I2C_LCD 用户手册

中文版 v1.0

2015-4-8

Sparking Work Space

Shenzhen, China

第 1 章 前言

I2C_LCD12864 是一款简单易用的显示模块，以下简称 I2C_LCD。

I2C_LCD 的分辨率为 128X64，可支持黑、白双色显示。最多可显示 168 个字符，最大可显示 128x64 像素点的双色图片。

I2C_LCD 具有独立的控制器，大部分复杂图形运算在 I2C_LCD 控制器中完成，可以减轻用户控制器（Arduino、MCU）的运算负担。

我们为用户提供了全功能的 Arduino 库，在使用库函数的情况下，只需要简单的几行程序，便可以实现较为复杂的字符、图形等显示功能。

1.1 如何使用本文档

在本文档中，讲述了 Arduino 库中 API 函数的使用方法，并提供相关的示例代码帮助用户使用 I2C_LCD。

用户可以首先通过每章的示例工程来学习如何使用 I2C_LCD。示例工程中已经包含了详细的程序注释和函数说明，当使用示例工程遇到困难时，可通过查阅每章第二节的 API 函数详细说明，来了解相关 API 函数的使用方法。

当用户在学习、使用的过程中遇到问题时，可以通过关键字搜索本文档来寻找解决问题的方法。

1.2 I2C_LCD 相关参数

1.2.1 产品特点

I2C_LCD 是一款简单易用的显示模块。为您提供一个高效率的双色 UI 界面设计途径。

主要特点：

1. 通过 I2C 串行同步通讯接口控制， 只占用两个 I/O 口；
2. 支持标准 I2C 模式（100Kbit/s）与快速 I2C 模式（400Kbit/s）；

3. 兼容多种通讯逻辑电平：2.8~5VDC；
4. 全功能 Arduino 库支持，仅仅几行代码即可完成显示；
5. 内部集成 7 种 ASCII 字体，5 种图形函数；
6. 提供专用的图片数据转换软件（Bitmap Converter）；
7. 大部分运算由 I2C_LCD 独立控制器处理，节省用户控制器资源；
8. 支持光标功能，可设置 16 级光标闪烁频率；
9. 支持 128 级背光亮度调整；
10. 支持 64 级屏幕对比度调整；
11. 支持通过程序修改写器件地址；
12. 最多支持 127 个 I2C_LCD 同时工作；
13. 调试代码时，它可以代替串口监视器监控程序运行状态；
14. 提供两种异常恢复方式：复位、恢复出厂设置；
15. 兼容 Grove 接口和 4Pin-100mil 间距插针接口（Grove 插座下方）；
16. 4 个对称的固定孔设计，方便用户安装；
17. 独特的中国风外形设计；

1.2.2 电气特性

符号	说明	最小值	典型值	最大值	单位
5V	电源输入	4.5	5	5.5	V
SCL\SDA	I2C 总线	2.8	5	5.5	V
I_{in}	工作电流	13	20	37.1	mA

1.3 使用 I2C_LCD 所需要的资源

硬件要求：任何 Arduino 兼容板、任何具有 I2C 通讯功能的控制器；

电压要求：电源电压 5V，通讯逻辑电压 2.8~5V；

ROM 要求：大于 4Kb；

RAM 需求：大于 512Byte；

1.4 屏幕与坐标

屏幕由能够被单独控制的许多点组成，这些被称作像素。用户通过程序能够在任何指定像素上绘制。

水平刻度被称作 X 轴，而垂直刻度被称作 Y 轴。一个二维坐标用 X 轴和 Y 轴坐标表示，即值(X,Y)。在程序中需要同时用到 X 和 Y 坐标时，X 坐标总在前面。I2C_LCD 显示屏的左上角为默认的坐标（0,0）。正的 X 值方向被总是向右；正的 Y 值方向总是向下。

图 1.4.1 说明了 I2C_LCD 的坐标系和 X 轴和 Y 轴的方向。所有传递到一个 API 函数的坐标总是以 1 像素为 1 刻度指定。

图 1.4.1 坐标示意图

第 2 章 显示字符

I2C_LCD 内置了 7 种 ASCII 字体：Font_6x8、Font_6x12、Font_8x16_1、 Font_8x16_2、Font_10x20、Font_12x24、Font_16x32，满足您的个性化需求。

在 I2C_LCD 的 Arduino 库的支持下，实现字符显示功能会变得非常简单，只需要一行程序就可以实现字符显示。

本章将介绍如何显示不同大小、不同颜色的字符。

备注：Font_10x20 表示宽度为 10 个像素，高度为 20 个像素的字体。

2.1 I2C_LCD 支持的字符

I2C_LCD 支持美国标准信息交换标准码(ASCII)中的所有字符。按 8 位字符进行编译，允许最大为 126 的不同的字符代码。在 I2C_LCD 中为了方便用户，将 28~31 定义为 4 个方向的箭头“← → ↑ ↓”。

I2C_LCD 支持的所有 ASCII 字符、扩充字符如下表所示：

I2C_LCD 支持字符表：

DEC	HEX	字符	DEC	HEX	字符	DEC	HEX	字符	DEC	HEX	字符
28	0x1C	←	53	0x35	5	78	0x4E	N	103	0x67	g
29	0x1D	→	54	0x36	6	79	0x4F	O	104	0x68	h
30	0x1E	↑	55	0x37	7	80	0x50	P	105	0x69	i
31	0x1F	↓	56	0x38	8	81	0x51	Q	106	0x6A	j
32	0x20		57	0x39	9	82	0x52	R	107	0x6B	k
33	0x21	!	58	0x3A	:	83	0x53	S	108	0x6C	l
34	0x22	"	59	0x3B	;	84	0x54	T	109	0x6D	m
35	0x23	#	60	0x3C	<	85	0x55	U	110	0x6E	n
36	0x24	$	61	0x3D	=	86	0x56	V	111	0x6F	o
37	0x25	%	62	0x3E	>	87	0x57	W	112	0x70	p
38	0x26	&	63	0x3F	?	88	0x58	X	113	0x71	q
39	0x27	'	64	0x40	@	89	0x59	Y	114	0x72	r
40	0x28	(	65	0x41	A	90	0x5A	Z	115	0x73	s
41	0x29	)	66	0x42	B	91	0x5B	[	116	0x74	t
42	0x2A	*	67	0x43	C	92	0x5C	\	117	0x75	u
43	0x2B	+	68	0x44	D	93	0x5D	]	118	0x76	v
44	0x2C	,	69	0x45	E	94	0x5E	^	119	0x77	w
45	0x2D	-	70	0x46	F	95	0x5F	_	120	0x78	x
46	0x2E	.	71	0x47	G	96	0x60	`	121	0x79	y
47	0x2F	/	72	0x48	H	97	0x61	a	122	0x7A	z
48	0x30	0	73	0x49	I	98	0x62	b	123	0x7B	{
49	0x31	1	74	0x4A	J	99	0x63	c	124	0x7C	|
50	0x32	2	75	0x4B	K	100	0x64	d	125	0x7D	}
51	0x33	3	76	0x4C	L	101	0x65	e	126	0x7E	~
52	0x34	4	77	0x4D	M	102	0x66	f			

2.2 字符显示示例工程

2.2.1 字体与颜色设置

2.2.1.1 示例程序 1

目的：演示如何显示不同字体、不同颜色的字符。

工程目录：UserManual\DemoCode\Sec_2211_Text

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16，将字符地址更新方式设为 FM_ANL_AAA 方式，将字符显示模式设为 BLACK _BAC；

在（0,10）处显示字符串“Sparking”；

延迟 2s；

将字体更改为 Font_10x20，将字符显示模式改为 WHITE _BAC，在（0,30）处显示字符串“Sparking”。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);    //显示字符串
    delay(2000);            //延迟 2s

    //10X20 像素字体，自动换行，白色字符-黑色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, WHITE_BAC);
    LCD.DispStringAt("Sparking", 0, 30);    //显示字符串

    while(1);               //使程序停止运行
}
```

运行结果：

2.2.2 字符背景色设置

2.2.2.1 示例程序 1

目的：演示字符背景色（有或无）设置。

工程目录：UserManual\DemoCode\Sec_2221_Text

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_10x20，将字符地址更新方式设为 FM_ANL_AAA 方式，将字符显示模式设为 BLACK_BAC；

在（0,10）处显示字符串“Sparking”；

延迟 2s；

在（0,20）处显示字符串“YES_BAC”；

延迟 2s；

将字符显示模式更改为 BLACK_NO_BAC；

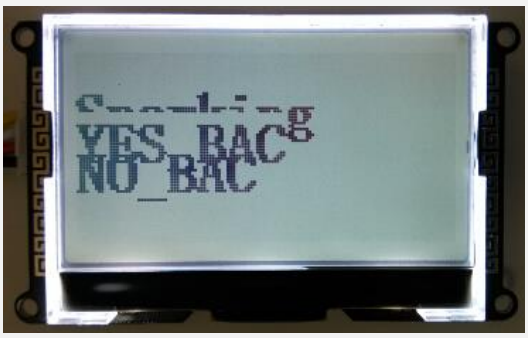
在（0,27）处显示字符串“NO_BAC”；

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>

I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51
```

```
void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //10X20 像素字体，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);    //显示字符串
    delay(2000);            //延迟 2s

    LCD.DispStringAt("YES_BAC", 0, 20);     //显示字符串
    //"Sparking"字符串的下半部分会被当前字符的白色背景擦除
    delay(2000);            //延迟 2s

    //10X20 像素字体，自动换行，黑色字符-无背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_NO_BAC);
    LCD.DispStringAt("NO_BAC", 0, 27);      //显示字符串
    //当前字符无背景色，"YES_BAC"字符串的下半部分将与当前字符叠加

    while(1); //使程序停止运行
}
```

运行结果：



2.2.3 字符地址累加方式设置

2.2.3.1 示例程序 1

目的：演示字符写入两种换行方式的区别（自动换行、手动换行）。

工程目录：UserManual\DemoCode\Sec_2231_Text

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_6x8，将字符地址更新方式设为 FM_ANL_AAA 方式，将字符显示模式设为 BLACK_BAC；

在（108,10）处显示字符串“Sparking”；

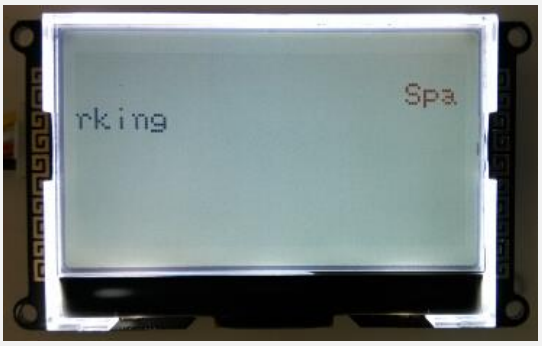
延迟 3s；

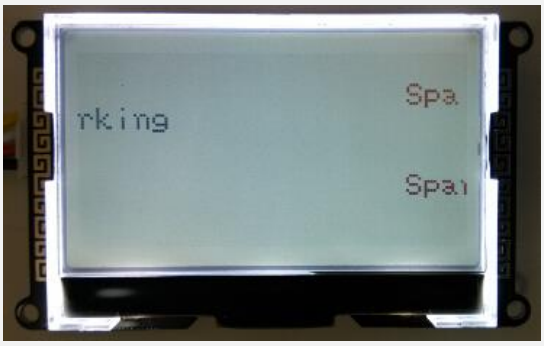
将字符地址更新方式设为 FM_MNL_AAA 方式，并在（108,40）处显示字符串“Sparking”。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //6X8 像素字体，自动换行，自动换行，黑色字符-白色背景色
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_6x8, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 108, 10);   //显示字符，自动换行
    delay(3000);            //延迟 3s

    //6X8 像素字体，手动换行，自动换行，黑色字符-白色背景色
    //FM_MNL_AAA: FM_ManualNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_6x8, FM_MNL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 108, 40);   //显示字符，不自动换行

    while(1); //使程序停止运行
}
```

运行结果：





2.3 字符显示 API 函数的说明

通过阅读本小节用户可以充分的了解每个字符显示 API 函数的详细信息与调用方法。

下表列出了 I2C_LCD 的 Arduino 库中与文本处理相关的 API 函数。

函数	说明
DispCharAt( )	在指定位置显示单个字符
DispStringAt( )	在指定位置显示多个字符
FontModeConf( )	字体模式配置

2.3.1 DispCharAt ( )

描述：在指定位置显示指定的单个字符。

函数原型：

void DispCharAt(char buf, uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
buf	要写入的字符	美国标准信息交换标准码（ASCII）	ASCII 字符集中的任意字符
x	x 坐标	0~127	x 轴一共有 128 个点，编码 0~127

y	y 坐标	0~63	y 轴一共有 64 个点，编码 0~63

返回值：无

调用范例：在屏幕的（0,0）位置显示‘A’；

```
LCD.DispCharAt(‘A’, 0, 0);
```

2.3.2 DispStringAt ( )

描述：在指定位置显示指定的多个字符。

函数原型：

```
void DispStringAt(char *buf, uint8_t x, uint8_t y)
```

参数	参数说明	可以的取值	取值说明
*buf	要写入的字符串的指针	字符串的数组名或地址	无
x	x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	y 坐标	0~63	y 轴一共有 64 个点，编码 0~63

返回值：无

调用范例：在屏幕的（0,30）位置显示字符串“Sparking”。

```
LCD.DispStringAt("Sparking", 0, 30);
```

2.3.3 FontModeConf( )

描述：配置字体显示模式。

函数原型：

```
void FontModeConf(enum LCD_FontSort font,
                  enum LCD_FontMode mode, enum LCD_CharMode cMode)
```

参数	参数说明	可以的取值	取值说明
font	字体类型	Font_6x8	大小为 6*8 像素的字体，默认值
		Font_6x12	大小为 6*12 像素的字体
		Font_8x16_1	大小为 8*16 像素的字体
		Font_8x16_2	大小为 8*16 像素的字体
		Font_10x20	大小为 10*20 像素的字体
		Font_12x24	大小为 12*24 像素的字体
		Font_16x32	大小为 16*32 像素的字体
mode	字符地址更新模式	FM_ANL_AAA	字符自动换行，地址自动累加，默认值 FM_AutoNewLine_AutoAddrAdd 的缩写
		FM_MNL_AAA	字符手动换行，地址自动累加 FM_ManualNewLine_AutoAddrAdd 的缩写
		FM_MNL_MAA	字符手动换行，地址手动累加 FM_ManualNewLine_ManualAddrAdd 的缩写
cMode	字符显示模式	WHITE_BAC	白色字符，有背景色，默认值
		WHITE_NO_BAC	白色字符，无背景色
		BLACK_BAC	黑色字符，有背景色
		BLACK_NO_BAC	黑色字符，无背景色

返回值：无

调用范例：将字体设为 Font_6x8，将字符地址更新方式设为 FM_ANL_AAA，将字符显示模式设为 BLACK_BAC。

```
LCD.FontModeConf(Font_6x8, FM_ANL_AAA, BLACK_BAC);
```

第 3 章 光标的使用

在很多应用的交互界面中，都需要使用光标来指示当前的焦点位置。为了方便用户快速的搭建交互界面，I2C_LCD 集成了光标功能，并可以通过相关 API 函数配置光标的开关、位置、大小、闪烁频率等属性。

本章将介绍如何配置与使用光标。

3.1 光标使用示例工程

3.1.1 光标的属性配置

3.1.1.1 示例程序 1

目的：演示光标开关、位置、闪烁频率的设置方法。

工程目录：UserManual\DemoCode\Sec_3111_Cusor

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16_1，将字符地址更新方式设为 FM_ANL_AAA 方式，将字符显示模式设为 BLACK _BAC；

在（0,30）位置显示时间字符串“12:00:00”；

延迟 2s；

将光标位置设置为秒的个位；

开启光标，并将闪烁频率等级设为 6；

延迟 5s；

将光标指示位置移动到秒的十位。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("12:00:00", 0, 30);    //显示时钟字符串
    delay(2000);            //延迟 2s

    //将光标指示到秒的个位
    //按照字体宽度计算秒个位的位置:x=8X7=56
    //根据字符的起始位置计算光标起始 y 坐标:y=30
    //按照字体宽度计算光标宽度:width=8
    //按照字体高度计算光标高度:height=16
    //Prototype: void CursorGotoXY(x, y, width, height)
    LCD.CursorGotoXY(56, 30, 8, 16);
    LCD.CursorConf(ON, 6);  //开启光标，闪烁等级 6
    delay(5000);            //延迟 5s

    //将光标指示到秒的十位
    LCD.CursorGotoXY(48, 30, 8, 16);

    delay(5000);            //延迟 5s
    LCD.CursorConf(OFF, 6); //关闭光标

    while(1); //使程序停止运行
}
```

运行结果：

3.2 光标 API 函数的说明

本小节将讲述与光标有关的 API 函数的详细属性以及调用方式。

下表列出了 I2C_LCD 的 Arduino 支持库中与光标相关的 API 函数。

函数	说明
CursorConf( )	光标属性配置
CursorGotoXY( )	光标位置配置

3.2.1 CursorConf ( )

描述：光标配置，设置光标的开关与闪烁频率。

函数原型：

void CursorConf(enum LCD_SwitchState swi, uint8_t freq)

参数	参数说明	可以的取值	取值说明
swi	光标开/关	OFF	关闭光标，默认值
		ON	开启光标
freq	光标闪烁频率	0~15	16 级光标闪烁频率设置

返回值：无

调用范例：将光标显示开关设为“ON”，将光标闪烁频率等级设为 6。

```
LCD.CursorConf(ON, 6);
```

3.2.2 CursorGotoXY ( )

描述：光标配置，设置光标的开关与闪烁频率。

函数原型：

void CursorGotoXY(uint8_t x, uint8_t y, uint8_t width, uint8_t height)

参数	参数说明	可以的取值	取值说明
x	光标的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	光标的起始 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
width	光标 x 轴方向宽度	0~127	x+width 最大取值为 127
height	光标 y 轴方向高度	0~63	y+height 最大取值为 63

返回值：无

调用范例：将光标起始点设为（0,0），将光标宽度设为 20，光标高度设为 8。

```
LCD.CursorGotoXY(0, 0, 20, 8);
```

第 4 章 2D 图形绘制

为了满足用户图形界面的设计需求，I2C_LCD 集成了 2D 图形绘制功能，能够绘制点、线、圆、矩形。实现这些图形的绘制，只需要调用支持库中提供的 API 函数即可。例如画圆，只需要将圆心坐标、半径作为参数传递给 API 函数，即可实现圆形的绘制，用户无需涉及复杂的图形算法，大大的减轻用户开发的难度。

由于图形相关的算法都是在 I2C_LCD 的独立控制器中运行，因此可以大量节省用户控制器的 ROM 与 RAM 资源。

本章介绍如何使用 I2C_LCD 的 Arduino 支持库中 API 函数来快速的绘制 2D 图形。

4.1 2D 图形绘制示例工程

4.1.1 画点与画线

4.1.1.1 示例程序 1

目的：演示画线函数的使用与线条颜色设置方法。

工程目录：UserManual\DemoCode\Sec_4111_Graphic

步骤：

以白色为背景色擦除全屏；

延迟 1s；

从屏幕的（0,20）到（127,20）画一条黑色水平实线；

延迟 2s；

从屏幕的（20,0）到（20,63）画一条黑色垂直实线；

延迟 2s；

在屏幕的（63,50）画一个黑色点。

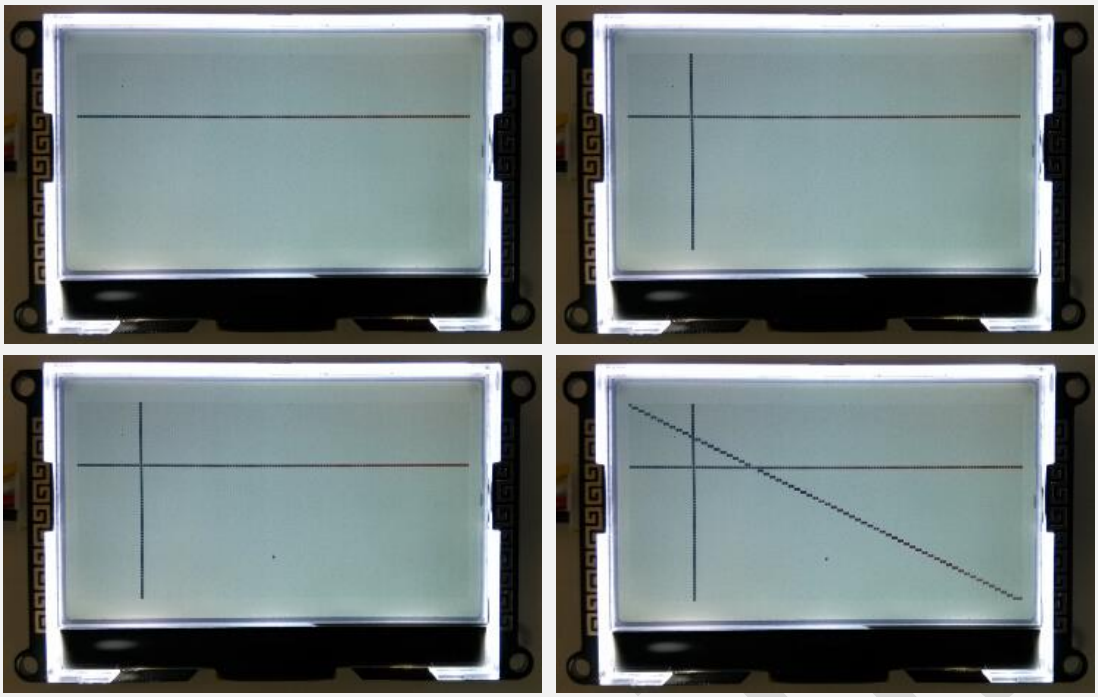
延迟 2s；

从屏幕的（0,0）到（127,63）画一条黑色倾斜实线。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //画水平线，黑色
    //Prototype: void DrawHLineAt(startX, endX, y, color)
    LCD.DrawHLineAt(0, 127, 20, BLACK);
    delay(2000);            //延迟 2s

    //画垂直线，黑色
    //Prototype: void DrawVLineAt(startY, endY, x, color)
    LCD.DrawVLineAt(0, 63, 20, BLACK);
    delay(2000);            //延迟 2s

    //画点，黑色
    //Prototype: void DrawDotAt(x, y, color)
    LCD.DrawDotAt(63, 50, BLACK);
    delay(2000);            //延迟 2s

    //画任意线，黑色
    //Prototype: void DrawLineAt(startX, endX, startY, endY, color)
    LCD.DrawLineAt(0, 127, 0, 63, BLACK);

    while(1); //使程序停止运行
}
```

运行结果：



4.1.1.2 示例程序 2

目的：演示画线函数的使用与颜色更改。

工程目录：UserManual\DemoCode\Sec_4112_Graphic

步骤：

以黑色为背景色擦除全屏；

延迟 1s；

从屏幕的（0,20）到（127,20）画一条白色水平实线；

延迟 2s；

从屏幕的（20,0）到（20,63）画一条白色垂直实线；

延迟 2s；

在屏幕的（63,30）画一个白色点。

延迟 2s；

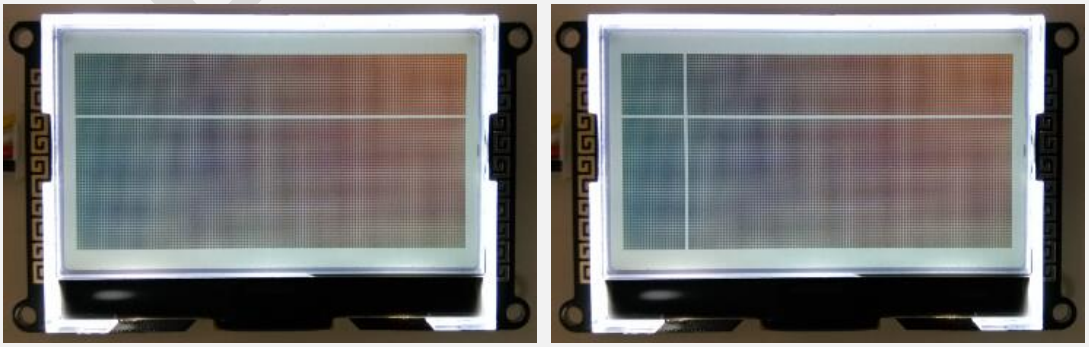
从屏幕的（0,0）到（127,63）画一条白色倾斜实线。

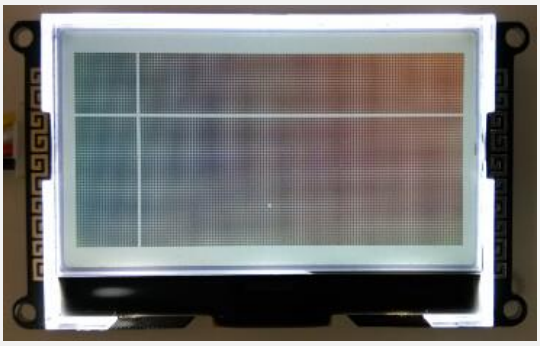
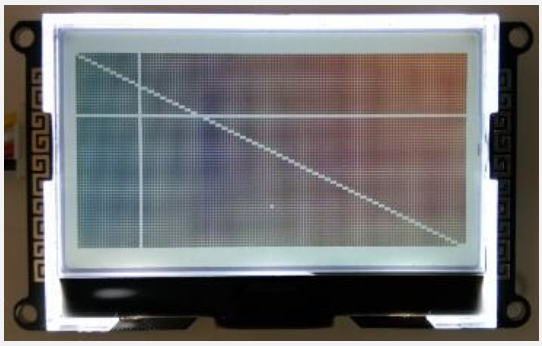
程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
```

```
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(BLACK);    //黑色清屏
    delay(1000);            //延迟 1s

    //画水平线，白色
    //Prototype: void DrawHLineAt(startX, endX, y, color)
    LCD.DrawHLineAt(0, 127, 20, WHITE);
    delay(2000);            //延迟 2s

    //画垂直线，白色
    //Prototype: void DrawVLineAt(startY, endY, x, color)
    LCD.DrawVLineAt(0, 63, 20, WHITE);
    delay(2000);            //延迟 2s

    //画点，白色
    //Prototype: void DrawDotAt(x, y, color)
    LCD.DrawDotAt(63, 50, WHITE);
    delay(2000);            //延迟 2s

    //画任意线，白色
    //Prototype: void DrawLineAt(startX, endX, startY, endY, color)
    LCD.DrawLineAt(0, 127, 0, 63, WHITE);

    while(1); //使程序停止运行
}
```

运行结果：



4.1.2 画矩形与画圆

4.1.2.1 示例程序 1

目的：演示画矩形、圆的方法与颜色设置。

工程目录：UserManual\DemoCode\Sec_4121_Graphic

步骤：

以白色为背景色擦除全屏；

延迟 1s；

在屏幕的（10,15）画一个宽 107，高 33 的黑色填充矩形；

延迟 2s；

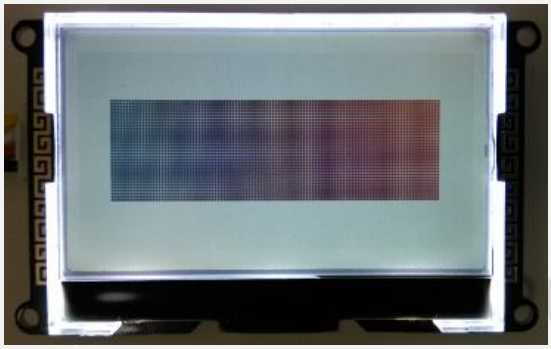
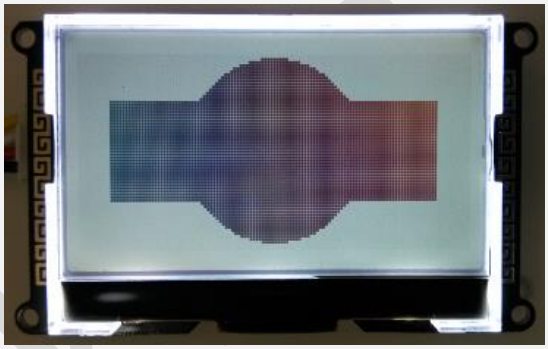
以屏幕的（63,31）为圆心，画一个半径为 30 的黑色填充圆。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s
```

```
    //画矩形，黑色填充
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(10, 15, 107, 33, BLACK_FILL);
    delay(2000);              //延迟 2s

    //画圆，黑色填充
    //Prototype: void DrawCircleAt(x, y, r, mode)
    LCD.DrawCircleAt(63, 31, 30, BLACK_FILL);

    while(1); //使程序停止运行
}
```

运行结果：

4.1.2.2 示例程序 2

目的：演示画矩形、圆方法与颜色改变。

工程目录：UserManual\DemoCode\Sec_4122_Graphic

步骤：

以黑色为背景色擦除全屏；

延迟 1s；

在屏幕的（10,15）画一个宽 107，高 33 的白色填充矩形；

延迟 2s；

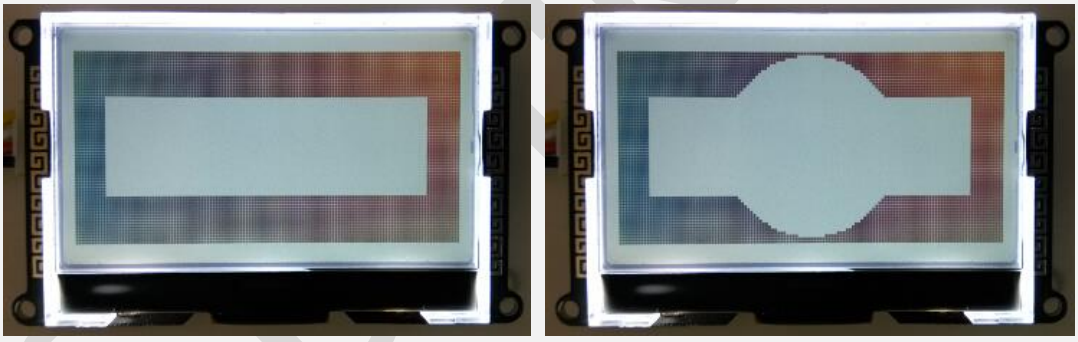
以屏幕的（63,31）为圆心，画一个半径为 30 的白色填充圆。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51
```

```
void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(BLACK);    //黑色清屏
    delay(1000);            //延迟 1s

    //画矩形，白色填充
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(10, 15, 107, 33, WHITE_FILL);
    delay(2000);            //延迟 2s

    //画圆，白色填充
    //Prototype: void DrawCircleAt(x, y, r, mode)
    LCD.DrawCircleAt(63, 31, 30, WHITE_FILL);

    while(1); //使程序停止运行
}
```

运行结果：



4.1.2.3 示例程序 3

目的：演示画矩形、圆方法与填充方式更改。

工程目录：UserManual\DemoCode\Sec_4123_Graphic

步骤：

以白色为背景色擦除全屏；

延迟 1s；

以屏幕的（63,31）为圆心，画一个半径为 50 的黑色填充圆；

延迟 2s；

在屏幕的（33,21）画一个宽 60，高 20 的白色无填充矩形。

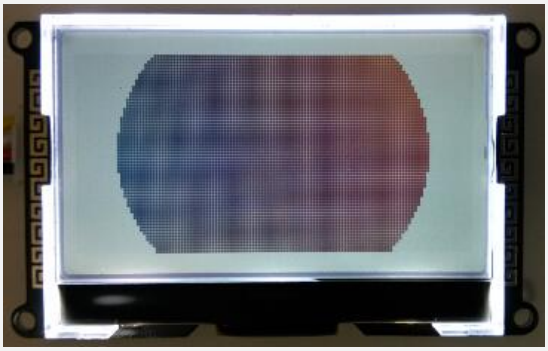
延迟 2s；

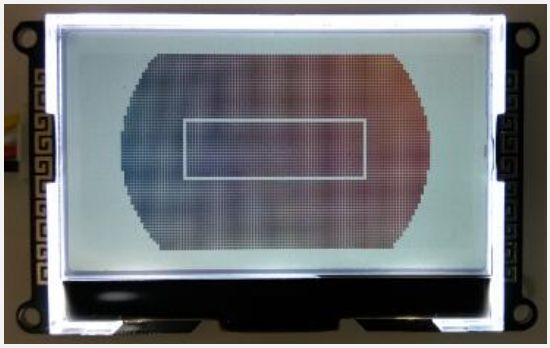
在屏幕的（37,25）画一个宽 52，高 12 的白色填充矩形。

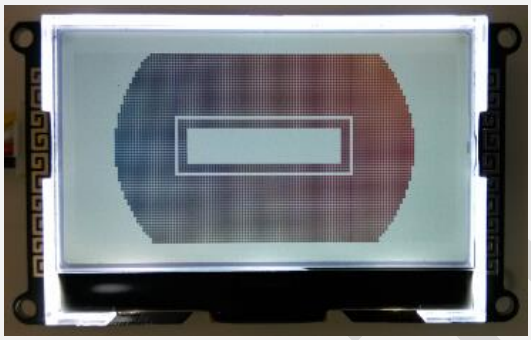
程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //画圆，黑色填充
    //Prototype: void DrawCircleAt(x, y, r, mode)
    LCD.DrawCircleAt(63, 31, 50, BLACK_FILL);
    delay(2000);            //延迟 2s

    //画矩形，白色不填充
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(33, 21, 60, 20, WHITE_NO_FILL);
    delay(2000);            //延迟 2s

    //画矩形，白色填充
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(37, 25, 52, 12, WHITE_FILL);

    while(1); //使程序停止运行
}
```

运行结果：







4.2 2D 图形 API 函数的说明

本小节将讲述与 2D 图形功能有关的 API 函数的详细属性以及调用方法。

下表列出了与 2D 图形绘制相关的 API 函数。

函数	说明
DrawDotAt( )	在指定位置，以指定颜色画点
DrawHLineAt( )	在指定位置，以指定颜色画横线
DrawVLineAt( )	在指定位置，以指定颜色画竖线
DrawLineAt( )	在指定位置，以指定颜色画线，任意斜率
DrawRectangleAt( )	在指定位置，以指定方式画矩形
DrawCircleAt( )	在指定位置，以指定方式画圆

4.2.1 DrawDotAt ( )

描述：在指定位置，以指定颜色画一个点。

函数原型：

void DrawDotAt(uint8_t x, uint8_t y, enum LCD_ColorSort color)

参数	参数说明	可以的取值	取值说明
x	画点的 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	画点的 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
color	点的颜色	WHITE	白色
		BLACK	黑色

返回值：无

调用范例：在屏幕的（0,0）位置画一个黑色点。

```
LCD.DrawDotAt(0, 0, BLACK);
```

4.2.2 DrawHLineAt ( )

描述：在指定的终点一一起点之间，以指定颜色画一条水平线。

函数原型：

```
void DrawHLineAt(uint8_t startX, uint8_t endX,
                 uint8_t y, enum LCD_ColorSort color)
```

参数	参数说明	可以的取值	取值说明
startX	横线的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
endX	横线的终止 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	横线的 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
color	线的颜色	WHITE	白色
		BLACK	黑色

返回值：无

调用范例：从屏幕的（0,0）到（127,0）画一条黑色水平线。

```
LCD.DrawHLineAt(0, 127, 0, BLACK);
```

4.2.3 DrawVLineAt ( )

描述：在指定的终点一一起点之间，以指定颜色画一条垂直线。

函数原型：

```
void DrawVLineAt(uint8_t startY, uint8_t endY,
                 uint8_t x, enum LCD_ColorSort color)
```

参数	参数说明	可以的取值	取值说明
startY	竖线的起始 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
endY	竖线的终止 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
x	竖线的 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
color	线的颜色	WHITE	白色
		BLACK	黑色

返回值：无

调用范例：从屏幕的（0,0）到（0,63）画一条黑色垂直线。

```
LCD.DrawVLineAt(0, 63, 0, BLACK);
```

4.2.4 DrawLineAt ( )

描述：在指定的终点与起点之间，以指定颜色画一条垂直线。

函数原型：

```
void DrawLineAt(uint8_t startX, uint8_t endX,
                uint8_t startY, uint8_t endY, enum LCD_ColorSort color);
```

参数	参数说明	可以的取值	取值说明
startX	线的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
endX	线的终止 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
startY	线的起始 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
endY	线的终止 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
color	线的颜色	WHITE	白色
		BLACK	黑色

返回值：无

调用范例：从屏幕的（0,0）到（127,63）画一条黑色斜线。

```
LCD.DrawLineAt(0, 127, 0, 63, BLACK);
```

4.2.5 DrawRectangleAt ( )

描述：从指定起点，以指定的宽度、高度、模式，画一个矩形。

函数原型：

```
void DrawRectangleAt(uint8_t x, uint8_t y, uint8_t width,
                     uint8_t height, enum LCD_DrawMode mode);
```

参数	参数说明	可以的取值	取值说明
x	矩形的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	矩形的起始 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
width	矩形的 x 轴方向宽度	0~127	x+width 最大取值为 127
height	矩形的 y 轴方向高度	0~63	y+height 最大取值为 63
mode	矩形的颜色与填充方式	WHITE_NO_FILL	白色无填充
		WHITE_FILL	白色填充
		BLACK_NO_FILL	黑色无填充
		BLACK_FILL	黑色填充

返回值：无

调用范例：以（0,0）为起点，画一个宽度为 60，高度为 40 的黑色无填充矩形。

```
LCD.DrawRectangleAt(0, 0, 60, 40, BLACK_NO_FILL);
```

4.2.6 DrawCircleAt ( )

描述：以指定的位置为圆心，以指定的长度为半径，以指定的模式画一个圆。

函数原型：

```
void DrawCircleAt(int8_t x, int8_t y, uint8_t r,
                  enum LCD_DrawMode mode);
```

参数	参数说明	可以的取值	取值说明
x	圆的圆心 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	圆的圆心 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63
r	圆的半径	0~127	x+width 最大取值为 127
mode	圆的颜色与填充方式	WHITE_NO_FILL	白色无填充
		WHITE_FILL	白色填充
		BLACK_NO_FILL	黑色无填充
		BLACK_FILL	黑色填充

返回值：无

调用范例：以（60,30）为圆心，画一个半径为 25 的黑色填充圆形。

```
LCD.DrawCircleAt(60, 30, 25, BLACK_FILL);
```

第 5 章 位图显示

为了丰富用户界面的元素，I2C_LCD 集成了位图显示功能，可以在 I2C_LCD 的屏幕上显示任意图片（最大支持分辨率为 128x64 的黑白双色图片）。

本章介绍如何使用 Arduino 支持库中的 API 函数实现所需图片的显示功能。

5.1 位图显示示例工程

5.1.1 位图显示

5.1.1.1 示例程序 1

目的：演示位图的绘制方法。

工程目录：UserManual\DemoCode\Sec_5111_Bitmap

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将 I2C_LCD 工作模式切换为绘图模式；

将资料中提供的兔斯基图片（Tuzki.bmp）显示在屏幕的（30,0）位置，图片数据包生成方法请参照 5.3 小节；

延迟 5s；

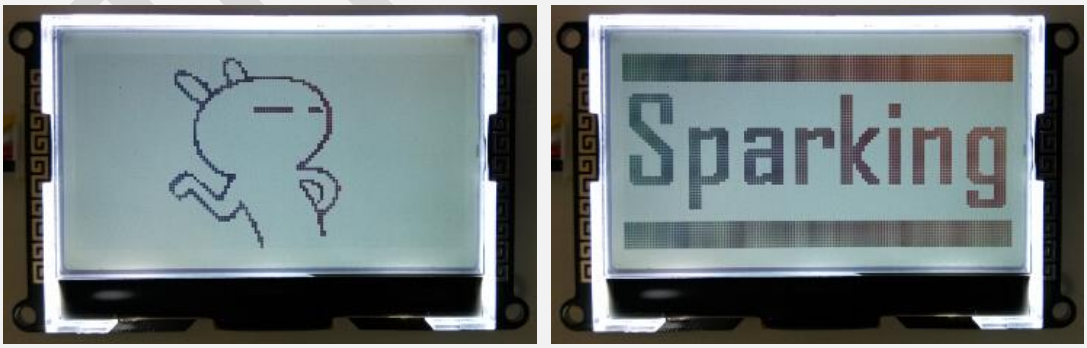
将 I2C_LCD 的启动画面背景图（Sparking）显示在屏幕上；

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
```

```
I2C_LCD LCD;
extern GUI_Bitmap_t bmTuzki;        //声明位图数据包
extern GUI_Bitmap_t bmSPLogo;      //声明位图数据包
uint8_t I2C_LCD_ADDRESS = 0x51;  //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //开启开机 Logo 显示，开启背光，位图工作模式
    //如要显示字符，需将模式切换为字符模式 WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_BitmapMode);

    //在指定位置显示位图(需要位图数据包文件),
    //数据包文件生成方法请参照 5.3 小节
    LCD.DrawScreenAreaAt(&bmTuzki, 30, 0);
    delay(5000);            //延迟 5s

    //在指定位置显示位图(需要位图数据包文件)
    LCD.DrawScreenAreaAt(&bmSPLogo, 0, 0);

    while(1); //使程序停止运行
}
```

运行结果：



5.2 位图显示 API 函数的说明

本小节将讲述与位图显示功能有关的 API 函数的详细属性以及调用方法。

下表列出了与位图显示相关的 API 函数。

函数	说明
DrawScreenAreaAt( )	以软件生成的图片数据，在指定位置绘图

5.2.1 DrawScreenAreaAt ( )

描述：在指定的屏幕区域，绘制指定的图片。

函数原型：

void DrawScreenAreaAt(GUI_BITMAP *bitmap, uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
*bitmap	图片数据包存储区的指针	GUI_BITMAP 结构体类型数据地址	数据包由软件生成，需稍加修改
x	图片的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	图片的起始 y 坐标	0~63	y 轴一共有 64 个点，编码 0~63

返回值：无

调用范例：以（20,10）为起点，将指定的图片绘制到屏幕。

```
LCD.DrawScreenAreaAt(&bmTuzki, 0, 0);
```

备注：bmTuzki 为图片数据包指针，数据包中包含了位图大小、像素点数据等内容，数据包通过软件生成，详细步骤见 5.3 小节。

5.3 图片数据包文件生成方法

本小节将介绍任意图片的数据包文件的生成方法。为了方便用户使用，我们团队为用户开发了 **Bitmap Converter** 软件，该软件现已支持 **Windows、Mac OS、Linux** 三大 PC 系统平台的主流版本。用户只需要简单的几步操作，就可以将所需的图片显示在 I2C_LCD 的屏幕上。

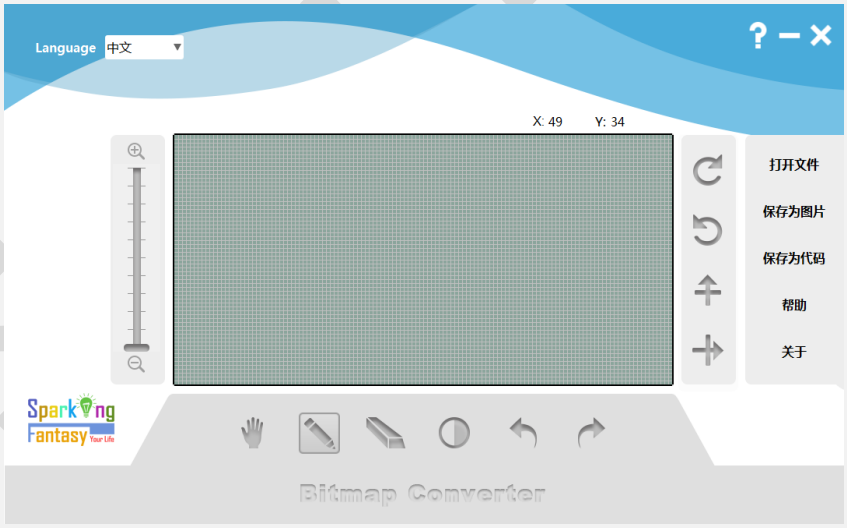
本小节以 UserManual\DemoCode\BitmapDisplay 目录下的兔斯基图片（Tuzki_1.JPG）为例，讲述如何获取该图片的数据包，并将该图片显示在 I2C_LCD 的屏幕上。

1. *用户可以使用图片编辑软件（比如 Photoshop 软件、系统自带的画图软件等）对图片进行编辑、修改。由于 I2C_LCD 只能显示黑、白双色的内容，因此建议编辑时只使用黑、白两色，不然可能影响最终显示效果；

2. *将调整后的图片保存（支持的类型：*.bmp, *.jpg, *.jpeg）；

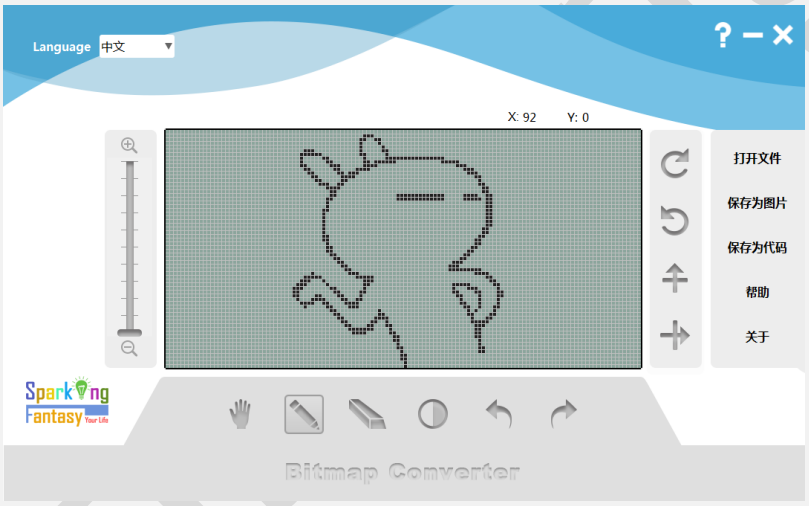
备注：文件名中禁止包含“+-*\”等运算符号以及特殊符号，否则将无法生成图片数据包。

3. 运行 BitmapConverter 目录下的 BitmapConverter 软件，点击“打开文件”浏览并打开“Tuzki_1.jpg”文件；

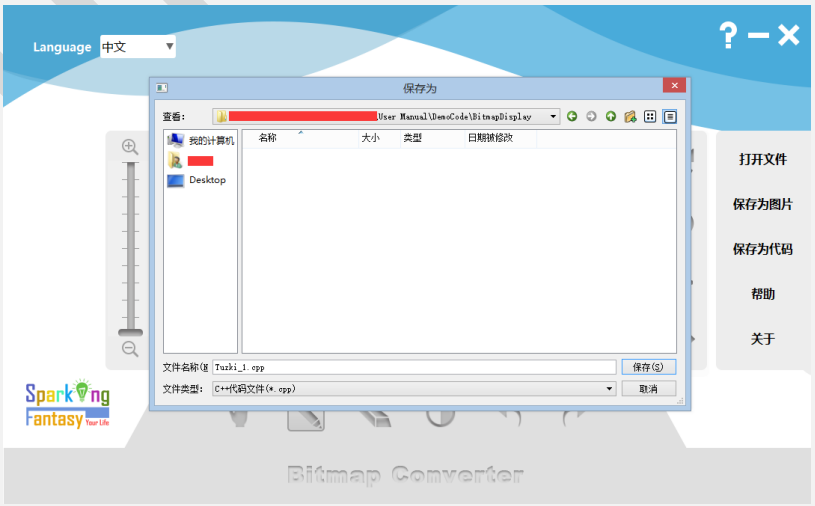


4. 使用裁剪工具选取最终想要转换的图片区域，然后点击“确定”按钮，如下图；

5. 图片已经加载到编辑区域，用户可以使用橡皮、铅笔等工具，对图片进行简单的编辑、修改、消除毛刺像素点。

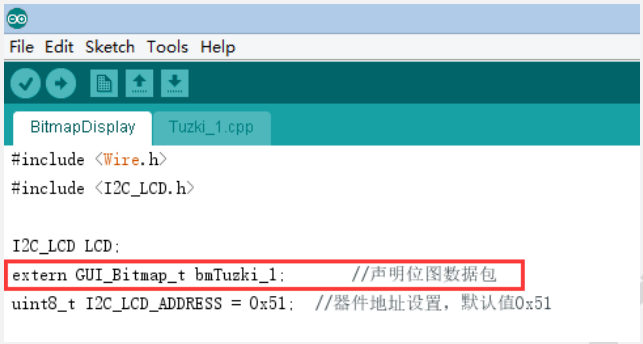


6. 然后，点击“保存为代码”按钮，选择要保存的路径，这里设为UserManual\DemoCode\BitmapDisplay；



7. 打开 UserManual\DemoCode\BitmapDisplay 目录下的 Arduino 工程（BitmapDisplay.ino），并在程序开头按照如下格式声明 bmTuzki_1 数据包；

备注：若已打开该工程，请关闭后重新打开；

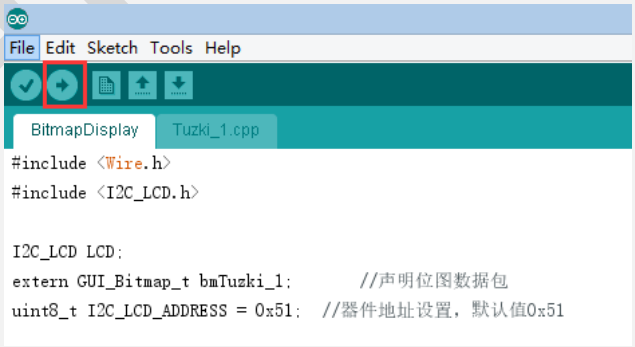


8. 修改图片数据显示函数，包含数据指针、显示起始坐标；

备注：图片显示函数的具体使用方法请见 5.2.1 小节；

```
void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟1s

    //开启开机Logo显示，开启背光，位图工作模式
    //如要显示字符，需将模式切换为字符模式WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_BitmapMode);

    //在指定位置显示位图(需要位图数据包文件),
    //数据包文件生成方法请参照5.3小节
    LCD.DrawScreenAreaAt(&bmTuzki_1, 30, 0);

    while(1); //使程序停止运行
}
```

9. 点击“Upload”按钮，下载程序到 Arduino，观察 I2C_LCD 显示结果。



第 6 章 *显存直接读/写操作

本章为高级用户选读内容，主要介绍 I2C_LCD 显存读、写 ADI 函数的使用方法，这些功能是为满足高级用户的特殊需求而设计的，普通用户可以跳过本章。

I2C_LCD 设计灵活、开放，不仅可以满足普通用户的 DIY 需求，还有面向高级用户的接口、功能、操作方式，以此来满足不同层次用户的需求。

本章将主要介绍如何通过 API 函数实现显存的读、写操作。

6.1 显存操作示例工程

6.1.1 单字节读/写显存

6.1.1.1 示例程序 1

功能：写入、读取 1 字节显存。

工程目录：UserManual\DemoCode\Sec_6111_DisRam

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将 I2C_LCD 工作模式切换为显存操作模式（WM_RamMode）；

将 0xAE 作为显存，直接写入到（0,1）位置；

延迟 3s；

从（0,1）位置，读取 1 个字节（Byte）显存，存储到 buf 变量中；

将 buf 值打印到 Arduino 串口监视器。

程序：

```
#include <Wire.h>
```

```
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
    Serial.begin(115200);   //初始化串口，115200 波特率
}

void loop(void)
{
    uint8_t buf = 0;        //缓存

    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //开启开机 Logo 显示，开启背光，显存操作模式
    //如要显示字符，需将模式切换为字符模式 WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //写入 1 个字节显存到指定位置
    //Prototype: void WriteByteDispRAM(buf, x, y)
    LCD.WriteByteDispRAM(0xAE, 0, 1);
    delay(3000);            //延迟 3s

    //从指定位置读取 1 个字节显存
    //Prototype:uint8_t ReadByteDispRAM(x, y)
    buf = LCD.ReadByteDispRAM(0, 1);

    //将读到的显存数据打印到串口监视器
    //以 16 进制格式打印
    Serial.print("buf = 0x");
    Serial.println(buf, HEX);

    while(1); //使程序停止运行
}
```

运行结果：

6.1.2 多字节读/写显存

6.1.2.1 示例程序 1

功能：写入、读取多个字节显存。

工程目录：UserManual\DemoCode\Sec_6121_DisRam

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将 buf1 数组中存储的“0xfa、0xaf、0xa5”打印到串口监视器；

将 I2C_LCD 工作模式切换为显存操作模式（WM_RamMode）；

将 buf1 数组中的数据作为显存，从（1,1）位置开始写入；

延迟 3s；

从（1,1）位置开始读取 3 个 Byte 长度的显存，并存储到 buf2 数组中；

将 buf2 数组内容打印到串口监视器。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
    Serial.begin(115200);   //初始化串口，115200 波特率
}

void loop(void)
{
    uint8_t buf1[3] = {0xFA,0xAF,0xA5}; //写入缓存
    uint8_t buf2[3] = {0};              //读取缓存
    uint8_t adder;

    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s
```

```
    //将 buf1 数组中的数据打印到串口监视器
    Serial.print("buf1[3] =");
    for(adder=0; adder<3; adder++)
    {
        //以 16 进制格式打印
        Serial.print(" 0x");
        Serial.print(buf1[adder], HEX);
    }
    Serial.println("");     //换行

    //开启开机 Logo 显示，开启背光，显存操作模式
    //如要显示字符，需将模式切换为字符模式 WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //将 buf1 中存储的数据作为显存连续写入到指定位置
    //Prototype: void WriteSeriesDispRAM(*buf, length, x, y)
    LCD.WriteSeriesDispRAM(buf1, 3, 1, 1);
    delay(3000);            //延迟 3s

    //从指定位置连续读取显存，并存储到 buf2 数组中
    //Prototype: void ReadSeriesDispRAM(*buf, length, x, y)
    LCD.ReadSeriesDispRAM(buf2, 3, 1, 1);

    //将 buf2 数组中的数据打印到串口监视器
    Serial.print("buf2[3] =");
    for(adder=0; adder<3; adder++)
    {
        //以 16 进制格式打印
        Serial.print(" 0x");
        Serial.print(buf2[adder], HEX);
    }
    Serial.println("");     //换行

    while(1); //使程序停止运行
}
```

运行结果：

6.2 显存读/写 API 函数的说明

本小节将讲述与显存读/写有关的 API 函数的详细属性以及调用方法。

下表列出了与显存读/写相关的 API 函数。

函数	说明
ReadByteDispRAM( )	从指定位置读取一个字节（Byte）显存
WriteByteDispRAM( )	指定位置写入一个字节（Byte）显存
ReadSeriesDispRAM( )	从指定位置开始，连续读取多个字节（Byte）显存
WriteSeriesDispRAM( )	指定位置开始，连续写入多个字节（Byte）显存

6.2.1 ReadByteDispRAM ( )

描述：从屏幕的指定位置读取一个 Byte 长度的显存。

函数原型：

uint8_t ReadByteDispRAM(uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
x	读取 RAM 的 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	读取 RAM 的 y 坐标	0~7	纵向 8 点为一行，一共 64/8=8 行，编码 0~7

返回值：读取到的一个字节长度显存

调用范例：从（0,0）位置，读取一个字节（Byte）显存，存储到 buf 变量中。

```
Buf = LCD.ReadByteDispRAM(0, 0);
```

6.2.2 WriteByteDispRAM ( )

描述：写入一个 Byte 长度的显存到指定的屏幕位置。

函数原型：

void WriteByteDispRAM(uint8_t buf, uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
buf	要写入 RAM 的数据	0x00~0xff	一个 Byte
x	写入 RAM 的 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	写入 RAM 的 y 坐标	0~7	纵向 8 点为一行，一共 64/8=8 行，编码 0~7

返回值：无

调用范例：将 buf 中存储的一个字节（Byte）显存，写入到（0,0）位置。

```
LCD.WriteByteDispRAM(buf, 0, 0);
```

6.2.3 ReadSeriesDispRAM ( )

描述：从屏幕的指定位置开始，读取多个 Byte 的显存数据。

函数原型：

void ReadSeriesDispRAM(uint8_t *buf, int8_t length, uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
*buf	数据存储区的指针	数组的数组名或地址	缓存数组的数组名
length	要读取的 Byte 个数	0~255	一次最多读取 255 个 Byte 显存数据
x	读取 RAM 的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	读取 RAM 的起始 y 坐标	0~7	纵向 8 点为一行，一共 64/8=8 行，编码 0~7

返回值：无

调用范例：从（0,0）位置开始，连续读取 10 个字节（Byte）显存，并存储到 buf[10]数组中。

```
LCD.ReadSeriesDispRAM(buf, 10, 0, 0);
```

6.2.4 WriteSeriesDispRAM ( )

描述：从屏幕的指定位置开始，读取多个 Byte 的显存数据。

函数原型：

void WriteSeriesDispRAM(uint8_t *buf, int8_t length, uint8_t x, uint8_t y)

参数	参数说明	可以的取值	取值说明
*buf	数据存储区的指针	数组的数组名或地址	缓存数组的数组名
length	要写入的 Byte 个数	0~255	一次最多写入 255 个 Byte 显存数据
x	写入 RAM 的起始 x 坐标	0~127	x 轴一共有 128 个点，编码 0~127
y	写入 RAM 的起始 y 坐标	0~7	纵向 8 点为一行，一共 64/8=8 行，编码 0~7

返回值：无

调用范例：将 buf[10]中存储的 10 个字节（Byte）显存，从（0,0）位置开始，连续写入到显存中。

```
LCD.WriteSeriesDispRAM(buf, 10, 0, 0);
```

第 7 章 系统设置

I2C_LCD 设计灵活、开放，可以满足不同用户的个性需求，尽可能的给用户最大的 DIY 空间。

本章将讲述 I2C_LCD 的系统设置的内容与方法，主要包含显示模式、背光亮度、开机 LOGO、工作模式、屏幕对比度、器件地址、恢复出厂等设置。

本章将主要介绍如何通过 API 函数实现系统设置与个性化设置。

7.1 系统设置示例工程

7.1.1 显示模式（反向/正常）切换

7.1.1.1 示例程序 1

功能：将 I2C_LCD 显示模式切换为反向，然后切换回正常。

工程目录：UserManual\DemoCode\Sec_7111_System

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_10x20，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（0,20）位置显示“Sparking”；

延迟 3s；

将 I2C_LCD 显示模式修改为反向显示；

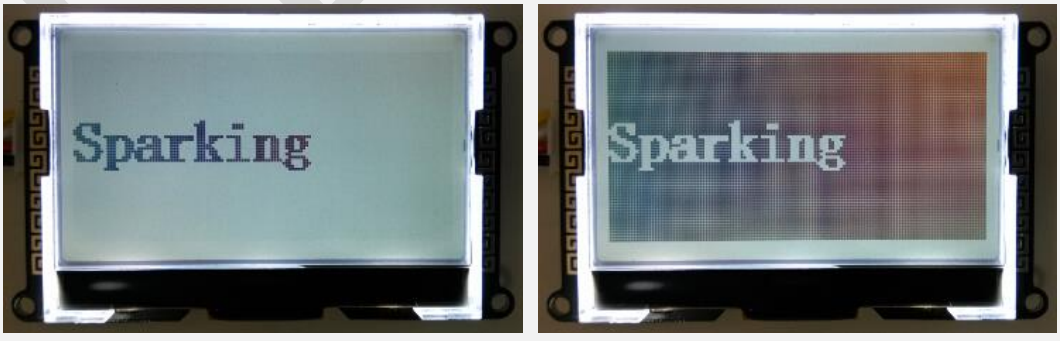
延迟 3s；

将 I2C_LCD 显示模式修改为正常显示。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //10X20 像素字体，自动换行，白色字符-黑色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 20);    //显示字符串
    delay(3000);            //延迟 3s

    //设置为反向显示模式，黑变白，白变黑
    LCD.DisplayConf(AllREV);
    delay(3000);            //延迟 3s

    //设置为正常显示模式
    LCD.DisplayConf(AllNOR);

    while(1); //使程序停止运行
}
```

运行结果：



7.1.2 工作模式切换

7.1.2.1 示例程序 1

功能：I2C_LCD 字符工作模式与绘图工作模式的切换。

工程目录：UserManual\DemoCode\Sec_7121_System

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16_1，地址更新方式设为自动换行、自动累加，字符显示模式设为黑色有背景色；

在屏幕的（50,10）位置显示“Hello！”；

延迟 3s；

将 I2C_LCD 工作模式切换为绘图模式；

将资料中提供的“摄像机”图片显示在屏幕（0,0）位置；

延迟 3s；

将 I2C_LCD 工作模式切换为字符模式；

在屏幕的（50,30）位置显示“Director.”；

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>

I2C_LCD LCD;
```

```
extern GUI_Bitmap_t bmCamera;   //声明位图数据包
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 50, 10);     //显示字符串
    delay(3000);            //延迟 3s

    //开启开机 Logo 显示，开启背光，位图工作模式
    //如要显示字符，需将模式切换为字符模式 WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_BitmapMode);

    //在指定位置显示位图(需要位图数据文件),
    //数据文件生成方法请参考用户手册-位图绘制章节
    LCD.DrawScreenAreaAt(&bmCamera, 0, 0);
    delay(3000);            //延迟 3s

    //开启开机 Logo 显示，开启背光，字符工作模式(系统默认)
    LCD.WorkingModeConf(ON, ON, WM_CharMode);
    LCD.DispStringAt("Director.", 50, 30);

    while(1); //使程序停止运行
}
```

运行结果：

7.1.2.2 示例程序 2

功能：I2C_LCD 字符工作模式与显存操作模式的切换。

工程目录：UserManual\DemoCode\Sec_7122_System

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16_1，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（20,20）位置显示“Hello！”；

延迟 3s；

将 I2C_LCD 工作模式切换为显存操作模式（高级用户适用模式）；

将 buf 数组中存储的“0xfa、0xaf、0xa5”从（20,2）位置开始，连续写入到显存中。

延迟 3s；

将 I2C_LCD 工作模式切换为字符模式；

在屏幕的（20,40）位置显示“Director.”；

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51
```

```
void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    uint8_t buf[3] = {0xFA,0xAF,0xA5};  //缓存

    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 50, 10);     //显示字符串
    delay(3000);            //延迟 3s

    //开启开机 Logo 显示，开启背光，显存操作模式
    //如要显示字符，需将模式切换为字符模式 WM_CharMode
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //将 buf 中存储的数据作为显存连续写入到指定位置
    //Prototype: void WriteSeriesDispRAM(*buf, length, x, y)
    LCD.WriteSeriesDispRAM(buf, 3, 20, 2);
    delay(3000);            //延迟 3s

    //开启开机 Logo 显示，开启背光，字符工作模式(系统默认)
    LCD.WorkingModeConf(ON, ON, WM_CharMode);
    LCD.DispStringAt("Director.", 50, 30);  //显示字符串

    while(1); //使程序停止运行
}
```

运行结果：

7.1.3 背光与开机 Logo 开关

7.1.3.1 示例程序 1

功能：I2C_LCD 背光与开机 Logo 的关闭。

工程目录：UserManual\DemoCode\Sec_7131_System

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16_1，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（20,20）位置显示“Hello！”；

延迟 2s；

将屏幕背光设为“OFF”，开机 Logo 设为“OFF”；

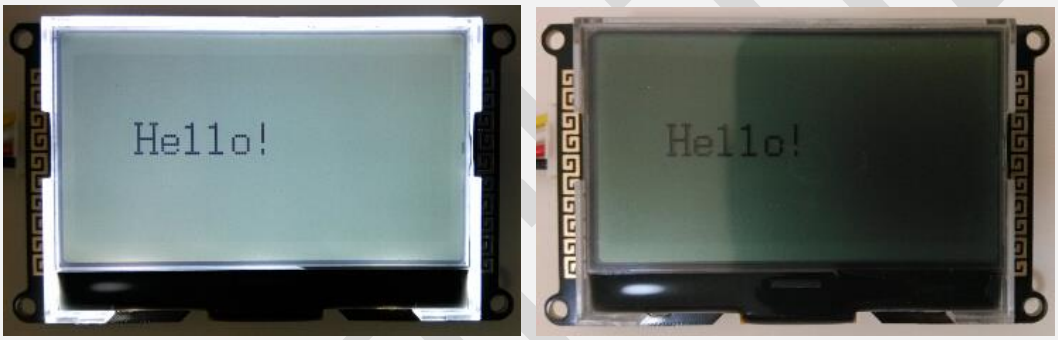
按下 I2C_LCD 顶部的复位键，I2C_LCD 将保持刚才的背光与开机 Logo 设置。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}
```

```
void loop(void)
{
    LCD.CleanAll(WHITE);     //白色清屏
    delay(1000);             //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 50, 10);      //显示字符串
    delay(2000);             //延迟 2s

    //关闭开机 Logo 显示，关闭背光，字符工作模式(系统默认)
    LCD.WorkingModeConf(OFF, OFF, WM_CharMode);

    while(1); //使程序停止运行
}
```

运行结果：



7.1.3.2 示例程序 2

功能：I2C_LCD 背光与开机 Logo 的开启。

工程目录：UserManual\DemoCode\Sec_7132_System

步骤：

以白色为背景色擦除全屏；

延迟 1s；

将字体设为 Font_8x16_1，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（20,20）位置显示“Hello！”；

延迟 2s；

将屏幕背光设为“ON”，开机 Logo 设为“ON”；

按下 I2C_LCD 顶部的复位键，I2C_LCD 将保持刚才的设置。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //8X16 像素字体 1，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 20, 20);     //显示字符串
    delay(2000);            //延迟 2s

    //开启开机 Logo 显示，开启背光，字符工作模式(系统默认)
    LCD.WorkingModeConf(ON, ON, WM_CharMode);

    while(1); //使程序停止运行
}
```

运行结果：

7.1.4 背光亮度设置

7.1.4.1 示例程序 1

功能：I2C_LCD 背光亮度等级调整，调整结果是否掉电保存。

工程目录：UserManual\DemoCode\Sec_7141_System

步骤：

用白色背景色擦除全屏；

延迟 1s；

将屏幕背光开关设为“ON”；

将背光亮度等级调整为 20，调整结果保存到 RAM 中；

按下 I2C_LCD 顶部的复位键，I2C_LCD 将恢复原来亮度等级。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //开启开机 Logo 显示，开启背光，字符工作模式(系统默认)
    LCD.WorkingModeConf(ON, ON, WM_CharMode);

    //调整背光亮度，调整结果掉电、复位不保存
    LCD.BacklightConf(LOAD_TO_RAM, 20);

    while(1); //使程序停止运行
}
```

运行结果：


7.1.5 对比度设置

7.1.5.1 示例程序 1

功能：I2C_LCD 对比度等级调整，调整结果是否掉电保存。

工程目录：UserManual\DemoCode\Sec_7151_System

步骤：

用白色背景色擦除全屏；

延迟 1s；

将字体设为 Font_10x20，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（0,20）位置显示“Sparking”；

延迟 2s；

将对比度等级调整为 12，调整结果保存到 RAM 中。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
```

```
{
    LCD.CleanAll(WHITE);     //白色清屏
    delay(1000);             //延迟 1s

    //10X20 像素字体，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 20);     //显示字符
    delay(2000);             //延迟 2s

    //调整对比度，调整结果掉电、复位不保存
    LCD.ContrastConf(LOAD_TO_RAM, 12);

    while(1); //使程序停止运行
}
```

运行结果：

7.1.6 器件地址修改

7.1.6.1 示例程序 1

功能：I2C_LCD 器件地址修改，调整结果是否掉电保存。

工程目录：UserManual\DemoCode\Sec_7161_System

步骤：

用白色背景色擦除全屏；

延迟 1s；

将字体设为 Font_10x20，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（0,10）位置显示“Sparking”；

延迟 2s；

将 I2C_LCD 的器件地址修改为 0x52；

尝试在屏幕的（0,30）位置显示“Hello！”。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51
//uint8_t I2C_LCD_ADDRESS = 0x52; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //白色清屏
    delay(1000);            //延迟 1s

    //10X20 像素字体，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);    //显示字符
    delay(2000);            //延迟 2s

    //修改 I2C_LCD 的器件地址，出厂默认值 0x51
    LCD.DeviceAddrEdit(0x52);

    //显示字符，但是由于器件地址与程序目标地址不同，而无法正常工作。
    LCD.DispStringAt("Hello!", 0, 30);

    //通过 I2C_LCD_ADDRESS 变量设置当前程序的目标器件地址，
    //将 I2C_LCD_ADDRESS 变量的值修改为 0x52 后，I2C_LCD 即可正常工作。
    //在忘记 I2C_LCD 器件地址的情况下，可按照用户手册-恢复出厂章节的说明，
    //将 I2C_LCD 器件地址恢复为出厂设置:0x51

    while(1); //使程序停止运行
}
```

运行结果：

7.1.7 全屏清除

7.1.7.1 示例程序 1

功能：清除屏幕上的所有显示内容。

工程目录：UserManual\DemoCode\Sec_7171_System

步骤：

用白色背景色擦除全屏；

延迟 1s；

将字体设为 Font_10x20，地址更新方式设为自动换行、累加，字符显示模式设为黑色有背景色；

在屏幕的（0,10）位置显示“Sparking”；

延迟 2s；

用黑色背景色擦除全屏。

延迟 2s；

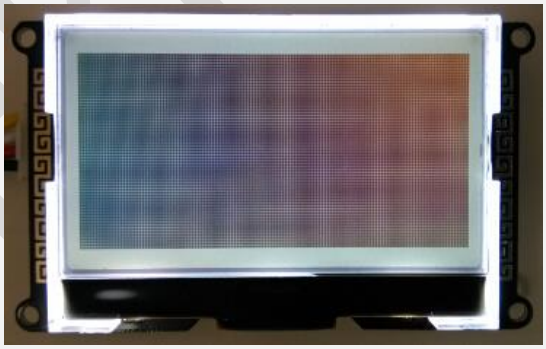
用白色背景色擦除全屏。

程序：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //器件地址设置，默认值 0x51

void setup(void)
{
    Wire.begin();           //初始化 I2C 控制器
```

```
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //白色清屏
    delay(1000);             //延迟 1s

    //10X20 像素字体，自动换行，黑色字符-白色背景色
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);     //显示字符
    delay(2000);             //延迟 2s

    LCD.CleanAll(BLACK);     //黑色清屏
    delay(2000);             //延迟 2s

    LCD.CleanAll(WHITE);     //白色清屏

    while(1); //使程序停止运行
}
```

运行结果：




7.2 支持库中的宏定义、变量说明

下表是支持库中所用到的预编译宏定义与器件地址设置的说明。

在不使用相关功能的 API 函数时，可以在支持库目录下的“I2C_LCD.h”文件开头部分，将相关宏定义设为“FALSE”，程序编译器将不再编译这部分

函数，节省用户控制器 ROM 资源。相反，若需要使用这部分 API 函数，则需要将相关宏定义设为“TRUE”。

I2C_LCD_ADDRESS 是当前程序操作的目标器件地址，如果用户修改了 I2C_LCD 模块的器件地址，对应的需要将程序中 I2C_LCD_ADDRESS 的也设为相同的值，才能继续操作 I2C_LCD 模块。

变量或宏定义	类型	可以的取值	取值说明
SUPPORT_2D_GRAPHIC_LIB	宏定义	FALSE	关闭 2D 图形库支持
		TRUE，默认值	开启 2D 图形库支持
I2C_LCD_ADDRESS	uint8_t	0~127 (0x00~0x7f)默认值：0x51	当前器件地址设置

7.3 支持库系统设置 API 函数的说明

本小节将讲述与系统设置有关的 API 函数的详细属性以及调用方法。

下表列出了与系统设置相关的 API 函数。

函数	说明
DisplayConf( )	配置屏幕的显示模式，正常/反向
WorkingModeConf( )	配置背光、开机 Logo 的开关与屏幕的工作模式
BacklightConf( )	配置背光亮度与参数保存方式
ContrastConf( )	配置对比度与参数保存方式
DeviceAddrEdit( )	更改 I2C_LCD 模块的器件地址
CleanAll( )	以指定的颜色（黑/白）擦除全屏

7.3.1 DisplayConf ( )

描述：配置屏幕的显示模式。

函数原型：

void DisplayConf(enum LCD_DisplayMode mode)

参数	参数说明	可以的取值	取值说明
mode	屏幕显示模式	AllREV	全部反向显示，黑变白，白变黑
		AllNOR	全部正常显示，默认值

返回值：无

调用范例：将 I2C_LCD 显示模式设为“AllREV”；

```
LCD.DisplayConf(AllREV);
```

7.3.2 WorkingModeConf ( )

描述：配置背光、开机 Logo 的开关与屏幕的工作模式。

函数原型：

void WorkingModeConf(enum LCD_SwitchState logoSwi,
enum LCD_SwitchState backLightSwi,
enum LCD_WorkingMode mode)

参数	参数说明	可以的取值	取值说明
logoSwi	开机 LOGO 开/关	OFF	关闭开机 LOGO 显示，并存储至 EEPROM
		ON	开启开机 LOGO 显示，并存储至 EEPROM，默认值
backLightSwi	背光开/关	OFF	关闭背光，并存储至 EEPROM
		ON	开启背光，并存储至 EEPROM，默认值

mode	工作模式	WM_CharMode	字符工作模式，默认值
		WM_BitmapMode	绘图工作模式
		WM_RamMode	显存操作模式

返回值：无

调用范例：将 I2C_LCD 开机 LOGO 设为“ON”，背光设为“OFF”，工作模式设为“WM_CharMode”；

```
LCD.WorkingModeConf(ON, OFF, WM_CharMode);
```

7.3.3 BacklightConf ( )

描述：配置背光亮度与参数保存方式。

函数原型：

```
void BacklightConf(enum LCD_SettingMode mode, uint8_t buf)
```

参数	参数说明	可以的取值	取值说明
mode	背光设置模式	LOAD_TO_RAM	设置保存到 RAM，掉电不保存
		LOAD_TO_EEPROM	设置保存到 EEPROM，掉电保存
buf	背光亮度设置	0~127	128 级背光可调，默认值 127

返回值：无

调用范例：将 I2C_LCD 背光设置模式设为“LOAD_TO_RAM”，背光亮度等级设为 20；

```
LCD.BacklightConf(LOAD_TO_RAM, 20);
```

7.3.4 ContrastConf ( )

描述：配置对比度与参数保存方式。

函数原型：

void ContrastConf(enum LCD_SettingMode mode, uint8_t buf)

参数	参数说明	可以的取值	取值说明
mode	对比度设置模式	LOAD_TO_RAM	设置保存到 RAM，掉电不保存
		LOAD_TO_EEPROM	设置保存到 EEPROM，掉电保存
buf	对比度设置	0~63	64 级对比度可调，默认值 21

返回值：无

调用范例：将 I2C_LCD 背光设置模式设为“LOAD_TO_RAM”，背光亮度等级设为 10；

```
LCD.ContrastConf(LOAD_TO_RAM, 10);
```

7.3.5 DeviceAddrEdit ( )

描述：更改 I2C_LCD 模块的器件地址。

函数原型：

void DeviceAddrEdit(uint8_t newAddr)

参数	参数说明	可以的取值	取值说明
newAddr	器件新地址	0~127	新的器件地址，立刻生效

返回值：无

调用范例：将器件地址设为 0x51。

```
LCD.DeviceAddrEdit(0x51);
```

7.3.6 CleanAll ( )

描述：以指定的颜色（黑/白）擦除全屏。

函数原型：

void CleanAll(enum LCD_ColorSort color)

参数	参数说明	可以的取值	取值说明
color	清屏的颜色	WHITE	使用白色清屏，完成后屏幕为白色
		BLACK	使用黑色清屏，完成后屏幕为黑色

返回值：无

调用范例：用黑色“BLACK”，将全屏内容擦除。

```
LCD.CleanAll(BLACK);
```

第 8 章 故障排除与恢复

本章主要讲述在使用 I2C_LCD 的过程中，遇到故障时的恢复方法。

8.1 故障排查与解决方法

在产品使用过程中遇到问题时，用户可以通过查询下面的故障列表，寻找解决问题的方法。

故障现象	排除方法
屏幕黑屏，无任何反应	1、检查供电是否为 5v
	2、按照 8.3 小节说明，进行恢复出厂
I2C_LCD 不受控制	1、按下复位键进行复位
	2、连接不良，更换连线再试
	3、按照 8.3 小节说明，进行恢复出厂
	4、检查程序中地址配置是否为默认的 0x51
忘记 I2C_LCD 模块的器件地址	1、按照 8.3 小节说明，进行恢复出厂

备注：若以上步骤仍无法解决故障，请与我们联系，并在邮件中详细描述您所遇到的故障现象。（若您的 I2C_LCD 浸水、跌落或非正常供电而导致损坏，将不在我们的保修范围内。）

E-mail：joney.s@foxmail.com

8.2 复位按键说明

在 I2C_LCD 的顶部有一颗复位按键，若 I2C_LCD 的状态不正常时，可通过按下此按键恢复。

备注：通过复位按键恢复，保存在 I2C_LCD 内存的参数设置将丢失，EEPROM 中的参数设置不受影响。

8.3 恢复出厂设置说明

若通过其他途径无法解决所遇到的故障，请按照以下步骤将 I2C_LCD 模块恢复出厂。

1、 在正常供电的情况下，用导电的物体（镊子、螺丝刀等），短接 I2C_LCD 模块右下角背面的 REC 触点；

2、 保持 REC 触点处于短接状态，按下顶部的复位按键（RST）；

3、 当屏幕出现“Resetting...”时，代表 I2C_LCD 正在复位，当显示“OK！”时，说明恢复出厂设置成功，所有寄存器的值将恢复默认值。

备注：

1. 若一次未能成功，请按照步骤重试几次，直到出现步骤 3 中的“OK！”提示；
2. 恢复出厂成功后，所有参数设置将恢复为默认值，请确认程序中的器件地址配置（I2C_LCD_ADDRESS）的值为默认的器件地址（0x51），不然将导致 I2C_LCD 无反应；