

## 目录

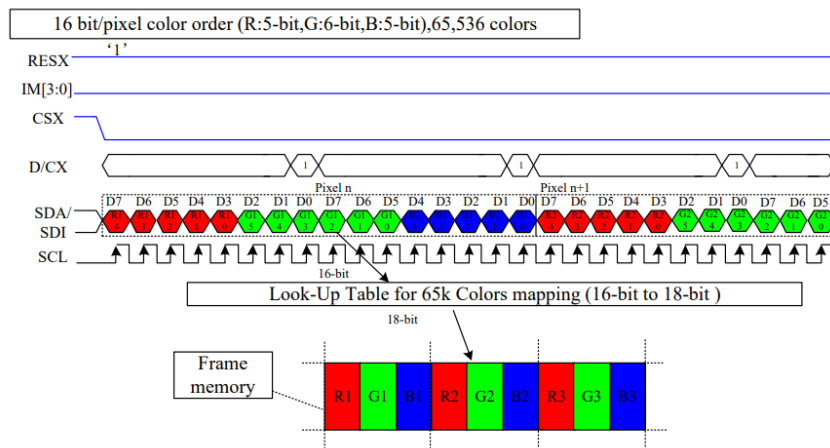
目录.....	1
1 基础介绍 .....	2
1.1 TFT-lcd 屏介绍 .....	2
1.2 TFT-lcd 屏使用流程图 .....	2
2 取模软件 .....	3
2.1 使用步骤 .....	3
3 代码实现 .....	4
3.1 外设初始化 .....	4
3.2 TFT-LCD 屏初始化.....	5
3.3 数据搬运分析 .....	6
4 更改记录 .....	7

## 1 基础介绍

### 1.1 TFT-LCD 屏介绍

TFT-LCD 液晶显示屏是薄膜晶体管型液晶显示屏，也就是“真彩”（TFT）。TFT 液晶为每个像素都设有一个半导体开关，每个像素都可以通过点脉冲直接控制，因而每个节点都相对独立，并可以连续控制，不仅提高了显示屏的反应速度，同时可以精确控制显示色阶，所以 TFT 液晶的色彩更真。

不同 TFT 屏使用的驱动芯片不一样，但其显示功能的接口函数是互通的。这里以 GC9A01 为例。GC9A01 液晶控制器自带显存，其显存总大小为 172800byte（240\*320\*18/8），即 18 位模式（26 万色）下的显存量。在 16 位模式下，GC9A01 采用 RGB565 格式存储颜色数据，最低为 5 位代表蓝色，中间 6 位为绿色，最高 5 位为红色。数值越大表示改颜色越深。如图所示：

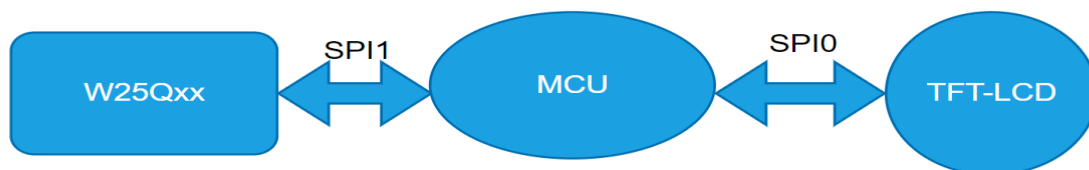


不同的 TFT-LCD 模块支持的通讯方式不同，这款 TFT-LCD 模块采用 SPI 的通讯方式，也是 TFT-LCD 屏常用通讯方式之一。在使用 TFT-LCD 屏时，单片机内部的 FLASH 不够，通常会采用外置 FLASH 来存储图片数据。下面将以 GC9A01 采用 RGB565 模式为例，采用 DMA 配合 SPI，实现从外置 FLASH 读出图片数据，搬运到 TFT-LCD 屏显示。（此 TFT-LCD 模块的像素大小为 240\*240）

### 1.2 TFT-LCD 屏使用流程图



TFT-LCD 使用流程图



硬件连接示意图

## 2 取模软件

将需要显示的图片，利用取模软件按照一定的格式输出取模数据。

### 2.1 使用步骤

以 Image2Lcd 软件为例：

#### 1. 导入图片。



#### 2. 输出数据类型选择。

选择的项：“二进制(\*.bin)”，“C 语言数组(\*.c)”，“BMP 格式(\*.bmp)”，“WBMP 格式(\*.wbmp)”。

常用的是 C 语言数组和二进制。

#### 3. 选择扫描模式。

图中第二个红框是扫描模式的附加选项。



#### 4. 输出灰度选择。

可以选择“单色/4 灰/16 灰/256 色/4096 色/16 位真彩色/18 位真彩色/24 位真彩色/32 位真彩色”中的一种。  
示例选择的 16 位真彩色，颜色位数 16bit。



#### 5. 最大宽度和高度设置。

设置图片大小 (和 LCD 的规格长宽像素点有关)

#### 6. 高位在前需勾选。

#### 7. 点击保存，完成取模。

## 3 代码实现

### 3.1 外设初始化

#### 1. SPI0、DMA0 初始化

```

32  /* PA3、PA7、PA14设置为推挽输出 */
33  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_7 | GPIO_Pin_14;
34  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
35  GPIO_InitStruct.GPIO_DriveLevel = 0;
36  GPIO_Init(GPIOA, &GPIO_InitStruct);
37
38  /* 配置SPI0为主机 */
39  SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge; //SPI第二沿采集数据
40  SPI_InitStruct.SPI_CPOL = SPI_CPOL_High; //SPI空闲电平为高电平
41  SPI_InitStruct.SPI_DataSize = SPI_DataSize_8B; //数据长度为8位
42  SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB; //数值传输为高位在前
43  SPI_InitStruct.SPI_Mode = SPI_Mode_Master; //SPI工作在主机模式
44  SPI_InitStruct.SPI_Prescaler = SPI_Prescaler_2; //SPI传输频率为APB0时钟2分频
45  SPI_Init(SPI0, &SPI_InitStruct);
46
47  /* 使能SPI0 */
48  SPI_Cmd(SPI0, ENABLE);
49
50  /* DMA外设配置 */
51  DMA_InitStruct.DMA_BufferSize = 16; //DMA搬运数量为16个
52  DMA_InitStruct.DMA_CircularMode = DMA_CircularMode_Disable; //关闭循环模式
53  DMA_InitStruct.DMA_DataSize = DMA_DataSize_Byte; //传输宽度为字节
54  DMA_InitStruct.DMA_SourceMode = DMA_SourceMode_INC_CIRC; //源地址循环递增
55  DMA_InitStruct.DMA_TargetMode = DMA_TargetMode_FIXED; //目标地址固定
56  DMA_InitStruct.DMA_SrcAddress = (uint32_t)0;
57  DMA_InitStruct.DMA_DstAddress = (uint32_t)&SPI0->SPI_DATA;
58  DMA_InitStruct.DMA_Request = DMA_Request_SPI0_TX; //SPI0 发送触发DMA0
59  DMA_InitStruct.DMA_Priority = DMA_Priority_HIGH; //设置优先级为高
60  DMA_Init(DMA0, &DMA_InitStruct);
61 }
62

```

## 2.SPI1、DMA1、DAM2 初始化

```
277 void SPI_Flash_Init(void)
278 {
279     GPIO_InitTypeDef GPIO_InitStructure;
280     SPI_InitTypeDef SPI_InitStructure; //定义SPI初始化结构体变量
281     DMA_InitTypeDef DMA_InitStructure; //定义DMA初始化结构体变量
282     /* 使能SPI1和DMA的时钟 */
283     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA, ENABLE);
284     RCC_APB1Cmd(ENABLE); //SPI1外设挂载在APB1时钟上
285
286     /* 结构体成员初始化 */
287     DMA_StructInit(&DMA_InitStructure);
288
289     /* PB6设置为推挽输出 */
290     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
291     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT_PP;
292     GPIO_InitStructure.GPIO_DriveLevel = 0;
293     GPIO_Init(GPIOB, &GPIO_InitStructure);
294
295     /* 配置SPI1为主机 */
296     SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge; //SPI第一沿采集数据
297     SPI_InitStructure.SPI_CPOL = SPI_CPOL_High; //SPI空闲电平为低电平
298     SPI_InitStructure.SPI_DataSize = SPI_DataSize_8B; //数据长度为8位
299     SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB; //数值传输为高位在前
300     SPI_InitStructure.SPI_Mode = SPI_Mode_Master; //SPI工作在主机模式
301     SPI_InitStructure.SPI_Prescaler = SPI_Prescaler_2; //SPI传输频率为APB1时钟1分频
302     SPI_Init(SPI1, &SPI_InitStructure);
303     /* 使能SPI1 */
304     SPI_PinsRemapConfig(SPI1, SPI_PinsRemap_A);
305     SPI_Cmd(SPI1, ENABLE);
306
307     /* DMA外设配置 */
308     DMA_InitStructure.DMA_BufferSize = 16; //DMA搬运数量为16个
309     DMA_InitStructure.DMA_CircularMode = DMA_CircularMode_Disable; //关闭循环模式
310     DMA_InitStructure.DMA_DataSize = DMA_DataSize_Byte; //传输宽度为字节
311     DMA_InitStructure.DMA_SourceMode = DMA_SourceMode_Fixed; //源地址固定
312     DMA_InitStructure.DMA_TargetMode = DMA_TargetMode_Fixed; //目标地址固定
313     DMA_InitStructure.DMA_SrcAddress = (uint32_t)0;
314     DMA_InitStructure.DMA_DstAddress = (uint32_t)&SPI1->SPI_DATA;
315     DMA_InitStructure.DMA_Request = DMA_Request_SPI1_RX; //SPI1_RX 接收触发DMA1
316     DMA_InitStructure.DMA_Priority = DMA_Priority_High; //设置优先级为高
317     DMA_Init(DMA1, &DMA_InitStructure);
318
319     /* DMA外设配置 */
320     DMA_InitStructure.DMA_BufferSize = 16; //DMA搬运数量为16个
321     DMA_InitStructure.DMA_CircularMode = DMA_CircularMode_Disable; //关闭循环模式
322     DMA_InitStructure.DMA_DataSize = DMA_DataSize_Byte; //传输宽度为字节
323     DMA_InitStructure.DMA_SourceMode = DMA_SourceMode_Fixed; //源地址固定
324     DMA_InitStructure.DMA_TargetMode = DMA_TargetMode_Fixed; //目标地址固定
325     DMA_InitStructure.DMA_SrcAddress = (uint32_t)&SPI1->SPI_DATA;
326     DMA_InitStructure.DMA_DstAddress = (uint32_t)&SPI0->SPI_DATA;
327     DMA_InitStructure.DMA_Request = DMA_Request_SPI1_RX; //SPI1_RX 接收触发DMA2
328     DMA_InitStructure.DMA_Priority = DMA_Priority_VERY_High; //设置优先级为非常高
329     DMA_Init(DMA2, &DMA_InitStructure);
330 }
331
```

注意：SPI0、SPI1 的频率应该相等，DMA2 的优先级要高于 DMA1。

## 3.2 TFT-LCD 屏初始化

不同类型的 TFT-LCD 屏幕的初始化写入的数据可能不同。主要是对功能寄存器写入数值进行初始化设置。（初始化代码一般都包含在屏幕资料里）

```
1 void LCD_GC9A01_Init(void)
2 {
3     LCD_Peripherals_Init(); //初始化GPIO SPI
4
5     LCD_RES_Clr(); //复位
6     delay_ms(10);
7     LCD_RES_Set();
8     delay_ms(10);
9
10
11     LCD_WR_REG(0xFE);
12     LCD_WR_REG(0xEF);
13
14     LCD_WR_REG(0x84);
15     LCD_WR_DATA8(0x40);
16
17
18     LCD_WR_REG(0xB6);
19     LCD_WR_DATA8(0x00);
20     LCD_WR_DATA8(0x20);
21
22     LCD_WR_REG(0x36);
23     LCD_WR_DATA8(0xC8);
24
25     LCD_WR_REG(0x3A);
26     LCD_WR_DATA8(0x05);
27
28     LCD_WR_REG(0xC3);
29     LCD_WR_DATA8(0x13);
30     LCD_WR_REG(0xC4);
31     LCD_WR_DATA8(0x13);
32     LCD_WR_REG(0xC9);
33     LCD_WR_DATA8(0x22);
34
35
36     LCD_WR_REG(0xF0);
37     LCD_WR_DATA8(0x45);
38     LCD_WR_DATA8(0x09);
39     LCD_WR_DATA8(0x08);
40     LCD_WR_DATA8(0x08);
41     LCD_WR_DATA8(0x26);
42     LCD_WR_DATA8(0x2A);
43
44     LCD_WR_REG(0xF1);
45     LCD_WR_DATA8(0x43);
46     LCD_WR_DATA8(0x70);
47     LCD_WR_DATA8(0x72);
48     LCD_WR_DATA8(0x36);
49     LCD_WR_DATA8(0x37);
50
```



### 3.3 数据搬运分析

```

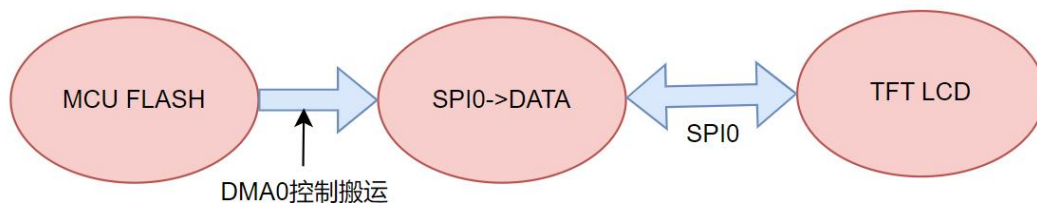
550 *****
551 void LCD_ShowPicture(uint16_t x, uint16_t y, uint16_t length, uint16_t width, const uint8_t pic[])
552 {
553     uint32_t LCD_num = length * width * 2;
554     SPI_DMACmd(SPI0, SPI_DMAReq_TX, DISABLE); //关闭上一次 SPI DMA传输
555     LCD_CS_Set();
556     LCD_Address_Set(x, y, x + length - 1, y + width - 1);
557     LCD_CS_Clr();
558     DMA_SetSrcAddress(DMA0, (uint32_t)pic);
559     DMA_SetCurrDataCounter(DMA0, LCD_num); //设置DMA传输数
560     SPI_DMACmd(SPI0, SPI_DMAReq_TX, ENABLE); //开启 SPI DMA传输
561     DMA_Cmd(DMA0, ENABLE); //使能DMA0
562     DMA_SoftwareTrigger(DMA0);
563 }
564
565

```

此函数的作用是：图片数据数组存储在 MCU 内部时，从数组搬运数据到 TFT-LCD 屏显示。

实现方法：

设置 DMA0 传输数，开启 SPI0 的 DMA 传输，软件触发一次 DMA0 搬运，DMA0 搬运完 1byte，会产生一次 SPI0\_TX 的 DMA 请求，DMA0 会继续搬运直到 DMA0 传输数为 0，停止搬运。



DMA0 搬运数据示意图

```

192 void SPI_Flash_Read(uint32_t ReadAddr, uint32_t Len)
193 {
194     uint8_t tmp = 0;
195     SPI_DMACmd(SPI1, SPI_DMAReq_TX | SPI_DMAReq_RX, DISABLE); //关闭上一次 SPI DMA传输
196     CS = 1; //取消片选
197     LCD_CS_Set();
198     LCD_Address_Set(0, 0, 239, 239);
199     LCD_CS_Clr();
200
201     while(W25Q16_ReadStatus() & 0x01); //判断是否忙
202     WriteEnable();
203     CS = 0; //使能器件
204     SPI_WriteByte(W25_ReadDATA8); //发送读取命令
205     SPI_WriteByte((uint8_t)(ReadAddr >> 16)); //发送24bit地址
206     SPI_WriteByte((uint8_t)(ReadAddr >> 8));
207     SPI_WriteByte((uint8_t)ReadAddr);
208
209     DMA_SetSrcAddress(DMA1, (uint32_t)&tmp);
210     DMA_SetCurrDataCounter(DMA1, Len); //设置DMA传输数
211     DMA_SetCurrDataCounter(DMA2, Len); //设置DMA传输数
212     SPI_DMACmd(SPI1, SPI_DMAReq_TX | SPI_DMAReq_RX, ENABLE); //开启 SPI DMA传输
213     DMA_Cmd(DMA1, ENABLE); //使能DMA1
214
215     /* 开启DMA2传输完成中断 */
216     DMA_ITConfig(DMA2, DMA_IT_INTEN | DMA_IT_TCIE, ENABLE);
217     NVIC_EnableIRQ(DMA2_IRQn);
218     DMA_Cmd(DMA2, ENABLE); //使能DMA2
219     DMA_SoftwareTrigger(DMA1);
220 }

```

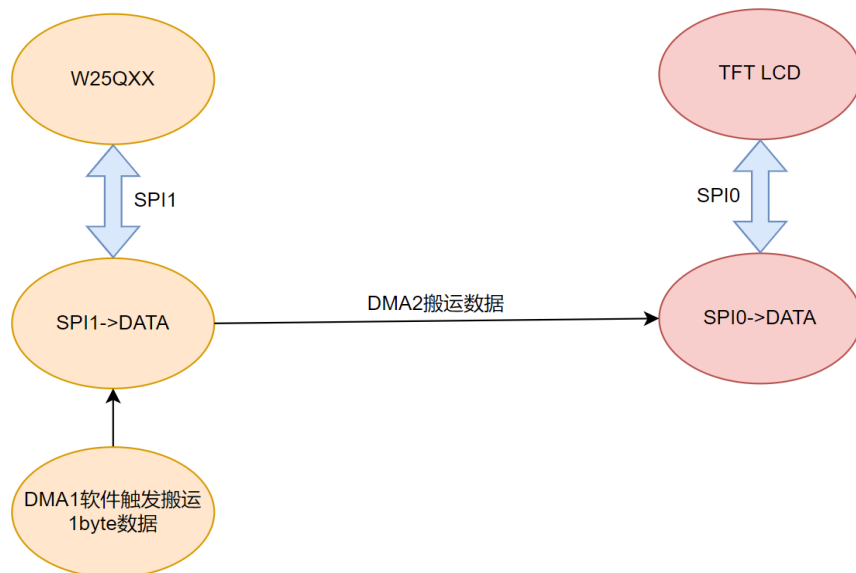
此函数的作用是：从外置 FLASH 读出数据，再传输给 TFT-LCD 屏显示。

实现的方法：

使用到 DMA1 和 DMA2，DMA2 的优先级高于 DMA1，确保数据是先从外置 FLASH 读出后再传输到 TFT 屏。

1. 首先给外置 FLASH 发送读命令，再发送 24bit 地址（图片存在外置 FLASH 的首地址），表示从该地址开始读数据。
2. 设置 DMA 的传输数（图片数据大小：240\*240\*2），使能 DMA1、DMA2 以及 SPI1 的 DMA 传输通道。
3. 软件触发一次 DMA1，DMA1 搬运 1byte 数据给外置 FLASH，外置 FLASH 会推出图片数据给到 SPI1->DATA，会产生一次 SPI\_RX 的 DMA 请求。由于 DMA1、DMA2 的请求源一致且 DMA2 的优先级高于

DAM1，会先执行 DMA2 搬运，把 SPI1->DATA 数据搬运到 SPI0->DATA，即数据给到 TFT 屏显示图片。DMA2 执行完后 DMA1 再执行，给外置 FLASH 传输 1byte 数据，一直重复这个过程直到 DMA 传输数为 0 停止搬运。搬运过程中不需要 CPU 参与。



DMA1、DMA2 搬运数据示意图

## 4 更改记录

版本	记录	日期
V1.0	初版。	2023 年 12 月