目录

**[1.绪论 1](#_Toc22711)**

**[2.STM32嵌入式介绍 1](#_Toc25061)**

[2.1 STM32F103ZET6芯片介绍 1](#_Toc17067)

[2.2 优缺点 1](#_Toc8147)

**[3. KEIL MDK软件的使用 2](#_Toc27810)**

[3.1 创建STM32工程 2](#_Toc29013)

[3.2 使用库函数控制LED闪亮 3](#_Toc6665)

[3.3 使用库函数控制蜂鸣器 5](#_Toc21233)

[3.4 使用库函数检测按键 7](#_Toc18658)

**[4. STM32常用功能的库函数实现 10](#_Toc6151)**

[4.1 使用STM32的串口收发数据 10](#_Toc7903)

[4.2 使用STM32控制彩色LCD显示图形和英文字符 12](#_Toc22512)

[4.3 使用STM32控制彩色LCD显示图片和中文字符 15](#_Toc1188)

**[5. 基于STM32的单总线温度检测系统设计 18](#_Toc29892)**

[5.3 软件代码设计 20](#_Toc2016)

[5.4 显示界面设计 22](#_Toc4485)

**[6.总结和展望 22](#_Toc9047)**

**[7.参考文献 23](#_Toc14990)**

**基于STM32的单总线温度检测系统设计**

**[1.绪论](#_Toc24848)**

项目结构

一个STM32F103ZET6单片机，一个DS18B20温度传感器，一个lcd屏幕，以及一些连接线和电源。

将DS18B20的VCC引脚连接到STM32F103的3.3V电源引脚，将GND引脚连接到STM32F103的GND引脚，将DQ引脚连接到STM32F103的任意GPIO引脚（本次使用了17、18）。

使用单总线协议来与DS18B20通信，发送和接收不同的指令和数据。

你需要使用SPI方式来将温度数据显示在lcd屏幕上。

项目功能

本次课程设计使用了DS18B20温度传感器、STM32F103ZET6的开发板和lcd屏幕实现了对温度的实时检测并显示在lcd屏幕。

研究目的

掌握STM32F103ZET6的使用，熟悉它的串口通信，外设使用，lcd屏幕的使用。

**[2.STM32嵌入式介绍](#_Toc25691)**

**2.1 STM32F103ZET6芯片介绍**

STM32F103ZET6芯片是一款基于ARM Cortex-M3 32位RISC内核的微控制器，具有高性能和低功耗的特点。它包含了高速嵌入式存储器（高达512KB的闪存和高达64KB的SRAM），以及多种增强型I/O和外围设备，如电动机控制、CAN、USB、ADC、DAC、定时器、串口、SPI、I2C等。它可以在2.0V到3.6V的电源下工作，温度范围为-40°C到+85°C或-40°C到+105°C12。它适用于广泛的应用领域，如电机驱动、应用控制、医疗和手持设备、PC和游戏外围设备、GPS平台、工业应用等。

**2.2 优缺点**

优势：

性能强大，支持高达72MHz的工作频率，内置浮点单元（FPU），具有丰富的外围设备和存储器。

低功耗，支持多种低功耗模式，如睡眠、停止和待机模式，具有电源复位和可编程电压检测器。

兼容性好，支持多种通信协议，如CAN、USB、SPI、I2C、USART等，可以与多种外部设备连接。

开发方便，提供了丰富的开发工具和资源，如STM32CubeMX、STM32CubeIDE、STM32CubeProgrammer等，以及多种开发板和模块。

缺点：

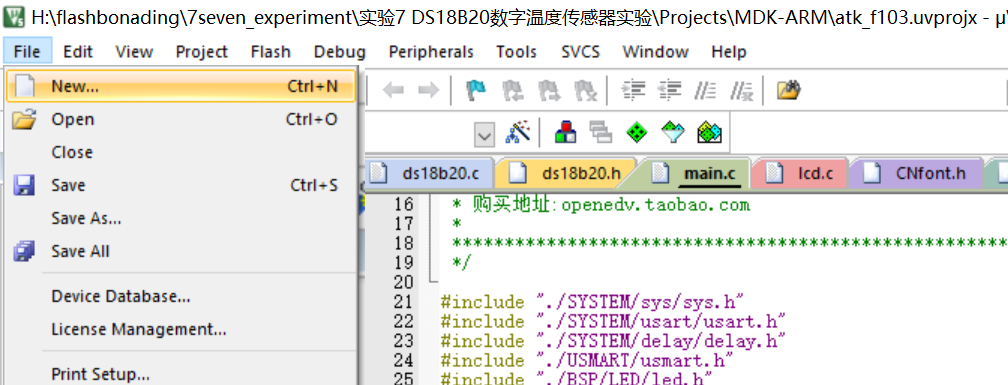
价格相对较高，与一些其他的8位或16位单片机相比，STM32F103ZET6的价格可能会更高一些。

外围电路较复杂，由于STM32F103ZET6的引脚数量较多，且部分引脚有多重功能，因此在设计外围电路时需要注意引脚的配置和分配 。

学习成本较高，由于STM32F103ZET6的功能较多，涉及到多种外围设备和协议，因此在学习和使用时需要掌握更多的知识和技能。

**[3. KEIL MDK软件的](#_Toc20280)使用**

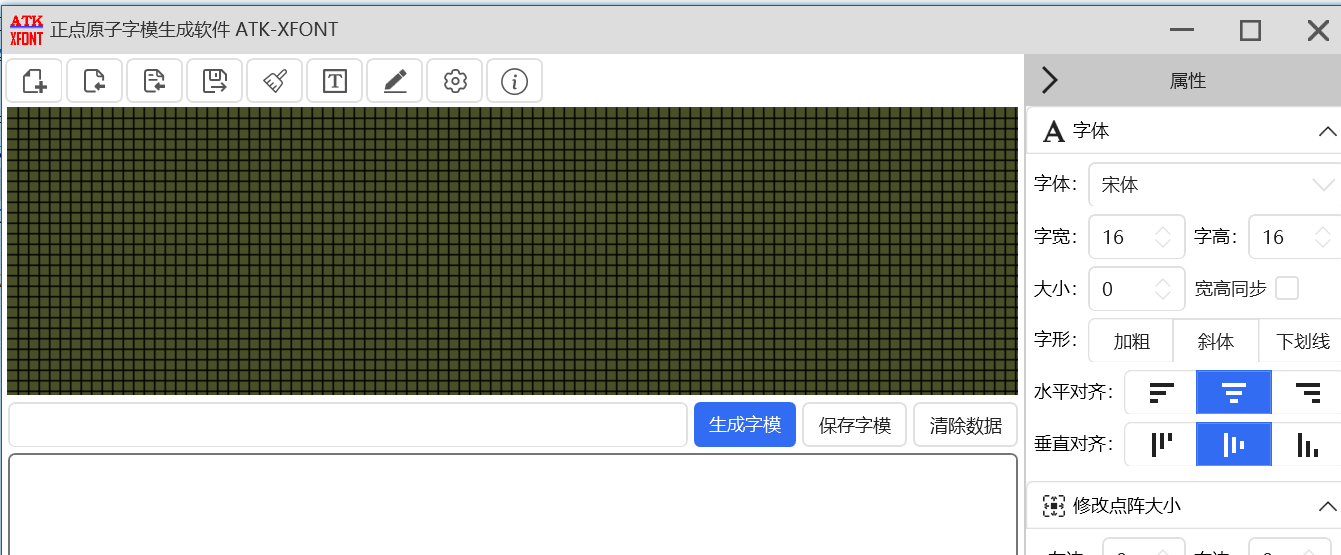
**[3.1](#_Toc15576) 创建STM32工程**



如图打开KEIL MDK软件，在file新建一个，并且在整个项目的文件需要添加项目所需的库函数。进行编译后生成HEX文件。



然后使用FlyMcu将hex文件烧录进开发板。



如果需要让屏幕上可以显示文字，需要使用这个软件生成字模，添加到lcd.c和lcd.h文件中。

**[3.2](#_Toc20082) 使用库函数控制LED闪亮**

运行KEIL MDK工具，选择STM32F103ZET6板子。

在Pinout标签页，检查SYS外设的调试接口是否选择了Serial Wire，并且PA13和PA14被自动分配和配置。

在Clock Configuration标签页，检查STM32CubeMX是否自动配置了内部振荡器和PLL@80MHz，并且系统时钟选择了PLLCLK。

在GPIO Configuration标签页，检查LD2（绿色LED）是否被设置为GPIO\_Output模式，并且对应的引脚是PA5。

配置项目并生成源代码，使用STM32CubeIDE打开项目。

编辑main.c文件，添加HAL\_GPIO\_TogglePin函数来切换LED的状态，并添加HAL\_Delay函数来延时一定的时间。

编译项目并下载到板子上，观察LED是否闪亮。

代码如下：

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/delay/delay.h"

void led\_init(void); /\* LED初始化函数声明 \*/

int i ;

int main(void)

{

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟, 72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

led\_init(); /\* LED初始化 \*/

while(1)

{

i = 500 ;

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_5,GPIO\_PIN\_SET); /\* PB5置1 \*/

delay\_ms(i);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_5,GPIO\_PIN\_RESET); /\* PB5置0 \*/

delay\_ms(i);

HAL\_GPIO\_WritePin(GPIOE,GPIO\_PIN\_5,GPIO\_PIN\_SET); /\* PE5置1 \*/

delay\_ms(i);

HAL\_GPIO\_WritePin(GPIOE,GPIO\_PIN\_5,GPIO\_PIN\_RESET); /\* PE5置0 \*/

delay\_ms(i);

}

}

/\*\*

\* @brief 初始化LED相关IO口, 并使能时钟

\* @param 无

\* @retval 无

\*/

void led\_init(void)

{

GPIO\_InitTypeDef gpio\_initstruct;

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE(); /\* IO口PB时钟使能 \*/

\_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE();

gpio\_initstruct.Pin = GPIO\_PIN\_5; /\* LED0引脚 \*/

gpio\_initstruct.Mode = GPIO\_MODE\_OUTPUT\_PP; /\* 推挽输出 \*/

gpio\_initstruct.Speed = GPIO\_SPEED\_FREQ\_HIGH; /\* 高速 \*/

HAL\_GPIO\_Init(GPIOB, &gpio\_initstruct); /\* 初始化LED0引脚 \*/

gpio\_initstruct.Pin = GPIO\_PIN\_5; /\* LED1引脚 \*/

gpio\_initstruct.Mode = GPIO\_MODE\_OUTPUT\_PP; /\* 推挽输出 \*/

gpio\_initstruct.Speed = GPIO\_SPEED\_FREQ\_HIGH; /\* 高速 \*/

HAL\_GPIO\_Init(GPIOE, &gpio\_initstruct); /\* 初始化LED1引脚 \*/

}

**[3.3](#_Toc7846) 使用库函数控制蜂鸣器**

需要使用GPIO\_Init函数来初始化蜂鸣器对应的引脚，然后使用GPIO\_SetBits或GPIO\_ResetBits函数来控制蜂鸣器的高低电平。

打开原理图，找到蜂鸣器对应的引脚，例如PB8。

使能GPIO端口，RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE)。

配置GPIO\_InitStructure结构体，设置引脚模式为推挽输出，速度为72MHz。

调用GPIO\_Init函数，传入GPIOB和GPIO\_InitStructure参数，初始化PB8引脚。

使用GPIO\_SetBits或GPIO\_ResetBits函数，传入GPIOB和GPIO\_Pin\_8参数，设置PB8引脚的高低电平。

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/delay/delay.h"

void beep\_init(void);

void led\_init(void);

int i = 200 ;

int main(void)

{

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟,72M \*/

delay\_init(72); /\* 初始化延时函数 \*/

led\_init(); /\* 初始化LED \*/

beep\_init(); /\* 初始化蜂鸣器 \*/

while (1)

{

i += 100 ;

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_5, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_5, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET);

delay\_ms(i);

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_5, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_5, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_SET);

delay\_ms(i);

if (i > 800 )

{

i = 200 ;

}

}

}

/\* 蜂鸣器\*/

void beep\_init(void)

{

GPIO\_InitTypeDef gpio\_init\_struct;

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE(); /\* BEEP时钟使能 \*/

gpio\_init\_struct.Pin = GPIO\_PIN\_8; /\* 蜂鸣器引脚 \*/

gpio\_init\_struct.Mode = GPIO\_MODE\_OUTPUT\_PP; /\* 推挽输出 \*/

gpio\_init\_struct.Pull = GPIO\_PULLUP; /\* 上拉 \*/

gpio\_init\_struct.Speed = GPIO\_SPEED\_FREQ\_HIGH; /\* 高速 \*/

HAL\_GPIO\_Init(GPIOB, &gpio\_init\_struct); /\* 初始化蜂鸣器引脚 \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET); /\* 关闭蜂鸣器 \*/

}

void led\_init(void)

{

GPIO\_InitTypeDef gpio\_init\_struct;

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE(); /\* LED0时钟使能 \*/

\_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE(); /\* LED1时钟使能 \*/

gpio\_init\_struct.Pin = GPIO\_PIN\_5; /\* LED0引脚 \*/

gpio\_init\_struct.Mode = GPIO\_MODE\_OUTPUT\_PP; /\* 推挽输出 \*/

gpio\_init\_struct.Pull = GPIO\_PULLUP; /\* 上拉 \*/

gpio\_init\_struct.Speed = GPIO\_SPEED\_FREQ\_HIGH; /\* 高速 \*/

HAL\_GPIO\_Init(GPIOB, &gpio\_init\_struct); /\* 初始化LED0引脚 \*/

gpio\_init\_struct.Pin = GPIO\_PIN\_5; /\* LED1引脚 \*/

HAL\_GPIO\_Init(GPIOE, &gpio\_init\_struct); /\* 初始化LED1引脚 \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_5, GPIO\_PIN\_SET); /\* 关闭 LED0 \*/

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_5, GPIO\_PIN\_SET); /\* 关闭 LED1 \*/

}

**3.4 使用库函数检测按键**

需要使用GPIO\_ReadInputDataBit函数来读取按键对应的引脚的电平状态，然后使用延时函数来消除抖动，再次读取电平状态，判断按键是否按下。

打开原理图，找到按键对应的引脚，例如KEY0接在PE3上，低电平有效。

使能GPIOE端口的时钟，RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOE, ENABLE)。

配置GPIO\_InitStructure结构体，设置引脚模式为上拉输入，速度为72MHz。

调用GPIO\_Init函数，传入GPIOE和GPIO\_InitStructure参数，初始化PE3引脚。

调用GPIO\_ReadInputDataBit函数，传入GPIOE和GPIO\_Pin\_3参数，读取PE3引脚的电平状态。

如果PE3为低电平，说明按键按下，延时一小段时间（10~30ms），再次读取PE3的电平状态。

如果PE3仍为低电平，说明按键有效，执行相应的操作。

如果PE3为高电平，说明按键松开或无效，继续等待下一次按键。

#include "./stm32f1xx\_it.h"

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/delay/delay.h"

#include "./BSP/LED/led.h"

#include "./BSP/BEEP/beep.h"

#include "./BSP/KEY/key.h"

int main(void)

{

uint8\_t key;

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟, 72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

led\_init(); /\* 初始化LED \*/

beep\_init(); /\* 初始化蜂鸣器 \*/

key\_init(); /\* 初始化按键 \*/

LED0(0); /\* 先点亮LED0 \*/

while(1)

{

key = key\_scan(0); /\* 得到键值 \*/

if (key)

{

switch (key)

{

case WKUP\_PRES: /\* 控制蜂鸣器 \*/

BEEP\_TOGGLE(); /\* BEEP状态取反 \*/

break;

case KEY6\_PRES:

BEEP\_TOGGLE();

LED0\_TOGGLE(); /\* LED0状态取反 \*/

LED1\_TOGGLE();

break;

case KEY1\_PRES: /\* 控制LED1(GREEN)翻转 \*/

LED1\_TOGGLE(); /\* LED1状态取反 \*/

break;

case KEY0\_PRES: /\* 同时控制LED0, LED1翻转 \*/

LED0\_TOGGLE(); /\* LED0状态取反 \*/

LED1\_TOGGLE(); /\* LED1状态取反 \*/

break;

}

}

else

{

delay\_ms(10);

}

}

}

**[4.](#_Toc8630) STM32常用功能的库函数实现**

**[4.1 使用](#_Toc30061)STM32的串口收发数据**

需要使用USART\_Init函数来初始化串口的波特率、数据位、停止位等参数，然后使用USART\_ITConfig函数来使能串口的发送或接收中断，再使用USART\_SendData或USART\_ReceiveData函数来发送或接收数据。

打开原理图，找到串口对应的引脚，例如USART1对应PA9和PA10。

使能GPIOA和USART1的时钟，例如RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_USART1, ENABLE)。

配置GPIO\_InitStructure结构体，设置PA9为复用推挽输出，设置PA10为浮空输入。

调用GPIO\_Init函数，传入GPIOA和GPIO\_InitStructure参数，初始化PA9和PA10引脚。

配置USART\_InitStructure结构体，设置波特率为115200，数据位为8位，停止位为1位，无奇偶校验，无硬件流控。

调用USART\_Init函数，传入USART1和USART\_InitStructure参数，初始化USART1。

调用USART\_ITConfig函数，传入USART1和USART\_IT\_RXNE参数，使能串口接收中断。

调用NVIC\_Init函数，传入NVIC\_InitStructure参数，配置中断优先级和使能中断。

在中断服务函数中，判断是否有接收中断产生，如果有，则调用USART\_ReceiveData函数读取接收到的数据，并存入缓冲区或进行其他处理。

在主函数中或其他地方，调用USART\_SendData函数发送数据，并等待发送完成标志。

#include "./stm32f1xx\_it.h"

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/usart/usart.h"

#include "./SYSTEM/delay/delay.h"

#include "./BSP/LED/led.h"

void beep\_init(void);

int main(void)

{

uint8\_t len;

uint16\_t times = 0;

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟为72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

usart\_init(115200); /\* 串口初始化为115200 \*/

led\_init(); /\* 初始化LED \*/

beep\_init(); /\* 初始化蜂鸣器 \*/

printf("\r\n请发送消息:\r\n");

while (1)

{

if (g\_usart\_rx\_sta & 0x8000) /\* 接收到了数据? \*/

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET); //

//delay\_ms(100);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_SET);

delay\_ms(200);

len = g\_usart\_rx\_sta & 0x3fff; /\* 得到此次接收到的数据长度 \*/

printf("\r\n您发送的消息为:\r\n");

HAL\_UART\_Transmit(&g\_uart1\_handle,(uint8\_t\*)g\_usart\_rx\_buf, len, 1000); /\* 发送接收到的数据 \*/

while(\_\_HAL\_UART\_GET\_FLAG(&g\_uart1\_handle,UART\_FLAG\_TC) != SET); /\* 等待发送结束 \*/

printf("\r\n\r\n"); /\* 插入换行 \*/

g\_usart\_rx\_sta = 0;

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET);

delay\_ms(100);

times++;

if (times % 1000 == 0)

{

printf("\r\n正点原子 STM32开发板 串口实验\r\n");

printf("正点原子@ALIENTEK\r\n\r\n\r\n");

}

if (times % 200 == 0) printf("请输入数据,以回车键结束\r\n");

if (times % 30 == 0) LED0\_TOGGLE(); /\* 闪烁LED,提示系统正在运行. \*/

delay\_ms(10);

}

}

}

void beep\_init(void)

{

GPIO\_InitTypeDef gpio\_init\_struct;

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE(); /\* BEEP时钟使能 \*/

gpio\_init\_struct.Pin = GPIO\_PIN\_8; /\* 蜂鸣器引脚 \*/

gpio\_init\_struct.Mode = GPIO\_MODE\_OUTPUT\_PP; /\* 推挽输出 \*/

gpio\_init\_struct.Pull = GPIO\_PULLUP; /\* 上拉 \*/

gpio\_init\_struct.Speed = GPIO\_SPEED\_FREQ\_HIGH; /\* 高速 \*/

HAL\_GPIO\_Init(GPIOB, &gpio\_init\_struct); /\* 初始化蜂鸣器引脚 \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET); /\* 关闭蜂鸣器 \*/

}

[4.2 使用](#_Toc19527)STM32控制彩色LCD显示图形和英文字符

需要使用SPI协议来与LCD模块通信，然后使用LCD驱动函数来初始化LCD模块，设置显示区域，发送显示数据等。

打开原理图，找到LCD模块对应的引脚，例如CS、Reset、DC、SDI、SCK、LED等。

使能GPIO端口的时钟，例如RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE)。

配置GPIO\_InitStructure结构体，设置CS、Reset、DC、SDI、SCK为复用推挽输出，设置LED为普通推挽输出。

调用GPIO\_Init函数，传入GPIOB和GPIO\_InitStructure参数，初始化各个引脚。

配置SPI\_InitStructure结构体，设置SPI的工作模式为主机模式，时钟极性为低电平空闲，时钟相位为第二个跳变沿采样数据，数据位为8位，波特率预分频为256等。

调用SPI\_Init函数，传入SPI2和SPI\_InitStructure参数，初始化SPI2。

调用SPI\_Cmd函数，传入SPI2和ENABLE参数，使能SPI2。

调用LCD\_Init函数，传入LCD的驱动芯片型号，初始化LCD模块。

调用LCD\_Clear函数，传入一个颜色值，例如WHITE等，清屏并填充背景色。

调用LCD\_SetWindows函数，传入显示区域的左上角和右下角的坐标值，例如(0,0)和(239,319)等，设置显示区域。

调用LCD\_DrawPoint函数或LCD\_WriteRAM\_Prepare函数和LCD\_WriteRAM函数，传入像素点的颜色值或数组指针等，向显示区域写入显示数据。

调用LCD\_ShowString函数或LCD\_ShowChar函数或LCD\_ShowNum函数等，传入字符串或字符或数字的坐标值和颜色值等，显示英文字符。

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/usart/usart.h"

#include "./SYSTEM/delay/delay.h"

#include "./BSP/LED/led.h"

#include "./BSP/LCD/lcd.h"

#include "./QQHead.h"

#include "./snowmountain.h"

int main(void)

{

uint8\_t x = 0;

uint8\_t lcd\_id[12];

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟, 72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

usart\_init(115200); /\* 串口初始化为115200 \*/

led\_init(); /\* 初始化LED \*/

lcd\_init(); /\* 初始化LCD \*/

g\_point\_color = RED;

sprintf((char \*)lcd\_id, "LCD ID:%04X", lcddev.id); /\* 将LCD ID打印到lcd\_id数组 \*/

//2.8MCU屏为 9341

while (1)

{ //lcd\_display\_dir(0); /\*竖屏;\*/

switch (x)

{

case 0:

lcd\_clear(WHITE);

break;

case 1:

lcd\_clear(BLACK);

break;

case 2:

lcd\_clear(BLUE);

break;

case 3:

lcd\_clear(RED);

break;

case 4:

lcd\_clear(MAGENTA);

break;

case 5:

lcd\_clear(GREEN);

break;

case 6:

lcd\_clear(CYAN);

break;

case 7:

lcd\_clear(YELLOW);

break;

case 8:

lcd\_clear(BRRED);

break;

case 9:

lcd\_clear(GRAY);

break;

case 10:

lcd\_clear(LGRAY);

break;

case 11:

lcd\_clear(BROWN);

break;

}

lcd\_show\_string(10, 40, 240, 32, 32, "STM32", RED); //240\*320

lcd\_show\_string(10, 80, 240, 24, 24, "TFTLCD TEST", RED);

lcd\_show\_string(10, 110, 240, 16, 16, "ATOM@ALIENTEK", RED);

lcd\_show\_string(10, 130, 240, 16, 16, (char \*)lcd\_id, RED); /\* 显示LCD ID \*/

int i = 0, j = 160, k = 24, l = 184;

while (l<320)

{

lcd\_draw\_rectangle(i, j, k, l, RED);

lcd\_fill(i+2, j+2, k+2, l+2, MAGENTA)

i += 24;

k += 24;

if (k > 240)

{

j += 24;

l += 24;

i = 0;

k = 24;

}

}

x++;

lcd\_fill(20,180, 220, 280, YELLOW);

lcd\_fill\_circle(100, 230, 20,MAGENTA);

if (x == 12)

x = 0;

delay\_ms(3000);

}

}

**4.3 [使用](#_Toc19527)STM32控制彩色LCD显示图片和中文字符**

打开原理图，找到LCD模块对应的引脚，例如CS、Reset、DC、SDI、SCK、LED等。

使能GPIO端口的时钟，例如RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE)。

配置GPIO\_InitStructure结构体，设置CS、Reset、DC、SDI、SCK为复用推挽输出，设置LED为普通推挽输出。

调用GPIO\_Init函数，传入GPIOB和GPIO\_InitStructure参数，初始化各个引脚。

配置SPI\_InitStructure结构体，设置SPI的工作模式为主机模式，时钟极性为低电平空闲，时钟相位为第二个跳变沿采样数据，数据位为8位，波特率预分频为256等。

调用SPI\_Init函数，传入SPI2和SPI\_InitStructure参数，初始化SPI2。

调用SPI\_Cmd函数，传入SPI2和ENABLE参数，使能SPI2。

调用LCD\_Init函数，传入LCD的驱动芯片型号，例如ILI9341等，初始化LCD模块。

调用LCD\_Clear函数，传入一个颜色值，例如WHITE等，清屏并填充背景色。

调用LCD\_SetWindows函数，传入显示区域的左上角和右下角的坐标值，例如(0,0)和(239,319)等，设置显示区域。

调用LCD\_DrawPoint函数或LCD\_WriteRAM\_Prepare函数和LCD\_WriteRAM函数，传入像素点的颜色值或数组指针等，向显示区域写入显示数据。

使用字模生成软件，例如PCtoLCD2002等，打开GBK字符集文件，并设置好字体大小、扫描方式、输出格式等参数。

将需要显示的中文字符复制到软件中，并生成对应的像素数据数组文件。

将像素数据数组文件转换为二进制文件，并烧录到FLASH或SD卡等外部存储器中。

在主程序中或其他地方，调用LCD\_ShowChinese函数或类似的函数，传入中文字符的坐标值、颜色值、内码值、存储器地址等参数，从外部存储器读取像素数据并显示中文字符。

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/usart/usart.h"

#include "./SYSTEM/delay/delay.h"

#include "./BSP/LED/led.h"

#include "./BSP/LCD/lcd.h"

//#include "./QQHead.h"

#include "./zhengjian.c"

//#include "./guanxi.c"

//#include "./zhangtao.c"

#include "./snowmountain.h"

int main(void)

{

uint8\_t lcd\_id[12];

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟, 72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

usart\_init(115200); /\* 串口初始化为115200 \*/

led\_init(); /\* 初始化LED \*/

lcd\_init(); /\* 初始化LCD \*/

g\_point\_color = RED;

sprintf((char \*)lcd\_id, "LCD ID:%04X", lcddev.id); /\* 将LCD ID打印到lcd\_id数组 \*/

//2.8MCU屏为 9341

{ lcd\_display\_dir(0); //设置LCD显示方向 0正常竖屏，1逆时针90度，2旋转180度，3顺时针90度

//lcd\_show\_string(10, 6, 240, 32, 32, "STM32", RED); // 以坐标(10,6)为左上角，在240\*32矩形范围内显示 32\*32点阵英文

//lcd\_show\_string(10, 40, 240, 24, 24, "TFTLCD TEST", RED); // 以坐标(10,40)为左上角，在240\*24矩形范围内显示24\*24点阵英文

//lcd\_show\_string(10, 70, 240, 16, 16, (char \*)lcd\_id, RED); /\* 显示显示LCD ID 以坐标(10,70)为左上角，在240\*16矩形范围内显示16\*16点阵英文 \*/

//lcd\_fill(20,140, 100, 200, YELLOW);//画实心矩形 左上角坐标(20,140) ,右下角坐标(100, 200) 填充黄色

//lcd\_draw\_rectangle(120, 140, 200, 200, YELLOW);/\* 画空心矩形 边框黄色\*/

// lcd\_fill\_circle(250, 40, 30,MAGENTA); //画实心圆 圆心(250, 40) 半径30 紫色填充

//lcd\_draw\_circle(250, 120, 30,BLUE); //画空心圆 边框蓝色

//g\_back\_color= YELLOW; //设置背景色黄色

//lcd\_show\_string(28, 162, 240, 16, 12, "zhangsan01", RED); //显示姓名zhangsan01 12\*12点阵英文

//g\_back\_color= BROWN;

//lcd\_show\_num(136,162,123456, 6,16,RED); //以坐标(136,162)为左上角显示数字： 123456 共显示6位数， 16\*16点阵英文

//LED0\_TOGGLE(); /\*红灯闪烁\*/

//delay\_ms(1500);

//lcd\_show\_pic(100, 140,40, 40, gImage\_qq); //以坐标(100, 140)为左上角 显示40\*40图片

//delay\_ms(1500);

//显示自己的照片

lcd\_show\_pic(60, 75, 120, 179, zhengjian);

//delay\_ms(5000);

//lcd\_display\_dir(0); /\*横屏;\*/

//lcd\_show\_pic(0, 0,320, 240, gImage\_SnowMountain); //以坐标(0, 0)为左上角 显示320\*240图片

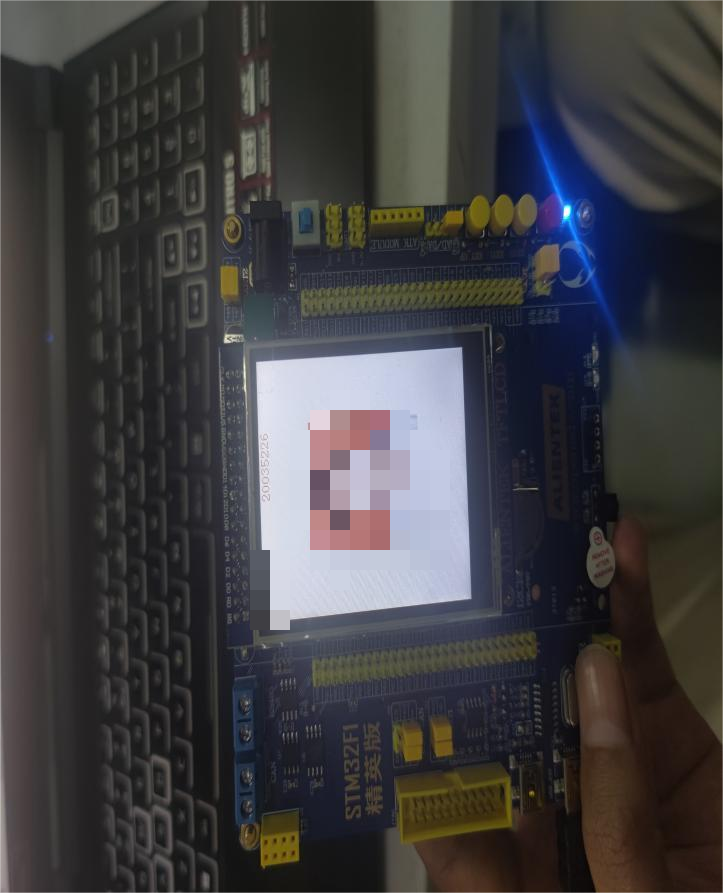
//GUI\_DrawCN\_Str(80,0,16,"深圳市电子技术",1,RED); // 以坐标(8,0)为左上角，显示16\*16点阵 中文

GUI\_DrawCN\_Str(0,0,16,"王云豪",1,BLACK);

lcd\_show\_num(110,0,20035226, 8,16,RED)

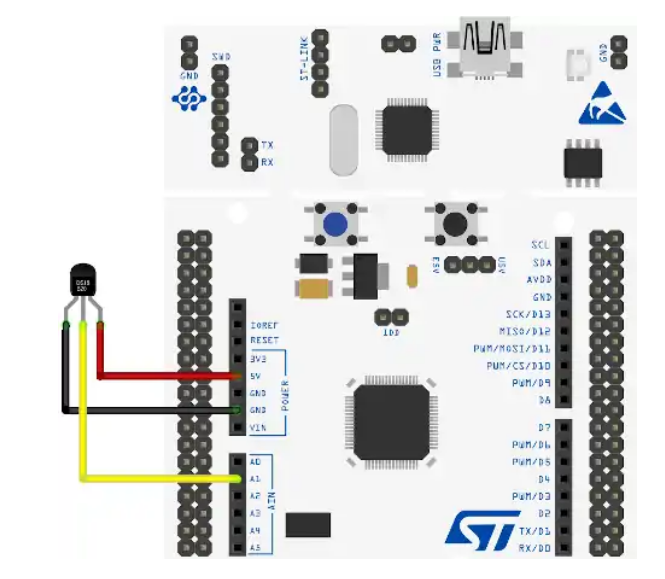
} while (1);

}



**[5.](#_Toc9491) 基于STM32的单总线温度检测系统设计**

5.1 系统原理电路图



5.2 系统工作原理

需要使用STM32的单总线接口来与DS18B20温度传感器通信，然后使用温度转换函数来获取温度值。

打开原理图，找到STM32的单总线引脚，例如PA0等。

使能GPIO端口的时钟，例如RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE)。

配置GPIO\_InitStructure结构体，设置PA0为开漏输出模式。

调用GPIO\_Init函数，传入GPIOA和GPIO\_InitStructure参数，初始化PA0引脚。

调用DS18B20\_Init函数，传入PA0引脚的地址，初始化DS18B20传感器。

调用DS18B20\_ReadId函数，传入PA0引脚的地址和一个存储ID的数组指针，读取DS18B20的ID号。

调用DS18B20\_Start函数，传入PA0引脚的地址和DS18B20的ID号，启动温度转换。

调用DS18B20\_ReadTemp函数，传入PA0引脚的地址和DS18B20的ID号，读取温度值。

5.3 软件代码设计

#include "./SYSTEM/sys/sys.h"

#include "./SYSTEM/usart/usart.h"

#include "./SYSTEM/delay/delay.h"

#include "./USMART/usmart.h"

#include "./BSP/LED/led.h"

#include "./BSP/LCD/lcd.h"

#include "./BSP/DS18B20/ds18b20.h"

int main(void)

{

uint8\_t t = 0;

short temperature;

HAL\_Init(); /\* 初始化HAL库 \*/

sys\_stm32\_clock\_init(RCC\_PLL\_MUL9); /\* 设置时钟, 72Mhz \*/

delay\_init(72); /\* 延时初始化 \*/

usart\_init(115200); /\* 串口初始化为115200 \*/

led\_init(); /\* 初始化LED \*/

lcd\_init(); /\* 初始化LCD 白色清屏\*/

{

lcd\_display\_dir(0); //设置LCD显示方向 0正常竖屏，1逆时针90度，2旋转180度，3顺时针90度

g\_back\_color= YELLOW; //设置背景色黄色

lcd\_show\_string(30, 50, 200, 16, 16, "STM32", RED); //默认白色背景

GUI\_DrawCN\_Str(5,5,16,"王云豪",1,BLACK);

lcd\_show\_num(120,5,20035226, 8,16,RED);

lcd\_show\_string(30, 70, 200, 16, 16, "DS18B20", RED);

GUI\_DrawCN\_Str(100,70,16,"测试",1,BLACK);

lcd\_show\_string(30, 90, 200, 16, 16, "ATOM@ALIENTEK", RED);

}

while (ds18b20\_init()) /\* DS18B20初始化 \*/

{

lcd\_show\_string(30, 110, 200, 16, 16, "DS18B20", RED);

GUI\_DrawCN\_Str(100,110,16,"未找到",1,BLACK);

delay\_ms(200);

lcd\_fill(30, 110, 239, 130 + 16, WHITE);

//从x=30开始将两行文字直至屏幕最右侧区域填充白色

delay\_ms(200);

}

lcd\_show\_string(30, 110, 200, 16, 16, "DS18B20", RED);

GUI\_DrawCN\_Str(100,110,16,"准备就绪",1,BLACK);

//lcd\_show\_string(30, 130, 200, 16, 16, "Temp: . C", BLUE);

GUI\_DrawCN\_Str(30,130,16,"温度",1,BLUE);

lcd\_show\_string(62, 130, 200, 16, 16, ":", BLUE);

lcd\_show\_string(94, 130, 200, 16, 16, ". C", BLUE);

while (1)

{

if (t % 10 == 0) /\* 每100ms读取一次 \*/

{

temperature = ds18b20\_get\_temperature(); //温度已经放大10倍

if (temperature < 0)

{

lcd\_show\_char(30 + 40, 130, '-', 16, 0, BLUE); /\* 显示负号 \*/

temperature = -temperature; /\* 转为正数 \*/

}

else

{

lcd\_show\_char(30 + 40, 130, ' ', 16, 0, BLUE); /\* 去掉负号 \*/

}

lcd\_show\_num(30 + 40 + 8, 130, temperature / 10, 2, 16, BLUE); /\* 显示正数部分 \*/

lcd\_show\_num(30 + 40 + 32, 130, temperature % 10, 1, 16, BLUE); /\* 显示小数部分 \*/

}

delay\_ms(10);

t++;

if (t == 20)

{

t = 0;

LED0\_TOGGLE(); /\* LED0闪烁 \*/

}

}

}

5.4 显示界面设计



在图中STM32F103ZET6使用DS18B20温度传感器可以显示中文的提示，及温度的显示。

**6.总结和展望**

本次的课程设计让我掌握了STM32F103ZET6控制led、蜂鸣器、使用按键、串口通信、lcd屏幕使用、外设的使用如DS18B20温度传感器的使用等，激发了我对硬件学习的兴趣，掌握了对硬件程序烧录的流程，对未来学习更多的硬件设备打下了基础和学习使用的思路。

**7.参考文献**

[1]杨旻荟,田澄睿,顾浚哲.基于STM32F103ZET6的电路特性测试仪[J].电子测试,2022,36(12):25-27+83.DOI:10.16520/j.cnki.1000-8519.2022.12.015.

[2]王晓东,马长银,刘沛灵等.浅谈DS18B20温度传感器的数据获取及LED显示[J].数码世界,2018,No.157(11):286-288.

[3]王云飞.DS18B20温度传感器的应用设计[J].电子世界,2014,No.450(12):355.

[4]王世礼,杨彪,陈正标.基于STM32单片机的多点温度测量系统设计[J].价值工程,2016,35(26):196-198.DOI:10.14018/j.cnki.cn13-1085/n.2016.26.076.