乐升半导体
LEVETOP

LT7688

TFT 串口屏

二次开发使用说明
Application Note

V1.0

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

1. 工程开发环境和烧写

LT7688 内部的 MCU 是用 LT32U02 的 C0 核架构，对应着的编译工具则是 CCore_IDE，如图 1-1，可在我们官网 www.levetop.cn 的下载专区 “LT32 系列-32 位 MCU 芯片”内找到。

图 1-1

如图 1-2 所示，向我们工程师获取到串口屏公版工程，将之导入后，程序修改后，点击“Build Project”即开始编译，编译成功后在工程的指定文件夹下可找到刚生成的一个 Bin 文件。

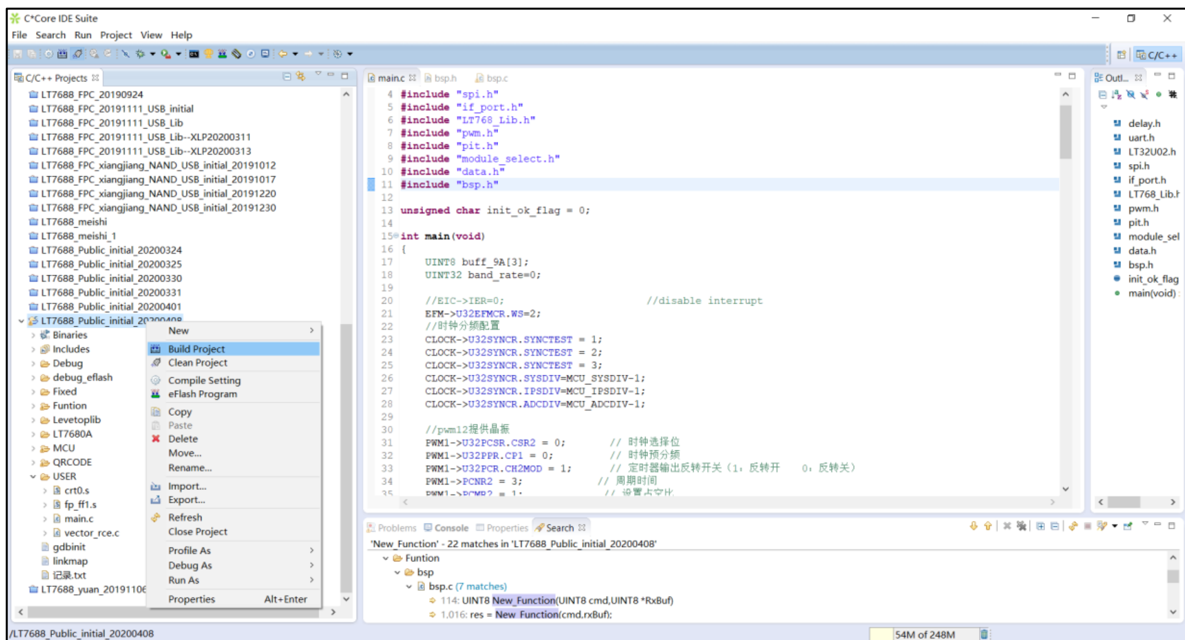

图 1-2

然后打开 LT_VCOM_GUI_2.0，USB 连接成功后将刚刚生成的 bin 文件导进来烧入。

© Levetop Semiconductor Co., Ltd.

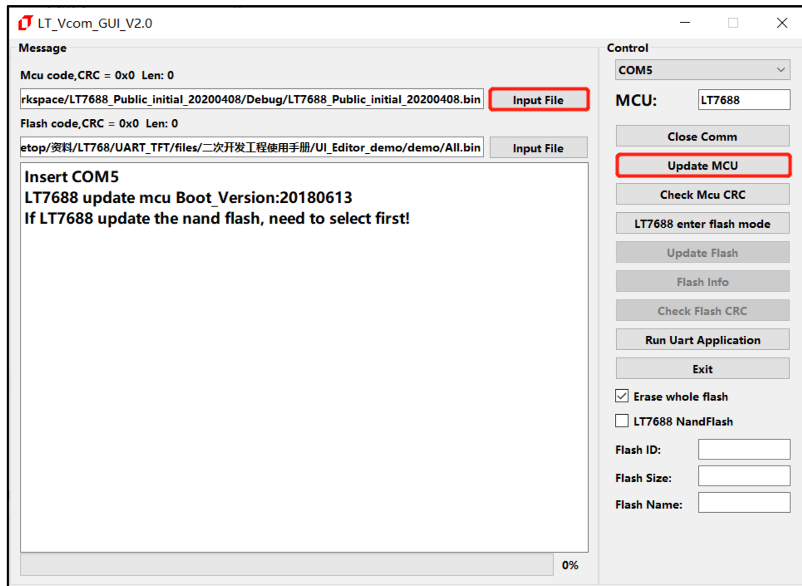

图 1-3

如图 1-4，识别到 Flash 型号后可将相应的的 bin 文件烧入到 Flash 中。

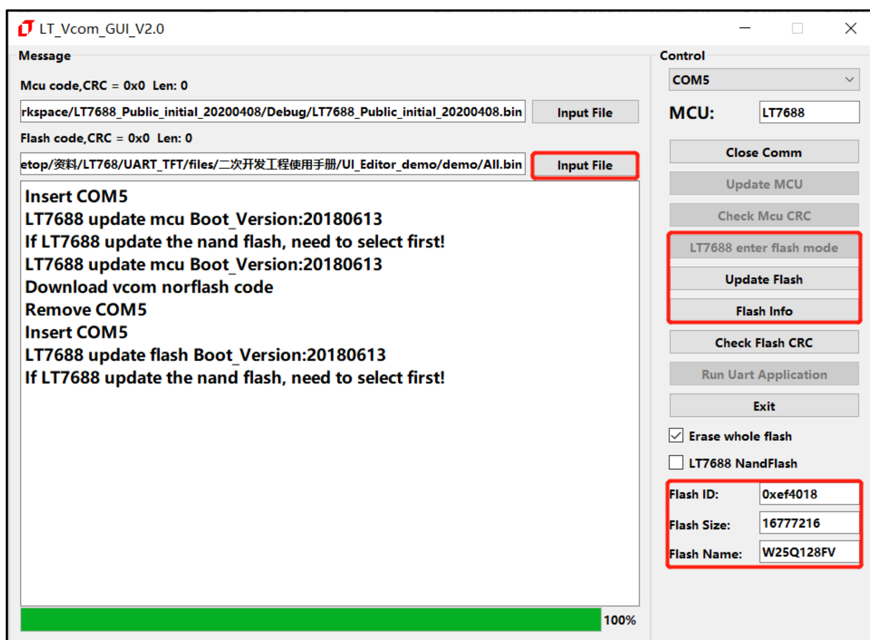

图 1-4

CCore_IDE 和 LT_VCOM_GUI_2.0 的具体使用方法可在我们官网上查阅相关手册或咨询我们的工程师，后面的章节会讲到如何二次开发和如何整合公版和客制化的资源。

© Levetop Semiconductor Co., Ltd.

2. 工程架构的了解与熟悉

2.1 串口指令的解析与执行

main.c 下的 LT_ReceiveCmd(gUsartRx.Buf) 是实时等待依次处理上位机发送过来的串口指令，符合帧头帧尾格式（0xAA 0xE4 0x1B 0x11 0xEE）的指令，会筛选进入到下一步 CRC 校验处理，校验无误后进入到 bsp.c 的 LT_AnalyzeCmdInfo(&buf[1])开始执行处理相应的指令，80h—FFh 指令是公版所使用到的指令 New_Function(cmd,rxBuf)就是对此部分指令做的解析和执行，这部分代码不建议私自大改，如有改动需求建议咨询我们工程师，每条指令对应的功能描述可参考“LT7688_UartTFT_APNote_V20_CH”文档。另外地我们留了一个区域来给客户进行二次开发，60h—7Fh 指令是客户可自定义的指令，通过 User_Function(cmd,rxBuf) 来处理自定义的指令。以 60h 指令举例，LT_ManageCmd_60(UINT8 *rxBuf)下可加入客户想加入的代码程序，一些简单的静态的显示功能可直接在此函数下完成，动态的显示功能后面部分会有说明。如图 2-1 所示，该 60h 指令的功能是执行完后向上位机反馈了相关的串口指令信息。

```
#if   MODULE_60
UINT8 LT_ManageCmd_60(UINT8 *rxBuf)
{
    UINT8 buf[3];

    if((gUsartRx.Count==9)||(cmd_flag== 1))
    {
        buf[0] = rxBuf[0];
        buf[1] = rxBuf[1];
        buf[2] = 0x00;
        LT_DataToPc(buf,3);
    }

    return OK;
}
#endif
```

图 2-1

© Levetop Semiconductor Co., Ltd.

2.2 动态效果的实现

如需加入我们串口屏公版方案没有的动态功能，请遵循以下的方式实现。以下以 81h 指令(循环显示图片)和 A0h 指令(Button 功能)作为例子。

在 2.1 已经讲解过串口指令的解析工作是由 main.c 下的 LT_ReceiveCmd(gUsartRx.Buf)来完成的，那么简单的显示一张图片是可以直接在解析完指令后就能够去执行，那么如果要实现动态的图片切换则要使用定时器来和主程序和一起配合来完成，如图 2-2、2-3 和 2-4 所示，main.c 的 TurnForm( ) 实际是进行图片切换的函数，bsp.c 的 LT_ManageCmd_81(UINT8 *rxBuf)成功解析指令后，打开定时器标志位 gOpen81，然后在 main.c 的 while 循环中有序地实现图片循环切换。

如 2-2、2-5 和 2-6 所示，bsp.c 的 LT_ManageCmd_A0(UINT8 *rxBuf)成功解析指令后，会打开 ControlFlag 标志位，剩下的触摸的判断与逻辑的工作就交由 main.c 的 button( )来执行，触摸相关的功能需要结合触摸读取函数 gTpInfo.scan( )和触摸状态 gTpInfo.sta 来编写程序。

因此要注意，TurnForm( )和 button( )这些在 main.c 的 while 循环中执行的函数，不要在程序里加入死循环或很长的延时，这样会影响到公版的串口指令解析和其他动态功能的运行。

```
181         if(gUsartRx.Flag)                   LT_ReceiveCmd(gUsartRx.Buf);
182
183 #if    MODULE_81
184         if(gTurnFlag)                               TurnForm();
185 #endif
186
187 #if    MODULE_86_87
188         if(gTPFlag)                                 AnalyzeTP();
189 #endif
190
191 #if    MODULE_88
192         if(gGifFlag)                                TurnGif();
193 #endif
194
195 #if    MODULE_B8_B9
196         if(gWavFlag)                            LT_PlayWav();
197 #endif
198
199 #if    MODULE_D8
200         if(rool_one_flag)                       TurnRoll_One();
201 #endif
202
203 #if    MODULE_D9
204         if(RollFlag)                            TurnPicture();
205 #endif
206
207 #if    MODULE_B4
208         if(gOpenGesture)                        TpGesture();
209 #endif
210
211 #if    MODULE_A0_A1
212         if(ControlFlag)                         button();
213 #endif
```

图 2-2

© Levetop Semiconductor Co., Ltd.

```
1295 UINT8 LT_ManageCmd_81(UINT8 *rxBuf)
1296 {
1297     UINT8 i = 0,j = 0,res = 0;
1298     UINT8 sum,t,oper,flag,pic[10] = {0};
1299     UINT16 canvas_w,x,y;
1300     if((gUsartRx.Count==9)||(cmd_flag== 1))
1301     {
1302             if(rxBuf[1] & 0x80)
1308             if(gOpen81 == 0)            gTurnCount = 0;
1309             res = Get_81_info1(rxBuf,&sum,&t,&flag,&x,&y,pic,&oper);
1310             if(res!=OK) return res;
1311             for(i = 0 ; i < gTurnCount ; i++)
1323             gTurnInfo[i].operation = oper;
1324             gTurnInfo[i].sum = sum;
1325             gTurnInfo[i].t = t;
1326             gTurnInfo[i].x = x;
1327             gTurnInfo[i].y = y;
1328             gTurnInfo[i].w = gPictureInfo.w;
1329             gTurnInfo[i].h = gPictureInfo.h;
1330             gTurnInfo[i].flag = flag;
1331             for(j = 0 ; j < gTurnInfo[i].sum ; j++)
1335             canvas_w = Get_Canvas_Width(gPictureInfo.w);
1336             Canvas_Image_Start_address(LAY_BUFF1);
1337             Canvas_image_width(canvas_w);
1338             LT768_DMA_24bit_Block_Or_Line_16bpp(SPI_X,CLK_DIV,0,0,
1339                                     gPictureInfo.w,gPictureInfo.h,
1340                                     gPictureInfo.w,gPictureInfo.addr
1341                                    ,LAY_BUFF1,canvas_w
1342                                     );
1343             if(gTurnInfo[i].flag == 0)
1351             else if(gTurnInfo[i].flag == 1)
1357             else if(gTurnInfo[i].flag == 2)
1365
1366             gOpen81 = 1;
1367             gTurnCount++;
```

图 2-3

```
26 void PIT3_Handler(void)
27 {
28 #if   MODULE_81
29             if(gOpen81==1)
30             {
31                 gTurnFlag = 1;
32             }
33 #endif
34 #if   MODULE_86_87
35             if(gOpenTP==1)
36             {
37                     gTPFlag = 1;
38             }
39 #endif
40 #if   MODULE_88
41             if(gOpen88==1)
42             {
43                     gGifFlag = 1;
44             }
45 #endif
46 #if MODULE_D8
47             if(gOpenD8==1)
48             {
49                     rool_one_flag = 1;
50             }
51 #endif
52 #if  MODULE_D9
53             if(gOpenD9==1)
54             {
55                     RollFlag = 1;
56             }
57 #endif
58     PIT3->U16PCSR.PIF =1;       // 4.clear PIF interrupt flag
59 }
```

图 2-4

© Levetop Semiconductor Co., Ltd.

```
3613 UINT8 LT_ManageCmd_A0(UINT8 *rxBuf)
3614 {
3615     UINT8 j;
3616     UINT16 i = 0;
3617     UINT8 oper = 0,flag = 0;
3618     UINT8 buf1[8] = {0},buf2[8] = {0};
3619     UINT16 x,y,canvas_w;
3620     UINT8 res = 0;
3621     if((gUsartRx.Count==9)||(cmd_flag== 1))
3622     {
3623         res = Get_A0_info(rxBuf,&oper,&flag,buf1,buf2,&x,&y);
3624         if(res!=OK) return res;
3625 
3626         for(i = 0 ; i < ControlCount ; i++)
3639         controlInfo[i].operation    = oper;
3640         controlInfo[i].x = x;
3641         controlInfo[i].y = y;
3642         controlInfo[i].flag = flag;
3643         for(j = 0 ;j < 8;j++)
3648         controlInfo[i].addr = gPictureInfo.addr;
3649         controlInfo[i].h = gPictureInfo.h;
3650         controlInfo[i].w = gPictureInfo.w;
3651 
3652         canvas_w = Get_Canvas_Width(gPictureInfo.w);
3653         Canvas_Image_Start_address(LAY_BUFF1);
3654         Canvas_image_width(canvas_w);
3655         LT768_DMA_24bit_Block_Or_Line_16bpp(SPI_X,CLK_DIV,0,0,
3656                                   gPictureInfo.w,gPictureInfo.h,
3657                                   gPictureInfo.w,gPictureInfo.addr
3658                                                       ,LAY_BUFF1,canvas_w
3659                                  );
3660         switch(flag)
3680         ControlCount++;
3681         if(ControlCount > CONTROL_SIZE)          ControlCount =  CONTROL_SIZE;
3682         ControlFlag = 1;
3683         return OK;
```

图 2-5

```
3735 void button(void)
3736 {
3737     UINT8 i,j;
3738     UINT8 buff[3];
3739     UINT16 canvas_w;
3740     gTpInfo.scan();
3741 
3742     if(gTpInfo.sta)
3743     {
3744         for(i = 0;i < ControlCount;i++)
3745         {
3746             if(gTpInfo.x[0] >= controlInfo[i].x && gTpInfo.x[0] <= controlInfo[i].x + controlInfo[i].w
3747             {//按下控件时
3748                 canvas_w = Get_Canvas_Width(controlInfo[i].w);
3749                 Canvas_Image_Start_address(LAY_BUFF1);
3750                 Canvas_image_width(canvas_w);
3751                 LT768_DMA_24bit_Block_Or_Line_16bpp(SPI_X,CLK_DIV,0,0,
3752                                            controlInfo[i].w,controlInfo[i].h,
3753                                            controlInfo[i].w,controlInfo[i].addr
3754                                            ,LAY_BUFF1,canvas_w
3755                                         );
```

图 2-6

© Levetop Semiconductor Co., Ltd.

2.3 流程图

以 81h 指令(循环显示图片)和 A0h 指令(Button 功能)作为例子。图 2-7 和图 2-8 分别为 81h 指令和 A0h 指令程序执行的流程图。

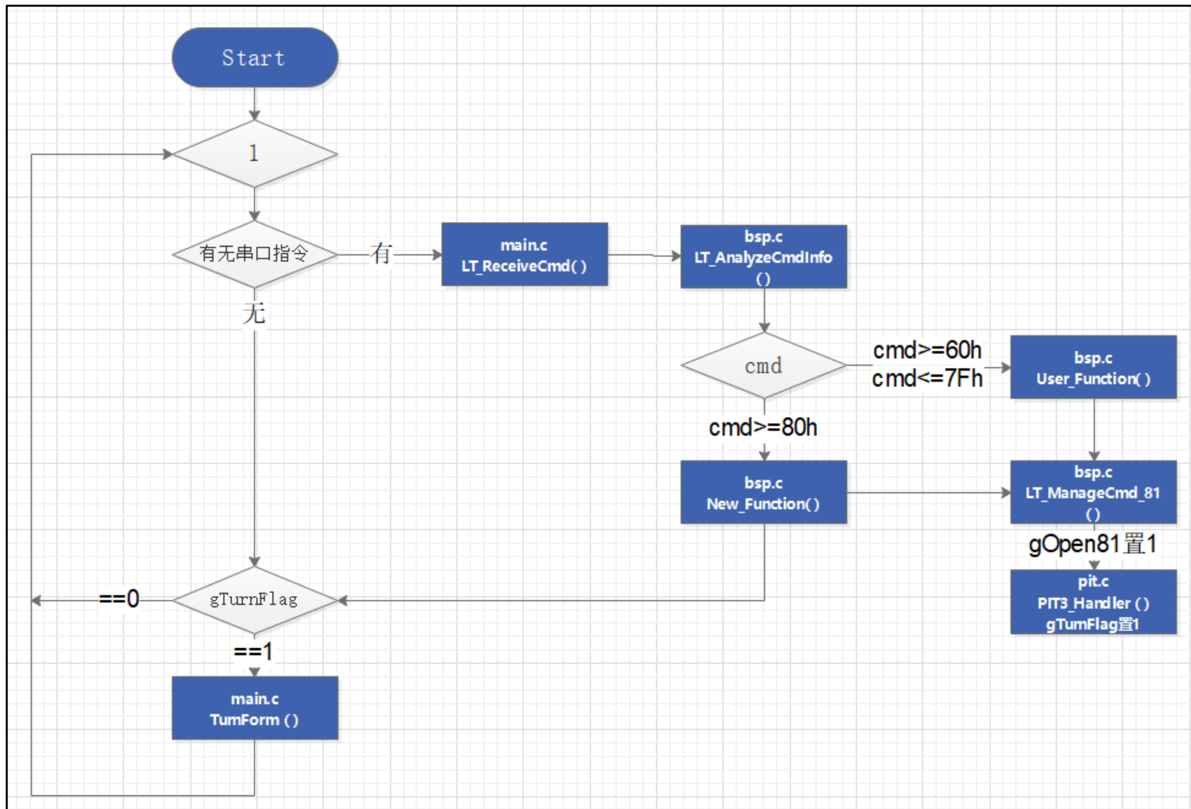

图 2-7

© Levetop Semiconductor Co., Ltd.

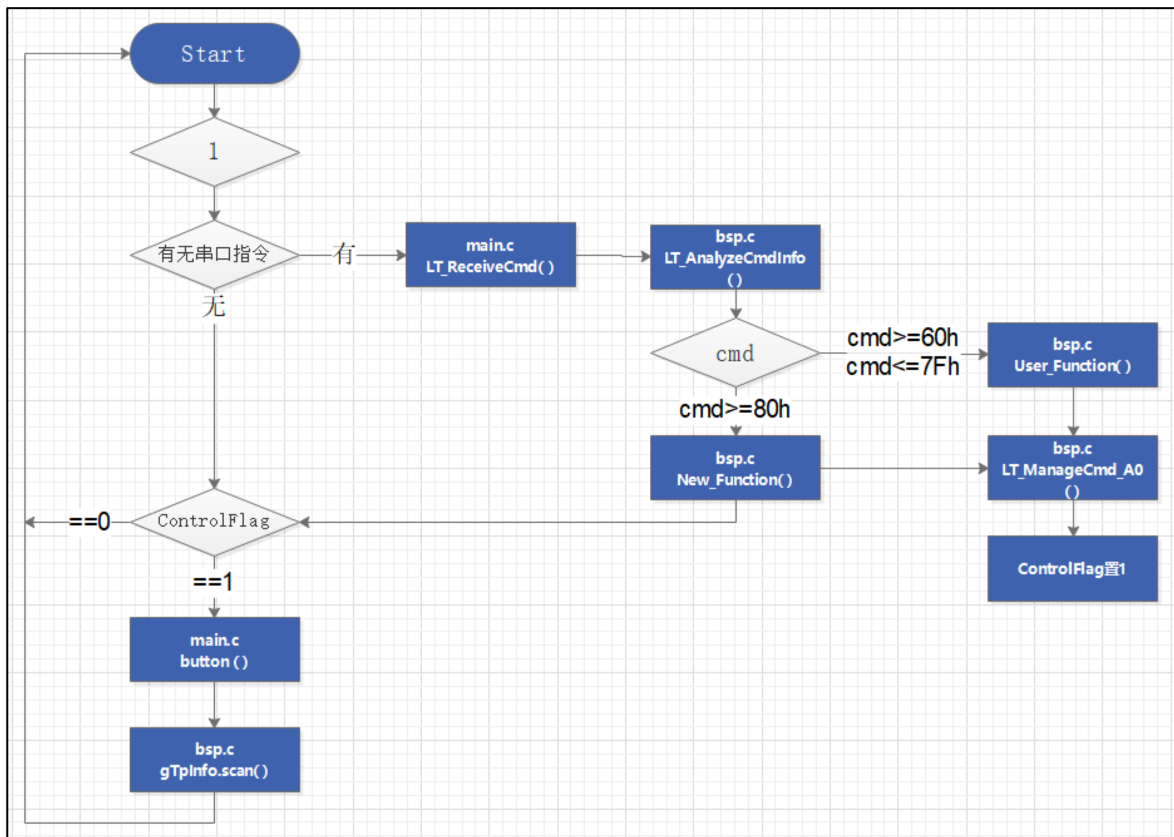

图 2-8

© Levetop Semiconductor Co., Ltd.

3. 工程资源的划分与使用

因为 7688 内部资源是有限的，编程部分的 RAM 是 7.5KBytes，ROM 是 63KBytes，显示驱动部分的 SRAM 是 16Mbytes，所以在二次开发的时候需要注意各资源的使用情况，以保证串口屏的工程能够正常的运行。

3.1 编程部分(RAM 和 ROM)

如图 3-1 所示，公版版工程的程序打开了大部分的功能，图中的宏定义选择，客户可根据实际需求来选择是否打开相应的指令功能。从图 3-2 中可以看到，以目前打开的大部分功能来看， 成功编译后，工程的 ROM，即程序量已经使用了 60064 Bytes (text)，RAM 使用了 6284 Bytes (data+bss)，因此二次开发要注意的是，新定义的函数里声明了局部变量不可过大，以目前工程举例的情况来说，局部变量定义的大小不可超过（7.5K - 6284)，否则会导致程序出错。所以有些指令功能用不到的话，建议是选择不打开。

```
main.c  bsp.h  bsp.c  bsp_user.c  bsp_user.h  module_select.h
1  #ifndef _module_select_h
2  #define _module_select_h
3  #include "LT32U02.h"
4
5  #define        R_TOUCH_FLAG       0        //1选择电阻屏，0不选电阻屏
6  #define        FT_TOUCH_FLAG      1     //1选择电容屏，0不选电容屏
7
8  #define        CRC_FLAG           1        //1选择CRC，0不选CRC
9  //#define      FPC                0
10
11 #define      MODULE_80          1        //1选择80指令，0不选80指令
12 #define      MODULE_81          1        //1选择81指令，0不选81指令
13 #define      MODULE_82          1        //1选择82指令，0不选82指令
14 #define      MODULE_84          1        //1选择81指令，0不选84指令
15 #define      MODULE_85          0        //1选择85指令，0不选85指令
16 #define      MODULE_86_87       0        //1选择86/87指令，0不选86/87指令
17 #define      MODULE_88          1        //1选择88指令，0不选88指令
18 #define      MODULE_89          1
19 #define      MODULE_8A          1        //1选择8A指令，0不选8A指令
20 #define      MODULE_8B          0        //1选择8B指令，0不选8B指令
21 #define      MODULE_8C_8D       0        //1选择8C/8D指令，0不选8C/8D指令
22 #define      MODULE_8E          1        //1选择8E指令，0不选8E指令
23 #define      MODULE_8F          1        //1选择8F指令，0不选8F指令
24 #define      MODULE_90          1        //1选择90指令，0不选90指令
25 #define      MODULE_91          1        //1选择91指令，0不选91指令
26 #define      MODULE_94_95       1        //1选择94指令，0不选94指令
27 #define      MODULE_98          0        //1选择98指令，0不选98指令
28 #define      MODULE_9A          1        //1选择9A指令，0不选9A指令
29 #define      MODULE_9B_9C       0        //1选择9B指令，0不选9B指令
30 #define      MODULE_A0_A1       1        //1选择A0指令，0不选A0指令
31 #define      MODULE_A2_A3       1        //1选择A2指令，0不选A2指令
32 #define      MODULE_B0          1        //1选择B0指令，0不选B0指令
33 #define      MODULE_B1          1        //1选择B1指令，0不选B1指令
34 #define      MODULE_B2          0        //1选择B1指令，0不选B1指令
```

图 3-1

```
Problems  Console  Properties  Search
CDT Build Console [LT7688_Public_initial_20200330]
make --no-print-directory post-build
mcore-elf-objcopy  -O binary LT7688_Public_initial_20200330.elf LT7688_Public_initial_20200330.bin& mcore-elf-objdump -D LT7688_Pu
   text     data      bss      dec      hex filename
  60064      368     5916    66348     1032c LT7688_Public_initial_20200330.elf
```

图 3-2

© Levetop Semiconductor Co., Ltd.

之前在第一节说过，我们留了个区域 User_Function(cmd,rxBuf)给客户自行添加指令，图 3-3 所示，建议客户是在 bsp_user.c 和 bsp_user.h 文件中添加自定义的功能函数。

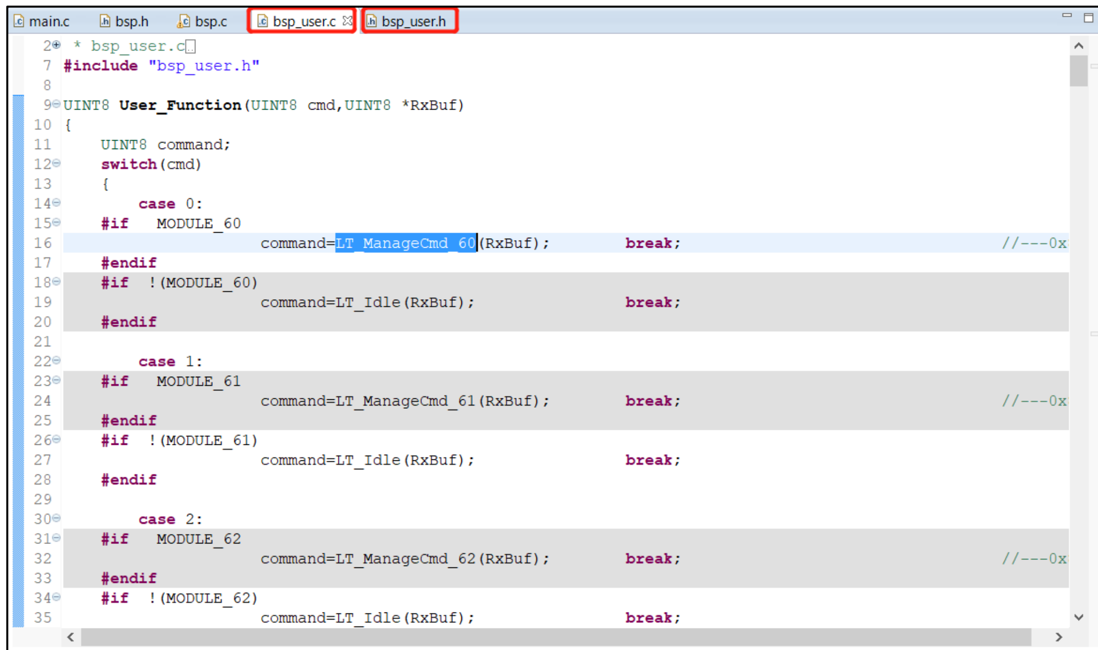

图 3-3

© Levetop Semiconductor Co., Ltd.

3.2 显示驱动部分(SRAM)

显示驱动部分的 SRAM 是 16Mbytes，以 800*480 分辨率的屏为例子，按照我们串口屏方案对图层的划分，有 21 层图层可使用，如图 3-4 所示，是我们已有的串口指令功能定义的图层，这部分宏定义建议不要乱改，若用不到的指令功能可关闭 ，那么对应的图层资源可空出作为自定义功能的图层资源，相关指令功能使用到的具体的图层资源可咨询我们工程师。

```
main.c   bsp.h   bsp.c   bsp_user.c   bsp_user.h
22 #define MAIN_BUFF   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*1
23 #define LAY_BUFF1   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*2
24 #define LAY_BUFF2   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*3
25 #define LAY_BUFF3   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*4
26 #define LAY_BUFF4   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*5
27 #define LAY_BUFF5   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*6
28 #define LAY_BUFF6   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*7
29
30 #define CircleTouch_BUFF   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*8
31 #define QR_BUFF     LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*9
32 #define TEMP_BUFF       LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*10
33
34 #define ROLL_BUFF1  Get_Buff(LCD_XSIZE_TFT,LCD_YSIZE_TFT,11)
35 #define ROLL_BUFF2  Get_Buff(LCD_XSIZE_TFT,LCD_YSIZE_TFT,13)
36
37 #define Password_BUFF1   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*17
38 #define Password_BUFF2   LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*18
39
40 #define SLID_BUFF       Get_Buff(LCD_XSIZE_TFT,LCD_YSIZE_TFT,19)
41
```

图 3-4

如图 3-5 所示，建议自行在 bsp.user.h 文件中添加对图层的宏定义：

```
main.c   bsp.h   bsp.c   bsp_user.c   bsp_user.h
39 #define     MODULE_77        0          //1选择77指令，0不选77指令
40 #define     MODULE_78        0          //1选择78指令，0不选78指令
41 #define     MODULE_79        0          //1选择79指令，0不选79指令
42 #define     MODULE_7A        0          //1选择7A指令，0不选7A指令
43 #define     MODULE_7B        0          //1选择7B指令，0不选7B指令
44 #define     MODULE_7C        0          //1选择7C指令，0不选7C指令
45 #define     MODULE_7D        0          //1选择7D指令，0不选7D指令
46 #define     MODULE_7E        0          //1选择7E指令，0不选7E指令
47 #define     MODULE_7F        0          //1选择7F指令，0不选7F指令
48
49 #define LAY_MyBUFF0    LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*5
50 #define LAY_MyBUFF1    LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*6
51 #define LAY_MyBUFF2    LCD_XSIZE_TFT*LCD_YSIZE_TFT*2*7
52
53 UINT8 User_Function(UINT8 cmd,UINT8 *RxBuf);
54
55 #if    MODULE_60
```

图 3-5

© Levetop Semiconductor Co., Ltd.

4. 电容触摸屏的调试说明

硬件电路若是按照我们提供的的原理图来设计的，正常调试完后，用 USB 烧入我们给的程序固件到 7688 和用对应版本的 UartTooL 或 UI_Editor 上位机工具生成的 bin 到 Flash，屏幕基本是能够正常显示的，剩下的可能就差 CTP 的调试，我们公版程序的 TP 部分是根据 FT5216 来调试的，客户若采用其它系列的 IC，只要触摸调试条件能达到我们制定的标准，是可以支持各种系列的 TP，我们调试过 FT、GT、ST 等类型的触摸屏，如有调试疑问可向我们的工程师咨询。

如图 4-1 所示，gTpInfo.sta 这个标志位即为触摸状态(1:按下　0:松开)，所以在调试电容触摸屏的时候，在读取坐标正确的同时，要保证手指按下时 gTpInfo.sta=1，松开后 gTpInfo.sta=0，另外 iic 读写和时序则需根据实际情况来调试。

```
UINT8 FT5216_Scan(void)
{
    if(fingerNumber==0&&TP_Get_INT()!= 0)
    {
        gTpInfo.sta = 0;
        return 0;
    }
    else
    {
        TP_read_XY();
        if(gTpInfo.x[0]>800)     gTpInfo.x[0] = 800;
        if(gTpInfo.y[0]>480)     gTpInfo.y[0] = 480;
        if(gTpInfo.sta == 0)     gTpInfo.sta = 1;
        Delay_ms(5);
        return 1;
    }
}
```

图 4-1

© Levetop Semiconductor Co., Ltd.

5. LT768 函数库的熟悉

在自定义编程的时候，若涉及到与显示相关的操作，则需要调用我们 768 的函数库，因此客户要客制化开发的话，学会使用我们的 768 函数库是非常有必要的，要显示图片无非就两个步骤，从 Flash 取数据到 SRAM，再 SRAM 到 SRAM 之间的处理，至于更多的编程细节可参考我们串口屏公版的相关程序，如有问题可咨询我们工程人员，如图 5-1 所示，LT768_Lib.c 和 LT768_Lib.h 是 768 的函数库。

```
main.c    LT768_Lib.c    LT768_Lib.h
151 void LT768_Init(void);
152
153 void LT768_PLL_Initial(void);
154 void LT768_SDRAM_initail(unsigned char mclk);
155 void Set_LCD_Panel(void);
156
157 /* 写数据到内存 */
158 void MPU8_8bpp_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,const unsig
159 void MPU8_16bpp_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,const unsi
160 void MPU8_24bpp_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,const unsi
161 void MPU16_16bpp_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,const uns
162 void MPU16_24bpp_Mode1_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,con
163 void MPU16_24bpp_Mode2_Memory_Write(unsigned short x,unsigned short y,unsigned short w,unsigned short h,con
164
165 /* 硬件画线段 */
166 void LT768_DrawLine(unsigned short X1,unsigned short Y1,unsigned short X2,unsigned short Y2,unsigned long L
167 void LT768_DrawLine_Width(unsigned short X1,unsigned short Y1,unsigned short X2,unsigned short Y2,unsigned
168
169 /* 硬件画圆 */
170 void LT768_DrawCircle(unsigned short XCenter,unsigned short YCenter,unsigned short R,unsigned long CircleCo
171 void LT768_DrawCircle_Fill(unsigned short XCenter,unsigned short YCenter,unsigned short R,unsigned long For
172 void LT768_DrawCircle_Width(unsigned short XCenter,unsigned short YCenter,unsigned short R,unsigned long Ci
173
174 /* 硬件画椭圆 */
175 void LT768_DrawEllipse(unsigned short XCenter,unsigned short YCenter,unsigned short X_R,unsigned short Y_R,
176 void LT768_DrawEllipse_Fill(unsigned short XCenter,unsigned short YCenter,unsigned short X_R,unsigned short
177 void LT768_DrawEllipse_Width(unsigned short XCenter,unsigned short YCenter,unsigned short X_R,unsigned shor
178
```

图 5-1

© Levetop Semiconductor Co., Ltd.

6. Flash 的 bin 档整合

前面我们说过了客户想要自定义的程序代码可在我们提供的指定接口内编写，那么另外的 Flash 的 bin 档这部分则需要 UI_Editor 和 UartTool 这两个上位机工具来配合，以下我们用一个简单的例子来提供参考。

6.1 公版部分(UI_Editor 制作)

如图 6-1 所示，是我们用 UI_Editor 的工具来生成的一个 UartTFT_Flash.bin 文件（UI_Editor 具体使用方法可参考 UI_Editor 应用手册，这里不一一细说），只需烧入这个 bin 档到 Flash 和烧录我们的公版程序到 MCU，这部分功能都可实现。

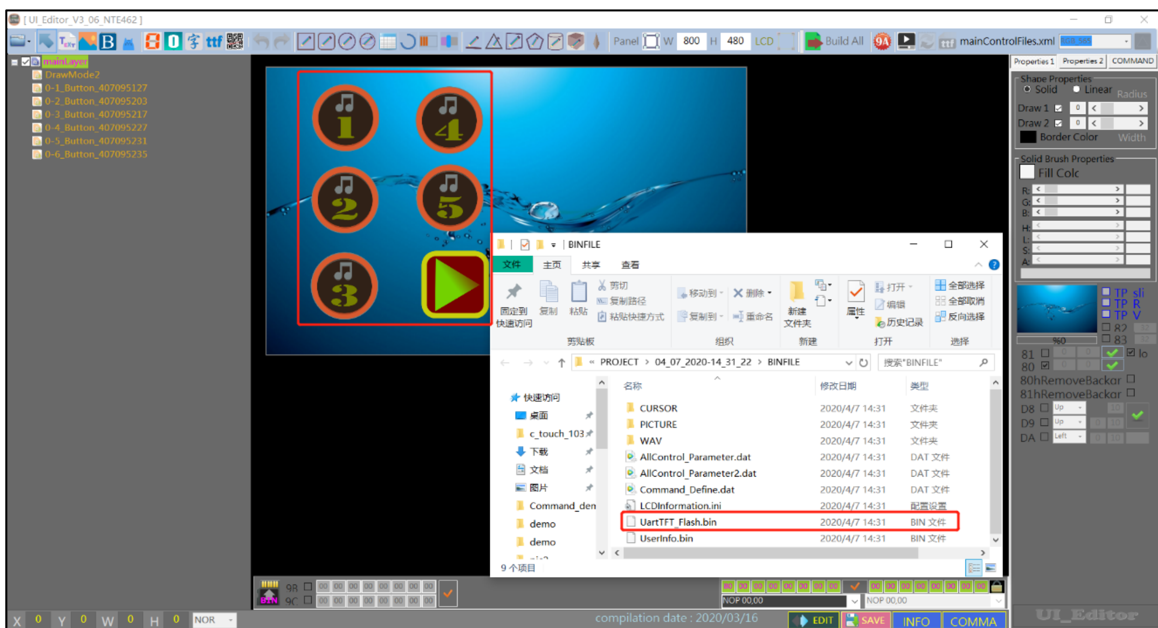

图 6-1

6.2 客制化部分(UartTool 制作)

如图 6-2 和 6-3 所示，用 UartTool 工具制作了两张图片的 bin 文件 (UartTool 具体使用方法可参考 UartTool 应用手册) 在 bin 文件生成的同时，有个 txt 文档也对应地生成了，此文档记录了该图片的宽度、高度和数据大小等，这里就以一张不带透明度的图片和一张带透明度的图片作为例子

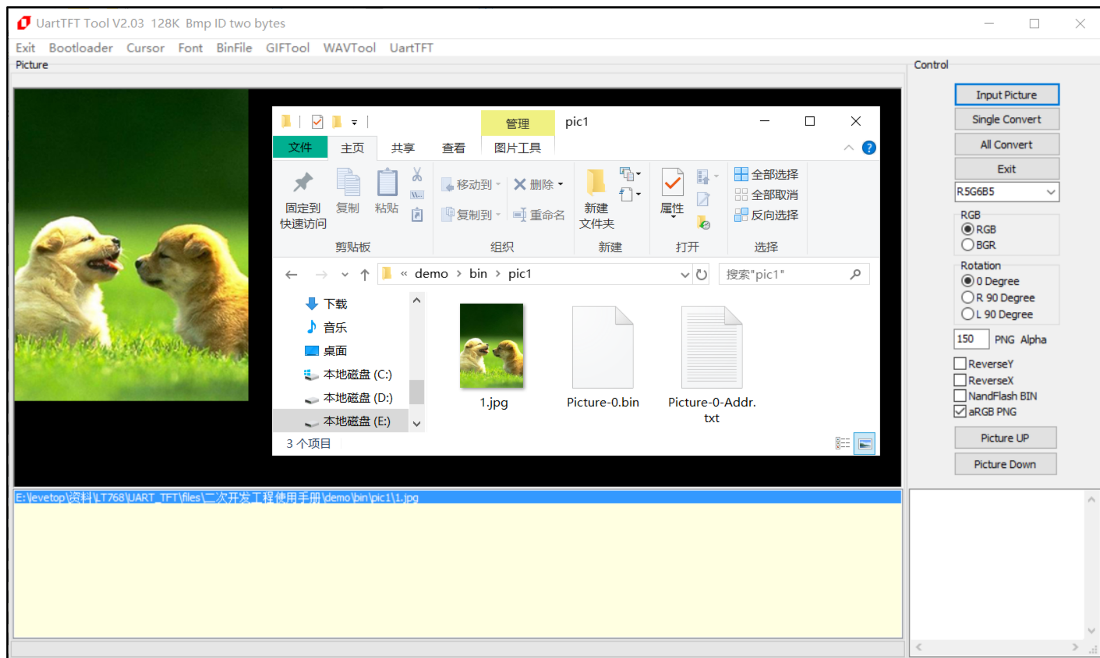

图 6-2

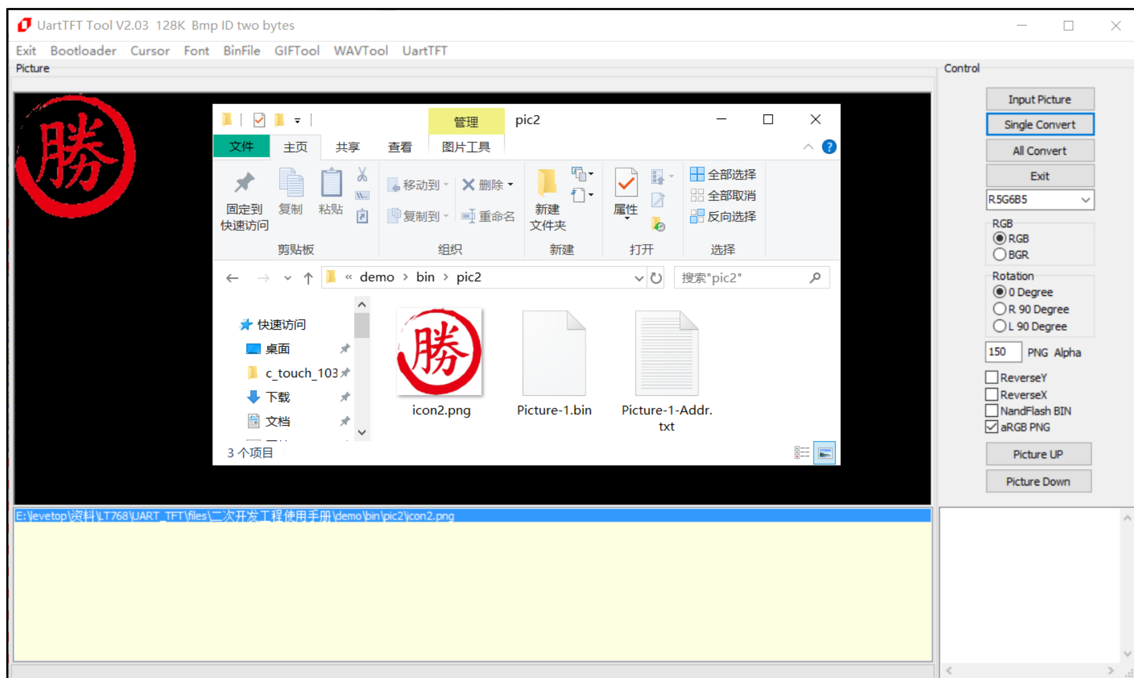

图 6-3

© Levetop Semiconductor Co., Ltd.

6.3 bin 档整合

如图 6-4 所示，需要将 UartTFT_Flash.bin 放在地址 0 的地方，后面就可以按顺序地添加自定义部分的 bin 文件，随后只要将 All.bin 烧入到 Flash 中，bin 档整合工作就算完成，图 6-5 中记录了各 bin 档的起始存放地址，客户自定义编程的时候就需要使用到这些地址。

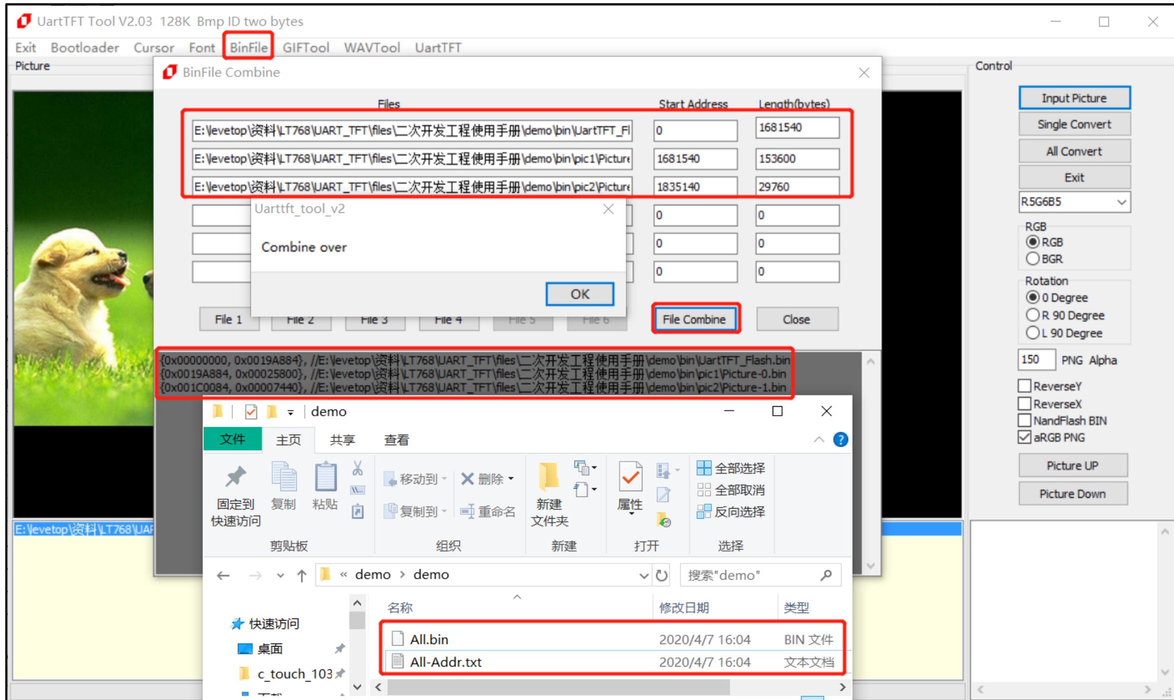

图 6-4

图 6-5

© Levetop Semiconductor Co., Ltd.

7. 范例

现在以第 6 节的 All.bin 烧入 Flash 作为前提，来简单地告知怎么结合串口指令来调用显示图片，以自定义 60h 指令作为例子。

如图 7-1 所示，我们先声明和定义图片信息的结构体，此目的是方便后面更改这些参数（分别为左上角 X 坐标、Y 坐标、图片宽度、图片高度、Flash 存放起始地址和图片类型标志位）。

```
272 #if   MODULE_60
273 typedef struct
274 {
275     UINT16 x;
276     UINT16 y;
277     UINT16 w;
278     UINT16 h;
279     UINT32 addr;
280     UINT8 flag;
281 }Pic_my_info;
282
283 Pic_my_info Mypic[2]={
284         {480,20,240,320,0x0019A884,0},
285         {650,150,120,124,0x001C0084,2}
286 };
287
288 UINT8 LT_ManageCmd_60(UINT8 *rxBuf)
289 {
290     UINT8 buf[3];
291     UINT16 canvas_w = 0;
292     UINT8 oper = 0;
293
```

图 7-1

如图 7-2 所示，**1** 部分的 LT768_DMA_24bit_Block_Or_Line_16bpp( )函数就是结合 DMA 从 Flash 读数据到我们规划的 LAY_BUFF1 显存(SRAM)区域，**2** 部分则是判断图片类型标志位的不同来对 LAY_BUFF1 中的数据进行对应的处理。

LT768_BTE_Memory_Copy( )这个是最基本的显存数据 copy 的函数，即将 LAY_BUFF1 的数据原样的复制到 0 地址（默认定义为主视窗的起始地址）的显存位置，此步骤操作后我们就能马上看到在屏幕上显示出来的图片。

LT768_BTE_Memory_Copy_Chroma_key 和 BTE_Pixel_16bpp_Alpha_Blending 则是针对带透明度图片的处理，功能是先对 LAY_BUFF1 里特定的数据做特定的处理再 copy 显存数据，具体使用方法可查阅 768 函数库的 AP_Note 或咨询我们的工程师。

© Levetop Semiconductor Co., Ltd.

```
UINT8 LT_ManageCmd_60(UINT8 *rxBuf)
{
    UINT8 buf[3];
    UINT16 canvas_w = 0;
    UINT8 oper = 0;
    if((gUsartRx.Count==9)||(cmd_flag== 1))
    {
        oper = rxBuf[1];
        canvas_w = Get_Canvas_Width(Mypic[oper].w);
        Canvas_Image_Start_address(LAY_BUFF1);
        Canvas_image_width(canvas_w);
        LT768_DMA_24bit_Block_Or_Line_16bpp(SPI_X,CLK_DIV,0,0,
                                    Mypic[oper].w,Mypic[oper].h,
                                    Mypic[oper].w,Mypic[oper].addr
                                    ,LAY_BUFF1,canvas_w
                      );
        if(Mypic[oper].flag == 0)
        {
            LT768_BTE_Memory_Copy(LAY_BUFF1,canvas_w,0,0,
                                  LAY_BUFF1,canvas_w,0,0,
                                  0,LCD_XSIZE_TFT,Mypic[oper].x,Mypic[oper].y,
                                  0x0c,Mypic[oper].w,Mypic[oper].h
                                 );
        }
        else if(Mypic[oper].flag == 1)
        {
            LT768_BTE_Memory_Copy_Chroma_key(LAY_BUFF1,canvas_w,0,0,
                                  0,LCD_XSIZE_TFT,Mypic[oper].x,Mypic[oper].y,
                                  Black,Mypic[oper].w,Mypic[oper].h);
        }
        else if(Mypic[oper].flag == 2)
        {
            BTE_Pixel_16bpp_Alpha_Blending(0,LCD_XSIZE_TFT,Mypic[oper].x,Mypic[oper].y,
                                  LAY_BUFF1,canvas_w,0,0,
                                  0,LCD_XSIZE_TFT,Mypic[oper].x,Mypic[oper].y,
                                  Mypic[oper].w,Mypic[oper].h
                                 );
        }
```

图 7-2

程序写完后，我们就可以用上位机或串口工具模拟分别发送 AA 60 00 0B 2A E4 1B 11 EE 和 AA 60 01 1B 0B E4 1B 11 EE 来验证一下显示的 All.bin 里自定义添加的 bin 对应的图片是否正常，如图 7-3 所示，左边显示的图片为 UI_Editor 生成的 bin 部分对应的图片和功能，调用了 A0 00 — A0 05 的指令，右边的两张图则分别为执行了指令 60 00 和 60 01 的效果。

图 7-3

© Levetop Semiconductor Co., Ltd.