



# LT268A

## TFT 串口屏

---

# 二次开发使用说明 Application Note

V1.0

[www.levetop.cn](http://www.levetop.cn)

Levetop Semiconductor Co., Ltd.

## 1. 工程开发环境和烧写

LT268A 内部的 MCU 是用 LT32U02 的 C0 核架构，对应的编译工具则是 CCore\_IDE，如图 1-1，可在我们官网 [www.levetop.cn](http://www.levetop.cn) 的下载专区 “LT32 系列-32 位 MCU 芯片” 内找到。

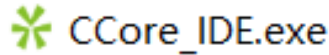


图 1-1

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。

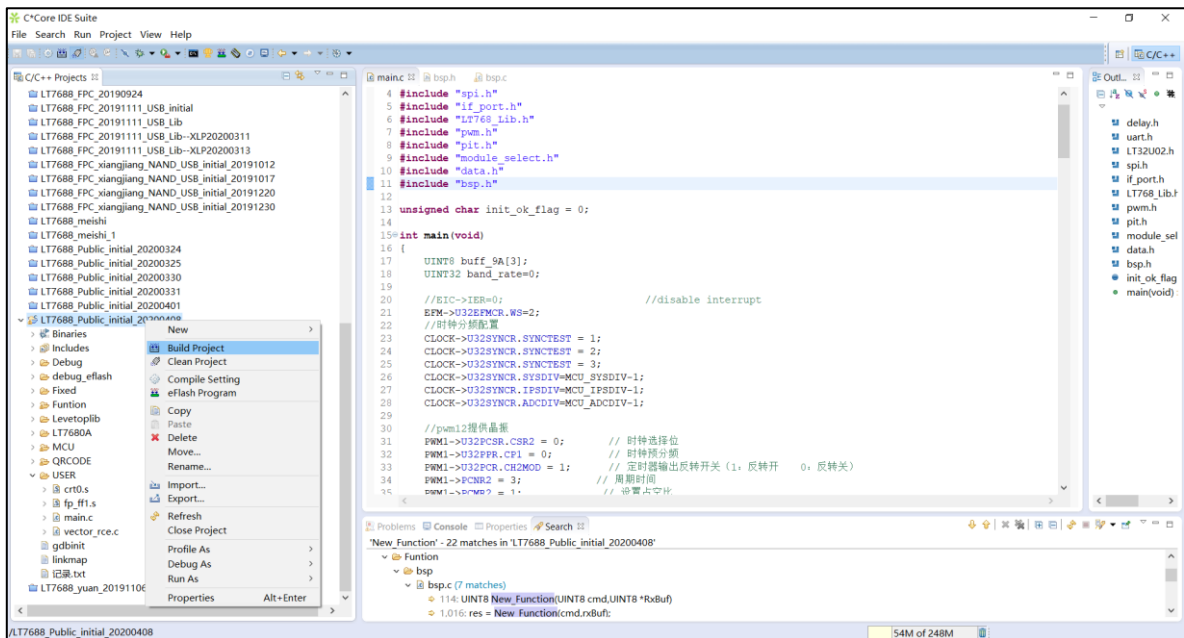


图 1-2

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。

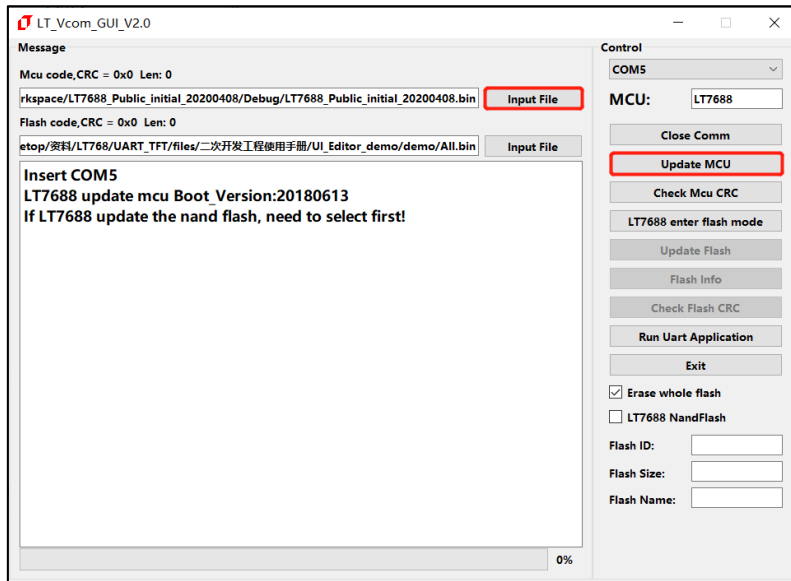


图 1-3

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。

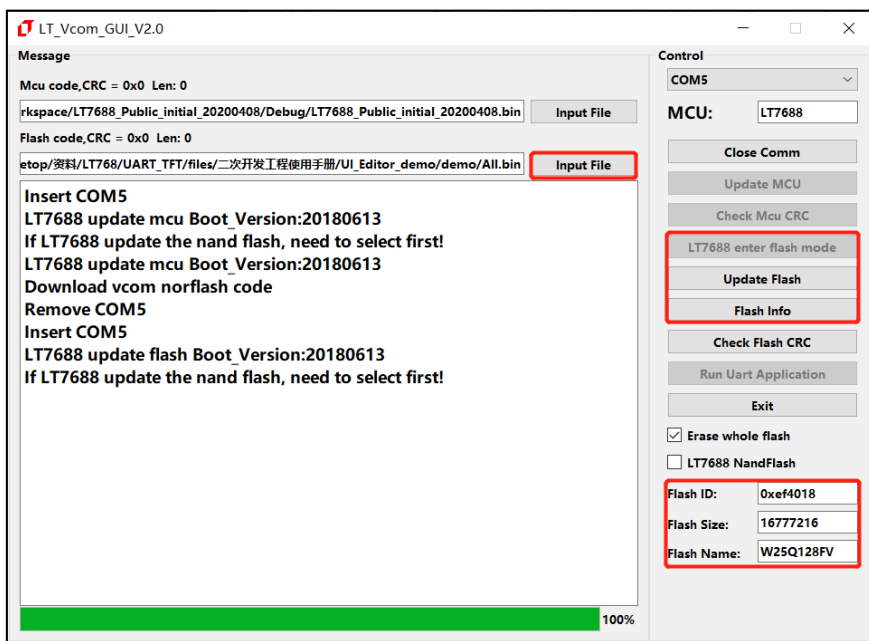


图 1-4

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定制化的资源。

## 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“LT268A\_UartTFT\_AP Note\_V20\_CH”文档。另外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

## 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，那么如果要想实现动态的图片切换则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。

如图 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

```

② #if MODULE_81
③ UINT8 LT_ManageCmd_81(UINT8 *rxBuf)
{
    UINT8 i = 0, j = 0, res = 0;
    UINT8 sum, t, oper, flag;
    UINT16 x, y;
    UINT16 pic[10] = {0};

    if((gUsartRx.Count==9) || (cmd_flag== 1))
    {
        if(rxBuf[1] & 0x80)
        if(gOpen81 == 0)                gTurnCount = 0;
        res = Get_81_infol(rxBuf, &sum, &t, &flag, &x, &y, pic, &oper);
        if(res!=OK) return res;
        for(i = 0 ; i < gTurnCount ; i++)
        gTurnInfo[i].operation = oper;
        gTurnInfo[i].sum = sum;
        gTurnInfo[i].t = t;
        gTurnInfo[i].x = x;
        gTurnInfo[i].y = y;
        gTurnInfo[i].w = gPictureInfo.w;
        gTurnInfo[i].h = gPictureInfo.h;
        gTurnInfo[i].flag = flag;

        for(j = 0 ; j < gTurnInfo[i].sum ; j++)

        if(gTurnInfo[i].flag == 0)        LT268_TFT_ShowPicture(gTurnInfo[i].x, gTurnInfo[i].y, gTurnInfo[i].w, gTurn
        else if(gTurnInfo[i].flag == 1)   LT268_TFT_ShowPicture_Png(gTurnInfo[i].x, gTurnInfo[i].y, gTurnInfo[i].w, g
        else if(gTurnInfo[i].flag == 2)   LT268_TFT_ShowPicture_Png(gTurnInfo[i].x, gTurnInfo[i].y, gTurnInfo[i].w, g

        gOpen81 = 1;
        gTurnCount++;

        return OK;
    }
    else

```

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

```

#ifndef MODULE_A0_A1
uint8_t LT_ManageCmd_A0(uint8_t *rxBuf)
{
    uint8_t j;
    uint16_t i = 0;
    uint8_t oper = 0, flag = 0;
    uint8_t buf1[8] = {0}, buf2[8] = {0};
    uint16_t x, y;
    uint8_t res = 0;

    if((gUsartRx.Count==9) || (cmd_flag== 1))
    {
        res = Get_A0_info(rxBuf, &oper, &flag, buf1, buf2, &x, &y);
        if(res!=OK) return res;

        for(i = 0 ; i < ControlCount ; i++)
        controlInfo[i].operation = oper;
        controlInfo[i].x = x;
        controlInfo[i].y = y;
        controlInfo[i].flag = gCmdInfo[1];
        for(j = 0 ; j < 8; j++)
        controlInfo[i].addr = gPictureInfo.addr;
        controlInfo[i].h = gPictureInfo.h;
        controlInfo[i].w = gPictureInfo.w;
        if(flag==0)
        if(flag==1)

        ControlCount++;
        if(ControlCount > CONTROL_SIZE) ControlCount = CONTROL_SIZE;
        ControlFlag = 1;
        return OK;
    }
}
#endif

```

图 2-5

```

void button(void)
{
    uint8_t i, j;
    uint8_t buff[3];
    gTpInfo.scan();

    if(gTpInfo.sta)
    {
        for(i = 0; i < ControlCount; i++)
    }
    if(gTpInfo.sta == 0 && button_flag == 1)
    {
        //控件按下之后再松开
        if(controlInfo[button_count].flag == 0)
        else if(controlInfo[button_count].flag == 1)
        else if(controlInfo[button_count].flag == 2)

        button_flag = 0;
        buff[0] = 0xA0;
        buff[1] = controlInfo[button_count].operation;
        buff[2] = 0x30;
        LT_DataToPc(buff, 3);
        for(j = 0; j < 8; j++)
        for(j = 0; j < 8; j++)
        {
            cmd_flag=1;
            if(control_buf[j][0]>=0x80) New_Function(control_buf[j][0]-0x80, control_buf[j]);
            #if USER_DEFINE_COMMAND
            cmd_flag=0;
        }
    }
}

```

图 2-6

### 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。

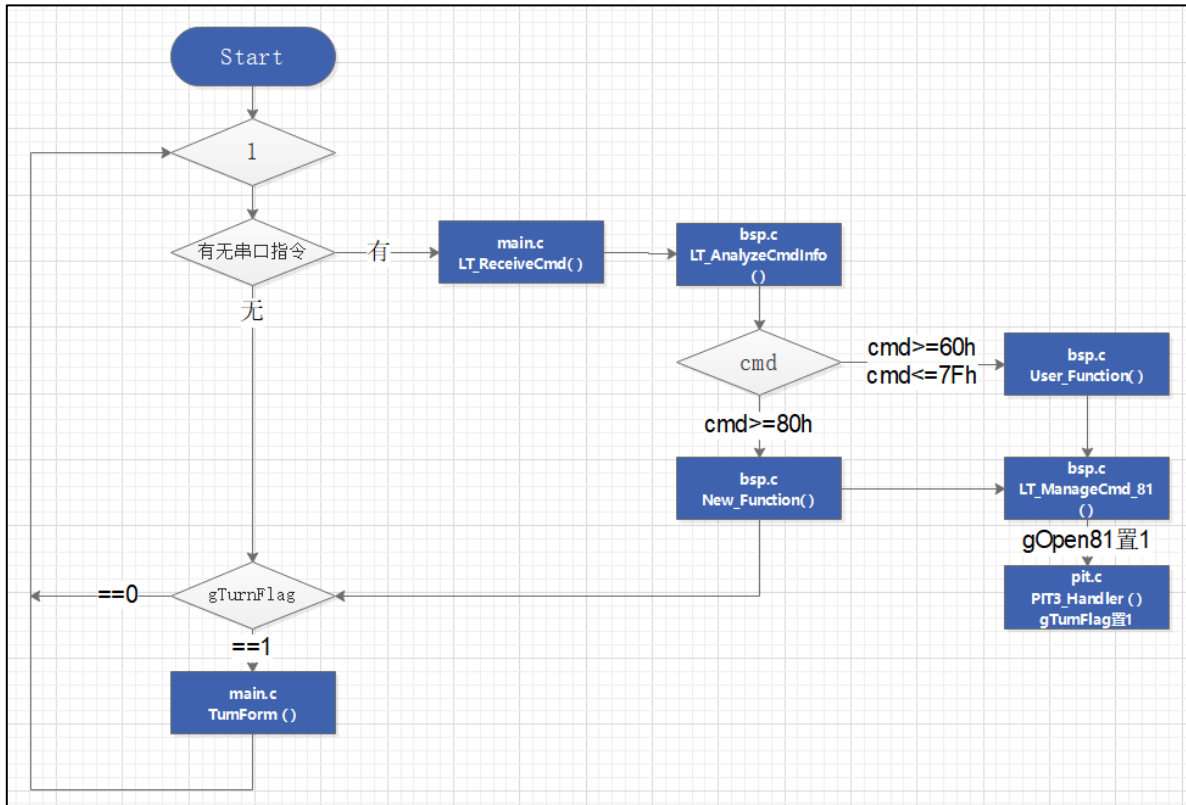


图 2-7



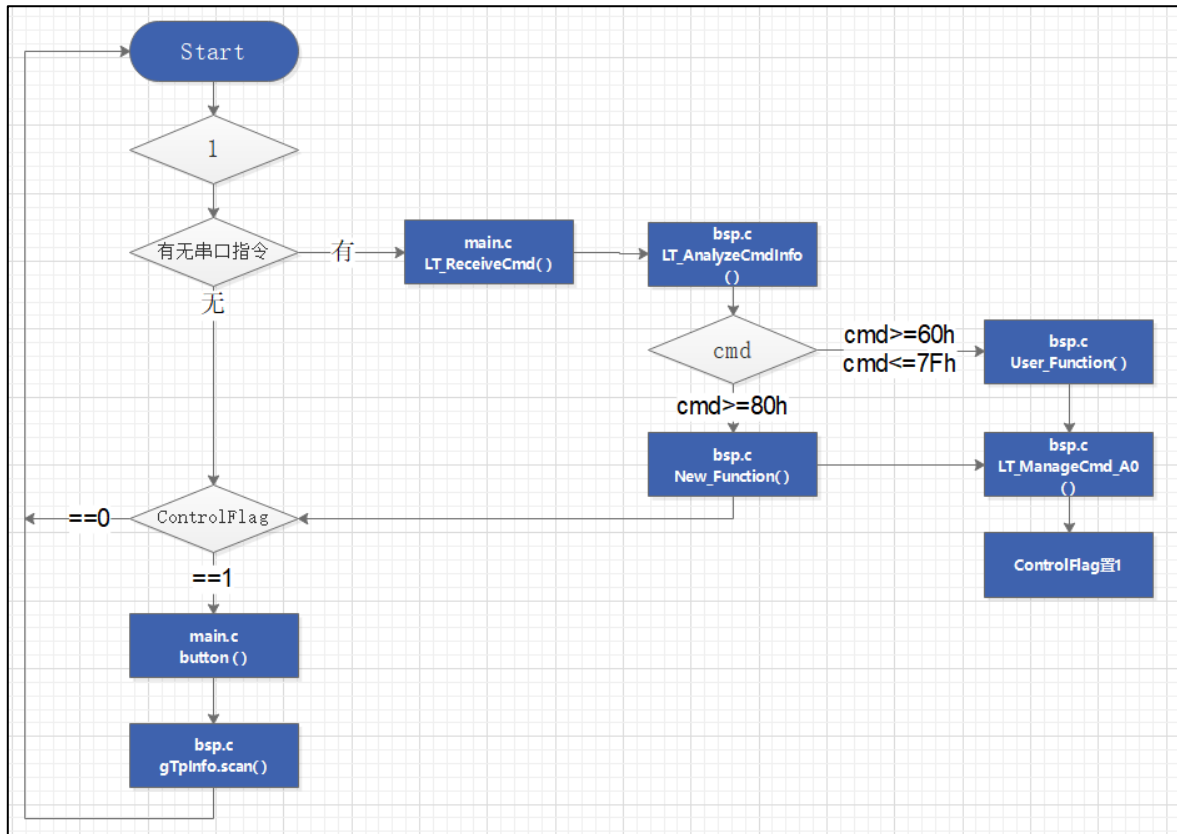


图 2-8

### 3. 工程资源的划分与使用

因为 268A 内部资源是有限的，编程部分的 RAM 是 7.5KBytes，ROM 是 59KBytes，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 56,276Bytes (text)，RAM 使用了 6,872Bytes (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 - 6,872)，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。

```

1 #ifndef _module_select_h
2 #define _module_select_h
3 #include "LT268A.h"
4
5 #define R_TOUCH_FLAG 1 //1选择电阻屏，0不选电阻屏
6 #define FT_TOUCH_FLAG 0 //1选择电容屏，0不选电容屏
7
8 #define LT268A_SPI 0 //SPI接口 //要更换需要更换libleveltop.a
9 #define LT268A_Data_8 1 //8位总线接口
10
11 #define Horizontal_screen 0 //1选择横屏，0选择竖屏
12
13 #define W25Q128 1
14
15 #define CRC_FLAG 0 //1选择CRC，0不选CRC
16
17 #define MODULE_80 1 //1选择80指令，0不选
18 #define MODULE_81 1 //1选择81指令，0不选
19 #define MODULE_82 0 //1选择82指令，0不选
20 #define MODULE_84 1 //1选择84指令，0不选
21 #define MODULE_85 0 //1选择85指令，0不选
22 #define MODULE_86_87 0 //1选择86/87指令，0不选
23 #define MODULE_88 1 //1选择88指令，0不选
24 #define MODULE_89 1 //1选择89指令，0不选
25 #define MODULE_8A 1 //1选择8A指令，0不选
26 #define MODULE_8B 1 //1选择8B指令，0不选
27 #define MODULE_8C_8D 0 //1选择8C/8D指令，0不选
28 #define MODULE_8F 1 //1选择8F指令，0不选
29 #define MODULE_90 1 //1选择90指令，0不选
30 #define MODULE_91 1 //1选择91指令，0不选
    
```

图 3-1

```

20200416 USB_public_8Bits_2Byte V
CDT build console [LT268A_TFT_240_320_20200511_USB_public_8Bits_2Byte V]
make -o binary LT268A_TFT_240_320_20200511_USB_public_8Bits_2
mcore-elf-objcopy -O binary LT268A_TFT_240_320_20200511_USB_public_8Bits_2
text data bss dec hex filename
56276 1112 5760 63148 f6ac LT268A_TFT_240_320_20200511_USB_pub
    
```

图 3-2

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。

```

2* * bsp_user.c[]
7 #include "bsp_user.h"
8
9 UINT8 User_Function(UINT8 cmd, UINT8 *RxBuf)
10 {
11     UINT8 command;
12     switch(cmd)
13     {
14     case 0:
15     #if MODULE_60
16         command=LT_ManageCmd_60(RxBuf); break; //---0x
17     #endif
18     #if !(MODULE_60)
19         command=LT_Idle(RxBuf); break;
20     #endif
21
22     case 1:
23     #if MODULE_61
24         command=LT_ManageCmd_61(RxBuf); break; //---0x
25     #endif
26     #if !(MODULE_61)
27         command=LT_Idle(RxBuf); break;
28     #endif
29
30     case 2:
31     #if MODULE_62
32         command=LT_ManageCmd_62(RxBuf); break; //---0x
33     #endif
34     #if !(MODULE_62)
35         command=LT_Idle(RxBuf); break;

```

图 3-3

## 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 268A 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

## 5. 电阻触摸屏的调试说明

硬件电路若是按照我们提供的的原理图来设计的，电阻屏调试要注意初始化函数 LCD\_TFT\_Init()和开窗函数 LT268\_TFT\_SetWindows(), 只需要替换这两个函数里面的相关代码，其它部分可以不修改。横竖屏的转换是通过修改初始化函数里面对应寄存器和 module\_select.h 里面的横竖屏模式来实现。电阻屏同样是用 gTpinfo.sta 标志位来判断触摸。

```

174 #endif
175 }
176
177 void LCD_TFT_Init(void)
178 {
179     // SPI0_Init();
180     Reset_Init();
181
182     #if LT268A_SPI
183         A0_Init();
184     #endif
185
186     #if LT268A_Data_8
187         Data_8_Port_Init();
188     #endif
189
190     LCD_SET;
191     Delay_ms(10);
192     LCD_RESET;
193     Delay_ms(10);
194     LCD_SET;
195     Delay_ms(120); //Delay 120ms
196
197     /*SPI屏的初始化代码部分, Horizontal_screen部分为设置横竖屏, 此处需根据实际情况调整*/
198     #if LT268A_SPI
199
200     /*8位并口屏的初始化代码部分*/
201     #if LT268A_Data_8
202
203     }
204 }
205

```

图 5-1

```

147
148
149 //-----设定窗口-----
150 void LT268_TFT_SetWindows(UINT16 XStar,UINT16 yStar,UINT16 Width,UINT16 High)
151 {
152     #if LT268A_SPI
153
154     #if LT268A_Data_8
155         TFT_CommandWrite(0x02);
156         TFT_DataWrite(XStar>>8);
157         TFT_CommandWrite(0x03);
158         TFT_DataWrite(XStar); //Column Start
159
160         TFT_CommandWrite(0x04);
161         TFT_DataWrite((XStar+Width-1)>>8);
162         TFT_CommandWrite(0x05);
163         TFT_DataWrite(XStar+Width-1); //Column End
164
165         TFT_CommandWrite(0x06);
166         TFT_DataWrite(yStar>>8);
167         TFT_CommandWrite(0x07);
168         TFT_DataWrite(yStar); //Row Start
169
170         TFT_CommandWrite(0x08);
171         TFT_DataWrite((yStar+High-1)>>8);
172         TFT_CommandWrite(0x09);
173         TFT_DataWrite(yStar+High-1); //Row End
174
175         TFT_CommandWrite(0x22);
176     #endif
177 }
178

```

图 5-2

## 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。

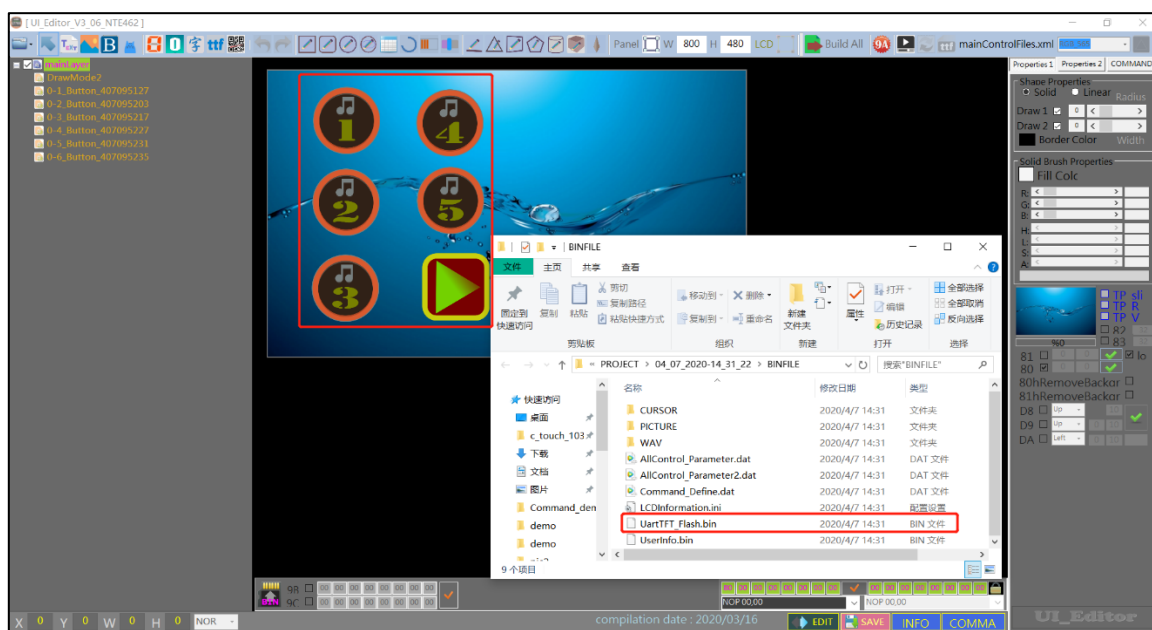


图 6-1

## 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

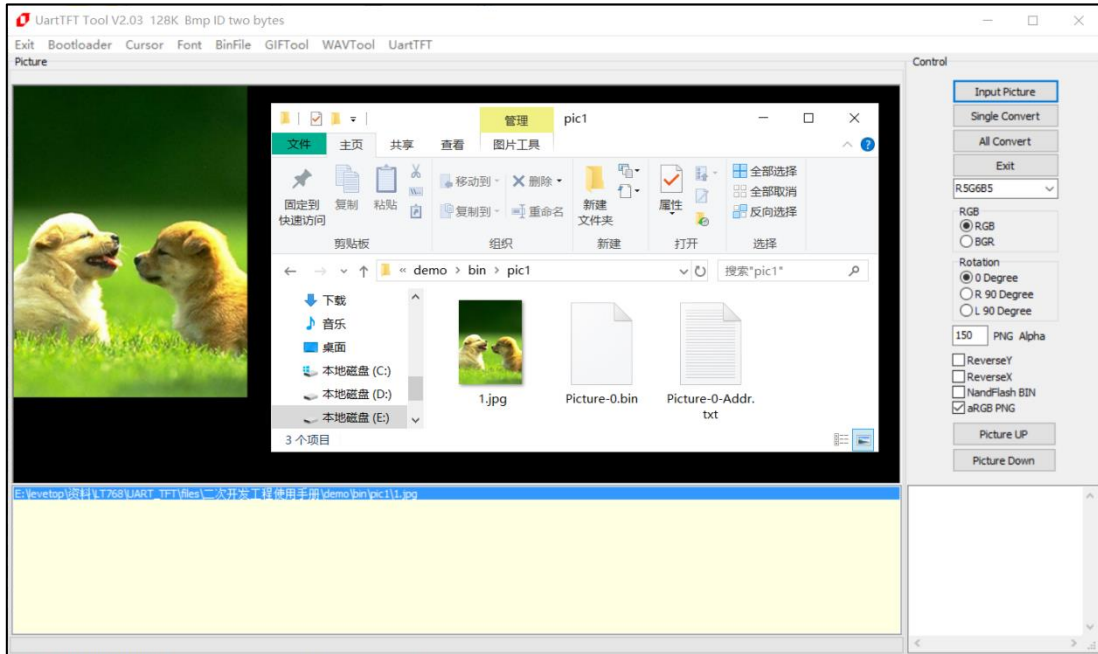


图 6-2

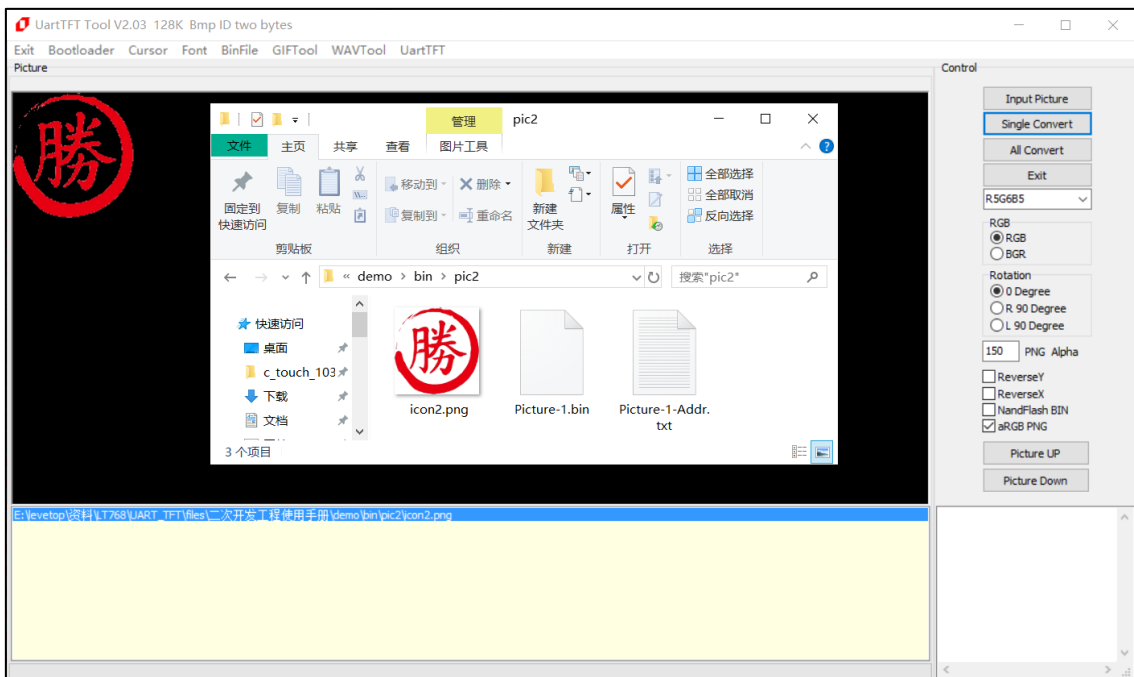


图 6-3

### 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。

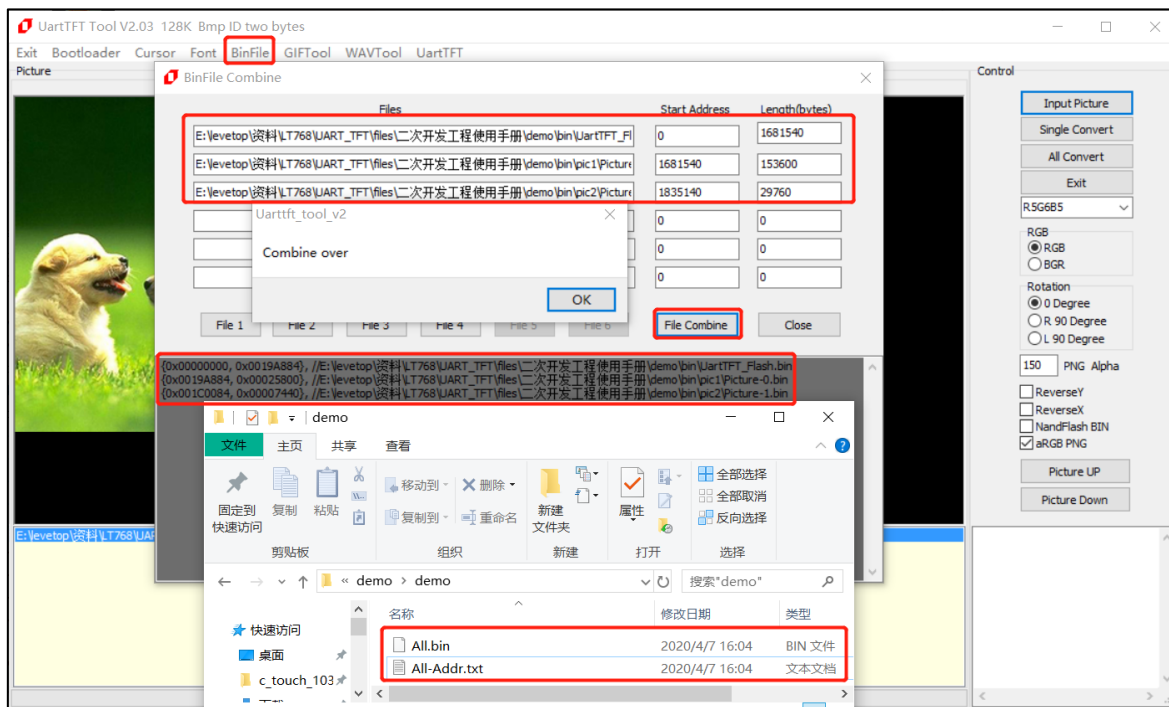


图 6-4



图 6-5



## 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     {100,100,47,67,0x0007CD78,0},
285     {0,0,47,67,0x0007E612,1}
286 };
287
288 UINT8 LT_ManageCmd_60 (UINT8 *rxBuf)
289 {
290     UINT8 buf[3];
291     UINT8 oper = 0;
```

图 7-1

如图 7-2 所示，函数 LT268\_TFT\_ShowPicture();是显示一张图片，函数 LT268\_TFT\_ShowPicture\_Png();是显示一张带透明度的图片

```

288=UINT8 LT_ManageCmd_60(UINT8 *rxBuf)
289 {
290     UINT8 buf[3];
291     UINT8 oper = 0;
292     if((gUsartRx.Count==9)||(cmd_flag== 1))
293     {
294         oper = rxBuf[1];
295         if(Mypic[oper].flag == 0)    LT268_TFT_ShowPicture(Mypic[oper].x,Mypic[oper].y,Mypic[oper].w,Mypic[oper].h,Mypic
296         else if(Mypic[oper].flag == 1)    LT268_TFT_ShowPicture_Png(Mypic[oper].x,Mypic[oper].y,Mypic[oper].w,Mypic[oper].h
297         else if(Mypic[oper].flag == 2)    LT268_TFT_ShowPicture_Png(Mypic[oper].x,Mypic[oper].y,gPictureInfo.w,gPictureInfo
298
299         buf[0] = rxBuf[0];
300         buf[1] = rxBuf[1];
301         buf[2] = 0x00;
302         LT_DataToPc(buf,3);
303     }
304
305     return OK;
306 }
    
```

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