**《计算系统设计与实现》**

实验报告

**姓名： 常添**

**班级： 2203102**

**学号： 2022111699**

**哈尔滨工业大学计算学部**

**2025年3月**

### 实验2 I2C数据传送设计

**一、实验目的（2分）**

** 掌握I2C总线的基本原理和通信协议，了解主机/从机模式的设置方法。**

** 学会在TivaC平台下使用TivaWare进行I2C通信编程，能独立完成I2C初始化、发送、接收等操作。**

** 通过主机与从机的回送实验，验证I2C通信的正确性，并通过UART串口打印调试结果。**

**二、实验环境（2分）**

** 硬件设备：**

**TM4C123GH6PM LaunchPad 开发板 (Tiva LaunchPad)**

**USB数据线，用于供电和调试**

** 软件环境：**

**Code Composer Studio (CCS) 12.x 或更高版本**

**TivaWare\_C\_Series-2.2.0.xxx 库文件**

**PuTTY / Tera Term等串口调试终端（波特率115200）**

**三、实验内容**

**1、I2C初始化及配置**

**（1）写出使能I2C外设的代码，并解释相关参数。（5分）**

// 使能 I2C0 模块

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);

// 使能 GPIOB 模块，PB2/PB3 用于 I2C0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB);

解释：

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0)：使能 MCU 内部 I2C0 外设的时钟，不加这行会导致 I2C0 寄存器不可用。

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB)：使能 GPIOB 的时钟，因为 I2C0 的 SCL/SDA 引脚位于 PB2/PB3 上。

**（2）写出使能用于I2C端口的代码，并指出端口信息。（5分）**

// 配置PB2/PB3为I2C功能

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL);

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA);

// 配置PB2为I2C SCL引脚 (带内部上拉)

GPIOPinTypeI2CSCL(GPIO\_PORTB\_BASE, GPIO\_PIN\_2);

// 配置PB3为I2C SDA引脚 (开漏+上拉)

GPIOPinTypeI2C(GPIO\_PORTB\_BASE, GPIO\_PIN\_3);

解释：

GPIOPinConfigure 通过PinMap.h中的宏将指定引脚功能复用为 I2C。

GPIOPinTypeI2CSCL、GPIOPinTypeI2C 则分别指定引脚作为SCL、SDA使用。

端口信息：本实验使用的端口是 GPIOB，引脚分别是 PB2 (SCL) 和 PB3 (SDA)。

**（3） SDA、SCL分别对应端口的管脚号：SDA\_\_PB3\_\_\_\_\_, SCL\_\_PB2\_\_\_\_\_ （2分）**

**（4）I2C的传输速率为\_\_ 100kbps\_\_\_\_\_\_（2分）**

**（5）写出I2C的时钟设置代码，并解释相关参数。（5分）**

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_OSC), 25000000);

I2CMasterInitExpClk(I2C0\_BASE, ui32SysClock, false);

解释：

SysCtlClockFreqSet：配置系统时钟为 25MHz（假设 LaunchPad 外部晶振为25MHz）。其中

SYSCTL\_XTAL\_25MHZ 表示25MHz晶振；

SYSCTL\_OSC\_MAIN 表示使用主振荡器；

SYSCTL\_USE\_OSC 表示直接使用外部振荡；

25000000 表示目标时钟 25MHz；

I2CMasterInitExpClk(I2C0\_BASE, ui32SysClock, false)：初始化 I2C0 主机的时钟，false 表示标准模式 100kHz 传输速率。

**2、I2C主机与从机模式配置**

**（1）写出设置I2C主机模式的代码，并分析设置的过程。（5分）**

// 使能 I2C0 Master 功能

HWREG(I2C0\_BASE + I2C\_O\_MCR) |= 0x01;

// 设置主机访问的从机地址(7位), false表示主机要向从机写

I2CMasterSlaveAddrSet(I2C0\_BASE, SLAVE\_ADDRESS, false);

分析：

HWREG(I2C0\_BASE + I2C\_O\_MCR) |= 0x01：将I2C0的Master Configuration寄存器最低位置1，启用I2C主机。

I2CMasterSlaveAddrSet(I2C0\_BASE, SLAVE\_ADDRESS, false)：指定此后要操作的从机地址为 SLAVE\_ADDRESS (本实验=0x3C)，并设置写模式。

（2）写出设置I2C从机模式的代码，并分析设置的过程。（5分）

I2CSlaveEnable(I2C0\_BASE);

I2CSlaveInit(I2C0\_BASE, SLAVE\_ADDRESS);

分析：

I2CSlaveEnable(I2C0\_BASE)：使能I2C从机功能。

I2CSlaveInit(I2C0\_BASE, SLAVE\_ADDRESS)：将本机作为从机时的地址设为0x3C（与主机保持一致）。这样I2C0就能同时充当主机/从机进行回送测试。

**（3）主机发送数据的指令是？（2分）**

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND)

该函数的第二个参数指明了要执行“单次发送”操作。

**（4）如何配置成主机接收，从机发送的模式？（2分）**

先调用I2CMasterSlaveAddrSet(I2C0\_BASE, SLAVE\_ADDRESS, true);

其中 true 表示主机“读”模式(接收)。随后使用 I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE) 发起接收请求。

**（5）从机发送数据的指令是？（2分）**

I2CSlaveDataPut(I2C0\_BASE, data);

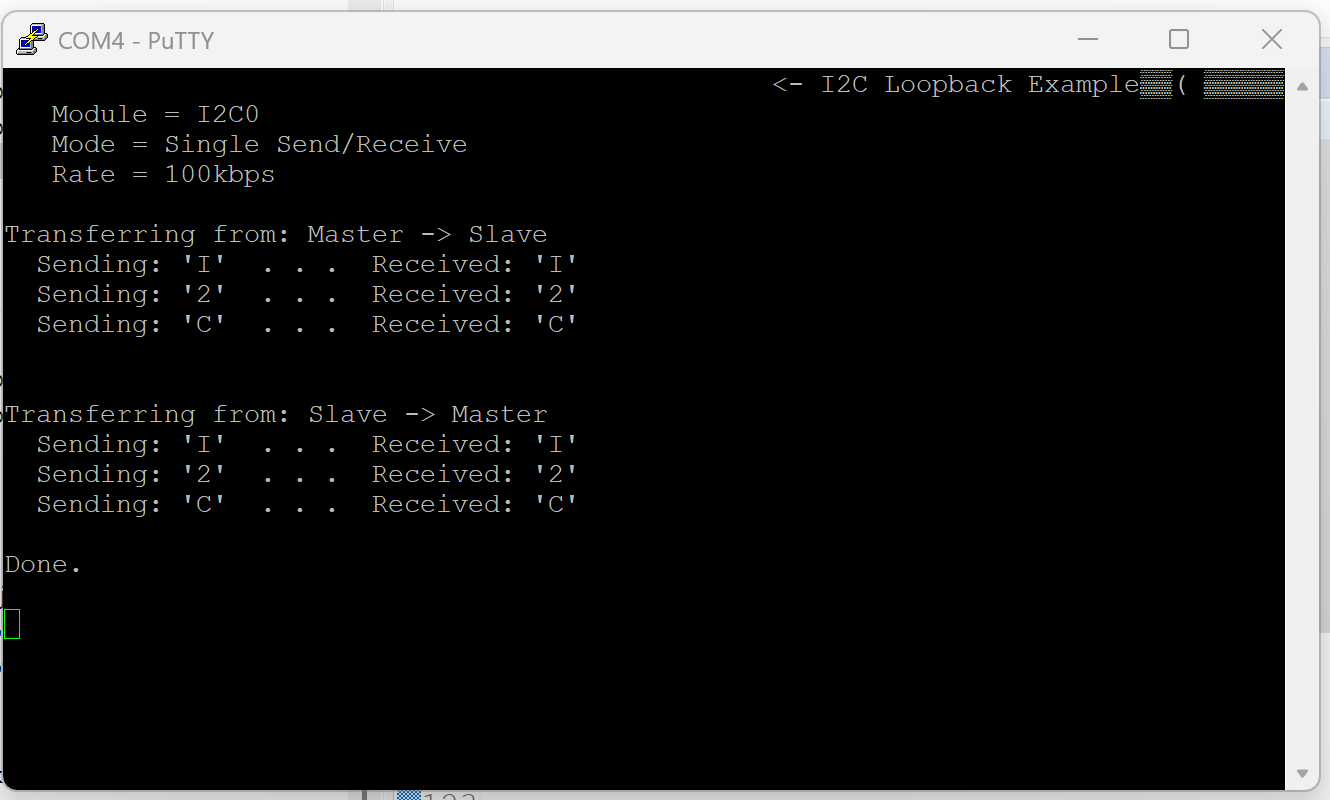
从机将要发送的数据写入该寄存器后，主机即可读到。

**3、I2C数据发送**

**（1）主机向从机发送的数据：（2分）**

通过程序可见，发送的内容为：'I', '2', 'C'。

串口助手输出截图：（3分）



**（2）从机向主机发送的数据：（2分）**

发送内容同样是：'I', '2', 'C'（回送测试）。

**串口助手输出截图：（3分）**

电脑萤幕的截图

AI 生成的内容可能不正确。

**（3）写出设置UART时钟的代码，并解释相关参数。（5分）**

// 使能GPIOA和UART0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

// 选择 UART0 的时钟源为 PIOSC(16MHz内部振荡)

UARTClockSourceSet(UART0\_BASE, UART\_CLOCK\_PIOSC);

// 配置 UART: 波特率 115200, 基于 16MHz

UARTStdioConfig(0, 115200, 16000000);

解释：

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA) / SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0)：启用GPIOA和UART0外设时钟。

UARTClockSourceSet(UART0\_BASE, UART\_CLOCK\_PIOSC)：将UART0的时钟源指定为内部16MHz振荡器(PIOSC)，与系统主时钟(25MHz)独立。

UARTStdioConfig(0, 115200, 16000000)：配置UART0的波特率为115200，第三个参数为UART模块时钟频率16MHz；内部自动计算波特率分频系数。

**四、思考题（4分）**

**1、本实验中UART工作时的波特率是如何获得的？**

本实验通过以下函数设置 UART 的波特率：

// 使能 UART0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

// 选择 PIOSC (16MHz) 作为 UART0 时钟源

UARTClockSourceSet(UART0\_BASE, UART\_CLOCK\_PIOSC);

// 配置 UART0 的波特率为 115200, 时钟16MHz

UARTStdioConfig(0, 115200, 16000000);

其中 UARTStdioConfig(0, 115200, 16000000) 会根据传入的时钟频率 (16 MHz) 和期望波特率 (115200) 计算并设置 UART 的分频系数，使串口实际通信达到 115200bps。TivaWare 库函数会自动完成对整数分频寄存器和小数分频寄存器的配置，从而实现精确的波特率。

**2、I2C回环模式中，从机地址应如何设置 ？**

在回环模式下（即同一个 MCU 上的 I2C0 同时作为主机和从机），从机地址要与主机通信时使用的从机地址保持一致。例如，本实验将从机地址设为 0x3C：

// 设置从机地址

I2CSlaveInit(I2C0\_BASE, 0x3C);

// 主机访问此从机地址

I2CMasterSlaveAddrSet(I2C0\_BASE, 0x3C, false);

如果从机端和主机端的地址不一致，主机发送的寻址信号就无法匹配该从机，也就无法在同一板子上完成数据回送。

**五、实验总结（2分）**

** 通过本次实验，进一步巩固了对 I2C 总线原理 和 UART 串口调试方法 的理解。实验中，将 I2C0 分别配置为主机与从机，利用回环机制验证收发数据的正确性；并在终端通过 UART 波特率 115200bps 打印出实验过程和结果。**

** 实践证明，只要硬件引脚配置正确（PB2/PB3）、I2C 从机地址匹配、主机/从机模式切换恰当，就能成功发送并接收相同的数据，从而完成一次完整的 I2C 回环通信。该思路在日后与外部 I2C 外设（如传感器、EEPROM）通信时同样适用。**