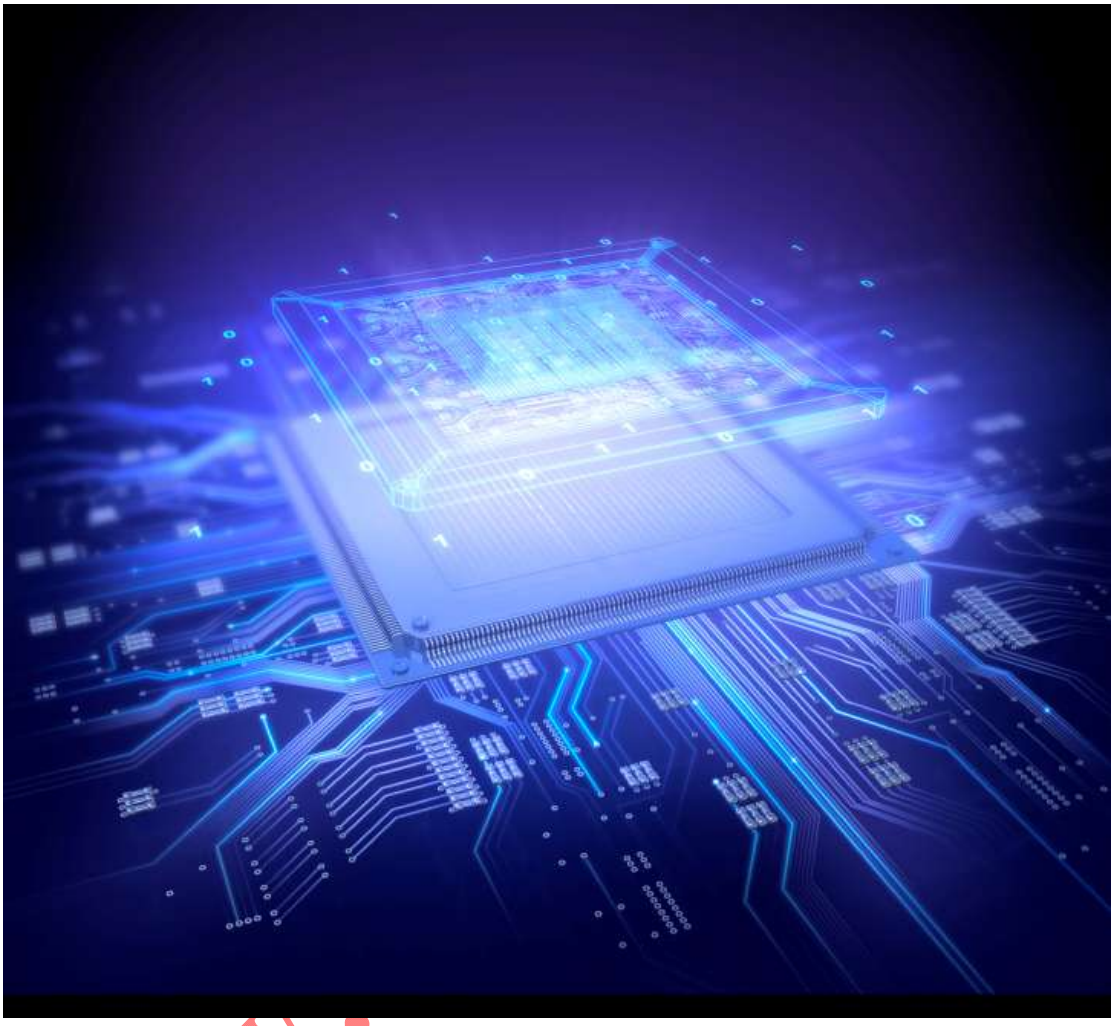
TFT 控制器用户手册



修订记录		
版本	编撰人	日期
V0.01	邓枭	2015.01.30

功能说明

- 分辨率支持 1024*600 及以下。
- 色彩支持 16bit(RGB565)、24bit(RGB888)(取决于 TFT 屏及外扩 SDRAM)。
- 支持多页缓存，页数取决于外扩 SDRAM 大小（例 4M*16Bit 的 SDRAM 支持 800*480 分辨率 8 页缓存）。显示页和读写页互相独立。
- 8080 或 6800 并行接口，数据位宽 16bit 或 24bit。
- 支持 HV、DE 模式，可扩展支持 VGA、LVDS、MIPI DSI 接口。
- 支持 0~16 级（或更高，可按需定制）分辨率的背光控制。
- 支持画面水平和垂直显示镜像。
- 支持指定区域突发读写，即只需要指定一次操作矩形区，X、Y 坐标即可自动增长，无需重复发送坐标位置。
- 用户读写最大带宽 50M*16bit（或 50M*24bit）。

电气规范

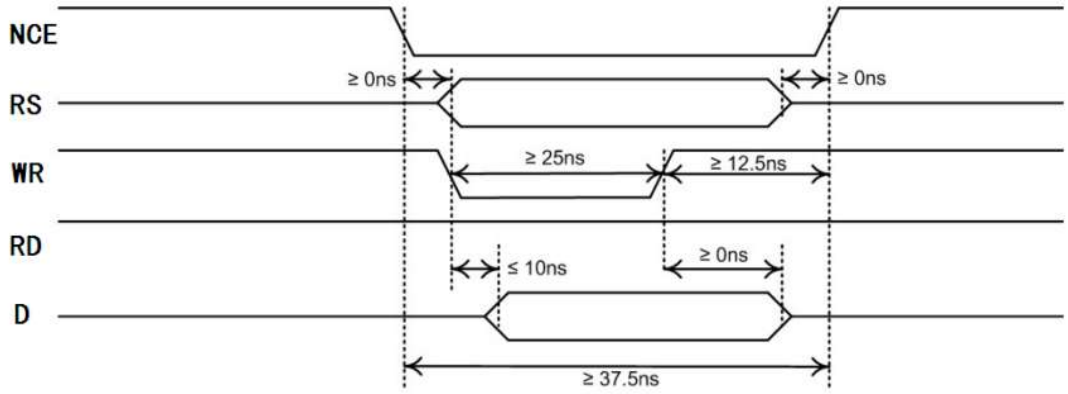
项目	说明
背光电压	5V
IO 电压	3.3V
总线接口	8080，可定制 6800
总线位宽	16bit ，可定制 24bit
总线带宽	50MB * 2 /s
电流	待测
功耗	待测

引脚定义

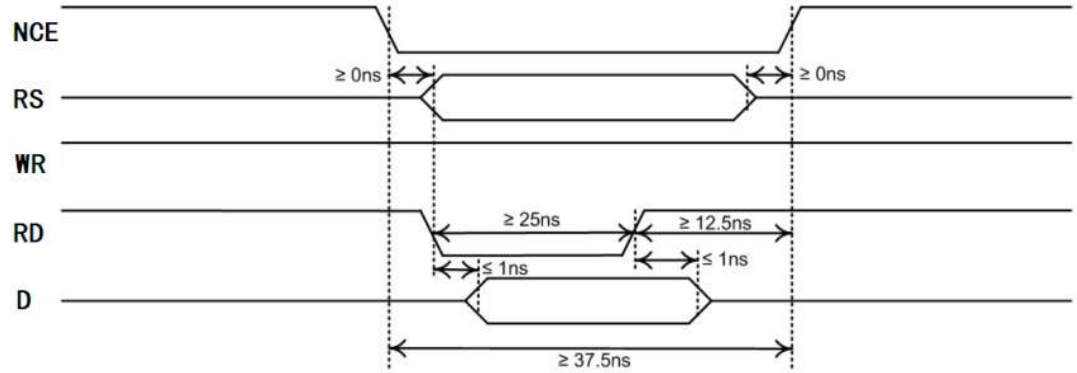
名称	定义	I/O 方向	备注
clk	控制器时钟	输入	频率默认 25M，可定制
nRst	控制器外部复位	输入	低电平有效
dataUsr[15..0]	8080 总线数据线	双向 IO	
cs	8080 总线片选	输入	低电平有效
rs	8080 总线寄存器地址与寄存器数据选择	输入	rs 为低表示寄存器地址，rs 为高表示寄存器数据
wr	8080 总线写使能	输入	低电平有效
rd	8080 总线读使能	输入	低电平有效
clkTft	TFT 模组的时钟	输出	
Vs	TFT 模组的场信号	输出	
Hs	TFT 模组的行信号	输出	
de	TFT 模组的时钟信号	输出	
rDataTft[4..0]	TFT 模组的 R 通道信号	输出	5 位，默认 16 位色，可定制 24 位色
gDataTft[5..0]	TFT 模组的 G 通信信号	输出	6 位，默认 16 位色，可定制 24 位色
bDataTft[4..0]	TFT 模组的 B 通信信号	输出	5 位，默认 16 位色，可定制 24 位色
lrTft	TFT 模组水平扫描方向信号	输出	
udTft	TFT 模组垂直扫描方向信号	输出	
modeTft	TFT 模组行场模式、DE 模式选择信号	输出	
ledTft	TFT 模组背光 PWM 控制信号	输出	

clkSdr	SDRAM 时钟信号	输出	
ckeSdr	SDRAM 时钟使能信号	输出	
nCsSdr	SDRAM 片选信号	输出	
nCasSdr	SDRAM 列地址有效信号	输出	
nRasSdr	SDRAM 行地址有效信号	输出	
dqmSdr[1..0]	SDRAM 掩码信号	输出	
baSdr[1..0]	SDRAM 的 BANK 地址信号	输出	
addrSdr[12..00]	SDRAM 的地址线	输出	
dataSdr[15..0]	SDRAM 的数据线	双向 IO	

控制器读写时序



总线写时序



总线读时序

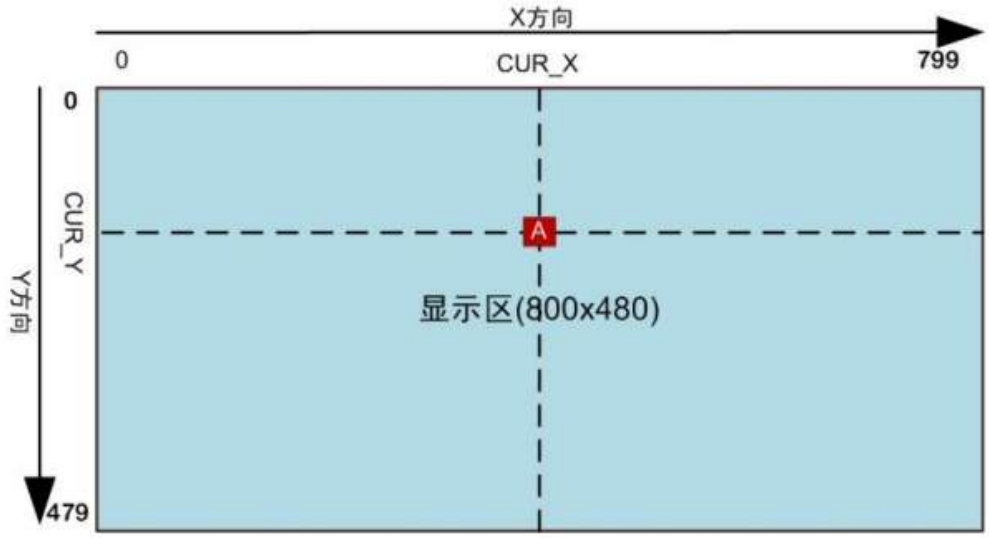
控制器寄存器说明

RS	操作	位宽	寄存器地址	寄存器名称	功能	复位值	备注
0	W	16	0xXXXX	REG_ADDR	设置要操作的寄存器地址	0x00	
0	R	16	0xXXXX	STATE	读状态寄存器	0x00	Bit0 为 1 表示数据已准备好
1	W	16	0x00	CUR_X	X 当前坐标	0x00	
1	W	16	0x01	CUR_Y	Y 当前坐标	0x00	
1	W	16	0x02	END_X	X 结束坐标	水平分辨率	默认为 TFT 屏的水平像素数
1	W	16	0x03	END_Y	Y 结束坐标	垂直分辨率	默认为 TFT 屏的垂直像素数
1	W	16	0x04	PIXELS	像素数据	0xXXXX	随机值
1	W	3	0x05	PAGE_DISP	显示页	0x00	取值范围 0x00~0x07，默认值 0x00
1	W	3	0x06	PAGES_OP	操作页	0x00	取值范围 0x00~0x07，默认值 0x00
1	W	4	0x07	LED_BK	背光占空比	0x0F	范围:0x00~0x0F, 频率 195kHz
1	W	2	0x08	MIRROR	水平镜像和垂直镜像	0x00	默认无镜像，扫描由左至右由上至下
1	W	1	0x09	MODE_TFT	HV 模式、DE 模式选择	0x00	默认 HV 模式
1	R	16	0x04	PIXELS	像素数据	0xXXXX	随机值
1	R	16	0x00	CUR_X	X 当前坐标	0x00	默认不支持
1	R	16	0x01	CUR_Y	Y 当前坐标	0x00	默认不支持
1	R	16	0x02	END_X	X 结束坐标	0x00	默认不支持
1	R	16	0x03	END_Y	Y 结束坐标	0x00	默认不支持
1	R	3	0x05	PAGE_DISP	显示页	0x00	默认不支持
1	R	3	0x06	PAGES_OP	操作页	0x00	默认不支持
1	R	4	0x07	LED_BK	背光占空比	0x00	默认不支持
1	R	2	0x08	MIRROR	水平、垂直镜像	0x00	默认不支持
1	R	1	0x09	MODE_TFT	HV 模式、DE 模式	0x00	默认不支持

注： *为节省逻辑资源，采用更小器件降低成本，不常用功能默认不支持*

CUR_X 寄存器(0x00)和 CUR_Y 寄存器(0x01)

寄存器 CUR_X 和 CUR_Y 用于设置待操作像素点的坐标， TFTLCD 屏幕上坐标的排列，以 800*480 分辨率为例如图所示：



坐标排列

当 CUR_Y 和 CUR_X 的值确定后，像素点 A 的位置便被唯一的确定了，随后的写入的像素数据会被准确的放置在 A 点。

PIXELS 寄存器(0x04)

寄存器 PIXELS 默认对应着 16 位的颜色数据，如果当前显示页与当前操作页相同，那么写入 PIXELS 的数据会被立即呈现在由 CUR_X 和 CUR_Y 选中的当前激活点上，如果当前显示页与当前操作页不相同，那么写入 PIXELS 的数据不会被立即呈现出来。

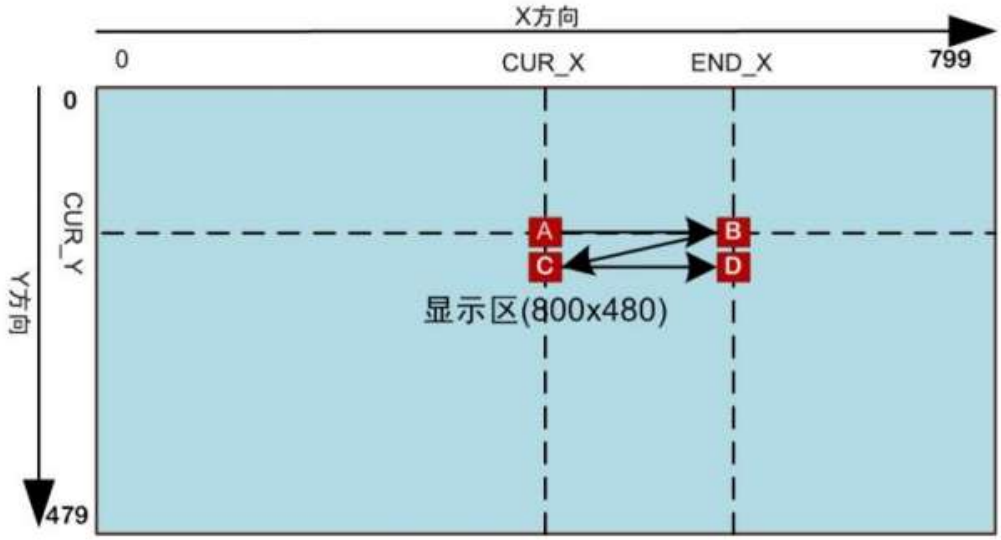
颜色格式默认为 RGB565，具体的颜色与每个位对应关系如表所示：

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

颜色与位对应关系

END_X 寄存器(0x02)、END_Y 寄存器（0x03）

为了提高像素数据连续读写的效率，当设置好 CUR_X 和 CUR_Y 后，每读取/写入一个像素，当前激活点的 X 坐标就会自动加一，当激活点的 X 坐标等于 END_X 后，便会自动返回 CUR_X 同时 Y 坐标自动加一。



X 坐标自动增长示意图

以写数据为例，假设 CUR_X、 CUR_Y、 END_X 分别为 400、 200、 500， A 点、 B 点、 C 点、 D 点的坐标分别为（ 400， 200）、（ 500， 200）、（ 400， 201）、（ 500， 201）。设置好 CUR_X、CUR_Y 后，第一个像素写到了 A 点，第 100 个像素写到 B 点，第 101 个像素写到 C 点，第 200 个像素写到 D 点，依此类推。借助 END_X 、END_Y 寄存器，可以简化 MCU 批量数据读写的流程，假设 MCU 需要以（ 100，200）为起始坐标写入一个 10×20 的矩形，那么只需要将 CUR_X 设为 100， CUR_Y 设为 200，END_X 设为 210，END_Y 设为 220，然后进行 200 次的像素点读/写操作即可，期间不需要再进行坐标设置操作，所有的坐标都会被自动推算。

显示页寄存器（0x05）、操作页寄存器（0x06）

默认支持 8 个缓存页。当前显示页由 PAGE_DISP 指定，表示屏幕上实际显示的显存分页，当前操作页由 PAGE_OP 指定，表示当前读写操作的显存分页。如果 PAGE_DISP 与 PAGE_OP 指向同一显存分页，那么写显存操作的结果会被立即呈现在屏幕上，如果 PAGE_DISP 与 PAGE_OP 指向不同的显存分页，那么对 PAGE_OP 的任何操作都不会影响屏幕上的显示内容，只有在 PAGE_DISP 切换到 PAGE_OP 后，PAGE_OP 中数据才会被显示出来。

背光占空比寄存器（0x07）

BK_PWM 用于设置背光信号的占空比，从而调节 TFT 背光的亮度，取值范围为 0~15，0 代表背光关闭， 15 代表背光最亮。上电复位后 BK_PWM 的值默认为 0，也就是背光关闭，在 MCU 对 BK_PWM 赋以非零值后，背光才能点亮。

水平镜像和垂直镜像寄存器(0x08)

bit0 代表水平扫描方向，bit1 代表垂直扫描方向，bit15~bti2 保留。lrTft、udTft 的输出直接对应于 bit0, bit1，因此其取值与方向的对应关系取决于 TFT 模组自身的定义，详情请参考 TFT 模组的规格书。

HV 模式、DE 模式选择寄存器(0x09)

bit0 为 0 表赤 HV 模式，为 1 表示 DE 模式，默认为 HV 模式。

读 STATE 寄存器

读取该寄存器会自动启动像素点的读操作，当 MCU 查询到 STATE 的 DATA_OK 位（ b0 位）为 1 后，表示像素数据有效，然后 MCU 读 PIXELS 寄存器即可获得对应点的像素数据，与写像素数据的操作相同，读像素数据的像素点位置也是由当前的 CUR_X 和 CUR_Y 定义的。当 MCU 读取 PIXELS 寄存器后， DATA_OK 位会被自动清零。需要注意的是，读 STATE 寄器时， b15~b1 位是随机值，因此在判断 DATA_OK 时，需要屏蔽掉这些位。

代码示例

普通 IO 口模拟总线操作 TFT 控制器

初始化 IO 口，及复位液晶

```
void _LCD_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GP
IOE,ENABLE);

    // Set PD.00(D2), PD.01(D3), PD.04(NOE), PD.05(NWE), PD.07(NE1), PD.08(D13),
    // PD.09(D14), PD.10(D15), PD.13(A18), PD.14(D0), PD.15(D1) as alternate
    // function push pull
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5 |
                                  GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10|
                                  GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);


    // Set PE.07(D4), PE.08(D5), PE.09(D6), PE.10(D7), PE.11(D8), PE.12(D9), PE.13(D10),
    // PE.14(D11), PE.15(D12) as alternate function push pull */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
                                   GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |

                                   GPIO_Pin_15;
    GPIO_Init(GPIOE, &GPIO_InitStructure);



    //PB14 --> LCD_RST
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //LCD_REST=0;
    GPIO_ResetBits(GPIOB, GPIO_Pin_1);
    delay_ms(50); // delay 20 ms
    //LCD_REST=1;
    GPIO_SetBits(GPIOB, GPIO_Pin_1);
    delay_ms(50); // delay 20 ms
}
```

```
2
5
2
6
2
7
2
8
2
9
3
0
3
1
3
2
3
3
3
4
3
5
3
6
3
7
```

采用蓝色清屏

```
#define DELAY 0

void _LCD_Clear(void)
{
    u32 x,y ;//index=0;
    //由于 TFT 控制器默认初始化矩形区为(0, 0, maxHs, maxVs),因此整屏操作可不需初始化指针及矩形区。
    //设置写点寄存器地址
    CS_H(); RS_H(); WR_H(); RD_H();
    SET_PIEXL_REG();        //准备写入
    _delay(DELAY);
    CS_L(); RS_L(); WR_L(); RD_H();
    _delay(DELAY);
    CS_H(); RS_H(); WR_H(); RD_H();
    _delay(DELAY);

    //采用蓝色清屏
    for(y=0;y<480;y++)
    {
        for(x=0;x<800;x++)
        {
            CS_H(); RS_H(); WR_H(); RD_H();
            SET_PIEXL_REG();
            _delay(DELAY);
            OUT_BLUE();
            CS_L(); RS_H(); WR_L(); RD_H();
            _delay(DELAY);
            CS_H(); RS_H(); WR_H(); RD_H();
            _delay(DELAY);
        }
    }
}
```

设置寄存器地址为像素数据寄存器地址

```
void SET_PIEXL_REG()  //0x04
{
    GPIO_ResetBits(GPIOD, GPIO_Pin_14);
    GPIO_ResetBits(GPIOD, GPIO_Pin_15);
    GPIO_SetBits(GPIOD, GPIO_Pin_0);
    GPIO_ResetBits(GPIOD, GPIO_Pin_1);

    GPIO_ResetBits(GPIOE, GPIO_Pin_7);
```

```
    GPIO_ResetBits(GPIOE, GPIO_Pin_8);
    GPIO_ResetBits(GPIOE, GPIO_Pin_9);
    GPIO_ResetBits(GPIOE, GPIO_Pin_10);

    GPIO_ResetBits(GPIOE, GPIO_Pin_11);
    GPIO_ResetBits(GPIOE, GPIO_Pin_12);
    GPIO_ResetBits(GPIOE, GPIO_Pin_13);
    GPIO_ResetBits(GPIOE, GPIO_Pin_14);

    GPIO_ResetBits(GPIOE, GPIO_Pin_15);
    GPIO_ResetBits(GPIOD, GPIO_Pin_8);
    GPIO_ResetBits(GPIOD, GPIO_Pin_9);
    GPIO_ResetBits(GPIOD, GPIO_Pin_10);
}
```

写蓝色数据

```
void OUT_BLUE()       //0x1f
{
    GPIO_SetBits(GPIOD, GPIO_Pin_14);
    GPIO_SetBits(GPIOD, GPIO_Pin_15);
    GPIO_SetBits(GPIOD, GPIO_Pin_0);
    GPIO_SetBits(GPIOD, GPIO_Pin_1);

    GPIO_SetBits(GPIOE, GPIO_Pin_7);
    GPIO_ResetBits(GPIOE, GPIO_Pin_8);
    GPIO_ResetBits(GPIOE, GPIO_Pin_9);
    GPIO_ResetBits(GPIOE, GPIO_Pin_10);

    GPIO_ResetBits(GPIOE, GPIO_Pin_11);
    GPIO_ResetBits(GPIOE, GPIO_Pin_12);
    GPIO_ResetBits(GPIOE, GPIO_Pin_13);
    GPIO_ResetBits(GPIOE, GPIO_Pin_14);

    GPIO_ResetBits(GPIOE, GPIO_Pin_15);
    GPIO_ResetBits(GPIOD, GPIO_Pin_8);
    GPIO_ResetBits(GPIOD, GPIO_Pin_9);
    GPIO_ResetBits(GPIOD, GPIO_Pin_10);
}
```

STM32 通过 FSMC 总线操作 TFT 控制器示例

初始化 FSMC 相关 IO 及时钟

```
    GPIO_InitTypeDef GPIO_InitStructure;
    FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
    FSMC_NORSRAMTimingInitTypeDef  readWriteTiming;
    FSMC_NORSRAMTimingInitTypeDef  writeTiming;

    // Enable FSMC, GPIOD, GPIOE clocks
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GP
IOE,ENABLE);

    // Set PD.00(D2), PD.01(D3), PD.04(NOE), PD.05(NWE), PD.07(NE1), PD.08(D13),
    // PD.09(D14), PD.10(D15), PD.13(A18), PD.14(D0), PD.15(D1) as alternate
    // function push pull
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5 |
                                  GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10|
                                  GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    // Set PE.07(D4), PE.08(D5), PE.09(D6), PE.10(D7), PE.11(D8), PE.12(D9), PE.13(D10),
    // PE.14(D11), PE.15(D12) as alternate function push pull */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
                                   GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |

                                   GPIO_Pin_15;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    //PB14 --> LCD_RST
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

初始化 FSMC

```
    readWriteTiming.FSMC_AddressSetupTime = 2;   //地址建立时间(ADDSET)为 2 个 HCLK 1/36M=27ns
    readWriteTiming.FSMC_AddressHoldTime = 1;    //地址保持时间（ADDHLD）模式 A 未用到
    readWriteTiming.FSMC_DataSetupTime = 10;         // 数据保存时间为 16 个 HCLK,因为液晶驱动
IC 的读数据的时候，速度不能太快，尤其对 1289 这个 IC。
    readWriteTiming.FSMC_BusTurnAroundDuration = 1;
    readWriteTiming.FSMC_CLKDivision = 1;
    readWriteTiming.FSMC_DataLatency = 1;
    readWriteTiming.FSMC_AccessMode = FSMC_AccessMode_A;     //模式 A

    writeTiming.FSMC_AddressSetupTime = 4;   //2 地址建立时间（ADDSET）为 1 个 HCLK
    writeTiming.FSMC_AddressHoldTime = 2;    //1 地址保持时间（A
    writeTiming.FSMC_DataSetupTime = 20;         //10 数据保存时间为 4 个 HCLK
    writeTiming.FSMC_BusTurnAroundDuration = 0;
    writeTiming.FSMC_CLKDivision = 1;
    writeTiming.FSMC_DataLatency = 1;
    writeTiming.FSMC_AccessMode = FSMC_AccessMode_A;     //模式 A

    FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM1;//  这里我们使用 NE1 ，也就对应
BTCR[6],[7]。
    FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable; // 不复用
数据地址
    FSMC_NORSRAMInitStructure.FSMC_MemoryType =FSMC_MemoryType_SRAM;// FSMC_MemoryType_SRA
M;  //SRAM
    FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;//存储器数据
宽度为 16bit
    FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =FSMC_BurstAccessMode_Disable;// FSMC_B
urstAccessMode_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;
    FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait=FSMC_AsynchronousWait_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitStat
e;
    FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable; //  存储器
写使能
    FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
    FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Enable; // 读写使用不同
的时序
    FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
    FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &readWriteTiming; //读写时序
    FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &writeTiming;  //写时序

    FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);  //初始化 FSMC 配置

```

```
    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);  // 使能 BANK1
    // 复位 LCD 控制器
    LCD_REST=0;
    delay_ms(50); // delay 20 ms
    LCD_REST=1;
    delay_ms(50); // delay 20 ms
```

使用 FSMC 对 TFT 控制器进行写入操作

```
//LCD 地址结构体定义
typedef struct
{
    u16 LCD_REG;
    u16 LCD_RAM;
} LCD_TypeDef;

//FSMC 地址设置
#define LCD_BASE        ((u32)(0x60000000 | 0x0007FFFE))
#define LCD             ((LCD_TypeDef *) LCD_BASE)

//用指定色彩填充指定区域
void LCD_CleartRect(u16 xStart, u16 yStart, u16 xEnd, u16 yEnd, u16 color)
{
    //u16 xStart, yStart, xEnd, yEnd;
    u16 i,j;
    LCD_SetRect(xStart, yStart, xEnd, yEnd);                //设置矩形区
    LCD_WriteRAM_Prepare();     //开始写入 GRAM
    for(i=yStart; i<=yEnd; i++)
        for(j=xStart; j<=xEnd; j++)
            LCD_WR_DATA(color);
}

//设置要进行操作的矩形区
void LCD_SetRect(u16 xStart, u16 yStart, u16 xEnd, u16 yEnd)
{
    LCD_WR_REG_DATA(0x00, xStart);
    LCD_WR_REG_DATA(0x01, yStart);
    LCD_WR_REG_DATA(0x02, xEnd);
    LCD_WR_REG_DATA(0x03, yEnd);
}

//准备开始写显存
void LCD_WriteRAM_Prepare(void)
{
```

```
35      LCD_WR_REG(0x04);
36  }
37
38  //往总线上写寄存器地址
39  void LCD_WR_REG(u16 regval)
40  {
41      LCD->LCD_REG=regval;//写入要写的寄存器序号
42  }
43
44  //往总线上写数据
45  void LCD_WR_DATA(u16 data)
46  {
47      LCD->LCD_RAM=data;
48      LCD->LCD_RAM=data<<8;
49  }
50
51
```