乐升半导体
LEVETOP

LT768x

High Performance TFT-LCD Graphics Controller

Application Notes

V1.2

www.levetop.tw
Levetop Semiconductor Co., Ltd.

Directory

1. Preface

This application note mainly describes the hardware interface, software libraries of LT768x, and the realization of the internal function. In the meantime, by utilizing the demo program, function library, and schematic diagram, customers can quickly set up the design environment of LT768x and develop their application with the TFT panel product. This manual can help users to get a quick start and shorten their development time on exploring LT768x.

1.1 Description

In the last few chapters of this manual, STM32+LT768x demo board and STC8051+LT768x demo board are introduced. In addition, a reference design and design notice for TFT module makers is presented. It is also explained how users can program BIN file to SPI Flash.

The software library and circuit schematics used in the manual are free and open, including the register control software of LT768x. Also, demo programs for common STM 32-bit MCU and STC8051 are available. Meanwhile, a dedicated program – "LT_IMAGE_TOOL.EXE" is provided to transfer pictures/Chinese font to BIN files and integrate BIN files. The detail operation flow of this program is described in this application manual. The schematic diagrams about the connection of LT768x and MCU (STM32F103VE and STC8A8K64S which represent 32-bit/8-bit MCU respectively) are also provided. This application note also provides the reference schematic diagram for designing LT768x on TFT modules. These software and hardware resources are fully open to users. Users can either download the relevant data/files from our company website (www.levetop.tw), or contact with our sales person to acquire those data/files.

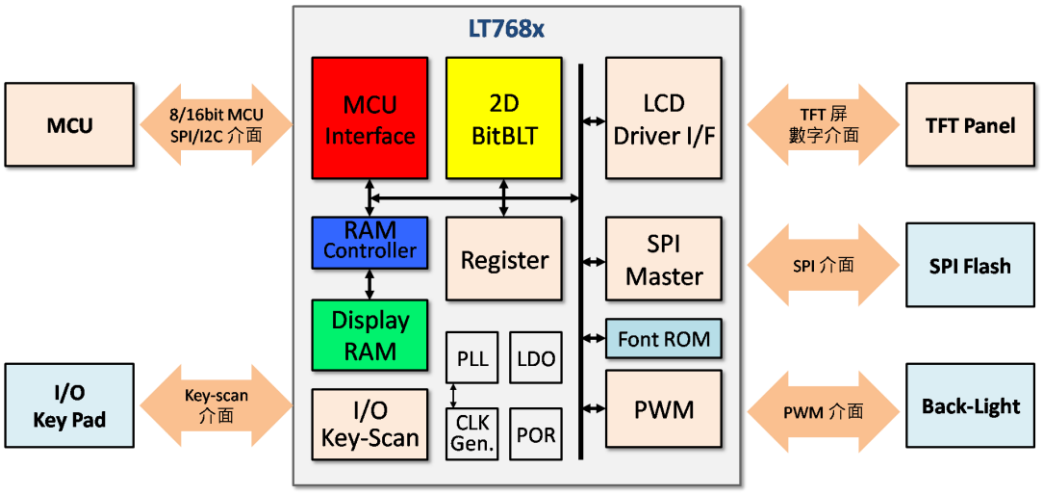

Figure 1-1: Internal Block Diagram of LT768x

1.2 System Architecture

To display an image on TFT panel, the image data must first be converted into electrical signals, and then the TFT driver transmits and continuously updates(scan) these signals to the LCD panel. As a result of the continuous transmission of scanned data, and the visual retention effect of human eyes, the image displayed on the TFT panel will be stable and complete as seen by human eyes. However, since the TFT driver does not have data store functions, it must continuously get the image data from the system side (such as MCU). The main function of the TFT controller is to assist the system to provide the image data to the TFT driver, and keep uploading the image data to the TFT panel continuously.

LT768x is a high-performance TFT-LCD graphics accelerated display controller. In addition to assist the MCU transfering display data to TFT driver as mentioned above, LT768x also provides 2D graphics acceleration, PIP (picture-in-picture), geometric graphics and other functions. In order to reduce the time taken by MCU to transmit image data, LT768x provides a SPI Master Interface to retrieve image data from SPI Flash through DMA transfer mode, and then save the data to the embedded display RAM of LT768x. SPI Flash is used to save image data such as pictures, fonts etc. LT768x will then transmit the specified display memory data of the selected display window through the RGB interface to the driver inside the external TFT panel. Therefore, LT768x not only enhance the display performance, but also greatly alleviate the processing burden on MCU.

Even if an 8bits MCU is used as the host, it can work well with a TFT display if LT768x is applied properly. The following diagram is the basic application architecture for LT768x:

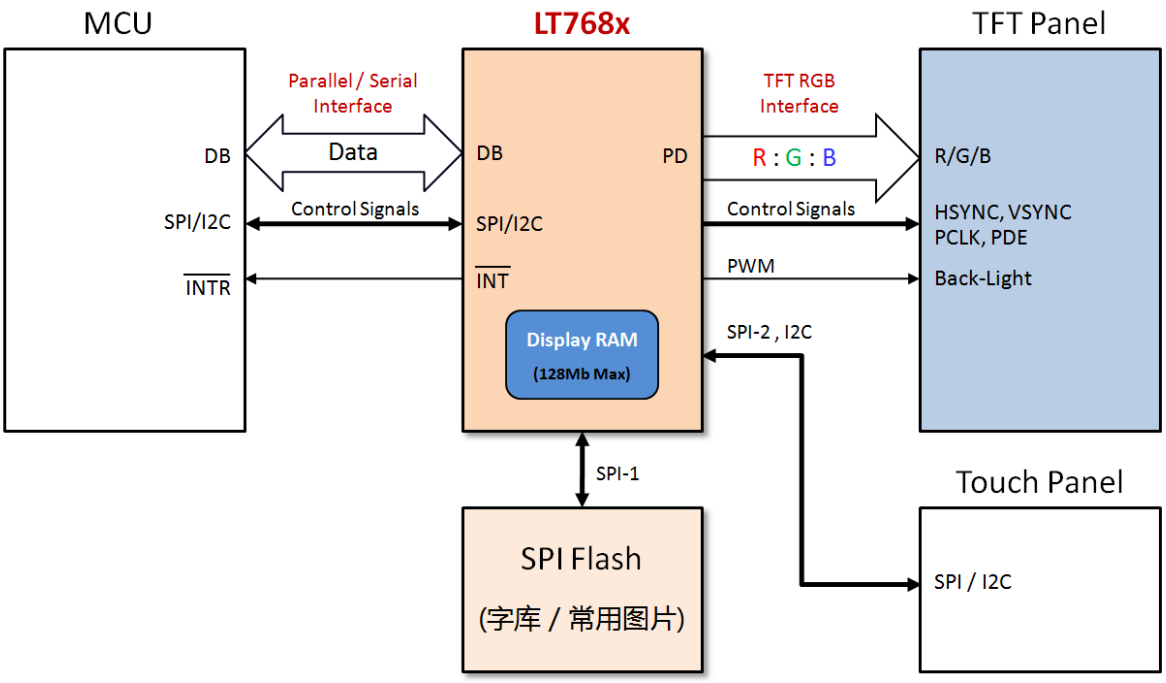

Figure 1-2: System Architecture of LT768x

LT768x provides 2 sets of SPI master interfaces and 1 set of I2C interfaces. If the system wants to provide a touch screen function, the interface of touch controller can either connect to I2C or SPI interface of LT768x, and then the MCU can read or control the touch controller through LT768x. This will simplifies the interface of touch controller connection.

2. Model Options

Table 2-1: LT768x Model Options

Support Features		LT7680A	LT7680B	LT7681	LT7683	LT7686
Items	Function					
Package	LQFP / QFN	QFN-68	QFN-68	LQFP-128	LQFP-128	LQFP-128
LCD Spec.	Resolution/Max.	800*600	480*320	640*480	1024*768	1280*1024
	Colors	262K	262K	16.7M	16.7M	16.7M
	TFT Interface	RGB (18bits/Max)	RGB (18bits/Max)	RGB (24bits/Max)	RGB (24bits/Max)	RGB (24bits/Max)
Display RAM	Embed. RAM Size	64Mbit	64Mbit	128Mbit	128Mbit	128Mbit
	Layers	7 Layers/Min	12 Layers /Min	18 Layers/Min	11 Layers /Min	4 Layers /Min
MCU Interface	8080 8bit			V	V	V
	6800 8bit			V	V	V
	8080 16bit			V	V	V
	6800 16bit			V	V	V
	3 wire SPI	V	V	V	V	V
	4 wire SPI	V	V	V	V	V
	I2C			V	V	V
Others Interface	SPI Master (DMA Flash)	V (2)	V (2)	V (2)	V (2)	V (2)
	I2C Master			V	V	V
	PWM O/P	2	2	2	2	2
	GPIO O/P	7/max	7/max	28/max	28/max	28/max
	Smart Key-scan			5*5	5*5	5*5
Graphics Features	2D BTE Engine	V	V	V	V	V
	Geometric Drawing Engine	V	V	V	V	V
	Polygon Drawing	V	V	V	V	V
	Picture in Picture (PIP)	V	V	V	V	V
	Virtual Display	V	V	V	V	V
	Vertical Scrolling	V	V	V	V	V
	Horizontal Scrolling	V	V	V	V	V
	Mirror and Rotation	V	V	V	V	V
	Alpha-Blending	V	V	V	V	V
	Graphic Cursor	V	V	V	V	V
	Power-on Display	V	V	V	V	V
	Color-Bar Test	V	V	V	V	V
Text Features	Embed. ISO/IEC 8859	ISO8259	ISO8259	ISO8259	ISO8259	ISO8259
	Ext. Font (ext. SPI Flash)	V	V	V	V	V
	Text Enlargement	4*4	4*4	4*4	4*4	4*4
	Text Rotate	V	V	V	V	V
	Text Cursor	V	V	V	V	V
	User-defined Character	V	V	V	V	V
Power	Sleep Mode ($S_{tandby}/S_{uspend}/S_{leep}$)	V	V	V	V	V
	Power Source	3.3V	3.3V	3.3V	3.3V	3.3V

Figure 2-1: LT768x Series

The LT768x of the same package are compatible. For example, LT7681, LT7683 and LT7686 are all 128Pin LQFP packages, and their pin-assignments are compatible. The LT7680A and LT7680B are 68Pin QFN package, their pin-assignment are also compatible. The resolution is backward compatible. For example, the resolution of the LT7686 is 1280*1024, it can also be used on the lower resolution of the TFT screen.

LT7680 is a 68Pin QFN (8mm*8mm) package with a smaller dimension. It can be used in the system PCB board or LCM PCB. It can also be welded on the FPC to form a standard TFT module with SPI interface, as shown in Figure 2-2 below. The designers can also integrate a standard TFT module and a LT768x control board to form a complete module so that most 8/16/32bits MCU can connect to this complete module directly, as shown in Figure 2-3.

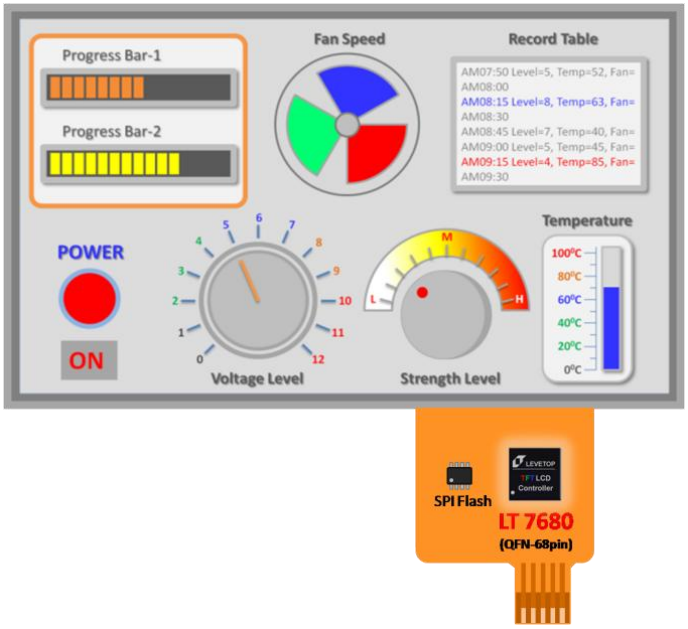

Figure 2-2: Standard SPI Interface TFT Module with LT7680

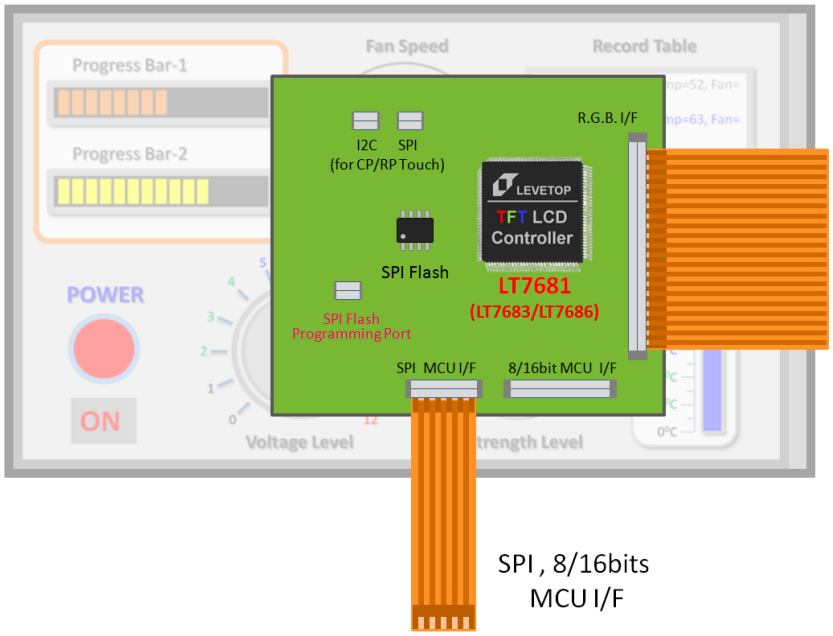

Figure 2-3: Standard TFT Module with LT7681/7683/7686

LT768x is controlled by the MCU so it can also be placed in the system board, and then connect with the standard RGB type TFT Panel. Show as below Figure 2-4.

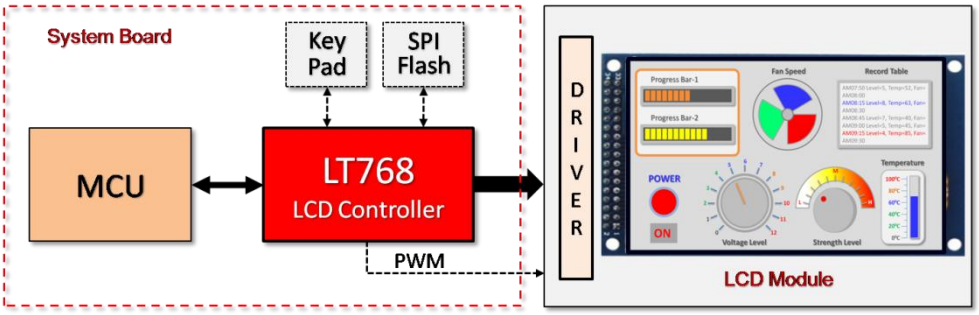

Figure 2-4: Design-in LT768x on System Board

3. Reset

3.1 Power-on Reset

LT768x has an embedded Power-On-Reset circuit. POR can issue an active low signal to synchronize the whole system through RST# pin. When system power (3.3V) on, internal reset will be active for at least 256 OSC clocks until the internal power is stable.

3.2 External Reset

External reset signal RST# allows LT768x to synchronize with external systems. The external reset signal must be stabilized for at least 256 crystal (OSC) clocks to be approved as shown in Figure 3-1. The MCU should check the BIT1(working mode status indication bit) of the state register STSR before setting up LT768x to ensure that LT768x is currently in "Normal Running State". External reset can be done through power-on reset or hardware reset issued by MCU, as shown in Figure 3-2, Figure 3-3.

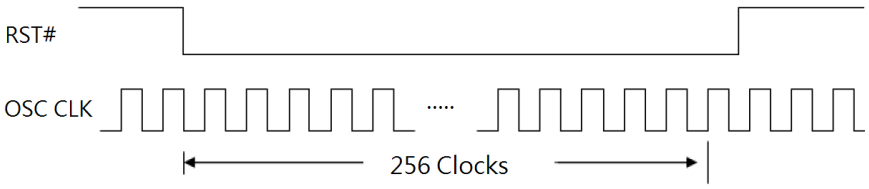

Figure 3-1: External Reset Signal - RST#

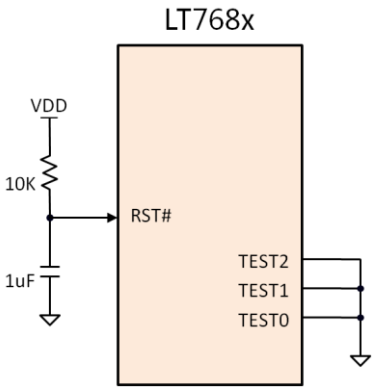

Figure 3-2: External Reset (1)

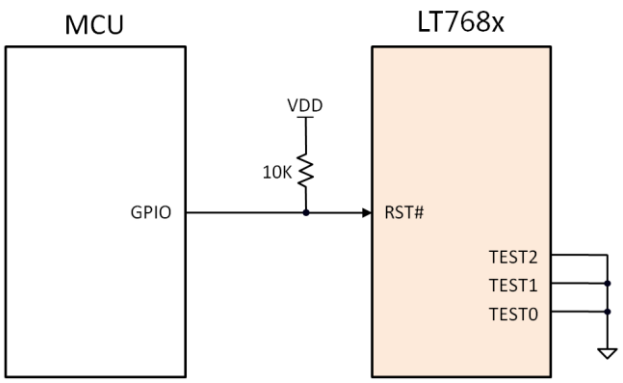

Figure 3-3: External Reset (2)

3.3 Software Reset

If the Host writes 1 to registers REG[00h] bit0, LT768x will be reset by software. The software reset will only reset the internal state machine of LT768x, and the other registers values will not be affected or cleared. After the software reset is complete, the REG[00h] bit0 will automatically be cleared to 0.

To perform a software reset, simply call the following function:

```
void LT768_SW_Reset(void);
```

3.4 Test Signals

TEST[2:0] are the test signals of LT768x, which is provided to LT768x for testing purposes. These pins should be connected to ground (GND) in normal use. As shown in above Figure 3-2 and 3-3.

If the system is off, and users want to update image datat to SPI Flash that connects to LT768x, then TEST[2] should be pulled low, and TEST[1] should be pulled high so that LT768x can enter TEST mode and disconnect external SPI Flash. This action will allow the data to be programmed to SPI Flash without being affected by LT768x. Please refer to Sections 18.4, 19.4, 21.3, and 22.3 for more details.

4. Clock Setting

4.1 Clock and PLL

LT768x needs an external clock or 10MHz crystal oscillator as the clock source for the three internal PLL. (Refer to Figure 4-1 to 4-3) The three PLL will generate thee different clock as following:

- CCLK: Used for Core Clock. Maximum is 100MHz.
- MCLK: Used for Display Memory Clock. Maximum is 133MHz.
- PCLK: Used for TFT Pixel Clock. Maximum is 80MHz.

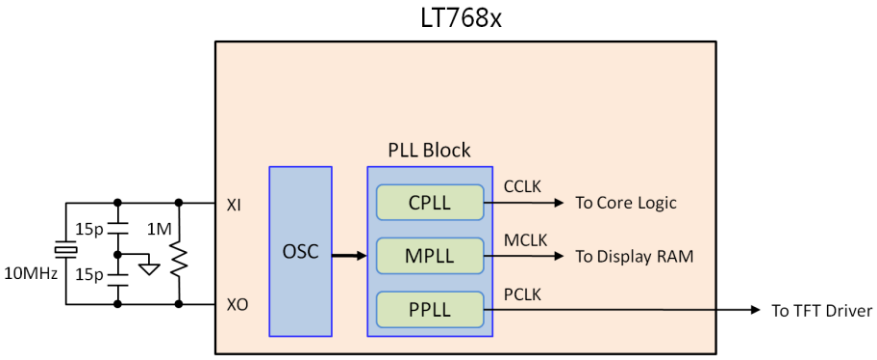

Figure 4-1: Clock Diagram of LT768x(1)

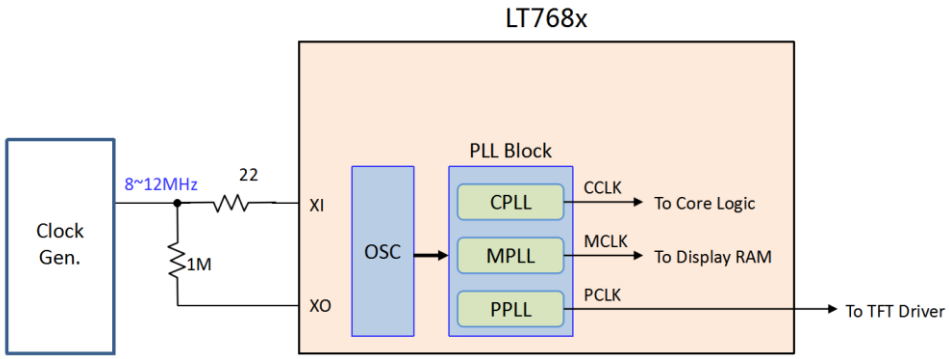

Figure 4-2: Clock Diagram of LT768x(2)

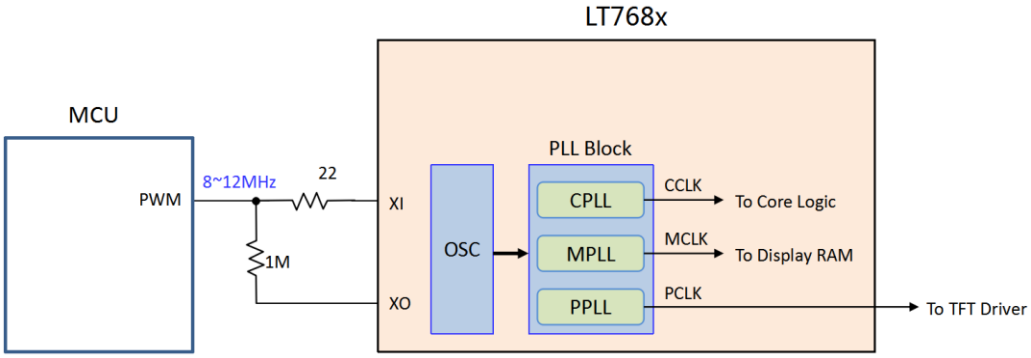

Figure 4-3: Clock Diagram of LT768x(3)

The three PLL are operation independent. The PLL output frequency is calculated from the following formula:

$$F_{OUT} = XI * (N / R) \div OD$$

In above formula, XI is the external 10MHz oscillator input. “R” is Input Divider Ratio, its value is between 2 ~ 31. “OD” is Output Divider Ratio that has to be set to 1, 2 or 4. “N” is the Feedback Divider Ratio of Loop represented by 9bits and its value is between 2 ~ 511.

Table 4-1: PLL Register Setting (1)

R[4:0]	Input Divider Ratio (R)	N[8:0]	Feedback Divider Ratio (N)
00010	2	000000010	2
00011	3	000000011	3
00101	4	000000101	4
⋮	⋮	⋮	⋮
11101	29	111111101	509
11110	30	111111110	510
11111	31	111111111	511

Table 4-2: PLL Register Setting (2)

OD[1:0]	Input Divider Ratio (OD)
00	1
01	2
10	3
11	4

For example, XI is 10MHz, R[4:0] is 01010 (i.e. 10), N[8:0] is 100000000 (i.e. 256), OD[1:0] is 11 (i.e. 4), then:

$$F_{OUT} = 10MHz * (256 / 10) \div 4 = 64MHz$$

The three clock frequency rule of LT768x is as following:

1. CCLK*2 > = MCLK >= CCLK
2. CCLK >= PCLK *1.5

Usually TFT panel manufacturers will provide the best Pixel Clock (PCLK) for the display, based on its TFT characteristics. Therefore, the registers should be setup according to the the requirement of the PCLK. After that, users can setup the frequencies of CCLK and MCLK by the above rules.

4.2 Clock Initialize

Users can modify the relevant parameters of **Lt768_lib.h** header file to setup the three clocks.

```
//Resolution
#define LCD_XSIZE_TFT      1024
#define LCD_YSIZE_TFT      600
//Parameters
#define LCD_VBPD           20
#define LCD_VFPD           12
#define LCD_VSPW           3
#define LCD_HBPD           140
#define LCD_HFPD           160
#define LCD_HSPW           20
```

The Pixel Clock (PCLK) can be obtained according to the above 6 parameters, the specific formula is as follows:

PCLK = (LCD_HBPD + LCD_HFPD + LCD_HSPW + LCD_XSIZE_TFT) * (LCD_VBPD + LCD_VFPD + LCD_VSPW+LCD_YSIZE_TFT) * 60;

When the above 6 parameters are set, call the function directly:

```
void LT768_PLL_Initial(void)
```

The three clocks will then be set automatically.

5. MCU Interface Setting

5.1 MCU Interface

LT768x provides 8bits and 16bits parallel modes, serial SPI mode, and I2C mode for the communication with Host. These interface modes are setup by PSM[2:0] pins:

Table 5-1: MCU Interface Mode

PSM[2:0]	MCU Interface
0 0 X	8bits or 16bits 8080 Parallel Interface Mode
0 1 X	8bits or 16bits 6800 Parallel Interface Mode
1 0 0	3-Wire SPI Mode
1 0 1	4-Wire SPI Mode
1 1 X	I2C Mode

When using the Parallel Host mode, 8bits or 16bits data transfer is determined by the bit0 of Register REG[01h]. When bit0=0, then 8bits mode is selected. If bit0=1 then 16bits mode is selected.

LT768x series support different Host interfaces. For example, LT7680 is a 68pin QFN chip, which only supports 3-wires and 4-wires serial SPI mode. The following table is the list of host interfaces supported by LT768x series:

Table 5-2: MCU Interface Supporting List of LT768x Series

No.	Host Interface Mode	LT7681 LT7683 LT7686	LT7680A-R LT7680B-R
1	8bits 8080 Parallel Interface Mode	v	
2	16bits 8080 Parallel Interface Mode	v	
3	8bits 6800 Parallel Interface Mode	v	
4	16bits 6800 Parallel Interface Mode	v	
5	3-Wire SPI Mode	v	v
6	4-Wire SPI Mode	v	v
7	I2C Mode	v	

In LT7680, the PSM[0] pin is already connected to GND within the LT7680, while PSM[2] must be pull-up to VDD. When PSM[1] = 0, then the 3-wires Serial SPI mode is selected. When PSM[1] = 1, then the 4-wires serialmode is selected.

The followings are the reference circuits for different MCU interfaces:

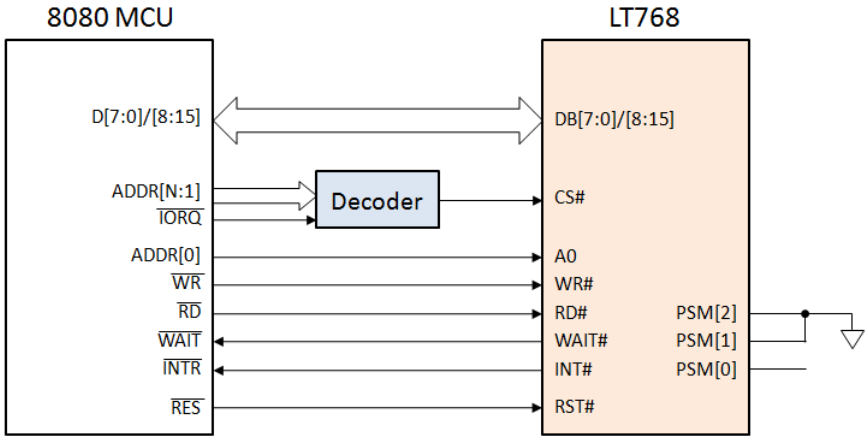

Figure 5-1: 8080 Parallel Mode Interface

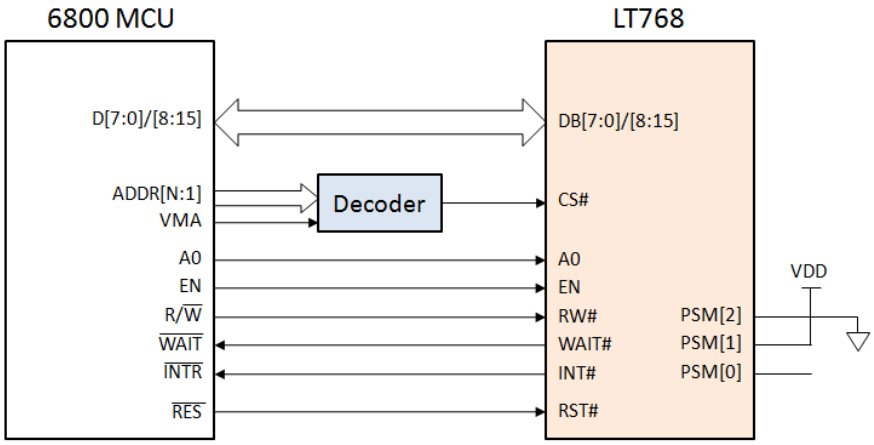

Figure 5-2: 6800 Parallel Mode Interface

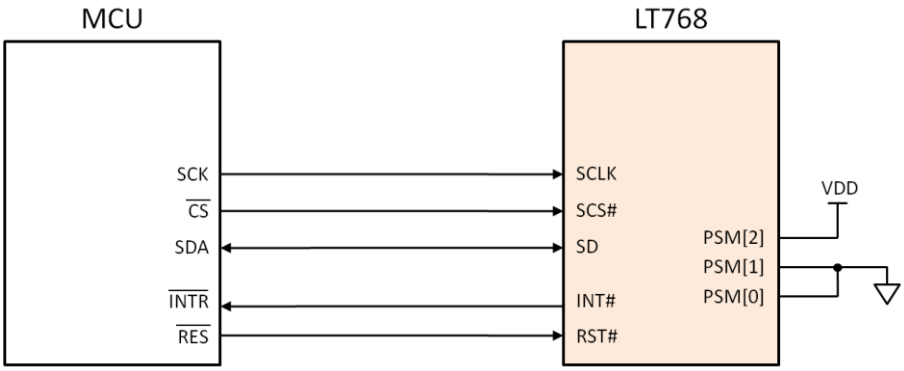

Figure 5-3: 3-Wire SPI Interface

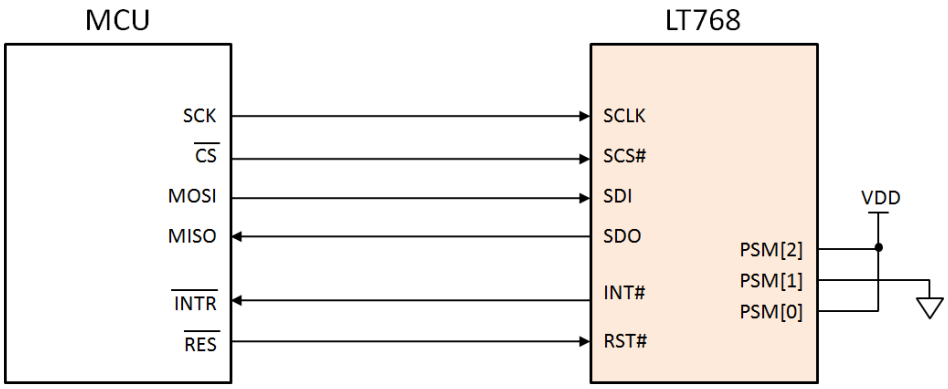

Figure 5-4: 4-Wire SPI Interface

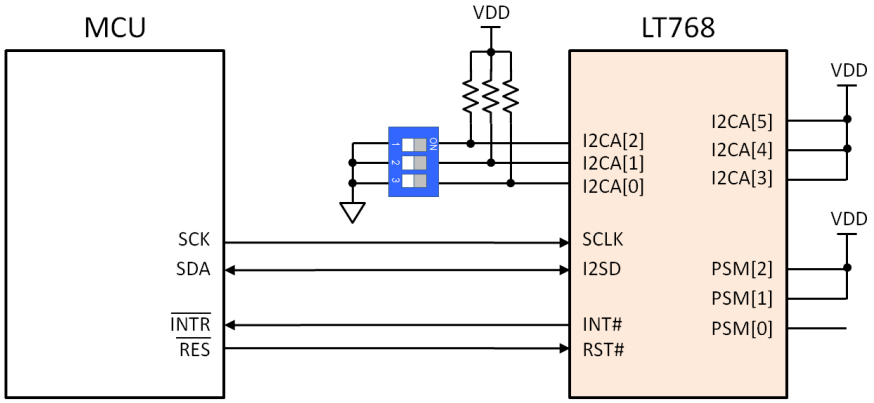

Figure 5-5: I2C Interface

Because the different MCU interfaces cannot be used at the same time, LT768x provides a shared pin mode. When Serial Mode is used, other parallel pins can be set to GPIO mode. Please refer to the following table:

Table 5-3: The Pin Definition of Host Interface

Pin Name	8080 I/F		6800 I/F		SPI 3-Wires	SPI 4-Wires	I2C
	8-bits	16-bits	8-bits	16-bits			
DB[15:8]	--	DB[15:0]	--	DB[15:0]	GPIOA[0:7]	GPIOA[0:7]	GPIOA[0:7]
DB[7]	DB[7:0]		DB[7:0]		SCLK	SCLK	SCLK
DB[6]					GND	SDI	I2C_SDA
DB[5]					SD	SDO	I2CA[5]
DB[4]					SCS#	SCS#	I2CA[4]
DB[3:0]					GND	GND	I2CA[3:0]
CS#	CS#		CS#		GPIB[0]	GPIB[0]	GPIB[0]
RD#	RD#		EN		GPIB[1]	GPIB[1]	GPIB[1]
WR#	WR#		RW#		GPIB[2]	GPIB[2]	GPIB[2]
A0	A0		A0		GPIB[3]	GPIB[3]	GPIB[3]
INT#	INT#		INT#		INT#	INT#	INT#
WAIT#	WAIT#		WAIT#		--	--	--

5.2 8-bit 8080 Interface

No matter which interface is used, these interfaces primarily implement the following five functions. Other functions are implemented based on the five functions.

1、void LT768_CmdWrite(u8 cmd); // Write Command to LT768

2、void LT768_DataWrite(u8 data); // Write Data to LT768

3、void LT768_DataWrite_Pixel(u16 data); // Write Pixel Data to LT768

4、u8 LT768_StatusRead(void); // Read Status Register Data

5、u16 LT768_DataRead(void); // Read Register Data

When using the 8-bit 8080 interface, be aware that REG[01h] bit[0] must be set to 0 (Host bus is 8bit) when initializing, otherwise the pixel data will be garbled.

- **Use STM32's FMSC to simulate 8-bit 8080 interfaces**

```
void FMSC_8_CmdWrite(u8 cmd)
{
*(vu8*) (LCD_BASE0)= (cmd);
}

void FMSC_8_DataWrite(u8 data)
{
*(vu8*) (LCD_BASE1)= (data);
}

void FMSC_8_DataWrite_Pixel(u16 data)
{
*(vu8*) (LCD_BASE1)= (data);
*(vu8*) (LCD_BASE1)= (data>>8);
}

u8 FMSC_8_StatusRead(void)
{
u8 temp = 0;
temp = *(vu8*)(LCD_BASE0);
return temp;
}
```

```
u16 FMSC_8_DataRead(void)
{
u16 temp = 0;
temp =    *(vu8*)(LCD_BASE1);
return temp;
}
```

5.3 16-bit 8080 Interface

- **Use STM32's FMSC to simulate 16-bit 8080 interfaces**

```
void FMSC_16_CmdWrite(u8 cmd)
{
*(vu16*) (LCD_BASE0)= (cmd);
}

void FMSC_16_DataWrite(u8 data)
{
*(vu16*) (LCD_BASE1)= (data);
}

void FMSC_16_DataWrite_Pixel(u16 data)
{
*(vu16*) (LCD_BASE1)= (data);
}

u8 FMSC_16_StatusRead(void)
{
u8 temp = 0;
temp = *(vu16*)(LCD_BASE0);
return temp;
}

u16 FMSC_16_DataRead(void)
{
u16 temp = 0;
temp =    *(vu16*)(LCD_BASE1);
return temp;
}
```

5.4 SPI Interface

```
void SPI_CmdWrite(u8 cmd)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x00);          // MCU write Instruction Register Address
    SPI2_ReadWriteByte(cmd);
    SS_SET;
}

void SPI_DataWrite(u8 data)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x80);          // MCU write data to Register or Display RAM
    SPI2_ReadWriteByte(data);
    SS_SET;
}

void SPI_DataWrite_Pixel(u16 data)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x80);
    SPI2_ReadWriteByte(data);
    SS_SET;

    SS_RESET;
    SPI2_ReadWriteByte(0x80);
    SPI2_ReadWriteByte(data>>8);
    SS_SET;
}

u8 SPI_StatusRead(void)
{
    u8 temp = 0;
    SS_RESET;
    SPI2_ReadWriteByte(0x40);          // MCU read Data from Status Register
    temp = SPI2_ReadWriteByte(0xff);
    SS_SET;
    return temp;
}
```

```
u16 SPI_DataRead(void)
{
    u16 temp = 0;
    SS_RESET;
    SPI2_ReadWriteByte(0xc0);              // MCU read Data from Instruction Register
    temp = SPI2_ReadWriteByte(0xff);
    SS_SET;
    return temp;
}
```

Note: The specific contents can refer to the Serial Host Interface of Section 2.2 of LT768x data sheet.

5.5 I2C Interface

```
u8 IIC_CmdWrite(u8 cmd)
{
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x00);        // MCU write Instruction Register Address
    if(IIC_Wait_Ack())                  // Waiting for Ack message
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(cmd);
    if(IIC_Wait_Ack())                  // Waiting for Ack message
    {
        IIC_Stop();
        return 1;
    }
    IIC_Stop();
    return 0;
}

u8 IIC_DataWrite(u8 data)
{
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x02);        // MCU write data to Register or Display RAM
    if(IIC_Wait_Ack())                  // Waiting for Ack message
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(data);
    if(IIC_Wait_Ack())                  // Waiting for Ack message
    {
        IIC_Stop();
        return 1;
    }
    IIC_Stop();
    return 0;
}
```

```
void IIC_DataWrite_Pixel(u16 data)
{
    IIC_DataWrite(data);
    IIC_DataWrite(data>>8);
}


u8 IIC_StatusRead(void)
{
    u8 res;
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x01);          // MCU read Data from Status Register
    IIC_Wait_Ack();
    res=IIC_Read_Byte(0);
    IIC_NAck();
    IIC_Stop();
    return res;
}


u8 IIC_DataRead(void)
{
    u8 res;
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x03);          // MCU read Data from Instruction Register
    IIC_Wait_Ack();
    res=IIC_Read_Byte(0);
    IIC_NAck();
    IIC_Stop();
    return res;
}
```

Notes:

1. The address represented by LT_ADDR is set by LT768x pin I2CA[5:0].
2. Please refer to the Serial Host Interface of Section 2.2 of LT768x data sheet for detail information.

6. Display Memory Setting

LT768x has two different Display Memory Capacities as listed in the below table:

Table 6-1: LT768x Display Memory Capacity

Model	Display Memory Capacity	Resolution (Max.)
LT7681	128Mb	640*480
LT7683	128Mb	1024*768
LT7686	128Mb	1280*1024
LT7680A	64Mb	800*600
LT7680B	64Mb	480*320

The Display Memory Capacity is set as follows:

```
#define SDRAM_128    1
//#define SDRAM_64    1
void LT768_SDRAM_initail(u8 MCLK);
```

For example: If you want to use 128Mbit Display Memory, and set the MCLK to 100MHz:

```
#define SDRAM_128    1
//#define SDRAM_64    1
LT768_SDRAM_initail(100);
```

Note: The value of the MCLK needs to be the same as the clock PLL setting. The initialization function is different for the two display memory capacities. The following examples are 128Mbit (LT7681/7683/7686) and 64Mbit (LT7680A/LT7680B) Display Memory initialization functions:

```
void LT768_SDRAM_initail(u8 mclk)
{
 unsigned short sdram_itv;                    // 128Mbit (LT7681/7683/7686)
 LCD_RegisterWrite(0xe0, 0x29);
 LCD_RegisterWrite(0xe1, 0x03);
 sdram_itv = (64000000 / 8192) / (1000/ mclk);
 sdram_itv-=2;

 LCD_RegisterWrite(0xe2, sdram_itv);
 LCD_RegisterWrite(0xe3, sdram_itv >>8);
 LCD_RegisterWrite(0xe4, 0x01);
 Check_SDRAM_Ready();
 Delay_ms(1);
}
```


The following function is a 64Mbit Display memory initialization for LT7680A and LT7680B.

```
void LT768_SDRAM_initail(u8 mclk)
{
 unsigned short sdram_itv;                          // 64Mbit (LT7680A/7680B)
 LCD_RegisterWrite(0xe0, 0x21);
 LCD_RegisterWrite(0xe1, 0x03);
 sdram_itv = (64000000 / 2048) / (1000/ mclk);
 sdram_itv-=2;

 LCD_RegisterWrite(0xe2, sdram_itv);
 LCD_RegisterWrite(0xe3, sdram_itv >>8);
 LCD_RegisterWrite(0xe4, 0x01);
 Check_SDRAM_Ready();
 Delay_ms(1);
}
```

7. LCD Panel Interface

7.1 RGB Type TFT Interface

LT768x Supports 16, 18, 24bits RGB interface of TFT Panel, no matter the color depth is 24bpp (RGB 8:8:8), 16bpp (RGB 5:6:5) or 8bpp (RGB 3:3:2), the display data can be sent to the TFT Driver on the TFT panel through these RGB interfaces. The LT768x LCD Display data that correspond to the RGB data are shown in the below Table 7-1. The Host can setup REG[01h] Bit[4:3] to select 16bits, 18bits or 24bits resolution.

The RGB data supported by different models of LT768x are also different. For example, when using LT7680A, even if REG[01h] bit[4:3] is set to 00b (24bits), the shown color depth will still be 18bits only. Please refer to Table 7-2 for RGB data supported by different models of LT768x.

Table 7-1: RGB Interface VS. RGB Display Data

LCD Data Bus	TFT-LCD Interface		
	REG[01h] bit[4:3] = 10b (16bits)	REG[01h] bit[4:3] = 01b (18bits)	REG[01h] bit[4:3] = 00b (24bits)
PD[0]			B0
PD[1]			B1
PD[2]		B0	B2
PD[3]	B0	B1	B3
PD[4]	B1	B2	B4
PD[5]	B2	B3	B5
PD[6]	B3	B4	B6
PD[7]	B4	B5	B7
PD[8]			G0
PD[9]			G1
PD[10]	G0	G0	G2
PD[11]	G1	G1	G3
PD[12]	G2	G2	G4
PD[13]	G3	G3	G5
PD[14]	G4	G4	G6
PD[15]	G5	G5	G7
PD[16]			R0
PD[17]			R1
PD[18]		R0	R2
PD[19]	R0	R1	R3
PD[20]	R1	R2	R4
PD[21]	R2	R3	R5
PD[22]	R3	R4	R6
PD[23]	R4	R5	R7

Table 7-2: RGB Data Signals of LT768x

Model	LCD Data Bus	RGB Signal Number	Colors	Description
LT7681	PD[23~0]	R:G:B = 8:8:8	16.7M Colors	Support 16/18/24bits
LT7683	PD[23~0]	R:G:B = 8:8:8	16.7M Colors	Support 16/18/24bits
LT7686	PD[23~0]	R:G:B = 8:8:8	16.7M Colors	Support 16/18/24bits
LT7680A	PD[23~18], PD[15~10], PD[7~23]	R:G:B = 6:6:6	262K Colors	Support 16/18bits
LT7680B	PD[23~18], PD[15~10], PD[7~23]	R:G:B = 6:6:6	262K Colors	Support 16/18bits

In addition to the RGB data lines mentioned above, LT768x also provides PCLK (Panel Scan Clock), VSYNC Pulse, HSYNC Pulse and Data Enable signals. Figure 7-1 is the interface diagram of LT768x and RGB type TFT panel, and Figure 7-2 is the timing diagram of these LCD signals.

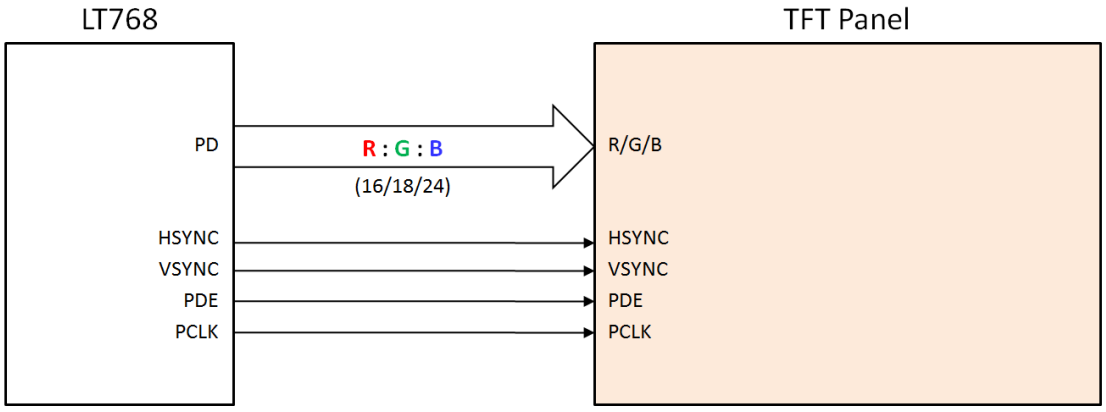

Figure 7-1: Interface Diagram of LT768x and RGB type TFT Panel

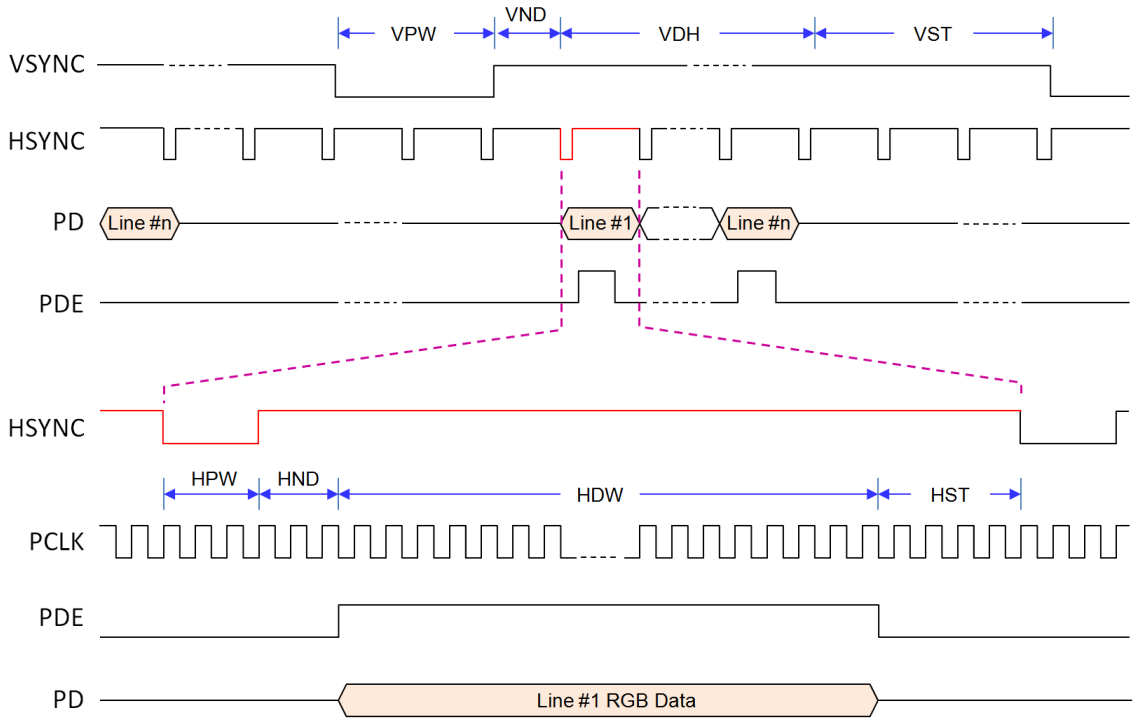

Figure 7-2: TFT-LCD Interface Timing Diagram

7.2 LCD Setting

For different LCD panel, you can set different parameters in the following function.

```
void Set_LCD_Panel(void);
```

The specific settings can be modified in this function, such as:

LCD Scan Type:

```
VSCAN_T_to_B();          //REG[12h]: from Top to Bottom
//VSCAN_B_to_T();        //from Bottom to Top
```

RBG Output Sequence:

```
PDATA_Set_RGB();         //REG[12h]: Select RGB output
//PDATA_Set_RBG();       // Select RBG output
//PDATA_Set_GRB();       // Select GRB output
//PDATA_Set_GBR();       // Select GBR output
//PDATA_Set_BRG();       // Select BRG output
//PDATA_Set_BGR();       // Select BGR output
```

HSYNC Active Level:

```
HSYNC_Low_Active();      //REG[13h]: Low
//HSYNC_High_Active();   // High
```

VSYNC Active Level:

```
VSYNC_Low_Active();      //REG[13h]: Low
//VSYNC_High_Active();   // High
```

DE Active Level:

```
DE_High_Active();        //REG[13h]: High
//DE_Low_Active();       // Low
```

8. Display Functions

8.1 Display Windows

The display window includes Main Window, Canvas Window and Active Window. All of the three windows is corresponded to the location of the LT768x's internal Display RAM (SDRAM). Their definitions are as follows:

- **Main Window:** Used to define an area of Display Memory as a display area for the TFT Panel.
- **Canvas Window:** Used to define an area of the Display Memory as a region to be written to when transferring data, such as DMA.
- **Active Window:** In the main window area, the LCD work area specified for drawing geometry or text display action.

8.1.1 Main Window Setting

- **Color Depth**

```
Select_Main_Window_8bpp();      // 256 Color
Select_Main_Window_16bpp();     // 65K Color
Select_Main_Window_24bpp();     // 16.7M Color
```

- **Start Display Address Setting**

```
Main_Image_Start_Address(unsigned long layer);
```

For Example: Setup Display Memory Address 0x8000 is the Start Display Address of Main Window:

```
Main_Image_Start_Address(0x800);
```

- **Width Setting**

```
Main_Image_Width(unsigned short LCD_XSIZE_TFT);
```

For Example: Setup the Width of Main Window is 1024:

```
Main_Image_Width(1024);
```

Note: Normally, the Width of Main Window is the same as the LCD Panel Size.

- **Start Coordinate Address Setting**

```
void Main_Window_Start_XY(unsigned short WX, unsigned short HY)
```

8.1.2 Canvas Window Setting

The Canvas Window includes two settings:

- **Start Address Setting**

  void Canvas_Image_Start_address(unsigned long Addr) ;

- **Width Setting**

  void Canvas_image_width(unsigned short WX);

For Example: Setup a Canvas Window which Start Address is 0x8000, and window Width is 480:

```
Canvas_Image_Start_address(0x8000);
Canvas_image_width(480);
```

8.1.3 Active Window Setting

The Active Window includes two settings:

- **Start Coordinate Address Setting**

  void Active_Window_XY(unsigned short WX, unsigned short HY);

- **Active Window Size Setting**

  void Active_Window_WH(unsigned short WX, unsigned short HY)

For Example: Setup an Active Window that Start Coordinate Address is (0, 0), and window size is (1024, 600):

```
Active_Window_XY(0, 0);
Active_Window_WH(1024, 600);
```

Note: The image data can be displayed normally on the LCD Panel only after the three Windows are properly set.

8.2 MCU Write Data to Display RAM

In parallel mode, the MCU can drive LT768x through 8-bit or 16-bit interface, and LT768x can display a variety of color depth pictures, therefore, there are many ways for the MCU to write data to the Display RAM. The following functions are for MCU Write Data to Display RAM, based on different data bus and different color depth.

void MPU8_8bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);

//MCU is 8-bit, write 8-bit color depth data to Display Memory

void MPU8_16bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);

//MCU is 8-bit, write 16-bit color depth data to Display Memory

void MPU8_24bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);

//MCU is 8-bit, write 24-bit color depth data to Display Memory

void MPU16_16bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);

//MCU is 16-bit, write 16-bit color depth data to Display Memory

void MPU16_24bpp_Mode1_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);

//MCU is 16-bit, write 24-bit color depth data to Display Memory (Mode 1)

void MPU16_24bpp_Mode2_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);

//MCU is 16-bit, write 24-bit color depth data to Display Memory (Mode 2)

8.3 Show Picture on Main Window

The following two examples explain how a picture can be displayed in the main window.
Given below assumptions:

1.The Resolution of LCD Panel is 1024*600.

2.The Picture Size is also 1024*600, and color depth is 16-bit.

3.The Picture is stored in picture_data[].

Example 1: Write a Picture (Resolution: 1024*600, Color Depth: 16-bit) to Display Memory (Start Address: 0), and show it on the TFT Panel.

1. Setup Color Depth of Main Window

```
Select_Main_Window_16bpp();          // 16bit Color Depth
```

2. Setup the Start Address and Width of Main Window

```
Main_Image_Start_Address(0);         // Setup Start Address is 0
Main_Image_Width(1024);              // Width of Main Window
```

3. Setup Start Coordinate of Main Window

```
Main_Window_Start_XY(0, 0);          // Start Coordinate is (0, 0)
```

4. Setup Start Address and Width of Canvas Window

```
Canvas_Image_Start_address(0);       // Start Address is 0 (Write Data)
Canvas_image_width(1024);            // Width of Canvas
```

5. Setup the Start Coordinate and Display Area of Active Window

```
Active_Window_XY(0, 0);              // Start Display Coordinate is (0, 0)
Active_Window_WH(1024, 600);         // LCD Display Area is 1024*600
```

6. MCU Write Picture Data to Display Memory

```
MPU16_16bpp_Memory_Write(0, 0, 1024, 600, picture_data);
```

Example 2: Get a picture(Resolution: 1024*600, Color Depth: 16-bit) from external SPI Flash (Address: 0) by DMA mode, then show it on the TFT Panel. Assume the picture data are already in the Flash Memory.

1. Setup Color Depth of Main Window

```
Select_Main_Window_16bpp();          // 16bit Color Depth
```

2. Setup the Start Address and Width of Main Window

```
Main_Image_Start_Address(0);         // Setup Start Address of Main Window is 0
Main_Image_Width(1024);              // Width of Main Window
```

3. Setup Start Coordinate of Main Window

```
Main_Window_Start_XY(0, 0);          // Start Coordinate is (0, 0)
```

4. Setup Start Address and Width of Canvas Window

```
Canvas_Image_Start_address(0);       // Start Address is 0 (Write Data)
Canvas_image_width(1024);            // Width of Canvas
```

5. Setup the Start Coordinate and Display Area of Active Window

```
Active_Window_XY(0, 0);              // Start Display Coordinate is (0, 0)
Active_Window_WH(1024, 600);         // LCD Display Area is 1024*600
```

6. Get Picture Data from External SPI Flash(DMA Mode) then Write to Internal Display Memory

```
LT768_DMA_24bit_Block(0, 0, 0, 0, 1024, 600, 0);
```

The first 5 steps of example 2 and example 1 are identical, but the function of the 6th step is different. The sixth step is use DMA transfer feature, please refer to Section 15.1.2.

8.4 Picture-In-Picture (PIP)

■ PIP Window Setting

```
void LT768_PIP_Init
(
unsigned char On_Off,              // 0: Disable PIP, 1: Enable PIP, 2: Keep State
unsigned char Select_PIP,          // 1: Use PIP1, 2: Use PIP2
unsigned long PAddr,               // Start Address of PIP
unsigned short XP,                 // X-axis of PIP Window, must be divisible by 4.
unsigned short YP,                 // Y-axis of PIP Window, must be divisible by 4.
unsigned long ImageWidth,          // Width of Image
unsigned short X_Dis,              // X-axis of Display Window
unsigned short Y_Dis,              // Y-axis of Display Window
unsigned short X_W,                // Width of Display Window, must be divisible by 4
unsigned short Y_H                 // Length of Display Window, must be divisible by 4
)
```

Example: Show a 300*300 PIP1 picture on Main Window

```
LT768_PIP_Init(1, 1, LCD_XSIZE_TFT*LCD_YSIZE_TFT*2, 0, 0, LCD_XSIZE_TFT, 0, 0, 300, 300);
```

■ Setup Display Coordinate of PIP Window

```
void LT768_Set_DisWindowPos
(
unsigned char On_Off,              // 0: Disable PIP, 1: Enable PIP, 2: Keep State
unsigned char Select_PIP,          // 1: Use PIP1, 2: Use PIP2
unsigned short X_Dis,              // Setup X-Coordinate
unsigned short Y_Di                // Setup Y-Coordinate
)
```

The function can change the position of the PIP by modifying the coordinates.

■ PIP Disable Setting

```
Disable_PIP1();                    // Disable PIP1
Disable_PIP2();                    // Disable PIP2
```

8.5 Image Rotate and Mirror

When the MCU writes data to the display memory, LT768X provides the function of data rotation and mirroring. Therefore, the MCU needs to set this function before writing data to display memory. However, data rotation and mirroring are not supported when using DMA to transfer data from SPI Flash to display memory.

The register REG[02h] bit[2:1] is used to control the direction of MCU write data to Display Memory:

```
void MemWrite_Left_Right_Top_Down(void);     // REG[02h] bit[2:1]=00b,
                                             // Left→Right, then Top→Down (Default)

void MemWrite_Right_Left_Top_Down(void);     // REG[02h] bit[2:1]=01b
                                             // Right→Left, then Top→Down (Flip Horizontal)

void MemWrite_Top_Down_Left_Right(void);     // REG[02h] bit[2:1]=10b, Top→Down, then
                                             // Left→Right (Right Rotate 90° & Flip Horizontal)

void MemWrite_Down_Top_Left_Right(void);     // REG[02h] bit[2:1]=11b, Down→Top, then
                                             // Left→Right (Left Rotate 90°)
```

The register REG[12h] bit3 (VDIR) is used to control the direction of vertical scan:

```
void VSCAN_T_to_B(void);      // REG[12h] bit3=0, from Top to Bottom

void VSCAN_B_to_T(void);      // REG[12h] bit3=1, from Bottom to Top
```

Therefore, there are 8 combinations of rotation and mirroring.

```
1、VSCAN_T_to_B();
   MemWrite_Left_Right_Top_Down();

2、VSCAN_T_to_B();
   MemWrite_Right_Left_Top_Down();

3、VSCAN_T_to_B();
   MemWrite_Top_Down_Left_Right();

4、VSCAN_T_to_B();
   MemWrite_Down_Top_Left_Right();

5、VSCAN_B_to_T ();
   MemWrite_Left_Right_Top_Down();

6、VSCAN_B_to_T ();
   MemWrite_Right_Left_Top_Down();

7、VSCAN_B_to_T ();
   MemWrite_Top_Down_Left_Right();

8、VSCAN_B_to_T ();
   MemWrite_Down_Top_Left_Right();
```


8.6 Display Color-Bar

LT768X provides a Color-Bar display feature that can be used as a display test and does not require the use of internal display memory.

```
void LT768_Color_Bar_ON(void);      // Color-Bar Enable
void LT768_Color_Bar_OFF(void);     // Color-Bar Disable
```

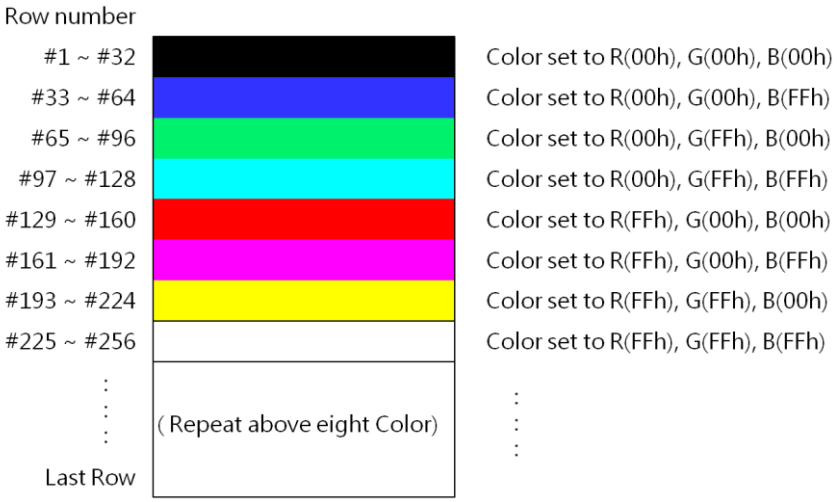

Figure 8-1: Display Color-Bar

9. Geometric Drawing

9.1 Drawing Line

9.1.1 Drawing a Thin Line

```
void LT768_DrawLine
(
unsigned short X1,          // X1-Axis
unsigned short Y1,          // Y1-Axis
unsigned short X2,          // X2-Axis
unsigned short Y2,          // Y2-Axis
unsigned long LineColor     // Line Color
)
```

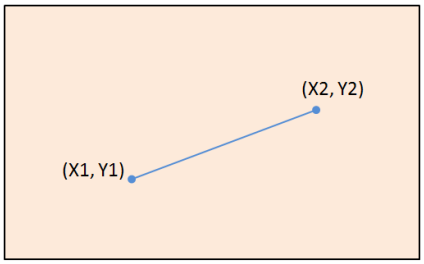

Figure 9-1: Drawing a Thin Line

Example: Drawing a red Thin Line from (100, 100) to (200, 200)

```
LT768_DrawLine(100, 100, 200, 200, Red);
```

9.1.2 Drawing a Thick Line

```
void LT768_DrawLine_Width
(
unsigned short X1,          // X1-Axis
unsigned short Y1,          // Y1-Axis
unsigned short X2,          // X2-Axis
unsigned short Y2,          // Y2-Axis
unsigned long LineColor,    // Line Color
unsigned short Width        // Line Width
)
```

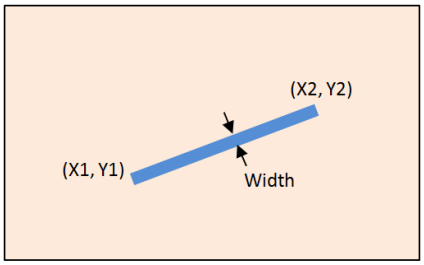

Figure 9-2: Drawing a Thick Line

Example: Drawing a red Thick Line from (120, 140) to (220, 260), and set the line width = 10.

```
LT768_DrawLine(120, 140, 220, 260, Red, 10);
```

9.2 Drawing Circle

9.2.1 Drawing a Hollow Circle

```
void LT768_DrawCircle
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short R,                // Radius
unsigned long CircleColor        // Color
)
```

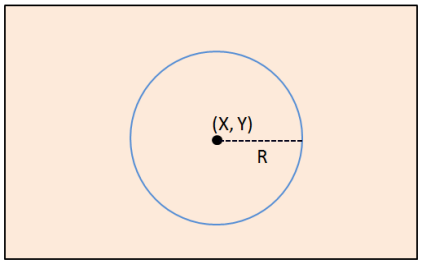

Figure 9-3: Drawing a Hollow Circle

Example: Drawing a red Circle with a radius of 100, and the center is at (200, 200)

```
LT768_DrawCircle(200, 200, 100, Red);
```

9.2.2 Drawing a Solid Circle

```
void LT768_DrawCircle_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short R,                // Radius
unsigned long ForegroundColor    // Color
)
```

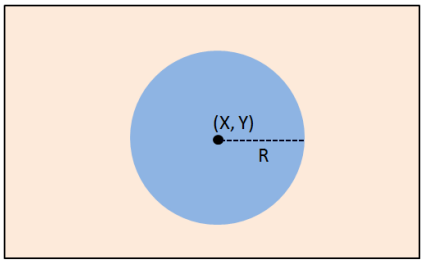

Figure 9-4: Drawing a Solid Circle

Example: Drawing a red Solid Circle with a radius of 100, and the center is at (200, 200)

```
LT768_DrawCircle_Fill (200, 200, 100, Red);
```

9.2.3 Drawing a Solid Circle with Frame

```
void LT768_DrawCircle_Width
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short R,                // Radius
unsigned long CircleColor,       // Frame Color
unsigned long ForegroundColor,   // Foreground Color
unsigned short Width             // Frame Width
)
```

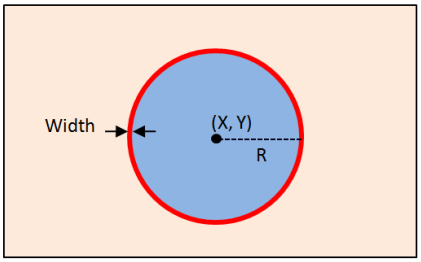

Figure 9-5: Drawing a Solid Circle with Frame

Example: Drawing a white solid circle with a red frame, the frame is 10, the radius is 100, and the center is at (200, 200)

```
LT768_DrawCircle_Width (200, 200, 100, Red, White, 10);
```

Note: This function is completed by drawing two Solid Circle. The frame color is determined by the first Solid Circle, and foreground color is determined by the second Solid Circle.

9.3 Drawing Ellipse

9.3.1 Drawing a Hollow Ellipse

```
void LT768_DrawEllipse
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long EllipseColor       // Color
)
```

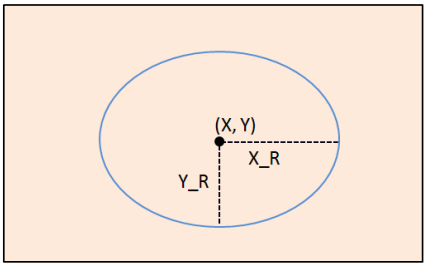

Figure 9-6: Drawing a Hollow Ellipse

Example: Drawing a red Ellipse with one radius = 80 and another = 50, and the center is at (100, 100)

```
LT768_DrawEllipse(100, 100, 80, 50, Red);
```

9.3.2 Drawing a Solid Ellipse

```
void LT768_DrawEllipse_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long ForegroundColor    // Foreground Color
)
```

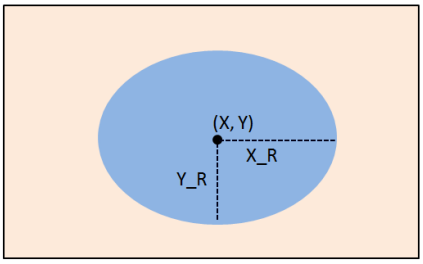

Figure 9-7: Drawing a Solid Ellipse

Example: Drawing a red Solid Ellipse with one radius = 80 and another = 50, and the center is at (100, 100)

```
LT768_DrawEllipse_Fill (100, 100, 80, 50, Red);
```

9.3.3 Drawing a Solid Ellipse with Frame

```
void LT768_DrawEllipse_Width
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long EllipseColor,      // Frame Color
unsigned long ForegroundColor,   // Foreground Color
unsigned short Width             // Frame Width
)
```

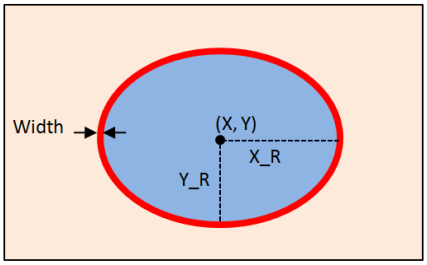

Figure 9-8: Drawing a Solid Ellipse with Frame

Example: Drawing a white Solid Ellipse with a red frame, the frame is 10, the radiuses are 80 and 50, and the center is at (100, 100)

```
LT768_DrawEllipse_Width (100, 100, 80, 50, Red, White, 10);
```

Note: This function is completed by drawing two Solid Ellipses. The frame color is determined by the first Solid Ellipse, and foreground color is determined by the second Solid Ellipse.

9.4 Drawing Rectangle

9.4.1 Drawing a Hollow Rectangle

```
void LT768_DrawSquare
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X1-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned long SquareColor       // Color
)
```

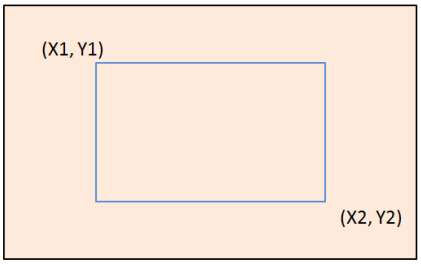

Figure 9-9: Drawing a Hollow Rectangle

Example: Drawing a red Rectangle with one corner = (50, 60) and another = (200,150).

```
LT768_DrawSquare(50, 60, 200, 150, Red);
```

9.4.2 Drawing a Solid Rectangle

```
void LT768_DrawSquare_Fill
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned long ForegroundColor   // Foreground Color
)
```

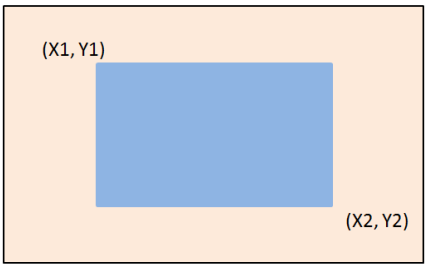

Figure 9-10: Drawing a Solid Rectangle

Example: Drawing a red Solid Rectangle with one corner = (50, 60) and another = (200,150).

```
LT768_DrawSquare_Fill(50, 60, 200, 150, Red);
```

9.4.3 Drawing a Solid Rectangle with Frame

```
void LT768_DrawSquare_Width
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned long SquareColor,      // Frame Color
unsigned long ForegroundColor,  // Foreground Color
unsigned short Width            // Frame Width
)
```

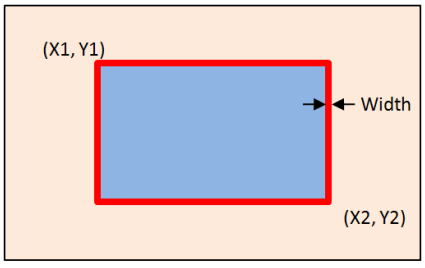

Figure 9-11: Drawing a Solid Rectangle with Frame

Example: Drawing a white solid Rectangle with a red frame. The frame width is 10, and the two corners are (50, 60) and (200, 150).

```
LT768_DrawSquare_Width(50, 60, 200, 150, Red, White, 10);
```

Note: This function is completed by drawing two Solid Rectangles. The frame color is determined by the first Solid Rectangle, and foreground color is determined by the second Solid Rectangle.

9.5 Drawing Rounded-Rectangle

9.5.1 Drawing a Hollow Rounded-Rectangle

```
void LT768_DrawCircleSquare
(
unsigned short X1,                  // Corner X1-Axis
unsigned short Y1,                  // Corner Y1-Axis
unsigned short X2,                  // Corner X2-Axis
unsigned short Y2,                  // Corner Y2-Axis
unsigned short X_R,                 // X-Axis Radius
unsigned short Y_R,                 // Y-Axis Radius
unsigned long CircleSquareColor     // Color
)
```

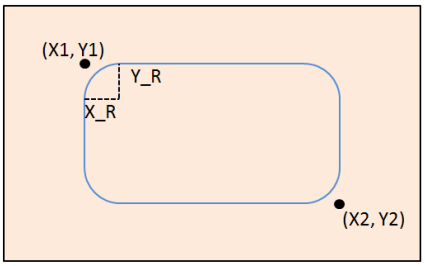

Figure 9-12: Drawing a Hollow Rounded-Rectangle

Example: Drawing a red Rounded-Rectangle with one corner radius = 30, and another = 20, , and the corners are at (50, 60) and (200, 150).

```
LT768_DrawCircleSquare(50, 60, 200, 150, 30, 20, Red);
```

9.5.2 Drawing a Solid Rounded-Rectangle

```
void LT768_DrawCircleSquare_Fill
(
unsigned short X1,                  // Corner X1-Axis
unsigned short Y1,                  // Corner Y1-Axis
unsigned short X2,                  // Corner X2-Axis
unsigned short Y2,                  // Corner Y2-Axis
unsigned short X_R,                 // X-Axis Radius
unsigned short Y_R,                 // Y-Axis Radius
unsigned long ForegroundColor,      // Foreground Color
)
```

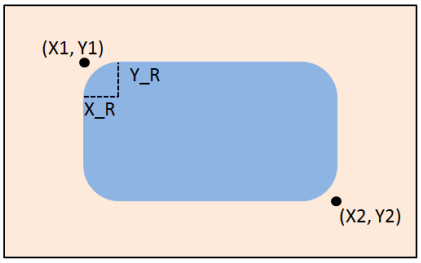

Figure 9-13: Drawing a Solid Rounded-Rectangle

Example: Drawing a red Solid Rounded-Rectangle with one corner radius = 30, and another = 20, and the corners are at (50, 60) and (200, 150).

```
LT768_DrawCircleSquare_Fill(50, 60, 200, 150, 30, 20, Red);
```

9.5.3 Drawing a Rounded-Rectangle with Frame

```
void LT768_DrawCircleSquare_Width
(
unsigned short X1,                  // Corner X1-Axis
unsigned short Y1,                  // Corner Y1-Axis
unsigned short X2,                  // Corner X2-Axis
unsigned short Y2,                  // Corner Y2-Axis
unsigned short X_R,                 // X-Axis Radius
unsigned short Y_R,                 // Y-Axis Radius
unsigned long CircleSquareColor,    // Frame Color
unsigned long ForegroundColor,      // Foreground Color
unsigned short Width                // Frame Width
)
```

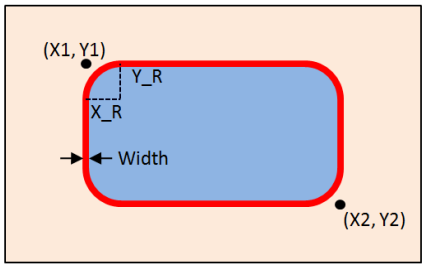

Figure 9-14: Drawing a Solid Rounded-Rectangle with Frame

Example: Drawing a white Rounded-Rectangle with a red frame, the radiuses are 30 and 20, the frame width is10, and the corners are at (50, 60) and (200, 150).

```
LT768_DrawCircleSquare_Width (50, 60, 200, 150, 30, 20, Red, White, 10);
```

Note: This function is completed by drawing two Rounded-Rectangles. The frame color is determined by the first solid Rounded-Rectangle, and foreground color is determined by the second Rounded-Rectangle.

9.6 Drawing Triangle

9.6.1 Drawing a Hollow Triangle

```
void LT768_DrawTriangle
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned long TriangleColor     // Color
)
```

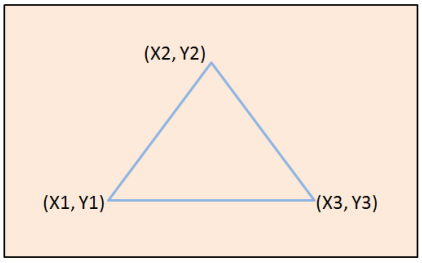

Figure 9-15: Drawing a Hollow Triangle

Example: Drawing a red Hollow Triangle with corners = (100,100), (50, 200), and (150, 150) respectively.

```
LT768_DrawTriangle(100, 100, 50, 200, 150, 150, Red);
```

9.6.2 Drawing a Solid Triangle

```
void LT768_DrawTriangle_Fill
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned long ForegroundColor   // Foreground Color
)
```

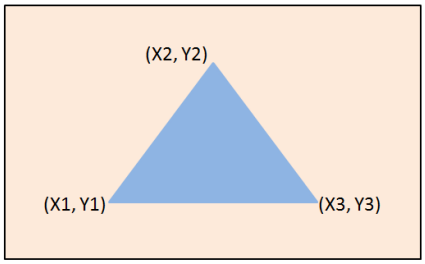

Figure 9-16: Drawing a Solid Rectangle

Example: Drawing a red Solid Triangle with corners = (100,100), (50, 200), and (150, 150) respectively.

```
LT768_DrawTriangle_Fill (100, 100, 50, 200, 150, 150, Red);
```

9.6.3 Drawing a Solid Rectangle with Frame

```
void LT768_DrawTriangle_Frame
(
unsigned short X1,                  // Corner X1-Axis
unsigned short Y1,                  // Corner X1-Axis
unsigned short X2,                  // Corner X1-Axis
unsigned short Y2,                  // Corner X1-Axis
unsigned short X3,                  // Corner X1-Axis
unsigned short Y3,                  // Corner X1-Axis
unsigned long TriangleColor,        // Frame Color
unsigned long ForegroundColor       // Foreground Color
)
```

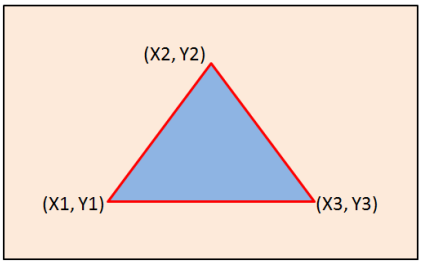

Figure 9-17: Drawing a Solid Rectangle with Frame

Example: Drawing a white Solid Triangle with a red Frame, and corners = (100,100), (50, 200), and (150, 150) respectively.

```
void LT768_DrawTriangle_Frame(100, 100, 50, 200, 150, 150, Red, White);
```

Note: This function is completed by drawing a Solid and a Hollow Triangle. The foreground color is determined by the Solid Rectangle, and frame color is determined by the Hollow Rectangle.

9.7 Drawing Curve

9.7.1 Drawing an Upper-Left Curve

```
void LT768_DrawLeftUpCurve
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long CurveColor         // Color
)
```

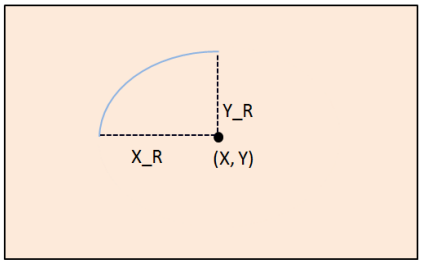

Figure 9-18: Drawing an Upper-Left Curve

Example: Drawing a red Upper-Left Curve with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawLeftUpCurve(100, 100, 100, 70, Red);
```

9.7.2 Drawing a Lower-Left Curve

```
void LT768_DrawLeftDownCurve
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long CurveColor         // Color
)
```

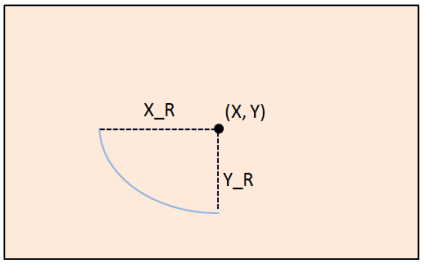

Figure 9-19: Drawing a Lower-Left Curve

Example: Drawing a red Lower-Left Curve with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawLeftDownCurve(100, 100, 100, 70, Red);
```

9.7.3 Drawing an Upper-Right Curve

```
void LT768_DrawRightUpCurve
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long CurveColor         // Color
)
```

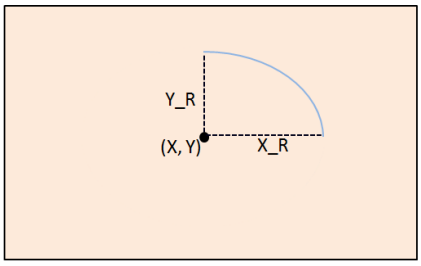

Figure 9-20: Drawing an Upper-Right Curve

Example: Drawing a red Upper-Right Curve with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawRightUpCurve(100, 100, 100, 70, Red);
```

9.7.4 Drawing a Lower-Right Curve

```
void LT768_DrawRightDownCurve
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long CurveColor         // Color
)
```

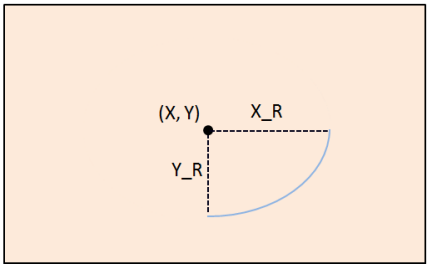

Figure 9-21: Drawing a Lower-Right Curve

Example: Drawing a red Lower-Right Curve with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawRightDownCurve(100, 100, 100, 70, Red);
```

9.8 Drawing 1/4 Ellipse

9.8.1 Drawing an Upper-Left 1/4 Ellipse

```
void LT768_DrawLeftUpCurve_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long ForegroundColor    // Color
)
```

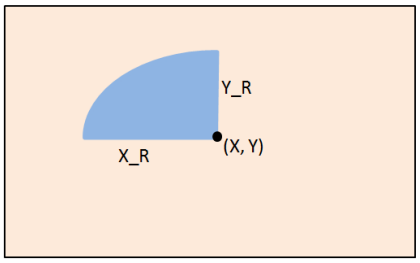

Figure 9-22: Drawing an Upper-Left 1/4 Ellipse

Example: Drawing a red Upper-Left 1/4 Ellipse with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawLeftUpCurve_Fill(100, 100, 100, 70, Red);
```

9.8.2 Drawing an Lower-Left 1/4 Ellipse

```
void LT768_DrawLeftDownCurve_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long ForegroundColor    // Color
)
```

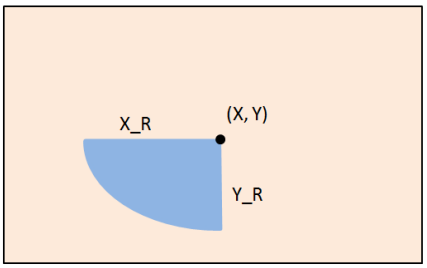

Figure 9-23: Drawing a Lower-Left 1/4 Ellipse

Example: Drawing a red Lower-Left 1/4 Ellipse with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawLeftDownCurve_Fill(100, 100, 100, 70, Red);
```

9.8.3 Drawing an Upper-Right 1/4 Ellipse

```
void LT768_DrawRightUpCurve_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long ForegroundColor    // Color
)
```

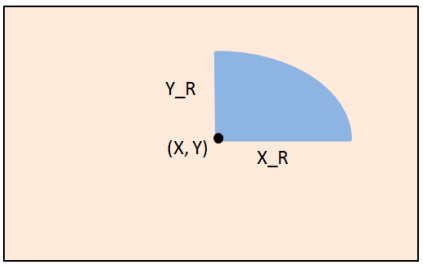

Figure 9-24: Drawing an Upper-Right 1/4 Ellipse

Example: Drawing a red Upper-Right 1/4 Ellipse with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawRightUpCurve_Fill(100, 100, 100, 70, Red);
```

9.8.4 Drawing an Lower-Right 1/4 Ellipse

```
void LT768_DrawRightDownCurve_Fill
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center Y-Axis
unsigned short X_R,              // X-Axis Radius
unsigned short Y_R,              // Y-Axis Radius
unsigned long ForegroundColor    // Color
)
```

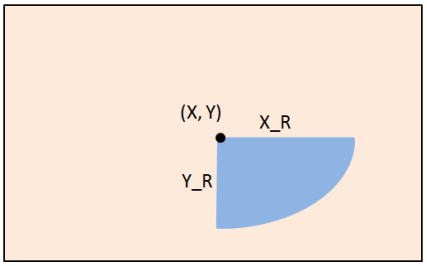

Figure 9-25: Drawing a Lower-Right 1/4 Ellipse

Example: Drawing a red Lower-Right 1/4 Ellipse with one radius = 100 and another = 70, and the center is at (100, 100)

```
LT768_DrawRightDownCurve_Fill(100, 100, 100, 70, Red);
```

9.9 Drawing Quadrilateral

9.9.1 Drawing a Hollow Quadrilateral

```
void LT768_DrawQuadrilateral
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned short X4,              // Corner X4-Axis
unsigned short Y4,              // Corner Y4-Axis
unsigned long ForegroundColor   // Frame Color
)
```

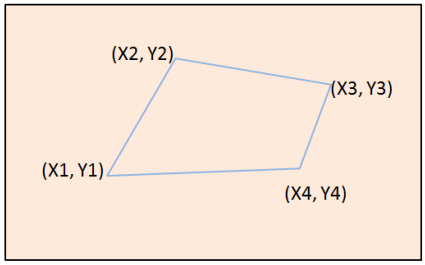

Figure 9-26: Drawing a Hollow Quadrilateral

Example: Drawing a red Hollow Quadrilateral with the corners = (50, 50), (200, 80), (150, 130) and (60, 100) respectively.

```
LT768_DrawQuadrilateral(50, 50, 200, 80, 150, 130, 60, 100, Red);
```

Note: The quadrilateral can be arbitrarily set coordinates of four corners. But the rectangle only need sets the coordinates of two corners.

9.9.2 Drawing a Solid Quadrilateral

```
void LT768_DrawQuadrilateral_Fill
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned short X4,              // Corner X4-Axis
unsigned short Y4,              // Corner Y4-Axis
unsigned long ForegroundColor   // Foreground Color
)
```

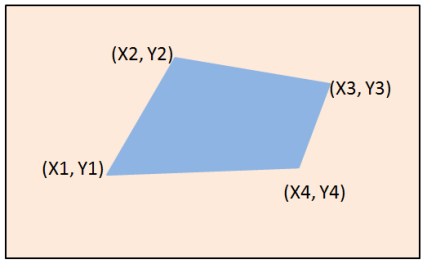

Figure 9-27: Drawing a Solid Quadrilateral

Example: Drawing a red Solid Quadrilateral with the corners = (50, 50), (200, 80), (150, 130) and (60, 100) respectively.

```
LT768_DrawQuadrilateral_Fill(50, 50, 200, 80, 150, 130, 60, 100, Red);
```

9.10 Drawing Pentagonal

9.10.1 Drawing a Hollow Pentagonal

```
void LT768_DrawPentagon
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned short X4,              // Corner X4-Axis
unsigned short Y4,              // Corner Y4-Axis
unsigned short X5,              // Corner X5-Axis
unsigned short Y5,              // Corner Y5-Axis
unsigned long ForegroundColor   // Frame Color
)
```

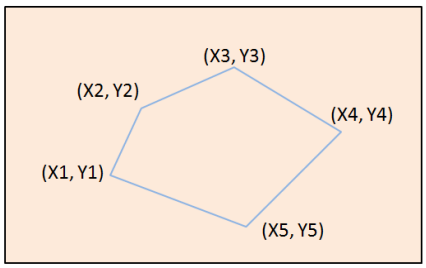

Figure 9-28: Drawing a Hollow Pentagonal

Example: Drawing a red Hollow Pentagonal with the coners = (50, 100), (120, 130), (150, 160), (100, 180) and (80, 140) respectively.

```
void LT768_DrawPentagon(50, 100, 120, 130, 150, 160, 100, 180, 80, 140, Red);
```

9.10.2 Drawing a Solid Pentagonal

```
void LT768_DrawPentagon_Fill
(
unsigned short X1,              // Corner X1-Axis
unsigned short Y1,              // Corner Y1-Axis
unsigned short X2,              // Corner X2-Axis
unsigned short Y2,              // Corner Y2-Axis
unsigned short X3,              // Corner X3-Axis
unsigned short Y3,              // Corner Y3-Axis
unsigned short X4,              // Corner X4-Axis
unsigned short Y4,              // Corner Y4-Axis
unsigned short X5,              // Corner X5-Axis
unsigned short Y5,              // Corner Y5-Axis
unsigned long ForegroundColor   // Foreground Color
)
```

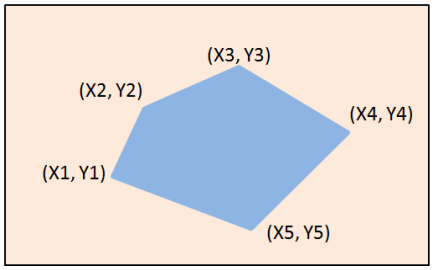

Figure 9-29: Drawing a Solid Pentagonal

Example: Drawing a red Solid Pentagonal with corners = (50, 100), (120, 130), (150, 160), (100, 180) and (80, 140) respectively.

```
void LT768_DrawPentagon_Fill (50, 100, 120, 130, 150, 160, 100, 180, 80, 140, Red);
```

9.11 Drawing Cylinder

```
unsigned char LT768_DrawCylinder
(
unsigned short XCenter,          // Center X-Axis
unsigned short YCenter,          // Center X-Axis
unsigned short X_R,              // X Radius
unsigned short Y_R,              // Y Radius
unsigned short H,                // Cylinder Height
unsigned long CylinderColor,     // Frame Color
unsigned long ForegroundColor    // Foreground Color
)
```

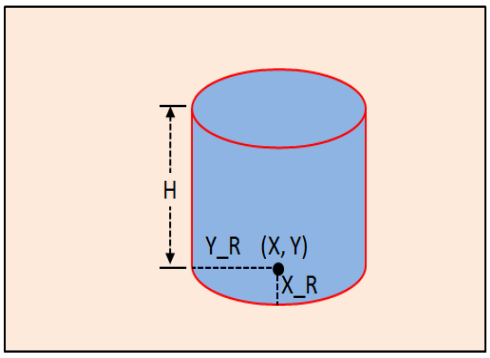

Figure 9-30: Drawing a Cylinder

Example: Drawing a blue Cylinder with a red frame, the radiuses are 100 and 50, the height is 150, and the center is at (200, 300).

```
LT768_DrawCylinder (200, 300, 50, 100, 150, Red, Blue);
```

Note: The implementation of drawing a Cylinder is integrated by some of the drawing functions mentioned in this Chapter, and the parameters of the Ellipse in this function are based on the bottom Ellipse.

9.12 Drawing Cube

```
void LT768_DrawQuadrangular
(
 unsigned short X1,                  // Upper-Left X1-Axis
 unsigned short Y1,                  // Upper-Left Y1-Axis
 unsigned short X2,                  // Upper-Left X2-Axis
 unsigned short Y2,                  // Upper-Left Y2-Axis
 unsigned short W,                   // Width
 unsigned short H,                   // Height
 unsigned long QuadrangularColor,    // Frame Color
 unsigned long ForegroundColor       // Foreground Color
)
```

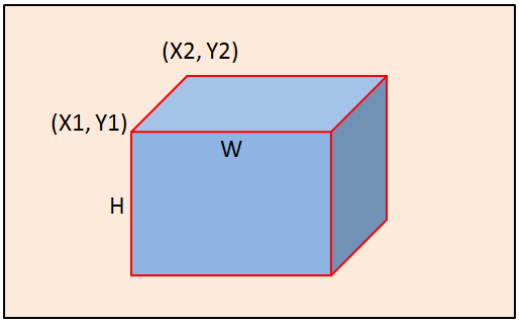

Figure 9-31: Drawing a Cube

Example: Drawing a blue Cube with a red frame, the Parallelogram Corners are (100, 100), (140, 60), the width is 150, and height is 200.

```
LT768_DrawQuadrangular (100, 100, 140, 60, 150, 200, Red, Blue);
```

Note: The implementation of drawing a Cube is integrated by some of the drawing functions mentioned in this Chapter.

9.13 Drawing Table

```
void LT768_MakeTable
(
unsigned short X1,                  // Start Corner X-Axis
unsigned short Y1,                  // Start Corner Y-Axis
unsigned short W,                   // Sub-Table Width (W)
unsigned short H,                   // Sub-Table Height (H)
unsigned short Line,                // Column Number (CN)
unsigned short Row,                 // Row Number (RN)
unsigned long   TableColor,         // Frame Color (C1)
unsigned long   ItemColor,          // Item Table Color (C2)
unsigned long   ForegroundColor,    // Content Table Color (C3)
unsigned short width1,              // Inner Frame Width
unsigned short width2,              // Outer Frame Width
unsigned char mode                  // 0: Item Table is Vertical, 1: Item Table is Horizontal
)
```

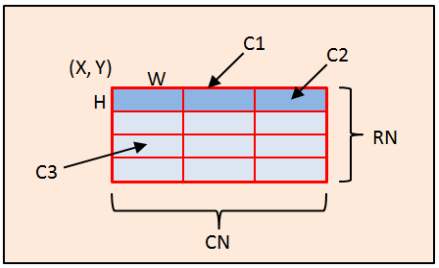

Figure 9-32: Drawing a Horizontal Table

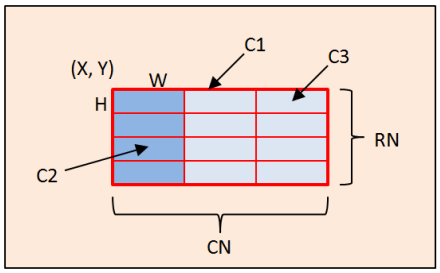

Figure 9-33: Drawing a Vertical Table

Example: Drawing a horizontal table window at (100, 100) position. The sub-table size is 80*40, table number of row is 3, table number of column is 4, the frame color is red, foreground color of Item table is green, foreground color of content table is blue, inner frame width is 1, outer frame width is 3.

```
LT768_DrawQuadrangular (100, 100, 80, 40, 3, 4, Red, Green, Blue, 1, 3, 1);
```

Note: The implementation of drawing a Table is integrated by some of the drawing functions mentioned in this Chapter.

10. Block Transfer Engine(BTE)

10.1 BTE Operation Mode

LT768x has an embedded high performance hardware engine - Block Transfer Engine (BTE), it is designed to accelerate data Loading, Transferring, and additional Logic Processing. BTE supports 13 basic BTE Operations (Table 10-1), as well as additional functions of Raster Operation (ROP, Table 10-2), Chroma Key, Color Expansion, and so on. After getting properly set and enabled, BTE can perform the required operations and functions automatically and rapidly regardless of MCU. BTE can extremely ease MCU working loads and resources, and greatly improve the performance for the whole system.

Befroe using BTE, users must assign the operation mode of BTE. Please refer to Table10-1 for the details of the operation modes. Also, LT768x supports 16 Raster Operations (ROP, Table10-2) that allow users to combine source data (source 0 and source 1) into desitnation data in various ways.

There are two ways to get BTE working status: (1) check Register BTE_CTRL0(REG[90h]) Bit4 or Status Register (STSR) Bit3; (2) check hardware interrupt. When hardware interrupt is triggered, the interruption source can be confirmed by checking REG[0Ch].

Table 10-1: BTE Operation

BTE Operation Code REG[91h] Bits [3:0]	BTE Operation Description
0000b	MCU Write with ROP.
0010b	Memory Copy (move) with ROP.
0100b	MCU Write with chroma keying (w/o ROP)
0101b	Memory Copy (move) with chroma keying (w/o ROP)
0110b	Pattern Fill with ROP
0111b	Pattern Fill with chroma keying (w/o ROP)
1000b	MCU Write with Color Expansion (w/o ROP)
1001b	MCU Write with Color Expansion and chroma keying (w/o ROP)
1010b	Memory Copy with opacity (w/o ROP)
1011b	MCU Write with opacity (w/o ROP)
1100b	Solid Fill (w/o ROP)
1110b	Memory Copy with Color Expansion (w/o ROP)
1111b	Memory Copy with Color Expansion and chroma keying (w/o ROP)
Other combinations	Reserved

Table 10-2: ROP Function

ROP Function Code REG[91h] bit[7:4]	Boolean Function Operation (Destination, DT)
0000b	0 (Blackness)
0001b	~S0・~S1 or ~ (S0+S1)
0010b	~S0・S1
0011b	~S0
0100b	S0・~S1
0101b	~S1
0110b	S0^S1
0111b	~S0+~S1 or ~ (S0・S1)
1000b	S0・S1
1001b	~ (S0^S1)
1010b	S1
1011b	~S0+S1
1100b	S0
1101b	S0+~S1
1110b	S0+S1
1111b	1 (Whiteness)

Note:

1. Source 0 (S0) Data: comes from MCU Writing or Memory
2. Source 1 (S1) Data: comes from Memory
3. Destination (DT) Data: be wrriten to Memory
4. Memory means internal Display RAM
5. Memory, S0, S1, DT; these Short Names will be always quoted in this Chapter. For Example: If ROP function REG[91h] bit[7:4]=0xCh, then “DT = S0”, means “Transfer Source 0 data to Destination”; If ROP function REG[91h] bit[7:4]=0xEh, then “DT = S0 + S1”, means “Source 0 data + Source 1 data then transfer to Destination”

10.2 BTE Operation Detail

10.2.1 MCU Write with ROP

This feature can increase the speed at which the MCU writes data to SDRAM. The data written can be combined with raster (ROP) operations to fill in the destination memory. BTE itself provides 16 types of ROP. As depicted in Figure10-1, it is clear that Source 0 (S0) must be provided by the MCU. In this example, the POP register (REG[91h]) Bit[7:4] is set to 0x0C.

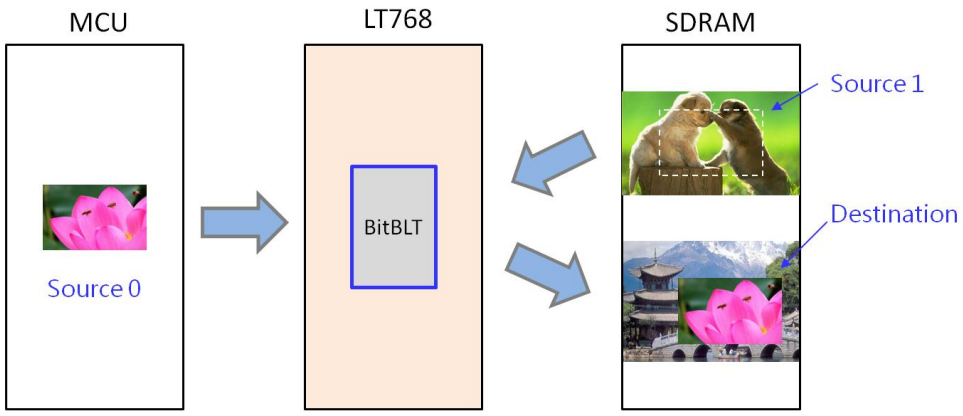

Figure 10:1: Example of MCU Write with ROP

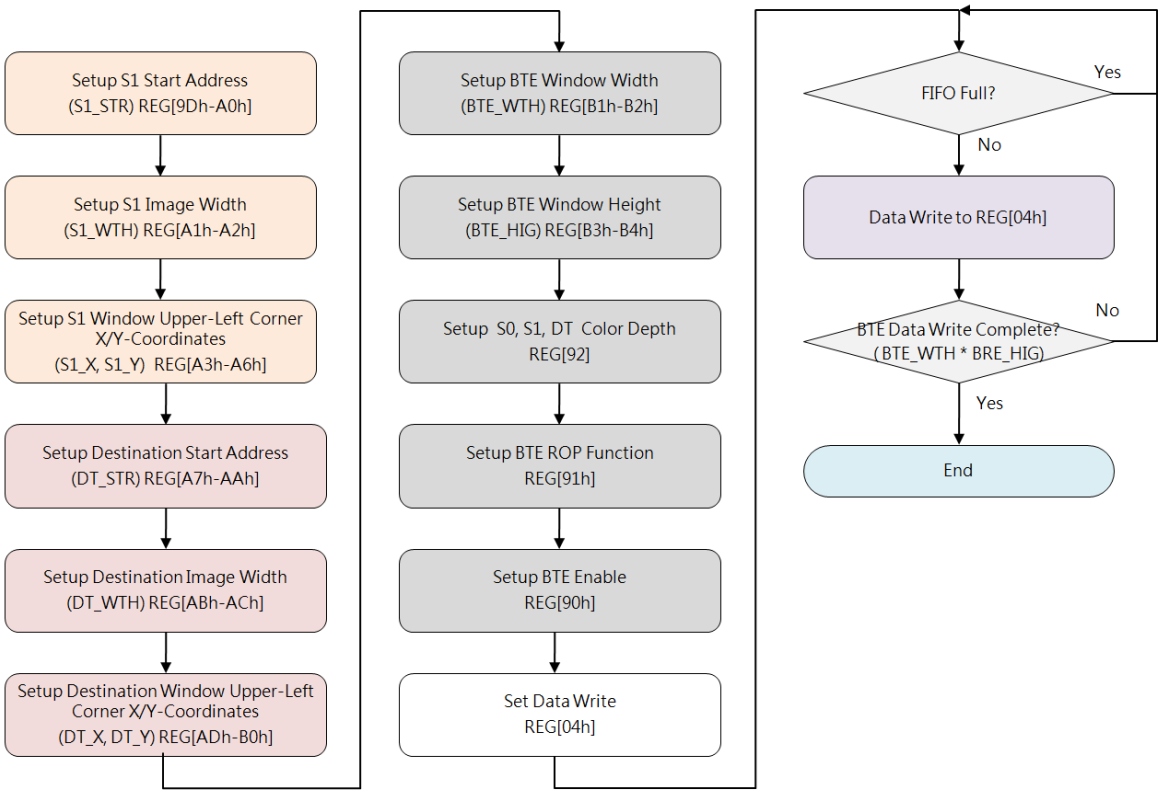

Figure 10-2: Flow Chart of MCU Wtite with ROP

```
void LT768_BTE_MCU_Write_MCU_16bit
(
unsigned long S1_Addr,              // Start address of S1 at SDRAM
unsigned short S1_W,                // Image width of S1
unsigned short XS1,                 // Upper-Left corner X-Coordinate of S1 image
unsigned short YS1,                 // Upper-Left corner Y- Coordinate of S1 image
unsigned long Des_Addr,             // Start address of DT at SDRAM
unsigned short Des_W,               // Max. Width of DT image
unsigned short XDes,                // Upper-Left corner X-Coordinate of DT
unsigned short YDes,                // Upper-Left corner Y-Coordinate of DT
unsigned int ROP_Code,              // ROP mode
unsigned short X_W,                 // Width of DT window
unsigned short Y_H,                 // Height of DT window
const unsigned short *data          // Start address of S0 image data
)
```

Example: (Assume the resolution of the LCD Panel is 1024*600)

S0: MCU write a 100*100 image with 16bits color (unsigned short Picture_Data[100*100])
S1: Start address is 1024*600*2, at (50, 50)
DT: From the Address 0 of Display RAM, at (200, 200)
DT Display Window Size: 100*100
ROP Mode: 0x0C (Show S0)

→

```
LT768_BTE_MCU_Write_MCU_16bit (1024*600*2, 1024, 50, 50, 0, 1024, 200, 200, 0x0C, 100, 100, &Picture_Data[0]);
```

10.2.2 Memory Copy (move) with ROP

This feature will move the specified memory source area to the specified memory destination area. This operation can reduce the MCU processing time, and increase the speed of copying/moving the memory data. In this example, the POP register (REG[91h]) Bit[7:4] is set to 0x0C.

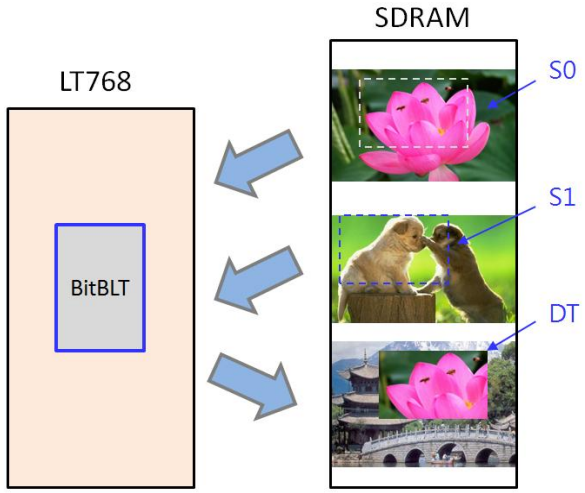

Figure 10-3: Example of Memory Copy with ROP

There are two ways to get BTE status. Figure 10-4 shows one way to get it by checking the Status Register STSR bit3, and Figure 10-5 shows another by checking hardware interruption.

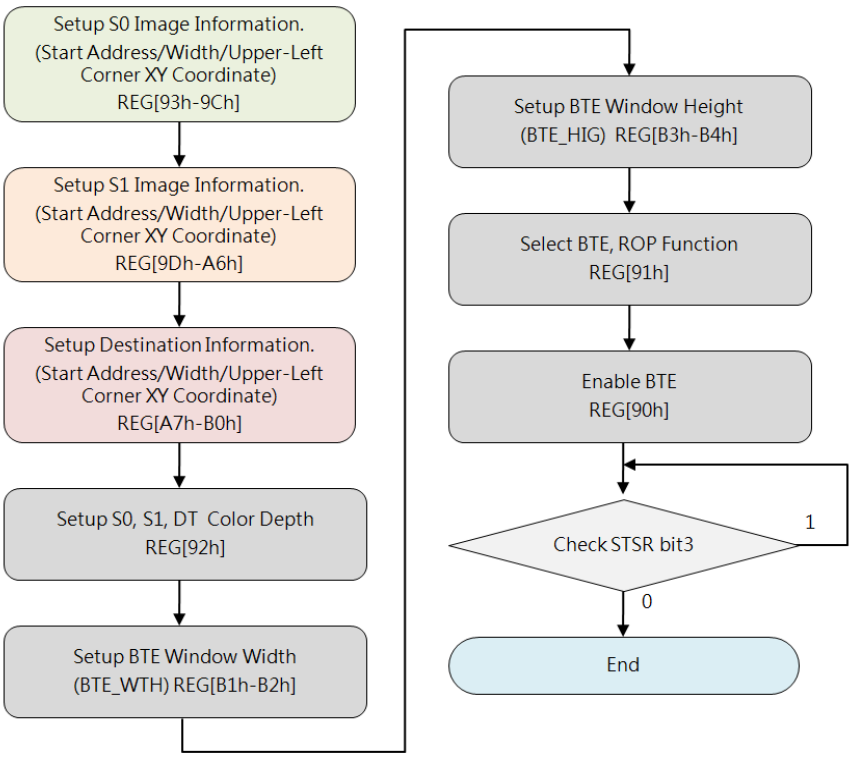

Figure 10-4: Flow Chart 1 of Memory Copy with ROP

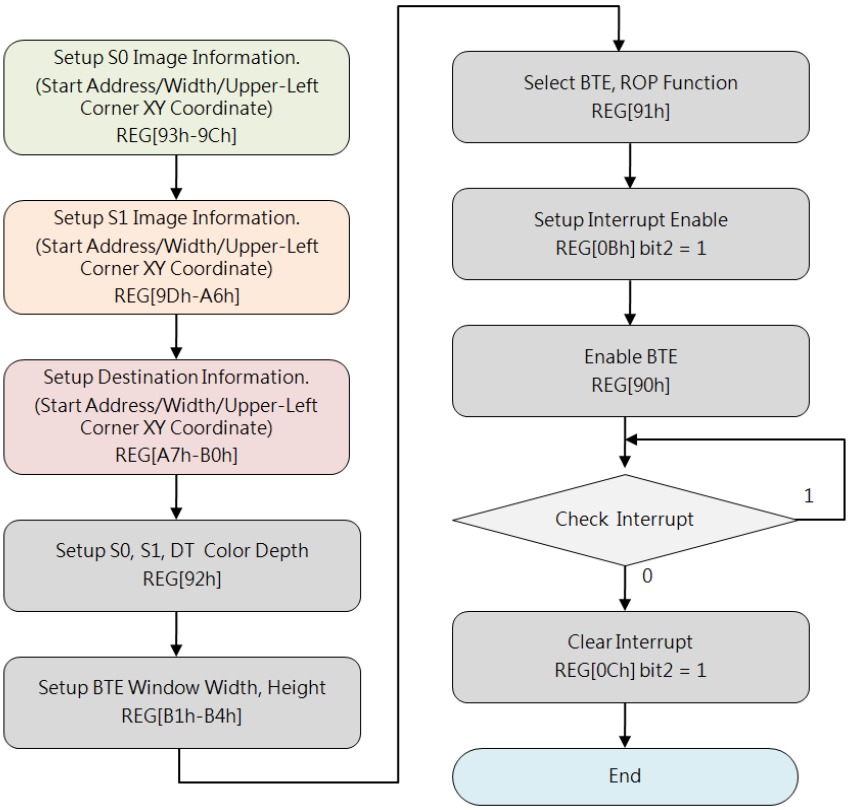

Figure 10-5: Flow Chart 2 of Memory Copy with ROP

```
void LT768_BTE_Memory_Copy
(
unsigned long S0_Addr,          // Start address of S0 at SDRAM
unsigned short S0_W,            // Image width of S0
unsigned short XS0,             // Upper-Left corner X-Coordinate of S0 image
unsigned short YS0,             // Upper-Left corner Y-Coordinate of S0 image
unsigned long S1_Addr,          // Start address of 1 at SDRAM
unsigned short S1_W,            // Image width of S1
unsigned short XS1,             // Upper-Left corner X-Coordinate of S1 image
unsigned short YS1,             // Upper-Left corner Y-Coordinate of S1 image
unsigned long Des_Addr,         // Start address of DT at SDRAM
unsigned short Des_W,           // Max. Width of DT image
unsigned short Xdes,            // Upper-Left corner X-Coordinate of DT
unsigned short Ydes,            // Upper-Left corner Y-Coordinate of DT
unsigned int ROP_Code,          // ROP mode
unsigned short X_W,             // Width of DT window
unsigned short Y_H,             // Height of DT window
)
```

Example:

S0: Start address is 0, at (100, 100)
S1: Start address is 1024*600*2 of Display RAM, at (50, 50)
DT: Start address is 1024*600*2*2 of Display RAM, at (200, 200)
DT Display Window Size: 100*100
ROP Mode: 0x0A (Show S1)

→

LT768_BTE_Memory_Copy (0, 1024, 100, 100, 1024*600*2, 1024, 50, 50, 1024*600*2*2, 1024, 200, 200, 0x0A, 100, 100);

10.2.3 MCU Write with Chroma Key (w/o ROP)

This is a MCU write data operation with Chroma-key feature. This feature can increase the speed at which the MCU writes data to SDRAM. Once this function is enabled, the BTE engine will remain busy until all data are written.

When processing data, if the color data from the MCU is the same as the Chroma-key color, then such data will be ignored. The key color is set in the "BTE background color" register. Take Figure 10-6 as an example, the picture from MCU shows “TOP” letters in red and its background color is green. If Green is selected as the Chroma-key color, then the data of green color will not be written to the memory, and the picture shown on the Destination will be the three red letters (TOP) only. Please refer to the following diagram and program flow chart:

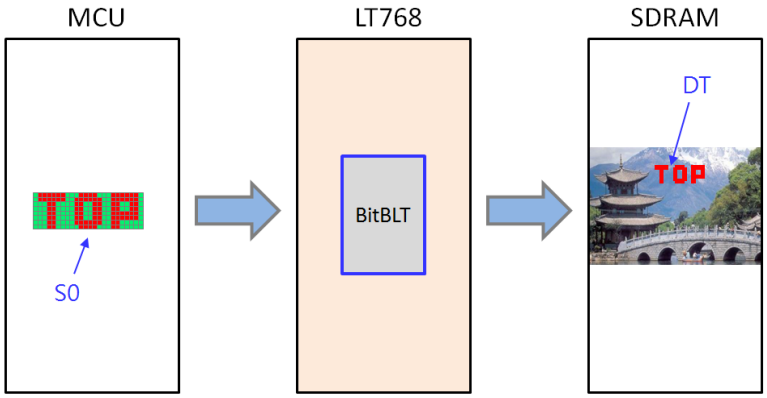

Figure 10-6: Example of MCU Write with Chroma Key

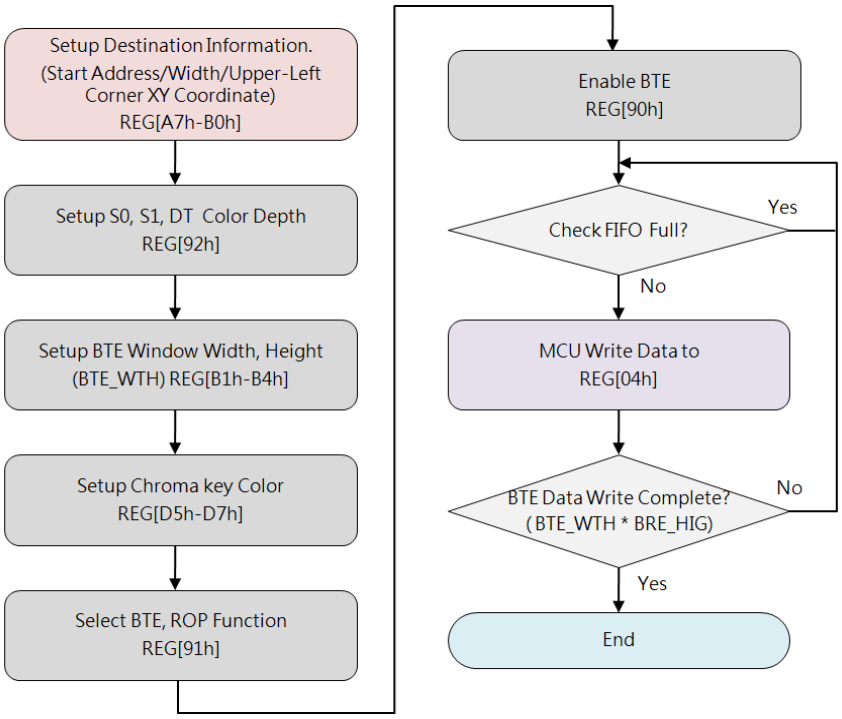

Figure 10-7: Flow Chart of MCU Write with Chroma Key

```
void LT768_BTE_MCU_Write_Chroma_key_MCU_16bit
(
unsigned long Des_Addr,                 // Start address of DT at SDRAM
unsigned short Des_W,                   // Max. Width of DT image
unsigned short Xdes,                    // Upper-Left corner X-Coordinate of DT
unsigned short YDes                     // Upper-Left corner Y-Coordinate of DT
unsigned long Background_color,         // Chroma Key (Transparent Color)
unsigned short X_W,                     // Width of DT window
unsigned short Y_H,                     // Width of DT window
const unsigned short *data              // Start address of S0 image data
)
```

Example:

S0: MCU writes a 100*100 image with 16bits color (unsigned short Picture_Data[100*100])
DT: From the Address 0 of Display RAM, at (200, 200)
DT Display Window Size: 100*100
Transparent Color: Red

→

```
LT768_BTE_MCU_Write_Chroma_key_MCU_16bit (0, 1024, 200, 200, Red, 100, 100,
&Picture_Data[0]);
```

10.2.4 Memory Copy with Chroma Key (w/o ROP)

This function is to replicate and move a specified memory Source area to the memory Destination area, and each color data of the Source will be compared to that of the Chroma Key druing the replicating process. If the color data of the Source is the same as that of the Chroma key, the data of the Destination will remain unchanged. The Chroma key color is set in the “BTE Background color” Register (REG[D7h:F5h]).

Both of the Source data and Destination data are located in Memory. Figure 10-8 shows an example: if the background color of the Source is Green and the Chroma Key color is also set to Green, then the graph written by this function will be a Red "TOP". Green becomes transparent and is not written to memory. Please refer to the following diagram and program flow chart:

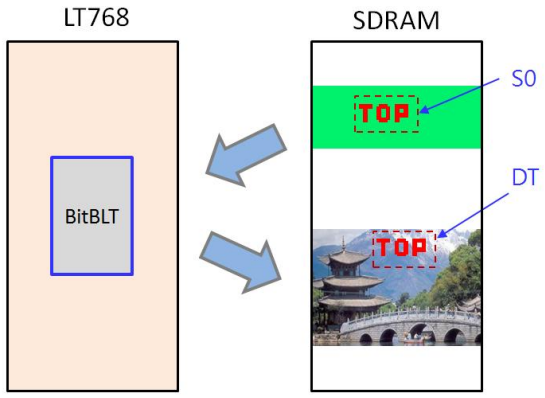

Figure 10-8: Example of Memory Copy with Chroma Key

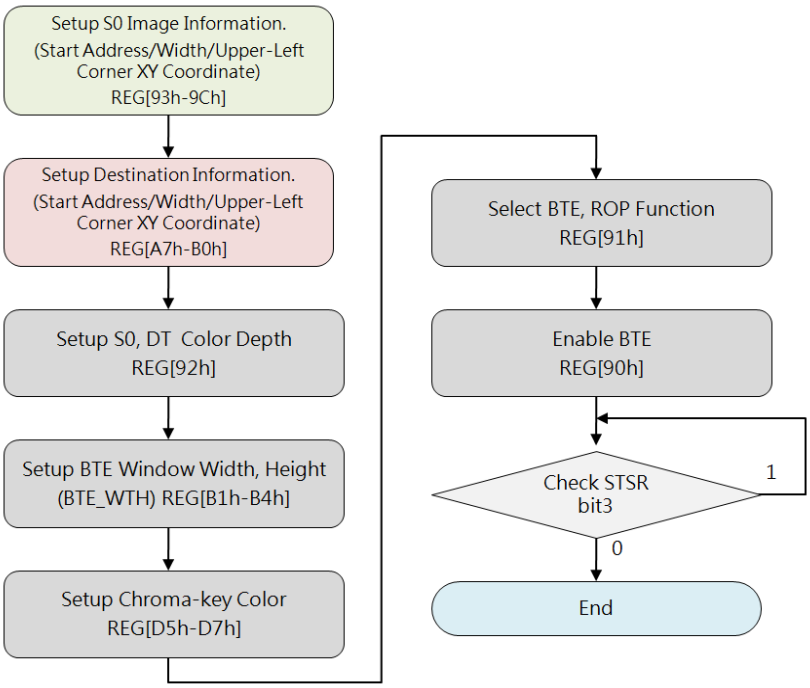

Figure 10-9: Flow Chart of Memory Copy with Chroma Key

```
void LT768_BTE_Memory_Copy_Chroma_key
(
unsigned long S0_Addr,                  // Start address of S0 at SDRAM
unsigned short S0_W,                    // Image width of S0
unsigned short XS0,                     // Upper-Left corner X-Coordinate of S0 image
unsigned short YS0,                     // Upper-Left corner Y-Coordinate of S0 image
unsigned long Des_Addr,                 // Start address of DT at SDRAM
unsigned short Des_W,                   // Width of DT image
unsigned short Xdes,                    // Upper-Left corner X-Coordinate of DT
unsigned short Ydes,                    // Upper-Left corner Y-Coordinate of DT
unsigned long Background_color,         // Chroma Key (Transparent Color)
unsigned short X_W,                     // Width of DT window
unsigned short Y_H                      // Height of DT window
)
```

Example:

S0: Start address is 0, at (200, 200)
DT: From the Address 1024*600*2 of Display RAM, at (100, 100)
DT Display Window Size: 100*100
Transparent Color: Red

→

```
LT768_BTE_Memory_Copy_Chroma_key (0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, Red, 100, 100);
```

10.2.5 Pattern Fill with ROP

This function is to repeatly fill a specified 8*8 or 16*16 pattern in a specified area. Please note that the 8*8 or 16*16 pixel pattern must be stored in the memory before applying this function. This function can also be combined with 16 kinds of raster (ROP) operations. This function can help shorten the time needed for copying patterns in a specified area.

Following is an example of Pattern Fill with ROP, where the POP register (REG[91h]) bit [7:4] is set to 0x0C.

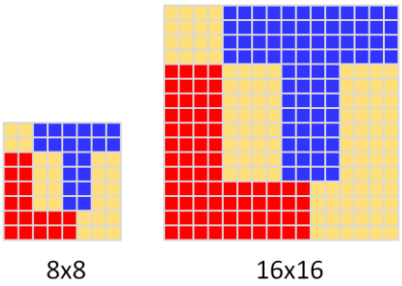

Figure 10-10: Pattern Format

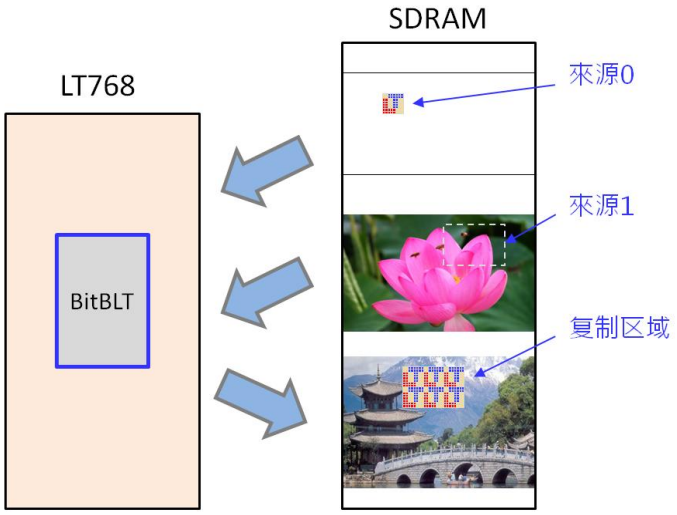

Figure 10-11: Example of Pattern Fill with ROP

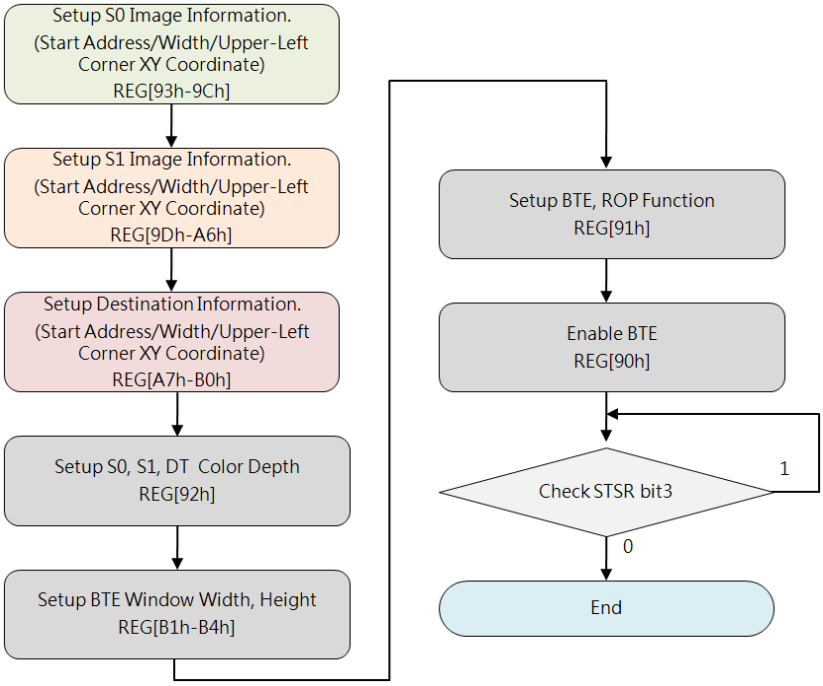

Figure 10-12: Flow Chart of Pattern Fill with ROP

```
void LT768_BTE_Pattern_Fill
(
unsigned char P_8x8_or_16x16,        // 0: select 8x8 Icon, 1: select 16x16 Icon.
unsigned long S0_Addr,               // Start address of S0 at SDRAM
unsigned short S0_W,                 // Image width of S0
unsigned short XS0,                  // Upper-Left corner X-Coordinate of S0 image
unsigned short YS0,                  // Upper-Left corner Y-Coordinate of S0 image
unsigned long Des_Addr,              // Start address of DT at SDRAM
unsigned short Des_W,                // Max. Width of DT image
unsigned short Xdes,                 // Upper-Left corner X-Coordinate of DT
unsigned short Ydes,                 // Upper-Left corner Y-Coordinate of DT
unsigned int ROP_Code,               // ROP mode
unsigned short X_W,                  // Width of DT window
unsigned short Y_H                   // Height of DT window
)
```

Example:

P_8x8_or_16x16: Select 16x16 Icon
S0: Start address is 0, at (200, 200)
DT: From the Address 1024*600*2 of Display RAM, at (100, 100)
ROP Mode: 0x0C (Show S0)
DT Display Window Size: 100*100

→

```
LT768_BTE_Pattern_Fill (1, 0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, 0x0C, 100, 100);
```

10.2.6 Pattern Fill with Chroma Key

This function is to repeatly fill an 8*8 or 16*16 pattern in a specified area, but if the color data of the Source is the same as that of Chroma Key, then such data will not be written to the Destination. Therefore, the background color of the pattern will become transparent. The chroma key color is set by REG[D5h] ~ [D7h] registers.

In below example, Orange is the background color of the Red “T” pattern, and Orange is set as Chroma Key. Therefore, BTE will repeatly fill RED “T” only to the Destination:

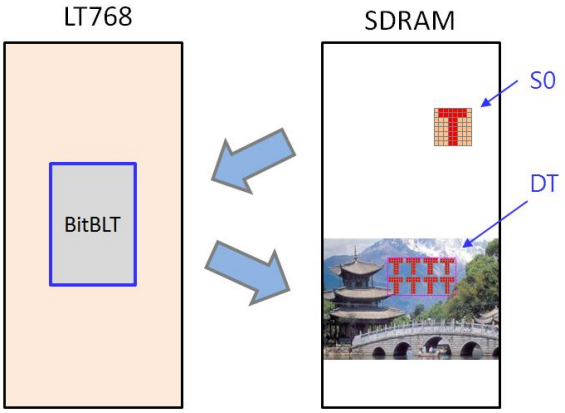

Figure 10-13: Example of Pattern Fill Chroma Key

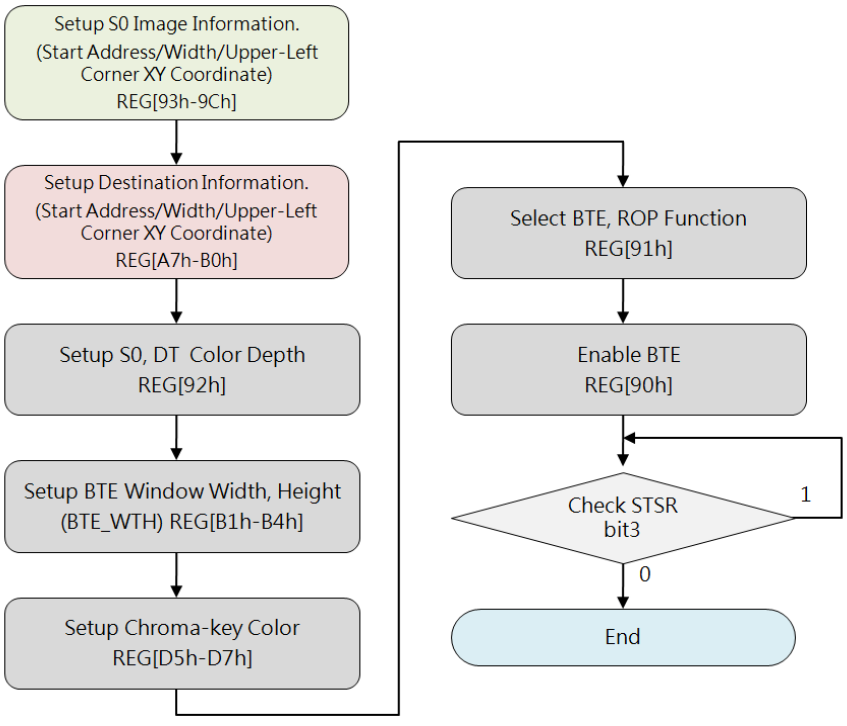

Figure 10-14: Flow Chart of Pattern Fill with Chroma Key

```
void LT768_BTE_Pattern_Fill_With_Chroma_key
(
unsigned char P_8x8_or_16x16,        // 0: use 8x8 Icon, 1: use 16x16 Icon.
unsigned long S0_Addr,               // Start address of S0 at SDRAM
unsigned short S0_W,                 // Image width of S0
unsigned short XS0,                  // Upper-Left corner X-Coordinate of S0 image
unsigned short YS0,                  // Upper-Left corner Y-Coordinate of S0 image
unsigned long Des_Addr,              // Start address of DT at SDRAM
unsigned short Des_W,                // Max. Width of DT image
unsigned short XDes,                 // Upper-Left corner X-Coordinate of DT
unsigned short YDes,                 // Upper-Left corner Y-Coordinate of DT
unsigned int ROP_Code,               // ROP mode
unsigned long Background_color,      // Chroma Key (Transparent Color)
unsigned short X_W,                  // Width of DT window
unsigned short Y_H                   // Height of DT window
)
```

Example:

P_8x8_or_16x16: Select 16x16 Icon
S0: Start address is 0, at (200, 200)
DT: From the Address 1024*600*2 of Display RAM, at (100, 100)
ROP Mode: 0x0C (Show S0)
Transparent Color: Red
DT Display Window Size: 100*100

→

```
LT768_BTE_Pattern_Fill_With_Chroma_key (1, 0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, 0x0C, Red, 100, 100);
```


10.2.7 MCU Write with Color Expansion

This function performs Color Expansion for S0 (from MCU Write). LT768x gets the monochrome data from MCU, and converts them to color data. In this operation, the source data is in Bitmap monochrome format. If the bit of the monochrome image is "1", it becomes the foreground color, and if the bit of the monochrome image is "0", it turns into a background color. This function makes it easy for users to turn a monochrome system into a color system.

Monochrome image within the BTE is processed by each scan line separately, when a scan line is processed, the corresponding monochrome scan line data will be discarded. The data of the next line will be generated by the next packet, and each of the data that will be written to Destination memory is processed from MSB to the LSB when the color is extended. If the MCU interface is set to 16bits, the starting bit of ROP can be set from 15 to 0. If the MCU interface is set to 8bits, the ROP starting bit can be set from 7 to 0. The Source 0 Color Depth (REG[92H] bit[7:6]) will not be referenced in this operation.

In the following example, the foreground color is set to Red and the background color is set to Blue. Therefore, when MCU writes monochrome data (Source 0) through BLT to specified memory area of Destination, the result is shown as Figure 10-15:

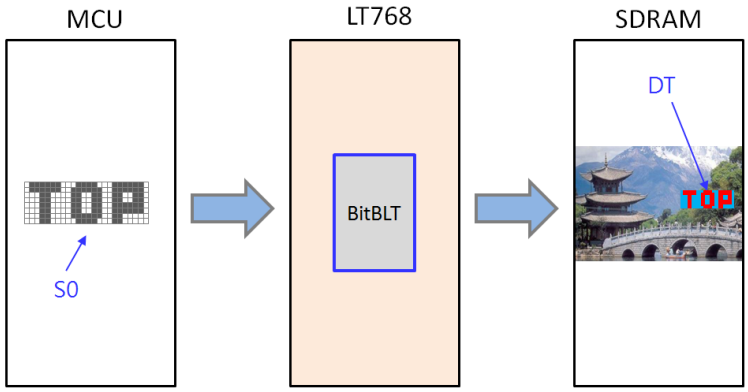

Figure 10-15: Example of MCU Write with Color Expansion

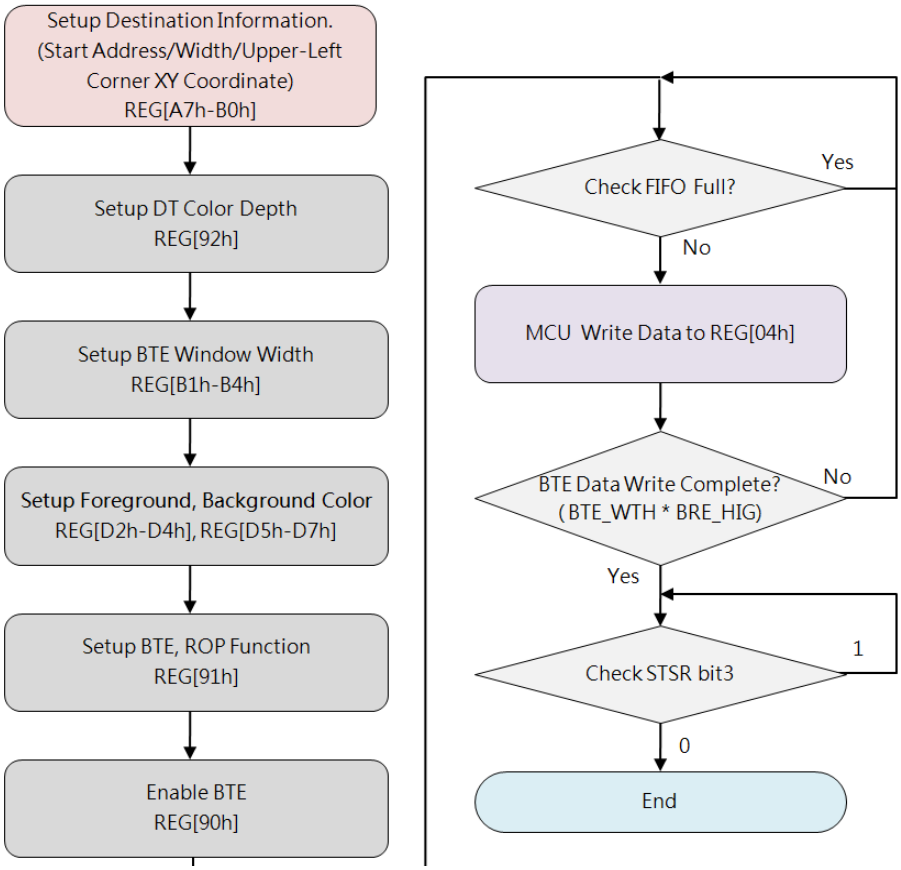

Figure 10-16: Flow Chart of MCU Write with Color Expansion

For example, ROP is set to 7, foreground color register is set to Red, and the background color register is set to Yellow. If BTE Width is 23, then the result of color extension displays as following Figure 10-17. Figure 10-18 is another example where ROP is set to 3, and other settings remain the same.

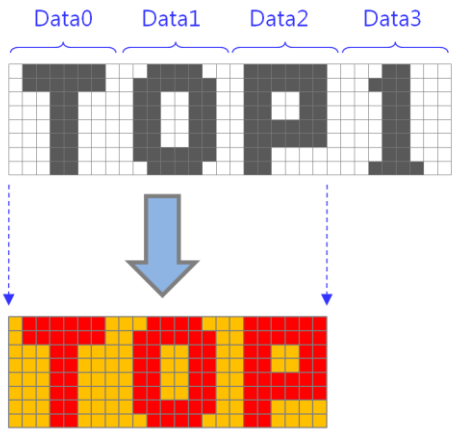

Figure 10-17: Example 1 of MCU Write with Color Expansion (ROP=7)

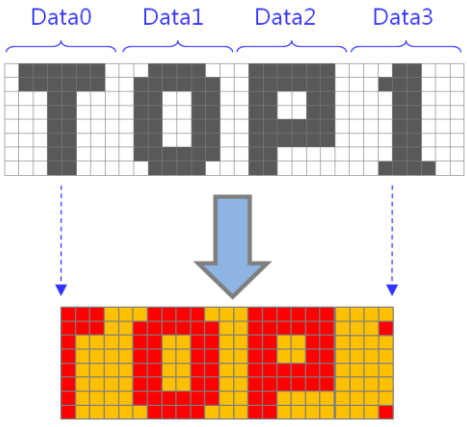

Figure 10-18: Example 2 of MCU Write with Color Expansion (ROP=3)

Note:

1. Sent Data Numbers per Row
   = [BitBLT Width + (MCU I/F bits – Start bit - 1)] / (MCU I/F bits) ,
   (Take integer with unconditional carry)
2. Total Data Number = (Sent Data Numbers per Row) * BitBLT Height

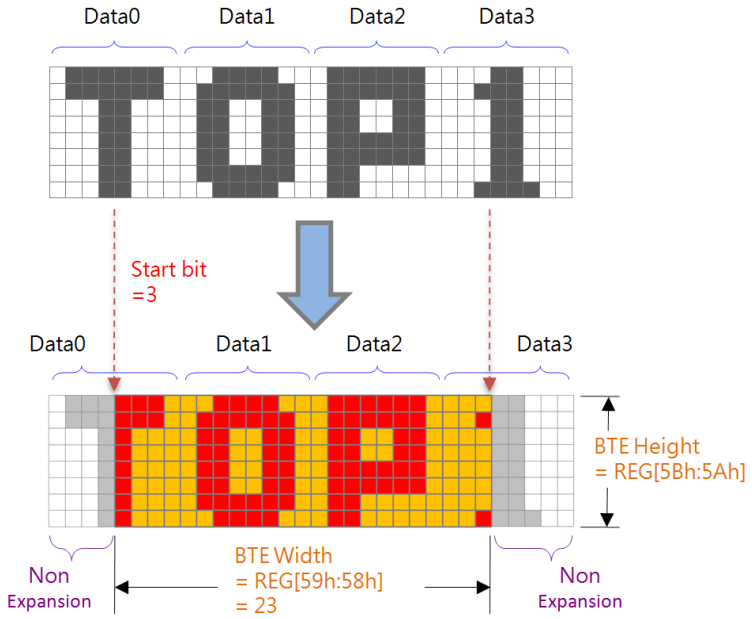

Figure 10-19: Data Format for Color Expansion

```
void LT768_BTE_MCU_Write_ColorExpansion_MCU_16bit
(
unsigned long Des_Addr,                 // Start address of DT at SDRAM
unsigned short Des_W,                   // Max. Width of DT image
unsigned short Xdes,                    // Upper-Left corner X-Coordinate of DT
unsigned short Ydes,                    // Upper-Left corner Y-Coordinate of DT
unsigned short X_W,                     // Width of DT window
unsigned short Y_H,                     // Height of DT window
unsigned long Foreground_color,         // Foreground color
unsigned long Background_color,         // Background color
const unsigned short *data              // 16-bit data
)
```

Example:

S0: MCU write a 100*100 image data (unsigned short Data[100*100])

DT: From the Address 1024*600*2 of Display RAM, at (200, 200)

Foreground Color: Red

Background Color: Blue

DT Display Window Size: 100*100

→

LT768_BTE_MCU_Write_ColorExpansion_MCU_16bit (1024*600*2, 1024, 200, 200, 100, 100, Red, Blue, &Data[0]);

10.2.8 MCU Write with Color Expansion and Chroma Key

This Operation is similar to previous section “MCU Write with Color Expansion”, but the difference is that background color will be ignored.

In the following example, the foreground color is set to Red. When MCU write monochrome data (Source 0) through BLT to specified memory area of Destination, the result is shown as Figure 10-20:

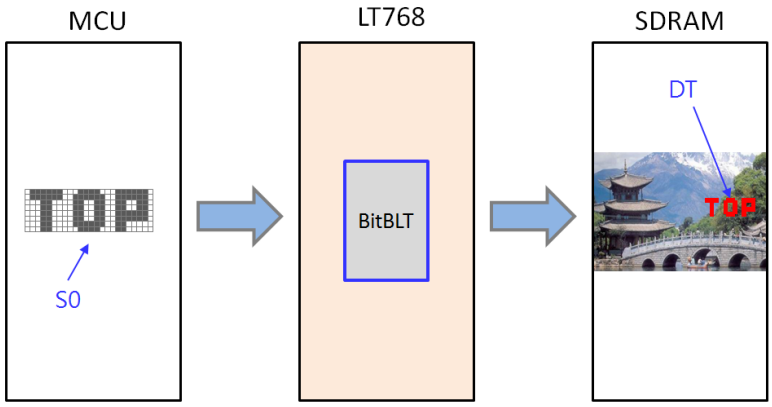

Figure 10-20: Example of MCU Write with Color Expansion and Chroma key

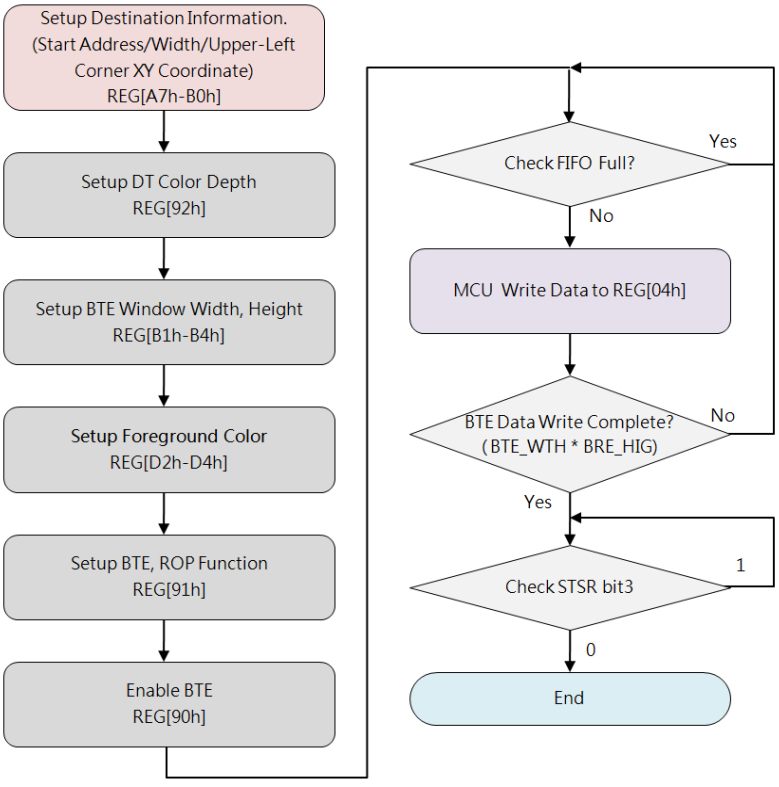

Figure 10-21: Flow Chart of MCU Write with Color Expansion and Chroma key

10.2.9 Memory Copy with Opacity

This operation performs blending S0 and S1 data and transferring the result to DT, two mode are available: Picture Mode and Pixel Mode.

Picture Mode is for 8 bpp/16bpp/24bpp format and it uses the same Opacity Value (Alpha Level) for whole bitmap picture. Opacity Value of Picture Mode is defined in REG[B5h].

Pixel Mode is for 8bpp/16bpp format and and it uses individual Opacity Value of S1, each pixel of S1 has its own Opacity Value, such as: for each 16bpp data, the bit[15:12] is Opacity Value, the bit[11:0] is color data; for each 8bpp data of S1, the bit[7:6] is Opacity Value, the bit[5:0] is the Index (Address) of Palette Color RAM pointing to the initialized 12-bits Color Depth data.

- **Picture Mode:**
  Destination Data
  = (Source 0 * alpha Level) + [Source 1 * (1- alpha Level)];
- **Pixel Mode 8bpp**:
  Destination Data
  = (Source 0 * alpha Level) + [Index palette (Source 1[5:0]) * (1 - alpha Level)]
- **Pixel Mode 16bpp**:
  Destination Data
  = (Source 0 * alpha Level) + [Source 1 [11:0] * (1 - alpha Level)]

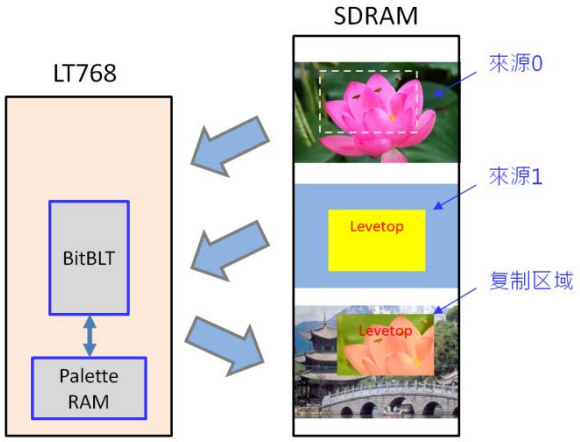

Figure 10-22: Example of Pixel Mode - 8bpp

Table 10-3: Alpha Blending of Pixel Mode - 8bpp

Bit[7:6]	Alpha Level
0h	0
1h	10/32
2h	21/32
3h	1

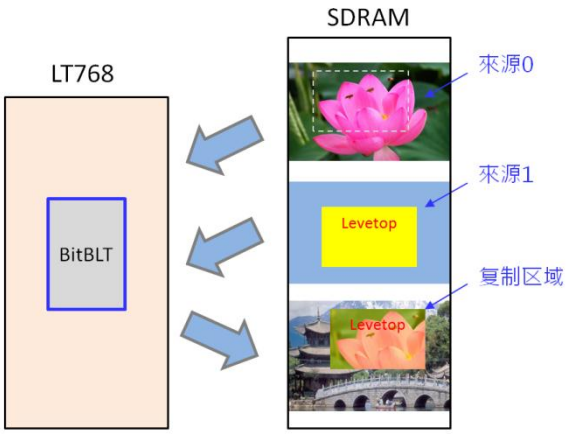

Figure 10-23: Example of Pixel Mode - 16bpp

Table 10-4: Alpha Level of Pixel Mode - 16bpp

Bit[15:12]	Alpha Level
0h	0
1h	2/32
2h	4/32
3h	6/32
4h	8/32
5h	10/32
6h	12/32
7h	14/32
8h	16/32
9h	18/32
Ah	20/32
Bh	22/32
Ch	24/32
Dh	26/32
Eh	28/32
Fh	1

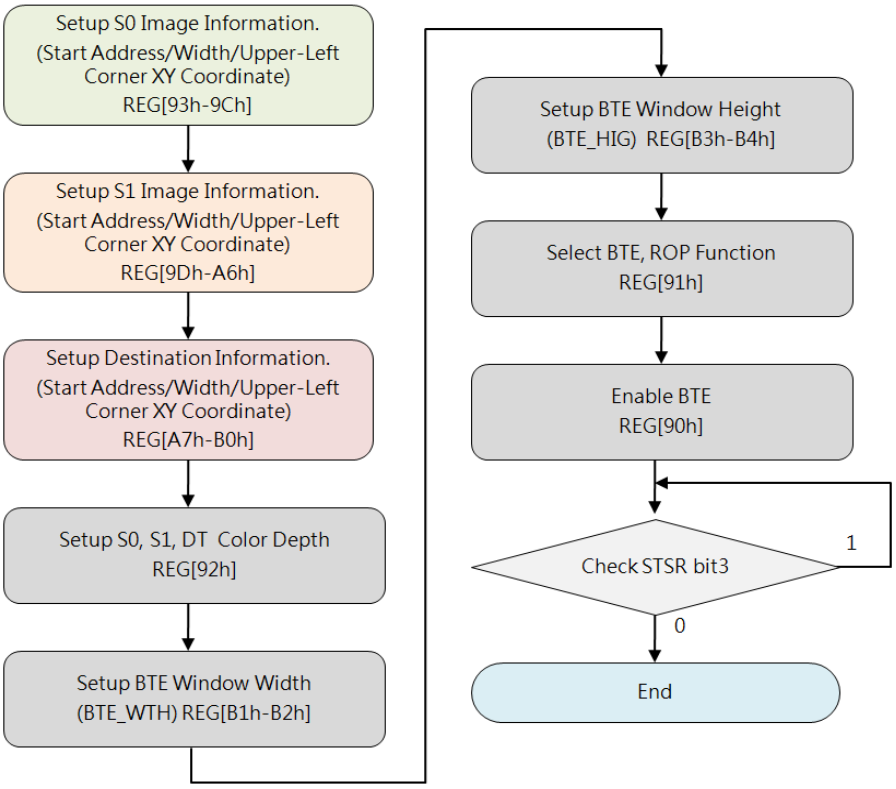

Figure 10-24: Flow Chart of Memory Copy with Opacity -Pixel Mode

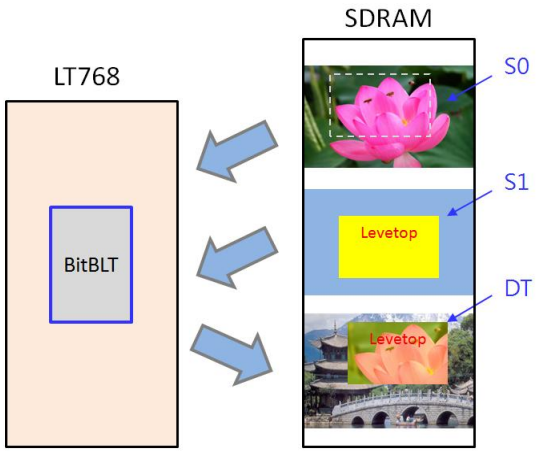

Figure 10-25: Example of Memory Copy with Opacity - Picture Mode

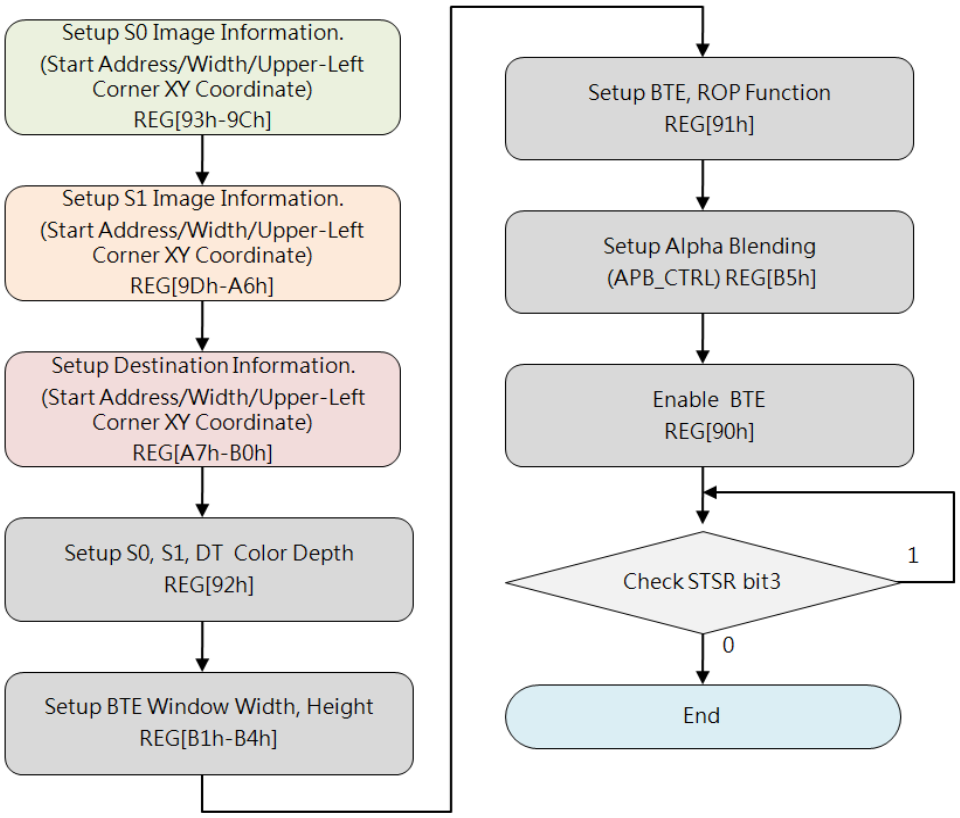

Figure 10-26: Flow Chart of Memory Copy with Opacity - Picture Mode

```
void BTE_Alpha_Blending
(
unsigned long S0_Addr,          // Start address of S0 at SDRAM
unsigned short S0_W,            // Image width of S0
unsigned short XS0,             // Upper-Left corner X-Coordinate of S0 image
unsigned short YS0,             // Upper-Left corner Y-Coordinate of S0 image
unsigned long S1_Addr,          // Start address of S1 at SDRAM
unsigned short S1_W,            // Image width of S1
unsigned short XS1,             // Upper-Left corner X-Coordinate of S1 image
unsigned short YS1,             // Upper-Left corner Y-Coordinate of S1 image
unsigned long Des_Addr,         // Start address of DT at SDRAM
unsigned short Des_W,           // Max. Width of DT image
unsigned short Xdes,            // Upper-Left corner X-Coordinate of DT
unsigned short Ydes,            // Upper-Left corner Y-Coordinate of DT
unsigned short X_W,             // Width of DT window
unsigned short Y_H,             // Height of DT window
unsigned char alpha             // Alpha Blending effect 0 ~ 32, Destination data =
                                // (Source 0 * (1- alpha)) + (Source 1 * alpha)
)
```

Example:

S0: Start address is 0, at (0, 0)
S1: Start address is 1024*600*2, at (0, 0)
DT: From the Address 1024*600*2*2 of Display RAM, at (50, 50)
Alpha Blending: 5 (0~31)
DT Display Window Size: 100*100

→

BTE_Alpha_Blending (0, 1024, 0, 0, 1024*600*2, 1024, 0, 0, 1024*600*4, 1024, 50, 50, 100, 100, 5);

10.2.10 Solid Fill

This function is to fill a rectangle area with a specified color. This function is used to fill a large area of SDRAM. The filled color is set in the BTE foreground color register.

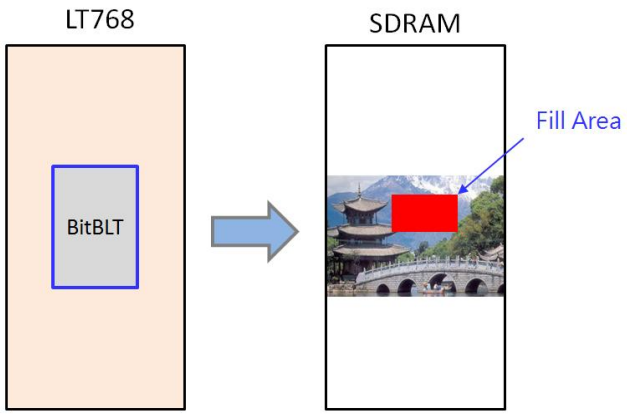

Figure 10-27: Example of Solid Fill

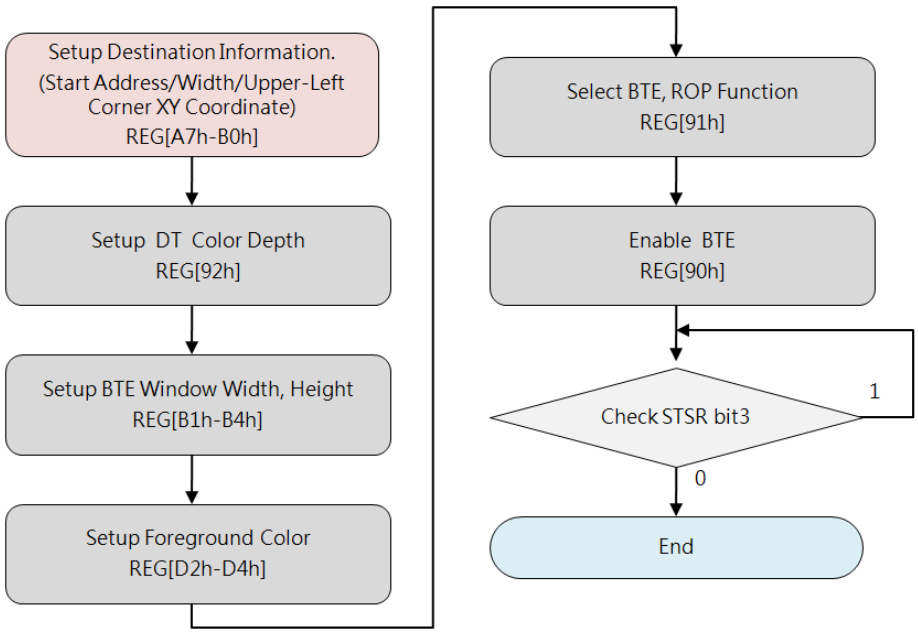

Figure 10-28: Flow Chart of Solid Fill

```
void BTE_Solid_Fill
(
unsigned long Des_Addr,          // Start address of DT at SDRAM
unsigned short Des_W,            // Max. Width of DT image
unsigned short XDes,             // Upper-Left corner X-Coordinate of DT
unsigned short YDes,             // Upper-Left corner Y-Coordinate of DT
unsigned short color,            // Fill color
unsigned short X_W,              // Width of DT window
unsigned short Y_H               // Height of DT window
)
```

Example:

DT: Start address is 0 of Display RAM, at (200, 200)
Fill Color: Red
DT Display Window Size: 100*100

→

```
BTE_Solid_Fill (0, 1024, 200, 200, Red, 100, 100);
```

11. Display Text

11.1 Use Embedded Character Font

The built-in fonts need to be initialized before they can be invoked to use the display function of the internal character font.

```
void LT768_Select_Internal_Font_Init
(
unsigned char Size,              // Setup Font Size 16: 8*16; 24: 12*24; 32: 16*32
unsigned char XxN,               // Width magnification: 1~4
unsigned char YxN,               // Height magnification: 1~4
unsigned char ChromaKey,         // 0: Background color transparent; 1: Background color
unsigned char Alignment          // 0: Non-alignment; 1: Alignment
)
```

```
void LT768_Print_Internal_Font_String
(
unsigned short x,                // Start position X
unsigned short y,                // Start position Y
unsigned long FontColor,         // Font color
unsigned long BackGroundColor,   // Background color (Invalid if Background color transparent)
char *c                          // Character string
)
```

Example: Use embedded 16*32 character font to show a Red string -“LeveTop LT768”at (10,100) position of the LCD panel. Settings: (1) no magnification; (2) with Background color transparent; (3) with font alignment.

```
LT768_Select_Internal_Font_Init(32, 1, 1, 0, 1);
LT768_Print_Internal_Font_String(10, 100, Red, 0, “LeveTop LT768”);
```

Note: The embedded characters do not support Chinese fonts. If you want to use Chinese fonts, you have to build them in the external SPI Flash.

11.2 Create Chinese Font Library

11.2.1 Get Font Library

To create a new font, it needs to use the corresponding font-taking software, and then generate bin format files. Levetop provides a software tool – LT_IMAGE_TOOL.EXE to help user create Chinese font. LT768x supports GB2312 Simplified Chinese, GB2312 Traditional Chinese, and BIG5 format Chinese font. Each Font Library supports five font sizes for 16*16, 24*24, 32*32, 48*48 and 72*72.

11.2.2 Save Bin File of Font

The Bin file of the font needs to be stored in the Flash Memory. The font needs to be read from the Flash and then be written to the SDRAM first. After setting the appropriate parameters, simply send the font code, then the corresponding font will be be displayed on the LCD screen.

11.2.3 Display Chinese Font (16*16、24*24、32*32)

The external font needs to be initialized before using. That is, the font needs to be read from the Flash and then be stored to the SDRAM, and then set the appropriate parameters.

```
void LT768_Select_Outside_Font_Init
(
unsigned char SCS,              // Select external SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,              // SPI Clock = System Clock /{(Clk+1)*2}
unsigned long FlashAddr,        // Flash's Source Address
unsigned long MemoryAddr,       // SDRAM's Destination Address
unsigned long Num,              // Data size of the Font Library
unsigned char Size,             // Font Size; 16: 16*16, 24: 24*24, 32: 32*32
unsigned char XxN,              // Width magnification of font: 1~4
unsigned char YxN,              // Height magnification of font: 1~4
unsigned char ChromaKey,        // 1: Background Color Transparent, 0: Background Color
                                // can be set
unsigned char Alignment         // 0: Font is not aligned, 1: Font aligned,
)
```

After initialization, call the following function to display Chinese String with GB2312 Format on the LCD screen.

```
void LT768_Print_Outside_Font_String
(
unsigned short x,                   // The x position at which the font starts to appear.
unsigned short y,                   // The y position at which the font starts to appear.
unsigned long FontColor,            // Color of Font
unsigned long BackGroundColor,      // Background Color (Note: When Background Color
                                    // Transparent then this value is not valid)
unsigned char *c                    // The first address of the Font_String buffer
)
```

If user want to display Chinese String with BIG5 Format, then call the following function:

```
void LT768_Print_Outside_Font_String_BIG5
(
unsigned short x,                   // The x position at which the font starts to appear.
unsigned short y,                   // The y position at which the font starts to appear.
unsigned long FontColor,            // Color of Font
unsigned long BackGroundColor,      // Background Color (Note: When Background Color
                                    // Transparent then this value is not valid)
unsigned char *c                    // The first address of the Font_String buffer
)
```

Example: From the 0x00 address of the plug-in Flash0 read 24*24 regular script Chinese font to SDRAM 0X003E537E0 first address. The amount of data in this font is 0x0009B520. This font is displayed in the position of the LCD screen (10, 100) with the Red "东莞市乐升电子有限公司-LT768". and does not enlarge the height and width of the font, the background color is transparent, the font is also aligned:

GB2312 Encoding Format:

```
LT768_Select_Outside_Font_Init(0, 0, 0x00, 0x003E537E0, 0x0009B520, 24, 1, 1, 1, 1);
LT768_Print_Outside_Font_String(10, 100, Red, 0, “东莞市乐升电子有限公司-LT768”);
```

BIG5 Encoding Format:

```
LT768_Select_Outside_Font_Init(0, 0, 0x00, 0x003E537E0, 0x0009B520, 24, 1, 1, 1, 1);
LT768_Print_Outside_Font_String_BIG5(10, 100, Red, 0, “东莞市乐升电子有限公司-LT768”);
```

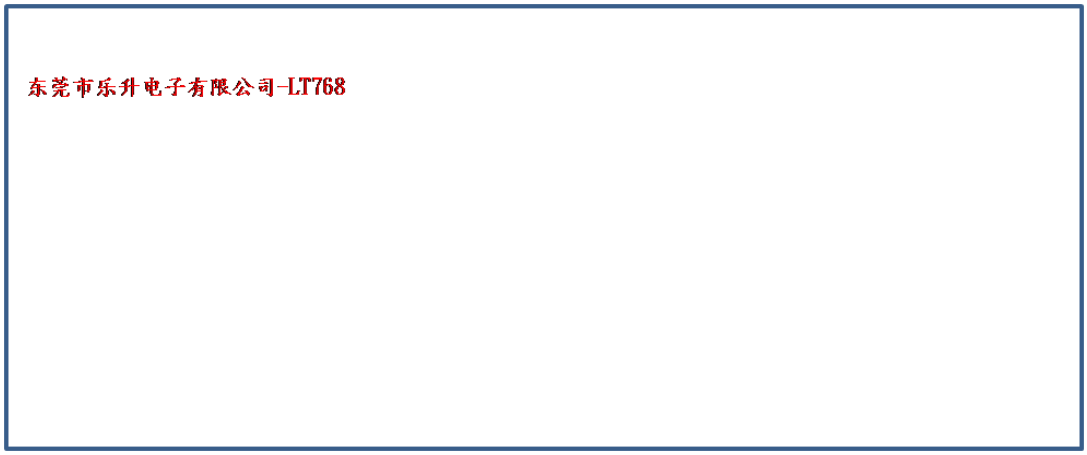

Figure 11-1: Display 24*24 Chinese String

Note:

1、This function is only valid for GB2312 or BIG5 Chinese encoding. If you want to use other Chinese coding, you need to modify the corresponding part of the function.

2、Some compilers may not compile in Chinese according to GB2312 or BIG5 encoding coding rules. Then the user will need to configure the compiler settings themselves.

11.2.4 Display Chinese Font (48*48、72*72)

If users want to show the big Chinese String of GB2312 encoding format as 48*48 or 72*72 on the screen, then call the following function:

```
void LT768_Print_Outside_Font_GB2312_48_72
(
unsigned char SCS,                  // Select external SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,                  // SPI Clock = System Clock /{(Clk+1)*2}
unsigned long FlashAddr,            // Flash's Source Address
unsigned long MemoryAddr,           // SDRAM's Destination Address
unsigned char Size,                 // Font Size; 48: 48*48; 72: 72*72
unsigned char ChromaKey,            // 1: Background Color Available,
                                    // 0: Background is Transparent
unsigned short x,                   // The X position at which the font starts to appear
unsigned short y,                   // The Y position at which the font starts to appear
unsigned long FontColor,            // Font Color
unsigned long BackGroundColor,      // Background Color (Note: When Background Color
                                    // Transparent then this value is not valid
unsigned short w,                   // Bold Level Setting: 1~3
unsigned short s,                   // Line Distance
unsigned char *c                    // The first address of the Font_String buffer
)
```

If users want to display Chinese String (48*48 or 72*72) with BIG5 Format, then call the following function:

```
void LT768_Print_Outside_Font_BIG5_48_72
(
unsigned char SCS,                  // Select external SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,                  // SPI Clock = System Clock /{(Clk+1)*2}
unsigned long FlashAddr,            // Flash's Source Address
unsigned long MemoryAddr,           // SDRAM's Destination Address
unsigned char Size,                 // Font Size; 48: 48*48; 72: 72*72
unsigned char ChromaKey,            // 1: Background Color Available,
                                    // 0: Background is Transparent
unsigned short x,                   // The X position at which the font starts to appear
unsigned short y,                   // The Y position at which the font starts to appear
unsigned long FontColor,            // Font Color
unsigned long BackGroundColor,      // Background Color (Note: When Background Color
                                    // Transparent then this value is not valid
unsigned short w,                   // Bold Level Setting: 1~3
unsigned short s,                   // Line Distance
unsigned char *c                    // The first address of the Font_String buffer
)
```

Example: From the 0x005DC000 address of the plug-in Flash0 read 48*48 regular script Chinese font to SDRAM 0X003E537E0 first address. This font is displayed in the position of the LCD screen (10, 100) with the Red "东莞市乐升电子有限公司-LT768". and does not enlarge the height and width of the font, the background color is transparent, the font is also aligned:

GB2312 Encoding Format:

```
LT768_Print_Outside_Font_GB2312_48_72(1, 0, 0x005DC000, 0x003E537E0, 48, 0, 10, 100,
color256_red, color256_white, 1, 0，“东莞市乐升电子有限公司-LT768”);
```

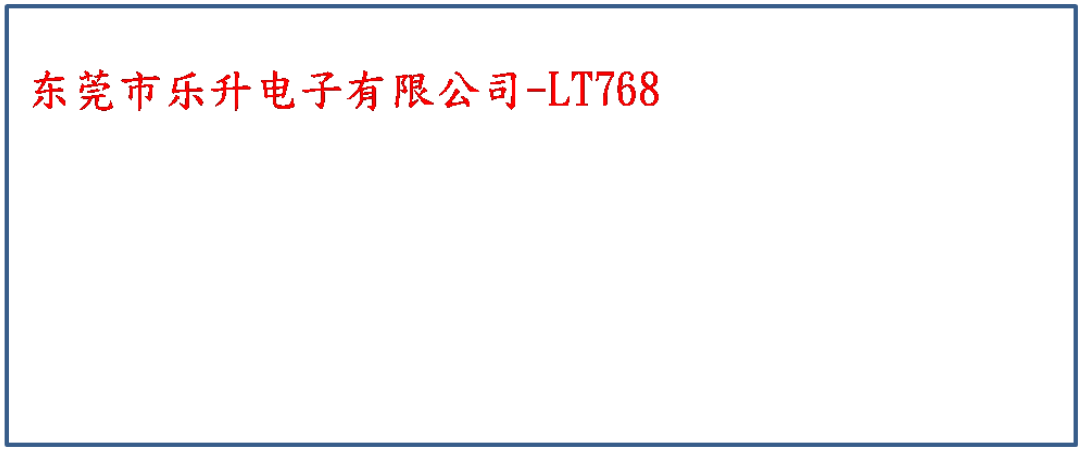

Figure 11-2: Display 48*48 Chinese String

BIG5 Encoding Format:

LT768_Print_Outside_Font_BIG5_48_72(1, 0, 0x005DC000, 0x003E537E0, 48, 0, 10, 100, color256_red, color256_white, 1, 0，“东莞市乐升电子有限公司-LT768”);

Note:

1、 This function is only valid for GB2312 or BIG5 Chinese encoding. If you want to use other Chinese coding, you need to modify the corresponding part of the function.

2、 Some compilers may not fllow GB2312 or BIG5 encoding coding rules. Users will need to configure the compiler settings themselves.

3、 In the use of large Chinese fonts, if you want to display English and punctuation, you need to enter full-width format.

4、 Large fonts (48*48, 72*72) must be applied with 256-bit color depth for font color and background color.

11.2.5 Line Spacing

This function is to set the line spacing of the previous row when the text is written to the last word to wrap. The size of the parameter “temp” is the number of pixels between the specified line spacing.

void Font_Line_Distance(unsigned char temp);

11.3 Create Bin File of Font

To use the Chines font, please download LT_IMAGE_TOOL.EXE from our website. This tool can convert the font information into Bin files, and then save the font data to built-in display memory of LT768x through DMA transfer method. After that, if you want to display Chinese on the TFT screen, the MCU only has to send GB code (2 bytes) to the dedicated registers. Therefore, it can improve the performance of Chinese display and ease the processing burden of MCU. To generate the font Bin file, users can refer to the user manual of LT_IMAGE_TOOL.EXE, or the instructions below. The following is an example to generate a 16*16 font Bin file:

1、 Execute LT_IMAGE_TOOL.EXE then click “font” to open the Chinese font Bin file production interface:

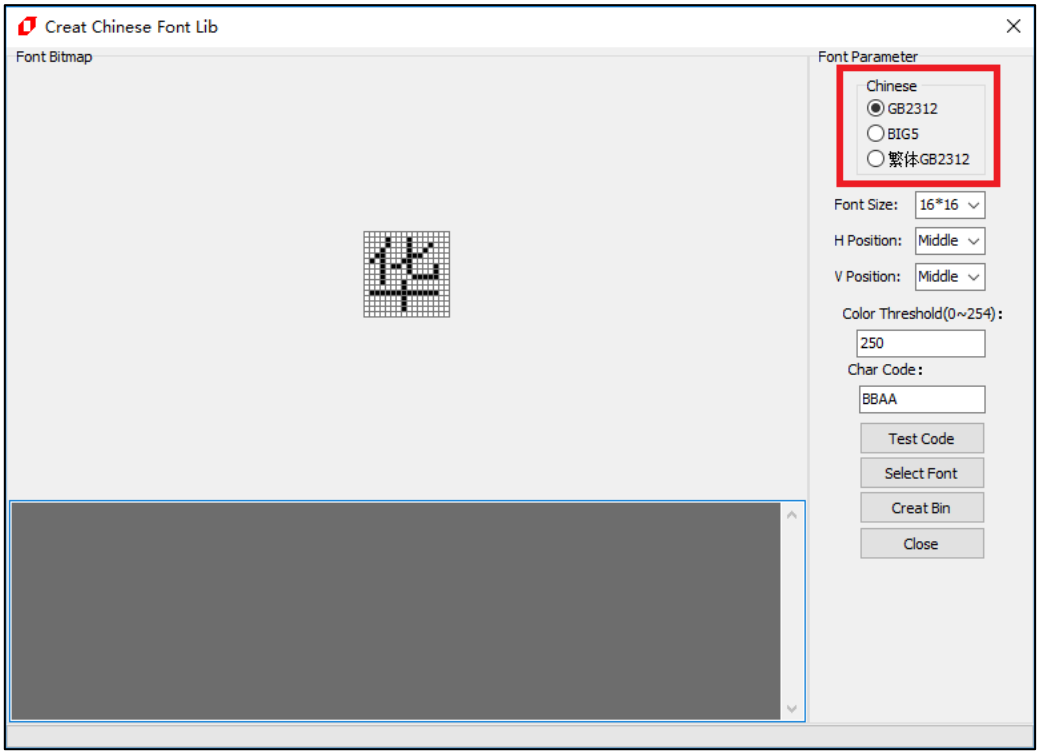

Figure 11-3: Create Chinese Font Library

2、Click "Select Font" button to set fonts, font style, size, and so on. When Setup is complete, press OK to save:

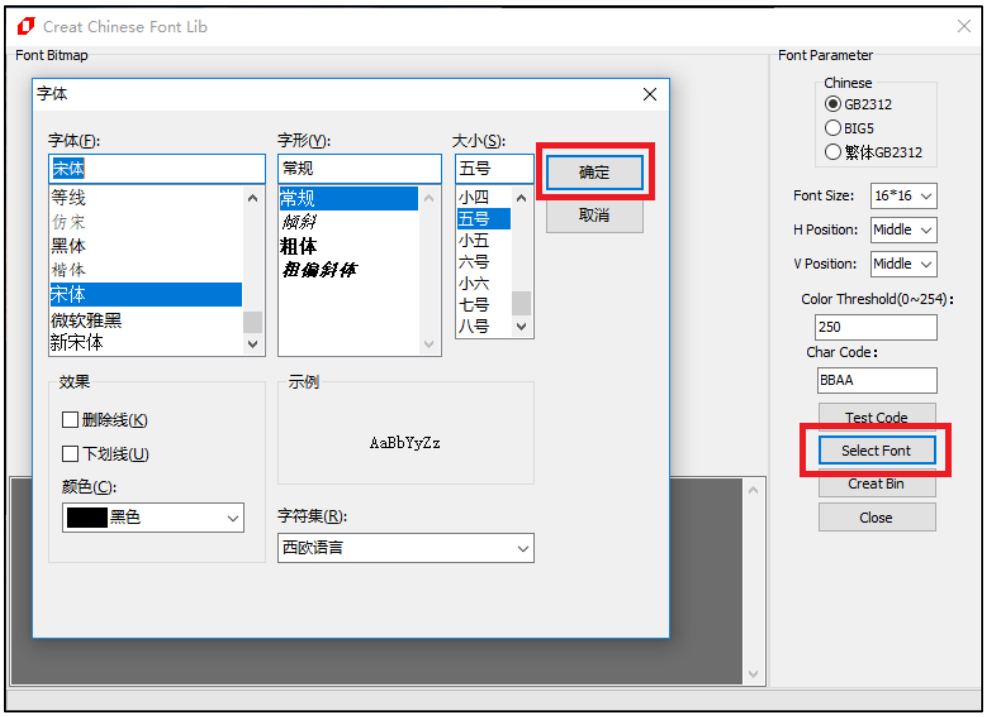

Figure 11-4: Select Font Style

3、Set the font size from the list of 16*16, 24*24, 32*32, 48*48, 72*72. You can also set the font horizontal position (Left, Middle, and Right), vertical position (Top, Middle, and Bottom), and Color threshold (0~254), and then preview the text. Click the "Test Code" button to view the character's code(GB2312).

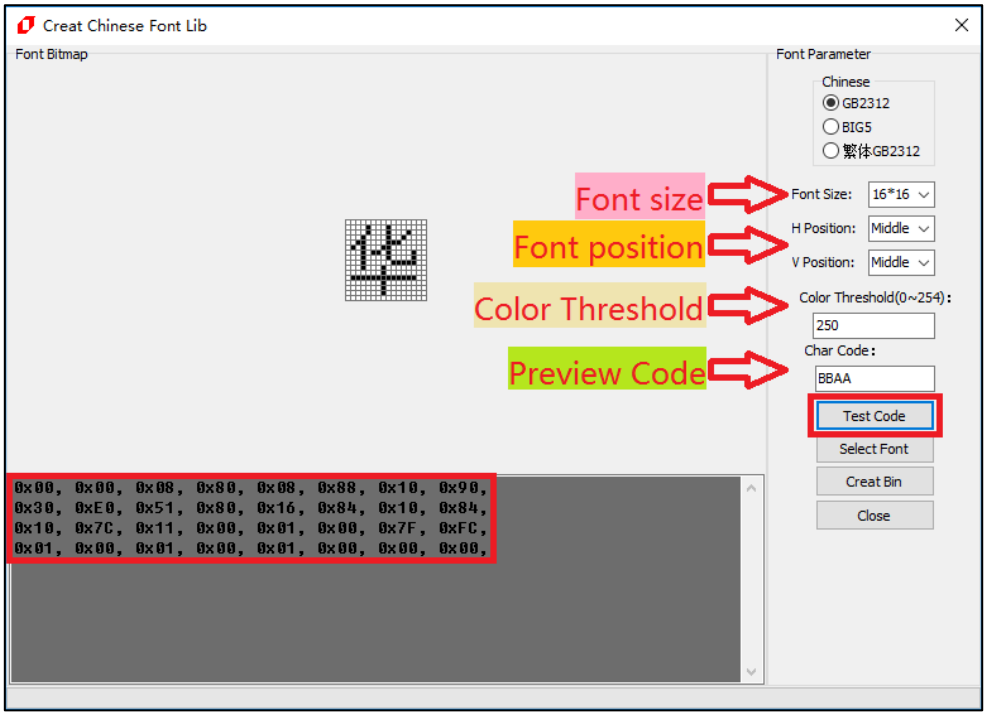

Figure 11-5: Setup Library

4、Click "Create Bin" to output the font Bin file. Please note when you enter a file name, the file name cannot contain special characters such as:? */* *: |, otherwise it cannot be saved.

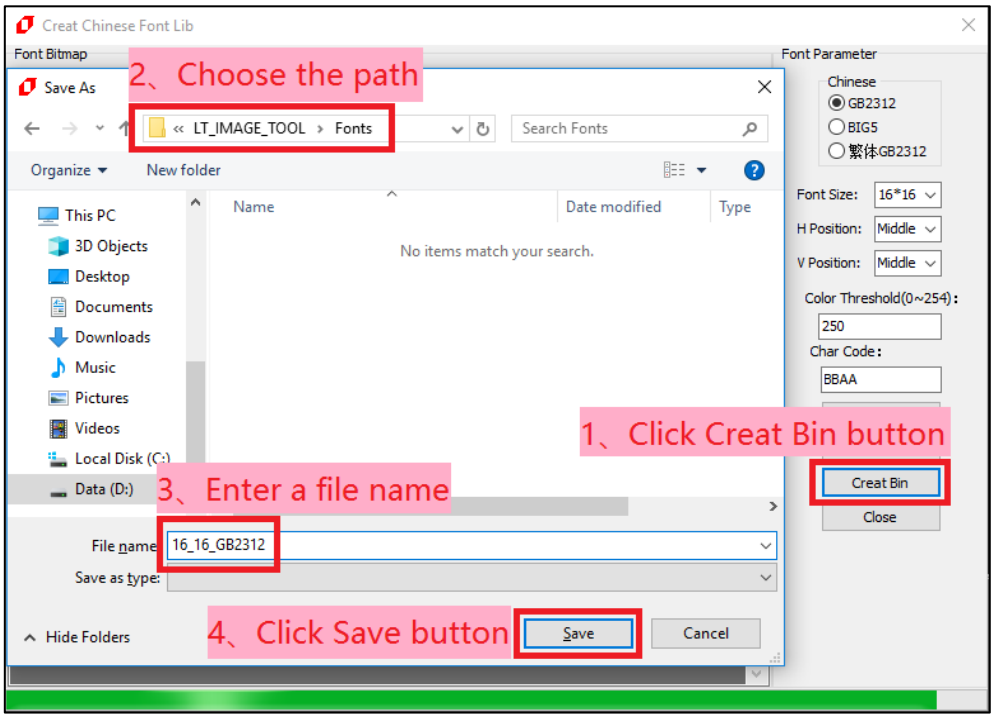

Figure 11-6: Save Font

If the file is saved successfully, a prompt message, “Font Lib ok”, will pop up.

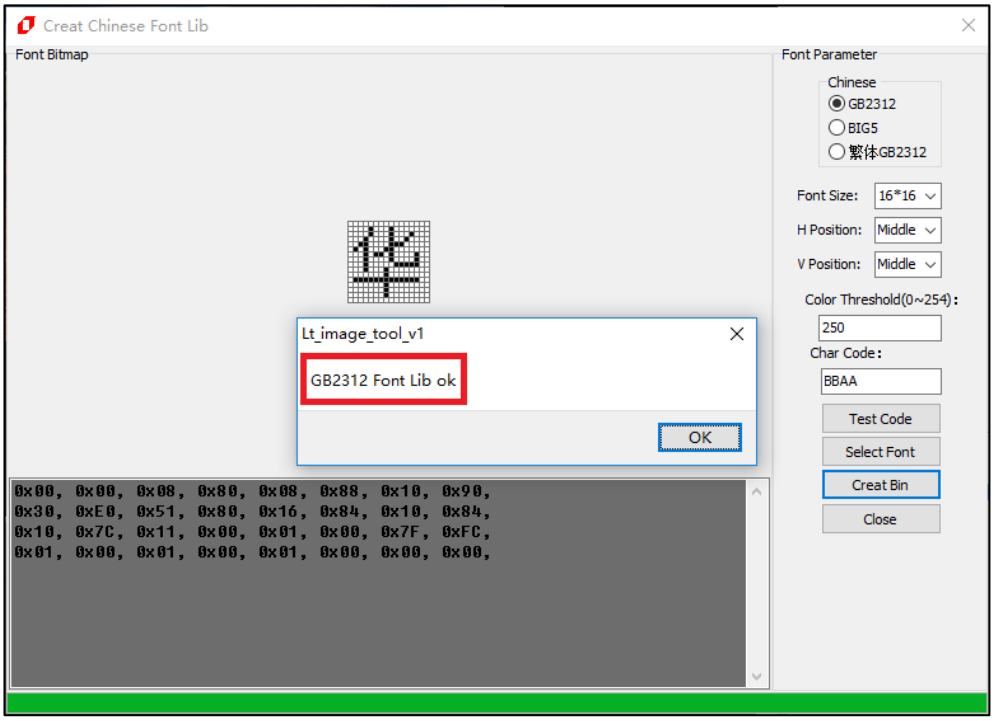

Figure 11-7: Complete the production of the font

5、 When font creatingis done, you can see the exported “16_16_GB2312.bin” in the destination folder:

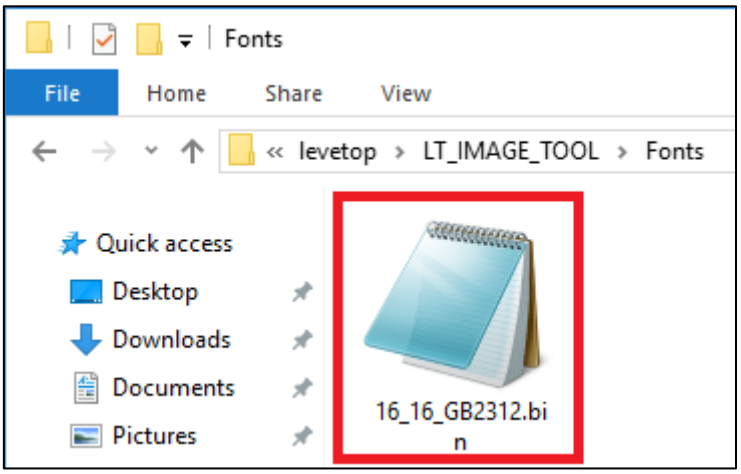

Figure 11-8: Generated Font Bin file

12. Display Cursor

LT768x supports 2 types of Cursor, Graphic Cursor and Text Cursor. This function allows users to display the cursor easily, enhance the display efficiency and ease the loading on MCU.

12.1 Display Text Cursor

A text cursor shows where the text will be written to. The width and height of the text cursor can be set, the move position can be set to automatically increase or not, and the cursor can be set to be flashing or not. When a text is written, the text cursor automatically move to the next text entry. The distance between each move is related to the text size and direction. When exceeding the edge of the Active Window, the cursor will move to the next line. Please note that the cursor automatic movement function is only effective within the Active Window. The row height size can be set in pixels. The following table lists the description of related registers.

Table 12-1: The Related Registers for Text Cursor

Register Address	Register Name	Description
REG[03h]	ICR	bit2: Graphic/Text Mode Selection (Text Mode Enable)
REG[3Ch]	GTCCR	bit1: Text Cursor Enable
		bit0: Text Cursor Blinking Enable)
REG[64h:63h]	F_CURX	X_Position: Text Input X coordinate
REG[66h:65h]	F_CURY	Y_Position: Text Input X coordinate
REG[D0h]	FLDR	Text Line Gap Setting

The height and width of the text cursor can be set through registers CURHS (REG[3Eh]) and CURVS (REG[3FH]).The height and width of the text cursor are also affected by whether the text is magnified (REG[CDH] bit[3:0]).

The cursor width can be set to 1 ~ 32 pixels under normal display. When using text magnify function, the width and height of the cursor will be magnified accordingly. The following figure is the horizontal and vertical setting of the text cursor:

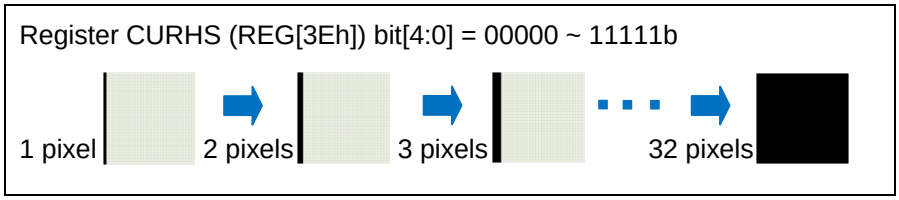

Figure 12-1: Example of Text Cursor Width Settings

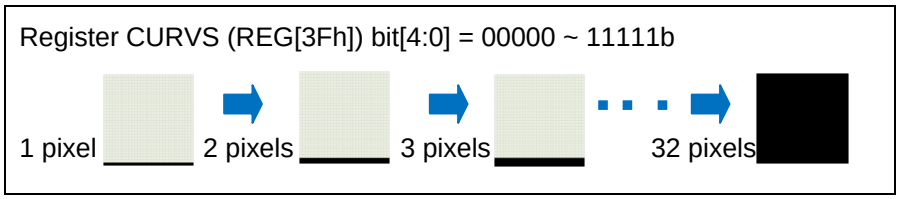

Figure 12-2: Example of Text Cursor Height Settings

The text cursor can be set to blink at a fixed frequency. GTCCR (REG[3Ch]) is used to set Blinking enabled (bit[0]=1) or disabled (bit[0]=0). The blinking interval can be calculated by the below formula:

Blink Time (sec) = BTCR[3Dh] * (1/Frame_Rate)

In the below example, the cursor stays behind the last written word and keeps blinking.

Figure 12-3: Example of Text Cursor Blinking

Before using the text cursor, it needs to be initialized. After the parameters are properly set, the cursor will then be displayed.

```
void LT768_Text_cursor_Init
(
unsigned char On_Off_Blinking,        // 0: Blink Disable;   1: Blink Enable
unsigned short Blinking_Time,         // Setup Blinking Time
unsigned short X_W,                   // Horizontal size of the text cursor (Max. 32)
unsigned short Y_W                    // Vertical size of the text cursor (Max. 32)
)

void LT768_Enable_Text_Cursor(void);  // Text Cursor Enable
void LT768_Disable_Text_Cursor(void); // Text Cursor Disable
```

Example: To set a blinking text cursor with a horizontal size of 10 and a vertical size of 2:

```
LT768_Text_cursor_Init(1, 15, 10, 2);
```

12.2 Display Graphic Cursor

LT768x Graphic Cursor is 32*32 pixel graphic, and the color of each pixel is represented by 2 bits data. The color data are defined as shown in Table 12-2:

Table 12-2: Graphic Color Definition

2'b00	Color-0 (defined by REG[44h])
2'b01	Color-1 (defined by REG[45h])
2'b10	Background Color
2'b11	Inversed Background Color

Therefore, 256bytes are required to build a graphics cursor. LT768X provides 4 graphic cursors that users can choose from. The MCU can select the cursor by setting the related registers. The graphic cursor position can be set by GCHP0 (REG[40h]), GCHP1 (REG[41h]), GCVP0 (REG[42h]) and GCVP1 (REG[43h]). The color of the Color-0 is set by the register REG[44h], and the Color-1 is set by the register REG[45h]. The following figure shows an example of the stored data format for the 32*32 graphic cursor. This assumes that the value of REG[44h] is set to Blue and the value of REG[45h] is set to Red.

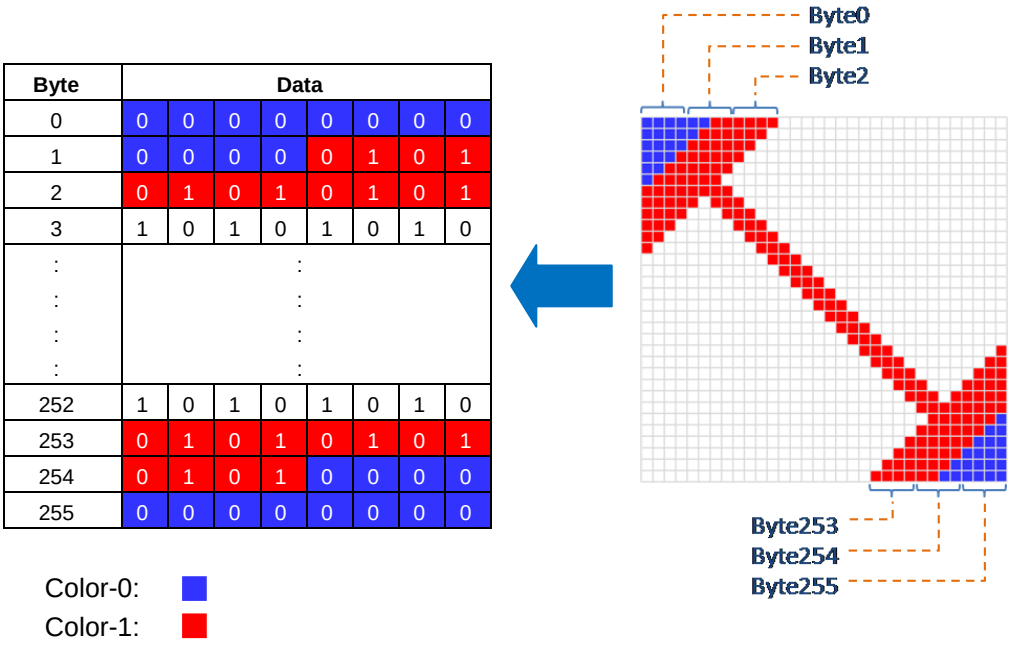

Figure 12-4: Graphic Cursor Data Format and Example

The graphic cursor needs to be customized by the user and then written to RAM. A total of 4 graphic cursors can be set, and then the cursors can be easily switched to another. The graphic cursors need to be initialized and set the appropriate parameters before using.

```
void LT768_Graphic_cursor_Init
(
unsigned char Cursor_N,          // Select Cursor; 1: Cursor1, 2: Cursor2, 3: Cursor3,
                                 // 4: Cursor4
unsigned char Color1,            // Color 1
unsigned char Color2,            // Color 2
unsigned short X_Pos,            // Coordinate X for Display
unsigned short Y_Pos,            // Coordinate Y for Display
unsigned char *Cursor_Buf        // The Start Address of Graphics Cursor Data
)
```

Toggle the position of the Graphic Cursor.

```
void LT768_Set_Graphic_cursor_Pos
(
unsigned char Cursor_N,          // Select Cursor; 1: Cursor1, 2: Cursor2, 3: Cursor3,
                                 // 4: Cursor4
unsigned short X_Pos,            // Coordinate X for Display
unsigned short Y_Pos             // Coordinate Y for Display
)
```

```
void LT768_Enable_Graphic_Cursor(void);      // Graphics Cursor Enable
void LT768_Disable_Graphic_Cursor(void);     // Graphics Cursor Disable
```

Example: Initialize 4 graphic cursors, then display the 4 graphic cursors in different locations in cycle.

```
extern const unsigned char gImage_pen_il[];
extern const unsigned char gImage_arrow_il[];
extern const unsigned char gImage_busy_im[];
extern const unsigned char gImage_no_im[];
```

Suppose that the above four arrays are the data for each graphic cursor.

```
LT768_Graphic_cursor_Init(1, 0xff, 0x00, 0, 0, (unsigned char*)gImage_pen_il);
LT768_Graphic_cursor_Init(2, 0xff, 0x00, 0, 0, (unsigned char*)gImage_arrow_il);
LT768_Graphic_cursor_Init(3, 0xff, 0x00, 0, 0, (unsigned char*)gImage_busy_im);
LT768_Graphic_cursor_Init(4, 0xff, 0x00, 0, 0, (unsigned char*)gImage_no_im);
for(i = 0 ; i < 4 ; i++)
{
   for(j = 0 ; j < 300 ; j++)
   {
      LT768_Set_Graphic_cursor_Pos(i, j, j);
   }
}
```

Example: Data for Arrows Cursor 

```
const unsigned char gImage_arrow_il[256] = { /* 0X00, 0X02, 0X20, 0X00, 0X20, 0X00,  */
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0X96, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X91, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XA4, 0X15, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X00, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XA9, 0X01, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X46, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0XA4, 0X05, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X01, 0X46, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X6A, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0XA4, 0X05, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X01, 0X46, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X69, 0XAA, 0XAA,
0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X14, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X44, 0X1A, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X11, 0X06, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X04, 0X41, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X10, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X44, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X11, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X04, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X95, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
} ;
```


12.3 Graphic Cursor Generation Tool

Levetop provides a dedicated tool, LT_IMAGE_TOOL.EXE, which can help generate graphic cursors. It allows users to draw a graphic cursor (32*32 pixels) on the PC side, and then export 256bytes cursor data. Users then copy the cursor data to the program that defines the cursor data. Users can refer to the manual of LT_IMAGE_TOOL.EXE or the following instructions:

12.3.1 Create Graphic Cursor

1、 Click “LT_IMAGE_TOOL menu > Cursor” to open the graphical cursor making interface:

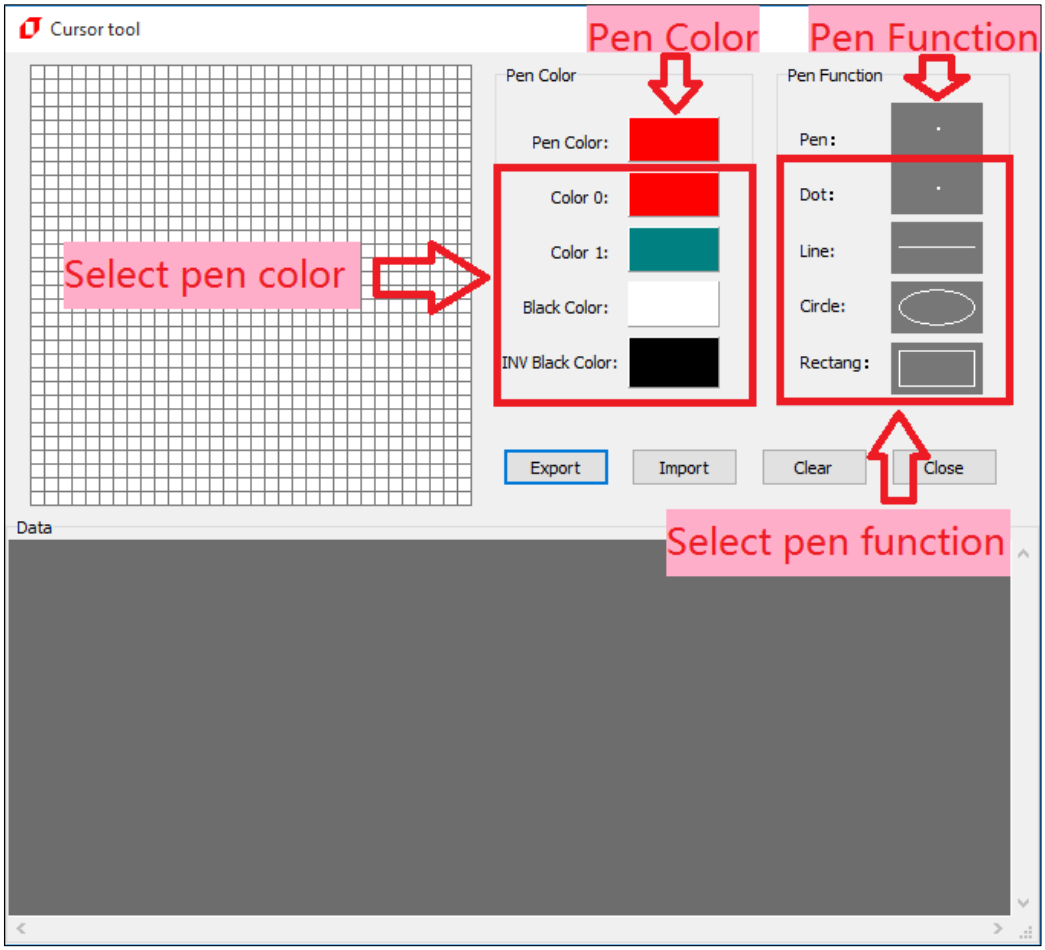

Figure 12-5: Open Cursor Tool

2、 After drawing the graphic cursor, click on the "Export" button, the graphic cursor data will be shown in the Data area.

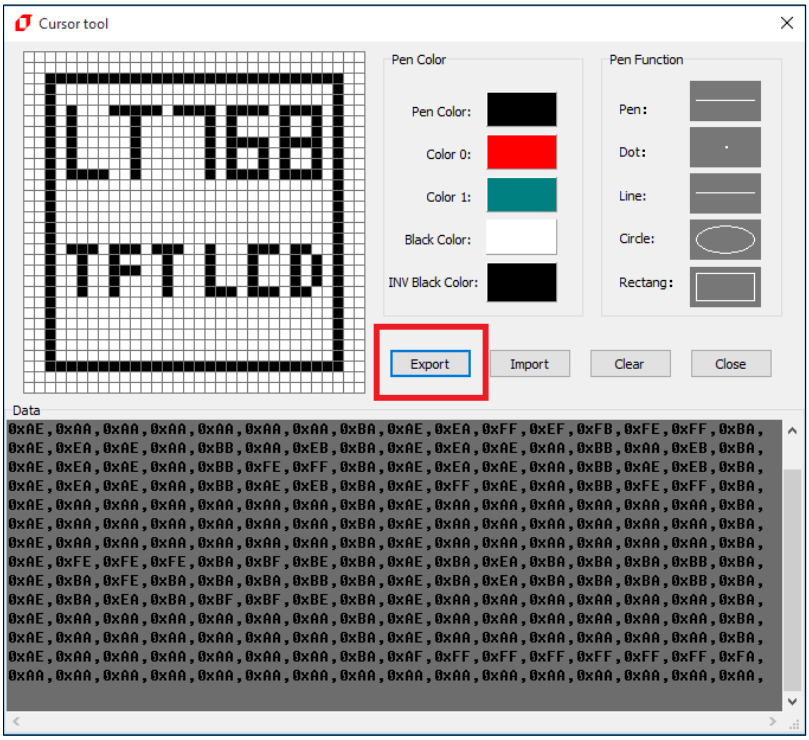

Figure 12-6: Generate Graphics Cursor Data

3、 Copy the Graphics Cursor Data:

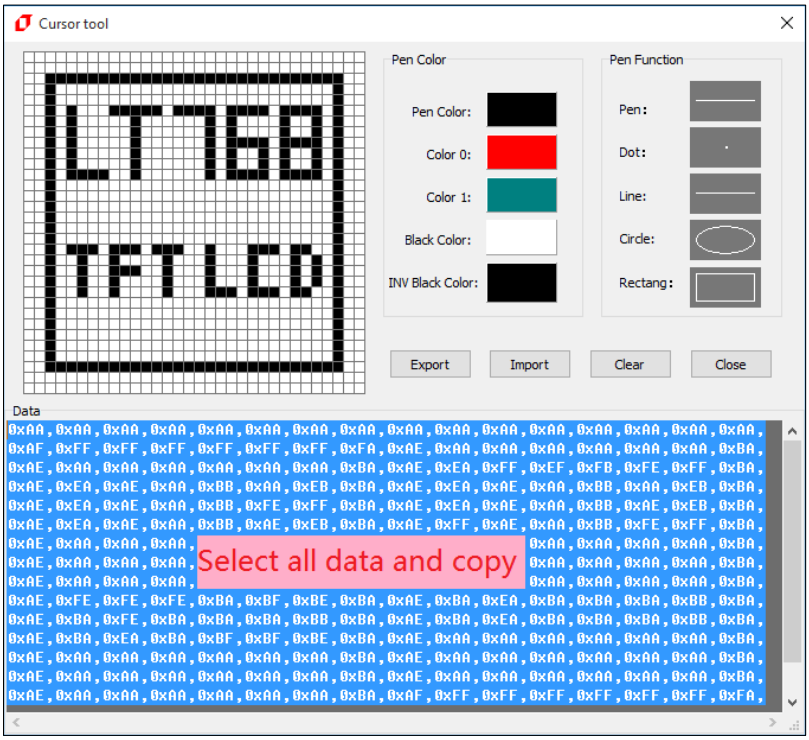

Figure 12-7: Copy the Graphics Cursor Data

4、 Paste the Graphic Cursor Data to program:

Image.h

```
#ifndef _Image_h
#define _Image_h

const unsigned char LT768_LCD[256] = {
0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,
0xAF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFA,0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,
0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,0xAE,0xEA,0xFF,0xEF,0xFB,0xFE,0xFF,0xBA,
0xAE,0xEA,0xAE,0xAA,0xBB,0xAA,0xEB,0xBA,0xAE,0xEA,0xAE,0xAA,0xBB,0xAA,0xEB,0xBA,
0xAE,0xEA,0xAE,0xAA,0xBB,0xFE,0xFF,0xBA,0xAE,0xEA,0xAE,0xAA,0xBB,0xAE,0xEB,0xBA,
0xAE,0xEA,0xAE,0xAA,0xBB,0xAE,0xEB,0xBA,0xAE,0xFF,0xAE,0xAA,0xBB,0xFE,0xFF,0xBA,
0xAE,0xAA,                                                              0xAA,0xBA,
0xAE,0xAA,Paste data to program and it can be used                      0xAA,0xBA,
0xAE,0xAA,                                                              0xAA,0xBA,
0xAE,0xFE,0xFE,0xFE,0xBA,0xBF,0xBE,0xBA,0xAE,0xBA,0xEA,0xBA,0xBA,0xBA,0xBB,0xBA,
0xAE,0xBA,0xFE,0xBA,0xBA,0xBA,0xBB,0xBA,0xAE,0xBA,0xEA,0xBA,0xBA,0xBA,0xBB,0xBA,
0xAE,0xBA,0xEA,0xBA,0xBF,0xBF,0xBE,0xBA,0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,
0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,
0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,
0xAE,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xBA,0xAF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFA,
0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,
};
```

Figure 12-8: Paste the Graphics Cursor Data to Program

12.3.2 Import and Modify the Graphics Cursor

Users can also import Graphic Cursor Data to generate the image of Cursor, and then modify it to get updated Data.

1. Copy the Graphic Cursor Data

Pen - Notepad

File Edit Format View Help

Copy cursor data

```
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0X96, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X91, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XA4, 0X15, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X00, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XA9, 0X01, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X46, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0XA4, 0X05, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X01, 0X46, 0XAA, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X6A, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0XA4, 0X05, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X01, 0X46, 0XAA, 0XAA, 0XAA,
0XAA, 0XAA, 0XAA, 0X40, 0X51, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X14, 0X69, 0XAA, 0XAA,
0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X14, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X44, 0X1A, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X11, 0X06, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X04, 0X41, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X10, 0X6A, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X44, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X11, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0X90, 0X04, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XA4, 0X01, 0X1A, 0XAA, 0XAA, 0XAA, 0XAA, 0XA9, 0X00, 0X1A,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X40, 0X6A, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0X95, 0XAA,
0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA, 0XAA,
```

Figure 12-9: Copy the Graphics Cursor Data

2. Paste the Cursor date to “Data” area, then click “Import” button

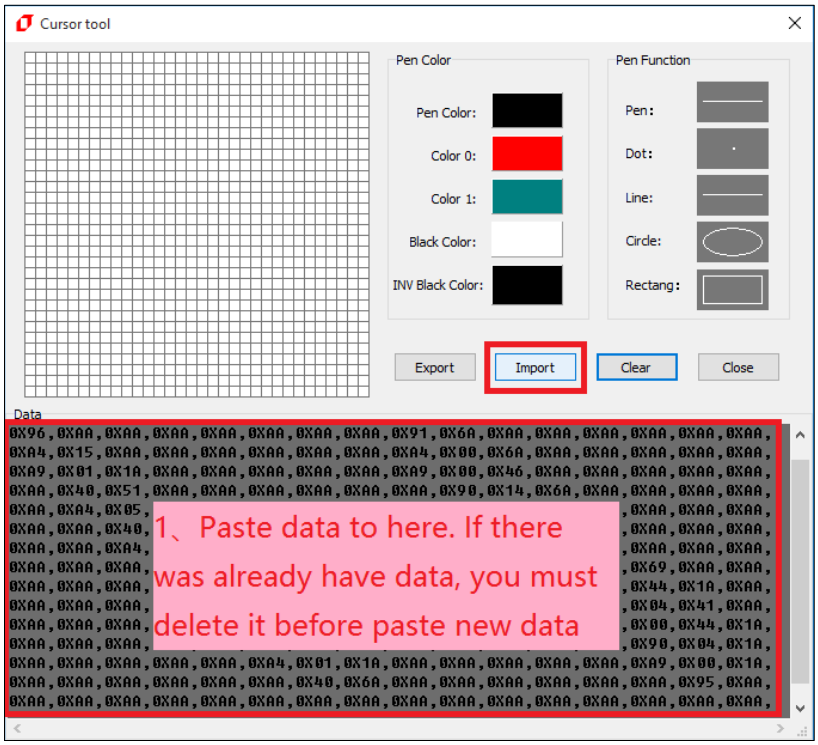

Figure 12-10: Paste the Cursor date to “Data” area

3. If the cursor image is shown, it means the import is successful. Users can modify the cursor image as the steps described above, and then export the cursor data after the modification is done.

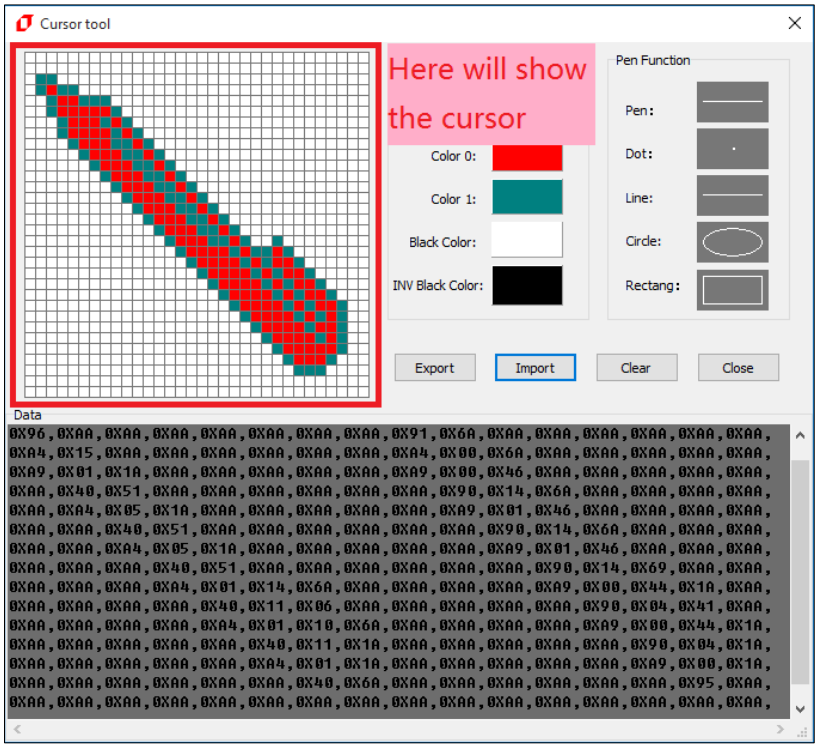

Figure 12-11: Import Success

13. PWM Setting

LT768x has built-in PWM functions, and provides two PWM output signals: PWM0 and PWM1. LT768x has embedded two 16bits counters, Timer-0 and Timer-1, and their actions are related to the output state of the PWM signals. For example, before using the PWM0, the Host must set Timer-0 count registers (TCNTB0, REG[8Ah-8Bh]) and Timer-0 count comparison registers (TCMPB0, REG[88h-89h]). After the PWM function is enabled, the Timer-0 counter will first load the TCNTB0 value and start counting down according to the PWM clock frequency. When the value of the Timer-0 counter is equal to the value of the TCMPB0 register, PWM will active. This means if the original state of PWM0 is 0, it will change to 1. The Timer-0 counter will continue to count down, and when Timer-0 equals to 0 then an interrupt will be issued. The PWM0 will be back to the original 0, and also automatically reload TCNTB0 value. The above procedure represents a complete PWM cycle.

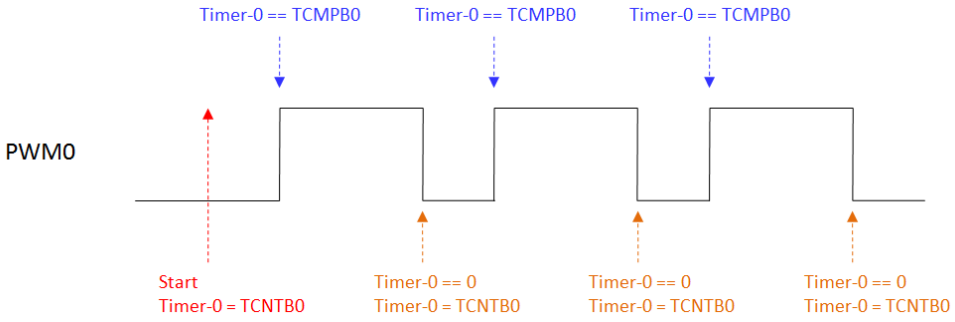

Figure 13-1: PWM0 Output Wave Form

Based on the above waveform and description, it is clear that the duty cycle of PWM0 is determined by comparison registers (TCMPB0, REG[88h-89h]). For example, when the PWM0 is initially set to 0, the larger the TCMPB0 value is, the larger equivalent voltage on PWM0 is. Conversely, when the PWM0 is initially set to 1, the smaller the TCMPB0 value is, the larger equivalent voltage on PWM0 is. Therefore, the backlight of TFT screen can be controlled by the approximate DC voltage generated by PWM. PWM can also be used to drive the buzzer to generate melody.

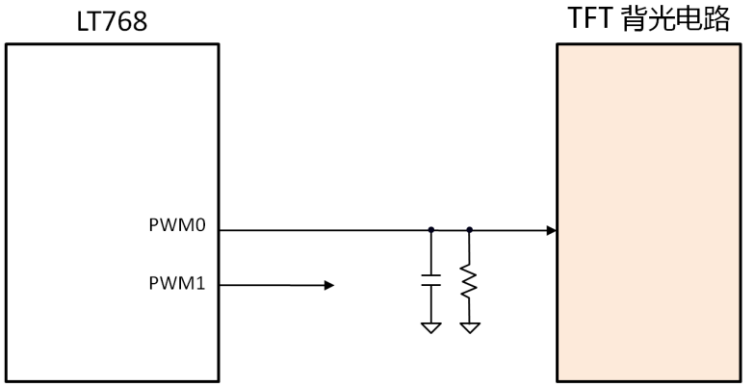

Figure 13-2: PWM Output Reference Circuit

You need to execute initialize function before using PWM.

```
void LT768_PWM0_Init
(
unsigned char on_off,                // 0: Disable PWM0; 1: Enable PWM0
unsigned char Clock_Divided,         // Divisor Range 0~3(1, 1/2, 1/4, 1/8)
unsigned char Prescaler,             // Prescaler is 1~256
unsigned short Count_Buffer,         // Setup the PWM output Cycle
unsigned short Compare_Buffer        // Setup the Duty
)

void LT768_PWM1_Init
(
unsigned char on_off,                // 0: Disable PWM1; 1: Enable PWM1
unsigned char Clock_Divided,         // Divisor Range 0~3(1, 1/2, 1/4, 1/8)
unsigned char Prescaler,             // Prescaler is 1~256
unsigned short Count_Buffer,         // Setup the PWM output Cycle
unsigned short Compare_Buffer        // Setup the Duty
)
```

After initialization, if users want to change the duty ratio, simply call the following corresponding function.

```
void LT768_PWM0_Duty(unsigned short Compare_Buffer);
void LT768_PWM1_Duty(unsigned short Compare_Buffer);
```

Example: Assume the system clock is 100Mhz, use PWM1 to output the 50Khz PWM waveform with a duty ratio of 30%:

```
LT768_PWM1_Init(1, 1, 10, 200, 60);          // 0.3=60/200
```

Modify the duty cycle of PWM1 to 50%:

```
LT768_PWM1_Duty(100);                // 0.5=100/200
```

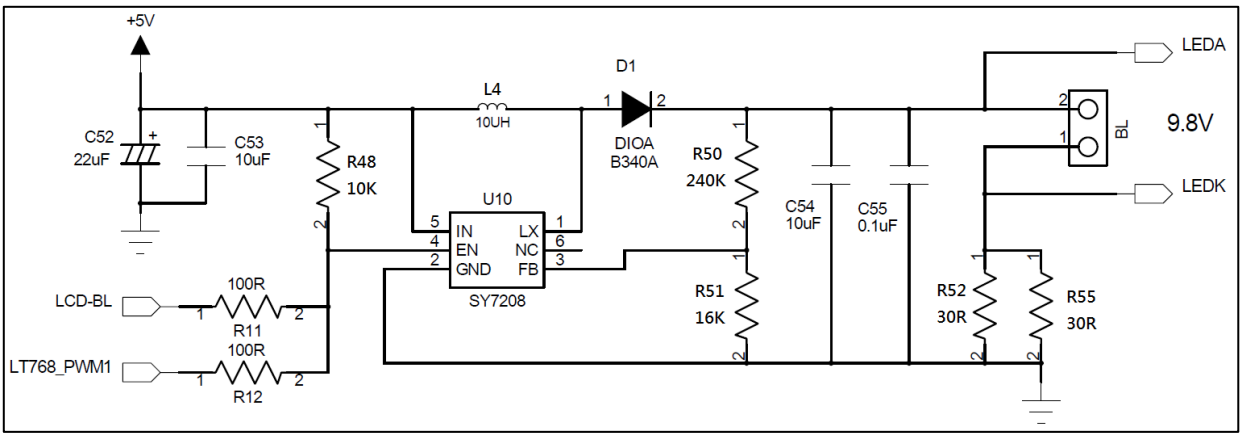

Figure 13-3: Reference Circuit of PWM Output to Control the Backlight (7” 1024*768 Panel)

14. Power-on Display

LT768x's Power-on Display circuit has embedded a small microprocessor unit. The main function is to quickly display the screen at boot time by executing the program code stored in Flash memory in the absence of Host, or when the Host is still in its start-up phase. To use this function, PWM[0] pin must connect to a 10K pull-up resistor, then the "Power-On Display" function will be enabled(Refer to Figure14-1). When Power-on Display is enabled, LT768x will automatically execute the program until the program code in the Flash memory is fully executed. After that, Host will retrieve the control of the system.

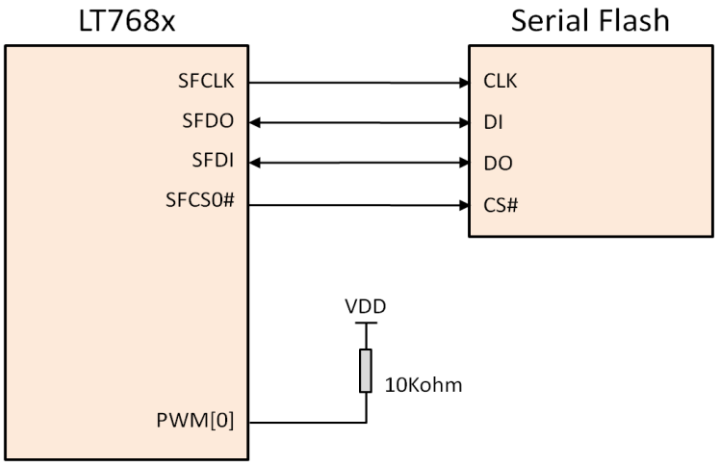

Figure 14-1: Enable the "Power-On Display" Function

The Power-on Display feature restricts that program code and display data must be stored in the same Flash memory. The Power-on Display unit supports 12 instructions as follows:

Table 14-1: 12 Power-on Display Instructions

No.	Instruction	Description	Instruction Length (Byte)
1	EXIT	Exit instruction (00h/FFh)	1
2	NOP	NOP instruction (AAh)	1
3	EN4B	Enter 4-Byte mode instruction (B7h)	1
4	EX4B	Exit 4-Byte mode instruction (E9h)	1
5	STSR	Status read instruction (10h)	2
6	CMDW	Command write instruction (11h)	2
7	DATR	Data read instruction (12h)	2
8	DATW	Data write instruction (13h)	2
9	REPT	Load repeat counter instruction (20h)	2
10	ATTR	Fetch Attribute instruction (30h)	2
11	JUMP	Jump instruction (80h)	5
12	DJNZ	Decrement & Jump instruction (81h)	5

After the Power-on Reset, the Power-on Display function will search the SPI Flash through the two SPI interfaces provided by LT768x. Within the SPI Flash, the first 8 bytes must be "61h, 72h, 77h, 63h, 77h, 62h, 78h, 67h". If the Flash memory is identified then the subsequent processing address will be (0008h), otherwise the master control will be transferred to the Host. During the execution of Power-on Display, if an “Exit” or undefined instruction is executed, the processor will hand over the control to the host. Here is an example of using an external serial Flash and show a 1024*768 picture on an 1024*768 resolution TFT panel:

```
// Addr: 'h0000
61   72   77   63   77   62   78   67       // ID

//Initial PLL
11   05   13   8A        // REG_WR('h05, 'h8A), Write 0x8A to REG[05]
11   06   13   41        // REG_WR('h06, 'h41)
11   07   13   8A        // REG_WR('h07, 'h8A)
11   08   13   64        // REG_WR('h08, 'h64)
11   09   13   8A        // REG_WR('h09, 'h8A)
11   0A   13   64        // REG_WR('h0A, 'h64)
11   00   13   80        // REG_WR('h00, 'h80)
11   01   13   82        // REG_WR('h01, 'h82)
11   01   12   82        // REG_WR('h01, 'h82)
11   02   13   40        // REG_WR('h02, 'h40)
AA   AA   AA   AA       // NOP

//Initial Display RAM
11   E0   13   29        // REG_WR('hE0, 'h29)
11   E1   13   03        // REG_WR('hE1, 'h03)
11   E2   13   0B        // REG_WR('hE2, 'h0B)
11   E3   13   03        // REG_WR('hE3, 'h03)
11   E4   13   01        // REG_WR('hE4, 'h01)
AA   AA   AA   AA       // NOP

//Setup LCD Panel
11   10   13   04        // REG_WR('h10, 'h04)
11   12   13   85        // REG_WR('h12, 'h85)
11   13   13   03        // REG_WR('h13, 'h03)
11   14   13   7F        // REG_WR('h14, 'h7F)
11   15   13   00        // REG_WR('h15, 'h00)
11   1A   13   FF        // REG_WR('h1A, 'hFF)
11   1B   13   02        // REG_WR('h1B, 'h02)

// Setup Main Window
11   20   13   00        // REG_WR('h20, 'h00)
11   21   13   00        // REG_WR('h21, 'h00)
11   22   13   00        // REG_WR('h22, 'h00)
11   23   13   00        // REG_WR('h23, 'h00)
11   24   13   00        // REG_WR('h24, 'h00)
11   25   13   04        // REG_WR('h25, 'h04)
11   26   13   00        // REG_WR('h26, 'h00)
11   27   13   00        // REG_WR('h27, 'h00)
11   28   13   00        // REG_WR('h28, 'h00)
11   29   13   00        // REG_WR('h29, 'h00)
AA   AA   AA   AA       // NOP

//Setup Canvas Window
11   50   13   00        // REG_WR('h50, 'h00)
11   51   13   00        // REG_WR('h51, 'h00)
11   52   13   00        // REG_WR('h52, 'h00)
11   53   13   00        // REG_WR('h53, 'h00)
11   54   13   00        // REG_WR('h54, 'h00)
```

```
11   55   13   04        // REG_WR('h55, 'h04)

//Setup Active Window
11   56   13   00        // REG_WR('h56, 'h00)
11   57   13   00        // REG_WR('h57, 'h00)
11   58   13   00        // REG_WR('h58, 'h00)
11   59   13   00        // REG_WR('h59, 'h00)
11   5A   13   00        // REG_WR('h5A, 'h00)
11   5B   13   04        // REG_WR('h5B, 'h04)
11   5C   13   00        // REG_WR('h5C, 'h00)
11   5D   13   03        // REG_WR('h5D, 'h03)
11   5E   13   02        // REG_WR('h5E, 'h02)

//Setup DMA Transfer Data from Flash to Display RAM
11   BC   13   00        // REG_WR('hBC, 'h00)
11   BD   13   02        // REG_WR('hBD, 'h02)
11   BE   13   00        // REG_WR('hBE, 'h00)
11   BF   13   00        // REG_WR('hBF, 'h00)
11   C0   13   00        // REG_WR('hC0,'h00)
11   C1   13   00        // REG_WR('hC1 ,'h00)
11   C2   13   00        // REG_WR('hC2, 'h00)
11   C3   13   00        // REG_WR('hC3, 'h00)
11   C6   13   00        // REG_WR('hC6, 'h00)
11   C7   13   04        // REG_WR('hC7, 'h04)
11   C8   13   00        // REG_WR('hC8, 'h00)
11   C9   13   03        // REG_WR('hC9, 'h03)
11   CA   13   00        // REG_WR('hCA, 'h00)
11   CB   13   04        // REG_WR('hCB, 'h04)
11   B7   13   C0        // REG_WR('hB7, 'hC0)
11   B6   13   01        // REG_WR('hB6, 'h01)
AA   AA   AA   AA        // NOP
11   B6   12   00        // REG_WR('hB6, 'h00)
11   12   13   40        // REG_WR('h12, 'h40)
00                       // Exit
```

Note: Power-on display unit requests program codes & display data, fonts, and required contents for program must be stored in the same serial Flash. If Host needs to switch to another serial Flash then all the codes & display data etc. will need to be stored to that serial Flash.

14.1 Setting The Power-on Boot Loader

Levetop provides a dedicated tool, LT_IMAGE_TOOL.EXE, which can help users set Power-on Boot Loader. This tool allows users to generate the program code of the power-on display on the PC side. Please refer to the user manual of LT_IMAGE_TOOL.EXE or the following:

1、Click “LT_IMAGE_TOOL Menu > Bootloader” to open the Boot Loader screen:

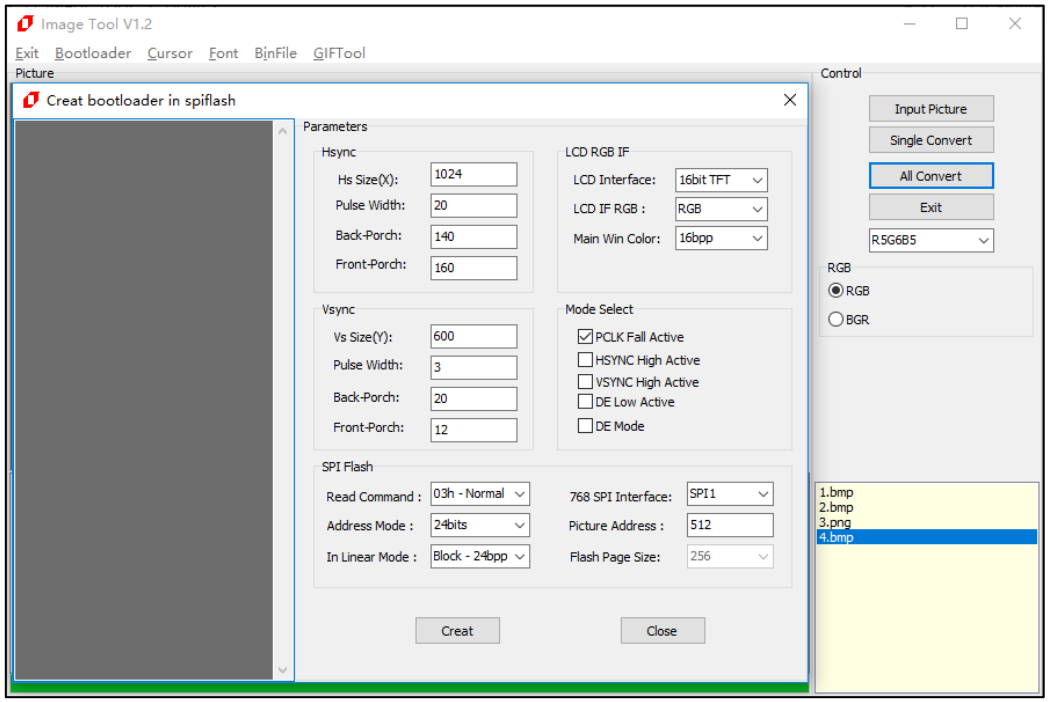

Figure 14-2: Setup the Bootloader

2、Set Panel Parameters: Hsync, Vsync and Mode Select. Set different parameters according to different LCD panel. This setting needs to be modified by reference to the TFT panel data:

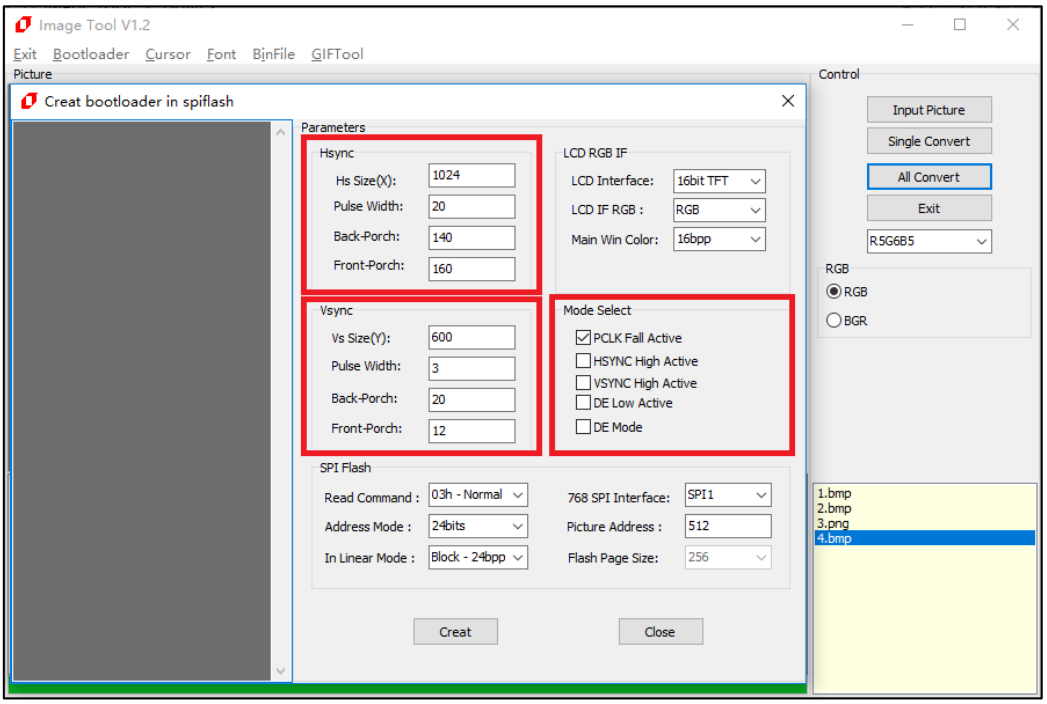

Figure 14-3: Setup Panel Parameters:

3、Setup LCD Interface

A. LCD Interface: Select 16、18、24bits RGB Interface
B. LCD IF RGB: Select RGB color Arrangement
C. Main Win Color: Select Color depth from the options of 24bpp(RGB 8:8:8), 16bpp(RGB 5:6:5) and 8bpp(RGB 3:3:2)

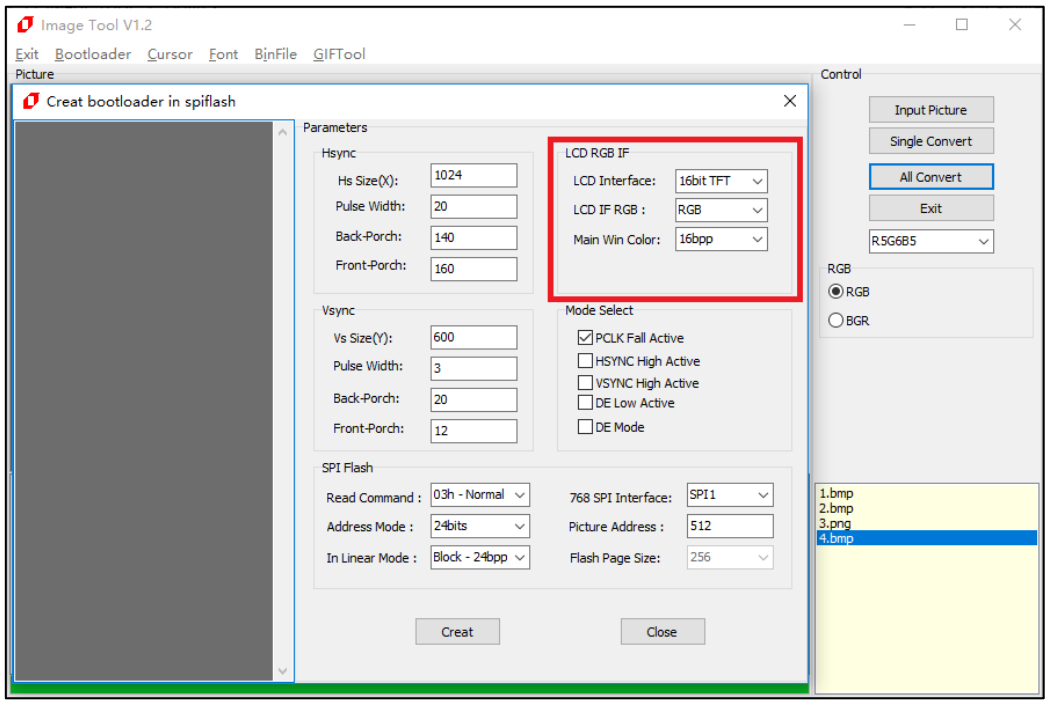

Figure 14-4: Setup LCD Interface

4、Setup SPI Flash Parameters

A. Read Command: Select SPI Flash Read Mode
B. LT768x SPI Interface: Select LT768x SPI I/F from the options of SPI0 and SPI1
C. Address Mode: Select Addressing Mode from the options of 24bits and 32bits
D. Picture Address: Setup the Flash Address of Boot Loader
E. In Liner Mode: Select the depth of Display RAM

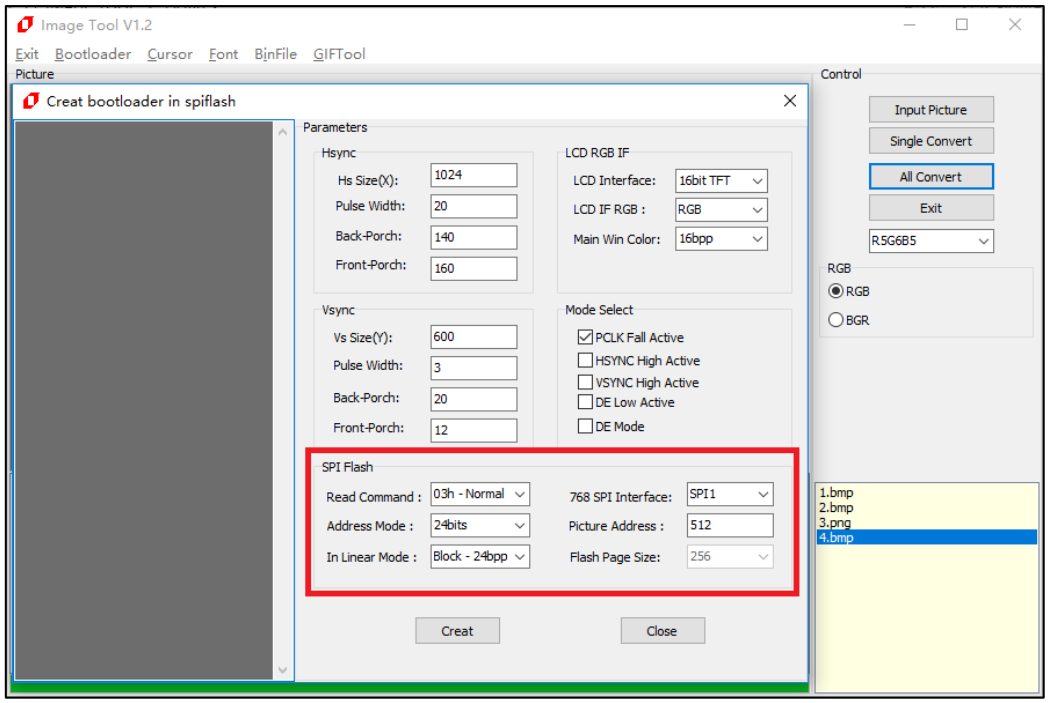

Figure 14-5: Setup SPI Flash Parameters

5、Click "Create" button to save the Bin file. Please note that the file name cannot contain special characters such as:? */* *: |, otherwise it cannot be saved.

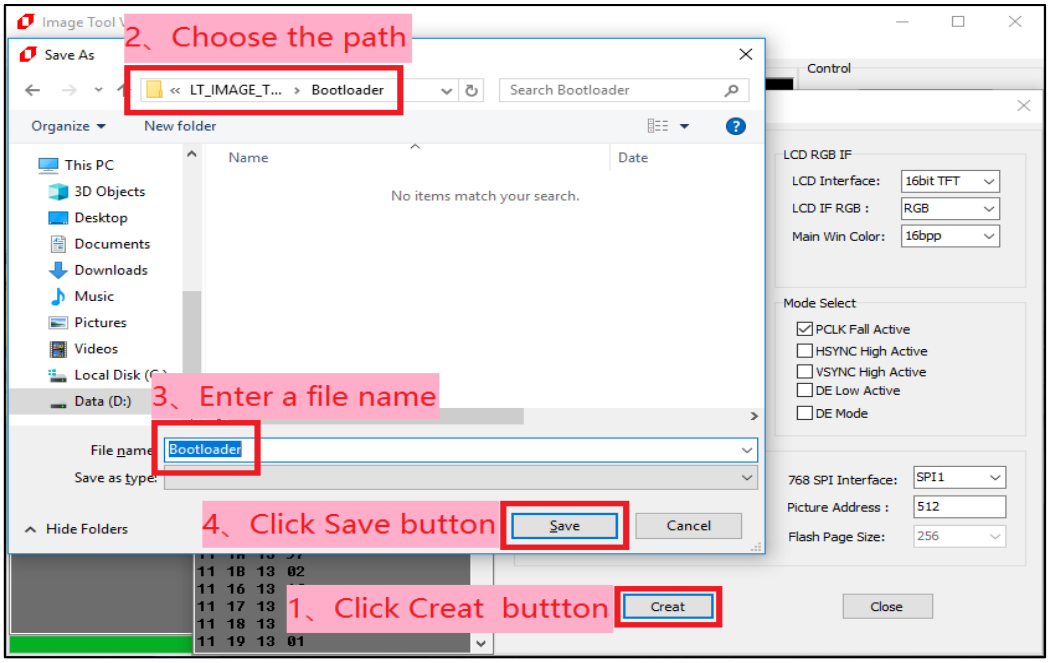

Figure 14-6: Save Bootloader Bin File

If the file is saved successfully, a prompt message, “OK”, will pop up.

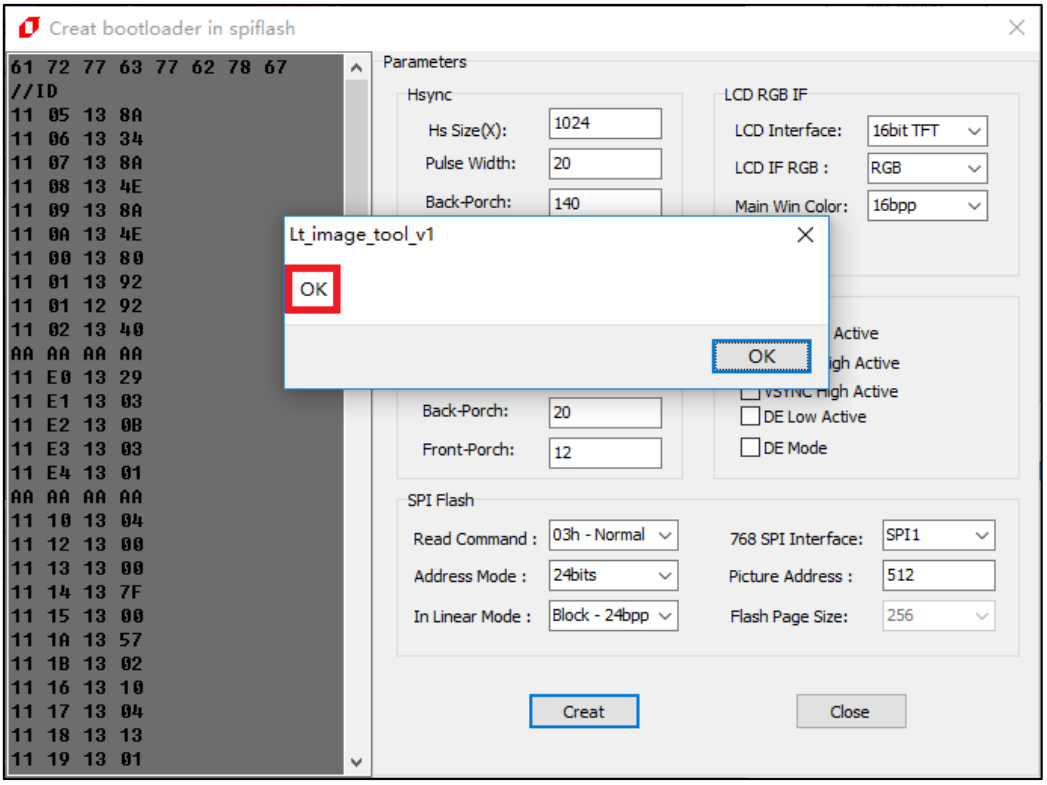

Figure 14-7: Save Success

6、When the bin file is created, the exported “Bootloader.bin” can be found in the destination folder:

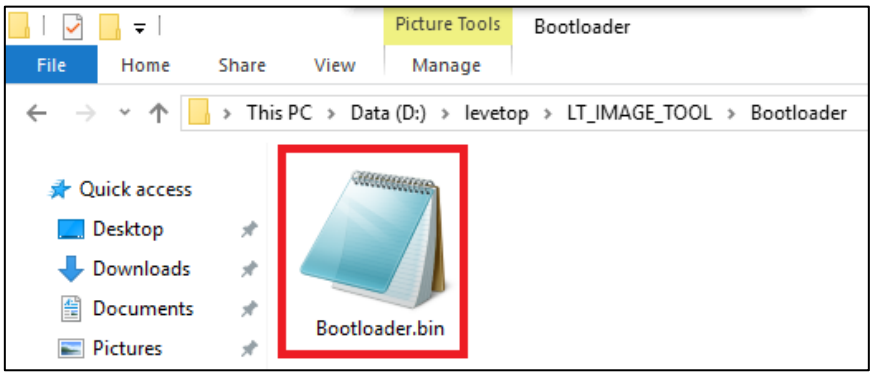

Figure 14-8: Export Bootloader.bin File

15. SPI Master

The SPI Master of LT768X is primarily used to control external SPI Flash or SPI components (such as touch chip). The SPI master supports various protocols such as 4-BUS (Normal Read), 5-BUS (FAST Read), Dual mode 0, and Dual mode 1 with Mode 0/Mode 3. Serial Flash/ROM function can be used for FONT mode and DMA mode. FONT mode means that the external serial Flash/ROM is treated as a source of character bitmap. DMA mode means that the external Flash/ROM is treated as the data source of DMA (Direct Memory Access). Through DMA mode, transferring data from Flash to display memory will be fast, and it does not need Host to process the data. For the applications of external SPI touch chip and I2C touch chip, please refer to chapter 16.

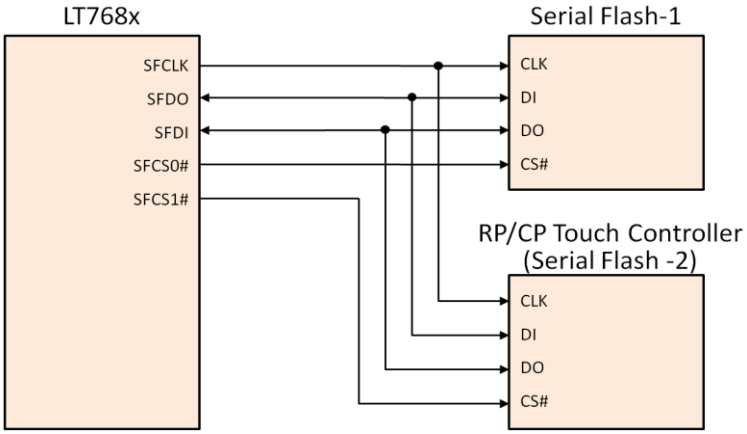

Figure 15-1: LT768x Serial Flash/ROM Application Circuit

15.1 DMA Transfer of SPI Flash

There are two kinds of DMA transmission modes in serial Flash: Linear transfer mode and block transfer mode. The purposes of the two modes are the same, which is to transfer the serial Flash data to the display memory. But they are different in the principle of transmission.

- **Linear Mode:** When the Chip Select signal CS is pulled low, the data will be transmitted. After the amount of data is transferred to the set value, CS will be pulled high.
- **Block Mode:** When Chip Select signal CS is pulled low, as long as each transmission data quantity reaches the amount of bytes of the set width multiplied by the set color depth, CS will be pulled high. After that, CS will be pulled low again to transfer data. This action will be repeated until the data are all transmitted. When the transmission is done, the CS will be pulled high.

There are two addressing modes for DMA transfer - 24bit and 32bit. Which addressing mode will be used depends on the operating sequence used for the serial Flash.

Example 1: The external serial Flash is the 128Mbit W25Q128 model (Winbond Serial NOR Flash).

Its reading data format is: When the CS is pulled low, the host sends 03h+24bit address + the amount of transmission data to Flash, and then CS will be pulled high. For such reading data format, users can use 24bit addressing transfer functions to read data from external serial Flash and transfer those data to SDRAM.

Example 2: The external serial Flash is the 1Gbit W25N01GVZEIG model (Winbond Serial NAND Flash).

This Flash reads data in pages mode, and there are two ways to read it. Therefore, users must set the sequential read mode, which is to send a command of the page address before sending the command to read the data. Therefore, this Flash requires one more step than W25Q128 to read the data. Users need to send a page command before calling the DMA function to transfer data. (See the Flash Data Manual for detail). If users need to use great amounts of Chinese font or pictures, this kind of high capacity NAND Flash may be taken into consideration.

Other than the explanation of the above examples, the DMA transfer operation also needs to cooperate with the operation timing sequence of the Flash. The access sequence diagram for SPI Flash memory can be referenced in the description of Section 10.3 of the LT768x specification.

15.1.1 DMA Transmission of Serial Flash in Linear Mode

1. 24bit Addressing Mode

This function supports 24bit addressing mode for the external Flash memory of LT768x. It can transfer data directly from the Flash to the embedded Display RAM. It is generally used to transfer the font data of the Flash to the Display RAM.

```
void LT768_DMA_24bit_Linear
(
unsigned char SCS,              // Select SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,              // SPI Clock = System Clock /{(Clk+1)*2}
unsigned long flash_addr,       // Start Address to Read Data from Flash
unsigned long memory_addr,      // Start Address of SDRAM which Data to be received
unsigned long data_num          // Amount of Data Transferred
)
```

Example: Assume a 24*24 format of Chinese regular script font is stored in the external Flash (use SPI-0), the start address is 0x0078DC20, and the total font data is 0x0009B520. These font data will be transferred to the embedded SDRAM of LT768x, and the first address of SDRAM is 0x003E537E0. Then the function is as following:

```
LT768_DMA_24bit_Linear(0, 0, 0x0078DC20, x003E537E0, 0x0009B520);
```


32bit Addressing Mode

This function supports 32bit addressing mode for the external Flash memory of LT768x. It can transfer data directly from the Flash to the embedded Display RAM. It is generally used to transferthe font data of the Flash to the Display RAM.

```
void LT768_DMA_32bit_Linear
(
unsigned char SCS,              // Select SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,              // SPI Clock = System Clock /{(Clk+1)*2}
unsigned long flash_addr,       // Start Address to Read Data from Flash
unsigned long memory_addr,      // Start Address of SDRAM which Data to be received
unsigned long data_num          // Amount of Data Transferred
)
```

Example: Assume a 24*24 format of Chinese regular script font is stored in the external Flash (use SPI-0), the start address is 0x0078DC20, and the total font data is 0x0009B520. These font data will be transferred to the embedded SDRAM of LT768x, and the first address of SDRAM is 0x003E537E0. Then the function is as following:

```
LT768_DMA_32bit_Linear(0, 0, 0x0078DC20, x003E537E0, 0x0009B520);
```

15.1.2 DMA Transmission of Serial Flash in Block Mode

2. 24bit Addressing Mode

This function supports 24bit addressing mode for the external Flash memory of LT768x. It can transfer data directly from the Flash to the embedded Display RAM. It is generally used to transfer the Picture data of the Flash to the Display RAM.

```
void LT768_DMA_24bit_Block
(
unsigned char SCS,        // Select SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,        // SPI Clock = System Clock /{(Clk+1)*2}
unsigned short X1,        // X1 Location of the SDRAM
unsigned short Y1,        // Y1 Location of the SDRAM
unsigned short X_W,       // The width of the DMA transfer data
unsigned short Y_H,       // The Height of the DMA transfer data
unsigned short P_W,       // the Width of Picture
unsigned long Addr        // Start Address of the Flash
)
```

Example: Assume there is a 1024*600, 16-bit color picture stored in the external Flash(use SPI-1), and the first address is at 0x80000. To display this picture(1024*600) on the TFT LCD screen, call the functions as following:

```
Select_Main_Window_16bpp();
Main_Image_Start_Address(0);
Main_Image_Width(1024);
Main_Window_Start_XY(0, 0);
Canvas_Image_Start_address(0);
Canvas_image_width(1024);
Active_Window_XY(0, 0);
Active_Window_WH(1024, 600);
LT768_DMA_24bit_Block(1, 0, 0, 0, 1024, 600, 1024, 0x80000);
```

3. 32bit Addressing Mode

This function supports 32bit addressing mode for the external Flash memory of LT768x. It can transfer data directly from the Flash to the embedded Display RAM. It is generally used to transfer the Picture data of the Flash to the Display RAM

```
void LT768_DMA_32bit_Block
(
unsigned char SCS,        // Select SPI Flash, 0: SPI-0; 1: SPI-1
unsigned char Clk,        // SPI Clock = System Clock /{(Clk+1)*2}
unsigned short X1,        // X1 Location of the SDRAM
unsigned short Y1,        // Y1 Location of the SDRAM
unsigned short X_W,       // The width of the DMA transfer data
unsigned short Y_H,       // The Height of the DMA transfer data
unsigned short P_W,       // the Width of Picture
unsigned long Addr        // Start Address of the Flash
)
```

Example: Assume there is a 1024*600, 16-bit color picture stored in the external Flash(use SPI-1), and the first address is at 0x80000. To display this picture(1024*600) on the TFT LCD screen, call the functions as following:

```
Select_Main_Window_16bpp();
Main_Image_Start_Address(0);
Main_Image_Width(1024);
Main_Window_Start_XY(0, 0);
Canvas_Image_Start_address(0);
Canvas_image_width(1024);
Active_Window_XY(0, 0);
Active_Window_WH(1024, 600);
LT768_DMA_32bit_Block(1, 0, 0, 0, 1024, 600, 1024, 0x80000);
```

15.2 Create Image Bin File for SPI Flash

The images that are commonly used on the application can be turned into bin files and saved in SPI Flash. Through LT768x DMA transfer mode, the bin file stored in the Flash can be transferred to the built-in display memory of LT768x. This can greatly ease the process loading on MCU. Levetop provides a dedicated tool, LT_IMAGE_TOOL.EXE, which has a function to make the bin file of pictures. It allows users to import pictures on the PC side, and then generate a bin file of the pictures. Users may refer to the manual of LT_IMAGE_TOOL.EXE for detail or the following explanation:

1、Execute the software tool: LT_IMAGE_TOOL.EXE

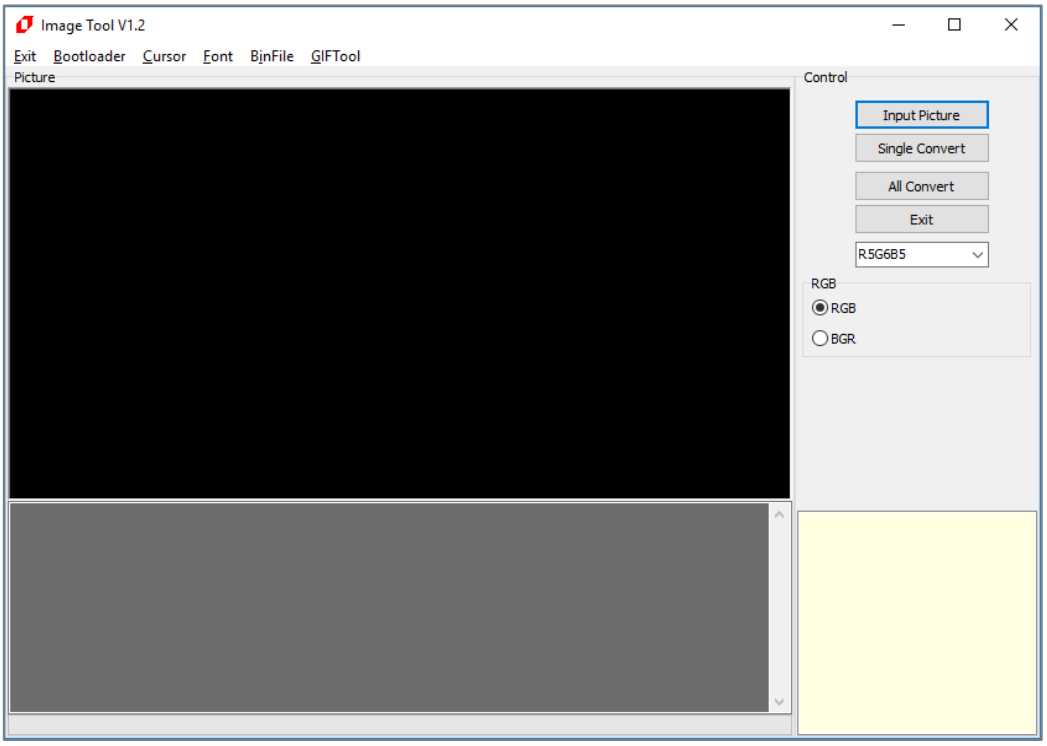

Figure 15-2: Execute LT_IMAGE_TOOL.EXE

2、Click “Input Picture” button to select picture, then click “Open”:

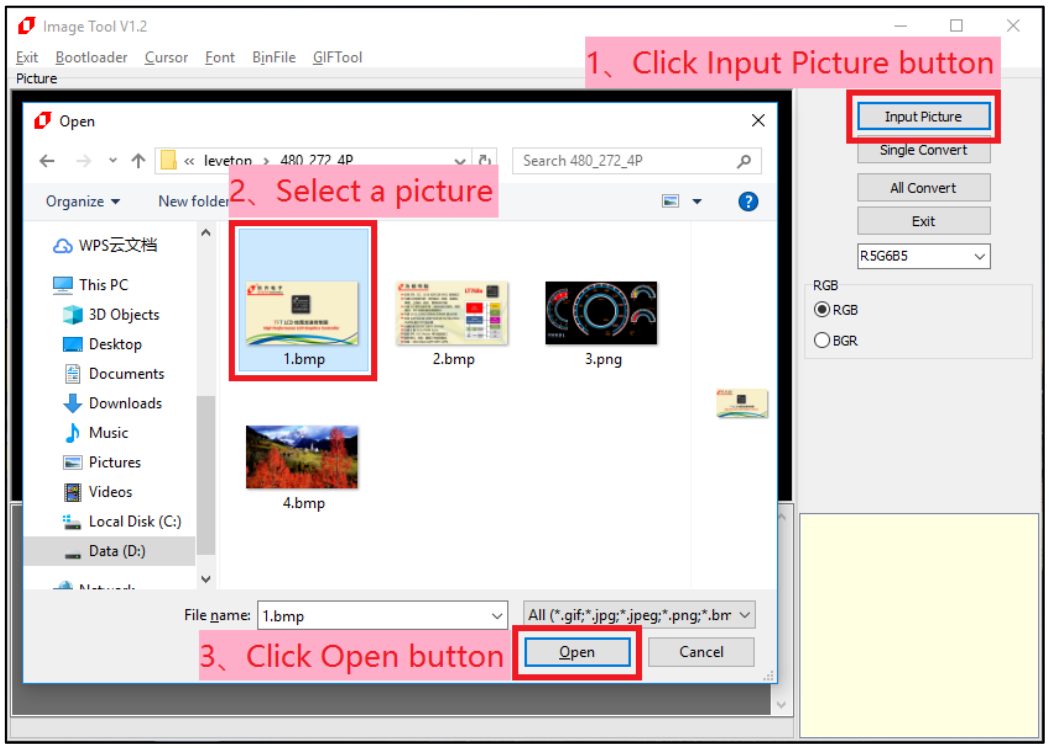

Figure 15-3: Import Picture

Figure 15-4: Import Picture Success

3、Setup the output format of picture: 16bpp or 24bpp color depth; and RGB or BGR format.

Figure 15-5: Setup the Picture Output Format

4、Export a picture or all the pictures. Please note the file name should not contain special characters such as:? */* *: |, otherwise the file will not be saved.

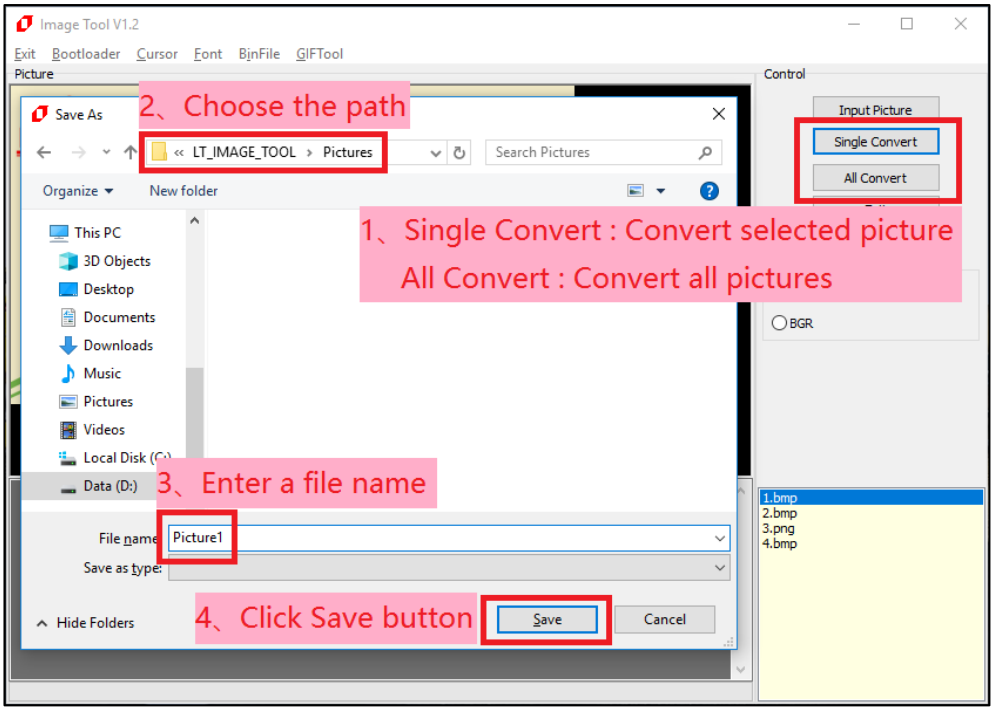

Figure 15-6: Export Picture (1/2)

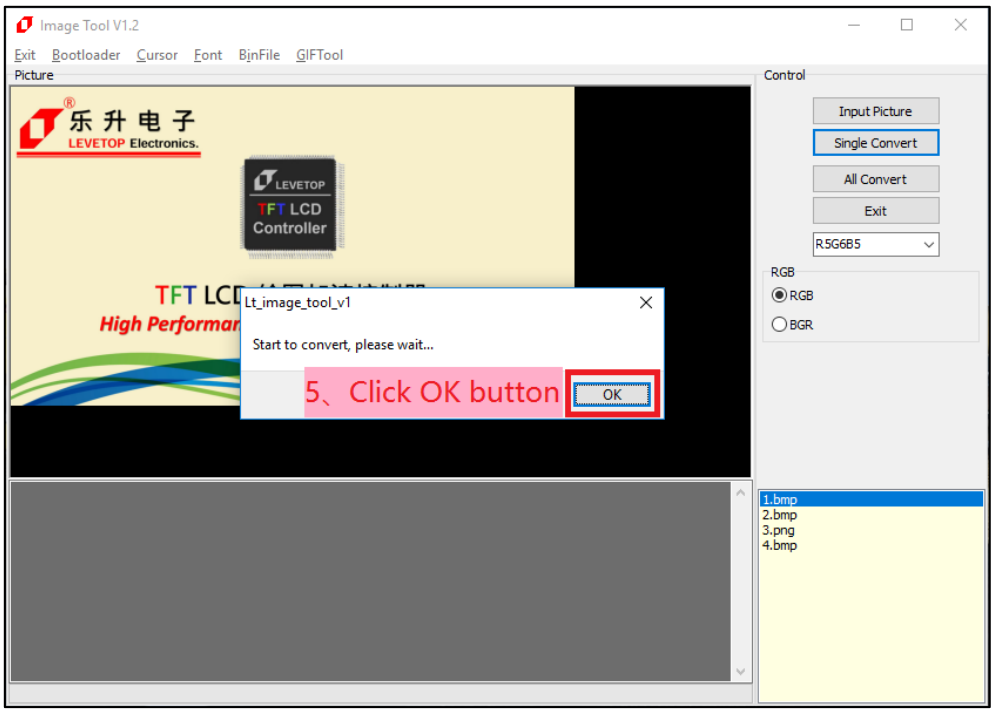

Figure 15-7: Export Picture (2/2)

5、 If the file is saved successfully, a prompt message, “Convert ok” will pop up.

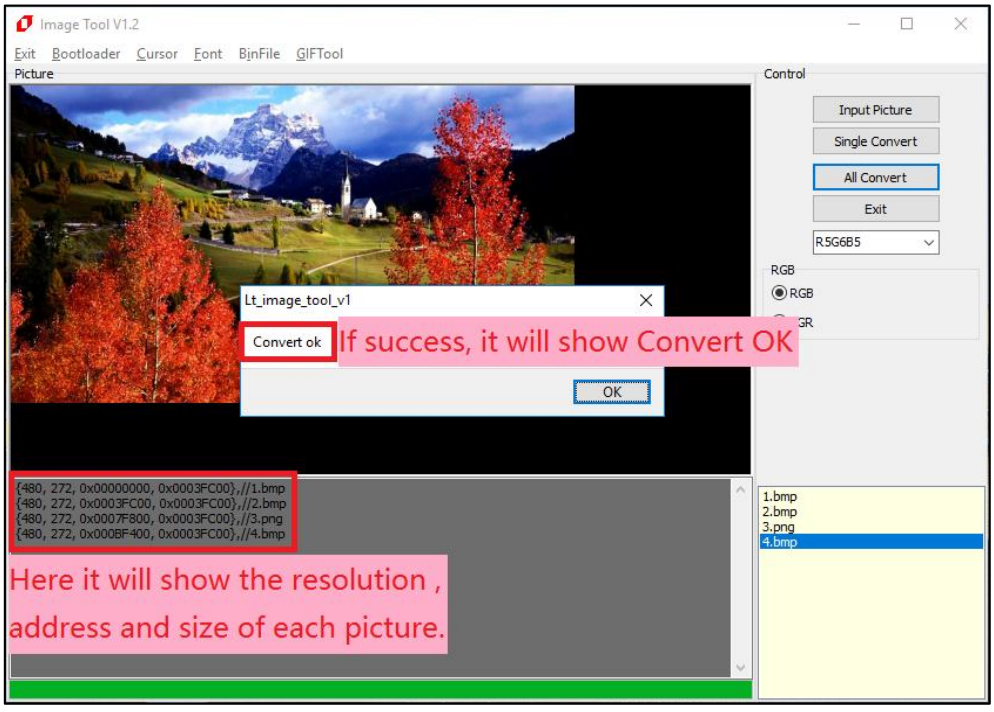

Figure 15-8: Create Picture Bin File

6、 After the Picture Bin file is saved, the exported “Picture1.bin” will be found in the destination folder.

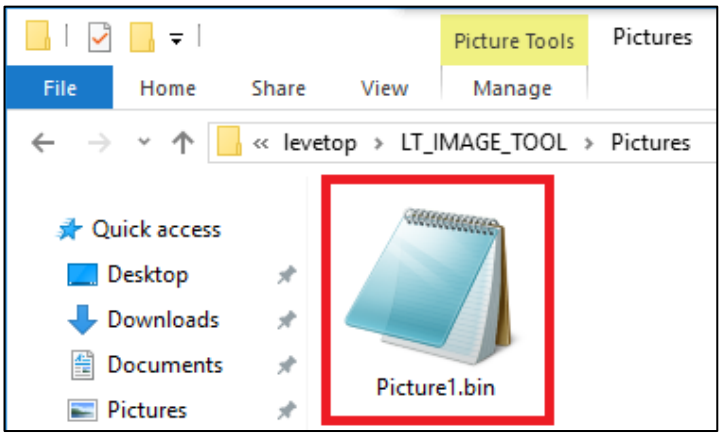

Figure 15-9: Generated Picture Bin file

15.3 Create GIF's Bin File

To play a GIF animation, users can disassemble the GIF file into pictures and then use DMA transfer to save those pictures to the LT768x built-in display memory. This will achieve the effect of displaying the animation. Users can use LT_IMAGE_TOOL.EXE to import an animated GIF file, and then generate a bin file of the disassembled pictures. Users may refer to the manual of LT_IMAGE_TOOL.EXE for detail or the following explanation:

1. Execute LT_IMAGE_TOOL.EXE then click "GIFTool" to open the Gif Picture Bin file production interface:

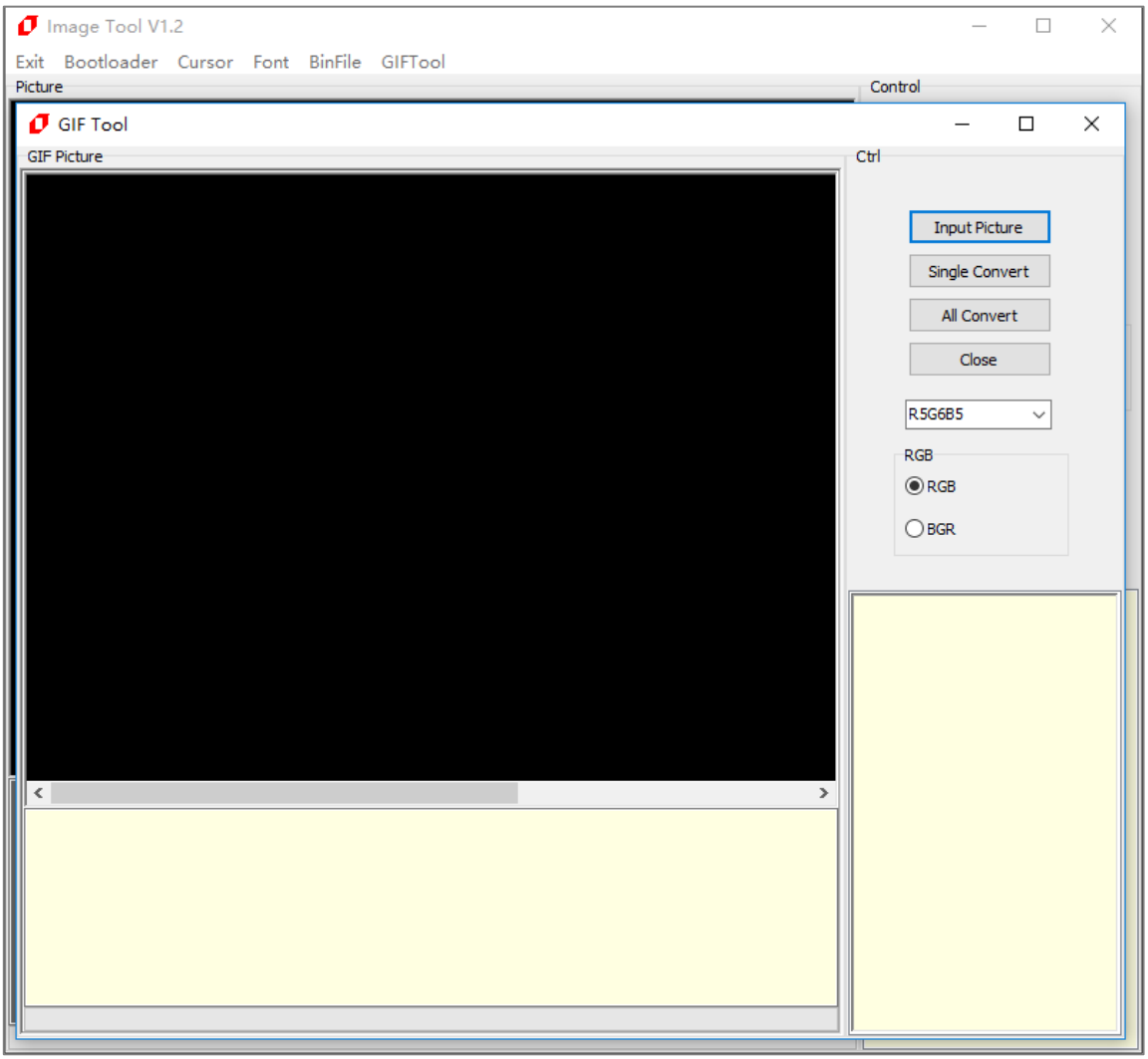

Figure 15-10: Open the Create Gif Picture Window

2. Click “Input Picture” button to select Gif file:

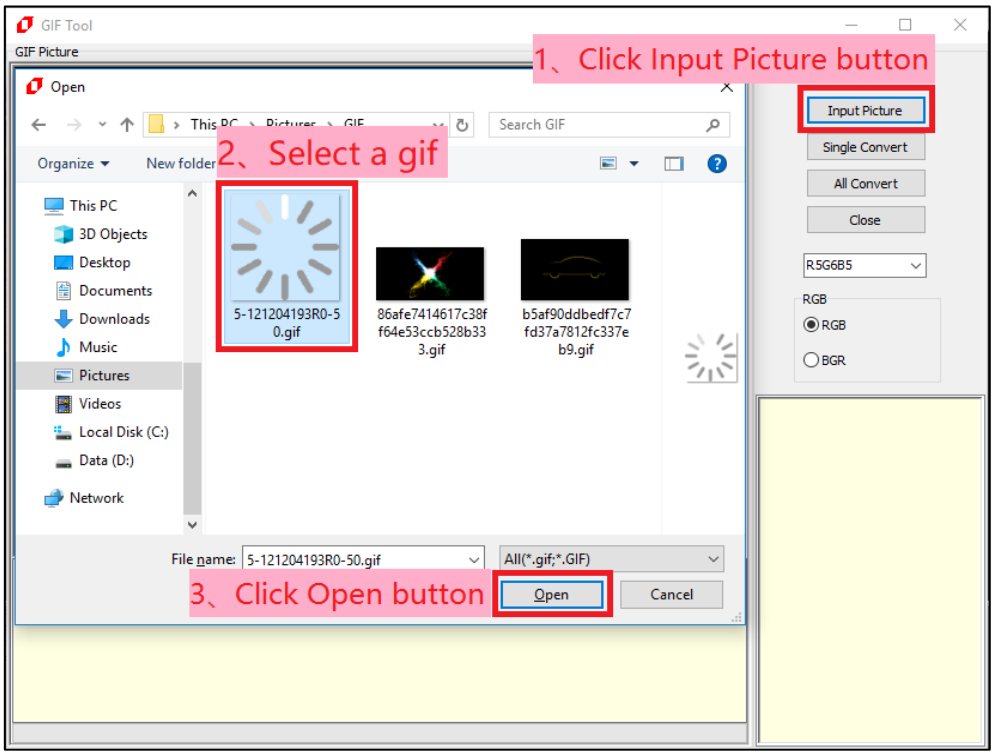

Figure 15-11: Slect Gif File

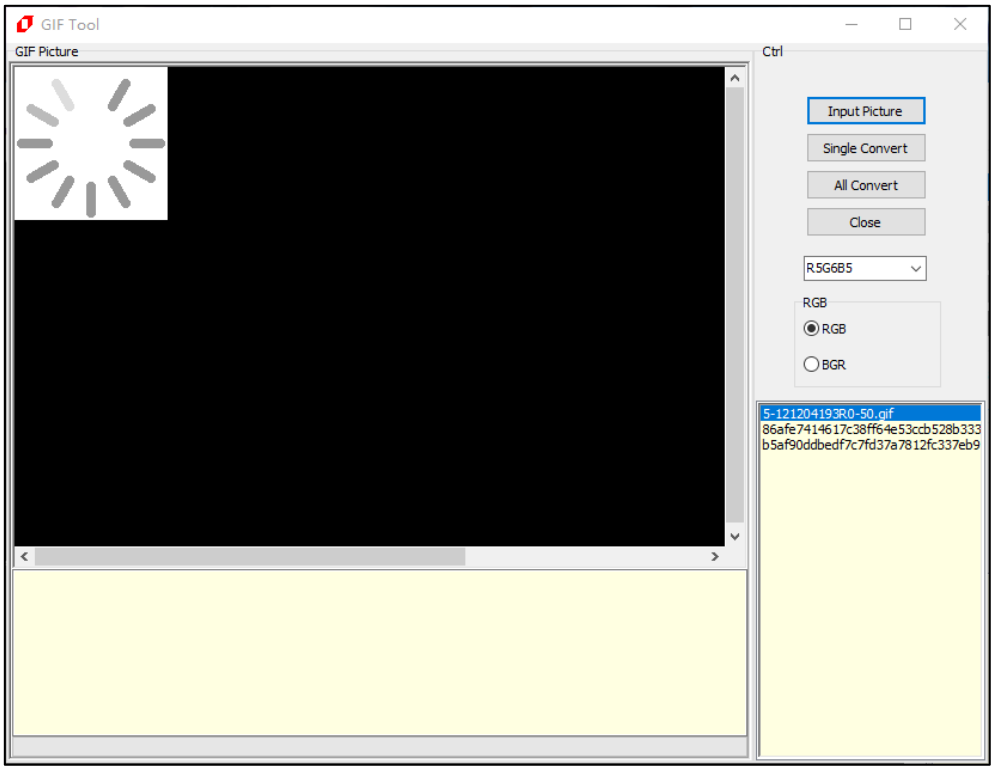

Figure 15-12: Loading the Gif File

3. Setup the output format of picture: 16bpp or 24bpp color depth; and RGB or BGR format.

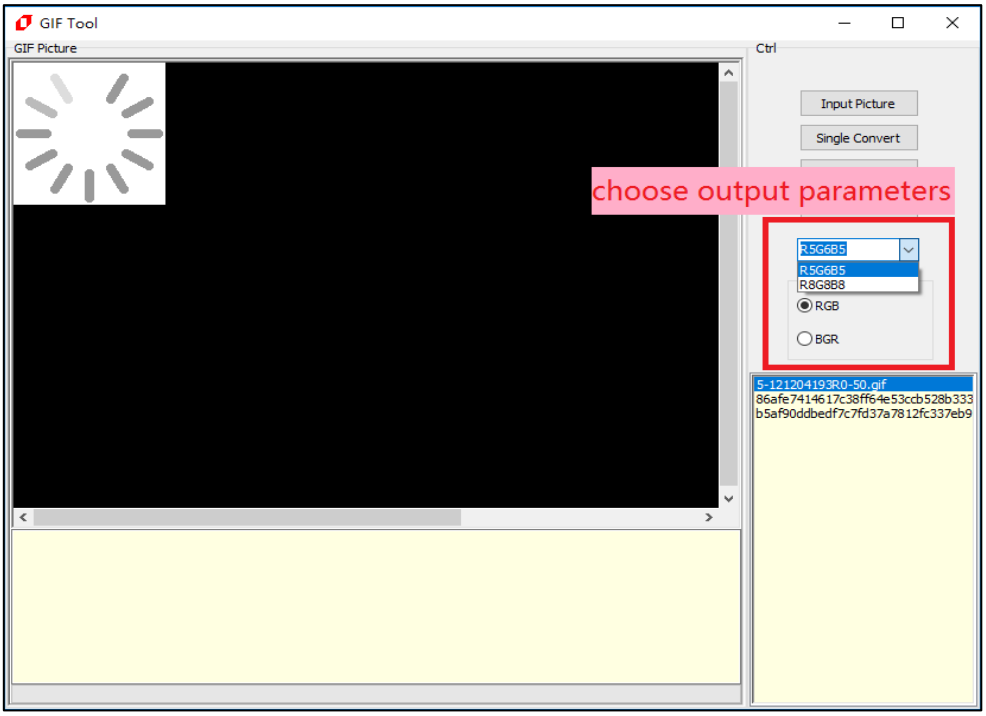

Figure 15-13: Setup the Picture Output Format

4. Export a Gif or all the Gif pictures. Please note the file name should not contain special characters such as:? */* *: |, otherwise the file will not be saved.

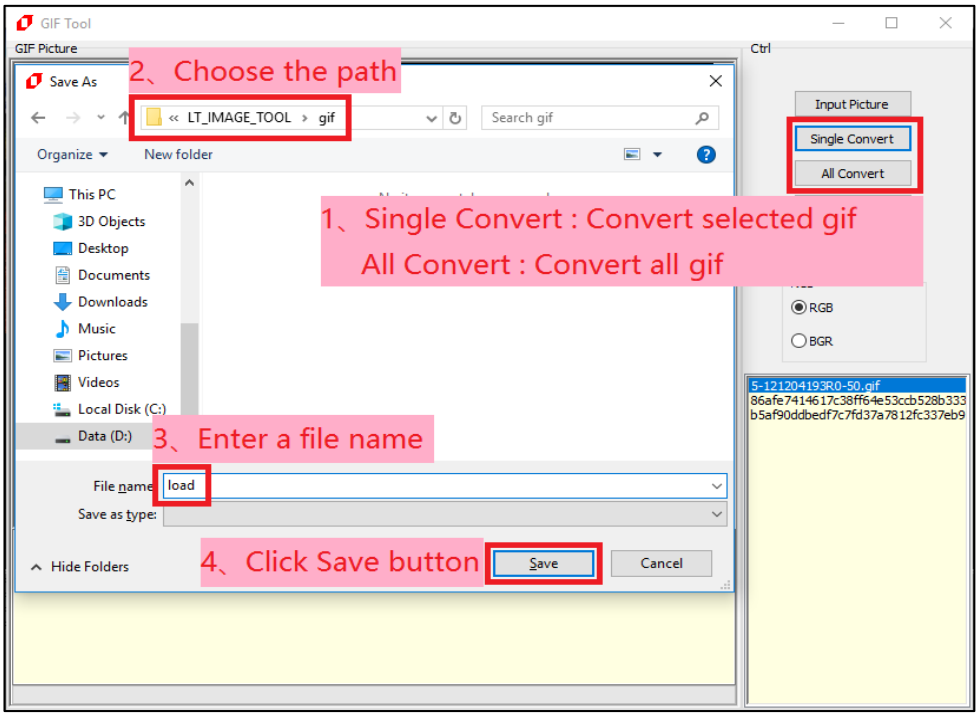

Figure 15-14: Export Picture (1/2)

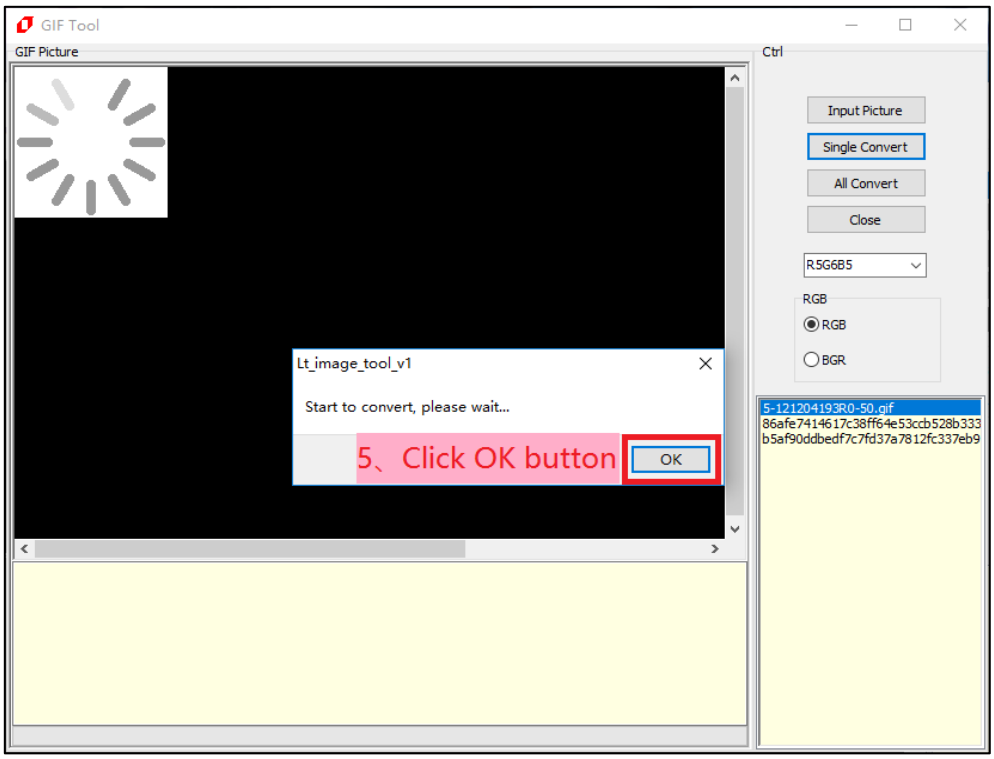

Figure 15-15: Export Picture (2/2)

5. If the file is saved successfully, a prompt message, “Convert ok”, will pop up.

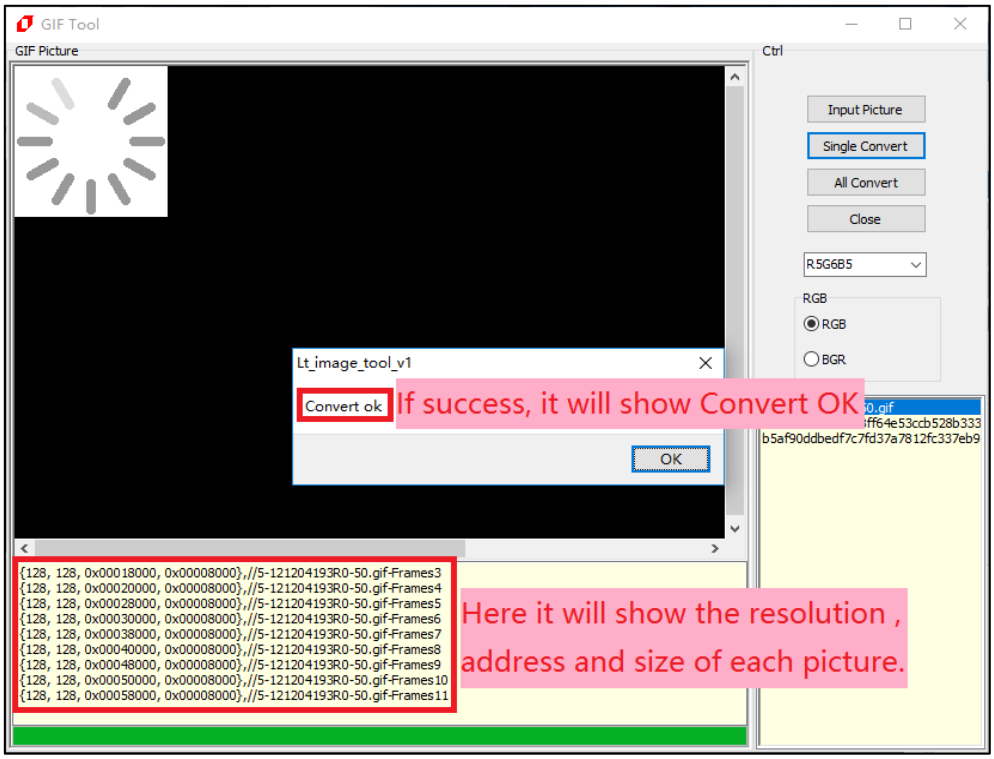

Figure 15-16: Create Picture Bin File

6. After the Picture Bin file is saved, the exported “load.bin” can be found in the destination folder, and a text file that lists the detail information of the pictures will also be found in the folder.

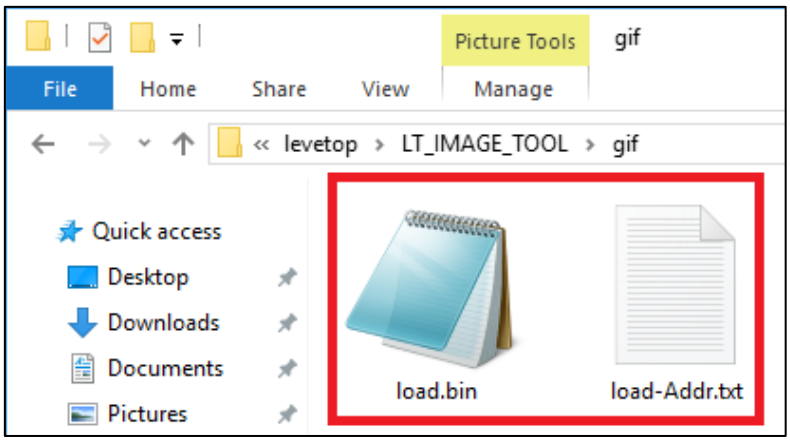

Figure 15-17: Generated Picture’s Bin File and Text File

Users can find the resolution, address, size, and the number of pictures for each picture of the GIF in the TXT document.

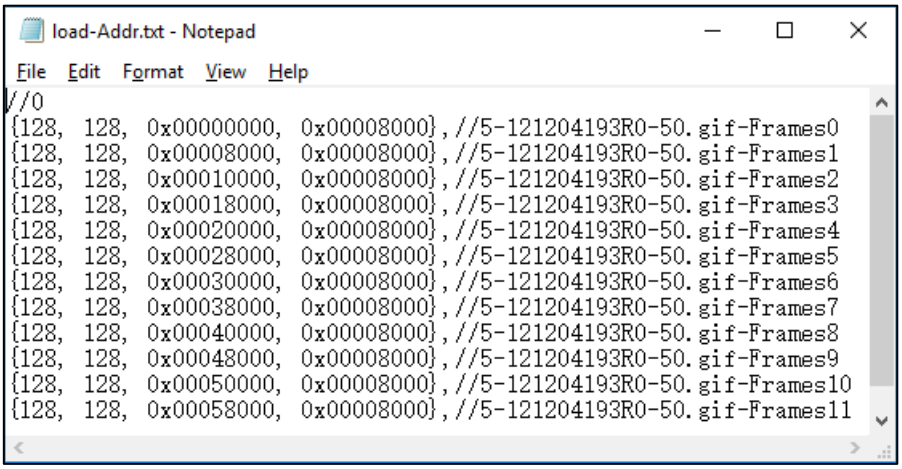

Figure 15-18: Picture’s Information

15.4 The Combination of Bin Files

In the previous section, it is explained that users can create bin files for both pictures and fonts. Users can also combine all of the bin files into one bin file, and then program it into the SPI Flash. Levetop provides a dedicated tool, LT_IMAGE_TOOL.EXE, which has a bin file combination feature that allows users to combine different bin files into one bin file. Users can refer to the manual of LT_IMAGE_TOOL.EXE for detail or the following explanation:

1、Execute LT_IMAGE_TOOL.EXE then click “Binfile”. This program can consolidate up to 6 bin files, and click "File 1~6" to add them sequentially. Please note that the Bootloader bin file needs to be placed at address 0, that is, File1.

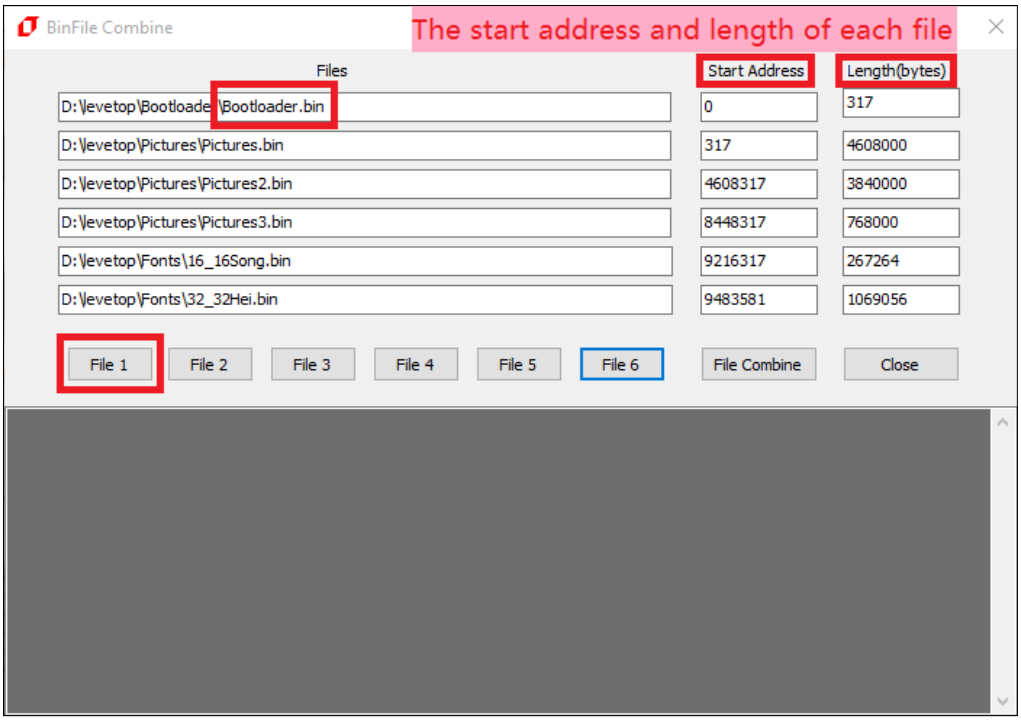

Figure 15-19: The Combination of Bin Files

2、Click the "File Combine" button to save the consolidated file. Please note the file name should not contain special characters such as:? */* *: |, otherwise the fill will not be saved.

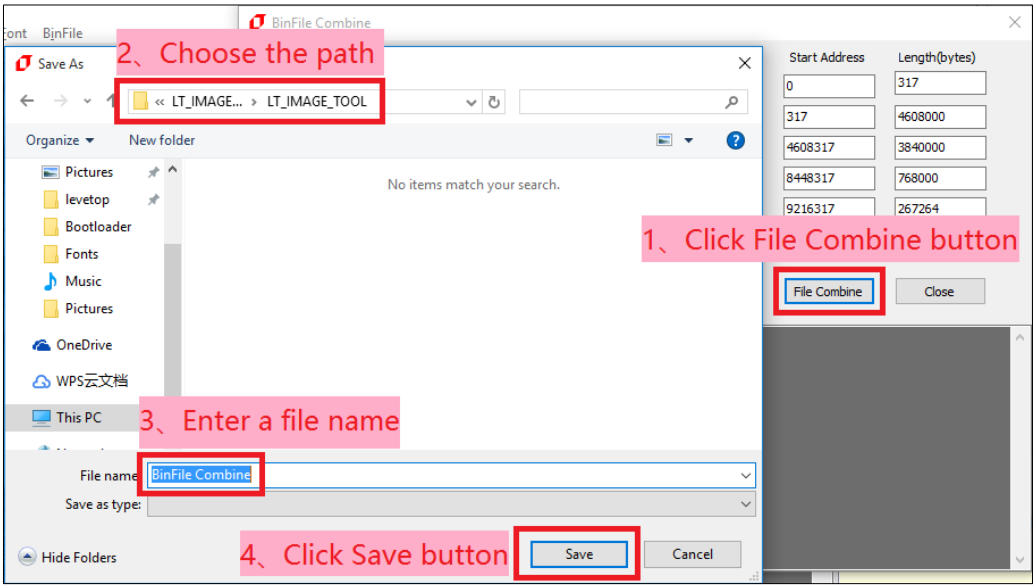

Figure 15-20: Save the Consolidated Bin File

If a prompt message, "Combine over", pops up, it means the combination succeeds, and the address and size of each source file will be displayed int the below window. The tool will also generate a "BinFile Combine-Addr.txt" file which allows users to review the details of each source file's address, size, and so on.

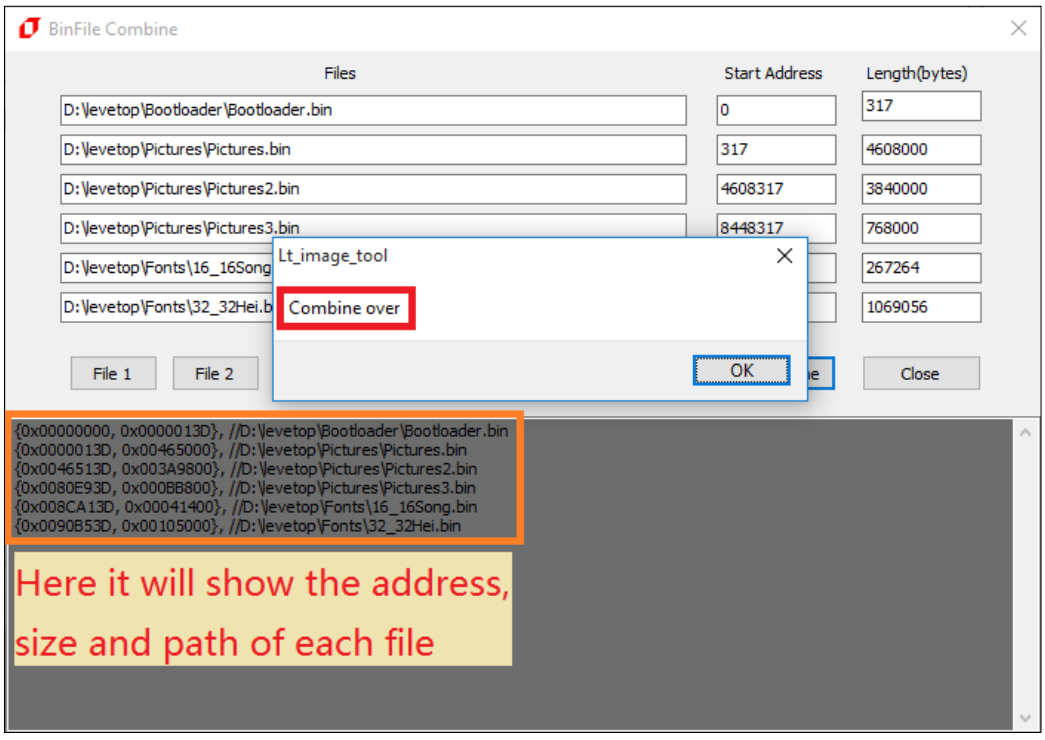

Figure 15-21: Bin Files Combination Success

3、Created a “BinFile Combine-Addr.txt” file:

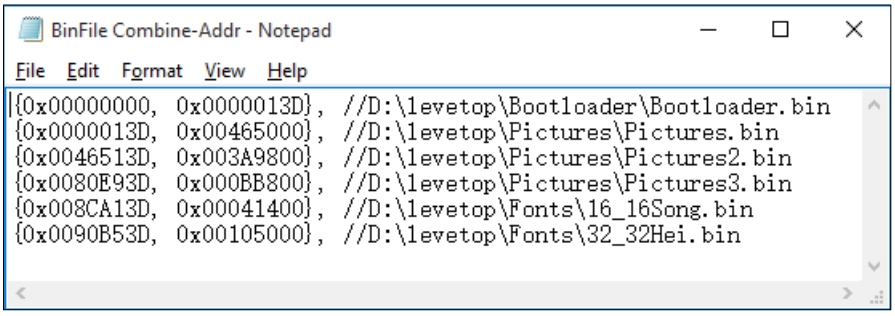

Figure 15-22: The contents of “BinFile Combine-Addr.txt”

After the consolidation is complete, the exported "Binfile Combine.bin" file can be found in the destination folder. Users can use the SPI Flash Programmer to write this file to an external SPI Flash connected to LT768x.

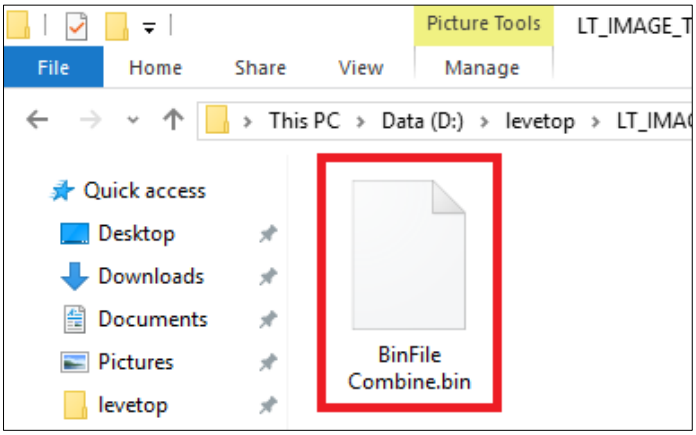

Figure 15-23: Generated "Binfile Combine.bin" File

15.5 How The Program Calls The Bin File of SPI Flash

Here are two examples of how programs invoke bin file data that has been programmed into SPI Flash. One is to display the pictures of "Picture1" and "Picture2" in the program. The other is to call the font “16_16 标楷体” in SPI Flash in the program.

Example 1: Use DMA to read the "Picture1" and "Picture2" from LT768 's external Flash(SPI-0), and display these two pictures circularly on the screen. These two pictures are stored next to each other in the Flash, and both of their resolution are 800*480 with 16bit color depth. The TFT Panel Size is 1024*600, and the top-left corner of the picture is located at the screen (200, 100).

```
Select_Main_Window_16bpp();              // Setup the Color Depth to 16bit
Main_Image_Start_Address(0);             // Setup the Start Address(0) of Main Window
Main_Image_Width(1024);                  // Setup the Width(1024) of Main Window
Main_Window_Start_XY(0, 0);              // Setup Start Coordinate(0, 0) of Main Window
Canvas_Image_Start_address(0);           // Setup the Start Address(0) of Canvas Window
Canvas_image_width(1024);                // Width of Canvas Window
Active_Window_XY(0, 0);                  // Setup Start Coordinate(0, 0) of Active Window
Active_Window_WH(1024, 600);             // Setup Active Window Size: 1024*600
while(1)
{
/*----- Display Picture1, address is 0x0000013D-----*/
LT768_DMA_24bit_Block(0, 0, 200, 100, 800, 480, 800, 0x0000013D);
delay_ms(500);

/*-----Display Picture2, Address is 0x0000013D+800*480*2-----*/
LT768_DMA_24bit_Block(0, 0, 200, 100, 800, 480, 800, 0x0000013D+800*480*2);
delay_ms(500);
}
```

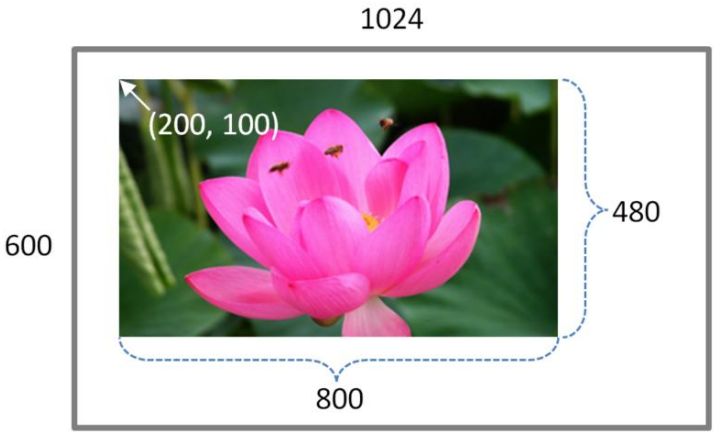

Figure 15-24: Show the Picture which Stored in SPI Flash (Example-1)

Example 2: Use DMA mode to read "16*16 标楷体" font data from the LT768's external Flash (SPI-1), and show a Red text string, "东莞市乐电子有限公司", starting from the position (625, 150) of the TFT screen. The string size is not enlarged, the background color is transparent, and the font is aligned. The TFT panel resolution is still 1024*600 in this example.

```
Select_Main_Window_16bpp();              // Setup the Color Depth to 16bit
Main_Image_Start_Address(0);             // Setup the Start Address(0) of Main Window
Main_Image_Width(1024);                  // Setup the Width(1024) of Main Window
Main_Window_Start_XY(0, 0);              // Setup Start Coordinate(0, 0) of Main Window
Canvas_Image_Start_address(0);           // Setup the Start Address(0) of Canvas Window
Canvas_image_width(1024);                // Width of Canvas Window
Active_Window_XY(0, 0);                  // Setup Start Coordinate(0, 0) of Active Window
Active_Window_WH(1024, 600);             // Setup Active Window Size: 1024*600

LT768_DrawSquare_Fill(0, 0, 1024, 600,White);   // Fill background to White

/*----- Initialization before using the external font -----*/
/*----- The start address of the Font Library: 0x0115E1BD, Library Size: 0x00041400 -----*/
LT768_Select_Outside_Font_Init(1, 0, 0x0115E1BD, 1024*600*2, 0x00041400, 16, 1, 1, 1, 1);

/*----- Display Text String -----*/
LT768_Print_Outside_Font_String(625, 150, Red, White, (u8*)"东莞市乐电子有限公司");
```

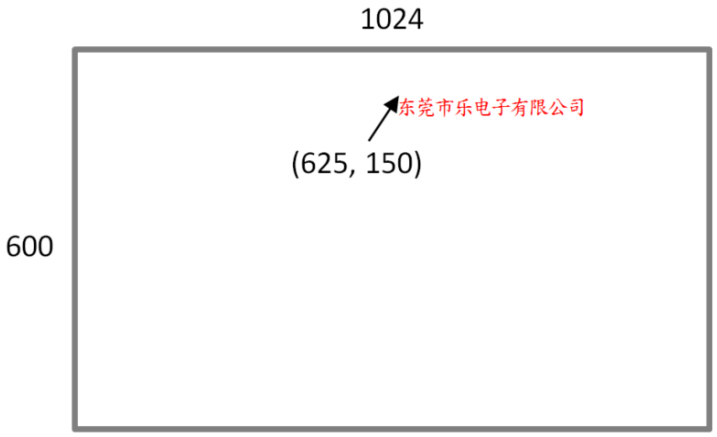

Figure 15-25: Call the Font in SPI Flash and Display a Text String (Example-2)

16. Touch Panel Interface

The touch screen is often used when using a TFT color display. And the touch controller on the touch screen is usually connected to the main control board through the FPC with SPI or I2C interface. Because LT768x provides the SPI master and I2C master features, the FPC pins on the touch screen can be directly attached to LT768x. Then the MCU on the main control board will be able to directly read and control the touch chip through the LT768x MCU interface.

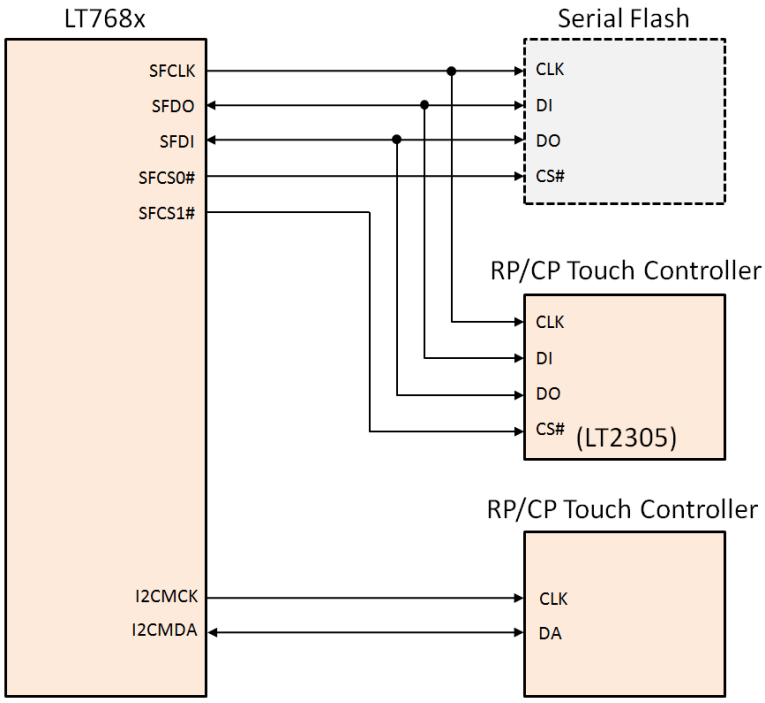

Figure 16-1: Touch Screen Interface of SPI and I2C Master

Figure 16-1 shows an example of the LT768x SPI/I2C touch screen interface. If you use our capacitive touch controller - LT2305, then you can use the SPI Master interface. The detail description of the control mode of SPI/I2C Master and the related sequence diagrams can be referenced in Chapter 10th of LT768x specification.

17. Power Management

In the management mode of power supply, there are four operation modes of LT768x. Depending on the amount of power consumption, from high to Low: Normal mode, Standby mode, Suspend mode, and Sleep mode. The four modes of operation are set by the register REG[DFh]. Following is the comparison table for the related clock action of the four operation modes

Table 17-1: Power Management vs. Clock Active

Item	Normal Mode	Standby Mode		Suspend Mode		Sleep Mode	
	PLL Enable	Parallel MCU	Serial MCU	Parallel MCU	Serial MCU	Parallel MCU	Serial MCU
MCLK	MPLL Clock	MPLL Clock	MPLL Clock	OSC	OSC	Stop	Stop
CCLK	CPLL Clock	OSC	OSC	Stop	OSC	Stop	OSC
PCLK	PPLL Clock	Stop	Stop	Stop	Stop	Stop	Stop
CPLL	ON	ON	ON	OFF	OFF	OFF	OFF
MPLL	ON	ON	ON	OFF	OFF	OFF	OFF
PPLL	ON	ON	ON	OFF	OFF	OFF	OFF

Note:

1. When LT768x enters the power saving mode, the LCD interface will not output the signal. Therefore, before entering the power-saving mode, the Host needs to issue “display off” or “power off” procedure on the LCD module to avoid LCD polarization damage.
2. The “OSC” means external X’tal oscillator.

17.1 Normal Mode

In this mode, three of the internal PLL functions are working normally. That is, Host setup Registers to control the three PLL to generate CCLK (Core Clock), MCLK (Display RAM Clock) and PCLK (LCD scan Clock) so that the whole system can work normally. Because it will take some time for the PLL to work steadily, Host must first check if the PLL frequency is in a stable state through register 01h bit7.

17.2 Standby Mode

```
void LT768_Standby(void);          // Enter Standby Mode
void LT768_Wkup_Standby(void);     // Wake up from Standby Mode
```

17.3 Suspend Mode

```
void LT768_Suspend(void);          // Enter Suspend Mode
void LT768_Wkup_Suspend(void);     // Wake up from Suspend Mode
```

17.4 Sleep Mode

```
void LT768_SleepMode(void);          // Enter Sleep Mode
void LT768_Wkup_Sleep(void);         // Wake up from Sleep Mode
```

17.5 Wake Up

There are three ways to awaken the system from power-saving mode: External interrupt wakeup, keyboard scan wakeup, and software wakeup. If the MCU interface is set to parallel mode, PSM[0] will be the external interrupt input pin. When using an external interrupt or a keyboard scan to wake LT768x up, LT768x will send an interrupt signal INT# to the MCU.

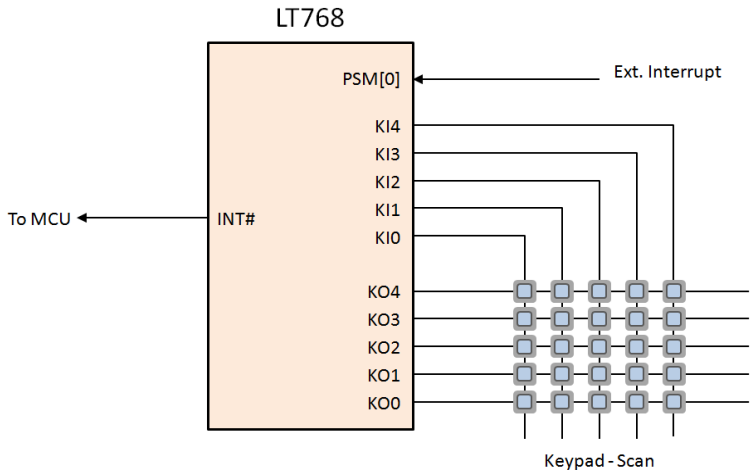

Figure 17-1: Wake Up

Writing "0" to register REG[DFH] BIT7 of LT768x will generate a software wakeup. The bit will be cleared to "0" after LT768x is awakened. REG[DFH] BIT7 will keep “1” before LT768x is fully awkened. The MCU must wait for LT768x to jump out of the power-saving mode before writing data to registers. The MCU can check this bit or the status register bit1 (power saving) to see if the system has returned to normal operating mode. The wake up functions are as follows:

```
void LT768_Wkup_Standby(void);       // Wake up from Standby Mode
void LT768_Wkup_Suspend(void);       // Wake up from Suspend Mode
void LT768_Wkup_Sleep(void);         // Wake up from Sleep Mode
```

18. STM32+LT768x Demo Board

18.1 PCB Interface

The following figure is a demo board composed of STM32F103VE and LT768. It can connect to the standard 50PIN TFT RGB FPC Interface (J5). STM32F103VE (U4) already contains a demo program for 1024*600 resolution TFT panel. LT768x (U1) connects an external 128Mb SPI NOR Flash (U3) that contains the fonts and pictures required for the demo program.

SW1 can be used to choose the interface models (8080/6800-port, SP/I2C serial port) between STM32F103 and LT768x. Users can write or modify STM32F103's demo program through the compilation environment, and then download the program through J2 to internal Flash of STM32F103. The required font and pictures can be converted to Bin file by the tool introduced in the Chapter 11 and 15, and then users can program the Bin file to the SPI Flash through J1.

Users can also update the STM32F103 program and SPI Flash data directly through our assistive tools via the SD card (U5) interface.

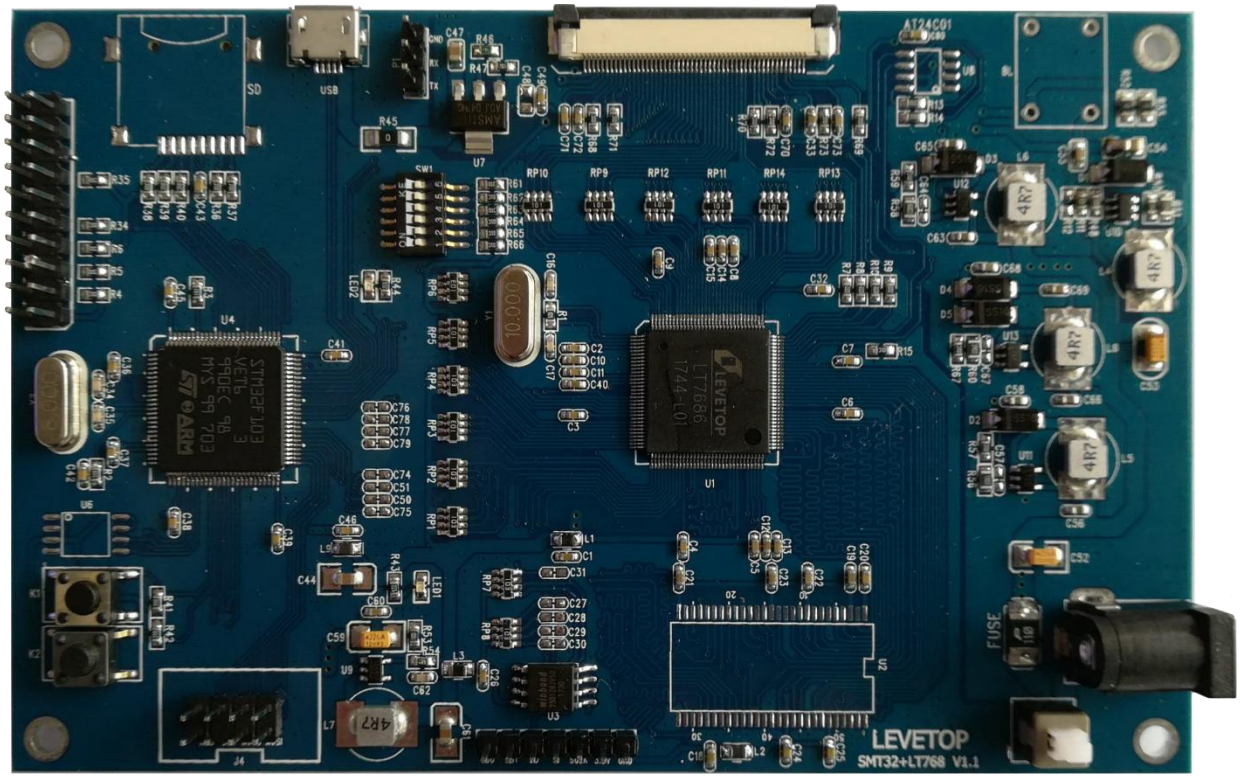

Figure 18-1: STM32+LT768x Demo Board

18.2 Circuit Diagram

For STM32+LT768x Demo Board schematic diagram information, please contact our Sales or FAE department.

File Name: STM32+LT768_Demo Board_V.x.rar

18.3 Demo Program

For STM32+LT768x Demo program information, please contact our Sales or FAE department.

File Name: STM32_LT768_Demo_128Mbit_V10.rar

18.4 SPI Flash Programming

For STM32+LT768x Demo board, if you want to update the data (Bin file) in SPI Flash, there are three ways to do this:

Mode 1: Use the MCU serial (UART) port to receive data from the PC, while updating the SPI Flash data.

The specific control flow chart is as follows:

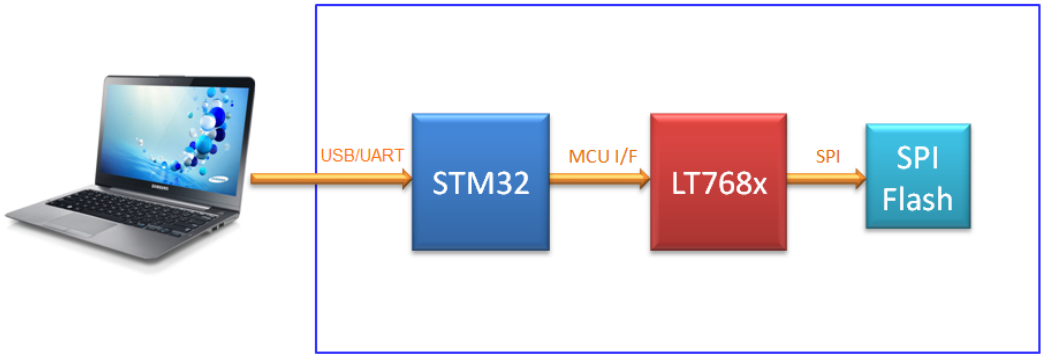

Figure 18-2: Update the data in SPI Flash via MCU UART Serial Port

The source of the data(Bin file) is provided by the PC/NB, and then the MCU receives the data through the UART interface, then MCU transfers Bin file data into SPI Flash through the LT768x SPI Master interface. Please note that the UART speed has to match with the data transmission speed between MCU and LT768x. Because STM32 has a DMA function to receive data through the serial port, this DMA function can be used to receive data from the PC/NB end. In addition, the speed of data writing from LT768x to Flash needs to be faster than the speed of data received by MCU.

Mode 2: Use the SD card connected to the STM32 to update the data in the SPI Flash.

The specific control flow chart is as follows:

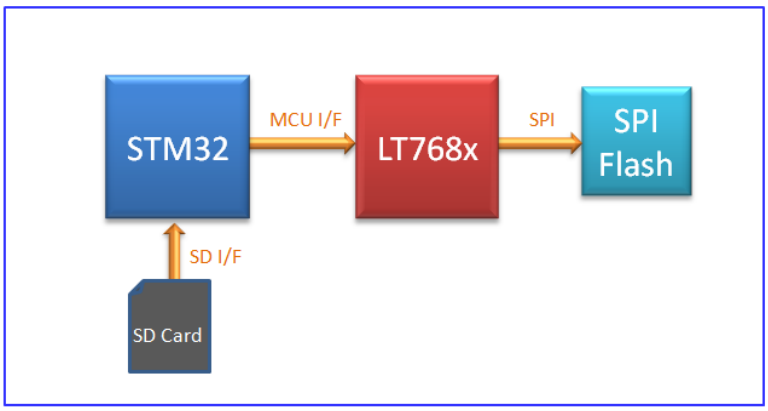

Figure 18-3: Update the Data in SPI Flash through the MCU SD Card

Mode 3: Using a dedicated Flash programmer, the data in SPI Flash is updated directly through the Flash control pins reserved on the board.

Using this method, it requires to pull low LT768x TEST[2] pin, and pull high TEST[1] pin so that LT768x can enter the test mode and disconnect the control of Flash.

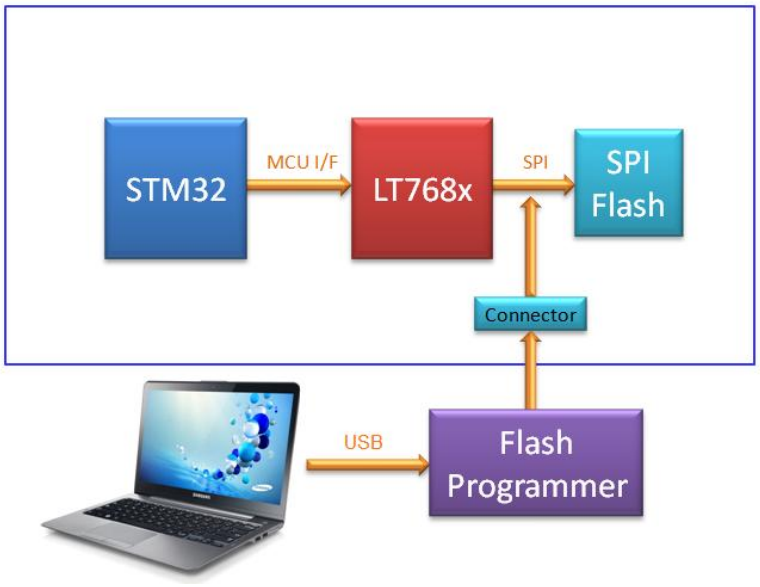

Figure 18-4: Update Data in SPI Flash by PC through Dedicated Flash Programmer

The dedicated Flash Programmer can be easily purchased on the website. It is better to confirm that the SPI Flash model used is supported before purchasing. If you have questions, please contact with us.

Note: Refer to Sections 11.3 and 15.2~15.4 for the creation of Bin files.

19. STC51+LT768x Demo Board

19.1 PCB Interface

The following figure is a demo board composed of STC8051 and LT768. It can connect to the standard 50PIN TFT RGB FPC Interface (J5). STC8051 (U5) already contains a demo program for 1024*600 resolution TFT panel. LT768x (U1) connects an external 128Mb SPI NOR Flash (U3) that contains the fonts and pictures required for the demo program.

SW1 can be used to choose the interface (8080/6800-port, SP/I2C serial port) between STC8051 and LT768x. Users can write or modify STC8051's demo program through the compilation environment, and then download the program through USB to the internal Flash of STC8051. The required font and pictures can be converted to Bin file by the tool introduced in the Chapter 11 and 15, and then users can program the Bin file to the SPI Flash through J1.

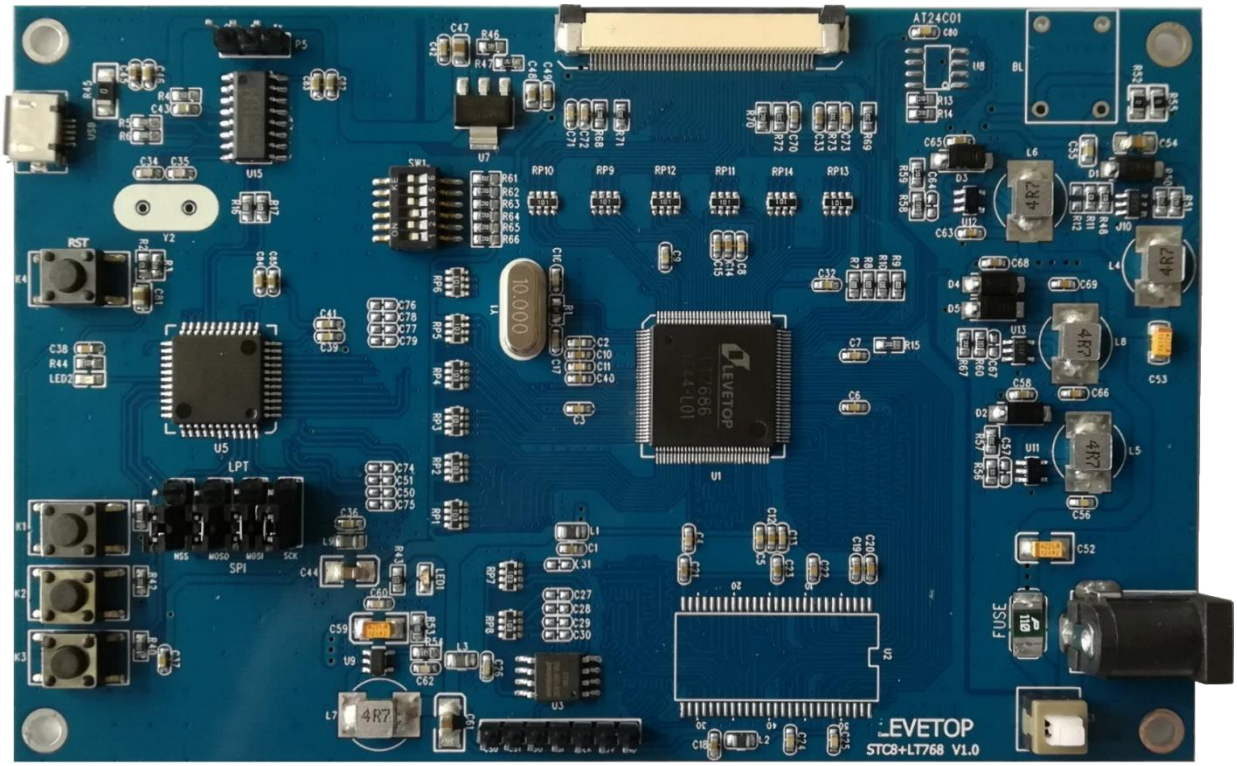

Figure 19-1: STC8051+LT768x Demo Board

19.2 Circuit Diagram

For STC8051+LT768x Demo Board schematic diagram information, please contact our Sales or FAE department.

File Name: STC8051+LT768_Demo Board_V1.x.rar

19.3 Demo Program

For STC8051+LT768x Demo program information, please contact our Sales or FAE department.

File Name: STC8051_LT768_Demo_128Mbit_V10.rar

19.4 SPI Flash Programming

For STC8051+LT768x Demo board, if you want to update the data (Bin file) in SPI Flash, there are three ways to do this:

Mode 1: Use the MCU serial (UART) port to receive data from the PC, while updating the SPI Flash data.

The specific control flow chart is as follows:

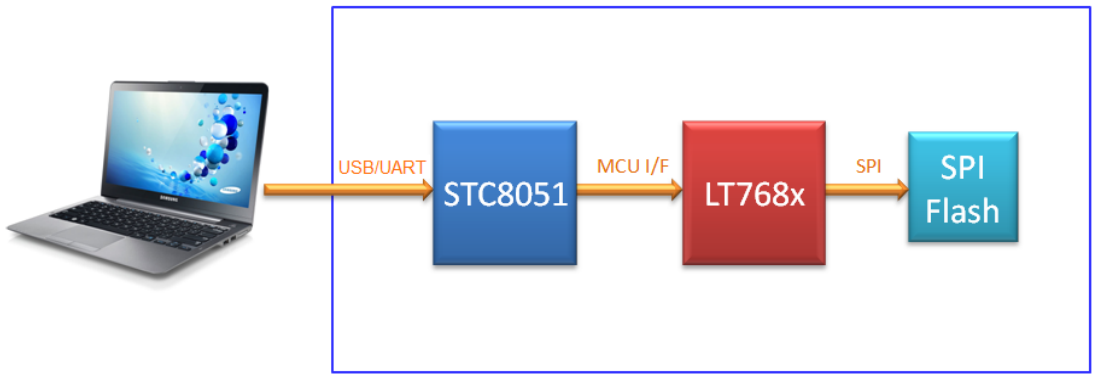

Figure 19-2: Update the data in SPI Flash via MCU UART Serial Port

The source of the data(Bin file) is provided by the PC/NB, and then the MCU receives the data through the UART interface, then MCU transfers Bin file data into SPI Flash through the LT768x SPI Master interface. Please note that the UART speed has to match with the data transmission speed between MCU and LT768x. STC8051 can use the interrupt of the serial port to receive data and use two 1K buffer respectively. One of the buffers is receiving PC/NB data, and another buffer writes the data to Flash. The speed of writing data to Flash needs to be faster than the speed of data received by the serial port.

Mode 2: Using a dedicated Flash programmer, the data in SPI Flash is updated directly through the Flash control pins reserved on the board.

Using this method, it requires to pull low LT768x TEST[2] pin, and pull high TEST[1] pin so that LT768x can enter the test mode and disconnect the control of Flash.

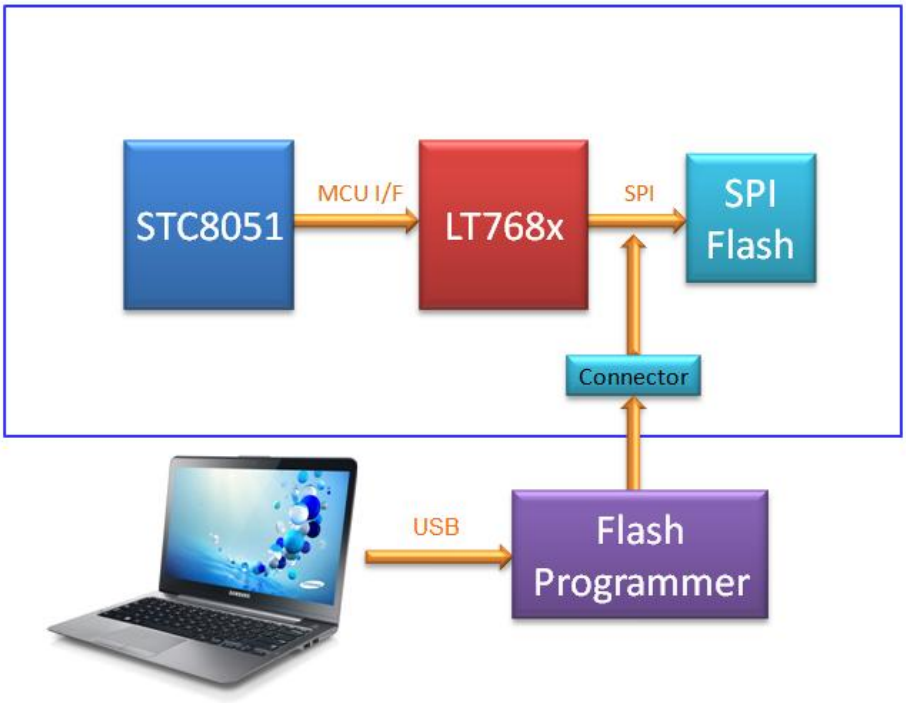

Figure 19-3: Update Data in SPI Flash by PC through Dedicated Flash Programmer

Since STC8051 is an 8-bit MCU, it does not have the ability to receive data by DMA. Therefore, if you use mode 1 from 8051 to program data to SPI Flash, it will be slower. It is suggested that users use dedicated Flash programmer to update SPI Flash.

20. LT7680 SPI Demo Board

20.1 PCB Interface

The following figure is a demo board composed of Levetop's LT32A01 and LT768. It can connect to the standard 40PIN TFT RGB FPC Interface. LT32A01 (U3) is a 32bit MCU developed by Levetop, and it already contains a simple demo program for 480*272 resolution TFT panel. LT7680A (U1) connects an external 128Mb SPI NOR Flash (U2) that contains the fonts and pictures required for the demo program.

LT7680 supports SPI interface only. Users can write or modify LT32A01's demo program through the compilation environment, and then download the program through programmer to internal Flash of LT32A01. The required font and pictures can be converted to Bin file by the tool introduced in the Chapter 11 and 15, and then users can program the Bin file to the SPI Flash through the connector COM8.

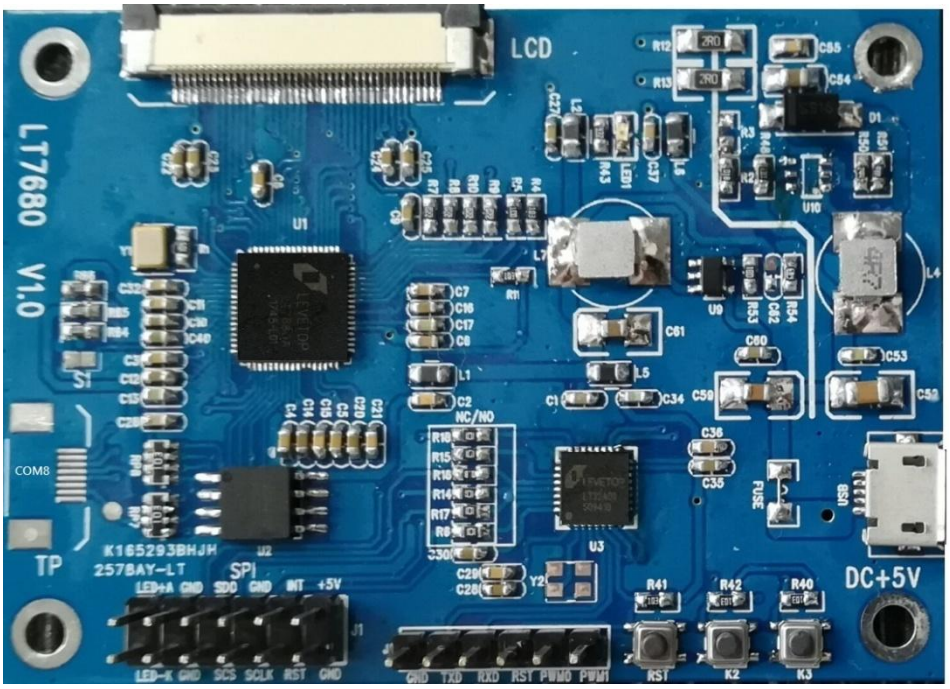

Figure 20-1: LT7680 Demo Board

20.2 Circuit Diagram

For LT7680 Demo Board schematic diagram information, please contact our Sales or FAE department.

File Name: LT7680_SPI Demo Board_V1.x.rar

20.3 Demo Program

For LT7680 Demo Program information, please contact our Sales or FAE department.

File Name: LT7680_SPI Demo_64Mbit_V10.rar

20.4 SPI Flash Programming

For LT32A01+LT768x Demo board, if you want to update the data (Bin file) in SPI Flash, you can use a dedicated Flash programmer to update the data through connector – COM8. This method needs to pull low the RST pin of LT7680, then LT7680 will enter the test mode and disconnect the control of Flash. The data can then be burned to Flash without being affected by LT7680.

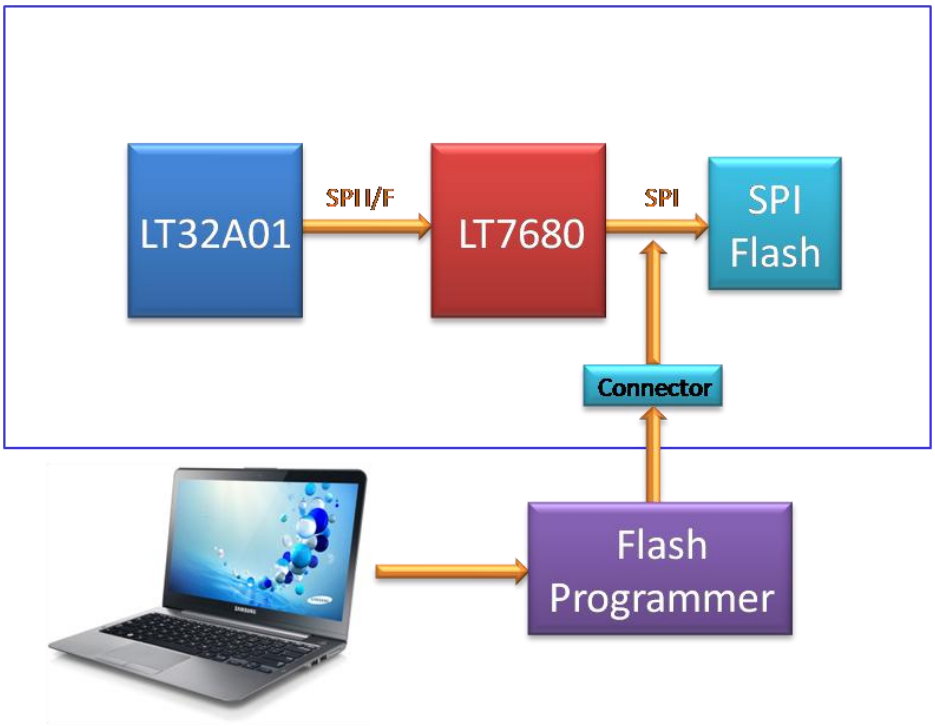

Figure 20-2: Update Data in SPI Flash by PC through Dedicated Flash Programmer

The dedicated Flash Programmer can be easily purchased on the website. It is better to confirm that the SPI Flash model used is supported before purchasing. If you have questions, please contact with us.

Note: Refer to Sections 11.3 and 15.2 ~ 15.4 for the creation of Bin files.

20.5 Use External MCU

If users want to use an external MCU to control LT7680, the R14, R15, R16, and R17 on the demo board must be disconnected. Through the SPI interface (J1), the external MCU can control the LT7680. There is no need to remove LT7680 from the board. Users can refer to the schematic diagram of the LT7680 SPI Demo board.

21. Use LT7681/7683/7686 to Make Standard TFT Module (LCM)

LT768x is suitable to transform the traditional RBG interface TFT module into a TFT module with MCU interface. This chapter will introduce the schematic diagram and precuasons of using LT7681/7683/7686 (LQFP-128pin) to form a standard MCU interface module.T

21.1 Block Diagram

Figure 21-1 is a LT7681/7683/7686 LCM block diagram. LT768x outputs RGB signal directly to the standard RGB TFT-LCD display. The RGB data can be 16bit (5/6/5), 18bit (6/6/6), or 24bit (8/8/8). MCU interface can be chosen from I2C/SPI serial and 6800/8080 port by setting PSM[2:0] (refer to the SW1 of the module schematic diagram).

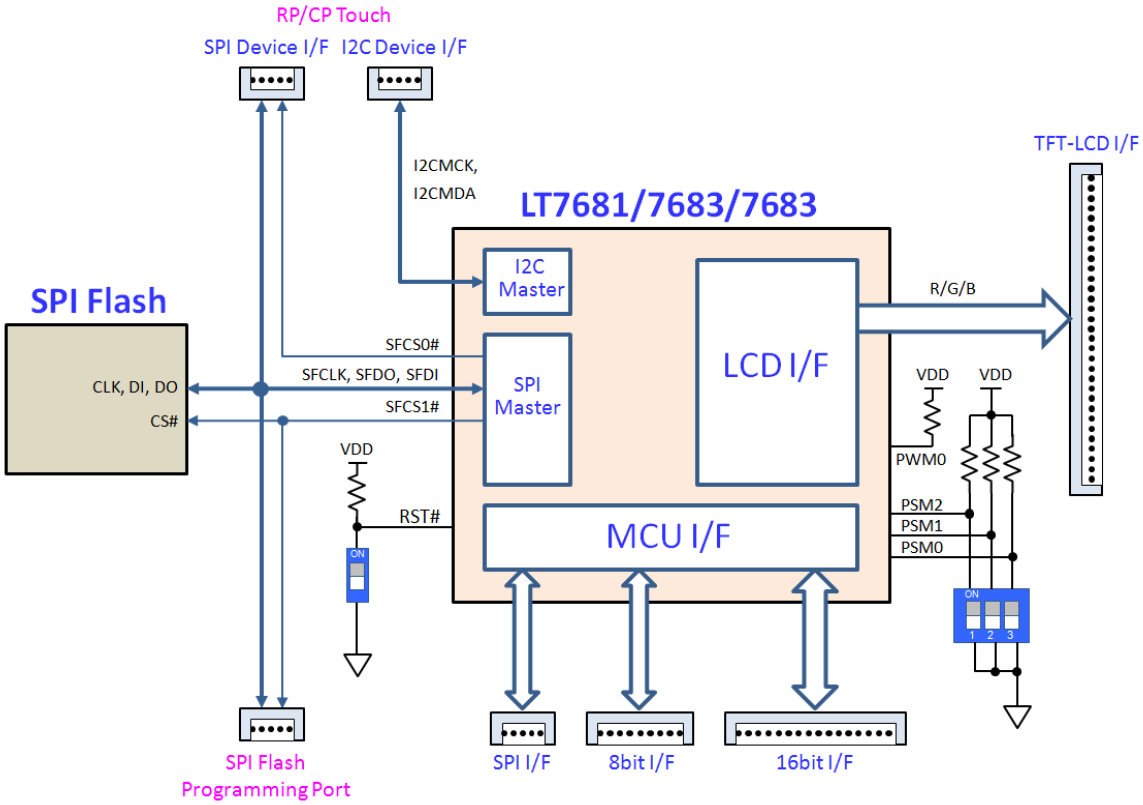

Figure 21-1: LT7681/7683/7686 Block Diagram

When the parallel mode is set, the 8-bit or 16-bit data transfer is determined by the bit0 of the Register REG[01h]. If bit0=0, set to 8bit data bus. If bit0=1, set to 16bit data bus. As shown in Table 21-1 below or in Chapter 2 of the Reference specification:

Table 21-1: MCU Interface Mode

PSM[2:0]	MCU I/F Mode
0 0 X	8bits or 16bits 8080 Parallel Interface Mode
0 1 X	8bits or 16bits 6800 Parallel Interface Mode
1 0 0	3-Wire SPI Mode
1 0 1	4-Wire SPI Mode
1 1 X	I2C Mode

0 represents the DIP switch to ON (grounded) state.

The SPI Flash on the block diagram is mainly used to store the image data and font file used in the application. Module makers can program SPI Flash in advance and then solder it on the PCB for their customers, or use the MCU to transfer the data into SPI Flash through the LT768x SPI Master interface. Users can also program SPI Flash directly with the programmer. It is suggested that the TFT module makers have the SPI Flash programming interface reserved.

If the PWM[0] pin is connected to a 10K pull-up resistor, then the "Power on Display" feature will be enabled. The primary purpose of this feature is to quickly display the screen without an external master processor, or when the primary processor is still in the starting phase. LT768X contains SPI master and I2C master circuits that can be connected to the resistive touch screen or capacitive touch screen. The end customer can control or read touch screen data through LT768x to save MCU interface resources.

TFT module Factory can use the existing RGB TFT screen to connect with LT7681 (or LT7683/LT7686) Control Board to form a standard TFT module with the MCU interface for speeding up the market promotion. As shown in Figure 21-2. The type of MCU interface can be fixed or open to the client to set up (such as the SW1 of the module schematic diagram).

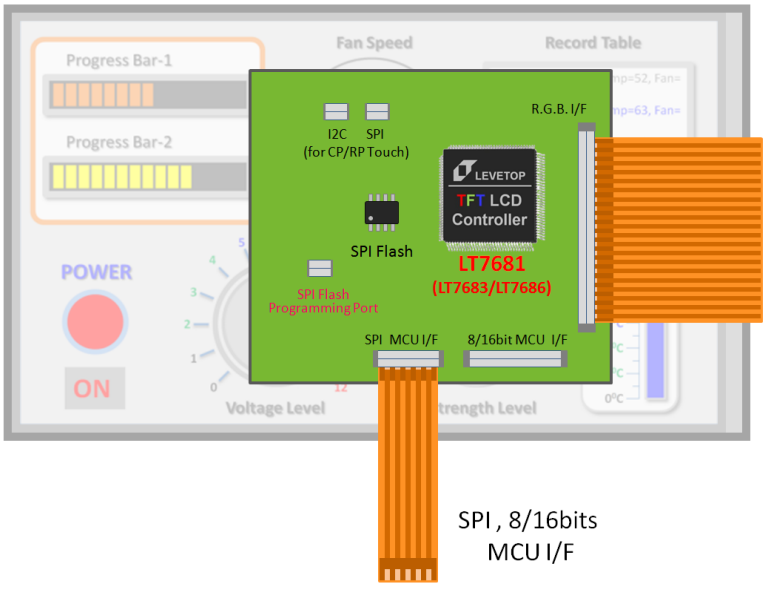

Figure 21-2: TFT Module with MCU Interface

21.2 Schematic Diagram

For LT7681/7683/7686 standard MCU interface module schematic diagram, please contact our Sales or FAE department.

File Name: LT7681.3.6 LCM Demo V1.0.zip

21.3 The Programming of SPI Flash

In the previous Section 21.1, it is mentioned that the SPI Flash can be programmed by MCU through the LT768x's MCU interface, or use a dedicated programmer to program the SPI Flash directly.

SPI Flash's programming interface including 4 control pins and power source (refer to module schematic diagram of the J1 Connector), and then use a dedicated Flash programmer to update the data. Please note that LT768x must be disconnected before writing data to the Flash. Users can refer to the below methods to disconnect LT768x from the SPI Flash:

1. Connect RST# pin of LT768x to ground; or
2. Pull-Low TEST[2] pin of LT768x, and Pull-High TEST[1] pin

It is suggested that the RST# pins can be connected to the Flash programming interface and be externally controlled directly. This will make the Flash programming more convenient.

22. Use LT7680 to Make Standard TFT Module (LCM)

This chapter will introduce the schematic diagram and precautions of using LT7680(QFN-68pin) to form the standard MCU interface module.

22.1 Block Diagram

Figure 22-1 is a LT7680 LCM block diagram. LT7680 outputs RGB signal directly to the standard RGB TFT-LCD display. The RGB data can be 16bit (5/6/5) or 18bit (6/6/6).

LT7680 MCU interface only supports SPI serial port, and can be set through the PSM[1]. As shown in below block diagram, the DIP switch "ON" state is to select 3-wires SPI serial port, "OFF" state is to select 4-wires SPI serial port.

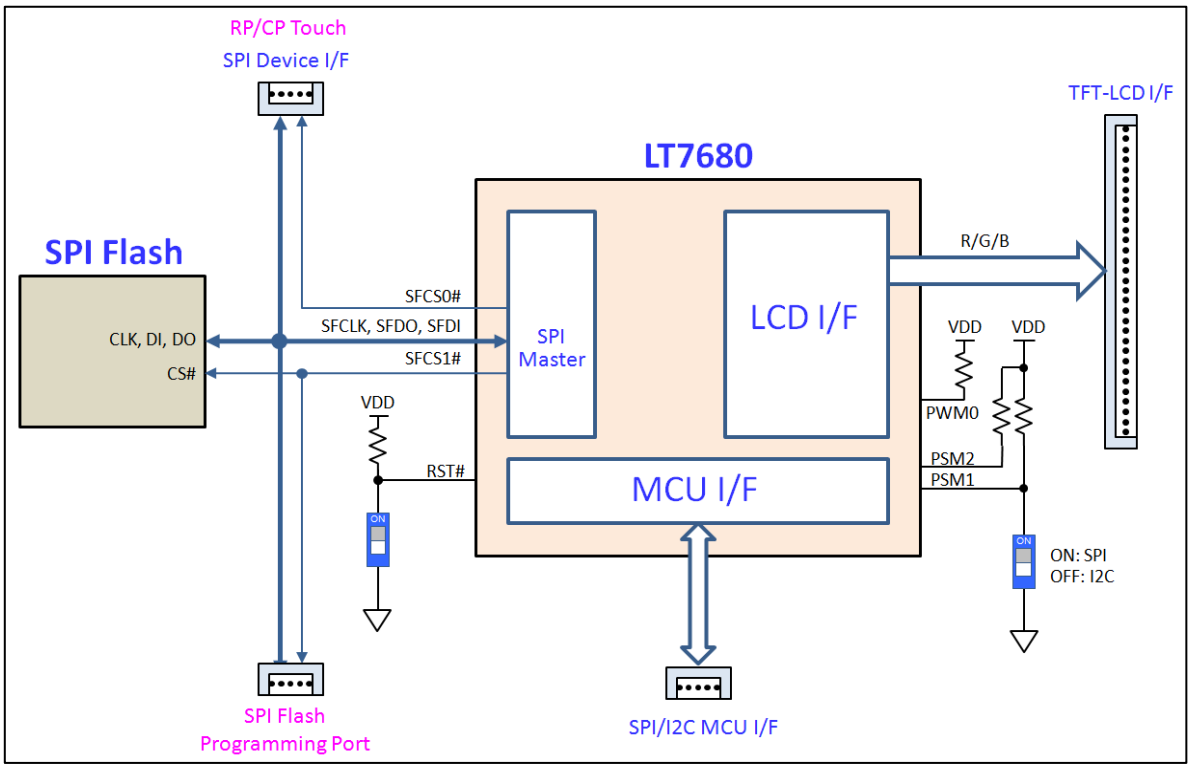

Figure 22-1: LT7680 Block Diagram

The SPI Flash in the block diagram is mainly used to store the image data and the font files used in the application. Module makers can program SPI Flash in advance and then solder it on the PCB for their customers, or use the MCU to transfer the data into SPI Flash through the LT768x SPI Master interface. Users can also program SPI Flash directly with the programmer. It is suggested that the TFT module makers have the SPI Flash programming interface reserved.

If the PWM[0] pin is connected to a 10K pull-up resistor, then the "Power on Display" feature will be enabled (enable). The primary purpose of this feature is to quickly display the screen without an external master processor, or when the primary processor is still in the starting phase. LT7680 contains SPI master circuits that can be connected to the resistive touch screen or capacitive touch screen. The end customer can control or read touch screen data through LT7680 to save MCU interface resources.

TFT module Factory can use the existing RGB TFT screen to connect LT7680 Control Board to form a standard TFT module with the SPI interface for speeding up the market promotion. TFT module Factory can also use the existing RGB TFT Panel and add LT7680's related circuit to modify FPC, as shown in Figure 22-3.

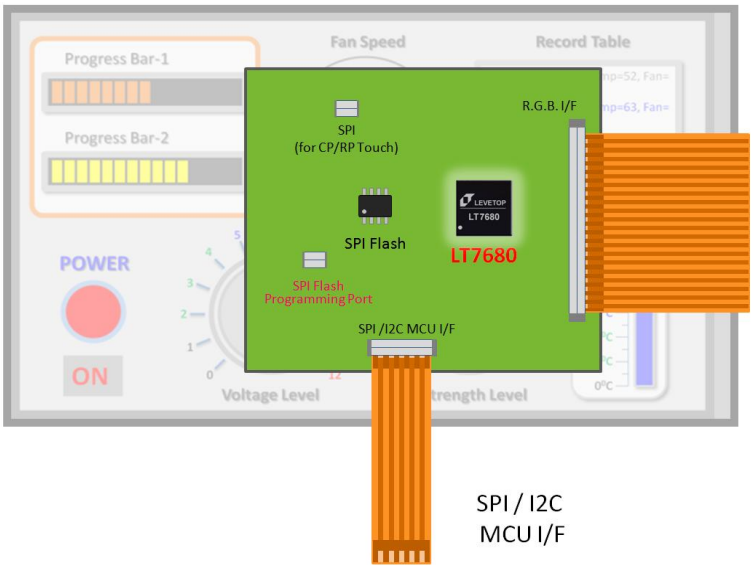

Figure 22-2: TFT Module with SPI/I2C MCU Interface (1)

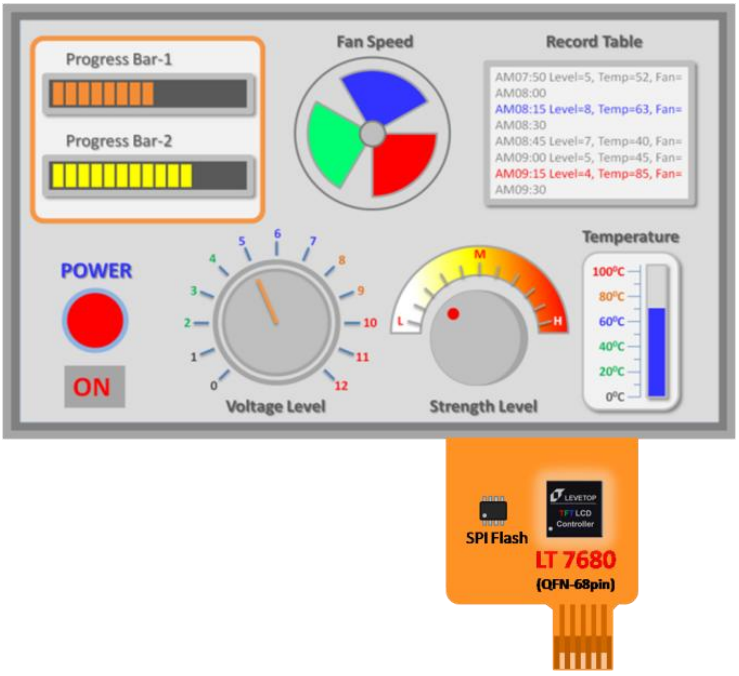

Figure 22-3: TFT Module with SPI/I2C MCU Interface (2)

22.2 Schematic Diagram

For LT7680 standard MCU interface module schematic diagram, please contact our Sales or FAE department.

File Name: LT7680 LCM Demo V1.0.zip

22.3 The Programming of SPI Flash

In the previous Section 21.1, it is mentioned that the SPI Flash can be programmed by MCU through the LT7680's MCU interface, or use a dedicated programmer to program the SPI Flash directly.

SPI Flash's programming interface including 4 control pins and power source (refer to LT7680 module schematic diagram of the J1 Connector), and then use a dedicated Flash programmer to update the data. In order to be able to write data to Flash without being affected by LT7680, you must take one of the following measures before programming:

1. Disconnect LT7680 power supply, only supply power to SPI Flash when programming.
2. The RST# of the LT7680 has to connect to GND.

These two ways can make the LT768x disconnected Flash control, can also use Flash programmer to write successfully. As shown in Figure 22-1 of the preceding Section 22.1, the 2nd method is used (the S1 of the module schematic diagram). That is, the DIP switch is "OFF" in normal state (RST# pin is pull high). When the user want to programming the Flash through the SPI interface, DIP switch has to change to "ON" state (RST# pin is pull low), then the Flash programmer can successfully copy the Bin file data to Flash.

It is suggested that the RST# pins can be connected to the Flash programming interface and be externally controlled directly. This will make the Flash programming more convenient.

23. Function Library Description

23.1 Function Library

- The Register Control Functions of LT768

  File Name: LT768.c

- The Declaration of Register Control Function

  File Name: LT768.h

- The Function Library of LT768

  File Name: LT768_Lib.c

- The Declaration of Function Library

  File Name: LT768_Lib.h

23.2 Application Software

- Bin File Generator for SPI Flash

  File Name: LT_IMAGE_TOOL.EXE

LT_IMAGE_TOOL.EXE is a dedicated program offered by Levetop Semiconductor. It can convert and integrate the pictures, boot program, graphics cursor and font into Bin file format. Users can then write these Bin files to SPI Flash. This program includes 6 features:

- Make Picture's Bin file
- Set up Boot Loader
- Make a Graphics Cursor
- Make the Font's Bin file
- Make GIF Bin file
- Bin file Integration

For the detail, please contact us or refer to the user manual of LT_IMAGE_TOOL.EXE.

24. Function Library List

Table 24-1: Function Library List

No.	Function Name	Function Description	Page
26	VSCAN_B_to_T()	LCD Scan – down to up	32
27	PDATA_Set_RGB()	RBG O/P Format: RGB	32
28	PDATA_Set_RBG()	RBG O/P Format: RBG	32
29	PDATA_Set_GRB()	RBG O/P Format: GRB	32
30	PDATA_Set_GBR()	RBG O/P Format: GBR	32
31	PDATA_Set_BRG()	RBG O/P Format: BRG	32
32	PDATA_Set_BGR()	RBG O/P Format: BGR	32
33	HSYNC_Low_Active()	HSYNC Active Low	32
34	HSYNC_High_Active()	HSYNC Active High	32
35	VSYNC_Low_Active()	VSYNC Active Low	32
36	VSYNC_High_Active()	VSYNC Active High	32
37	DE_High_Active()	DE Active High	32
38	DE_Low_Active()	DE Active Low	32
Main Window's Color Depth			
39	Select_Main_Window_8bpp();	256 Colors	33
40	Select_Main_Window_16bpp();	65K Colors	33
41	Select_Main_Window_24bpp();	262K / 16.7M Colors	33
Setup Main Window			
42	Main_Image_Start_Address()	Start Address of Main Window	33
43	Main_Image_Width()	Width of Main Window	33
44	void Main_Window_Start_XY()	Start Coordinates of Main Window	33
Setup Canvas Window			
45	void Canvas_Image_Start_address()	Start Address of Canvas Window	34
46	void Canvas_image_width()	Width of Canvas Window	34
Setup Active Window			
47	void Active_Window_XY()	Start Address of Active Window	34
48	void Active_Window_WH()	Start Coordinates of Canvas Window	34
MCU Write to Display RAM			
49	void MPU8_8bpp_Memory_Write()	MCU Data Bus: 8bit; Color Depth: 8bpp	35
50	void MPU8_16bpp_Memory_Write()	MCU Data Bus: 8bit; Color Depth: 16bpp	35
51	void MPU8_24bpp_Memory_Write()	MCU Data Bus: 8bit; Color Depth: 24bpp	35
52	void MPU16_16bpp_Memory_Write()	MCU Data Bus: 16bit; Color Depth: 168bpp	35
53	void MPU16_24bpp_Mode1_Memory_Write()	MCU Data Bus: 16bit; Color Depth: 24bpp	35

No.	Function Name	Function Description	Page
54	void MPU16_24bpp_Mode2_Memory_Write()	MCU Data Bus: 16bit; Color Depth: 24bpp	35
Picture-in-Picture (PIP)			
55	LT768_PIP_Init()	PIP Initialize	38
56	LT768_Set_DisWindowPos()	Picture Position of PIP	38
57	Disable_PIP1();	PIP1 Disable	38
58	Disable_PIP2();	PIP2 Disable	38
Direction Control of the MCU Write Data to Memory			
59	MemWrite_Left_Right_Top_Down()	Left→Right, then Up→Down	39
60	MemWrite_Right_Left_Top_Down()	Left→Right, then Up→Down	39
61	MemWrite_Top_Down_Left_Right()	Left→Right (Right Rotate 90° & Flip Horizontal)	39
62	MemWrite_Down_Top_Left_Right()	Left→Right (Left Rotate 90°)	39
Color Bar Display			
63	LT768_Color_Bar_ON()	Color Bar Display Enable	40
64	LT768_Color_Bar_OFF()	Color Bar Display Disable	40
Geometric Drawing			
65	LT768_DrawLine()	Drawing a Thin Line	41
66	LT768_DrawLine_Width()	Drawing a Thick Line	41
67	LT768_DrawCircle()	Drawing a Hollow Circle	42
68	LT768_DrawCircle_Fill()	Drawing a Solid Circle	42
69	LT768_DrawCircle_Width()	Drawing a Solid Circle with Frame	42
70	LT768_DrawEllipse()	Drawing a Hollow Ellipse	44
71	LT768_DrawEllipse_Fill()	Drawing a Solid Ellipse	44
72	LT768_DrawEllipse_Width()	Drawing a Solid Ellipse with Frame	44
73	LT768_DrawSquare()	Drawing a Hollow Rectangle	46
74	LT768_DrawSquare_Fill()	Drawing a Solid Rectangle	46
75	LT768_DrawSquare_Width()	Drawing a Solid Rectangle with Frame	46
76	LT768_DrawCircleSquare()	Drawing a Hollow Rounded-Rectangle	48
77	LT768_DrawCircleSquare_Fill()	Drawing a Solid Rounded-Rectangle	48
78	LT768_DrawCircleSquare_Width()	Drawing a Rounded-Rectangle with Frame	49
79	LT768_DrawTriangle()	Drawing a Hollow Triangle	50
80	LT768_DrawTriangle_Fill()	Drawing a Solid Triangle	50
81	LT768_DrawTriangle_Frame()	Drawing a Solid Rectangle with Frame	51

Version History

Table A-1: Version History

Version	Date	Revision Description
V1.0	2018/03/10	Preliminary version.
V1.1B	2018/12/28	1. Revised Section 11.2.3, 11.2.4 and 11.2.5 for displaying the external Chinese font (GB2312/BIG5 Encoding Format) and ways to use Large (48*48, 72*72) Chinese fonts 2. Add Figure 13-3: Reference Circuit of Backlight Control. 3. Add Section 15.3 for make Bin file of Gif. 4. Update the Section 20.1, 20.4 and 20.5 for LT7680 Demo Board.
V1.2	2019/8/3	1. Update Table 2-1: LT768x Model Options 2. Update Figure 21-1: LT7681/7683/7686 Block Diagram

Copyright Notice