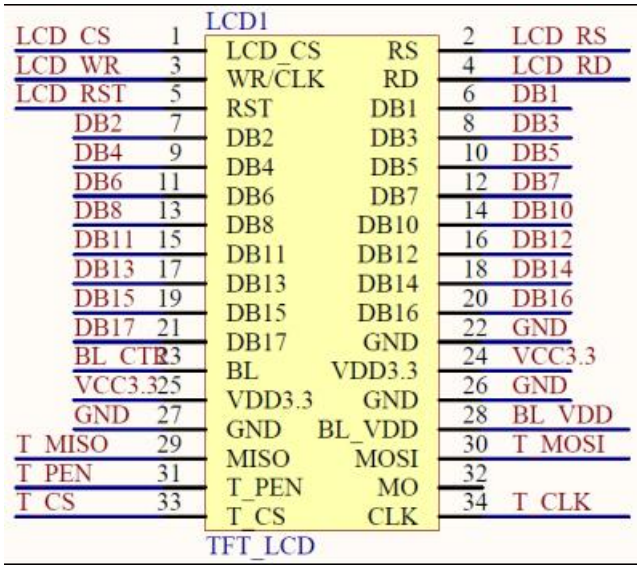
一、8080 并口协议

要是用 IO 模拟的方式驱动 LCD，首先要了解 80 并口协议，因为 IO 模拟需要遵守这个协议才可以正常驱动 LCD。



LCD 屏常用的时序为 8080 时序和 6800 时序， 6800 总线又叫做摩托罗拉总线、8080 时序也叫做英特尔总线。

Intel 总线的控制线有四根,RD 写使能, WR 读使能, ALE 地址锁存, CS 片选。而 moto 总线只有 三根,R/W 读/写,ALE 地址锁存,CE 片使能。8080 总线存在许多接口 8/9/16/18 位接口

1、8080 模式：

LCD 控制以及传输数据所需要的的管脚列表

管脚名称	功能描述
CS	片选信号线
RS(D/I)	数据/命令选择管脚(1:数据读写,0:命令读写)
WR	MPU向LCD写入数据控制线
RD	MPU从LCD读出数据控制线
DB[15：0]	16位双向数据线
RST	硬复位LCD信号
BL	LCD背光控制信号
IM0	IM0=0时为16bit数据总线，IM0=1时为8bit数据总线

2、8080 并口读/写的过程：

（1）读取数据：

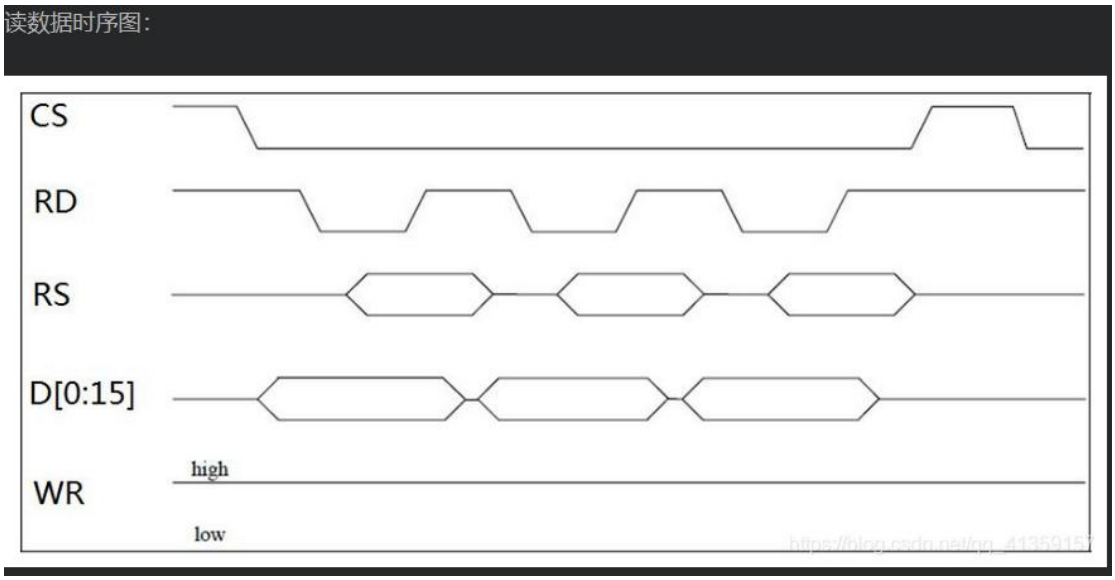
伪代码：

1、CS 为低

2、RS 为高（数据）
3、在 RD 的上升沿，读取数据线上的数据（D[15:0]），
4、WR 为高
5、CS 为高，结束一组数据读取

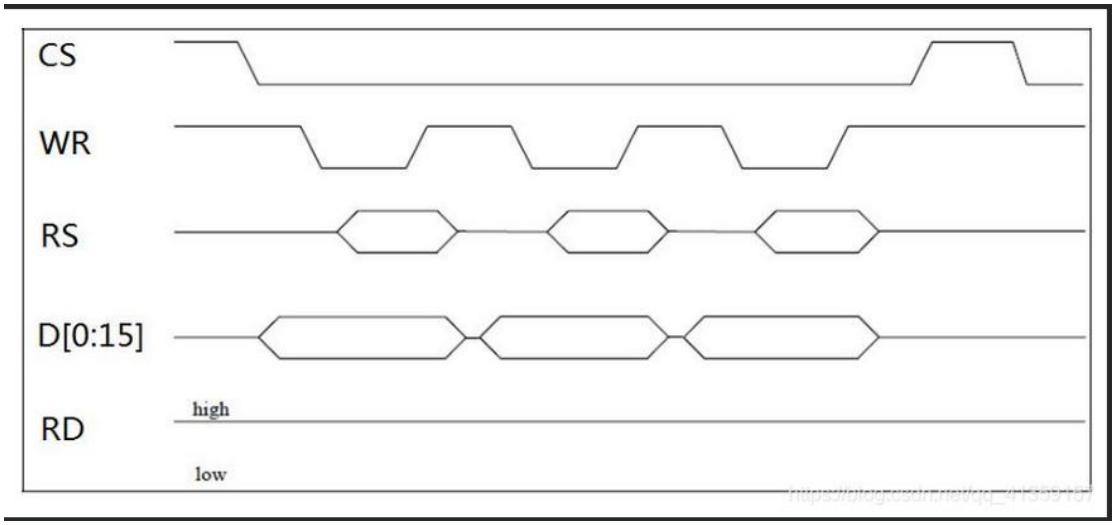
```
LCD_CS = 0;          //开始片选
LCD_RS = 1;          //读数据
LCD_WR = 1;          //禁止写
LCD_RD = 0;          //上升沿读数据
data = DATAIN();//读取数据
LCD_RD = 1;          //上升沿读数据
LCD_CS = 1;          //结束片选
```

读数据时序图：



（2）写入数据：
伪代码：
1、CS 为低
2、RS 为高（数据）
3、在 WR 的上升沿，使数据写入到 驱动 IC 里面
4、RD 为高
5、CS 为高，结束一组数据读取

```
LCD_CS = 0;          //开始片选
LCD_RS = 1;          //写数据
LCD_RD = 1;          //禁止读
DATAOUT(Data);       //输出数据
LCD_WR = 0;          //写入开始
LCD_WR = 1;          //写入结束
LCD_CS = 0;          //结束片选
```



二、ILI9341

这款 LCD 驱动芯片我相信大家都很熟悉，具体情况可以翻阅 ILI9341 数据手册。我在网上看到卖 9341 的屏幕有 37pin 和 40pin 的。原子的使用 37pin，野火的使用 40pin。下面是我找到管脚定义图：

37pin：

引脚编号	引脚名称	功能说明
1	DB0	LCD 数据信号线
2	DB1	LCD 数据信号线
3	DB2	LCD 数据信号线
4	DB3	LCD 数据信号线
5	GND	地
6	VDDI	I/O 口电压（+2.8V~+3.3V)
7	CSX	片选信号，低电平有效
8	DCX	指令/数据选择端，L:指令，H:数据
9	WRX	LCD 写控制端
10	RDX	LCD 读控制端
11	NC	悬空
12	X+	触摸屏信号线
13	Y+	触摸屏信号线
14	X-	触摸屏信号线
15	Y-	触摸屏信号线
16	LEDA	背光 LED 正极
17	LEDK1	背光 LED 负极
18	LEDK2	背光 LED 负极
19	LEDK3	背光 LED 负极
20	LEDK4	背光 LED 负极
21	NC/FMARK	悬空
22	DB4	LCD 数据信号线
23	DB8	LCD 数据信号线
24	DB9	LCD 数据信号线
25	DB10	LCD 数据信号线
26	DB11	LCD 数据信号线
27	DB12	LCD 数据信号线
28	DB13	LCD 数据信号线
29	DB14	LCD 数据信号线
30	DB15	LCD 数据信号线
31	RESX	复位信号线
32	VCI	模拟电路电源（+2.8V~+3.3V)
33	VDDI	I/O 口电压（+2.8V~+3.3V)
34	GND	地
35	DB5	LCD 数据信号线
36	DB6	LCD 数据信号线
37	DB7	LCD 数据信号线

40pin :

引脚编号	引脚名称	功能说明
1	X-	触摸屏信号线
2	Y-	触摸屏信号线
3	X+	触摸屏信号线
4	Y+	触摸屏信号线
5	GND	地
6	VDDI	I/O 口电压（+2.8V~+3.3V)
7	VDD	模拟电路电源（+2.8V~+3.3V)
8	NC/FMARK	悬空
9	CSX	片选信号，低电平有效
10	DCX	指令/数据选择端，L:指令，H:数据
11	WRX	LCD 写控制端
12	RDX	LCD 读控制端
13	SPI SDI	串口信号线- 输入
14	SPI SDO	串口信号线- 输出
15	RESX	复位信号线
16	GND	地
17	DB0	LCD 数据信号线
18	DB1	LCD 数据信号线
19	DB2	LCD 数据信号线
20	DB3	LCD 数据信号线
21	DB4	LCD 数据信号线
22	DB5	LCD 数据信号线
23	DB6	LCD 数据信号线
24	DB7	LCD 数据信号线
25	DB8	LCD 数据信号线
26	DB9	LCD 数据信号线
27	DB10	LCD 数据信号线
28	DB11	LCD 数据信号线
29	DB12	LCD 数据信号线
30	DB13	LCD 数据信号线
31	DB14	LCD 数据信号线
32	DB15	LCD 数据信号线
33	LED-A	背光 LED 正极性端
34	LED-K	背光 LED 负极性端
35	LED-K	背光 LED 负极性端
36	LED-K	背光 LED 负极性端
37	GND	地
38	IM0	模式选择
39	IM1	模式选择
40	IM2	模式选择

我们通过对比可以看到 40pin 比 37pin 多了模式选择（IM），以及支持串口信号 SPI。
三、GPIO 口的配置

对于 stm32，我们使用 IO 模拟 8080 驱动 LCD 屏幕时，除了 CS、WR、RD、RS、RST、BL 控制引脚，可以根据自己需要，定义任意 IO 去控制。对于数据端口 DB[15:0],建议使用同一个 GPIO 端口使用，因为操作方便（当然后面也提供数据端口也使用任意 IO 控制的方法）。

接下来是 GPIO 配置部分：

1、DB[15:0]使用同一个 GPIO 口。

```
static void ILI9341_GPIO_Config ( void )
{
    GPIO_InitTypeDef GPIO_InitStructure;
    /* 使能复用 IO 时钟*/
    //  RCC_APB2PeriphClockCmd ( RCC_APB2Periph_AFIO, ENABLE );
    //复位引脚直接使用 NRST，开发板复位的时候会使液晶复位
    /* 使能对应相应管脚时钟*/
    RCC_APB2PeriphClockCmd (/*控制信号*/
                            ILI9341_CS_CLK|ILI9341_DC_CLK|ILI9341_WR_CLK|
                            ILI9341_RD_CLK  |ILI9341_BK_CLK|
                            /*数据信号*/
                            ILI9341_DATA_CLK, ENABLE );

    //开启 SWD，失能 JTAG（部分 PB 引脚用在了 jtag 接口，改成 SWD 接口就不会有干扰）
    //  GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable , ENABLE);
    /* 配置液晶相对应的数据线,PORT-D0~D15 */
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_Out_PP;

    GPIO_InitStructure.GPIO_Pin = ILI9341_DATA_PIN;
    GPIO_Init ( ILI9341_DATA_PORT, &GPIO_InitStructure );
       /* 配置液晶相对应的控制线
     * 读                :LCD-RD
     * 写                :LCD-WR
     * 片选              :LCD-CS
     * 数据/命令         :LCD-DC
     */
    GPIO_InitStructure.GPIO_Pin = ILI9341_RD_PIN;
    GPIO_Init (ILI9341_RD_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_WR_PIN;
    GPIO_Init (ILI9341_WR_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_CS_PIN;
```

```
    GPIO_Init ( ILI9341_CS_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_DC_PIN;
    GPIO_Init ( ILI9341_DC_PORT, & GPIO_InitStructure );

    /* 配置 LCD 背光控制管脚 BK*/
    GPIO_InitStructure.GPIO_Pin = ILI9341_BK_PIN;
    GPIO_Init ( ILI9341_BK_PORT, &GPIO_InitStructure );
```

脚位						管脚名称	类型(1)	I/O电平(2)	主功能(3)(复位后)	可选的复用功能	
BGA144	BGA100	WLCSP64	LQFP64	LQFP100	LQFP144					默认复用功能	重定义功能
A7	A7	A4	55	89	133	PB3	I/O	FT	JTDO	SPI3_SCK / I2S3_CK	PB3/TRACESWO TIM2_CH2/ SPI1_SCK
A6	A6	B4	56	90	134	PB4	I/O	FT	NJTRST	SPI3_MISO	PB4/TIM3_CH1/ SPI1_MISO

配置的时候我们要注意，如果数据端口使用的是 GPIOB（PB3、PB4），我们要禁用 JTAG。对于 STM32F103，我们需要将上面代码注释部分还原。如果不使用 GPIOB 端口，使用其他 GPIO，我们则无需改动，注释保留。

2、DB[15:0]使用不同的 GPIO 口。

宏定义方式，便于移植

```
static void ILI9341_GPIO_Config ( void )
{
    GPIO_InitTypeDef GPIO_InitStructure;
        RCC_APB2PeriphClockCmd (
                                /*控制信号*/
                                ILI9341_CS_CLK|ILI9341_DC_CLK|ILI9341_WR_CLK|
                                ILI9341_RD_CLK|ILI9341_BK_CLK|
                                /*数据信号*/
                                ILI9341_D0_CLK|ILI9341_D1_CLK|ILI9341_D2_CLK|
                                ILI9341_D3_CLK|ILI9341_D4_CLK|ILI9341_D5_CLK|
                                ILI9341_D6_CLK|ILI9341_D7_CLK|ILI9341_D8_CLK|

    ILI9341_D9_CLK|ILI9341_D10_CLK|ILI9341_D11_CLK|

    ILI9341_D12_CLK|ILI9341_D13_CLK|ILI9341_D14_CLK|
                                ILI9341_D15_CLK, ENABLE );

    /* 配置液晶相对应的数据线,PORT-D0~D15 */
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_Out_PP;

GPIO_InitStructure.GPIO_Pin = ILI9341_D0_PIN;
GPIO_Init ( ILI9341_D0_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D1_PIN;
GPIO_Init ( ILI9341_D1_PORT, &GPIO_InitStructure );

GPIO_InitStructure.GPIO_Pin = ILI9341_D2_PIN;
GPIO_Init ( ILI9341_D2_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D3_PIN;
GPIO_Init ( ILI9341_D3_PORT, &GPIO_InitStructure );

GPIO_InitStructure.GPIO_Pin = ILI9341_D4_PIN;
GPIO_Init ( ILI9341_D4_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D5_PIN;
GPIO_Init ( ILI9341_D5_PORT, &GPIO_InitStructure );

GPIO_InitStructure.GPIO_Pin = ILI9341_D6_PIN;
GPIO_Init ( ILI9341_D6_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D7_PIN;
GPIO_Init ( ILI9341_D7_PORT, &GPIO_InitStructure );

GPIO_InitStructure.GPIO_Pin = ILI9341_D8_PIN;
GPIO_Init ( ILI9341_D8_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D9_PIN;
GPIO_Init ( ILI9341_D9_PORT, &GPIO_InitStructure );

GPIO_InitStructure.GPIO_Pin = ILI9341_D10_PIN;
GPIO_Init ( ILI9341_D10_PORT, &GPIO_InitStructure );


GPIO_InitStructure.GPIO_Pin = ILI9341_D11_PIN;
GPIO_Init ( ILI9341_D11_PORT, &GPIO_InitStructure );
```

```
    GPIO_InitStructure.GPIO_Pin = ILI9341_D12_PIN;
    GPIO_Init ( ILI9341_D12_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D13_PIN;
    GPIO_Init ( ILI9341_D13_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D14_PIN;
    GPIO_Init ( ILI9341_D14_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D15_PIN;
    GPIO_Init ( ILI9341_D15_PORT, &GPIO_InitStructure );
    /* 配置液晶相对应的控制线
     * 读                :LCD-RD
     * 写                :LCD-WR
     * 片选          :LCD-CS
     * 数据/命令     :LCD-DC
     */
    GPIO_InitStructure.GPIO_Pin = ILI9341_RD_PIN;
    GPIO_Init (ILI9341_RD_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_WR_PIN;
    GPIO_Init (ILI9341_WR_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_CS_PIN;
    GPIO_Init ( ILI9341_CS_PORT, & GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_DC_PIN;
    GPIO_Init ( ILI9341_DC_PORT, & GPIO_InitStructure );

    /* 配置 LCD 背光控制管脚 BK*/
    GPIO_InitStructure.GPIO_Pin = ILI9341_BK_PIN;
    GPIO_Init ( ILI9341_BK_PORT, &GPIO_InitStructure );
}
```

四、读写函数实现

1、写数据函数：

```
/**
  * @brief  向 ILI9341 写入数据
  * @param  usData :要写入的数据
  * @retval 无
```

```
  */
__inline void ILI9341_Write_Data ( uint16_t usData )
{
    ILI9341_CS_CLR;//开始片选
    ILI9341_DC_SET;//写数据
    ILI9341_RD_SET;//禁止读
    DATAOUT(usData);//输出数据
    ILI9341_WR_CLR;//写入开始
    ILI9341_WR_SET;//写入结束
    ILI9341_CS_SET;//结束片选

}
```

2、写命令函数：

```
/**
  * @brief  向 ILI9341 写入命令
  * @param  usCmd :要写入的命令（表寄存器地址）
  * @retval 无
  */
__inline void ILI9341_Write_Cmd ( uint16_t usCmd )
{
    ILI9341_CS_CLR;//开始片选
    ILI9341_DC_CLR;//写命令
    ILI9341_RD_SET;//禁止读
    DATAOUT(usCmd);//输出命令
    ILI9341_WR_CLR;//写入开始
    ILI9341_WR_SET;//写入结束
    ILI9341_CS_SET;//结束片选
}
```

3、读数据函数：

```
/**
  * @brief  从 ILI9341 读取数据
  * @param  无
  * @retval 读取到的数据
  */
__inline uint16_t ILI9341_Read_Data ( void )
{
    uint16_t data;

#if      IL9341_DATA_USE_ONEPORT == 1

    ILI9341_DATA_PORT->CRL=0X88888888; //上拉输入
    ILI9341_DATA_PORT->CRH=0X88888888; //上拉输入
    ILI9341_DATA_PORT->ODR=0X0000;     //全部输出 0
```

```
#elif  IL9341_DATA_USE_ANYPORT == 1
    GPIO_SET_DATA_OUT(GPIO_Set_Mode_IN);
    DATAOUT(0X0000);
#endif

    ILI9341_DC_SET;
    ILI9341_WR_SET;

    ILI9341_CS_CLR;
    //读取数据
    ILI9341_RD_CLR;

    data = DATAIN();
    ILI9341_RD_SET;
    ILI9341_CS_SET;

#if      IL9341_DATA_USE_ONEPORT == 1

    ILI9341_DATA_PORT->CRL=0X33333333; // 上拉输出
    ILI9341_DATA_PORT->CRH=0X33333333; // 上拉输出
    ILI9341_DATA_PORT->ODR=0XFFFF;    //全部输出高

#elif  IL9341_DATA_USE_ANYPORT == 1
    GPIO_SET_DATA_OUT(GPIO_Set_Mode_OUT);
    DATAOUT(0XFFFF);
#endif
    return data;
}
```

这里就需要说一说细节点了。可以看到我这里用了两个宏，IL9341_DATA_USE_ONEPORT 、IL9341_DATA_USE_ANYPORT 分别对应于使用同一个 GPIO 端口和任意 GPIO 端口。其实 GPIO 初始化时也是这么定义的，只是为了便于区分，就此分开编写，后面工程中有体现。

对于使用同一个 GPIO 总共 16 个引脚，正好对应 DB[15:0]，所以 DATAOUT()/DATAIN()函数对应如下所示：

1、数据输出 DATAOUT（）：

```
//数据线输入输出
#define DATAOUT(x)      ILI9341_DATA_PORT->ODR=x; //数据输出
#define DATAIN()        ILI9341_DATA_PORT->IDR;   //数据输入
```

那么我如果使用不同的 GPIO 端口作为 DB[15:0]数据线，肯定就不能使用这种方法了。这个时候就需要使用如下：

```
//使用宏定义方式，或者直接定义为一个 DATAOUT 函数。
#if 1
```

```
//空间换时间
//位带操作，与一个缺点，不能根据宏定义更改，操作时，需要对这个部分单独带入对应 GPIO
#define DATAOUT(x) \
{\
     D0_W =  (x>>0&0x0001);\
     D1_W =  (x>>1&0x0001);\
     D2_W =  (x>>2&0x0001);\
     D3_W =  (x>>3&0x0001);\
     D4_W =  (x>>4&0x0001);\
     D5_W =  (x>>5&0x0001);\
     D6_W =  (x>>6&0x0001);\
     D7_W =  (x>>7&0x0001);\
     D8_W =  (x>>8&0x0001);\
     D9_W =  (x>>9&0x0001);\
     D10_W = (x>>10&0x0001);\
     D11_W = (x>>11&0x0001);\
     D12_W = (x>>12&0x0001);\
     D13_W = (x>>13&0x0001);\
     D14_W = (x>>14&0x0001);\
     D15_W = (x>>15&0x0001);\
}
#endif

//调用库函数实现，可以解决直接修改宏定义全局修改，不必像位带操作，针对每个 GPIO 带入
//比起使用未位带操作满了大概 800ms，肉眼可见
#if 0
void DATAOUT(unsigned int x)
{
//   ILI9341_D0_WRITE =  (x>>0&0x0001)&ILI9341_D0_PIN;
//   ILI9341_D1_WRITE =  (x>>1&0x0001)&ILI9341_D1_PIN;
//   ILI9341_D2_WRITE =  (x>>2&0x0001)&ILI9341_D2_PIN;
//   ILI9341_D3_WRITE =  (x>>3&0x0001)&ILI9341_D3_PIN;
//   ILI9341_D4_WRITE =  (x>>4&0x0001)&ILI9341_D4_PIN;
//   ILI9341_D5_WRITE =  (x>>5&0x0001)&ILI9341_D5_PIN;
//   ILI9341_D6_WRITE =  (x>>6&0x0001)&ILI9341_D6_PIN;
//   ILI9341_D7_WRITE =  (x>>7&0x0001)&ILI9341_D7_PIN;
//   ILI9341_D8_WRITE =  (x>>8&0x0001)&ILI9341_D8_PIN;
//   ILI9341_D9_WRITE =  (x>>9&0x0001)&ILI9341_D9_PIN;
//   ILI9341_D10_WRITE = (x>>10&0x0001)&ILI9341_D10_PIN;
//   ILI9341_D11_WRITE = (x>>11&0x0001)&ILI9341_D11_PIN;
//   ILI9341_D12_WRITE = (x>>12&0x0001)&ILI9341_D12_PIN;
//   ILI9341_D13_WRITE = (x>>13&0x0001)&ILI9341_D13_PIN;
//   ILI9341_D14_WRITE = (x>>14&0x0001)&ILI9341_D14_PIN;
//   ILI9341_D15_WRITE = (x>>15&0x0001)&ILI9341_D15_PIN;
```

```
//-----------------------------------------------
//-----------------------------------------
    GPIO_WriteBit(ILI9341_D0_PORT,ILI9341_D0_PIN,(BitAction)(x>>0&0x0001));
    GPIO_WriteBit(ILI9341_D1_PORT,ILI9341_D1_PIN,(BitAction)(x>>1&0x0001));
    GPIO_WriteBit(ILI9341_D2_PORT,ILI9341_D2_PIN,(BitAction)(x>>2&0x0001));
    GPIO_WriteBit(ILI9341_D3_PORT,ILI9341_D3_PIN,(BitAction)(x>>3&0x0001));
    GPIO_WriteBit(ILI9341_D4_PORT,ILI9341_D4_PIN,(BitAction)(x>>4&0x0001));
    GPIO_WriteBit(ILI9341_D5_PORT,ILI9341_D5_PIN,(BitAction)(x>>5&0x0001));
    GPIO_WriteBit(ILI9341_D6_PORT,ILI9341_D6_PIN,(BitAction)(x>>6&0x0001));
    GPIO_WriteBit(ILI9341_D7_PORT,ILI9341_D7_PIN,(BitAction)(x>>7&0x0001));
    GPIO_WriteBit(ILI9341_D8_PORT,ILI9341_D8_PIN,(BitAction)(x>>8&0x0001));
    GPIO_WriteBit(ILI9341_D9_PORT,ILI9341_D9_PIN,(BitAction)(x>>9&0x0001));
    GPIO_WriteBit(ILI9341_D10_PORT,ILI9341_D10_PIN,(BitAction)(x>>10&0x0001));
    GPIO_WriteBit(ILI9341_D11_PORT,ILI9341_D11_PIN,(BitAction)(x>>11&0x0001));
    GPIO_WriteBit(ILI9341_D12_PORT,ILI9341_D12_PIN,(BitAction)(x>>12&0x0001));
    GPIO_WriteBit(ILI9341_D13_PORT,ILI9341_D13_PIN,(BitAction)(x>>13&0x0001));
    GPIO_WriteBit(ILI9341_D14_PORT,ILI9341_D14_PIN,(BitAction)(x>>14&0x0001));
    GPIO_WriteBit(ILI9341_D15_PORT,ILI9341_D15_PIN,(BitAction)(x>>15&0x0001));
}
#endif
```

2、数据输入 DATAIN（）：

```
#if 1
unsigned short DATAIN(void)
{
    volatile unsigned short data = 0;
    data |= D15_R;data <<= 1;
    data |= D14_R;data <<= 1;
    data |= D13_R;data <<= 1;
    data |= D12_R;data <<= 1;
    data |= D11_R;data <<= 1;
    data |= D10_R;data <<= 1;
    data |= D9_R;data <<= 1;
    data |= D8_R;data <<= 1;
    data |= D7_R;data <<= 1;
    data |= D6_R;data <<= 1;
    data |= D5_R;data <<= 1;
    data |= D4_R;data <<= 1;
    data |= D3_R;data <<= 1;
    data |= D2_R;data <<= 1;
    data |= D1_R;data <<= 1;
    data |= D0_R;

    return data;
```

```
}
#endif
```

注意：在 ILI9341_Read_Data（）函数中，需要切换数据端口的 GPIO 的模式。 开始读取时需要切换为输入模式，进行读取数据，读取完数据之后，再切换为输出模式，便于后续操作。

五、功能函数验证及实现

经过上述的配置，现在我们需要知道是否可以正常的读写。所以我们采用读取 ILI9341ID 号的方式来验证是否正常。

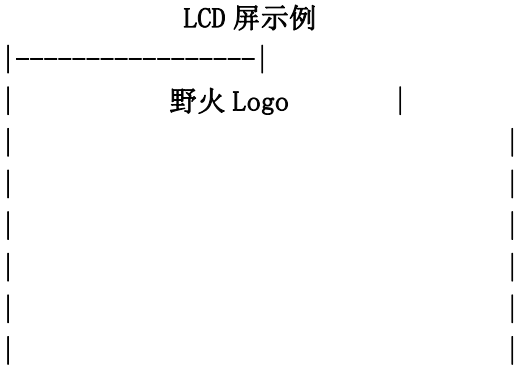
```
/**
 * @brief  ILI9341 读取芯片 ID 函数，可用于测试底层的读写函数
 * @param  无
 * @retval 正常时返回值为 0x9341
 */
uint16_t ILI9341_Read_ID(void)
{
    uint16_t id = 0;

    ILI9341_Write_Cmd(0xD3);
    ILI9341_Read_Data();
    ILI9341_Read_Data();
    id = ILI9341_Read_Data();
    id<<=8;
    id|=ILI9341_Read_Data();

    return id;

}
```

经过验证，上述读写函数、IO 配置等都是正常的。接下来就是，画点、画线等函数了。这些驱动部分可以参考野火或者原子的亦或自己编写都可以。下面贴上野火的。以便大家不用打开工程即可查阅。

1、设置 ILI9341 的屏幕方向显示

```
/**
 * @brief  设置 ILI9341 的 GRAM 的扫描方向
 * @param  ucOption ：选择 GRAM 的扫描方向
 *     @arg 0-7 :参数可选值为 0-7 这八个方向
 *
 *  ！！！其中 0、3、5、6 模式适合从左至右显示文字，
 *              不推荐使用其它模式显示文字  其它模式显示文字会有镜像效果

 *
 *  其中 0、2、4、6 模式的 X 方向像素为 240，Y 方向像素为 320
```

```
 *  其中 1、3、5、7 模式下 X 方向像素为 320，Y 方向像素为 240
 *
 *  其中 6 模式为大部分液晶例程的默认显示方向
 *  其中 3 模式为摄像头例程使用的方向
 *  其中 0 模式为 BMP 图片显示例程使用的方向
 *
 * @retval 无
 * @note  坐标图例：A 表示向上，V 表示向下，<表示向左，>表示向右
                   X 表示 X 轴，Y 表示 Y 轴

------------------------------------------------------------
模式 0：              .       模式 1：      .   模式 2：          .   模式 3：

                      A       .                     A       .       A
        .       A
                      |       .                     |       .       |
        .       |
                      Y       .                     X       .       Y
        .       X
                      0       .                     1       .       2
        .       3
    <--- X0 o         .   <----Y1 o         .       o 2X--->  .     o 3Y--->
------------------------------------------------------------
模式 4：              .   模式 5：          .   模式 6：          .   模式 7：

    <--- X4 o         .   <--- Y5 o         .       o 6X--->  .     o 7Y--->
                      4       .                     5       .       6
        .       7
                      Y       .                     X       .       Y
        .       X
                      |       .                     |       .       |
        .       |
                      V       .                     V       .       V
        .       V
------------------------------------------------------------
```



```
                                |                                   |
                                |                                   |
                                |-----------------|
                                屏幕正面（宽 240，高 320）

 *******************************************************/
void ILI9341_GramScan ( uint8_t ucOption )
{
    //参数检查，只可输入 0-7
    if(ucOption >7 )
        return;

    //根据模式更新 LCD_SCAN_MODE 的值，主要用于触摸屏选择计算参数
    LCD_SCAN_MODE = ucOption;

    //根据模式更新 XY 方向的像素宽度
    if(ucOption%2 == 0)
    {
        //0 2 4 6 模式下 X 方向像素宽度为 240，Y 方向为 320
        LCD_X_LENGTH = ILI9341_LESS_PIXEL;
        LCD_Y_LENGTH =  ILI9341_MORE_PIXEL;
    }
    else
    {
        //1 3 5 7 模式下 X 方向像素宽度为 320，Y 方向为 240
        LCD_X_LENGTH = ILI9341_MORE_PIXEL;
        LCD_Y_LENGTH =  ILI9341_LESS_PIXEL;
    }

    //0x36 命令参数的高 3 位可用于设置 GRAM 扫描方向
    ILI9341_Write_Cmd ( 0x36 );
    ILI9341_Write_Data ( 0x08 |(ucOption<<5));//根据 ucOption 的值设置 LCD 参数，
共 0-7 种模式
    ILI9341_Write_Cmd ( CMD_SetCoordinateX );
    ILI9341_Write_Data ( 0x00 );        /* x 起始坐标高 8 位 */
    ILI9341_Write_Data ( 0x00 );        /* x 起始坐标低 8 位 */
    ILI9341_Write_Data ( ((LCD_X_LENGTH-1)>>8)&0xFF ); /* x 结束坐标高 8 位 */

    ILI9341_Write_Data ( (LCD_X_LENGTH-1)&0xFF );              /* x 结束坐标低
8 位 */

    ILI9341_Write_Cmd ( CMD_SetCoordinateY );
    ILI9341_Write_Data ( 0x00 );        /* y 起始坐标高 8 位 */
    ILI9341_Write_Data ( 0x00 );        /* y 起始坐标低 8 位 */
```

```
    ILI9341_Write_Data ( ((LCD_Y_LENGTH-1)>>8)&0xFF ); /* y 结束坐标高 8 位 */

    ILI9341_Write_Data ( (LCD_Y_LENGTH-1)&0xFF );              /* y 结束坐标低
8 位 */

    /* write gram start */
    ILI9341_Write_Cmd ( CMD_SetPixel );
}
```

3、开窗函数：

```
/**
 * @brief  在 ILI9341 显示器上开辟一个窗口
 * @param  usX ：在特定扫描方向下窗口的起点 X 坐标
 * @param  usY ：在特定扫描方向下窗口的起点 Y 坐标
 * @param  usWidth ：窗口的宽度
 * @param  usHeight ：窗口的高度
 * @retval 无
 */
void ILI9341_OpenWindow ( uint16_t usX, uint16_t usY, uint16_t usWidth, uint16_t usHeight )
{
    ILI9341_Write_Cmd ( CMD_SetCoordinateX );                    /* 设置 X 坐标 */
    ILI9341_Write_Data ( usX >> 8  );    /* 先高 8 位，然后低 8 位 */
    ILI9341_Write_Data ( usX & 0xff  );  /* 设置起始点和结束点*/
    ILI9341_Write_Data ( ( usX + usWidth - 1 ) >> 8  );
    ILI9341_Write_Data ( ( usX + usWidth - 1 ) & 0xff  );

    ILI9341_Write_Cmd ( CMD_SetCoordinateY );                   /* 设置 Y 坐标*/
    ILI9341_Write_Data ( usY >> 8  );
    ILI9341_Write_Data ( usY & 0xff  );
    ILI9341_Write_Data ( ( usY + usHeight - 1 ) >> 8 );
    ILI9341_Write_Data ( ( usY + usHeight - 1) & 0xff );

}
```

4、光标设置

```
/**
 * @brief  设定 ILI9341 的光标坐标
 * @param  usX ：在特定扫描方向下光标的 X 坐标
 * @param  usY ：在特定扫描方向下光标的 Y 坐标
 * @retval 无
 */
static void ILI9341_SetCursor ( uint16_t usX, uint16_t usY )
{
    ILI9341_OpenWindow ( usX, usY, 1, 1 );
}
```

5、像素点填充

```
/**
 * @brief  在 ILI9341 显示器上以某一颜色填充像素点
 * @param  ulAmout_Point ：要填充颜色的像素点的总数目
 * @param  usColor ：颜色
 * @retval 无
 */
static __inline void ILI9341_FillColor ( uint32_t ulAmout_Point, uint16_t usColor )
{
    uint32_t i = 0;


    /* memory write */
    ILI9341_Write_Cmd ( CMD_SetPixel );

    for ( i = 0; i < ulAmout_Point; i ++ )
        ILI9341_Write_Data ( usColor );
}
```

6、对 ILI9341 显示器的某一点以某种颜色进行填充

```
/**
 * @brief  对 ILI9341 显示器的某一点以某种颜色进行填充
 * @param  usX ：在特定扫描方向下该点的 X 坐标
 * @param  usY ：在特定扫描方向下该点的 Y 坐标
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_SetPointPixel ( uint16_t usX, uint16_t usY )
{
    if ( ( usX < LCD_X_LENGTH ) && ( usY < LCD_Y_LENGTH ) )
  {
        ILI9341_SetCursor ( usX, usY );

        ILI9341_FillColor ( 1, CurrentTextColor );
    }

}
```

7、读取 ILI9341 GRAN 的一个像素数据

```
/**
 * @brief  读取 ILI9341 GRAN 的一个像素数据
 * @param  无
 * @retval 像素数据
 */
static uint16_t ILI9341_Read_PixelData ( void )
```

```
{
    uint16_t usR=0, usG=0, usB=0 ;


    ILI9341_Write_Cmd ( 0x2E );   /* 读数据 */

    usR = ILI9341_Read_Data ();     /*FIRST READ OUT DUMMY DATA*/

    usR = ILI9341_Read_Data ();     /*READ OUT RED DATA  */
    usB = ILI9341_Read_Data ();     /*READ OUT BLUE DATA*/
    usG = ILI9341_Read_Data ();     /*READ OUT GREEN DATA*/

  return ( ( ( usR >> 11 ) << 11 ) | ( ( usG >> 10 ) << 5 ) | ( usB >> 11 ) );

}
```

8、获取 ILI9341 显示器上某一个坐标点的像素数据

```
/**
 * @brief
 * @param  usX ：在特定扫描方向下该点的 X 坐标
 * @param  usY ：在特定扫描方向下该点的 Y 坐标
 * @retval 像素数据
 */
uint16_t ILI9341_GetPointPixel ( uint16_t usX, uint16_t usY )
{
    uint16_t usPixelData;


    ILI9341_SetCursor ( usX, usY );

    usPixelData = ILI9341_Read_PixelData ();

    return usPixelData;

}
```

9、Bresenham 算法画线函数

```
/**
 * @brief  在 ILI9341 显示器上使用 Bresenham 算法画线段
 * @param  usX1 ：在特定扫描方向下线段的一个端点 X 坐标
 * @param  usY1 ：在特定扫描方向下线段的一个端点 Y 坐标
 * @param  usX2 ：在特定扫描方向下线段的另一个端点 X 坐标
 * @param  usY2 ：在特定扫描方向下线段的另一个端点 Y 坐标
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
```

```
 * @retval 无
 */
void ILI9341_DrawLine ( uint16_t usX1, uint16_t usY1, uint16_t usX2, uint16_t usY2 )
{
    uint16_t us;
    uint16_t usX_Current, usY_Current;

    int32_t lError_X = 0, lError_Y = 0, lDelta_X, lDelta_Y, lDistance;
    int32_t lIncrease_X, lIncrease_Y;


    lDelta_X = usX2 - usX1; //计算坐标增量
    lDelta_Y = usY2 - usY1;

    usX_Current = usX1;
    usY_Current = usY1;


    if ( lDelta_X > 0 )
        lIncrease_X = 1; //设置单步方向

    else if ( lDelta_X == 0 )
        lIncrease_X = 0;//垂直线

    else
  {
    lIncrease_X = -1;
    lDelta_X = - lDelta_X;
  }


    if ( lDelta_Y > 0 )
        lIncrease_Y = 1;

    else if ( lDelta_Y == 0 )
        lIncrease_Y = 0;//水平线

    else
  {
    lIncrease_Y = -1;
    lDelta_Y = - lDelta_Y;
  }
```

```
    if (  lDelta_X > lDelta_Y )
        lDistance = lDelta_X; //选取基本增量坐标轴

    else
        lDistance = lDelta_Y;


    for ( us = 0; us <= lDistance + 1; us ++ )//画线输出
    {
        ILI9341_SetPointPixel ( usX_Current, usY_Current );//画点

        lError_X += lDelta_X ;
        lError_Y += lDelta_Y ;

        if ( lError_X > lDistance )
        {
            lError_X -= lDistance;
            usX_Current += lIncrease_X;
        }

        if ( lError_Y > lDistance )
        {
            lError_Y -= lDistance;
            usY_Current += lIncrease_Y;
        }

    }


}
```

10、Bresenham 算法画圆

```
/**
 * @brief  在 ILI9341 显示器上使用 Bresenham 算法画圆
 * @param  usX_Center ：在特定扫描方向下圆心的 X 坐标
 * @param  usY_Center ：在特定扫描方向下圆心的 Y 坐标
 * @param  usRadius：圆的半径（单位：像素）
 * @param  ucFilled ：选择是否填充该圆
  *   该参数为以下值之一：
  *     @arg 0 :空心圆
  *     @arg 1 :实心圆
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DrawCircle ( uint16_t usX_Center, uint16_t usY_Center, uint16_t usRadius, uint8_t ucFilled )
```

```
{
    int16_t sCurrentX, sCurrentY;
    int16_t sError;


    sCurrentX = 0; sCurrentY = usRadius;

    sError = 3 - ( usRadius << 1 );     //判断下个点位置的标志


    while ( sCurrentX <= sCurrentY )
    {
        int16_t sCountY;


        if ( ucFilled )
            for ( sCountY = sCurrentX; sCountY <= sCurrentY; sCountY ++ )
            {
                ILI9341_SetPointPixel ( usX_Center + sCurrentX, usY_Center + sCountY );           //1,
研究对象
                ILI9341_SetPointPixel ( usX_Center - sCurrentX, usY_Center + sCountY );           //2
                ILI9341_SetPointPixel ( usX_Center - sCountY,   usY_Center + sCurrentX );         //3
                ILI9341_SetPointPixel ( usX_Center - sCountY,   usY_Center - sCurrentX );         //4
                ILI9341_SetPointPixel ( usX_Center - sCurrentX, usY_Center - sCountY );           //5
            ILI9341_SetPointPixel ( usX_Center + sCurrentX, usY_Center - sCountY );           //6
                ILI9341_SetPointPixel ( usX_Center + sCountY,    usY_Center - sCurrentX );          //7

            ILI9341_SetPointPixel ( usX_Center + sCountY,   usY_Center + sCurrentX );         //0

            }

        else
        {
            ILI9341_SetPointPixel ( usX_Center + sCurrentX, usY_Center + sCurrentY );            //1, 研
究对象
            ILI9341_SetPointPixel ( usX_Center - sCurrentX, usY_Center + sCurrentY );            //2
            ILI9341_SetPointPixel ( usX_Center - sCurrentY, usY_Center + sCurrentX );            //3
            ILI9341_SetPointPixel ( usX_Center - sCurrentY, usY_Center - sCurrentX );            //4
            ILI9341_SetPointPixel ( usX_Center - sCurrentX, usY_Center - sCurrentY );            //5
            ILI9341_SetPointPixel ( usX_Center + sCurrentX, usY_Center - sCurrentY );            //6
            ILI9341_SetPointPixel ( usX_Center + sCurrentY, usY_Center - sCurrentX );            //7
            ILI9341_SetPointPixel ( usX_Center + sCurrentY, usY_Center + sCurrentX );            //0
  }
```

```
            sCurrentX ++;

            if ( sError < 0 )
                sError += 4 * sCurrentX + 6;
            else
            {
                sError += 10 + 4 * ( sCurrentX - sCurrentY );
                sCurrentY --;
            }
        }
}
```

11、显示一个英文字符

```
/**
 * @brief  在 ILI9341 显示器上显示一个英文字符
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下该点的起始 Y 坐标
 * @param  cChar ：要显示的英文字符
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispChar_EN ( uint16_t usX, uint16_t usY, const char cChar )
{
    uint8_t  byteCount, bitCount,fontLength;
    uint16_t ucRelativePositon;
    uint8_t *Pfont;

    //对 ascii 码表偏移（字模表不包含 ASCII 表的前 32 个非图形符号）
    ucRelativePositon = cChar - ' ';

    //每个字模的字节数
    fontLength = (LCD_Currentfonts->Width*LCD_Currentfonts->Height)/8;

    //字模首地址
    /*ascii 码表偏移值乘以每个字模的字节数，求出字模的偏移位置*/
    Pfont = (uint8_t *)&LCD_Currentfonts->table[ucRelativePositon * fontLength];

    //设置显示窗口
    ILI9341_OpenWindow ( usX, usY, LCD_Currentfonts->Width, LCD_Currentfonts->Height);

    ILI9341_Write_Cmd ( CMD_SetPixel );

    //按字节读取字模数据
    //由于前面直接设置了显示窗口，显示数据会自动换行
    for ( byteCount = 0; byteCount < fontLength; byteCount++ )
```

```
    {
            //一位一位处理要显示的颜色
            for ( bitCount = 0; bitCount < 8; bitCount++ )
            {
                if ( Pfont[byteCount] & (0x80>>bitCount) )
                    ILI9341_Write_Data ( CurrentTextColor );
                else
                    ILI9341_Write_Data ( CurrentBackColor );
            }
    }
}
```

12、显示英文字符串

```
/**
 * @brief  在 ILI9341 显示器上显示英文字符串
 * @param  line ：在特定扫描方向下字符串的起始 Y 坐标
  *   本参数可使用宏 LINE(0)、LINE(1)等方式指定文字坐标，
  *   宏 LINE(x)会根据当前选择的字体来计算 Y 坐标值。
    *         显示中文且使用 LINE 宏时，需要把英文字体设置成 Font8x16
 * @param  pStr ：要显示的英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispStringLine_EN (  uint16_t line,  char * pStr )
{
    uint16_t usX = 0;

    while ( * pStr != '\0' )
    {
        if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) > LCD_X_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            line += LCD_Currentfonts->Height;
        }

        if ( ( line - ILI9341_DispWindow_Y_Star + LCD_Currentfonts->Height ) > LCD_Y_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            line = ILI9341_DispWindow_Y_Star;
        }

        ILI9341_DispChar_EN ( usX, line, * pStr);

        pStr ++;
```

```
        usX += LCD_Currentfonts->Width;
    }
}
```

13、通过行命令显示英文字符串

```
/**
 * @brief  在 ILI9341 显示器上显示英文字符串
 * @param  line ：在特定扫描方向下字符串的起始 Y 坐标
  *   本参数可使用宏 LINE(0)、LINE(1)等方式指定文字坐标，
  *   宏 LINE(x)会根据当前选择的字体来计算 Y 坐标值。
    *       显示中文且使用 LINE 宏时，需要把英文字体设置成 Font8x16
 * @param  pStr ：要显示的英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispStringLine_EN (  uint16_t line,  char * pStr )
{
    uint16_t usX = 0;

    while ( * pStr != '\0' )
    {
        if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) >
LCD_X_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            line += LCD_Currentfonts->Height;
        }

        if ( ( line - ILI9341_DispWindow_Y_Star + LCD_Currentfonts->Height ) >
LCD_Y_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            line = ILI9341_DispWindow_Y_Star;
        }

        ILI9341_DispChar_EN ( usX, line, * pStr);

        pStr ++;

        usX += LCD_Currentfonts->Width;
    }
}
```

14、显示英文字符串

```
/**
```

```
 * @brief  在 ILI9341 显示器上显示英文字符串
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下字符的起始 Y 坐标
 * @param  pStr ：要显示的英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispString_EN (   uint16_t usX ,uint16_t usY,  char * pStr )
{
    while ( * pStr != '\0' )
    {
        if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) >
LCD_X_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            usY += LCD_Currentfonts->Height;
        }

        if ( ( usY - ILI9341_DispWindow_Y_Star + LCD_Currentfonts->Height ) >
LCD_Y_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            usY = ILI9341_DispWindow_Y_Star;
        }
        ILI9341_DispChar_EN ( usX, usY, * pStr);
        pStr ++;
        usX += LCD_Currentfonts->Width;
    }
}
```

15、显示英文字符串(沿 Y 轴方向)

```
/**
 * @brief  在 ILI9341 显示器上显示英文字符串(沿 Y 轴方向)
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下字符的起始 Y 坐标
 * @param  pStr ：要显示的英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispString_EN_YDir (   uint16_t usX,uint16_t usY ,  char * pStr )
{
    while ( * pStr != '\0' )
    {
        if      (      (      usY      -      ILI9341_DispWindow_Y_Star      +
LCD_Currentfonts->Height ) >LCD_Y_LENGTH  )
```

```
        {
            usY = ILI9341_DispWindow_Y_Star;
            usX += LCD_Currentfonts->Width;
        }

        if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) >
LCD_X_LENGTH)
        {
            usX = ILI9341_DispWindow_X_Star;
            usY = ILI9341_DispWindow_Y_Star;
        }

        ILI9341_DispChar_EN ( usX, usY, * pStr);

        pStr ++;

        usY += LCD_Currentfonts->Height;
    }
}
```

17、显示一个中文字符

```
/**
 * @brief  在 ILI9341 显示器上显示一个中文字符
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下字符的起始 Y 坐标
 * @param  usChar ：要显示的中文字符（国标码）
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispChar_CH ( uint16_t usX, uint16_t usY, uint16_t usChar )
{
    uint8_t rowCount, bitCount;
    uint8_t ucBuffer [ WIDTH_CH_CHAR*HEIGHT_CH_CHAR/8 ];
    uint16_t usTemp;

    //设置显示窗口
    ILI9341_OpenWindow ( usX, usY, WIDTH_CH_CHAR, HEIGHT_CH_CHAR );

    ILI9341_Write_Cmd ( CMD_SetPixel );

    //取字模数据
    GetGBKCode ( ucBuffer, usChar );

    for ( rowCount = 0; rowCount < HEIGHT_CH_CHAR; rowCount++ )
```

```
    {
    /* 取出两个字节的数据，在 lcd 上即是一个汉字的一行 */
        usTemp = ucBuffer [ rowCount * 2 ];
        usTemp = ( usTemp << 8 );
        usTemp |= ucBuffer [ rowCount * 2 + 1 ];

        for ( bitCount = 0; bitCount < WIDTH_CH_CHAR; bitCount ++ )
        {
            if ( usTemp & ( 0x8000 >> bitCount ) )  //高位在前
              ILI9341_Write_Data ( CurrentTextColor );
            else
                ILI9341_Write_Data ( CurrentBackColor );
        }
    }
}
```

18、显示中文字符串

```
/**
 * @brief  在 ILI9341 显示器上显示中文字符串
 * @param  line ：在特定扫描方向下字符串的起始 Y 坐标
  *   本参数可使用宏 LINE(0)、LINE(1)等方式指定文字坐标，
  *   宏 LINE(x)会根据当前选择的字体来计算 Y 坐标值。
    *       显示中文且使用 LINE 宏时，需要把英文字体设置成 Font8x16
 * @param  pStr ：要显示的英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispString_CH (    uint16_t usX , uint16_t usY, char * pStr )
{
    uint16_t usCh;
    while( * pStr != '\0' )
    {
        if ( ( usX - ILI9341_DispWindow_X_Star + WIDTH_CH_CHAR ) > LCD_X_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            usY += HEIGHT_CH_CHAR;
        }

        if ( ( usY - ILI9341_DispWindow_Y_Star + HEIGHT_CH_CHAR ) > LCD_Y_LENGTH )
        {
            usX = ILI9341_DispWindow_X_Star;
            usY = ILI9341_DispWindow_Y_Star;
        }
```

```
        usCh = * ( uint16_t * ) pStr;
        usCh = ( usCh << 8 ) + ( usCh >> 8 );

        ILI9341_DispChar_CH ( usX, usY, usCh );
        usX += WIDTH_CH_CHAR;
        pStr += 2;           //一个汉字两个字节
    }
}
```

19、显示中英文字符串

```
/**
 * @brief  在 ILI9341 显示器上显示中英文字符串
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下字符的起始 Y 坐标
 * @param  pStr ：要显示的字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispString_EN_CH (     uint16_t usX , uint16_t usY, char * pStr )
{
    uint16_t usCh;

    while( * pStr != '\0' )
    {
        if ( * pStr <= 126 )                //英文字符
        {
            if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) >
LCD_X_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                usY += LCD_Currentfonts->Height;
            }

            if ( ( usY - ILI9341_DispWindow_Y_Star + LCD_Currentfonts->Height ) >
LCD_Y_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                usY = ILI9341_DispWindow_Y_Star;
            }
          ILI9341_DispChar_EN ( usX, usY, * pStr );
          usX +=  LCD_Currentfonts->Width;
          pStr ++;
        }
        else                                //汉字字符
```

```
        {
            if ( ( usX - ILI9341_DispWindow_X_Star + WIDTH_CH_CHAR ) >
LCD_X_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                usY += HEIGHT_CH_CHAR;
            }

            if ( ( usY - ILI9341_DispWindow_Y_Star + HEIGHT_CH_CHAR ) >
LCD_Y_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                usY = ILI9341_DispWindow_Y_Star;
            }

            usCh = * ( uint16_t * ) pStr;
            usCh = ( usCh << 8 ) + ( usCh >> 8 );
            ILI9341_DispChar_CH ( usX, usY, usCh );
            usX += WIDTH_CH_CHAR;
            pStr += 2;           //一个汉字两个字节
    }
  }
}
```

20、显示中英文字符串

```
/**
 * @brief  在 ILI9341 显示器上显示中英文字符串
 * @param  line ：在特定扫描方向下字符串的起始 Y 坐标
  *   本参数可使用宏 LINE(0)、LINE(1)等方式指定文字坐标，
  *   宏 LINE(x)会根据当前选择的字体来计算 Y 坐标值。
    *       显示中文且使用 LINE 宏时，需要把英文字体设置成 Font8x16
 * @param  pStr ：要显示的字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispStringLine_EN_CH (  uint16_t line, char * pStr )
{
    uint16_t usCh;
    uint16_t usX = 0;

    while( * pStr != '\0' )
    {
        if ( * pStr <= 126 )               //英文字符
        {
```

```
            if ( ( usX - ILI9341_DispWindow_X_Star + LCD_Currentfonts->Width ) >
LCD_X_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                line += LCD_Currentfonts->Height;
            }

            if ( ( line - ILI9341_DispWindow_Y_Star + LCD_Currentfonts->Height ) >
LCD_Y_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                line = ILI9341_DispWindow_Y_Star;
            }

          ILI9341_DispChar_EN ( usX, line, * pStr );
          usX +=  LCD_Currentfonts->Width;
          pStr ++;

        }

        else                                    //汉字字符
        {
            if  (  (  usX  -  ILI9341_DispWindow_X_Star  +  WIDTH_CH_CHAR  )  >
LCD_X_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                line += HEIGHT_CH_CHAR;
            }

            if  (  (  line  -  ILI9341_DispWindow_Y_Star  +  HEIGHT_CH_CHAR  )  >
LCD_Y_LENGTH )
            {
                usX = ILI9341_DispWindow_X_Star;
                line = ILI9341_DispWindow_Y_Star;
            }

            usCh = * ( uint16_t * ) pStr;

            usCh = ( usCh << 8 ) + ( usCh >> 8 );

            ILI9341_DispChar_CH ( usX, line, usCh );

            usX += WIDTH_CH_CHAR;
```

```
            pStr += 2;            //一个汉字两个字节

    }

  }
}
```

21、显示中英文字符串(沿 Y 轴方向)

```
/**
 * @brief  在 ILI9341 显示器上显示中英文字符串(沿 Y 轴方向)
 * @param  usX ：在特定扫描方向下字符的起始 X 坐标
 * @param  usY ：在特定扫描方向下字符的起始 Y 坐标
 * @param  pStr ：要显示的中英文字符串的首地址
 * @note 可使用 LCD_SetBackColor、LCD_SetTextColor、LCD_SetColors 函数设置颜色
 * @retval 无
 */
void ILI9341_DispString_EN_CH_YDir (  uint16_t usX,uint16_t usY , char * pStr )
{
    uint16_t usCh;

    while( * pStr != '\0' )
    {
            //统一使用汉字的宽高来计算换行
            if     (     (     usY     -     ILI9341_DispWindow_Y_Star     +
HEIGHT_CH_CHAR ) >LCD_Y_LENGTH  )
            {
                usY = ILI9341_DispWindow_Y_Star;
                usX += WIDTH_CH_CHAR;
            }
            if  (  (  usX  -  ILI9341_DispWindow_X_Star  +  WIDTH_CH_CHAR  )  >
LCD_X_LENGTH)
            {
                usX = ILI9341_DispWindow_X_Star;
                usY = ILI9341_DispWindow_Y_Star;
            }

        //显示
        if ( * pStr <= 126 )                //英文字符
        {
            ILI9341_DispChar_EN ( usX, usY, * pStr);

            pStr ++;

            usY += HEIGHT_CH_CHAR;
```

```
        }
        else                                    //汉字字符
        {
            usCh = * ( uint16_t * ) pStr;
            usCh = ( usCh << 8 ) + ( usCh >> 8 );
            ILI9341_DispChar_CH ( usX,usY , usCh );
            usY += HEIGHT_CH_CHAR;
            pStr += 2;          //一个汉字两个字节
    }
  }
}
```

22、缩放字体部分

```
#define ZOOMMAXBUFF 16384
uint8_t zoomBuff[ZOOMMAXBUFF] = {0};    //用于缩放的缓存，最大支持到 128*128
uint8_t zoomTempBuff[1024] = {0};

/**
 * @brief  缩放字模，缩放后的字模由 1 个像素点由 8 个数据位来表示
                                        0x01 表示笔迹，0x00 表示空白区
 * @param  in_width ：原始字符宽度
 * @param  in_heig ：原始字符高度
 * @param  out_width ：缩放后的字符宽度
 * @param  out_heig：缩放后的字符高度
 * @param  in_ptr ：字库输入指针    注意：1pixel 1bit
 * @param  out_ptr ：缩放后的字符输出指针 注意: 1pixel 8bit
 *      out_ptr 实际上没有正常输出，改成了直接输出到全局指针 zoomBuff 中
 * @param  en_cn ：0 为英文，1 为中文
 * @retval 无
 */
void ILI9341_zoomChar(uint16_t in_width,    //原始字符宽度
                      uint16_t in_heig,     //原始字符高度
                      uint16_t out_width,   //缩放后的字符宽度
                      uint16_t out_heig,    //缩放后的字符高度
                      uint8_t *in_ptr,      //字库输入指针  注意：1pixel 1bit
                      uint8_t *out_ptr,     //缩放后的字符输出指针 注意: 1pixel
8bit
                      uint8_t en_cn)        //0 为英文，1 为中文
{
    uint8_t *pts,*ots;
    //根据源字模及目标字模大小，设定运算比例因子，左移 16 是为了把浮点运算转成定
点运算
    unsigned int xrIntFloat_16=(in_width<<16)/out_width+1;
  unsigned int yrIntFloat_16=(in_heig<<16)/out_heig+1;
```

```
    unsigned int srcy_16=0;
    unsigned int y,x;
    uint8_t *pSrcLine;

    uint16_t byteCount,bitCount;

    //检查参数是否合法
    if(in_width >= 32) return;                                          //
字库不允许超过 32 像素
    if(in_width * in_heig == 0) return;
    if(in_width * in_heig >= 1024 ) return;                    //限制输入最大
32*32

    if(out_width * out_heig == 0) return;
    if(out_width * out_heig >= ZOOMMAXBUFF ) return; //限制最大缩放 128*128
    pts = (uint8_t*)&zoomTempBuff;

    //为方便运算，字库的数据由 1 pixel/1bit 映射到 1pixel/8bit
    //0x01 表示笔迹，0x00 表示空白区
    if(en_cn == 0x00)//英文
    {
        //英文和中文字库上下边界不对，可在此处调整。需要注意 tempBuff 防止溢出
            for(byteCount=0;byteCount<in_heig*in_width/8;byteCount++)
            {
                for(bitCount=0;bitCount<8;bitCount++)
                    {
                        //把源字模数据由位映射到字节
                        //in_ptr 里 bitX 为 1，则 pts 里整个字节值为 1
                        //in_ptr 里 bitX 为 0，则 pts 里整个字节值为 0
                        *pts++ = (in_ptr[byteCount] & (0x80>>bitCount))?1:0;
                    }
            }
    }
    else //中文
    {
            for(byteCount=0;byteCount<in_heig*in_width/8;byteCount++)
            {
                for(bitCount=0;bitCount<8;bitCount++)
                    {
                        //把源字模数据由位映射到字节
                        //in_ptr 里 bitX 为 1，则 pts 里整个字节值为 1
                        //in_ptr 里 bitX 为 0，则 pts 里整个字节值为 0
                        *pts++ = (in_ptr[byteCount] & (0x80>>bitCount))?1:0;
```

```
                }
            }
    }

    //zoom 过程
    pts = (uint8_t*)&zoomTempBuff;  //映射后的源数据指针
    ots = (uint8_t*)&zoomBuff;  //输出数据的指针
    for (y=0;y<out_heig;y++)    /*行遍历*/
    {
            unsigned int srcx_16=0;
        pSrcLine=pts+in_width*(srcy_16>>16);
        for (x=0;x<out_width;x++) /*行内像素遍历*/
        {
            ots[x]=pSrcLine[srcx_16>>16]; //把源字模数据复制到目标指针中
            srcx_16+=xrIntFloat_16;         //按比例偏移源像素点
        }
        srcy_16+=yrIntFloat_16;               //按比例偏移源像素点
        ots+=out_width;
    }
    /*！！！缩放后的字模数据直接存储到全局指针 zoomBuff 里了*/
    out_ptr = (uint8_t*)&zoomBuff;  //out_ptr 没有正确传出，后面调用直接改成了全局变量指针！

    /*实际中如果使用 out_ptr 不需要下面这一句！！！
        只是因为 out_ptr 没有使用，会导致 warning。强迫症*/
    out_ptr++;
}


/**
 * @brief  利用缩放后的字模显示字符
 * @param  Xpos ： 字符显示位置 x
 * @param  Ypos ： 字符显示位置 y
 * @param  Font_width ： 字符宽度
 * @param  Font_Heig：字符高度
 * @param  c ： 要显示的字模数据
 * @param  DrawModel ： 是否反色显示
 * @retval 无
 */
void ILI9341_DrawChar_Ex(uint16_t usX,             //字符显示位置 x
                         uint16_t usY,             //字符显示位置 y
                         uint16_t Font_width,      //字符宽度
                         uint16_t Font_Height,     //字符高度
                         uint8_t *c,               //字模数据
```

```
                        uint16_t DrawModel)          //是否反色显示
{
  uint32_t index = 0, counter = 0;

    //设置显示窗口
    ILI9341_OpenWindow ( usX, usY, Font_width, Font_Height);

    ILI9341_Write_Cmd ( CMD_SetPixel );

    //按字节读取字模数据
    //由于前面直接设置了显示窗口，显示数据会自动换行
    for ( index = 0; index < Font_Height; index++ )
    {
            //一位一位处理要显示的颜色
            for ( counter = 0; counter < Font_width; counter++ )
            {
                    //缩放后的字模数据，以一个字节表示一个像素位
                    //整个字节值为 1 表示该像素为笔迹
                    //整个字节值为 0 表示该像素为背景
                    if ( *c++ == DrawModel )
                        ILI9341_Write_Data ( CurrentBackColor );
                    else
                        ILI9341_Write_Data ( CurrentTextColor );
            }
    }
}


/**
 * @brief  利用缩放后的字模显示字符串
 * @param  Xpos ：字符显示位置 x
 * @param  Ypos ：字符显示位置 y
 * @param  Font_width ：字符宽度，英文字符在此基础上/2。注意为偶数
 * @param  Font_Heig：字符高度，注意为偶数
 * @param  c ：要显示的字符串
 * @param  DrawModel ：是否反色显示
 * @retval 无
 */
void ILI9341_DisplayStringEx(uint16_t x,               //字符显示位置 x
                             uint16_t y,               //字符显示位置 y
                             uint16_t Font_width,      //要显示的字体宽度,英文
字符在此基础上/2。注意为偶数
                             uint16_t Font_Height,     //要显示的字体高度,注意
为偶数
```

```
                            uint8_t *ptr,               //显示的字符内容
                            uint16_t DrawModel)         //是否反色显示


{
    uint16_t Charwidth = Font_width; //默认为Font_width，英文宽度为中文宽度的一
半
    uint8_t *psr;
    uint8_t Ascii;  //英文
    uint16_t usCh;  //中文
    uint8_t ucBuffer [ WIDTH_CH_CHAR*HEIGHT_CH_CHAR/8 ];

    while ( *ptr != '\0')
    {
            /****处理换行*****/
            if ( ( x - ILI9341_DispWindow_X_Star + Charwidth ) > LCD_X_LENGTH )
            {
                x = ILI9341_DispWindow_X_Star;
                y += Font_Height;
            }

            if ( ( y - ILI9341_DispWindow_Y_Star + Font_Height ) > LCD_Y_LENGTH )
            {
                x = ILI9341_DispWindow_X_Star;
                y = ILI9341_DispWindow_Y_Star;
            }

        if(*ptr > 0x80) //如果是中文
        {
            Charwidth = Font_width;
            usCh = * ( uint16_t * ) ptr;
            usCh = ( usCh << 8 ) + ( usCh >> 8 );
            GetGBKCode ( ucBuffer, usCh );  //取字模数据
            //缩放字模数据，源字模为16*16

    ILI9341_zoomChar(WIDTH_CH_CHAR,HEIGHT_CH_CHAR,Charwidth,Font_Height,(uint8
_t *)&ucBuffer,psr,1);
            //显示单个字符

    ILI9341_DrawChar_Ex(x,y,Charwidth,Font_Height,(uint8_t*)&zoomBuff,DrawMode
l);
            x+=Charwidth;
            ptr+=2;
```

```
        }
        else
        {
                Charwidth = Font_width / 2;
                Ascii = *ptr - 32;
                //使用 16*24 字体缩放字模数据
                ILI9341_zoomChar(16,24,Charwidth,Font_Height,(uint8_t
*)&Font16x24.table[Ascii * Font16x24.Height*Font16x24.Width/8],psr,0);
              //显示单个字符

    ILI9341_DrawChar_Ex(x,y,Charwidth,Font_Height,(uint8_t*)&zoomBuff,DrawMode
l);
                x+=Charwidth;
                ptr++;
        }
    }
}


/**
 * @brief  利用缩放后的字模显示字符串(沿 Y 轴方向)
 * @param  Xpos : 字符显示位置 x
 * @param  Ypos : 字符显示位置 y
 * @param  Font_width : 字符宽度，英文字符在此基础上/2。注意为偶数
 * @param  Font_Heig: 字符高度，注意为偶数
 * @param  c : 要显示的字符串
 * @param  DrawModel : 是否反色显示
 * @retval 无
 */
void ILI9341_DisplayStringEx_YDir(uint16_t x,              //字符显示位置 x
                                  uint16_t y,              //字符显示位置 y
                                  uint16_t Font_width,     //要显示的字体宽度，
英文字符在此基础上/2。注意为偶数
                                  uint16_t Font_Height,    //要显示的字体高度，
注意为偶数
                                  uint8_t *ptr,            //显示的字符内容
                                  uint16_t DrawModel)      //是否反色显示
{
    uint16_t Charwidth = Font_width; //默认为 Font_width，英文宽度为中文宽度的一
半
    uint8_t *psr;
    uint8_t Ascii;  //英文
    uint16_t usCh;  //中文
    uint8_t ucBuffer [ WIDTH_CH_CHAR*HEIGHT_CH_CHAR/8 ];
```

```
    while ( *ptr != '\0')
    {
            //统一使用汉字的宽高来计算换行
            if ( ( y - ILI9341_DispWindow_X_Star + Font_width ) > LCD_X_LENGTH )
            {
                y = ILI9341_DispWindow_X_Star;
                x += Font_width;
            }

            if ( ( x - ILI9341_DispWindow_Y_Star + Font_Height ) > LCD_Y_LENGTH )
            {
                y = ILI9341_DispWindow_X_Star;
                x = ILI9341_DispWindow_Y_Star;
            }

        if(*ptr > 0x80) //如果是中文
        {
            Charwidth = Font_width;
            usCh = * ( uint16_t * ) ptr;
            usCh = ( usCh << 8 ) + ( usCh >> 8 );
            GetGBKCode ( ucBuffer, usCh );  //取字模数据
            //缩放字模数据，源字模为16*16

    ILI9341_zoomChar(WIDTH_CH_CHAR,HEIGHT_CH_CHAR,Charwidth,Font_Height,(uint8
_t *)&ucBuffer,psr,1);
            //显示单个字符

    ILI9341_DrawChar_Ex(x,y,Charwidth,Font_Height,(uint8_t*)&zoomBuff,DrawMode
l);
            y+=Font_Height;
            ptr+=2;
        }
        else
        {
                Charwidth = Font_width / 2;
                Ascii = *ptr - 32;
                //使用16*24字体缩放字模数据
                ILI9341_zoomChar(16,24,Charwidth,Font_Height,(uint8_t
*)&Font16x24.table[Ascii *
    Font16x24.Height*Font16x24.Width/8],psr,0);
              //显示单个字符

    ILI9341_DrawChar_Ex(x,y,Charwidth,Font_Height,(uint8_t*)&zoomBuff,DrawMode
```

```
1);
                y+=Font_Height;
                ptr++;
        }
    }
}
```

23、设置英文字体类型

```
/**
  * @brief  设置英文字体类型
  * @param  fonts: 指定要选择的字体
    *       参数为以下值之一
  *     @arg：Font24x32;
  *     @arg：Font16x24;
  *     @arg：Font8x16;
  * @retval None
  */
void LCD_SetFont(sFONT *fonts)
{
  LCD_Currentfonts = fonts;
}
```

24、获取当前字体类型

```
/**
  * @brief  获取当前字体类型
  * @param  None.
  * @retval 返回当前字体类型
  */
sFONT *LCD_GetFont(void)
{
  return LCD_Currentfonts;
}
```

25、设置 LCD 的前景(字体)及背景颜色

```
/**
  * @brief  设置 LCD 的前景(字体)及背景颜色,RGB565
  * @param  TextColor: 指定前景(字体)颜色
  * @param  BackColor: 指定背景颜色
  * @retval None
  */
void LCD_SetColors(uint16_t TextColor, uint16_t BackColor)
{
  CurrentTextColor = TextColor;
  CurrentBackColor = BackColor;
```

```
}
```

27、获取 LCD 的前景(字体)及背景颜色

```
/**
  * @brief  获取 LCD 的前景(字体)及背景颜色,RGB565
  * @param  TextColor: 用来存储前景(字体)颜色的指针变量
  * @param  BackColor: 用来存储背景颜色的指针变量
  * @retval None
  */
void LCD_GetColors(uint16_t *TextColor, uint16_t *BackColor)
{
  *TextColor = CurrentTextColor;
  *BackColor = CurrentBackColor;
}
```

28、设置 LCD 的前景(字体)颜色

```
/**
  * @brief  设置 LCD 的前景(字体)颜色,RGB565
  * @param  Color: 指定前景(字体)颜色
  * @retval None
  */
void LCD_SetTextColor(uint16_t Color)
{
  CurrentTextColor = Color;
}
/**
  * @brief  设置 LCD 的背景颜色,RGB565
  * @param  Color: 指定背景颜色
  * @retval None
  */
void LCD_SetBackColor(uint16_t Color)
{
  CurrentBackColor = Color;
}
```

30、清除某行文字

```
/**
  * @brief  清除某行文字
  * @param  Line: 指定要删除的行
  *   本参数可使用宏 LINE(0)、LINE(1)等方式指定要删除的行，
  *   宏 LINE(x)会根据当前选择的字体来计算 Y 坐标值，并删除当前字体高度的第 x 行。
  * @retval None
  */
void LCD_ClearLine(uint16_t Line)
```

```
{
  ILI9341_Clear(0,Line,LCD_X_LENGTH,((sFONT *)LCD_GetFont())->Height); /* 清屏，显示全黑 */

}
//==========================================
```
分割线

2021/06/17 添加：

```
void GPIO_SET_DATA_OUT(uint8_t Mode)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    /* 配置液晶相对应的数据线,PORT-D0~D15 */

  if(Mode == GPIO_Set_Mode_OUT)
    {
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_Out_PP;
    }
    else{
        GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_IPU;
    }


    GPIO_InitStructure.GPIO_Pin = ILI9341_D0_PIN;
    GPIO_Init ( ILI9341_D0_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D1_PIN;
    GPIO_Init ( ILI9341_D1_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D2_PIN;
    GPIO_Init ( ILI9341_D2_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D3_PIN;
    GPIO_Init ( ILI9341_D3_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D4_PIN;
    GPIO_Init ( ILI9341_D4_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D5_PIN;
    GPIO_Init ( ILI9341_D5_PORT, &GPIO_InitStructure );
```

```
    GPIO_InitStructure.GPIO_Pin = ILI9341_D6_PIN;
    GPIO_Init ( ILI9341_D6_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D7_PIN;
    GPIO_Init ( ILI9341_D7_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D8_PIN;
    GPIO_Init ( ILI9341_D8_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D9_PIN;
    GPIO_Init ( ILI9341_D9_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D10_PIN;
    GPIO_Init ( ILI9341_D10_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D11_PIN;
    GPIO_Init ( ILI9341_D11_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D12_PIN;
    GPIO_Init ( ILI9341_D12_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D13_PIN;
    GPIO_Init ( ILI9341_D13_PORT, &GPIO_InitStructure );

    GPIO_InitStructure.GPIO_Pin = ILI9341_D14_PIN;
    GPIO_Init ( ILI9341_D14_PORT, &GPIO_InitStructure );


    GPIO_InitStructure.GPIO_Pin = ILI9341_D15_PIN;
    GPIO_Init ( ILI9341_D15_PORT, &GPIO_InitStructure );
}
```