乐升半导体
LEVETOP

LT7580
LT7586B

TFT-LCD 绘图显示控制芯片

High Performance TFT-LCD Graphics Controller

应用手册

V1.2

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

版本记录

版 本	日 期	说 明
V1.0	2024/11/12	● 初版
V1.2	2026/02/27	● 更新图 2-1：LT7586B 引脚图 ● 更新图 20-2：STM32F407+LT7586 演示板原理图 ● 移除 LT7583 信息

版权说明

目　录

版本记录 2
版权说明 2
目　录 3
图附录 8
表附录 12
1. 前言 13
1.1 手册说明 13
1.2 应用架构 14
2. 型号选择 15
2.1 芯片脚位图 15
2.2 LT758x 比较表 17
3. 复位 20
3.1 电源开启复位 20
3.2 外部复位信号 20
3.3 软件复位 21
3.4 测试信号 21
4. 时钟设定 22
4.1 时钟与 PLL 22
4.2 时钟的启始设定 25
5. MCU 接口 26
5.1 MCU 硬件接口 26
5.2 使用 8 位 8080 接口 29
5.3 使用 16 位 8080 接口 31
5.4 使用 8 位 SPI 接口 32
5.5 使用 16 位 SPI 接口 34
5.6 使用 I2C 接口 36

6. 显示内存（SDRAM）设定 38
7. LCD 界面 39
7.1 LCD 屏的接口 39
7.2 LCD 屏设定 42
8. 显示功能 43
8.1 显示视窗（Display Windows） 43
8.1.1 主视窗的设定 43
8.1.2 底图视窗的设定 44
8.1.3 工作视窗的设置 44
8.2 MCU 写入数据到内存 45
8.3 主视窗中显示图片 46
8.4 画中画（Picture-In-Picture, PIP） 48
8.5 旋转与镜像 49
8.6 彩条（Color Bar）显示 50
9. 几何绘图 51
9.1 画线 51
9.1.1 画细线 51
9.1.2 画粗线 51
9.2 画圆形 52
9.2.1 画空心圆形 52
9.2.2 画实心圆形 53
9.2.3 画带框实心圆形 53
9.3 画椭圆形 54
9.3.1 画空心椭圆形 54
9.3.2 画实心椭圆形 55
9.3.3 画带框实心椭圆形 55
9.4 画矩形 56
9.4.1 画空心矩形 56
9.4.2 画实心矩形 57
9.4.3 画带框实心矩形 57
9.5 画圆角矩形 58
9.5.1 画空心圆角矩形 58
9.5.2 画实心圆角矩形 59
9.5.3 画带框实心圆角矩形 59

9.6 画三角形 60
9.6.1 画空心三角形 60
9.6.2 画实心三角形 61
9.6.3 画带框实心三角形 61
9.7 画曲线 62
9.7.1 左上方曲线 62
9.7.2 左下方曲线 63
9.7.3 右上方曲线 63
9.7.4 右下方曲线 64
9.8 画 1/4 椭圆形 64
9.8.1 左上方 1/4 椭圆 64
9.8.2 左下方 1/4 椭圆 65
9.8.3 右上方 1/4 椭圆 66
9.8.4 右下方 1/4 椭圆 66
9.9 画四边形 67
9.9.1 画空心四边形 67
9.9.2 画实心四边形 68
9.10 五边形 69
9.10.1 画空心五边形 69
9.10.2 画实心五边形 70
9.11 圆柱体 71
9.12 方柱体 72
9.13 表格视窗 73
10.区块传输引擎（BTE） 74
10.1 BTE 操作模式 74
10.2 BTE 功能详述 76
10.2.1 结合光栅操作的 MCU 写入 76
10.2.2 结合光栅操作的 BTE 内存复制 78
10.2.3 结合 Chroma Key 的 MCU 写入 82
10.2.4 结合 Chroma Key 的内存复制 84
10.2.5 结合光栅操作的图样填满 86
10.2.6 结合 Chroma Key 的图样填满 88
10.2.7 结合扩展色彩的 MCU 写入 90
10.2.8 结合扩展色彩与 Chroma key 的 MCU 写入 94
10.2.9 结合透明度的内存复制 96
10.2.10 结合透明度的 MCU 写入 101
10.2.11 结合扩展色彩的内存复制 103

10.2.12 结合扩展色彩与 Chroma Key 的内存复制 106
10.2.13 区域填满（Solid Fill） 108
11.显示文字 110
11.1 使用内建字库 110
11.2 建立中文字库 111
11.2.1 取得字库 111
11.2.2 存入字库档方法 111
11.2.3 显示中文字（16*16、24*24、32*32） 111
11.2.4 显示大型中文字（48*48、72*72） 113
11.3 制作字库的 Bin 文件 115
12.光标 122
12.1 显示文字光标 122
12.2 显示图形光标 124
12.3 图形光标产生工具 127
12.3.1 制作图形光标 127
12.3.2 导入及修改图形光标 130
13.脉宽调制-PWM 132
14.SPI Master 135
14.1 串行闪存的 DMA 传输 136
14.1.1 串行闪存在线性模式下的 DMA 传输 136
14.1.2 串行闪存在区块模式下的 DMA 传输 138
14.2 Bin 文件的结合 140
14.3 程序如何调用 SPI Flash 的 Bin 文件 144
15.图像解码单元（Image Decode Unit） 145
15.1 JPG 图像解码流程 145
15.2 JPG 解码程序例程 146
15.3 制作 JPG 图片的 Bin 文件 147
16.电源管理 153
16.1 正常模式 153
16.2 待命模式（Standby） 153
16.3 暂停模式（Suspend） 153
16.4 休眠模式（Sleep） 154

16.5 唤醒（Wake up） 154
17.程序库说明 155
17.1 程序库 155
17.2 应用软件 155
17.3 程序库列表 156
18.点屏範例 162
18.1 初始化 LT758x 162
18.2 从 MCU 端发送图片到显示屏上 163
18.3 从 LT758x 外部的 SPI Flash 读取图片到显示屏上 167
19.点亮 TFT 屏流程及注意事项 171
19.1 电源部分 171
19.2 晶振部分 171
19.3 复位部分 172
19.4 测试接脚 173
19.5 MCU 的接口 173
19.6 初始化部分 174
19.7 显示部分 174
19.8 SPI Flash 部分 175
19.9 其他注意事项 175
20.LT7586 演示板 176
20.1 PCB 接口说明 176
20.2 演示程序 176
20.3 SPI Flash 烧录方式 177
20.4 原理图 178
21.LT7580 演示板 179
21.1 PCB 接口说明 179
21.2 演示程序 179
21.3 SPI Flash 烧录方式 179
21.4 原理图 180

图附录

图 1-1：LT758x TFT Controller 13
图 1-2：LT758x 内部方块图 13
图 1-3：LT758x 应用架构图 14
图 2-1：LT7586B 引脚图（LQFP-128Pin） 15
图 2-2：LT7580 引脚图（QFN-80Pin） 16
图 2-3：LT7580 标准 SPI/8Bit TFT 模块 18
图 2-4：标准带控制器的 TFT 模块 19
图 2-5：LT7586 设置在系统主板上 19
图 2-6：LT7586 设置在 LCD 模块上 19
图 3-1：外部复位信号 20
图 3-2：外部复位方式（1） 20
图 3-3：外部复位方式（2） 20
图 5-1：8080 MCU 并口电路图 27
图 5-2：6800 MCU 并口电路图 27
图 5-3：MCU3 线 SPI 串联接口电路图 28
图 5-4：MCU4 线 SPI 串联接口电路图 28
图 5-5：MCU I2C 串联接口电路图（不支持连读模式） 28
图 5-6：MCU I2C 串联接口电路图（支持连读模式） 29
图 7-1：LT758x 与 TFT 屏驱动器的接口 40
图 7-2：TFT-LCD RGB 接口时序图 41
图 8-1：彩条（Color Bar）显示 50
图 8-2：垂直方向的彩条（Color Bar）显示 50
图 9-1：画细线 51
图 9-2：画粗线 52
图 9-3：画实心圆形 52
图 9-4：画实心圆形 53
图 9-5：画带框实心圆形 53
图 9-6：画空心椭圆形 54
图 9-7：画实心椭圆形 55
图 9-8：画带框实心椭圆形 55
图 9-9：画空心矩形 56
图 9-10：画实心矩形 57
图 9-11：画带框实心矩形 57
图 9-12：画空心圆角矩形 58
图 9-13：画实心圆角矩形 59
图 9-14：画带框实心圆角矩形 60
图 9-15：画空心三角形 60
图 9-16：画实心三角形 61

图 9-17：画带框实心三角形 62
图 9-18：左上方曲线 62
图 9-19：左下方曲线 63
图 9-20：右上方曲线 63
图 9-21：右下方曲线 64
图 9-22：左上方 1/4 椭圆 65
图 9-23：左下方 1/4 椭圆 65
图 9-24：右上方 1/4 椭圆 66
图 9-25：右下方 1/4 椭圆 66
图 9-26：画空心四边形 67
图 9-27：画实心四边形 68
图 9-28：画空心五边形 69
图 9-29：画实心五边形 70
图 9-30：圆柱体 71
图 9-31：方柱体 72
图 9-32：横向表格视窗图 73
图 9-33：纵向表格视窗 73
图 10-1：结合光栅操作的 BTE 写入范例 76
图 10-2：结合光栅操作的 MCU 写入流程图 76
图 10-3：结合光栅操作的 BTE 内存复制范例 78
图 10-4：结合光栅操作的 BTE 内存复制流程图（1） 79
图 10-5：结合光栅操作的 BTE 内存复制流程图（2） 79
图 10-6：结合 Chroma Key 的 MCU 写入范例 82
图 10-7：结合 Chroma Key 的 MCU 写入流程图 82
图 10-8：结合 Chroma Key 的内存复制范例 84
图 10-9：结合 Chroma Key 的内存复制流程图 85
图 10-10：图样格式 86
图 10-11：结合光栅操作的图样填满范例 86
图 10-12：结合光栅操作的图样填满流程图 87
图 10-13：结合 Chroma Key 的图样填满范例 88
图 10-14：结合 Chroma Key 的图样填满流程图 89
图 10-15：结合扩展色彩的 MCU 写入范例 90
图 10-16：结合扩展色彩的 MCU 写入流程图 91
图 10-17：色彩扩展显示范例 1 91
图 10-18：色彩扩展显示范例 2 92
图 10-19：色彩扩展显示数据格式 92
图 10-20：结合扩展色彩与 Chroma key 的 MCU 写入范例 94
图 10-21：结合扩展色彩与 Chroma key 的 MCU 写入流程图 94
图 10-22：8bpp Pixel Mode 范例 96
图 10-23：16bpp Pixel Mode 范例 97

图 10-24：结合透明度的内存复制（Pixel Mode）流程图 99
图 10-25：Picture Mode 范例 99
图 10-26：结合透明度的内存复制（Picture Mode）流程图 100
图 10-27：结合透明度的 MCU 写入范例 101
图 10-28：结合透明度的 MCU 写入流程图 102
图 10-29：结合扩展色彩的内存复制范例 103
图 10-30：色彩扩展显示范例 1 104
图 10-31：色彩扩展显示范例 2 104
图 10-32：结合扩展色彩的内存复制流程图 105
图 10-33：结合扩展色彩与 Chroma Key 的内存复制范例 106
图 10-34：结合扩展色彩与 Chroma Key 的内存复制流程图 107
图 10-35：区域填满范例 108
图 10-36：区域填满流程图 108
图 11-1：显示 24*24 楷书中文字 112
图 11-2：显示 48*48 楷书中文字 114
图 11-3：制作 115
图 11-4：选择字体 116
图 11-5：设置字库-1 116
图 11-6：设置字库-2 117
图 11-7：参数设置字体效果 118
图 11-8：预览文字 119
图 11-9：保存字库 120
图 11-10：字库制作完成 120
图 11-11：导出的字库 Bin 文件 121
图 12-1：文字光标的宽度 122
图 12-2：文字光标的高度 123
图 12-3：光标的闪烁范例 123
图 12-4：图形光标范例 124
图 12-5：打开图像光标制作界面 127
图 12-6：导出图形光标数据 128
图 12-7：保存图形光标数据 128
图 12-8：复制导出的光标数据 129
图 12-9：粘贴导出的光标数据 129
图 12-10：复制导入的光标数据 130
图 12-11：粘贴导入的光标数据 130
图 12-12：导入成功 131
图 13-1：PWM 波形图 132
图 13-2：PWM 控制 132
图 13-3：PWM 控制背光参考原理图（7” 1024*600 屏） 134
图 14-1：LT758x 串行 SPI Flash 应用电路 135

图 14-2：LT758x 串行 4 线 QSPI Flash 应用电路 135
图 14-3：Bin 文件整合 140
图 14-4：保存整合文件 141
图 14-5：整合成功 142
图 14-6：保存文件信息 142
图 14-7：导出的 Bin 整合文件 143
图 14-8：调用 SPI Flash 内的字库（范例二） 144
图 15-1：JPG 图像解码显示流程图 145
图 15-2：执行 UartTFT.exe 147
图 15-3：导入图片 148
图 15-4：导入完成 149
图 15-5：设定输出格式 149
图 15-6：导出图片 150
图 15-7：导出成功 151
图 15-8：导出的图片 Bin 文件 151
图 15-9：StartAddr.txt 文件 152
图 16-1：外部中断唤醒 154
图 18-1：宽 80 高 69 的 JPG 图片 163
图 18-2：将 JPG 图片转成 bin 文件 163
图 18-3：复制图片 C 源码 164
图 18-4：图片信息的数组 165
图 18-5：宽 68 高 60 的 JPG 图片 167
图 18-6：将 JPG 图片转成 bin 文件 167
图 18-7：将 bin 文件命名为 ALL.bin 168
图 18-8：SD 卡文件夹命名 168
图 18-9：ALL.bin 文件放入 Updata_Flash 文件夹 169
图 18-10：图片起始地址 169
图 18-11：将图片的起始地址存入数组 170
图 19-1：LT758 时钟电路（一） 171
图 19-2：LT758 时钟电路（二） 171
图 19-3：LT758 时钟电路（三） 172
图 19-4：外部复位方式（1） 172
图 19-5：外部复位方式（2） 172
图 19-6：LT758x 与 TFT 屏驱动器的接口 174
图 20-1：STM32F407+LT7586B 演示板 176
图 20-2：STM32F407+LT7586 演示板原理图 178
图 21-1：STM32F103+LT7580 演示板 179
图 21-2：STM32F103+LT7580 演示板原理图 180

表附录

表 2-1：LT758x 型号比较表 17
表 4-1：PLL 寄存器 - Feedback Divider Ratio（M） 23
表 4-2：PLL 寄存器 - Input Divider Ratio（N） 23
表 4-3：PLL 寄存器 - Output Divider Ratio（OD） 24
表 5-1：MCU 接口模式设定 26
表 5-2：不同 MCU 模式的接口定义 26
表 6-1：LT758x 内存容量 38
表 7-1：LT758x LCD 接口对应 RGB 的数据 39
表 7-2：不同型号的 LT758x 所支持的 RGB 数据 40
表 10-1：BitBLT 的工作模式 74
表 10-2：光栅操作模式（ROP Function） 75
表 10-3：Alpha Blending Pixel Mode -- 8bpp 96
表 10-4：Alpha Blending Pixel Mode -- 16bpp 97
表 10-5：Alpha Blending Effect 98
表 12-1：字光标相关的寄存器表 122
表 12-2：图形光标像素定义 124
表 16-1：电源管理模式的时钟动作比较表 153
表 17-1：程序库列表 156
表 19-1：LT7586B MCU 接口模式设定 173
表 19-2：LT7580 MCU 接口模式设定 173

1. 前言

本应用手册主要在说明LT758x（LT7586B、LT7580）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 LT758x 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。

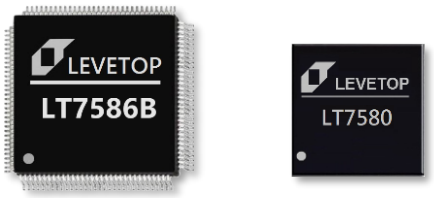


图 1-1：LT758x TFT Controller

1.1 手册说明

手册中除了硬件及软件的设置说明外，在最后几章提供了两个点屏范例，并对 TFT 模块厂将 LT758x 设计到 TFT 模块上时所要注意的事项，及 SPI Flash 烧录的方式做了完整说明，同时介绍了本公司所提供的三个 STM32 + LT758x 演示板。

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 LT758x 寄存器控制的软件，还有配合演示板的 STM 32 位 MCU 的演示程序，同时本公司还提供一个专用整合软件 UartTFT_V4.30.exe（或 V4.30 以上的版本）可以用来制作图片、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，取得方式请参考第 17.2 节。原理图的部分，我们提供了 STM32F407VE 代表 32 位 MCU 与 LT758x 连接的原理图，还有将 LT758x 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。

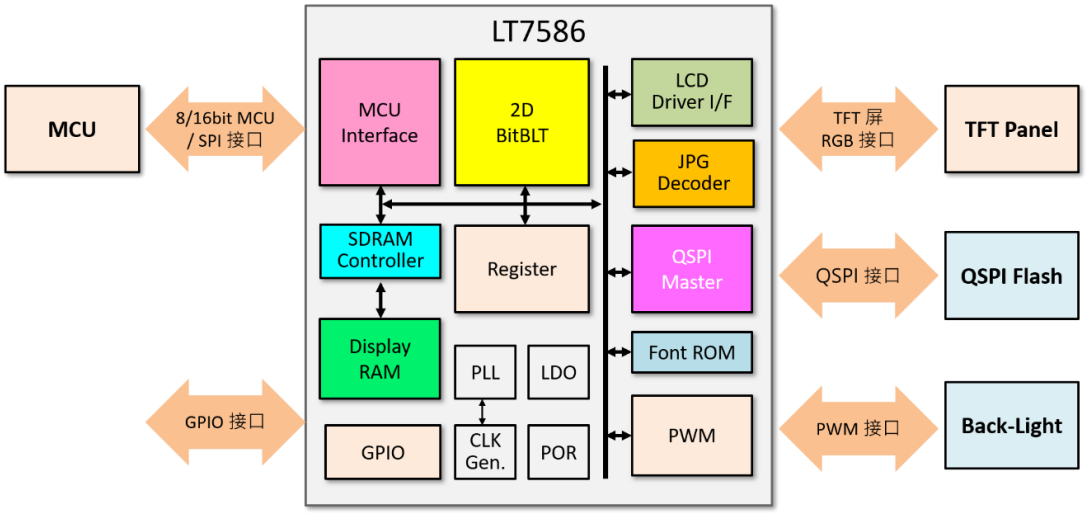


图 1-2：LT758x 内部方块图

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 控制器的脚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。

LT758x 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，LT758x 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 LT758x 内建的高容量显示内存（Display RAM，64/128Mbits），然后 LT758x 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 LT758x 的基本应用架构：

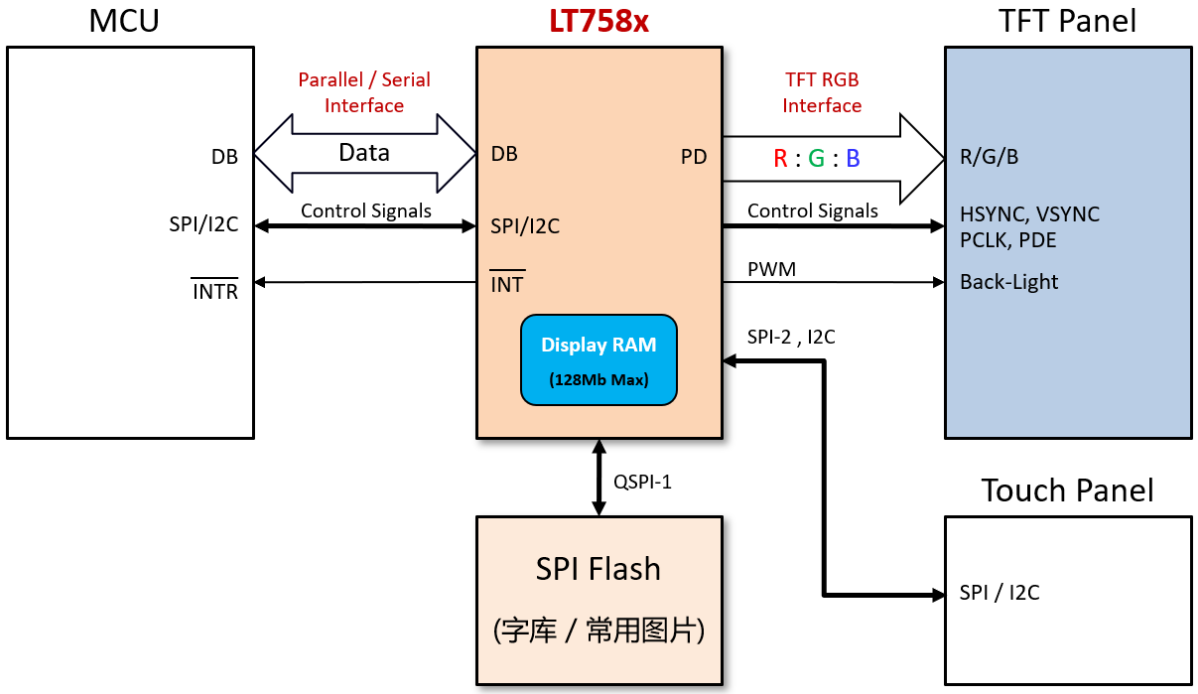


图 1-3：LT758x 应用架构图

LT758x 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 LT758x 上，再由 MCU 透过 LT758x 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

2.1 芯片脚位图

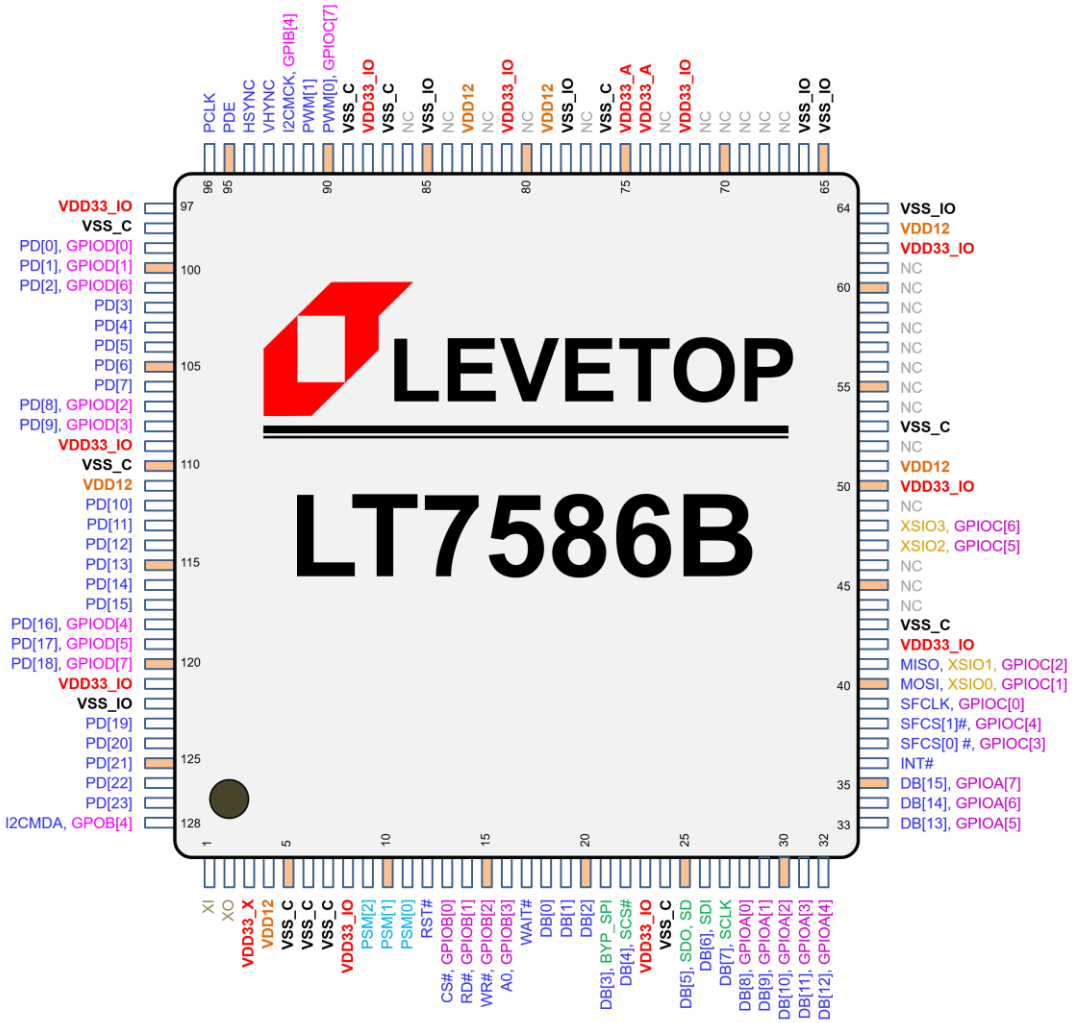


图 2-1：LT7586B 引脚图（LQFP-128Pin）

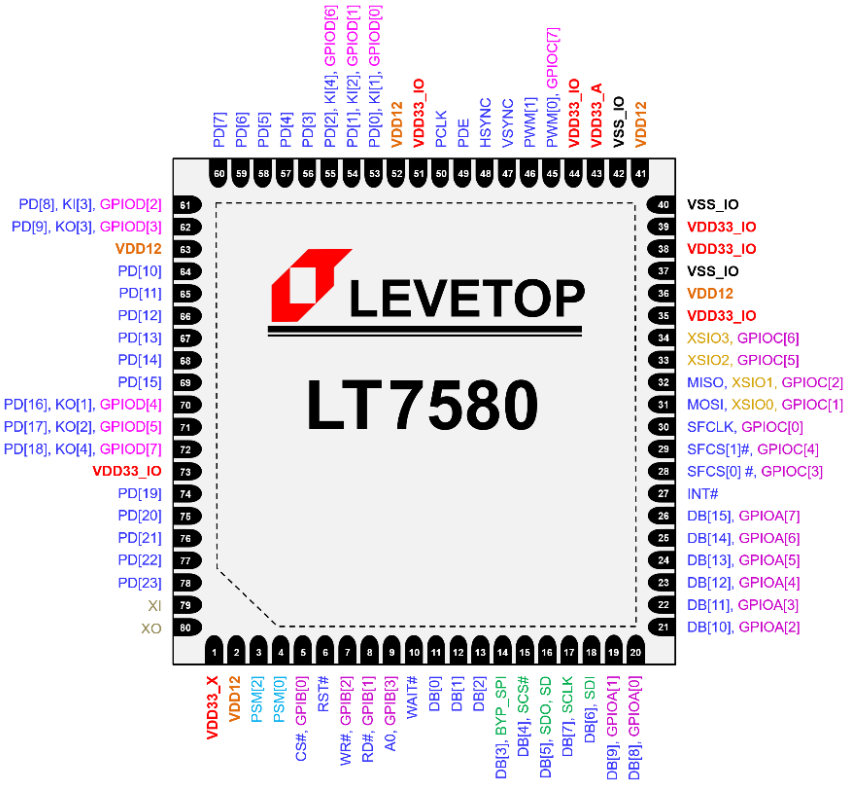


图 2-2：LT7580 引脚图（QFN-80Pin）

2.2 LT758x 比较表

表 2-1：LT758x 型号比较表

支持功能		LT7580	LT7586B
类别	功能说明		
封装	LQFP / QFN	QFN-80	LQFP-128
LCD 规格	分辨率	480*480 | 1024*768	480*480 | 1280*1024
	色彩	65K / 16M 色	65K / 16M 色
	Alpha RGB	8:8:8:8	4:4:4:4
	TFT 接口	RGB (16bits/24bits)	RGB (16bits/24bits)
显示解碼	显示内存	128Mbit	128Mbit
	JPG 解碼	V	V
MCU 接口	8080 8bit	V	V
	6800 8bit	--	V
	8080 16bit	V	V
	6800 16bit	--	V
	3 线 SPI 串口	V	V
	4 线 SPI 串口	V	V
	I2C 串口	--	V（支持连读模式）
其他接口	SPI Master (DMA Flash)	QSPI (2 组)	QSPI (2 组)
	SPI I/F by Pass Mode	V	V
	NAND Flash 坏块处理	V	V
	I2C Master	--	V
	PWM 输出	1	2
	GPIO 输出	V	V
绘图功能	2D 绘图加速器	V	V
	几何绘图加速器	V	V
	画中画 PIP	V	V
	Virtual Display	V	V
	垂直画面卷动	V	V
	水平画面卷动	V	V
	画面旋转	V	V
	Alpha-Blending	V	V
	图形光标	V	V
	彩条测试	V (V & H)	V (V)
文字功能	内建英文字库	ISO8259	ISO8259
	支持中文字库 (外挂 Flash)	V (16*16, 24*24, 32*32)	V (16*16, 24*24, 32*32)
	文字放大	4*4 倍	4*4 倍
	文字旋转	V	V
	文字光标	V	V
	自定义文字	V	V
电源	休眠模式 (S_{tandby}/S_{uspend}/S_{leep})	V	V
	电源	3.3V	3.3V

LT7580 是 80Pin QFN 封装，LT7586 是 128pin LQFP 封装；分辨率则是向下兼容，例如 LT7586 的分辨率是 1280*1024，它也可以用在更低分辨率的 TFT 屏上。

LT7580 是 80Pin 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 8 位並口或是 SPI 串口的标准 TFT 模块，如下图 2-4 所示。也可以将标准 TFT 驱动模块加上 LT758x 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 2-5 所示。

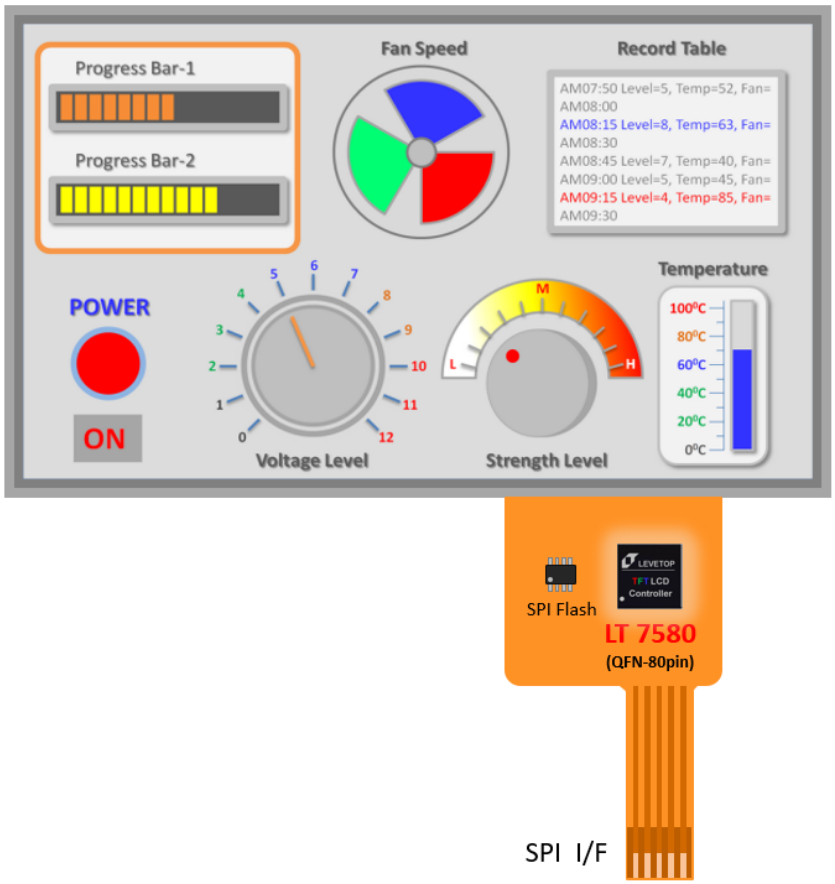


图 2-3：LT7580 标准 SPI/8Bit TFT 模块

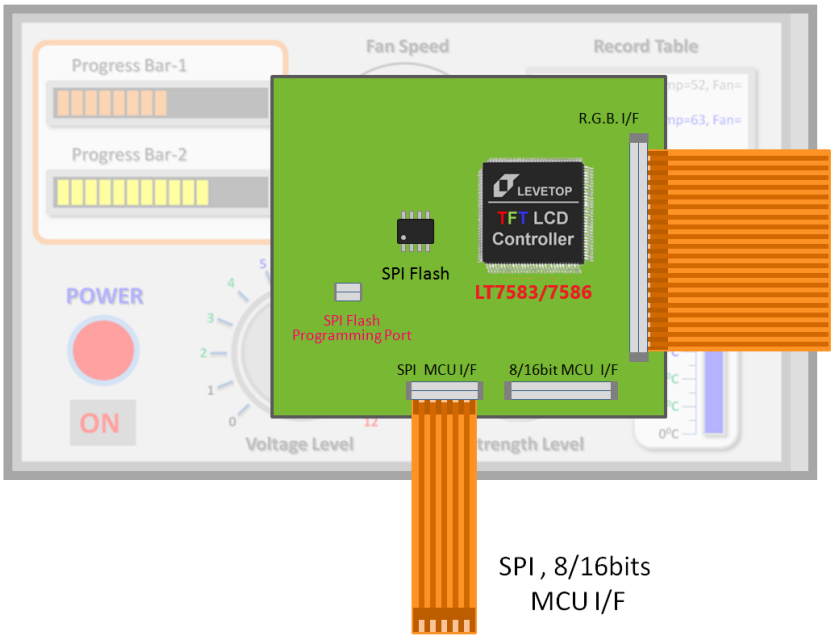


图 2-4：标准带控制器的 TFT 模块

LT758x 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 2-6。

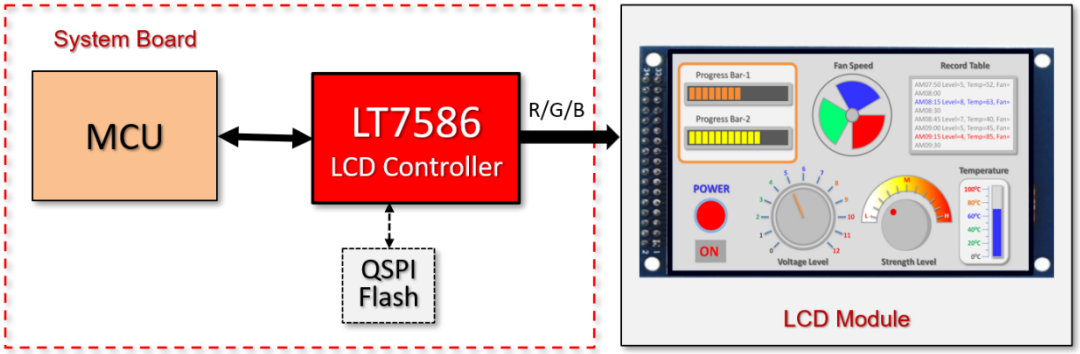


图 2-5：LT7586 设置在系统主板上

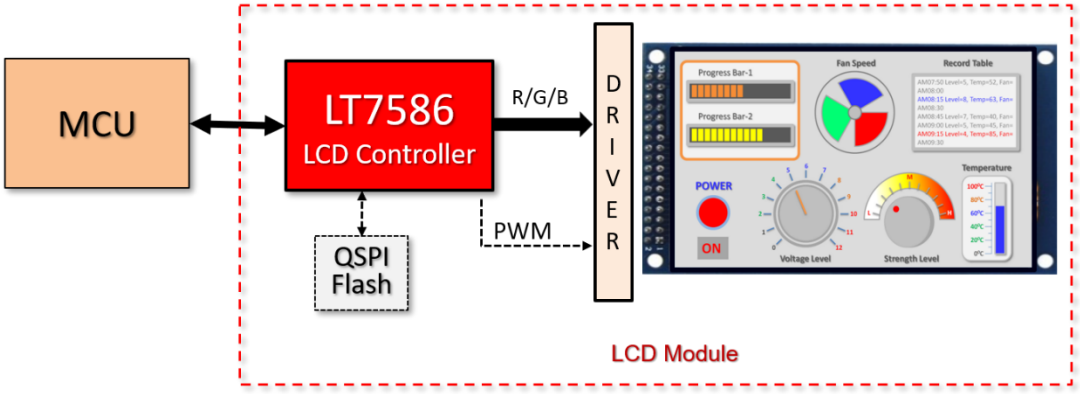


图 2-6：LT7586 设置在 LCD 模块上

3. 复位

3.1 电源开启复位

LT758x 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 32 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 LT758x 与外部系统同步，外部复位信号必须稳定至少 32 个晶振频率（OSC）时钟才会被承认，如图 3-1 所示。而 MCU 在开始设定 LT758x 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 LT758x 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。

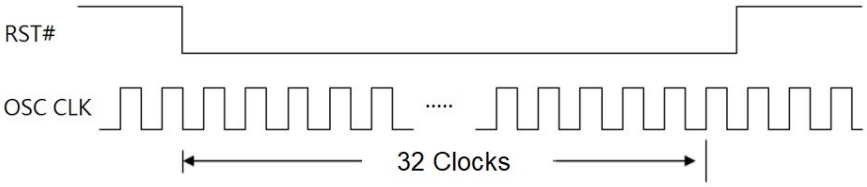


图 3-1：外部复位信号

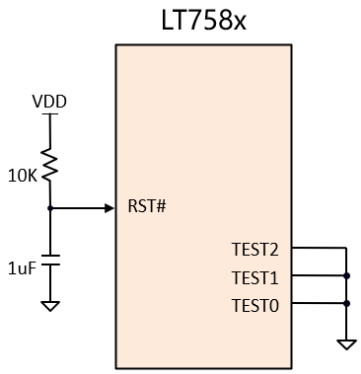


图 3-2：外部复位方式（1）

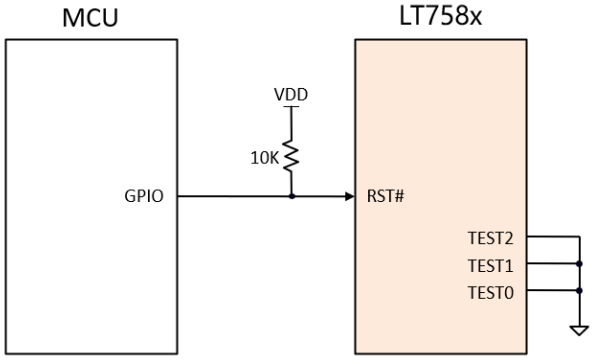


图 3-3：外部复位方式（2）

3.3 软件复位

如果 MCU 对寄存器 REG[00h] bit0 写入 1，LT758 将会进行软件复位（Software Reset），软件复位只会复位 LT758 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 LT758_SW_Reset(void);
```

提示： 本应用手册提到的应用函数，使用者可以到乐升半导体的官网下载区下载“LT758_Library_202xxxxx.rar”再解压缩后取得，详细的说明请参考第 17.1 节。

3.4 测试信号

LT758x 的 TEST[2:0]是测试模式信号，这些引脚是提供给 LT758x 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 LT758x SPI Master 上的 SPI Flash 进行数据更新，可以把 LT758x 的 TEST[2] 引脚拉低、TEST[1] 引脚拉高，让 LT758x 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 LT758x 的影响（请参考本手册的第 20.3 节的说明）

4. 时钟设定

4.1 时钟与 PLL

LT758 需要外接一个 12MHz 的晶振，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：

- **CPLL** ：产生 **CCLK**（Core Clock）提供 MCU 接口、BTE 引擎、绘图引擎、文字 DMA 引擎使用，如使用 12MHz 的晶振则预设频率为 96MHz，最大 170MHz。
- **MPLL**：产生 **MCLK**（Memory Clock）以提供给内部显示内存使用，预设 96MHz。，最大 180MHz。
- **PPLL** ：产生 **PCLK**（Pixel Clock）提供 LCD 屏幕扫描工作频率，预设 36MHz，最大 100MHz。

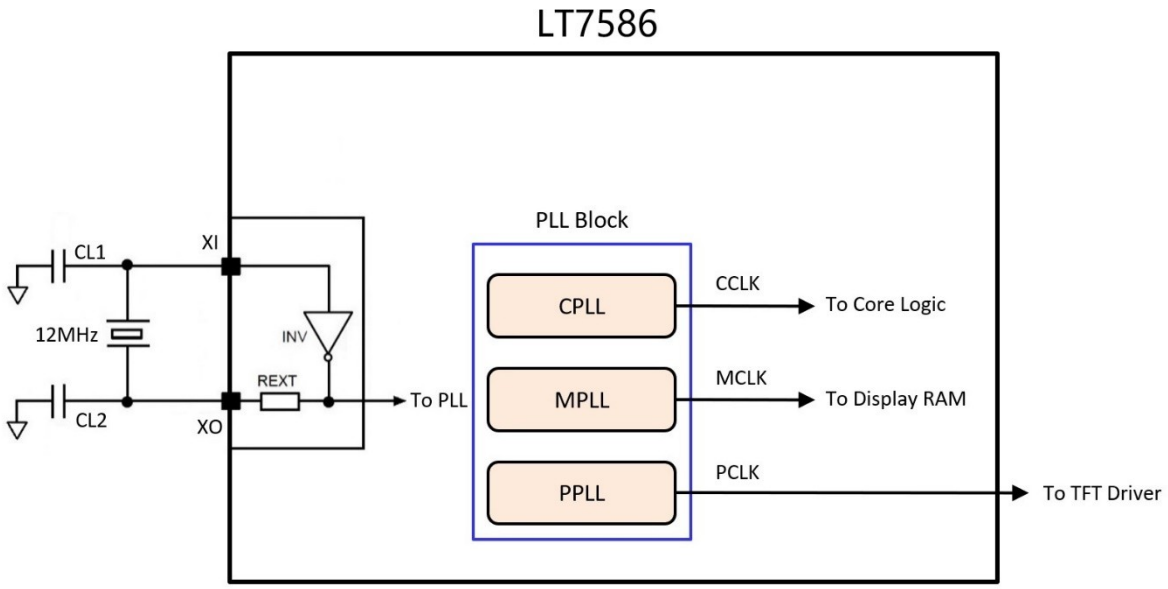


图 4-1：LT758x 时钟电路

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：

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

XI 是外部晶振/时钟信号输入，建议值为 12MHz；M 是 Feedback Divider Ratio of Loop，共 8 个 bits，数值介于 4~255 之间；N 是输入除频器比率值（Input Divider Ratio），介于 1~15 之间；OD 是输出除频器比率值（Output Divider Ratio），可以设成 1、2、4 或是 8。

PLL 寄存器的设定规则除了上述公式外，还需要遵循下面的条件**:**

a) $3.5MHz \le XI \div N \le 35MHz$

b) $200MHz \le F_{OUT} * OD \le 400MHz$

c) $M \ge 4; N \ge 1;$

表 4-1：PLL 寄存器 - Feedback Divider Ratio（M）

M[7:0]	Feedback Divider Ratio（M）
0000_0000	NA
0000_0001	NA
0000_0010	NA
0000_0011	NA
0000_0100	4
0000_0101	5
0000_0110	6
⋮	⋮
1111_1101	253
1111_1110	254
1111_1111	255

表 4-2：PLL 寄存器 - Input Divider Ratio（N）

N[3:0]	Input Divider Ratio（N）
0000	NA
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

表 4-3：PLL 寄存器 - Output Divider Ratio（OD）

OD[1:0]	Output Divider Ratio（OD）
00	1
01	2
10	4
11	8

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。

依照 Panel 分辨率大小不同，每个 PLL 输出（F_{OUT}）要设定不同，以 640*480 Panel 分辨率为例，如果 TFT 屏厂商建议 PCLK = 30MHz，则可以选择设 MCLK = 60MHz，CCLK = 60MHz，那么各 PLL 输出与寄存器设定如下：

PCLK = XI *（M ÷ N）÷ OD
= 12MHz *（60 ÷ 3）÷ 8
= 30MHz
REG[06h] [7:0] (M) = 3Ch,
REG[05h] [4:1] (N) = 0011b,
REG[05h] [7:6] (OD) = 11b,

MCLK = XI *（M ÷ N）÷ OD
= 12MHz *（60 ÷ 3）÷ 4
= 60MHz
REG[08h] [7:0] (M) = 3Ch,
REG[07h] [4:1] (N) = 0011b,
REG[07h] [7:6] (OD) = 10b,

CCLK = XI *（M ÷ N）÷ OD
= 12MHz *（60 ÷ 3）÷ 4
= 60MHz
REG[0Ah] [7:0] (M) = 3Ch,
REG[09h] [4:1] (N) = 0011b,
REG[09h] [7:6] (OD) = 10b,

4.2 时钟的启始设定

在 **LT758_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 LT758_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

5.1 MCU 硬件接口

LT758x 的动作是受到外部 MCU 所控制，而 MCU 是通过接口直接对 LT758x 寄存器或是显示内存（Display RAM）进行资料的读写。LT758x 提供了 2 种 8 位、16 位的并行接口，以及 SPI、I2C 的串行接口，让不同的 MCU 以适合的接口来控制 LT758x。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	LT7586B	LT7580
0 0 X	选择并口 8 位或 16 位的 8080 模式	V	V
0 1 X	选择并口 8 位或 16 位的 6800 模式	V	--
1 0 0	选择串口 3 线式 SPI 模式	V	V
1 0 1	选择串口 4 线式 SPI 模式（支持旁通 By-Pass 模式）	V	V
1 1 0	选择串口 I2C 模式（不支持连读模式）	V	--
1 1 1	选择串口 I2C 模式（支持连读模式）	V	--

由于不同的 MCU 接口无法同时被使用，因此 LT758x 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 5-2：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]					接地	**BYP_SPI**(1)	I2CA[3]
DB[2:0]					接地	接地	I2CA[2:0]
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#		--	--	--

提示(1)：详细请参考 LT758x 规格书第 13.2.1 节的说明。

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。

提示：所有寄存器访问仅为 8 位，内存数据端口除外。即使 MCU 接口为 16 位宽，LSB（DB[7:0]）也用于除内存数据端口（REG[04h]）以外的所有寄存器。对于内存数据端口（REG[04h]），当 MCU 接口类型设定位（REG[01h]、bit[0]）设置为 16 位宽时，使用全 16 位，而当它设置为 8 位宽时，仅使用低 8 位。参考 LT758x 规格书第 17 章寄存器说明。

以下为不同的 MCU 接口参考电路图：

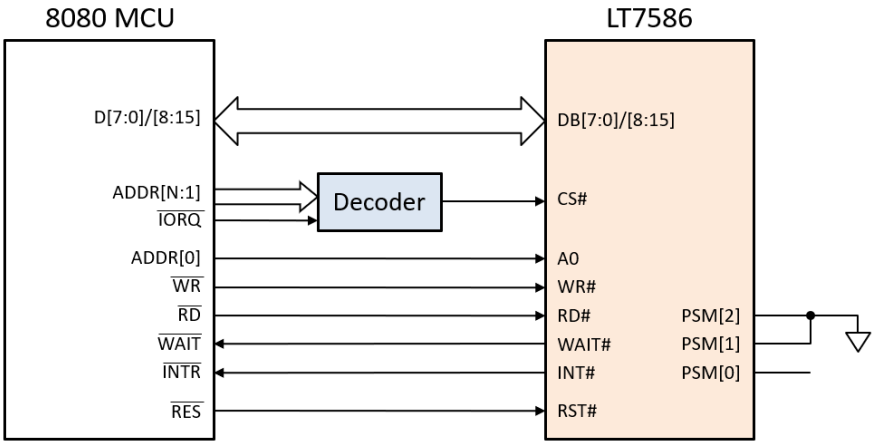


图 5-1：8080 MCU 并口电路图

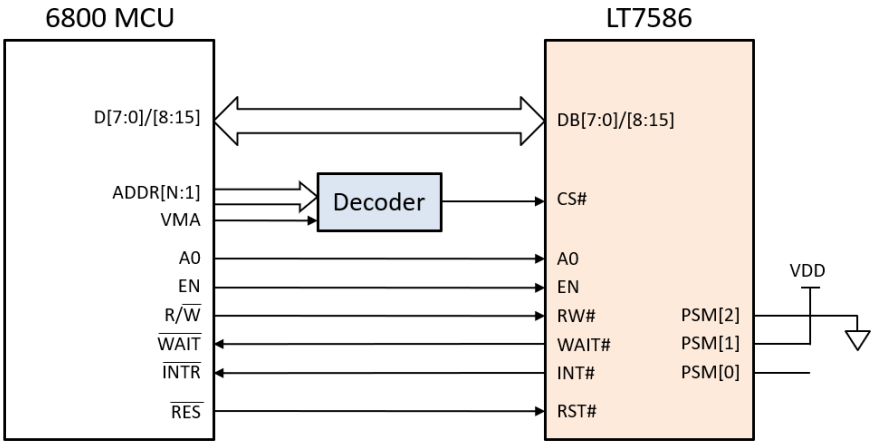


图 5-2：6800 MCU 并口电路图

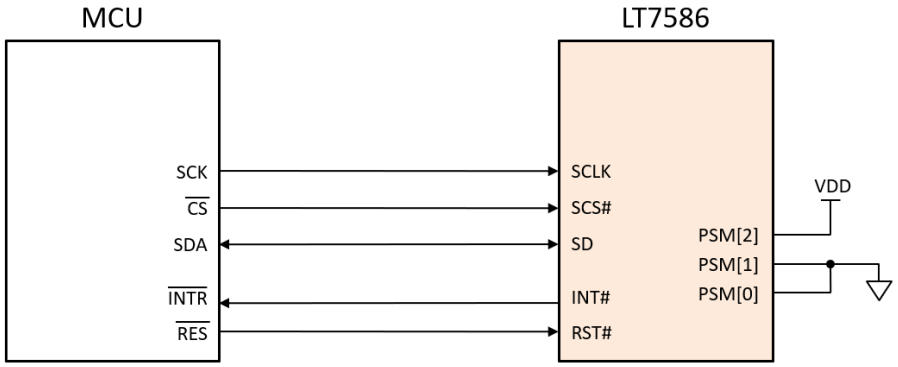


图 5-3：MCU3 线 SPI 串联接口电路图

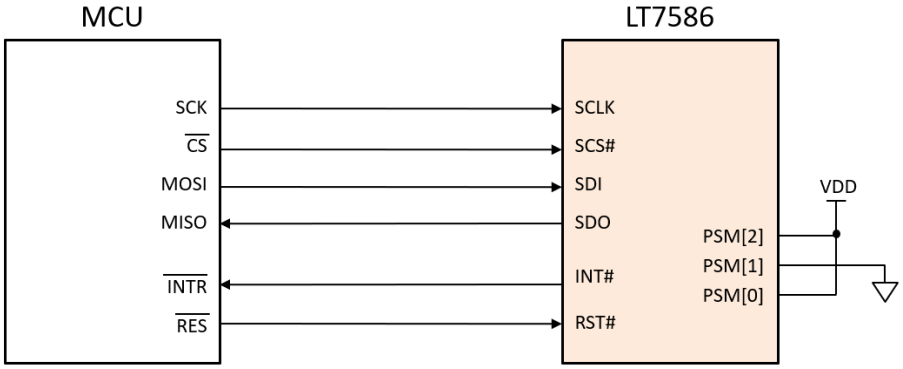


图 5-4：MCU4 线 SPI 串联接口电路图

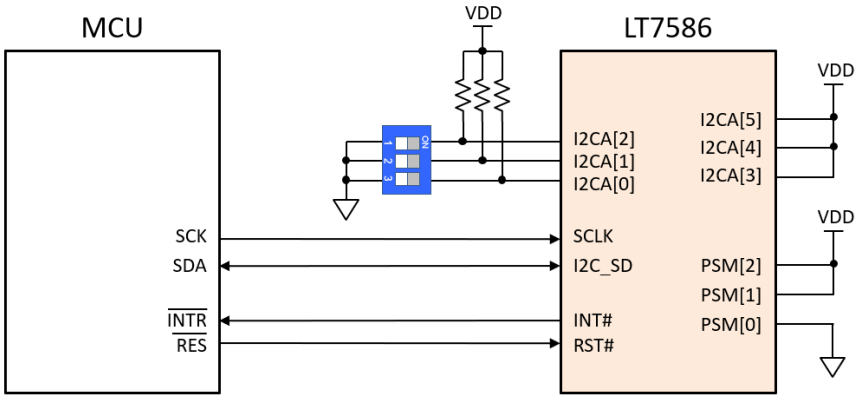


图 5-5：MCU I2C 串联接口电路图（不支持连读模式）

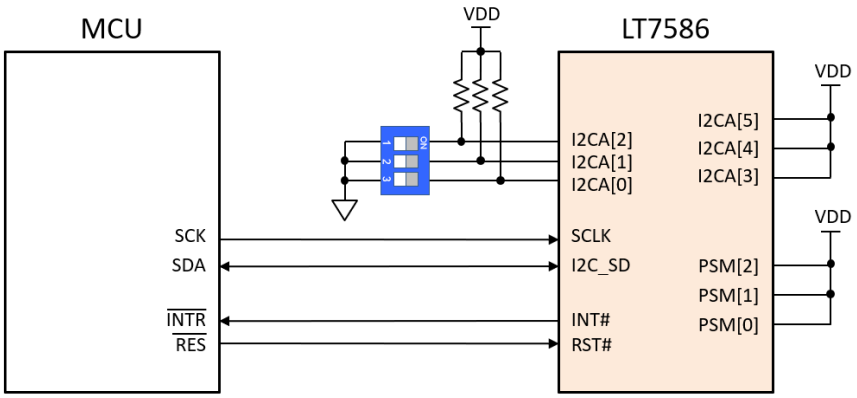


图 5-6：MCU I2C 串联接口电路图（支持连读模式）

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 LT758 的其他的功能都是基于以上的 5 个函数来实现的。

```
1、void LT758_CmdWrite(u8 cmd);              // 向 LT758 写命令
2、void LT758_DataWrite(u8 data);            // 向 LT758 写数据
3、void LT758_DataWrite_Pixel(u16 data);     // 向 LT758 写像素点的数据
4、 u8 LT758_StatusRead(void);               // 读状态寄存器
5、 u16 LT758_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 FSMC 来模拟 8 位的 8080 接口：**

```
void FSMC_8_CmdWrite(u8 cmd)
{
*(vu8*) (LCD_BASE0)= (cmd);
}

void FSMC_8_DataWrite(u8 data)
{
*(vu8*) (LCD_BASE1)= (data);
}
```



```
void FSMC_8_DataWrite_Pixel(u16 data)
{
*(vu8*) (LCD_BASE1)= (data);
*(vu8*) (LCD_BASE1)= (data>>8);
}

void FSMC_8_DataWrite_Pixel_24(u32 data)
{
*(vu8*) (LCD_BASE1)= (data);
*(vu8*) (LCD_BASE1)= data>>8);
*(vu8*) (LCD_BASE1)= data>>16);
}
u8 FSMC_8_StatusRead(void)
{
u8 temp = 0;
temp = *(vu8*)(LCD_BASE0);
return temp;
}

u16 FSMC_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 FSMC 来模拟 16 位的 8080 接口：**

```
void FSMC_16_CmdWrite(u8 cmd)
{
*(vu16*) (LCD_BASE0)= (cmd);
}

void FSMC_16_DataWrite(u8 data)
{
*(vu16*) (LCD_BASE1)= (data);
}

void FSMC_16_DataWrite_Pixel(u16 data)
{
*(vu16*) (LCD_BASE1)= (data);
}

void FSMC_16_DataWrite_Pixel_24(u32 data)
{
*(vu16*) (LCD_BASE1)= (data);
*(vu16*) (LCD_BASE1)= (data >> 16);
}

u8 FSMC_16_StatusRead(void)
{
u8 temp = 0;
temp = *(vu16*)(LCD_BASE0);
return temp;
}

u16 FSMC_16_DataRead(void)
{
u16 temp = 0;
temp = *(vu16*)(LCD_BASE1);
return temp;
}
```


5.4 使用 8 位 SPI 接口

```
void SPI_CmdWrite_8(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_8(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_8(u16 data)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x80);
    SPI2_ReadWriteByte(data);
    SS_SET;

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



```
void SPI_DataWrite_Pixel_8_24(u32 data)
{
    SS_RESET;
    SPI2_ReadWriteByte(0x80);
    SPI2_ReadWriteByte(data);
    SS_SET;

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

    SS_RESET;
    SPI2_ReadWriteByte(0x80);
    SPI2_ReadWriteByte(data>>16);
    SS_SET;
}

u8 SPI_StatusRead_8(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_8(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;
}
```


5.5 使用 16 位 SPI 接口

```
void SPI_CmdWrite_16(u8 cmd)
{
 SS_RESET;
 SPI2_ReadWriteByte(0x00<<8|cmd);
 SS_SET;
}

void SPI_DataWrite_16(u8 data)
{
 SS_RESET;
 SPI2_ReadWriteByte(0x80<<8|data);
 SS_SET;
}

void SPI_DataWrite_Pixel_16(u16 data)
{
 SS_RESET;
 SPI2_ReadWriteByte(0x80<<8|(data&0x00ff));
 SS_SET;

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

void SPI_DataWrite_Pixel_16_24(u32 data)
{
 SS_RESET;
 SPI2_ReadWriteByte(0x80<<8|(data&0x00ff));
 SS_SET;

 SS_RESET;
 SPI2_ReadWriteByte((0x80<<8)|((data>>8)&0x00ff));
 SS_SET;

 SS_RESET;
 SPI2_ReadWriteByte((0x80<<8)|((data>>16)&0x00ff));
 SS_SET;
}
```



```
u8 SPI_StatusRead_16(void)
{
 u8 temp = 0;
 SS_RESET;
 temp = SPI2_ReadWriteByte(0x40ff);
 SS_SET;
 return temp;
}

u16 SPI_DataRead_16(void)
{
 u16 temp = 0;
 SS_RESET;
 temp = SPI2_ReadWriteByte(0xc0ff) & 0x00ff;
 SS_SET;
// SS_RESET;
// temp |= ((SPI2_ReadWriteByte(0xc0ff)  & 0x00ff) << 8);
// SS_SET;
 return temp;
}
```


5.6 使用 I2C 接口

```
u8   IIC_CmdWrite(u8 cmd)
{
     CT_IIC_Start();
     CT_IIC_Send_Byte(LT_ADDR);          // 代表 MCU 要写入指令寄存器地址
     If (IIC_Wait_Ack())                 // 等待应答
     {
       CT_IIC_Stop();
       return 1;
     }
     CT_IIC_Send_Byte(cmd);
     If (IIC_Wait_Ack())                 // 等待 ACK
     {
        CT_IIC_Stop();
        return 1;
     }
     CT_IIC_Stop();
     return 0;
}

u8   IIC_DataWrite(u8 data)
{
     CT_IIC_Start();
     CT_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
     {
        CT_IIC_Stop();
        return 1;
     }
     IIC_Send_Byte(data);
     If (IIC_Wait_Ack())                 // 等待 ACK
      {
        CT_IIC_Stop();
        return 1;
     }
     CT_IIC_Stop(); return 0;
}
```



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

void IIC_DataWrite_Pixel_24(u32 data)
{
    IIC_DataWrite(data);
    IIC_DataWrite(data>>8);
    IIC_DataWrite(data>>16);
}

u8 IIC_StatusRead(void)
{
    u8 res;
    CT_IIC_Start();
    CT_IIC_Send_Byte(LT_ADDR|0x01);          // 代表 MCU 要读取状态寄存器的数据
    IIC_Wait_Ack();
    res=CT_IIC_Read_Byte(0);
    CT_IIC_Ack();
    CT_IIC_Stop();
    return res;
}

u8 IIC_DataRead(void)
{
    u8 res;
    CT_IIC_Start();
    CT_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);
    CT_IIC_Ack();
    CT_IIC_Stop();
    return res;
}
```

提示:

1、LT_ADDR 所代表的地址是通过 LT758x 的引脚 I2CA[5:0]设定。

2、具体的内容可以参考 LT758x 规格书第 2.3 章节的 MCU 串行接口中的 IIC 接口。

6. 显示内存（SDRAM）设定

LT758x 所支持的显示内存容量为 128Mbit，如下表所示：

表 6-1：LT758x 内存容量

型 号	内 建 显 示 内 存	分 辨 率（最大）
LT7580	128Mb	1024*768
LT7586B	128Mb	1024*1024

显示内存的设定方式如下：

```
void LT758_SDRAM_initial(u8 MCLK);
```

举例：如果要使用显示内存，设置 MCLK 为 100MHz：

```
LT758_SDRAM_initial(100);
```

提示：参数 MCLK 的数值需要和时钟 PLL 的设置一样。
以下为显示内存初始化函数：

```
void LT758_SDRAM_initial(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

LT758x 支持 16、24bits RGB 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 RGB 接口将信号送到 TFT 面板上的驱动器。LT758x 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 或是 24bits。请参考表 7-2 不同型号的 LT758x 所支持的 RGB 数据。

表 7-1：LT758x LCD 接口对应 RGB 的数据

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

表 7-2：不同型号的 LT758x 所支持的 RGB 数据

型 号	LCD 数据线	RGB 接口数	分辨率	色彩
LT7586B	PD[23~0]	R:G:B = 8:8:8	1280*1024	16.7M 色
	PD[23~19], PD[15~10], PD[7~3]	R:G:B = 5:6:5		65K 色
LT7580	PD[23~0]	R:G:B = 8:8:8	1024*768	16.7M 色
	PD[23~19], PD[15~10], PD[7~3]	R:G:B = 5:6:5		65K 色

除了 RGB 信号外，LT758x 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。

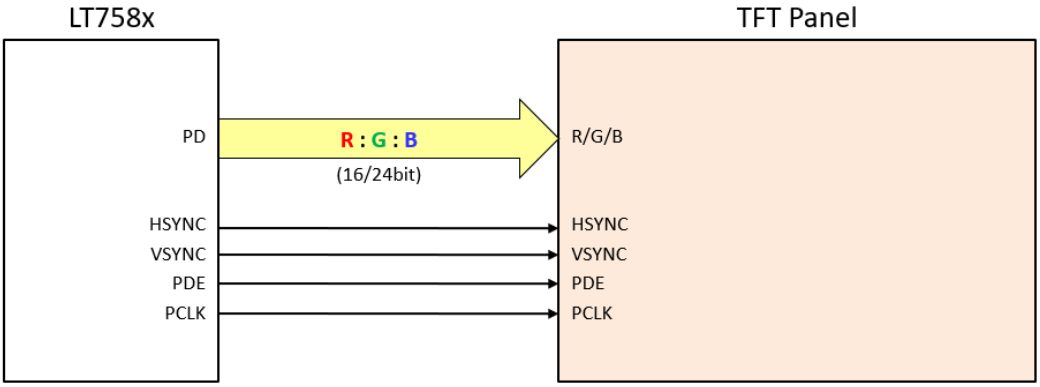


图 7-1：LT758x 与 TFT 屏驱动器的接口

下图是 LT758x 输出到 TFT-LCD 的接口时序图，除了上述的 PD 数据线外还提供了 PCLK 屏幕扫描时钟信号、VSYNC 垂直同步信号、HSYNC 水平同步信号、屏幕数据使能信号。PCLK 的频率是由 REG[05h]、[06h] 所设定，请参考第 4.1 节及 LT758x 规格书第 17 章的寄存器说明。

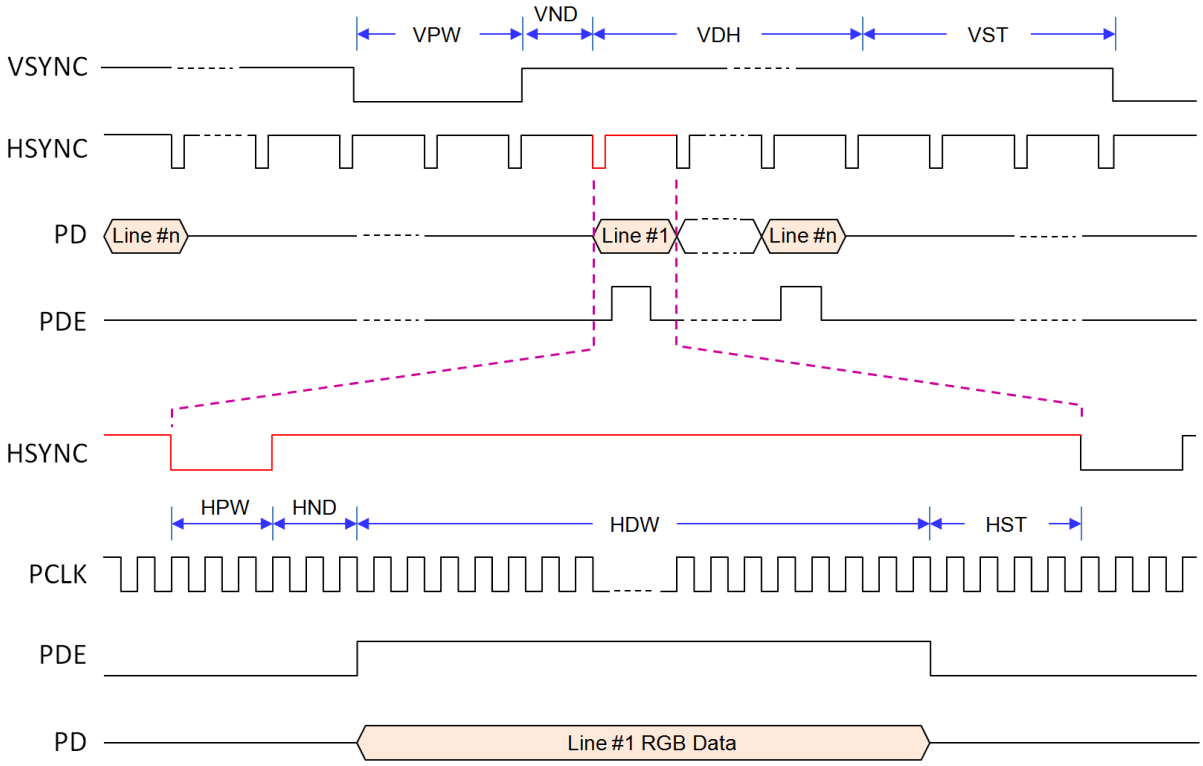


图 7-2：TFT-LCD RGB 接口时序图

7.2 LCD 屏设定

针对不同的 LCD 屏，LT758x 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
//PDATA_Set_GRAY();      // Select GRAY 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 LT758x 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_16bpp();          // 65K 色
Select_Main_Window_24bpp();          // 16.7M 色
```

- **设置主视窗的开始显示地址**

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

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**

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

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：

- **工作视窗大小的设置**

```
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 LT758x 有 8 位和 16 位，而且 LT758 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_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_16Memory_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 LT758x 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

```
LT758_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 LT758_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。

```
LT758_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 LT758_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_Dis,      // 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
Enable_PIP1();               // 使能 PIP1
Enable_PIP1();               // 使能 PIP2
```


8.5 旋转与镜像

LT758x 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

LT758x 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   Color_Bar_ON(void);          // 显示彩条
void   Color_Bar_OFF(void);         // 关闭彩条
```

图 8-1：彩条（Color Bar）显示

当 REG[12h] bit4（VDIR）为 0 时，显示水平方向的彩条（如上图），当 VDIR 为 1 时，显示垂直方向的彩条（如下图）。

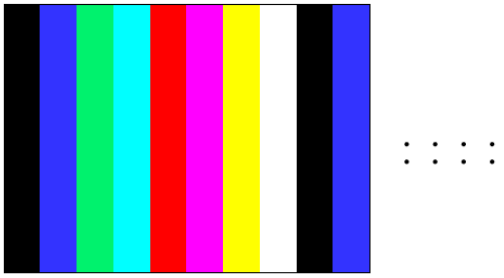


图 8-2：垂直方向的彩条（Color Bar）显示

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 LT758_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
)
```

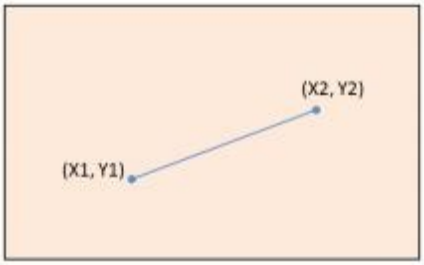


图 9-1：画细线

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

```
LT758_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 LT758_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
)
```

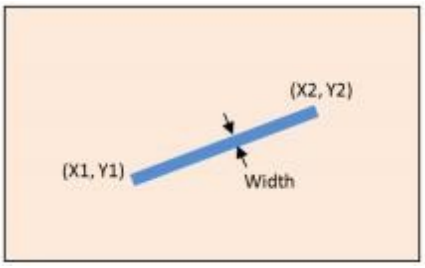



图 9-2：画粗线

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）。

```
LT758_DrawLine_Width(120, 140, 220, 260, Red, 10);
```

9.2 画圆形

9.2.1 画空心圆形

```
void LT758_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
 )
```

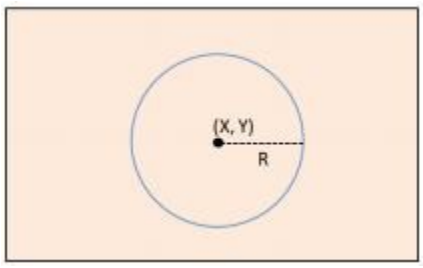


图 9-3：画实心圆形

举例：画一个圆心为坐标（200, 200)、半径 100 的红色空心圆。

```
LT758_DrawCircle(200, 200, 100, Red);
```


9.2.2 画实心圆形

```
void LT758_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
)
```

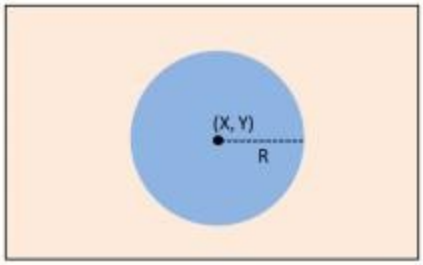


图 9-4：画实心圆形

举例：画一个圆心为坐标（200, 200)、半径 100 的红色实心圆。

```
LT758_DrawCircle_Fill(200, 200, 100, Red);
```

9.2.3 画带框实心圆形

```
void LT758_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
)
```

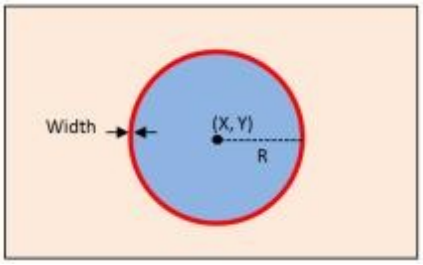


图 9-5：画带框实心圆形

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。

```
LT758_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 LT758_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
)
```

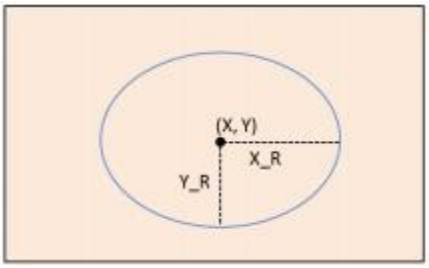


图 9-6：画空心椭圆形

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。

```
LT758_DrawEllipse(100, 100, 80, 50, Red);
```


9.3.2 画实心椭圆形

```
void LT758_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
)
```

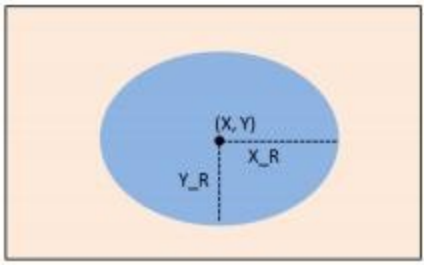


图 9-7：画实心椭圆形

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。

```
LT758_DrawEllipse_Fill(100, 100, 80, 50, Red);
```

9.3.3 画带框实心椭圆形

```
void LT758_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
)
```

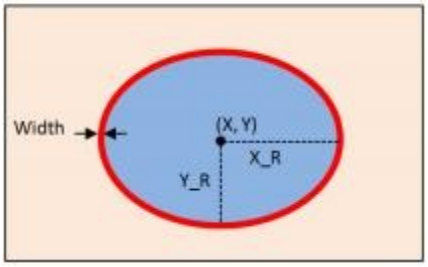


图 9-8：画带框实心椭圆形

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：

```
LT758_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 LT758_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
)
```

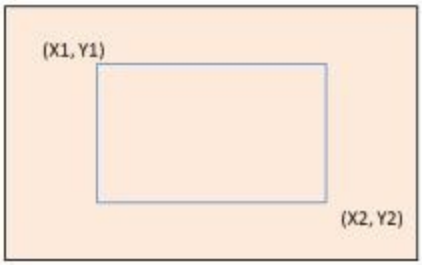


图 9-9：画空心矩形

举例：画一个起点(50, 60)、终点(200, 150)的红色空心矩形。

```
LT758_DrawSquare(50, 60, 200, 150, Red);
```


9.4.2 画实心矩形

```
void LT758_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
)
```

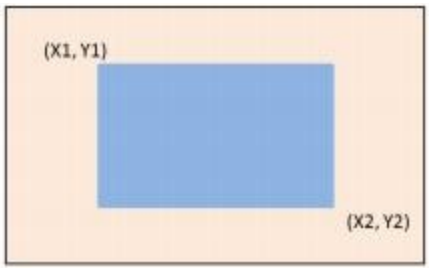


图 9-10：画实心矩形

举例：画一个起点(50,60)、终点(200,150)的红色实心矩形。

```
LT758_DrawSquare_Fill(50, 60, 200, 150, Red);
```

9.4.3 画带框实心矩形

```
void LT758_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
)
```

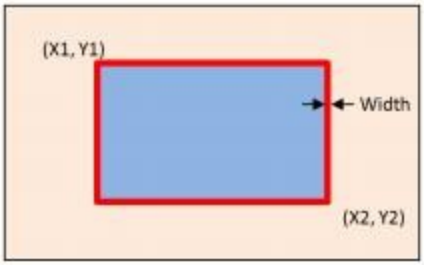


图 9-11：画带框实心矩形

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。

```
LT758_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 LT758_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
)
```

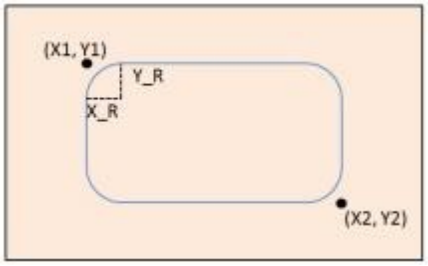


图 9-12：画空心圆角矩形

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：

```
LT758_DrawCircleSquare(50, 60, 200, 150, 30, 20, Red);
```


9.5.2 画实心圆角矩形

```
void LT758_DrawCircleSquare_Fill
(
unsigned short X1,              // X1 位置
unsigned short Y1,              // X1 位置
unsigned short X2,              // X1 位置
unsigned short Y2,              // X1 位置
unsigned short X_R,             // 宽半径
unsigned short Y_R,             // 长半径
unsigned long ForegroundColor   // 前景颜色
)
```

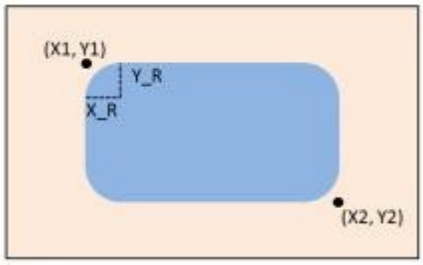


图 9-13：画实心圆角矩形

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：

```
LT758_DrawCircleSquare_Fill(50, 60, 200, 150, 30, 20, Red);
```

9.5.3 画带框实心圆角矩形

```
void LT758_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
)
```

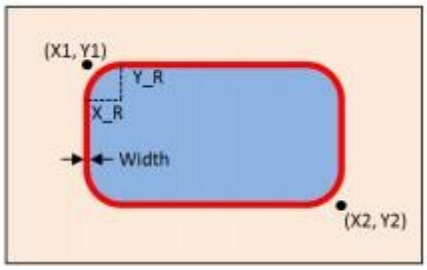



图 9-14：画带框实心圆角矩形

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：

```
LT758_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

```
void LT758_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
)
```

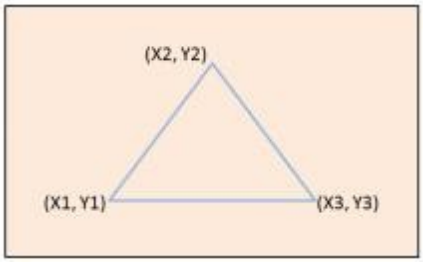


图 9-15：画空心三角形

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：

```
LT758_DrawTriangle(100, 100, 50, 200, 150, 150, Red);
```


9.6.2 画实心三角形

```
void LT758_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
)
```

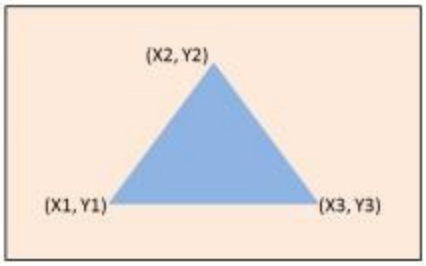


图 9-16：画实心三角形

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：

```
LT758_DrawTriangle_Fill(100, 100, 50, 200, 150, 150, Red);
```

9.6.3 画带框实心三角形

```
void LT758_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
)
```

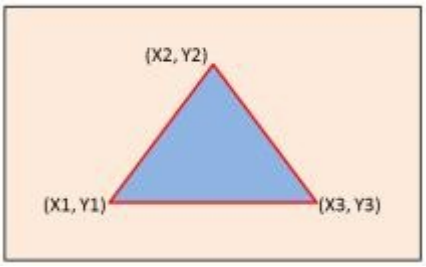



图 9-17：画带框实心三角形

9.7 画曲线

9.7.1 左上方曲线

```
void LT758_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
)
```

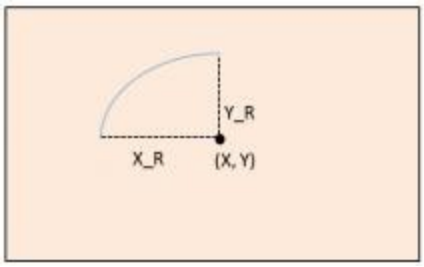


图 9-18：左上方曲线

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：

```
LT758_DrawLeftUpCurve(100, 100, 100, 70, Red);
```


9.7.2 左下方曲线

```
void LT758_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
)
```

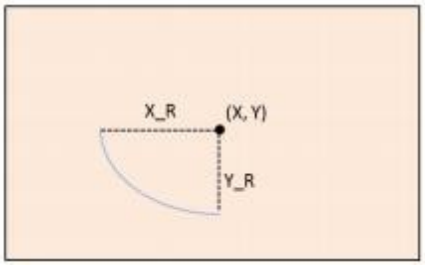


图 9-19：左下方曲线

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：

```
LT758_DrawLeftDownCurve(100, 100, 100, 70, Red);
```

9.7.3 右上方曲线

```
void LT758_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
)
```

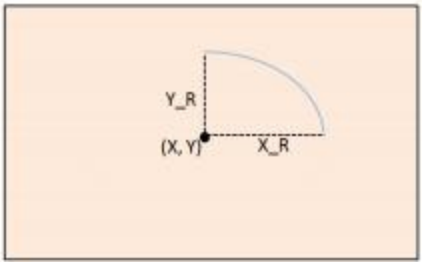


图 9-20：右上方曲线

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：

```
LT758_DrawRightUpCurve(100, 100, 100, 70, Red);
```

9.7.4 右下方曲线

```
void LT758_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
)
```

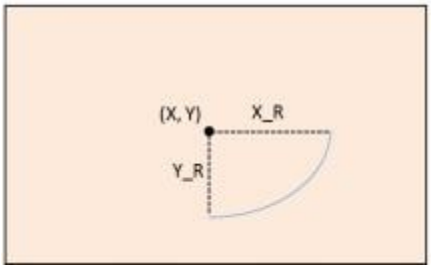


图 9-21：右下方曲线

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：

```
LT758_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 LT758_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
)
```

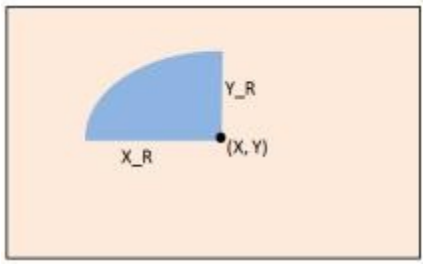



图 9-22：左上方 1/4 椭圆

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：

```
LT758_DrawLeftUpCurve_Fill(100, 100, 100, 70, Red);
```

9.8.2 左下方 1/4 椭圆

```
void LT758_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
)
```

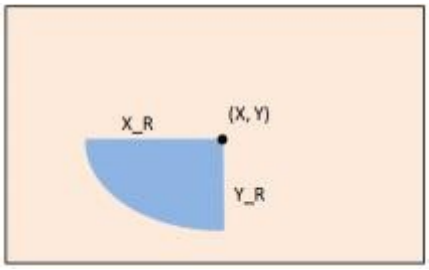


图 9-23：左下方 1/4 椭圆

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：

```
LT758_DrawLeftDownCurve_Fill(100, 100, 100, 70, Red);
```


9.8.3 右上方 1/4 椭圆

```
void LT758_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
)
```

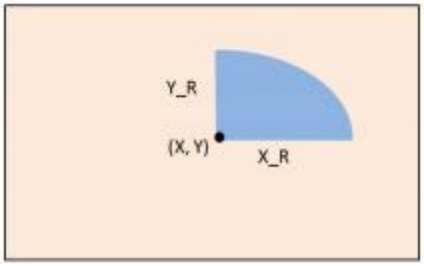


图 9-24：右上方 1/4 椭圆

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：

```
LT758_DrawRightUpCurve_Fill(100, 100, 100, 70, Red);
```

9.8.4 右下方 1/4 椭圆

```
void LT758_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
)
```

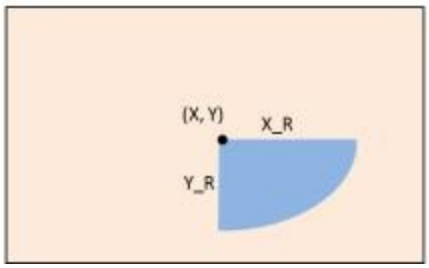


图 9-25：右下方 1/4 椭圆

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：

```
LT758_DrawRightDownCurve_Fill(100, 100, 100, 70, Red);
```

9.9 画四边形

9.9.1 画空心四边形

```
void LT758_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
)
```

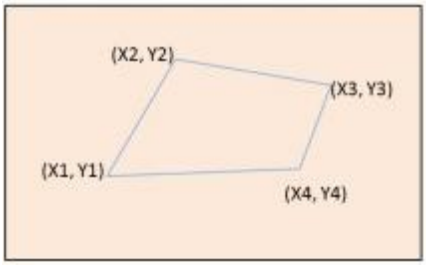


图 9-26：画空心四边形

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：

```
LT758_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 LT758_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
)
```

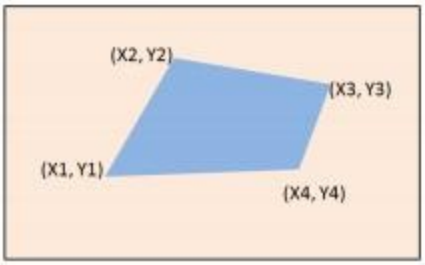


图 9-27：画实心四边形

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：

```
LT758_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 LT758_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
)
```

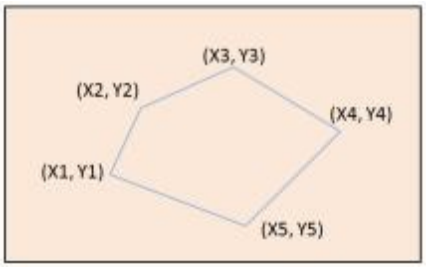


图 9-28：画空心五边形

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 LT758_DrawPentagon(50, 100, 120, 130, 150, 160, 100, 180, 80, 140, Red);
```


9.10.2 画实心五边形

```
void LT758_DrawPentagon_Fill
(
unsigned short X1,                  // X1 位置
unsigned short Y1,                  // X1 位置
unsigned short X2,                  // Y2 位置
unsigned short Y2,                  // X2 位置
unsigned short X3,                  // Y3 位置
unsigned short Y3,                  // X3 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
)
```

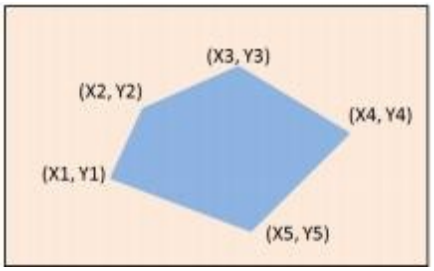


图 9-29：画实心五边形

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 LT758_DrawPentagon_Fill (50, 100, 120, 130, 150, 160, 100, 180, 80, 140, Red);
```


9.11 圆柱体

```
Unsigned char LT758_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
)
```

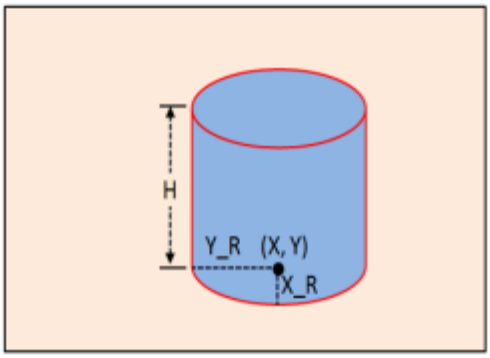


图 9-30：圆柱体

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。

```
LT758_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 T758_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
)
```

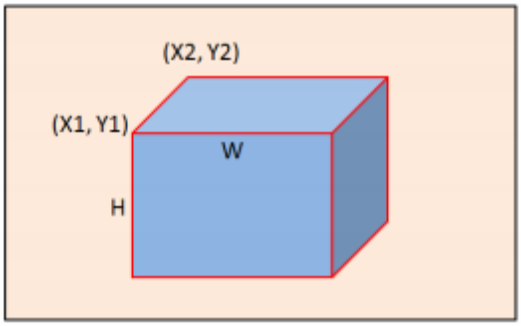


图 9-31：方柱体

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。

```
LT758_DrawQuadrangular(100, 100, 140, 60, 150, 200, Red, Blue);
```


9.13 表格视窗

```
void LT758_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
)
```

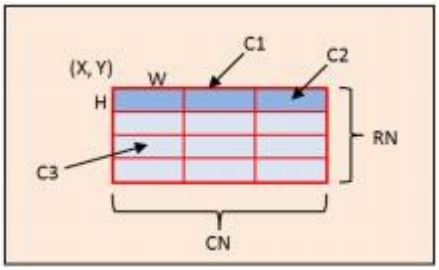


图 9-32：横向表格视窗图

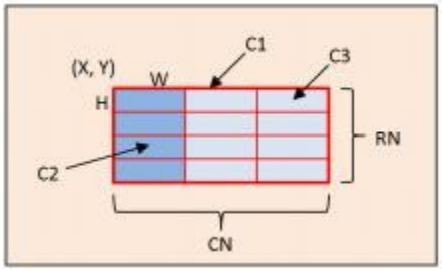


图 9-33：纵向表格视窗

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。

```
LT758_MakeTable(100, 100, 80, 40, 3, 4, Red, Green, Blue, 1, 3, 1);
```


10. 区块传输引擎（BTE）

LT758x 内建 2D 区块传输引擎（BTE），可以增加区块传输的效率。在指定区块数据结合某些逻辑传输操作中，LT758x 内的 BTE 硬件可以提升传输速度，这可以简化 MCU 的程序。因此这个章节主要是讨论 BTE 的功能与操作模式。

10.1 BTE 操作模式

在使用 BTE 功能之前，使用者必须选择指定的 BTE 操作模式。关于操作上的描述，请参考下面说明，而对于不同的应用，LT758x 支持 16 种光栅操作（**Raster Operation，简称 ROP**），来源端与目的端可以健行多样的 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。

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 10-1：BitBLT 的工作模式

BitBLT 工作模式 REG[91h] Bits [3:0]	BitBLT 操 作 说 明
0000b	MCU Write with ROP.
0001b	未使用
0010b	Memory Copy（move）with ROP.
0011b	未使用
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）
1101b	未使用
1110b	Memory Copy with Color Expansion（w/o ROP）
1111b	Memory Copy with Color Expansion and Chroma Keying（w/o ROP）

表 10-2：光栅操作模式（ROP Function）

ROP Bits REG[91h] bit[7:4]	Boolean Function Operation
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 MCU 写入

当 REG[91h] Bits [3:0]=0000b，这是一个 MCU 写入数据到内存中的功能，它可以增加 MCU 写入显示内存的速度，且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，当通过 BTE 引擎做写入数据至目的内存时，会自动处理光栅运算。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。

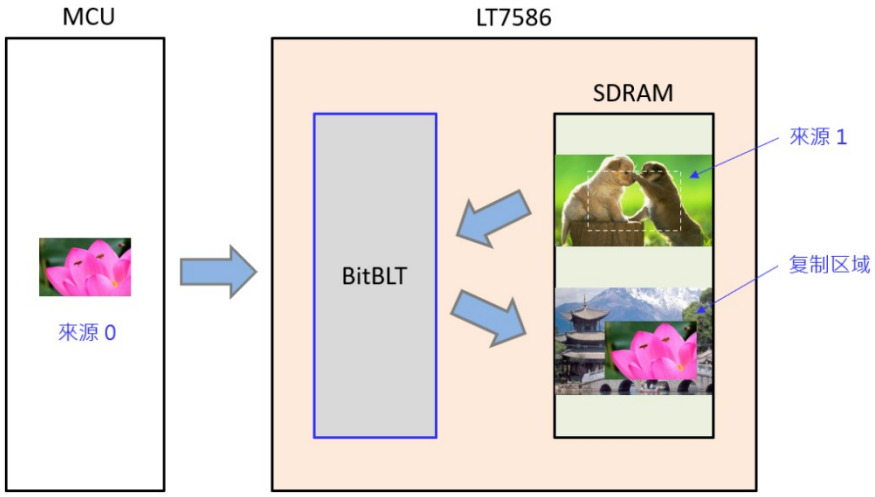


图 10-1：结合光栅操作的 BTE 写入范例

完成这个功能的程序流程图如下：

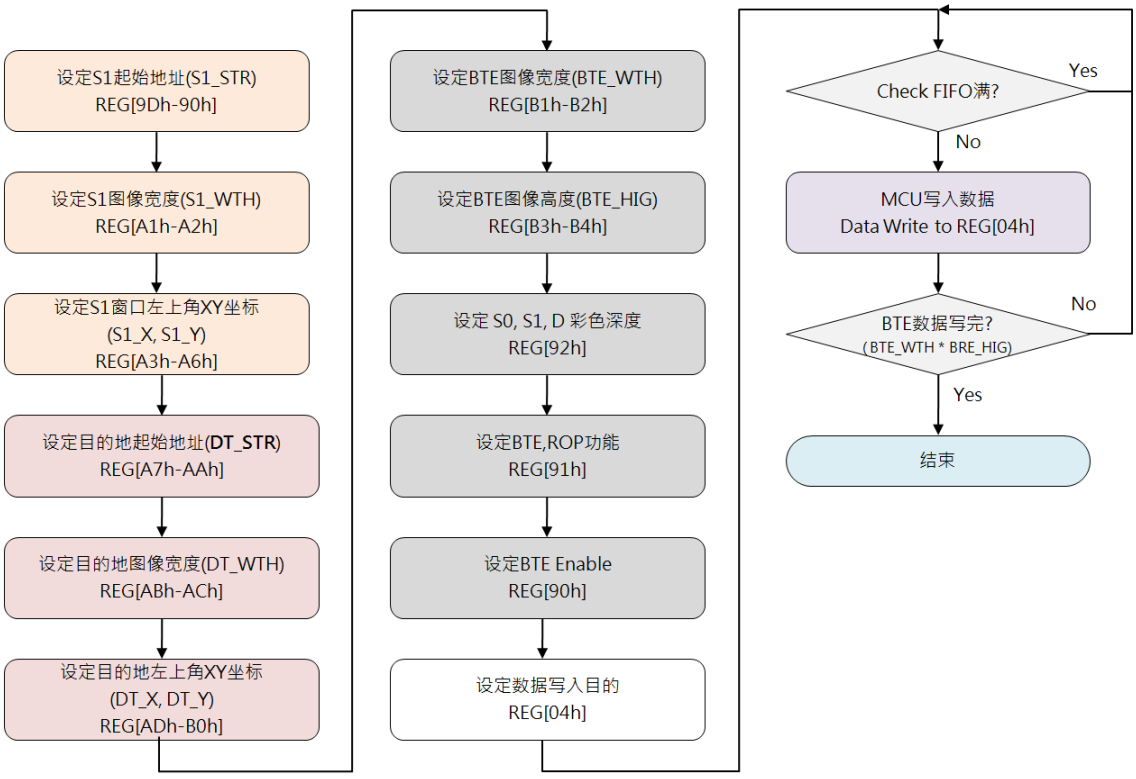


图 10-2：结合光栅操作的 MCU 写入流程图


```
void LT758_BTE_MCU_Write_MCU8
(
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
unsigned char Color_depth           // 色深
)
```

```
void LT758_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 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：

```
LT758_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

当 REG[91h] Bits [3:0]=0010b，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，此功能可以结合16种光栅（ROP）操作。下图范例是当 POP 寄存器（REG[91h]）bit[7:4] 设成 0x0C 得到的结果。

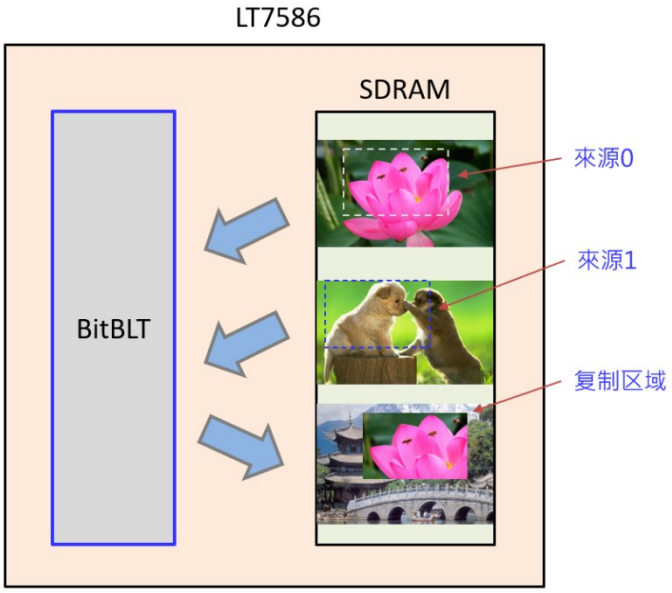


图 10-3：结合光栅操作的 BTE 内存复制范例

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。

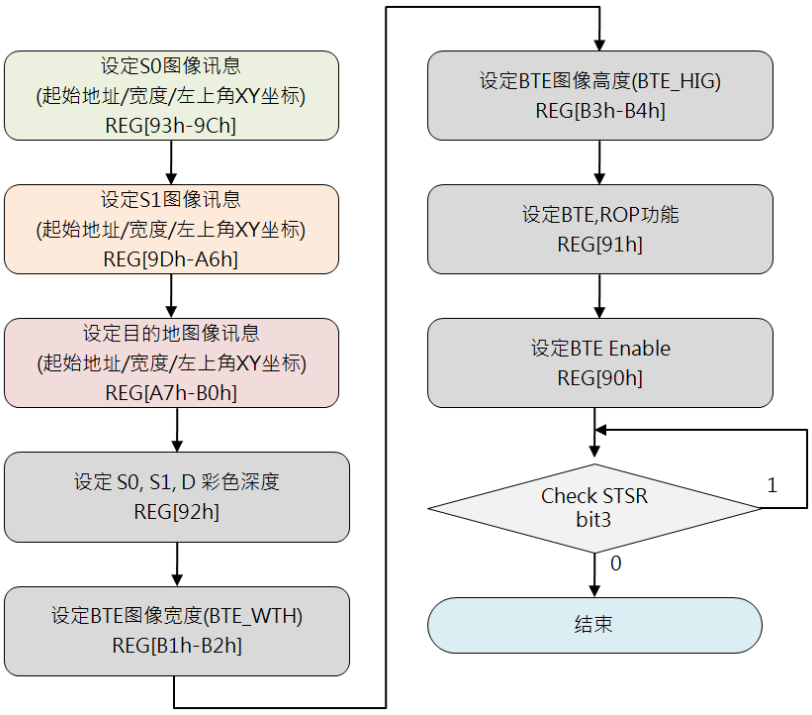


图 10-4：结合光栅操作的 BTE 内存复制流程图（1）

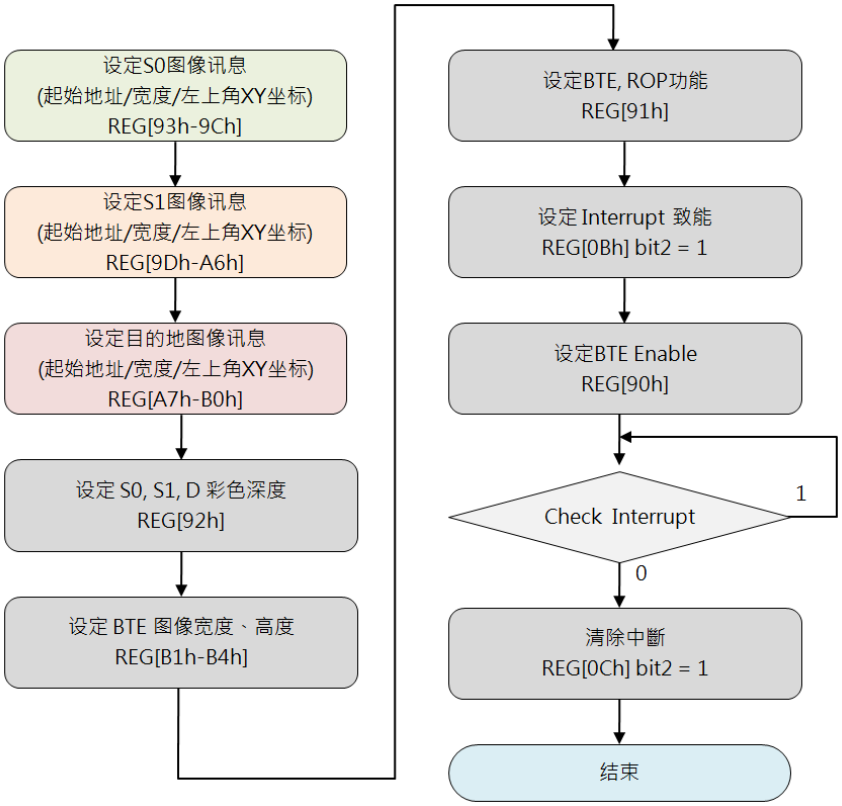


图 10-5：结合光栅操作的 BTE 内存复制流程图（2）


```
void LT758_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
unsigned char Color_depth           // 色位深度
)
```

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)

色位深度：16

实现函数：

```
LT758_BTE_Memory_Copy(0, 1024, 100, 100, 1024*600*2, 1024, 50, 50, 1024*600*2*2,
1024, 200, 200, 0x0A, 100, 100, 16);
```



```
void LT758_BTE_Memory_Copy_JPG
(
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
unsigned char Color_depth           // 色位深度
)
```

```
void LT758_BTE_Memory_Copy_ColorExpansion_8_ColorDepth
(
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 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
unsigned char Color_depth           // 色位深度
)
```


10.2.3 结合 Chroma Key 的 MCU 写入

当 REG[91h] Bits [3:0]=0100b，这也是一个 MCU 写入数据到内存中的功能，与 10.6.1 节的 MCU 写入差异在它写入的数据可以与关键色 Chroma Key 结合，如果 MCU 写入数据与关键色相同，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。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，本功能不支持光栅（ROP）操作。请参考下面图示及程序流程：

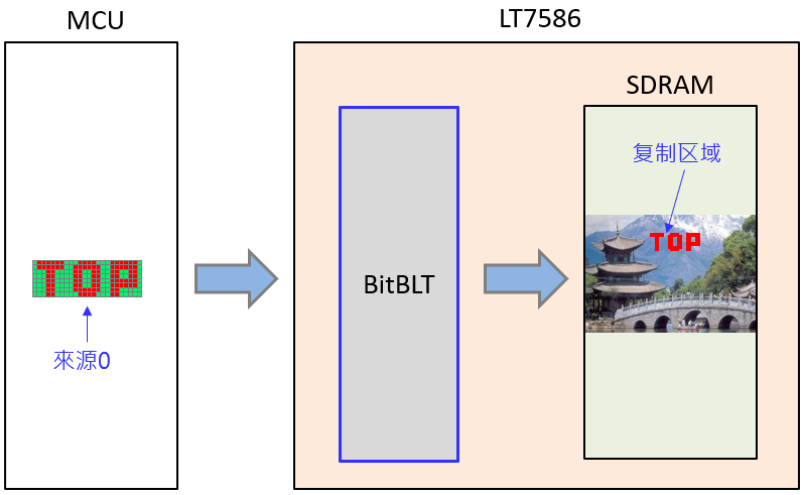


图 10-6：结合 Chroma Key 的 MCU 写入范例

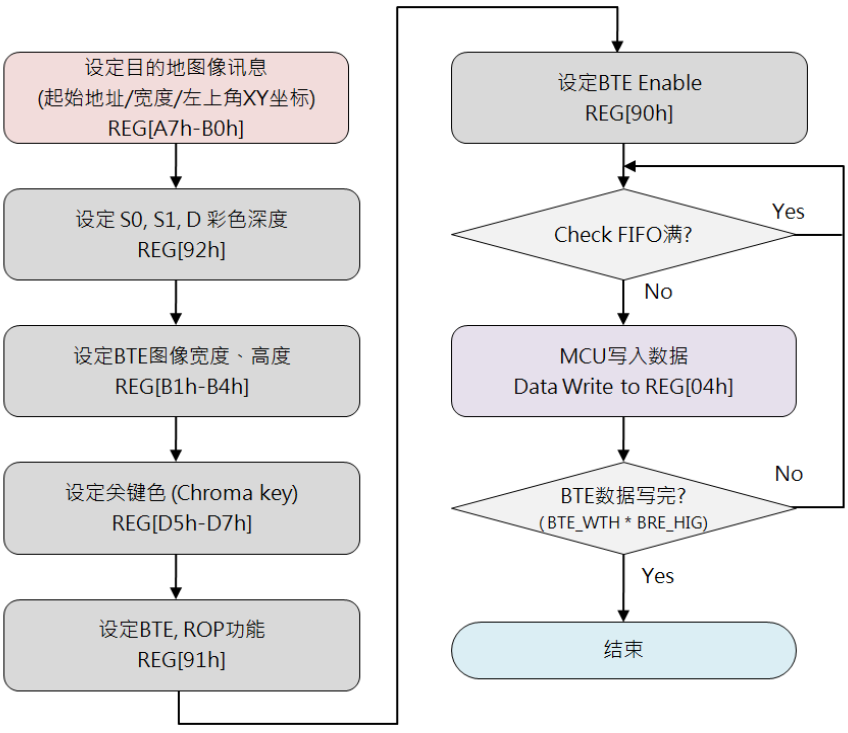


图 10-7：结合 Chroma Key 的 MCU 写入流程图


```
void LT758_BTE_MCU_Write_Chroma_key_MCU8
(
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
unsigned char Color_depth               // 色位深度
)

void LT758_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：

```
LT758_BTE_MCU_Write_Chroma_key_MCU_16bit（0, 1024, 200, 200, Red, 100, 100, &Picture_Data[0]）；
```



```
void LT758_BTE_Memory_Copy_ColorExpansion_Chroma_key_8_ColorDepth
(
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 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 char Color_depth               // 色位深度
)
```

10.2.4 结合 Chroma Key 的内存复制

当 REG[91h] Bits [3:0]=0101b，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，而来源与目的数据是在显示内存上不同的区域，并且在复制搬移的过程中会比较来源端数据与关键色（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。本功能不支持光栅（ROP）操作。请参考下面图示及程序流程：

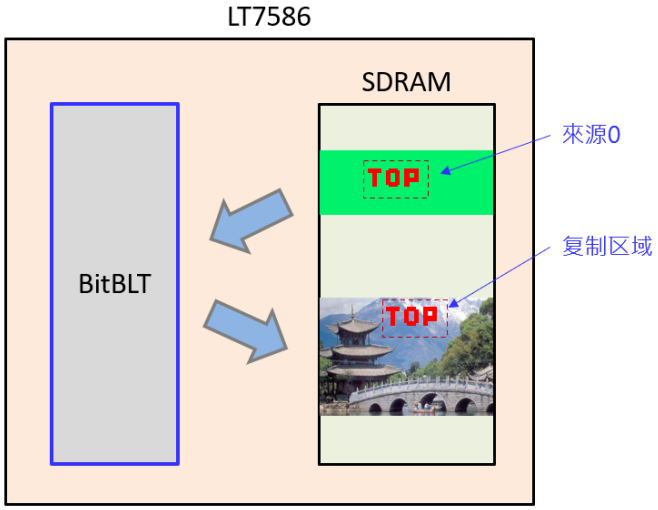


图 10-8：结合 Chroma Key 的内存复制范例

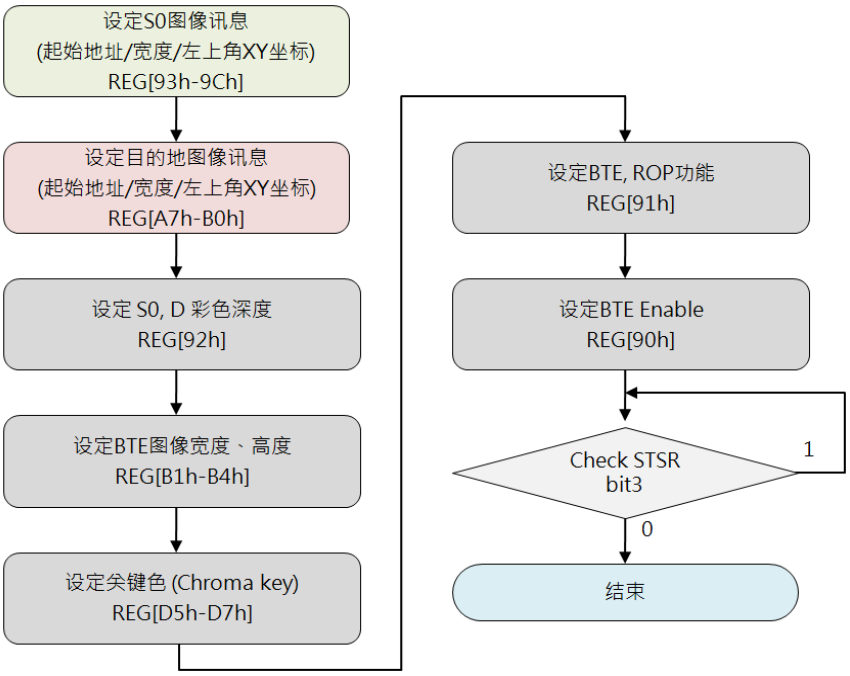


图 10-9：结合 Chroma Key 的内存复制流程图

```
void LT758_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
unsigned char Color_depth           // 色位深度
)
```

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

色位深度：24

实现函数：

LT758_BTE_Memory_Copy_Chroma_key(0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, Red, 100, 100, 24);

10.2.5 结合光栅操作的图样填满

当 REG[91h] Bits [3:0]=0110b，此功能将一 BTE 指定的内存区域重复填满指定的 8*8、16*16 图样（Pattern），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。

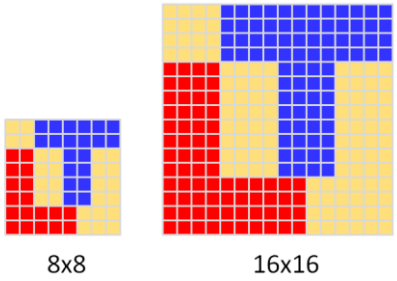


图 10-10：图样格式

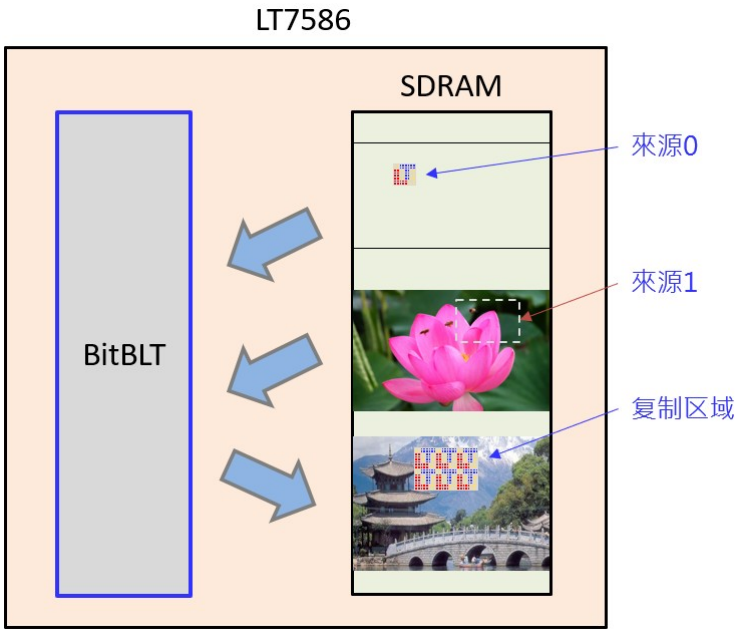


图 10-11：结合光栅操作的图样填满范例

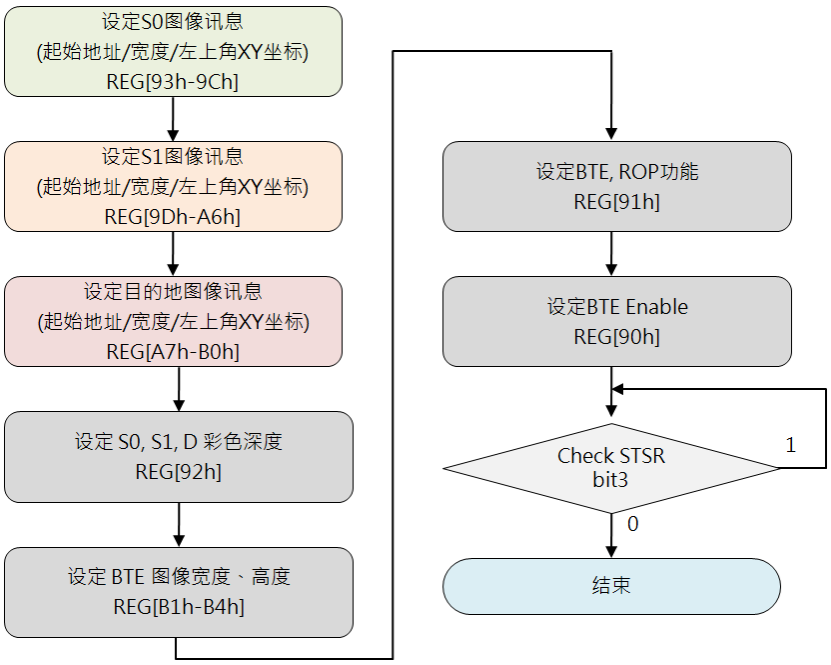


图 10-12：结合光栅操作的图样填满流程图

```
void LT758_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
unsigned char Color_depth          // 色位深度
)
```

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
色位深度：16

实现函数：

```
LT758_BTE_Pattern_Fill(1, 0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, 0x0C, 100, 100, 16);
```

10.2.6 结合 Chroma Key 的图样填满

当 REG[91h] Bits [3:0]=0111b，此功能将一 BTE 指定的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（不会被更新），因此看到的效果将会是透明的。而关键色被设定在 BTE 背景色寄存器 REG[D5h] ~ [D7h] 中，这个功能没有光栅（ROP）操作。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。

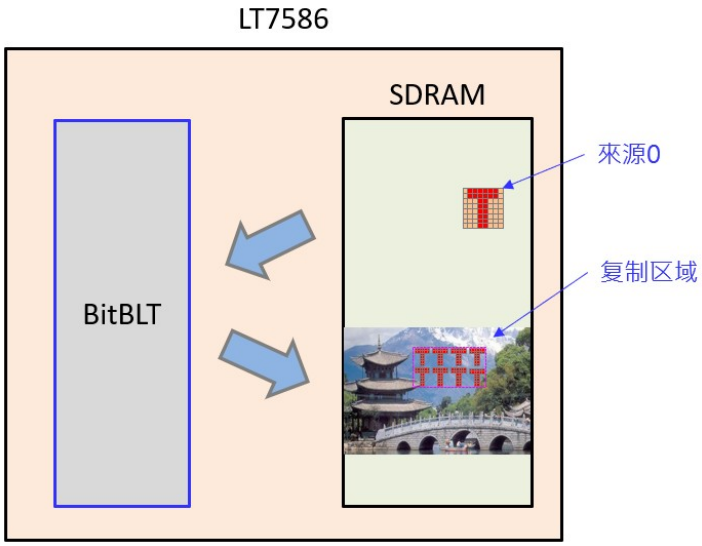


图 10-13：结合 Chroma Key 的图样填满范例

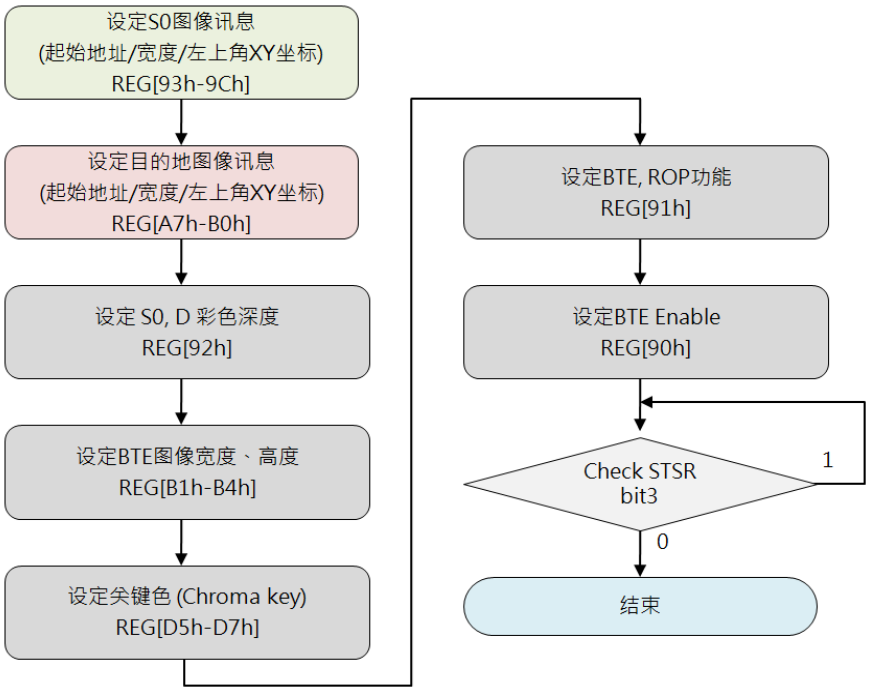


图 10-14：结合 Chroma Key 的图样填满流程图

```
void LT758_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
unsigned char Color_depth            // 色位深度
)
```

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
色位深度：24

实现函数：

LT758_BTE_Pattern_Fill_With_Chroma_key(1, 0, 1024, 200, 200, 1024*600*2, 1024, 100, 100, 0x0C, Red, 100, 100, 24);

10.2.7 结合扩展色彩的 MCU 写入

当 REG[91h] Bits [3:0]=1000b，此功能为 MCU 将单色数据扩展为 8/16/24bpp 彩色数据格式，然后写入到内存中，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。同时要注意的是无论是否使能背景透明功能，前景与背景寄存器（D5h ~ D7h）不可设相同的值。

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。

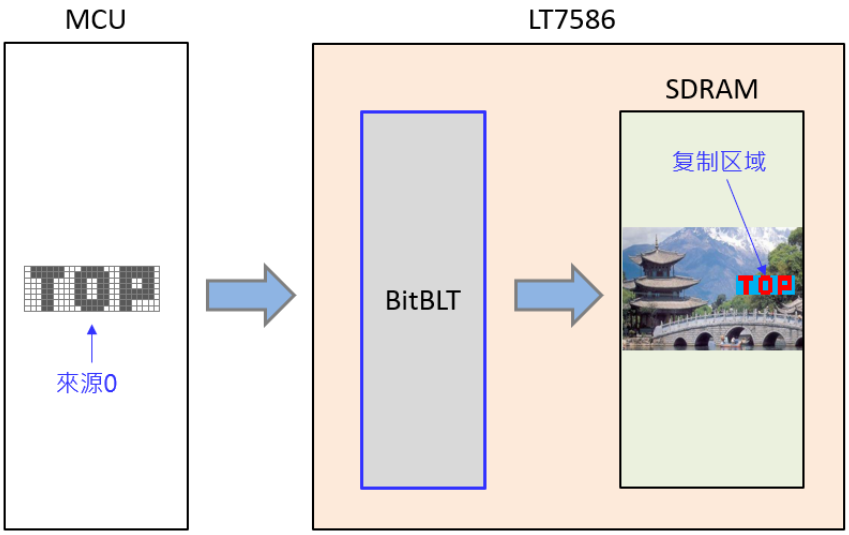


图 10-15：结合扩展色彩的 MCU 写入范例

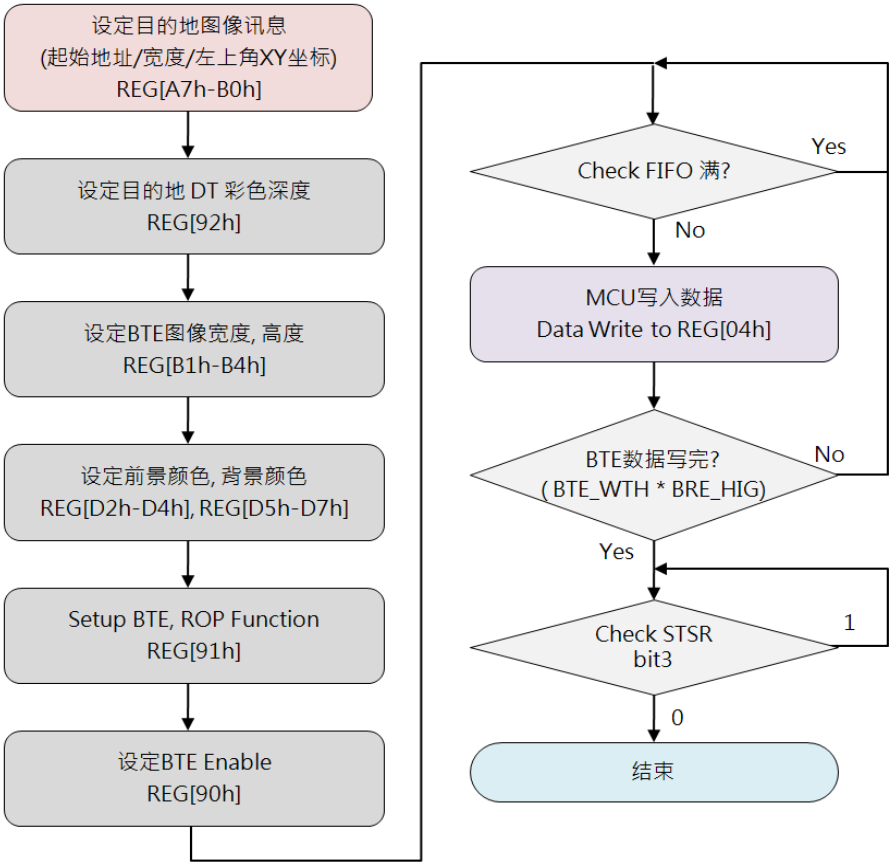


图 10-16：结合扩展色彩的 MCU 写入流程图

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。

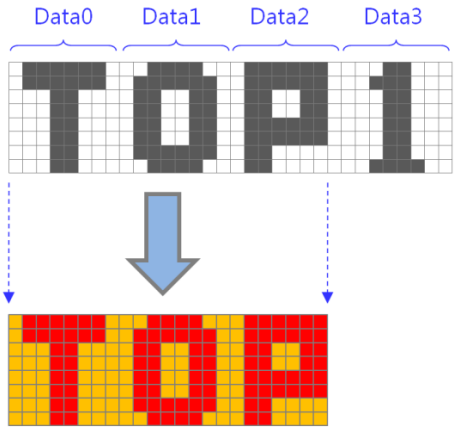


图 10-17：色彩扩展显示范例 1

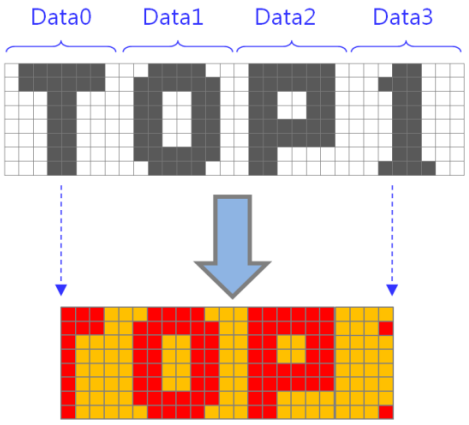


图 10-18：色彩扩展显示范例 2

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

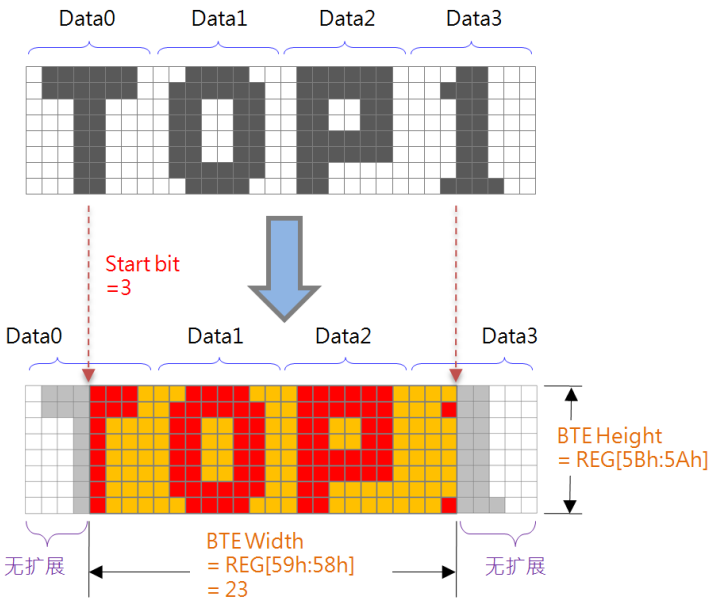


图 10-19：色彩扩展显示数据格式

```
void LT758_BTE_MCU_Write_ColorExpansion_MCU8
(
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,         // 8-bit Data
unsigned char Color_depth           // 色位深度
)

void LT758_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：

```
LT758_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

当 REG[91h] Bits [3:0]=1001b，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。

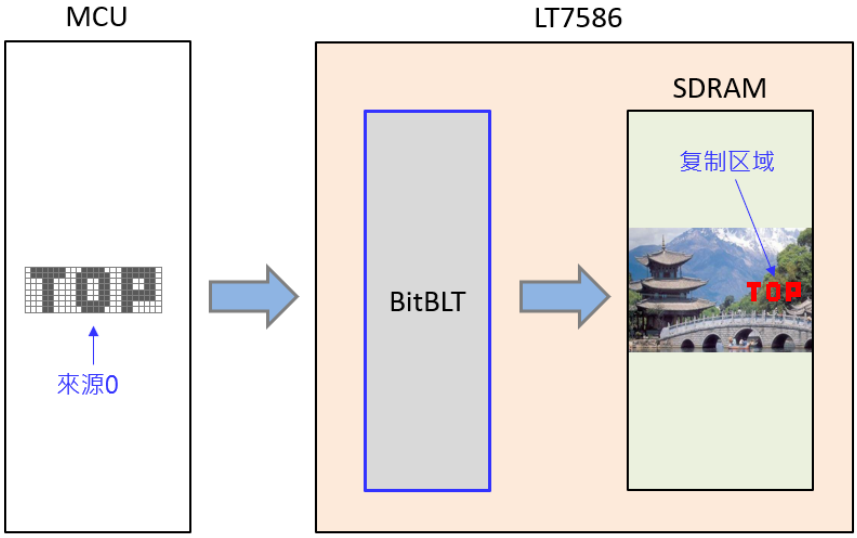


图 10-20：结合扩展色彩与 Chroma key 的 MCU 写入范例

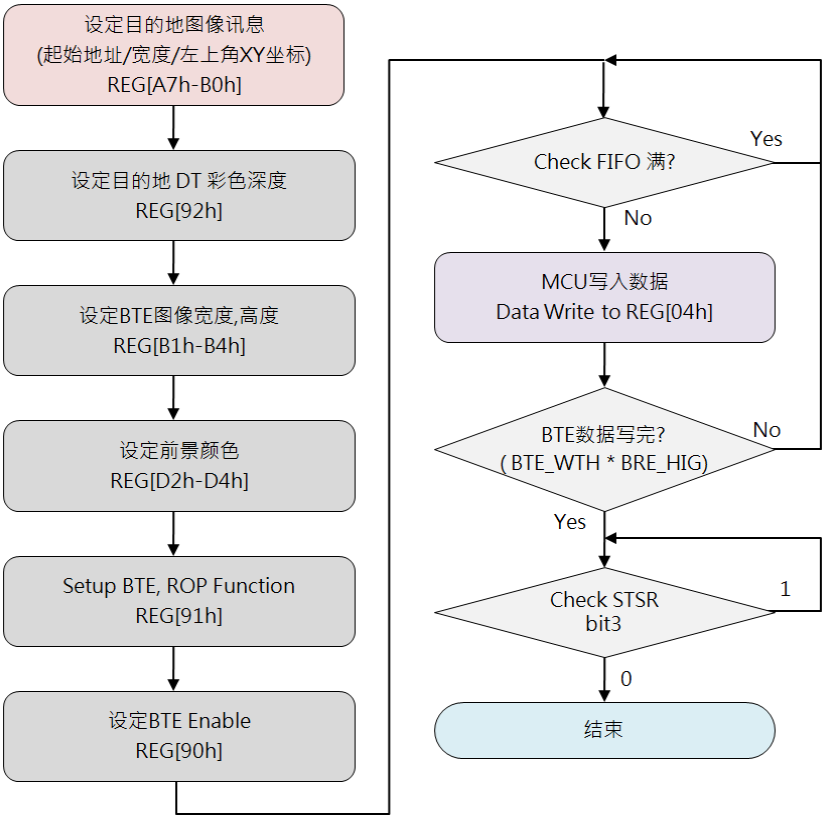


图 10-21：结合扩展色彩与 Chroma key 的 MCU 写入流程图


```
void LT758_BTE_MCU_Write_ColorExpansion_Chroma_key_MCU8
(
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
const unsigned short *data           // 8-bit Data
unsigned char Color_depth            // 色位深度
)
```

```
void LT758_BTE_MCU_Write_ColorExpansion_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 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
const unsigned short *data           // 16-bit Data
)
```


10.2.9 结合透明度的内存复制

当 REG[91h] Bits [3:0]=1010b，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），混合度被定义在寄存器 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。来源 0（S0）、来源 1（S1）及目的（D）都是指在显示内存内。

- **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）]

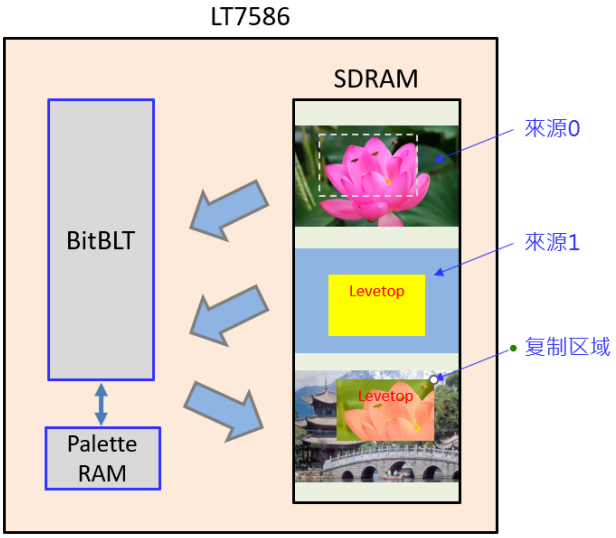


图 10-22：8bpp Pixel Mode 范例

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

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

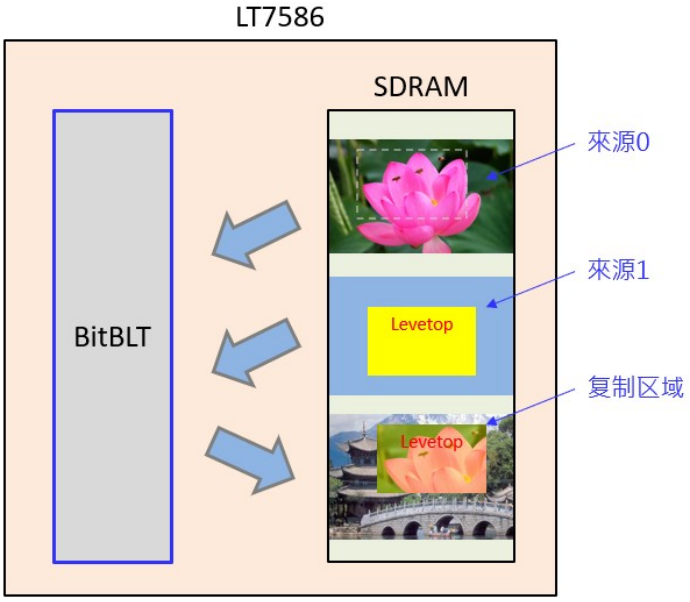


图 10-23：16bpp Pixel Mode 范例

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

表 10-5：Alpha Blending Effect

Bit[15:12]	Alpha Level
00h	0
01h	1/32
02h	2/32
03h	3/32
04h	4/32
05h	5/32
06h	6/32
07h	7/32
08h	8/32
09h	9/32
0Ah	10/32
0Bh	11/32
0Ch	12/32
0Dh	13/32
0Eh	14/32
0Fh	15/32
10h	16/32
11h	17/32
12h	18/32
13h	19/32
14h	20/32
15h	21/32
16h	22/32
17h	23/32
18h	24/32
19h	25/32
1Ah	26/32
1Bh	27/32
1Ch	28/32
1Dh	29/32
1Eh	30/32
1Fh	31/32
2Xh	1

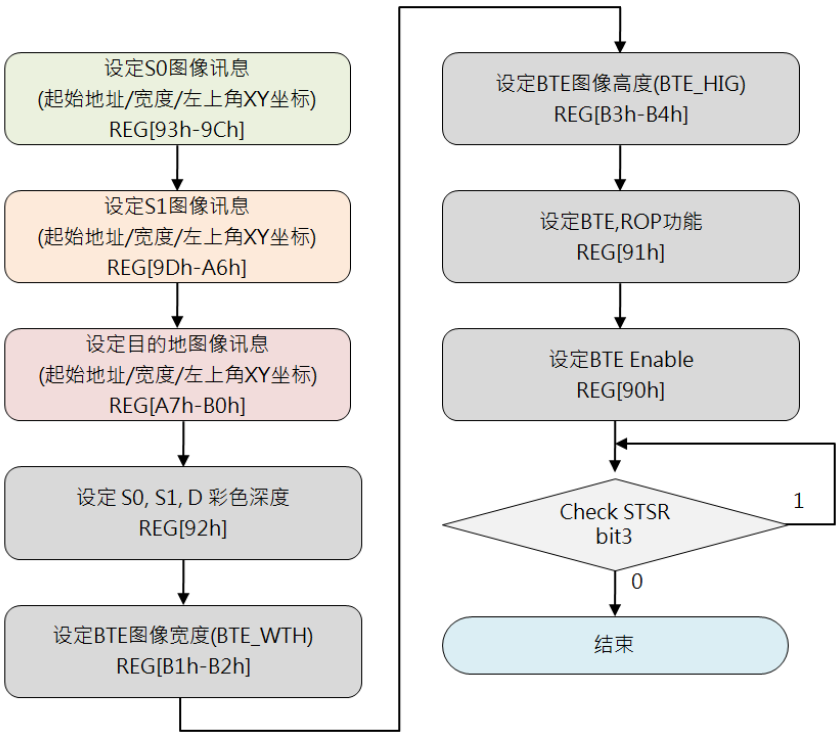


图 10-24：结合透明度的内存复制（Pixel Mode）流程图

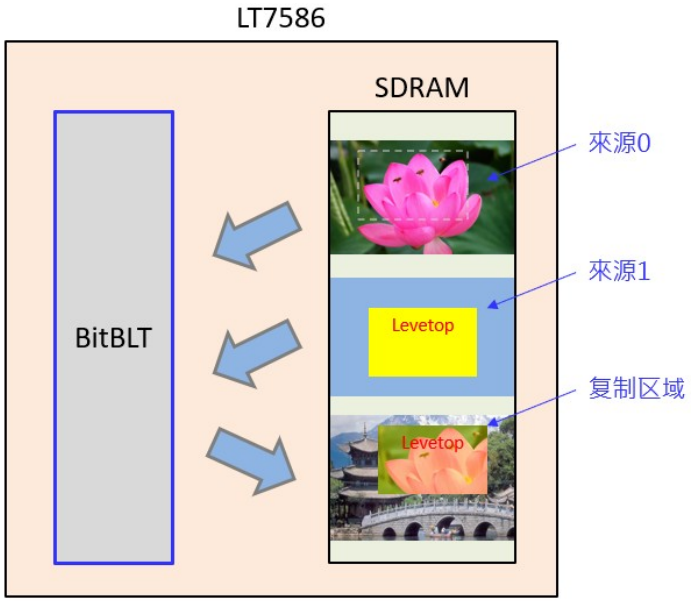


图 10-25：Picture Mode 范例

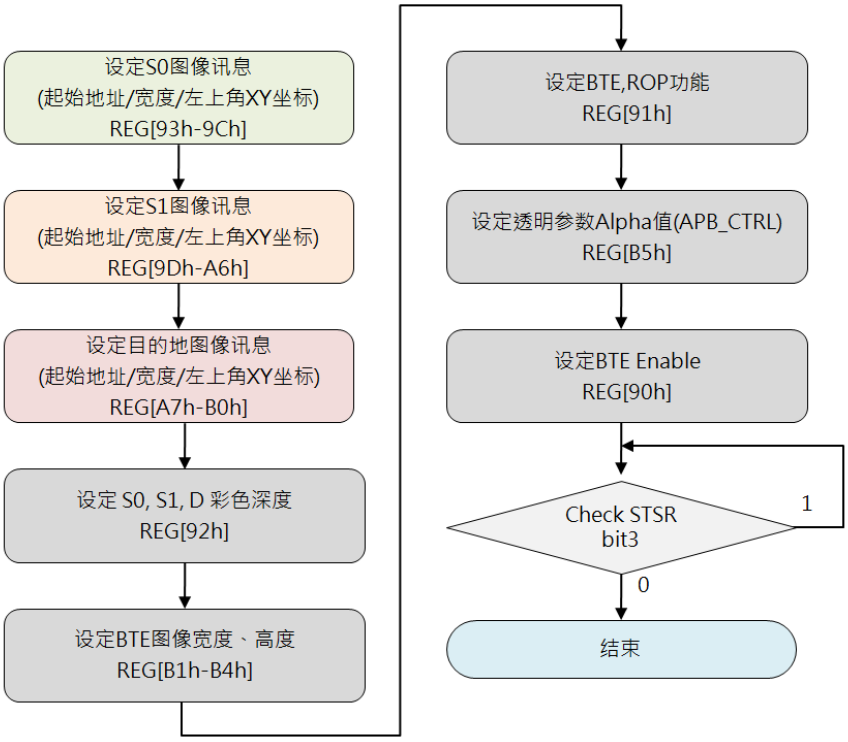


图 10-26：结合透明度的内存复制（Picture Mode）流程图

```
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 结合透明度的 MCU 写入

当 REG[91h] Bits [3:0]=1011b，此功能混合了来源 0 与来源 1 的数据并写入目的内存，而来源 0 的数据是从 MCU 来的，来源 1 数据则由显示内存，写入的目的（D）也是在显示内存。此 Alpha Blending 的功能也具有 Picture 与 Pixel 两种模式。Picture 模式是指说 BTE 处理区域都是具有相同的 Alpha 透明参数值，这个值通过寄存器读取可得到。Pixel 模式是只说 BTE 处理区域内每个像素具有不同的 Alpha 透明参数值，这个透明的参数值纪录在每个像素本身的高位中。可参考上一节的结合透明度的内存复制（Memory Copy with Opacity”）。

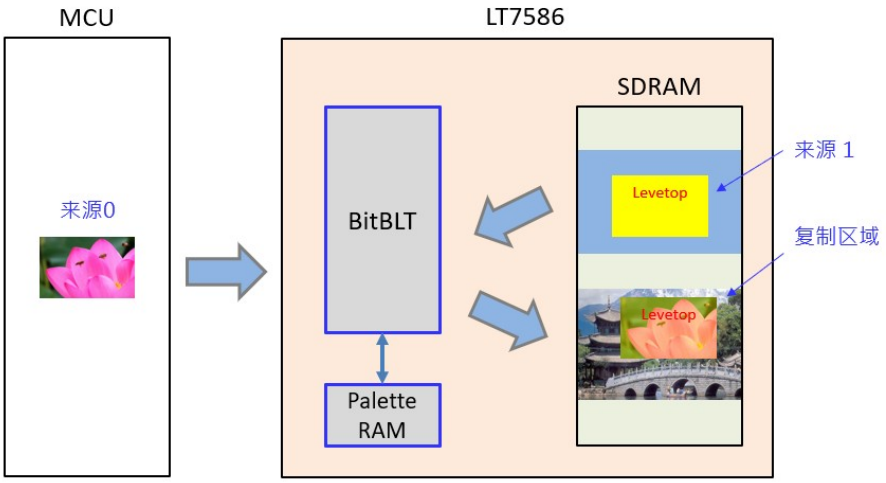


图 10-27：结合透明度的 MCU 写入范例

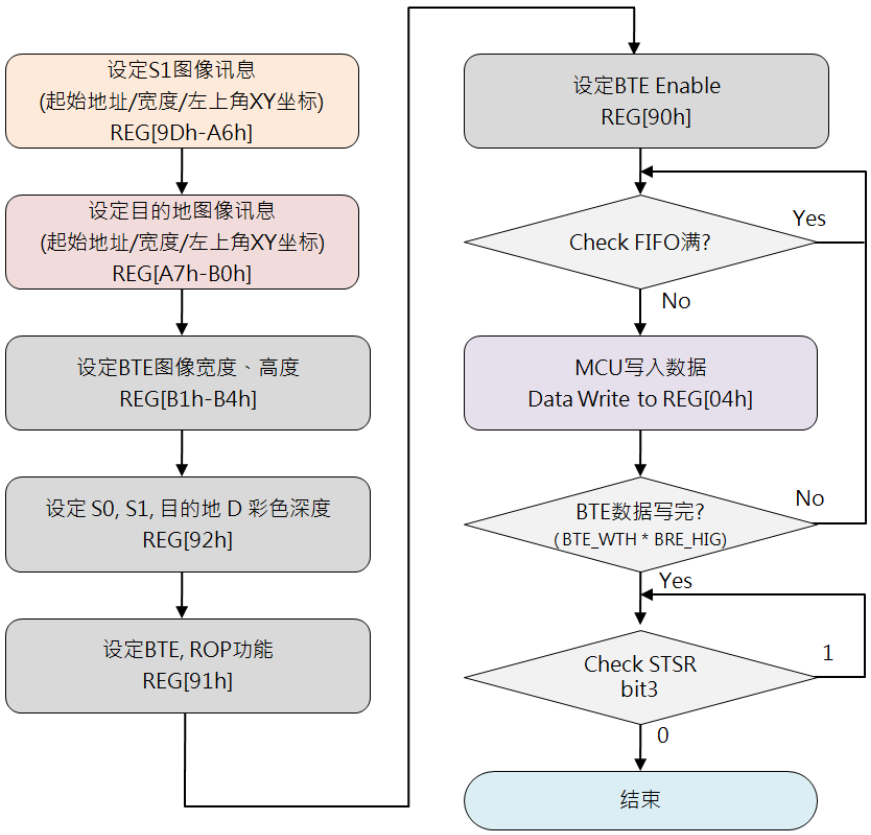


图 10-28：结合透明度的 MCU 写入流程图

```
void LT758_BTE_MCU_Write_Picture_Alpha_Blending
(
unsigned long S1_Addr,              // 来源 S1 的起始地址
unsigned short S1_W,                // 来源 S1 的宽度
unsigned short XS1,                 // 来源 S1 的左上角 X 坐标
unsigned short YS1,                 // 来源 S1 的左上角 Y 坐标
unsigned long Des_Addr,             // 目标图片的起始地址
unsigned short Des_W,               // 目标图片的宽度
unsigned short XDes,                // 目标图片左上角的 X 坐标
unsigned short YDes,                // 目标图片的左上角 Y 坐标
unsigned short X_W,                 // 活动串口宽度
unsigned short Y_H,                 // 活动窗口高度
unsigned char *data,                // 来源 S0 的图片数据的起始地址
unsigned char alpha,                // 透明度等级
unsigned char Color_depth           // 颜色深度
)
```


举例：（假设 LCD 屏为 1024*600 的分辨率）

来源 1：内存 0 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 地址起的（50, 50）坐标

活动窗口的宽：100，高：100

S0：MCU 写入图片数据的起始地址 RWBuff（unsigned char RWBuff[8192];）

透明度：16

颜色深度：24bit

实现函数：

LT758_BTE_MCU_Write_Picture_Alpha_Blending(0, 800, 50, 50, 1024*600*2 ,800, 50, 50, 100, 100, RWBuff, 16, 24);

10.2.11 结合扩展色彩的内存复制

当 REG[91h] Bits [3:0]=1110b，此功能会将从显示内存读取的来源 0（S0）单色影像数据（bit-map）转成 8/16/24bpp 彩色影像数据，并且写入显示内存目的内存中。如果单色数据 bit 为“1”那么将会转换成前景色寄存器设定的颜色。如果单色数据 bit 为“0”，那么将会转换成背景色寄存器设定的颜色。如果背景透明被使能，那么当来源数据是“0”时，目的内存数据不会有任何更改。单色数据宽度则是由 REG[92h] 来定义，来源 0 单色数据宽度可以定义为 8bit/16bit。如果单色数据宽度定义为 8bit，那么 ROP（Start Bit）可设定值可由 bit7 ~ bit0 来当起始位；如果单色数据宽度定义为 16bit，那么 ROP（Start Bit）可设定值可由 bit15 ~ bit0 来当起始位。同样要留意的是无论是否使能背景透明功能，前景与背景寄存器（D5h ~ D7h）不可设相同的值。

例如下图，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蓝色，所得到的色彩扩展结果。

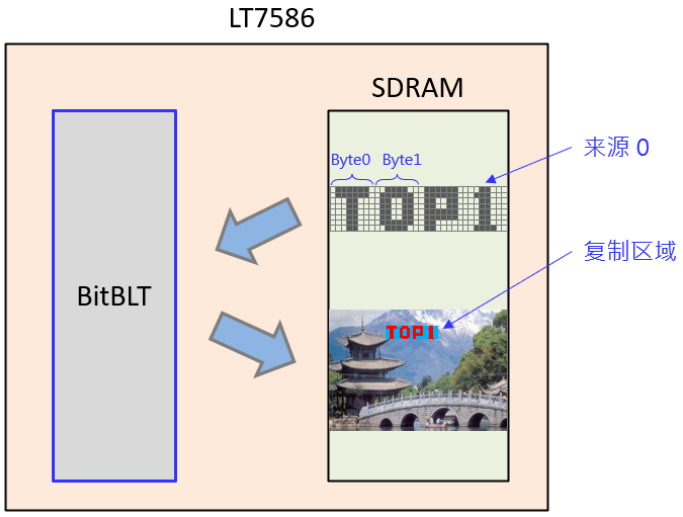


图 10-29：结合扩展色彩的内存复制范例

例如下图，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，BTE Width = 23 时，所得到的色彩扩展结果。

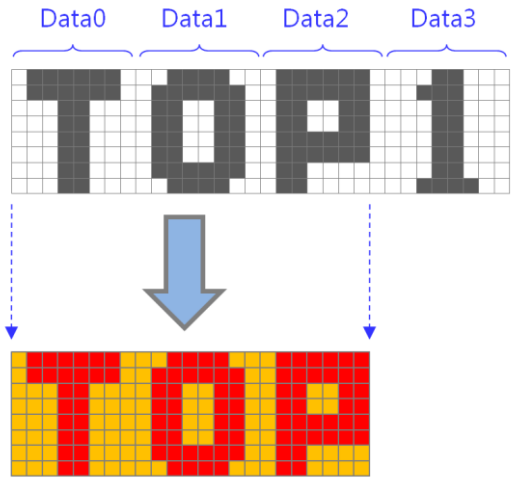


图 10-30：色彩扩展显示范例 1

下图范例则为 ROP = 4，其他设定不变时所得到的色彩扩展结果。

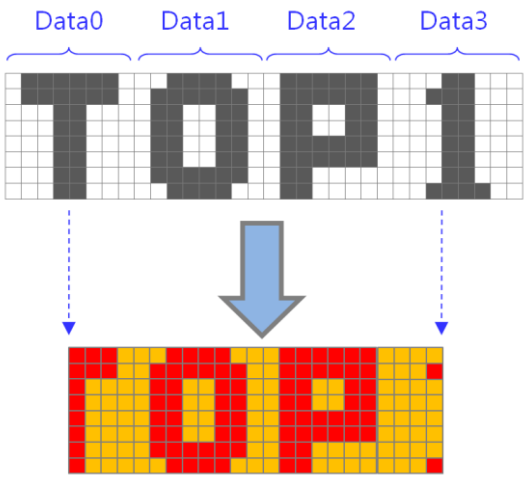


图 10-31：色彩扩展显示范例 2

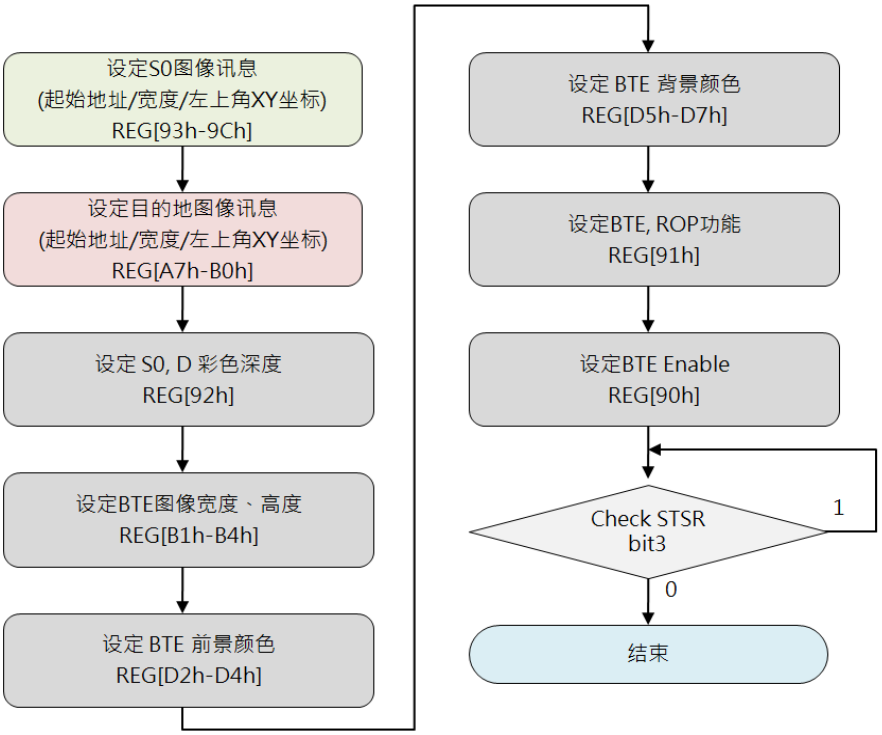


图 10-32：结合扩展色彩的内存复制流程图

```
void LT758_BTE_Memory_Copy_ColorExpansion_8_ColorDepth
(
unsigned long S0_Addr,                  // 来源 S0 图片在 SDRAM 的起始地址
unsigned short S0_W,                    // 来源 S0 图片的宽度
unsigned short XS0,                     // 来源 S0 图片的左上角 x 坐标
unsigned short YS0,                     // 来源 S0 图片的左上角 y 坐标
unsigned long Des_Addr,                 // 目标图片在 SDRAM 的起始地址
unsigned short Des_W,                   // 目标图片的宽度
unsigned short XDes,                    // 目标图片的左上角 x 坐标
unsigned short YDes,                    // 目标图片的左上角 y 坐标
unsigned short X_W,                     // 活动窗口的宽度
unsigned short Y_H,                     // 活动窗口的高度
unsigned long Foreground_color,         // 前景颜色
unsigned long Background_color,         // 背景颜色
unsigned char Color_depth               // 颜色深度
)
```


举例：（假设 LCD 屏为 1024*600 分辨率）

来源 0：内存 1024*600*3 地址起的（100, 100）坐标

目的图像的内存地址及位置：内存 0 地址起（200, 200）坐标

工作视窗：宽度：100；高度：100

前景颜色：红色，背景颜色：黄色，颜色深度：16bit

实现函数：

LT758_BTE_Memory_Copy_ColorExpansion_8_ColorDepth(1024*600*3 ,800, 100, 100, 0, 800, 200, 200, 100, 100, 0xF800, 0xFFE0, 16);

10.2.12 结合扩展色彩与 Chroma Key 的内存复制

当 REG[91h] Bits [3:0]=1111b，此功能会将从显示内存读取的来源 0（S0）单色影像数据（bit-map）转成彩色影像数据，并且写入显示内存目的内存中。如果单色数据 bit 为“1”，那么将会转换成前景色寄存器设定的颜色。如果单色数据 bit 为“0”，那么将不会对目的内存做任何的更动，以达成透明的效果。

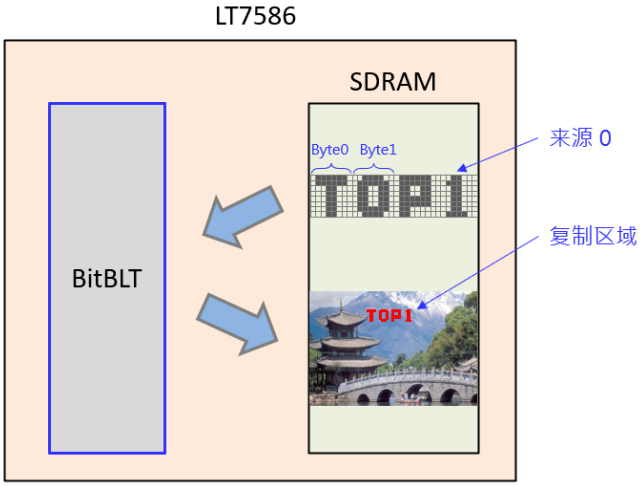


图 10-33：结合扩展色彩与 Chroma Key 的内存复制范例

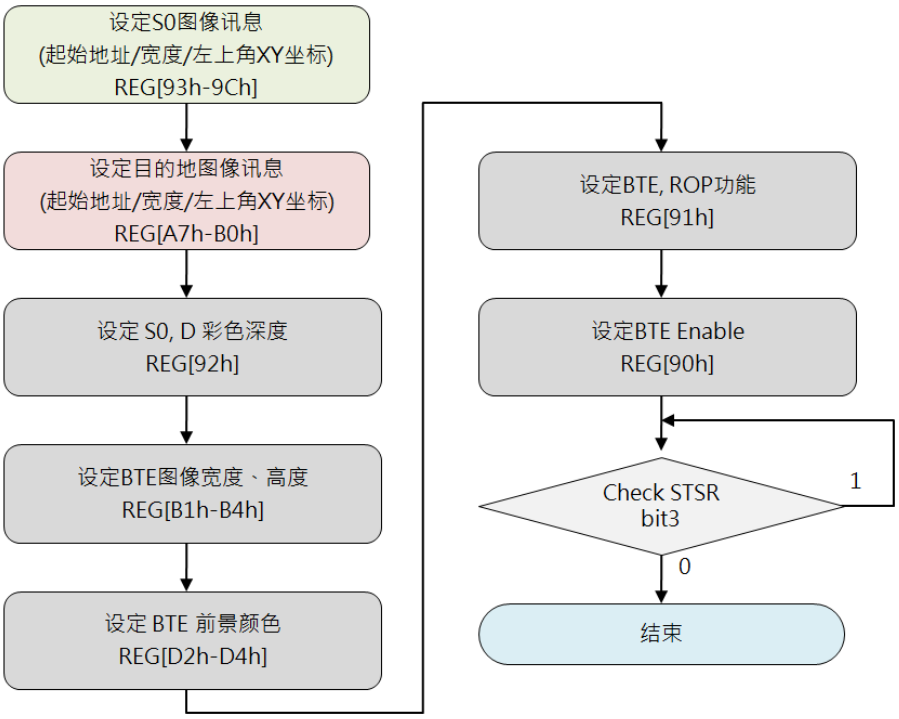


图 10-34：结合扩展色彩与 Chroma Key 的内存复制流程图

```
void LT758_BTE_Memory_Copy_ColorExpansion_Chroma_key_8_ColorDepth
(
unsigned long S0_Addr,                  // 来源 S0 图片在 SDRAM 的起始地址
unsigned short S0_W,                    // 来源 S0 图片的宽度
unsigned short XS0,                     // 来源 S0 图片的左上角 x 坐标
unsigned short YS0,                     // 来源 S0 图片的左上角 y 坐标
unsigned long Des_Addr,                 // 目标图片在 SDRAM 的起始地址
unsigned short Des_W,                   // 目标图片的宽度
unsigned short XDes,                    // 目标图片的左上角 x 坐标
unsigned short YDes,                    // 目标图片的左上角 y 坐标
unsigned short X_W,                     // 活动窗口的宽度
unsigned short Y_H,                     // 活动窗口的高度
unsigned long Foreground_color,         // 前景颜色
unsigned char Color_depth               // 颜色深度
)
```

举例： (假设 LCD 屏为 1024*600 分辨率)

来源 0：内存 1024*600*3 地址起的（100, 100）坐标

目的图像的内存地址及位置：内存 0 地址起（200, 200）坐标

工作视窗：宽度：100；高度：100

前景颜色：红色，颜色深度：16bit

实现函数：

LT758_BTE_Memory_Copy_ColorExpansion_Chroma_key_8_ColorDepth(1024*600*3, 800, 100, 100, 0, 800, 200, 200, 100, 100, 0xF800, 16);

10.2.13 区域填满（Solid Fill）

当 REG[91h] Bits [3:0]=1100b，此功能会针对 BTE 指定的目的内存的一个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。

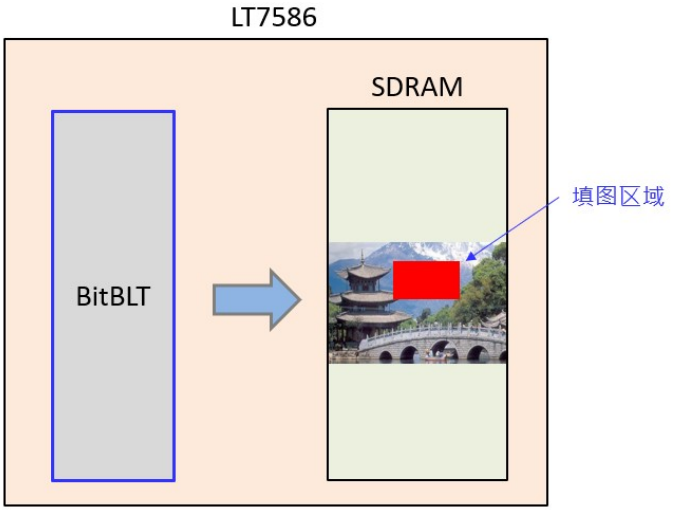


图 10-35：区域填满范例

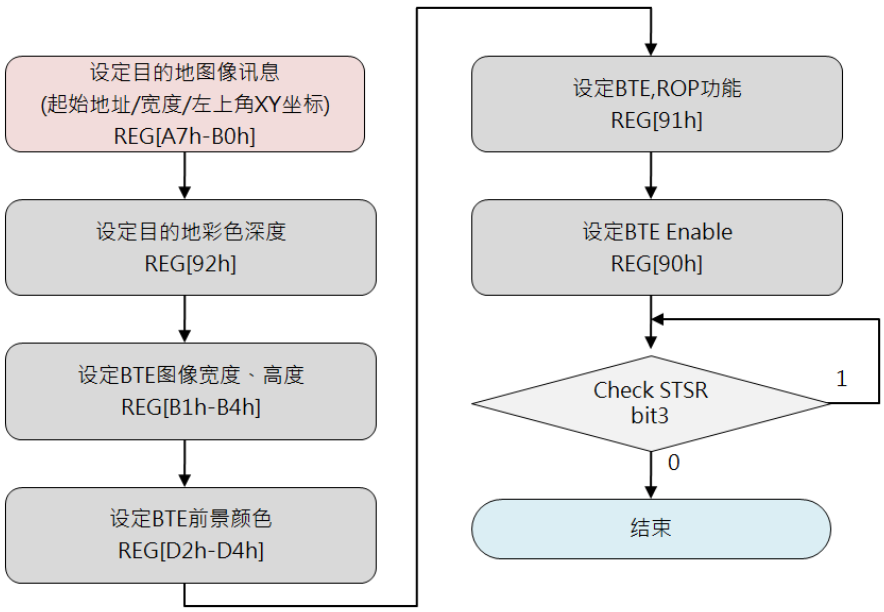


图 10-36：区域填满流程图


```
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
unsigned char Color_depth      // 色位深度
)
```

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

色位深度：16

实现函数：

```
BTE_Solid_Fill(0, 1024, 200, 200, Red, 100, 100, 16);
```


11. 显示文字

LT758x 有两种文字图形来源：

- 内建 ASCII 字型
- 使用者定义字型（CGRAM）

LT758x 除了内建 4 组 ASCII 字型外，也允许 MCU 写入数据到字型寄存器进行造字功能，与字型相关的寄存器是（REG[CCh] ~ REG[DEh]），而文字颜色可以在前景色与背景色寄存器（REG[D2h] ~ REG[D7h]）中被设定。

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 LT758_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 LT758_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）
unsigned char *c                     // 显示字符
)
```

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 LT758”，且不放大字体的高度和宽度、背景透明，字体也对齐：

```
LT758_Select_Internal_Font_Init(32, 1, 1, 0, 1);
LT758_Print_Internal_Font_String(10, 100, Red, 0, “LeveTop LT758”);
```

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 LT758 对 Bin 文件进行调用，请参考 11.3 节。LT758 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 LT758_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,        // 0：可以设置字体的背景色；1：字体背景色透明
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 LT758_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 ASCII 的编码规则下， 则需要调用以下的函数：

```
void LT758_Print_Outside_Ascii_String
(
unsigned short x,                  // 字体开始显示的 x 位置
unsigned shorty,                   // 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 Flash 0 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“深圳市乐升半导体有限公司”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：

```
LT758_Select_Outside_Font_Init(0, 0, 0x00, 0x003E537E0, 0x0009B520, 24, 1, 1, 1, 1);
LT758_Print_Outside_Font_String(10, 100, Red, 0, “深圳市乐升半导体有限公司”);
```

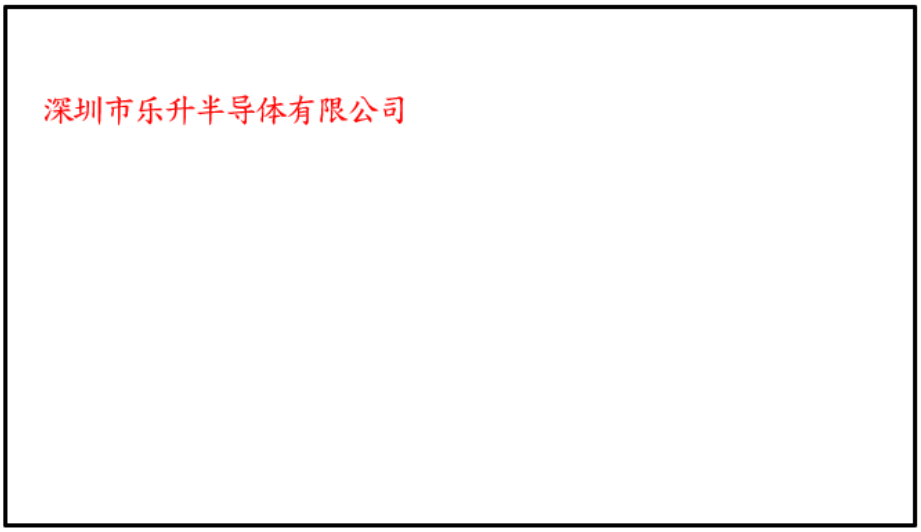


图 11-1：显示 24*24 楷书中文字

提示：

1、该函数，只是对 GB2312 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 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 LT758_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)*
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 LT758_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“深圳市乐升半导体有限公司”字串，且不加粗、背景色透明、行距为 0：

GB2312 编码规则：

LT758_Print_Outside_Font_GB2312_48_72(1, 0, 0x005DC000, 0x003E537E0, 48, 0, 10, 100, color256_red, color256_white, 1, 0 , “深圳市乐升半导体有限公司”);

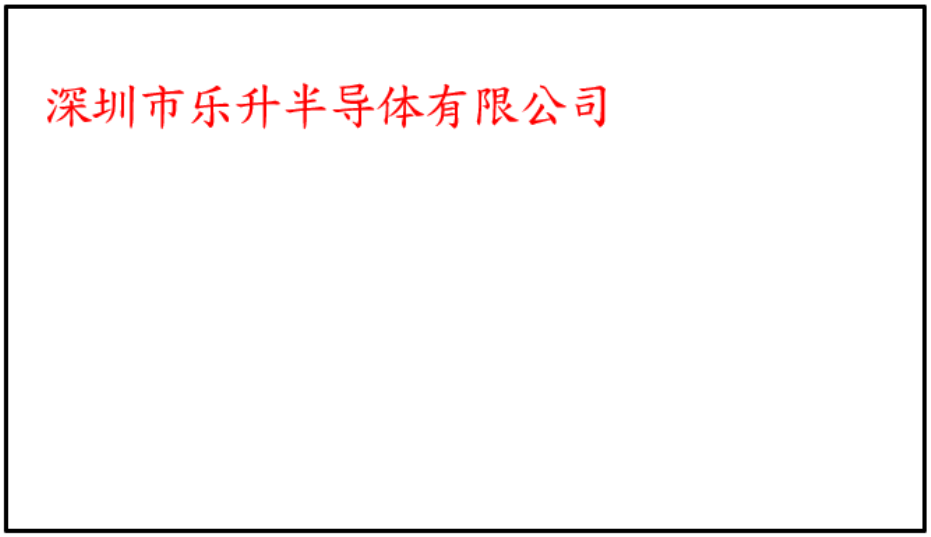


图 11-2：显示 48*48 楷书中文字

BIG5 编码规则：

LT758_Print_Outside_Font_BIG5_48_72(1, 0, 0x005DC000, 0x003E537E0, 48, 0, 10, 100, color256_red, color256_white, 1, 0，“深圳市乐升半导体有限公司”);

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。

11.3 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整合软件 UartTFT_Vxxx.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 LT758x 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，使用者可以参考以下的说明，举例产生一个 16*16 的宋体字库 Bin 文件：

1. 点击【UartTFT 菜单栏>BWFont】打开中文字库 Bin 文件制作界面。在界面右侧如图单击即可查看可供选择的字库列表，并选择字库：

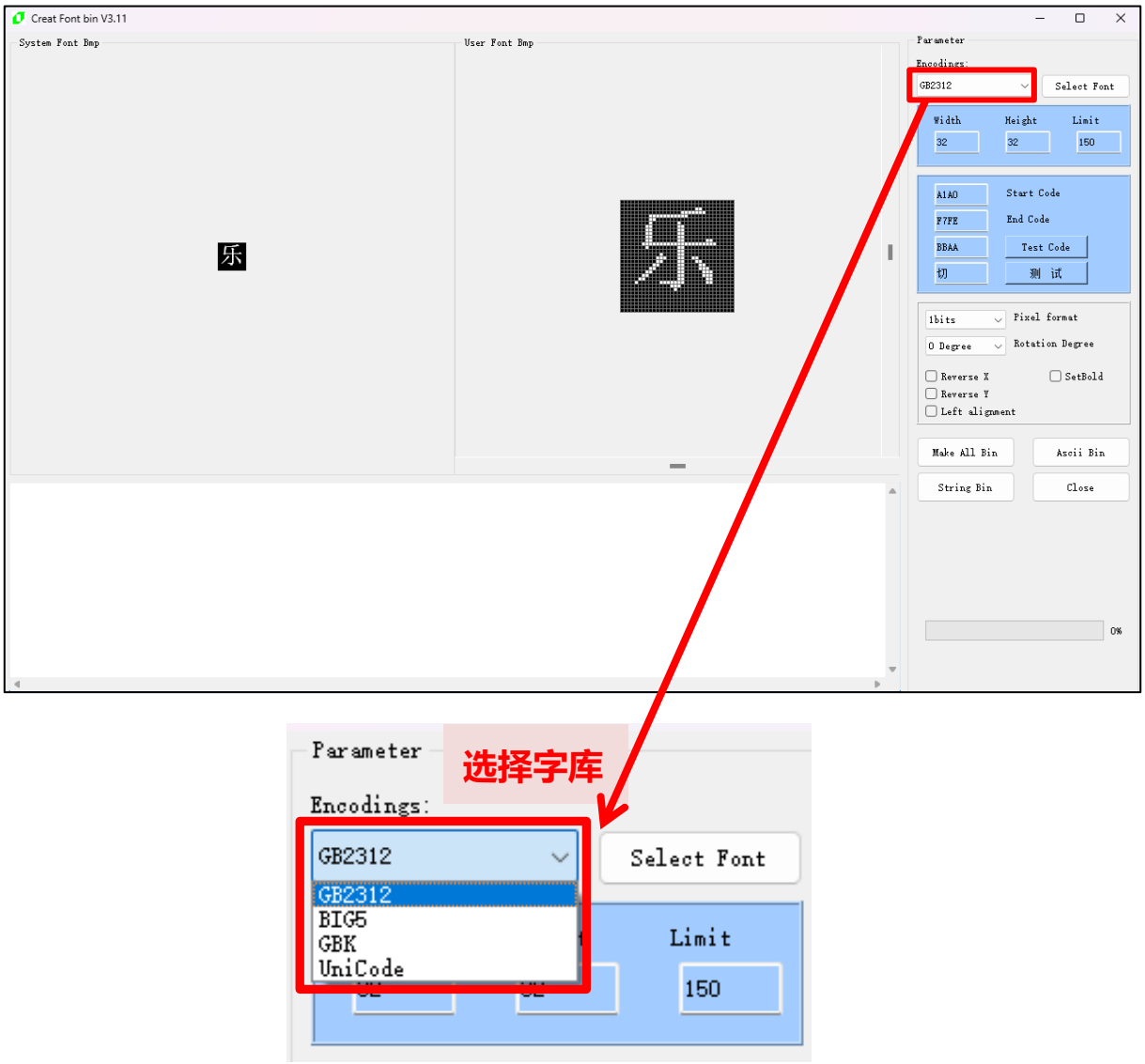


图 11-3：制作

提示：专用整合软件 UartTFT_V4.30.exe（或 V4.30 以上的版本）取得方式请参考第 17.2 节。

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

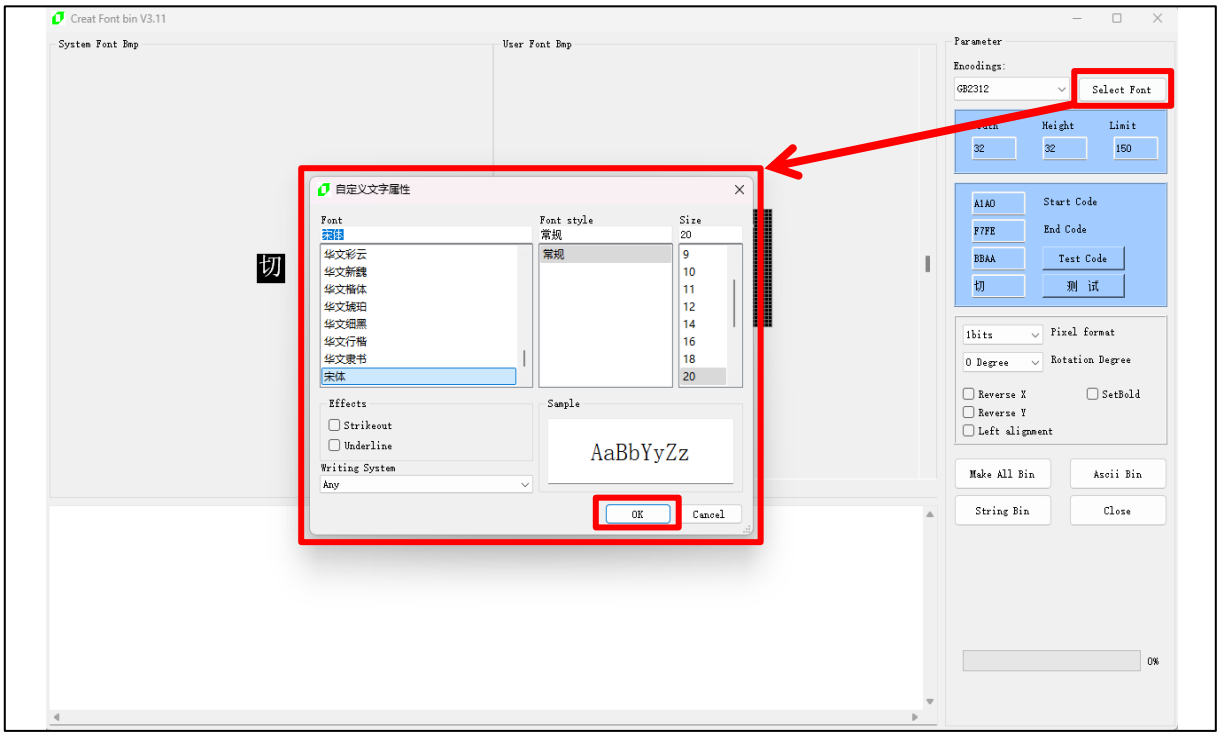


图 11-4：选择字体

3. 点击 Width、Height 和 Limit 下框并修改框内的参数，就可以调整字体大小。点击 Start Code、End Code 左框并修改框内的参数，就可以设置开始字码和结束字码：

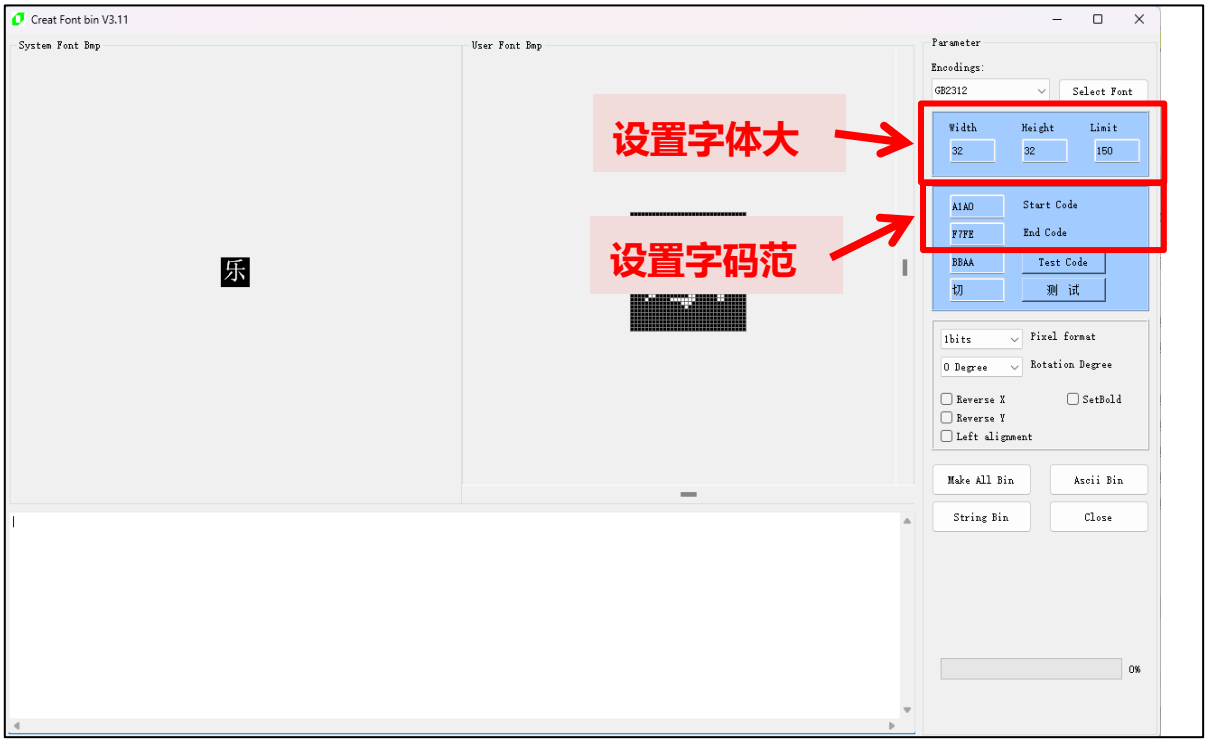


图 11-5：设置字库-1

4. 该工具提供的像素格式有 1bits、2bits、4bits、8bits、aRGB4444 这 5 种，可以通过单击 Pixel format 左侧的框查看并设置。字体可以被设置的旋转角度有 0 Degree、R 90 Degree、L 90 Degree 这 3 种，可以通过单击 Rotation Degree 左侧的框查看并设置。你还可以通过勾选信息来设置字体参数，如“Reverse X”即沿 X 轴翻转，“ Reverse Y”即沿 Y 轴翻转，“ Left alignment”即左对齐，“ SetBold”即设置为粗体：

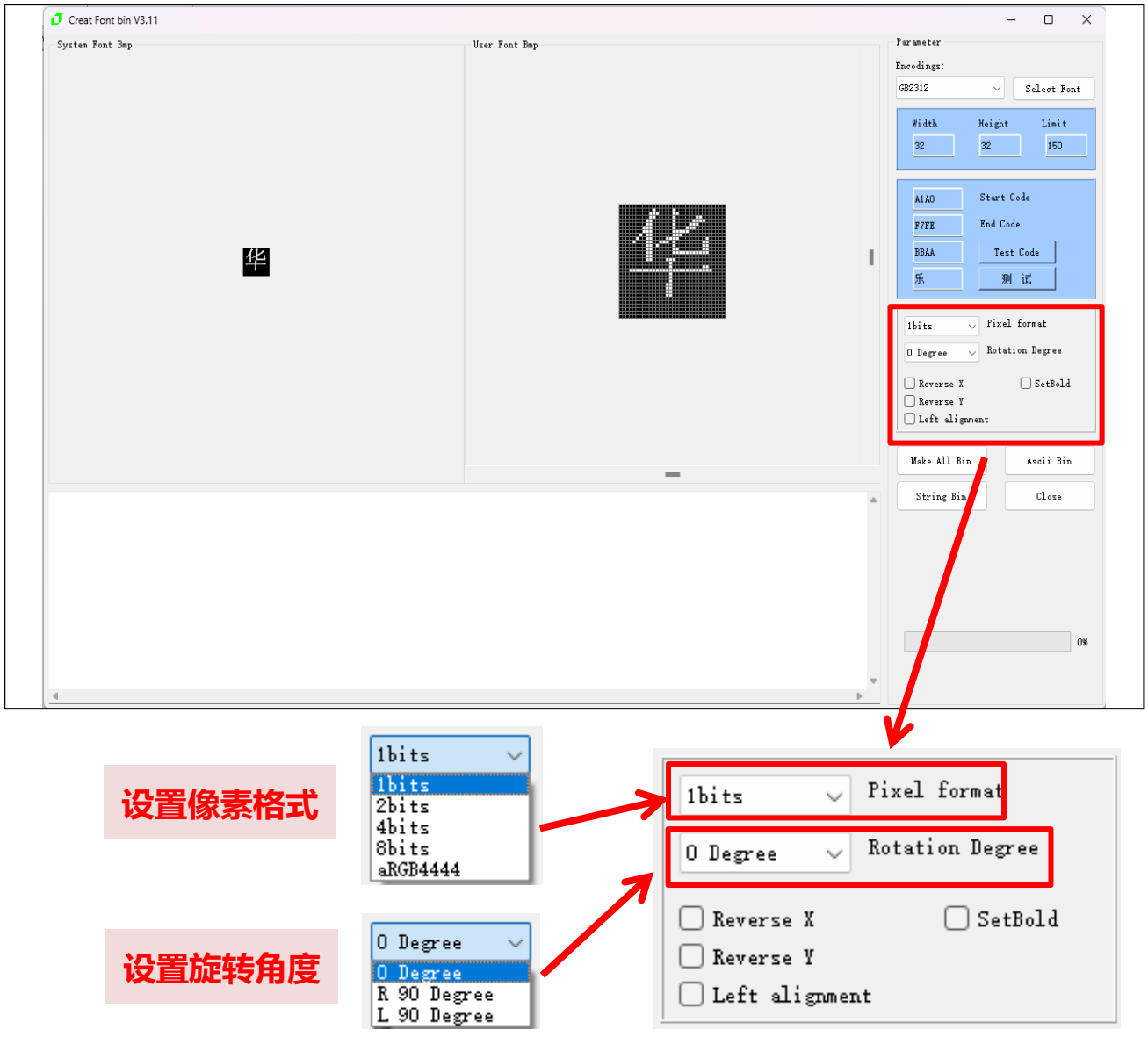
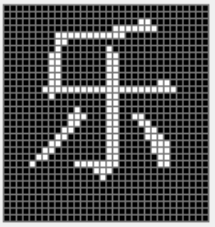
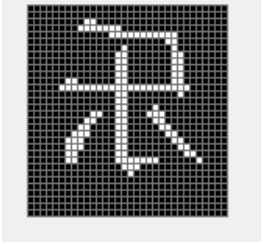


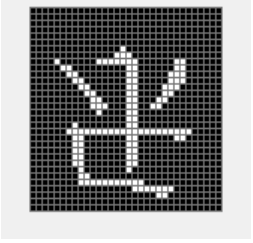
图 11-6：设置字库-2



无效果

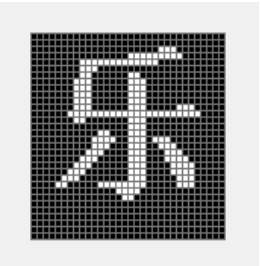


“Reverse X”效果



“Reverse Y”效果”

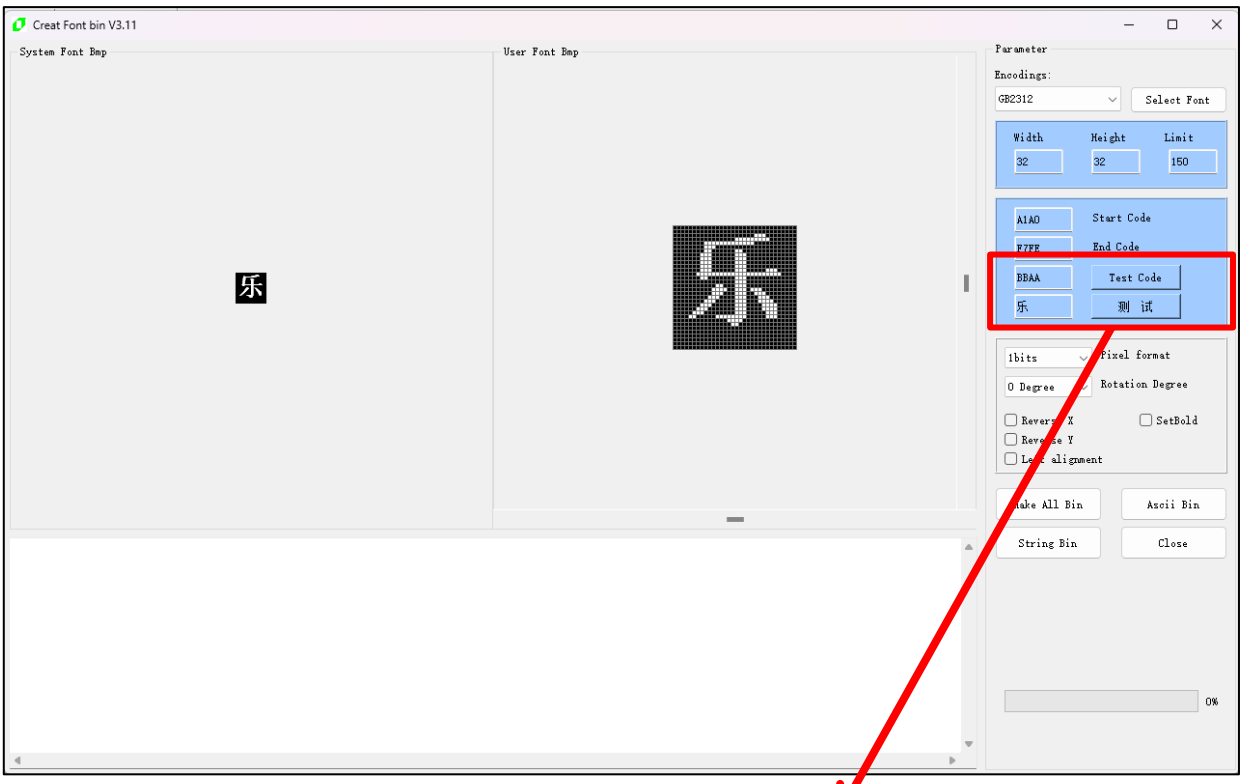
“Left alignment”效果



“SetBold”效果

图 11-7：参数设置字体效果

5. 我们提供了两种方式预览文字，一是通过设置字码来预览，再点击【Test Code】按钮显示预览文字；二是通过设置文字来预览，再点击【测试】按钮显示预览文字：



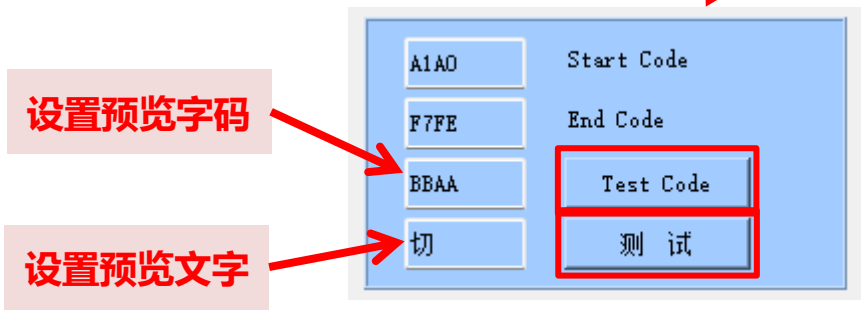


图 11-8：预览文字

6. 点击【Make All Bin】即可输出字库 Bin 文件，或是点击【Ascii Bin】、【String Bin】分别输出字库 Bin 文件。注意输入文件名时文件名中不能包含如下字符，如：？* / \ < > : " |，否则无法保存。

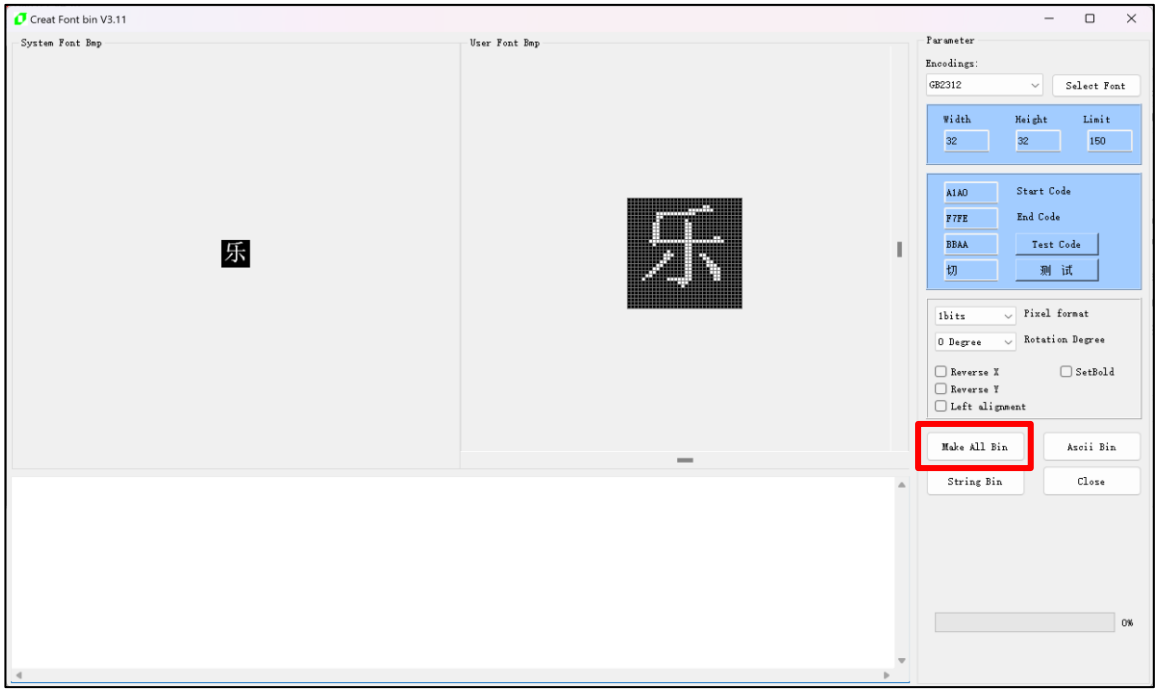


图 11-9：保存字库

当显示“(字库)+Font Lib ok”时，即保存成功：

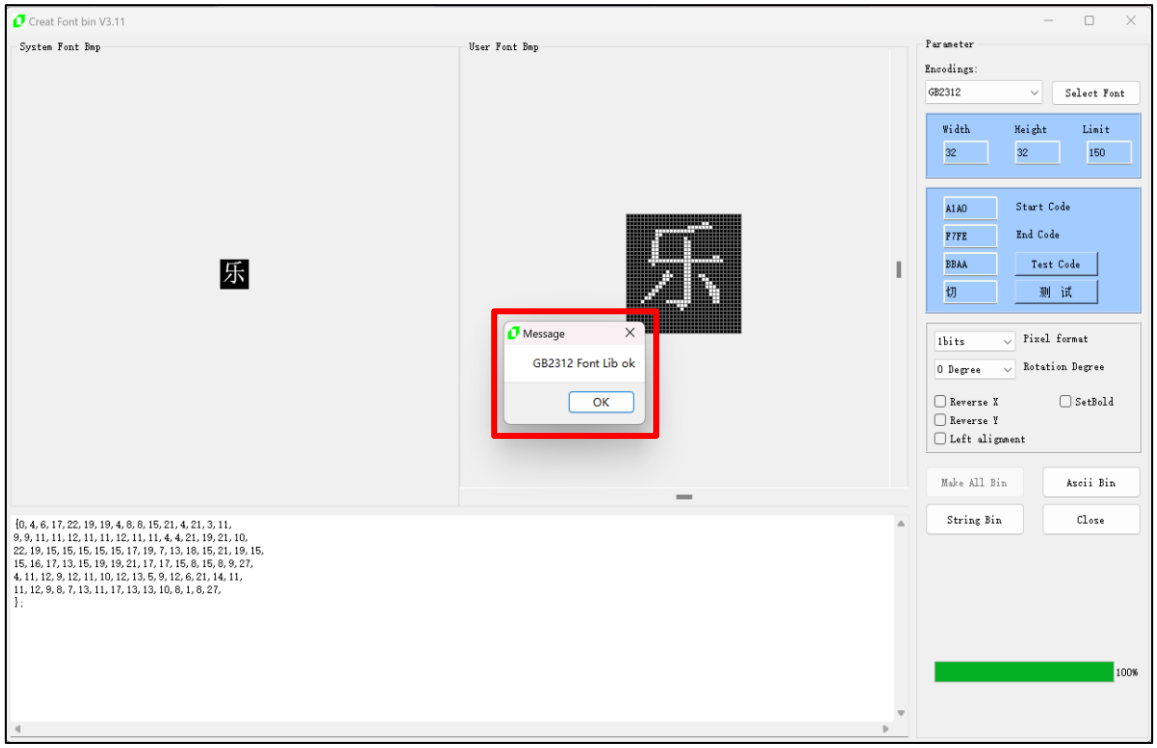


图 11-10：字库制作完成

7. 制作完成后可以在目标文件夹中看到导出的宋体 32_32.bin 文件：

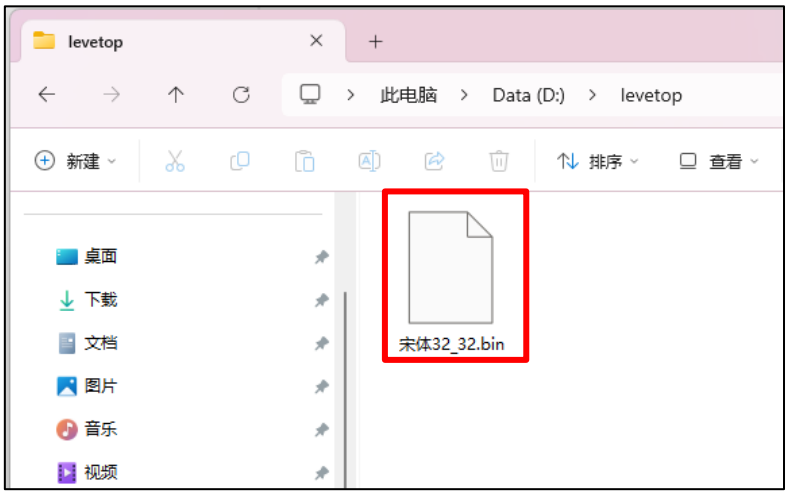


图 11-11：导出的字库 Bin 文件

12. 光标

LT758x 提供图形光标与文字光标功能。图形光标是由 32*32 或 64*64 像素的像素图形组成，它可以被显示在使用者定义的位置上，当设定位置改变时，图形光标就会被移动。而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。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。

提示：显示优先级为图型光标 > 文字光标 > PIP1 > PIP2 > Main

表 12-1：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 Y 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:

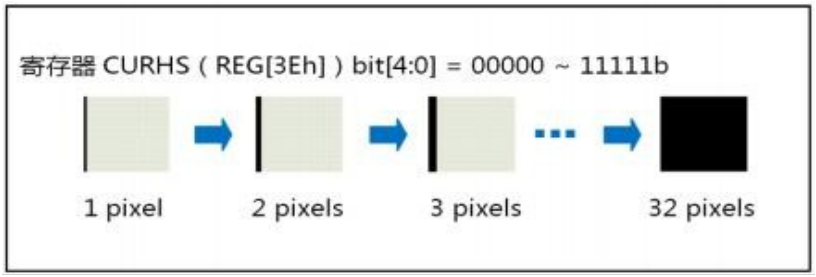


图 12-1：文字光标的宽度

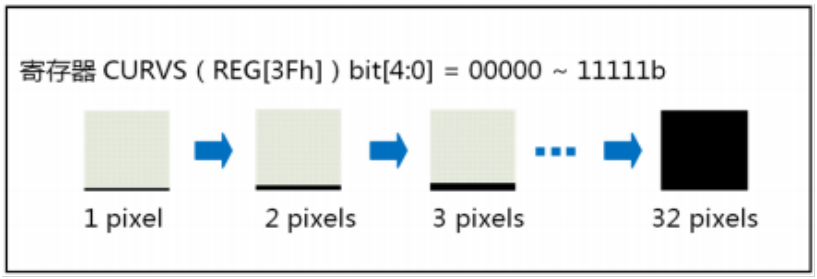


图 12-2：文字光标的高度

文字光标可以设定成固定频率的的闪烁或不闪烁，由寄存器 GTCCR（REG[3Ch]）设定，闪烁时，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 LT758_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 LT758_Enable_Text_Cursor(void);  // 使能文字光标
void LT758_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：

```
LT758_Text_cursor_Init(1, 15, 10, 2);
```

12.2 显示图形光标

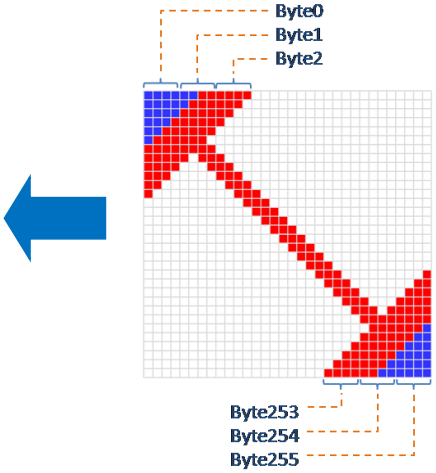
LT758x 的图形光标为 32*32 像素或 64*64 像素组成，而每个像素占用 2 个 bit，用来指定四种颜色：Color-0、Color-1、底圖背景色、底圖背景反向色：

表 12-2：图形光标像素定义

光标像素	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	底圖背景色
2’ b11	底圖背景反向色

因此在自建一个 32*32 像素图形光标时需要 256bytes 大小，而 64*64 像素图形光标需要 1KB 大小。LT758x 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。

Byte	Data
0	0 0 0 0 0 0 0 0
1	0 0 0 0 0 1 0 1
2	0 1 0 1 0 1 0 1
: : ::	: : ::
253	0 1 0 1 0 0 0 0
254	0 1 0 1 0 0 0 0
255	0 0 0 0 0 0 0 0



Color-0 的颜色：

Color-1 的颜色：

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 LT758_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 LT758_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 LT758_Enable_Graphic_Cursor(void);   // 使能图形光标
void LT758_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。

```
LT758_Graphic_cursor_Init(1, 0xff, 0x00, 0, 0, (unsigned char*)gImage_pen_il);
LT758_Graphic_cursor_Init(2, 0xff, 0x00, 0, 0, (unsigned char*)gImage_arrow_il);
LT758_Graphic_cursor_Init(3, 0xff, 0x00, 0, 0, (unsigned char*)gImage_busy_im);
LT758_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++) {
     LT758_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

本公司提供的专用整合软件 UartTFT_Vxxx.exe，里面有一项图形光标制作的软件工具，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，请参考以下的说明：

12.3.1 制作图形光标

1、点击【UartTFT 菜单>Cursor】即可打开图形光标制作界面：

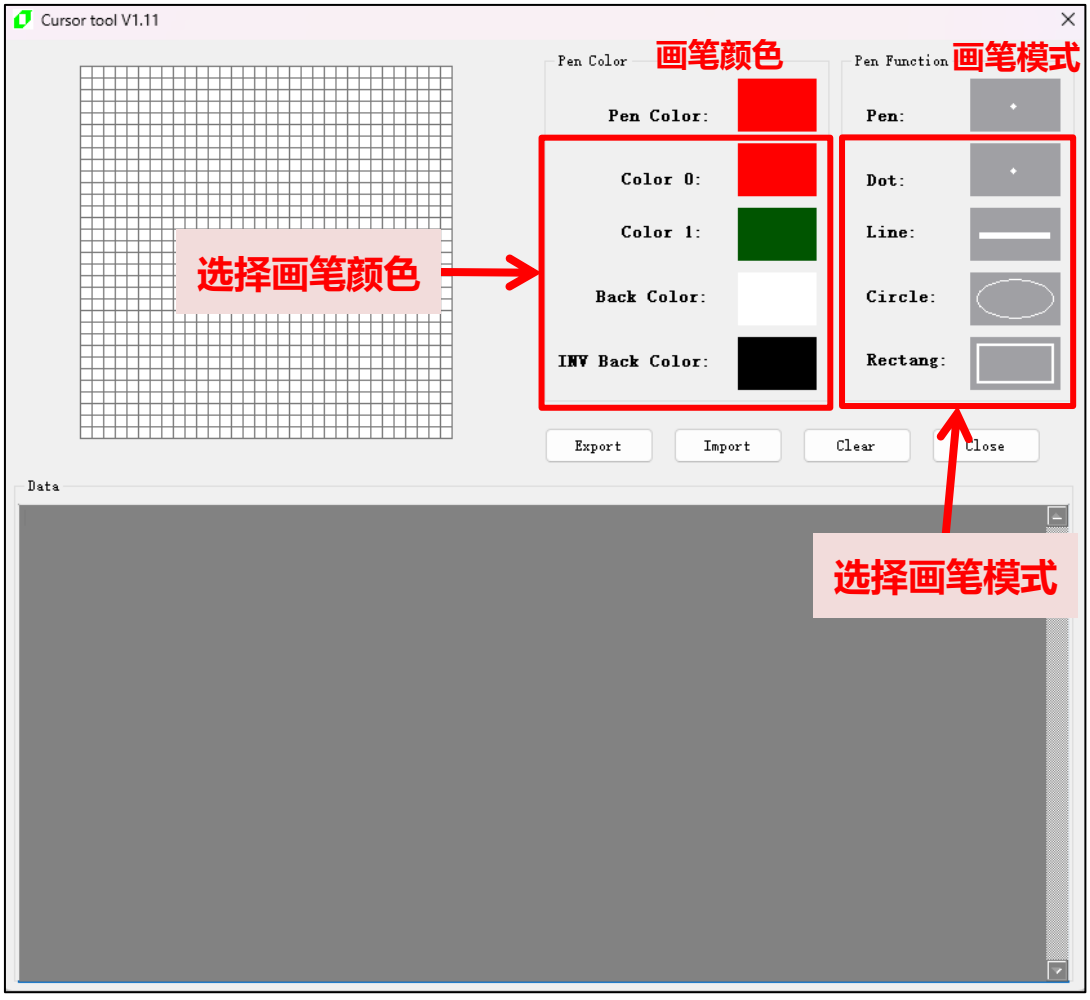


图 12-5：打开图像光标制作界面

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。

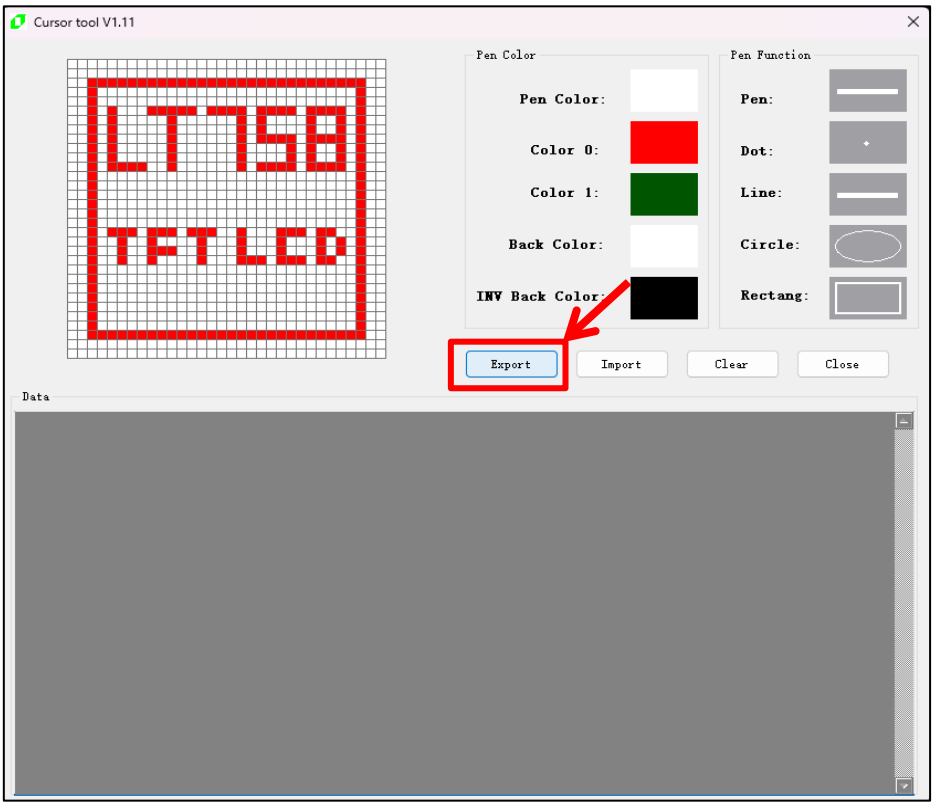


图 12-6：导出图形光标数据

在弹出的窗口选择保存路径，设置保存文件名，单击【保存】按钮。随后即可在保存路径下找到保存的 Bin 文件。

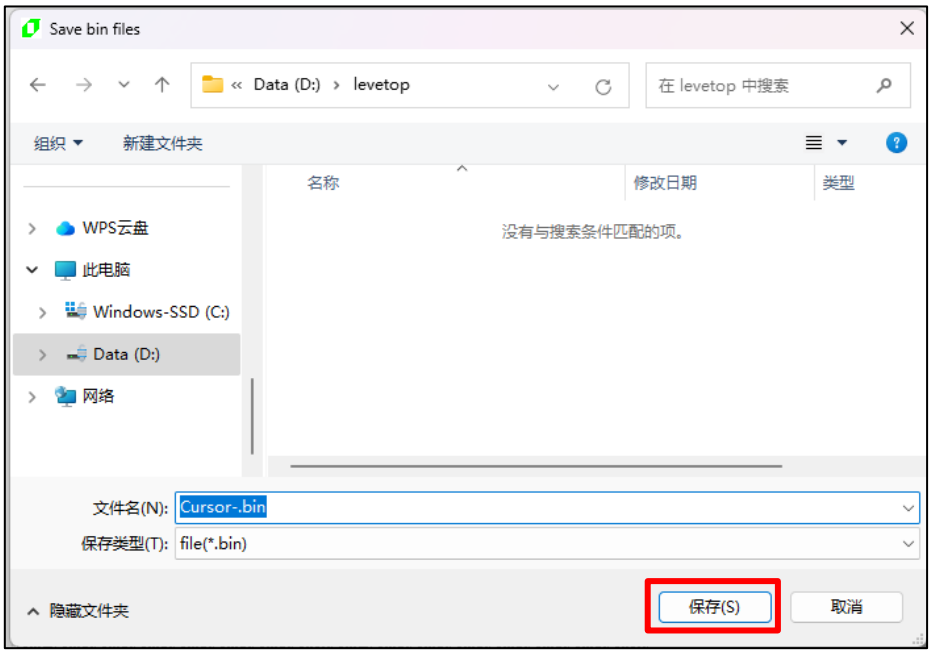


图 12-7：保存图形光标数据

3、全选 Data 框内的数据，右键复制图形光标数据：

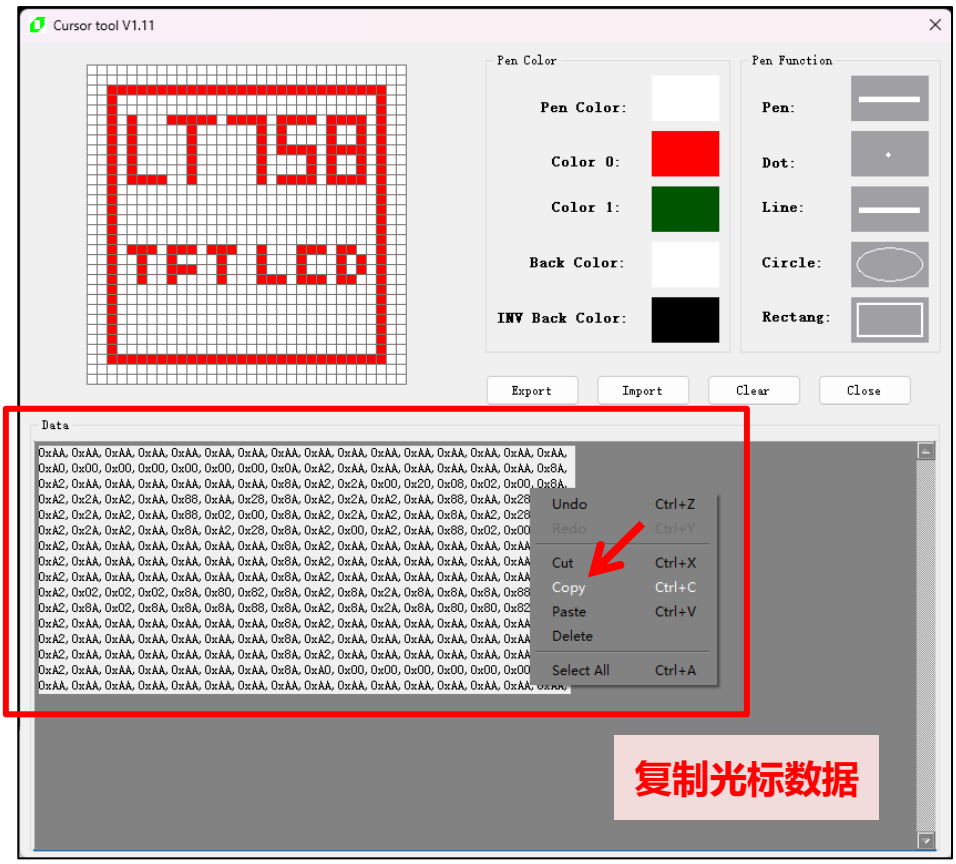


图 12-8：复制导出的光标数据

4、粘贴复制的光标数据到程序内：

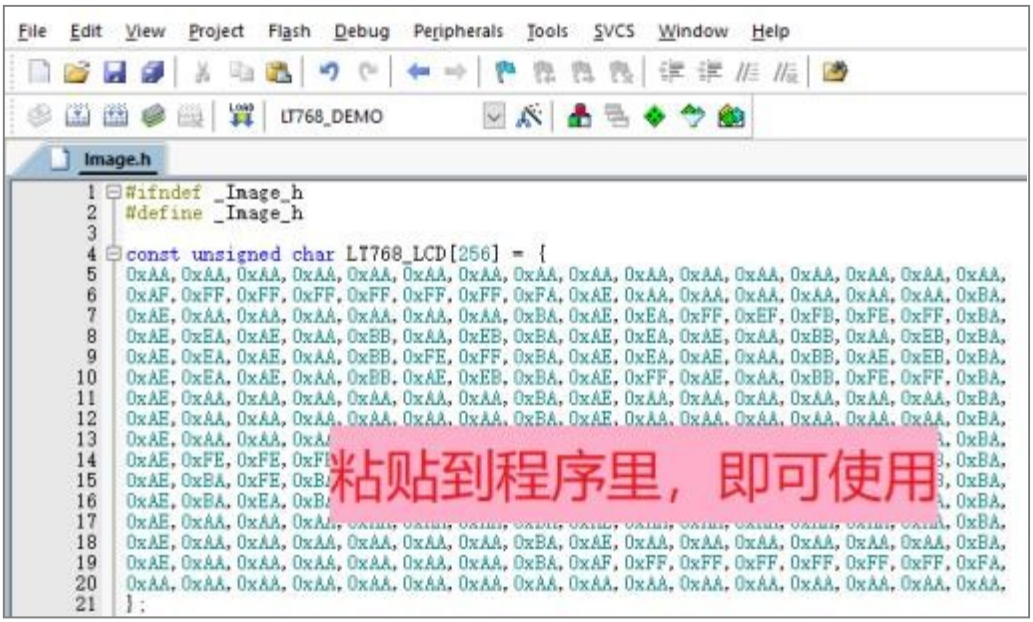


图 12-9：粘贴导出的光标数据

12.3.2 导入及修改图形光标

1. 复制要导入的光标数据：

图 12-10：复制导入的光标数据

2. 将光标数据粘贴到图形光标制作界面的 Data 内后，点击【Import】按钮：

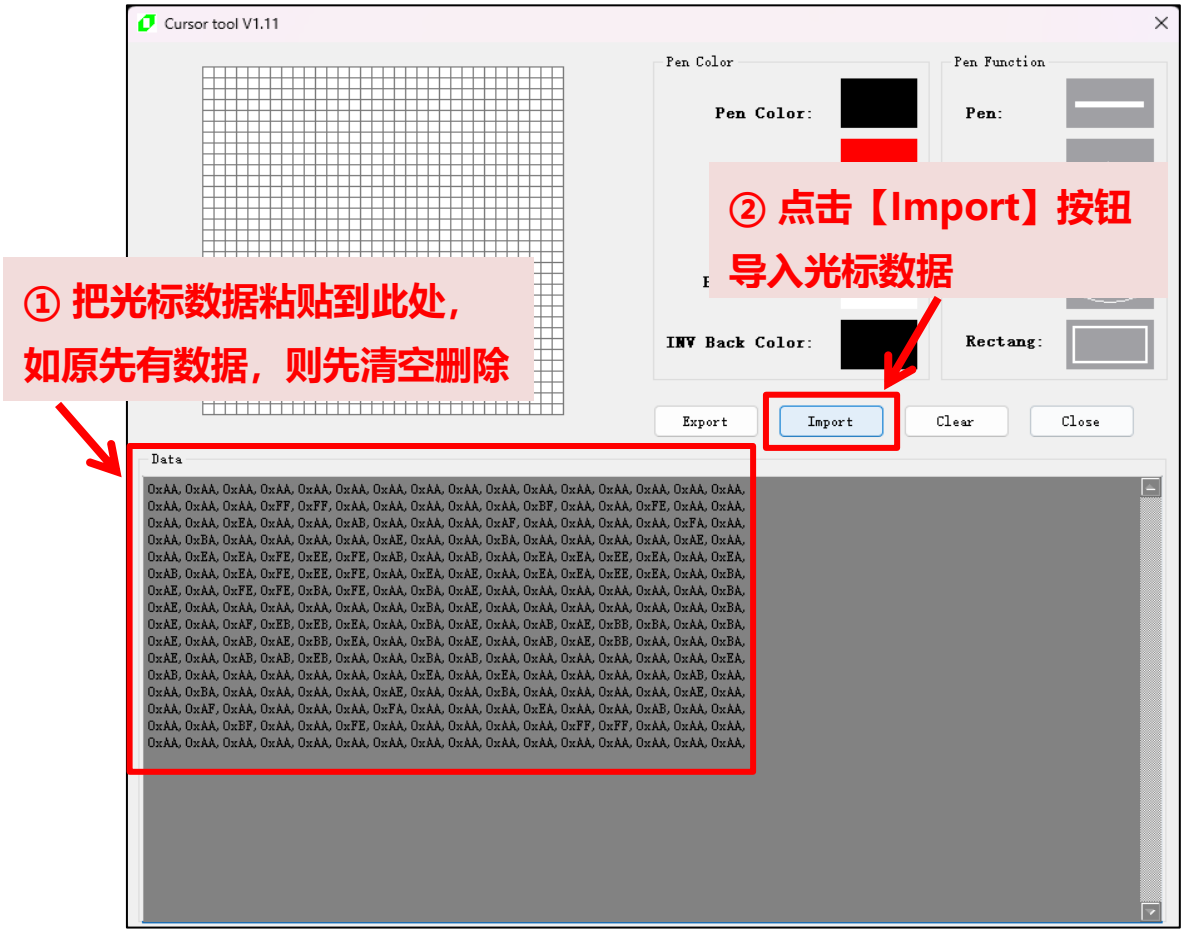


图 12-11：粘贴导入的光标数据

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 12-12：导入成功

13. 脉宽调制-PWM

LT758x 内建 PWM 功能，并且提供 2 个 PWM 输出：PWM0、PWM1。LT758x 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。

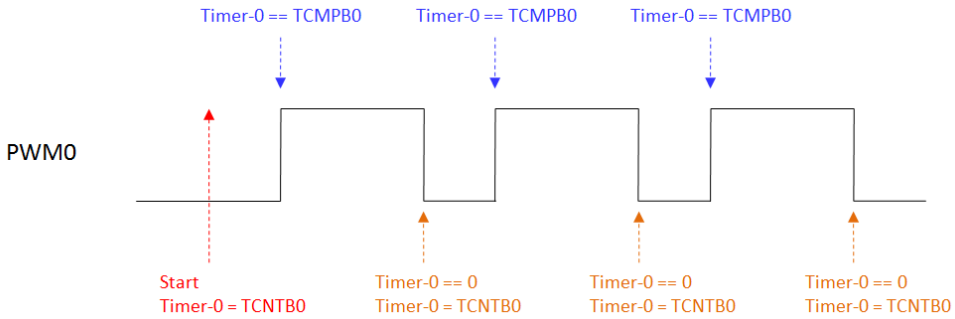


图 13-1：PWM 波形图

由以上动作可以了解，PWM0 的 Duty 决定于比较寄存器（REG[88h-89h]，TCMPB0]），例如想要藉由 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，因此可以藉由 PWM 产生的近似 DC 准位电压来控制 TFT 屏的背光。

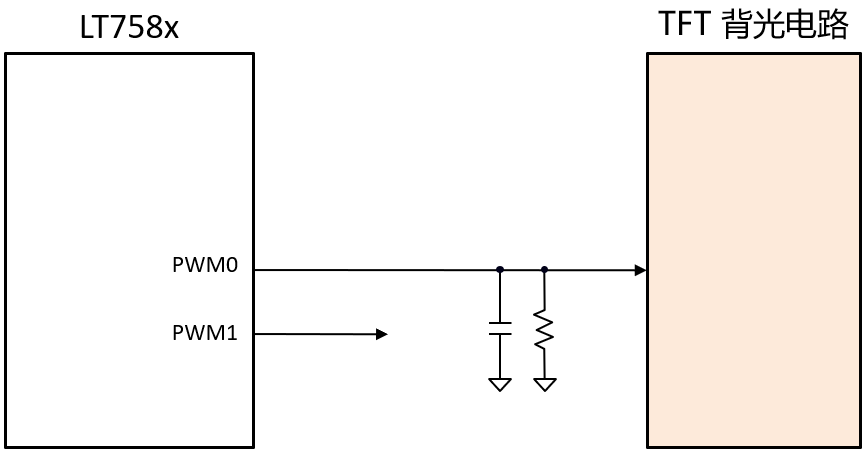


图 13-2：PWM 控制

在使用 PWM 之前需要对此先进行初始化。

```
void LT758_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 LT758_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 LT758_PWM0_Duty(unsigned short Compare_Buffer);
void LT758_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%：

```
LT758_PWM1_Init(1, 1, 10, 20060);        // 0.3=60/200
```

要修改 PWM1 的占空比为 50%：

```
LT758_PWM1_Duty(100);                    // 0.5=100/200
```

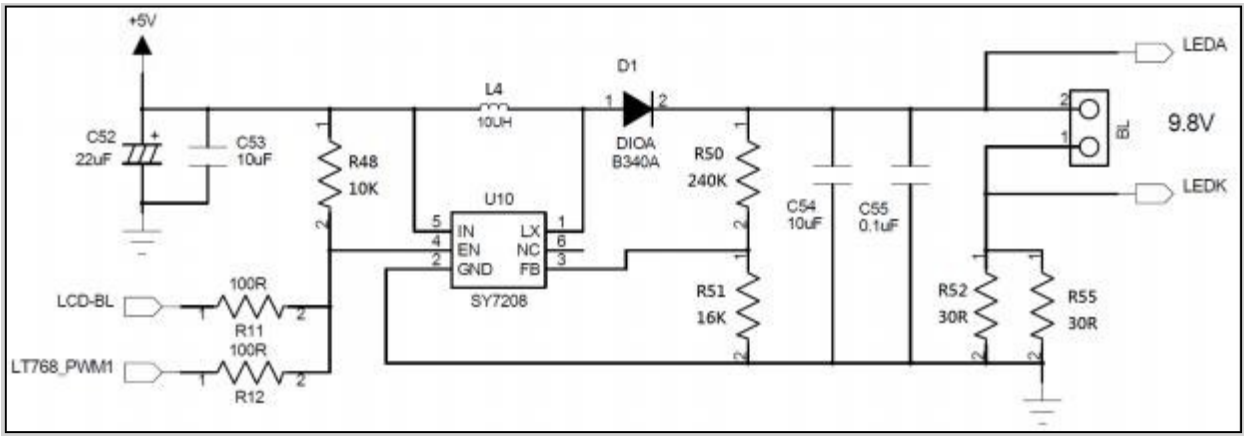



图 13-3：PWM 控制背光参考原理图（7”1024*600 屏）

14. SPI Master

LT758x 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 Mode1 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。

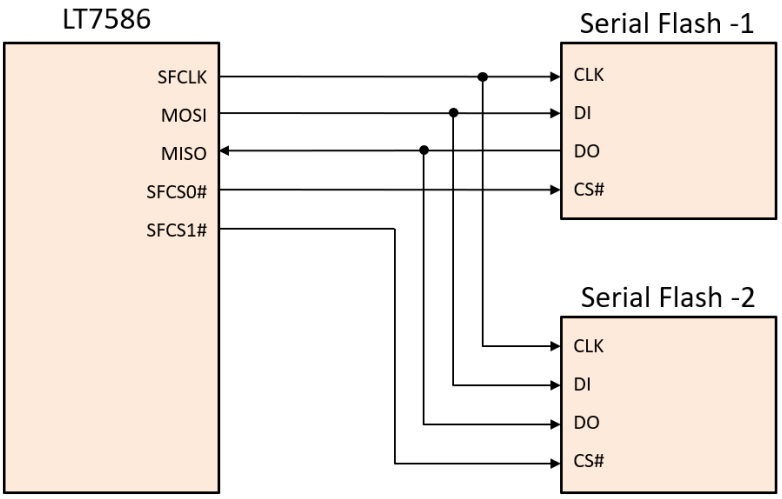


图 14-1：LT758x 串行 SPI Flash 应用电路

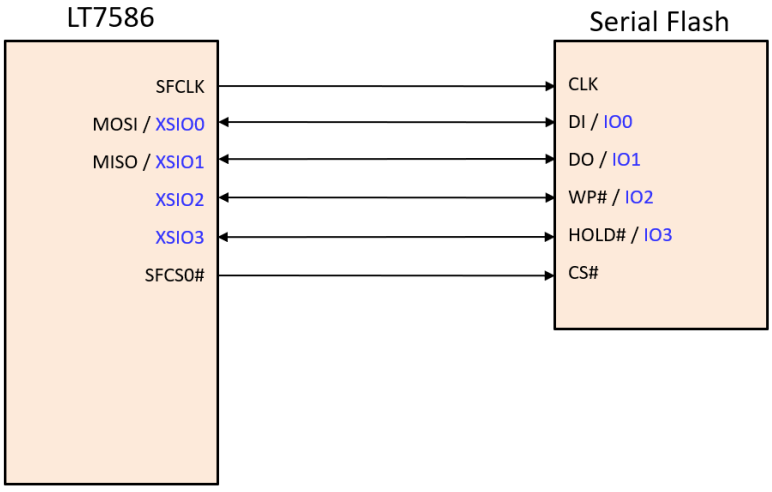


图 14-2：LT758x 串行 4 线 QSPI Flash 应用电路

14.1 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 Flash。

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 LT758x 规格书第 13 章的说明。

14.1.1 串行闪存在线性模式下的 DMA 传输

- **24 位寻址**

该函数支持 LT758 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 LT758_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)*
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：

```
LT758_DMA_24bit_Linear(0, 0, 0x0078DC20, x003E537E0, 0x0009B520);
```

- **32 位寻址**

该函数支持 LT758 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 LT758_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：

```
LT758_DMA_32bit_Linear(0, 0, 0x0078DC20, 0x003E537E0, 0x0009B520);
```


14.1.2 串行闪存在区块模式下的 DMA 传输

■ **24 位寻址**

该函数支持 LT758 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 LT758_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);
LT758_DMA_24bit_Block(1, 0, 0, 0, 1024, 600, 1024, 0x80000);
```


■ 32 位寻址

该函数支持 LT758 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 LT758_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);
LT758_DMA_32bit_Block(1, 0, 0, 0, 1024, 600, 1024, 0x80000);
```


14.2 Bin 文件的结合

在 14.3 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，专用的整合软件 UartTFT_Vxxx.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。详细步骤使用者可以参考以下的说明：

1、点击【UartTFT 菜单>BinTool】进入文件整合界面，该工具最多能整合 6 个 Bin 文件，点击【File 1~6】可依次添加。

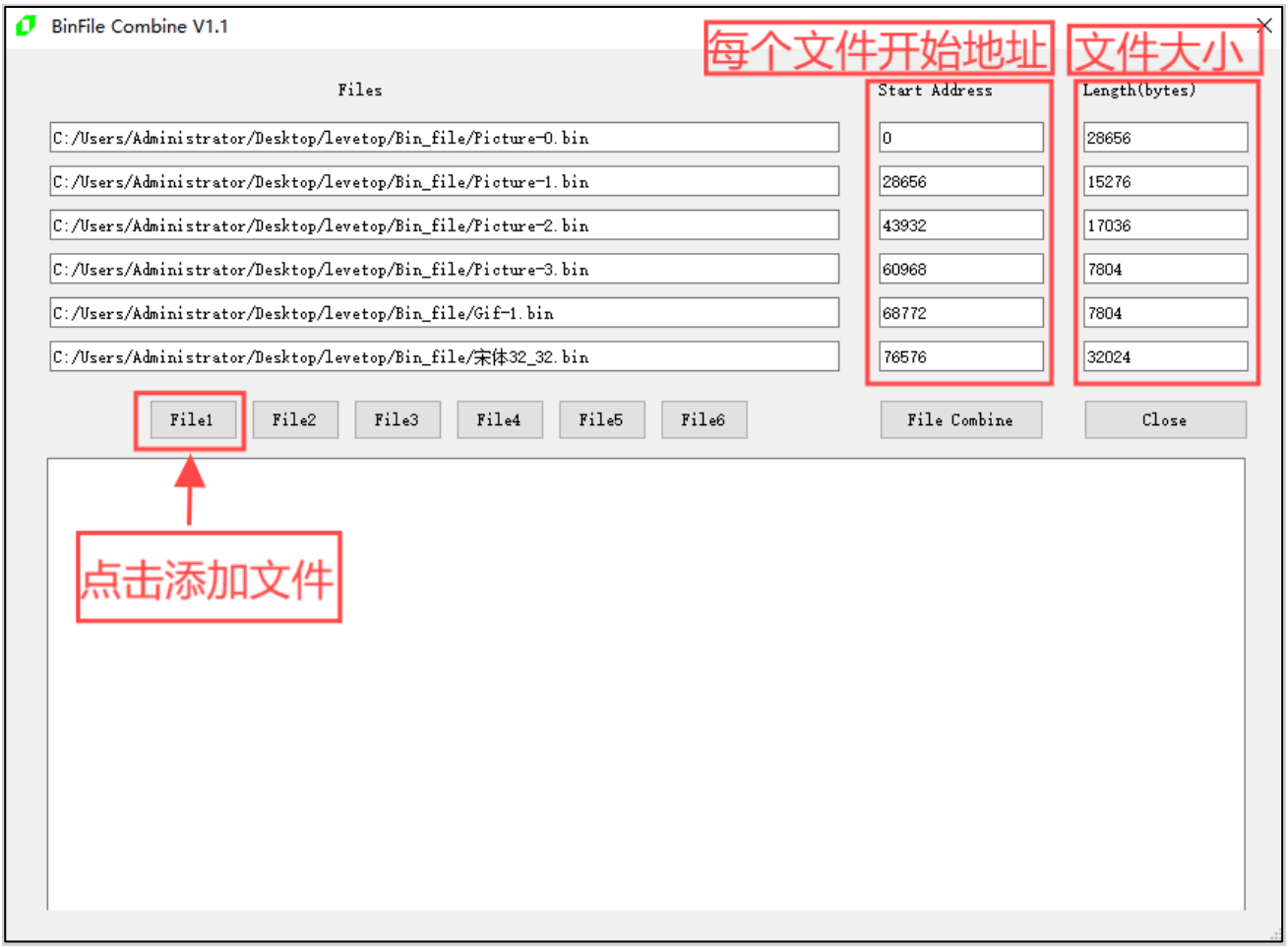


图 14-3：Bin 文件整合

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。

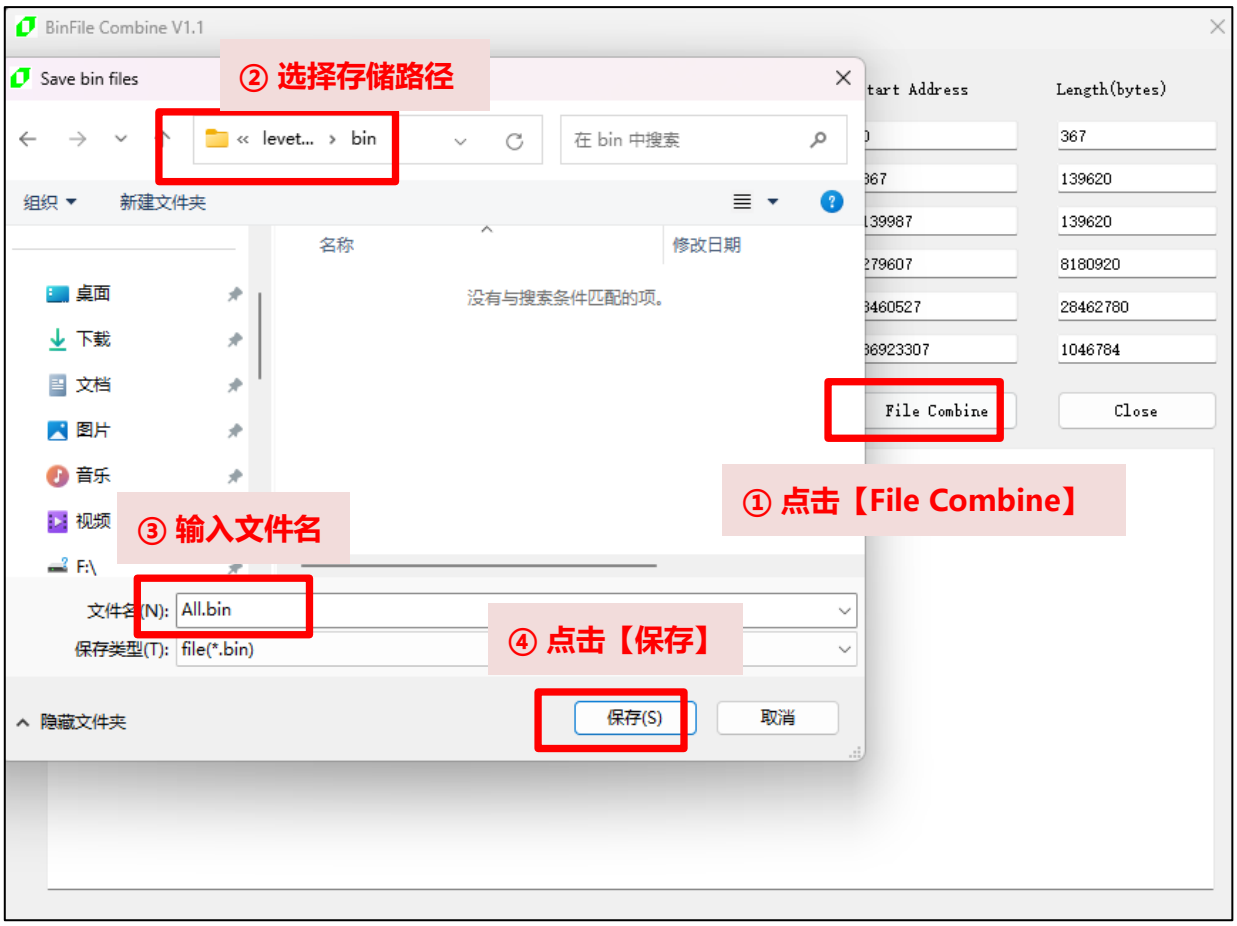


图 14-4：保存整合文件

当显示“Combine over”时，整合成功，且界面窗口显示每个源文件的地址和大小，同时生成一个 All-Addr.txt 文件，便于查阅每个源文件的地址、大小等详细信息。

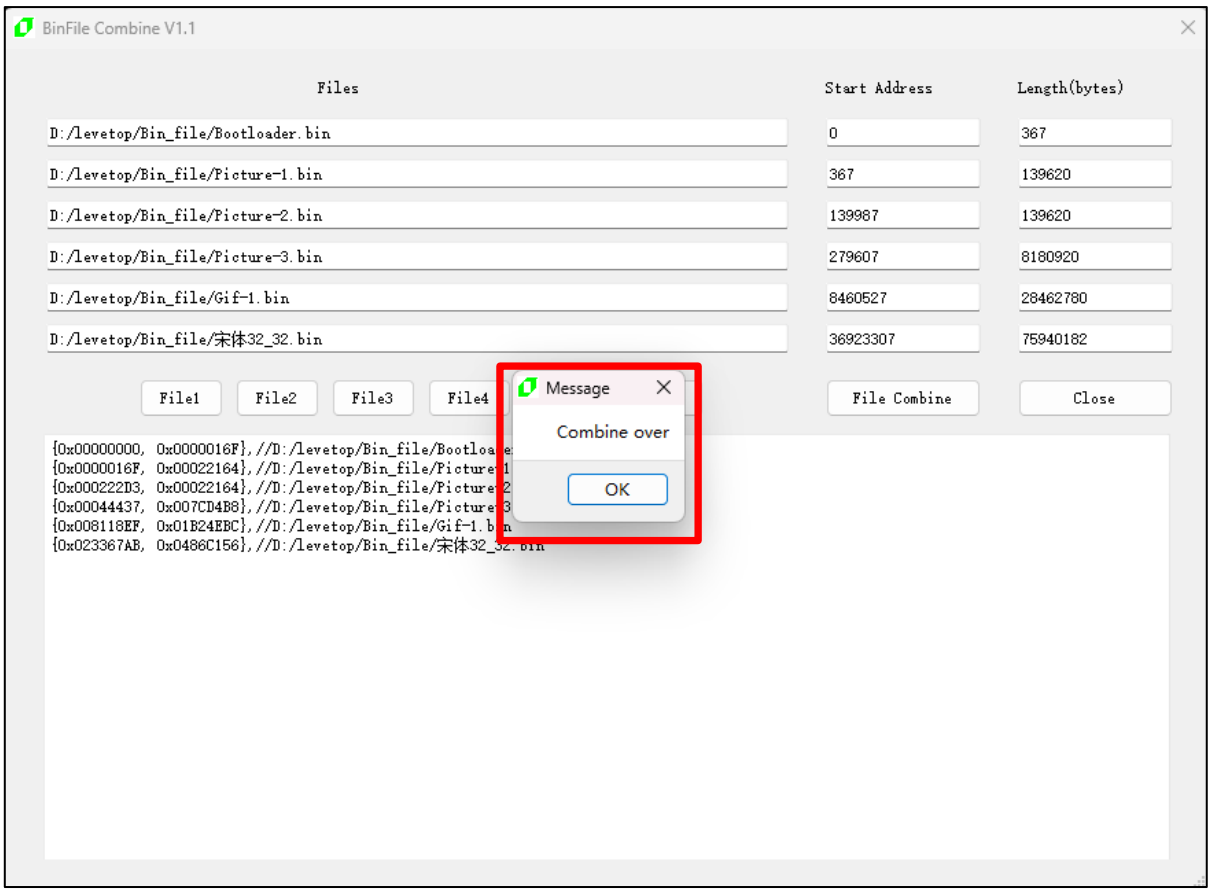


图 14-5：整合成功

3、生成的 All.txt 文件：

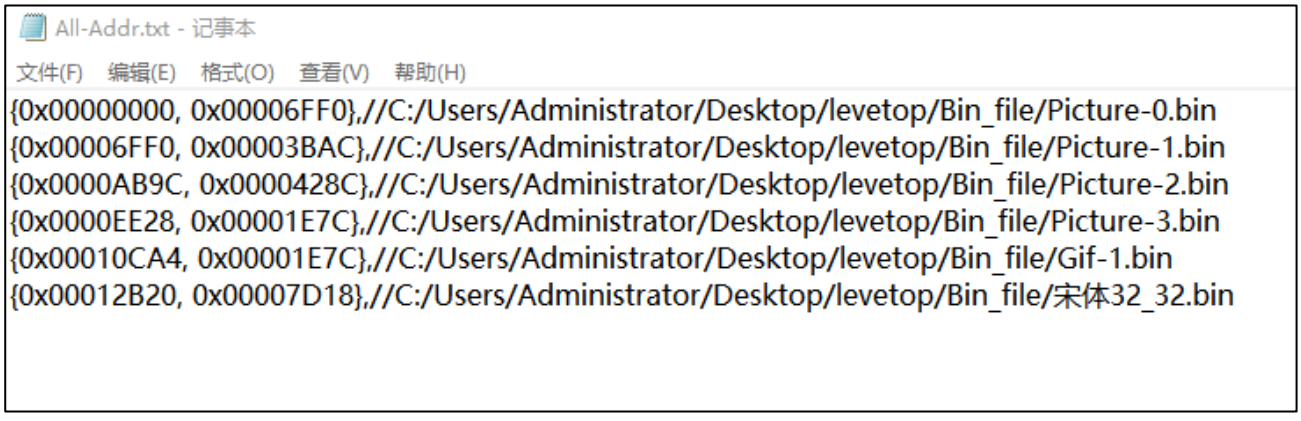


图 14-6：保存文件信息

整合完成后可以在目标文件夹中看到导出的 All.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 LT758x 的 SPI Flash。

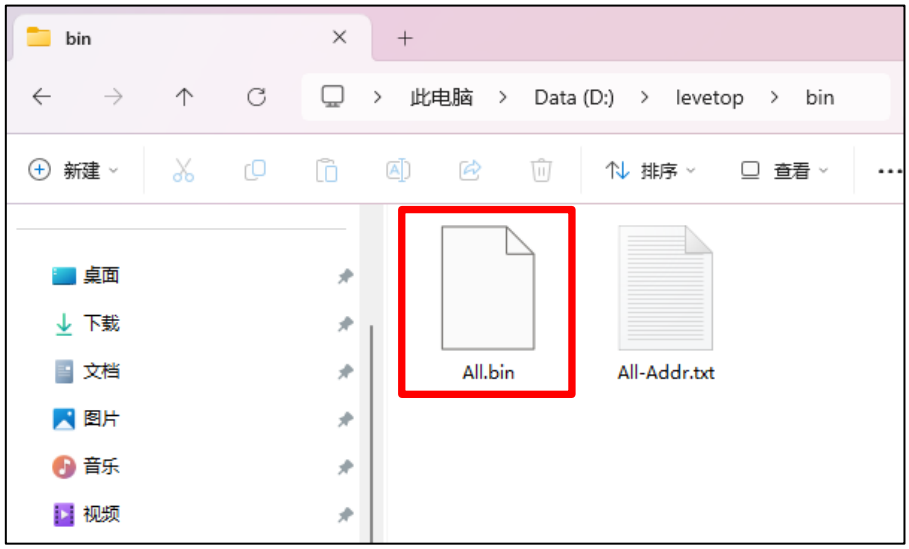


图 14-7：导出的 Bin 整合文件

14.3 程序如何调用 SPI Flash 的 Bin 文件

以下用 1 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，在程序中调用 SPI Flash 内的字库（16_16 标楷体）。

范例：从 LT758 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“深圳市乐升半导体有限公司”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

LT758_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）-----*/
LT758_Select_Outside_Font_Init(1, 0, 0x0115E1BD, 1024*600*2, 0x00041400, 16, 1, 1, 1, 1);

/*-----显示文字-----*/
LT758_Print_Outside_Font_String(625, 150, Red, White, (u8*)"深圳市乐升半导体有限公司");
```

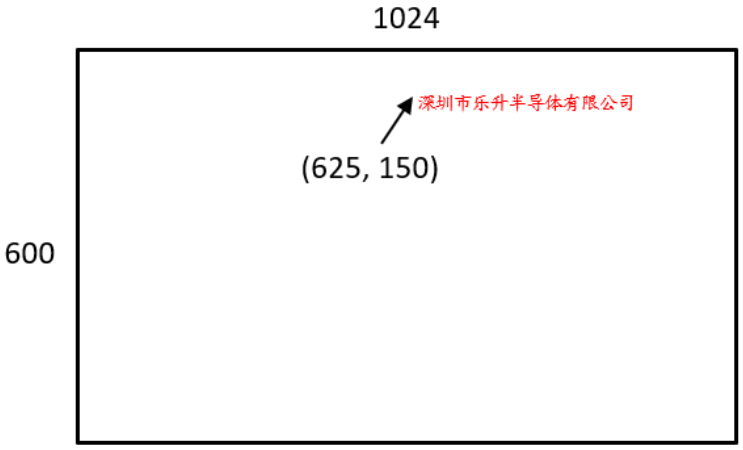


图 14-8：调用 SPI Flash 内的字库（范例二）

15. 图像解码单元（Image Decode Unit）

LT758x 提供图像解码器单元，支持 JPEG 和 BMP 图像格式。LT758x 可以自动区分上述两种格式，并自动将它们解析到相对解码器。JPEG（Joint Photographic Experts Group）是一种常见的图像压缩格式，它采用有损压缩方法以减小文件大小，LT758x 支持标准的 JPEG 基线（Baseline）格式，用户应提前将图像加载到外部的 Serial Flash 中，并通过设置 DMA、CANVAS 寄存器将它们显示在 LCD 屏幕上。

15.1 JPG 图像解码流程

LT758x 支持基线（Baseline）的 JPG 格式解码。

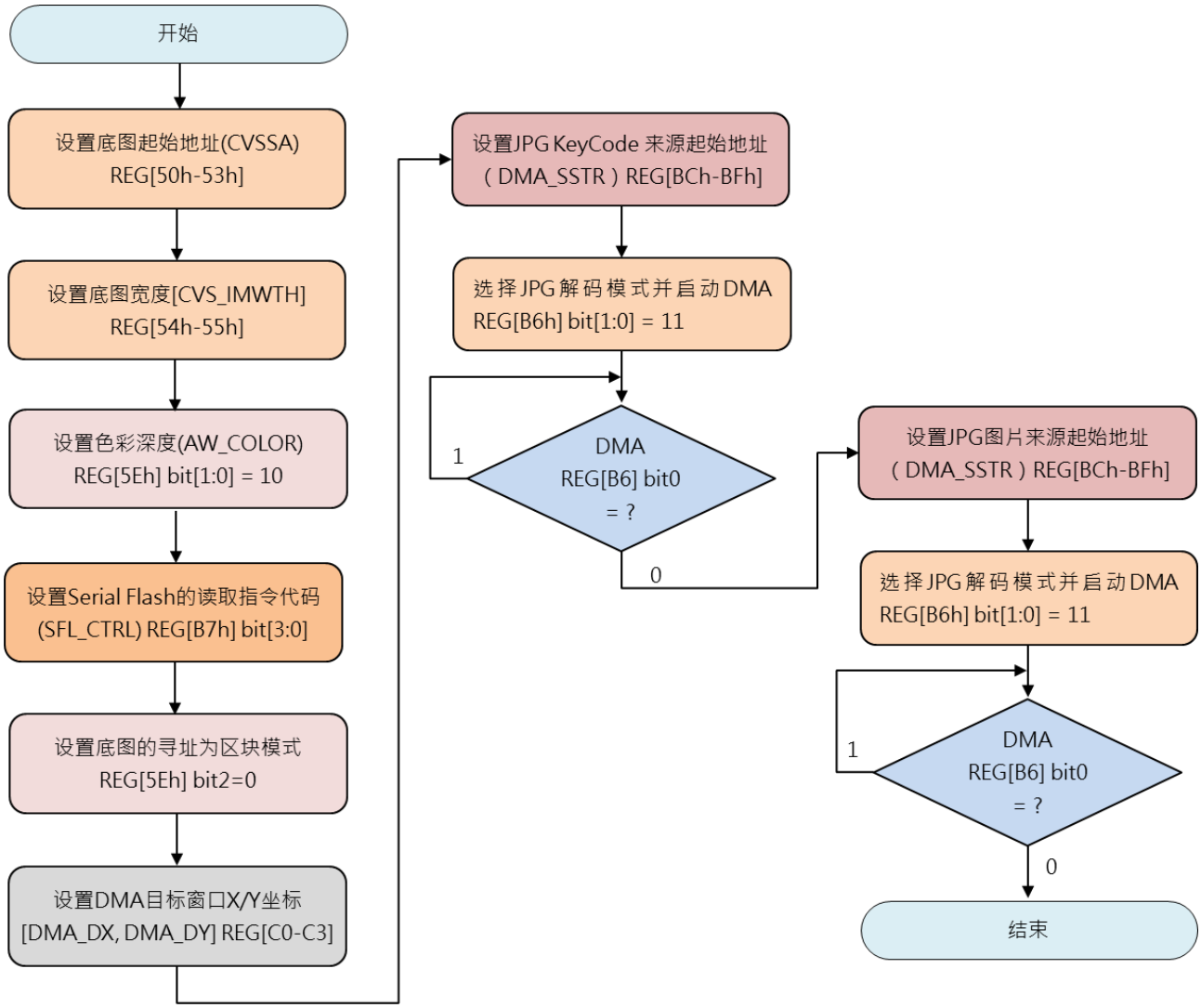


图 15-1：JPG 图像解码显示流程图

15.2 JPG 解码程序例程

JPG 解碼设置例程：从 SPI Flash 读取一张 JPG 图档，分辨率 800x480，解碼后写入 LT758x SDRAM。

```
Main_Image_Start_Address(0);              //主窗口：设为显示图层
Main_Image_Width(800);                    //主窗口宽度：800
Main_Window_Start_XY(0,0);                //主窗口起点坐标：(0, 0)
Canvas_Image_Start_address(0);            //画布起点坐标：(0, 0)
Canvas_image_width(800);                  //画布宽度：800
Active_Window_XY(0,0);                    //活动窗口起点坐标：(0, 0)
Active_Window_WH(800, 480);               //活动窗口宽高：800, 480
Memory_24bpp_Mode();                      //设置颜色深度：24bpp
Graphic_Mode();                           //设置为图形模式，REG[03] bit2

Enable_SFlash_SPI();                      //使能 Flash 的 SPI 接口，REG[01h] bit1= 1
DMA_Read_6BH();                           //设置 Flash 的’读取’指令代码，REG[B7h] bit[3:0]
Reset_CPOL();                             //设置 SPI 通信模式：Mode 0，REG[B9h] bit1
Reset_CPHA();                             //设置 SPI 通信模式：Mode 0，REG[B9h] bit0
SPI_Clock_Period(0);                      //设置 SPI 分频，REG[BBh] bit[3:0]

Memory_XY_Mode();                         //设置为区块模式，REG[5Eh] bit2 = 0

SFI_DMA_Destination_Upper_Left_Corner(X1,Y1); //设置 DMA 目标窗口的 X/Y 坐标, REG[C0h-C3h]

//设置 JPG KeyCode 来源起始地址 (in Flash)
SFI_DMA_Source_Start_Address(JPG_Key_Addr);     // REG[BCh-BFh]
Start_JPG_DMA();                          //启动 JPG 解碼及 DMA，REG[B6h] bit[1:0] = 11
Check_Busy_JPG_DMA();                     //检测 DMA 动作是否完成，REG[B6h] bit0

//设置 JPG 图片来源起始地址(in Flash),
SFI_DMA_Source_Start_Address(JPG_Pic_Addr);     // REG[BCh-BFh]
Start_JPG_DMA();                          //启动 JPG 解碼及 DMA，REG[B6h] bit[1:0] = 11
Check_Busy_JPG_DMA();                     //检测 DMA 动作是否完成，REG[B6h] bit0
```

注意：

1. 在进行 JPG 图片解码前，需先对 16 Bytes 的 JPG KeyCode 运行解码程序，以启动解码功能。使用者需将该 JPG KeyCode 烧录在 SPI Flash (可自行指定地址) 中，以便在代码中指定该地址。
2. JPG KeyCode 共有 16bytes： { FF DB FF E0 00 10 4A 45 52 52 00 01 01 01 00 0A }
3. 利用乐升半导体提供的专用整合软件(UartTFT_Vxxx.exe)内的 JPG 转档工具，会在转换 JPG 图片时，在记录每张 JPG 图片的数据前，先加入前述 16Bytes 的 JPG KeyCode。

15.3 制作 JPG 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 LT758x 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，专用的整合软件 UartTFT_Vxxx.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，详细步骤使用者可以参考以下的说明：

1、打开软件 UartTFT_Vxxx.exe：

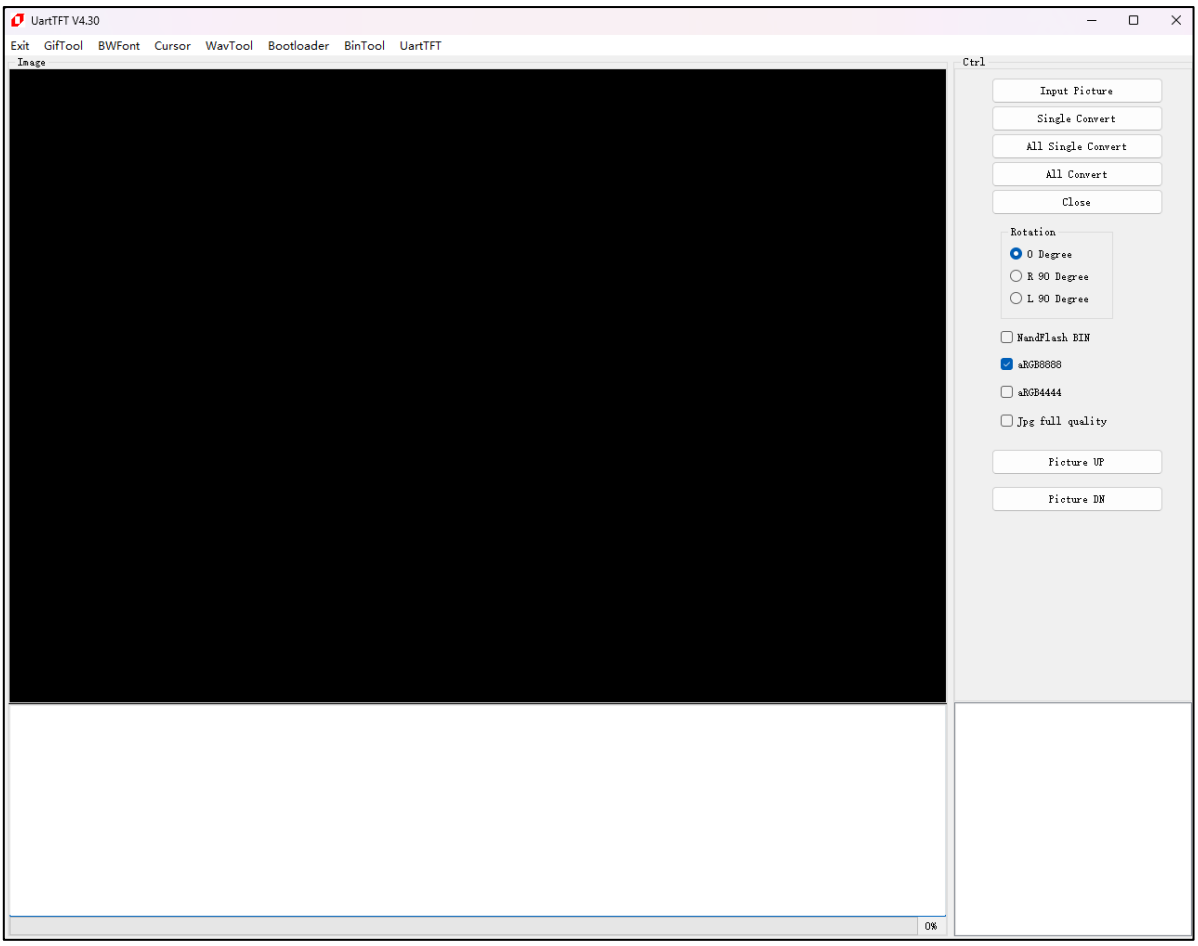


图 15-2：执行 UartTFT.exe

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：

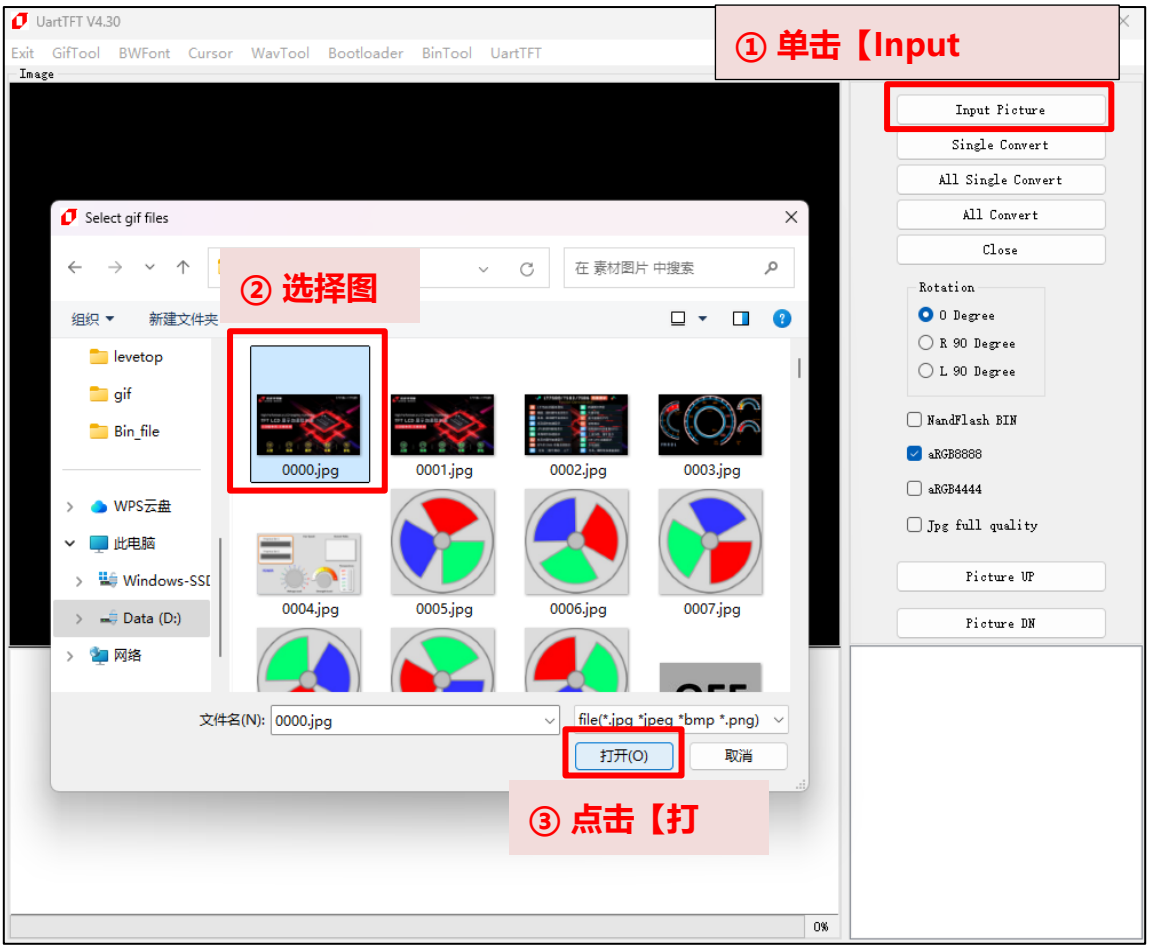


图 15-3：导入图片

图 15-4：导入完成

3、图片输出设定，可勾选 16bpp (aRGB4444)或 32bpp (aRGB8888) 格式，以及选择输出 JPG 全质量格式，你还可以设置图片的旋转角度，可选无旋转、向左旋转 90 度或向右旋转 90 度效果：

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

5、成功导出图片 Bin 文件：

图 15-7：导出成功

6、导出图片后可以在目标文件夹中看到导出的 Picture-1.bin 文件以及 Picture-1-Addr.txt 和 Picture-1-StartAddr.txt 文件：

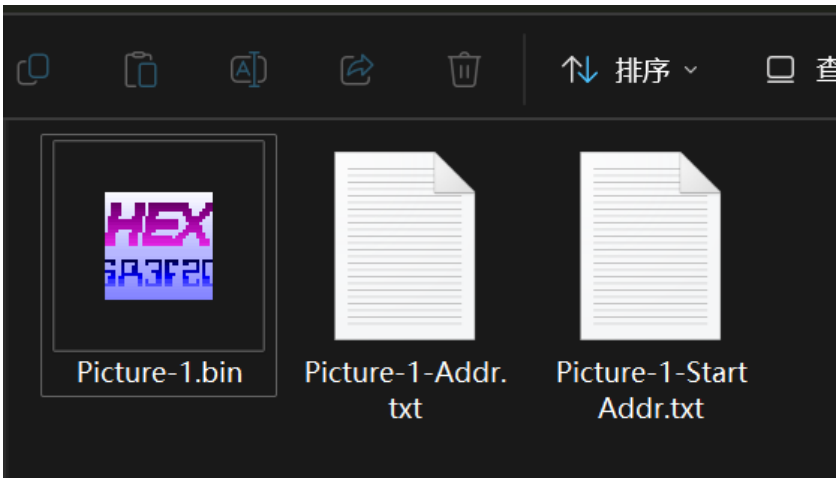


图 15-8：导出的图片 Bin 文件

7、其中后缀为 StartAddr.txt 文件里包含了每一张图片在 Bin 文件里的起始地址，可直接复制到程序中调用：

图 15-9：StartAddr.txt 文件

16. 电源管理

在电源的管理模式上，LT758x 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 16-1：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. LT758x 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。

16.1 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。

16.2 待命模式（Standby）

```
void LT758_Standby(void);          // 进入待命模式
void LT758_Wkup_Standby(void);     // 从待机模式中唤醒
```

16.3 暂停模式（Suspend）

```
void LT758_Suspend(void);          // 进入暂停模式
void LT758_Wkup_Suspend(void);     // 从暂停模式中唤醒
```


16.4 休眠模式（Sleep）

```
void LT758_SleepMode(void);        // 进入休眠模式
void LT758_Wkup_Sleep(void);       // 从休眠模式中唤醒
```

16.5 唤醒（Wake up）

有两种方法可以从省电模式中唤醒：外部中断唤醒、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。使用外部中断唤醒唤醒后 LT758x 会发出中断信号 INT#给 MCU。

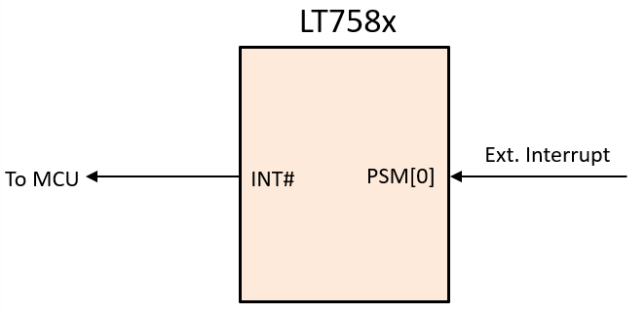


图 16-1：外部中断唤醒

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 LT758_Wkup_Standby(void);     // 从待机模式中唤醒
void LT758_Wkup_Suspend(void);     // 从暂停模式中唤醒
void LT758_Wkup_Sleep(void);       // 从休眠模式中唤醒
```

17. 程序库说明

17.1 程序库

LT758x 的程序库包含如下四个部分，使用者可以到乐升半导体的官网下载区下载“LT758_Library_202xxxxx.rar”，再解压缩后取得。

- T758x 底层寄存器的封装函数：

  档案名：LT758.c

- LT758x 底层寄存器封装函数声明：

  档案名：LT758.h

- LT758x 的功能封装函数：

  档案名：LT758_Lib.c

- LT758x 的功能封装函数声明：

  档案名：LT758_Lib.h

17.2 应用软件

乐升半导体提供了一个 LT758x 专用的整合软件，包含如下六个功能，使用者可以到乐升半导体的官网下载区下载“UartTFT_Vx.xx_date.rar”，再解压缩之后取得。

- 专用整合软件：

  档案名：UartTFT_V4.30.exe（或 V4.30 以上的版本）

UartTFT_V4.30.exe 这个专用整合软件针对 LT758x 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，此软件的六个功能，分别为：

一、 制作图片 Bin 文件；参考第 15.3 节的范例

二、 制作字库 Bin 文件；参考第 11.3 节的范例

三、 制作「GIF Bin 文件」；

四、 制作音频 Bin 文件；

五、 制作图形光标；参考第 12.3 节的范例

六、 Bin 文件整合；参考第 14.2 节的范例

17.3 程序库列表

表 17-1：程序库列表

No.	函 数 名 称	功 能	页数
启始設定			
1	LT758_SW_Reset()	软件复位	
2	LT758_PLL_Initial()	时钟與 PLL 設定	
MCU 接口：8 位的 8080 接口			
3	FSMC_8_ CmdWrite()		
4	FSMC_ 8_ DataWrite()		
5	FSMC_ 8_ DataWrite_Pixel()		
6	FSMC_8_DataWrite_Pixel_24()		
7	u8 FSMC_8_StatusRead()		
8	u16 FSMC_8_DataRead()		
MCU 接口：16 位的 8080 接口			
9	FSMC_16_CmdWrite()		
10	FSMC_16_DataWrite()		
11	FSMC_16_DataWrite_Pixel()		
12	FSMC_16_DataWrite_Pixel_24()		
13	u8 FSMC_16_StatusRead(void)		
14	u16 FSMC_16_DataRead(void)		
MCU 接口：SPI 接口			
15	SPI_CmdWrite_8()		
16	SPI_DataWrite_8()		
17	SPI_DataWrite_Pixel_8()		
18	SPI_DataWrite_Pixel_8_24()		
19	u8 SPI_StatusRead_8()		
20	u16 SPI_DataRead_8()		
21	SPI_CmdWrite_16()		
22	SPI_DataWrite_16()		
23	SPI_DataWrite_Pixel_16()		
24	SPI_DataWrite_Pixel_16_24()		
25	u8 SPI_StatusRead_16()		
26	u16 SPI_DataRead_16()		

No.	函 数 名 称	功 能	页数
MCU 接口：I2C 接口			
27	u8 IIC_CmdWrite()		
28	u8 IIC_DataWrite()		
29	IIC_DataWrite_Pixel()		
30	IIC_DataWrite_Pixel_24()		
31	u8 IIC_StatusRead()		
32	u8 IIC_DataRead()		
显示内存（SDRAM）设定			
33	LT758_SDRAM_initial()	顯示內存（SDRAM）設定	
控制 LCD 的输出信号			
34	Set_LCD_Panel()	LCD 屏設定	
35	VSCAN_T_to_B()	LCD 的扫描方式从上到下	
36	VSCAN_B_to_T()	LCD 的扫描方式从下到上	
37	PDATA_Set_RGB()	RBG 的输出方式:RGB	
38	PDATA_Set_RBG()	RBG 的输出方式:RBG	
39	PDATA_Set_GRB()	RBG 的输出方式:GRB	
40	PDATA_Set_GBR()	RBG 的输出方式:GBR	
41	PDATA_Set_BRG()	RBG 的输出方式:BRG	
42	PDATA_Set_BGR()	RBG 的输出方式:BGR	
43	PDATA_Set_GRAY()	RBG 的输出方式:GRAY	
45	HSYNC_Low_Active()	HSYNC 的动作方式：Low	
46	HSYNC_High_Active()	HSYNC 的动作方式：High	
47	VSYNC_Low_Active()	VSYNC 的动作方式：Low	
48	VSYNC_High_Active()	VSYNC 的动作方式：High	
49	DE_High_Active()	DE 的动作方式：High	
50	DE_Low_Active()	DE 的动作方式：Low	
设置主视窗是要以几位的颜色来显示			
51	Select_Main_Window_8bpp()	256 色	
52	Select_Main_Window_16bpp()	65K 色	
53	Select_Main_Window_24bpp()	262K/16.7M 色	
设置主视窗			
54	Main_Image_Start_Address()	设置主视窗的开始显示地址	

No.	函 数 名 称	功 能	页数
55	Main_Image_Width()	设置主视窗的宽度	
56	Main_Window_Start_XY()	设置主视窗的起始坐标	
设置底图视窗			
57	Canvas_Image_Start_address()	设置底图的起始位置	
58	Canvas_image_width()	设置底图的宽度	
设置工作视窗			
59	Active_Window_XY()	设置工作视窗的起始位置	
60	Active_Window_WH()	设置工作视窗的大小	
MCU 写入数据到内存			
61	MPU8_8bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 8 位色深来显示	
62	MPU8_16bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 16 位色深来显示	
63	MPU8_24bpp_Memory_Write()	MCU 使用 8 位数据驱动，而且用 24 位色深来显示	
64	MPU16_16bpp_Memory_Write()	MCU 使用 16 位数据驱动，而且用 16 位色深来显示	
65	MPU16_24bpp_Mode1_Memory_Write()	MCU 使用 16 位数据驱动，而且用 24 位色深来显示	
66	MPU16_24bpp_Mode2_Memory_Write()	MCU 使用 16 位数据驱动，而且用 24 位色深来显示	
画中画 PIP			
67	LT758_PIP_Init()	画中画（PIP）视窗的设定	
68	LT758_Set_DisWindowPos()	画中画视窗显示位置与图像位置	
69	Enable_PIP1()	使能 PIP1	
70	Disable_PIP1()	失能 PIP1	
71	Enable_PIP2()	使能 PIP2	
72	Disable_PIP2()	失能 PIP2	
主控端写入的内存方向控制			
73	MemWrite_Left_Right_Top_Down()	左→右，然后上→下（初始值）	
74	MemWrite_Right_Left_Top_Down()	右→左，然后上→下（水平翻转）	
75	MemWrite_Top_Down_Left_Right()	上→下，然后左→右（向右旋转 90°并且水平翻转）	
76	MemWrite_Down_Top_Left_Right()	下→上，然后左→右（向左旋转 90°）	

No.	函 数 名 称	功 能	页数
彩条（Color Bar）显示			
77	LT758_Color_Bar_ON()	显示彩条	
78	LT758_Color_Bar_OFF()	关闭彩条	
几何绘图			
79	LT758_DrawLine()	画细线	
80	LT758_DrawLine_Width()	画粗线	
81	LT758_DrawCircle()	画空心圆形	
82	LT758_DrawCircle_Fill()	画实心圆形	
83	LT758_DrawCircle_Width()	画带框實心圆形	
84	LT758_DrawEllipse()	画空心椭圆形	
85	LT758_DrawEllipse_Fill()	画實心椭圆形	
86	LT758_DrawEllipse_Width()	画带框實心椭圆形	
87	LT758_DrawSquare()	画空心矩形	
88	LT758_DrawSquare_Fill()	画實心矩形	
89	LT758_DrawSquare_Width()	画带框實心矩形	
90	LT758_DrawCircleSquare()	画空心圓角矩形	
91	LT758_DrawCircleSquare_Fill()	画實心圓角矩形	
92	LT758_DrawCircleSquare_Width()	画带框實心圓角矩形	
93	LT758_DrawTriangle()	画空心三角形	
94	LT758_DrawTriangle_Fill()	画實心三角形	
95	LT758_DrawTriangle_Frame()	画带框實心三角形	
96	LT758_DrawLeftUpCurve()	画左上方曲线	
97	LT758_DrawLeftDownCurve()	画左下方曲线	
98	LT758_DrawRightUpCurve()	画右上方曲线	
99	LT758_DrawRightDownCurve()	画右下方曲线	
101	LT758_DrawLeftUpCurve_Fill()	画左上方 1/4 椭圆	
102	LT758_DrawLeftDownCurve_Fill()	画左下方 1/4 椭圆	
103	LT758_DrawRightUpCurve_Fill()	画右上方 1/4 椭圆	
104	LT758_DrawRightDownCurve_Fill()	画右下方 1/4 椭圆	
106	LT758_DrawQuadrilateral()	画空心四边形	
107	LT758_DrawQuadrilateral_Fill()	画实心四边形	
108	LT758_DrawPentagon()	画空心五边形	

No.	函 数 名 称	功 能	页数
109	LT758_DrawPentagon_Fill()	画实心五边形	
110	unsigned char LT758_DrawCylinder()	画圆柱体	
111	LT758_DrawQuadrangular()	画方柱体	
112	LT758_MakeTable()	画视窗	
BTE 功能			
113	LT758_BTE_MCU_Write_MCU_16bit()	结合光栅操作的 BTE 写入	
114	LT758_BTE_MCU_Write_Chroma_key_MCU_16bit()	结合 Chroma Key 的 MCU 写入	
115	LT758_BTE_MCU_Write_ColorExpansion_MCU_16bit()	结合扩展色彩的 MCU 写入	
116	LT758_BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit()	结合扩展色彩和 Chroma Key 的 MCU 写入	
117	LT758_BTE_Memory_Copy_JPG()	对于 JPG 的 BTE 内存复制	
118	LT758_BTE_Memory_Copy()	结合光栅操作的 BTE 内存复制	
119	LT758_BTE_Memory_Copy_Chroma_key()	结合 Chroma Key 的内存复制	
120	LT758_BTE_Pattern_Fill()	结合光栅操作的图样填满	
121	LT758_BTE_Pattern_Fill_With_Chroma_key()	结合 Chroma Key 的图样填满	
122	LT758_BTE_MCU_Write_MCU8()	结合光栅操作的 BTE 写入	
123	LT758_BTE_MCU_Write_Chroma_key_MCU8()	结合 Chroma Key 的 MCU 写入	
125	LT758_BTE_MCU_Write_ColorExpansion_MCU8()	结合扩展色彩的 MCU 写入	
126	LT758_BTE_MCU_Write_ColorExpansion_Chroma_key_MCU8()	结合扩展色彩和 Chroma Key 的 MCU 写入	
127	LT758_BTE_Memory_Copy_ColorExpansion_8_ColorDepth()	结合扩展色彩的内存复制	
128	LT758_BTE_Memory_Copy_ColorExpansion_Chroma_key_8_ColorDepth()	结合扩展色彩和 Chroma Key 的你内存复制	
130	LT758_BTE_Pixel_Alpha_Blending()	结合扩展色彩和 Chroma Key 的你内存复制	
131	BTE_Alpha_Blending()	结合透明度的内存复制	
132	BTE_Solid_Fill()	区域填满	
显示文字			
133	LT758_Select_Internal_Font_Init()	使用内建字库 – 初始化	
134	LT758_Print_Internal_Font_String()	使用内建字库 – 设定	
135	LT758_Select_Outside_Font_Init()	使用外建字库 – 初始化	
136	LT758_Print_Outside_Font_String()	显示外建字库	

No.	函 数 名 称	功 能	页数
137	LT758_Print_Outside_Ascii_String()	显示外建字库(Ascii)	
138	LT758_Print_Outside_Font_GB2312_48_72()	显示大的外建字库	
139	LT758_Print_Outside_Font_BIG5_48_72()	显示大的外建字库(BIG5)	
显示光标			
140	LT758_Text_cursor_Init()	使用文字光标之前初始化	
141	LT758_Enable_Text_Cursor()	使能文字光标	
142	LT758_Disable_Text_Cursor();	禁止文字光标	
143	LT758_Graphic_cursor_Init()	使用图光标之前初始化	
144	LT758_Set_Graphic_cursor_Pos()	切换光标的位置	
145	LT758_Enable_Graphic_Cursor()	使能图形光标	
146	LT758_Disable_Graphic_Cursor()	禁止图形光标	
PWM 控制			
147	LT758_PWM0_Init()	PWM0 进行初始化	
148	LT758_PWM1_Init()	PWM1 进行初始化	
149	LT758_PWM0_Duty()	改动 PWM0 占空比	
150	LT758_PWM1_Duty()	改动 PWM1 占空比	
串行闪存的 DMA 传输			
151	LT758_DMA_24bit_Linear()	线性模式下的 DMA 传输：24 位寻址	
152	LT758_DMA_32bit_Linear()	线性模式下的 DMA 传输：32 位寻址	
153	LT758_DMA_24bit_Block()	区块模式下的 DMA 传输：24 位寻址	
154	LT758_DMA_32bit_Block()	区块模式下的 DMA 传输：32 位寻址	
电源管理			
155	LT758_Standby()	进入待命模式（Standby）	
156	LT758_Suspend()	进入暂停模式（Suspend）	
157	LT758_SleepMode()	进入休眠模式（Sleep）	
158	LT758_Wkup_Standby()	从待机模式中唤醒	
159	LT758_Wkup_Suspend()	从暂停模式中唤醒	
160	LT758_Wkup_Sleep()	从休眠模式中唤醒	

18. 点屏範例

本章提供两个范例供用户参考，第一个范例是从 MCU 端发送一张图片到显示屏上，第二个范例是从 SPI Flash 读取图片到显示屏上，范例中使用者可以用 V4.30 版（或 V4.30 以上版本）的专用整合软件（UartTFT_V4.30.exe）工具来协助。

18.1 初始化 LT758x

流程如下：

1、初始化单片机与 LT758x 的硬件通讯接口

```
Parallel_Init();
```

2、初始化 LT758x

```
LT758_Init();
```

3、设定显示窗口的颜色深度

```
Select_Main_Window_24bpp();
```

4、设定显示窗口在显存中的起始地址

```
Main_Image_Start_Address(layer1_start_addr);
```

5、设定显示窗口的宽度

```
Main_Image_Width(LCD_XSIZE_TFT);
```

6、设定显示窗口的起始坐标

```
Main_Window_Start_XY(0,0);
```

7、设定底图窗口的起始地址（即写入显存的起始地址，此处设为与显示窗口一致）

```
Canvas_Image_Start_address(layer1_start_addr);
```

8、设定底图窗口的宽度

```
Canvas_image_width(LCD_XSIZE_TFT);
```

9、设定初始工作窗口在底图窗口的坐标

```
Active_Window_XY(0,0);
```

10、设定初始工作窗口的宽高

```
Active_Window_WH(LCD_XSIZE_TFT,LCD_YSIZE_TFT);
```

11、先往图层写入全白背景

```
LT758_DrawSquare_Fill(0,0,LCD_XSIZE_TFT-1,LCD_YSIZE_TFT-1,White);
```

12、打开 RGB 信号输出

```
Display_ON();
```

13、延时 100ms 后，打开背光

```
delay_ms(100);

LT758_PWM1_Init(1,0,99,99,50);
```


18.2 从 MCU 端发送图片到显示屏上

假设 MCU 端想要传送一张如图 18-1 的图片到显示屏上，其流程如下：

图 18-1：宽 80 高 69 的 JPG 图片

1. 通过专用整合软件（UartTFT_V4.30.exe）工具，导入并转成 bin 文件，如图 18-2 所示：

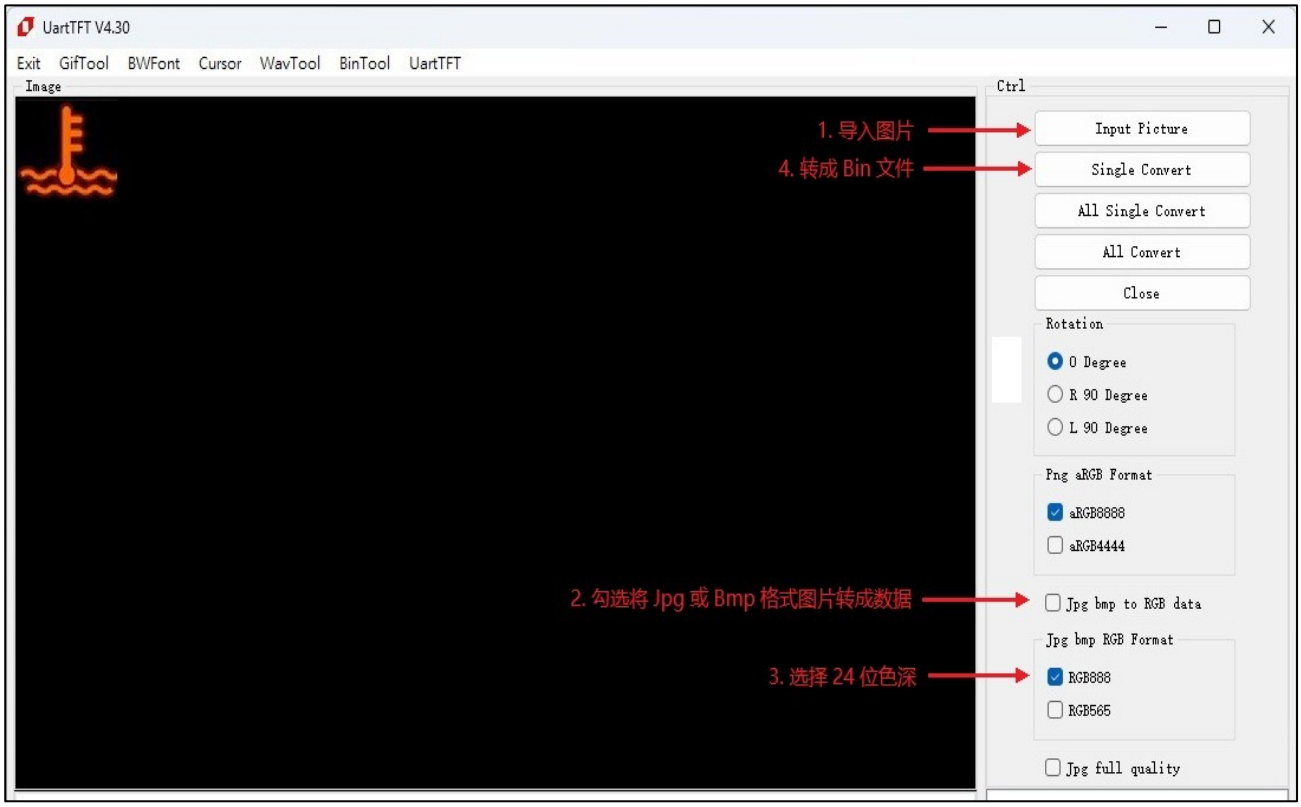


图 18-2：将 JPG 图片转成 bin 文件

2. 得到 bin 文件后，用 WinHex 软件打开 Picture-1.bin 文件，并选择以 C 源码复制，如图 18-3 所示：

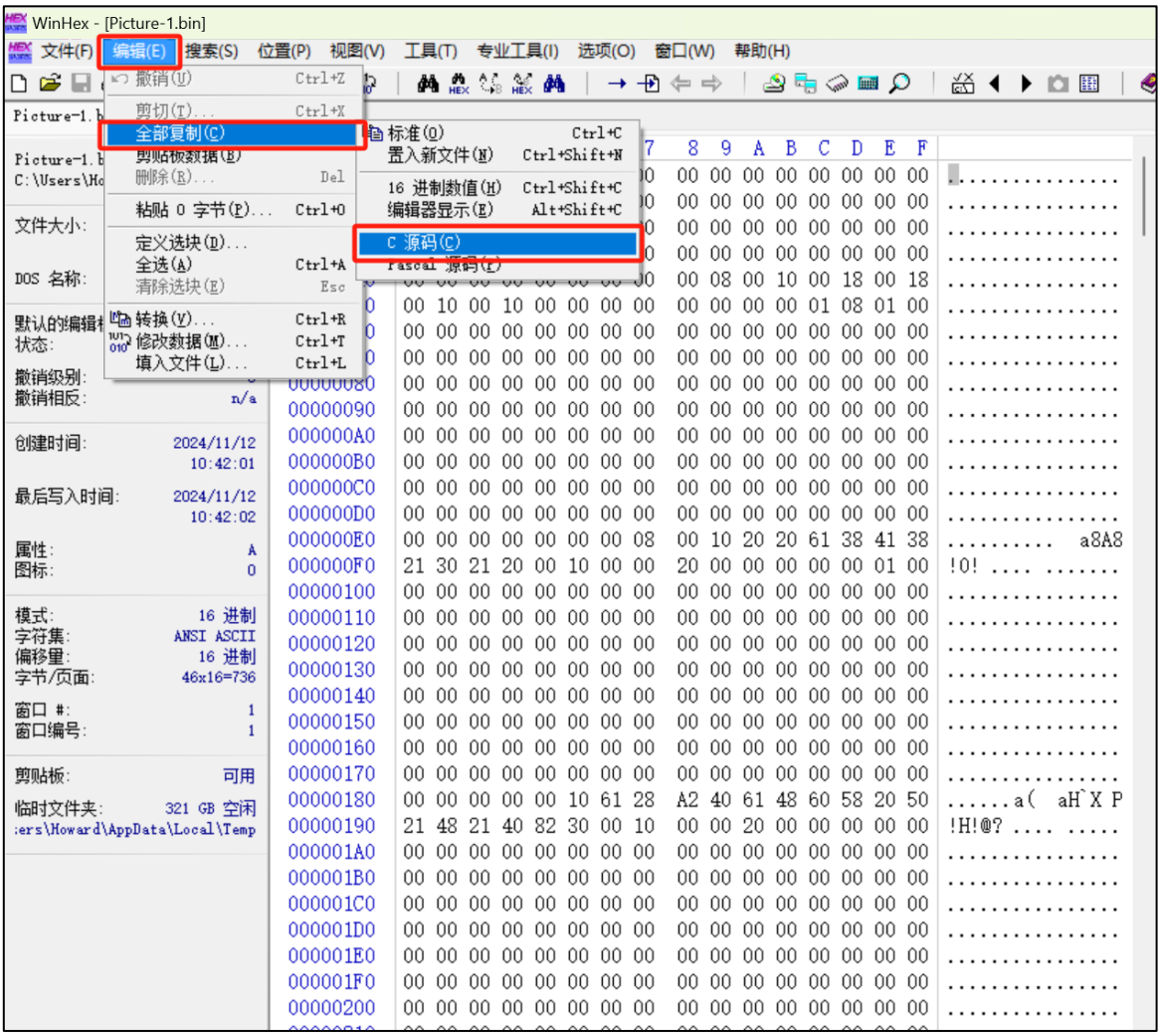


图 18-3：复制图片 C 源码

3. 复制后得到以下数组，粘贴到 txt 文档中，并命名为 pic_data，如图 18-4 所示，可复制到程序中调用：

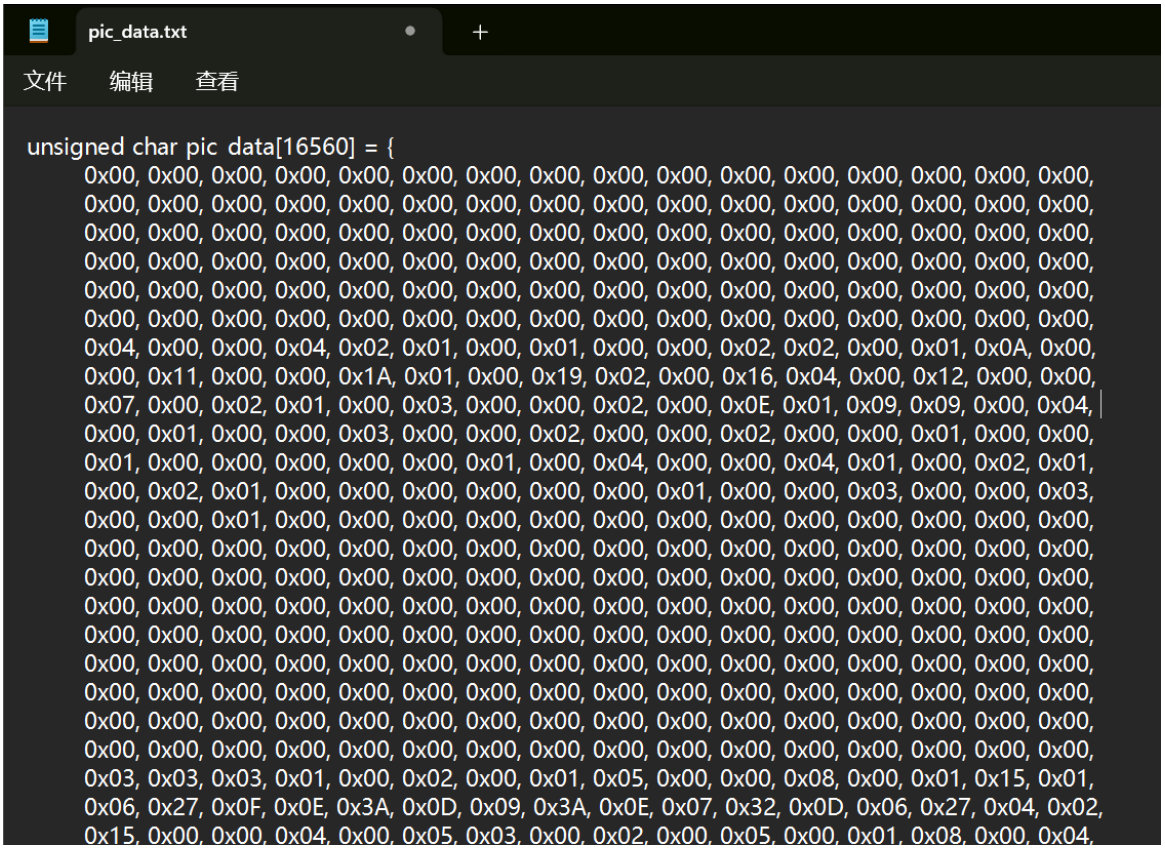


图 18-4：图片信息的数组

程序部分，初始化 LT758x 完成后，通过开窗后描点的方式，把 pic_data 数组的内容发送到 LT758x，执行流程如下：

1、设定工作窗口的坐标（即图片坐标）：

Active_Window_XY(50,50);

2、设定工作窗口的宽高（即图片的宽度和高度）：

Active_Window_WH(80,69);

3、设定描点的起始坐标（与工作窗口保持一致，即从工作窗口的左上角开始描点）：

Goto_Pixel_XY(50,50);

4、写 0x04 寄存器指向显存：

LCD_CmdWrite(0x04);

5、循环发送图片数据：

8080 并口方式（16bits）（24bpp Mode1）：

```
for(i=0;i<16560;i+=6)
{
 temp = (pic_data[i+1]<<8)+pic_data[i+0];
 LCD_DataWrite_Pixel(temp);
 temp = (pic_data[i+3]<<8)+pic_data[i+2];
 LCD_DataWrite_Pixel(temp);
 temp = (pic_data[i+5]<<8)+pic_data[i+4];
 LCD_DataWrite_Pixel(temp);
}
```

SPI 方式（8bits）：

```
for(i=0;i<16560;i++)
{
 LCD_DataWrite(pic_data[i]);
}
```


18.3 从 LT758x 外部的 SPI Flash 读取图片到显示屏上

假设 MCU 端想要传送一张存在 LT758x 外部 SPI Flash 内的如图 18-5 的图片到显示屏上，其流程如下：

图 18-5：宽 68 高 60 的 JPG 图片

1、将此图片通过专用整合软件（UartTFT_V4.30.exe）工具，导入并转成 bin 文件，如图 18-6 所示：

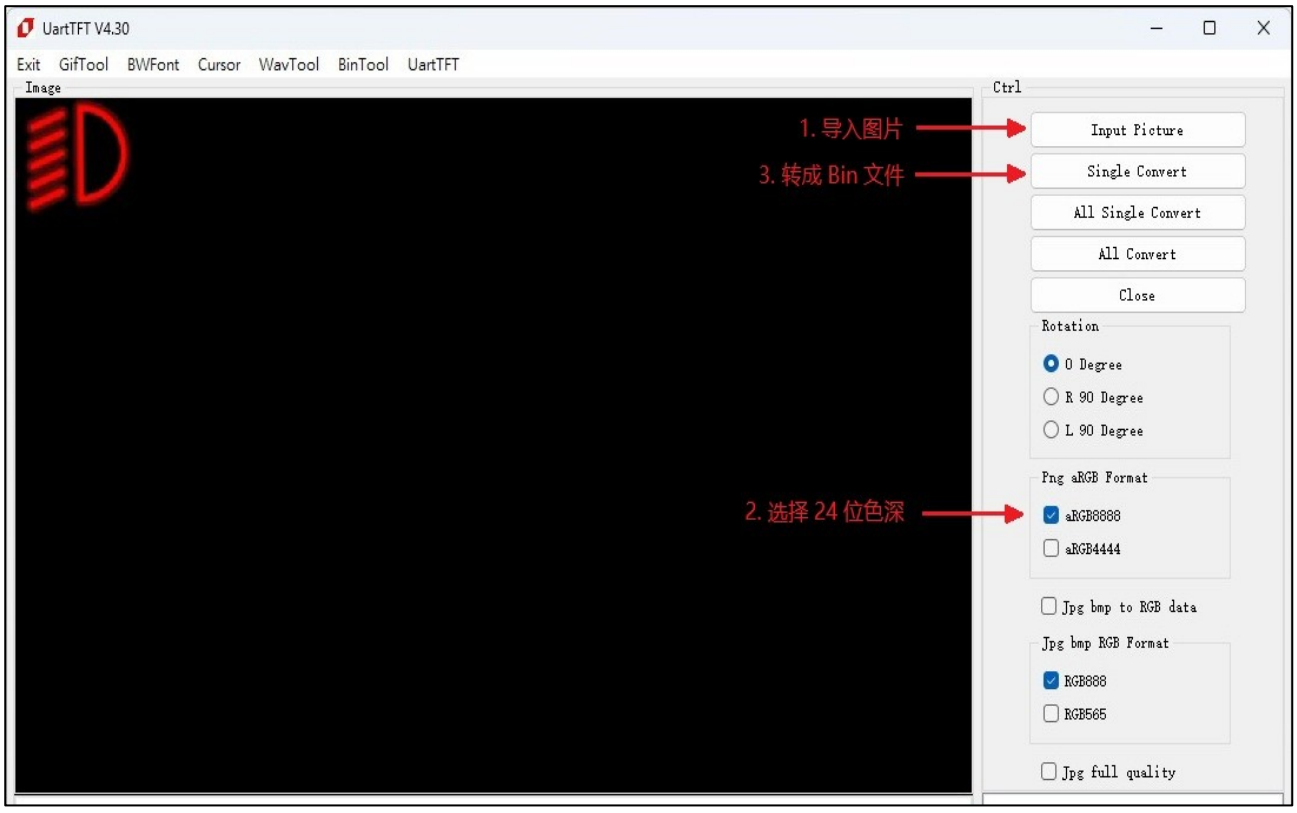


图 18-6：将 JPG 图片转成 bin 文件

2、将转换的 bin 文件命名为 ALL.bin,如图 18-7 所示:

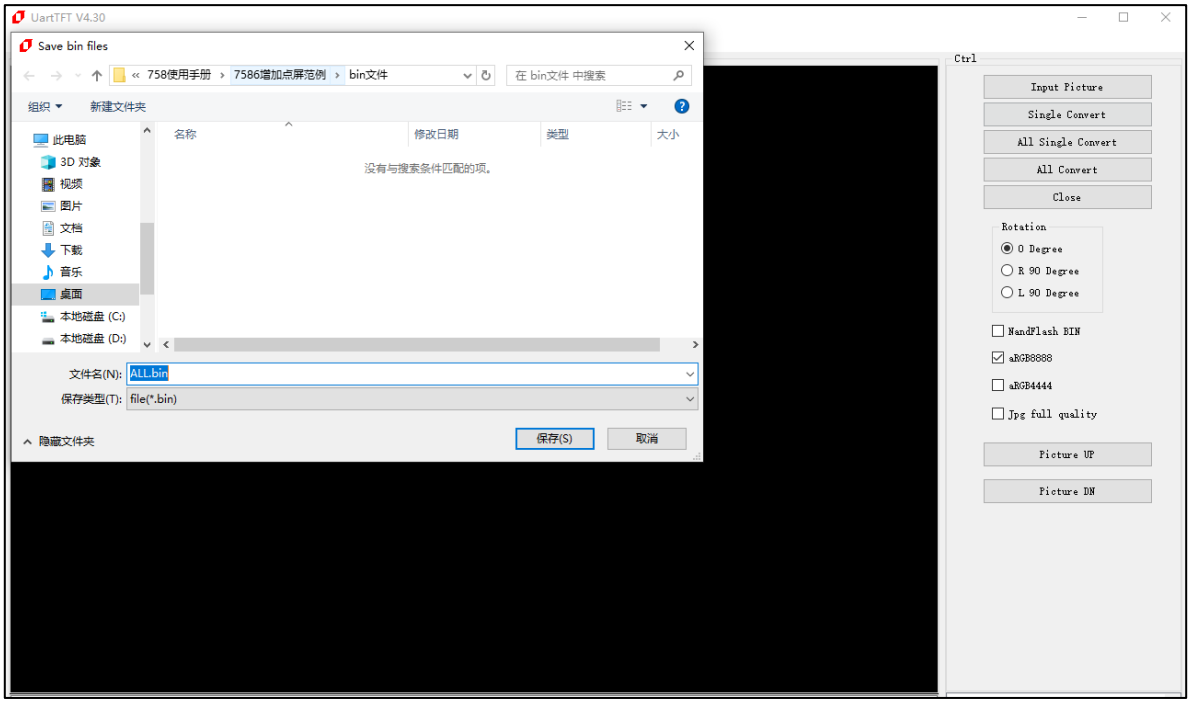


图 18-7：将 bin 文件命名为 ALL.bin

3、在 SD 卡新建文件夹，并命名为 Updata_Flash 如图 18-8 所示:

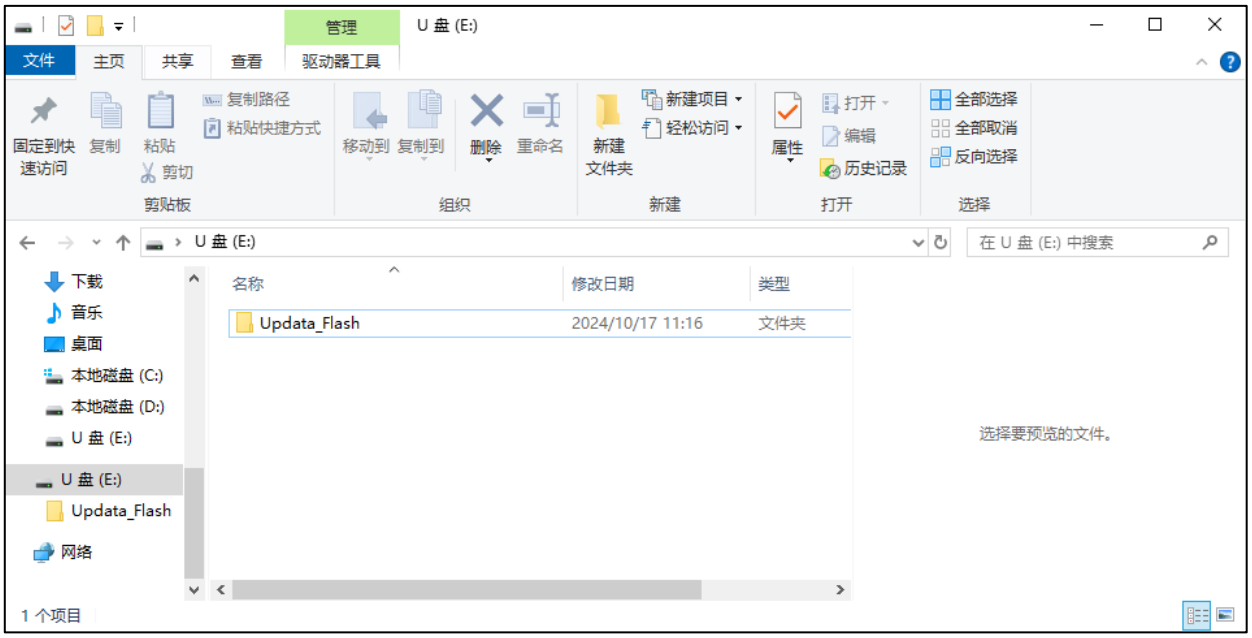


图 18-8：SD 卡文件夹命名

4、在 Updata_Flash 文件夹里面将 ALL.bin 文件放入，如图 18-9 所示

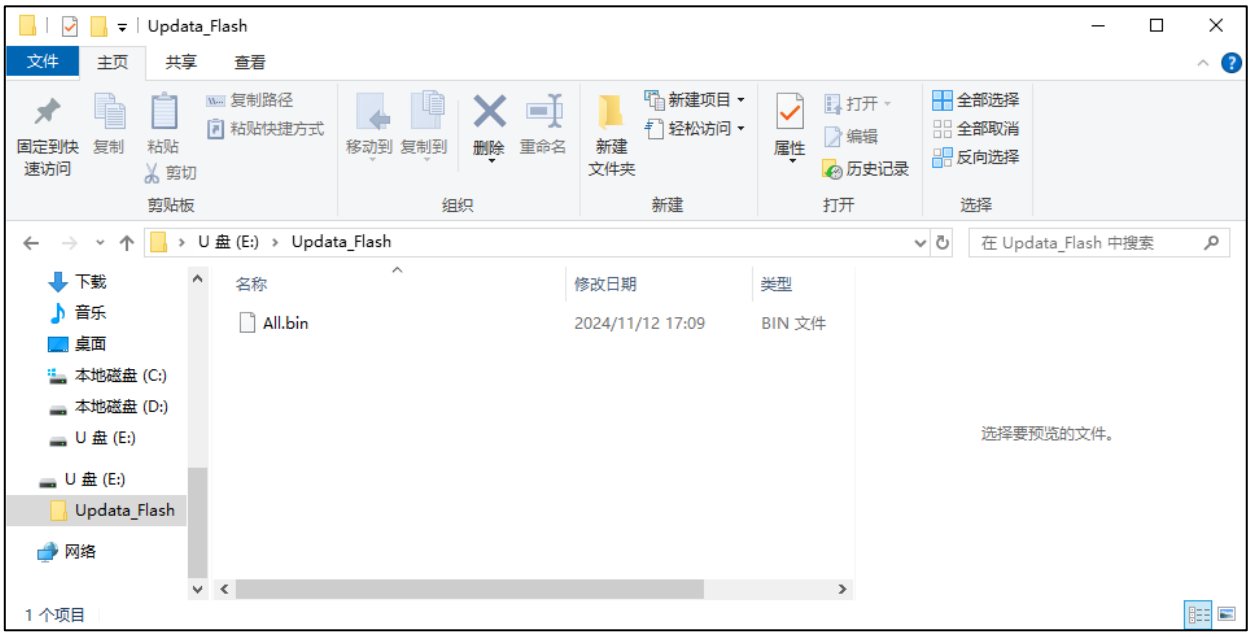


图 18-9：ALL.bin 文件放入 Updata_Flash 文件夹

5、将 SD 卡插入 LT758x 板子上 SD 卡卡槽中，等待 Flash 更新完成。

6、在步骤 3 导出的 bin 文件中会同时导出两个文本文件，在 ALL-StartAddr 文件中可以看到图片的起始地址如图 18-10 所示：

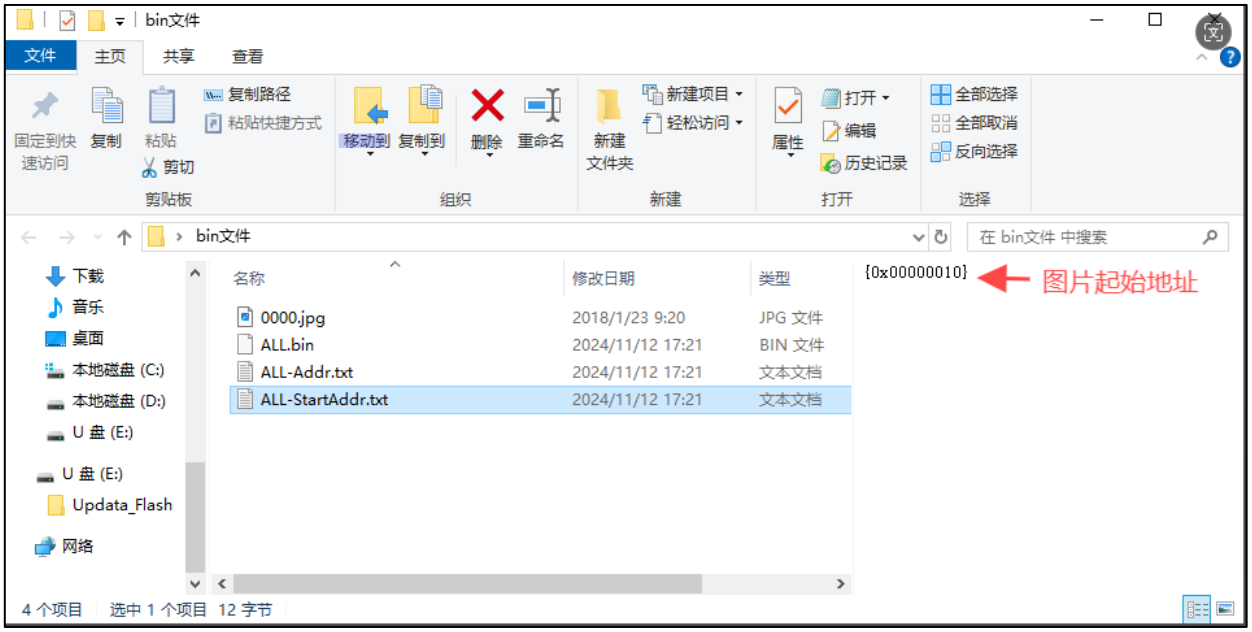


图 18-10：图片起始地址

7、打开 LT758x Demo 程序在 LT758_Function.c 文件将图片的起始地址存入 H_JPG_Addr[]这个数组中如图 18-11 所示：

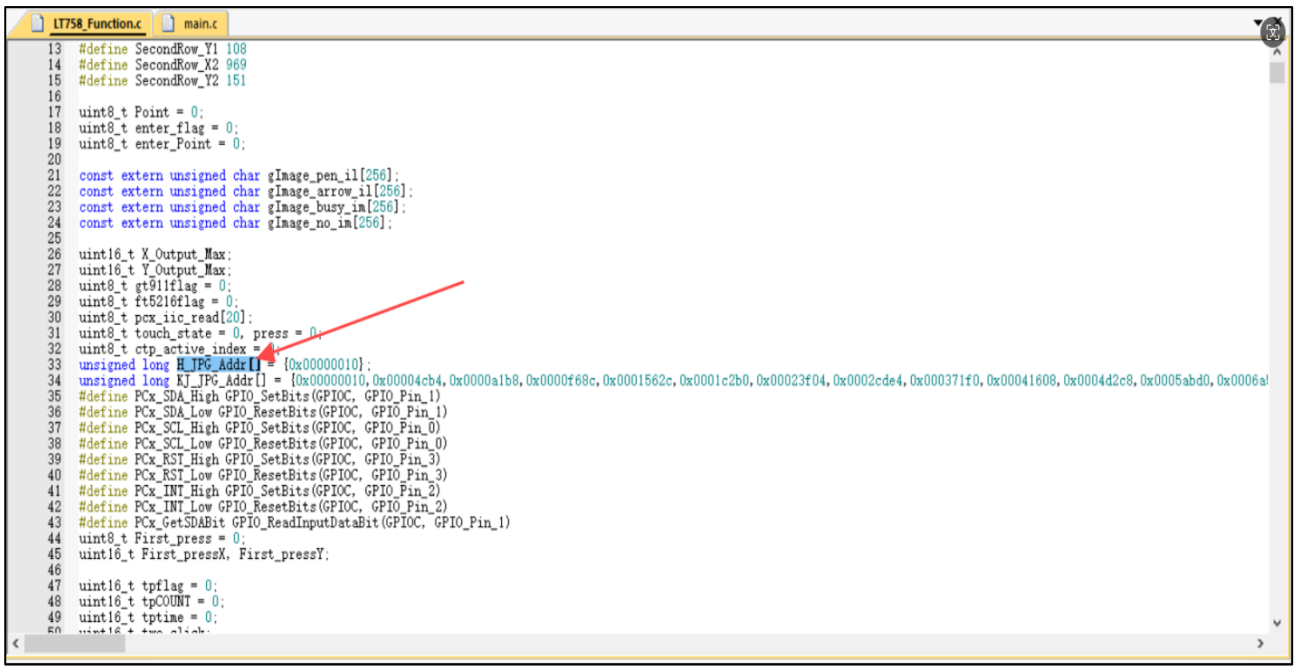


图 18-11：将图片的起始地址存入数组

8、最后在程序中调用以下函数；即可将图片显示出来：

```
Main_Image_Start_Address(0);          // 从显示的 0 地址开始映像到主视窗图层中
Main_Image_Width(1024);               // 设置主视窗的宽度
Main_Window_Start_XY(0,0);            // 主视窗从（0,0）地址开始
Canvas_Image_Start_address(0);        // 从显示内存的 0 地址开始写数据
Canvas_image_width(60);               // 宽度设置
LT758_DMA_JPG(0, 0, H_JPG_Addr[0]);   // 将图片数据从 SPI Flash 通过 MA 传输到显存
                                      // SDRAM
```


19. 点亮 TFT 屏流程及注意事项

19.1 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 LT758x 的 VDD33 电压是否为稳定的 3.3V。（参考 LT758x 引脚图）

3、检查 LT758x 的 VDD12 电压是否为稳定的 1.2V。

19.2 晶振部分

检查 XI 及 XO 所接时钟输入或是晶振（通常为 12MHz）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。

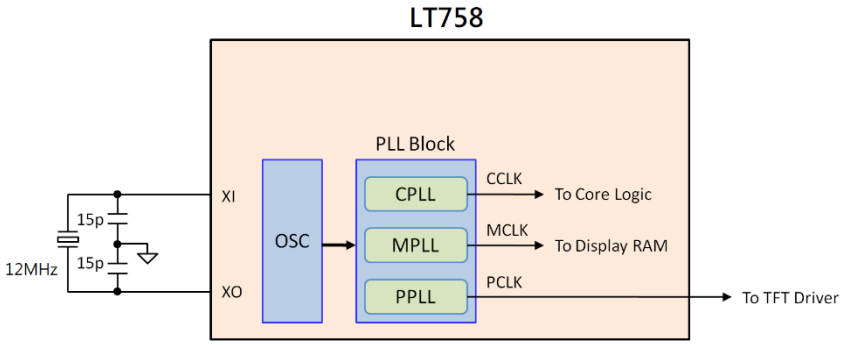


图 19-1：LT758 时钟电路（一）

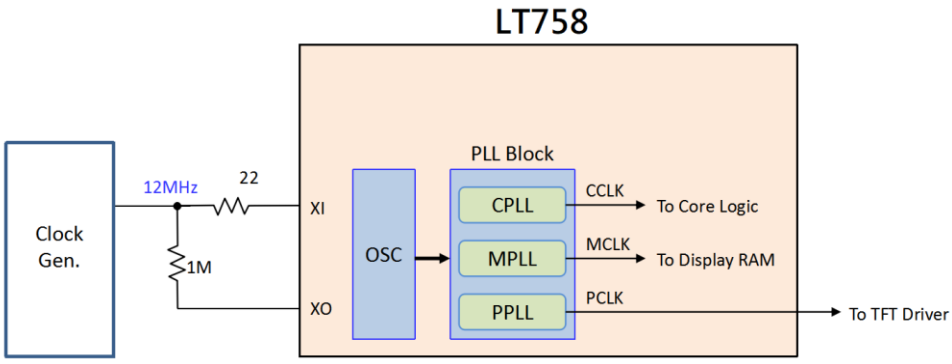


图 19-2：LT758 时钟电路（二）

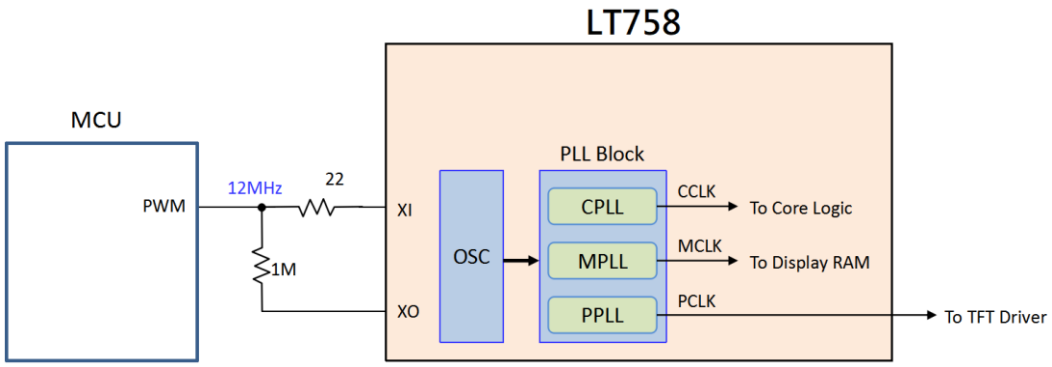


图 19-3：LT758 时钟电路（三）

19.3 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。

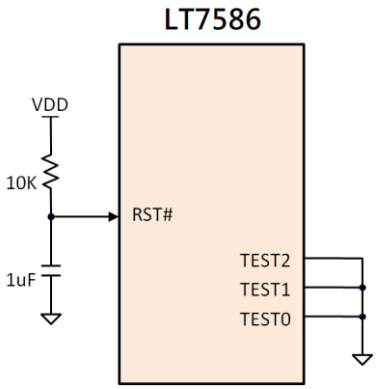


图 19-4：外部复位方式（1）

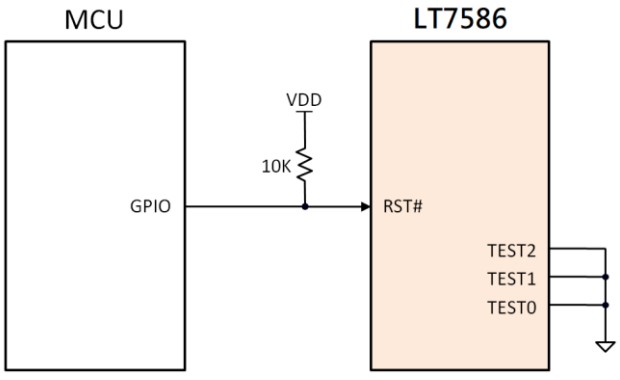


图 19-5：外部复位方式（2）

19.4 测试接脚

确认 TEST[2:0]引脚为接地（如图 19-4、图 19-5）。若使用 LT7580 型号，则忽略此步骤。

19.5 MCU 的接口

1、根据所设计的 MCU 接口模式，检查是否与 LT758x PSM[2:0] 三个引脚的设定一致。

表 19-1：LT7586B 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（支持旁通 By-Pass 模式）
1 1 0	选择串口 I2C 模式（不支持连读模式）
1 1 1	选择串口 I2C 模式（支持连读模式）

2、LT7580 只支持 8 位或 16 位的并口 8080 模式、3 线 SPI 串口及 4 线 SPI 串口模式，其 PSM[1] 引脚已经在 IC 内部接到地。

表 19-2：LT7580 MCU 接口模式设定

PSM[2], PSM[0]		MCU 接口模式
0	X	选择并口 8 位或 16 位的 8080 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

3、MCU 接口到 LT758x 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。

19.6 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 LT758，即复位后，System_Check_Temp 函数是否通过。若没有，则检查 MCU 和 LT758 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 LT758。若中间某一个函数不通过，比如 LT758_SDRAM_initail 函数，则尝试降低 MCU 和 LT758 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 LT758 是否通讯正常。

5、可以透过本公司提供的 Display ON 函数让 LT758 输出 RGB 信号。

19.7 显示部分

1、若以上步骤均正常，且 MCU 的程序已经控制 LT758x 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 LT758x 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 LT75x8 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）。

3、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。

4、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

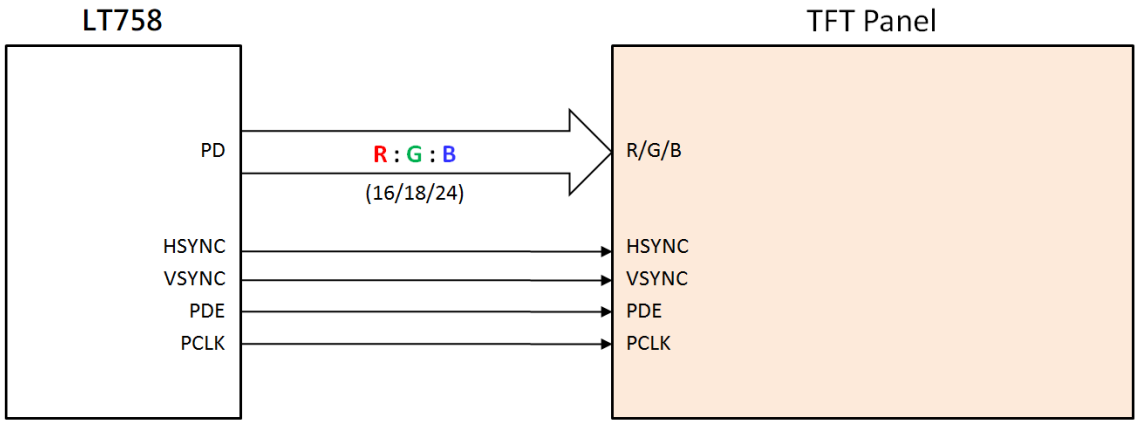


图 19-6：LT758x 与 TFT 屏驱动器的接口

19.8 SPI Flash 部分

1、若与 SPI Flash 的通讯不成功，则先检查 Flash 与 LT758 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 STM32F407(或是 STM32F103) + LT7580/3/6 平台，可以直接使用 SD 卡进行 SPI Flash 烧录。

3、烧录 Flash 所需的 Bin 文件，需使用专用整合软件 UartTFT_Vxxx.exe 生成，该软件具有制作图片、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。

19.9 其他注意事项

1、用户可以参考本应用手册及 LT758x 的规格书，尤其是应用手册说明了 LT758x 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 LT758x 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 STM32Fxx + LT758x 演示板，还有针对 TFT 模块厂将 LT758x 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、S 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 程序让用户能迅速的点亮 TFT 屏和确认软硬件是否正常。相关的 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。

20. LT7586B 演示板

20.1 PCB 接口说明

下图是采用 STM32F407VET6 与 LT7586B 组合成的演示板，可以外接标准 50pin 的 TFT RGB FPC 接口（TFT-1），STM32F407VET6（U4）已经内含 1024*600 的演示程序，LT7586（U3）外接一个 128Mb SPI NOR Flash（U5）或 1Gb SPI NAND Flash 内含有演示程序所需的字库与图片。

SW4 可以用来选择 STM32F407 与 LT7586B 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 STM32F407 的演示程序，然后透过 ST-LINK V2 下载到 STM32F407 内的 Flash；而所需的字库与图片可以参考第 11、14 章的说明产生 Bin 档案，使用者也可以透过我们的辅助工具经由 SD 卡（CARD1）接口直接更新 STM32F407 的程序与 SPI Flash 的数据。

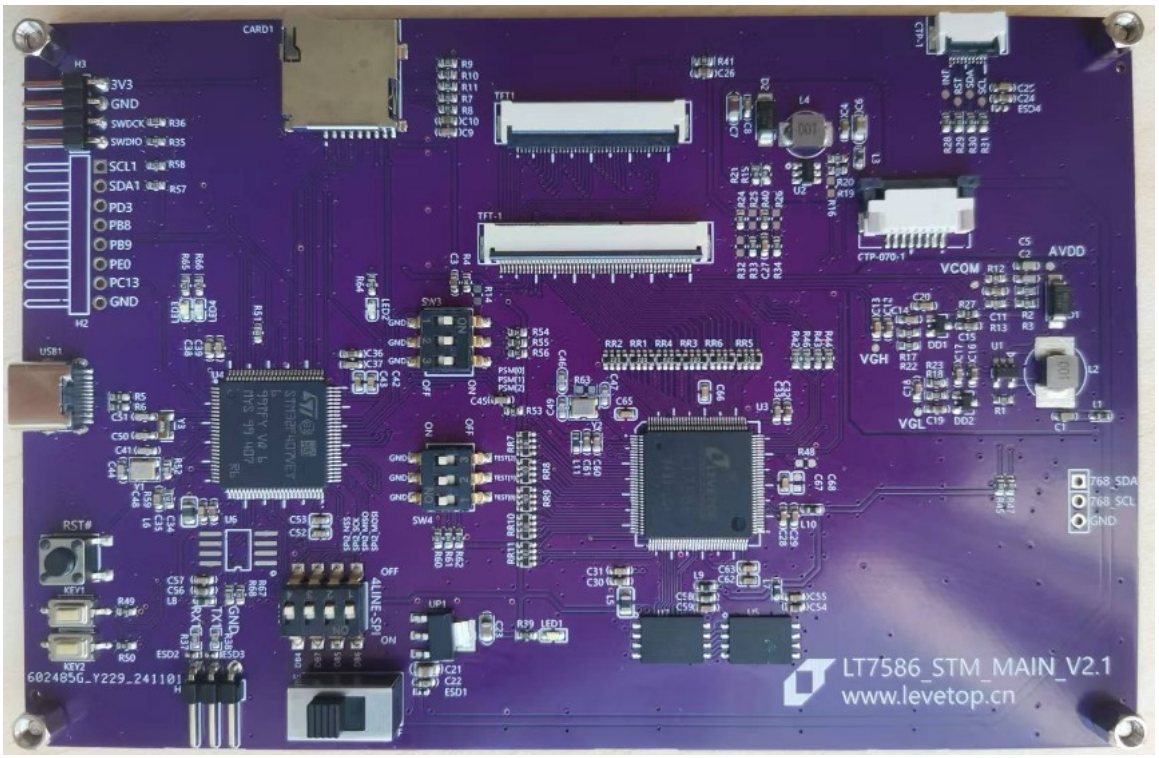


图 20-1：STM32F407+LT7586B 演示板

20.2 演示程序

如需 STM32F407+LT7586B 演示板的演示程序可到本公司官网（http://www.levetop.cn）下载，或与我们业务、FAE 部门联系取得。

20.3 SPI Flash 烧录方式

STM32F407+LT7586 演示板如果要更新 SPI Flash 内的数据（Bin 文件），可用 SD 卡更新，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：

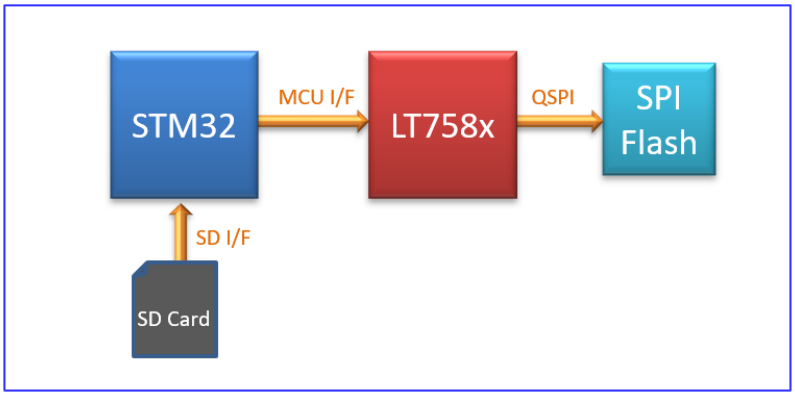


图 20-3：透过 MCU 的 SD 卡更新 SPI Flash 内的数据

提示：有关 Bin 文件的产生请参考第 14.2 ~ 14.5 节。LT758x 提供兼容 4 种形式的 MCU 接口，而内部的寄存器读写就是通过这些 MCU 接口来完成的。

20.4 原理图

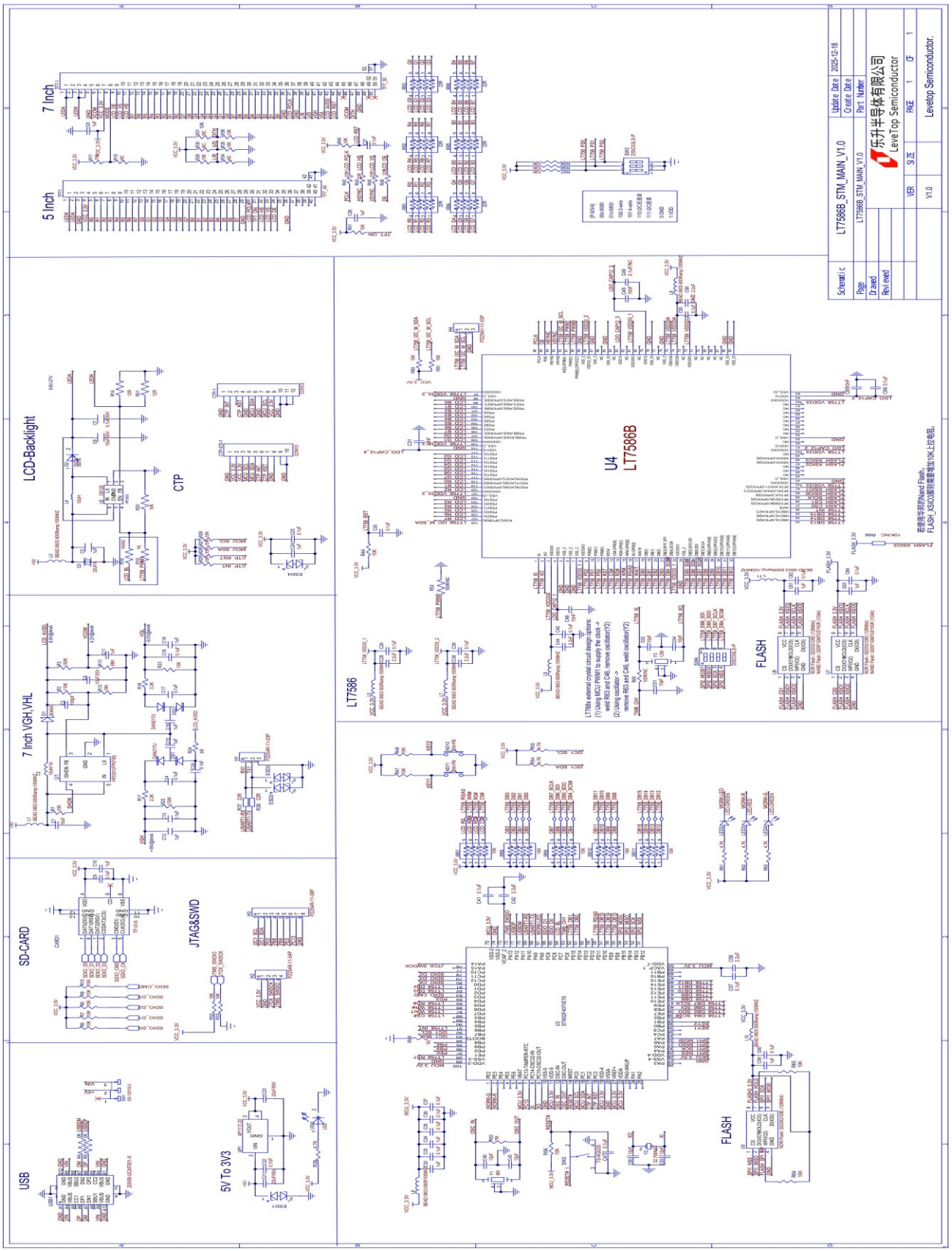


图 20-2：STM32F407+LT7586 演示板原理图

21. LT7580 演示板

21.1 PCB 接口说明

下图是采用 STM32F103VET6 与 LT7580 组合成的演示板，可以外接标准 50pin 的 TFT RGB FPC 接口（TFT-1），STM32F103VET6（U4）已经内含 1024*600 的演示程序，LT7580（U3）外接一个 128MbSPI NOR Flash（U5）或 1Gb SPI NAND Flash 内含有演示程序所需的字库与图片。

SW4 可以用来选择 STM32F103 与 LT7580 之间是使用哪一种接口（8080/6800 并口，SPI 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 ST-LINK V2 下载到 STM32F103 内的 Flash；而所需的字库与图片可以参考第 11、14 章的说明产生 Bin 档案，使用者也可以透过我们的辅助工具经由 SD 卡（CARD1）接口直接更新 STM32F103 的程序与 SPI Flash 的数据。

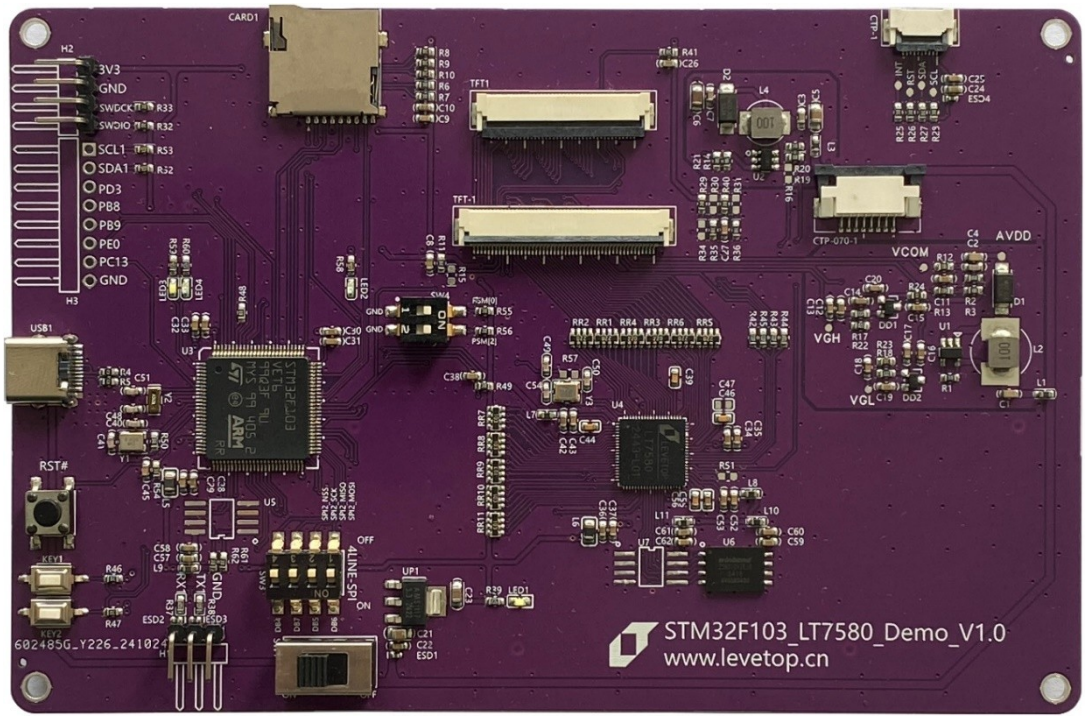


图 21-1：STM32F103+LT7580 演示板

21.2 演示程序

如需 STM32F103+LT7580 演示板的演示程序可到本公司官网下载，或与我们业务、FAE 部门联系取得。

21.3 SPI Flash 烧录方式

STM32F103+LT7580 演示板如果要更新 SPI Flash 内的数据（Bin 文件），可用 SD 卡更新，具体的控制流程如第 20.4 节及图 20-3 所示。

提示：有关 Bin 文件的产生请参考第 14.2 ~ 14.5 节。

21.4 原理图

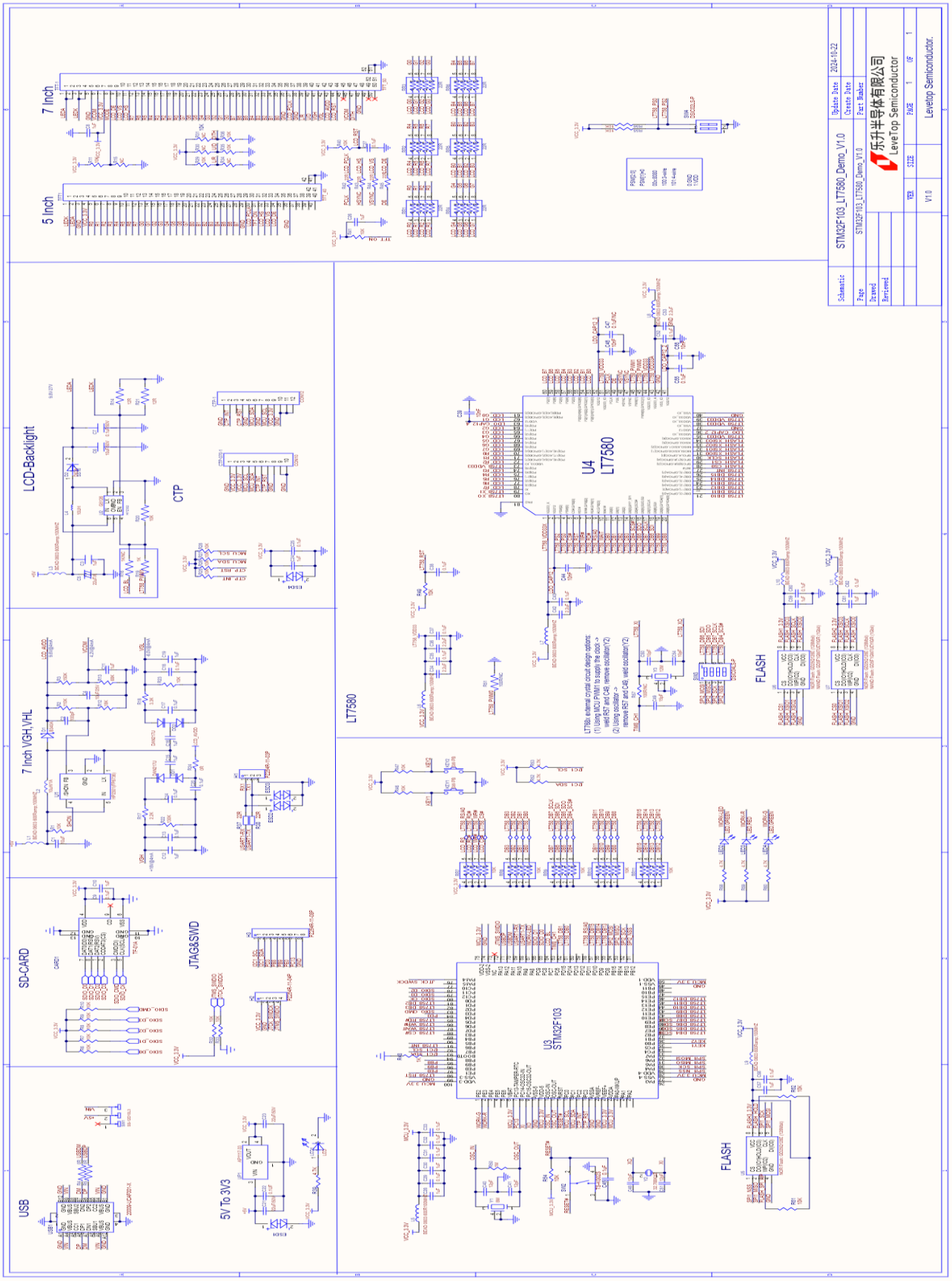


图 21-2：STM32F103+LT7580 演示板原理图