乐升半导体
LEVETOP

LT768x

TFT-LCD 绘图加速控制芯片

High Performance TFT-LCD Graphics Controller

应 用 手 册

V3.0

www.levetop.cn

Levetop Semiconductor Co., Ltd.

目　　录

1. 前言

本应用手册主要在说明 LT768x 的硬件接口与内部功能的实现，同时配合本公司所提供的演示程序、程序库、及原理图，让 TFT 模块厂或是系统端的应用客户很快的能对 LT768x 进行设置及应用开发，能轻易上手并且缩短自行摸索的时间。

1.1 说明

手册中除了硬件及软件的设置说明外，在最后几章也介绍了本公司所提供的 STM32+LT768 演示板、STC8051+LT768 演示板，还有针对 TFT 模块厂将 LT768x 设计到 TFT 模块上时所要注意的事项，及对 SPI Flash 烧录的方式做了完整说明。

手册中所使用到的程序库、原理图都是免费开放的，包括最底层 LT768x 寄存器控制的软件，而本公司也提供了常用的 STM 32 位 MCU，及 8051 系列的 8 位 MCU 演示程序，同时本公司还提供一个专用程序 LT_IMAGE_TOOL.EXE 可以制作图片、字库 Bin 文檔，及整合 Bin 文檔，其进行的步骤也在应用手册有详细的说明。原理图的部分，我们提供了 STM32F103VE 及 STC8A8K64S 分别代表 32 位/8 位 MCU 与 LT768x 连接的原理图，还有将 LT768x 设计到 TFT 模块上时的参考原理图，这些软、硬件资源完全开放给用户，相关数据文件可至本公司网页下载（http://www.levetop.cn）或与本公司业务人员联系。

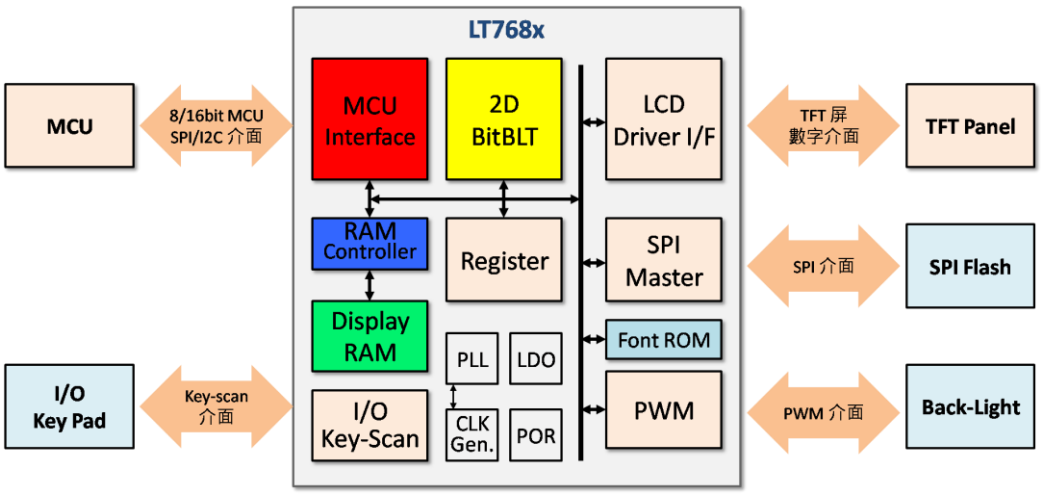


图 1-1：LT768x 内部方块图

1.2 应用架构

TFT 显示器是将图像数据转换成电信号，然后透过 TFT 驱动器不断的利用扫描方式传送这些代表图像数据的信号到液晶屏上，由于扫描数据不断地传送加上人眼的视觉暂留现象，使我们能由 TFT 屏上看到完整的画面，也就是因为 TFT 驱动器并没有储存数据的功能，所以 TFT 显示器不论是呈现动态或是静待图像都必须由系统端（如 MCU）不断地传送图像数据，TFT 控制器的角色就是协助系统端将数据接口转换成 TFT 驱动器接口，让图像数据可以顺利传到 TFT 屏。

LT768x 是一款高效能 TFT-LCD 图形加速显示控制器，除了上面所说协助 MCU 将所要显示到 TFT 屏的内容传递给 TFT 驱动器（Driver）外，它还提供了 2D 图形加速、PIP（Picture-in-Picture）、几何图形绘图等功能，而为了减少 MCU 传递图像数据所花费的时间，LT768x 还提供 SPI Master 接口，将存在 SPI Flash 的图像数据 - 如在应用端会常用到的图片、字库等，透过 DMA 传输方式将显示数据存到 LT768x 内建的高容量显示内存（Display RAM，64/128Mbits），然后 LT768x 内部会依据所选择的显示窗口将指定的显示内存数据透过 RGB 接口不断的传送到外部 TFT Panel 内的驱动器，因此不只提升了显示效能，还大大的降低 MCU 处理图形显示的负担，甚至当系统端只使用 8bits MCU 都可能做得到带 TFT 屏显示的方案。下图是 LT768x 的基本应用架构：

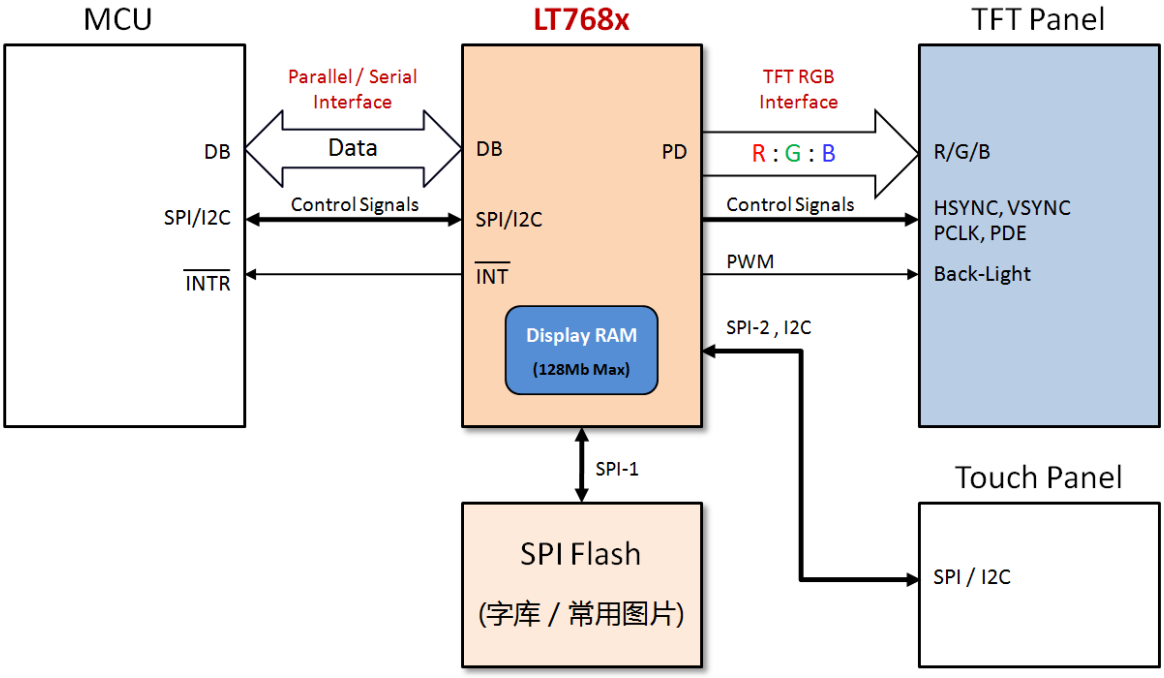


图 1-2：LT768x 应用架构图

LT768x 提供 2 组 SPI Master 接口，1 组 I2C 接口，如果使用触控屏还可以将触控芯片的 I2C 或是 SPI 接口接到 LT768x 上，再由 MCU 透过 LT768x 的 MCU 接口去读取或是控制触控芯片，简化触控芯片的连接方式。

2. 型号选择

表 2-1：LT768x 型号选型表

支持功能		LT7680A-R	LT7680B-R	LT7681	LT7683+	LT7686
类别	功能说明					
封装	LQFP / QFN	QFN-68	QFN-68	LQFP-128	LQFP-128	LQFP-128
LCD 规格	分辨率	1280*1024	480*320	640*480	1024*768	1280*1024
	色彩	262K 色	262K 色	16.7M 色	16.7M 色	16.7M 色
	TFT 接口	RGB (18bits/Max)	RGB (18bits/Max)	RGB (24bits/Max)	RGB (24bits/Max)	RGB (24bits/Max)
显示内存	显示内存	128Mbit	128Mbit	128Mbit	128Mbit	128Mbit
	图层显示	4 层/Min	22 层/Min	18 层/Min	11 层/Min	4 层/Min
MCU 接口	8080 8bit			V	V	V
	6800 8bit			V	V	V
	8080 16bit			V	V	V
	6800 16bit			V	V	V
	3 线 SPI 串口	V	V	V	V	V
	4 线 SPI 串口	V	V	V	V	V
	I2C 串口			V	V	V
其他接口	SPI Master (DMA Flash)	V (2 组)	V (2 组)	V (2 组)	V (2 组)	V (2 组)
	I2C Master			V	V	V
	PWM 输出	2	2	2	2	2
	GPIO 输出	7/max	7/max	28/max	28/max	28/max
	智能键盘接口			5*5	5*5	5*5
绘图功能	2D 绘图加速器	V	V	V	V	V
	几何绘图加速器	V	V	V	V	V
	多边形几何绘图	V	V	V	V	V
	画中画 PIP	V	V	V	V	V
	Virtual Display	V	V	V	V	V
	垂直画面卷动	V	V	V	V	V
	水平画面卷动	V	V	V	V	V
	画面旋转	V	V	V	V	V
	Alpha-Blending	V	V	V	V	V
	图形光标	V	V	V	V	V
	开机画面	V	V	V	V	V
	彩条测试	V	V	V	V	V
文字功能	内建英文字库	ISO8259	ISO8259	ISO8259	ISO8259	ISO8259
	支持中文字库 (外挂 Flash)	V	V	V	V	V
	文字放大	4*4 倍	4*4 倍	4*4 倍	4*4 倍	4*4 倍
	文字旋转	V	V	V	V	V
	文字光标	V	V	V	V	V
	自定义文字	V	V	V	V	V
电源	休眠模式 (S_{tandby}/S_{uspend}/S_{leep})	V	V	V	V	V
	电源	3.3V	3.3V	3.3V	3.3V	3.3V

图 2-1：LT768x 系列

LT768x 相同封装的脚位都是兼容的，例如 LT7681、LT7683+、LT7686 都是 128Pin LQFP 封装，它们的脚位是兼容的；而 LT7680A、LT7680B 都是 68Pin QFN 封装，它们的脚位也是兼容的。分辨率则是向下兼容，例如 LT7686 的分辨率是 1280*1024，它也可以用在更低分辨率的 TFT 屏上。

LT7680 是 68Pin QFN（8mm*8mm）封装，外观尺寸较小，除了可以使用在系统 PCB 板或是 LCM PCB 上也可以焊在 FPC 上做成 SPI 标准 TFT 模块，如下图 2-2 所示。也可以将标准 TFT 驱动模块加上 LT768x 控制板变成标准带控制器的 TFT 模块，让大多数的 8/16/32bits 单片机可以直接使用，如图 2-3 所示。

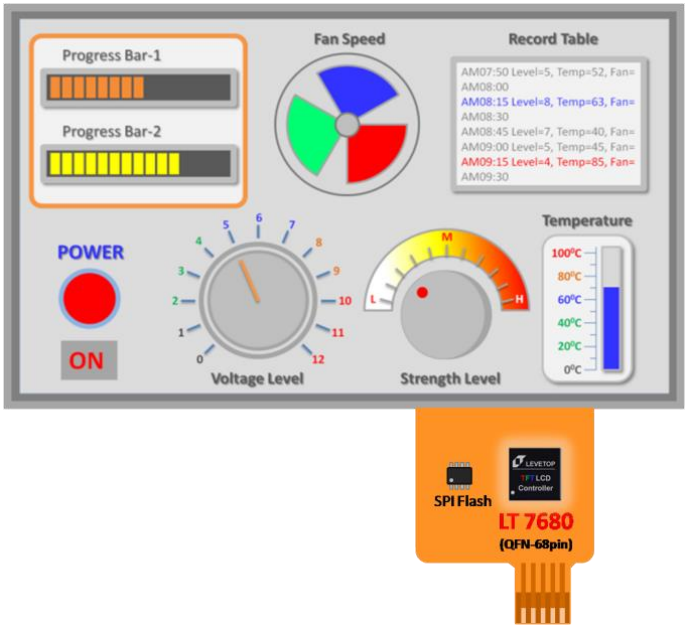


图 2-2：LT7680 标准 SPI TFT 模块

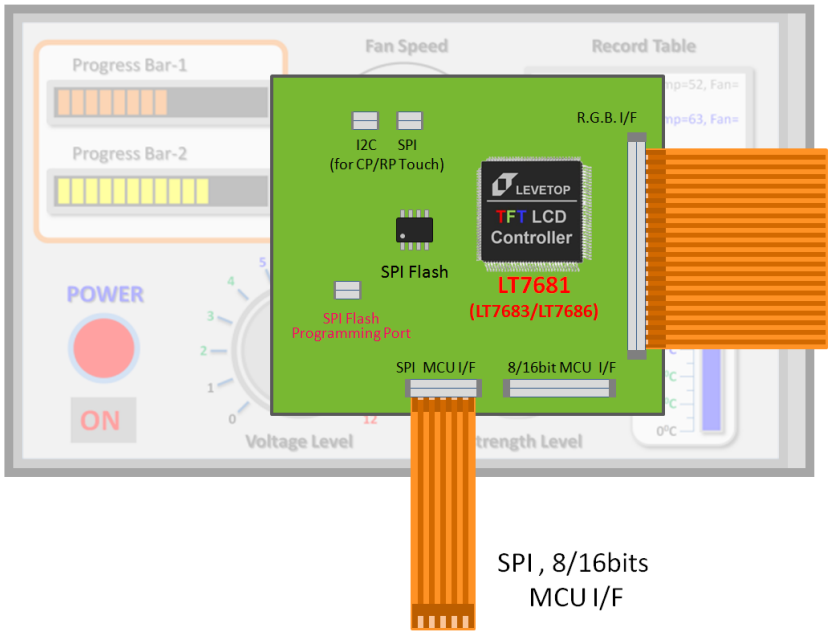


图 2-3：标准带控制器的 TFT 模块

LT768x 是受到主控 MCU 所控制，因此也可以放置在系统主控端，再搭配 RGB 标准 TFT 驱动模块使用，如下图 2-4。

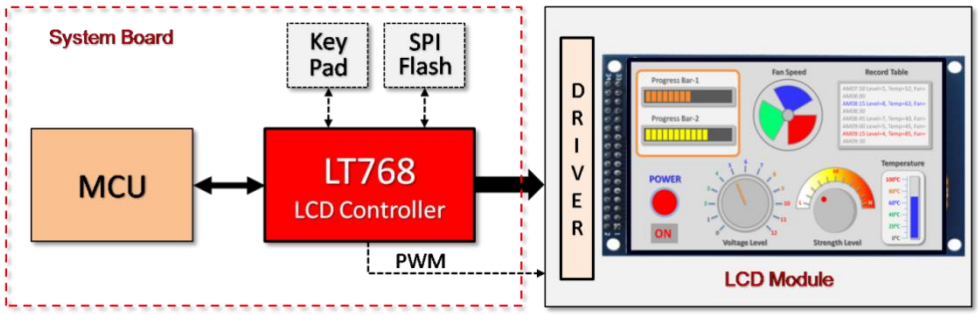


图 2-4：LT768x 放在系统主控端

3. 复位

3.1 电源开启复位

LT768x 内建电源重置 POR（Power On Reset）电路，会产生一个主动的低信号，可以通过 RST#引脚输出到外部电路来同步整个系统。当系统电源（3.3V）启动时，内部复位将启动，直到内部电源稳定，也就是 256 个晶振频率（OSC）时钟之后。

3.2 外部复位信号

外部复位信号 RST#可以让 LT768x 与外部系统同步，外部复位信号必须稳定至少 256 晶振时钟才会被承认，如图 3-1 所示。而 MCU 在开始设定 LT768x 之前，应检查状态寄存器 STSR 的 bit1 - 工作模式状态指示位，以确保 LT768x 目前处于“正常运行状态"。外部复位可以采用 Power-On 复位或是透过 MCU I/O 口进行硬件复位，如图 3-2、图 3-3 所示。

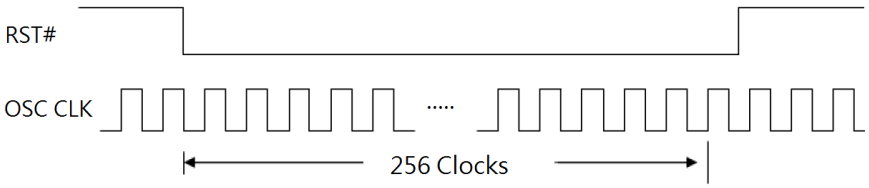


图 3-1：外部复位信号

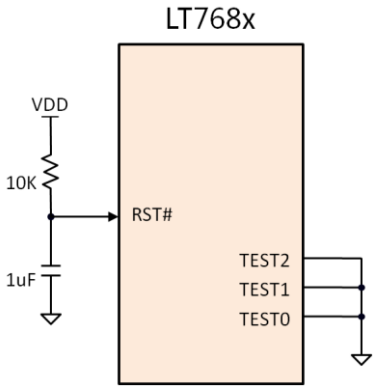


图 3-2：外部复位方式（1）

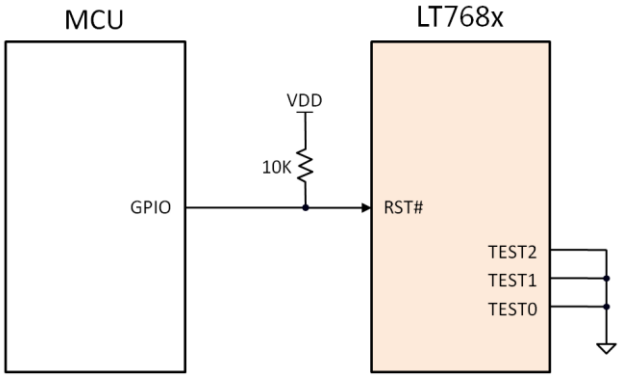


图 3-3：外部复位方式（2）

3.3 软件复位

如果 MCU 对寄存器 REG[00h] bit0 写入 1，LT768 将会进行软件复位（Software Reset），软件复位只会复位 LT768 内部的状态机，其他寄存器值不会被影响或是被清除。复位完成后 REG[00h] bit0 也会自动被清为 0。

当要进行软件复位，只需调用以下的这个函数：

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

3.4 测试信号

LT768x 的 TEST[2:0]是测试模式信号，这些引脚是提供给 LT768 在测试时使用，正常使用应连接到地（GND），如上图 3-2、3-3 所示。

如果要在关机时想要单独对接在 LT768x SPI Master 上的 SPI Flash 进行数据更新，可以把 LT768x 的 TEST[2] 引脚拉低、TEST[1] 引脚拉高，让 LT768x 进入测试模式及断开对外部 SPI Flash 的控制，这样就能将数据烧录到 Flash 而不受 LT768 的影响(请参考本手册的第 18.4、19.4、21.3 及 22.3 节的说明)。

4. 时钟设定

4.1 时钟与 PLL

LT768 需要外接一个 10MHz 的晶振（如图 4-1），或是时钟信号（如图 4-2），如果 MCU 资源足够还可以用一 PWM 输出来产生时钟信号（如图 4-3），用来作为内部 3 组 PLL 的时钟来源，所产生的 3 个时钟分别为：

- CCLK：用于 Core 的时钟信号（Core Clock），最大 100MHz；
- MCLK：用于 Display RAM 的时钟信号（Memory Clock），最大 133MHz；
- PCLK：用于提供给 TFT 屏驱动器的时钟信号（Pixel Clock），最大 80MHz。

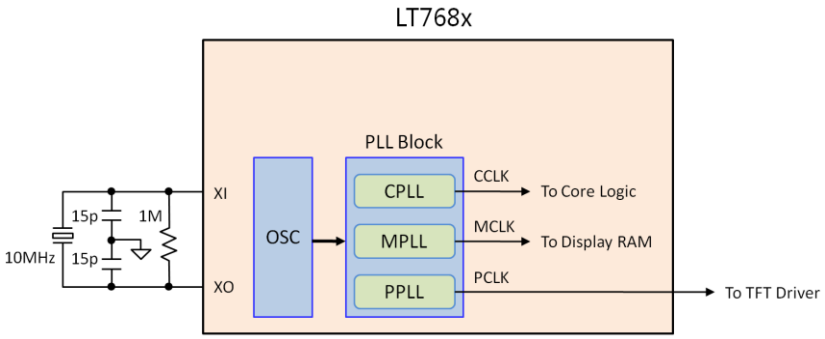


图 4-1：LT768x 时钟电路（一）

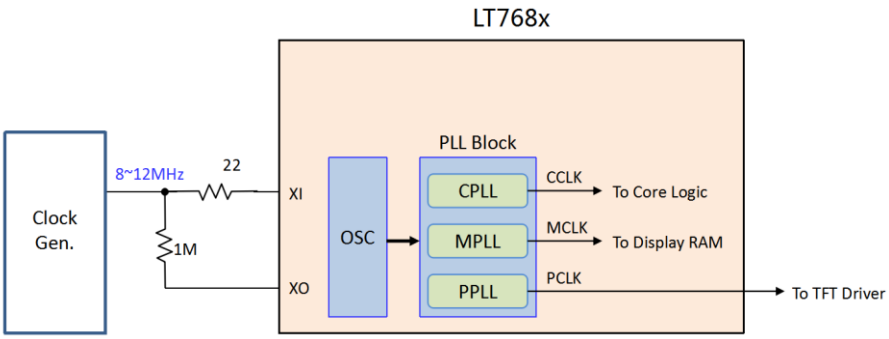


图 4-2：LT768x 时钟电路（二）

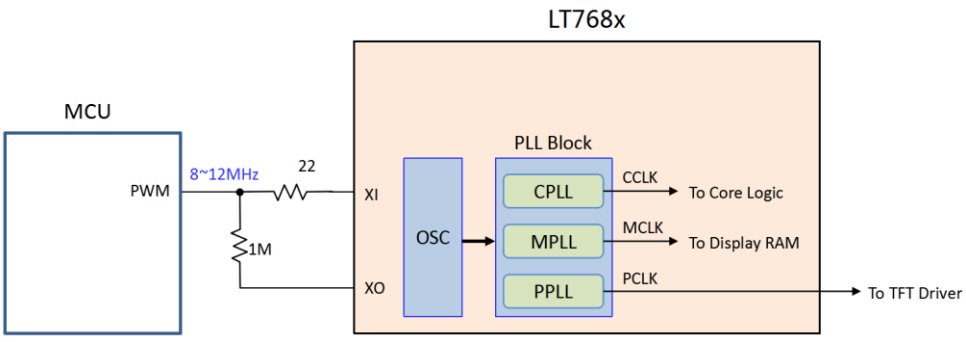


图 4-3：LT768x 时钟电路（三）

3 个 PLL 输出频率都是由 3 组独立的 PLL 寄存器设定，F_{OUT} 为任一组的 PLL 输出频率，其公式为：

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

XI 是外部晶振输入，以上图为例 XI 就是 10MHz，R 是输入除频器比率值（Input Divider Ratio），介于 2 ~ 31 之间，OD 是输出除频器比率值(Output Divider Ratio)，可以设成 1、2 或是 4，N 是 Feedback Divider Ratio of Loop，共 9 个 bits，数值介于 2 ~ 511 之间。

表 4-1：PLL 寄存器设定（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

表 4-2：PLL 寄存器设定（2）

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

例如 XI 是 10MHz，R[4:0] 是 01010（代表 10），N[8:0] 是 100000000（代表 256），OD[1:0] 是 11（代表 4），那么：

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

LT768x 的 3 组时钟信号设计准则为：

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

通常 TFT 屏厂商会依据其 TFT 特性告知最佳显示的 Pixel Clock（PCLK），因此可以依据其要求的 PCLK 完成寄存器设定，及依照上面的准则选定 CCLK 与 MCLK 的频率。根据不同的 LCD 屏，需要设置不同的参数。

注意：

1. 寄存器 R[4:0]的值必须大于寄存器 OD[1:0]。
2. 当屏幕 PCLK 要求小于或等于 8MHz 时，请直接设定 R = 4，OD = 2。

4.2 时钟的启始设定

在 **LT768_Lib.h** 的头文件中修改相关的参数，即可一次将 3 组的时钟设定完成。

```
//分辨率
#define LCD_XSIZE_TFT     1024
#define LCD_YSIZE_TFT     600
//参数
#define LCD_VBPD          20
#define LCD_VFPD          12
#define LCD_VSPW          3
#define LCD_HBPD          140
#define LCD_HFPD          160
#define LCD_HSPW          20
```

而 Pixel Clock（PCLK）是可以根据以上 6 个参数得到，具体的公式如下：

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

当设置好以上的 6 个参数后，直接调用函数：

```
void LT768_PLL_Initial(void)
```

即可设置好 3 组时钟，因为该函数已经根据这 6 个参数，使用该公式计算好这 3 组的时钟的频率了。

5. MCU 接口设定

5.1 MCU 硬件接口

LT768x 提供了 2 种 8 位、16 位的并行接口，以及 SPI、I2C 的串行 MCU 接口，让不同的 MCU 以适合的接口来控制 LT768x。MCU 接口的模式由 PSM[2:0] 引脚来设定，请参考下表设定：

表 5-1：MCU 接口模式设定

PSM[2:0]	MCU 接口模式
0 0 X	选择并口 8 位或 16 位的 8080 模式
0 1 X	选择并口 8 位或 16 位的 6800 模式
1 0 0	选择串口 3 线式 SPI 模式
1 0 1	选择串口 4 线式 SPI 模式
1 1 X	选择串口 I2C 模式

在使用并口模式时，选择 8 位或 16 位的数据传输是由寄存器 REG[01h] 的 bit0 来决定，如果 bit0=0，则设定为 8bit 数据总线，如果 bit0=1，则设定为 16bit 数据总线。

LT768x 系列有不同的封装，支持的 MCU 接口也有所差异，例如 LT7680 为 68pin 的 QFN 封装，只支持串口 3 线 SPI 及 4 线 SPI 模式，下表是 LT768x 系列支持的 MCU 接口对应表：

表 5-2：LT768x 系列支持的 MCU 接口

No.	MCU 接口模式	LT7681 LT7683+ LT7686	LT7680A-R Lt7680B-R
1	并口 8 位的 8080 模式	v	
2	并口 16 位的 8080 模式	v	
3	并口 8 位的 6800 模式	v	
4	并口 16 位的 6800 模式	v	
5	串口 3 线式 SPI 模式	v	v
6	串口 4 线式 SPI 模式	v	v
7	串口 I2C 模式	v	

在 LT7680，PSM[1] 引脚已经在 IC 内部接到地，只引出 PSM[2] 、PSM[0]，使用时 PSM[2] 则必须接到高电位，而由 PSM[0] 来选择 3 线 SPI 或是 4 线式 SPI 模式，当 PSM[0] = 0，选择串口 3 线式 SPI 模式，PSM[0] = 1，则选择 4 线式 SPI 模式。

以下为不同的 MCU 接口参考电路图：

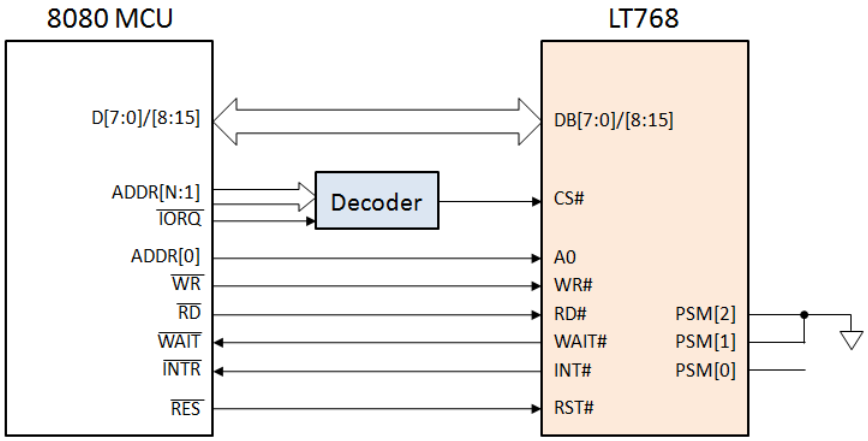


图 5-1：8080 MCU 并口电路图

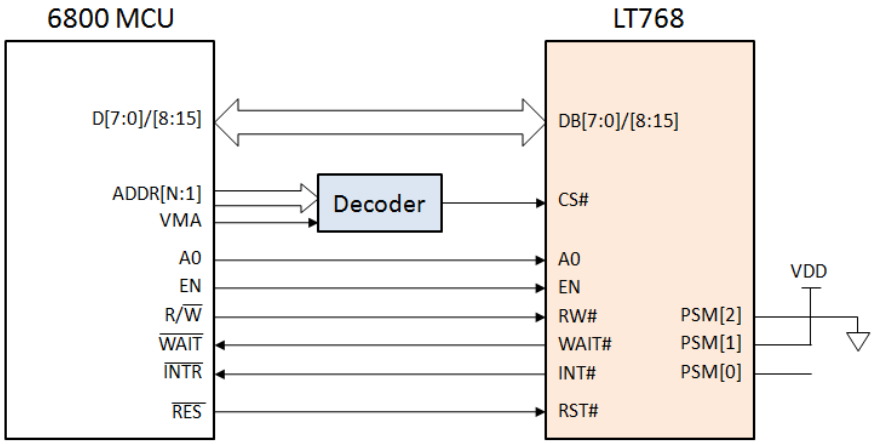


图 5-2：6800 MCU 并口电路图

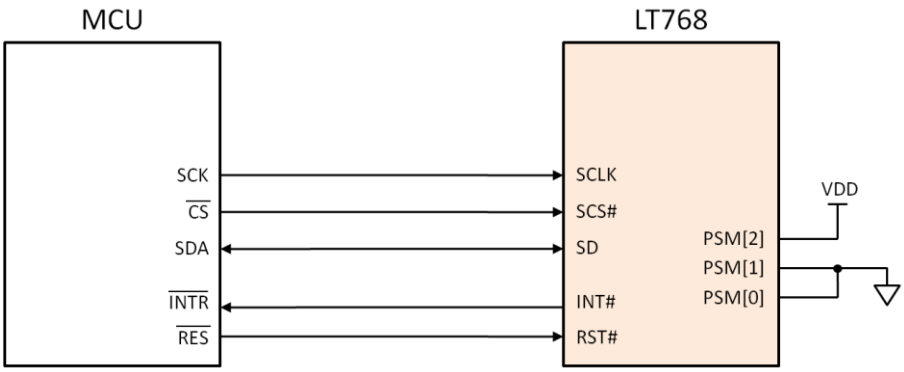


图 5-3：3 线 SPI 串联 MCU 接口电路图

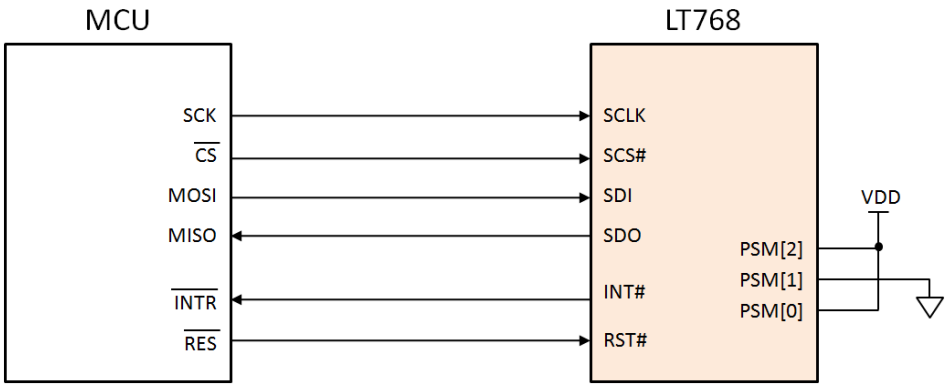


图 5-4：4 线 SPI 串联 MCU 接口电路图

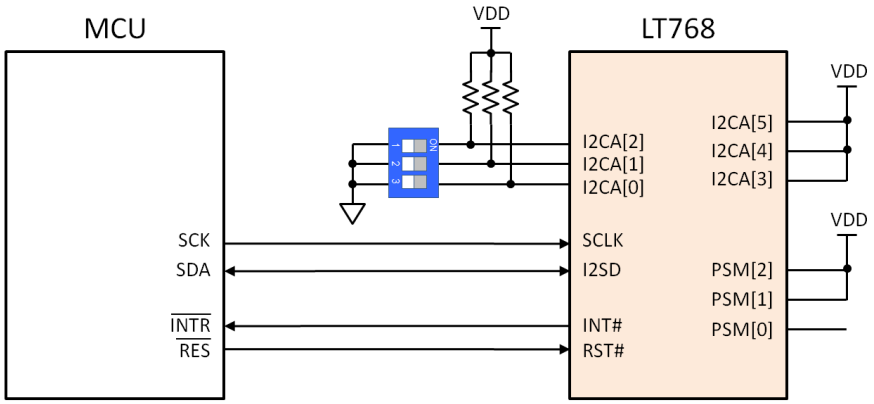


图 5-5：I2C 串联 MCU 接口电路图

由于不同的 MCU 接口无法同时被使用，因此 LT768x 提供了共享的引脚模式，而串口模式中使用较少的引脚，所以其他并口引脚也可以设成 GPIO 使用，请参考下表不同 MCU 模式的接口定义：

表 5-3：不同 MCU 模式的接口定义

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]					接地	SDI	I2C_SDA
DB[5]					SD	SDO	I2CA[5]
DB[4]					SCS#	SCS#	I2CA[4]
DB[3:0]					接地	接地	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 使用 8 位 8080 接口

不管用哪种接口，这些接口主要要实现以下 5 个函数的功能，而 LT768 的其他的功能都是基于以上的 5 个函数来实现的。

1、void LT768_CmdWrite(u8 cmd); // 向 LT768 写命令
2、void LT768_DataWrite(u8 data); // 向 LT768 写数据
3、void LT768_DataWrite_Pixel(u16 data); // 向 LT768 写像素点的数据
4、u8 LT768_StatusRead(void); // 读状态寄存器
5、u16 LT768_DataRead(void); // 读寄存器数据

在使用 8 位的 8080 接口时需要注意，在初始化时一定要把 REG[01h] bit[0] 设为 0（主机总线 8bit），否则像素数据会错乱。

■ **使用 STM32 的 FMSC 来模拟 8 位的 8080 接口：**

```
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 使用 16 位 8080 接口

■ **使用 STM32 的 FMSC 来模拟 16 位的 8080 接口:**

```
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 使用 SPI 接口

```
void SPI_CmdWrite(u8 cmd)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x00);            // 代表 MCU 要写入指令寄存器地址
    SPI2_ReadWriteByte(cmd);
    SS_SET;
}

void SPI_DataWrite(u8 data)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x80);            // 代表 MCU 要写入数据到寄存器或是显示内存中
    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);            // 代表 MCU 要读取状态寄存器的数据
    temp = SPI2_ReadWriteByte(0xff);
    SS_SET;
    return temp;
}
```



```
u16 SPI_DataRead(void)
{
    u16 temp = 0;
    SS_RESET;
    SPI2_ReadWriteByte(0xc0);              // 代表 MCU 要读取指令寄存器的的数据
    temp = SPI2_ReadWriteByte(0xff);
    SS_SET;
    return temp;
}
```

提示：具体的内容可以参考 LT768x 规格书第 2.2 章节的 MCU 串行接口中的 SPI 接口。

5.5 使用 I2C 接口

```
u8 IIC_CmdWrite(u8 cmd)
{
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x00);        // 代表 MCU 要写入指令寄存器地址
    if(IIC_Wait_Ack())                  // 等待应答
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(cmd);
    if(IIC_Wait_Ack())                  // 等待 ACK
    {
        IIC_Stop();
        return 1;
    }
    IIC_Stop();
    return 0;
}

u8 IIC_DataWrite(u8 data)
{
    IIC_Start();
    IIC_Send_Byte(LT_ADDR|0x02);        // 代表 MCU 要写入数据到寄存器或是显示内存中
    if(IIC_Wait_Ack())                  // 等待应答
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(data);
    if(IIC_Wait_Ack())                  // 等待 ACK
    {
        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);          // 代表 MCU 要读取状态寄存器的数据
    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);          // 代表 MCU 要读取指令寄存器的的数据
    IIC_Wait_Ack();
    res=IIC_Read_Byte(0);
    IIC_NAck();
    IIC_Stop();
    return res;
}
```

提示：

1、LT_ADDR 所代表的地址是通过 LT768x 的引脚 I2CA[5:0]设定。

2、具体的内容可以参考数据手册 2.2 章节的 MCU 串行接口中的 IIC 接口。

6. 显示内存（SDRAM）设定

LT768x 有 2 种容量的大小设置（128Mbit），如下表所示：

表 6-1：LT768x 内存容量

型 号	内 建 显 示 内 存	分 辨 率 (最大)
LT7681	128Mb	640*480
LT7683+	128Mb	1024*768
LT7686	128Mb	1280*1024
LT7680A-R	128Mb	1280*1024
LT7680B-R	128Mb	480*320

显示内存的设定方式如下：

```
void LT768_SDRAM_initail(u8 MCLK);
```

举例：如果要使用 128Mbit 容量的显示内存，而且设置的 MCLK 为 100MHz：

```
LT768_SDRAM_initail(100);
```

提示：参数 MCLK 的数值需要和时钟 PLL 的设置一样。对于不同容量的显示内存，其初始化函数也不同。以下为 128Mbit（LT7681/7683+/7686/LT7680x-R）显示内存初始化函数：

```
void LT768_SDRAM_initail(u8 mclk)
{
 unsigned short sdram_itv;
 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);
}
```

7. LCD 界面

7.1 LCD 屏的接口

LT768x 支持 16、18、24bits CMOS 接口面板，不论是 24bpp（RGB 8:8:8）、16bpp（RGB 5:6:5）或者是 8bpp (RGB 3:3:2) 的色度都可以通过这些 CMOS 接口将信号送到 TFT 面板上的驱动器。LT768x 的 LCD 数据线对应到 RGB 的数据如下表所示，MCU 可以通过寄存器 REG[01h] 的 bit[4:3] 去设定 16bits、18bits 或是 24bits，同时不同型号的 LT768x 所支持的 RGB 数据也不同，例如 REG[01h] bit[4:3] = 00b（24bits），而使用 LT7680（只支持 18bits RGB 接口），只能展现 18bits 的 262K 色彩效果。请参考表 7-2 不同型号的 LT768x 所支持的 RGB 数据。

表 7-1：CMOS 接口对应 RGB 的数据

LCD 数据线	数字 TFT-LCD 接口		
	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

表 7-2：不同型号的 LT768x 所支持的 RGB 数据

型 号	LCD 数据线	RGB 接口数	色彩（Max）	说明
LT7681	PD[23~0]	R:G:B = 8:8:8	16.7M 色	支持 16/18/24bits
LT7683+	PD[23~0]	R:G:B = 8:8:8	16.7M 色	支持 16/18/24bits
LT7686	PD[23~0]	R:G:B = 8:8:8	16.7M 色	支持 16/18/24bits
LT7680A-R	PD[23~18], PD[15~10], PD[7~23]	R:G:B = 6:6:6	262K 色	支持 16/18bits
LT7680B-R	PD[23~18], PD[15~10], PD[7~23]	R:G:B = 6:6:6	262K 色	支持 16/18bits

除了 RGB 信号外，LT768x 还提供了 LCD 屏幕扫描时钟信号 PCLK、垂直同步信号 VSYNC、水平同步信号 HSYNC、及屏幕数据使能信号 PDE 到 TFT 显示屏，图 7-1 为 LCD 屏的接口原理图，图 7-2 为 LCD 屏的时序图。

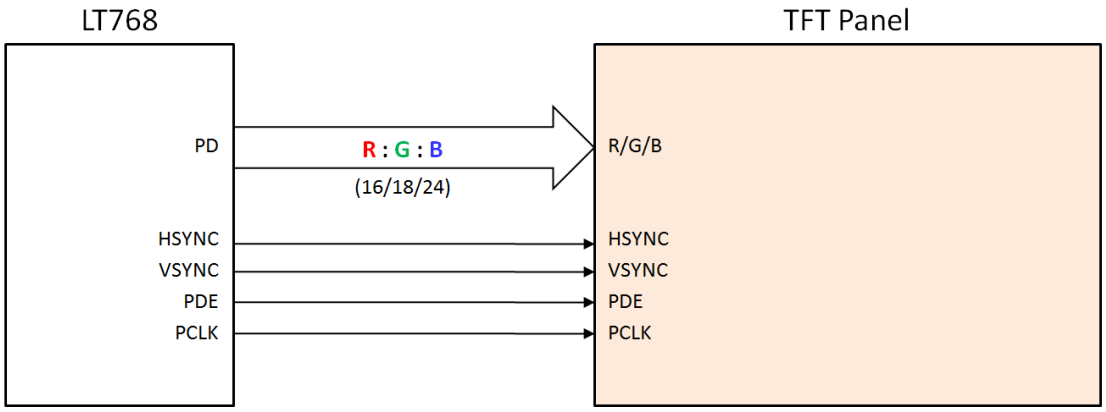


图 7-1：LT768x 与 TFT 屏驱动器的接口

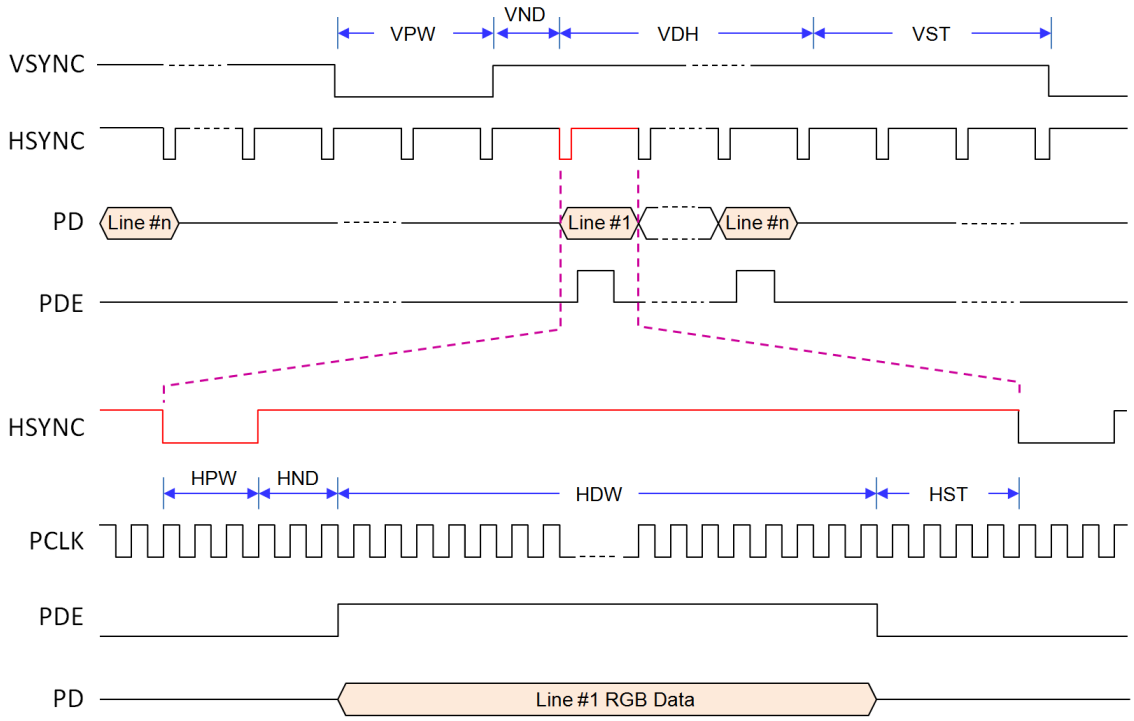


图 7-2：数字 TFT-LCD 接口时序图

7.2 LCD 屏设定

针对不同的 LCD 屏，LT768x 需要设定不同的参数。

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

具体的修改的设置，可以在该函数里修改，比如：

LCD 的扫描方式：

```
VSCAN_T_to_B();          //REG[12h]: 从上到下
//VSCAN_B_to_T();        //从下到上
```

RBG 的输出方式：

```
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 的动作方式：

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

VSYNC 的动作方式：

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

DE 的动作方式：

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

8. 显示功能

8.1 显示视窗（Display Windows）

显示视窗包括了主视窗、底图视窗、工作视窗，三者都是对应到 LT768x 内部显示内存 （Display RAM）的位置，其定义如下：

- 主视窗：用来定义显示内存的一个区域，作为对应到 TFT 屏的显示内容。
- 底图视窗：用来定义显示内存的一个区域，做为 DMA 传输时要写入数据的区域。
- 工作视窗：在主视窗区域中，针对绘制几何图形或文字显示动作所指定的 LCD 工作区域。

8.1.1 主视窗的设定

- **设置主视窗是要以几位的颜色来显示**

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

- **设置主视窗的开始显示地址**

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

举例：假设要以显示内存的 0x800 处来当成主视窗的开始显示地址：

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

- **设置主视窗的宽度**

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

举例：假设要设置的主视窗的宽度位 1024：

```
Main_Image_Width(1024);
```

提示：一般来说，主视窗的宽度要设置成和 LCD 屏幕一样大小。

- **主视窗的起始坐标**

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

8.1.2 底图视窗的设定

底图视窗包括两个方面的设置：

- **底图起始位置的设置**

```
void Canvas_Image_Start_address(unsigned long Addr);
```

- **底图宽度的设置**

```
void Canvas_image_width(unsigned short WX);
```

举例：设置一张起始位置为 0x8000，而且宽度为 480 的一张底图：

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

8.1.3 工作视窗的设置

工作视窗包括两个方面的设置：

- **工作视窗起始位置的设置**

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

- **工作视窗大小的设置**

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

举例：设置一个起始位置为(0, 0)，大小为(1024, 600)的工作视窗：

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

综上所述：只有设置了以上 3 个视窗才可以在 LCD 中正常的显示出写入的数据。

8.2 MCU 写入数据到内存

由于 MCU 驱动 LT768x 有 8 位和 16 位，而且 LT768 也可以显示多种色深的图片，所以 MCU 写入数据到内存的方式就有很多种，具体需要用到哪种可以根据自己的设定。

void MPU8_8bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);
//MCU 使用 8 位数据驱动，写到内存，而且用 8 位色深来显示

void MPU8_16bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);
// MCU 使用 8 位数据驱动，写到内存，而且用 16 位色深来显示

void MPU8_24bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned char *data);
// MCU 使用 8 位数据驱动，写到内存，而且用 24 位色深来显示

void MPU16_16bpp_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);
// MCU 使用 16 位数据驱动，写到内存，而且用 16 位色深来显示

void MPU16_24bpp_Mode1_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);
// MCU 使用 16 位数据驱动，写到内存，而且用 24 位色深来显示（模式 1）

void MPU16_24bpp_Mode2_Memory_Write(unsigned short x, unsigned short y, unsigned short w, unsigned short h, const unsigned short *data);
// MCU 使用 16 位数据驱动，写到内存，而且用 24 位色深来显示（模式 2）

8.3 主视窗中显示图片

综上所述：以下的 2 个范例说明如何在主视窗中显示所给定的内容。假设：

1、LCD 的屏为 1024*600 的分辨率
2、所要显示的图片的大小也为 1024*600，且色深为 16bit。
3、图片的数据存在数组 picture_data[]中。

范例一：要在显示内存的 0 地址中写入一张大小为 1024*600、色深为 16bit 的图片，并且在屏上显示出来。

1. 设置主视窗的色深。

```
Select_Main_Window_16bpp();          // 16bit 的深度
```

2. 、设置主视窗的起始位置和宽度。

```
Main_Image_Start_Address(0);         // 从显示的 0 地址起开始映像到主视窗图层中
Main_Image_Width(1024);              // 主视窗的宽度
```

3. 主视窗的起始坐标

```
Main_Window_Start_XY(0, 0);          // 主视窗从(0, 0)地址开始
```

4. 设置底图的起始位置和宽度

```
Canvas_Image_Start_address(0);       // 从底图（显示内存）的 0 地址开始写数据
Canvas_image_width(1024);            // 底图的宽度
```

5. 设置工作视窗的起始坐标和大小

```
Active_Window_XY(0, 0);              // LCD 从主视窗的(0, 0)地址开始显示
Active_Window_WH(1024, 600);         // LCD 显示的宽为 1024，长为 600
```

6. MCU 向内存中写入图片的数据

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

范例二：使用 DMA 的方式从 LT768 的 SFCS0# 中外挂的 Flash 的 0 地址中读取一张 1024*600、色深为 16bit 的图片到显示的 0 地址中（前提 Flash 中已有图片数据），并显示出来。

1. 设置主视窗的色深。

```
Select_Main_Window_16bpp();          // 16bit 的深度
```

2. 设置主视窗的起始位置和宽度。

```
Main_Image_Start_Address(0);         // 从显示的 0 地址起开始映像到主视窗图层中
Main_Image_Width(1024);              // 主视窗的宽度
```

3. 主视窗的起始坐标

```
Main_Window_Start_XY(0, 0);          // 主视窗从(0, 0)地址开始
```

4. 设置底图的起始位置和宽度

```
Canvas_Image_Start_address(0);       // 从底图（显示内存）的 0 地址开始写数据
Canvas_image_width(1024);            // 底图的宽度
```

5. 设置工作视窗的起始坐标和大小

```
Active_Window_XY(0, 0);              // LCD 从主视窗的(0, 0)地址开始显示
Active_Window_WH(1024, 600);         // LCD 显示的宽为 1024，长为 600
```

6. MCU 向内存中写入图片的数据

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

范例二的前 5 个步骤和范例一都完全相同，只是第 6 个步骤的函数不一样，该函数的使用具体请看第 15.1.2 节。

8.4 画中画（Picture-In-Picture, PIP）

■ 画中画（PIP）视窗的设定

```
void LT768_PIP_Init
(
unsigned char On_Off,              // 0：禁止 PIP；1：使能 PIP；2：保持原来的状态
unsigned char Select_PIP,          // 1：使用 PIP1；2：使用 PIP2
unsigned long PAddr,               // PIP 的开始地址
unsigned short XP,                 // PIP 窗口的 X 坐标，必须被 4 整除
unsigned short YP,                 // PIP 窗口的 Y 坐标，必须被 4 整除
unsigned long ImageWidth,          // 底图的宽度
unsigned short X_Dis,              // 显示窗口的 X 坐标
unsigned short Y_Dis,              // 显示窗口的 Y 坐标
unsigned short X_W,                // 显示窗口的宽度，必须被 4 整除
unsigned short Y_H,                // 显示窗口的长度，必须被 4 整除
)
```

举例：要在主视图上显示一个 300*300 的 PIP1 的图像。

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

■ 画中画视窗显示位置与图像位置

```
void LT768_Set_DisWindowPos
(
unsigned char On_Off,              // 0：禁止 PIP；1：使能 PIP；2：保持原来的状态
unsigned char Select_PIP,          // 1：使用 PIP1；2：使用 PIP2
unsigned short X_Dis,              // 显示窗口的 X 坐标
unsigned short Y_Di                // 显示窗口的 Y 坐标
)
```

该函数可以通过修改坐标来改变 PIP 的位置。

■ PIP Disable?

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

8.5 旋转与镜像

LT768x 支持从 MCU 写到显示内存的数据进行旋转与镜像的功能，但如果是用 DMA 从 Flash 传到显示内存的数据则不支持此功能，因此在 MCU 传送数据之前需要先做好功能的设定。

寄存器 REG[02h] bit[2:1] 提供主控端写入的内存方向控制（另提供给绘图模式）：

```
void MemWrite_Left_Right_Top_Down(void);      // REG[02h] bit[2:1]=00b：左→右
                                              // 然后 上→下（初始值）
void MemWrite_Right_Left_Top_Down(void);      // REG[02h] bit[2:1]=01b：右→左
                                              // 然后 上→下（水平翻转）
void MemWrite_Top_Down_Left_Right(void);      // REG[02h] bit[2:1]=10b：上→下
                                              // 然后 左→右（向右旋转 90°并且水平翻转）
void MemWrite_Down_Top_Left_Right(void);      // REG[02h] bit[2:1]=11b：下→上
                                              // 然后 左→右（向左旋转 90°）
```

寄存器 REG[12h] bit3（VDIR）用来控制 LCD 的垂直扫描方向：

```
void VSCAN_T_to_B(void);      // REG[12h] bit3=0：垂直扫描方向由上到下
void VSCAN_B_to_T(void);      // REG[12h] bit3=1：垂直扫描方向由下到上
```

因此，旋转与镜像就有 8 种的组合方式：

```
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 彩条（Color Bar）显示

LT768x 提供彩条（Color Bar）显示功能，可以做为显示测试使用，同时使用上并不需要用到显示内存。

```
void LT768_Color_Bar_ON(void);      //显示彩条
void LT768_Color_Bar_OFF(void);     //关闭彩条
```

图 8-1：彩条（Color Bar）显示

9. 几何绘图

9.1 画线

9.1.1 画细线

该函数只能画一条线，不能设置线宽。

```
void LT768_DrawLine
(
unsigned short X1,          // X1 坐标
unsigned short Y1,          // Y1 坐标
unsigned short X2,          // X2 坐标
unsigned short Y2,          // Y2 坐标
unsigned long LineColor     // 线段颜色
)
```

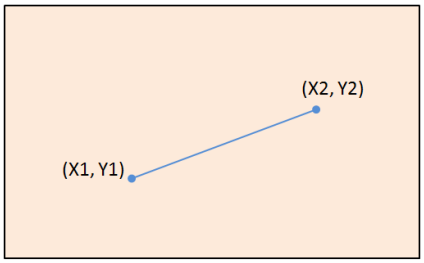


图 9-1：画细线

举例：画一条颜色为红色，且起点为（100, 100），终点为（200, 200）的线段。

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

9.1.2 画粗线

该函数能画一条线，且能设置线宽。

```
void LT768_DrawLine_Width
(
unsigned short X1,          // X1 坐标
unsigned short Y1,          // Y1 坐标
unsigned short X2,          // X2 坐标
unsigned short Y2,          // Y2 坐标
unsigned long LineColor,    // 线段颜色
unsigned short Width        // 线段宽度
)
```

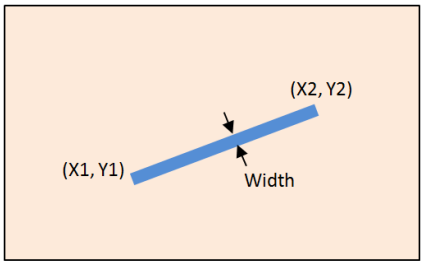


图 9-2：画粗线

举例：画一条线宽为 10 的红色线段，且起点为（120, 140），终点为（220, 260）。

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

9.2 画圆形

9.2.1 画空心圆形

```
void LT768_DrawCircle
(
unsigned short XCenter,          // 圆心 X 位置
unsigned short YCenter,          // 圆心 Y 位置
unsigned short R,                // 半径
unsigned long CircleColor        // 线颜色
)
```

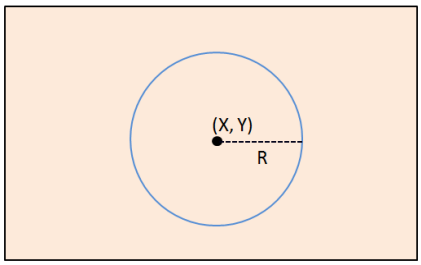


图 9-3：画实心圆形

举例：画一个圆心为坐标（200, 200)、半径 100 的红色空心圆。

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

9.2.2 画实心圆形

```
void LT768_DrawCircle_Fill
(
unsigned short XCenter,          // 圆心 X 位置
unsigned short YCenter,          // 圆心 Y 位置
unsigned short R,                // 半径
unsigned long ForegroundColor    // 前景颜色
)
```

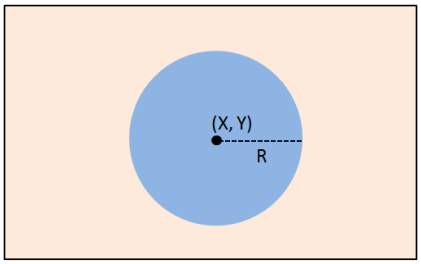


图 9-4：画实心圆形

举例：画一个圆心为坐标（200, 200)、半径 100 的红色实心圆。

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

9.2.3 画带框实心圆形

```
void LT768_DrawCircle_Width
(
unsigned short XCenter,          // 圆心 X 位置
unsigned short YCenter,          // 圆心 Y 位置
unsigned short R,                // 半径
unsigned long CircleColor,       // 外框颜色
unsigned long ForegroundColor,   // 前景颜色
unsigned short Width             // 外框宽度
)
```

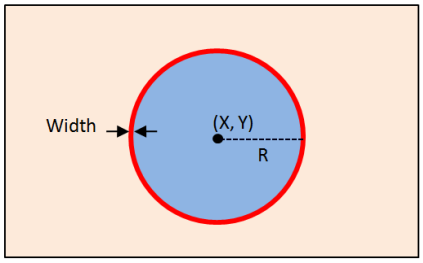


图 9-5：画带框实心圆形

举例：画一个圆心为坐标（200, 200)、半径 100、线宽 10 的的红色空心圆，且底色前景颜色为白色。

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

提示：该函数的线宽是两个实心圆的叠加，其中 CircleColor 是外部大的实心圆前景色，而 ForegroundColor 是内部小的实心圆前景色。

9.3 画椭圆形

9.3.1 画空心椭圆形

```
void LT768_DrawEllipse
(
unsigned short XCenter,          // 椭圆心 X 位置
unsigned short YCenter,          // 椭圆心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long EllipseColor       // 画线颜色
)
```

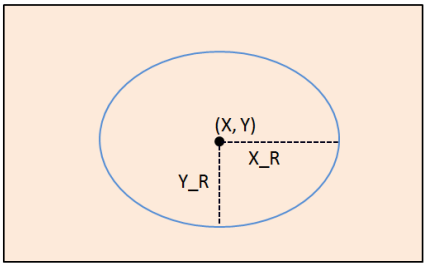


图 9-6：画空心椭圆形

举例：画一个椭圆心(100, 100)，长宽半径各为 80、50 的红色空心椭圆。

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

9.3.2 画实心椭圆形

```
void LT768_DrawEllipse_Fill
(
unsigned short XCenter,          // 椭圆心 X 位置
unsigned short YCenter,          // 椭圆心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long ForegroundColor    // 前景颜色
)
```

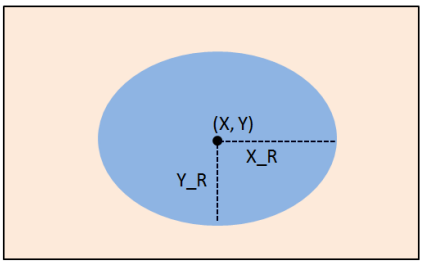


图 9-7：画实心椭圆形

举例：画一个椭圆心(100, 100)，长宽半径各为 80、50 的红色实心椭圆。

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

9.3.3 画带框实心椭圆形

```
void LT768_DrawEllipse_Width
(
unsigned short XCenter,          // 椭圆心 X 位置
unsigned short YCenter,          // 椭圆心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long EllipseColor,      // 外框颜色
unsigned long ForegroundColor,   // 前景颜色
unsigned short Width             // 外框宽度
)
```

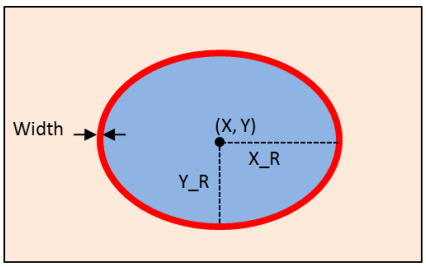


图 9-8：画带框实心椭圆形

举例：画一个椭圆心(100, 100)，长宽半径各为 80、50，线宽为 10 的红色空心椭圆，且前景色为白色：

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

提示：该函数的线宽是两个实心椭圆的叠加，其中 EllipseColor 是外部大的实心椭圆前景色，而 ForegroundColor 是内部小的实心椭圆前景色。

9.4 画矩形

9.4.1 画空心矩形

```
void LT768_DrawSquare
(
unsigned short X1,              // X1 位置
unsigned short Y1,              // Y1 位置
unsigned short X2,              // X2 位置
unsigned short Y2,              // Y2 位置
unsigned long SquareColor       // 画线颜色
)
```

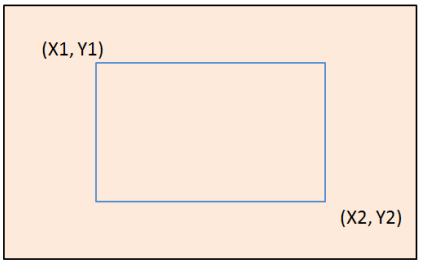


图 9-9：画空心矩形

举例：画一个起点(50, 60)、终点(200, 150)的红色空心矩形。

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

9.4.2 画实心矩形

```
void LT768_DrawSquare_Fill
(
unsigned short X1,              // X1 位置
unsigned short Y1,              // Y1 位置
unsigned short X2,              // X2 位置
unsigned short Y2,              // Y2 位置
unsigned long ForegroundColor   // 前景颜色
)
```

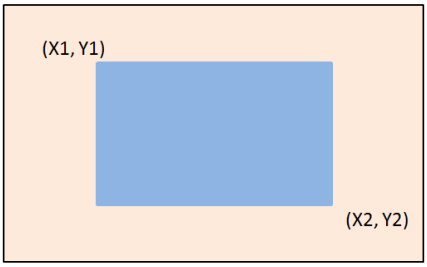


图 9-10：画实心矩形

举例：画一个起点(50,60)、终点(200,150)的红色实心矩形。

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

9.4.3 画带框实心矩形

```
void LT768_DrawSquare_Width
(
unsigned short X1,              // X1 位置
unsigned short Y1,              // Y1 位置
unsigned short X2,              // X2 位置
unsigned short Y2,              // Y2 位置
unsigned long SquareColor,      // 外框颜色
unsigned long ForegroundColor,  // 前景颜色
unsigned short Width            // 外框宽度
)
```

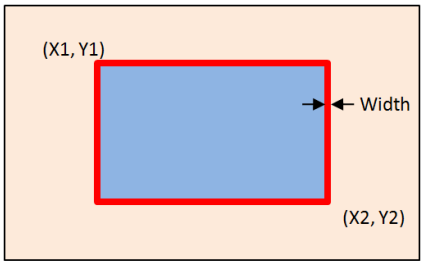


图 9-11：画带框实心矩形

举例：画一个起点(50,60)、终点(200,150)、线宽 10 的红色空心矩形，且前景为白色。

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

提示：该函数的线宽是两个实心矩形的叠加，其中 SquareColor 是外部大的实心矩形前景色，而 ForegroundColor 是内部小的实心矩形前景色。

9.5 画圆角矩形

9.5.1 画空心圆角矩形

```
void LT768_DrawCircleSquare
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X_R,                 // 宽半径
unsigned short Y_R,                 // 长半径
unsigned long CircleSquareColor     // 画线颜色
)
```

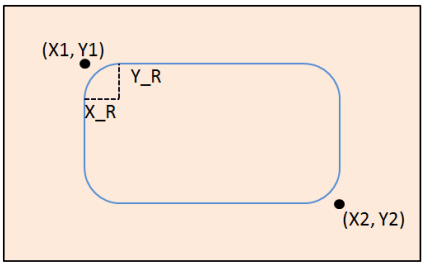


图 9-12：画空心圆角矩形

举例：画一个起点(50,60)、终点(200,150)、长宽半径各为 30、20 的红色空心圆角矩形：

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

9.5.2 画实心圆角矩形

```
void LT768_DrawCircleSquare_Fill
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X_R,                 // 宽半径
unsigned short Y_R,                 // 长半径
unsigned long ForegroundColor       // 前景颜色
)
```

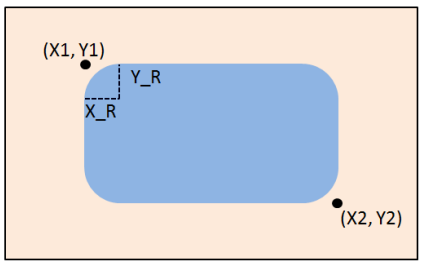


图 9-13：画实心圆角矩形

举例：画一个起点(50,60)、终点(200,150)、长宽半径各为 30、20 的红色实心圆角矩形：

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

9.5.3 画带框实心圆角矩形

```
void LT768_DrawCircleSquare_Width
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X_R,                 // 宽半径
unsigned short Y_R,                 // 长半径
unsigned long CircleSquareColor,    // 外框颜色
unsigned long ForegroundColor,      // 前景颜色
unsigned short Width                // 外框宽度
)
```

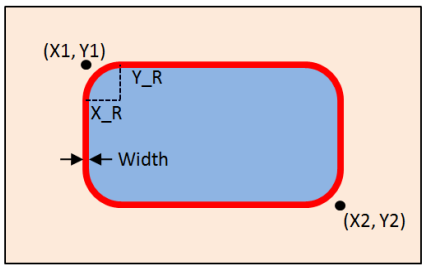


图 9-14：画带框实心圆角矩形

举例：画一个起点(50,60)、终点(200,150)、长宽半径各为 30、20，线宽为 10 的红色空心圆角矩形，且前景为白色：

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

提示：该函数的线宽是两个实心矩形的叠加，其中 CircleSquareColor 是外部大的实心圆角矩形前景色，而 ForegroundColor 是内部小的实心圆角矩形前景色。

9.6 画三角形

9.6.1 画空心三角形

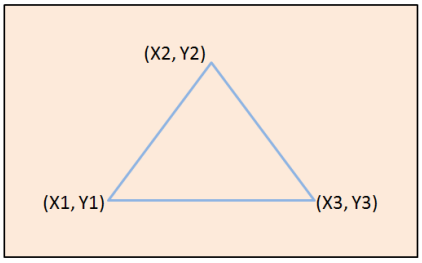


图 9-15：画空心三角形

```
void LT768_DrawTriangle
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned long TriangleColor         // 画线颜色
)
```

举例：画一个三点的坐标分别为(100,100)、(50,200)、(150,150) 的红色空心三角形：

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

9.6.2 画实心三角形

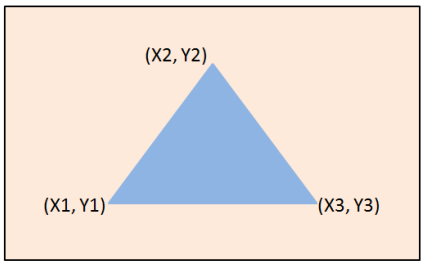


图 9-16：画实心三角形

```
void LT768_DrawTriangle_Fill
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned long ForegroundColor       // 前景颜色
)
```

举例：画一个三点的坐标分别为(100, 100)、(50, 200)、(150, 150) 的红色实心三角形：

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

9.6.3 画带框实心三角形

```
void LT768_DrawTriangle_Frame
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned long TriangleColor,        // 画线颜色
unsigned long ForegroundColor       // 前景颜色
)
```

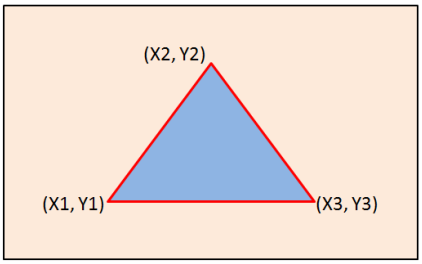


图 9-17：画带框实心三角形

9.7 画曲线

9.7.1 左上方曲线

```
void LT768_DrawLeftUpCurve
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long CurveColor         // 画线颜色
)
```

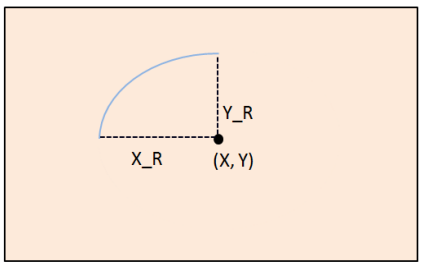


图 9-18：左上方曲线

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色左上方曲线：

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

9.7.2 左下方曲线

```
void LT768_DrawLeftDownCurve
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long CurveColor         // 画线颜色
)
```

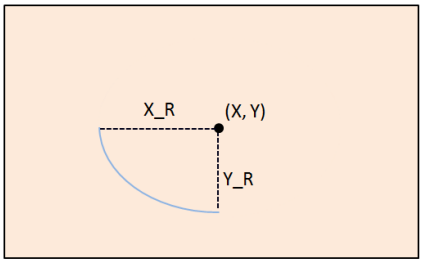


图 9-19：左下方曲线

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色左下方曲线：

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

9.7.3 右上方曲线

```
void LT768_DrawRightUpCurve
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long CurveColor         // 画线颜色
)
```

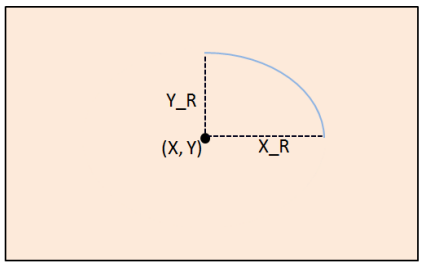


图 9-20：右上方曲线

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色右上方曲线：

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

9.7.4 右下方曲线

```
void LT768_DrawRightDownCurve
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long CurveColor         // 画线颜色
)
```

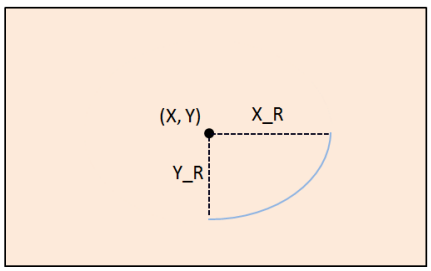


图 9-21：右下方曲线

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色右下方曲线：

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

9.8 画 1/4 椭圆形

9.8.1 左上方 1/4 椭圆

```
void LT768_DrawLeftUpCurve_Fill
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long ForegroundColor    // 前景颜色
)
```

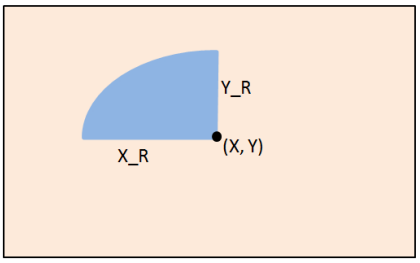


图 9-22：左上方 1/4 椭圆

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色左上方 1/4 椭圆：

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

9.8.2 左下方 1/4 椭圆

```
void LT768_DrawLeftDownCurve_Fill
(
unsigned short XCenter,          // 曲心 X 位置
unsigned short YCenter,          // 曲心 Y 位置
unsigned short X_R,              // 宽半径
unsigned short Y_R,              // 长半径
unsigned long ForegroundColor    // 前景颜色
)
```

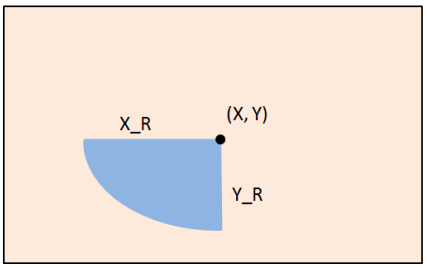


图 9-23：左下方 1/4 椭圆

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色左下方 1/4 椭圆：

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

9.8.3 右上方 1/4 椭圆

```
void LT768_DrawRightUpCurve_Fill
(
unsigned short XCenter,              // 曲心 X 位置
unsigned short YCenter,              // 曲心 Y 位置
unsigned short X_R,                  // 宽半径
unsigned short Y_R,                  // 长半径
unsigned long ForegroundColor        // 前景颜色
)
```

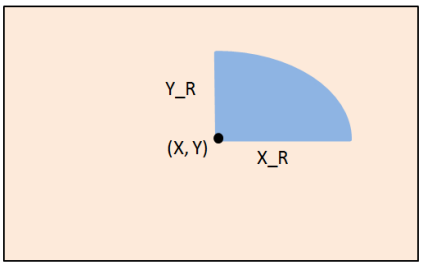


图 9-24：右上方 1/4 椭圆

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色右上方 1/4 椭圆：

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

9.8.4 右下方 1/4 椭圆

```
void LT768_DrawRightDownCurve_Fill
(
unsigned short XCenter,              // 曲心 X 位置
unsigned short YCenter,              // 曲心 Y 位置
unsigned short X_R,                  // 宽半径
unsigned short Y_R,                  // 长半径
unsigned long ForegroundColor        // 前景颜色
)
```

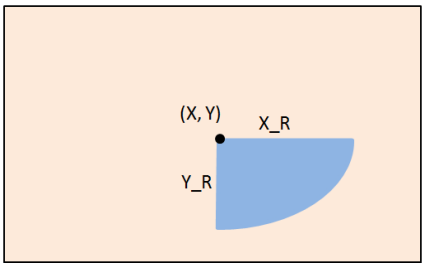


图 9-25：右下方 1/4 椭圆

举例：画一个曲心(100, 100)、长宽半径各为 100、70 的红色右下方 1/4 椭圆：

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

9.9 画四边形

9.9.1 画空心四边形

```
void LT768_DrawQuadrilateral
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned short X4,                  // X4 位置
unsigned short Y4,                  // Y4 位置
unsigned long ForegroundColor       // 画线颜色
)
```

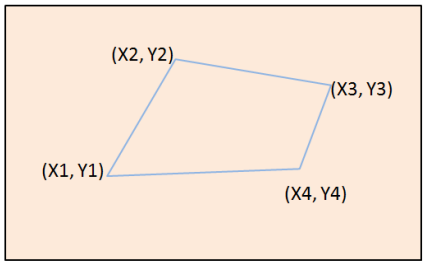


图 9-26：画空心四边形

举例：画一个 4 点分别为(50, 50)、(200, 80)、(150, 130)、(60, 100)的红色空心四边形：

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

四边形于矩形的区别：四边形可以任意设置四个点的坐标，不一定会是矩形，而矩形只需设置两个点的坐标。

9.9.2 画实心四边形

```
void LT768_DrawQuadrilateral_Fill
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned short X4,                  // X4 位置
unsigned short Y4,                  // Y4 位置
unsigned long ForegroundColor       // 前景颜色
)
```

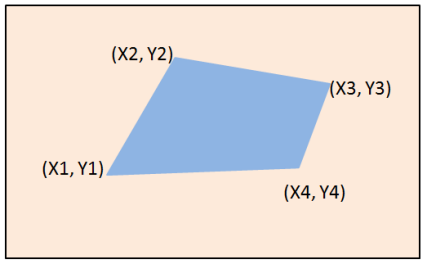


图 9-27：画实心四边形

举例：画一个 4 点分别为(50, 50)、(200, 80)、(150, 130)、(60, 100)的红色实心四边形：

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

9.10 五边形

9.10.1 画空心五边形

```
void LT768_DrawPentagon
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned short X4,                  // X4 位置
unsigned short Y4,                  // Y4 位置
unsigned short X5,                  // X5 位置
unsigned short Y5,                  // Y5 位置
unsigned long ForegroundColor       // 画线颜色
)
```

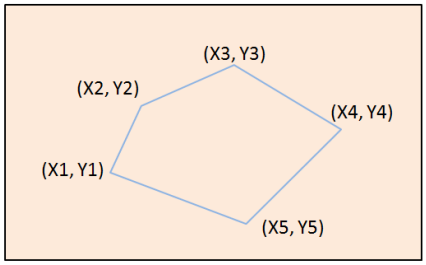


图 9-28：画空心五边形

举例：画一个 5 点分别为(50, 100)、(120, 130)、(150, 160)、(100, 180)、(80, 140) 的红色空心五边形。

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

9.10.2 画实心五边形

```
void LT768_DrawPentagon_Fill
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // Y1 位置
unsigned short X2,                  // X2 位置
unsigned short Y2,                  // Y2 位置
unsigned short X3,                  // X3 位置
unsigned short Y3,                  // Y3 位置
unsigned short X4,                  // X4 位置
unsigned short Y4,                  // Y4 位置
unsigned short X5,                  // X5 位置
unsigned short Y5,                  // Y5 位置
unsigned long ForegroundColor       // 前景颜色
)
```

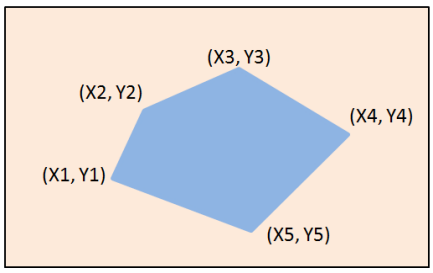


图 9-29：画实心五边形

举例：画一个 5 点分别为(50, 100)、(120, 130)、(150, 160)、(100, 180)、(80, 140)的红色实心五边形：

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

9.11 圆柱体

```
unsigned char LT768_DrawCylinder
(
unsigned short XCenter,              // 椭圆心 X 位置
unsigned short YCenter,              // 椭圆心 Y 位置
unsigned short X_R,                  // 宽半径
unsigned short Y_R,                  // 长半径
unsigned short H,                    // 高度
unsigned long CylinderColor,         // 外框颜色
unsigned long ForegroundColor        // 前景颜色
)
```

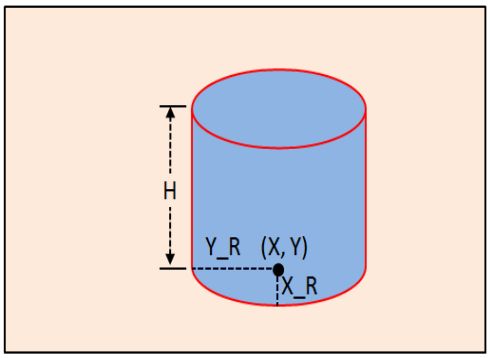


图 9-30：圆柱体

该圆柱体的的实现方法，主要是整合了以上的的一些硬件画图功能而成的。

举例：画一个椭圆心（200，300），长宽半径各为 100、50，且高为 150，画线颜色为红色，前景颜色为蓝色的圆柱体。

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

提示：该函数中椭圆的参数都是针对底部的椭圆。

9.12 方柱体

```
void LT768_DrawQuadrangular
(
unsigned short X1,              // 顶面左下角 X 位置
unsigned short Y1,              // 顶面左下角 Y 位置
unsigned short X2,              // 顶面左上角 X 位置
unsigned short Y2,              // 顶面左上角 Y 位置
unsigned short W,               // 宽度
unsigned short H,               // 高度
unsigned long   QuadrangularColor,    // 外框颜色
unsigned long   ForegroundColor       // 前景颜色
)
```

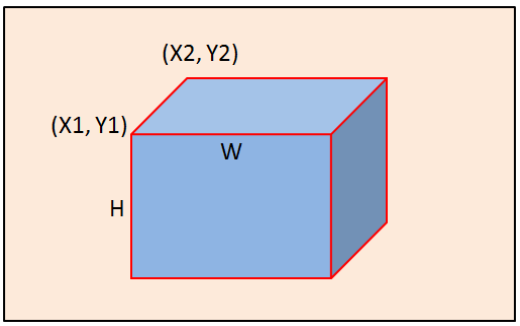


图 9-31：方柱体

该圆柱体的的实现方法，主要是整合了以上的的一些硬件画图功能而成的。

举例：画一个六个点分别为（100，100）、（140，60）、宽度为 150，且高为 200，且画线颜色为红色，前景颜色为蓝色的方柱体。

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

9.13 表格视窗

```
void LT768_MakeTable
(
unsigned short X1,                  // 起始位置 X1
unsigned short Y1,                  // 起始位置 Y1
unsigned short W,                   // 宽度
unsigned short H,                   // 高度
unsigned short Line,                // 行数 CN
unsigned short Row,                 // 列数 RN
unsigned long   TableColor,         // 线框颜色 C1
unsigned long   ItemColor,          // 项目拦背景色 C2
unsigned long   ForegroundColor,    // 内部窗口背景色 C3
unsigned short width1,              // 内框宽度
unsigned short width2,              // 外框宽度
unsigned char mode                  // 0：项目栏纵向    1：项目栏横向
)
```

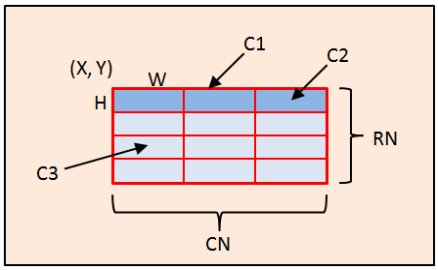
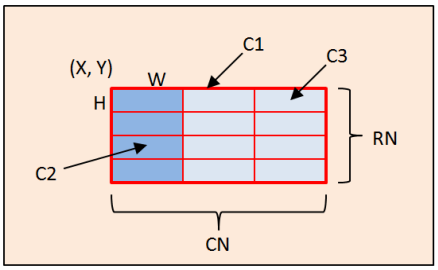


图 9-32：横向表格视窗图



9-33：纵向表格视窗

该表格视窗的的实现方法，主要也是整合了以上的的一些硬件画图功能而成的。

举例：画一个项目栏横向的表格视窗在（100, 100）位置，单一表格窗口大小为 80*40，表格行数为 3，列数为 4，画线颜色为红色，项目拦表格背景为绿色，其他表格为蓝色，内框宽度为 1，外框宽度为 3。

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

10. 区块传输引擎（BTE）

10.1 BTE 操作模式

LT768x 内建 2D 区块传输引擎（BTE），在指定区块数据结合某些逻辑传输操作中，LT768x 内的 BTE 硬件可以提升传输速度，简化 MCU 的程序。在使用 BTE 功能之前，使用者必须选择指定的 BTE 操作模式，如表 10-1。LT768x 同時支持 16 种光栅操作（Raster Operation，简称 ROP），来源端与目的端可以进行多样的 ROP 组合，经由组合 BTE 功能的光栅操作，使用者可以达到不同的应用。

表 10-1：BTE 的操作模式

BTE 操作模式 REG[91h] Bits [3:0]	BTE 操 作 说 明
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

MCU 可以使用检查 BTE 忙碌信号与硬件中断来确认 BTE 执行状况。如果使用者要读取 BTE 状态可以由 BTE_CTRL0（REG[90h]）bit4 或是状态寄存器（STSR）bit3 得到。另一种方法，使用者可以检查硬件中断，当有硬件中断 INT#产生时，可以去中断旗标寄存器 REG[0Ch] 确认中断来源，而硬件线路上 INT#中断信号必须要连接 MCU 的中断输入脚。

表 10-2：光栅操作模式（ROP Function）

ROP Bits REG[91h] bit[7:4]	Boolean Function Operation (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）

上表中，S0 代表来源 0 的数据，S1 代表来源 1 的数据，DT 代表目的端的数据。例如 ROP 功能设定为 0010b，那么目的端数据 DT = ~S0・S1；如果 ROP 功能设定为 1010b，那么目的端数据 DT = 来源 1 的数据（DT = S1）；如果 ROP 功能设定为 1100b，那么目的端数据 DT = 来源 0 的数据（DT = S0）；如果 ROP 功能设定为 1110b，那么目的端数据 DT = S0 + S1。

10.2 BTE 功能详述

10.2.1 结合光栅操作的 BTE 写入

此功能可以增加 MCU 写入 SDRAM 的速度，写入的数据可以结合光栅（ROP）操作填入目的内存中。BTE 本身提供 16 种 ROP，由下图可以得知来源 0（S0）必须由 MCU 提供，此范例是当 POP 寄存器（REG[91h]）bit[7:4] 设成 0x0C 得到的结果。

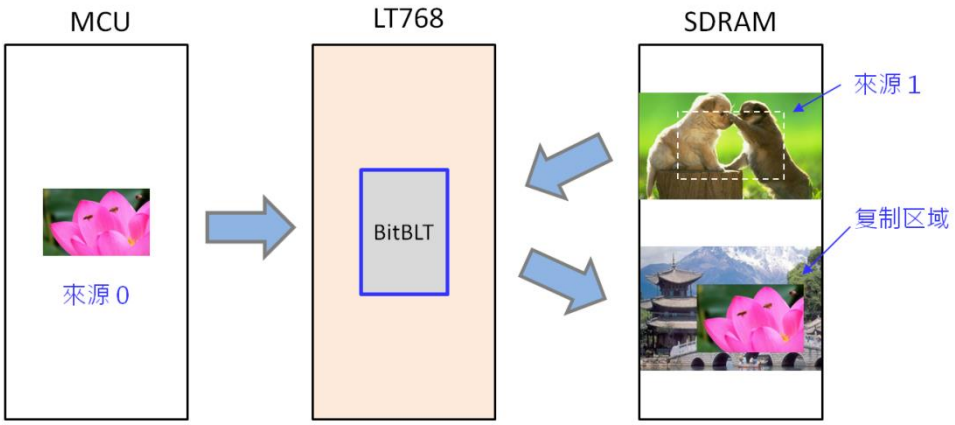


图 10-1：结合光栅操作的 BTE 写入范例

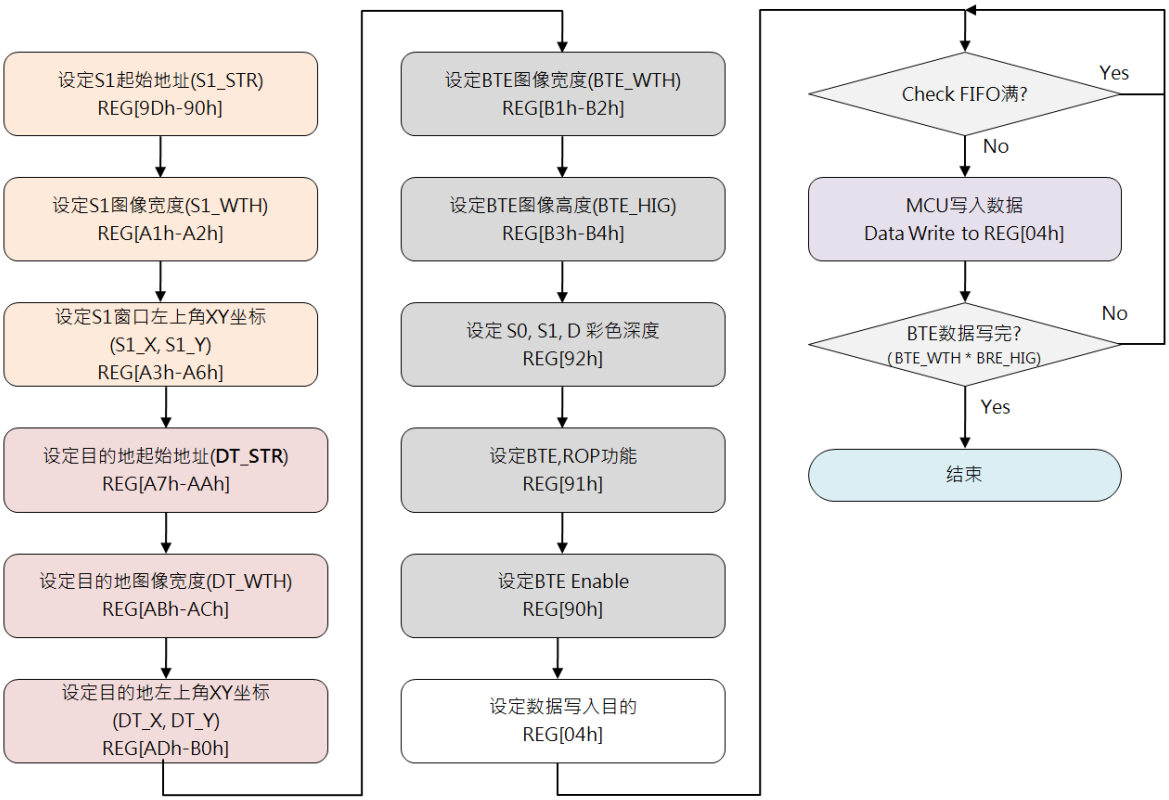


图 10-2：结合光栅操作的 BTE 写入流程图

```
void LT768_BTE_MCU_Write_MCU_16bit
(
unsigned long S1_Addr,                  // S1 图像的内存起始地址
unsigned short S1_W,                    // S1 图像的宽度
unsigned short XS1,                     // S1 图像的左上方 X 坐标
unsigned short YS1,                     // S1 图像的左上方 Y 坐标
unsigned long Des_Addr,                 // 目的图像的内存起始地址
unsigned short Des_W,                   // 目的图像的总宽度
unsigned short XDes,                    // 目的图像的左上方 X 坐标
unsigned short YDes,                    // 目的图像的左上方 Y 坐标
unsigned int ROP_Code,                  // 光栅操作模式
unsigned short X_W,                     // 目的图像的宽度
unsigned short Y_H,                     // 目的图像的长度
const unsigned short *data              // S0 数据的起始地址
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：MCU 写入的一张 16 色的 100*100 的图像（unsigned short Picture_Data[100*100]）
来源 1：内存 1024*600*2 地址起的（50, 50）坐标
目的图像的内存地址及位置：内存 0 地址起的（200, 200）坐标
目的图像的大小：100*100
光栅的操作模式：0x0C（显示来源 0）

实现函数：

```
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 结合光栅操作的 BTE 内存复制

这个功能将会从指定的内存来源区域复制搬移至指定的内存目的区域。这个操作可以减少 MCU 处理时间，进而提升内存数据复制搬移的速度。下图范例是当 POP 寄存器（REG[91h]）bit[7:4] 设成 0x0C 得到的结果。

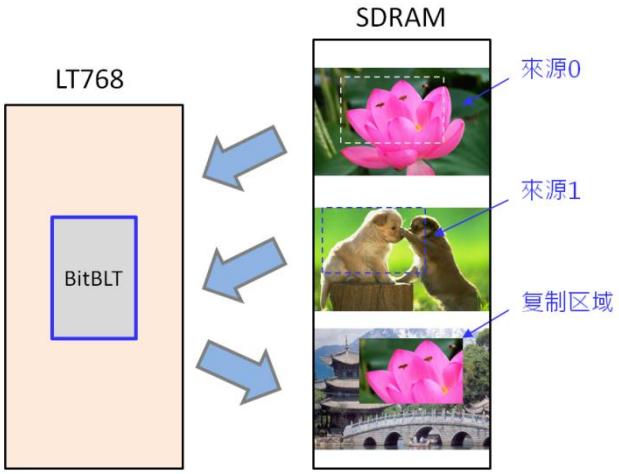


图 10-3：结合光栅操作的 BTE 内存复制范例

图 10-4 是此功能之流程图，MCU 是以检查 BTE 忙碌信号来确认 BTE 执行状况。图 10-5 之流程图，MCU 是以检查硬件中断来确认 BTE 执行状况。

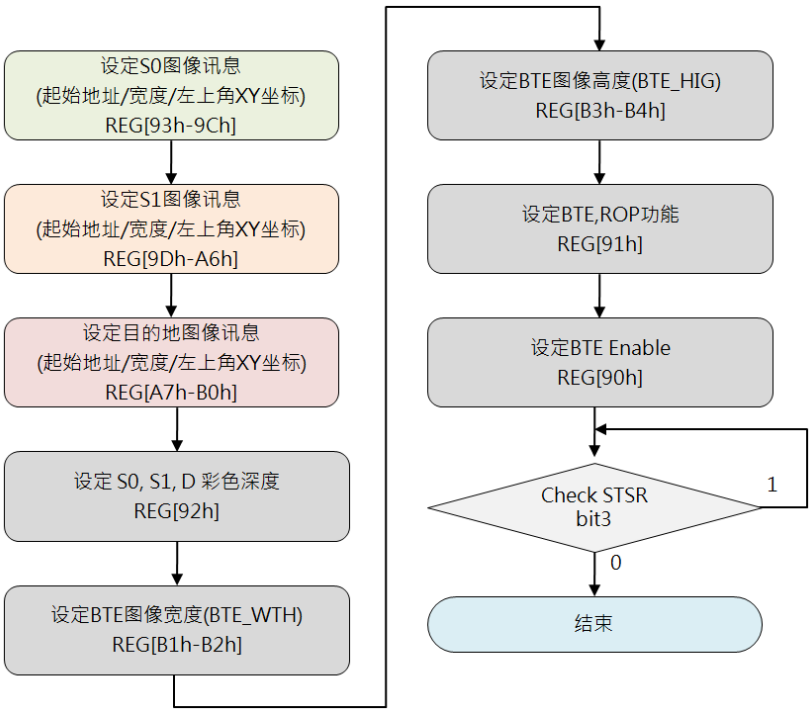


图 10-4：结合光栅操作的 BTE 内存复制流程图(1)

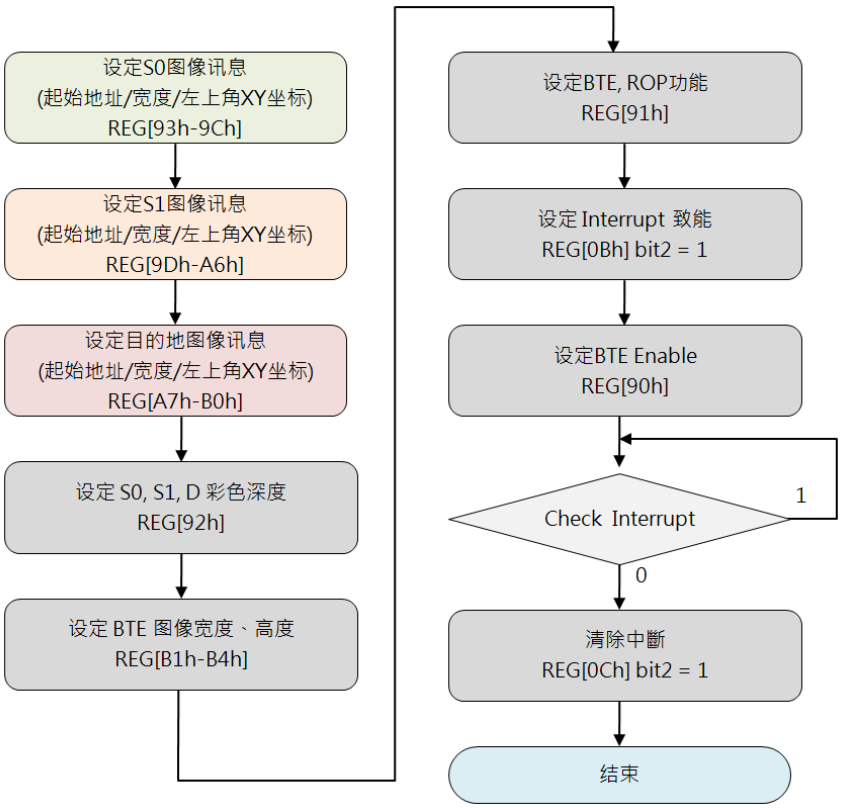


图 10-5：结合光栅操作的 BTE 内存复制流程图(2)

```
void LT768_BTE_Memory_Copy
(
unsigned long S0_Addr,          // S0 图像的内存起始地址
unsigned short S0_W,            // S0 图像的宽度
unsigned short XS0,             // S0 图像的左上方 X 坐标
unsigned short YS0,             // S0 图像的左上方 Y 坐标
unsigned long S1_Addr,          // S1 图像的内存起始地址
unsigned short S1_W,            // S1 图像的宽度
unsigned short XS1,             // S1 图像的左上方 X 坐标
unsigned short YS1,             // S1 图像的左上方 Y 坐标
unsigned long Des_Addr,         // 目的图像的内存起始地址
unsigned short Des_W,           // 目的图像的总宽度
unsigned short Xdes,            // 目的图像的左上方 X 坐标
unsigned short Ydes,            // 目的图像的左上方 Y 坐标
unsigned int ROP_Code,          // 光栅操作模式
unsigned short X_W,             // 目的图像的宽度
unsigned short Y_H              // 目的图像的长度
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：内存 0 地址起的（100, 100）坐标
来源 1：内存 1024*600*2 地址起的（50, 50）坐标
目的图像的内存地址及位置：内存 1024*600*2*2 地址起的（200, 200）坐标
目的图像的大小：100*100
光栅的操作模式：0x0A(显示来源 1)

实现函数：

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 结合 Chroma Key 的 MCU 写入

此功能为 MCU 具有关键色的写入数据功能。此功能可以提升 MCU 写入 SDRAM 的速度。一但这个功能被使能后，BTE 引擎会维持忙碌状态直到所有数据被写入为止。

与“BTE 写入”功能不同的是“结合 Chroma Key 的 MCU 写入”功能在处理数据时，如果 MCU 写入数据与关键色（Chroma key）相同，则写入 S0 数据会忽略掉。而关键色是被设定在“BTE background Color”寄存器中。举例说明如果来源端红色 TOP 背景是绿色的，经由选择绿色为透明色的话，那么通过此功能写出来的图就是一个红色的 TOP，绿色则不会被写入内存中。请参考下面图示及程序流程：

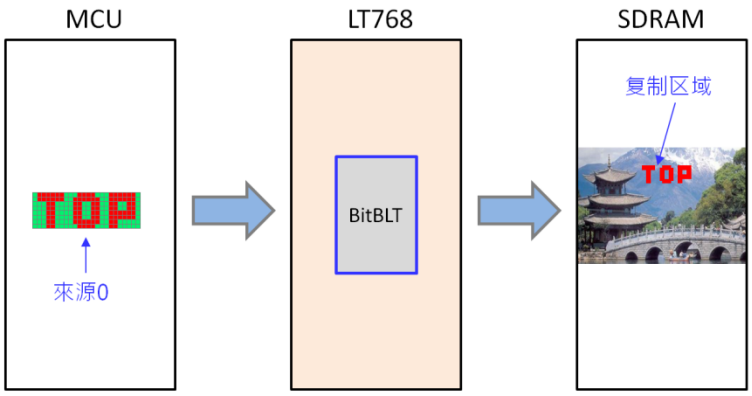


图 10-6：结合 Chroma Key 的 MCU 写入范例

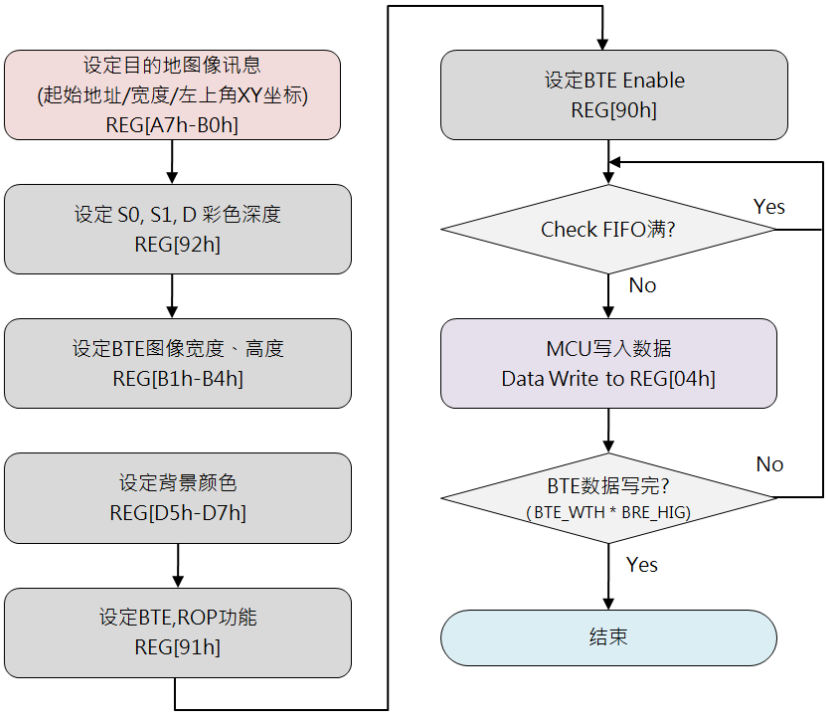


图 10-7：结合 Chroma Key 的 MCU 写入流程图

```
void LT768_BTE_MCU_Write_Chroma_key_MCU_16bit
(
unsigned long Des_Addr,                 // 目的图像的内存起始地址
unsigned short Des_W,                   // 目的图像的总宽度
unsigned short Xdes,                    // 目的图像的左上方 X 坐标
unsigned short YDes                     // 目的图像的左上方 Y 坐标
unsigned long Background_color,         // 透明色
unsigned short X_W,                     // 目的图像的宽度
unsigned short Y_H,                     // 目的图像的长度
const unsigned short *data              // S0 数据的起始地址
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：MCU 写入的一张 16 色的 100*100 的图像（unsigned short Picture_Data[100*100]）
目的图像的内存地址及位置：内存 0 地址起的（200, 200）坐标
目的图像的大小：100*100
透明色：红色

实现函数：

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

10.2.4 结合 Chroma Key 的内存复制（不含 ROP）

此功能可以复制搬移一指定的内存来源区域到内存目的区域，并且在复制搬移的过程中会比较来源端数据与（Chroma Key）的颜色，当两者相同时，不去更改内存目的端的数据，表现出来就是与关键色相同的会被透明处理。而关键色的设定在“BTE Background Color”寄存器（REG[D7h:D5h]）中。来源端与目的端皆是内存为来源。举例说明如果来源端背景是绿色，关键色也是设成绿色的，那么通过此功能写出来的图就是一个红色的 TOP，绿色变成透明色则不会被写入内存中。请参考下面图示及程序流程：

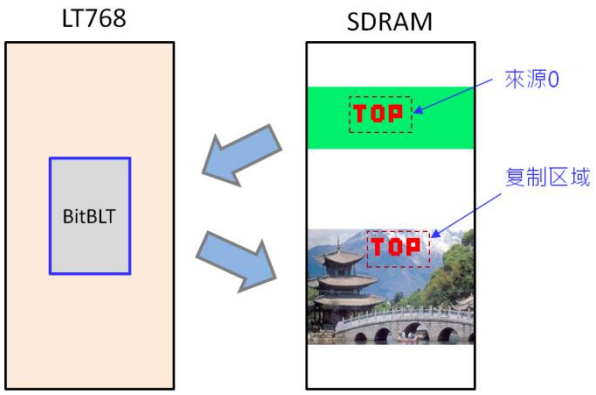


图 10-8：结合 Chroma Key 的内存复制范例

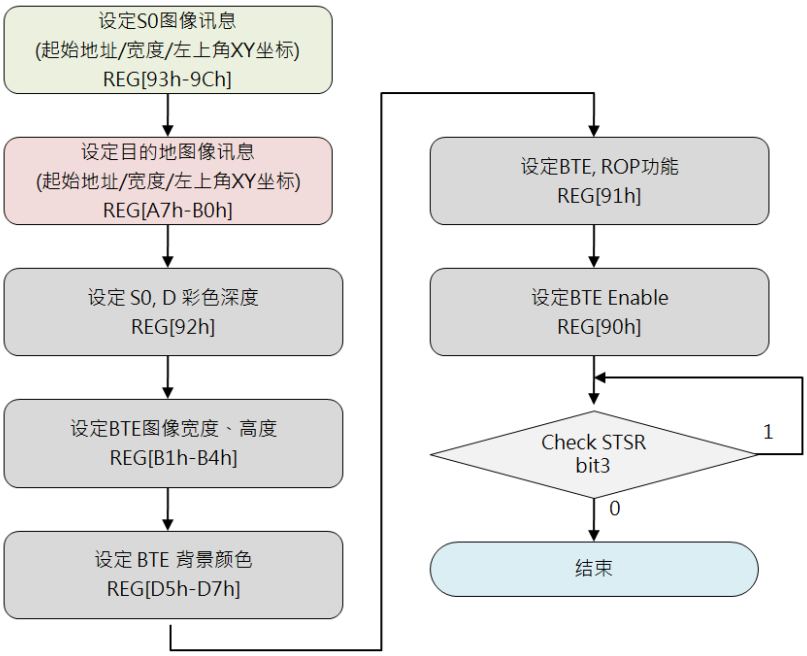


图 10-9：结合 Chroma Key 的内存复制流程图

```
void LT768_BTE_Memory_Copy_Chroma_key
(
unsigned long S0_Addr,                  // S0 图像的内存起始地址
unsigned short S0_W,                    // S0 图像的宽度
unsigned short XS0,                     // S0 图像的左上方 X 坐标
unsigned short YS0,                     // S0 图像的左上方 Y 坐标
unsigned long Des_Addr,                 // 目的图像的内存起始地址
unsigned short Des_W,                   // 目的图像的总宽度
unsigned short Xdes,                    // 目的图像的左上方 X 坐标
unsigned short Ydes,                    // 目的图像的左上方 Y 坐标
unsigned long Background_color,         // 透明色
unsigned short X_W,                     // 目的图像的宽度
unsigned short Y_H                      // 目的图像的长度
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：内存 0 地址起的（200, 200）坐标
目的图像的内存地址及位置：内存 1024*600*2 地址起的（100, 100）坐标
目的图像的大小：100*100
透明色：红色

实现函数：

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

10.2.5 结合光栅操作的图样填满

此功能将一指定区域重复填满指定的 8*8、16*16 图案，而 8*8、16*16 像素的图案是使用此功能前已经预先储存在内存中。这个功能也可以结合 16 种光栅（ROP）操作。这个功能可以在一指定的区域加速复制图样，如快速背景张贴上。下图以 8*8 图案为例，POP 寄存器（REG[91h]）bit[7:4]设成 0x0C 得到的结果。

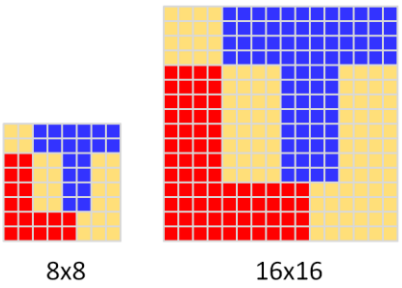


图 10-10：图样格式

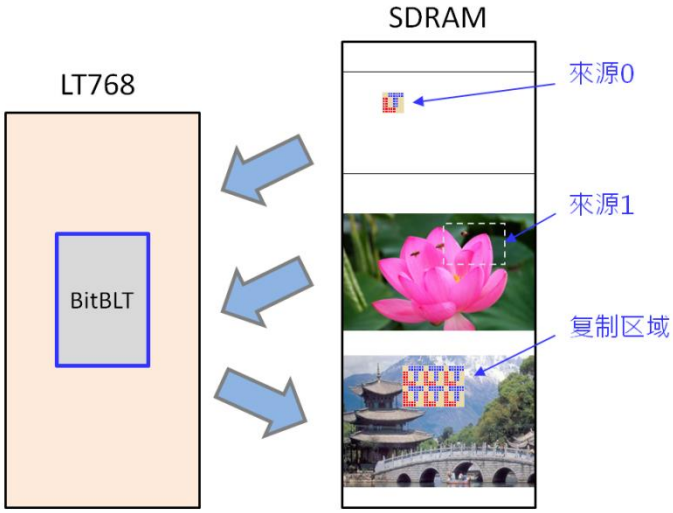


图 10-11：结合光栅操作的图样填满范例

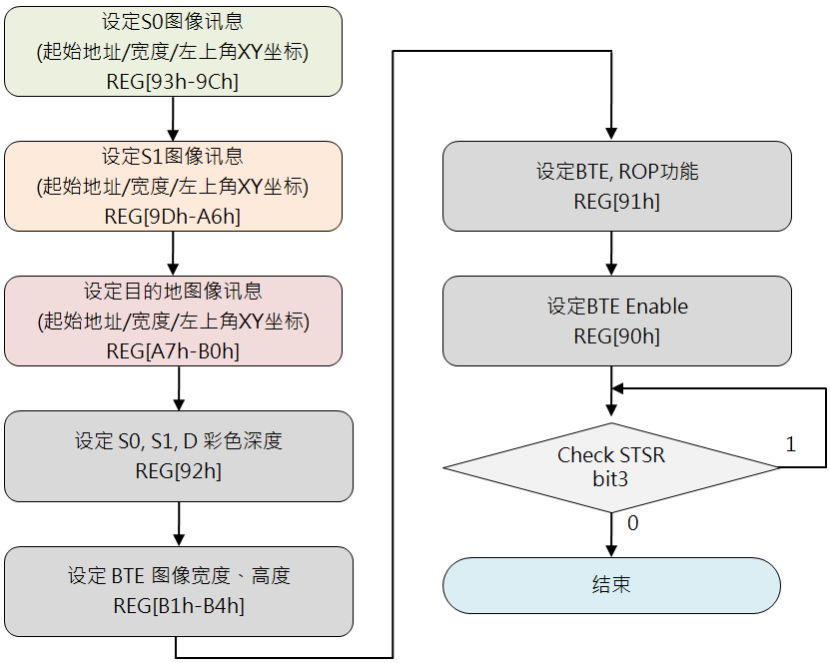


图 10-12：结合光栅操作的图样填满流程图

```
void LT768_BTE_Pattern_Fill
(
unsigned char P_8x8_or_16x16,        // 0：使用 8x8 Icon；1：使用 16x16 Icon.
unsigned long S0_Addr,               // S0 图像的内存起始地址
unsigned short S0_W,                 // S0 图像的宽度
unsigned short XS0,                  // S0 图像的左上方 X 坐标
unsigned short YS0,                  // S0 图像的左上方 Y 坐标
unsigned long Des_Addr,              // 目的图像的内存起始地址
unsigned short Des_W,                // 目的图像的总宽度
unsigned short Xdes,                 // 目的图像的左上方 X 坐标
unsigned short Ydes,                 // 目的图像的左上方 Y 坐标
unsigned int ROP_Code,               // 光栅操作模式
unsigned short X_W,                  // 目的图像的宽度
unsigned short Y_H                   // 目的图像的长度
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

P_8x8_or_16x16：选用 16x16 Icon
来源 0：内存 0 地址起的（200, 200）坐标
目的图像的内存地址及位置：内存 1024*600*2 地址起的（100, 100）坐标
光栅的操作模式：0x0C(显示来源 0)
目的图像的大小：100*100

实现函数：

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

10.2.6 结合 Chroma Key 的图样填满

此功能将一指定内存区域重复填满指定的 8*8、16*16 图案，但是在处理的过程中如果来源端颜色与关键色 (Chroma key) 相同那么对于目的端就不做写入，因此看到的效果将会是透明的。而关键色（Chroma key）被设定 REG[D5h] ~ [D7h]寄存器中。下图的范例关键色被设定为粉橘色，因此在指定内存区域重复填满后看到的效果将只有红色会出现。

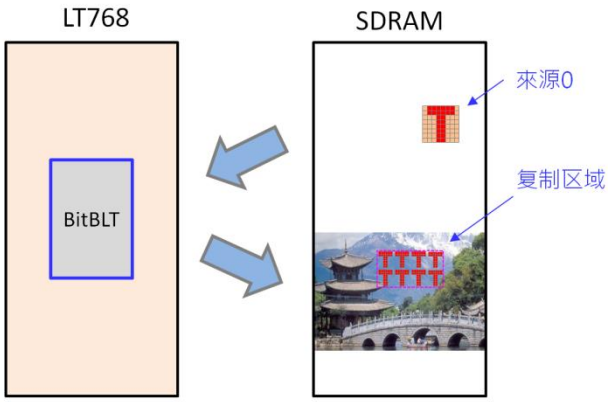


图 10-13：结合 Chroma Key 的图样填满范例

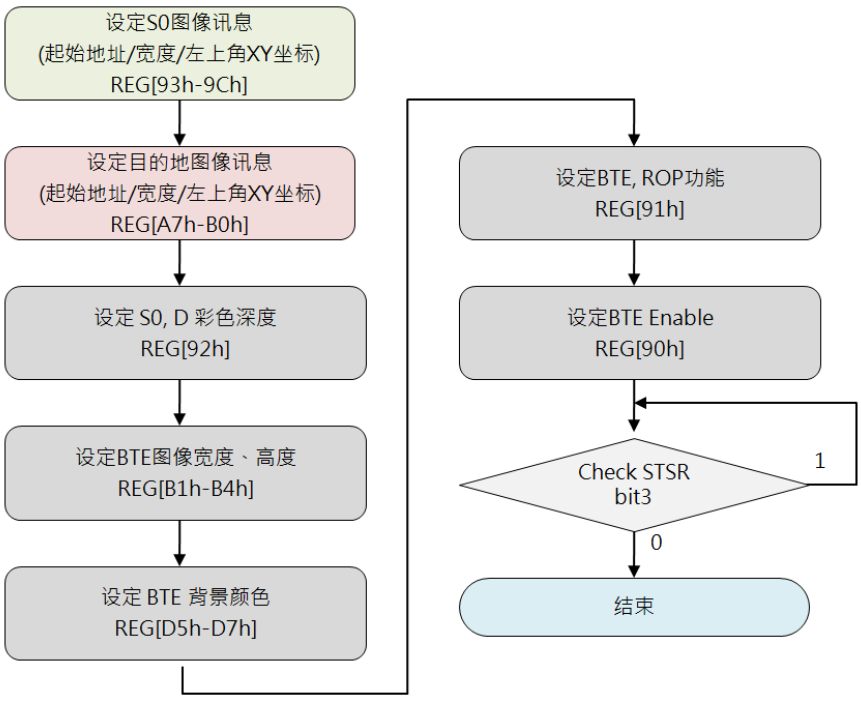


图 10-14：结合 Chroma Key 的图样填满流程图

```
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,               // S1 图像的内存起始地址
unsigned short S0_W,                 // S0 图像的宽度
unsigned short XS0,                  // S0 图像的左上方 X 坐标
unsigned short YS0,                  // S0 图像的左上方 Y 坐标
unsigned long Des_Addr,              // 目的图像的内存起始地址
unsigned short Des_W,                // 目的图像的总宽度
unsigned short XDes,                 // 目的图像的左上方 X 坐标
unsigned short YDes,                 // 目的图像的左上方 Y 坐标
unsigned int ROP_Code,               // 光栅操作模式
unsigned long Background_color,      // 透明色
unsigned short X_W,                  // 目的图像的宽度
unsigned short Y_H                   // 目的图像的长度
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

P_8x8_or_16x16：选用 16x16 Icon
来源 0：内存 0 地址起的（200，200）坐标
目的图像的内存地址及位置：内存 1024*600*2 地址起的（100，100）坐标
光栅的操作模式：0x0C(显示来源 0)
透明色：红色
目的图像的大小：100*100

实现函数：

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 结合扩展色彩的 MCU 写入

此功能为 MCU 将单色数据写入内存中，在这个操作中来源图档是单色（bit-map）的数据，经过 BTE 功能可以转成多位的图文件数据。如果单色图档的 bit 为“1”则转为前景色，如果单色图档的 bit 为“0”则转成背景色。这个功能让使用者方便由单色系统转成彩色系统。单色图在 BTE 内部是每个扫描线分开处理的，当一条扫描线处理完时，没有被处理的单色扫描线数据就被舍弃。下一行的数据则由下一笔数据包产生，每一笔写目的内存的数据做颜色扩展时都是由 MSB 处理到 LSB。如果 MCU 接口被设定为 16bits 时，那么 ROP 的起始位可以被设为 15 到 0 的任一位，MCU 接口被设定为 8bits，那么 ROP 起始位可以被设为 7 到 0 的任一位。来源 0 颜色深度 REG [92h] bit[7:6] 在此功能中将不被参考。

下图的范例中前景色被设定为红色，背景色被设定为蓝色，因此 MCU 将单色数据（来源 0）经过 BLT 填入指定内存区域看到的效果就是这样。

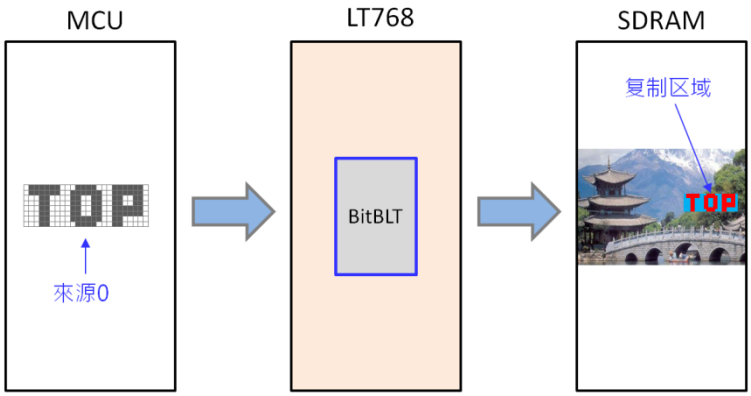


图 10-15：结合扩展色彩的 MCU 写入范例

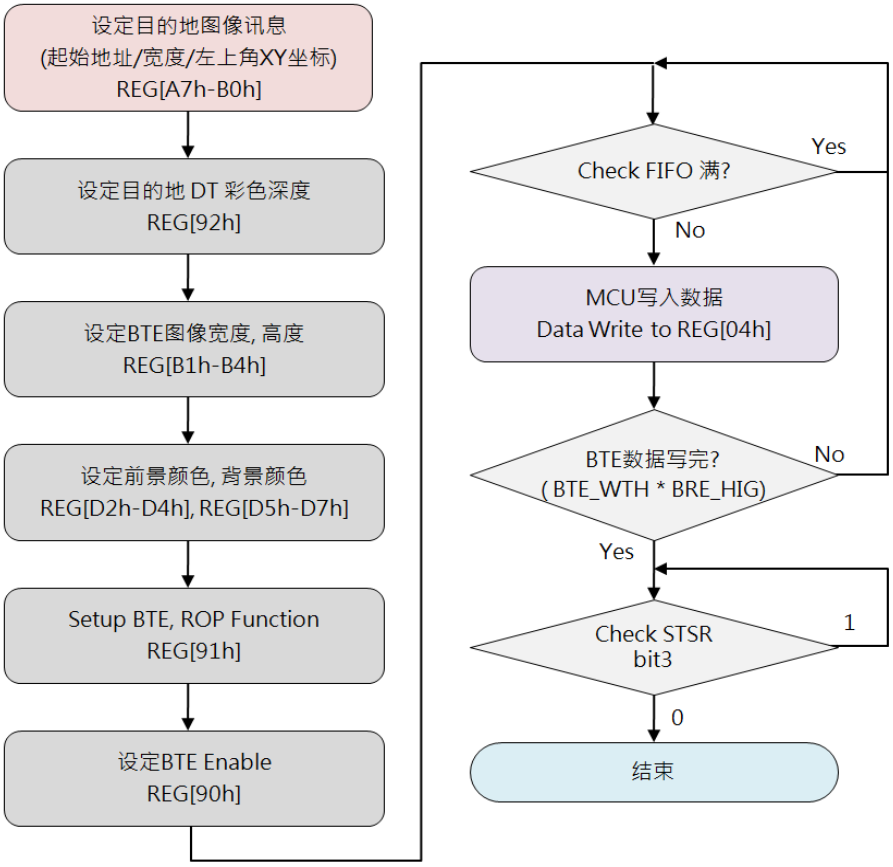


图 10-16：结合扩展色彩的 MCU 写入流程图

例如下图 10-17，ROP = 7，前景色寄存器设定的颜色为红色，背景色寄存器设定的颜色为土黄色，如果 BTE Width = 23 时，所得到的色彩扩展显示结果。图 10-18 范例则为 ROP = 3，其他设定不变时所得到的色彩扩展结果。

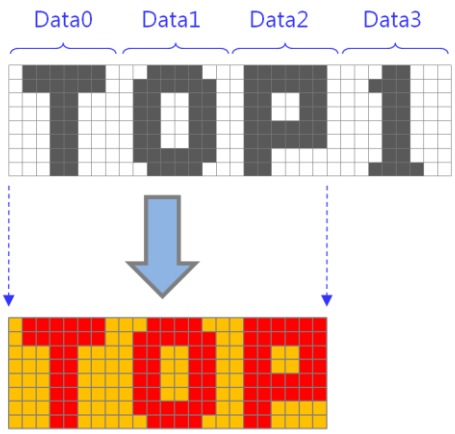


图 10-17：色彩扩展显示范例 1

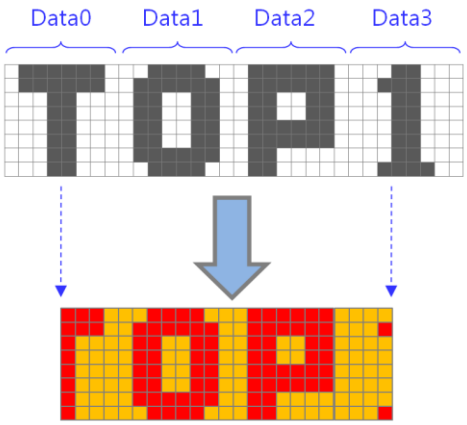


图 10-18：色彩扩展显示范例 2

提示：

1. Sent Data Numbers per Row
   = [BitBLT Width + (MCU I/F bits – Start bit - 1)] / (MCU I/F bits)，取無條件進位的整数。

2. Total Data Number = (Sent Data Numbers per Row) * BitBLT Height

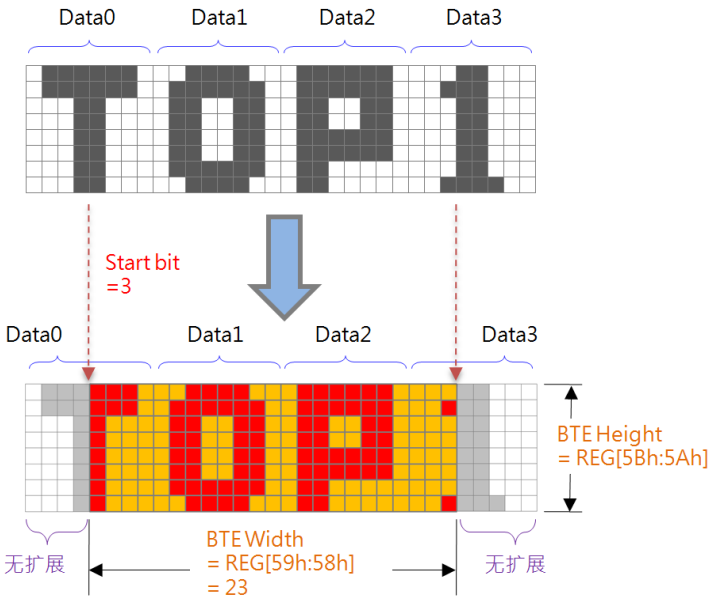


图 10-19：色彩扩展显示数据格式

```
void LT768_BTE_MCU_Write_ColorExpansion_MCU_16bit
(
unsigned long Des_Addr,                 // 目的图像的内存起始地址
unsigned short Des_W,                   // 目的图像的总宽度
unsigned short Xdes,                    // 目的图像的左上方 X 坐标
unsigned short Ydes,                    // 目的图像的左上方 Y 坐标
unsigned short X_W,                     // 目的图像的宽度
unsigned short Y_H,                     // 目的图像的长度
unsigned long Foreground_color,         // 前景色
unsigned long Background_color,         // 背景色
const unsigned short *data              // 16-bit Data
)
```


举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：MCU 写入的数据（unsigned short Data[100*100]）
目的图像的内存地址及位置：内存 1024*600*2 地址起的（200, 200）坐标
前景色：红色
背景色：蓝色
目的图像的大小：100*100

实现函数：

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

10.2.8 结合扩展色彩与 Chroma key 的 MCU 写入

此 BitBLT 操作除了背景色被完全忽略外，与颜色扩展 BLT 几乎完全相同，来源单色位图中设置为 1 的所有位都将颜色扩展到 BitBLT 前景色;而来源单色位图中设置为 0 的所有位将不会扩展到 BitBLT 背景色。

下图的范例中前景色被设定为红色，因此 MCU 将单色数据（来源 0）经过 BLT 填入指定内存区域看到的效果就是这样。

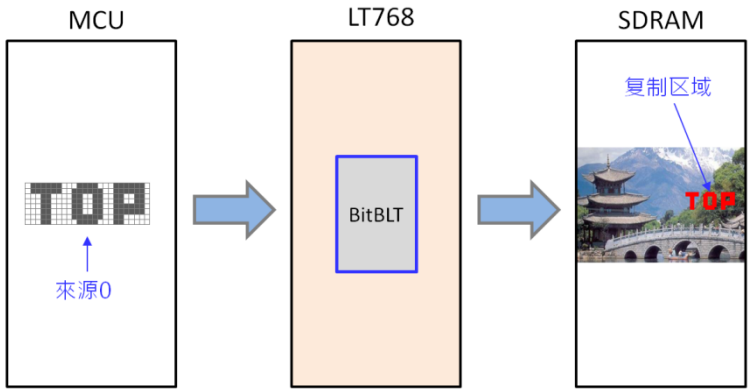


图 10-20：结合扩展色彩与 Chroma key 的 MCU 写入范例

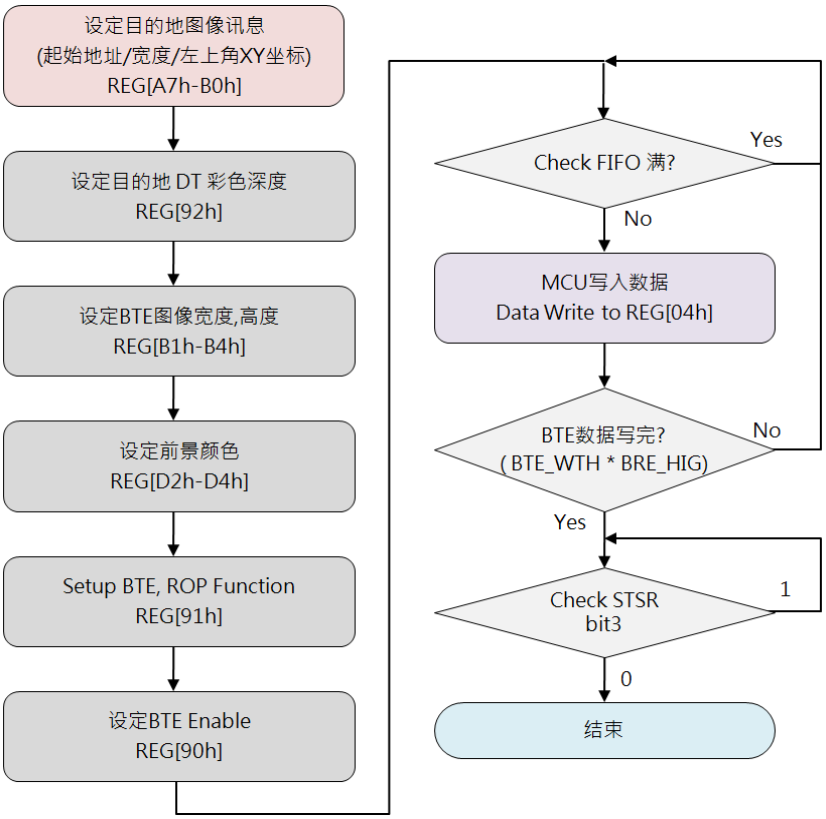


图 10-21：结合扩展色彩与 Chroma key 的 MCU 写入流程图

10.2.9 结合透明度的内存复制

此功能可以混合来源 0 数据与来源 1 数据然后再写入目的内存。这个功能有两个模式 – Picture 模式与 Pixel 模式。Picture 模式可以被操作在 8/16/24bpp 色深下并且对于全图只具有一种混合透明度（Alpha Level），混合度被定义在 REG[B5h]。Pixel 模式只能被操作在来源 1 端是 8/16bpp 模式，而各个 Pixel 具有其各自的混合度，在来源 1 为 16bpp 色深下像素的 bit[15:12] 是透明度，剩余的 bit 则为色彩数据；而来源 1 为 8bpp 色深情形下像素 bit[7:6] 是透明度，bit[5:0]则是被使用在索引调色盘（Palette Color）的颜色。

- **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）]

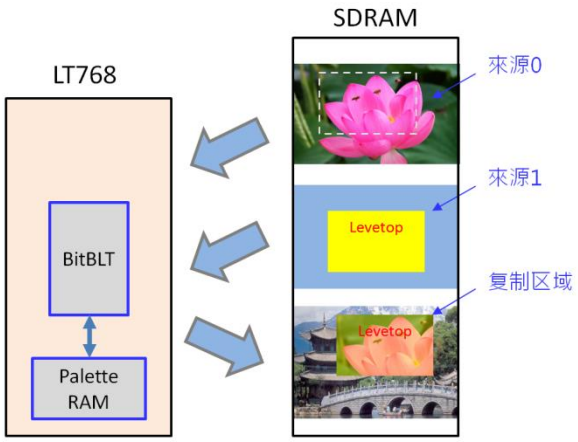


图 10-22：8bpp Pixel Mode 范例

表 10-3：Alpha Blending Pixel Mode -- 8bpp

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

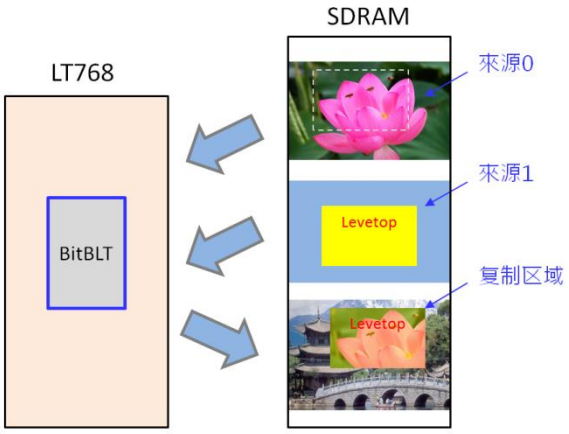


图 10-23：16bpp Pixel Mode 范例

表 10-4：Alpha Blending 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

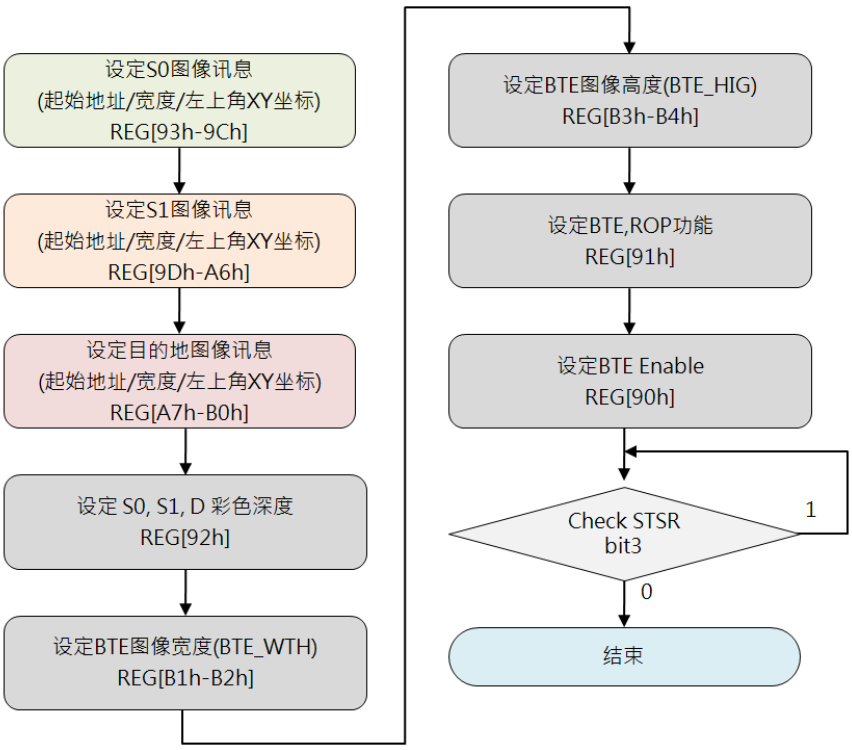


图 10-24：结合透明度的内存复制（Pixel Mode）流程图

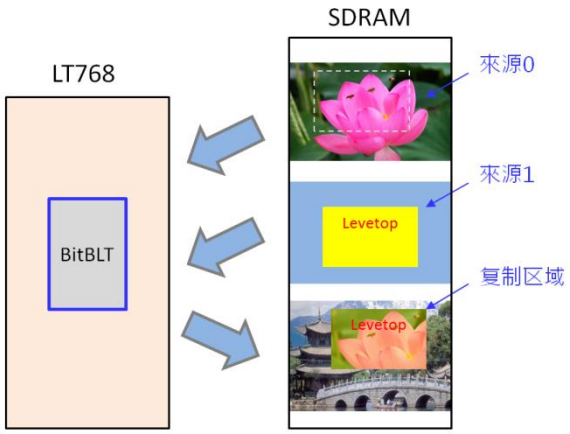


图 10-25：Picture Mode 范例

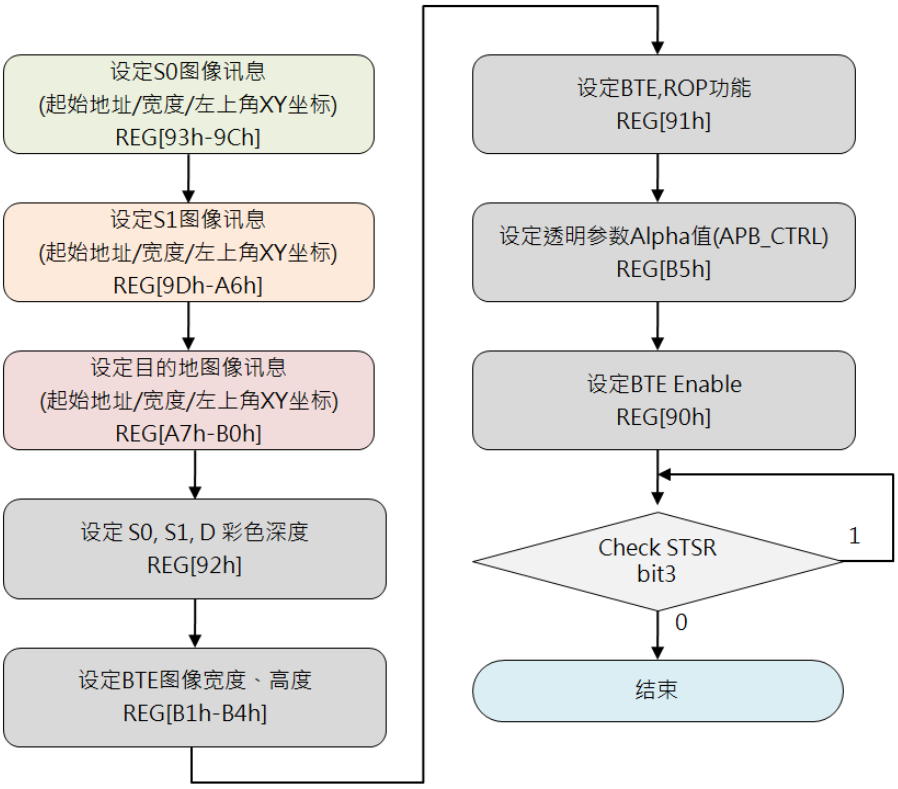


图 10-26：结合透明度的内存复制（Picture Mode）流程图

```
void BTE_Alpha_Blending
(
unsigned long S0_Addr,          // S0 图像的内存起始地址
unsigned short S0_W,            // S0 图像的宽度
unsigned short XS0,             // S0 图像的左上方 X 坐标
unsigned short YS0,             // S0 图像的左上方 Y 坐标
unsigned long S1_Addr,          // S1 图像的内存起始地址
unsigned short S1_W,            // S1 图像的宽度
unsigned short XS1,             // S1 图像的左上方 X 坐标
unsigned short YS1,             // S1 图像的左上方 Y 坐标
unsigned long Des_Addr,         // 目的图像的内存起始地址
unsigned short Des_W,           // 目的图像的总宽度
unsigned short Xdes,            // 目的图像的左上方 X 坐标
unsigned short Ydes,            // 目的图像的左上方 Y 坐标
unsigned short X_W,             // 目的图像的宽度
unsigned short Y_H,             // 目的图像的长度
unsigned char alpha             // Alpha Blending effect 0 ~ 32, Destination data =
                                // (Source 0 * (1- alpha)) + (Source 1 * alpha)
)
```


举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 0：内存 0 地址起的（0，0）坐标
来源 1：内存 1024*600*2 地址起的（0，0）坐标
目的图像的内存地址及位置：内存 1024*600*2*2 地址起的（50，50）坐标
透明度：5（0~31）
目的图像的大小：100*100

实现函数：

```
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 区域填满（Solid Fill)

此功能会针对 BTE 指定的矩形范围做指定颜色的填满。这个功能是被使用在填满一个大范围区域。而填满的颜色被设定在 BTE 的前景色寄存器中。

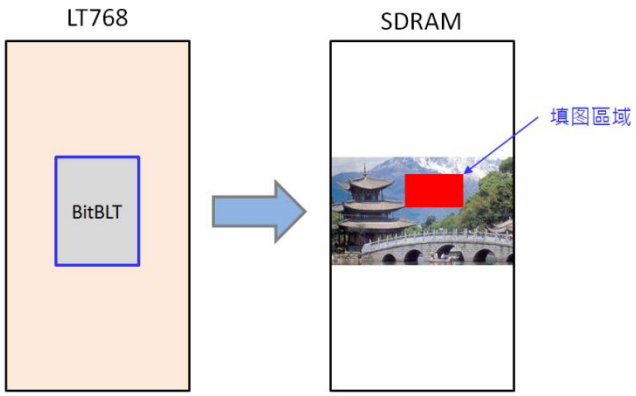


图 10-27：区域填满范例

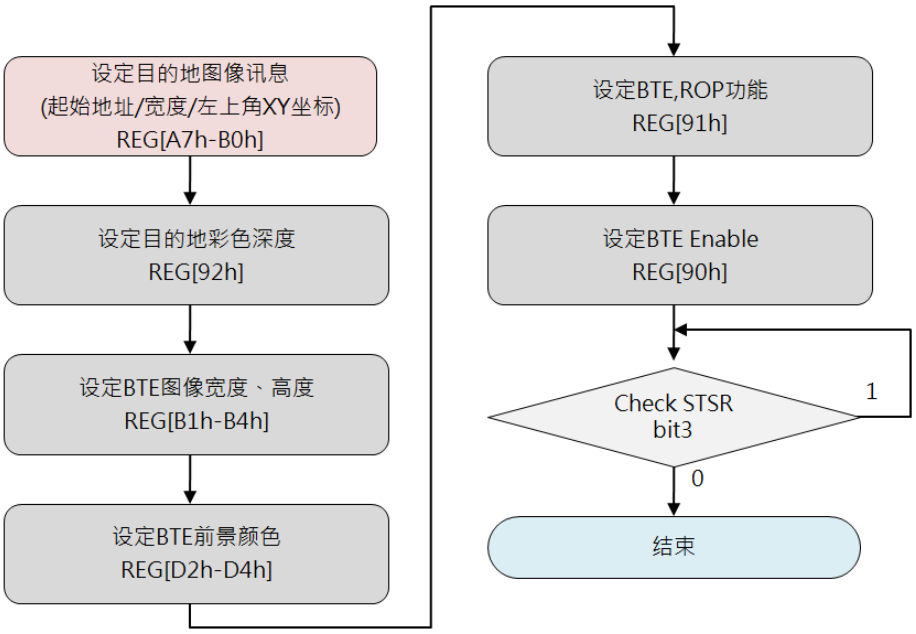


图 10-28：区域填满流程图

```
void BTE_Solid_Fill
(
unsigned long Des_Addr,          // 目的图像的内存起始地址
unsigned short Des_W,            // 目的图像的总宽度
unsigned short XDes,             // 目的图像的左上方 X 坐标
unsigned short YDes,             // 目的图像的左上方 Y 坐标
unsigned short color,            // 填充的颜色
unsigned short X_W,              // 目的图像的长度
unsigned short Y_H               // 目的图像的宽度
)
```

举例：（假设 LCD 屏为 1024*600 的分辨率）

目的图像的内存地址及位置：内存 0 地址起的（200，200）坐标
填充颜色：红色
目的图像的大小：100*100

实现函数：

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

11. 显示文字

11.1 使用内建字库

在使用内建字体之前需要先初始化，初始化后才可调用内置字库的显示函数。

```
void LT768_Select_Internal_Font_Init
(
unsigned char Size,              // 设置字体大小 16：8*16；24：12*24；32：16*32
unsigned char XxN,               // 字体的宽度放大倍数：1~4
unsigned char YxN,               // 字体的高度放大倍数：1~4
unsigned char ChromaKey,         // 0：字体背景色透明；1：可以设置字体的背景色
unsigned char Alignment          // 0：字体不对齐；1：字体对齐
)

void LT768_Print_Internal_Font_String
(
unsigned short x,                // 字体开始显示的 x 位置
unsigned short y,                // 字体开始显示的 y 位置
unsigned long FontColor,         // 字体的颜色
unsigned long BackGroundColor,   // 字体的背景色（提示：当字体背景初始化成透明时，设置该
                                 // 值无效）
char *c                          // 数据缓冲的首地址
)
```

举例：要使用 16*32 的内建字库在 LCD 屏的(10,100)位置起显示红色的“LeveTop LT768”，且不放大字体的高度和宽度、背景透明，字体也对齐:

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

提示：在内建字库中是不支持中文字体的，如果需要使用中文字体时需要自己外建字库。

11.2 建立中文字库

11.2.1 取得字库

字库的取模需要使用相应的字库取模软件，然后生成 Bin 格式的文件，需要显示中文字时再由 LT768 对 Bin 文件进行调用，请参考 11.3 节（本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项制作字库取模功能）。LT768 支持 GB2312 简体字库、GB2312 繁体字库，以及 BIG5 字库，每个字库都支持 16*16、24*24、32*32、48*48、72*72 的五种字体大小。其中，GB2312 的繁体版的编码规则与 GB2312 简体一致。

11.2.2 存入字库档方法

字库的 Bin 文件需要存入 Flash 存储器中，在使用字库前，要把字库数据从 Flash 中写到 SDRAM 中，然后设置好相应的参数，此时只需发送字库码，就可以在 LCD 屏上显示出来。

11.2.3 显示中文字（16*16、24*24、32*32）

在使用外建字库前需要先初始化，把字库从 Flash 中读取及储存到 SDRAM 中，设置好相应的参数。

```
void LT768_Select_Outside_Font_Init
(
unsigned char SCS,              // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,              // SPI 时钟分频参数：SPI Clock = System Clock /{(Clk+1)*2}
unsigned long FlashAddr,        // 源地址(Flash)
unsigned long MemoryAddr,       // 目的地址(SDRAM)
unsigned long Num,              // 字库的数据量大小
unsigned char Size,             // 设置字体大小；16：16*16；24:24*24；32:32*32
unsigned char XxN,              // 字体的宽度放大倍数：1~4
unsigned char YxN,              // 字体的高度放大倍数：1~4
unsigned char ChromaKey,        // 1：字体背景色透明；0：可以设置字体的背景色
unsigned char Alignment         // 0：字体不对齐；1：字体对齐
)
```

初始化完后，在编译器是 GB2312 的编码规则下，调用以下的函数就可以在 LCD 屏上显示中文。

```
void LT768_Print_Outside_Font_String
(
unsigned short x,                   // 字体开始显示的 x 位置
unsigned short y,                   // 字体开始显示的 y 位置
unsigned long FontColor,            // 字体的颜色
unsigned long BackGroundColor,      // 字体的背景色（提示：当字体背景初始化成透明时，
                                    // 设置该值无效）
unsigned char *c                    // 数据缓冲的首地址
)
```

如果编译器是 BIG5 的编码规则下，则需要调用以下的函数：

```
void LT768_Print_Outside_Font_String_BIG5
(
unsigned short x,                   // 字体开始显示的 x 位置
unsigned short y,                   // 字体开始显示的 y 位置
unsigned long FontColor,            // 字体的颜色
unsigned long BackGroundColor,      // 字体的背景色（提示：当字体背景初始化成透明时，
                                    // 设置该值无效）
unsigned char *c                    // 数据缓冲的首地址
)
```

举例：从外挂 Flash0 里的 0x00 地址起读取 24*24 楷书中文字体到 SDRAM 的 0x003E537E0 的首地址里，该字库的数据量为 0x0009B520。用该字体在 LCD 屏的(10,100)位置起显示红色的“东莞市乐升电子有限公司-LT768”字串，且不放大字体的高度和宽度、背景色透明、字体也对齐：

GB2312 编码规则：

```
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 编码规则：

```
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”);
```

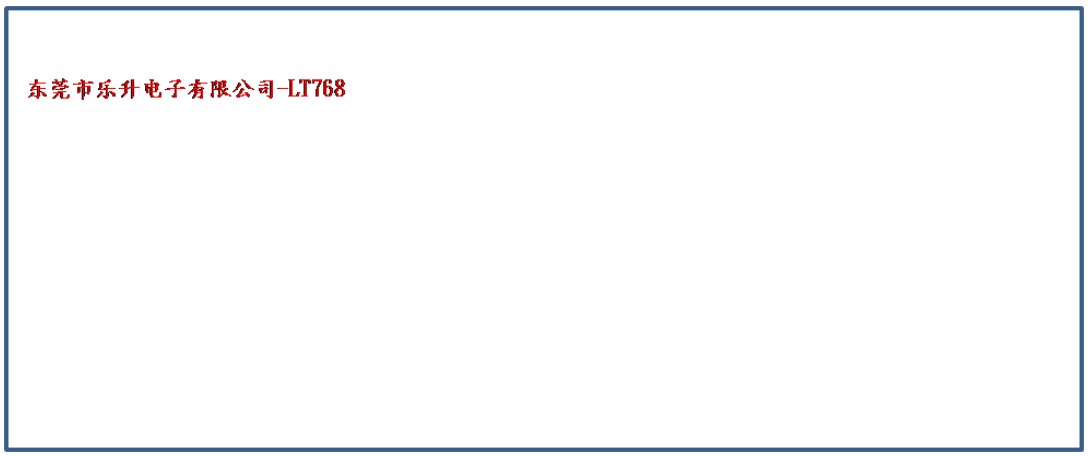


图 11-1：显示 24*24 楷书中文字

提示：

1、该函数，只是对 GB2312 或是 BIG5 的中文编码才有效，如若要使用其他的的中文编码，需要自己修改该函数对应的部分

2、有些编译器对中文的编译有可能不是按照 GB2312 或是 BIG5 的编码规则，此时需要自行配置自己编译器的设置了。

11.2.4 显示大型中文字（48*48、72*72）

在使用大型中文字库时，如果编译器是 GB2312 的编码规则下，调用以下函数：

```
void LT768_Print_Outside_Font_GB2312_48_72
(
unsigned char SCS,                // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,                // SPI 时钟分频参数：SPI Clock=System Clock /{(Clk+1)*2}
unsigned long FlashAddr,          // 源地址(Flash)
unsigned long MemoryAddr,         // 目的地址(SDRAM)
unsigned char Size,               // 设置字体大小；48：48*48；72:72*72
unsigned char ChromaKey,          // 1：可以设置字体的背景色；0：字体背景色透明
unsigned short x,                 // 字体开始显示的 x 位置
unsigned short y,                 // 字体开始显示的 y 位置
unsigned long FontColor,          // 字体的颜色
unsigned long BackGroundColor,    // 字体的背景色(注意：当字体背景色为透明时，设置该值无效)
unsigned short w,                 // 字体加粗：1：不加粗；2：加粗 1 级；3：加粗 2 级
unsigned short s,                 // 行距
unsigned char *c                  // 数据缓冲的首地址
)
```

如果编译器是 BIG5 的编码规则下，则需要调用以下的函数：

```
void LT768_Print_Outside_Font_BIG5_48_72
(
unsigned char SCS,                  // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,                  // SPI 时钟分频参数: SPI Clock = System Clock /{(Clk+1)*2}
unsigned long FlashAddr,            // 源地址(Flash)
unsigned long MemoryAddr,           // 目的地址(SDRAM)
unsigned char Size,                 // 设置字体大小；48：48*48；72:72*72
unsigned char ChromaKey,            // 1：可以设置字体的背景色；0：字体背景色透明
unsigned short x,                   // 字体开始显示的 x 位置
unsigned short y,                   // 字体开始显示的 y 位置
unsigned long FontColor,            // 字体的颜色
unsigned long BackGroundColor,      // 字体的背景色(注意：当字体背景色为透明时，设置该值无效)
unsigned short w,                   // 字体加粗：1：不加粗；2：加粗 1 级；3：加粗 2 级
unsigned short s,                   // 行距
unsigned char *c                    // 数据缓冲的首地址
)
```

举例：从外挂 Flash0 里的 0x005DC000 地址起读取 48*48 楷书中文字体到 SDRAM 的 0x003E537E0 的首地址里。用该字体在 LCD 屏的(10,100)位置起显示红色的“东莞市乐升电子有限公司-LT768”字串，且不加粗、背景色透明、行距为 0：

GB2312 编码规则：

```
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”);
```

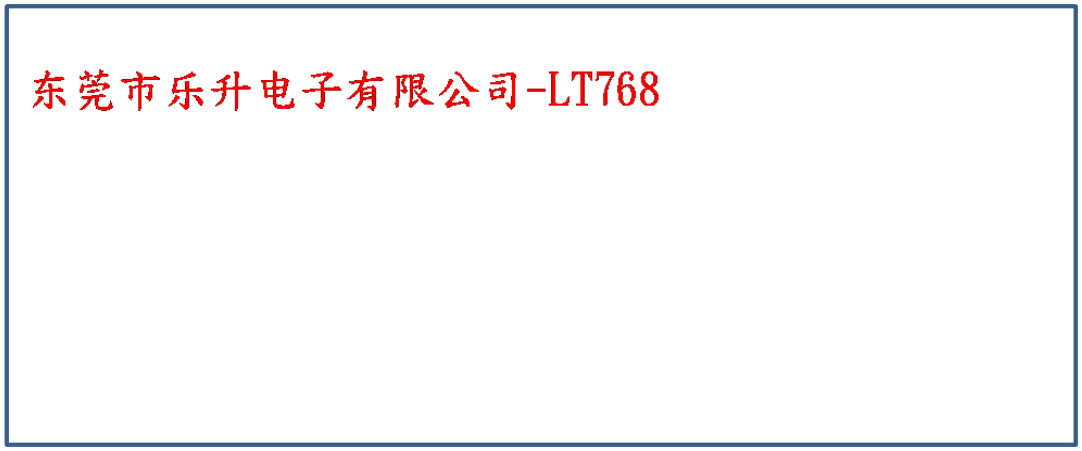


图 11-2：显示 48*48 楷书中文字

BIG5 编码规则：

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”);

提示：

1、以上函数，只是对 GB2312 和 BIG5 的中文编码规则才有效，如若要使用其他的的中文编码，需要自己修改该函数对应的部分。

2、有些编译器对中文的编译有可能不是按照 GB2312 或者 BIG5 的编码规则，此时需要自行配置自己编译器的设置。

3、在使用大型中文字库时，如果要显示英文及标点符号，需输入全角格式的英文及标点符号，才能正常调用字库。

4、使用大型字库(48*48、72*72)，设置字体颜色以及背景色时，需使用 256 位色。

11.2.5 文字行距:

此函数就是设置当文字写到最后自动换行时，与前一行的行距，参数 temp 的大小就是指定行距之间相差的像素点。

void Font_Line_Distance(unsigned char temp);

11.3 制作字库的 Bin 文件

在应用端如果要使用到中文字库，可以利用本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项制作字库 Bin 文件的功能,能将字库信息转成 Bin 文件,然后透过 DMA 传输方式将字库数据存到 LT768x 的内建显示内存中，之后如果要在 TFT 屏上显示中文，MCU 只须送 GB 码（2 个 Bytes）就可以在设定的位置上显示出中文，因此可以提升中文显示效能，还降低 MCU 处理中文显示的负担。对于字库 Bin 文件的制作，使用者可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明，举例产生一个 16*16 的宋体字库 Bin 文件：

1、点击【LT_IMAGE_TOOL 菜单>Font】即可打开中文字库 Bin 文件制作界面：

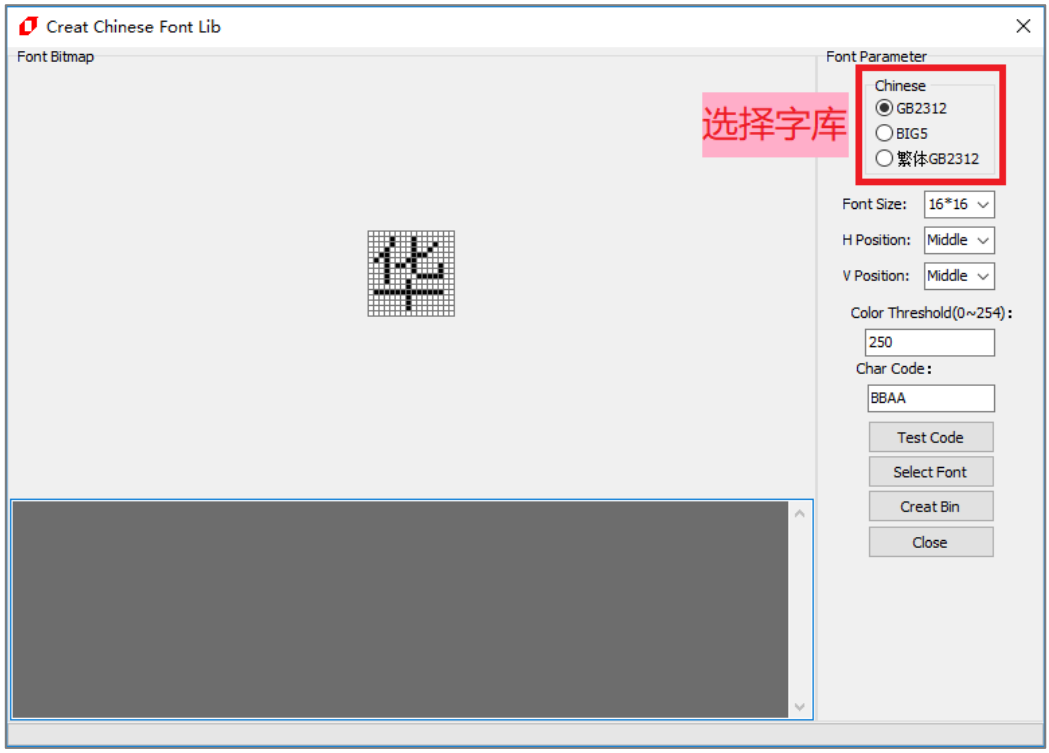


图 11-3：制作中文字库

2、点击【Select Font】按钮，可设置字体、字形、大小等，设置完毕后，按确定保存：

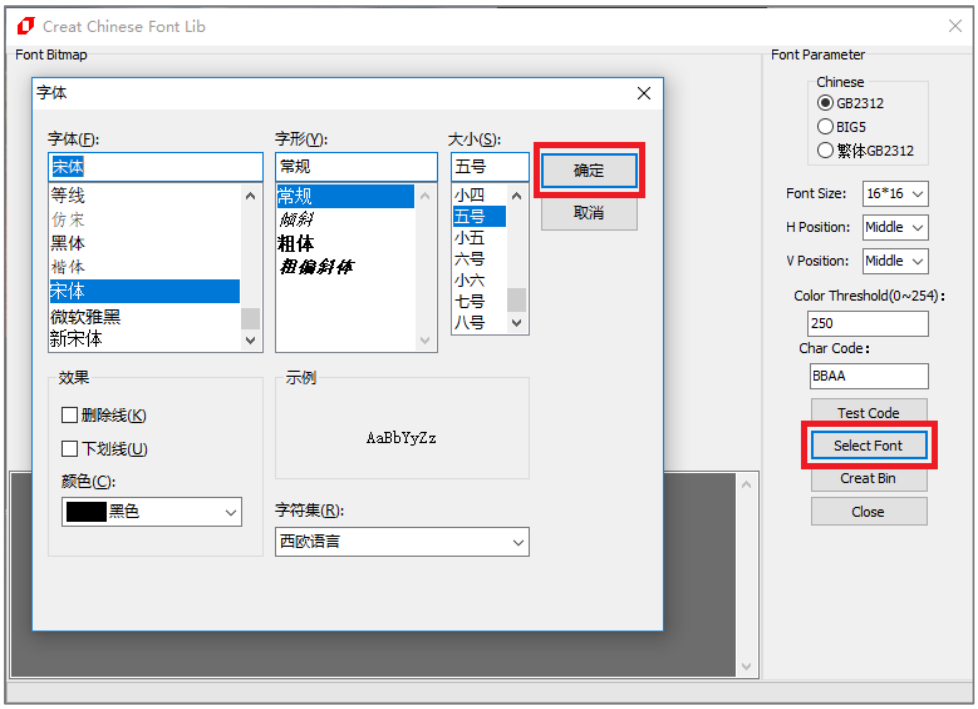


图 11-4：选择字体

3、设置字库有 16*16、24*24、32*32、48*48、72*72 五种字体大小，你也可以设置字体横向(偏左、居中、偏右)和纵向(偏上、居中、偏下)的位置、颜色阈值(0~254)、和预览文字，点击【Test Code】按钮即可查看该字符的数据。

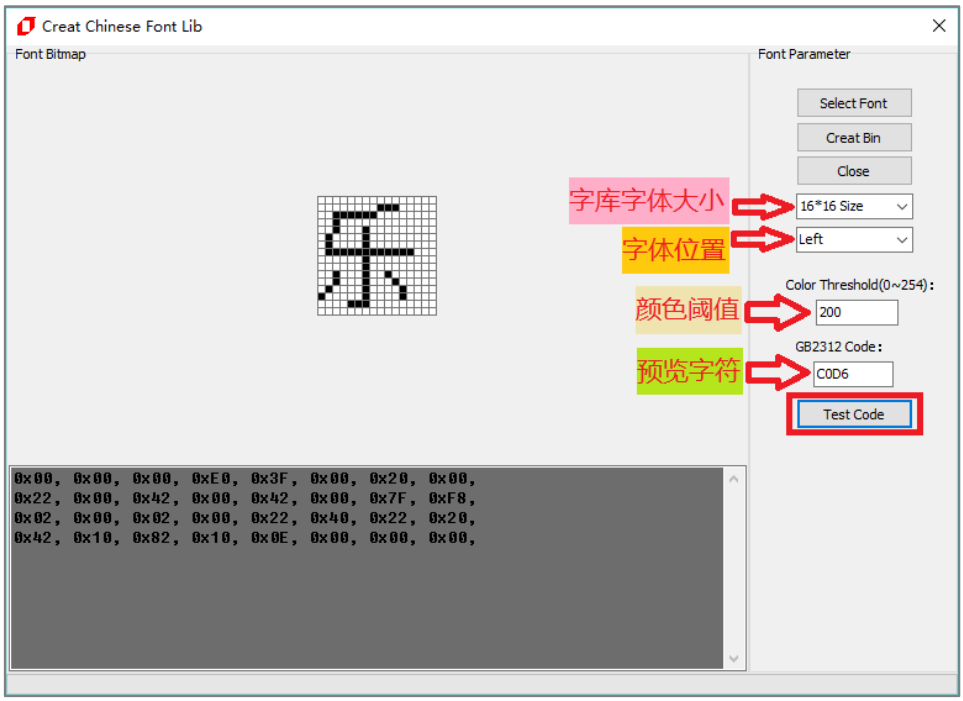


图 11-5：设置字库

4、点击【Create Bin】即可输出字库 Bin 文件。注意输入文件名时文件名中不能包含下面这些字符，如：? * / \ < > : " |，否则无法保存。

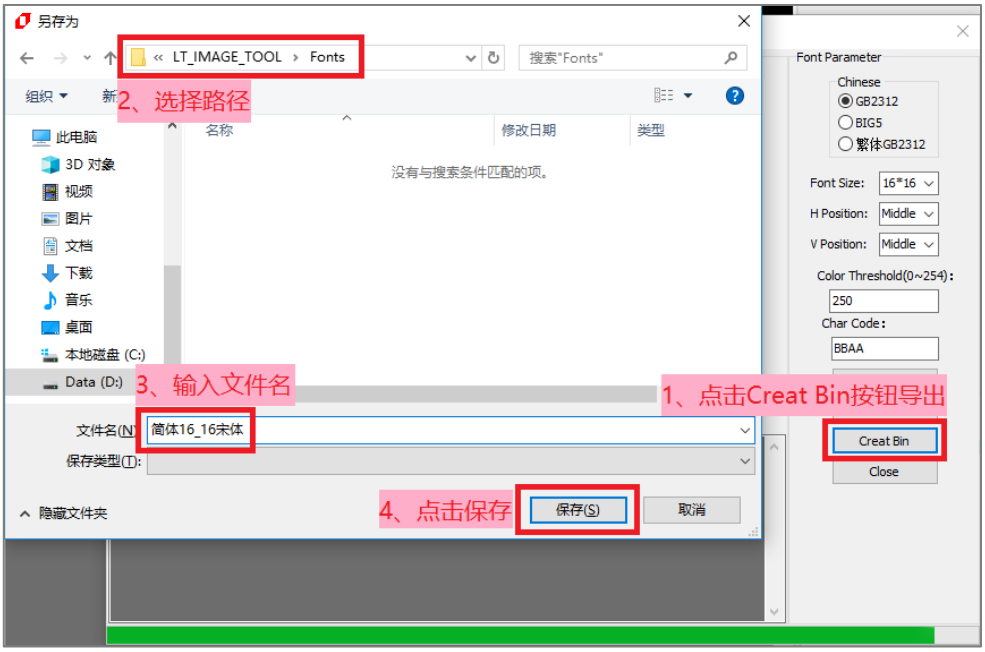


图 11-6：保存字库

当显示 (字库)+Font Lib ok 时，即保存成功：

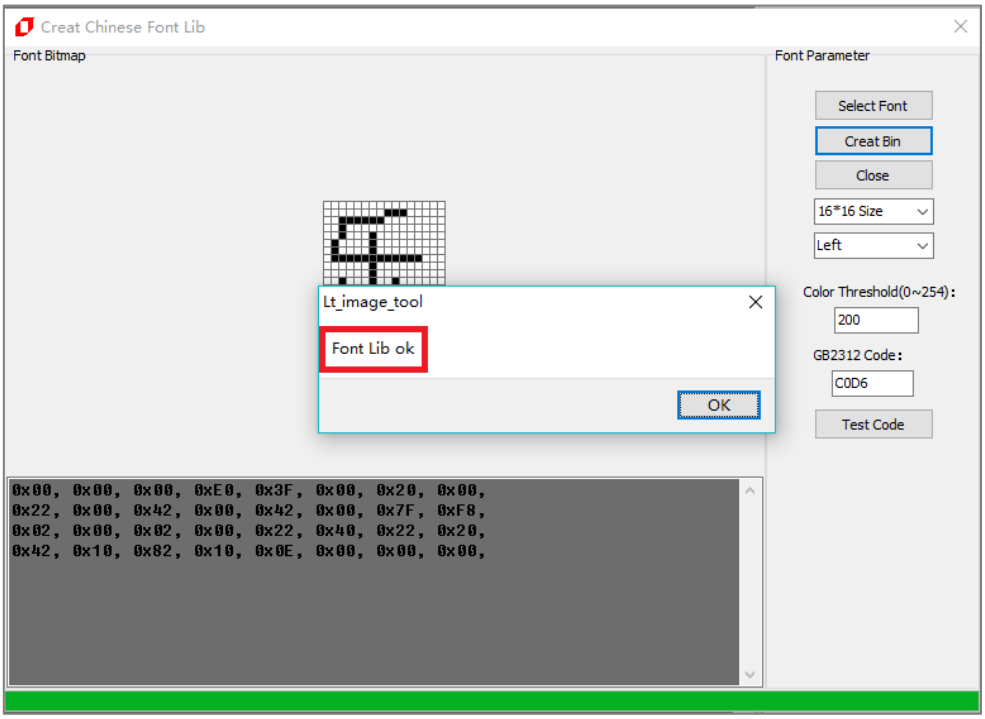


图 11-7：字库制作完成

5、制作完成后可以在目标文件夹中看到导出的 简体 16_16 宋体.bin 文件：

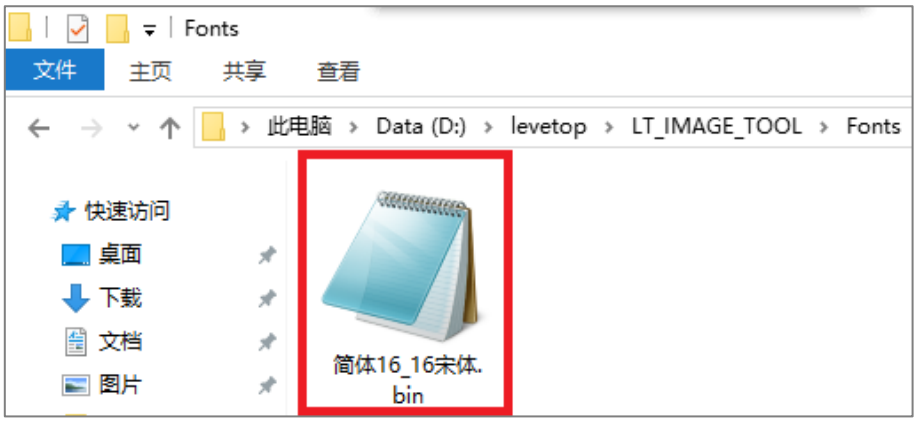


图 11-8：导出的字库 Bin 文件

12. 显示光标

LT768x 提供文字光标与图形光标功能，让用户在需要显示光标的画面时方便使用，增加处理效率及减轻 MCU 负担。

12.1 显示文字光标

文字光标是提供文字写入时的指示，它显示在目前文字可以写入的位置，文字光标的宽度与高度外观是可以被设定的，包括移动位置可以被设成自动累加或是不自动累加，及光标闪烁或是不闪烁。当文字写入时，文字光标会自动累加到下一个文字输入的位置，而每次移动的距离与文字大小与方向有关。当超过工作视窗的边缘时，光标将会移动到下一行，但是要注意光标自动移动功能必须是在工作视窗内。行高的大小可以以像素为单位来设定。下表列出相关的寄存器描述。

表 12-1：文字光标相关的寄存器表

Register Address	Register Name	说　明
REG[03h]	ICR	bit2：图形/文本模式选择（Text Mode Enable）
REG[3Ch]	GTCCR	bit1：文字光标设定（Text Cursor Enable
		bit0：字光标闪烁设定（Text Cursor Blinking Enable）
REG[64h:63h]	F_CURX	光标位置：写入文字时的 X 坐标
REG[66h:65h]	F_CURY	光标位置：写入文字时的 Y 坐标
REG[D0h]	FLDR	设定文字的行距（Character Line Gap Setting）

文字光标可以通过寄存器 CURHS（REG[3Eh]）与 CURVS（REG[3Fh]）去设定高度与宽度。同时文字光标的高度与宽度也会受文字是否被放大（REG[CDh] bit[3:0]）影响，正常显示下光标宽度可以被设为 1 ~ 32 像素；而使用文字放大功能时，光标的宽度与高度将会依倍数放大。下图水平、垂直的文字光标设定：

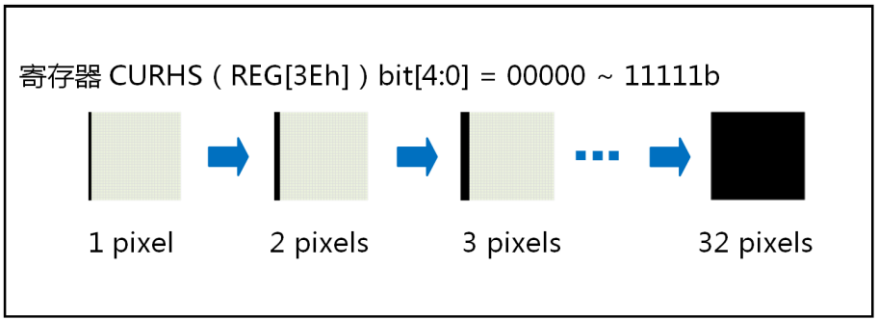


图 12-1：文字光标的宽度

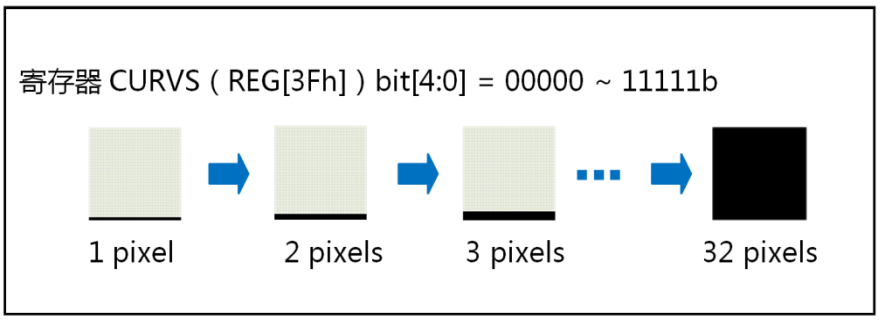


图 12-2：文字光标的高度

文字光标可以设定成固定频率的闪烁或不闪烁，由寄存器 GTCCR（REG[3Ch]）设定，闪烁时的闪烁时间可以被程序化，其计算公式如下：

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

下图光标闪烁的例子中，光标的位置会停留在最后一个写入字的后面。

图 12-3：光标的闪烁范例

在使用文字光标之前需要先初始化，设置好相应的参数，就可以显示出光标了。

```
void LT768_Text_cursor_Init
(
unsigned char On_Off_Blinking,        // 0：禁止光标闪烁；1：使能光标闪烁
unsigned short Blinking_Time,         // 设置文字光标闪烁时间
unsigned short X_W,                   // 文字光标水平大小（最大为 32）
unsigned short Y_W                    // 文字光标垂直大小（最大为 32）
)

void LT768_Enable_Text_Cursor(void);  // 使能文字光标
void LT768_Disable_Text_Cursor(void); // 禁止文字光标
```

举例：要设置一个闪烁的文字光标，且该光标的水平大小为 10，垂直大小为 2：

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

12.2 显示图形光标

LT768x 的图形光标为 32*32 像素组成，而每个像素占用 2 个 bit，用来指定四种颜色：Color-0、Color-1、背景色、背景反向色）：

表 12-2：图形光标像素定义

2’ b00	Color-0（颜色由 REG[44h] 的设定来决定）
2’ b01	Color-1（颜色由 REG[45h] 的设定来决定）
2’ b10	背景色（显示透明）
2’ b11	背景反向色

因此在自建一个图形光标时需要 256bytes 大小。LT768x 提供 4 个图形光标可供选择，MCU 可以经由设定相关的寄存器来选择光标，图形光标位置可由通过 GCHP0（REG[40h]）、GCHP1（REG[41h]）、GCVP0(REG[42h])与 GCVP1(REG[43h])来设定；Color-0 的颜色由寄存器 REG[44h] 设定，Color-1 的颜色则由寄存器 REG[45h] 设定，下图是 32*32 图形光标的储存数据格式的说明范例，在此假设 REG[44h] 的值设定为蓝色，REG[45h] 的值设定为红色。

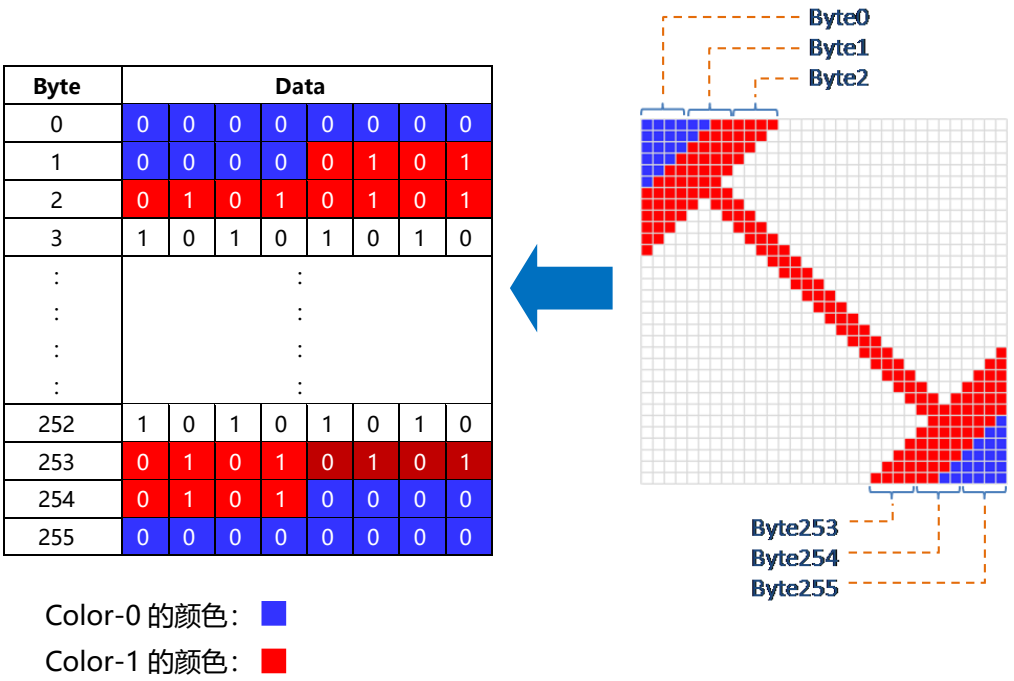


图 12-4：图形光标范例

图形光标需要使用者先自定义，然后写入 RAM 中。总共可以设置 4 个光标，然后再使用的过程中可以很方便的相互切换使用。在使用之前需要先初始化，设置好相应的参数。

```
void LT768_Graphic_cursor_Init
(
unsigned char Cursor_N,          // 选择光标 → 1:光标 1；2:光标 2；3:光标 3；4:光标 4
unsigned char Color1,            // 颜色 1
unsigned char Color2,            // 颜色 2
unsigned short X_Pos,            // 显示坐标 X
unsigned short Y_Pos,            // 显示坐标 Y
unsigned char *Cursor_Buf        // 光标数据的缓冲首地址
)
```

在使用中切换光标的位置。

```
void LT768_Set_Graphic_cursor_Pos
(
unsigned char Cursor_N,          // 选择光标 → 1:光标 1；2:光标 2；3:光标 3；4:光标 4
unsigned short X_Pos,            // 显示坐标 X
unsigned short Y_Pos             // 显示坐标 Y
)

void LT768_Enable_Graphic_Cursor(void);      // 使能图形光标
void LT768_Disable_Graphic_Cursor(void);     // 禁止图形光标
```

举例：初始化 4 个的图形光标，且可以循环切换，在不同的位置上显示。

```
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[];
```

假设以上四个数组的就是每个的图形光标的数据。

```
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);
  }
}
```

举例：箭头的光标数据 

```
const unsigned char gImage_pen_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 图形光标产生工具

本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项图形光标制作的功能，可以让使用者在 PC 端绘制图形光标（32*32 像素），然后导出 256bytes 的光标数据，用户再将光标数据拷贝到定义光标数据的程序内，使用者可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明：

12.3.1 制作图形光标

1、 点击【LT_IMAGE_TOOL 菜单>Cursor】即可打开图形光标制作界面：

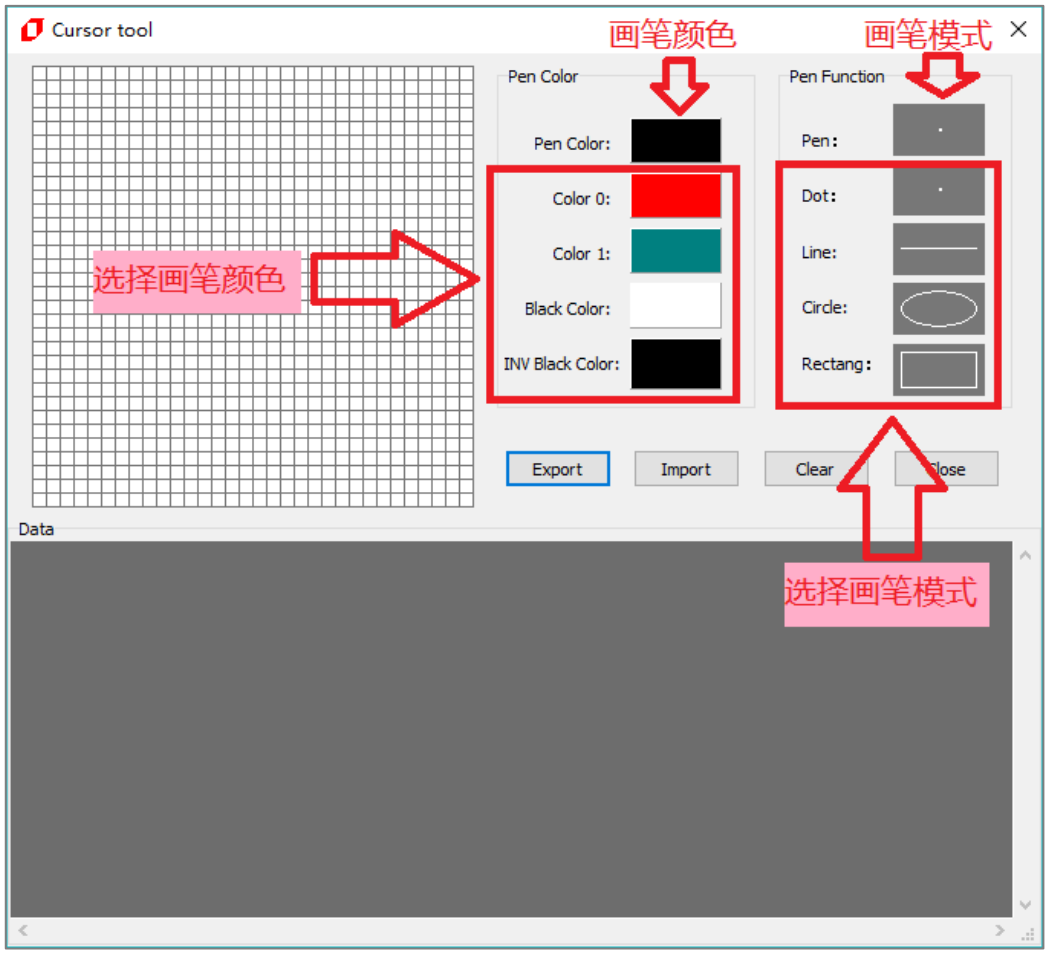


图 12-5：打开图像光标

2、 画好图形光标后，点击【Export】按钮，即可看到图形光标数据，位于 Data 区域。

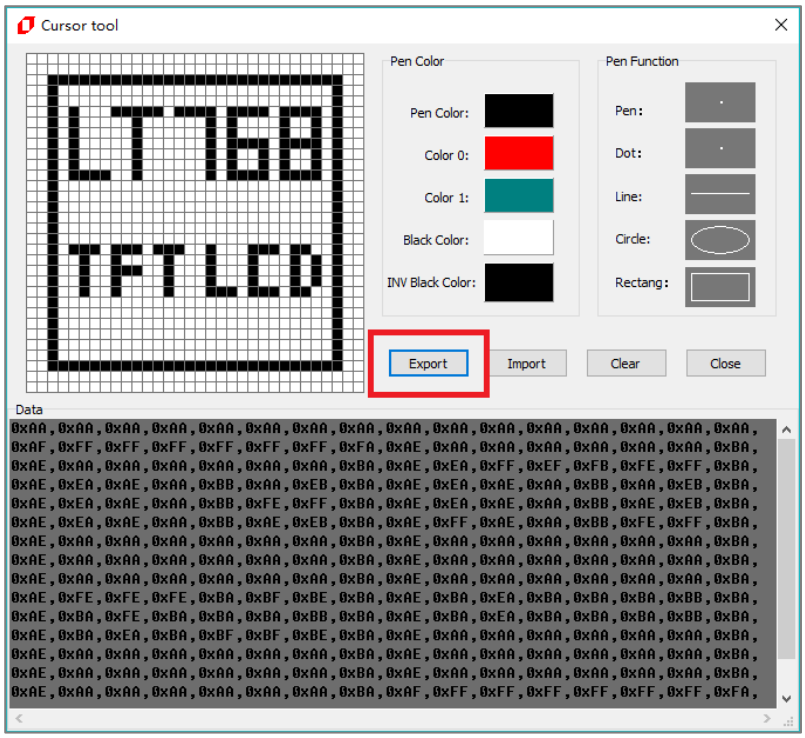


图 12-6：导出图形光标数据

3、 复制图形光标数据：

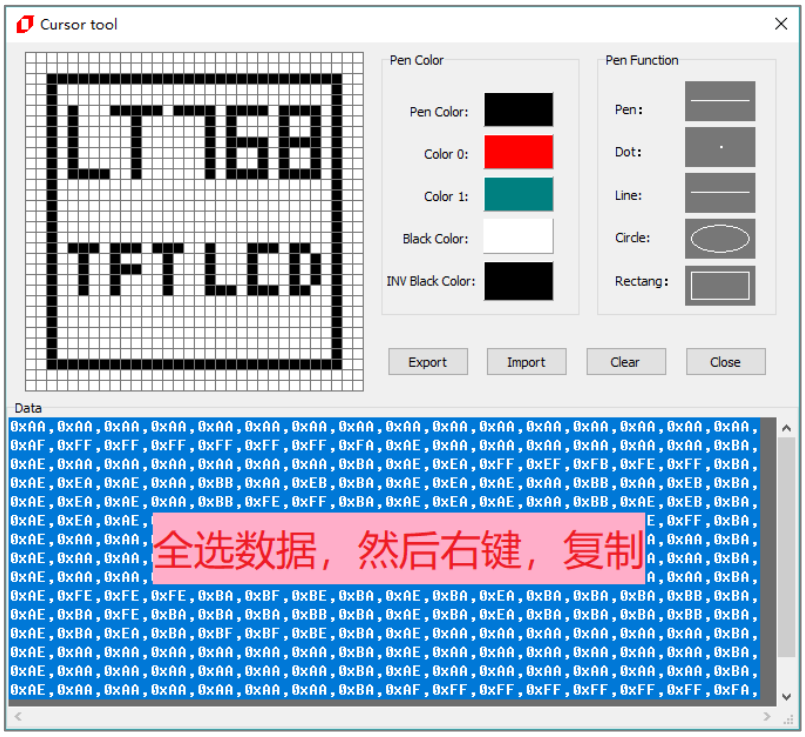


图 12-7：复制导出的光标数据

4、 粘贴复制的光标数据到程序内：

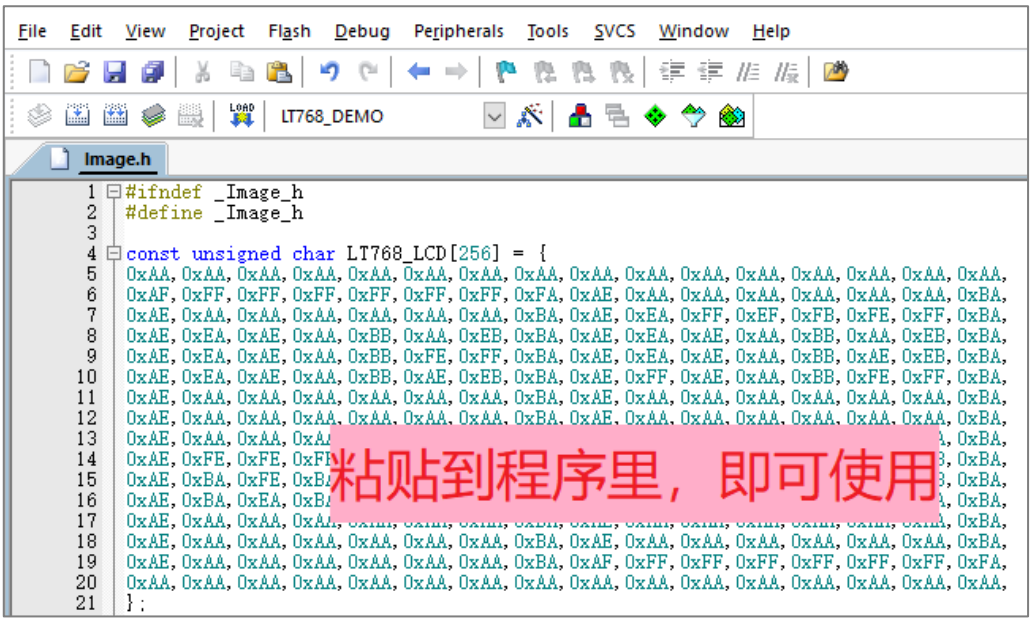


图 12-8：粘贴导出的光标数据

12.3.2 导入及修改图形光标

1. 复制要导入的光标数据：

图 12-9：复制导入的光标数据

2. 粘贴到图形光标制作界面的 Data 内后，点击【Import】按钮

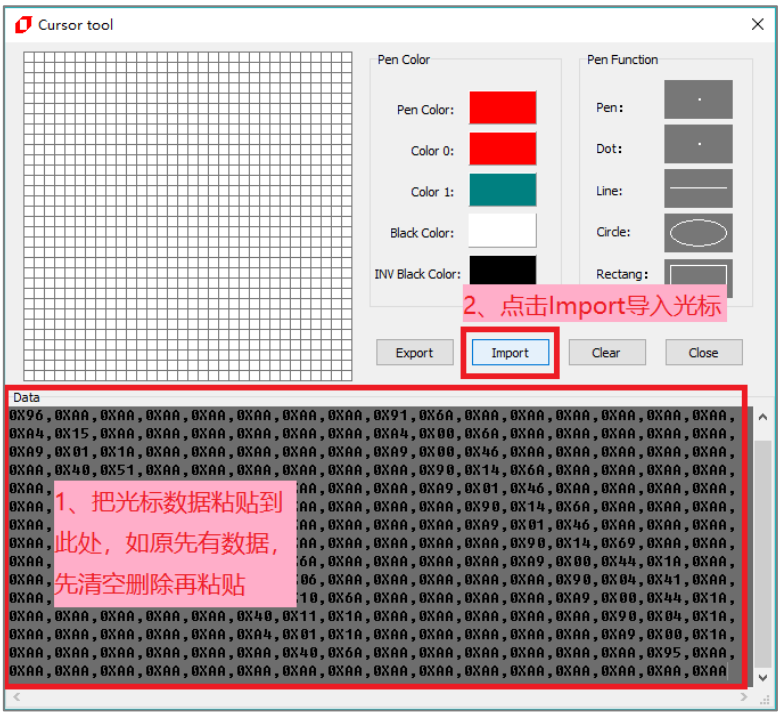


图 12-10：粘贴导入的光标数据

3. 如正确显示要导入的光标，表示导入成功，可自行修改，然后按照第 12.3.1 节的步骤导出光标数据。

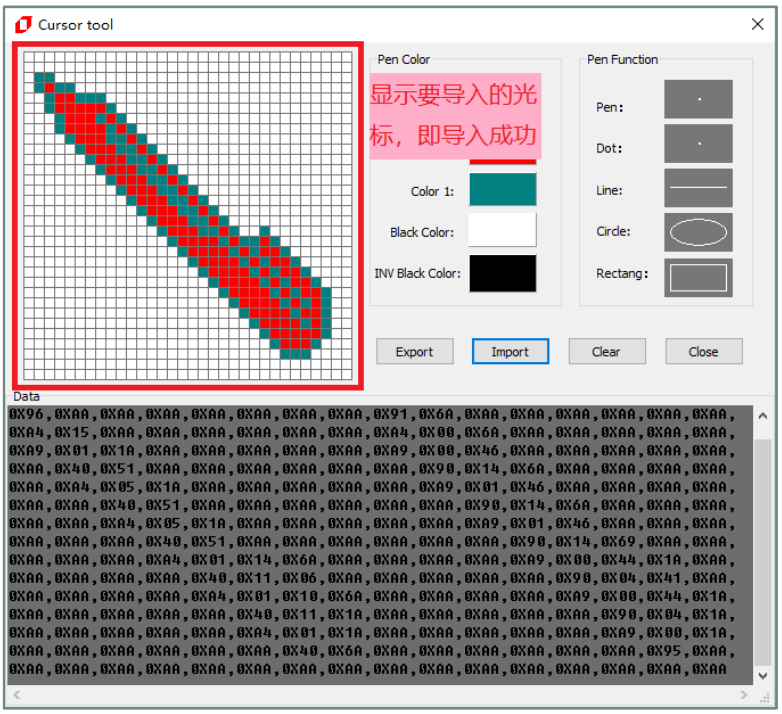


图 12-11：导入成功

13. PWM 设定

LT768x 内建 PWM 功能，并且提供 2 个 PWM 输出：PWM0、PWM1。LT768x 内部含有 2 个 16bits 的计数器 Timer-0 和 Timer-1，其动作关系着 PWM 的输出状态。以 PWM0 为例，在使用前必须先设置 Timer-0 计数寄存器（REG[8Ah-8Bh]，TCNTB0）及 Timer-0 计数比较寄存器（REG[88h-89h]，TCMPB0]），启动 PWM 功能后，Timer-0 计数器会先加载 TCNTB0 的值，并依照 PWM Clock 的设定频率开始下数，当 Timer-0 计数器下数的值等于 TCMPB0 寄存器的值时 PWM 就会动作，也就是 PWM0 如果原本为 0 就转态为 1，而 Timer-0 计数器依然会继续下数，当 Timer-0 继续下数等于 0 时会产生中断，PWM0 回到原本的准位 0，同时会自动加载寄存器 TCNTB0 的值，这就是代表一个完整的 PWM 周期。

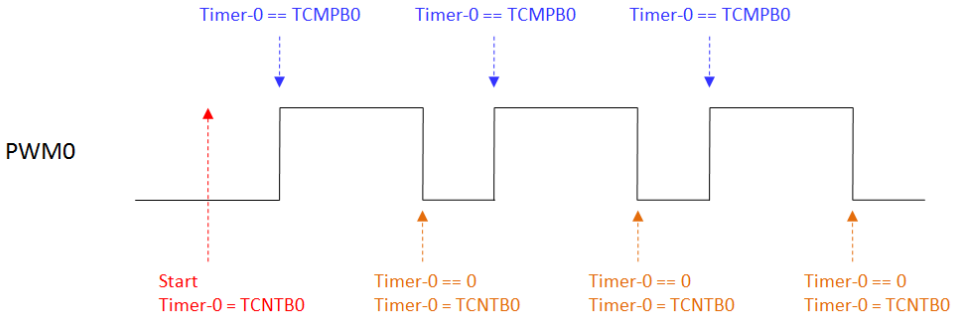


图 13-1：PWM 波形图

由以上动作可以了解，PWM0 的 Duty 决定于比较寄存器（REG[88h-89h], TCMPB0]），例如想要借由 PWM0 产生一个近似 DC 准位的电压，当 PWM0 初始设定为 0，想要产生的等效电压高，那么 TCMPB0 值就要设大一点；当 PWM0 初始设定为 1，想要产生的等效电压高，那么 TCMPB0 值就要设小一点，因此可以借由 PWM 产生的近似 DC 准位电压来控制 TFT 屏的背光。

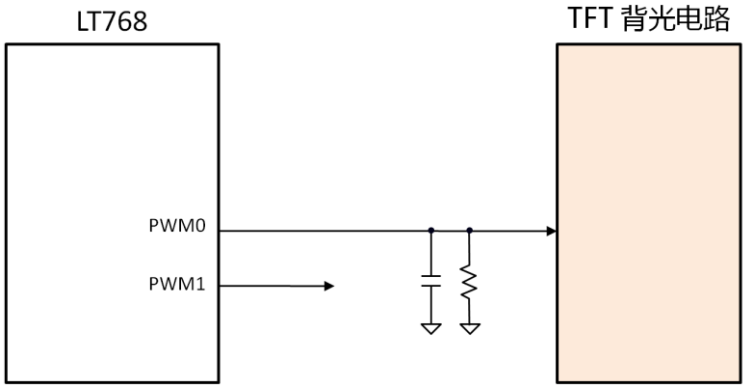


图 13-2：PWM 控制接口

在使用 PWM 之前需要对此先进行初始化。

```
void LT768_PWM0_Init
(
unsigned char on_off,                  // 0：禁止PWM0；1：使能PWM0
unsigned char Clock_Divided,           // PWM时钟分频；取值范围 0~3(1, 1/2, 1/4, 1/8)
unsigned char Prescalar,               // 时钟分频；取值范围 1~256
unsigned short Count_Buffer,           // 设置PWM的输出周期
unsigned short Compare_Buffer          // 设置占空比
)

void LT768_PWM1_Init
(
unsigned char on_off,                  // 0：禁止 PWM1；1：使能 PWM1
unsigned char Clock_Divided,           // PWM 时钟分频；取值范围 0~3(1, 1/2, 1/4, 1/8)
unsigned char Prescalar,               // 时钟分频；取值范围 1~256
unsigned short Count_Buffer,           // 设置 PWM 的输出周期
unsigned short Compare_Buffer,         // 设置占空比
)
```

在初始化完之后，要改动占空比时，只需调用以下对应的函数。

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

举例：假设内核的时钟为 100Mhz，要使用 PWM1 来输出 50Khz 的 PWM 波，且占空比为 30%：

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

要修改 PWMI 的占空比为 50%：

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

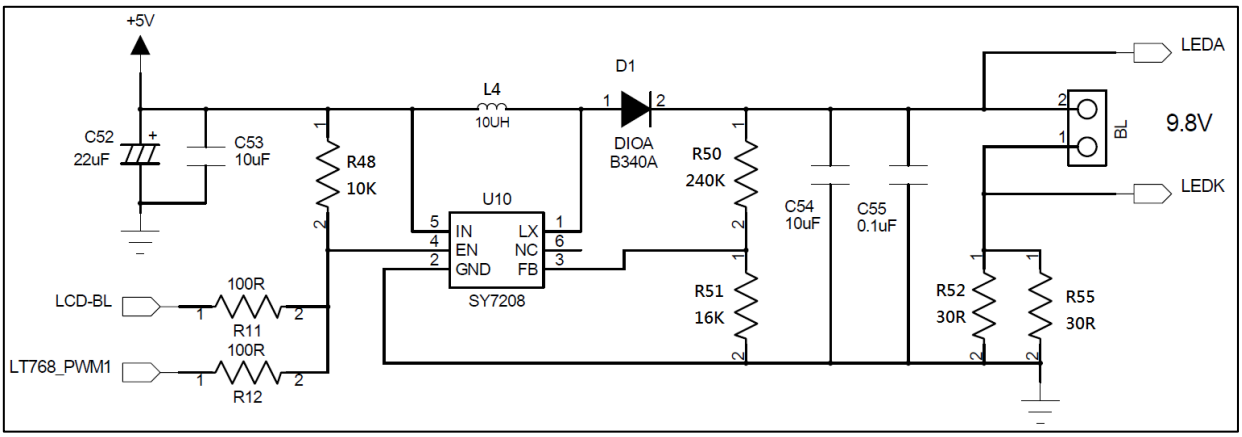


图 13-3：PWM 控制背光参考原理图（7”1024*600 屏）

14. 开机显示

LT768x 的开机显示模块有内建一小型微处理器，主要的功能是在没有外部主处理器的情况下，或是主处理器还在起始运行阶段，在开机时借由执行储存在闪存中的程序代码来迅速显示画面。欲采用此功能可以在 PWM[0] 引脚接上一个 10K 左右的上拉电阻，那么「开机显示」功能就会被使能 (Enable)，如图 14-1 所示，在此功能被使能的情形下，LT768 开机后会自动执行直到闪存中的程序代码被完全执行，也就是执行到 EXIT 指令或未定义的指令，然后就会将主控权交由外部的主处理器。开机显示功能限制程序代码与显示数据必须存在相同的闪存中。开机显示功能模块支持 12 种指令，指令功能如下：

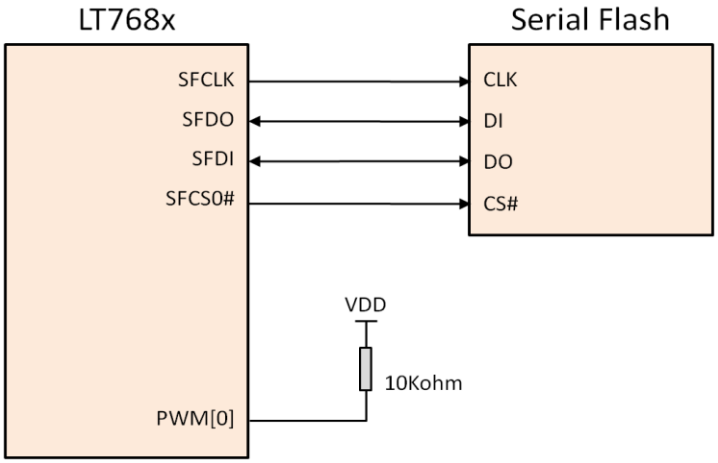


图 14-1：开机显示功能的设定

表 14-1：12 种开机显示功能指令

No.	指 令	指 令 说 明	指令长度 (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

在开机复位后，开机显示功能会针对 LT768x 提供的两个 SPI 接口搜寻，而 0000h ~ 0007h 前 8 个 byte 必须是“61h, 72h, 77h, 63h, 77h, 62h, 78h, 67h”，如果闪存被辨识出那么后续处理的地址将会是（0008h），否则会将主控权交由外部 MCU。LT768x 内部的微处理器从快闪记体的地址 0008h 开始执行指令，如果有遇到 EXIT 指令或未定义的指令才会将控制权交由外部的 MCU。以下为一个从外挂的 Flash（0x200 地址起），读取一张 1024*768 的图片在 LCD 屏上显示的范例。

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

//初始化 PLL
11   05   13   8A        // REG_WR  ('h05, 'h8A) 向寄存器 REG[05]  写入 0x8A
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       // 空指令

//初始化显示内存
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       // 空指令

//设置 LCD 屏
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)

//设置主窗口
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       // 空指令

//设置底图
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)

//设置活动窗口
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)

//设置 DMA 从 FLAHS 传输数据到显示内存
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       // 空指令
11   B6   12   00       // REG_WR  ('hB6, 'h00)
11   12   13   40       // REG_WR  ('h12, 'h40)

00                      // 离开
```

使用限制条件：开机显示功能、限制程序代码与显示数据、字型或其它被为程序代码所需要的数据，都必须存在同一个闪存中。如果使用者需要切换到另一个闪存中，那么相关的程序代码与数据都必须由另一个闪存得到。

14.1 设置开机启动加载程序

本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项设置开机启动加载程序的功能，可以让使用者在 PC 端更轻易的产生开机显示功能的程序代码，使用者可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明：

1、点击【LT_IMAGE_TOOL 菜单>Bootloader】即可打开开机启动设置界面：

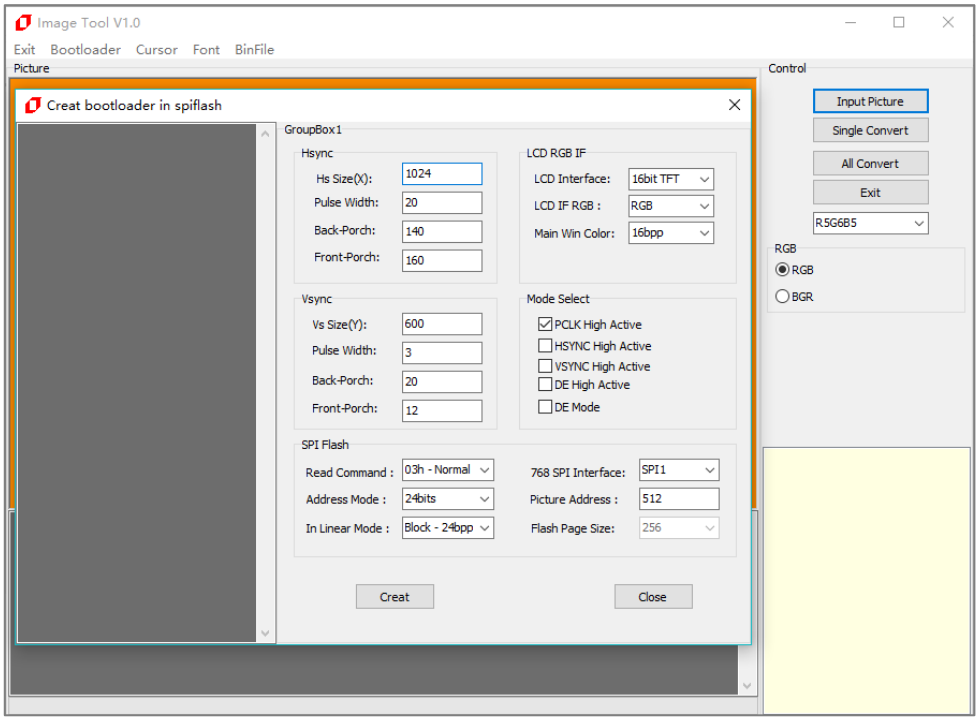


图 14-2：Bootloader 设置

2、设置屏幕参数：Hsync、Vsync、Mode Select：根据不同 LCD 屏幕设置不同的参数，需参照屏幕资料来修改。

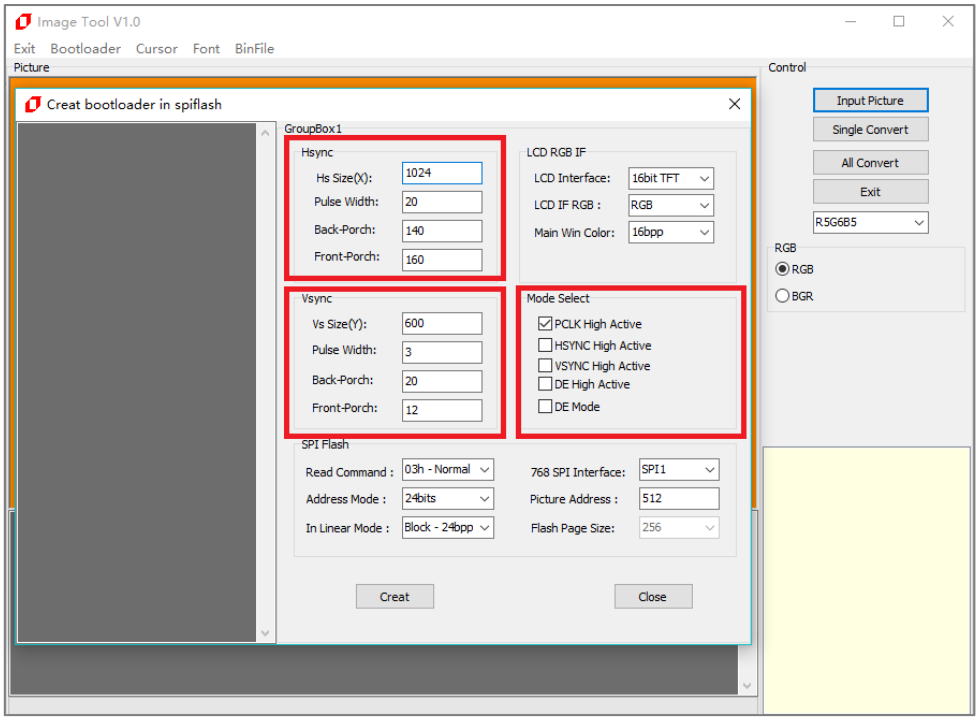


图 14-3：设置屏幕参数

3、设置 LCD 接口：

A. LCD Interface：选择 16、18、24bits CMOS 接口面板

B. LCD IF RGB：选择 RGB 颜色排列

C. Main Win Color：选择显示色度 24bpp（RGB 8:8:8）、16bpp（RGB 5:6:5）、或者是 8bpp（RGB 3:3:2）

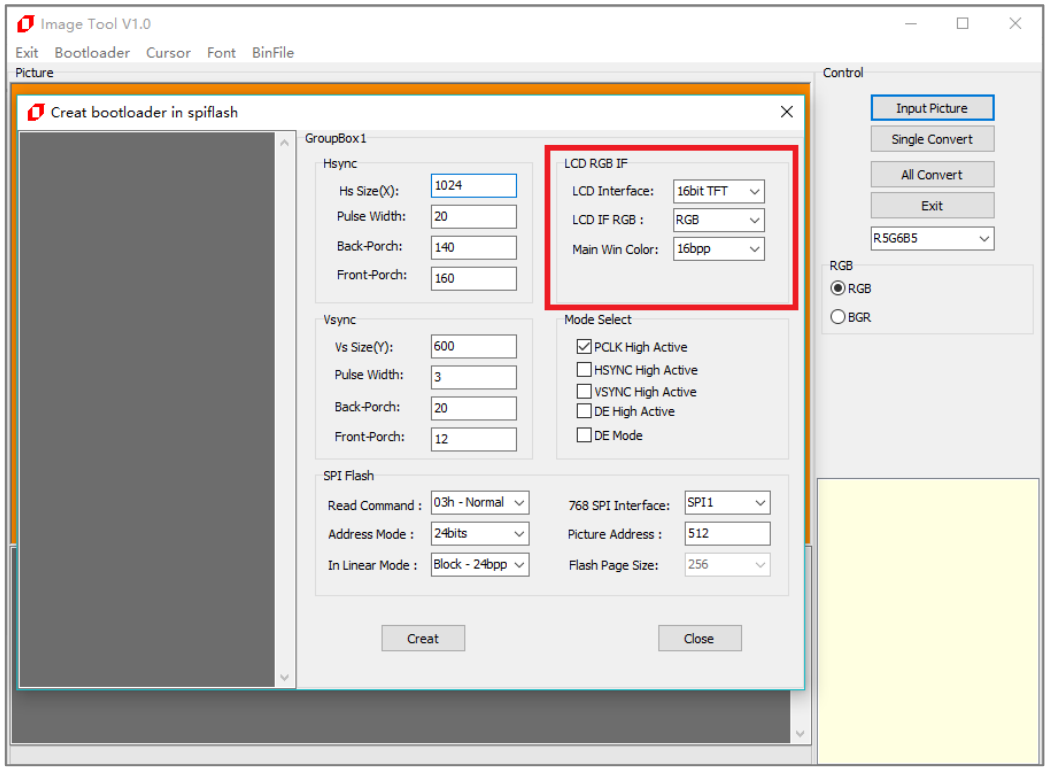


图 14-4：设置 LCD 接口

4、设置 SPI Flash 参数：

A. Read Command：选择 SPI Flash 读取方式

B. 768 SPI Interface：选择 LT768 的 SPI 接口：SPI-0 或 SPI-1

C. Address Mode：选择寻址方式：24bits 或 32bits

D. Picture Address：设定开机显示图片的 Flash 地址

E. In Liner Mode：选择内存显示深度

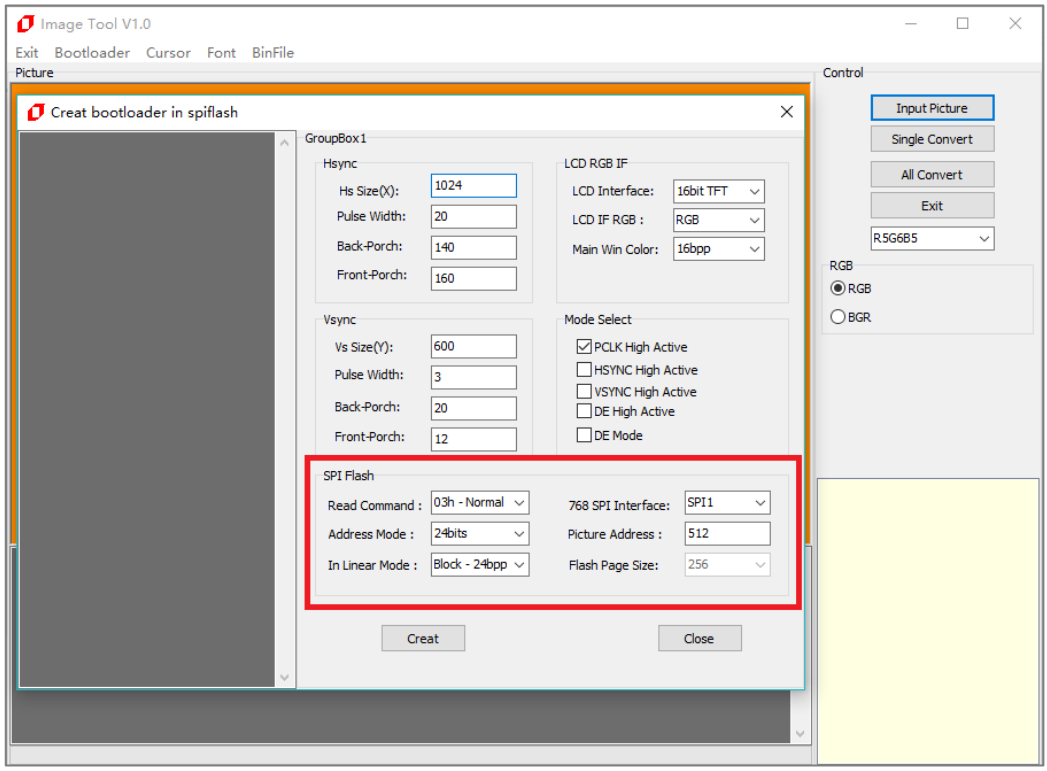


图 14-5：设置 SPI Flash 参数

5、以上参数都设定好后，点击【Create】按钮另存为 Bin 文件，注意输入文件名时文件名中不能包含下面这些字符，如：? * / \ < > : " |，否则无法保存。

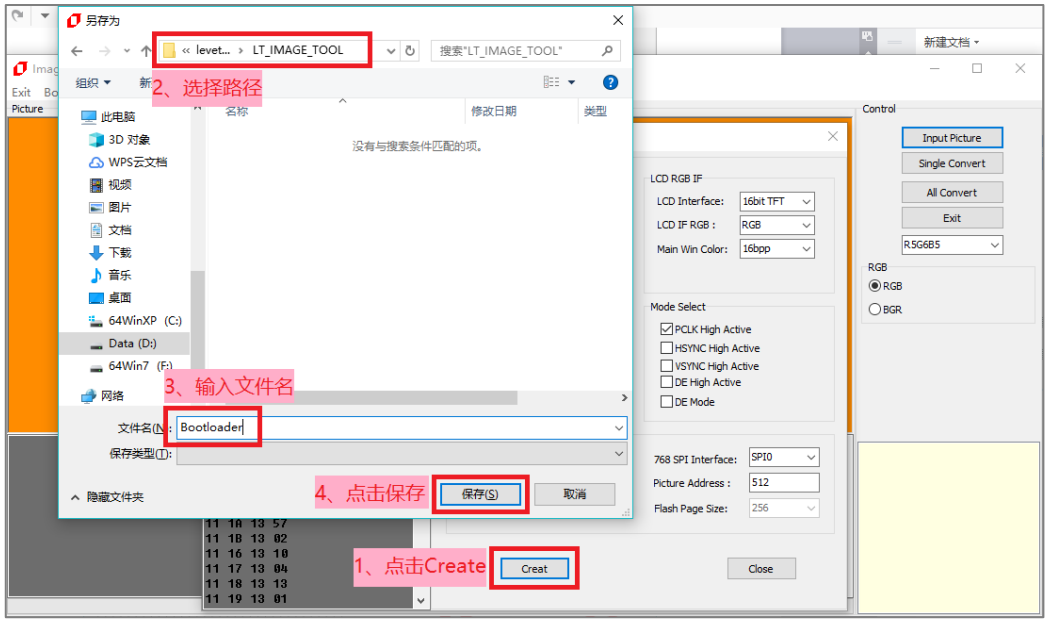


图 14-6：保存

当显示 ok 时，即保存成功。

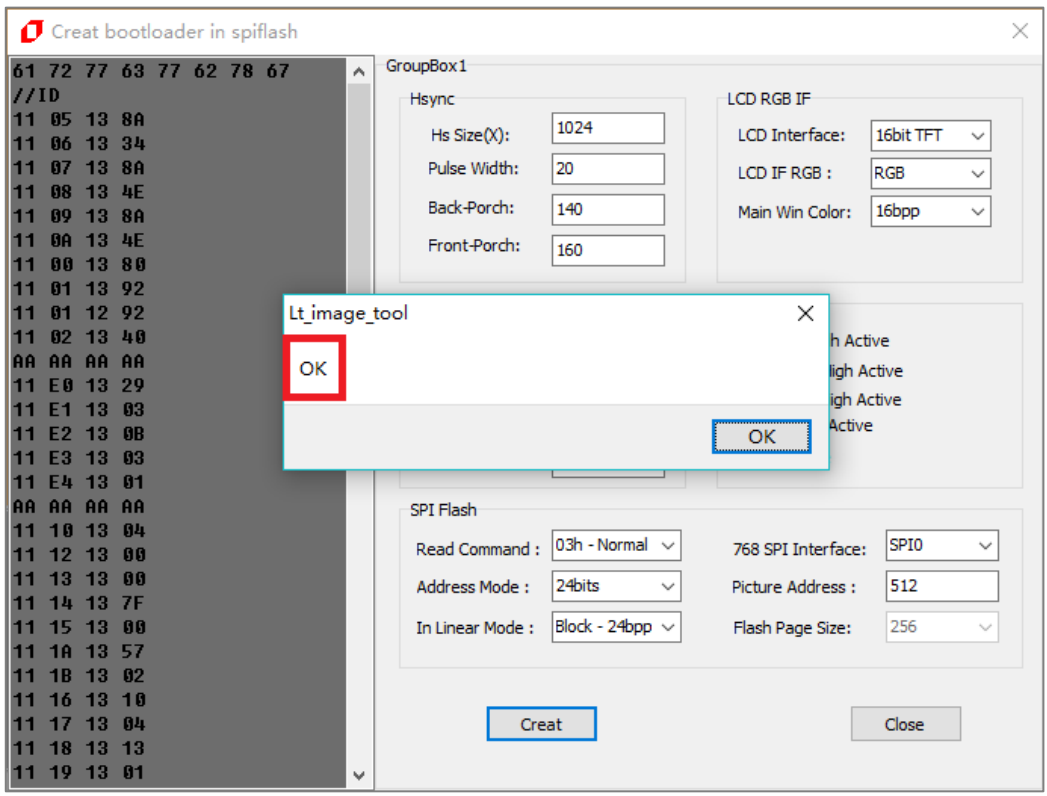


图 14-7：保存成功

6、保存后可以在目标文件夹中看到导出的 Bootloader.bin 文件：

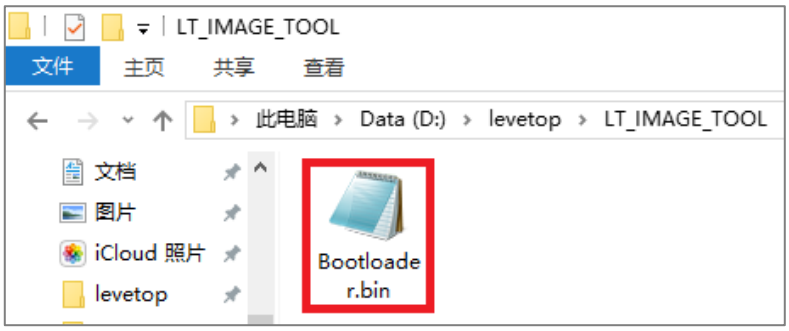


图 14-8：导出的 Bootloader.bin 文件

15. SPI Master

LT768x 的 SPI Master 主要是用于控制外部 SPI 串口闪存（Serial Flash）或是 SPI 串口元件（如触控芯片），对 SPI 串口闪存支持的协议有 4-BUS（Normal Read）、5-BUS（FAST Read）、Dual Mode 0、Dual Mode 1 与 Mode 0/Mode 3。闪存/ROM 功能可以被文字模式与 DMA 模式使用。文字模式表示外部的闪存/ROM 储存的是文字的 bitmap 图文件。DMA 模式表示外部的闪存储存的是 DMA（Direct Memory Access）的数据，通常是储存图档，使用者可以使用 DMA 加快显示数据由闪存传送至显示内存中，而这个处理过程不需要 MCU 的处理。至于外接 SPI 触控芯片或是 I2C 触控芯片可以参考的 16 章的说明。

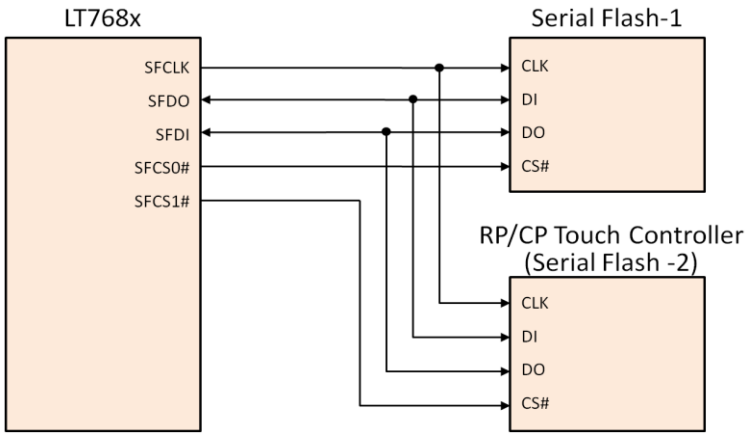


图 15-1：LT768x 串行 Flash/ROM 应用电路

15.1 串行闪存的 DMA 传输

串行闪存有两种的 DMA 传输方式：线性传输模式（Linear）和区块传输模式（Block）。这两种的传输目的都是一样的，把串行闪存下的数据传输到显示内存中。但是他们在传输的原理上是有所差别的：

- **线性传输模式**：在线性模式下，数据是在片选信号 CS 一拉低，就一直传输，直到传输的数据量到达所设定的值，这时才会拉高片选信号 CS。
- **区块传输模式**：在区块传输模式下，片选信号 CS 一拉低，只要每次传输的数据量达到所设定的宽度乘上设定颜色深度的字节数，就会被拉高，然后又被拉低，依次类推，直到数据全都传输完成，最后片选 CS 被拉高。

该 DMA 传输有两种寻址方式 - 24 位和 32 位的，具体要使用哪种寻址得看所使用的串行闪存的操作时序。

举例 1：外挂的串行闪存是 128Mbit 的华邦 W25Q128 型号（NOR Flash）。

该 Flash 正常读取数据的格式：当 CS 拉低后，主机给 Flash 发送 03H+24 位的地址+传输的数据，然后 CS 拉高，这样的读取数据格式，就可以用 24 位寻址的传输函数了。

举例 2：外挂的串行闪存是 1Gbit 的华邦 W25N01GVZEIG 型号（NAND Flash）。

该 Flash 读取数据是以页为单位操作的，并且有两种读取方式，当时必须要设置成连续的读取方式，即要先发送一个页地址的命令，然后才发送读取数据的命令，所以这个 Flash 比 W25Q128 读取数据需要多一步操作，需要先发送页命令后，才能在调用 DMA 的传输数据的函数。（具体地操作请看该 Flash 的数据手册）。如果用户的外建中文字库及图片很多，可以采用这种高容量的 NAND Falsh。

以上的两个例子只是简单的说明，具体的操作还需配合Flash的操作时序，这两种寻址在运用上是比较灵活的，有关SPI闪存的存取时序图可以参考规格书第 10.3 节的说明。

15.1.1 串行闪存在线性模式下的 DMA 传输

1. 24 位寻址

该函数支持 LT768 外挂闪存的 24 位寻址，可以把闪存里的数据直接通过 DMA 传输到内存中。一般是用于传输闪存中的字库数据到内存中。

```
void LT768_DMA_24bit_Linear
(
unsigned char SCS,              // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,              // SPI 时钟分频参数：SPI Clock = System Clock /{(Clk+1)*2}
unsigned long flash_addr,       // 要从 Flash 读取数据的起始地址
unsigned long memory_addr,      // 数据要传输到 SDRAM 的起始地址
unsigned long data_num          // 传输的数据量
)
```

举例：24*24 的楷书字体在外部 Flash （Flash 挂在 SFCS0#上）的 0x0078DC20 首地址存起，而且字库数据的大小为 0x0009B520，此时要把字库传输到 SDRAM，且首地址为 x003E537E0。只需这样使用该函数：

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

■ 32 位寻址

该函数支持 LT768 外挂闪存的 32 位寻址，可以把闪存里的数据直接通过 DMA 传输到内存中。一般是用于传输闪存中的字库数据到内存中。

```
void LT768_DMA_32bit_Linear
(
unsigned char SCS,              // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,              // SPI 时钟分频参数：SPI Clock = System Clock /{(Clk+1)*2}
unsigned long flash_addr,       // 要从 Flash 读取数据的起始地址
unsigned long memory_addr,      // 数据要传输到 SDRAM 的起始地址
unsigned long data_num          // 传输的数据量
)
```

举例： 24*24 的楷书字体在外部 Flash （Flash 挂在 SFCS0#上）的 0x0078DC20 首地址存起，而且字库数据的大小为 0x0009B520，此时要把字库传输到 SDRAM，且首地址为 0x003E537E0。只需这样使用该函数：

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

15.1.2 串行闪存在区块模式下的 DMA 传输

■ 24 位寻址

该函数支持 LT768 外挂闪存的 24 位寻址，可以把闪存里的数据直接通过 DMA 传输到内存中。一般是用于传输闪存中的图片数据到内存中。

```
void LT768_DMA_24bit_Block
(
unsigned char SCS,          // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,          // SPI 时钟分频参数：SPI Clock = System Clock /{(Clk+1)*2}
unsigned short X1,          // 内存 X1 的位置
unsigned short Y1,          // 内存 Y1 的位置
unsigned short X_W,         // DMA 传输数据的宽度
unsigned short Y_H,         // DMA 传输数据的高度
unsigned short P_W,         // 图片的宽度
unsigned long Addr          // Flash 的地址
)
```

举例：假设有张 1024*600 的 16 位色的图片在外部 Flash（Flash 挂在 SFCS1#）的 0x80000 首地址存起，此时要把该图片在 LCD 屏（1024*600）上显示出来。

```
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);
```

■ 32 位寻址

该函数支持 LT768 外挂闪存的 32 位寻址，可以把闪存里的数据直接通过 DMA 传输到内存中。一般是用于传输闪存中的图片数据到内存中。

```
void LT768_DMA_32bit_Block
(
unsigned char SCS,          // 选择外挂的 SPI Flash → 0：SPI-0；1：SPI-1
unsigned char Clk,          // SPI 时钟分频参数：SPI Clock = System Clock /{(Clk+1)*2}
unsigned short X1,          // 内存 X1 的位置
unsigned short Y1,          // 内存 Y1 的位置
unsigned short X_W,         // DMA 传输数据的宽度
unsigned short Y_H,         // DMA 传输数据的高度
unsigned short P_W,         // 图片的宽度
unsigned long Addr          // Flash 的地址
)
```

举例：假设有张 1024*600 的 16 位色的图片在外部 Flash（Flash 挂在 SFCS1#）的 0x80000 首地址存起，此时要把该图片在 LCD 屏（1024*600）上显示出来。

```
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 制作图片的 Bin 文件

在应用端会常用到的图片，可以透过 DMA 传输方式将显示数据存到 LT768x 内建的显示内存内，这样可以降低 MCU 处理传送图像数据的负担，使用这个功能可以先将图片转成 Bin 文件，然后预先烧录在 SPI Flash 内，本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项制作图片 Bin 文件的功能，可以让使用者在 PC 端导入图片，然后生成一个图片的 Bin 文件，使用者可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明：

1、打开软件 LT_IMAGE_TOOL.EXE：

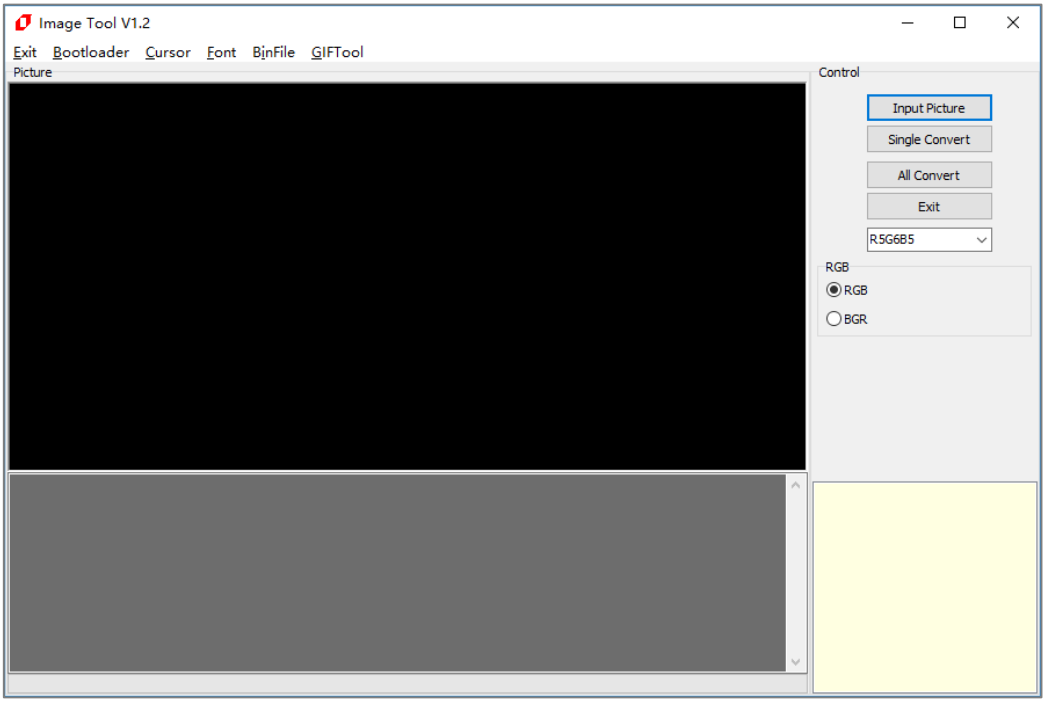


图 15-2：执行 LT_IMAGE_TOOL.EXE

2、导入图片，点击 Input Picture 按钮，选择需要的图片，点击打开，即可添加此文件夹下的所有图片：

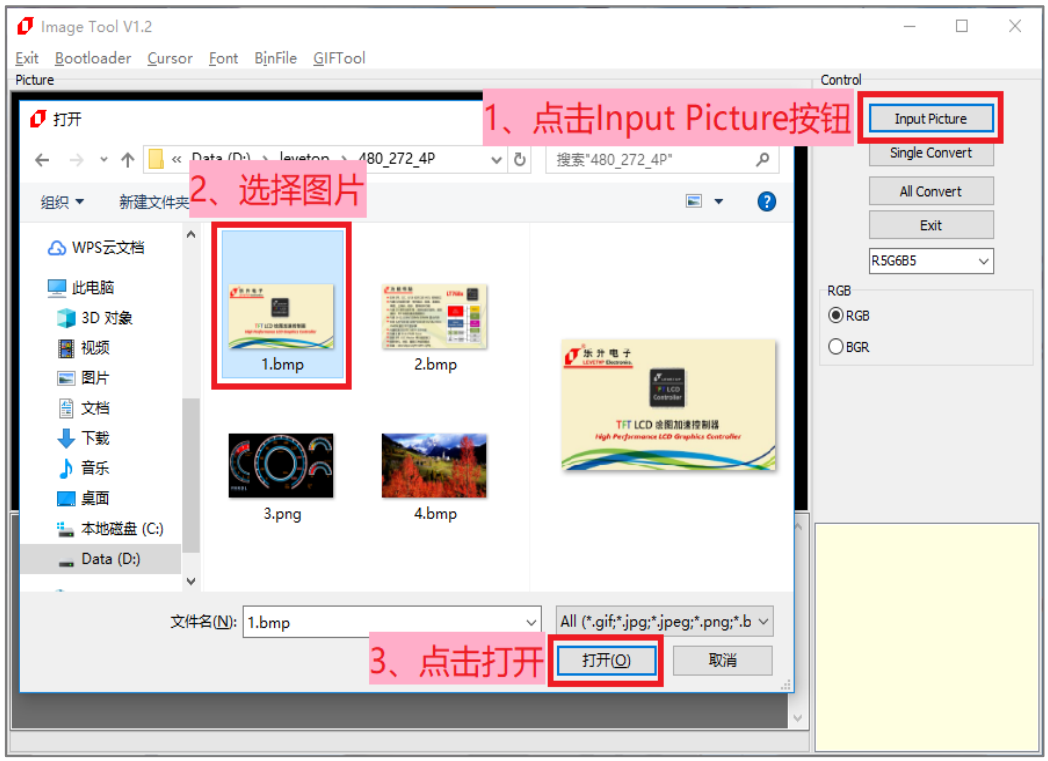


图 15-3：导入图片

图 15-4：导入完成

3、图片输出设定，可选 16bpp 或 24bpp 或 Black_White 格式，以及 RGB 或 BGR 格式：
注意：若需要制作 Black_White 格式的图片 bin 文件，源图片文件必须为只有黑色和白色的图片。

图 15-5：设定输出格式

4、导出某一张图片或导出全部图片，注意输入文件名时文件名中不能包含下面这些字符，如：? * / \ < > : " |，否则无法保存。

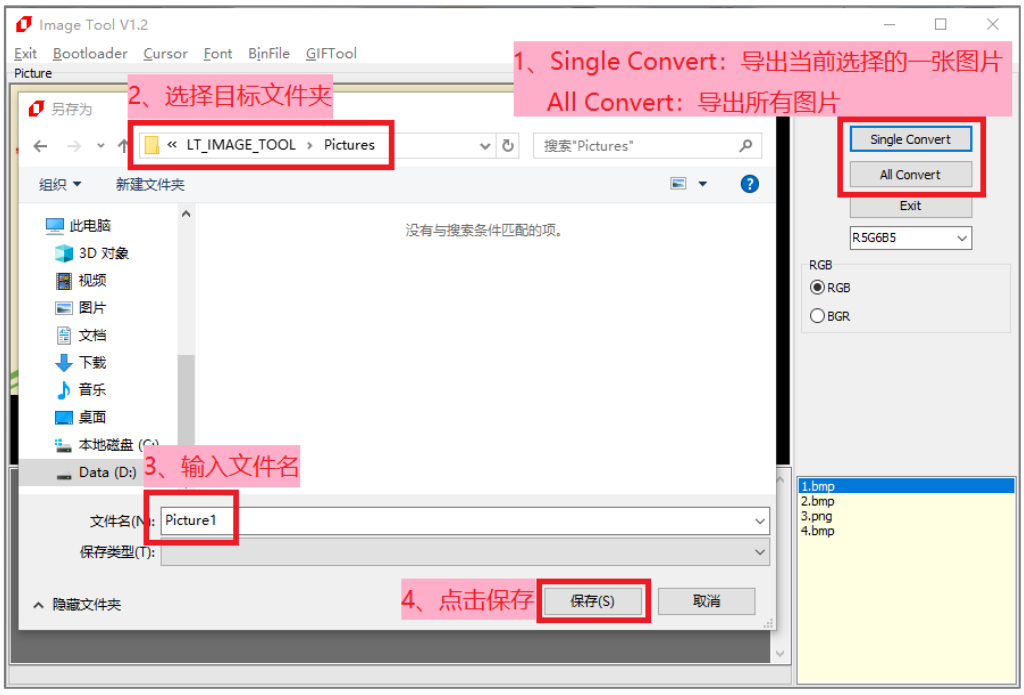


图 15-6：导出图片（1/2）

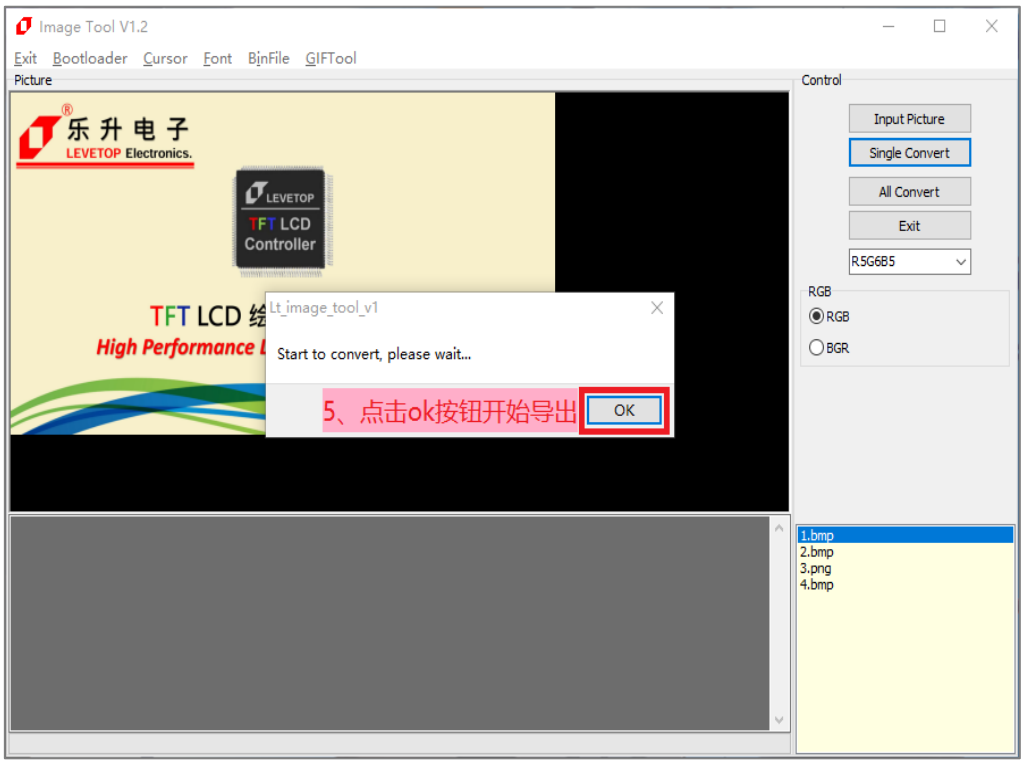


图 15-7：导出图片（2/2）

5、成功导出图片 Bin 文件：

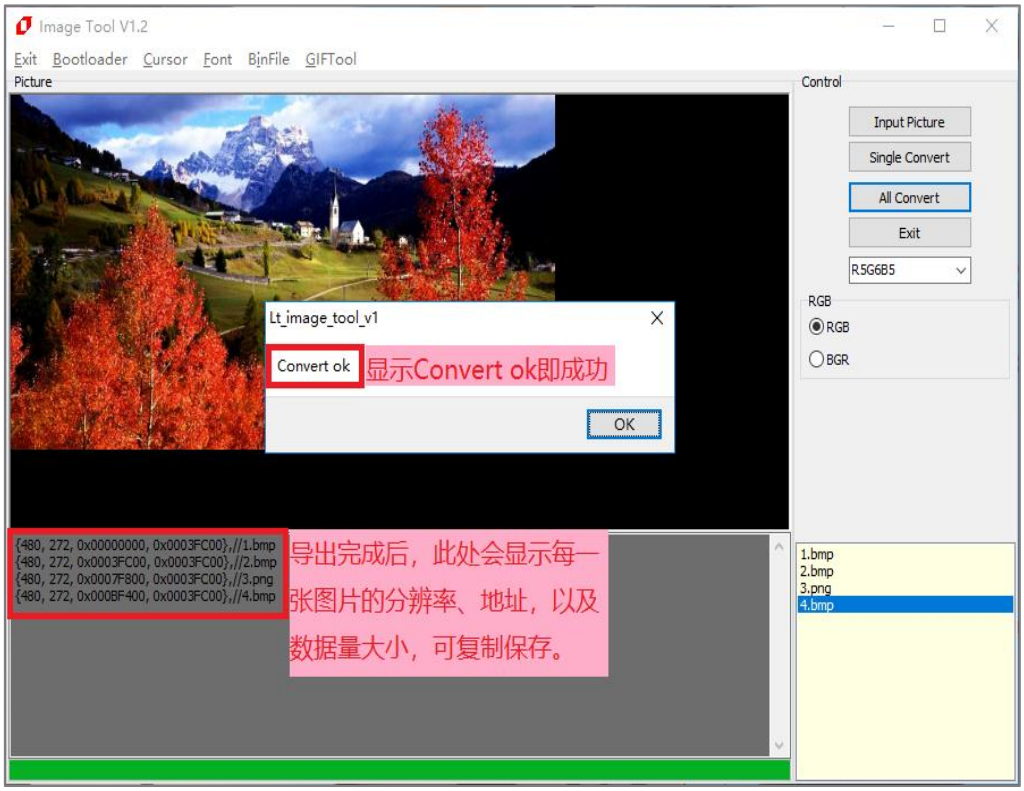


图 15-8：导出成功

6、导出图片后可以在目标文件夹中看到导出的 Picture1.bin 文件：

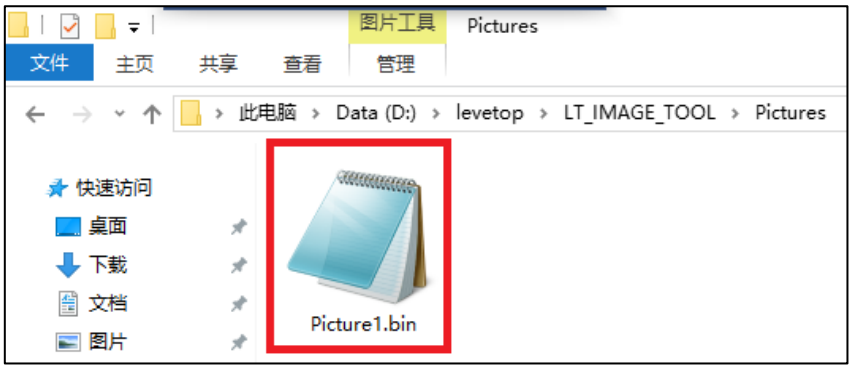


图 15-9：导出的图片 Bin 文件

15.3 制作 GIF 檔的 Bin 文件

应用端如果要拨放 GIF 的动画，可以将 GIF 档案切割成许多图片，然后透过 DMA 传输方式将这些图片分时存到 LT768x 内建的显示内存，这样就可以达到显示动画的效果。使用者可以用专用程序 LT_IMAGE_TOOL.EXE 来导入 GIF 的动画档案，然后生成一连串图片的 Bin 文件，详细步骤可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明：

1. 点击【LT_IMAGE_TOOL 菜单>GIFTool】即可打开 GIF Bin 文件制作界面：

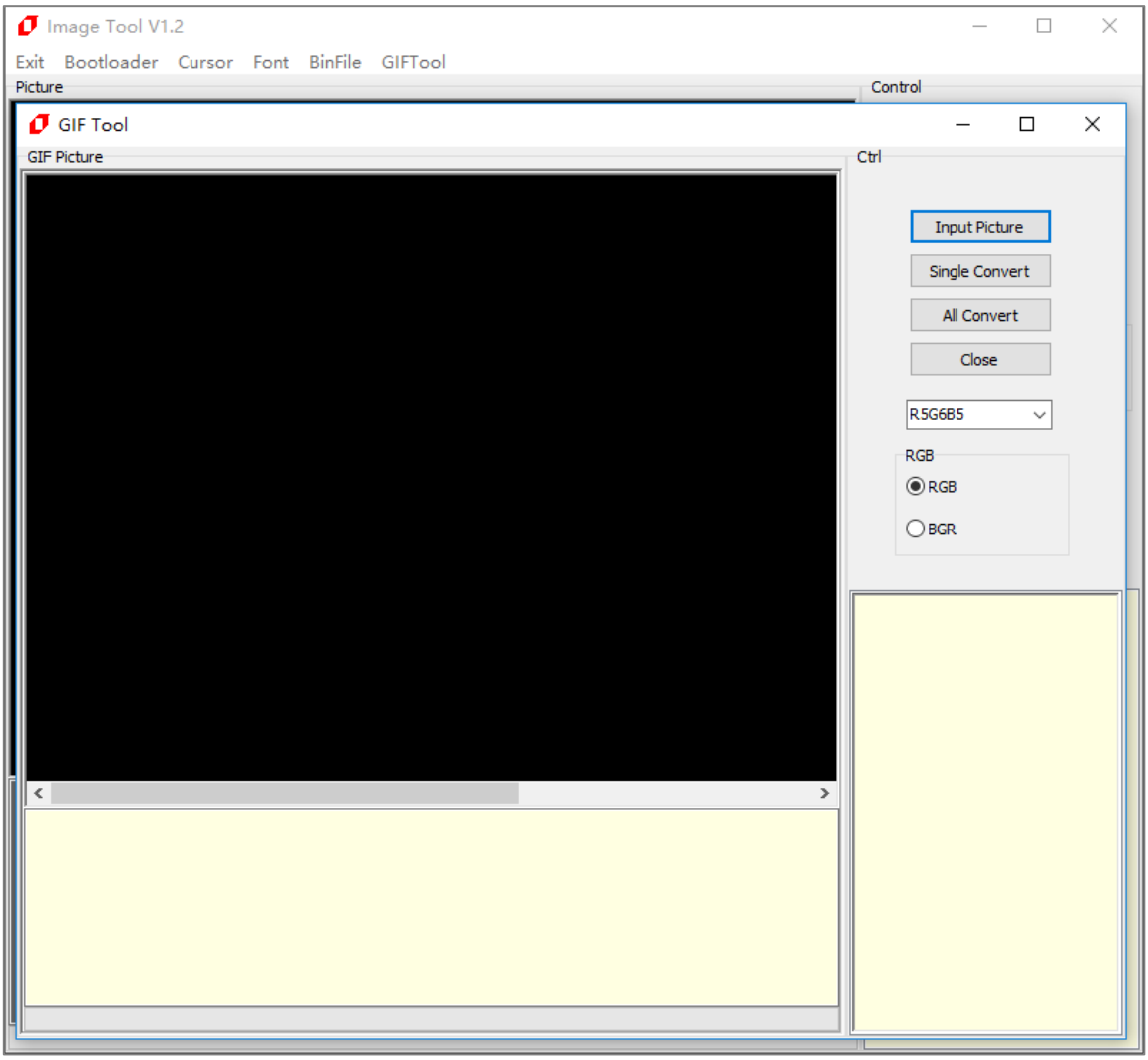


图 15-10：打开 GIF Bin 文件制作界面

2. 导入 GIF 图片，点击 Input Picture 按钮，选择需要的 GIF 图片，点击打开，即可添加此文件夹下的所有 GIF 图片：

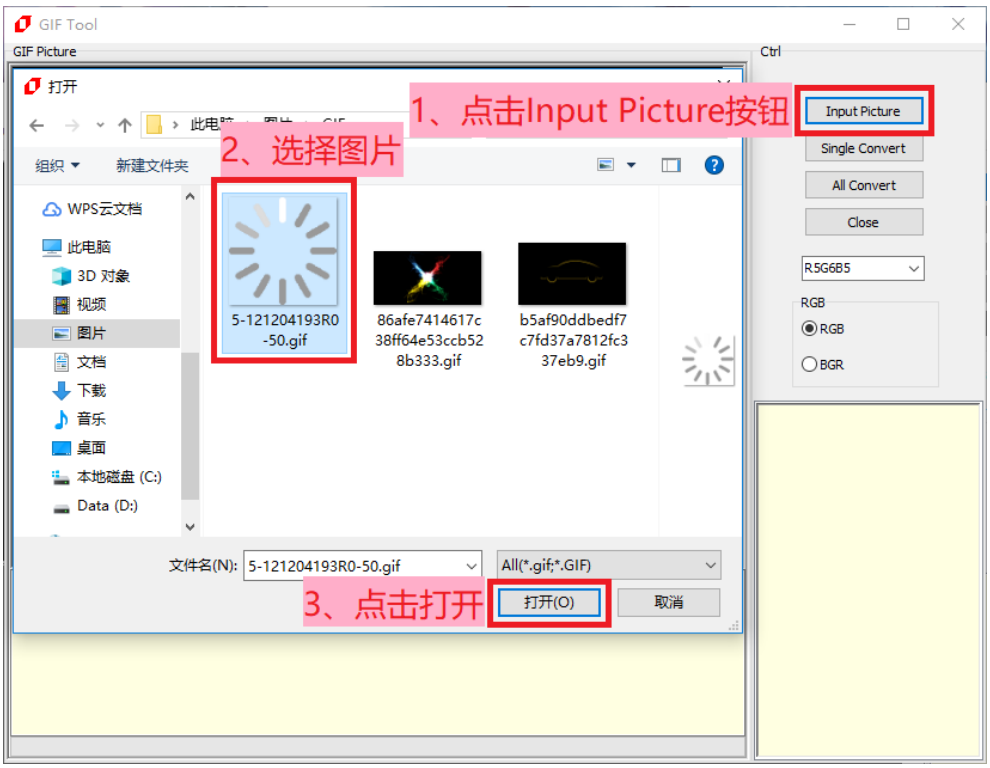


图 15-11：导入图片

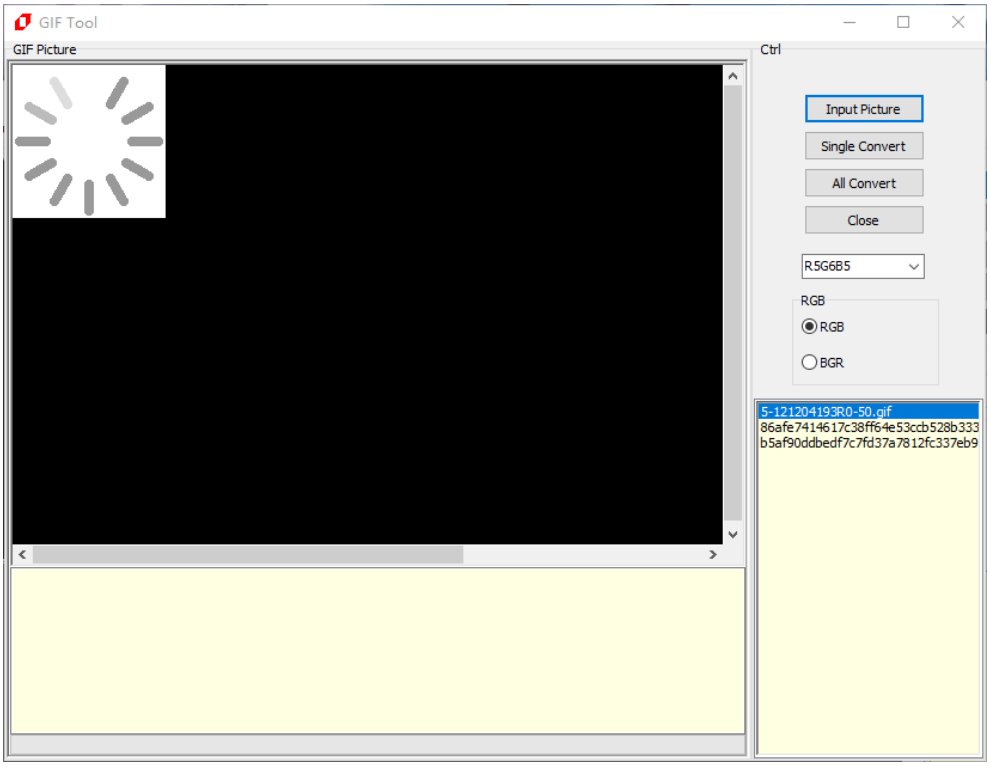


图 15-12：导入完成

3. GIF 图片输出设定，可选 16bpp 或 24bpp 格式，以及 RGB 或 BGR 格式：

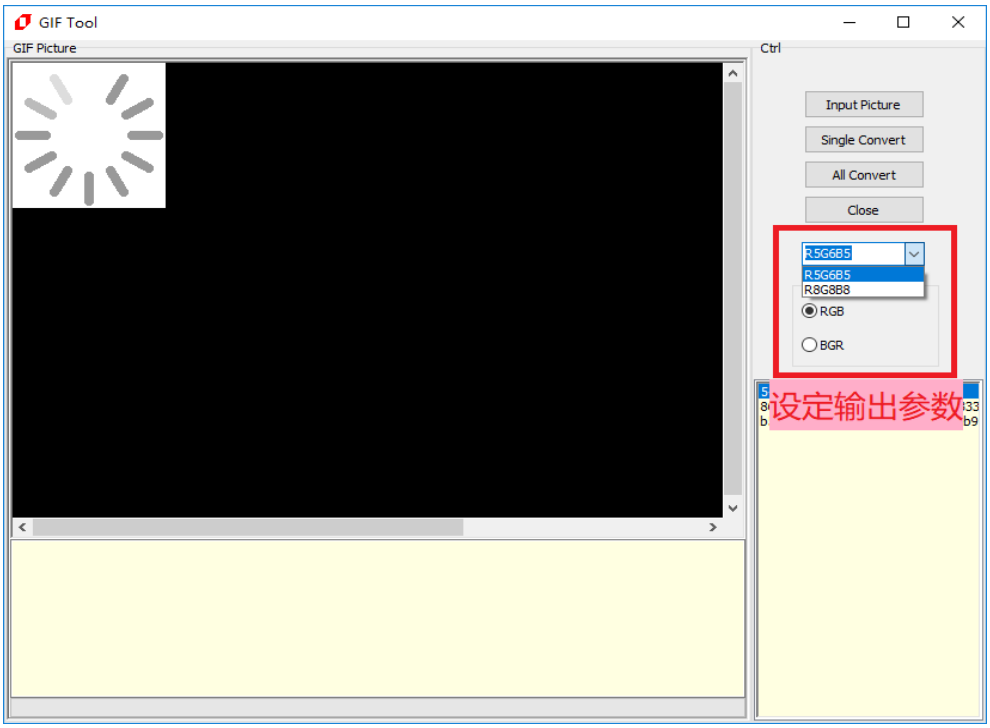


图 15-13：设定输出格式

4. 导出某一张 GIF 图片或导出全部 GIF 图片，注意输入文件名时文件名中不能包含下面这些字符，如：？ * / \ < > : " |，否则无法保存。

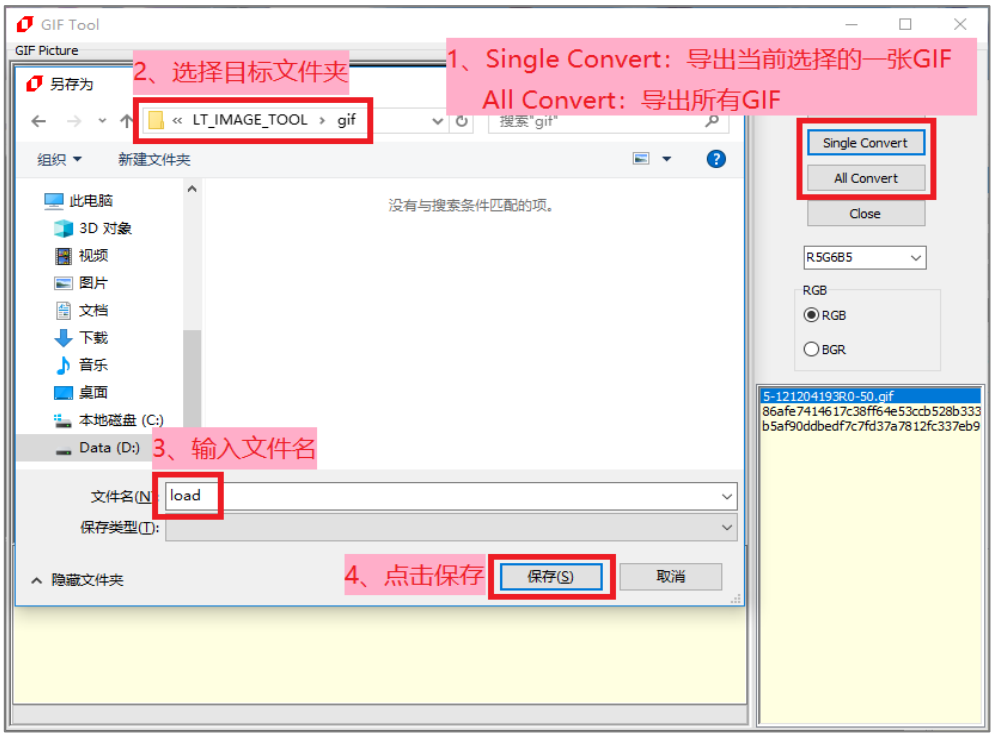


图 15-14：导出图片（1/2）

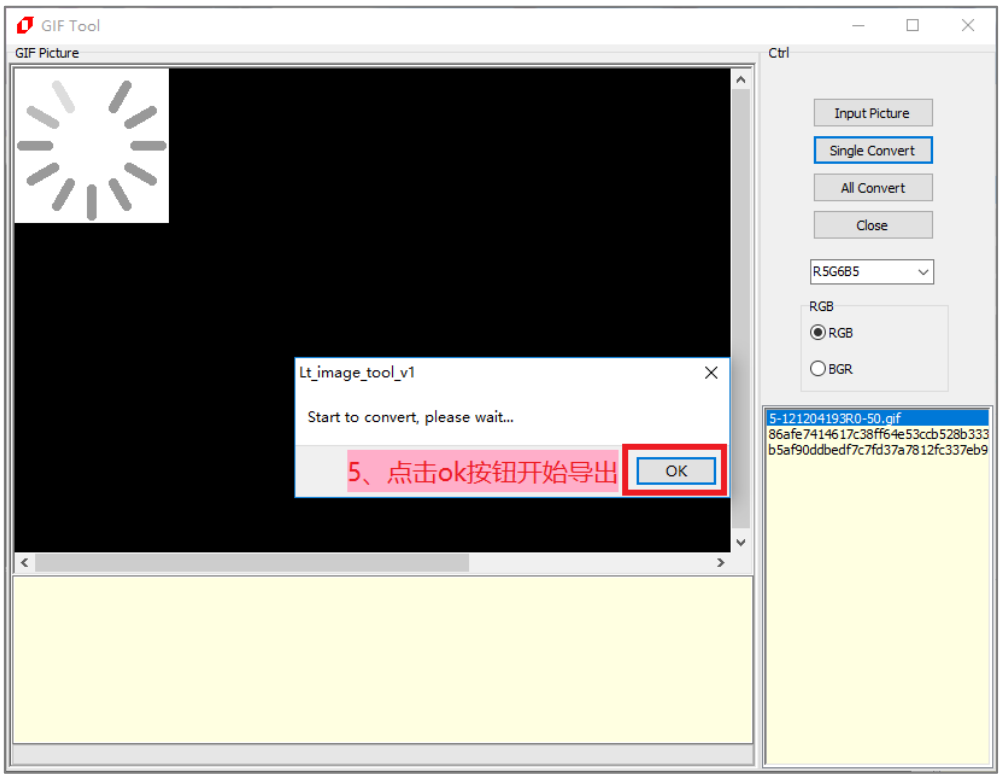


图 15-15：导出图片（2/2）

5. 成功导出图片 Bin 文件：

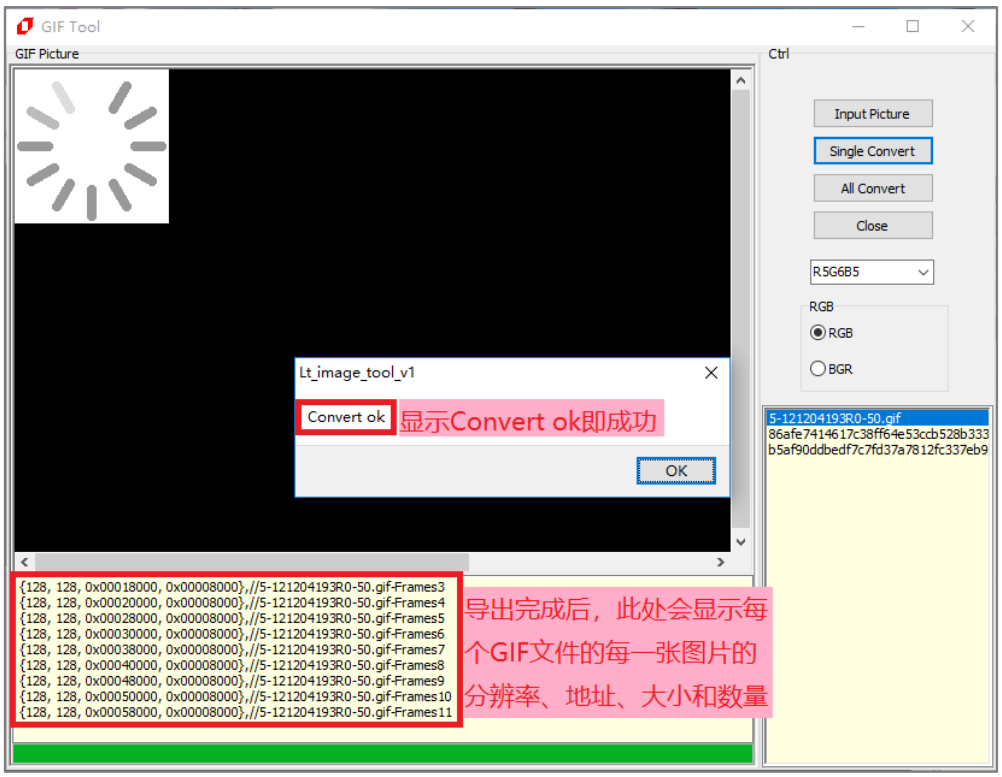


图 15-16：导出成功

6. 导出图片后可以在目标文件夹中看到导出的 load.bin 文件以及 txt 详情文档：

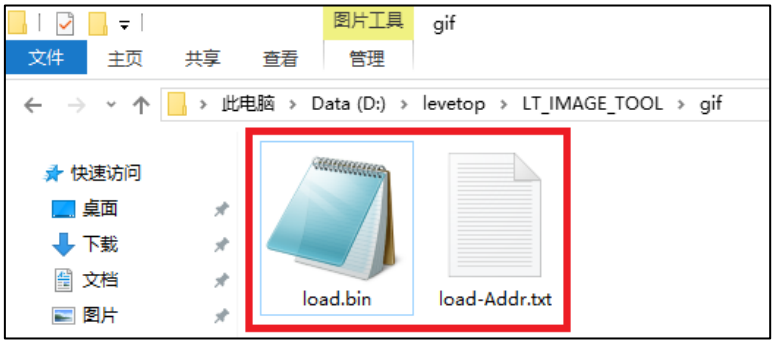


图 15-17：导出的图片 Bin 文件

在 txt 文档内能看到 Gif 内每张图片的分辨率、地址、大小以及图片数量：

图 15-18：导出的每张图片信息

15.4 Bin 文件的结合

在 10.3 节及提到制作串行闪存的字库 Bin 文件，实际应用可以将图片 Bin 文件与字库 Bin 文件结合成一个 Bin 文件，烧录在 SPI Flash 内，本公司提供的专用程序 LT_IMAGE_TOOL.EXE，里面有一项 Bin 文件整合的功能，可以让使用者在 PC 端将不同的 Bin 文件结合成一个 Bin 文件。详细步骤可以参考 LT_IMAGE_TOOL.EXE 的使用说明书，或是以下的说明：

1、点击【LT_IMAGE_TOOL 菜单>Binfile】文件整合界面，最多能整合 6 个 Bin 文件，点击【File 1~6】可依次添加。注意，Bootloader.bin 文件需放在地址 0 位置，即 File 1。

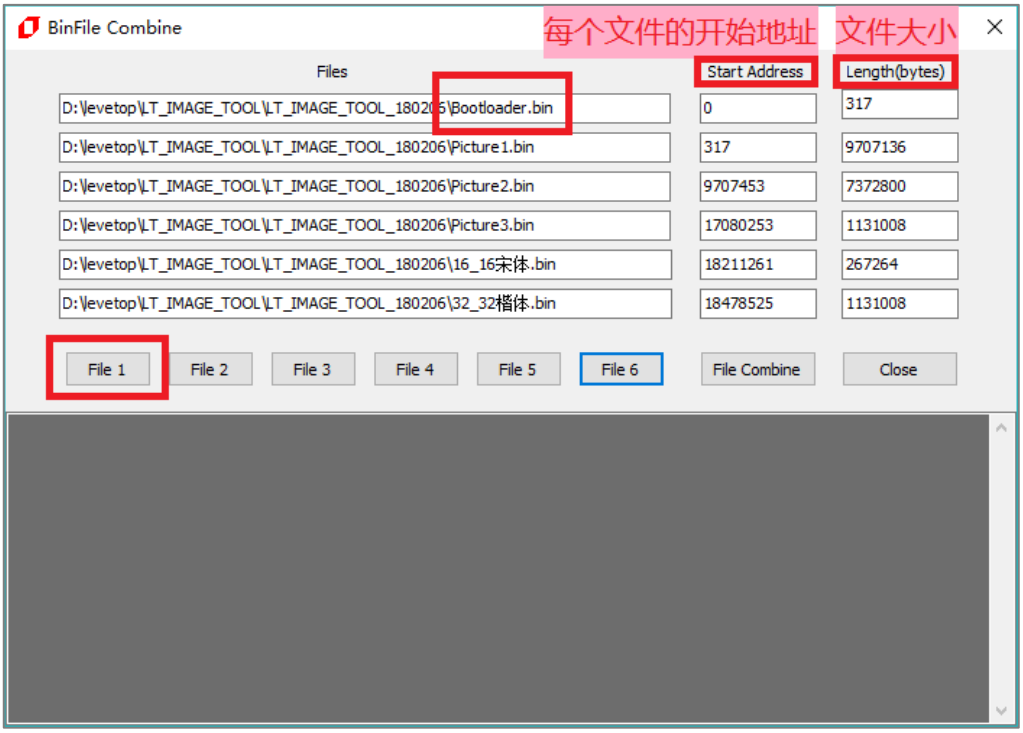


图 15-19：Bin 文件整合

2、点击【File Combine】按钮保存整合文件，注意输入文件名时文件名中不能包含下面这些字符，如：？ * / \ < > : " |，否则无法保存。

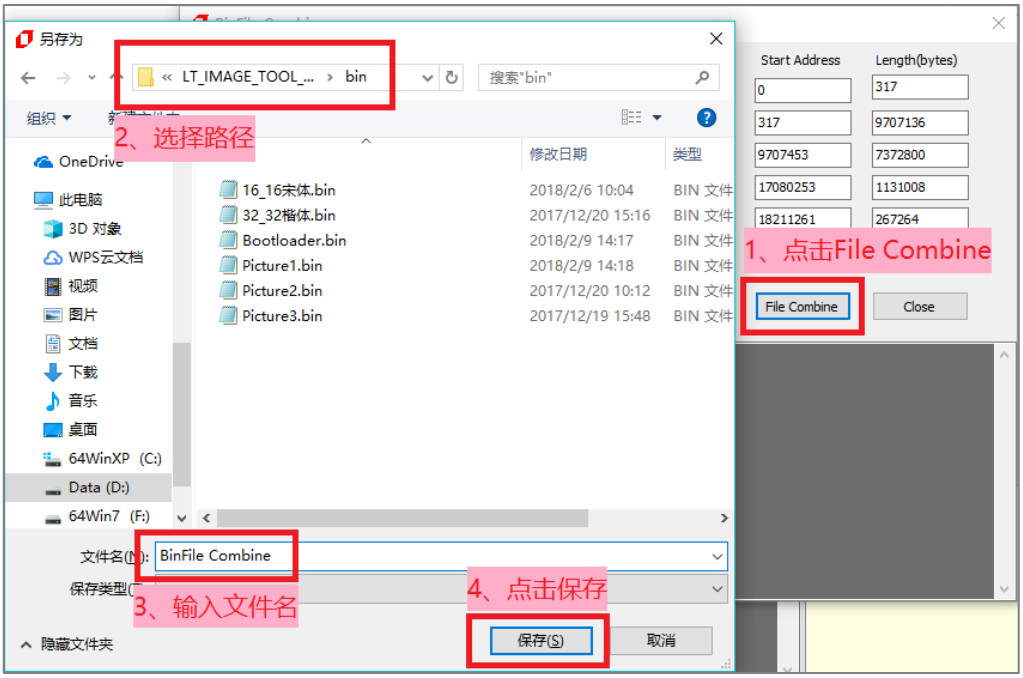


图 15-20：保存整合文件

当显示 Combine over 时，即整合成功，并显示每个源文件的地址和大小，同时生成一个 BinFile Combine-Addr.txt 文件，便于查阅每个源文件的地址、大小等详细信息。

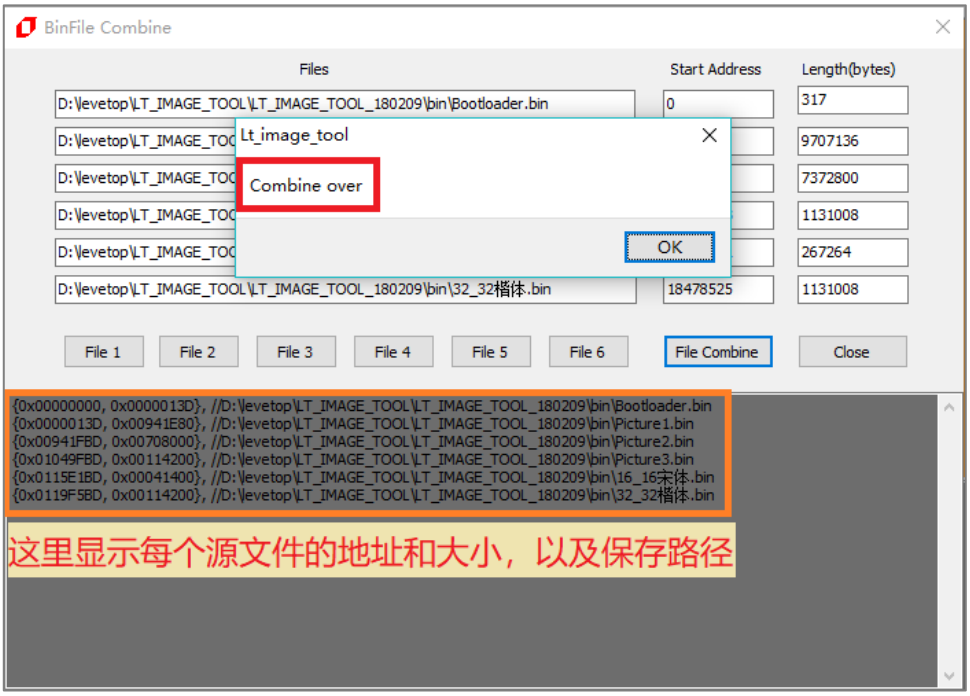


图 15-21：整合成功

3、生成的 BinFile Combine-Addr.txt 文件：

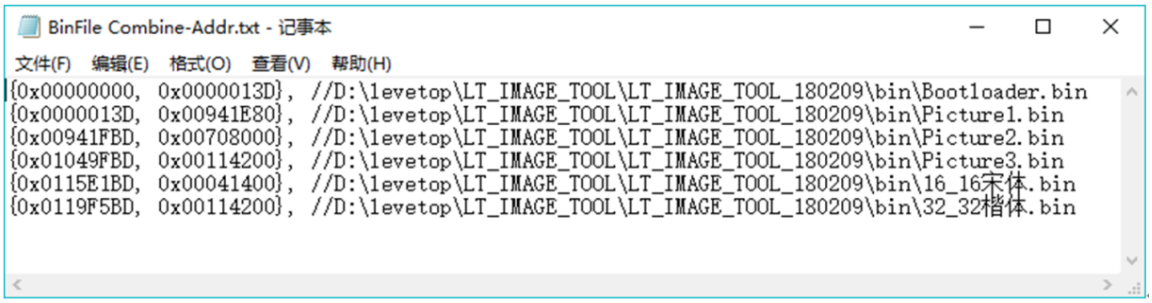


图 15-22：保存文件信息

整合完成后可以在目标文件夹中看到导出的 BinFile Combine.bin 文件，然后使用者可以用 SPI Flash 烧录器将此档案烧录到连接至 LT7868x 的 SPI Flash。

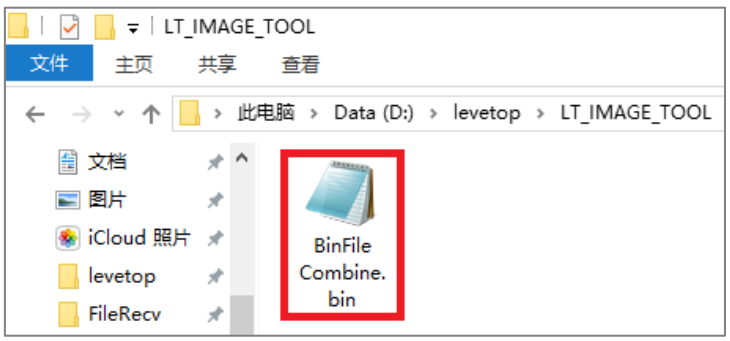


图 15-23：导出的 Bin 整合文件

15.5 程序如何调用 SPI Flash 的 Bin 文件

以下用 2 个例子说明程序如何调用已经烧录到 SPI Flash 内的 Bin 文件数据，一个是在程序中显示图片“Picture1”和“Picture2”，另一个是在程序中调用 SPI Flash 内的字库（16_16 标楷体）。

范例一：使用 DMA 的方式从 LT768 的 SFCS0# (SPI-0) 中外挂的 Flash 中读取第一张“Picture1”和第二张图片“Picture2”，并循环显示，而这 2 张图在 Flash 内是连续的存放的。两张图片的分辨率为 800*480、色深为 16bit。TFT Panel Size is 1024*600，图片的左上角在屏幕的（200, 100）位置 。

```
Select_Main_Window_16bpp();                 // 设置主规窗的色深 16bit 的深度
Main_Image_Start_Address(0);                // 从显示的 0 地址起开始映像到主规窗图层中
Main_Image_Width(1024);                     // 主规窗的宽度
Main_Window_Start_XY(0, 0);                 // 主视窗的起始坐标主规窗从(0, 0)地址开始
Canvas_Image_Start_address(0);              // 从底图（显示内存）的 0 地址开始写数据
Canvas_image_width(1024);                   // 底图的宽度
Active_Window_XY(0, 0,);                    // 工作视窗：LCD 从主规窗的(0, 0)地址开始显示
Active_Window_WH(1024, 600);                // 工作视窗：LCD 显示的宽为 1024，长为 600
while(1)
{
/*-----显示第一张图片，地址为 0x0000013D-----*/
LT768_DMA_24bit_Block(0, 0, 200, 100, 800, 480, 800, 0x0000013D);
delay_ms(500);

/*-----显示第二张图片，地址为 0x0000013D+1024*600*2-----*/
LT768_DMA_24bit_Block(0, 0, 200, 100, 800, 480, 800, 0x0000013D+800*480*2);
delay_ms(500);
}
```

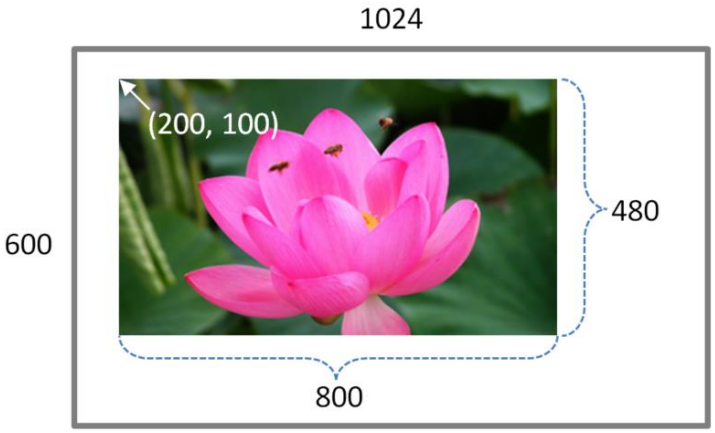


图 15-24：调用 SPI Flash 内的图片（范例一）

范例二：从LT768的SFCS1# (SPI-1) 中外挂的Flash中的“16_16标楷体”地址读取“16*16标楷体”字库数据，并在1024*600的TFT屏上的（625, 150）位置显示红色“东莞市乐电子有限公司”字符串，且不放大字体的高度和宽度、背景色透明、字体对齐。

```
Select_Main_Window_16bpp();                 // 设置主规窗的色深 16bit 的深度
Main_Image_Start_Address(0);                // 从显示的 0 地址起开始映像到主规窗图层中
Main_Image_Width(1024);                     // 主规窗的宽度
Main_Window_Start_XY(0, 0);                 // 主视窗的起始坐标主规窗从(0, 0)地址开始
Canvas_Image_Start_address(0);              // 从底图（显示内存）的 0 地址开始写数据
Canvas_image_width(1024);                   // 底图的宽度
Active_Window_XY(0, 0);                     // 工作视窗：LCD 从主规窗的(0, 0)地址开始显示
Active_Window_WH(1024, 600);                // 工作视窗：LCD 显示的宽为 1024，长为 600

LT768_DrawSquare_Fill(0, 0, 1024, 600, White);  //画白布

/*-----外挂字库初始化（字库起始地址：0x0115E1BD，字库大小：0x00041400）-----*/
LT768_Select_Outside_Font_Init(1, 0, 0x0115E1BD, 1024*600*2, 0x00041400, 16, 1, 1, 1, 1);

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

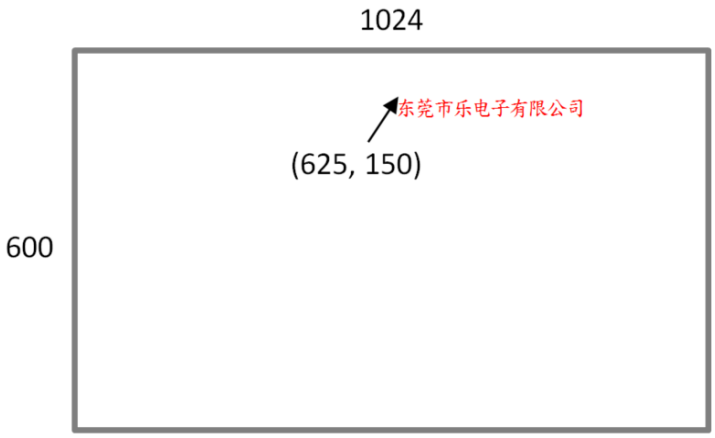


图 15-25：调用 SPI Flash 内的字库（范例二）

16. 触控屏接口

在实际使用 TFT 彩屏时常常会搭配触控屏使用，常见的做法就是将触控屏上的 FPC 接到主控板上，而在触控屏上的芯片通常是 SPI 或是 I2C 接口，由于 LT768x 提供了 SPI Master 及 I2C Master 功能，因此可以将触控屏上的 FPC 引脚直接接到 LT768x 上，然后主控板上的 MCU 直接透过 LT768x 的 MCU 接口对触控芯片进行读取控制。

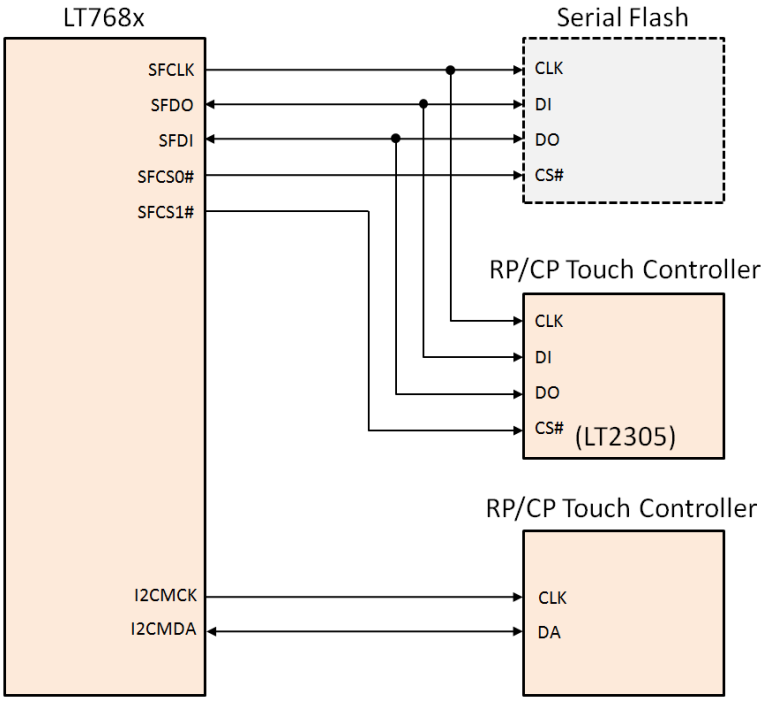


图 16-1：SPI 或是 I2C 的触控屏接口

图 16-1 是 LT768x 的 SPI/I2C 触控屏接口，如果使用本公司的电容触控芯片 LT2305，则可以接到 SPI 接口，有关 SPI Master 或是 I2C Master 的控制方式及相关的时序图可以参考 LT768x 规格书第 10 章的说明。

17. 电源管理

在电源的管理模式上，LT768x 总共有四种工作模式，依据功耗消量大小由高至低为：正常模式（Normal）、待命模式（Standby）、暂停模式（Suspend）、休眠模式（Sleep），四种工作模式由寄存器 REG[DFh] 来设定。下面是四种工作模式的时钟动作比较表：

表 17-1：电源管理模式的时钟动作比较表

Item	正常模式 (Normal)	待命模式 (Standby)		暂停模式 (Suspend)		休眠模式 (Sleep)	
	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

提示：

1. LT768x 进入省电模式时，LCD 接口将不输出讯号，因此进入省电模式前，MCU 需先将 LCD 模块做 Display Off 或 Power Down 的动作，以避免 LCD 极化损坏。
2. OSC 是指外部的晶振频率。

17.1 正常模式

在此模式下内部的三个 PLL 都正常运作，也就是 MCU 通过设定让三个 PLL 分别产生 CCLK（Core Clock）、MCLK（显示内存 Clock）、PCLK（LCD 扫描 Clock），让所有接口包括 LCD 都可以正常显示，要注意的是 PLL 启动需要一段时间，因此 MCU 必须先通过寄存器 01h 的 bit7 得知 PLL 频率是否处于稳定状态。

17.2 待命模式（Standby）

```
void LT768_Standby(void);          // 进入待命模式
void LT768_Wkup_Standby(void);     // 从待机模式中唤醒
```


17.3 暂停模式（Suspend）

```
void LT768_Suspend(void);          // 进入暂停模式
void LT768_Wkup_Suspend(void);     // 从暂停模式中唤醒
```

17.4 休眠模式（Sleep）

```
void LT768_SleepMode(void);        // 进入休眠模式
void LT768_Wkup_Sleep(void);       // 从休眠模式中唤醒
```

17.5 唤醒（Wake up）

有三种方法可以从省电模式中唤醒：外部中断唤醒、键盘扫描唤醒、软件唤醒。如果 MCU 接口设置为并行模式，则 PSM[0] 为外部中断输入引脚。使用外部中断唤醒或是键盘扫描唤醒后 LT768x 会发出中断信号 INT#给 MCU。

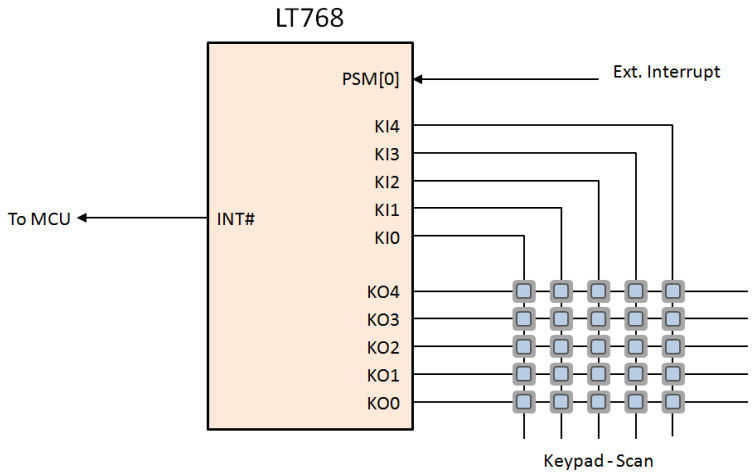


图 17-1：唤醒（Wake up）

对寄存器 REG[DFh] bit7 写 0 可以产生软件唤醒，在系统唤醒后此 bit 才会被清为 0，在系统未完全苏醒时，读取此 bit 仍为 1。MCU 必须等待系统跳出省电模式才能允许写寄存器。MCU 可以检查这个 bit 或是检查状态寄存器位 bit1 (Power Saving) 来得知系统是否已经回到标准操作模式了。唤醒函数如下：

```
void LT768_Wkup_Standby(void);     // 从待机模式中唤醒
void LT768_Wkup_Suspend(void);     // 从暂停模式中唤醒
void LT768_Wkup_Sleep(void);       // 从休眠模式中唤醒
```

18. STM32+LT768x 演示板

18.1 PCB 接口说明

下图是采用 STM32F103VE 与 LT768 组合成的演示板,可以外接标准 50pin 的 TFT RGB FPC 接口(J5),STM32F103VE（U4）已经内含 1024*600 的演示程序，LT768（U1）外接一个 128Mb SPI NOR Flash（U3）内含有演示程序所需的字库与图片。

SW1 可以用来选择 STM32F103 与 LT768 之间是使用哪一种接口（8080/6800 并口，SPI/I2C 串口），使用者可以经由编译环境自行撰写或是修改 STM32F103 的演示程序，然后透过 J2 下载到 STM32F103 内的 Flash；而所需的字库与图片可以参考第 11、15 章的说明产生 Bin 档案，然后透过 J1 由烧录器下载到 SPI Flash。

使用者也可以透过我们的辅助工具经由 SD 卡（U5）接口直接更新 STM32F103 的程序与 SPI Flash 的数据。

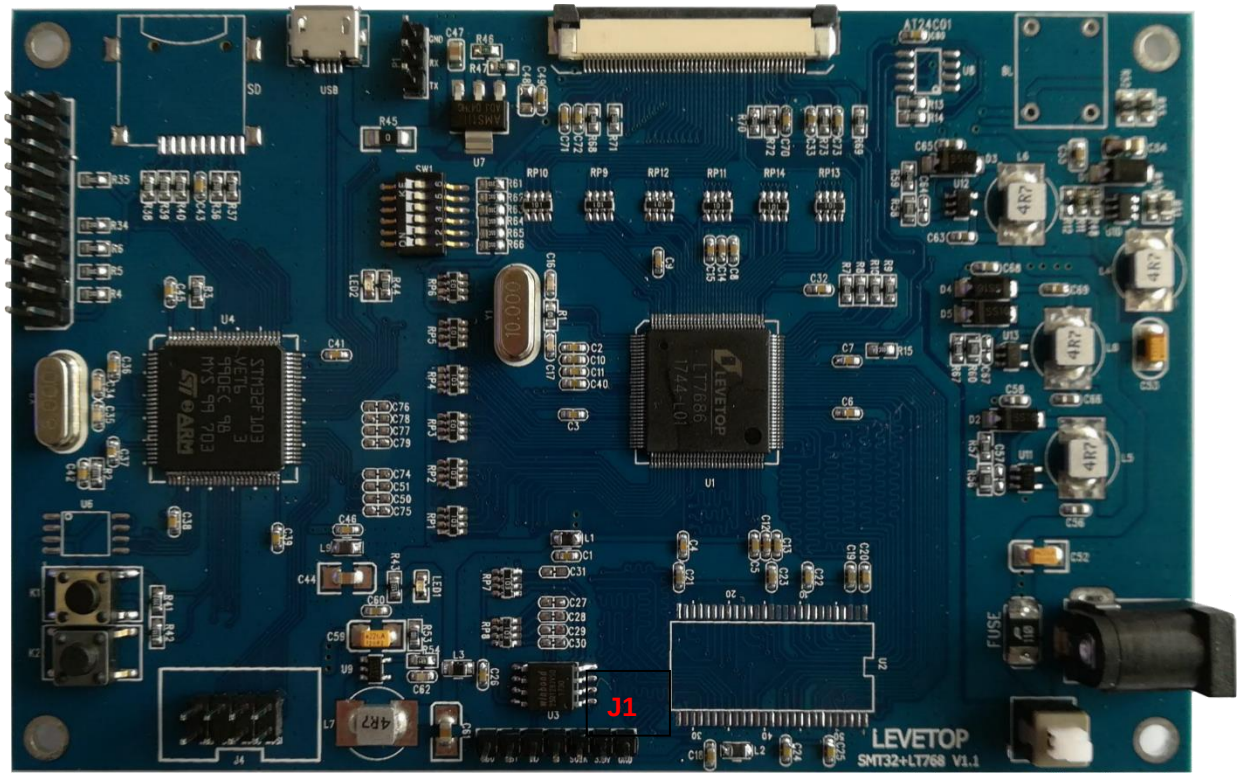


图 18-1：STM32+LT7686 演示板

18.2 原理图

如需 STM32+LT768 演示板的原理图请与我们业务或是 FAE 部门联系取得。

档案名：STM32+LT768_Demo Board_V1.x.rar

18.3 演示程序

如需 STM32+LT768 演示板的演示程序请与我们业务或是 FAE 部门联系取得。

档案名：STM32_LT768_Demo_128Mbit_V10.rar

18.4 SPI Flash 烧录方式

STM32+LT768 演示板如果要更新 SPI Flash 内的数据（Bin 文件），有以下三种方式：

方式 1：使用 MCU 的串口(UART）一边接收来自 PC 机的数据，一边更新 SPI Flash 内的数据。
具体的控制流程图如下：

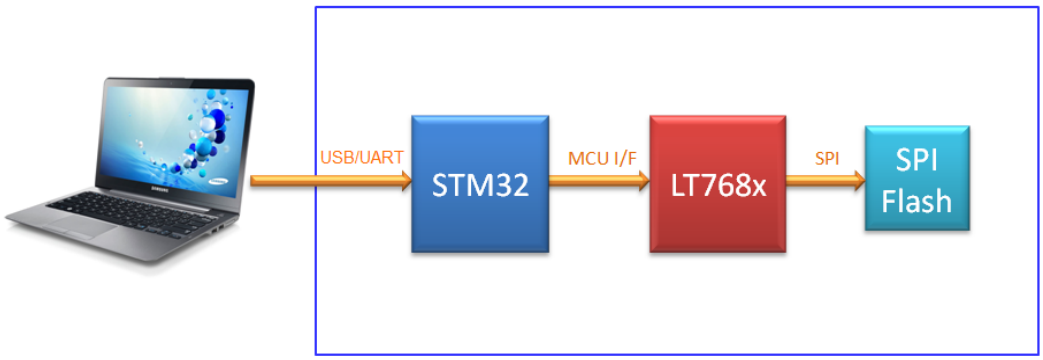


图 18-2：透过 MCU 的 UART 串口更新 SPI Flash 内的数据

数据的来源由 PC/NB 端提供，然后透过 UART 接口传到 MCU，再由 MCU 透过 LT768x 的 SPI Master 接口将数据烧录进 SPI Flash，但是要特别注意 UART 的速度和 MCU 写入数据到 LT768 速度要匹配。因为 STM32 的串口多了个 DMA 接收数据的功能，因此可以用到 DMA 来接收来自 PC/NB 端的数据，而写数据到 Flash 中的速度需要比 MCU 串口接收数据的速度还要快。

方式 2：使用连接在 STM32 的 SD 卡来更新 SPI Flash 内的数据。

具体的控制流程图如下：

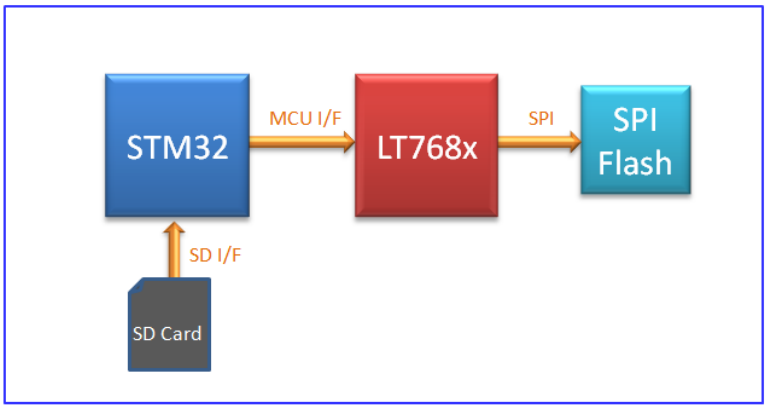


图 18-3：透过 MCU 的 SD 卡更新 SPI Flash 内的数据

方法 3：用专用的 Flash 烧录器，透过板上预留的 Flash 控制引脚直接更新 SPI Flash 内的数据。

此方法需要把 LT768x 的 TEST[2] 引脚拉低、TEST[1] 引脚拉高，使 LT768 进入测试模式及断开对 Flash 的控制，才能将数据烧录到 Flash 而不受 LT768 的影响。

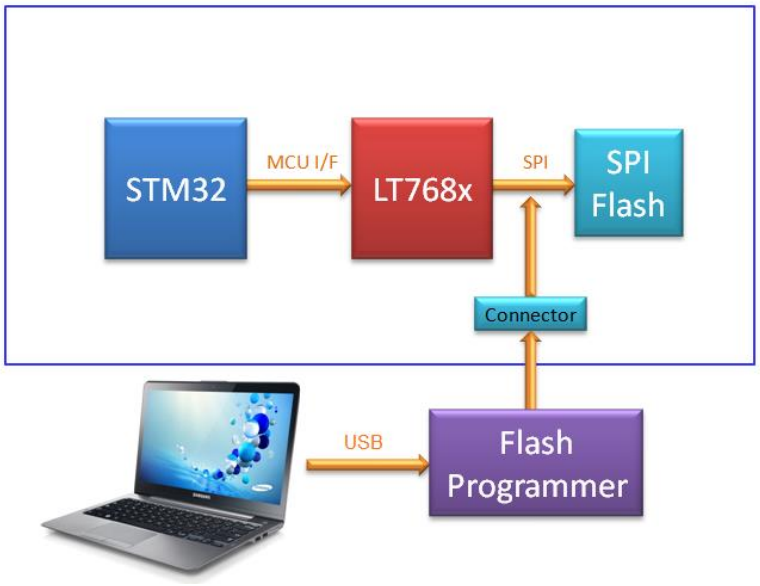


图 18-4：由 PC 透过专用的 Flash 烧录器更新 SPI Flash 内的数据

至于专用的 Flash 烧录器（Programmer）在网站上都可以轻易的采购到，在采购前最好先确认是否支持所使用的 SPI Flash 型号，如有疑问可与本公司 FAE 人员询问。

提示：有关 Bin 文件的产生请参考第 11.3、15.2 ~ 15.4 节。

19. STC51+LT768x 演示板

19.1 PCB 接口说明

下图是采用 STC8051 与 LT768 组合成的演示板，可以外接标准 50pin 的 TFT RGB FPC 接口（J5），STC8051（U5）已经内含 1024*600 的演示程序，LT768（U1）外接一个 128Mb SPI NOR Flash（U3）内含有演示程序所需的字库与图片。

SW1 可以用来选择 STC8051 与 LT768 之间是使用哪一种接口（8080/6800 并口，SPI/I2C 串口），使用者可以经由编译环境自行撰写或是修改 STC8051 的演示程序，然后透过 USB 下载到 STC8051 内的 Flash；而所需的字库与图片可以参考第 11、15 章的说明产生 Bin 档案，然后透过 J1 由烧录器下载到 SPI Flash 。

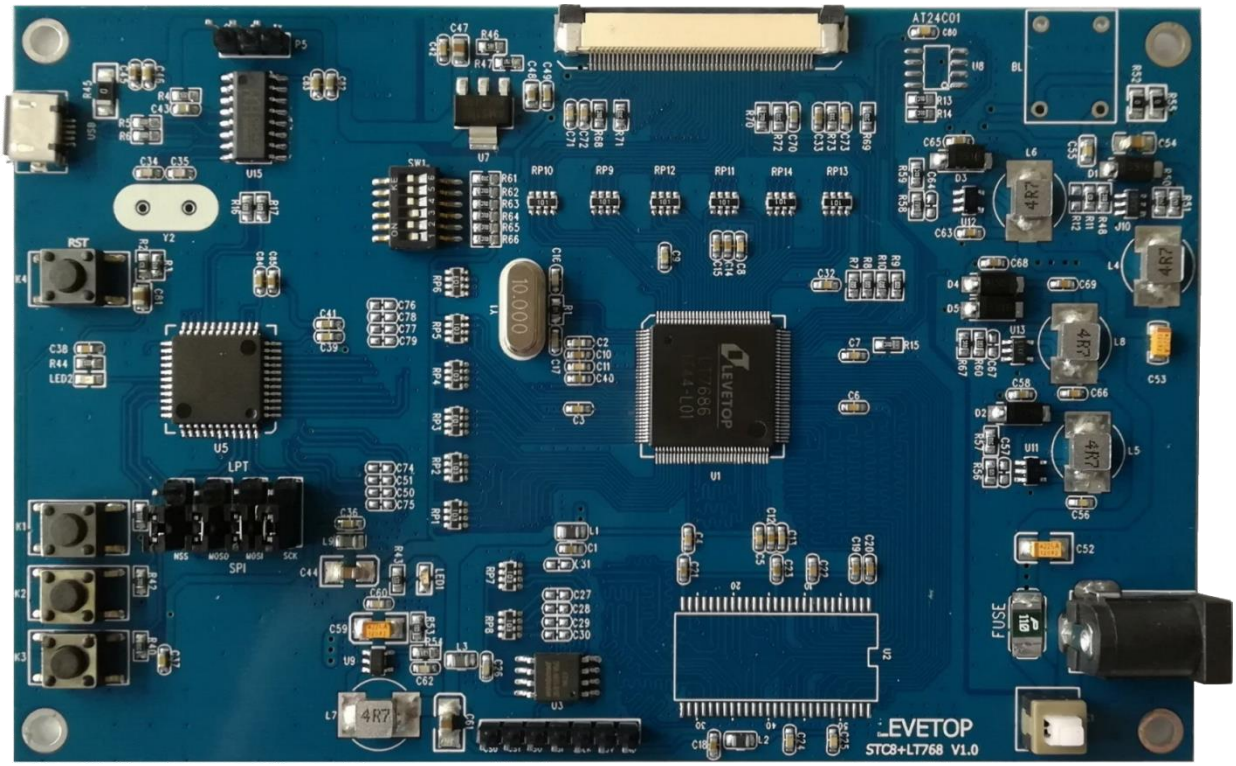


图 19-1：STC8051+LT7686 演示板

19.2 原理图

如需 STC8051+LT768 演示板的原理图请与我们业务或是 FAE 部门联系取得。

档案名：STC8051+LT768_Demo Board_V1.x.rar

19.3 演示程序

如需 STC8051+LT768 演示板的演示程序请与我们业务或是 FAE 部门联系取得。

档案名：STC8051_LT768_Demo_128Mbit_V10.rar

19.4 SPI Flash 烧录方式

STC8051+LT768 演示板如果要更新 SPI Flash 内的数据（Bin 文件），有以下 2 种方式：

方式 1：使用 MCU 的串口（UART）一边接收来自 PC 机的数据，一边更新 SPI Flash 内的数据。
具体的控制流程图如下：

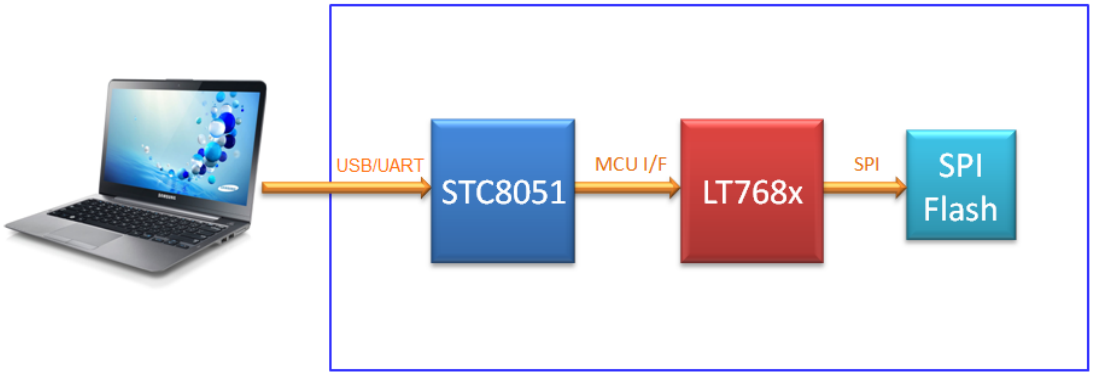


图 19-2：透过 MCU 的 UART 串口更新 SPI Flash 内的数据

数据的来源由 PC/NB 端提供，然后透过 UART 接口传到 MCU，再由 MCU 透过 LT768x 的 SPI Master 接口将数据烧录进 SPI Flash，但是要特别注意 UART 的速度和 MCU 写入数据到 LT768x 速度要匹配。STC8051 可以用串口的中断来接收数据，并使用两个分别为 1K 的缓存，一个缓存在接收 PC/NB 端的数据，一个缓存的数据在写到 Flash 中，此时需要注意写数据到 Flash 中的速度需要比串口接收数据的速度要快。

方式 2：用专用的 Flash 烧录器，透过板上预留的 Flash 控制引脚直接更新 SPI Flash 内的数据。

此方法需要把 LT768x 的 TEST[2] 引脚拉低、TEST[1] 引脚拉高，使 LT768 进入测试模式及断开对 Flash 的控制，才能将数据烧录到 Flash 而不受 LT768 的影响。

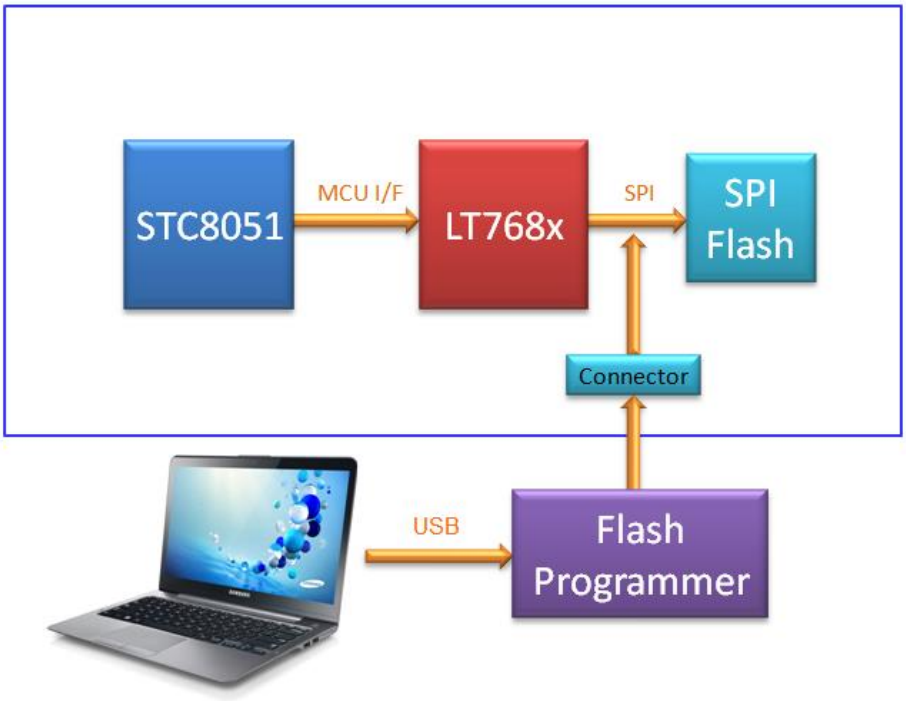


图 19-3：由 PC 透过专用的 Flash 烧录器更新 SPI Flash 内的数据

由于 STC8051 是 8 位 MCU，而且没有 DMA 接收数据的功能，如果使用方式 1 由 8051 烧录数据到 SPI Flash 会比较慢，建议使用方式 2，由专用的 Flash 烧录器去更新 SPI Flash。至于专用的 Flash 烧录器（Programmer）在网站上都可以轻易的采购到，在采购前最好先确认是否支持所使用的 SPI Flash 型号，如有疑问可与本公司 FAE 人员询问。

提示：有关 Bin 文件的产生请参考第 11.3、15.2 ~ 15.4 节。

20. LT32A01+LT7680 SPI 演示板

20.1 接口说明

下图是采用 LT32A01 与 LT7680 组合成的演示板，可以外接标准 40pin 的 TFT RGB FPC 接口，LT32A01（U3）是本公司自行研发生产的 32 位，已经内含 480*272 的简易演示程序，LT7680A（U1）外接一个 128Mb SPI NOR Flash（U2）内含有演示程序所需的字库与图片。

LT7680 只支持 SPI（串口），使用者可以经由编译环境自行撰写或是修改 LT32A01 的演示程序，然后透过 LT32A01 烧录器下载到 LT32A01 内的 Flash；而所需的字库与图片可以参考第 11、15 章的说明产生 Bin 档案，然后透过 Connector 由烧录器下载到 SPI Flash 。

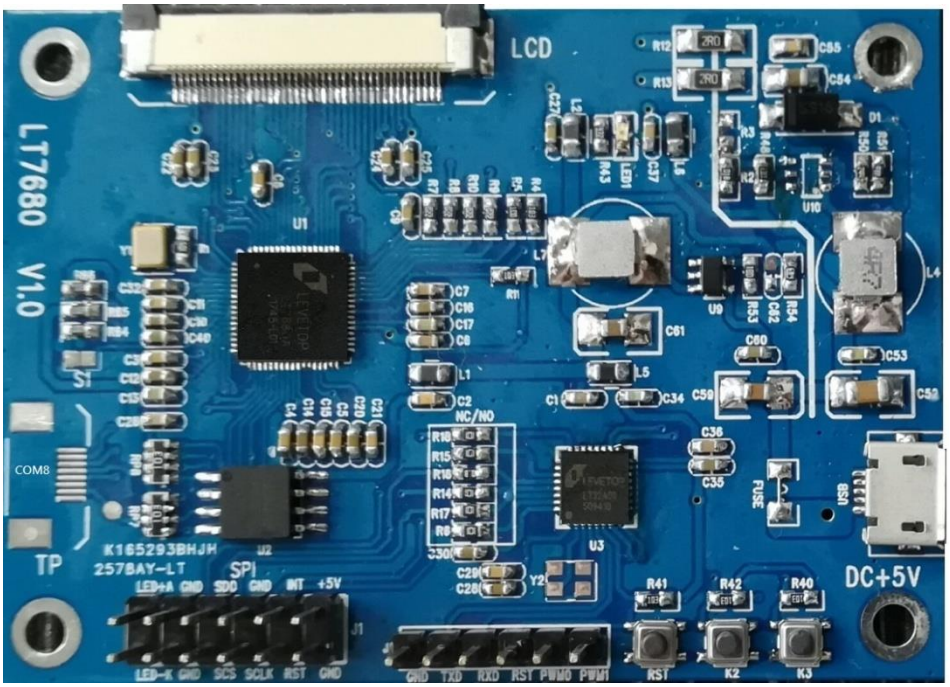


图 20-1：LT7680 SPI 演示板

20.2 原理图

如需 LT7680 SPI 演示板的原理图请与我们业务或是 FAE 部门联系取得。

档案名：LT7680_SPI Demo Board_V1.1.rar

20.3 演示程序

如需 LT7680 SPI 演示板的演示程序请与我们业务或是 FAE 部门联系取得。

档案名：LT7680_SPI Demo_64Mbit_V10.rar

20.4 SPI Flash 烧录方式

LT32A01+LT7680 演示板如果要更新 SPI Flash 内的数据（Bin 文件），可以**用专用的 Flash 烧录器，透过板上预留的 Flash 控制引脚（COM8）直接更新 SPI Flash 内的数据。**此方法需要把 LT768 的 RST 引脚拉低，使 LT7680 断开对 Flash 的控制，才能将数据烧录到 Flash 而不受 LT7680 的影响。

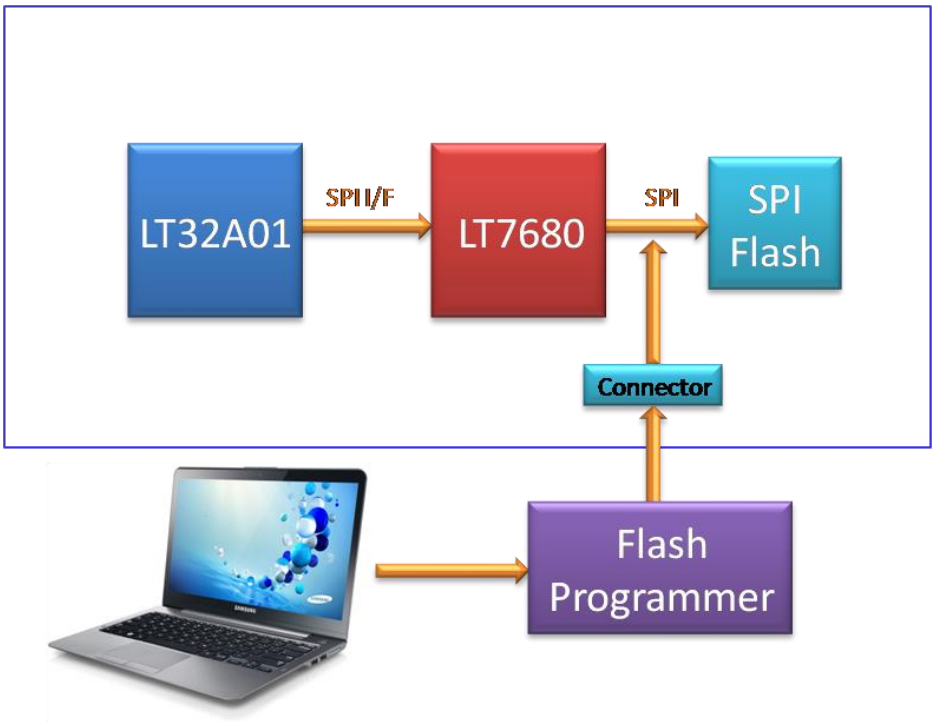


图 20-2：由 PC 透过专用的 Flash 烧录器更新 SPI Flash 内的数据

至于专用的 Flash 烧录器（Programmer）在网站上都可以轻易的采购到，在采购前最好先确认是否支持所使用的 SPI Flash 型号，如有疑问可与本公司 FAE 人员询问。

提示：有关 Bin 文件的产生请参考第 11.3、15.2 ~ 15.4 节。

20.5 使用外部 MCU

如果使用者用外部 MCU 来控制 LT7680，可以将演示板上的 **R14, R15, R16, R17** 断开，再透过 J1 的 SPI 接口接到 LT7680，外部 MCU 就可以控制 LT7680 了，不需要把 LT7680 焊下来，使用者可以参考 LT7680 SPI 演示板的原理图。

21. 用 LT7681/7683+/7686 做标准 TFT 模块（LCM）

LT768x 适合将传统 RBG 接口的 TFT 模块变成 MCU 接口的 TFT 模块，本章将介绍以 LT7681/7683+/7686（LQFP-128Pin）做标准 MCU 接口模块时的原理图及注意事项。

21.1 方块图

图 21-1 是 LT7681/7683+/7686 的 LCM 模块方块图，LT768x 输出的 RGB 信号直接接到标准的 RGB TFT-LCD 显示屏，RGB 数据可以为 16bit(5/6/5)、18bit(6/6/6) ，或是 24bit(8/8/8)。而透过 PSM[2:0] 的准位可以设定（参考模块原理图的 SW1）MCU 的接口是 I2C/SPI 串口还是 6800/8080 并口。

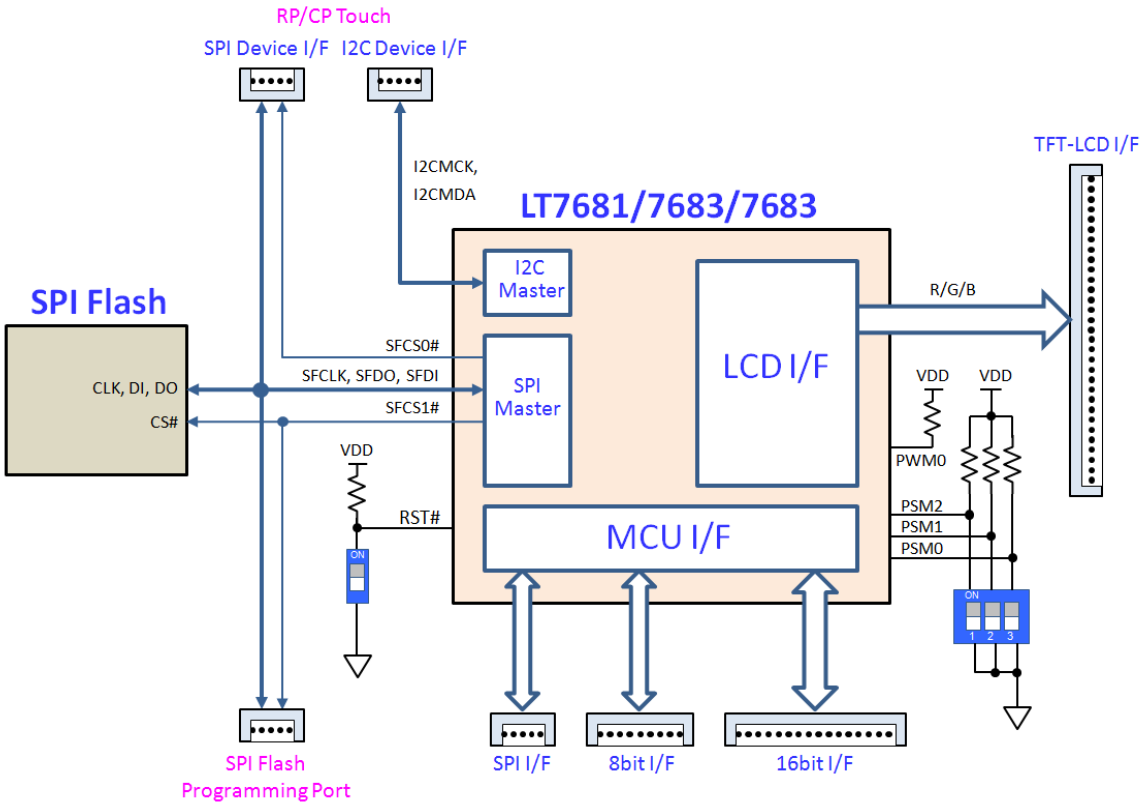


图 21-1：LT7681/7683+/7686 模块方块图

设定并口模式时，选择 8 位或 16 位的数据传输是由寄存器 REG[01h] 的 bit0 来决定，如果 bit0=0，则设定为 8bit 数据总线，如果 bit0=1，则设定为 16bit 数据总线。如下表 21-1 或是参考规格书第二章的说明：

表 21-1：MCU 接口模式设定

PSM[2:0]	MCU 接 口 模 式
0 0 X	选择并口 8 位或 16 位的 8080 模式
0 1 X	选择并口 8 位或 16 位的 6800 模式
1 0 0	选择串口 3 线式 SPI 模式
1 0 1	选择串口 4 线式 SPI 模式
1 1 X	选择串口 I2C 模式

0 代表 DIP Switch 为 ON（接地）状态。

方块图上的 SPI Flash 主要是用来储存应用时常用的图片数据，或是字库档案，模块厂可以事先将客户的需求烧录在 SPI Flash，后加工时再焊在 PCB 上；或者是由 MCU 端透过 LT768 的 MCU 接口将数据（Bin 文件）烧录到 SPI Flash；也可以直接透过 SPI Flash 的烧录接口（Programming Port）直接用烧录器烧录 Bin 文件。我们建议 TFT 模块厂设计时将 SPI Flash 的烧录接口预留出来。

如果在 PWM[0] 引脚接上一个 10K 左右的上拉电阻，那么「开机显示」功能就会被使能（Enable），主要的功能是在没有外部主处理器的情况下，或是主处理器还在起始运行阶段，在开机时借由执行储存在闪存中的程序代码来迅速显示画面。此外终端客户有可能使用触控屏，而 LT768x 内含 SPI Master 及 I2C Master 电路，因此可以将 SPI 及 I2C 接口引出，变成电阻触控屏或是电容触控屏的接口，终端客户再透过 MCU 接口去控制或是读取触控屏数据，也可以减少浪费 MCU 的接口资源。

TFT 模块厂可以用现有的 RGB TFT 屏，加上 LT7681（LT7683+/LT7686）的控制板后变成标准具有 MCU 接口的 TFT 模块，并且迅速的推出市场，如图 21-2 所示。而 MCU 接口的型式可以固定或是开放客户自行设定（如模块原理图的 SW1）。

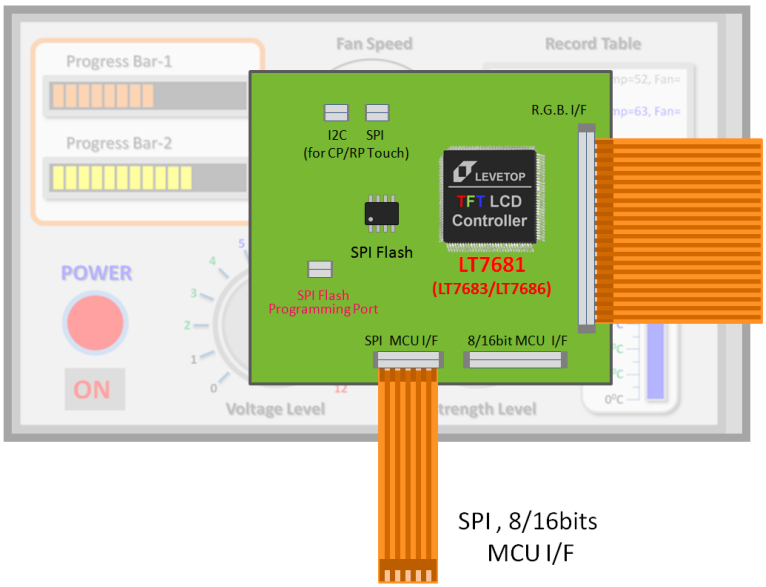


图 21-2：具有 MCU 接口的 TFT 模块

21.2 参考原理图

如需 LT7681/7683+/7686 的标准 MCU 接口模块原理图，请与我们业务或是 FAE 部门联系取得。

档案名： LT7681.3.6 LCM Demo V1.0.zip

21.3 SPI Flash 烧录方式

在前面第 21.1 节提到烧录 SPI Flash 可以由 MCU 端透过 LT768x 的 MCU 接口将数据（Bin 文件）烧录到 SPI Flash；也可以直接透过 SPI Flash 的烧录接口（Programming Port）直接用烧录器烧录 Bin 文件。

SPI Flash 的烧录接口实际上就是将 Flash 的 4 个控制引脚及电源引出（参考模块原理图的 J1 连接器），然后用专用的 Flash 烧录器来更新数据，而为了能将数据烧录到 Flash 而不受 LT768 的影响，烧录前必须采取以下三种措施的任一种：

1. 断开 LT768x 的电源，烧录时只对 SPI Flash 供电
2. 将 L768x 的 RST#接到地(GND)
3. 将 LT768x 的 TEST[2] 引脚拉低（Pull-Low），及 TEST[1] 引脚拉高（Pull-High）

这三种方式都可以使 LT768x 断开对 Flash 的控制，也才能用 Flash 烧录器顺利烧写成功。如前面第 21.1 节的图 21-1 是采用第 2 种方式（可参考模块原理图的 SW1），也就是平时使用时 DIP Switch 为 ON 状态（TEST[1] 引脚被拉低），当烧录器要透过 SPI Flash 的烧录接口对 Flash 烧录时，DIP Switch 要为 OFF 状态（TEST[1] 引脚被拉高），Flash 烧录器才能顺利透过 J1 连接器对 Flash 进行烧写。

我们建议可以将 RST# 引脚引出到烧录接口由外部直接控制，就不须要在接上 DIP Switch，进行 Flash 烧录时会更方便。

22. 用 LT7680 做标准 TFT 模块（LCM）

本章将介绍以 LT7680（QFN-68Pin）做标准 MCU 接口模块时的原理图及注意事项。

22.1 方块图

图 22-1 是 LT7680 的 LCM 模块方块图，LT7680 输出的 RGB 信号直接接到标准的 RGB TFT-LCD 显示屏，RGB 数据可以为 16bit(5/6/5) 或是 18bit(6/6/6)。LT7680 的 MCU 接口只支持 SPI 串口，可以透过 PSM[0] 的准位去设定，如下图 DIP Switch 为 ON 则选择 3 线式 SPI 串口，DIP Switch 为 OFF 则选择 4 线式 SPI 串口。

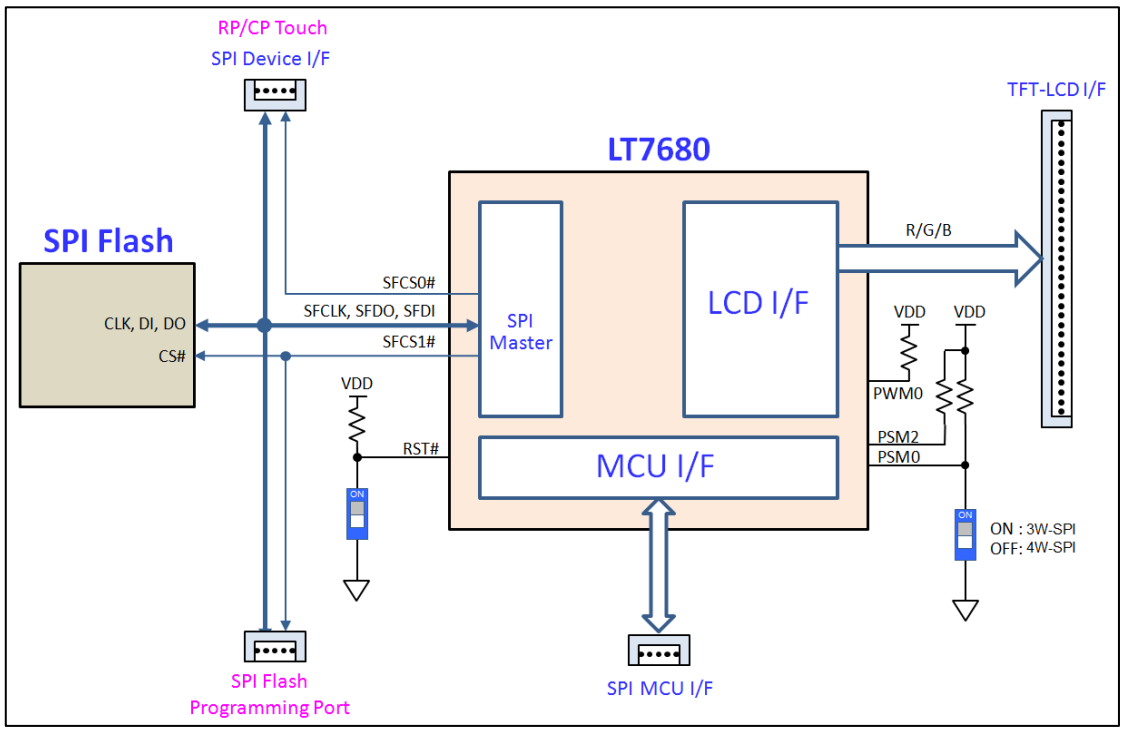


图 22-1：LT7680 模块方块图

方块图上的 SPI Flash 主要是用来储存应用时常用的图片数据，或是字库档案，模块厂可以事先将客户的需求烧录在 SPI Flash，后加工时再焊在 PCB（或是 FPC）上；或者是由 MCU 端透过 LT768 的 MCU 接口将数据（Bin 文件）烧录到 SPI Flash；也可以直接透过 SPI Flash 的烧录接口（Programming Port）直接用烧录器烧录 Bin 文件。我们建议 TFT 模块厂设计时将 SPI Flash 的烧录接口预留出来。

如果在 PWM[0] 引脚接上一个 10K 左右的上拉电阻，那么「开机显示」功能就会被使能（Enable），主要的功能是在没有外部主处理器的情况下，或是主处理器还在起始运行阶段，在开机时借由执行储存在闪存中的程序代码来迅速显示画面。此外终端客户有可能使用触控屏，而 LT7680 内含 SPI Master 电路，可以将 SPI 接口引出，变成电阻触控屏或是电容触控屏的接口，终端客户再透过 MCU 接口去控制或是读取触控屏数据，也可以减少浪费 MCU 的接口资源。

TFT 模块厂可以用现有的 RGB TFT 屏，加上 LT7680 的控制板后变成标准具有 SPI 接口的 TFT 模块，并且迅速的推出市场，如图 22-2 所示。而 MCU 接口的型式可以固定为 3 线 SPI 或是 4 线 SPI。TFT 模块厂也可以用现有 RGB TFT 屏，修改 FPC，将 LT7680 相关电路放在 FPC 上，如图 22-3 所示。

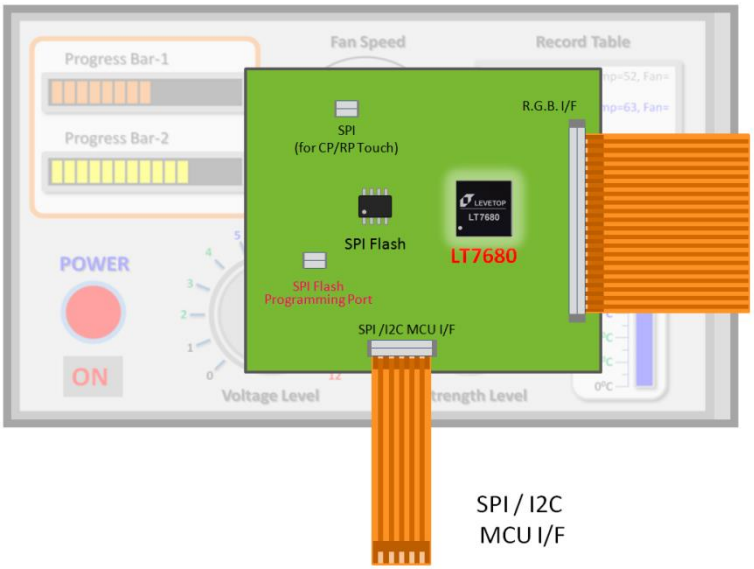


图 22-2：具有 SPI 接口的 TFT 模块（1）

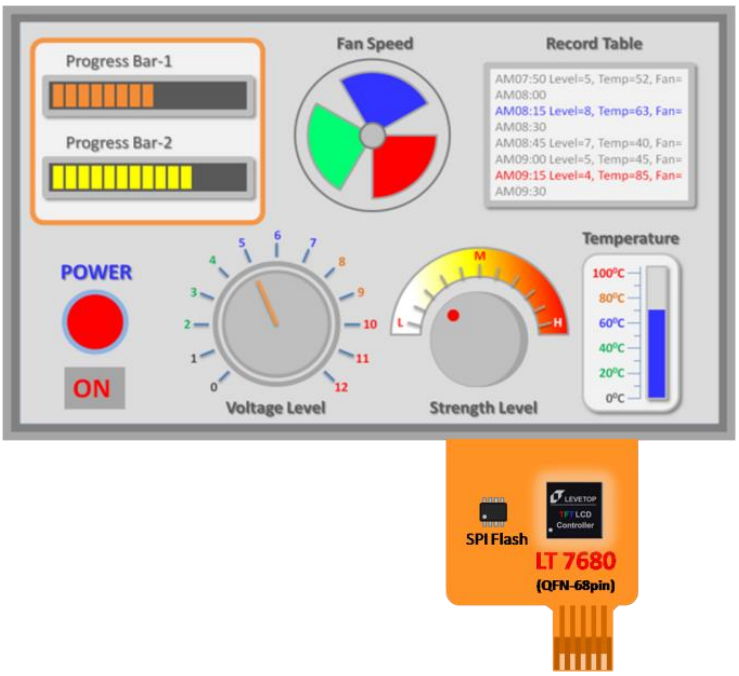


图 22-3：具有 SPI 接口的 TFT 模块（2）

22.2 参考原理图

如需 LT7680 的标准 MCU 接口模块原理图，请与我们业务或是 FAE 部门联系取得。

档案名：LT7680 LCM Demo V1.0.zip

22.3 SPI Flash 烧录方式

在前面第 22.1 节提到烧录 SPI Flash 可以由 MCU 端透过 LT7680 的 MCU 接口将数据（Bin 文件）烧录到 SPI Flash；也可以直接透过 SPI Flash 的烧录接口（Programming Port）直接用烧录器烧录 Bin 文件。

SPI Flash 的烧录接口实际上就是将 Flash 的 4 个控制引脚及电源引出（参考 LT7680 模块原理图的 J1 连接器），然后用专用的 Flash 烧录器来更新数据，而为了能将数据烧录到 Flash 而不受 LT7680 的影响，烧录前必须采取以下 2 种措施的任一种：

1. 断开 LT7680 的电源，烧录时只对 SPI Flash 供电
2. 将 L7680 的 RST#接到地（GND）

这 2 种方式都可以使 LT7680 断开对 Flash 的控制，也才能用 Flash 烧录器顺利烧写成功。如前面第 22.1 节的图 22-1 是采用第 2 种方式（可参考模块原理图的 S1），也就是平时使用时 DIP Switch 为 OFF 状态（RST# 引脚被拉高），当烧录器要透过 SPI Flash 的烧录接口对 Flash 烧录时，DIP Switch 要为 ON 状态（RST# 引脚被拉低），Flash 烧录器才能顺利对 Flash 进行烧写。

建议可以将 RST# 引脚引出到烧录接口由外部直接控制，就不须要在接上 DIP Switch，进行 Flash 烧录时会更方便。

23. 程序库说明

23.1 程序库

- LT768x 底层寄存器的封装函数：

  档案名：LT768.c

- LT768x 底层寄存器封装函数声明：

  档案名：LT768.h

- LT768x 的功能封装函数：

  档案名：LT768_Lib.c

- LT768x 的功能封装函数声明：

  档案名：LT768_Lib.h

23.2 应用软件

- Bin 檔生成软件：

  档案名：LT_IMAGE_TOOL.EXE

LT_IMAGE_TOOL.EXE 是 乐升半导体 提供的一个 Bin 檔生成软件，针对 LT768x TFT 控制器外接的 SPI Flash，将图片 Bin 文件、字库 Bin 文件、图形光标和开机启动程序等整合起来，产生可以烧录到 SPI Flash 的 Bin 文件，此软件的 6 个功能，分别为：

一. 制作图片 Bin 文件
二. 制作字库 Bin 文件
三. 制作「GIF 檔 Bin 文件」
四. 制作图形光标
五. 设置开机启动加载程序
六. Bin 文件整合

详细说明请参考 LT_IMAGE_TOOL.EXE 的使用说明书。

24. 程序库列表

表 24-1：程序库列表

No.	函 数 名 称	功 能	页数
启始設定			
1	LT768_SW_Reset()	软件复位	13
2	LT768_PLL_Initial()	时钟與 PLL 設定	16
MCU 接口：8 位的 8080 接口			
3	FMSC_8_CmdWrite()		20
4	FMSC_8_DataWrite()		20
5	FMSC_8_DataWrite_Pixel()		20
6	u8 FMSC_8_StatusRead()		20
7	u16 FMSC_8_DataRead()		21
MCU 接口：16 位的 8080 接口			
8	FMSC_16_CmdWrite()		22
9	FMSC_16_DataWrite()		22
10	FMSC_16_DataWrite_Pixel()		22
11	u8 FMSC_16_StatusRead(void)		22
12	u16 FMSC_16_DataRead(void)		22
MCU 接口：SPI 接口			
13	SPI_CmdWrite()		23
14	SPI_DataWrite()		23
15	SPI_DataWrite_Pixel()		23
16	u8 SPI_StatusRead()		23
17	u16 SPI_DataRead()		24
MCU 接口：I2C 接口			
18	u8 IIC_CmdWrite()		25
19	u8 IIC_DataWrite()		25
20	IIC_DataWrite_Pixel()		26
21	u8 IIC_StatusRead()		26
22	u8 IIC_DataRead()		26
显示内存（SDRAM）设定			
23	LT768_SDRAM_initail()	顯示内存（SDRAM）設定	27
控制 LCD 的输出信号			
24	Set_LCD_Panel()	LCD 屏設定	32
25	VSCAN_T_to_B()	LCD 的扫描方式从上到下	32
26	VSCAN_B_to_T()	LCD 的扫描方式从下到上	32

No.	函 数 名 称	功 能	页数
27	PDATA_Set_RGB()	RBG 的输出方式: RGB	32
28	PDATA_Set_RBG()	RBG 的输出方式: RBG	32
29	PDATA_Set_GRB()	RBG 的输出方式: GRB	32
30	PDATA_Set_GBR()	RBG 的输出方式: GBR	32
31	PDATA_Set_BRG()	RBG 的输出方式: BRG	32
32	PDATA_Set_BGR()	RBG 的输出方式: BGR	32
33	HSYNC_Low_Active()	HSYNC 的动作方式：Low	32
34	HSYNC_High_Active()	HSYNC 的动作方式：High	32
35	VSYNC_Low_Active()	VSYNC 的动作方式：Low	32
36	VSYNC_High_Active()	VSYNC 的动作方式：High	32
37	DE_High_Active()	DE 的动作方式：High	32
38	DE_Low_Active()	DE 的动作方式：Low	32
设置主视窗是要以几位的颜色来显示			
39	Select_Main_Window_8bpp();	256 色	33
40	Select_Main_Window_16bpp();	65K 色	33
41	Select_Main_Window_24bpp();	262K / 16.7M 色	33
设置主视窗			
42	Main_Image_Start_Address()	设置主视窗的开始显示地址	33
43	Main_Image_Width()	设置主视窗的宽度	33
44	void Main_Window_Start_XY()	设置主视窗的起始坐标	33
设置底图视窗			
45	void Canvas_Image_Start_address()	设置底图的起始位置	34
46	void Canvas_image_width()	设置底图的宽度	34
设置工作视窗			
47	void Active_Window_XY()	设置工作视窗的起始位置	34
48	void Active_Window_WH()	设置工作视窗的大小	34
MCU 写入数据到内存			
49	void MPU8_8bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 8 位色深来显示	35
50	void MPU8_16bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 16 位色深来显示	35
51	void MPU8_24bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 24 位色深来显示	35
52	void MPU16_16bpp_Memory_Write()	MCU 使用 16 位数据驱动，而且用 16 位色深来显示	35
53	void MPU16_24bpp_Mode1_Memory_Write()	MCU 使用 16 位数据驱动，而且用 24 位色深来显示	35

25. 点亮 TFT 屏流程及注意事项

25.1 电源部分

1、未上电前，检查电路各个部分是否有短路现象，确认没有异常再上电。

2、检查 LT768x 的 VDD 电压是否为稳定的 3.3V。（参考 **LT768x 引脚图)**

3、检查 LT768x 的 VDD_C 电压是否为稳定的 1.8V。

25.2 晶振部分

检查 XI 及 XO 所接时钟输入或是晶振（通常为 10MHz）是否起振（用示波器测量），若没有起振，则检查时钟来源或是晶振的 RC 电路及使用的阻容值是否正确，或是更换晶振。

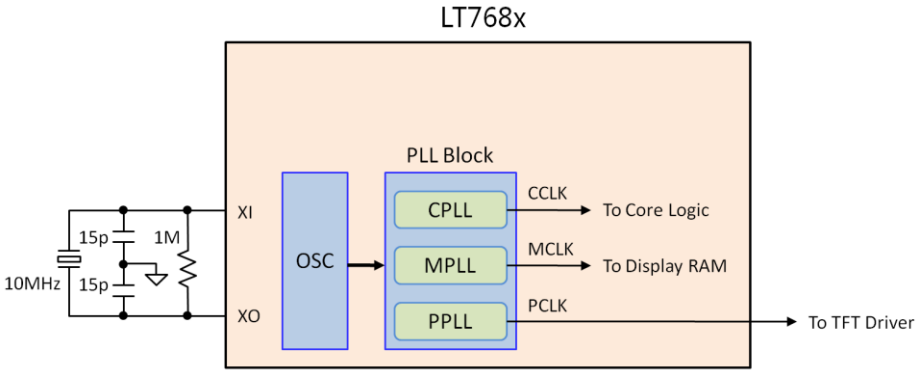


图 25-1：LT768x 时钟电路（一）

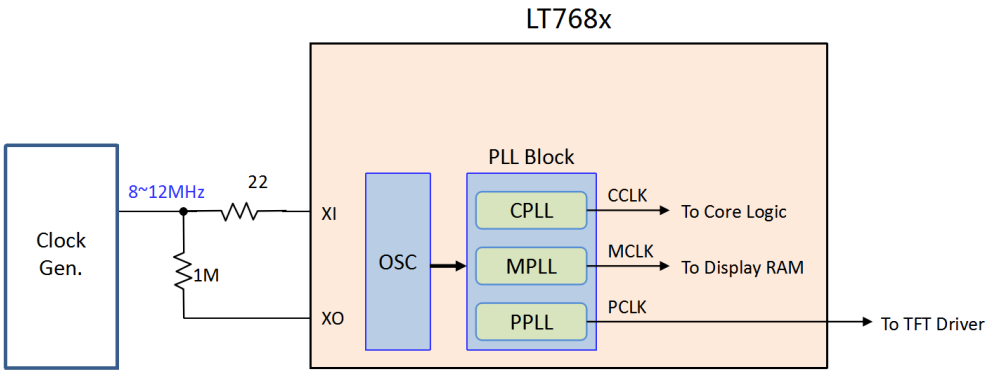


图 25-2：LT768x 时钟电路（二）

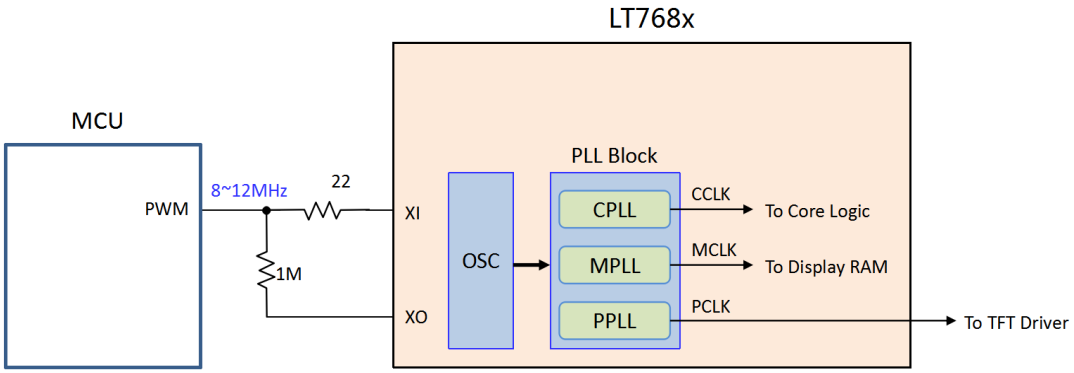


图 25-3：LT768x 时钟电路（三）

25.3 复位部分

确认 RST#复位接脚可通过 MCU 正确控制，复位后此脚应为高电位。

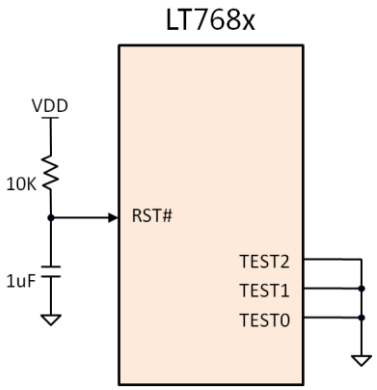


图 25-4：外部复位方式（1）

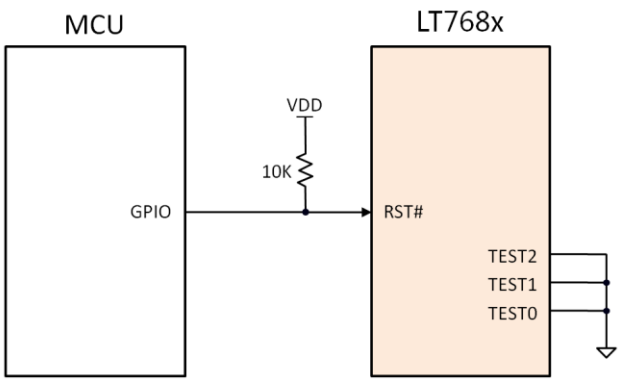


图 25-5：外部复位方式（2）

25.4 测试接脚

确认 TEST[2:0]引脚为接地（如图 25-4、图 25-5）。若使用 LT7680A/B 型号，则忽略此步骤。

25.5 MCU 的接口

1、根据所设计的 MCU 接口模式，检查是否与 LT768x PSM[2:0]三个引脚的设定一致。

表 25-1：LT7681/3/6 MCU 接口模式设定

PSM[2:0]	MCU 接口模式
0 0 X	选择并口 8 位或 16 位的 8080 模式
0 1 X	选择并口 8 位或 16 位的 6800 模式
1 0 0	选择串口 3 线式 SPI 模式
1 0 1	选择串口 4 线式 SPI 模式
1 1 X	选择串口 I2C 模式

2、LT7680A/B 只支持串口 3 线 SPI 及 4 线 SPI 模式，其 PSM[1] 引脚已经在 IC 内部接到地，而 PSM[2] 则必须接到高电位。PSM[0] = 0，选择串口 3 线式 SPI 模式；PSM[0] = 1，选择串口 4 线 SPI 模式。

表 25-2：LT7680 MCU 接口模式设定

PSM[2], PSM[0]	MCU 接口模式
1 0	选择串口 3 线式 SPI 模式
1 1	选择串口 4 线式 SPI 模式
0 X	不允许

3、MCU 接口到 LT768 的接线尽可能小于 15cm 内，如果太长需要加上拉电阻或是降低传输速度。

25.6 初始化部分

1、测试 MCU 是否能烧录程序，通过串口打印检查上电后系统初始化是否通过。若没有，则详细检查 MCU 的周边电路是否正常。

2、通过串口打印调试，测试 MCU 是否成功复位 LT768x，即复位后，System_Check_Temp 函数是否通过。若没有，则检查 MCU 和 LT768x 的接口和复位引脚是否正确连接。

3、通过串口打印调试，测试 MCU 是否成功初始化 LT768x。若中间某一个函数不通过，比如 LT768_SDRAM_initail 函数，则尝试降低 MCU 和 LT768x 的接口的通讯速度。

4、可以透过本公司提供的初始化函数确认 MCU 与 LT768 是否通讯正常。

5、可以透过本公司提供的 Display ON 函数让 LCD 输出正常信号。

25.7 显示部分

1、若以上步骤均正常，且 MCU 的程序已经控制 LT768x 输出画面，如红、绿、蓝三种颜色的前提下，屏幕没有显示，则首先检测 LT768x 的 LCD 输出信号 PCLK、DE、HSYNC 和 VSYNC 是否有波形输出，波形是否正确（用示波器测量），以及是否送到 TFT 屏的 FPC 上。若没有波形，则检查芯片是否有虚焊，或者更换芯片测试。

2、若第一步 LT768x 的 LCD 输出信号 PCLK、DE、HSYNC 和 VSYNC 没有波形输出，则检查是否呼叫 Display ON 函数。

3、若第一步 LT768x 的 LCD 输出信号 PCLK、DE、HSYNC 和 VSYNC 有波形输出，则检查 PCLK 的频率是否与屏幕匹配，以及 HSYNC 和 VSYNC 的时序是否选择正确（高电平有效或低电平有效）。

4、若第一步正常，则检查屏幕部分的驱动电路电压，背光电路（LEDA+、LEDK-）电压是否符合屏幕需求。若不符合或没有电压，则详细检查并调整电路。

5、若屏幕带有开启/关闭（Display ON/OFF）控制引脚，则检查次引脚的电平是否达到要求，若不符合或为低电平，则详细检查并调整电路

6、

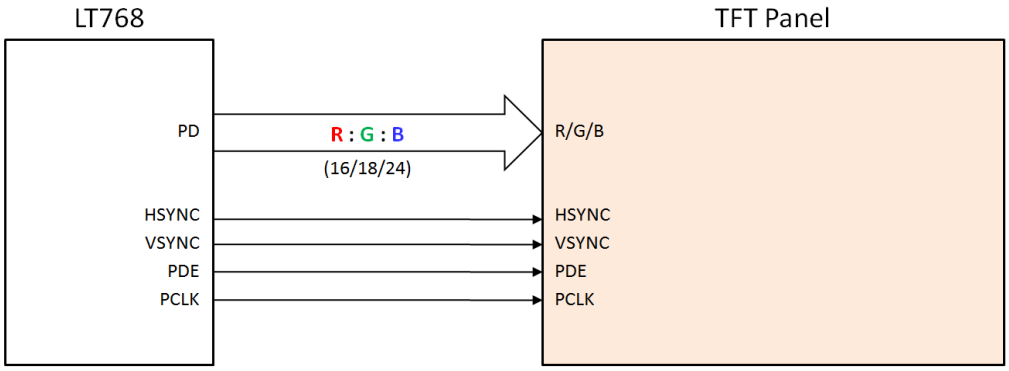


图 25-6：LT768x 与 TFT 屏驱动器的接口

25.8 SPI Flash 部分

1、若与 SPI Flash 的通讯不成功，则先检查 Flash 与 LT768x 连接的电路是否正常，Flash 周边的电路焊接是否有短接现象。

2、SPI Flash 的烧录：若使用 STM32F103+LT7681/3/6 平台，则可以使用 STM32_BinToFlash 程序来烧录 Flash（程序及说明文档请联系本司业务）。若使用 LT7680A/B 平台，则需使用外部烧录器烧录。需注意，烧录 Flash 时，要把 LT768x 的复位引脚接地，否则无法烧录。

3、烧录 Flash 所需的 Bin 文件，需使用 LT_IMAGE_TOOL.exe 软件生成，该软件具有制作图片、Bootloader、光标、字库、GIF 和 WAV bin 文件的功能，同时还具有 bin 文件的合并功能。

25.9 其他注意事项

1、用户可以参考 LT768x 的规格书及应用手册（LT768x_AP-Note_Vxx_CH.pdf），尤其是应用手册说明了 LT768x 的硬件接口与内部功能的实现，同时配合本公司所提供的演示程序、程序库、及原理图，可以让 TFT 模块厂或是系统端的应用客户很快的能对 LT768x 进行设置及应用开发，能轻易上手并且缩短自行摸索的时间。手册中除了硬件及软件的设置说明外，也在最后几章也介绍了本公司所提供的 STM32+LT768 演示板、STC8051+LT768 演示板，还有针对 TFT 模块厂将 LT768x 设计到 TFT 模块上时所要注意的事项，及对 SPI Flash 烧录的方式做了完整说明。

2、针对不同的 MCU、不同的串并口，及 24bit RGB 或是 16bit RGB，本公司提供了简易的 Demo 程序（STM32+LT768 Simple DEMO.rar；STC8+L768 Simple DEMO.rar），让用户能迅速的点亮 TFT 屏和确认软硬件是否正常。简易的 Demo 程序请自我们的网站 Http://www.levetop.cn 下载，或与我们业务、FAE 部门联系取得。

版本记录

表 A-1：应用手册版本记录

版 别	发 布 日 期	改 版 說 明
V1.0	2017/12/25	初版
V1.1	2018/01/12	修订版 1. 增加 18.4 节、19.4 节 SPI Flash 刻录方式 2. 增加 21 章 LT7681/7683+/7686 做标准模块（LCM） 3. 增加 22 章 LT7680 做标准模块（LCM）
V2.0	2018/02/24	1. 修订 11.3 节 制作串行闪存的字库 Bin 文件 2. 修订 12.3 节 图形光标产生工具 3. 增加 14.1 节 设置开机启动加载程序 4. 修订 15.2 节 制作图片的 Bin 文件 5. 修订 15.3 节 Bin 文件的结合 6. 修订表 24-1：程序库列表
V2.1	2018/05/04	1. 修订 11.2.3、11.2.4、11.2.5 节 关于显示外建中文字库（GB2312/BIG5 码）的函数，及使用大型（48*48、72*72）中文字库的方式 2. 修订 11.3 节 制作串行闪存的字库 Bin 文件 3. 增加图 13-3 PWM 控制背光参考原理图 4. 增加 15.3 节 制作 GIF 檔的 Bin 文件 5. 增加 20.1、20.4、20.5 节有关 LT7680 SPI 演示板的说明
V2.2B	2018/12/28	1. 增加 4.1 节（时钟与 PLL）注意事项 2. 增加图 4-2、图 4-2 LT768x 时钟电路 3. 增加 25 章：点亮 TFT 屏流程及注意事项
V2.5	2019/10/08	1. 更新图 22-4：4 线式 MCU SPI 转 3 线式 SPI 2. 修改 LT7683+产品信息
V3.0	2020/7/15	1. 修改 LT7680 MCU 端的 I2C 接口为 4 线 SPI 接口

版权说明