I2C_LCD
USER MANUAL

English Version 1.2

2014-9-8

Sparking Work Space

Shenzhen, China

Release Note

Version	Note	Date
v1.0	Initial version	20150620
v1.1	Add bitmap converter illustration	20150712
V1.2	1、 Update bitmap converter illustration 2、 Modify the description of bitmap display function	20150827

Chapter 1. Introduction

I2C_LCD12864 is an easy-to-use display module，Hereinafter referred to as I2C_LCD.

The resolution of I2C_LCD is 128*64, support black and white display. Maximum 168 characters and maximum 128*64 pixel black & white picture can be displayed.

I2C_LCD has an independent controller, most of the complex operations run in the independent controller, it can reduce the user controller (MCU, Arduino) of the computing burden.

We provide users with full-featured Arduino library, in the case of using the library, simply a few lines of the program, you can achieve the characters, graphics and other display functions.

1.1 How to use this manual

In this manual, describes the use of API functions in the Arduino library, and provides the relevant sample code to help users use I2C_LCD.

Firstly, the user can learn to use the I2C_LCD by demo code of each chapter. Demo code already contains the detailed annotation and illustration. When using the demo code encountered difficulties, please referring to the second section of each chapter to understand the use of API function.

When you get problems, you can search through this manual with keywords to find the solution of the problems.

1.2 **Parameters of I2C_LCD**

1.2.1 **Features**

I2C_LCD is an easy-to-use display module. Provide a highly efficient dual color UI interface design approach.

1. Only 2 Arduino pins are occupied (Use I2C interface).

2. Supports standard I2C mode (100Kbit/s) and fast I2C mode (400Kbit/s).

3. Compatible with multiple communication logic levels: 2.8~5VDC.

4. Arduino library supported, use a line of code to complete the display.

5. Integrate 7 sizes of ASCII fonts, 5 graphics functions.

6. Provide dedicated picture data convert software (Bitmap Converter).

7. Most of the complex operation is processed by I2C_LCD independent controller, saving user controller resources.

8. Supports cursor function, can set up 16 cursor flicker frequency.

9. Supports 128 level backlight brightness adjustment.

10. Support 64 level screen contrast adjustment.

11. Support device address modification.

12. Supports 127 I2C_LCD work in parallel.

13. When debugging code, it can take the place of the serial monitor to monitor the program running state.

14. Two abnormal recovery methods are provided: reset and restore the factory settings.

15. Compatible with Grove interface and 4Pin-100mil interface (under the Grove socket).

16. 4 symmetrical fixed hole design for easy user installation.

17. China style unique appearance.

1.2.2 **Electrical Characteristics**

Symbol	Description	Minimum	Typical	Maximum	Unit
5V	Power in	4.5	5	5.5	V
SCL\SDA	I2C bus	2.8	5	5.5	V
I_{in}	Current	13	20	37.1	mA

1.3 **Resource Requirements**

Hardware requirements: Any Arduino compatible board, any controller with I2C communication function;

Voltage requirement: Power supply voltage is 5V, communication logic voltage is 2.8~5V;

ROM requirements: Biger than 4Kb;

RAM requirements: Biger than 512Byte;

1.4 **Screen & Coordinates**

The screen is made up of many points that can be individually controlled, which are called pixels. Users can draw on any specified pixel by the program.

The horizontal scale is called the X-axis, and the vertical scale is called the Y-axis. X-axis and Y-axis coordinate representation of a two-dimensional coordinate (X, Y). In the program need to use both X and Y coordinates, X coordinates always in the front. The upper left corner of the I2C_LCD screen is the default coordinate (0, 0). The positive X direction is always right and the positive Y direction is always down.

Figure 1.4.1 shows the positive direction of the I2C_LCD coordinate system. All the coordinates passed to the API function are always specified at 1 pixels for 1.
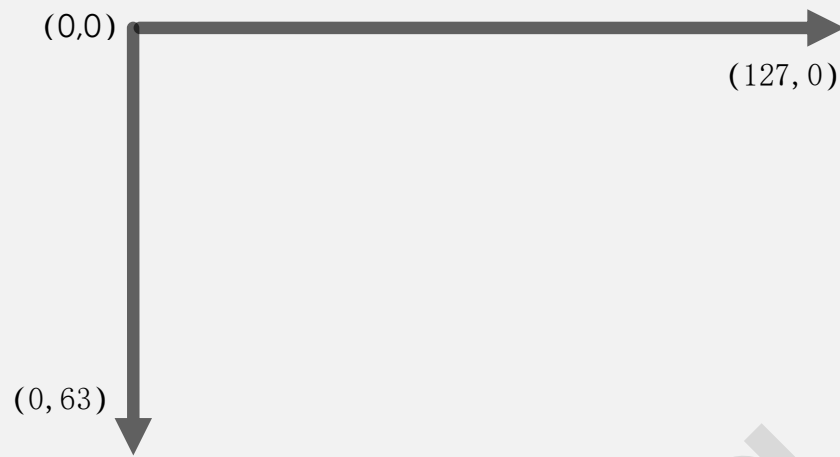
(0,0)

$(127, 0)$

$(0, 63)$

Figure 1.4.1 Coordinate system

Chapter 2. Character Display

I2C_LCD integrated 7 ASCII fonts: Font_6x8, Font_6x12, Font_8x16_1, Font_8x16_2, Font_10x20, Font_12x24, Font_16x32, to meet your personalized needs.

With the support of the Arduino I2C_LCD library, the realization of the character display function will become very simple, just a line of program needed.

This chapter will show how to display different sizes, different colors of characters.

Notes: Font_10x20 represents a font, its width is 10 pixels, 20 pixels high.

2.1 I2C_LCD Supported Characters

I2C_LCD supports American Standard Code for information interchange (ASCII). Compiled as 8-bit format characters, allowing for the maximum of 126 different character code. For the convenience of users, in I2C_LCD the 28~31 is defined as 4 direction arrow "← → ↑ ↓".

The ASCII characters and extended characters which I2C_LCD supported are shown in the following table:

Character set for I2C_LCD:

DEC	HEX	Char	DEC	HEX	Char	DEC	HEX	Char	DEC	HEX	Char
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 Sample Project Of Character Display

2.2.1 Display character and string

2.2.1.1 Sample 1

Objective: Demonstrate how to display character and string.

Path: UserManual\DemoCode\Sec_2211_Text

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_6x8, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display character 'A' at (0, 0);

Display the string "Sparking" at (0, 10);

Set the start coordinate as (0, 20);

Print "Hello, World!" on I2C_LCD at the start coordinate;

Circulating print the number of seconds since reset at (0 ,30);

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();           //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //Erase all.
    delay(1000);            //Delay for 1s.

    //8*16 font size, auto new line, black character on white back ground.
    LCD.FontModeConf(Font_6x8, FM_ANL_AAA, BLACK_BAC);
```
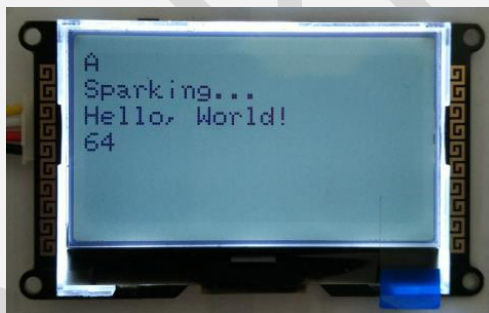
```
LCD.DispCharAt('A', 0, 0);      //Display character.

LCD.DispStringAt("Sparking...", 0, 10);     //Display string.

//Set the start coordinate.
LCD.CharGotoXY(0,20);
//Print string on I2C_LCD at the start coordinate.
LCD.print("Hello, World!");

while(1)
{
    //Set the start coordinate.
    LCD.CharGotoXY(0,30);
    //Print the number of seconds since reset.
    LCD.print(millis()/1000,DEC);
    delay(1000); //Delay for 1s.
}
}
```

Operating Results:

2.2.2 Font and color settings

2.2.2.1 Sample 1

Objective: Demonstrate how to display different fonts, different colors.

Path: UserManual\DemoCode\Sec_2221_Text

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display the string "Sparking" at (0, 10);

Delay for 2s;

To change the font to Font_10x20, change the character display mode to WHITE_BAC, then display the string "Sparking" at (0, 30).

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);       //Erase all.
    delay(1000);               //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);    //Display string.
    delay(2000);               //Delay for 2s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, WHITE_BAC);
    LCD.DispStringAt("Sparking", 0, 30);    //Display string.

    while(1);                  //Wait for ever.
}
```

Operating Results:
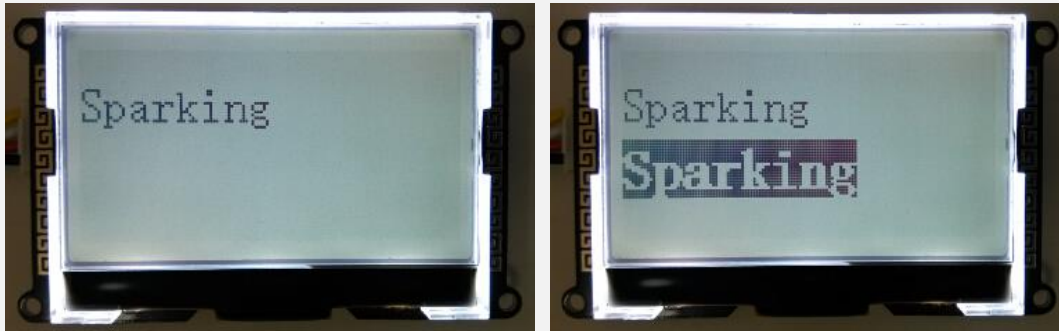
2.2.3 Setting Character Background Color

2.2.3.1 Sample 1

Objective: Demonstrate how to set the background colors of the characters.

Path: UserManual\DemoCode\Sec_2231_Text

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_10x20, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display the string "Sparking" at (0, 10);

Delay for 2s;

Display the string " YES_BAC " at (0, 20);

Delay for 2s;

Change the character display mode to BLACK_NO_BAC;

Display the string " NO_BAC " at (0, 27);

Code:

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

I2C_LCD LCD;
```

```
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);     //Display string.
    delay(2000);             //Delay for 2s.

    LCD.DispStringAt("YES_BAC", 0, 20);      //Display string.
    delay(2000);             //Delay for 2s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_NO_BAC);
    LCD.DispStringAt("NO_BAC", 0, 27);       //Display string.

    while(1); //Wait for ever.
}
```

Operating Results:
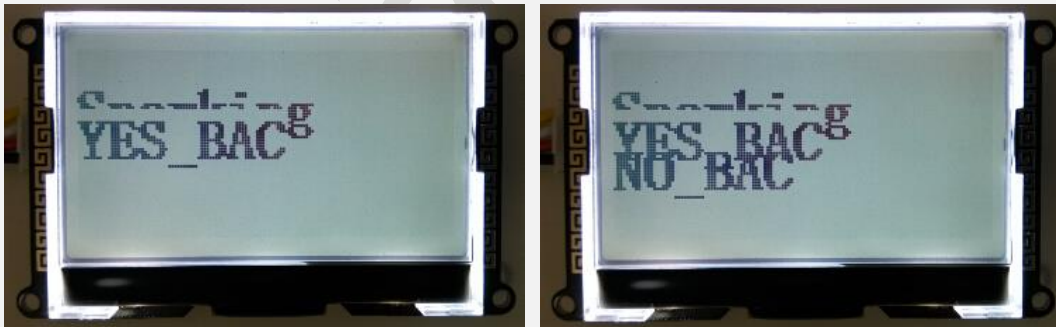
2.2.4   Character Address Accumulation Mode

2.2.4.1   Simple 1

Objective: Demonstrate the difference between auto new line mode (FM_ANL_AAA) and manual new line mode (FM_MNL_AAA).

Path: UserManual\DemoCode\Sec_2241_Text

Steps:

Erase full screen with white background color;

Delay for 1s;

The font is set to Font_6x8, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

In (108, 10) display the string "Sparking"";

Delay for 3s;

The character address update mode is set to FM_MNL_AAA, and display "Sparking" at (108, 40) ".

Code：

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Set the font, character address update mode, display mode.
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_6x8, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 108, 10);   //Display string.
    delay(3000);             //Delay for 3s.

    //Set the font, character address update mode, display mode.
    //FM_MNL_AAA: FM_ManualNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_6x8, FM_MNL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 108, 40);   //Display string.

    while(1); //Wait for ever.
}
```

Operating Results:

2.3 Character Display API Instruction

By reading this section the user can fully understand the details of the API function, and knows how to use it.

The following table lists the API functions associated with the text processing in the Arduino I2C_LCD library.

Function	Instruction
print( )	Display data or character at current position.
DispCharAt( )	Display a single character at the specified position.
DispStringAt( )	Display multiple characters at a specified position.
FontModeConf( )	Font mode configuration.

2.3.1 print ( )

Instruction: Display data or character at current position.

Function description:

This function can use for display data or character.

LCD.print(78) gives "78";

LCD.print(1.23456) gives "1.23";

LCD.print('N') gives "N";

LCD.print("Hello world.") gives "Hello world.";

This function can display data in special format.

LCD.print(78, BIN) gives "1001110";

LCD.print(78, OCT) gives "116";

LCD.print(78, DEC) gives "78";

LCD.print(78, HEX) gives "4E";

LCD.println(1.23456, 0) gives "1";

LCD.println(1.23456, 2) gives "1.23";

LCD.println(1.23456, 4) gives "1.2346";

Return: Null

Example: Display "Hello" at (10,0).

LCD. CharGotoXY(10, 0);

LCD. print("Hello");

2.3.2 **DispCharAt ( )**

Instruction: Displaying a single character at the specified position.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
buf	The character to be written.	ASCII characters	Any ASCII character.
x	X-coordinate	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Y-coordinate	0~63	The total number of points on the Y-axis is 64, encoding 0~63.

Return: Null

Example: Display 'A' at (0, 0) `coordinate`.

<div align="center">

`LCD.DispCharAt(‘A’, 0, 0);`

</div>

2.3.3   DispStringAt ( )

Instruction: Display multiple characters at specified position.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
*buf	Pointer to the string.	An array name or address of a string.	Null
x	X-coordinate	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Y-coordinate	0~63	The total number of points on the Y-axis is 64, encoding 0~63.

Return: Null

Example: Display "Sparking" at (0, 30) `coordinate`.

<div align="center">

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

</div>

2.3.4   FontModeConf( )

Instruction: Configure font display mode.

Function Prototype:

void FontModeConf(enum LCD_FontSort font,

enum LCD_FontMode mode, enum LCD_CharMode cMode)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction

font	Font type.	Font_6x8	Font size for 6*8 pixels, the default value.
		Font_6x12	Font size for 6*12 pixels.
		Font_8x16_1	Font size for 8*16 pixels.
		Font_8x16_2	Font size for 8*16 pixels.
		Font_10x20	Font size for 10*20 pixels.
		Font_12x24	Font size for 12*24 pixels.
		Font_16x32	Font size for 16*32 pixels.
mode	Character address update mode.	FM_ANL_AAA	Auto wrap, address automatic accumulation, this is short for FM_AutoNewLine_AutoAddrAdd.
		FM_MNL_AAA	Manual wrap, address automatic accumulation, this is short for FM_ManualNewLine_AutoAddrAdd.
		FM_MNL_MAA	Manual wrap, manual address accumulation, this is short for FM_ManualNewLine_ManualAddrAdd.
cMode	Character display mode.	WHITE_BAC	White character, black background, the default value.
		WHITE_NO_BAC	White character, no background.
		BLACK_BAC	Black character, white background.
		BLACK_NO_BAC	Black character, no background.

Return: Null

Example: Set the font to Font_6x8, set the character address

update mode to FM_ANL_AAA, set the character display mode to

BLACK_BAC.

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

Chapter 3. The Usage of Cursor

In many applications, the cursor is required to indicate the current focus. In order to facilitate the user to build a fast interactive interface, I2C_LCD integrated the cursor function. User can config the cursor's switch, position, size, flicker frequency and other properties by the relevant API functions.

This chapter will describe how to configure and use the cursor.

3.1 Sample Project For Cursor Usage

3.1.1 The property of the cursor

3.1.1.1 Sample 1

Objective: Demonstrate the method of setting up the cursor switch, position and flicker frequency.

Path: UserManual\DemoCode\Sec_3111_Cusor

Steps:

Erase full screen with white background color;

Delay for 1s;

The font is set to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display time string "12:00:00" at (0, 30);

Delay for 2s;

Set the cursor position to the second;

Open the cursor and set the flash frequency level to 6;

Delay for 5s;

Move the cursor to the ten bit of the second.

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("12:00:00", 0, 30);     //Display clcok string.
    delay(2000);             //Delay for 2s.

    //To move the cursor point to ones place of second.
    //Calculate the X position of the cursor in accordance with the font
width: x=8X7=56.
    //Calculate the starting position of the cursor based on the starting
position of the character: y=30.
    //Calculate the width of the cursor in accordance with the font width:
width=8.
    //Calculate the height of the cursor in accordance with the font
height: height=16
    //API Prototype: void CursorGotoXY(x, y, width, height);
    LCD.CursorGotoXY(56, 30, 8, 16);

    //Open the cursor, and set the flicker cycle to 6-level.
    LCD.CursorConf(ON, 6);
    delay(5000);             //Delay for 5s.

    //To move the cursor point to decade of second.
    LCD.CursorGotoXY(48, 30, 8, 16);

    delay(5000);             //Delay for 5s.
    LCD.CursorConf(OFF, 6); //Turn off the cursor.
```
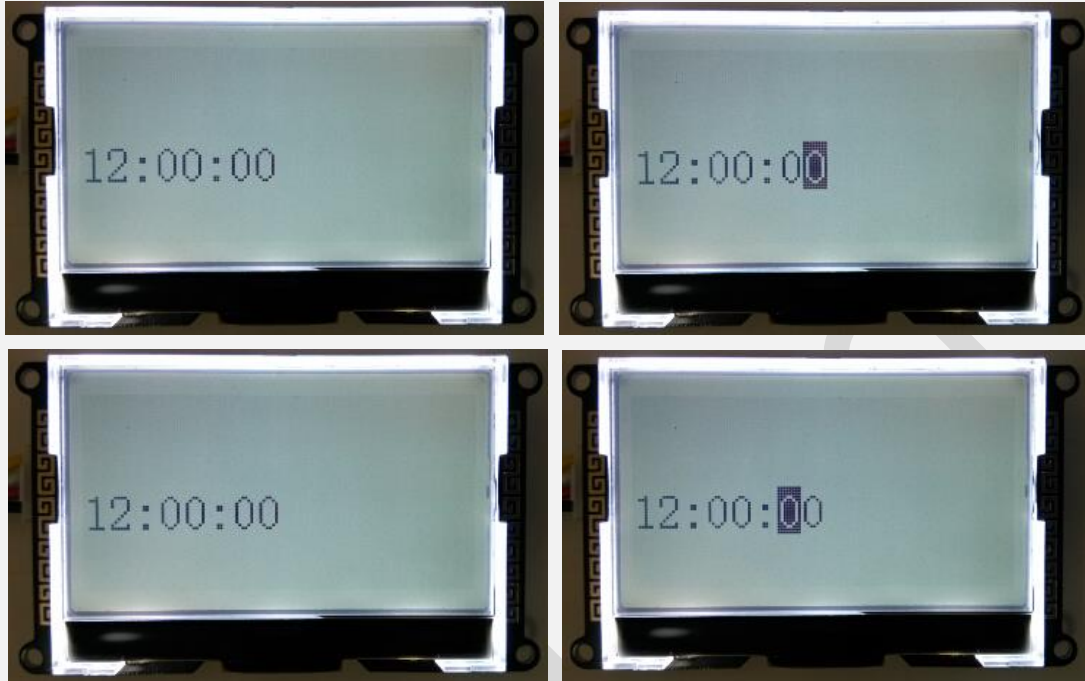
```
    while(1); //Wait for ever.
}
```

Operating Results:

3.2 Cursor Config API Instruction

This section describes the details of the cursor API functions, and show the usage of them.

The following table lists the API functions associated with the cursor in the I2C_LCD library for Arduino.

Function	Instruction
CursorConf( )	Cursor property configuration.
CursorGotoXY( )	Cursor position configuration.

3.2.1 CursorConf ( )

Instruction: Set the cursor's switch and the flicker frequency.

Function Prototype:

void CursorConf(enum LCD_SwitchState swi, uint8_t freq)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
swi	Cusor ON/OFF	OFF	Close the cursor, the default value.
		ON	Open the cursor.
freq	Cursor blink frequency.	0~15	16 stage cursor blink frequency setting.

Return: Null

Example: The cursor switch is set to ON, and the blink frequency is set to 6.

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

3.2.2 CursorGotoXY ( )

Instruction: Set the position of the cursor.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
x	Starting X-coordinates of the cursor.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Starting Y-coordinates of the cursor.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
width	Cursor width of X-axis direction.	0~127	The maximum value of (X+width) is 127.
height	Cursor height of Y-axis direction.	0~63	The maximum value of (Y+height) is 63.

Return: Null

Example: The starting position of the cursor is set to (0,0), the width of the cursor is set to 20, the height of the cursor is set to 8.

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

Chapter 4. Drawing 2D Graphics

In order to meet the needs of the user interface design, I2C_LCD integrates the functions of 2D graphics rendering, it can draw points, lines, circles and rectangles. To achieve these graphics rendering, you only need to call the API functions provided by the library. For example, to draw a circle, pass the center coordinates and radius parameters to an API function, you will achieve a circle drawing, users do not involve complex graphics algorithm, greatly reduce the difficulty.

Since the graphics related algorithms are running in the I2C_LCD independent controller, it can save the ROM and RAM resources of the user controller.

This chapter describes how to use I2C_LCD library to quickly draw 2D graphics with API functions.

4.1 Sample Project Of 2D Graphics

4.1.1 Drawing points and lines

4.1.1.1 Sample 1

Objective: Demonstrate drawing lines in different colors.

Path: UserManual\DemoCode\Sec_4111_Graphic

Steps:

Erase full screen with white background color;

Delay for 1s;

Draw a black horizontal line from (0, 20) to (127, 20) ;

Delay for 2s;

Draw a black vertical line from (20, 0) to (20, 63) ;

Delay for 2s;

Draw a black dot at (63, 50);

Delay for 2s;

Draw a tilt black line from (0, 0) to (127, 63) ;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();           //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Draw a black horizontal line.
    //Prototype: void DrawHLineAt(startX, endX, y, color)
    LCD.DrawHLineAt(0, 127, 20, BLACK);
    delay(2000);             //Delay for 2s.

    //Draw a black vertical line.
    //Prototype: void DrawVLineAt(startY, endY, x, color)
    LCD.DrawVLineAt(0, 63, 20, BLACK);
    delay(2000);             //Delay for 2s.

    //Draw a black point.
    //Prototype: void DrawDotAt(x, y, color)
    LCD.DrawDotAt(63, 50, BLACK);
    delay(2000);             //Delay for 2s.

    //Draw any black line.
    //Prototype: void DrawLineAt(startX, endX, startY, endY, color)
    LCD.DrawLineAt(0, 127, 0, 63, BLACK);

    while(1); //Wait for ever.
}
```

Operating Results:
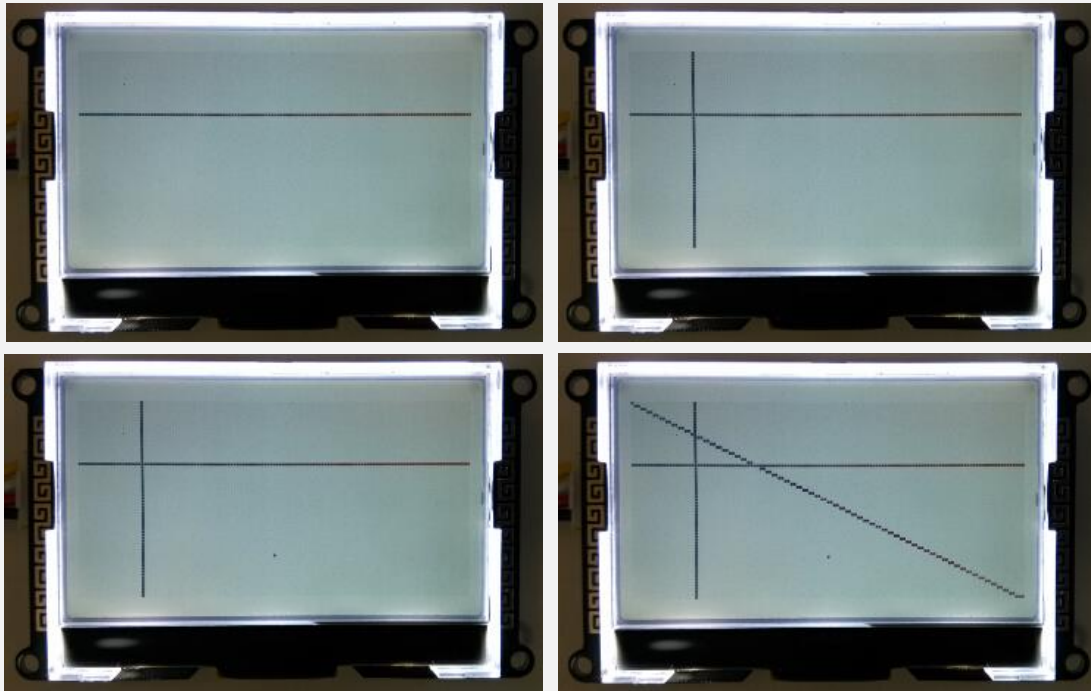
4.1.1.2 Sample 2

Objective: Demonstrate drawing lines in different color.

Path: UserManual\DemoCode\Sec_4112_Graphic

Steps:

Erase full screen with black background color;

Delay for 1s;

Draw a white horizontal line from (0, 20) to (127, 20) ;

Delay for 2s;

Draw a white vertical line from (20, 0) to (20, 63) ;

Delay for 2s;

Draw a white dot at (63, 50);

Delay for 2s;

Draw a tilt white line from (0, 0) to (127, 63) ;

Code:

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

```
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(BLACK);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Draw a white horizontal line.
    //Prototype: void DrawHLineAt(startX, endX, y, color)
    LCD.DrawHLineAt(0, 127, 20, WHITE);
    delay(2000);             //Delay for 2s.

    //Draw a white vertical line.
    //Prototype: void DrawVLineAt(startY, endY, x, color)
    LCD.DrawVLineAt(0, 63, 20, WHITE);
    delay(2000);               //Delay for 2s.

    //Draw a white point.
    //Prototype: void DrawDotAt(x, y, color)
    LCD.DrawDotAt(63, 50, WHITE);
    delay(2000);             //Delay for 2s.

    //Draw any white line.
    //Prototype: void DrawLineAt(startX, endX, startY, endY, color)
    LCD.DrawLineAt(0, 127, 0, 63, WHITE);

    while(1); //Wait for ever.
}
```
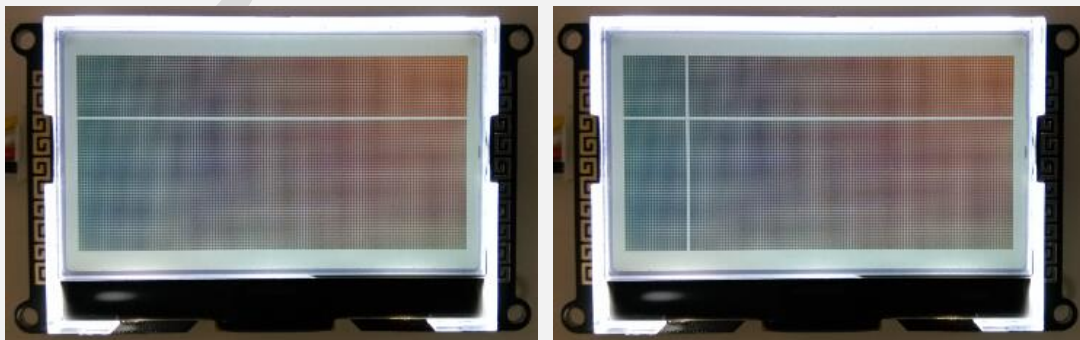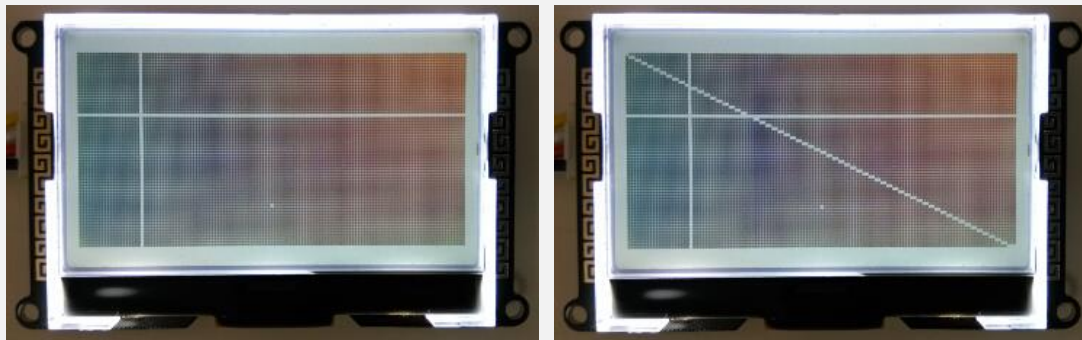
Operating Results:

4.1.2 Drawing rectangle and circle

4.1.2.1 Sample 1

Objective: Demonstrate drawing rectangle and circle in different colors.

Path: UserManual\DemoCode\Sec_4121_Graphic

Steps:

Erase full screen with white background color;

Delay for 1s;

Draw a 107-width, 33-height rectangle at (10, 15), and filled with black;

Delay for 2s;

Draw a circle with a radius of 30 at (63, 31), and filled with black.

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
```
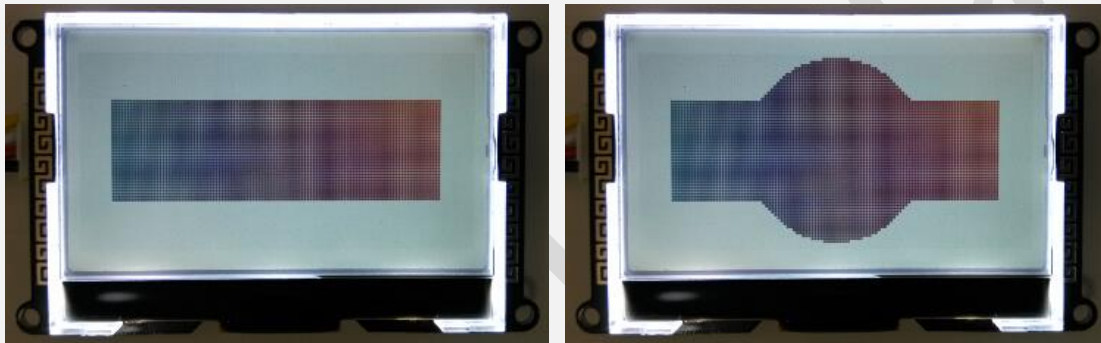
```
        LCD.CleanAll(WHITE);    //Erase all.
        delay(1000);            //Delay for 1s.

        //Draw a rectangle, and filled with black;
        //Prototype: void DrawRectangleAt(x, y, width, height, mode)
        LCD.DrawRectangleAt(10, 15, 107, 33, BLACK_FILL);
        delay(2000);            //Delay for 2s.

        //Draw a circle, and filled with black;
        //Prototype: void DrawCircleAt(x, y, r, mode)
        LCD.DrawCircleAt(63, 31, 30, BLACK_FILL);

        while(1); //Wait for ever.
}
```

Operating Results:

4.1.2.2   Sample 2

Objective: Demonstrate drawing rectangle and circle in different colors.

Path: UserManual\DemoCode\Sec_4122_Graphic

Steps:

Erase full screen with black background color;

Delay for 1s;

Draw a 107-width, 33-height rectangle at (10, 15), and filled with white;

Delay for 2s;

Draw a circle with a radius of 30 at (63, 31), and filled with white.

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(BLACK);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Draw a rectangle, and filled with white;
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(10, 15, 107, 33, WHITE_FILL);
    delay(2000);             //Delay for 2s.

    //Draw a circle, and filled with white;
    //Prototype: void DrawCircleAt(x, y, r, mode)
    LCD.DrawCircleAt(63, 31, 30, WHITE_FILL);

    while(1); //Wait for ever.
}
```
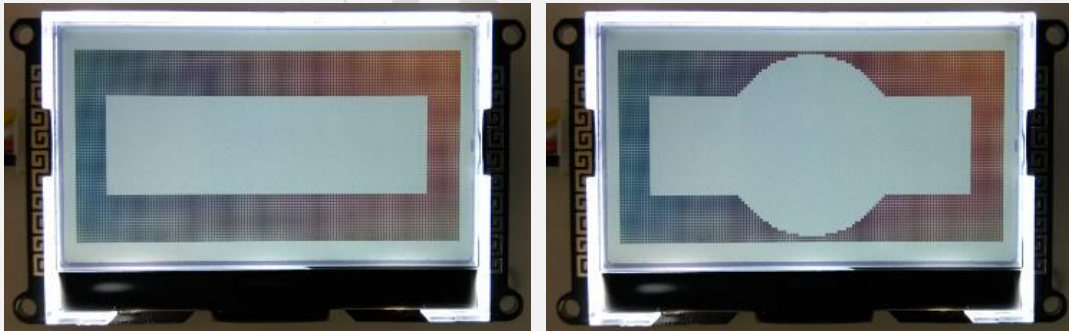
Operating Results:

4.1.2.3   Sample 3

Objective: Demonstrate the method and color setting of drawing rectangle and circle.

Path: UserManual\DemoCode\Sec_4123_Graphic

Steps:

Erase full screen with white background color;

Delay for 1s;

Draw a circle with a radius of 50 at (63, 31), and filled with black;

Delay for 2s;

Draw a 60-width, 20-height white rectangle at (33, 21), and without fill;

Delay for 2s;

Draw a 52-width, 12-height rectangle at (37, 25), and filled with white.

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Draw a circle, and filled with black.
    //Prototype: void DrawCircleAt(x, y, r, mode)
    LCD.DrawCircleAt(63, 31, 50, BLACK_FILL);
    delay(2000);             //Delay for 2s.

    //Draw a white rectangle, and without fill.
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(33, 21, 60, 20, WHITE_NO_FILL);
    delay(2000);             //Delay for 2s.

    //Draw a rectangle, and filled with white.
    //Prototype: void DrawRectangleAt(x, y, width, height, mode)
    LCD.DrawRectangleAt(37, 25, 52, 12, WHITE_FILL);

    while(1); //Wait for ever.
}
```
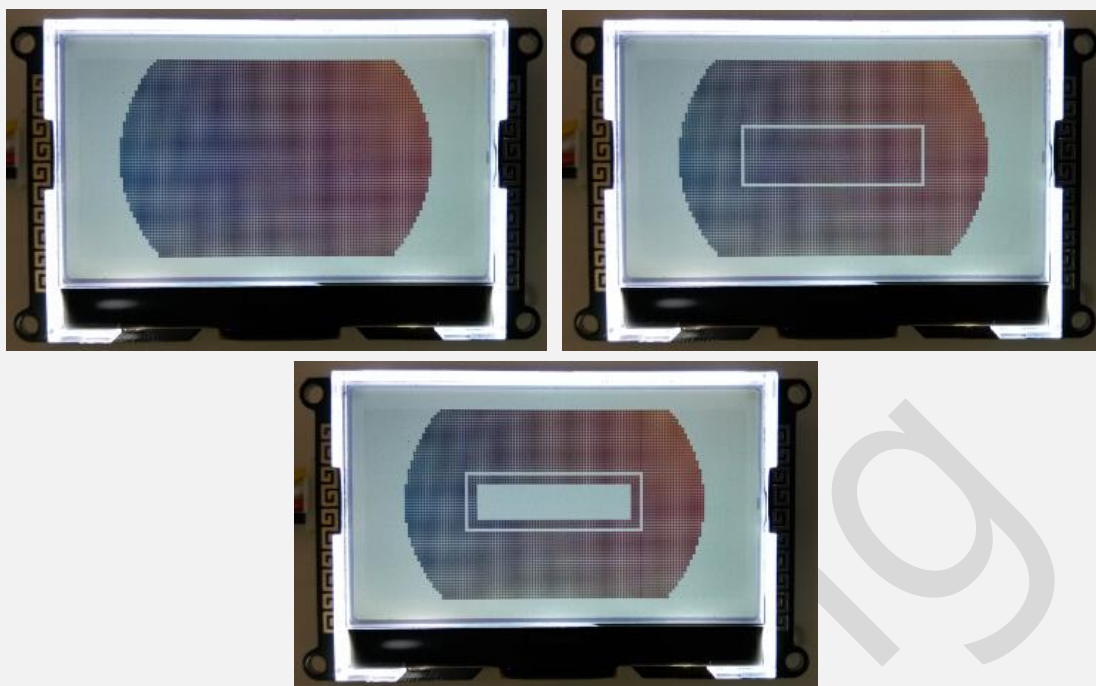
Operating Results:

4.2 2D Graphics API Instruction

This section describes the details of the 2D graphics API functions, and show the usage of them.

The following table lists the API functions associated with the 2D graphics in the I2C_LCD library for Arduino.

Function	Instruction
DrawDotAt( )	At the specified location, use the specify color to painting point.
DrawHLineAt( )	At the specified location, use the specify color to painting horizontal line.
DrawVLineAt( )	At the specified location, use the specify color to painting vertical line.
DrawLineAt( )	At the specified location, use the specify color to painting any line.
DrawRectangleAt( )	At the specified location, use the specify mode to painting rectangle.
DrawCircleAt( )	At the specified location, use the specify mode to painting circle.

4.2.1 DrawDotAt ( )

Instruction: At the specified location, use the specify color to painting a point.

Function Prototype:

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

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
x	X-coordinates of the points.	0~127	The total number of points on the X-axis is 128, encoding 0~127.

y	Y-coordinates of the points.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
color	The color of point.	WHITE	White point.
		BLACK	Black point.

Return: Null

Example: 在 Draw a black dot at (0, 0).

<div align="center">

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

</div>

4.2.2 DrawHLineAt ( )

Instruction: Draw a horizontal line use the specify color.

Function Prototype:

void DrawHLineAt(uint8_t startX, uint8_t endX,

uint8_t y, enum LCD_ColorSort color)

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
startX	Starting X-coordinates of the line.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
endX	Ending X-coordinates of the line.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Y-coordinate of the line.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
color	The color of line.	WHITE	White.
		BLACK	Black.

Return: Null

Example: Draw a black horizontal line from (0, 0) to (127, 0).

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

4.2.3 DrawVLineAt ( )

Instruction: Draw a vertical line use the specify color.

Function Prototype:

void DrawVLineAt(uint8_t startY, uint8_t endY,

uint8_t x, enum LCD_ColorSort color)

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
startY	Starting Y-coordinates of the line.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
endY	Ending Y-coordinates of the line.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
x	X-coordinate of the line.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
color	The color of line.	WHITE	White.
		BLACK	Black.

Return: Null

Example: Draw a black vertical line from (0, 0) to (0, 63).

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

4.2.4 DrawLineAt ( )

Instruction: Draw a line with the specified color from start point to the destination.

Function Prototype:

void DrawLineAt(uint8_t startX, uint8_t endX,

uint8_t startY, uint8_t endY, enum LCD_ColorSort color);

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
startX	Starting X-coordinates of the line.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
endX	Ending X-coordinates of the line.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
startY	Starting Y-coordinates of the line.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
endY	Ending Y-coordinates of the line.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
color	The color of the line.	WHITE	White.
		BLACK	Black.

Return: Null

Example: Draw a black slash from (0, 0) to (127, 63) .

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

4.2.5 **DrawRectangleAt ( )**

Instruction: Draw a rectangle with the specified parameters.

Function Prototype:

void DrawRectangleAt(uint8_t x, uint8_t y, uint8_t width,

uint8_t height, enum LCD_DrawMode mode);

Parameter	ParameterInstruction	Acceptable Value	Value Instruction

		WHITE_NO_FILL	White without filling.

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
x	Starting X-coordinates of the rectangle.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Starting Y-coordinates of the rectangle.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
width	X-axis direction width.	0~127	The maximum value of (X+width) is 127.
height	Y-axis direction height.	0~63	The maximum value of (Y+height) is 63.
mode	The color and fill mode of the rectangle.	WHITE_NO_FILL	White without filling.
		WHITE_FILL	White with filling.
		BLACK_NO_FILL	Black without filling.
		BLACK_FILL	Black with filling.

Return: Null

Example: Draw a 60-width, 40-height black rectangle at (0, 0), and without fill.

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

4.2.6 DrawCircleAt ( )

Instruction: Draw a circle with the specified parameters.

Function Prototype:

void DrawCircleAt(int8_t x, int8_t y, uint8_t r,

enum LCD_DrawMode mode);

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
x	The center X-coordinates of the circle.	0~127	The total number of points on the X-axis is 128, encoding 0~127.

y	The center Y-coordinates of the circle.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.
r	Radius of the circle.	0~127	The maximum value of r is 127.
mode	The color and fill mode of the circle.	WHITE_NO_FILL	White without filling.
		WHITE_FILL	White with filling.
		BLACK_NO_FILL	Black without filling.
		BLACK_FILL	Black with filling.

Return: Null

Example: Draw a circle with a radius of 25 at (60, 30), and filled with black.

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

Chapter 5. Bitmap Display

In order to enrich the user interface elements, I2C_LCD integrated the bitmap display function, you can display any image on the I2C_LCD (Maximum support for 128*64 dual color pictures).

This chapter describes how to use the API function to achieve desired picture display.

5.1 Sample Project Of Bitmap Display

5.1.1 Display a bitmap

5.1.1.1 Sample 1

Objective: Demonstrate how to display a bitmap.

Path: UserManual\DemoCode\Sec_5111_Bitmap

Steps:

Erase full screen with white background color;

Delay for 1s;

Switch I2C_LCD work mode to drawing mode;

Display the bitmap of Tuzki(Tuzki.bmp) at (30, 0), details of picture data packet generation, please refer to section 5.3;

Delay for 5s;

Display "Sparking" logo at (0, 0);

Code:

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

```
#include <I2C_LCD.h>

I2C_LCD LCD;
extern GUI_Bitmap_t bmTuzki;        //Declare bitmap data packet.
extern GUI_Bitmap_t bmSPLogo;       //Declare bitmap data packet.
uint8_t I2C_LCD_ADDRESS = 0x51;   //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);              //Delay for 1s.

    //Booting logo ON, backlight ON, bitmap work mode.
    //If you want to display characters please switch to WM_CharMode.
    LCD.WorkingModeConf(ON, ON, WM_BitmapMode);

    //Display bitmap at the specified location.
    //For details about bitmap display, please refer to the 5.3 section of
user manual.
    LCD.DrawScreenAreaAt(&bmTuzki, 30, 0);
    delay(5000);                 // Delay for 5s.

    //Display bitmap at the specified location.
    LCD.DrawScreenAreaAt(&bmSPLogo, 0, 0);

    while(1); //Wait for ever.
}
```

Operating Results:

5.2 Bitmap Display API Instruction

This section describes the details of the bitmap display API functions, and show the usage of them.

The following table lists the API functions associated with the bitmap display in the I2C_LCD library for Arduino.

Function	Instruction
DrawScreenAreaAt( )	Display bitmap at the specified location use the bitmap data package.

5.2.1 DrawScreenAreaAt ( )

Instruction: Display bitmap at the specified location.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
*bitmap	A pointer to the picture data package.	GUI_Bitmap_t structure data type address.	Data package generated by Bitmap Converter.
x	Starting X-coordinate of the bitmap.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	Starting Y-coordinate of the bitmap.	0~63	The total number of points on the Y-axis is 64, encoding 0~63.

Return: Null

Example: Display the bitmap at (20, 10).

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

Note: bmTuzki is the picture data packet pointer, the data packet contains the bitmap size, the pixel data and so on. The data packet generated by Bitmap Converter software, the detailed steps please refer to 5.3 section.

5.3 Picture Data Packet Generation

This section will introduce the method of generating a data packet file for any image. Our team has developed the Converter Bitmap software for the user, the software is now supporting Mac, OS Windows, Linux. Users only need a few steps to display the picture on I2C_LCD .

In this section, we will generate the data packet of Tuzki_1.bmp, and display it on I2C_LCD. (The picture file is under UserManual\DemoCode\BitmapDisplay directory.)

1. *Users can edit the bitmap by software (Photoshop, mspaint tool for windows, and so on). Due to the I2C_LCD can display only black and white dual color content. Therefore, it is suggested to edit the bitmap using only black and white , or may affect the final display effects (color bitmap will automatically be converted to black and white by BitmapConverter software).

2. *Save the bitmap which you edited. (supported by type: *.bmp, *.jpeg, *.jpg);

Note: Do not contains the operators and special symbols("+-*\") in the name of the file, otherwise it will can't generate data packets.
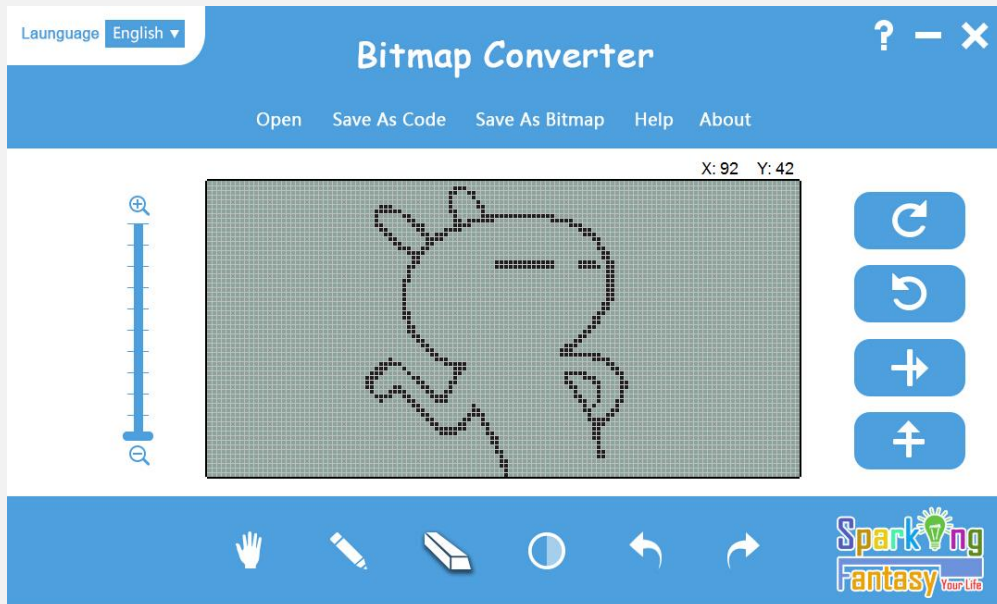
3. Running the BitmapConverter software, click "Open" to browse and open "Tuzki_1.bmp";
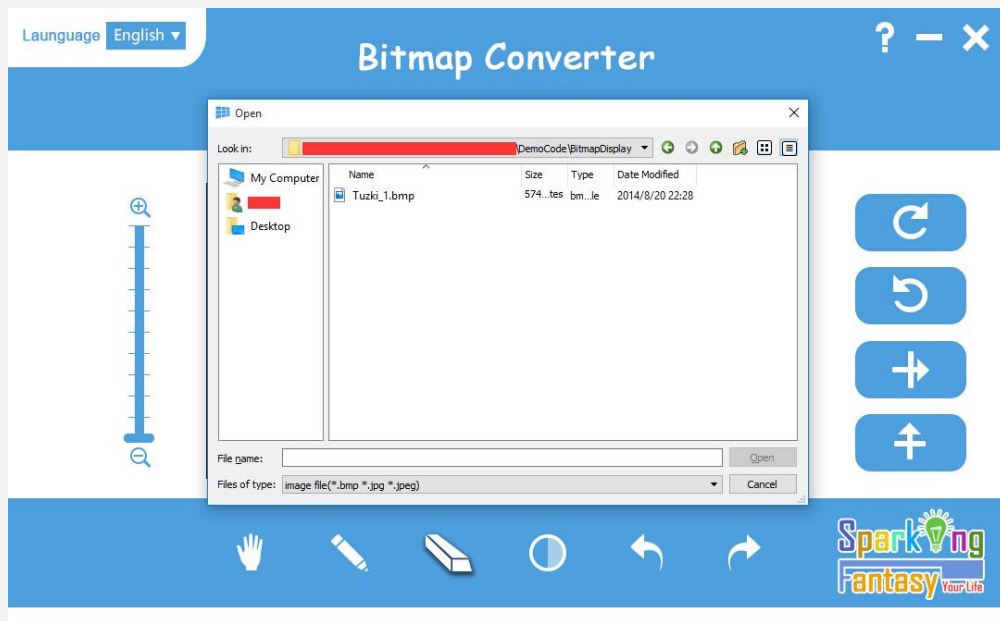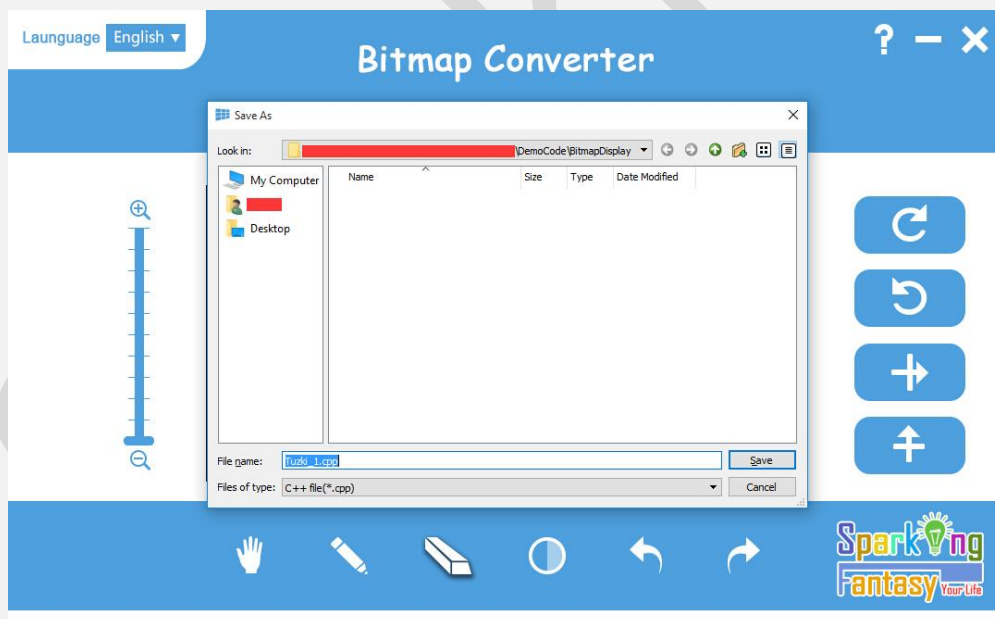
4. Use the cutting tool to select the final desired image area, and then click OK button, as shown below;

5. Pictures have been loaded into the editing area, you can use eraser, pencil and other tools to edit the bitmap, removal of burr pixels.

6. Then, click "Save As Code" button, select the path you want to save to, here is: UserManual\DemoCode\BitmapDisplay;
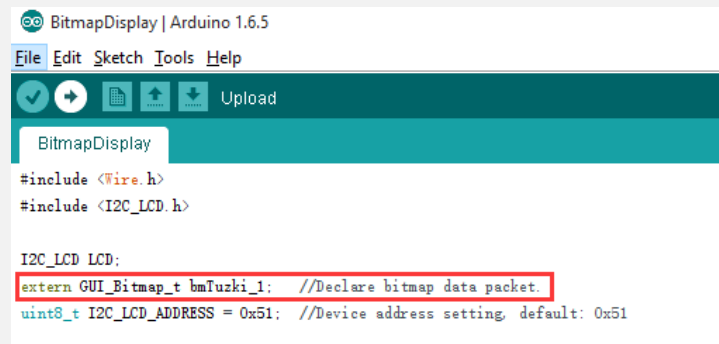

7. Open "BitmapDisplay.ino" under the directory:

UserManual\DemoCode\BitmapDisplay

Then declare the data packet("bmTuzki_1") at the beginning of the project;

Note: if you have opened the project, please close it and then open it again;

8. Modify the bitmap display function, change the data packet pointer, and starting coordinates;

Note: For details of bitmap display function, please refer to 5.2.1 section;
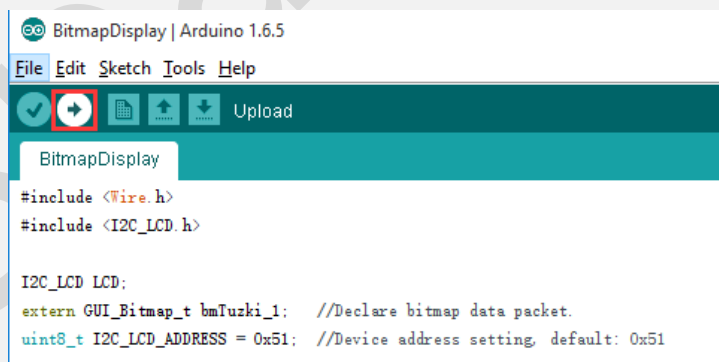

9. Click the "Upload" button to upload the program to Arduino board, then enjoy yourself.

Chapter 6. *Memory Read / Write operation

This chapter for advanced users reading content, mainly introduces to read and write I2C_LCD display memory directly by API function. These features are designed to meet the special needs of advanced users, ordinary users please skip this chapter.

LCD I2C design flexible, open, not only can meet the needs of ordinary users DIY, but also can satisfy the needs of advanced users. In order to meet the needs of users at different levels, It has special operation mode for advanced user , to achieve more advanced features.

This chapter mainly introduces how to read/write display memory directly by API functions.

6.1 Display Memory Operation Example

6.1.1 Single byte read / write

6.1.1.1 Sample 1

Objective: Read / write one byte of display memory.

Path: UserManual\DemoCode\Sec_6111_DisRam

Steps:

Erase full screen with white background color;

Delay for 1s;

Change the I2C_LCD operation mode to ram mode(WM_RamMode);

Write data 0xae(hex) to (0, 1);

Delay for 3s;

Read one byte of ram from (0, 1), and keep the value in the buf variable;

Print the value of buf to serial monitor;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.

    //Init serial port, and set the band rate as 115200.
    Serial.begin(115200);
}

void loop(void)
{
    uint8_t buf = 0;           //Buffer.

    LCD.CleanAll(WHITE);       //Erase all.
    delay(1000);               //Delay for 1s.

    //Booting logo ON, backlight ON, RAM work mode.
    //If you want to display characters please switch to WM_CharMode.
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //Write one byte ram to the specified location.
    //Prototype: void WriteByteDispRAM(buf, x, y);
    LCD.WriteByteDispRAM(0xAE, 0, 1);
    delay(3000);               //Delay for 3s.

    //Read one byte ram from the specified location.
    //Prototype:uint8_t ReadByteDispRAM(x, y)
    buf = LCD.ReadByteDispRAM(0, 1);

    //Print the data that just read from RAM to serial monitor.
    //Print in hex format.
    Serial.print("buf = 0x");
    Serial.println(buf, HEX);

    while(1); //Wait for ever.
```

}

Operating Results:

6.1.2 Write/Read multiple bytes of data

6.1.2.1 Sample 1

Objective: Write/Read multiple bytes of data to/from display ram;

Path: UserManual\DemoCode\Sec_6121_DisRam

Steps:

Erase full screen with white background color;

Delay for 1s;

Print the data (0xfa, 0xa5, 0xaf) to serial monitor that stored in the buf1 array;

Change the I2C_LCD operation mode to ram mode (WM_RamMode);

Write the data "0xFA, 0xAF, 0xA5" to (1, 1) that stored in buf1 array ;

Delay for 3s;

Read three bytes of ram from (1, 1), and keep the value in the buf2 array;

Print the data to serial monitor that stored in the buf2 array;

Code:

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

```
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.

    //Init serial port, and set the band rate as 115200.
    Serial.begin(115200);    }

void loop(void)
{
    uint8_t buf1[3] = {0xFA,0xAF,0xA5}; //Write buffer.
    uint8_t buf2[3] = {0};              //Read buffer.
    uint8_t adder;

    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Print the data stored in buf1 array to serial monitor.
    Serial.print("buf1[3] =");
    for(adder=0; adder<3; adder++)
    {
        //Print in hex format.
        Serial.print(" 0x");
        Serial.print(buf1[adder], HEX);
    }
    Serial.println("");      //Newline.

    //Booting logo ON, backlight ON, RAM work mode.
    //If you want to display characters please switch to WM_CharMode.
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //Starting from the specified location, continuous write multiple
bytes of data to ram.
    //Prototype: void WriteSeriesDispRAM(*buf, length, x, y)
    LCD.WriteSeriesDispRAM(buf1, 3, 1, 1);
    delay(3000);             //Delay for 3s.

    //Starting from the specified location, continuous read multiple bytes
of ram.
    //Prototype: void ReadSeriesDispRAM(*buf, length, x, y)
    LCD.ReadSeriesDispRAM(buf2, 3, 1, 1);

    ////Print the data stored in buf2 array to serial monitor.
    Serial.print("buf2[3] =");
    for(adder=0; adder<3; adder++)
    {
```

```
            //Print in hex format.
            Serial.print(" 0x");
            Serial.print(buf2[adder], HEX);
        }
        Serial.println("");      //Newline.

        while(1); //Wait for ever.
    }
```

Operating Results:

6.2 Ram Write/Read API Instruction

This section describes the details of the API function, and the usage of the function related to the ram write/read.

The following table lists the API functions associated with the ram write/read in I2C_LCD library for Arduino.

Function	Instruction
ReadByteDispRAM( )	Read one byte ram from the specified location.
WriteByteDispRAM( )	Write one byte ram to the specified location.
ReadSeriesDispRAM( )	Starting from the specified location, continuous read multiple bytes of ram.
WriteSeriesDispRAM( )	Starting from the specified location, continuous write multiple bytes to ram.

6.2.1 ReadByteDispRAM ( )

Instruction: Read one byte ram from the specified location.

Function Prototype:

uint8_t ReadByteDispRAM(uint8_t x, uint8_t y)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
x	The X-coordinate of where you want to read.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	The X-coordinate of where you want to read.	0~7	Vertical 8 points as a line, a total of 64/8=8 lines, encoding 0~7.

Return: One byte of ram just read out.

Example: Read one byte of ram from (0, 0), and keep the value in the buf variable.

Buf = LCD.ReadByteDispRAM(0, 0);

6.2.2 WriteByteDispRAM ( )

Instruction: Write one byte ram to the specified location.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
buf	The data you want to write.	0x00~0xff	One byte of data.
x	The X-coordinate of where you want to write.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	The Y-coordinate of where you want to write.	0~7	Vertical 8 points as a line, a total of 64/8=8 lines, encoding 0~7.

Return: Null

Example: Write one byte of ram to (0, 0).

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

6.2.3 **ReadSeriesDispRAM ( )**

Instruction: Starting from the specified location, continuous read multiple bytes of ram.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
*buf	Pointer to a storage area.	Address of storage area.	The name of array or a pointer to the storage area.
length	The num of bytes you want to read.	0~255	A maximum of 255 Byte data can be read one time.
x	The X-coordinate of where you want to read.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	The Y-coordinate of where you want to read.	0~7	Vertical 8 points as a line, a total of 64/8=8 lines, encoding 0~7.

Return: Null

Example: Read ten bytes of ram data from (0, 0), and store in buf[10] array;

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

6.2.4 **WriteSeriesDispRAM ( )**

Instruction: Starting from the specified location, continuous write multiple bytes of data to ram.

Function Prototype:

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

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
*buf	Pointer to a storage area.	Address of storage area.	The name of array or a pointer to the storage area.
length	The num of bytes you want to write.	0~255	A maximum of 255 Byte data can be write one time.
x	The X-coordinate of where you want to write.	0~127	The total number of points on the X-axis is 128, encoding 0~127.
y	The Y-coordinate of where you want to write.	0~7	Vertical 8 points as a line, a total of 64/8=8 lines, encoding 0~7.

Return: Null

Example: Continuous write 10 bytes of data to (0, 0), that stored in buf[10] array;

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

Chapter 7. System Configration

This chapter will talk about system setting of I2C_LCD, mainly comprises display mode, backlight brightness, boot logo on/off, work mode, screen contrast, device address, recovery factory settings.

This chapter will mainly introduce how to change the system setting by API function.

7.1 System Setup Example

7.1.1 Reverse/Normal display mode switch

7.1.1.1 Sample 1

Objective: Set I2C_LCD display mode to reverse, and then set back to normal.

Path: UserManual\DemoCode\Sec_7111_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_10x20, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Sparking" at (0, 20);

Delay for 3s;

Set I2C_LCD display mode to reverse;

Delay for 3s;

Set I2C_LCD display mode to normal;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);      //Erase all.
    delay(1000);              //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 20);     //Display string.
    delay(3000);              //Delay for 3s.

    //Set to reverse display mode.
    LCD.DisplayConf(AllREV);
    delay(3000);              //Delay for 3s.

    //Set to normal display mode.
    LCD.DisplayConf(AllNOR);

    while(1); //Wait for ever.
}
```

Operating Results:

7.1.2 **Work mode switch**

7.1.2.1 **Sample 1**

Objective: Switch to character or bitmap work mode.

Path: UserManual\DemoCode\Sec_7121_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Hello!" at (50, 10);

Delay for 3s;

Set I2C_LCD work mode to bitmap drawing mode;

Display bitmap of camer at (0, 0);

Delay for 3s;

Set I2C_LCD work mode back to character mode;

Display string "Director." at (50, 30);

Code:

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

```
I2C_LCD LCD;
extern GUI_Bitmap_t bmCamera;    //Declare bitmap data packet.
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();             //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);      //Erase all.
    delay(1000);              //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 50, 10);       //Display string.
    delay(3000);              //Delay for 3s.

    //Booting logo ON, backlight ON, bitmap work mode.
    //If you want to display characters please switch to WM_CharMode.
    LCD.WorkingModeConf(ON, ON, WM_BitmapMode);

    //Display bitmap at the specified location.
    //For details about bitmap display, please refer to the 5.3 section of
user manual.
    LCD.DrawScreenAreaAt(&bmCamera, 0, 0);
    delay(3000);              //Delay for 3s.

    //Booting logo ON, backlight ON, character work mode.
    LCD.WorkingModeConf(ON, ON, WM_CharMode);
    LCD.DispStringAt("Director.", 50, 30);

    while(1); //Wait for ever.
}
```

Operating Results:

7.1.2.2 Sample 2

Objective: Switch to character or ram operate work mode.

Path: UserManual\DemoCode\Sec_7122_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Hello!" at (50, 10);

Delay for 3s;

Set I2C_LCD work mode to ram operate mode (for advanced user);

Write the data "0xFA, 0xAF, 0xA5" to (20, 2) that stored in buf array ;

Delay for 3s;

Set I2C_LCD work mode back to character mode;

Display string "Director." at (50, 30);

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51
```

```
void setup(void)
{
    Wire.begin();           //I2C controler init.
}

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

    LCD.CleanAll(WHITE);    //Erase all.
    delay(1000);            //Delay for 1s.

    //Set the font, character address update mode, display mode.
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 50, 10);     //Display string.
    delay(3000);            //Delay for 3s.

    //Booting logo ON, backlight ON, RAM work mode.
    //If you want to display characters please switch to WM_CharMode.
    LCD.WorkingModeConf(ON, ON, WM_RamMode);

    //Starting from the specified location, continuous write multiple
bytes of data to ram.
    //Prototype: void WriteSeriesDispRAM(*buf, length, x, y)
    LCD.WriteSeriesDispRAM(buf, 3, 20, 2);
    delay(3000);            //Delay for 3s.

    //Booting logo ON, backlight ON, character work mode.
    LCD.WorkingModeConf(ON, ON, WM_CharMode);
    LCD.DispStringAt("Director.", 50, 30);  //Display string.

    while(1); //Wait for ever.
}
```

Operating Results:

7.1.3 **Backlight and boot logo display on/off**

7.1.3.1 **Sample 1**

Objective: Switch on/off the backlight and boot logo display.

Path: UserManual\DemoCode\Sec_7131_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Hello!" at (20, 20);

Delay for 2s;

Turn off the backlight and boot logo display;

Press the reset button of I2C_LCD, I2C_LCD will keep the configuration just now, because the configuration was saved in EEPROM;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
```

```
{
    Wire.begin();           //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);    //Erase all.
    delay(1000);            //Delay for 1s.

    //Set the font, character address update mode, display mode.
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 20, 20);    //Display string.
    delay(2000);            //Delay for 2s.

    //Booting logo OFF, backlight OFF, character work mode.
    LCD.WorkingModeConf(OFF, OFF, WM_CharMode);

    while(1); //Wait for ever.
}
```
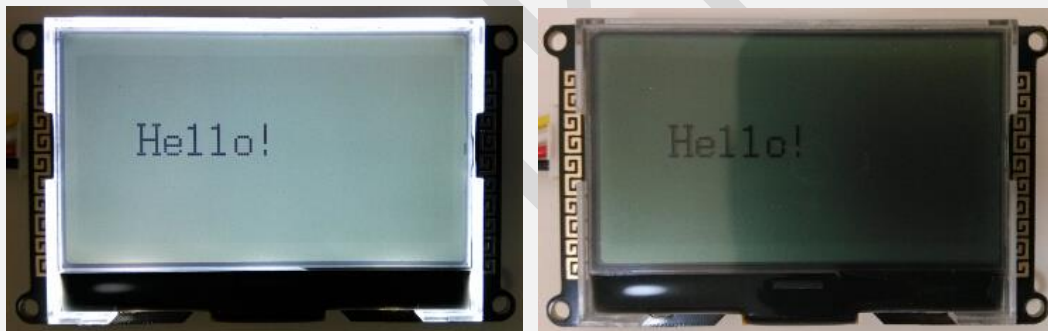
Operating Results:

7.1.3.2 Sample 2

Objective: Switch on/off the backlight and boot logo display.

Path: UserManual\DemoCode\Sec_7132_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_8x16_1, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Hello!" at (20, 20);

Delay for 2s;

Turn on the backlight and boot logo display;

Press the reset button of I2C_LCD, I2C_LCD will keep the configuration just now, because the configuration was saved in EEPROM;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);      //Erase all.
    delay(1000);              //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_8x16_1, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Hello!", 20, 20);     //Display string.
    delay(2000);              //Delay for 2s.

    //Booting logo ON, backlight ON, character work mode.
    LCD.WorkingModeConf(ON, ON, WM_CharMode);

    while(1); //Wait for ever.
}
```
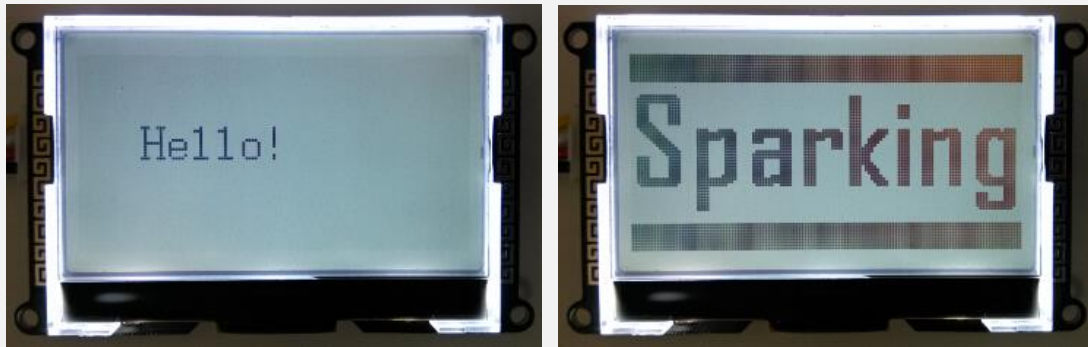
Operating Results:

7.1.4 **Backlight adjustment**

7.1.4.1 **Sample 1**

Objective: Adjust the backlight brightness of I2C_LCD, the configuration will recover when reboot.

Path: UserManual\DemoCode\Sec_7141_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Switch on the backlight;

Set the backlight brightness to 20, and save the configuration to ram;

Press the reset button of I2C_LCD, the brightness will recover to the configuration before, because the configuration was saved in ram;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}
```
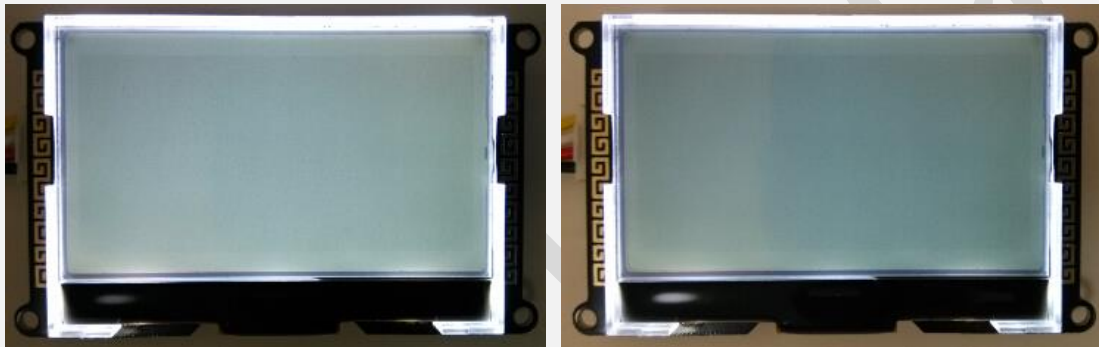
```
void loop(void)
{
    LCD.CleanAll(WHITE);      //Erase all.
    delay(1000);              //Delay for 1s.

    //Booting logo ON, backlight ON, character work mode.
    LCD.WorkingModeConf(ON, ON, WM_CharMode);

    //Set the backlight brightness to 20, and save the configuration to
ram.
    LCD.BacklightConf(LOAD_TO_RAM, 20);

    while(1); //Wait for ever.
}
```

Operating Results:

7.1.5   Contrast adjustment

7.1.5.1   Sample 1

Objective: Adjust the contrast of I2C_LCD, and the configuration will recover when reboot.

Path: UserManual\DemoCode\Sec_7151_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_10x20, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Sparking" at (0, 20);

Delay for 2s;

Set the contrast to 12, and save the configuration to ram;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);      //Erase all.
    delay(1000);              //Delay for 1s.

    //Set the font, character address update mode, display mode.
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 20);    //Display string.
    delay(2000);              //Delay for 2s.

    //Set the contrast to 12, and save the configuration to ram.
    LCD.ContrastConf(LOAD_TO_RAM, 12);

    while(1); //Wait for ever.
}
```

Operating Results:

7.1.6 Modify the device address

7.1.6.1 Sample 1

Objective: Modify the device address of I2C_LCD, and the configuration will save to EEPROM.

Path: UserManual\DemoCode\Sec_7161_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_10x20, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Sparking" at (0, 10);

Delay for 2s;

Set the device address to 0x52;

Try to display "Hello!" on (0, 30);

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51
//uint8_t I2C_LCD_ADDRESS = 0x52; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();            //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Set the font, character address update mode, display mode.
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
```

```
        LCD.DispStringAt("Sparking", 0, 10);    //Display string.
        delay(2000);               //Delay for 2s.

        //Modify the device address to 0x52, the default setting is 0x51.
        LCD.DeviceAddrEdit(0x52);

        //I want to display character, but because the device address is
        //different from the program current address setting,
        //so it can't work properly.
        LCD.DispStringAt("Hello!", 0, 30);

        //I2C_LCD_ADDRESS variable is the device address of the current
program,
        //after the value of the I2C_LCD_ADDRESS variable is changed to 0x52
as below, the I2C_LCD can work normally.
        //I2C_LCD_ADDRESS = 0x52;
        //LCD.DispStringAt("Hello!", 0, 30);

        //If you have forgotten the I2C_LCD device address, you can recover to
the factory settings: 0x51.
        //For details, please refer to the 8.3 section of the user manual.
        while(1); //Wait for ever.
    }
```
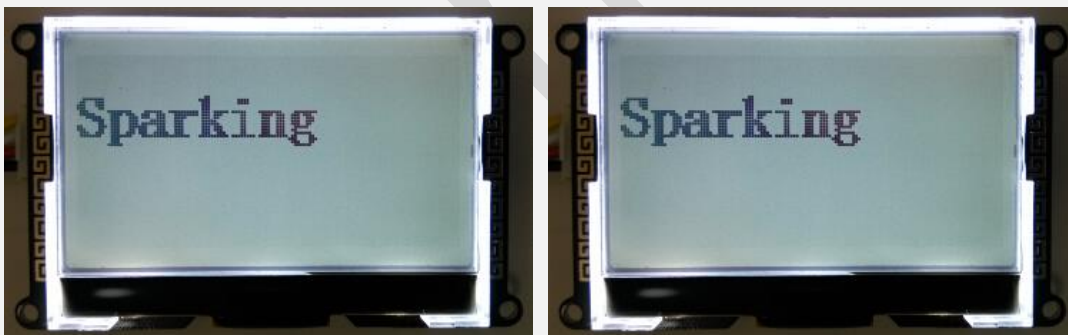
Operating Results:

7.1.7 Clean the screen

7.1.7.1 Sample 1

Objective: Erase the entire screen.

Path: UserManual\DemoCode\Sec_7171_System

Steps:

Erase full screen with white background color;

Delay for 1s;

Set the font to Font_10x20, the character address update mode is set to FM_ANL_AAA mode, the character display mode is set to BLACK_BAC;

Display string "Sparking" at (0, 10);

Delay for 2s;

Clean up the entire screen use black as the background color;

Delay for 2s;

Clean up the entire screen use white as the background color;

Code:

```
#include <Wire.h>
#include <I2C_LCD.h>
I2C_LCD LCD;
uint8_t I2C_LCD_ADDRESS = 0x51; //Device address setting, default: 0x51

void setup(void)
{
    Wire.begin();              //I2C controler init.
}

void loop(void)
{
    LCD.CleanAll(WHITE);     //Erase all.
    delay(1000);             //Delay for 1s.

    //Set the font, character address update mode, display mode.
    //FM_ANL_AAA: FM_AutoNewLine_AutoAddrAdd
    LCD.FontModeConf(Font_10x20, FM_ANL_AAA, BLACK_BAC);
    LCD.DispStringAt("Sparking", 0, 10);    //Display string.
    delay(2000);             //Delay for 2s.

    LCD.CleanAll(BLACK);     //Erase all use black background color.
    delay(2000);             //Delay for 2s.

    LCD.CleanAll(WHITE);     //Erase all use white background color.

    while(1); //Wait for ever.
}
```

Operating Results:

7.2 Macro And Variable Instruction

The following table lists the macro definitions and global variables in I2C_LCD library.

When you do not need the graphic API function, you can set the definition to "FALSE" at the beginning of "I2C I2C_LCD.h" file. The compiler will no longer compile this part of API function, this can saving the user controller ROM resources. Conversely, if you need to use this part of the API function, you need to set the definition to "TRUE".

I2C_LCD_ADDRESS is the device address of the current program, if the user modifies the I2C device address, also need to set the I2C_LCD_ADDRESS to the same value, in order to continue to use the I2C_LCD.

Variable Or Macro	Type	Acceptable Value	Value Instruction
SUPPORT_2D_GRAPHIC_LIB	Macro definitions	FALSE	No support 2D graphic library.
		TRUE, the default value.	Support 2D graphic library.

		0~127	
I2C_LCD_ADDRESS	uint8_t	(0x00~0x7f), Default value: 0x51	The current device address.

7.3 System Configration API Instruction

This section describes the details of the system configration API functions, and the usage of the function related to system configration.

The following table lists the API functions associated with system configration in I2C_LCD library for Arduino.

Function	Instruction
DisplayConf( )	Configure the display mode of the screen, normal/reverse.
WorkingModeConf( )	Configure the backlight on/off, boot logo on/off and work mode.
BacklightConf( )	Configure the backlight brightness and parameter save mode.
ContrastConf( )	Configure the contrast brightness and parameter save mode.
DeviceAddrEdit( )	Change the device address of I2C_LCD.
CleanAll( )	Erase full screen with the specified color (black / white).

7.3.1 DisplayConf ( )

Instruction: Configure the display mode of the screen.

Function Prototype:

void DisplayConf(enum LCD_DisplayMode mode)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
mode	Display mode.	AllREV	Reverse display mode, white change to black, and black change to white.
		AllNOR	Normal display mode, default value.

Return: Null

Example: Set display mode to "AllREV".

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

7.3.2 **WorkingModeConf ( )**

Instruction: Configure the backlight on/off, boot logo on/off and work mode.

Function Prototype:

void WorkingModeConf(enum LCD_SwitchState logoSwi,

enum LCD_SwitchState backLightSwi,

enum LCD_WorkingMode mode)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
logoSwi	Boot logo on/off.	OFF	Turn off the boot logo display, and save the configuration to EEPROM.
		ON	Turn on the boot logo display, and save the configuration to EEPROM.
backLightSwi	Backlight on/off.	OFF	Turn off the backlight, and save the configuration to EEPROM.
		ON	Turn on the backlight, and save the configuration to EEPROM.
mode	Work mode.	WM_CharMode	Character work mode.

		WM_BitmapMode	Bitmap work mode.
		WM_RamMode	Ram operate work mode.

Return: Null

Example: Set boot logo switch to "ON", set backlight switch to "OFF", and set work mode to "WM_CharMode".

LCD.WorkingModeConf(ON, OFF, WM_CharMode);

7.3.3 BacklightConf ( )

Instruction: Configure the backlight brightness, and configuration save mode.

Function Prototype:

void BacklightConf(enum LCD_SettingMode mode, uint8_t buf)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
mode	Configuration save mode.	LOAD_TO_RAM	Configuration save to ram, and discard when reboot.
		LOAD_TO_EEPROM	Configuration save to EEPROM, and remain when reboot.
buf	Backlight brightness set.	0~127	128 levels of backlight brightness, default 100.

Return: Null

Example: Set the configuration save mode to "LOAD_TO_RAM", and set the backlight brightness level to 20.

LCD.BacklightConf(LOAD_TO_RAM, 20);

7.3.4 ContrastConf ( )

Instruction: 配 Configure the contrast, and configuration save mode.

Function Prototype:

void ContrastConf(enum LCD_SettingMode mode, uint8_t buf)

Parameter	ParameterInstruction	Acceptable Value	Value Instruction
mode	Configuration save mode.	LOAD_TO_RAM	Configuration save to ram, and discard when reboot.
		LOAD_TO_EEPROM	Configuration save to EEPROM, and remain when reboot.
buf	Contrast set.	0~63	64 levels of contrast, default 21.

Return: Null

Example: Set the configuration save mode to "LOAD_TO_RAM", and set the contrast to 10.

LCD.ContrastConf(LOAD_TO_RAM, 10);

7.3.5   DeviceAddrEdit ( )

Instruction: Modify the device address of I2C_LCD.

Function Prototype:

void DeviceAddrEdit(uint8_t newAddr)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
newAddr	The new device address.	0~127	The new device address of I2C_LCD, and take effect immediately.

Return: Null

Example: Set "0x51" as the new device address.

LCD.DeviceAddrEdit(0x51);

7.3.6 **CleanAll ( )**

Instruction: Erase full screen with the specified color (black / white).

Function Prototype:

void CleanAll(enum LCD_ColorSort color)

Parameter	Parameter Instruction	Acceptable Value	Value Instruction
color	The color which to use.	WHITE	Erase full screen with white background color.
		BLACK	Erase full screen with black background color.

Return: Null

Example: Erase full screen with black background color.

LCD.CleanAll(BLACK);

Chapter 8. Troubleshooting And Recovery

This chapter will mainly introduce how to solve the problem when encountered.

8.1 FAQ

When you encountered problems, you can find the method to solve the problem by searching the following fault list.

Failure Phenomenon	Elimination Procedure
The screen is black, without any reaction.	1. Please make sure the power supply is 5V.
	2. Please refer to the 8.3 section, recover to the factory setting.
I2C_LCD is uncontrolled.	1. Press the RST button to retoot.
	2. Problem with connection, please check the connection is correctly, or try to use another Grove cable.
	3. Please refer to the 8.3 section, recover to the factory setting.
	4. Please make sure the value of "I2C_LCD_ADDRESS" variable is set to the default 0x51 in your program.
Forgot the device address of I2C_LCD.	1. Please refer to the 8.3 section, recover to the factory setting.

Note: If the above steps are still unable to resolve the problem, please contact with us by E-mail, and describe in detail of the failure phenomenon you encountered.

E-mail: joney.s@foxmail.com

Fall into the water, fall on the ground or abnormal power supply (Higher than 5.5v, or less than 4.5v) caused by the failure, will not be within the scope of our warranty.

8.2 Reset Button Instruction

At the top of the I2C_LCD there is a reset button, if the state of the I2C_LCD is not normal, can be recovered by pressing this button.

Note: After the reset button is pressed, the configurations saved in RAM are lost, and the configurations saved in EEPROM are not affected.

8.3 Restore Factory Settings

If you can not solve the failure, please follow the following steps to restore the I2C_LCD to factory settings.

1、 Make sure the I2C_LCD is under the typical power supply, then use conductive objects to short circuit the REC contacts on the back of the I2C_LCD.

2、 Keeping the REC contacts in short circuit contact state, and then press the reset button (RST);

3、 When the screen appears "Resetting...OK!", indicates that the recovery is successful, and all the configurations has resumed to the default value.

Note:

1. If you fail to recover, please try again several times, until the screen appears "Resetting... OK!".

2. After recover to factory setting successfully, all the configurations will recover to default values, please confirm the device address (I2C_LCD_ADDRESS) in your program was set to the default value 0x51, otherwise will cause I2C_LCD no response.