**实验标题：GPIO模拟I2C总线通信协议**

作者：叶青华

1. **实验要求**

编写软件，通过使用GPIO来模拟I2C总线通信协议，实现一主一从间的I2C通信。

实验硬件实现示意图如下：

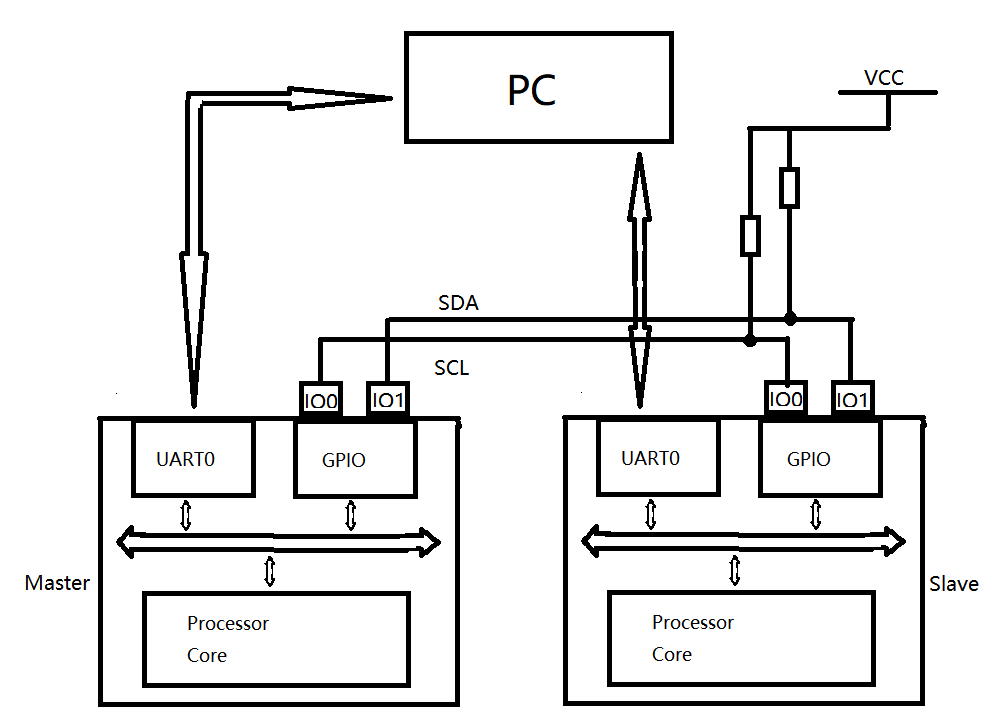


图1 硬件实现示意图

1. **实验原理**

基本思想：本次实验实现主机和从机之间相互的字节收发。

相应步骤：

① 主机、从机进行GPIO设置等初始化工作。最后使SDA、SCL为高电平状态。

② 在SCL为高电平期间，主机将SDA由高电平拉低至低电平，即发出START信号。从机准备接收接下来从机要发送的地址等数据。

③ 主机控制SCL。从机地址设置为7位时，在接下来的8个由主机释放和拉低SCL所形成的时钟脉冲内（10位地址模式下，则为11个时钟脉冲），主机控制SDA，由MSB到LSB，发送出目标从机的地址以及最后的读/写控制位，接着释放SDA。读/写控制位为高电平，表示主机将从从机中读取数据；为低电平，则表示主机将向从机写数据。从机在读取地址以及的读/写控制位后，将其与自身设定的地址进行比对，若地址匹配，则在接下来的一个时钟脉冲来临前，将SDA拉低，持续整个时钟脉冲高电平的时间，以向主机发送应答信号ACK；若地址不匹配，则将SDA维持为高电平，作为一个非应答信号NACK。

④ 当主机从从机读取数据时，接下来的8个SCL时钟脉冲内，从机通过拉低或释放SDA来向主机发送一个字节的数据，在再接下来的一个时钟脉冲内，主机需要将SDA拉低，向从机发送应答信号ACK；当主机向从机写入数据时，接下来的8个SCL时钟脉冲内，主机通过拉低或释放SDA来向从机发送一个字节的数据，在再接下来的一个时钟脉冲内，从机需要将SDA拉低，向主机发送应答信号ACK。

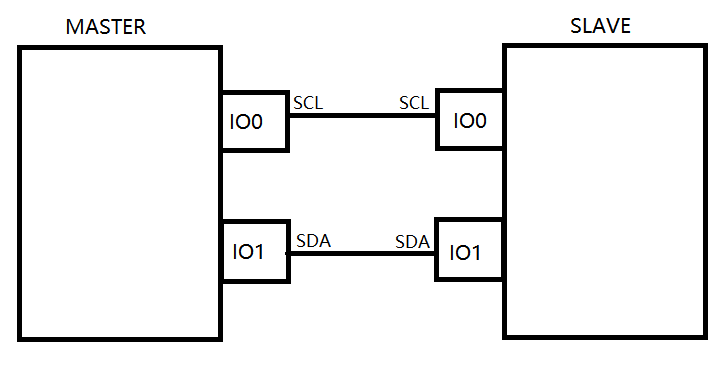
⑤ 若需要改变数据传输方向（比如由主机从从机读取改为主机向从机写入），需要回到步骤①。若以当前传输方向继续传输数据，则重复步骤④。若需停止传输，主机需在接下来的SCL时钟脉冲高电平期间，将SDA由低拉至高电平，以向从机发送STOP信号。

⑥ SCL时钟周期和占空比可通过使用延时函数来控制。设定每个时钟周期有4次延时，

高电平时两个，低电平时两个，占空比为50%。

1. GPIO模块介绍

GPIO（General-purpose input/output），即通用型之输入输出，可供使用者由程控自由使用，PIN脚可根据现实考量作输入、输出等用途。



SO

SI

SO

SI

**CS**

**CS**

图2 使用GPIO模拟的I2C结构示意图

连线介绍：

1. SDA – 串行数据线
2. SCL –  串行时钟线
3. IO0 – 第0个GPIO引脚
4. IO1 – 第1个GPIO引脚

本次试验中模块的使用：使用了GPIO0、GPIO1的通用输入输出功能。

具体硬件连接方式如下图所示：

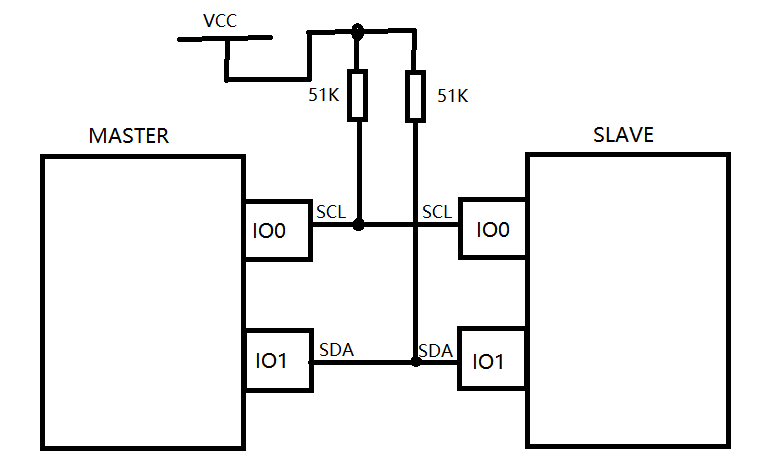


图3 本实验结构图

1. UART0模块介绍

UART0串口用于MCU和PC之间的通讯，可以将寄存器内的数据在电脑上显示出来。UART0可以实现异步串行收发数据的功能，它的异步传输以byte为单位，它发送byte里面的每一个bit，按顺序串行发送，在接收端另一个UART单元将接收到的每一个bit再重新组装回原来的byte，且必须以双方设定的统一波特率进行逐位串行传输。

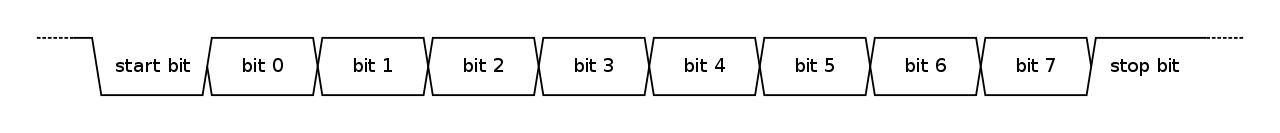


图4 异步串行通讯传输格式

1. **分析思路**
2. GPIO模块

主机的GPIO0（即SCL）通过设置的[SYS\_ IOCTL\_REG]寄存器的第0位为1来设为输出模式，从机的则设为0为设为输入模式。GPIO1（即SDA）的输入、输出模式需要灵活改变，可通过设置[SYS\_ IOCTL\_REG]寄存器的第1位为来实现；同理，置0为输入模式，置1为输出模式。输出模式时，通过改变[SYS\_ GPIO0\_REG]的第0或1位的值来改变对应IO引脚的电平；输入模式时，通过读取[SYS\_ GPIO1\_REG]的第0或1位的值来得到对应IO引脚的电平状态。

1. UART0模块

发送数据时，首先判断[UART\_BUSY\_REG]的值，若为1，则不可执行数据发送操作；若为0，则可执行发送数据操作。再将需要发送的数据写入[UART\_WRITE\_REG]。

本实验中，实际使用str.h库中的puts和putch函数。

1. **相关寄存器**
2. GPIO模块
3. SYS\_ IOCTL\_REG(0x1f80-0704): 设置GPIO输入或输出模式的寄存器
4. SYS\_ GPIO0\_REG (0x1f80-0705): 设置GPIO输出电平的寄存器
5. SYS\_ GPIO1\_REG (0x1f80-0706): 读取GPIO输入电平的寄存器
6. UART0模块
7. U1\_CTL0\_REG(0x1f80-0004): 本实验中只用到第4位使能位，置1使能
8. U1\_BUSY\_REG(0x1f80-0001): 判断RX是否为忙的寄存器
9. U1\_WRITE\_REG(0x1f80-0002): 用来写入待发送到PC的数据的寄存器
10. **实验流程**

（1）主、从机初始化GPIO，设置SDA、SCL对应的端口和初始电平，向PC发送程序状态信息

（2）主机发送START信号，向PC发送程序状态信息

（3）从机接收主机发送的START信号，向PC发送程序状态信息

（4）主机发送地址和读/写控制位，向PC发送程序状态信息

（5）从机读取地址和读/写控制位，向PC发送程序状态信息

（6）若读/写控制位为 0（写），主机发送三个字符，分别是‘S’、‘O’、’C’，向PC发送程序 状态信息。从机接收三个字符，向PC发送程序状态信息；若读/写控制位为 1（读）， 从机发送三个字符，分别是‘C’、‘O’、’S’，向PC发送程序状态信息。主机接收三个字 符，向PC发送程序状态信息；

（7）主机发送STOP信号，向PC发送程序状态信息

（8）从机接收STOP信号，向PC发送程序状态信息

（9）程序结束

实验流程图如下：

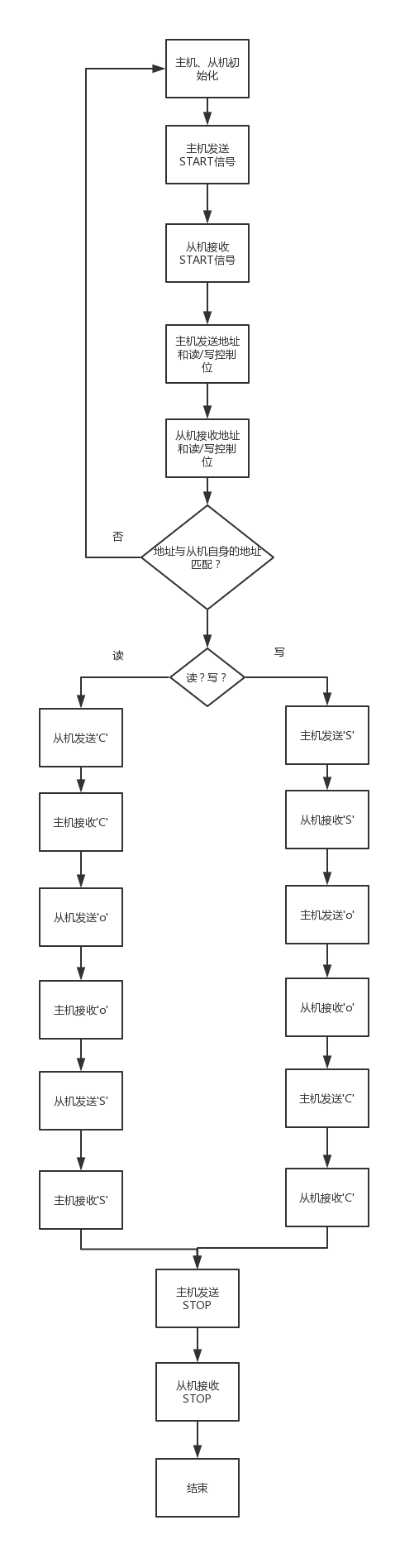


图5 实验流程

1. **基本代码：**
2. **主机向从机写入数据**

**main\_master\_MW.c**

#include "typedef.h"

#include "i2c.h"

#define READ\_MODE 1

#define WRITE\_MODE 0

#define R\_W\_MODE (WRITE\_MODE)

#define SLAVE\_ADDR\_7BITS 0x2a // 0b010-1010

#define SLAVE\_ADDR\_10BITS 0x155 // 0b01-0101-0101

void user\_interrupt(void){}

int main(void){

int i = 0;

int addr\_rw = 0;

uint8\_t data = 0; // unsigned char

while(1){

i2c\_debug("\n----I2C MASTER WRITE----\n"); // send string to PC

addr\_rw = ((SLAVE\_ADDR\_7BITS << 1) | WRITE\_MODE);

/\* I2C master init \*/

i2c\_master\_init();

i2c\_debug("\n0. Master init.\n");

/\* send START signal \*/

i2c\_master\_start();

i2c\_debug("1. Master has sent START signal.\n");

/\* send address and R/W bit \*/

if(!i2c\_master\_write\_byte(addr\_rw)){

i2c\_debug("2. Target slave has replied.\n");

}else{

i2c\_debug("2. No slave replies.\n");

}

#if 0 // 10-bit address

/\* send address and R/W bit \*/

if(!i2c\_master\_write\_11bits(addr\_rw)){

i2c\_debug("2. Target slave has replied(11-bit address).\n");

}else{

i2c\_debug("2. No slave replies.\n");

}

#endif

udelay(7\*I2C\_DELAY);

/\* write a byte to slave \*/

if(!i2c\_master\_write\_byte('S')){

i2c\_debug("3. Master wrote a byte('S') to slave successfully.\n");

}else{

i2c\_debug("3. Master wrote a byte('S') to slave fail.\n");

}

udelay(5\*I2C\_DELAY);

/\* write a byte to slave \*/

if(!i2c\_master\_write\_byte('o')){

i2c\_debug("4. Master wrote a byte('o') to slave successfully.\n");

}else{

i2c\_debug("4. Master wrote a byte('o') to slave fail.\n");

}

udelay(5\*I2C\_DELAY);

/\* write a byte to slave \*/

if(!i2c\_master\_write\_byte('C')){

i2c\_debug("5. Master wrote a byte('C') to slave successfully.\n");

}else{

i2c\_debug("5. Master wrote a byte('C') to slave fail.\n");

}

/\* send STOP signal \*/

i2c\_master\_stop();

i2c\_debug("6. Master has sent STOP signal.\n");

break;

}

while(1);

return 0;

}

**main\_slave\_MW.c**

#include "typedef.h"

#include "i2c.h"

#include "str.h"

#define READ\_MODE 1

#define WRITE\_MODE 0

// 0b010-1010

#define SLAVE\_ADDR\_7BITS 0x2a

// 0b01-0101-0101

#define SLAVE\_ADDR\_10BITS 0x155

void user\_interrupt(void){}

int main(void){

int data = 0x76;

while(1){

i2c\_debug("\n----I2C MASTER WRITE----\n");

/\* I2C slave init \*/

i2c\_slave\_init();

i2c\_debug("\n0. Slave init.\n");

/\* wait for START signal \*/

if(!i2c\_slave\_wait\_start()){

i2c\_debug("1. Slave has captured START signal.\n");

}else{

continue;

}

/\* capture address and R/W bit then ack or nack\*/

data = i2c\_slave\_read\_byte(1, SLAVE\_ADDR\_7BITS);

if(data != 2){

i2c\_debug("2. Slave address fits. ");

if(data == 0){

i2c\_debug("Master write.\n");

}else{

i2c\_debug("Master read.\n");

}

}else{

i2c\_debug("2. Slave address doesn't fit\n");

}

#if 0 // 10-bit address

/\* capture address and R/W bit then ack or nack\*/

data = i2c\_slave\_read\_11bits(SLAVE\_ADDR\_10BITS);

if(data != 2){

i2c\_debug("2. Slave address fits. ");

if(data == 1){

i2c\_debug("Master write.\n");

}else{

i2c\_debug("Master read.\n");

}

}else{

i2c\_debug("2. Slave address doesn't fit\n");

}

#endif

/\* read a byte from master \*/

data = i2c\_slave\_read\_byte(0, 0);

i2c\_debug("3. Slave read a byte('");

putch(data);

i2c\_debug("') from master successfully.\n");

/\* read a byte from master \*/

data = i2c\_slave\_read\_byte(0, 0);

i2c\_debug("4. Slave read a byte('");

putch(data);

i2c\_debug("') from master successfully.\n");

/\* read a byte from master \*/

data = i2c\_slave\_read\_byte(0, 0);

i2c\_debug("5. Slave read a byte('");

putch(data);

i2c\_debug("') from master successfully.\n");

/\* wait for STOP signal \*/

i2c\_slave\_wait\_stop();

i2c\_debug("6. Slave has captured STOP signal.\n");

break;

}

while(1);

return 0;

}

1. **主机从从机中读取数据**

**main\_master\_MR.c**

#include "typedef.h"

#include "i2c.h"

#define READ\_MODE 1

#define WRITE\_MODE 0

#define R\_W\_MODE (WRITE\_MODE)

#define SLAVE\_ADDR\_7BITS 0x2a // 0b010-1010

#define SLAVE\_ADDR\_10BITS 0x155 // 0b01-0101-0101

void user\_interrupt(void){}

int main(void){

int i = 0;

int addr\_rw = 0;

uint8\_t data = 0;

while(1){

i2c\_debug("\n----I2C MASTER READ----\n");

addr\_rw = ((SLAVE\_ADDR\_7BITS << 1) | READ\_MODE);

/\* I2C master init \*/

i2c\_master\_init();

i2c\_debug("\n0. Master init.\n");

/\* send START signal \*/

i2c\_master\_start();

i2c\_debug("1. Master has sent START signal.\n");

/\* send address and R/W bit \*/

if(!i2c\_master\_write\_byte(addr\_rw)){

i2c\_debug("2. Target slave has replied.\n");

}else{

i2c\_debug("2. No slave replies.\n");

}

udelay(7\*I2C\_DELAY);

/\* read a byte from slave \*/

data = i2c\_master\_read\_byte();

i2c\_debug("3. Master read a byte('");

putch(data);

i2c\_debug("') from slave successfully.\n");

udelay(5\*I2C\_DELAY);

/\* read a byte from slave \*/

data = i2c\_master\_read\_byte();

i2c\_debug("4. Master read a byte('");

putch(data);

i2c\_debug("') from slave successfully.\n");

udelay(5\*I2C\_DELAY);

/\* read a byte from slave \*/

data = i2c\_master\_read\_byte();

i2c\_debug("5. Master read a byte('");

putch(data);

i2c\_debug("') from slave successfully.\n");

/\* send STOP signal \*/

i2c\_master\_stop();

i2c\_debug("6. Master has sent STOP signal.\n");

break;

}

while(1);

return 0;

}

**main\_slave\_MR.c**

#include "typedef.h"

#include "i2c.h"

#include "str.h"

#define READ\_MODE 1

#define WRITE\_MODE 0

// 0b010-1010

#define SLAVE\_ADDR\_7BITS 0x2a

// 0b01-0101-0101

#define SLAVE\_ADDR\_10BITS 0x155

void user\_interrupt(void){}

int main(void){

int data = 0x76;

while(1){

i2c\_debug("\n----I2C MASTER READ----\n");

/\* I2C slave init \*/

i2c\_slave\_init();

i2c\_debug("\n0. Slave init.\n");

/\* wait for START signal \*/

if(!i2c\_slave\_wait\_start()){

i2c\_debug("1. Slave has captured START signal.\n");

}else{

continue;

}

/\* capture address and R/W bit then ack or nack\*/

data = i2c\_slave\_read\_byte(1, SLAVE\_ADDR\_7BITS);

if(data != 2){

i2c\_debug("2. Slave address fits. ");

if(data == 0){

i2c\_debug("Master write.\n");

}else{

i2c\_debug("Master read.\n");

}

}else{

i2c\_debug("2. Slave address doesn't fit\n");

}

/\* write a byte to master \*/

if(!i2c\_slave\_write\_byte('C')){

i2c\_debug("3. Slave wrote a byte('C') to master successfully.\n");

}else{

i2c\_debug("3. Slave wrote a byte('C') to master fail.\n");

}

/\* write a byte to master \*/

if(!i2c\_slave\_write\_byte('o')){

i2c\_debug("4. Slave wrote a byte('o') to master successfully.\n");

}else{

i2c\_debug("4. Slave wrote a byte('o') to master fail.\n");

}

/\* write a byte to master \*/

if(!i2c\_slave\_write\_byte('S')){

i2c\_debug("5. Slave wrote a byte('S') to master successfully.\n");

}else{

i2c\_debug("5. Slave wrote a byte('S') to master fail.\n");

}

/\* wait for STOP signal \*/

i2c\_slave\_wait\_stop();

i2c\_debug("6. Slave has captured STOP signal.\n");

break;

}

while(1);

return 0;

}

1. **在CamelStudio上编译**

（1） 打开 CamelStudio

在安装 CamelStudio的目录中，双击 CamelStudio图标，然后您将获得如下方图片所示的 CMStudio 主界面。

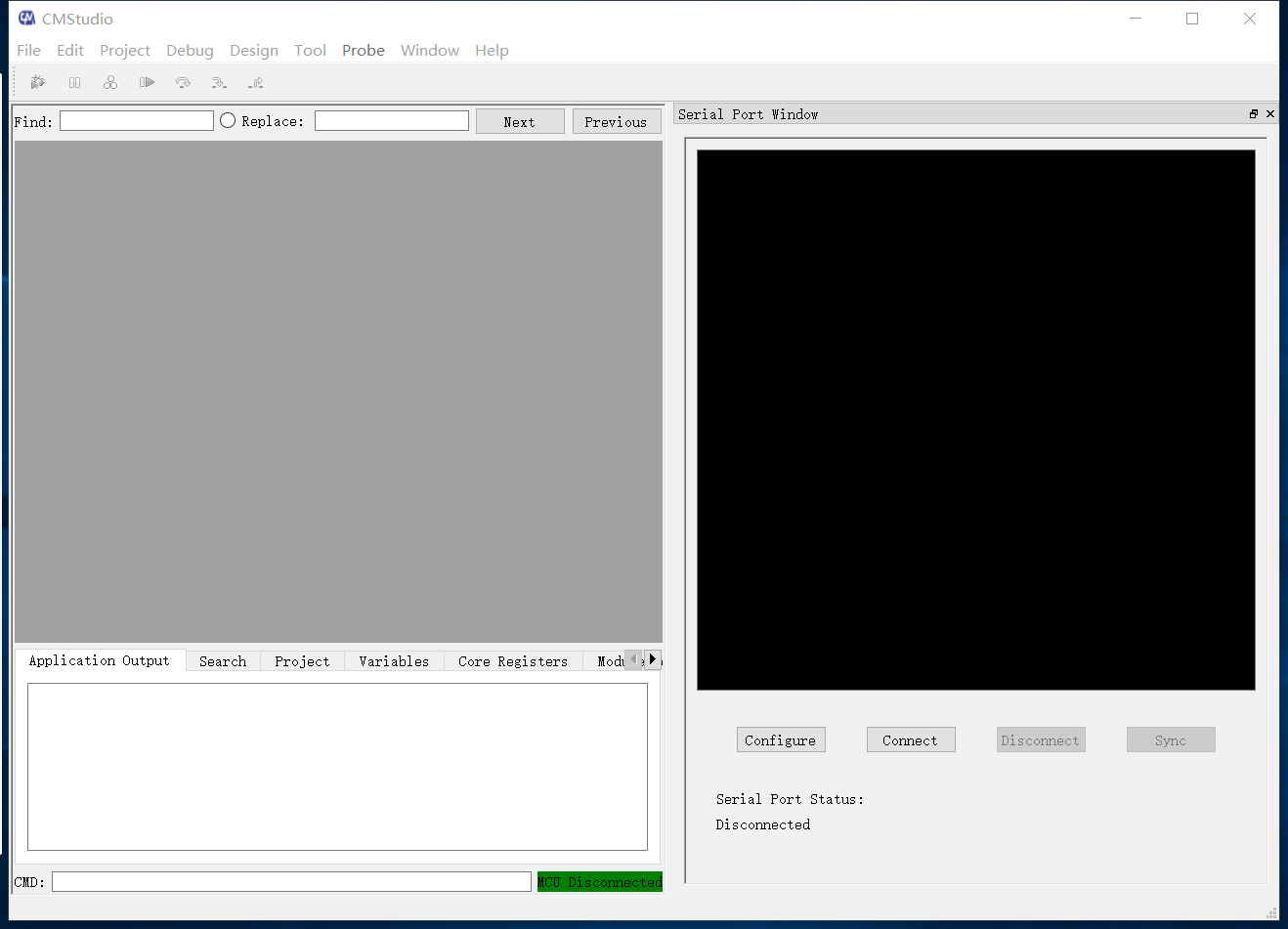


图6 CMStudio界面

（2） 新建项目

在 Project菜单中，点击New Project。在弹出的对话框里，输入 i2c作为项目名称，选择一个合适的文件夹作为保存路径，再选择M2，最后点击OK。这样，CMStudio就会在该路径下新建一个i2c文件夹和一个i2c.cms 项目文件。

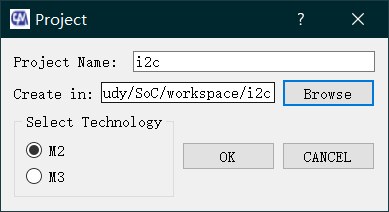


图7

在File中打开项目文件 i2c.cms，该文件包含了项目的参数设置以及要调用的源文件。



图8

1. 将文件添加到项目中

**主机向从机写数据实验：**

1. 烧写主机程序：

删去“A\_SOURCE=”和“C\_SOURCE=”前面的“//”，分别在其后面输入 delay.s 和 main\_master\_MW.c i2c.c。要注意的是，您应将源文件放在与项目文件相同的目录下，或者提供源文件的完整路径。

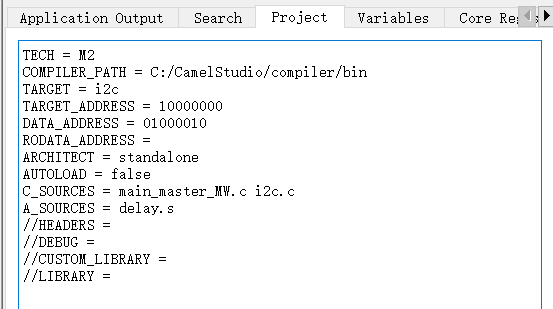


图9

（b）烧写从机程序

“C\_SOURCE=”后面改为main\_slave\_MW.c i2c.c。要注意的是，您应将源文件放在与项目文件相同的目录下，或者提供源文件的完整路径。

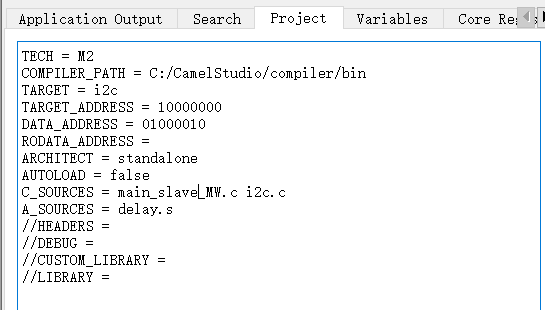


图 10

**主机从从机读数据实验：**

烧录主机、从机程序时的项目文件截图如下所示，具体步骤不再赘述。

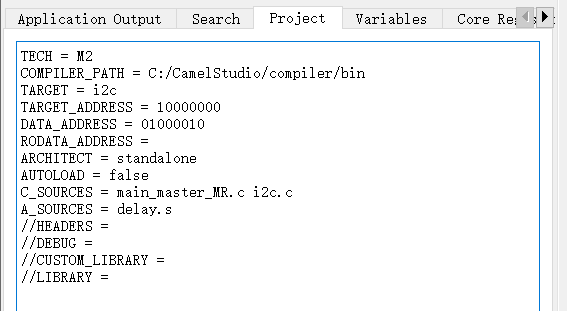


图 11 主机从从机读数据实验，主机程序

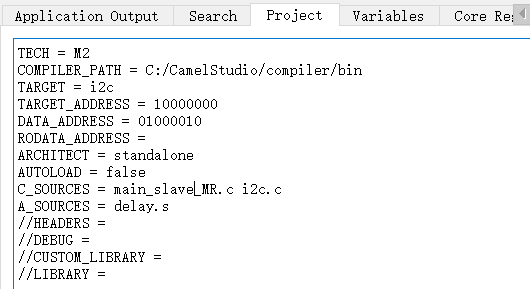


图 11 主机从从机读数据实验，从机程序

（4）构建程序

在Project菜单中，点击Build。程序构建过程中，详细信息会在Application Output窗口显示出来。如果一切正常，CMStudio会在 i2c/release目录下生成bin文件。下图为主机从从机读取数据的实验中，主机程序的构建结果，其他情况的程序构建过程同理可得，不再赘述。



图12

1. **上传到M2系统**

(1) 连接到 M2

点击串行端口窗口下的Configure按钮，在弹出的对话框中设置如下图所示参数后，单击Apply。再点击Connect并打开设备电源，屏幕上将显示启动菜单。

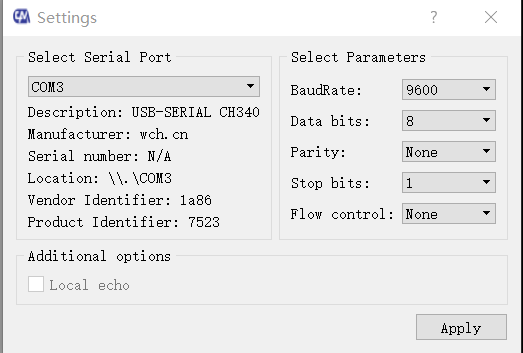


图13

(2) 上传

在Tool菜单中，点击UART LOAD后，加载程序窗口将出现。如果在构建过程中没有出现错误，它将自动填充load address和program处的值。单击OK， CMStudio 会把程序加载到目标单元上。几秒钟后，Application Output窗口会弹出类似于“load xxxx line code to load address”的信息，表明上传已完成。

主机向从机写数据和主机从从机读数据两个实验中，主、从机的程序要分别进行烧录。

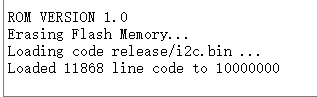
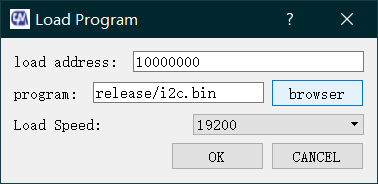


图14

1. **运行**

通过上一步骤，我们已经将用户程序烧录进芯片的Flash空间。接下来运行程序的方式有两种：

①当TESTB处于“0”时，对应芯片处于root模式，此时按下RSTB按钮，首先存放于Rom空间的boot程序被启动，在PC端对应显示一个界面控制菜单。该模式下通过界面控制菜单修改PC指针的值，从Flash空间开始执行用户程序。

②当TESTB处于“1”时，对应芯片处于user模式，此时按下RSTB按钮，PC指针直接指向Flash空间开始执行用户程序。

下列两组图片分别为：

1. 主机向从机写数据实验（见图15）
2. 主机从从机读数据实验（见图16）

主、从机程序运行的最终结果。可见实现了实验要求。

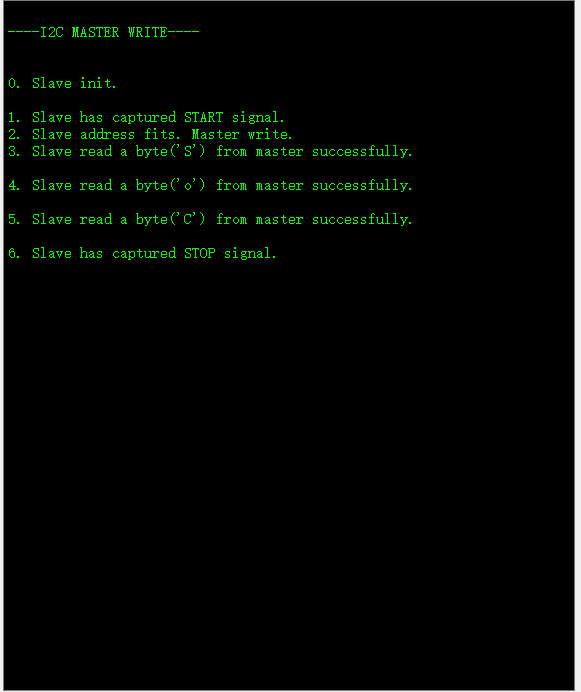
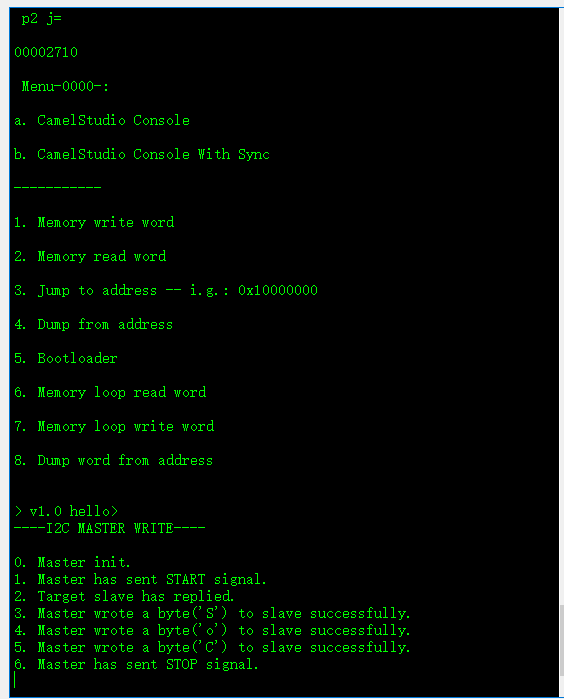


图15 主机向从机写数据实验运行结果图（左为主机，右为从机）

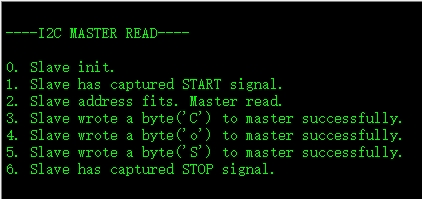
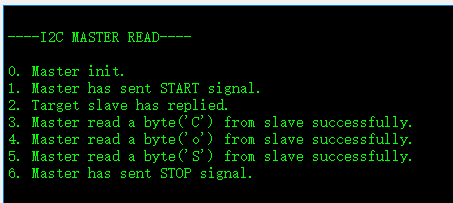


图15 主机从从机读数据实验运行结果图（左为主机，右为从机）

1. **实验拓展**

（1）本次实验中，M2e芯片运行频率为3MHz，比较慢。且每个SCL时钟脉冲期间，主从机都需要进行比较多的操作。如果脉冲周期太短，会导致从机错过一些数据。因此比特率比较低。是否有什么办法提高波特率？

（2）将芯片工作频率超频到12MHz，比特率至少可以提高4倍。