目 录

1. 实验目的 1

2. 程序流程示意图 2

3. 代码及实现思路分析 3

4. 实验结果 4

5. 感想与收获 5

6. 讨论题 6

7. 源码 10

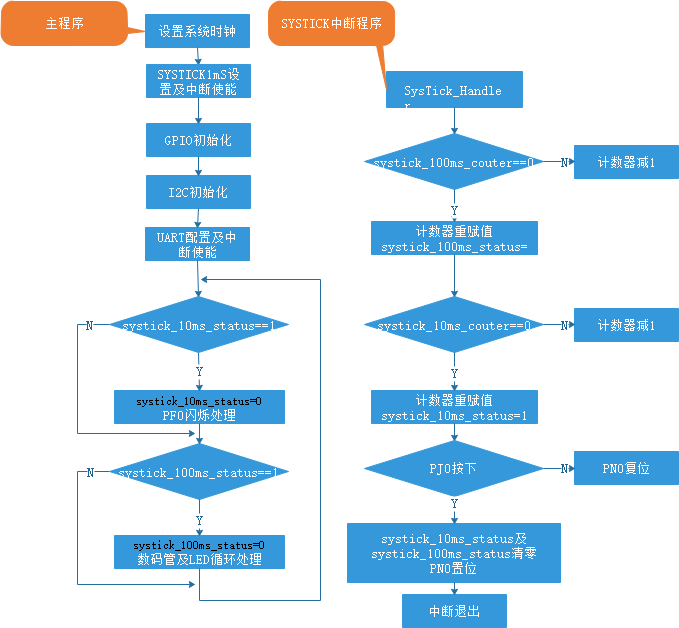
# 实验目的

了解 UART 串行通讯的工作原理

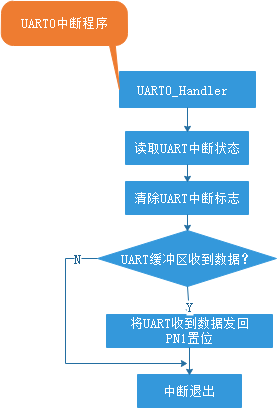
掌握在 PC 端通过串口调试工具与实验板通过 UART 通讯的方法

掌握 UART 的堵塞式与非堵塞式通讯方法

# 程序流程示意图

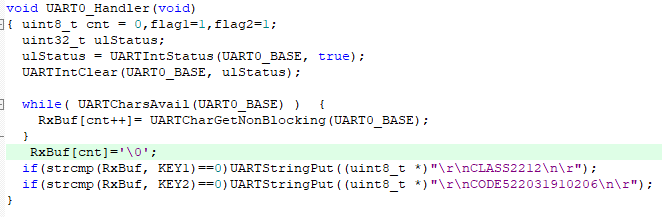


图一、主程序与SYSTICK中断程序逻辑



图二、UARTO中断程序逻辑

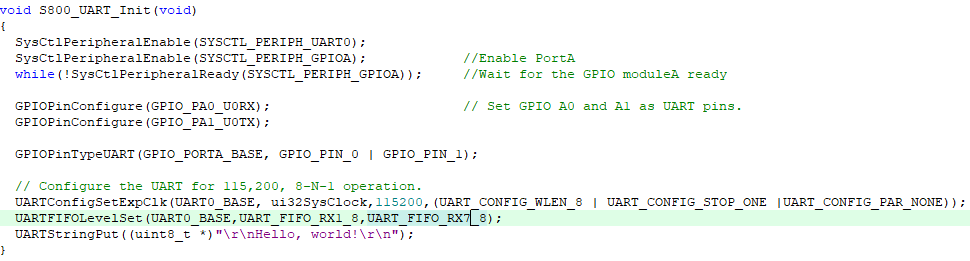
# 代码及实现思路分析



使用UARTCharsAvail(UART0\_BASE)检查 UART 接收 FIFO 是否有可用数据，如果有的话就通过UARTCharGetNonBlocking接收并存储到RxBuf数组中。当输入全部被接收后，就使用strcmp函数比较接收到的字符串与“AT+CLASS”和“AT+STUDENTCODE”是否相同，相同便作出对应的输出。

90d6e7ba8ccc5d7cb15919c8563fc0d

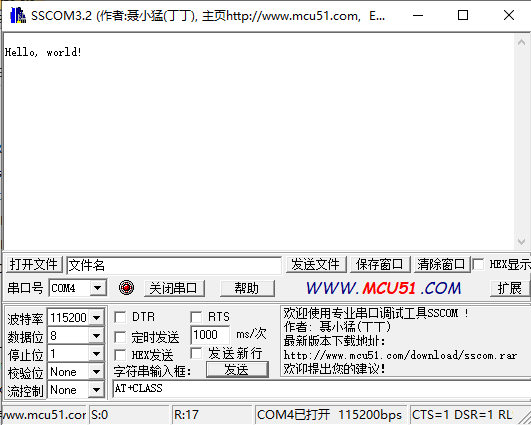
使用了非阻塞的方式接收字符串。



在初始化函数中加入UARTFIFOLevelSet(UART0\_BASE,UART\_FIFO\_RX1\_8,UART\_FIFO\_RX7\_8)，便可以成功讲AT+STUDENTCODE发送给串口通信模拟器并被接收。

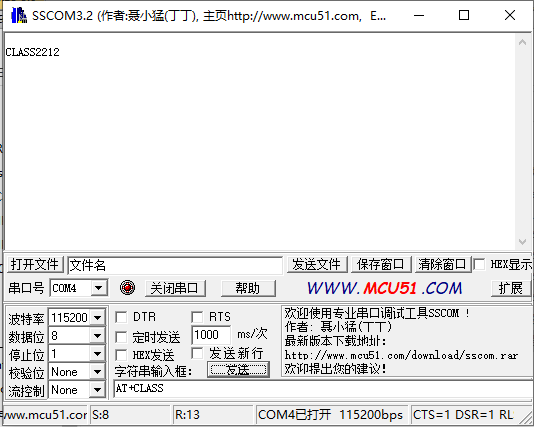
# 实验结果

按下RESET键后，实验板回以Hello, world!



图三、按下RESET

当PC端发来AT+CLASS后，实验板回以CLASS2212



图三、发送AT+CLASS

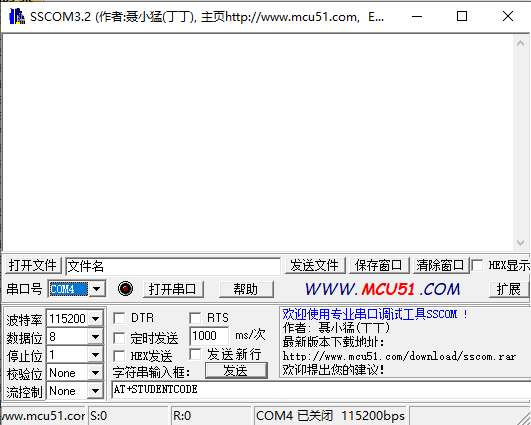
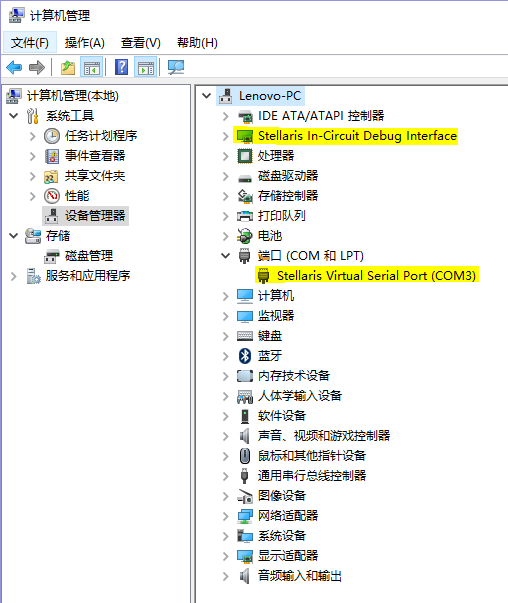
当PC端发来AT+STUDENTCODE后，实验板回以CODE522031910206



图四、发送AT+STUDENTCODE

# 感想与收获

学会了串口通信模拟器的使用方式：



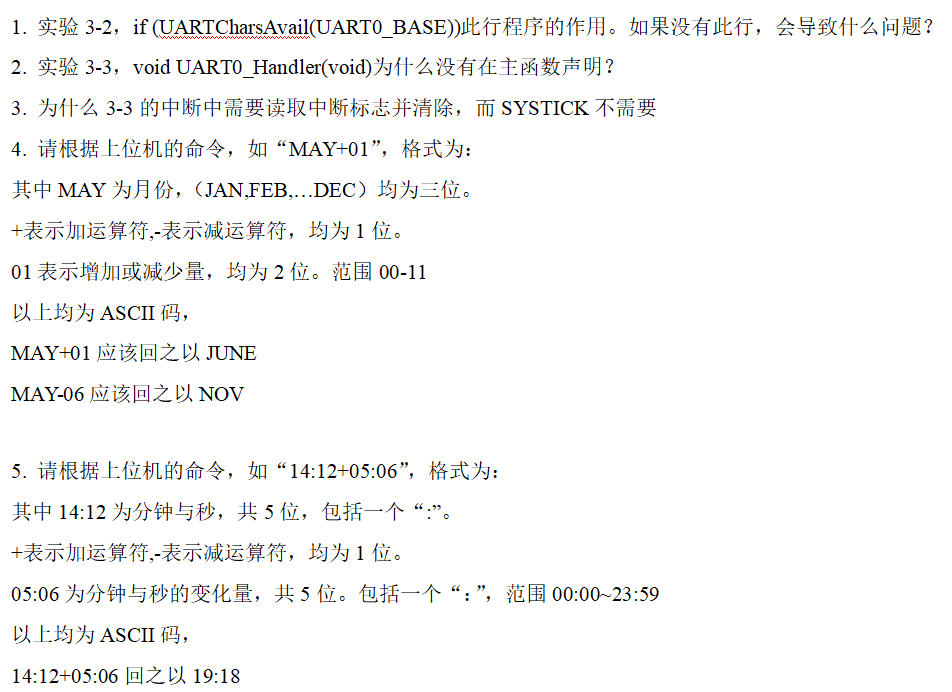
图五、配置虚拟串口通信模拟器

在虚拟机中讲串口号改为COM4,将波特率改为115200便可开始模拟通信。左下方的S代表发出的字节数，左下方的R代表读取到的字节数。

学会了使用库函数与UARTFIFOLevelSet函数来简化代码逻辑，实现性能更强的通信：

将传输的信息通过SSCOM的信息输入框发送给主板。在初始化函数中加入UARTFIFOLevelSet(UART0\_BASE,UART\_FIFO\_RX1\_8,UART\_FIFO\_RX7\_8)函数保证可以发送较长的文本。通过非阻塞的方式接收字符串后，利用C语言库中的strcmp函数来进行字符串的比较。并做出对应的返回。

# 讨论题



1. 作用：

UARTCharsAvail(UART0\_BASE)用于检查 UART 接收 FIFO 是否有可用数据。如果没有这行代码，会导致以下问题：

(1).数据读取错误：直接读取可能会发生在没有数据可读的情况下，读取到无效或错误数据。

(2).程序崩溃或陷入死循环：如果读取操作依赖于此检查，缺少它会导致程序读取错误地址或数据，可能崩溃或进入死循环。

2. 没有在主函数声明的原因：

中断处理函数 UART0\_Handler不需要在主函数中声明，因为它是在启动文件或系统中断向量表中声明和定义的。这是一个惯例，中断处理函数由硬件中断向量表直接调用。

3. 3-3的中断中需要读取中断标志并清除，而SYSTICK不需要的原因:

(1).UART中断：UART 中断标志在触发后需要被清除，以防止中断处理程序反复处理相同的中断事件。通常是通过读取相关寄存器并进行清除操作。

(2).SysTick中断：SysTick 计时器在每次溢出时自动触发中断，不需要手动清除中断标志。SysTick 定时器自动重装和溢出处理机制不需要用户代码清除标志。

4.

#include <stdio.h>

#include <string.h>

const char \*months[] = {

"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"

};

int getMonthIndex(const char \*month) {

for (int i = 0; i < 12; i++) {

if (strncmp(months[i], month, 3) == 0) {

return i;

}

}

return -1; // 错误月份

}

void processMonthCommand(const char \*command) {

char month[4] = {0};

strncpy(month, command, 3);

char op = command[3];

int value = (command[4] - '0') \* 10 + (command[5] - '0');

int monthIndex = getMonthIndex(month);

if (monthIndex == -1) {

printf("Invalid month\n");

return;

}

if (op == '+') {

monthIndex = (monthIndex + value) % 12;

} else if (op == '-') {

monthIndex = (monthIndex - value + 12) % 12;

} else {

printf("Invalid operation\n");

return;

}

printf("Result: %s\n", months[monthIndex]);

}

int main() {

processMonthCommand("MAY+01"); // Expected output: JUNE

processMonthCommand("MAY-06"); // Expected output: NOV

return 0;

}

5.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 解析时间字符串 "MM:SS"

void parseTime(const char \*timeStr, int \*minutes, int \*seconds) {

\*minutes = (timeStr[0] - '0') \* 10 + (timeStr[1] - '0');

\*seconds = (timeStr[3] - '0') \* 10 + (timeStr[4] - '0');

}

// 将分钟和秒数转换回字符串 "MM:SS"

void formatTime(int minutes, int seconds, char \*timeStr) {

snprintf(timeStr, 6, "%02d:%02d", minutes, seconds);

}

void processTimeCommand(const char \*command) {

int minutes1, seconds1;

int minutes2, seconds2;

char result[6];

parseTime(command, &minutes1, &seconds1);

parseTime(command + 6, &minutes2, &seconds2);

char op = command[5];

int totalSeconds1 = minutes1 \* 60 + seconds1;

int totalSeconds2 = minutes2 \* 60 + seconds2;

int resultSeconds;

if (op == '+') {

resultSeconds = totalSeconds1 + totalSeconds2;

} else if (op == '-') {

resultSeconds = totalSeconds1 - totalSeconds2;

} else {

printf("Invalid operation\n");

return;

}

if (resultSeconds < 0) {

resultSeconds += 24 \* 60 \* 60; // 处理负时间

}

int resultMinutes = (resultSeconds / 60) % (24 \* 60);

resultSeconds = resultSeconds % 60;

formatTime(resultMinutes, resultSeconds, result);

printf("Result: %s\n", result);

}

int main() {

processTimeCommand("14:12+05:06"); // Expected output: 19:18

return 0;

}

# 源码

#include <stdint.h>

#include <stdbool.h>

#include <string.h>

#include "hw\_memmap.h"

#include "debug.h"

#include "gpio.h"

#include "hw\_i2c.h"

#include "hw\_types.h"

#include "i2c.h"

#include "pin\_map.h"

#include "sysctl.h"

#include "systick.h"

#include "interrupt.h"

#include "uart.h"

#include "hw\_ints.h"

#define SYSTICK\_FREQUENCY 1000 //1000hz

#define I2C\_FLASHTIME 500 //500mS

#define GPIO\_FLASHTIME 300 //300mS

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//I2C GPIO chip address and resigster define

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define TCA6424\_I2CADDR 0x22

#define PCA9557\_I2CADDR 0x18

#define PCA9557\_INPUT 0x00

#define PCA9557\_OUTPUT 0x01

#define PCA9557\_POLINVERT 0x02

#define PCA9557\_CONFIG 0x03

#define TCA6424\_CONFIG\_PORT0 0x0c

#define TCA6424\_CONFIG\_PORT1 0x0d

#define TCA6424\_CONFIG\_PORT2 0x0e

#define TCA6424\_INPUT\_PORT0 0x00

#define TCA6424\_INPUT\_PORT1 0x01

#define TCA6424\_INPUT\_PORT2 0x02

#define TCA6424\_OUTPUT\_PORT0 0x04

#define TCA6424\_OUTPUT\_PORT1 0x05

#define TCA6424\_OUTPUT\_PORT2 0x06

void Delay(uint32\_t value);

void S800\_GPIO\_Init(void);

uint8\_t I2C0\_WriteByte(uint8\_t DevAddr, uint8\_t RegAddr, uint8\_t WriteData);

uint8\_t I2C0\_ReadByte(uint8\_t DevAddr, uint8\_t RegAddr);

void S800\_I2C0\_Init(void);

void S800\_UART\_Init(void);

//systick software counter define

volatile uint16\_t systick\_10ms\_couter,systick\_100ms\_couter;

volatile uint8\_t systick\_10ms\_status,systick\_100ms\_status;

volatile uint8\_t result,cnt,key\_value,gpio\_status;

volatile uint8\_t rightshift = 0x01;

uint32\_t ui32SysClock;

uint8\_t seg7[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x58,0x5e,0x079,0x71,0x5c};

uint8\_t uart\_receive\_char;

char \*uart\_receive\_string;

char \*KEY1="AT+CLASS";

char \*KEY2="AT+STUDENTCODE";

char RxBuf[100];

int main(void)

{

volatile uint16\_t i2c\_flash\_cnt,gpio\_flash\_cnt;

//use internal 16M oscillator, PIOSC

//ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_16MHZ |SYSCTL\_OSC\_INT |SYSCTL\_USE\_OSC), 16000000);

//ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_16MHZ |SYSCTL\_OSC\_INT |SYSCTL\_USE\_OSC), 8000000);

//use external 25M oscillator, MOSC

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

//use external 25M oscillator and PLL to 120M

//ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |SYSCTL\_CFG\_VCO\_480), 120000000);;

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL |SYSCTL\_CFG\_VCO\_480), 20000000);

SysTickPeriodSet(ui32SysClock/SYSTICK\_FREQUENCY);

SysTickEnable();

SysTickIntEnable(); //Enable Systick interrupt

S800\_GPIO\_Init();

S800\_I2C0\_Init();

S800\_UART\_Init();

IntEnable(INT\_UART0);

UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT); //Enable UART0 RX,TX interrupt

IntMasterEnable();

while (1)

{

if (systick\_10ms\_status)

{

systick\_10ms\_status = 0;

if (++gpio\_flash\_cnt >= GPIO\_FLASHTIME/10)

{

gpio\_flash\_cnt = 0;

if (gpio\_status)

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0,GPIO\_PIN\_0 );

else

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0,0);

gpio\_status = !gpio\_status;

}

}

if (systick\_100ms\_status)

{

systick\_100ms\_status = 0;

if (++i2c\_flash\_cnt >= I2C\_FLASHTIME/100)

{

i2c\_flash\_cnt = 0;

result = I2C0\_WriteByte(TCA6424\_I2CADDR,TCA6424\_OUTPUT\_PORT1,seg7[cnt+1]); //write port 1

result = I2C0\_WriteByte(TCA6424\_I2CADDR,TCA6424\_OUTPUT\_PORT2,rightshift); //write port 2

result = I2C0\_WriteByte(PCA9557\_I2CADDR,PCA9557\_OUTPUT,~rightshift);

cnt++;

rightshift= rightshift<<1;

if (cnt >= 0x8)

{

rightshift= 0x01;

cnt = 0;

}

}

}

}

}

void Delay(uint32\_t value)

{

uint32\_t ui32Loop;

for(ui32Loop = 0; ui32Loop < value; ui32Loop++){};

}

void UARTStringPut(uint8\_t \*cMessage)

{

while(\*cMessage!='\0')

UARTCharPut(UART0\_BASE,\*(cMessage++));

}

void UARTStringPutNonBlocking(const char \*cMessage)

{

while(\*cMessage!='\0')

UARTCharPutNonBlocking(UART0\_BASE,\*(cMessage++));

}

void S800\_UART\_Init(void)

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); //Enable PortA

while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOA)); //Wait for the GPIO moduleA ready

GPIOPinConfigure(GPIO\_PA0\_U0RX); // Set GPIO A0 and A1 as UART pins.

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Configure the UART for 115,200, 8-N-1 operation.

UARTConfigSetExpClk(UART0\_BASE, ui32SysClock,115200,(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |UART\_CONFIG\_PAR\_NONE));

UARTFIFOLevelSet(UART0\_BASE,UART\_FIFO\_RX1\_8,UART\_FIFO\_RX7\_8);

UARTStringPut((uint8\_t \*)"\r\nHello, world!\r\n");

}

void S800\_GPIO\_Init(void)

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF); //Enable PortF

while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOF)); //Wait for the GPIO moduleF ready

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOJ); //Enable PortJ

while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOJ)); //Wait for the GPIO moduleJ ready

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPION); //Enable PortN

while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPION)); //Wait for the GPIO moduleN ready

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0); //Set PF0 as Output pin

GPIOPinTypeGPIOOutput(GPIO\_PORTN\_BASE, GPIO\_PIN\_0); //Set PN0 as Output pin

GPIOPinTypeGPIOOutput(GPIO\_PORTN\_BASE, GPIO\_PIN\_1); //Set PN1 as Output pin

GPIOPinTypeGPIOInput(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0 | GPIO\_PIN\_1);//Set the PJ0,PJ1 as input pin

GPIOPadConfigSet(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0 | GPIO\_PIN\_1,GPIO\_STRENGTH\_2MA,GPIO\_PIN\_TYPE\_STD\_WPU);

}

void S800\_I2C0\_Init(void)

{

uint8\_t result;

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_I2C0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB);

GPIOPinConfigure(GPIO\_PB2\_I2C0SCL);

GPIOPinConfigure(GPIO\_PB3\_I2C0SDA);

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

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

I2CMasterInitExpClk(I2C0\_BASE,ui32SysClock, true); //config I2C0 400k

I2CMasterEnable(I2C0\_BASE);

result = I2C0\_WriteByte(TCA6424\_I2CADDR,TCA6424\_CONFIG\_PORT0,0x0ff); //config port 0 as input

result = I2C0\_WriteByte(TCA6424\_I2CADDR,TCA6424\_CONFIG\_PORT1,0x0); //config port 1 as output

result = I2C0\_WriteByte(TCA6424\_I2CADDR,TCA6424\_CONFIG\_PORT2,0x0); //config port 2 as output

result = I2C0\_WriteByte(PCA9557\_I2CADDR,PCA9557\_CONFIG,0x00); //config port as output

result = I2C0\_WriteByte(PCA9557\_I2CADDR,PCA9557\_OUTPUT,0x0ff); //turn off the LED1-8

}

uint8\_t I2C0\_WriteByte(uint8\_t DevAddr, uint8\_t RegAddr, uint8\_t WriteData)

{

uint8\_t rop;

while(I2CMasterBusy(I2C0\_BASE)){};

I2CMasterSlaveAddrSet(I2C0\_BASE, DevAddr, false);

I2CMasterDataPut(I2C0\_BASE, RegAddr);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

while(I2CMasterBusy(I2C0\_BASE)){};

rop = (uint8\_t)I2CMasterErr(I2C0\_BASE);

I2CMasterDataPut(I2C0\_BASE, WriteData);

I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH);

while(I2CMasterBusy(I2C0\_BASE)){};

rop = (uint8\_t)I2CMasterErr(I2C0\_BASE);

return rop;

}

uint8\_t I2C0\_ReadByte(uint8\_t DevAddr, uint8\_t RegAddr)

{

uint8\_t value,rop;

while(I2CMasterBusy(I2C0\_BASE)){};

I2CMasterSlaveAddrSet(I2C0\_BASE, DevAddr, false);

I2CMasterDataPut(I2C0\_BASE, RegAddr);

// I2CMasterControl(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

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

while(I2CMasterBusBusy(I2C0\_BASE));

rop = (uint8\_t)I2CMasterErr(I2C0\_BASE);

Delay(1);

//receive data

I2CMasterSlaveAddrSet(I2C0\_BASE, DevAddr, true);

I2CMasterControl(I2C0\_BASE,I2C\_MASTER\_CMD\_SINGLE\_RECEIVE);

while(I2CMasterBusBusy(I2C0\_BASE));

value=I2CMasterDataGet(I2C0\_BASE);

Delay(1);

return value;

}

/\*

Corresponding to the startup\_TM4C129.s vector table systick interrupt program name

\*/

void SysTick\_Handler(void)

{

if (systick\_100ms\_couter != 0)

systick\_100ms\_couter--;

else

{

systick\_100ms\_couter = SYSTICK\_FREQUENCY/10;

systick\_100ms\_status = 1;

}

if (systick\_10ms\_couter != 0)

systick\_10ms\_couter--;

else

{

systick\_10ms\_couter = SYSTICK\_FREQUENCY/100;

systick\_10ms\_status = 1;

}

if (GPIOPinRead(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0) == 0)

{

systick\_100ms\_status = systick\_10ms\_status = 0;

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0,GPIO\_PIN\_0);

}

else

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0,0);

}

/\*

Corresponding to the startup\_TM4C129.s vector table UART0\_Handler interrupt program name

\*/

/\*

void UART0\_Handler(void)

{

int32\_t uart0\_int\_status;

uart0\_int\_status = UARTIntStatus(UART0\_BASE, true); // Get the interrrupt status.

UARTIntClear(UART0\_BASE, uart0\_int\_status); //Clear the asserted interrupts

while(UARTCharsAvail(UART0\_BASE)) // Loop while there are characters in the receive FIFO.

{

///Read the next character from the UART and write it back to the UART.

//UARTCharPutNonBlocking(UART0\_BASE,UARTCharGetNonBlocking(UART0\_BASE));

//GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_1,GPIO\_PIN\_1 );

// Delay(1000);

}

//GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_1,0 );

}

\*/

void UARTStringGet(uint32\_t ui32Base,char \*cMessage,const char Iden)

{

while(1)

{

\*cMessage=UARTCharGet(ui32Base);

if(\*cMessage!=Iden)

{

cMessage=cMessage+1;

}

else

{

\*cMessage='\0';

break;

}

}

}

void UART0\_Handler(void)

{ uint8\_t cnt = 0,flag1=1,flag2=1;

uint32\_t ulStatus;

ulStatus = UARTIntStatus(UART0\_BASE, true);

UARTIntClear(UART0\_BASE, ulStatus);

while( UARTCharsAvail(UART0\_BASE) ) {

RxBuf[cnt++]= UARTCharGetNonBlocking(UART0\_BASE);

}

RxBuf[cnt]='\0';

if(strcmp(RxBuf, KEY1)==0)UARTStringPut((uint8\_t \*)"\r\nCLASS2212\n\r");

if(strcmp(RxBuf, KEY2)==0)UARTStringPut((uint8\_t \*)"\r\nCODE522031910206\n\r");

}