SWM32SRET6-50

TFT-LCD 驱动演示板
应用和注意事项

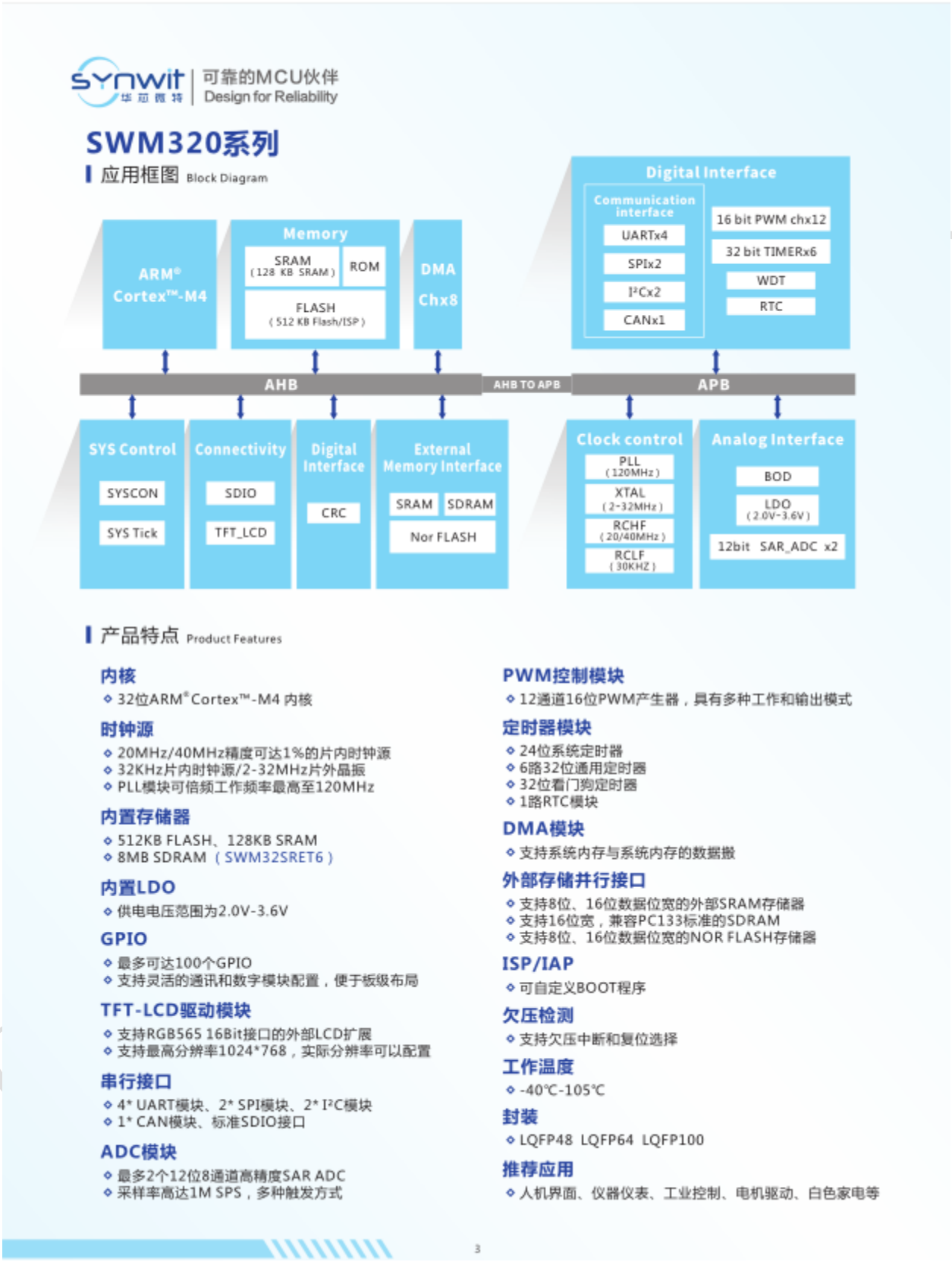

32位MCU产品及服务简介
PWR
LED
SWM32SRET7 v1
KEY
RST
www.synwit.cn


可靠的MCU伙伴
Design for Reliability
SWM32SRET7 v1
32位MCU产品及服务简介


synwit
华芯微特
可靠的MCU伙伴
Design for Reliability
华芯微特MCU产品手册

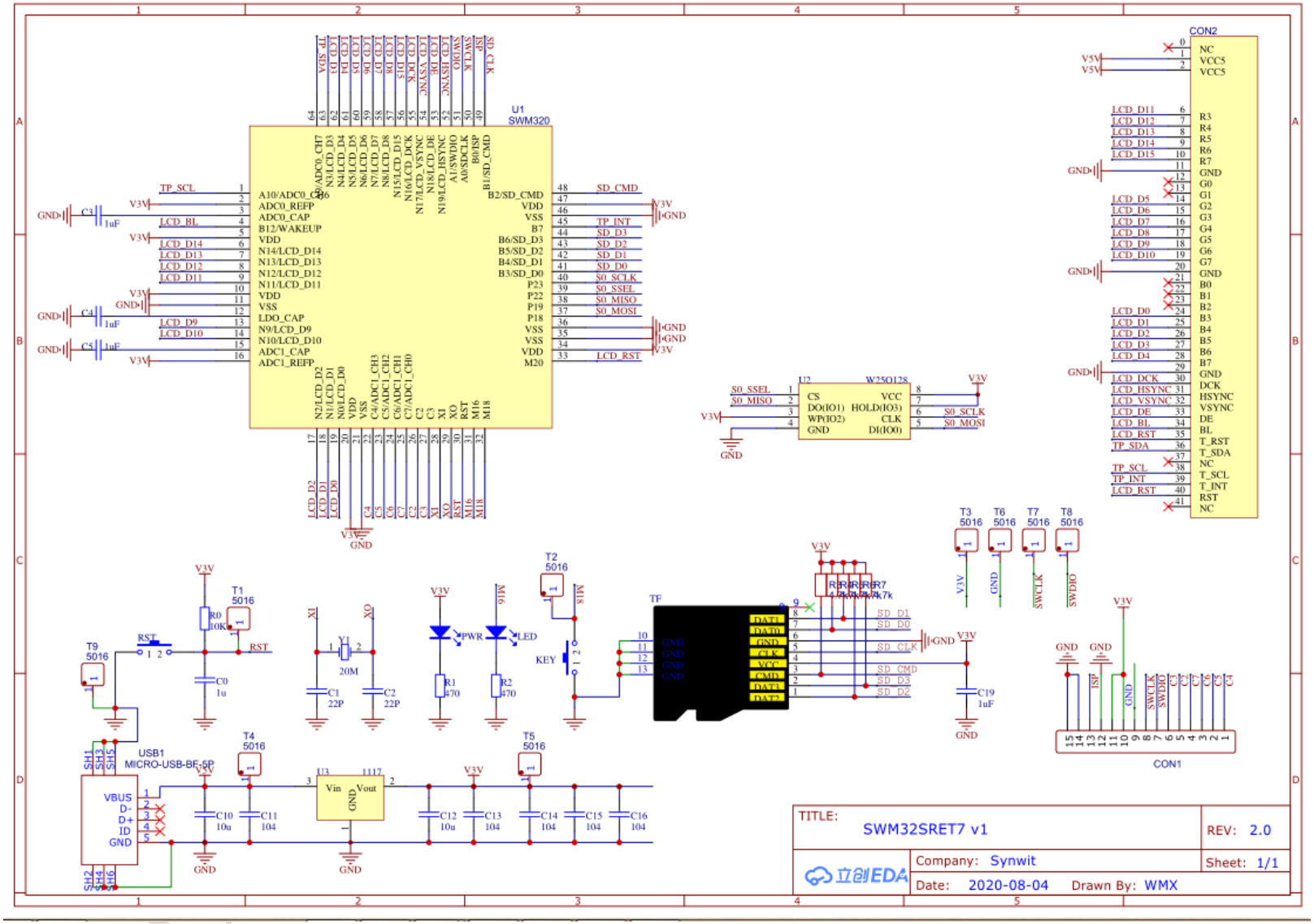
目录

一、SWM32x 资源状况

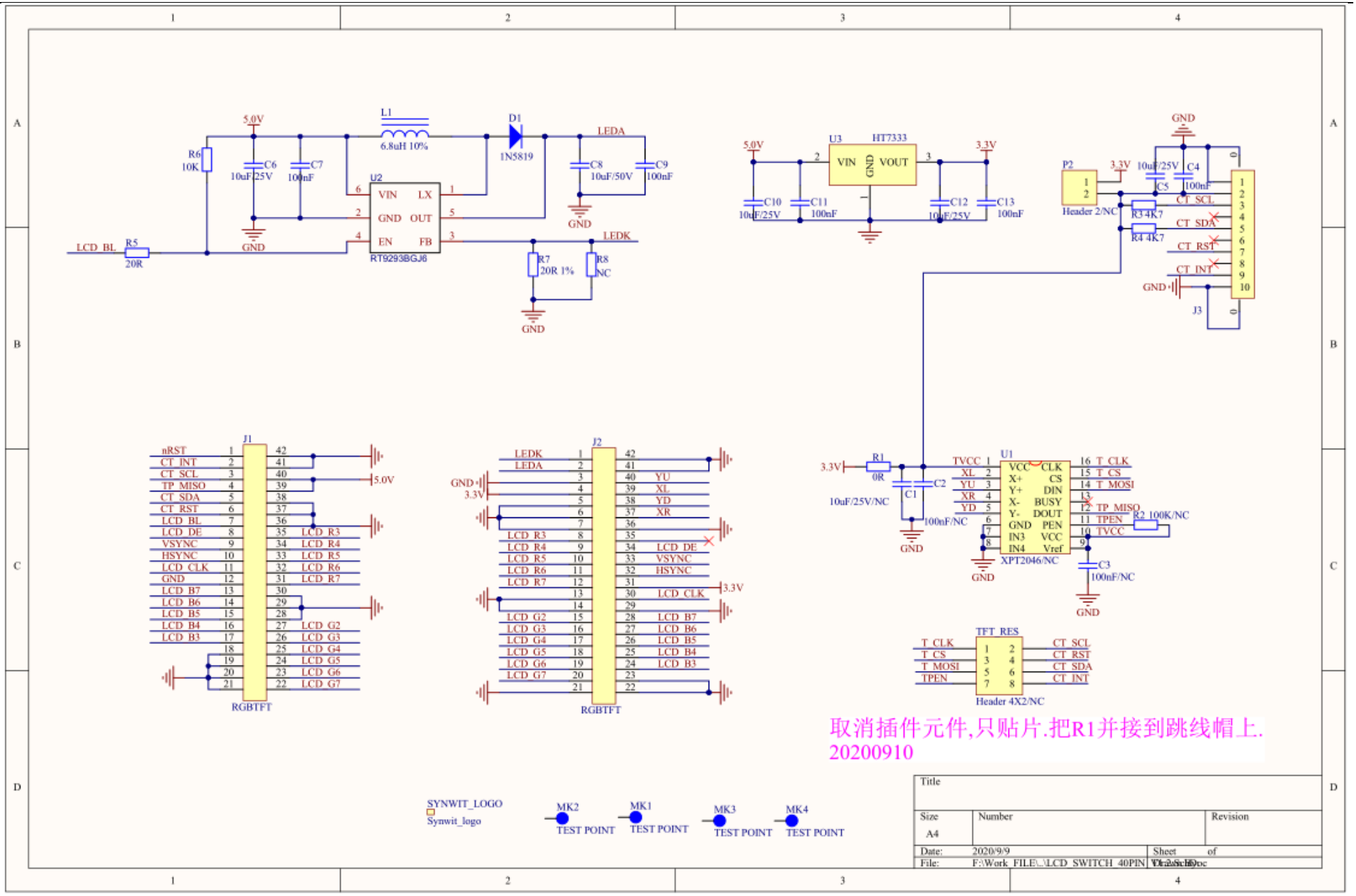


二、最小系统板介绍

2.1、最小系统板 V2.0

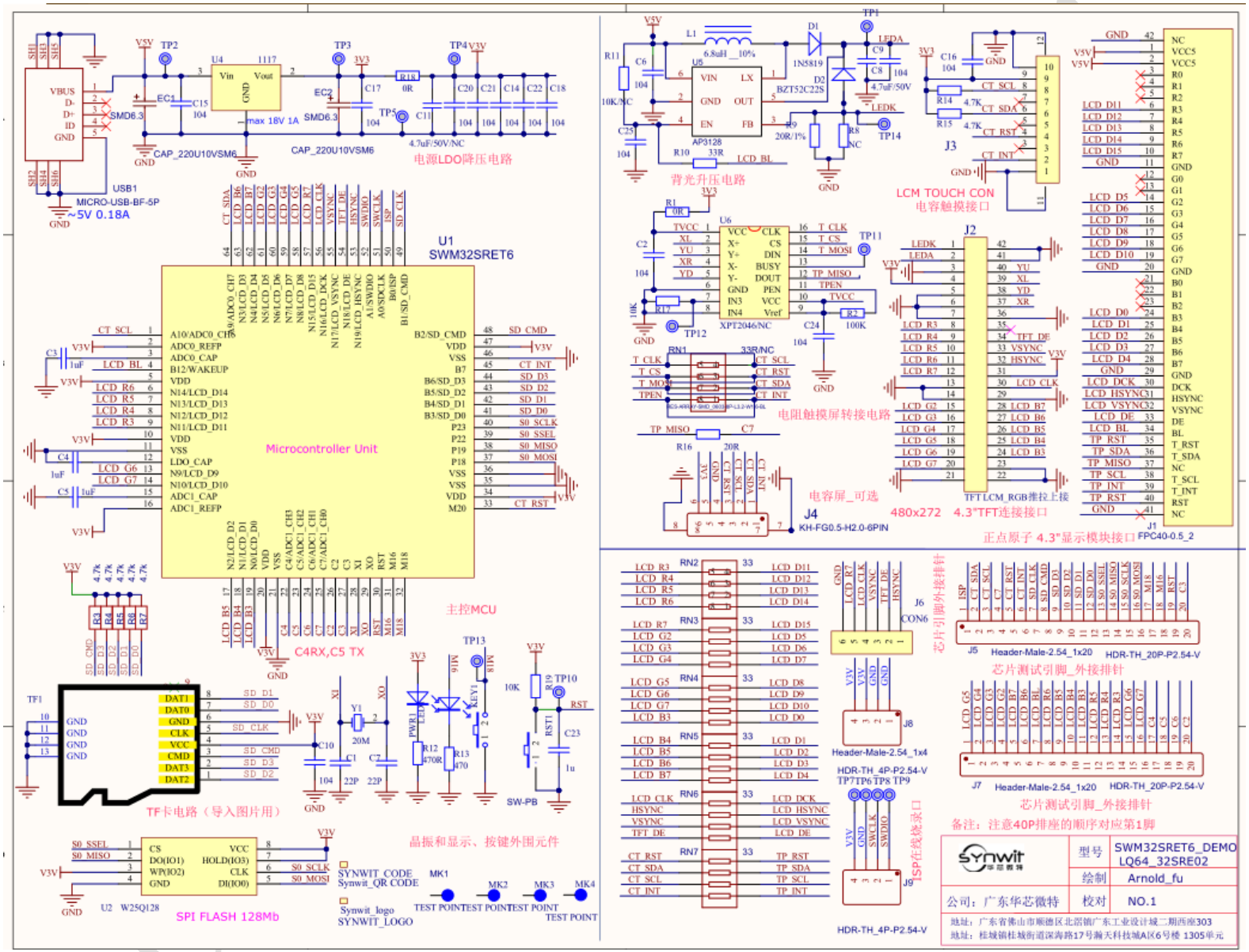


SWM32SRET6-50 v2.0 最小系统板原理图

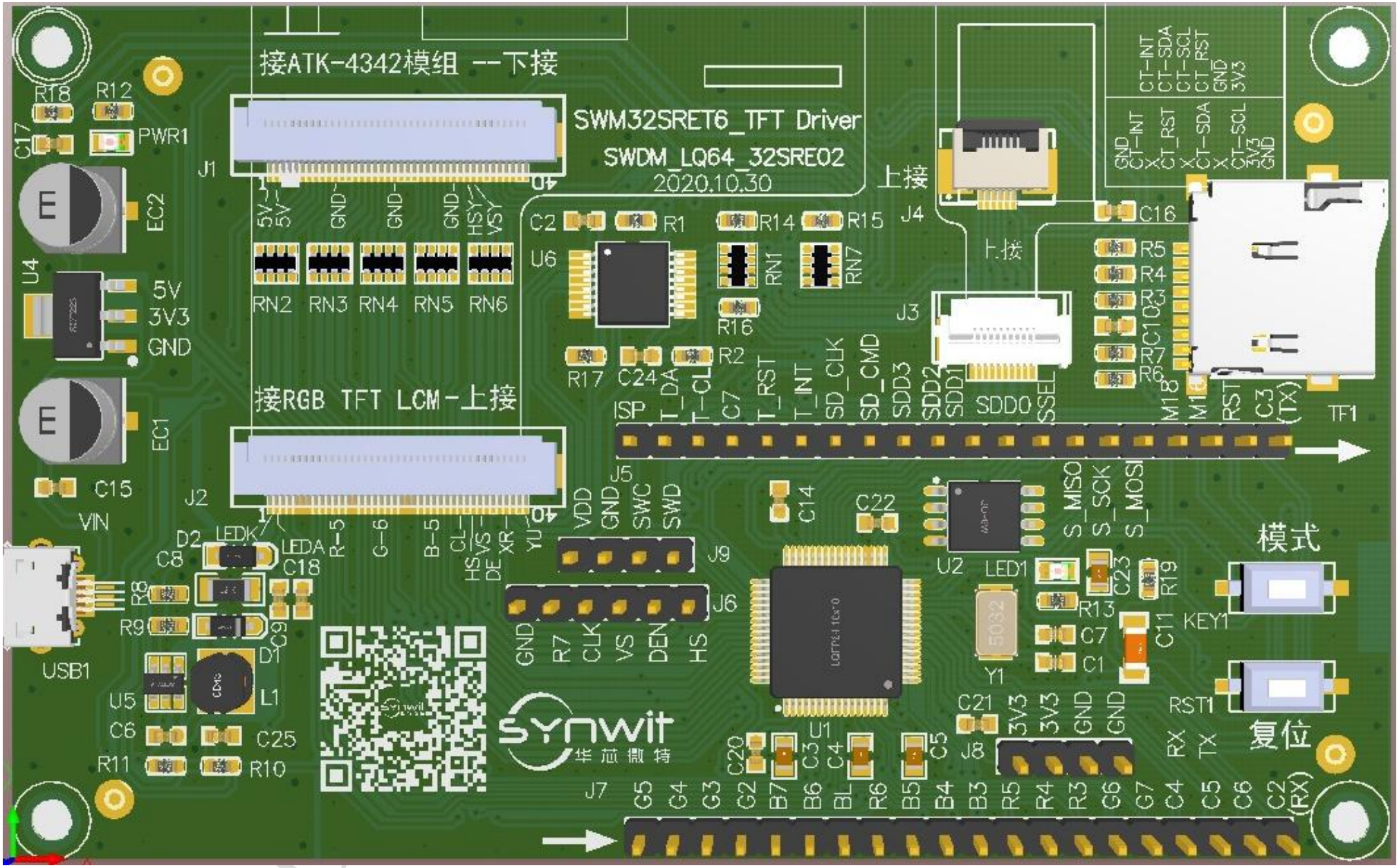

5.0V
L1
6.8uH 10%
D1
1N5819
LEDA
R6
10K
C6
10uF/25V
C7
100nF
U2
VIN
LX
GND
OUT
EN
FB
RT9293BGJ6
C8
10uF/50V
C9
100nF
LCD_BL
R5
20R
LEDK
R7
20R 1%
R8
NC
U3
HT7333
VIN
GND
VOUT
3.3V
C10
10uF/25V
C11
100nF
C12
10uF/25V
C13
100nF
P2
Header 2/NC
C4
C5
100nF
R3 4K7
R4 4K7
CT_SCL
CT_SDA
CT_RST
CT_INT
J3
J1
nRST
CT_INT
CT_SCL
TP_MISO
CT_SDA
CT_RST
LCD_BL
LCD_DE
VSYNC
HSYNC
LCD_CLK
LCD_B7
LCD_B6
LCD_B5
LCD_B4
LCD_B3
LCD_R3
LCD_R4
LCD_R5
LCD_R6
LCD_R7
LCD_G2
LCD_G3
LCD_G4
LCD_G5
LCD_G6
LCD_G7
RGBTFT
J2
LEDK
LEDA
YU
XL
YD
XR
VSYNC
HSYNC
RGBTFT
U1
R1
0R
C1
100nF/NC
C2
10uF/25V/NC
XPT2046/NC
T_CLK
T_CS
T_MOSI
TPEN
R2 100K/NC
C3
100nF/NC
TFT_RES
Header 4X2/NC
取消插件元件,只贴片.把R1并接到跳线帽上.
20200910
SYNWIT_LOGO
Synwit_logo
MK1
MK2
MK3
MK4
TEST POINT
Title
Size
A4
Number
Revision
Date:
2020/9/9
Sheet
of
File:
F:\Work_FILE\...\LCD_SWITCH_40PIN

背光驱动板原理图

2.2、最小系统板 V3.0

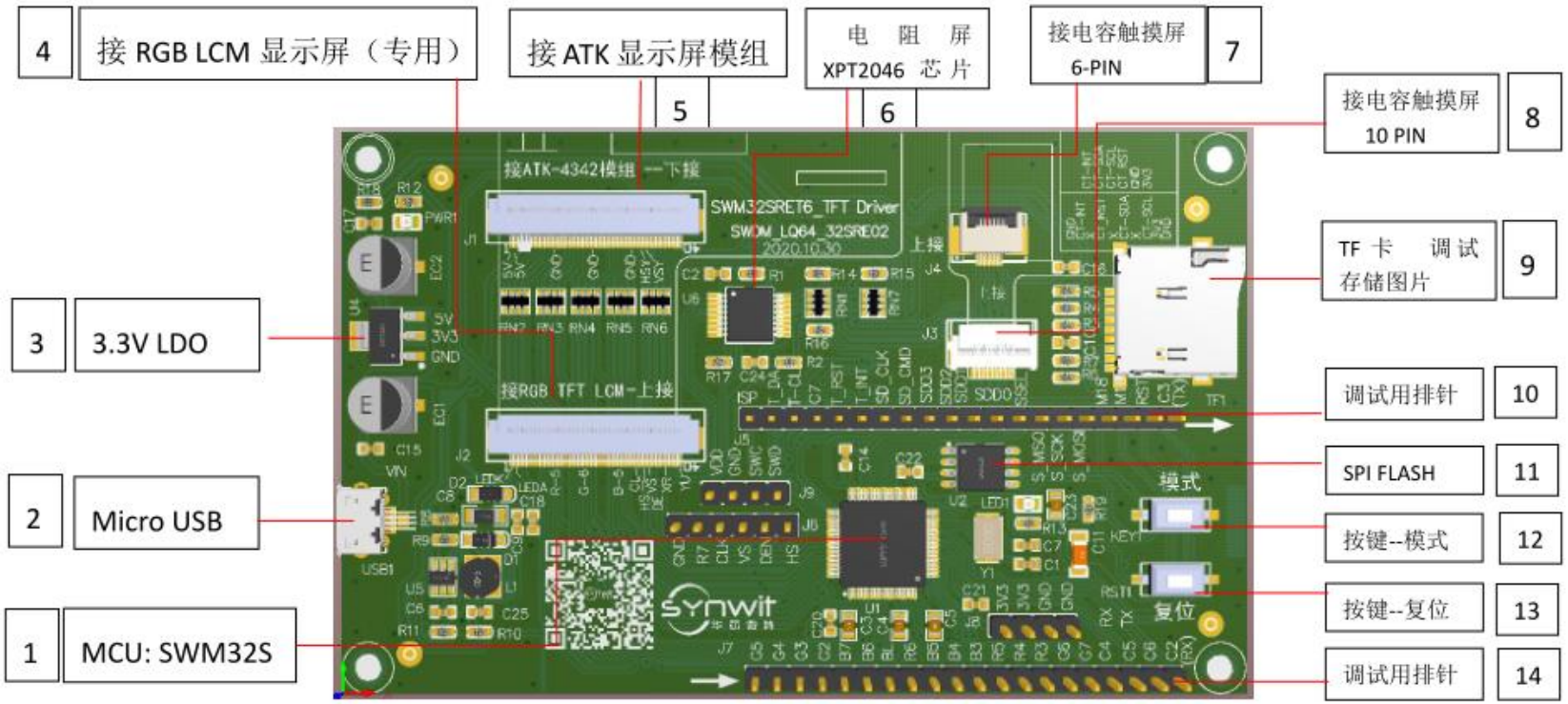


SWM32SRET6-50 v3.0 最小系统板原理图



SWM32SRET6-50 v3.0 最小系统板板框预览图

一、 SWM32SRET6 配 4.3 RGB 屏驱板 接口和配置：



二 、接口相关说明（从左至右）：

1、MCU：SWM32SRET6 (华芯微特 32SRET6 LQFP64)　　2、MICRO USB：输入供电电压 5V 0.2A

3、3.3V LDO :电压 5V 转 3.3V 给 MCU 和 Flash 供电；　　5、接 ATK 模组：参考正点原子 ATK4342 专用模组；

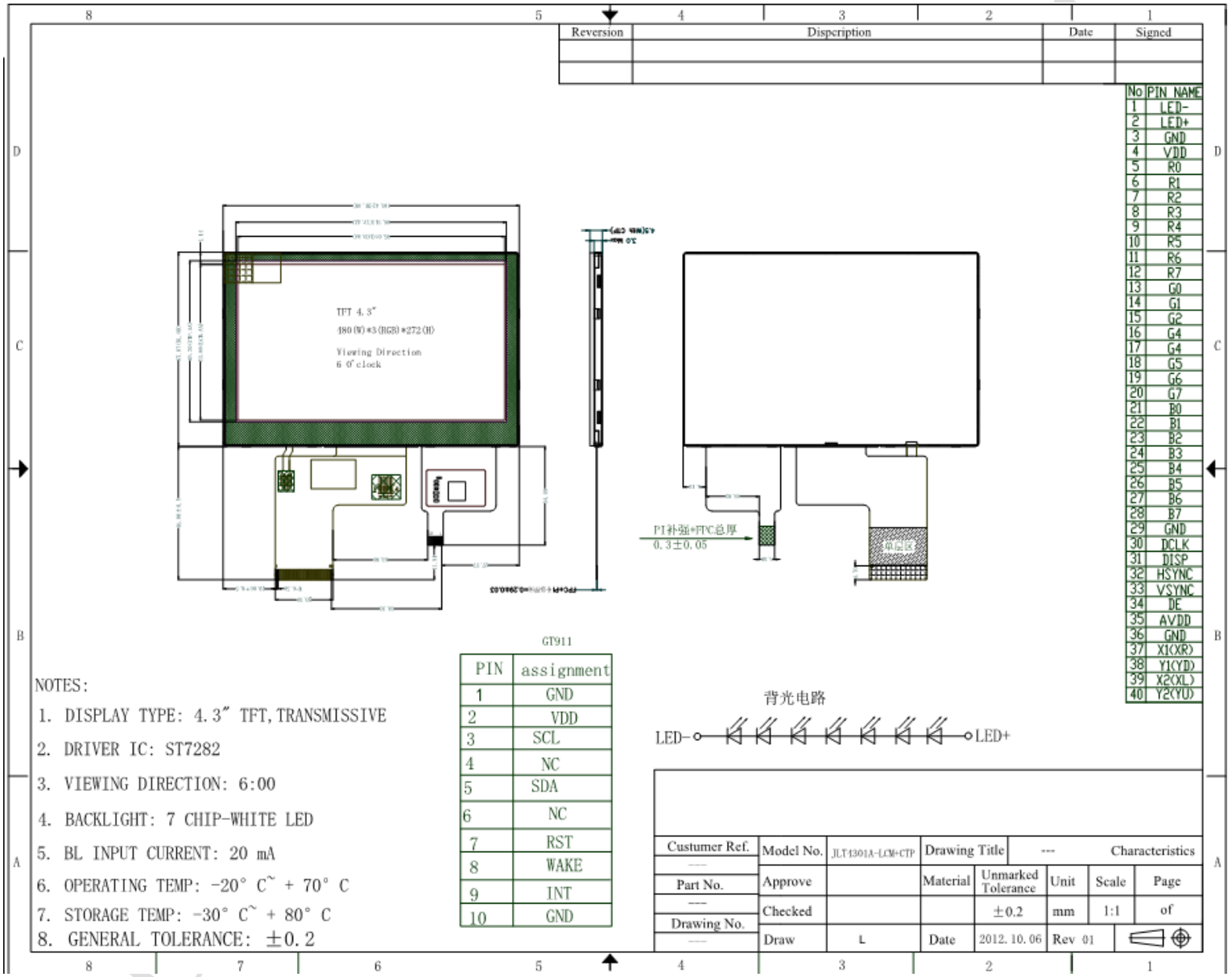
4 、RGB 显示屏 ，专用接口，接口顺序参考原理图，或 JLT4305A / JLT4301A 规格书（800x480 或 480x272 像素）

6、电阻屏 XPT2046：电阻屏转 SPI 芯片；　　7、（8） 接电容触摸屏：参考标示的连接顺序

9、TF 卡，把需要图片转存到 Flash 接口；　　10、（14）MCU 调试使用排针，测试备用；

11、SPI Flash: 存储需要显示的图片；　　12 （13）按键，转换不同模式，和复位按键

2.3、TFT-LCD 模组规格



No	PIN_NAME
1	LED-
2	LED+
3	GND
4	VDD
5	R0
6	R1
7	R2
8	R3
9	R4
10	R5
11	R6
12	R7
13	G0
14	G1
15	G2
16	G4
17	G4
18	G5
19	G6
20	G7
21	B0
22	B1
23	B2
24	B3
25	B4
26	B5
27	B6
28	B7
29	GND
30	DCLK
31	DISP
32	HSYNC
33	VSYNC
34	DE
35	AVDD
36	GND
37	X1(XR)
38	Y1(YD)
39	X2(XL)
40	Y2(YU)

PIN	assignment
1	GND
2	VDD
3	SCL
4	NC
5	SDA
6	NC
7	RST
8	WAKE
9	INT
10	GND

2.4、表 1　LCDC 端口分布　RGB565

序号	名称	SWM32SRET6-50 LQFP64	备注
1	**LCD_D0(B3)**	Pin19- N0	应用为 TFT-LCD 接口。 针对 TFT-LCD 为 RGB666 或 RGB888 接口，应用 SWM32x 系列进行驱动时，RGB 每组颜色从高位（B7）到低位（B0）顺接，低位接地即可。
2	**LCD_D1(B4)**	Pin18- N1	
3	**LCD_D2(B5)**	Pin17- N2	
4	**LCD_D3(B6)**	Pin63- N3	
5	**LCD_D4(B7)**	Pin62- N4	
6	**LCD_D5(G2)**	Pin61- N5	
7	**LCD_D6(G3)**	Pin60- N6	
8	**LCD_D7(G4)**	Pin59- N7	
9	**LCD_D8(G5)**	Pin58- N8	
10	**LCD_D9(G6)**	Pin13- N9	
11	**LCD_D10(G7)**	Pin14- N10	
12	**LCD_D11(R3)**	Pin9- N11	
13	**LCD_D12(R4)**	Pin8- N12	
14	**LCD_D13(R5)**	Pin7- N13	
15	**LCD_D14(R6)**	Pin6- N14	
16	**LCD_D15(R7)**	Pin57- N15	
17	**LCD_RD(LCD_CLK)**	Pin56-N16	
18	**LCD_CS(LCD_VSYNC)**	Pin55- N17	
19	**LCD_RS(LCD_DE)**	Pin54- N18	
20	**LCD_WNR(LCD_HSYNC)**	Pin53- N19	
21	**LCD_BL**	Pin4-B12	
22	**LCD_RESET**	Pin33-M20	DEMO 板设计与 TP_RESET 复位共用。产品要根据实际情况设计。
	Total	21	

2.5、表 2　SpiFlash 端口分布

序号	名称	SWM32SRET6-50 LQFP64	备注
1	**SPI0_CLK**	Pin40-P23 / FUNMUX1	硬件 SPI 接口，FUNMUX1 应用
2	**SPI0_SSN**	Pin39-P22 / FUNMUX0	普通 IO 应用
3	**SPI0_MISO**	Pin38-P19 / FUNMUX1	硬件 SPI 接口，FUNMUX1 应用
4	**SPI0_MOSI**	Pin37-P18 / FUNMUX0	硬件 SPI 接口，FUNMUX0 应用
	Total	4	

2.6、表 3　SDIO 端口分布

序号	名称	SWM32SRET6-50 LQFP64	备注
1	**SD_CLK**	Pin49- B1	SD 卡在产品应用中，如作为图片 UI 的传递或程序升级，SDIO 端口也可以进行互用。在产品设计中需要注意的是如何进行端口功能的转换。
2	**SD_CMD**	Pin48- B2	
3	**SD_DATA0**	Pin41- B3	
4	**SD_DATA1**	Pin42- B4	
5	**SD_DATA2**	Pin43- B5	
6	**SD_DATA3**	Pin44- B6	
	Total	6	

2.7、表 4　CTP 触摸 端口分布

序号	名称	SWM32SRET6-50 LQFP64	备注
1	**TP_RESET**	Pin33-M20 / FUNMUX0	普通 IO 应用
2	**TP_INT**	Pin45-B7 / FUNMUX1	普通 IO 应用
3	**TP_SCL**	Pin1-A10/ FUNMUX0/ADC_CH6	硬件 I2C，FUNMUX0 应用
4	**TP_SDA**	Pin64-A9 / FUNMUX1/ADC0_CH7	硬件 I2C，FUNMUX1 应用
	Total	4	

2.8、表 5　其它端口分布

序号	名称	SWM32SRET6-50 LQFP64	备注
1	**C4/ADC1_CH3/FUN0**	Pin22	可按实际项目需求进行设置。演示板设置为串口 RX
2	**C5/ADC1_CH2/FUN1**	Pin23	可按实际项目需求进行设置。演示板设置为串口 TX
3	**C6ADC1_CH1/FUN0**	Pin24	可按实际项目需求进行设置。演示板无应用
4	**C7/ADC1_CH0/FUN1**	Pin25	可按实际项目需求进行设置。演示板无应用
5	**C2/FUN0**	Pin26	可按实际项目需求进行设置。演示板无应用
6	**C3/FUN1**	Pin27	可按实际项目需求进行设置。演示板无应用
7	**M16/FUN0**	Pin31	可按实际项目需求进行设置。演示板无应用
8	**M18/FUN0**	Pin32	可按实际项目需求进行设置。演示板无应用
9	**B0/ISP/FUN0**	Pin50	可按实际项目需求进行设置。演示板无应用。 ISP 应用参考规格书描述。
10	**A0/SWCLK/FUN0**	Pin51	可按实际项目需求进行设置。演示板作为 SWD 应用。 如需要设置为普通 IO 应用，设计中要考虑如何再激活为 SWD 应用作为程序的调试或下载。
11	**A1/SWDIO/FUN1**	Pin52	可按实际项目需求进行设置。演示板作为 SWD 应用。 如需要设置为普通 IO 应用，设计中要考虑如何再激活为 SWD 应用作为程序的调试或下载。
	Total	11	

三、硬件设计

3.1、TFT-LCD 接口要求

接口要求为 RGB565，红色 5 bit，绿色 6 bit，蓝色 5 bit。如果 TFT-LCD 是 RGB888，或者是 RGB666 的，那么应该从高位起始顺接，即 R7/R6/R5/R4/R3，G7/G6/G5/G4/G3/G2，B7/B6/B5/B4/B3。 然后就是 Hsync、Vsync、PCLK、DE。

建议在原理图设计中，每个数据线的连接预留串接电阻 33 欧姆，作为抗干扰作用。

3.2、SD 卡的设计

布线设计中避开干扰信号。

SD 卡在产品应用中，如作为图片 UI 的传递或程序升级，SDIO 端口也可以进行互用，比如，在上电的某个时间段作为 SDIO 的应用，当检测到没有外挂 SD 卡，则转换为普通 IO 的驱动应用。在产品设计中需要注意的是如何进行端口功能的转换。

3.3、Demo 板上 SpiFlash 可以放置多少张 480x272 的整图？

Demo 板上的 SpiFlash 是 W25Q128，是 128M bit，即 16M Byte，16x1024 BYTE；一张 480x272 的图片 是 480x272x2= 261120 Byte。 (16 x 1024 x 1024) / 261120 = 64 张整图。

注意：在调试过程中，如转换的图片数据是正确的，从 SD 卡拷贝数据到 SpiFlash 过程中出现某些图片数据出错的现象，可以先来判断是否 SpiFlash 的存储空间出现了溢出。

3.4、SWM32Sxxx 内置 8M Byte SDRAM 可以放置多少张 480x272 的整图？

SWM32Sxxx 内置的 8M Byte SDRAM ，即 64M bit；一张 480x272 的图片 是 480x272x2= 261120 Byte。 (8 x 1024 x 1024) / 261120 = 32 张整图。在应用中，评估项目应用的 UI 素材大小，如小于 8M Byte，则可以将 UI 素材从 SpiFlash 拷贝到 SDRAM，显示速度效果更好； 如不够，则可以定义多缓冲区，将要显示的 UI 先进行缓冲。

3.5、ISP 烧写方式的应用

B0 端口是 ISP 方式的触发条件，在正常工作状态应该接下拉电阻，有效过滤外界的干扰导致系统上电后进入 ISP 模式。当需要 ISP 烧写是，通过外围接入高电平，系统上电过程检测 B0 端口位高电平进入 ISP 状态，配合 C2/C3 端口 和 PC 上位机软件“SYNWIT_ISP_V3.1.2.exe”，可以进行程序的下载烧写。

四、软件设计

4.1、TFT-LCD 的初始化

部分 TFT-LCD 的应用需要进行通过 SPI 方式进行初始化，SPI 通讯的每个参数需要了解，TFT-LCD 模组厂提供的初始化参数大部分基于 Android 平台应用；或者初始化参数只是初始化了大部分参数，并没有设置打开 TFT-LCD 的显示。所以，需要着重查看初始化参数。

4.2、TFT-LCD 驱动时序的设置

参考 void lcd_rgb_init(void) 函数，设置背光、TFT-LCD 驱动数据端口的功能状态、水平像素、垂直像素嗲、Hfp、Hbp、Vfp、Vbp、像素时钟等等。此参数可参考 TFT-LCD 屏规格书中的描述进行设置，简单清晰。

注意：LCD 的 CLK 信号和 CLKDiv 相关，如图，主时钟为 120Mhz 的情况下，LCDCLK 频率在 8~12Mhz，那分频设置必须在 LCD_CLKDIV_14~LCD_CLKDIV_10,否则显示不正常。

6.1 TIMING CHARACTERISTICS

6.1.1 PRALLEL RGB INPUT TIMING TABLE

Item		Symbol	Values			Unit	Remark
			Min.	Typ.	Max.		
DCLK Frequency		Fclk	8	9	12	MHz	
Hsync	Period time	Th	485	531	-	DCLK	
	Display Period	Thdisp	-	480-	-	DCLK	
	Back Porch	Thbp	3	43	-	DCLK	
	Front Porch	Thfp	2	8	-	DCLK	
Vsync	Period time	Tv	276	292	-	H	
	Display Period	Tvdisp	-	272	-	H	
	Back Porch	Tvbp	2	12	-	H	
	Front Porch	Tvfp	2	8	-	H	

```
void lcd_rgb_init(void)
{
    LCD_InitStructure LCD_initStruct;

    GPIO_Init(GPIOM, PIN20, 1, 0, 0);
    GPIO_ClrBit(GPIOM, PIN20);
    swm_delay(1);
    GPIO_SetBit(GPIOM, PIN20);

//    GPIO_Init(GPIOB, PIN12, 1, 0, 0); //背光控制
//    GPIO_SetBit(GPIOB, PIN12); //点亮背光
//    _LCD_BACKLIGHT_PORT_INIT(); //背光控制
//    _LCD_BALKLIGHT_ON();       //点亮背光


    PORT->PORTN_SEL0 = 0xAAAAAAAA;   //GPION.0~15  LCD_DATA0~15
    PORT->PORTN_SEL1 = 0xAA;

    LCD_initStruct.Interface = LCD_INTERFACE_RGB;
    LCD_initStruct.HnPixel = LV_HOR_RES_MAX;
    LCD_initStruct.VnPixel = LV_VER_RES_MAX;
    LCD_initStruct.Hfp = 5;
    LCD_initStruct.Hbp = 40;
    LCD_initStruct.Vfp = 8;
    LCD_initStruct.Vbp = 8;
    LCD_initStruct.ClkDiv = LCD_CLKDIV_6;
    LCD_initStruct.ClkAlways = 0;
    LCD_initStruct.SamplEdge = LCD_SAMPLEDGE_FALL;
    LCD_initStruct.HsyncWidth = LCD_HSYNC_1DOTCLK;
    LCD_initStruct.IntEOTEn = 1;
    LCD_Init(LCD, &LCD_initStruct);
}
```

4.3、系统时钟设置

系统时钟建议设置在 120Mhz。参考 system_SWM320.c 文件中的时钟设置，如下图所示采用内部 20Mhz 的 RC 震荡源，通过 PLL 到 120Mhz。 也可采用外部晶振，通过内部 PLL 到 120Mhz，在设计时建议预留外部晶体的位置。

```
/******************************************************************************************************************
 * 系统时钟设定
 ******************************************************************************************************************/
#define SYS_CLK_20MHz 0   //0 内部高频20MHz RC振荡器
#define SYS_CLK_40MHz 1   //1 内部高频40MHz RC振荡器
#define SYS_CLK_32KHz 2   //2 内部低频32KHz RC振荡器
#define SYS_CLK_XTAL  3   //3 外部晶体振荡器(2-30MHz)
#define SYS_CLK_PLL   4   //4 片内锁相环输出

#define SYS_CLK   SYS_CLK_PLL


#define SYS_CLK_DIV_1 0
#define SYS_CLK_DIV_2 1

#define SYS_CLK_DIV     SYS_CLK_DIV_1


#define __HSI   (20000000UL)    //高速内部时钟
#define __LSI   (   32000UL)    //低速内部时钟
#define __HSE   (20000000UL)    //高速外部时钟


/********************************** PLL 设定 ***********************************************
 * VCO输出频率 = PLL输入时钟 / INDIV * 4 * FBDIV
 * PLL输出频率 = PLL输入时钟 / INDIV * 4 * FBDIV / OUTDIV = VCO输出频率 / OUTDIV
 ******************************************************************************************************************/
#define SYS_PLL_SRC     SYS_CLK_20MHz  //SYS_CLK_XTAL //SYS_CLK_20MHz //可取值SYS_CLK_20MHz、SYS_CLK_XTAL

#define PLL_IN_DIV    5

#define PLL_FB_DIV    60


#define PLL_OUT_DIV8  0
#define PLL_OUT_DIV4  1
#define PLL_OUT_DIV2  2

#define PLL_OUT_DIV   PLL_OUT_DIV8



uint32_t SystemCoreClock  = __HSI;          //System Clock Frequency (Core Clock)
uint32_t CyclesPerUs      = (__HSI / 1000000);    //Cycles per micro second
```

4.4、SDRAM 的设置

SDRAM 的时钟是系统时钟的 4 分频。如系统时钟是 120Mhz，SDRAM 的时钟是 30Mhz。参考 void lcd_memory_init(void)函数

```
void lcd_memory_init(void)
{
#ifdef SWM_USING_SRAM
    SRAM_InitStructure SRAM_InitStruct;

    PORT->PORTP_SEL0 = 0xAAAAAAAA; //PP0-23 => ADDR0-23
    PORT->PORTP_SEL1 = 0x0000AAAA;

    PORT->PORTM_SEL0 = 0xAAAAAAAA; //PM0-15 => DATA15-0
    PORT->PORTM_INEN = 0xFFFF;

    PORT->PORTM_SEL1 = 0xAAA; //PM16 => OEN、PM17 => WEN、PM18 => NORFL_CSN、PM19 => SDRAM_CSN、PM20 => SRAM_CSN、PM21 => 

    SRAM_InitStruct.ClkDiv = SRAM_CLKDIV_8; //主频40MHz时可选SRAM_CLKDIV_5，主频120MHz时一般选SRAM_CLKDIV_8
    SRAM_InitStruct.DataWidth = SRAM_DATAWIDTH_16;
    SRAM_Init(&SRAM_InitStruct);
#endif
#ifdef SWM_USING_SDRAM
    SDRAM_InitStructure SDRAM_InitStruct;

    PORT->PORTP_SEL0 = 0xAAAAAAAA; //PP0-23 => ADDR0-23
//    PORT->PORTP_SEL1 = 0x00000A0A;  //此处需要调整P23\P22\P19\P18的端口，这几个端口用作SpiFlash的通讯，其中P22是片选，片
                                      //如此处没有注意，先初始化SpiFlash，再初始化SDRAM的情况下；或应用中disable SDRAM,再
    PORT->PORTP_SEL1 = 0x00004A5A;

    PORT->PORTM_SEL0 = 0xAAAAAAAA; //PM0-15 => DATA15-0
    PORT->PORTM_INEN = 0xFFFF;
    //PM16 => OEN, PM17 => WEN, PM18 => NORFL_CSN, PM19 => SDRAM_CSN, PM20 => SRAM_CSN, PM21 => SDRAM_CKE
    PORT->PORTM_SEL1 = 0x888;

    SDRAM_InitStruct.CellSize = SDRAM_CELLSIZE_64Mb;
    SDRAM_InitStruct.CellBank = SDRAM_CELLBANK_4;
    SDRAM_InitStruct.CellWidth = SDRAM_CELLWIDTH_16;
    SDRAM_InitStruct.CASLatency = SDRAM_CASLATENCY_2;
    SDRAM_InitStruct.TimeTMRD = SDRAM_TMRD_3;
    SDRAM_InitStruct.TimeTRRD = SDRAM_TRRD_2;
    SDRAM_InitStruct.TimeTRAS = SDRAM_TRAS_6;
    SDRAM_InitStruct.TimeTRC  = SDRAM_TRC_8;
    SDRAM_InitStruct.TimeTRCD = SDRAM_TRCD_3;
    SDRAM_InitStruct.TimeTRP  = SDRAM_TRP_3;
    SDRAM_Init(&SDRAM_InitStruct);
#endif
}
```

以下设置 LCDC 模块从 SDRAM 区域取显示数据的的地址，可以理解为显示内存

```
#ifdef SWM_USING_SDRAM
static uint32_t lcdbuf_1[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE)))           = {0x00000000};
static uint32_t lcdbuf_2[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE + 0x3FC00))) = {0x00000000};
#endif
```

4.5、SD 卡的设置

```
uint8_t bsp_SDIOInit(void)
{
  uint8_t resSD;

  #if _SWM320VET7V16_ //SWM320VET7-50 SD GPIO 设置，与SDRAM的ADDR6~ADDR11共用
  PORT_Init(PORTP, PIN11, PORTP_PIN11_SD_CLK, 0);
  PORT_Init(PORTP, PIN10, PORTP_PIN10_SD_CMD, 1);
  PORT_Init(PORTP, PIN9, PORTP_PIN9_SD_D0,  1);
  PORT_Init(PORTP, PIN8, PORTP_PIN8_SD_D1,  1);
  PORT_Init(PORTP, PIN7, PORTP_PIN7_SD_D2,  1);
  PORT_Init(PORTP, PIN6, PORTP_PIN6_SD_D3,  1);
  #endif

  #if _SWM320WET7V10_ //SWM320WET7-40 SD GPIO 设置，这里用另外一组独立的SD GPIO
  PORT_Init(PORTB, PIN1, PORTB_PIN1_SD_CLK, 0);
  PORT_Init(PORTB, PIN2, PORTB_PIN2_SD_CMD, 1);
  PORT_Init(PORTB, PIN3, PORTB_PIN3_SD_D0,  1);
  PORT_Init(PORTB, PIN4, PORTB_PIN4_SD_D1,  1);
  PORT_Init(PORTB, PIN5, PORTB_PIN5_SD_D2,  1);
  PORT_Init(PORTB, PIN6, PORTB_PIN6_SD_D3,  1);
  #endif

  #if _SWM32SRET7V10_ //SWM32SRET7-50 SD GPIO 设置，这里用另外一组独立的SD GPIO
  PORT_Init(PORTB, PIN1, PORTB_PIN1_SD_CLK, 0);
  PORT_Init(PORTB, PIN2, PORTB_PIN2_SD_CMD, 1);
  PORT_Init(PORTB, PIN3, PORTB_PIN3_SD_D0,  1);
  PORT_Init(PORTB, PIN4, PORTB_PIN4_SD_D1,  1);
  PORT_Init(PORTB, PIN5, PORTB_PIN5_SD_D2,  1);
  PORT_Init(PORTB, PIN6, PORTB_PIN6_SD_D3,  1);
  #endif


  resSD = SDIO_Init();

  return resSD;
}
```

4.6、简单画线程序应用

在调试 TFT-LCD 过程中，参考以下的画线程序，先在屏幕上进行画线测试，如果可以正常画线，则可以人为驱动正常。

```
LCD_Start(LCD);
uint32_t x,y;
for (y=0; y<1; y++)
{
    for (x=0; x<480; x++)
    {
        _HW_DrawPoint(x,y,0xffff);
    }
}

while(1)
{
}
```

4.7、SpiFlash 应用于文件系统中大小设定

在应用文件系统对 SpiFlash 进行管理设计时，需要对 SpiFlash 进行格式化大小的定义管理。如最小系统板应用的是 SpiFlash 是 W25Q128，其有 16MB 的存储空间，在提供的例程应用中，只格式化了 6MB 的大小空间，剩余的空间可以由用户进行直接的应用处理，如下图所示。

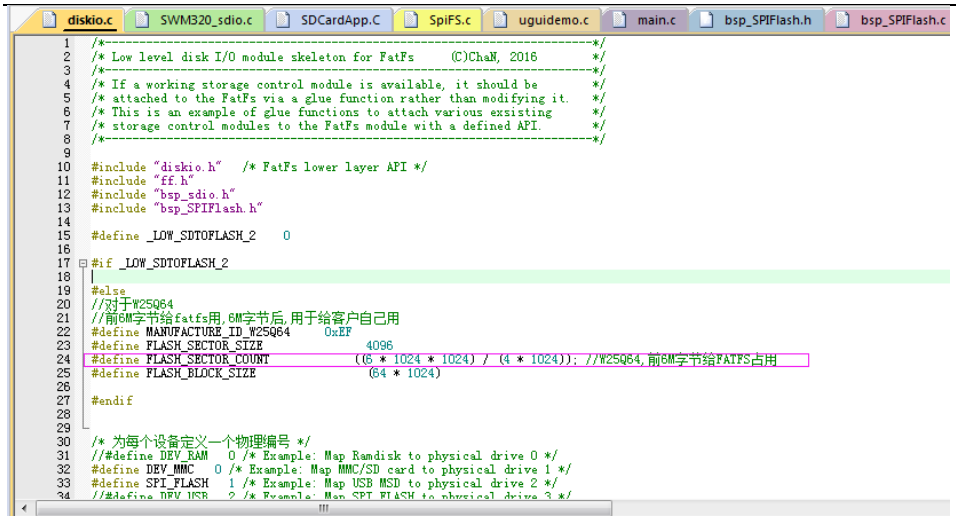
所以，用户应用例程时，当图片大小超过 6MB 时，图片显示会出现乱的现象。此时，用户可以进行如下更改：

```
#define FLASH_SECTOR_COUNT          ((6 * 1024 * 1024) / (4 * 1024));
```

更改为

```
#define FLASH_SECTOR_COUNT          ((16 * 1024 * 1024) / (4 * 1024));
```

或根据图片数量大小来进行调整。



4.8、LCD_Star(LCD)函数应用的注意事项

LCD_Star(LCD)函数是启动 SWM32S 的 LCDC 模块从显示中读取数据进行扫描的启动命令，在未取完数据的过程中不允许进行再次 LCD_Star(LCD)操作，否则会使得 LCDC 取数据出错，引起显示乱的现象，如白屏、黑屏。

正确的做法是如下：

4.8.1、在初始化工作完成后，进入 main()函数的 while 循环之前，启动一次 LCD_Star(LCD)；

```
148
149     LCD_Start(LCD);
150
151     while (1 == 1)
152     {
```

4.8.2、在 LCD_Handler(void) 应用中进行数据刷新的启动

```
60  void LCD_Handler(void)
61  {
62      LCD_INTClr(LCD);
63      LCD_Start(LCD);
64  }
65
```

在初始化 LCD_Star(LCD)启动后，后续的数据刷新都会在 LCD_Handler(void)启动后进行。

五、UI 设计应用

5.1、UI 过渡渐变色的设计和保存格式

驱动 TFT-LCD 的接口为 RGB565，即红色分辨率为 5bit，绿色分辨率为 6bit，蓝色分辨率为 5bit，即对灰阶的解析度比 RGB888 要稍逊，所以 UI 设计过程中渐变过渡色的处理不适合设计过多过细，稍作注意即可。

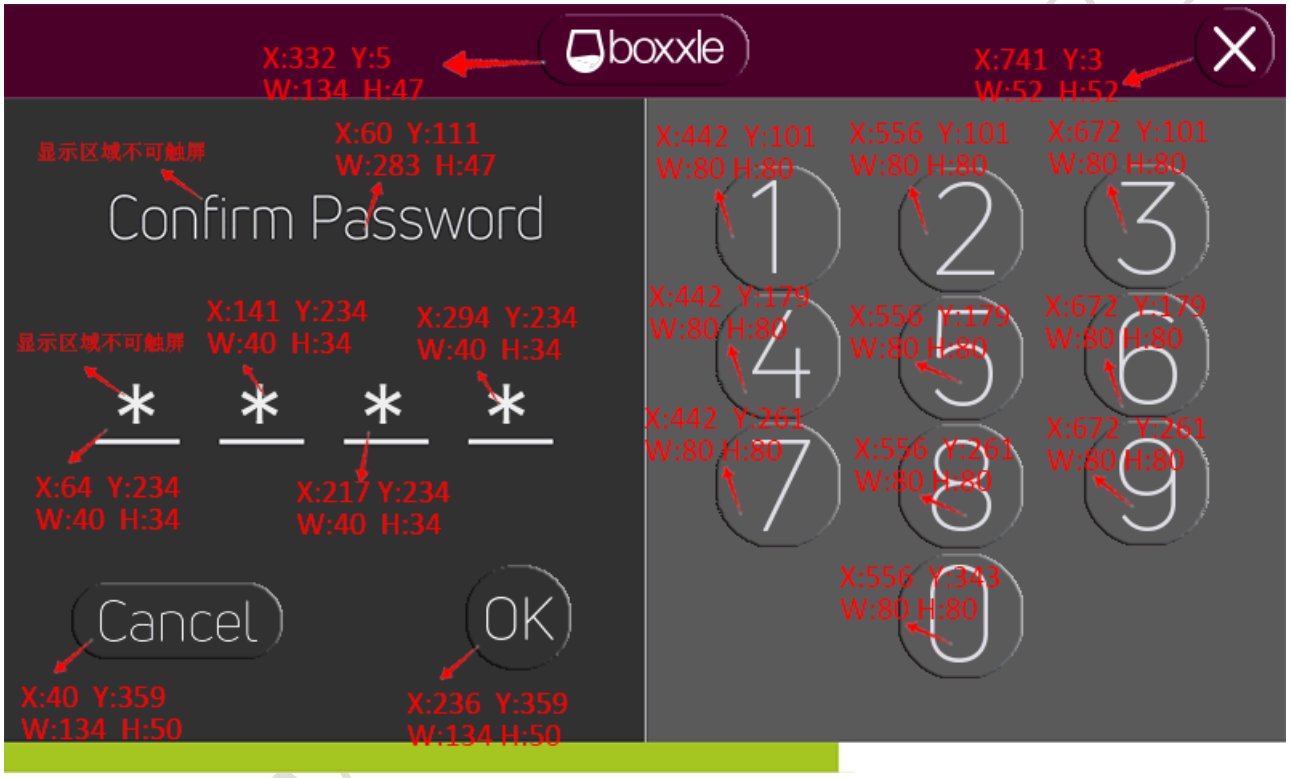
图片的保存格式采用 24bit 的 BMP 格式，在用辅助工具时可以进行图片的数据转换。

5.2、UI 图标的坐标位置设计

UI 设计时，将图标的坐标的水平起始位置定义在偶数像素点。如图 4.1 所示，右上角的“关闭”图标的坐标位置为“X:741 Y:3”，此处水平坐标为奇数，不符合规则，须进行调整。

5.3、分切 UI 图标大小规则

UI 设计时，进行图标分切时，需要将水平宽度的大小分切为偶数值，最好为 4 的倍数。如图 4.1 所示，左边的“Confirm Password”图标的大小“W:283 H:47”，此处水平宽度大小为奇数，不符合规则，须进行调整。



六、GUI 平台的应用

6.1、LittleVGL 的应用

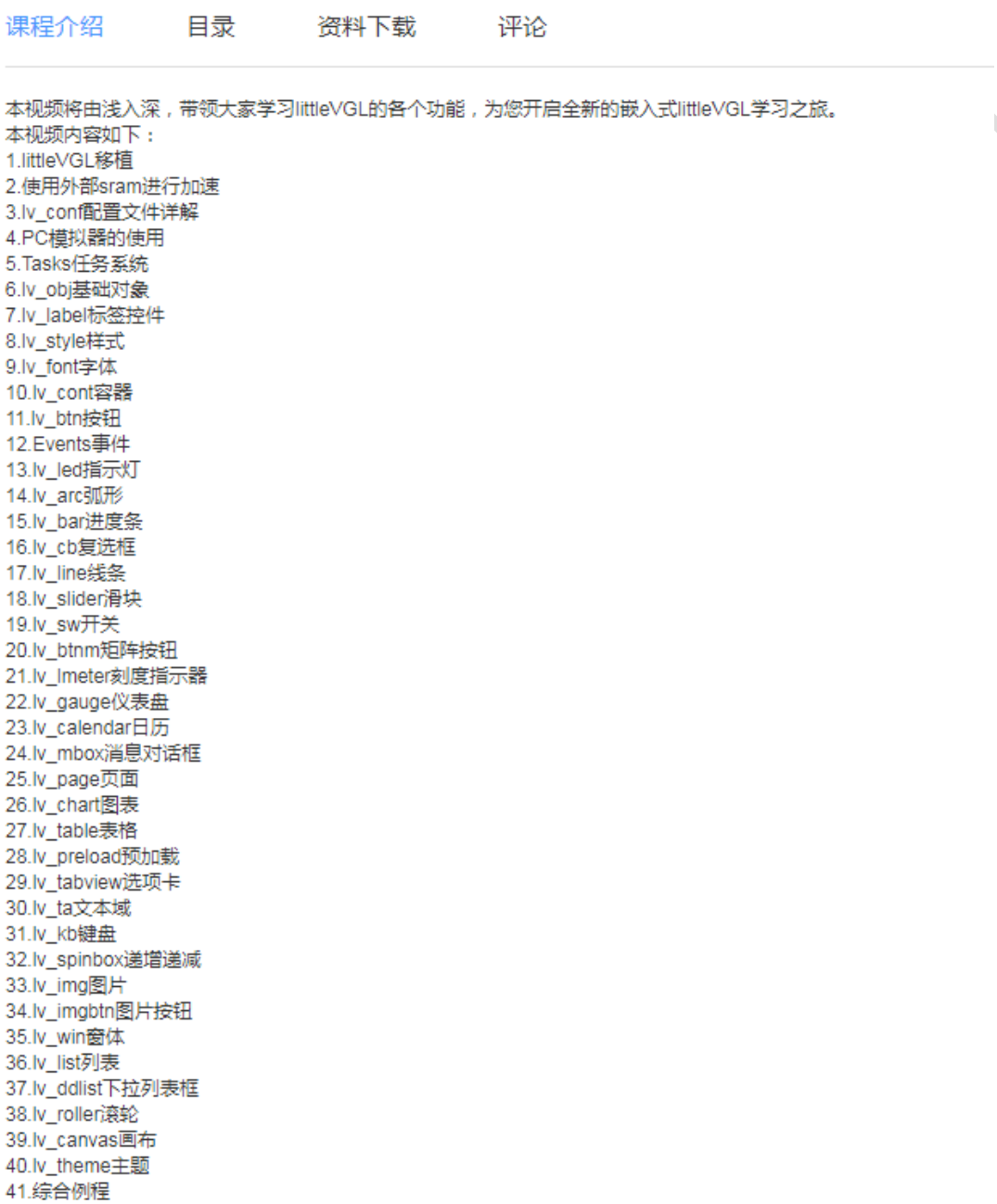
LittleVGL 的应用，可以参考 lvgl 的官网介绍和原子哥对 lvgl 的教学课，如下：

1)、官网首页 https://lvgl.io/

2)、有关文档介绍的官网链接 Introduction — LVGL documentation https://docs.lvgl.io/dev/en/html/intro/index.html

3)、原子哥的 lvgl 教学课程

手把手教你学 LittlevGL https://www.yuanzige.com/course/detail/80038



4)、Littlevgl 的转换工具参看如下链接

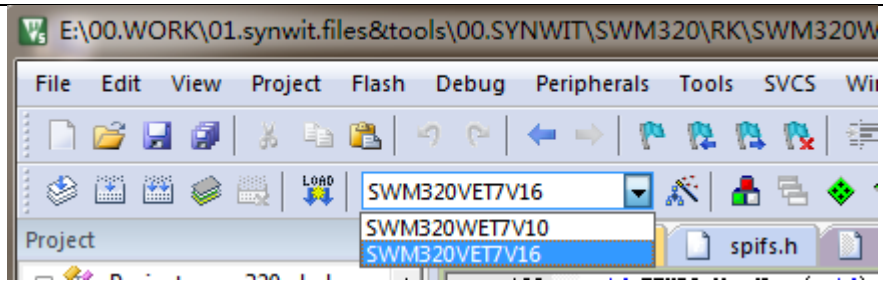
https://lvgl.io/tools/imageconverter

建议 PNG 格式图片用 lvgl 网站的转换工具，BMP 格式图片采用 Img2Lcd.exe

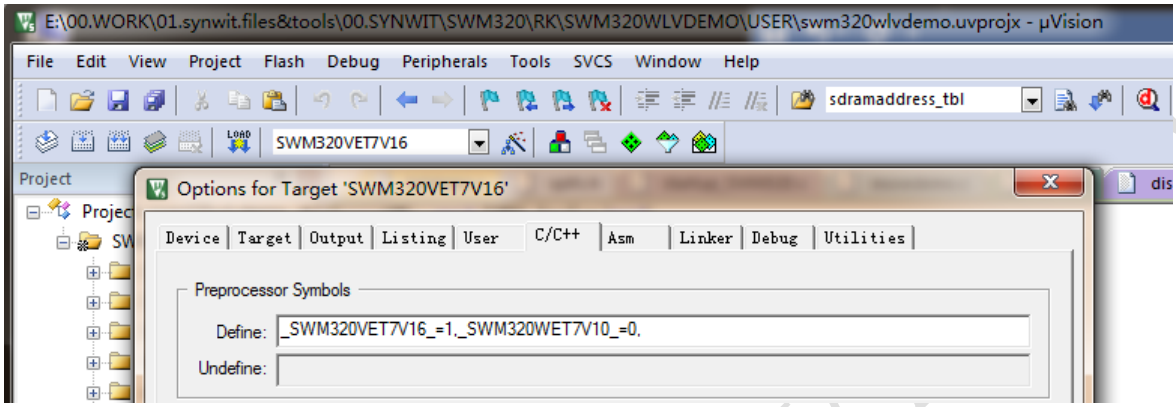
5)、参考文档“开源 GUI-littevGL 应用教程 V0.1.pdf”。

6)、例程工程“SWM320WLVDEMO”采用的是 lvgl V5.1 的版本进行的应，例程演示包括了几个：lvgl 自带的 DEMO，主要有 keyboard、text、list、page 等控件应用；PNG 上升箭头图片的叠图应用；BMP 图片叠图移动应用；叠图左进右出、右进左出等效果的应用，通过按键进行切换。

7)、例程工程“SWM320WLVDEMO”包含了 SWM320VET7-50 最小系统显示板 V1.6 和 SWM320WET7-40 最小系统显示板 V1.0 的硬件平台，通过下图进行选择。

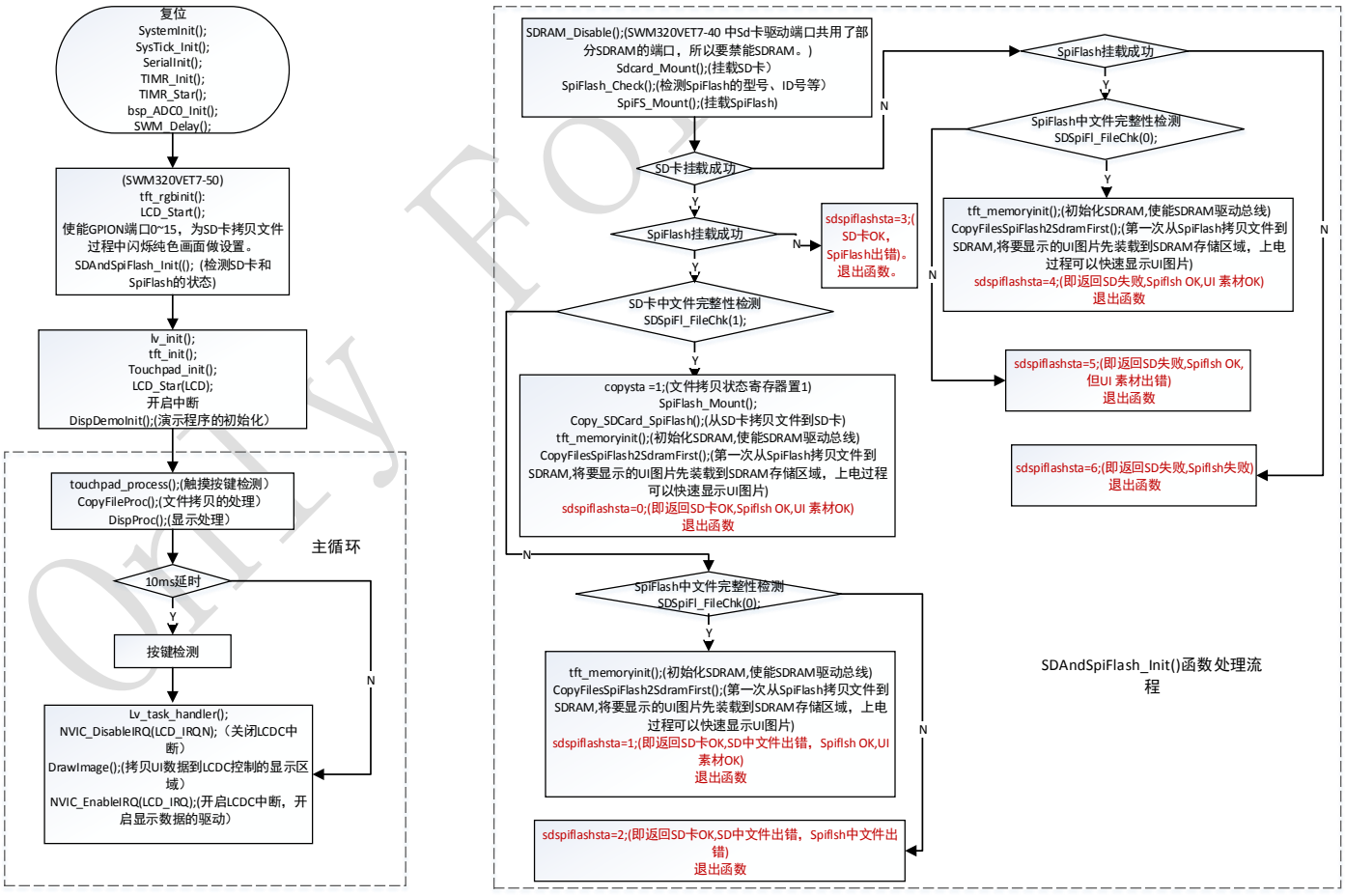


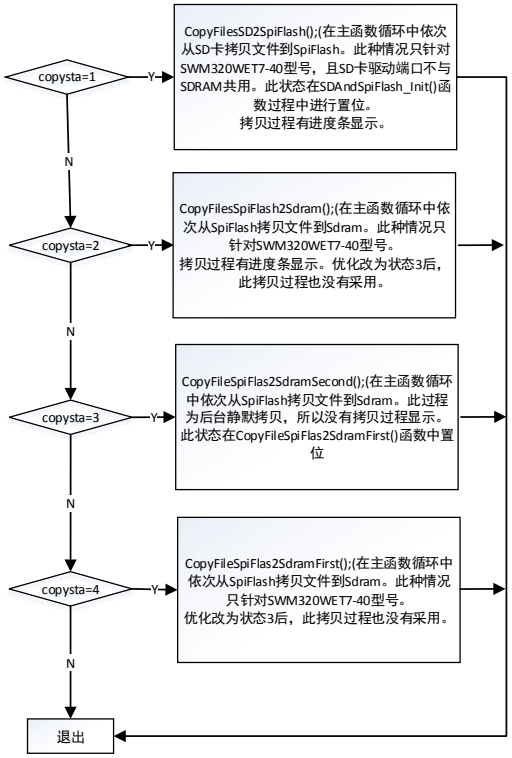
Prj -> “SWM320VET7V16” 和 “SWM320WET7V10” 的工程宏定义，如下：



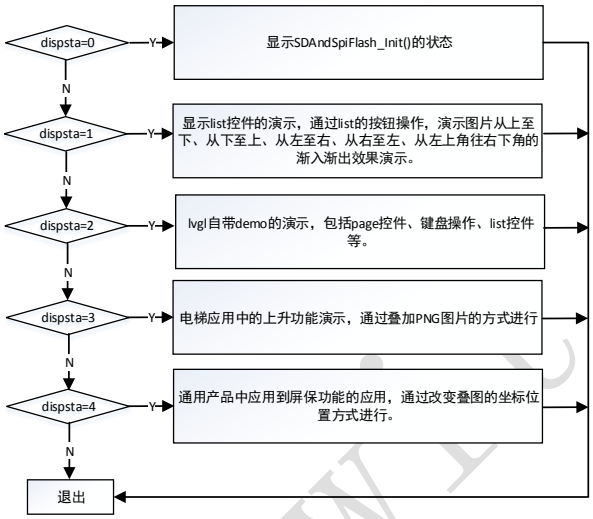
6.2、例程应用中素材拷贝处理逻辑

例程工程采用的是 4.3 寸 480x272 的 TFT-LCD（如正点原子 4.3 寸 TFT-LCD 模组），例程所用的 UI 图片的 BIN 文件，放置于 SD 卡根目录中。上电时，会检测 SD 卡中的 UI 文件是否完整，如完整会将 UI 文件拷贝到 SPIFLSH 中，然后再从 SPIFLASH 拷贝到 SDRAM 进行显示应用。流程如下以作参考：





CopyFileProc()函数处理流程



DispProc()函数处理流程，
dispsta的值通过按键进行切换

七、辅助工具的应用

SWM320VET7-50 / SWM32SRET6-50 中的 LCDC 模块时直接显存的 bin 数据进行显示，所以需要显示的图片素材转换成 BIN 的格式进行存储于 SDRAM 显示中。

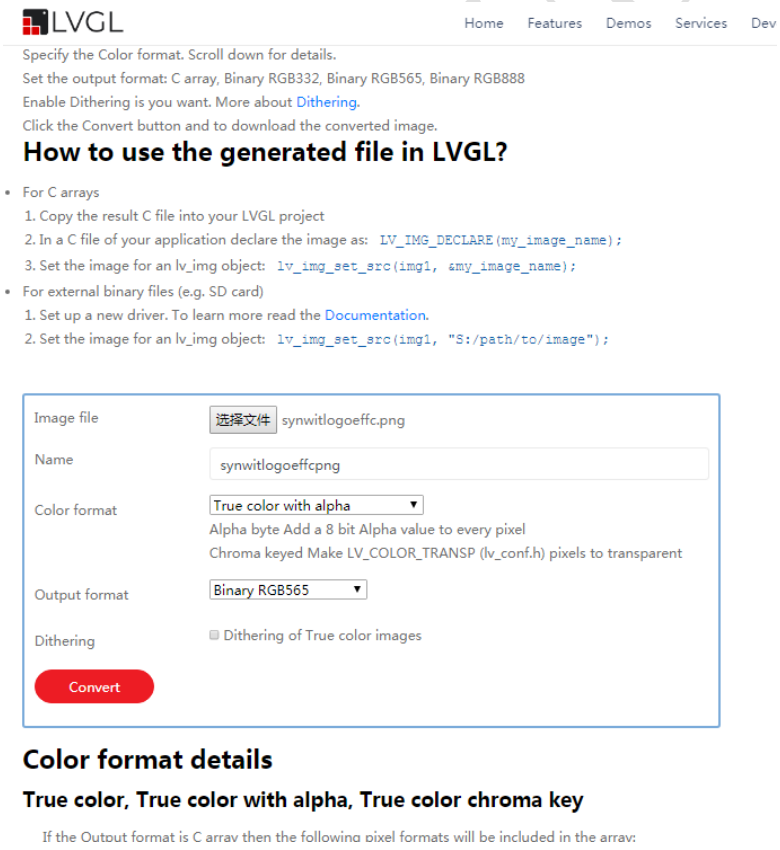
7.1、转换工具为 Img2Lcd.exe 软件，可以将 BMP 文件转换成 bin 格式。

参考上图设置❶❷❸❹❺选项，再点击“保存”，储存为转换工具默认的“*.ebm”文件格式后，可以直接更改文件后缀名为“*.bin”

7.2、LittleVGL 应用平台在线转换工具，参考如下链接：

https://lvgl.io/tools/imageconverter

PNG 图片格式转换参考



八、如何更改其它分辨率 TFT-LCD 模组的应用

以 LittleVGL 应用的演示为基础，参考“SWM32Slvgl612DEMO800480”工程，如驱动其它分辨率的 TFT-LCD 模组，需要改动以下几个参数：

8.1、调整 lv_conf.h 中的 LV_HOR_RES_MAX 和 LV_VER_RES_MAX 配置参数

lv_conf.h

```
/**
 * @file lv_conf.h
 *
 */

/*
 * COPY THIS FILE AS `lv_conf.h` NEXT TO the `lvgl` FOLDER
 */

#if 1 /*Set it to "1" to enable content*/

#ifndef LV_CONF_H
#define LV_CONF_H
/* clang-format off */

#include <stdint.h>

/*====================
   Graphical settings
 *====================*/
#define LV_DEMO_WALLPAPER   1
#define LV_USE_DEMO 1
/* Maximal horizontal and vertical resolution to support by the library.*/
#define LV_HOR_RES_MAX          (800)
#define LV_VER_RES_MAX          (480)
```

8.2、调整 lv_port_disp.c 中的显示缓存参数

调整 lv_port_disp.c 中的显示缓存参数，大小设置为 LV_HOR_RES_MAX * LV_VER_RES_MAX * 2 的值，如下图所示 800*480 的分辨率，显示缓存为 800*480*2 = 768000，即 0xBB800。

lv_port_disp.c

```
/**********************
/**********************
/**********************
static void disp_init(void);
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p);

/**********************
/**********************
/**********************
void lv_port_disp_init(void)
{
    /*-------------------------
    disp_init();

    /*-----------------------------
    /* LittlevGL requires a buffer where it draws the objects. The buffer's has to be greater than 1 display row
     *
     * There are three buffering configurations:
     * 1. Create ONE buffer with some rows:
     *      LittlevGL will draw the display's content here and writes it to your display
     *
     * 2. Create TWO buffer with some rows:
     *      LittlevGL will draw the display's content to a buffer and writes it your display.
     *      You should use DMA to write the buffer's content to the display.
     *      It will enable LittlevGL to draw the next part of the screen to the other buffer while
     *      the data is being sent form the first buffer. It makes rendering and flushing parallel.
     *
     * 3. Create TWO screen-sized buffer:
     *      Similar to 2) but the buffer have to be screen sized. When LittlevGL is ready it will give the
     *      whole frame to display. This way you only need to change the frame buffer's address instead of
     *      copying the pixels.
     * */

static lv_disp_buf_t disp_buf;
#ifdef SWM_USING_SRAM
static lv_color_t lcdbuf_1[LV_HOR_RES_MAX * LV_VER_RES_MAX] __attribute__((at(SRAMM_BASE)))          = {0x00000000};
static lv_color_t lcdbuf_2[LV_HOR_RES_MAX * LV_VER_RES_MAX] __attribute__((at(SRAMM_BASE + 0x3FC00))) = {0x00000000};
#endif
#ifdef SWM_USING_SDRAM
static uint32_t lcdbuf_1[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE)))          = {0x00000000};
static uint32_t lcdbuf_2[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE + 0xBB800))) = {0x00000000};
#endif
    lv_disp_buf_init(&disp_buf, lcdbuf_1, lcdbuf_2, LV_HOR_RES_MAX * LV_VER_RES_MAX); /*Initialize the display buffer*/
//    lv_disp_buf_init(&disp_buf_3, lcdbuf_1, NULL, LV_HOR_RES_MAX * LV_VER_RES_MAX); /*Initialize the display buffer*/
```

8.3、调整 LCDC 模块驱屏的参数。

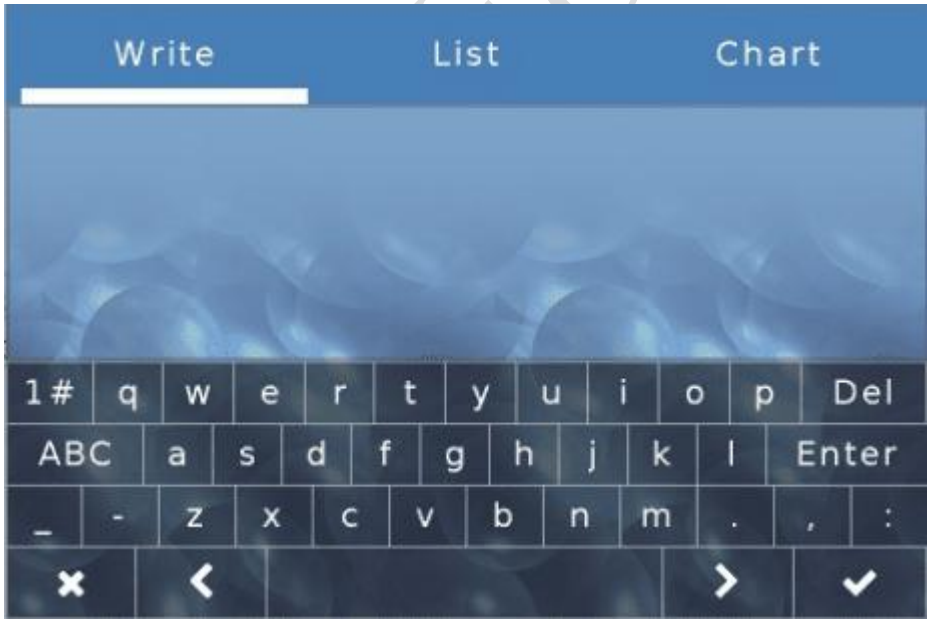
根据 TFT-LCD 模组的驱动时序，调整相应的参数。参考如下：

dev_rgblcd.c

```
#include "main.h"
#include "lv_conf.h"

void lcd_memory_init(void)
{
void lcd_rgb_init(void)
{
    LCD_InitStructure LCD_initStruct;

    GPIO_Init(GPIOM, PIN20, 1, 0, 0); //复位
    GPIO_ClrBit(GPIOM, PIN20);
    swm_delay(1);
    GPIO_SetBit(GPIOM, PIN20);

    GPIO_Init(GPIOB, PIN12, 1, 0, 0); //背光控制
    GPIO_SetBit(GPIOB, PIN12);        //点亮背光

    PORT->PORTN_SEL0 = 0xAAAAAAAA; //GPION.0~15  LCD_DATA0~15
    PORT->PORTN_SEL1 = 0xAA;

    LCD_initStruct.Interface = LCD_INTERFACE_RGB;
    LCD_initStruct.HnPixel = LV_HOR_RES_MAX;
    LCD_initStruct.VnPixel = LV_VER_RES_MAX;
    LCD_initStruct.Hfp = 5;
    LCD_initStruct.Hbp = 40;
    LCD_initStruct.Vfp = 8;
    LCD_initStruct.Vbp = 8;
    LCD_initStruct.ClkDiv = LCD_CLKDIV_4;
    LCD_initStruct.ClkAlways = 0;
    LCD_initStruct.SamplEdge = LCD_SAMPLEDGE_FALL;
    LCD_initStruct.HsyncWidth = LCD_HSYNC_1DOTCLK;
    LCD_initStruct.IntEOTEn = 1;
    LCD_Init(LCD, &LCD_initStruct);
}
```

8.4、LittleVGL 自带 DEMO

采用 LittleVGL 自带的 DEMO 例程，便于看到不依赖于图片的显示画面，如下图所示！



九、演示例程工程介绍

9.1、SWM32SLVGL612DEMO_202008210831.rar
SpiFlash 要求：W25Q32
演示程序如“图 8.1”~“8.4”

9.2、SWM32SLVGL612DEMO_202009241650.rar
SpiFlash 要求：W25Q128
演示程序如“图 8.1”~“8.5”

9.3、SWM32Slvgl612DEMO800480_202009271649.rar
不依赖于图片素材，如“图 8.3”。
此例程工程相当于纯净版的 LittleVGL 应用。

图 8.1

图 8.2

图 8.3

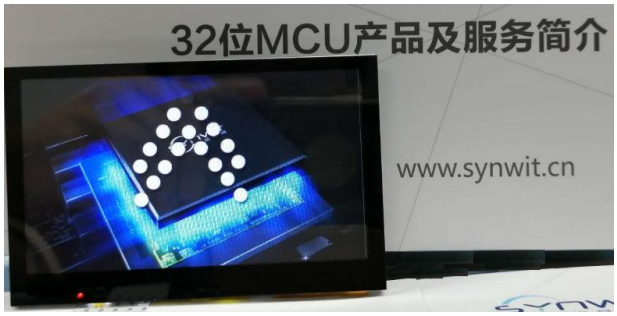


图 8.4

图 8.5

十、SWM32SRET6 LVGL 移植说明

10.1.SWM32SRET6 硬件资源介绍

Part Number	Flash	SRAM	IO	Timer	PWM	WDT	RTC	DMA	UART	I2C	SPI	CAN	ADC	EX-MEM	LCDC	SDIO	SDRAMC
SWM320CEU7-35	512	128	39	6	12	1	1	8	4	2	2	1	7(2)	0	0	0	0
SWM320RET7-50	512	128	50	6	12	1	1	8	4	2	2	1	11(2)	0	0	0	0
SWM320VET7-50	512	128	84	6	12	1	1	8	4	2	2	1	12(2)	1	1	1	1
SWM32SRET6-50	512	128	46	6	12	1	1	8	4	2	2	1	9(2)	1	1	1	1(stacked)

表格 3-1 SWM320 系列 MCU 选型表

10.1.1、 LCD 控制器（LCDC）

10.1.1.1 概述

本系列 LCDC 模块操作均相同，使用前需使能 LCDC 模块时钟。

LCDC 模块用于实现 MCU 与外部 LCD 的对接，在 MCU 的控制下，将需要显示的数据通过传送到外部 LCD 接口（支持 SYNC 的 LCD 接口）去显示。

10.1.1.2 特性

- 支持同步 LCD 接口
  - 接口时序可调
  - 输出时钟可配置为空闲时关闭
- 支持 565RGB 格式
- 支持最高分辨率 1024*768，实际分辨率可以配置
- LCDC 输出数据宽度 16bit
- 支持横屏和竖屏模式
- 内置单通道 DMA，FIFO 深度 32*32bit

```
void lcd_rgb_init(void)
{
    LCD_InitStructure LCD_initStruct;

    GPIO_Init(GPIOM, PIN20, 1, 0, 0); //复位
    GPIO_ClrBit(GPIOM, PIN20);
    swm_delay(1);
    GPIO_SetBit(GPIOM, PIN20);

    GPIO_Init(GPIOB, PIN12, 1, 0, 0); //背光控制
    GPIO_SetBit(GPIOB, PIN12);      //点亮背光

    PORT->PORTN_SEL0 = 0xAAAAAAAA; //GPION.0~15  LCD_DATA0~15
    PORT->PORTN_SEL1 = 0xAA;

    LCD_initStruct.Interface = LCD_INTERFACE_RGB;
    LCD_initStruct.HnPixel = LV_HOR_RES_MAX;
    LCD_initStruct.VnPixel = LV_VER_RES_MAX;
    LCD_initStruct.Hfp = 5;
    LCD_initStruct.Hbp = 40;
    LCD_initStruct.Vfp = 8;
    LCD_initStruct.Vbp = 8;
    LCD_initStruct.ClkDiv = LCD_CLKDIV_6;
    LCD_initStruct.ClkAlways = 0;
    LCD_initStruct.SamplEdge = LCD_SAMPLEDGE_FALL;
    LCD_initStruct.HsyncWidth = LCD_HSYNC_1DOTCLK;
    LCD_initStruct.IntEOTEn = 1;
    LCD_Init(LCD, &LCD_initStruct);
```

```
}
```

10.1.2、 SDRAM 控制器（SDRAMC）

10.1.2.1 概述

本系列所有型号 SDRAMC 模块操作均相同，主要功能在于完成 AHB 总线和外部 SDRAM 之间的数据搬移，使用前需使能 SDRAMC 模块时钟。模块支持标准 AHB 总线操作，仅支持 WORD 级别读写。

10.1.2.2 特性

- 仅支持 32 位 WORD 操作
- 支持 16bit 位宽的 SDRAM
- 支持兼容 PC133 标准的 SDRAM 颗粒
- 内部 8MB 的 SDRAM 颗粒

```
void lcd_memory_init(void)
{
  SDRAM_InitStructure SDRAM_InitStruct;

  PORT->PORTP_SEL0 = 0xAAAAAAAA; //PP0-23 => ADDR0-23
  PORT->PORTP_SEL1 &= ~0x00000F0F;
  PORT->PORTP_SEL1 |= 0x00000A0A;

  PORT->PORTM_SEL0 = 0xAAAAAAAA; //PM0-15 => DATA15-0
  PORT->PORTM_INEN = 0xFFFF;
  //PM16 => OEN,PM17 => WEN,PM18 => NORFL_CSN,PM19 => SDRAM_CSN,PM20 => SRAM_CSN,PM21 => SDRAM_CKE
  PORT->PORTM_SEL1 = 0x888;

  SDRAM_InitStruct.CellSize = SDRAM_CELLSIZE_64Mb;
  SDRAM_InitStruct.CellBank = SDRAM_CELLBANK_4;
  SDRAM_InitStruct.CellWidth = SDRAM_CELLWIDTH_16;
  SDRAM_InitStruct.CASLatency = SDRAM_CASLATENCY_2;
  SDRAM_InitStruct.TimeTMRD = SDRAM_TMRD_3;
  SDRAM_InitStruct.TimeTRRD = SDRAM_TRRD_2;
  SDRAM_InitStruct.TimeTRAS = SDRAM_TRAS_6;
  SDRAM_InitStruct.TimeTRC = SDRAM_TRC_8;
  SDRAM_InitStruct.TimeTRCD = SDRAM_TRCD_3;
  SDRAM_InitStruct.TimeTRP = SDRAM_TRP_3;
  SDRAM_Init(&SDRAM_InitStruct);
}
```

10.1.3、 SDIO 接口（SDIO）

10.1.3.1 概述

本系列 SDIO 模块操作均相同，部分型号可能不包含该模块。使用前需使能 SDIO 模块时钟。SDIO 模块控制器支持多媒体卡（MMC）、SD 存储卡、SDIO 卡等设备，可以使用软件方法或者 DMA 方法（SDIO 模块内部 DMA，与芯片 DMA 模块无关）进行数据传输。

10.1.3.2 特性

- 兼容 SD 主机控制标准规范 2.0
- 兼容 SDIO 卡规范 2.0
- 兼容 SD 存储卡规范 2.0（Draft 版本）
- 兼容 SD 存储卡安全规范 1.01
- 兼容 MMC 规范标准 3.31、4.2 和 4.3
- 支持 DMA 和非 DMA 操作两种模式
- 支持 MMC Plus 和 MMC Mobile

- 卡检测（插入/移除）
- 可变时钟频率：0~52MHz
- 支持 1 位、4 位、8 位的 SD 模式
- 支持多媒体卡中断模式
- 4 位 SD 模式下，传输速率高达 100Mbits/S
- 8 位 SD 模式下，传输速率高达 416Mbits/S
- 支持读写控制，暂停/恢复操作
- 支持 MMC4.3 卡纠错
- 支持 CRC 循环冗余校验

```
uint32_t sdcard_init(void)
{
  PORT_Init(PORTB, PIN1, PORTB_PIN1_SD_CLK, 0);
  PORT_Init(PORTB, PIN2, PORTB_PIN2_SD_CMD, 1);
  PORT_Init(PORTB, PIN3, PORTB_PIN3_SD_D0,  1);
  PORT_Init(PORTB, PIN4, PORTB_PIN4_SD_D1,  1);
  PORT_Init(PORTB, PIN5, PORTB_PIN5_SD_D2,  1);
  PORT_Init(PORTB, PIN6, PORTB_PIN6_SD_D3,  1);
  return SDIO_Init(30000000);
}
```

10.1.4 串行外设接口（SPI）控制器

10.1.4.1 概述

不同型号 SPI 数量可能不同。使用前需使能对应 SPI 模块时钟。

SPI 模块支持 SPI 模式及 SSI 模式。SPI 模式下支持 MASTER 模式及 SLAVE 模式。具备深度为 8 的 FIFO，速率及帧宽度可灵活配置。

10.1.4.2 特性

- 全双工串行同步收发
- 可编程时钟极性和相位
- 支持 MASTER 模式和 SLAVE 模式
- MASTER 模式下最高传输速度支持主时钟 4 分频
- 数据宽度支持 4BIT 至 16BIT
- 具备深度为 8 的接收和发送 FIFO

```
static void spi_configuration(spi_user_data_t spi)
{
  SPI_InitStructure SPI_initStruct;

  GPIO_Init(GPIOP, PIN22, 1, 0, 0);

  PORT_Init(PORTP, PIN23, FUNMUX1_SPI0_SCLK, 0);
  PORT_Init(PORTP, PIN18, FUNMUX0_SPI0_MOSI, 0);
  PORT_Init(PORTP, PIN19, FUNMUX1_SPI0_MISO, 1);

  SPI_initStruct.clkDiv = SPI_CLKDIV_4;
  SPI_initStruct.FrameFormat = SPI_FORMAT_SPI;
  SPI_initStruct.SampleEdge = SPI_SECOND_EDGE;
  SPI_initStruct.IdleLevel = SPI_HIGH_LEVEL;
  SPI_initStruct.WordSize = 8;
  SPI_initStruct.Master = 1;
  SPI_initStruct.RXHFullIEn = 0;
  SPI_initStruct.TXEmptyIEn = 0;
  SPI_initStruct.TXCompleteIEn = 0;
  SPI_Init(spi->spix, &SPI_initStruct);
  SPI_Open(spi->spix);
```

```
}
```

10.2 文件系统移植

10.2.1 SFUD 移植

10.2.1.1 SFUD 概述

SFUD 是一款开源的串行 SPI Flash 通用驱动库。由于现有市面的串行 Flash 种类居多，各个 Flash 的规格及命令存在差异，SFUD 就是为了解决这些 Flash 的差异现状而设计，让我们的产品能够支持不同品牌及规格的 Flash，提高了涉及到 Flash 功能的软件的可重用性及可扩展性，同时也可以规避 Flash 缺货或停产给产品所带来的风险。

10.2.1.2 SFUD 移植

移植文件位于`/sfud/port/sfud_port.c`，文件中的`sfud_err sfud_spi_port_init(sfud_flash *flash)`方法是库提供的移植方法，在里面完成各个设备 SPI 读写驱动（必选）、重试次数（必选）、重试接口（可选）及 SPI 锁（可选）的配置。更加详细的移植内容，可以参考 demo 中的各个平台的移植文件。

```
static spi_user_data spi0 = {.spix = SPI0, .cs_gpiox = GPIOP, .cs_gpio_pin = PIN22};
static void spi_configuration(spi_user_data_t spi)
{
    SPI_InitStructure SPI_initStruct;

    GPIO_Init(GPIOP, PIN22, 1, 0, 0);

    PORT_Init(PORTP, PIN23, FUNMUX1_SPI0_SCLK, 0);
    PORT_Init(PORTP, PIN18, FUNMUX0_SPI0_MOSI, 0);
    PORT_Init(PORTP, PIN19, FUNMUX1_SPI0_MISO, 1);

    SPI_initStruct.clkDiv = SPI_CLKDIV_4;
    SPI_initStruct.FrameFormat = SPI_FORMAT_SPI;
    SPI_initStruct.SampleEdge = SPI_SECOND_EDGE;
    SPI_initStruct.IdleLevel = SPI_HIGH_LEVEL;
    SPI_initStruct.WordSize = 8;
    SPI_initStruct.Master = 1;
    SPI_initStruct.RXHFullIEn = 0;
    SPI_initStruct.TXEmptyIEn = 0;
    SPI_initStruct.TXCompleteIEn = 0;
    SPI_Init(spi->spix, &SPI_initStruct);
    SPI_Open(spi->spix);
}

static void spi_lock(const sfud_spi *spi)
{
    __disable_irq();
}

static void spi_unlock(const sfud_spi *spi)
{
    __enable_irq();
}

/**
 * SPI write data then read data
 */
static sfud_err spi_write_read(const sfud_spi *spi, const uint8_t *write_buf, size_t write_size, uint8_t *read_buf,
                               size_t read_size)
```

```
{
    sfud_err result = SFUD_SUCCESS;
    uint8_t send_data, read_data;
    spi_user_data_t spi_dev = (spi_user_data_t)spi->user_data;

    if (write_size)
    {
        SFUD_ASSERT(write_buf);
    }
    if (read_size)
    {
        SFUD_ASSERT(read_buf);
    }

    GPIO_ClrBit(spi_dev->cs_gpiox, spi_dev->cs_gpio_pin);
    /* 开始读写数据 */
    for (size_t i = 0, retry_times; i < write_size + read_size; i++)
    {
        /* 先写缓冲区中的数据到 SPI 总线，数据写完后，再写 dummy(0xFF) 到 SPI 总线 */
        if (i < write_size)
        {
            send_data = *write_buf++;
        }
        else
        {
            send_data = SFUD_DUMMY_DATA;
        }
        /* 发送数据 */
        retry_times = 1000;
        while (SPI_IsTXFull(spi_dev->spix))
        {
            SFUD_RETRY_PROCESS(NULL, retry_times, result);
        }
        if (result != SFUD_SUCCESS)
        {
            goto exit;
        }
        SPI_Write(spi_dev->spix, send_data);
        /* 接收数据 */
        retry_times = 1000;
        while (SPI_IsRXEmpty(spi_dev->spix))
        {
            SFUD_RETRY_PROCESS(NULL, retry_times, result);
        }
        if (result != SFUD_SUCCESS)
        {
            goto exit;
        }
        read_data = SPI_Read(spi_dev->spix);
        /* 写缓冲区中的数据发完后，再读取 SPI 总线中的数据到读缓冲区 */
        if (i >= write_size)
        {
            *read_buf++ = read_data;
```

```
    }
  }

exit:
  GPIO_SetBit(spi_dev->cs_gpiox, spi_dev->cs_gpio_pin);

  return result;
}
```

10.2.2 FATFS 移植

10.2.2.1、FATFS 概述

FATFS 是一个通用的文件系统(FAT/exFAT)模块，用于在小型嵌入式系统中实现 FAT 文件系统。 FatFs 组件的编写遵循 ANSI C(C89)，完全分离于磁盘 I/O 层,因此不依赖于硬件平台。它可以嵌入到资源有限的微控制器中，如 8051, PIC, AVR, ARM, Z80, RX 等等，不需要做任何修改。

10.2.2.2、FATFS 移植

因为 FATFS 模块完全与磁盘 I/O 层分开，因此需要下面的函数来实现底层物理磁盘的读写与获取当前时间。底层磁盘 I/O 模块并不是 FATFS 的一部分，并且必须由用户提供。这些函数一般有 5 个，在 diskio.c 里面。

- disk_initialize
- disk_status
- disk_read
- disk_write
- disk_ioctl

```
/*-----------------------------------------------------------------------*/
/* 设备初始化                                                            */
/*-----------------------------------------------------------------------*/
DSTATUS disk_initialize(
  BYTE pdrv /* 物理编号 */
)
{
  DSTATUS status = STA_NOINIT;

  switch (pdrv)
  {
  case DEV_MMC: /* SD CARD */
    if (sdcard_init() == SD_RES_OK)
    {
      status = RES_OK;
    }
    break;
  case DEV_FLASH: /* SPI FLASH */
    if (sfud_init() == SFUD_SUCCESS)
    {
      status = RES_OK;
    }
    status = RES_OK;
    break;
  default:
    break;
  }
```

```
    return status;
}
```

```
/*-----------------------------------------------------------------------*/
/* 获取设备状态                                                          */
/*-----------------------------------------------------------------------*/
DSTATUS disk_status(
    BYTE pdrv /* 物理编号 */
)
{
    DSTATUS status = STA_NOINIT;

    switch (pdrv)
    {
    case DEV_MMC: /* SD CARD */
        status &= ~STA_NOINIT;
        break;
    case DEV_FLASH: /* SPI FLASH */
        status &= ~STA_NOINIT;
        break;
    default:
        break;
    }

    return status;
}
```

```
/*-----------------------------------------------------------------------*/
/* 读扇区：读取扇区内容到指定存储区                                      */
/*-----------------------------------------------------------------------*/
DRESULT disk_read(
    BYTE pdrv,    /* 设备物理编号(0..) */
    BYTE *buff,   /* 数据缓存区 */
    DWORD sector, /* 扇区首地址 */
    UINT count    /* 扇区个数(1..128) */
)
{
    DRESULT status = RES_PARERR;

    switch (pdrv)
    {
    case DEV_MMC: /* SD CARD */
        if (count == 1)
        {
            SDIO_BlockRead(sector, (uint32_t *)buff);
        }
        else
        {
            SDIO_MultiBlockRead(sector, count, (uint32_t *)buff);
        }
        status = RES_OK;
        break;
```

```
    case DEV_FLASH: /* SPI FLASH */
    {
        sfud_err result = SFUD_SUCCESS;
        sfud_flash *spi_flash = sfud_get_device(SFUD_W25_DEVICE_INDEX);
        result = sfud_read(spi_flash, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE, buff);
        if (result == SFUD_SUCCESS)
        {
            status = RES_OK;
        }
        else
        {
            status = RES_ERROR;
        }
    }
    break;
    default:
        break;
    }

    return status;
}
```

```
DRESULT disk_write(
    BYTE pdrv,      /* 设备物理编号(0..) */
    const BYTE *buff, /* 欲写入数据的缓存区 */
    DWORD sector,    /* 扇区首地址 */
    UINT count      /* 扇区个数(1..128) */
)
{
    DRESULT status = RES_PARERR;

    if (!count)
    {
        return RES_PARERR; /* Check parameter */
    }

    switch (pdrv)
    {
    case DEV_MMC: /* SD CARD */
        if (count == 1)
        {
            SDIO_BlockWrite(sector, (uint32_t *)buff);
        }
        else
        {
            SDIO_MultiBlockWrite(sector, count, (uint32_t *)buff);
        }
        status = RES_OK;
        break;
    case DEV_FLASH: /* SPI FLASH */
    {
        sfud_err result = SFUD_SUCCESS;
        sfud_flash *spi_flash = sfud_get_device(SFUD_W25_DEVICE_INDEX);
```

```
        sfud_erase_write(spi_flash, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE, buff);
        if (result == SFUD_SUCCESS)
        {
            status = RES_OK;
        }
        else
        {
            status = RES_ERROR;
        }
    }
    break;
    default:
        break;
    }

    return status;
}
```

```
DRESULT disk_ioctl(
    BYTE pdrv, /* 物理编号 */
    BYTE cmd,  /* 控制指令 */
    void *buff /* 写入或者读取数据地址指针 */
)
{
    DRESULT status = RES_PARERR;
    switch (pdrv)
    {
    case DEV_MMC: /* SD CARD */
        switch (cmd)
        {
        // Get R/W sector size (WORD)
        case GET_SECTOR_SIZE:
            *(WORD *)buff = 512;
            break;

        // Get erase block size in unit of sector (DWORD)
        case GET_BLOCK_SIZE:
            *(DWORD *)buff = SD_cardInfo.CardBlockSize;
            break;

        case GET_SECTOR_COUNT:
            *(DWORD *)buff = SD_cardInfo.CardCapacity / 512;
            break;

        case CTRL_SYNC:
        default:
            break;
        }

        status = RES_OK;
        break;
    case DEV_FLASH: /* SPI FLASH */
        switch (cmd)
```

```
    {
    // Get R/W sector size (WORD)
    case GET_SECTOR_SIZE:
      *(WORD *)buff = FLASH_SECTOR_SIZE;
      break;

    // Get erase block size in unit of sector (DWORD)
    case GET_BLOCK_SIZE:
      *(DWORD *)buff = 1;
      break;

    case GET_SECTOR_COUNT:
    {
      sfud_flash *spi_flash = sfud_get_device(SFUD_W25_DEVICE_INDEX);
      *(DWORD *)buff = spi_flash->chip.capacity / FLASH_SECTOR_SIZE;
    }
    break;

    case CTRL_SYNC:
    default:
      break;
    }

    status = RES_OK;
    break;
  default:
    status = RES_PARERR;
  }

  return status;
}
```

10.3、LittlevGL 移植

10.3.1、 LittlevGL 概述

LittlevGL 是一个免费的开放源代码图形库，它提供创建嵌入式 GUI 所需的一切，它具有易于使用的图形元素，精美的视觉效果和低内存占用。强大的构建块按钮，图表，列表，滑块，图像等，带有动画，抗锯齿，不透明度，平滑滚动的高级图形，各种输入设备的触摸板，鼠标，键盘，编码器等，多显示器支持，即同时使用更多的 TFT 和单色显示器，支持 UTF-8 编码的多语言，完全可定制的图形元素。

独立于任何微控制器或显示器使用的硬件，可扩展以使用较少的内存（80kB 闪存，12 kB RAM），支持操作系统，外部存储器和 GPU，但不是必需的，即使使用单帧缓冲区操作，也具有高级图形效果。

用 C 语言编写，以实现最大的兼容性（与 C++兼容），模拟器可在没有嵌入式硬件的 PC 上启动嵌入式 GUI 设计，快速 GUI 设计的教程，示例，主题，在线和离线文档，在 MIT 许可下免费和开源。

LittlevGL 官网：https://littlevgl.com

GitHub 地址：https://github.com/littlevgl

10.3.2、 LittlevGL 硬件要求

- 16、32 或 64 位微控制器或处理器
- 建议时钟频率大于 16MHz
- 闪存/ ROM：对于非常重要的组件，其大小大于 64 kB（建议大于 180 kB）
- 内存：
- 静态 RAM 使用量：大约 8 至 16 kB，具体取决于所使用的功能和对象类型
- 堆栈：大于 2kB（建议大于 4kB）
- 动态数据（堆）：大于 4 KB（如果使用多个对象，则建议大于 16 kB）。LV_MEM_SIZE 在 lv_conf.h 中设置

● 显示缓冲区：大于“水平分辨率”像素（建议大于 10× “水平分辨率”）
● C99 或更高版本的编译器
● 基本的 C（或 C ++）知识：指针，结构，回调

10.3.3、LittlevGL 移植

10.3.3.1、 屏幕介绍

开发板板载的是一个 RGB 接口的屏幕 JLT4301A，我们的板子使用的是 RGB565 接口。屏幕分辨率 480*272，显示方向为横向。

触摸采用的是 GT911 的电容触摸屏。

使用 RGB 屏，SDRAM 是必须的，因为 RGB 屏需要使用显存，例如 480*272 的 RGB565 屏幕，一个像素占用 2 字节的显存，总共需要 480*272*2=261120 折合 255KB 的显存，内部 RAM 很显然是不够用的。那么 RGB 屏的驱动只需要使能背光、配置 LCDC 的外设以及显存的地址就可以了。然后往屏幕填充内容就是往对应的显存发送数据就可以了。不同 RGB 屏的配置参数可能不一样，显示方向也不一样。

开发板使用的是电容触摸屏，使用 I2C 进行通信，利用 I2C 初始化触摸 IC GT911 后，我们就可以通过 I2C 读出触摸的绝对位置，跟屏幕是一一对应的。

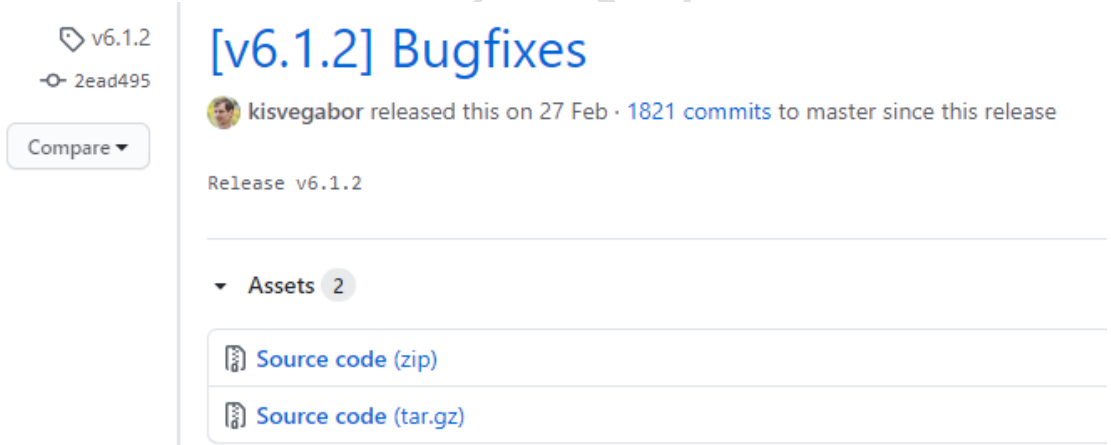
10.3.3.2、 移植流程

LittlevGL 的移植过程也非常简单，总结了以下几个步骤

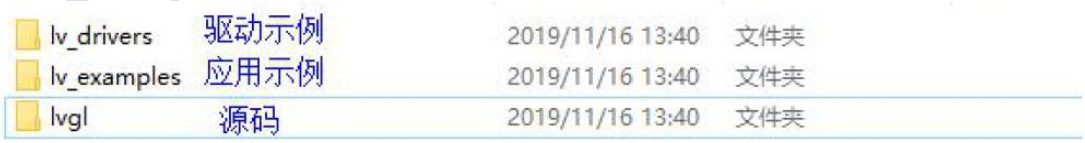
1. 添加库文件到工程
2. 配置屏幕大小以及颜色深度等跟显示相关的参数
3. 分配一个显示缓冲区并实现屏幕填充的接口
4. 实现输入设备接口，读取触摸屏坐标
5. 实现文件系统接口，实现文件的读取写入
6. 提供一个滴答时钟的接口 lv_tick_inc();
7. 完成库的初始化以及接口的初始化 lv_init();lv_port_disp_init();lv_port_indev_init(); lv_port_fs_init;
8. 定期调用任务处理函数，可设置为 5-10ms lv_task_handler();

10.3.3.3、 源码下载

LittlevGL 的源码可以在 GitHub 进行下载，https://github.com/littlevgl/lvgl，可以 clone 到本地也可以直接下载压缩包。除了下载源码以外，还可以下载 example 和 drivers。example 里面包含了各种应用展示和控件使用示例，drivers 里面包含了一些液晶屏驱动接口示例。



共下载 3 个文件夹



打开源码文件，里面包含了接口示例文件和配置示例文件，其中 src 文件夹下面又进行了分类，具体请查看源码。

10.3.3.4 添加库文件到工程

第一步，复制库文件

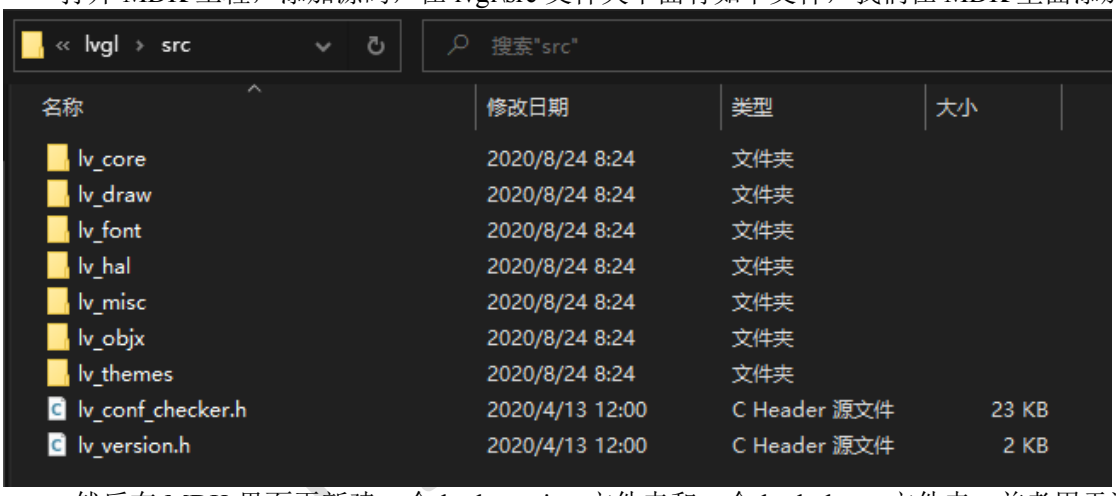
复制 lvgl 文件夹到工程文件夹 APP 下面。

将 lvgl\lv_conf_template.h 文件复制一份并改名为 lv_conf.h。这个文件是 lvgl 的配置文件，后面再介绍如何修改。

将 lv_examples 文件夹复制到工程文件夹下面，这里面包含了各种应用和基础示例，我们后面需要使用。

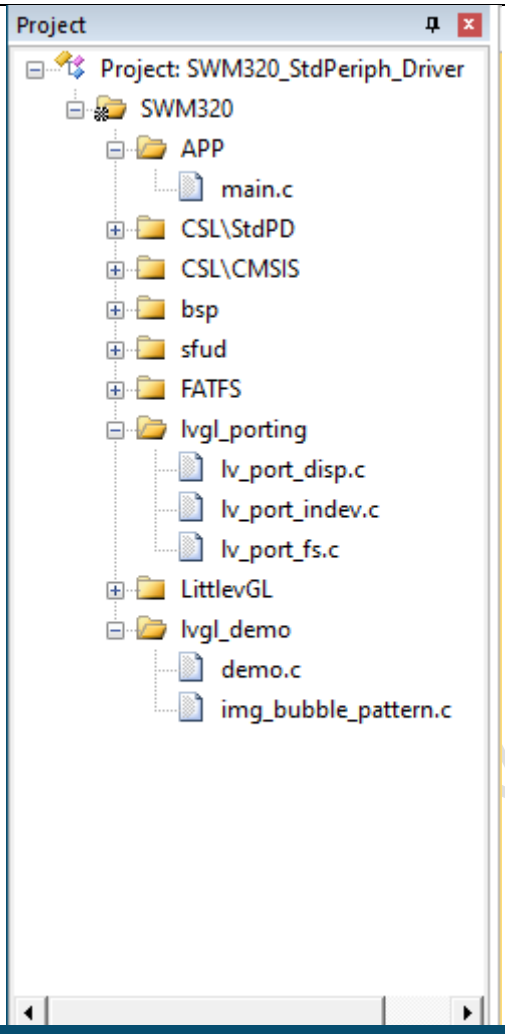
第二步，添加库文件到工程

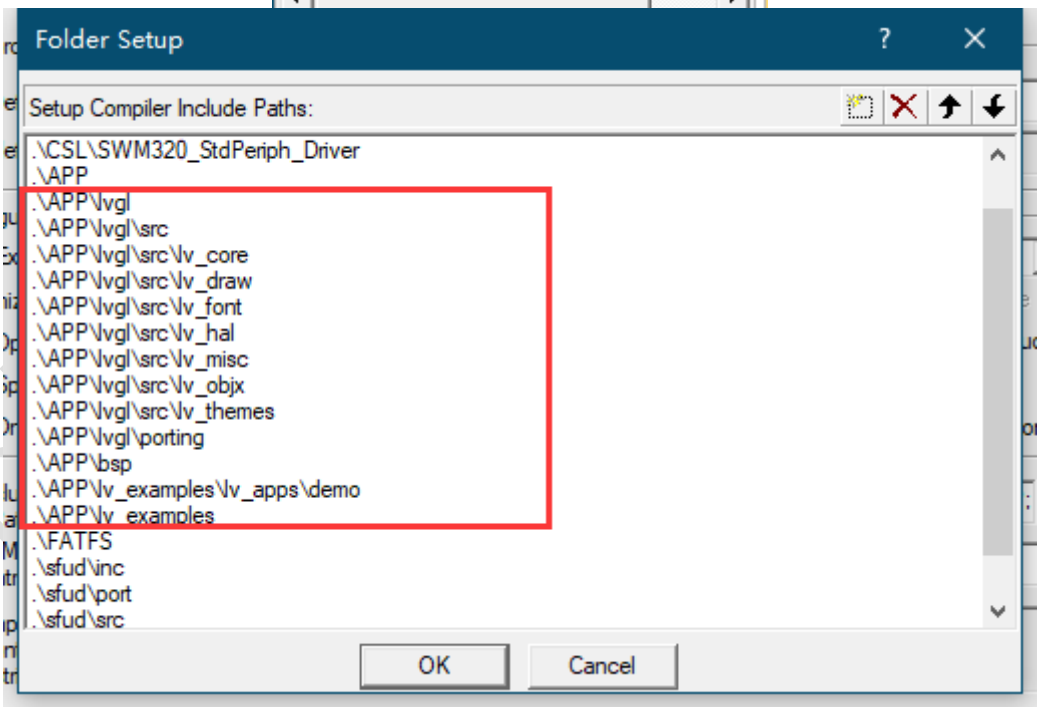
打开 MDK 工程，添加源码，在 lvgl\src 文件夹下面有如下文件，我们在 MDK 里面添加全部文件即可。



然后在 MDK 里面再新建一个 lvgl_porting 文件夹和一个 lvgl_demo 文件夹，前者用于添加接口文件，后者用于我们后面添加示例文件，将 lvgl\porting\lv_port_disp_template.c 和 lvgl\porting\lv_port_indev_template.c，lvgl\porting\lv_port_fs_template.c 各复制一份，分别改名为 lv_port_disp,c 和 lv_port_indev.c,lv_port_fs.c 添加到 MDK 的 lv_porting 文件夹

下面，后面再针对开发板的硬件进行对应的修改。





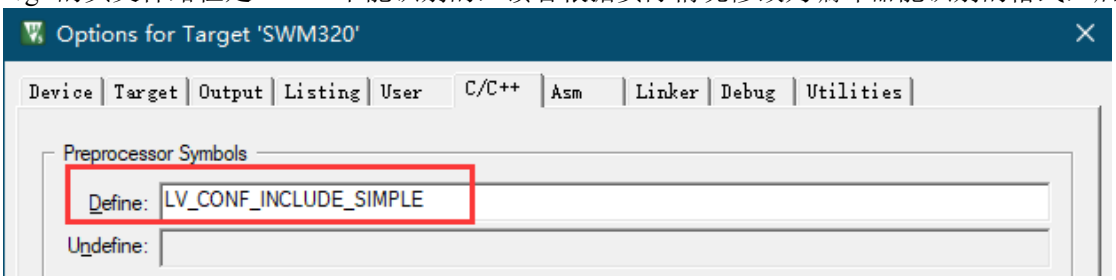
10.3.3.5 移植文件的适配

lvgl 的源码中包含头文件有完整路径和简单路径两种方式，在 MDK 里面我们直接使用简单的头文件包含形式。

```
/*********************
 *      INCLUDES
 *********************/
#ifdef LV_CONF_INCLUDE_SIMPLE
#include "lv_conf.h"
```

```
#else
#include "../../../lv_conf.h"
#endif
```

我们使用简单的头文件包含，在 MDK 的宏定义里面添加 LV_CONF_INCLUDE_SIMPLE 定义，如果后面还有 lvgl 的头文件路径是 MDK 不能识别的，读者根据实际情况修改为编译器能识别的格式，后面不再对此修改做介绍。



修改 lv_conf 文件，在我们之前复制过来的配置示例文件默认是被注释掉了。我们需要打开他。只需要将开头的#if 0 修改为#if 1 即可，如果后续的文件中再出现这种开关，读者自行打开即可。

```
/**
/*
 * COPY THIS FILE AS `lv_conf.h` NEXT TO the `lvgl` FOLDER
 */

#if 1 /*Set it to "1" to enable content*/

#ifndef LV_CONF_H
#define LV_CONF_H
/* clang-format off */

#include <stdint.h>
```

配置屏幕的大小，我们的液晶屏是 480*272

```
/* Maximal horizontal and vertical resolution to support by the library.*/
#define LV_HOR_RES_MAX          (480)
#define LV_VER_RES_MAX          (272)
```

配置颜色位数，我们使用 RGB565 模式，16 位色

```
/* Color depth:
 * - 1:  1 byte per pixel
 * - 8:  RGB233
 * - 16: RGB565
 * - 32: ARGB8888
 */
#define LV_COLOR_DEPTH     16

/* Swap the 2 bytes of RGB565 color.
 * Useful if the display has a 8 bit interface (e.g. SPI)*/
#define LV_COLOR_16_SWAP   0
```

LVGL 内存配置，这里区别与绘图缓冲区，这里配置的是 lvgl 的控件等使用的内存，需大于 2KB，这里可以使用默认的 32K 配置，也可以将内存定义 100K，从而分配更多的内存。

```
/* 1: use custom malloc/free, 0: use the built-in `lv_mem_alloc` and `lv_mem_free` */
#define LV_MEM_CUSTOM      0
#if LV_MEM_CUSTOM == 0
/* Size of the memory used by `lv_mem_alloc` in bytes (>= 2kB)*/
#  define LV_MEM_SIZE    (100 * 1024U)
```

GPU 配置，这里关闭，我们使用 LCD 中断进行 buffer 的填充

```
/* 1: Enable GPU interface*/
#define LV_USE_GPU              0
```

LOG 接口配置，这里暂未使用，读者在使用是可以将其配置为串口等

```
/*1: Enable the log module*/
#define LV_USE_LOG      0
#if LV_USE_LOG
/* How important log should be added:
 * LV_LOG_LEVEL_TRACE       A lot of logs to give detailed information
 * LV_LOG_LEVEL_INFO        Log important events
 * LV_LOG_LEVEL_WARN        Log if something unwanted happened but didn't cause a problem
 * LV_LOG_LEVEL_ERROR       Only critical issue, when the system may fail
 * LV_LOG_LEVEL_NONE        Do not log anything
 */
#  define LV_LOG_LEVEL    LV_LOG_LEVEL_WARN

/* 1: Print the log with 'printf';
 * 0: user need to register a callback with `lv_log_register_print_cb`*/
#  define LV_LOG_PRINTF   1
#endif  /*LV_USE_LOG*/
```

10.3.3.6、 配置显示接口

LittlevGL 绘制的过程需要有一个缓冲区 disp_buf,lvgl 内部将位图绘制到这个缓冲区，缓冲区满了以后调用 flush_cb 接口函数进行屏幕的填充，我们将这部分缓冲区的内容通过 LCD 中断搬运到液晶屏，当填充完成后，调用 lv_disp_flush_ready 函数通知库绘制已经完成，可以开始其他绘制过程。

LittlevGL 已经提供了一个示例文件，我们上面提到的复制文件也是使用他提供的文件，只需要我们修改几个接口函数即可。

定义 lvgl 绘制的缓冲区，这里需定义在外部 SDRAM 定义两个全屏的缓冲区。

```
static lv_disp_buf_t disp_buf;
#ifdef SWM_USING_SRAM
static lv_color_t lcdbuf_1[LV_HOR_RES_MAX * LV_VER_RES_MAX] __attribute__((at(SRAMM_BASE)))           = {0x00000000};
static lv_color_t lcdbuf_2[LV_HOR_RES_MAX * LV_VER_RES_MAX] __attribute__((at(SRAMM_BASE + 0x3FC00))) = {0x00000000};
#endif
#ifdef SWM_USING_SDRAM
static uint32_t lcdbuf_1[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE)))           = {0x00000000};
static uint32_t lcdbuf_2[LV_HOR_RES_MAX * LV_VER_RES_MAX / 2] __attribute__((at(SDRAMM_BASE + 0x3FC00))) = {0x00000000};
#endif
    lv_disp_buf_init(&disp_buf, lcdbuf_1, lcdbuf_2, LV_HOR_RES_MAX * LV_VER_RES_MAX); /*Initialize the display buffer*/
```

定义一个 lv_disp_drv_t 的变量并初始化。

```
    lv_disp_drv_t disp_drv;      /*Descriptor of a display driver*/
    lv_disp_drv_init(&disp_drv); /*Basic initialization*/
```

设置缓冲区。

```
    /*Set a display buffer*/
    disp_drv.buffer = &disp_buf;
```

设置屏幕填充接口，这里的 disp_fluash 是一个函数，在示例中已经定义，我们直接对其修改就行了。

```
    /*Used to copy the buffer's content to the display*/
    disp_drv.flush_cb = disp_flush;
```

注册驱动程序。

```
    /*Finally register the driver*/
    lv_disp_drv_register(&disp_drv);
```

修改 disp_flush 函数以适应我们自己的硬件和屏幕。

```
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p)
{
    LCD->SRCADDR = (uint32_t)disp_drv->buffer->buf_act;
    LCD_Start(LCD);

    /* IMPORTANT!!!
```

```
     * Inform the graphics library that you are ready with the flushing*/
    lv_disp_flush_ready(disp_drv);
}
```

10.3.3.7、 配置输入设备接口

lvgl 支持键盘、鼠标、按键、触摸屏作为输入设备，我们这里仅仅使用触摸屏进行输入。

在 lv_port_indev_template.c 中，注册一个触摸屏设备需要以下步骤，我们只需要对触摸屏读取的回调函数进行修改即可。

```
    /*Register a touchpad input device*/
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
    indev_touchpad = lv_indev_drv_register(&indev_drv);
```

打开触摸屏读取的回调函数 touchpad_read 函数

```
/* Will be called by the library to read the touchpad */
static bool touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    /*Save the pressed coordinates and the state*/
    if (touchpad_is_pressed())
    {
        touchpad_get_xy(&last_x, &last_y);
        data->state = LV_INDEV_STATE_PR;
    }
    else
    {
        data->state = LV_INDEV_STATE_REL;
    }

    /*Set the last pressed coordinates*/
    data->point.x = last_x;
    data->point.y = last_y;

    /*Return `false` because we are not buffering and no more data to read*/
    return false;
}
```

从 touchpad_read 函数可以看出，用户只需要完成两个接口函数即可。这两个函数的实现如下：

```
/*Return true is the touchpad is pressed*/
static bool touchpad_is_pressed(void)
{
    /*Your code comes here*/
    if(tp_dev.sta & 0x80)
    {
        return true;
    }
    return false;
}

/*Get the x and y coordinates if the touchpad is pressed*/
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y)
{
```

```
    /*Your code comes here*/

    (*x) = tp_dev.x[0];
    (*y) = tp_dev.y[0];
}
```

103.3.8 配置文件系统接口

文件系统使用 FATFS，对接前需要完成文件系统的挂载。

文件系统注册。

```
void lv_port_fs_init(void)
{
    /*-------------------------------------------------
     * Initialize your storage device and File System
     * ------------------------------------------------*/
    fs_init();

    /*-------------------------------------------------
     * Register the file system interface  in LittlevGL
     *------------------------------------------------*/

    /* Add a simple drive to open images */
    lv_fs_drv_t fs_drv;
    lv_fs_drv_init(&fs_drv);

    /*Set up fields...*/
    fs_drv.file_size     = sizeof(file_t);
    fs_drv.letter        = 'P';
    fs_drv.open_cb       = fs_open;
    fs_drv.close_cb      = fs_close;
    fs_drv.read_cb       = fs_read;
    fs_drv.write_cb      = fs_write;
    fs_drv.seek_cb       = fs_seek;
    fs_drv.tell_cb       = fs_tell;
    fs_drv.free_space_cb = fs_free;
    fs_drv.size_cb       = fs_size;
    fs_drv.remove_cb     = fs_remove;
    fs_drv.rename_cb     = fs_rename;
    fs_drv.trunc_cb      = fs_trunc;

    fs_drv.rddir_size    = sizeof(dir_t);
    fs_drv.dir_close_cb = fs_dir_close;
    fs_drv.dir_open_cb  = fs_dir_open;
    fs_drv.dir_read_cb  = fs_dir_read;

    lv_fs_drv_register(&fs_drv);
}
```

10.3.3.9 配置 LittlevGL 的嘀嗒心跳时钟接口

LittlevGL 的使用需要需要周期性的时钟支持，用户需要定期调用 lv_tick_inc(uint32_t tick_period)函数，我们这里利用滴答定时器的 1KHz 去调用这个函数。在滴答定时器的中断服务函数中添加 lvgl 时基函数。

```
uint8_t tick_indev = 0;
void SysTick_Handler_cb(void)
{
```

```
    lv_tick_inc(1);
    tick_indev++;
    if (tick_indev > 100)
    {
        tick_indev = 0;
        GT911_Scan();
    }
}
```

10.3.3.10 初始化

包含 lvgl 的初始化以及显示和触摸，文件系统接口的初始化。

```
    lv_init();
    lv_port_disp_init();
    lv_port_indev_init();
    lv_port_fs_init();
```

我们需要在主循环中周期性调用 LittelvGL 的任务处理函数。

```
    while (1 == 1)
    {
        lv_task_handler();
    }
```

然后就可以进行应用的开发了。

10.3.4 LVGL 叠图显示的基础应用

10.3.4.1 创建图片结构体 lv_img_dsc_t ,赋值 alpha 值，宽，高，大小，色彩格式，以及数据所在的地址。

```
lv_img_dsc_t my_image1 = //刷背景图用到的函数
{
.header.always_zero = 0,
.header.w = 480,
.header.h = 272,
.data_size = 480*272 * LV_COLOR_SIZE / 8,
.header.cf = LV_IMG_CF_TRUE_COLOR,
.data = (uint8_t *)(0x70000000  + _SYNWITLOGO1E_) //0x7F800 BF400 (783360
};

lv_img_dsc_t uparrow_png = //刷png向上箭头用到的函数
{
.header.always_zero = 0,
.header.w = 190,
.header.h = 210,
.data_size = 190*210 * LV_COLOR_SIZE / 8,
.header.cf = LV_IMG_CF_TRUE_COLOR_ALPHA,
.data = (uint8_t *)(0x70000000 + 0x4 + 0x177000)
};
```

10.3.4.2 创建图片对象 lv_img_create ，确立当前屏幕 lv_scr_act（）为父系。
lv_img_set_src( )把图片结构体的数据装载到创建好的图片对象中。
lv_obj_set_pos( )对应的图片对象设置到当前屏幕的 x,y 轴坐标（左上角为原点）。

```
    wp = lv_img_create(lv_scr_act(), NULL);//背景图
    lv_img_set_src(wp, &my_image1); //背景图片设置，不设置位置默认在0，0作为起始坐标
    //lv_obj_set_width(wp, LV_HOR_RES * 4);

      //lv_obj_set_protect(wp, LV_PROTECT_POS);

    //lv_obj_t *img_func = lv_img_create(lv_scr_act(), NULL);//背景图

      //lv_obj_del(img_func);


      obj_uparrow = lv_img_create(lv_scr_act(), NULL); //PNG图位置
      lv_obj_set_pos(obj_uparrow, 140, 0);      /*Align next to the source image*/
    }
```

10.3.4.3 通过反复调用 lv_img_set_src( )，以及改变图片数据源，把固定图片结构体，装载不同的数据到创建好的图片对 象中。

```
      math ++;
      if (math > 9)
      {
        bg ++;
//        if (bg > 4)
        if (bg > 1)
          bg = 0;
        math = 0;
      }
      //uparrow_png.pixel_map = (uint8_t *)(0x70000000 + 0x4 +  PNG1  + math*119704); //显示png图用的
      uparrow_png.data = (uint8_t *)(0x70000000 + 0x4 + _PNG1_ + math*119704); //显示png图用的 _PNG1_是基地址

      lv_img_set_src(obj_uparrow,&uparrow_png);
      //my_image1.pixel_map = (uint8_t *)(0x70000000 +  SYNWITLOGO1E  + bg*261120);
      my_image1.data = (uint8_t *)(0x70000000 + _SYNWITLOGO1E_ + bg*261120);//_SYNWITLOGO1E_是基地址
      lv_img_set_src(wp, &my_image1); //背景图片设置
    }
```

编写记录

日期	版本	描述	备注
2020-08-21	20200821	整理 SWM32SxET6-xx	rk
2020-09-30	20200930	增加开发板原理图 增加背光驱动板原理图 增加 TFT-LCD 接口介绍 增加驱动其它分辨率的应用指引 增加例程工程的描述 增加 LVGL 移植说明	rk lk
2020-10-22	20201022	修改 LCDC 端口分布、其它端口表格中管脚的描述问题	rk
2020-11-2	20201102	补充了 LCD 时钟信号频率的说明 增加 LVGL 基本叠图显示函数的说明	lzc
2020-12-18	20201218	更新 DEMO 演示板原理图	rk
2020-12-21	20201221	增加 SD 卡端口互用的应用 -- page11 增加 ISP 烧写方式的应用 -- page11 增加 SWM32SRET6-50 v3.0 最小系统板板介绍 -- page7 ~ page9	rk
2021-02-04	20210204	增加 LCD_Star(LCD)的应用注意事项 – page20	rk