STM32 CAT24WCxx 存储器工作原理

4.22.1 概述

24C01/24C02 是一个 1K/2K/4K/8K/16K 位串行 CMOS E2PROM, 内部含有 128/256/512/1024/2048 个 8 位字节,CATALYST 公司的先进 CMOS 技术实质上减少了器件的功耗.CAT24WC01 有一个 8 字节页写缓冲器,24C01/24C02 有一个 16 字节页写缓冲器.该器件通过 I2C 总线接口进行操作有一个专门的写保护功能。

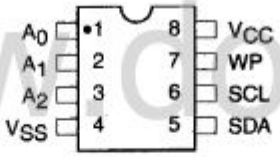
4.22.1.1 AT24C02WCxx 特点

- 1.8 到 6.0 伏工作电压范围；
- 与 400KHz I^2C 总线兼容；
- 低功耗 CMOS 技术
- 写保护功能：当 WP 为高电平时进入写保护状态；
- 页写缓冲器
- 自定时擦写周期
- 100 万次编程/擦除周期
- 可保存数据 100 年；
- 8 脚 DIP、SOIC 或者 TSSOP 封装；

具体详细的使用参数和时序请参考官方技术手册。

4.22.1.2 AT24C02 管脚描述

管脚配置



管脚描述

管脚名称	功能
A0、A1、A2	器件地址选择
SDA	串行数据/地址
SCL	串行时钟
WP	写保护
Vcc	+1.8V~6.0V 工作电压
Vss	地

4.22.1.3 AT24C02 时序说明

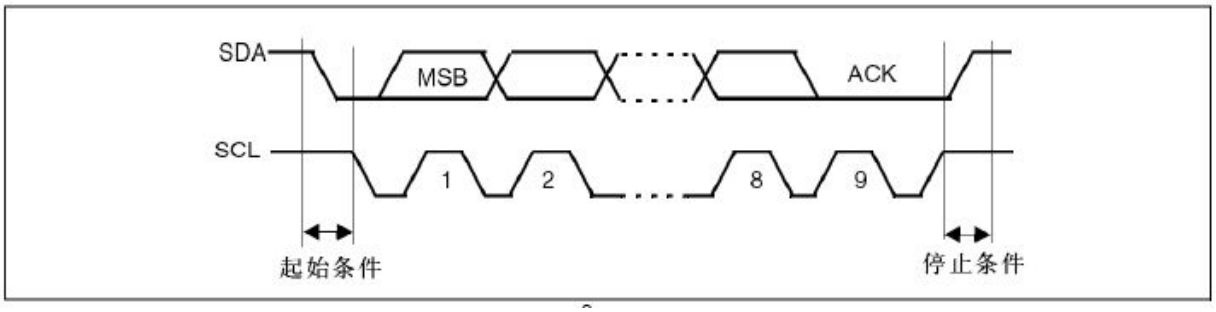


图 4.22.1.3 AT24C02 时序

具体详细的使用参数和时序请参考官方技术手册。

4.22.2 实验目的

使用 STM32 芯片模拟 I^2C 总线通讯方式读写 AT24C02WCxx 存储器。

4.22.3 硬件设计

利用实验板上的 AT24C02 电路，通过程序设计可以很方便的实现 I^2C 存

储功能。硬件设计见图 4.22.1 AT24C02 原理图。从硬件设计图上可以看 A0、A1、A2 接地，从而知道 24C02 片选地址为“0”。

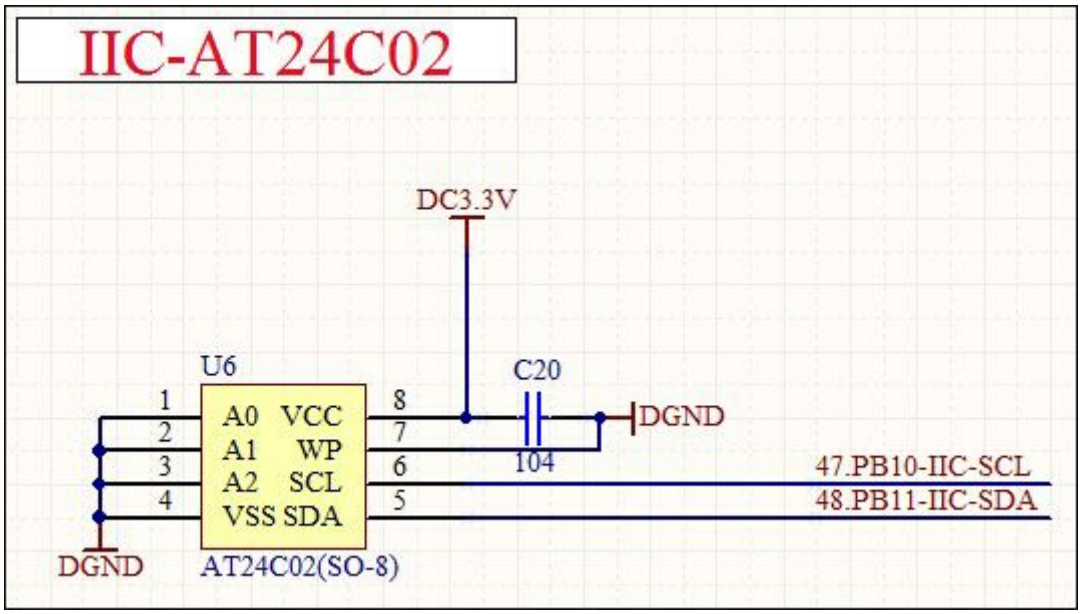


图 4.22.1 AT24C02 原理图

4.22.4 软件设计

4.22.4.1 软件设计说明

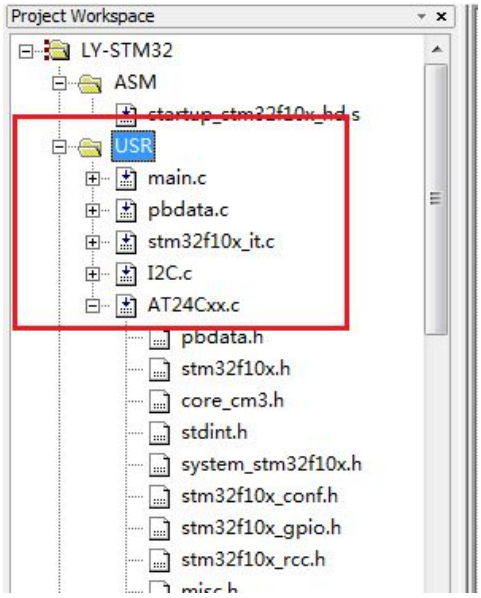
这次试验是先一组数据通过 I^2C 通讯写入到 24C02 存储器中，然后在读出来验证是否写入正确。并打印输出，送到显示器显示出来。设计思路是这样的：

1、准备一组数据通过 I^2C 通讯写入 24C04 中，然后在通过 I^2C 通讯读出来；

2、打印输出到显示器；

3、和原写入的数据进行比较；

在“USR”文件夹中引入我们上节课写的“I2C.c”，然后我们还要新建两个头文件，命名为“AT24Cxx.c”和“AT24Cxx.h”，并且引入到“USR”文件夹中，如下图所示。



4.22.4.2 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
```

本节实验及以后的实验我们都是用到库文件，其中 stm32f10x_gpio.h 头文件包含了 GPIO 端口的定义。stm32f10x_rcc.h 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 stm32f10x_gpio.c 和 stm32f10x_rcc.c 加到工程中；Misc.c 库函数主要包含了中断优先级的设置，stm32f10x_exti.c 库函数主要包含了外部中断设置参数，tm32f10x_tim.c 库函数主要包含定时器设置，tm32f10x_usart.c 库函数主要包含串行通讯设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.22.4.3 自定义头文件

```
pbdata.h
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.22.4.4 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "sttdio.h"
#include "I2C.h"
#include "AT24Cxx.h"

extern u8 dt;//定义变量

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。

4.22.4.5 pbdata.c 文件里的内容是

pbdata.c 文件的内容和上节讲的一样，在这里就不详细列出。在文件开始还是引用“pbdata.h”文件，书写格式详细参考《4.21 STM32 I2C 工作原理》。

4.22.5 STM32 系统用户文件 I2C.c 文件里的引用

在这个工程中我们要引用“I2C.h”文件。他们是模拟 I2C 总线的基础文件，

所以需要引入进来。具体内容和书写格式详细参考《4.21 STM32 I2C 工作原理》。

4.22.6 STM32 系统时钟配置 SystemInit()

我们总在强调，每个工程都必须在开始时配置并启动 STM32 系统时钟，这次也不例外。

4.22.7 GPIO 引脚时钟使能

```
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟使
能
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
```

本节实验用到了 PB 端口，所以要把 PB 端口的时钟打开；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.22.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待 I2C 总线通讯中断的到来。

```
    while(1);
```

4.22.9 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中仅串口 1 子函数非空，进入中断处理函数后，只有串口 1 有参数输出。

```
    #include "stm32f10x_it.h"
    #include "stm32f10x_exti.h"
    #include "stm32f10x_rcc.h"
    #include "misc.h"
    #include "pbdata.h"

    void NMI_Handler(void)
    {
    }
```

```
void USART1_IRQHandler(void)
{
   if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
   {
      USART_SendData(USART1,USART_ReceiveData(USART1));
      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
   }
}
```

4.22.10 AT24Cxx.h 文件里的内容是

自定义函数 AT24Cxx.h 在这里是核心内容，在 AT24Cxx.h 的内容如下。

```
#ifndef _AT24Cxx_H
#define _AT24Cxx_H   //定义头文件，防止两次或两次以上调用同一个头文件产生错误提示
#include "pbdata.h"  //引用头文件

//把所有大小的 AT24Cxx 存储器都定义出来，方便以后大家移植
#define AT24C01  127
#define AT24C02  255 //定义 AT24C02 范围
#define AT24C04  511
#define AT24C08  1023
#define AT24C16  2047
#define AT24C32  4095
#define AT24C64  8191
#define AT24C128 16383
#define AT24C256 32767

#define EE_TYPE  AT24C02   //定义类型，用 EE_TYPE 替代 AT24C02，这样做是为了更方便的修改和移植程序，也可以不这样写，在后文中直接使用 AT24Cxx。
u8 AT24Cxx_ReadOneByte(u16 addr);
u16 AT24Cxx_ReadTwoByte(u16 addr);
void AT24Cxx_WriteOneByte(u16 addr,u8 dt);
void AT24Cxx_WriteTwoByte(u16 addr,u16 dt);

#endif
```

4.22.11 AT24Cxx.c 文件里的内容是

自定义函数 AT24Cxx.c 在这里是核心内容，AT24Cxx.c 的内容如下。

```
#include "pbdata.h"
```

```
u8 AT24Cxx_ReadOneByte(u16 addr) //读一个字节子函数
{
   u8 temp=0;     //定义一个 8 位缓存，为以后存储数据使用

   I2C_Start();  //调用起始信号子函数

   if(EE_TYPE>AT24C16) //判断是单字节还是双字节寻址？
   {
      I2C_Send_Byte(0xA0);  //双字节寻址，根据硬件设计，外扩存储器地址是
“0”
      I2C_Wait_Ack();
      I2C_Send_Byte(addr>>8);  //发送数据地址高位
   }
   else
   {
      I2C_Send_Byte(0xA0+((addr/256)<<1));//器件地址+数据地址，这样写是为
了兼容稍大的存储器寻址
   }

   I2C_Wait_Ack();  //延时
   I2C_Send_Byte(addr%256);//双字节是数据地址低位
                           //单字节是数据地址低位
   I2C_Wait_Ack();

   I2C_Start();
   I2C_Send_Byte(0xA1);//发送读命令
   I2C_Wait_Ack();

   temp=I2C_Read_Byte(0); //  0 代表 NACK，读回来的数据存入 temp 中
   I2C_Stop();   //停止信号
   return temp;  //返回读入的数据
}

u16 AT24Cxx_ReadTwoByte(u16 addr)//读二个字节子函数
{
   u16 temp=0;

   I2C_Start();

   if(EE_TYPE>AT24C16)
   {
      I2C_Send_Byte(0xA0);
      I2C_Wait_Ack();
      I2C_Send_Byte(addr>>8);  //发送数据地址高位
```

```
    }
    else
    {
       I2C_Send_Byte(0xA0+((addr/256)<<1));//器件地址+数据地址
    }

    I2C_Wait_Ack();
    I2C_Send_Byte(addr%256);//双字节是数据地址低位
                            //单字节是数据地址低位
    I2C_Wait_Ack();

    I2C_Start();
    I2C_Send_Byte(0xA1);
    I2C_Wait_Ack();

    temp=I2C_Read_Byte(1); //  1   代表 ACK
    temp<<=8;//左移动 8 位，存数据高字节
    temp|=I2C_Read_Byte(0); //  0  代表 NACK   和高字节相或，就组成一字的
数据了，读一个字完成

    I2C_Stop();//停止位

    return temp;  //返回读出来的数据
}

void AT24Cxx_WriteOneByte(u16 addr,u8 dt)//写一个字节子函数
{
    I2C_Start();  //发送起始条件

    if(EE_TYPE>AT24C16)  //发送器件地址
    {
        I2C_Send_Byte(0xA0);
        I2C_Wait_Ack();
        I2C_Send_Byte(addr>>8);  //右移动 8 位，发送数据地址高位
    }
    else
    {
       I2C_Send_Byte(0xA0+((addr/256)<<1));//器件地址+数据地址
    }

    I2C_Wait_Ack();
    I2C_Send_Byte(addr%256);//双字节是数据地址低位
                            //单字节是数据地址低位
    I2C_Wait_Ack();
```

```
    I2C_Send_Byte(dt);  //发送数据
    I2C_Wait_Ack();
    I2C_Stop();       //结束

    delay_ms(10);     //很重要，必须延时，根据时序得来
}

void AT24Cxx_WriteTwoByte(u16 addr,u16 dt)//写二个字节子函数
{
    I2C_Start();

    if(EE_TYPE>AT24C16)
    {
        I2C_Send_Byte(0xA0);
        I2C_Wait_Ack();
        I2C_Send_Byte(addr>>8);  //发送数据地址高位
    }
    else
    {
       I2C_Send_Byte(0xA0+((addr/256)<<1));//器件地址+数据地址
    }

    I2C_Wait_Ack();
    I2C_Send_Byte(addr%256);//双字节是数据地址低位
                            //单字节是数据地址低位
    I2C_Wait_Ack();

    I2C_Send_Byte(dt>>8);//右移动 8 位，发送高位字节
    I2C_Wait_Ack();

    I2C_Send_Byte(dt&0xFF);//把高位与掉，然后发送
    I2C_Wait_Ack();
    I2C_Stop();
    delay_ms(10);
}
```

4.22.12 main.c 文件里的内容是

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
```

```
void NVIC_Configuration(void);
void USART_Configuration(void);

int fputc(int ch,FILE *f)
{
   USART_SendData(USART1,(u8)ch);
   while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
   return ch;
}

int main(void)
{
   u16 dt=0;

   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();
   I2C_Init();

   delay_ms(1);

   //读写一个字节
   //AT24Cxx_WriteOneByte(0,0x33);
   //dt=AT24Cxx_ReadOneByte(0);
   //printf("字符串输出 dt=%X\r\n",dt);//打印输出

   //读写二个字节
   AT24Cxx_WriteTwoByte(0,0x1234);
   dt=AT24Cxx_ReadTwoByte(0);
   printf("字符串输出 dt=%X\r\n",dt);//打印输出

   while(1);
}

void RCC_Configuration(void)
{
    SystemInit();//72m
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
}
```

```
    void GPIO_Configuration(void)
    {
        GPIO_InitTypeDef GPIO_InitStructure;
       //LED
       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
       GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
       GPIO_Init(GPIOA,&GPIO_InitStructure);

       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
       GPIO_Init(GPIOA,&GPIO_InitStructure);
    }

    void NVIC_Configuration(void)
    {
          NVIC_InitTypeDef NVIC_InitStructure;

       NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

       NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
       NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
       NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
       NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
       NVIC_Init(&NVIC_InitStructure);
    }

    void USART_Configuration(void)
    {
        USART_InitTypeDef  USART_InitStructure;

       USART_InitStructure.USART_BaudRate=9600;
       USART_InitStructure.USART_WordLength=USART_WordLength_8b;
       USART_InitStructure.USART_StopBits=USART_StopBits_1;
       USART_InitStructure.USART_Parity=USART_Parity_No;
       USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowContr
ol_None;
       USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

       USART_Init(USART1,&USART_InitStructure);
       USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
       USART_Cmd(USART1,ENABLE);
       USART_ClearFlag(USART1,USART_FLAG_TC);
    }
```

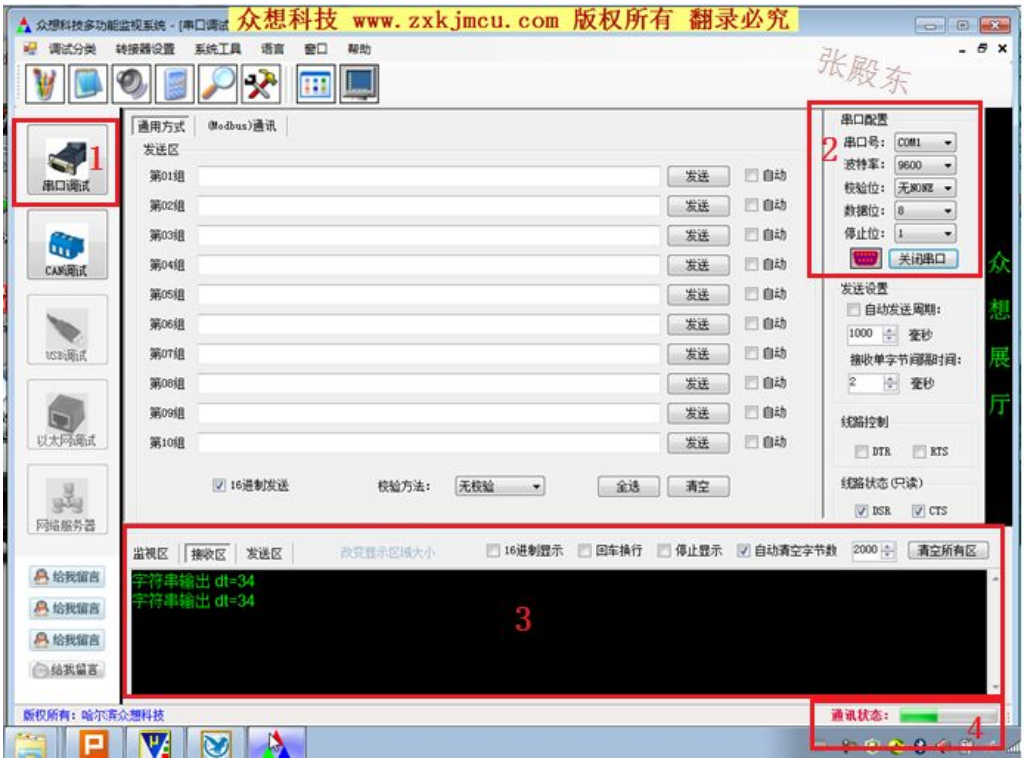
4.22.13 程序下载

在这一章节中要掌握 AT24C02 存储芯片 I^2C 总线时序，大家做到熟悉掌握程序，还要熟悉掌握外扩存储器地址的寻址程序的编写方法。

本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\2,外设篇\外设篇 04. STM32 I2C 读写 AT24Cxx 存储器(模拟方式)\程序）

4.22.14 实验效果图

使用众想科技多功能监控软件，可以观察到从 AT24C02 中读出来的数据，和我们设计的一样。



实验效果图