

图片及字符显示例程

实验平台：奋斗版STM32开发板MINI、V2、V2.1、V3

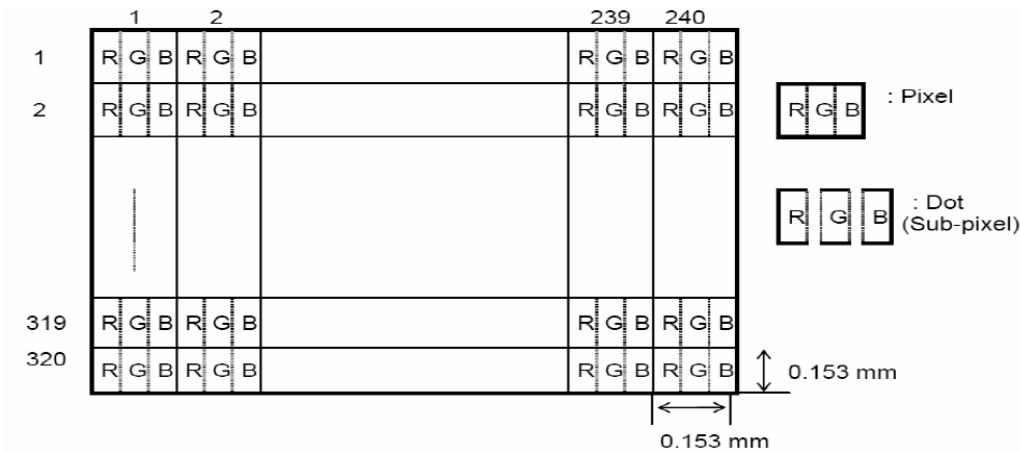
实验内容：本例程演示了在2.4寸TFT屏是显示一副16位色图片，并在图片上透明叠加两个不同显示方向的字符串，该实验学习了2.4寸TFT 16位色显示程序的编制。

预先需要掌握的知识

1. 2.4寸TFT显示模块。

2.4寸TFT显示器：（关于2.4寸TFT的详细资料请参考光盘奋斗板文档目录下\奋斗板配2.4寸显示模块文档\下的ILI9325.pdf和QD024CPS25-36AV0规格书.pdf）2.4显示模块采用的是基于ILI9325驱动的2.4寸TFT显示器（320X240），规格如下：

Item	Display Panel	Remark
Display Mode	Normally White, Transmissive LCD	
Viewing Direction	6 O'CLOCK	
Input Signals	16/18Bits	
Outside Dimensions	42.72mm(W)*59.46mm(H)*2.85(MAX)mm(T)	
Effective Area	-	
Active Area	36.72mm(W)×48.96mm(H)	
Number of Pixels	240×RGB×320Pixels	Note1)
Pixel Pitch	0.153mm(H)×0.153mm(W)	Note1)
Pixel Arrangement	RGB Vertical stripes	Note1)
Drive IC	ILI9325	



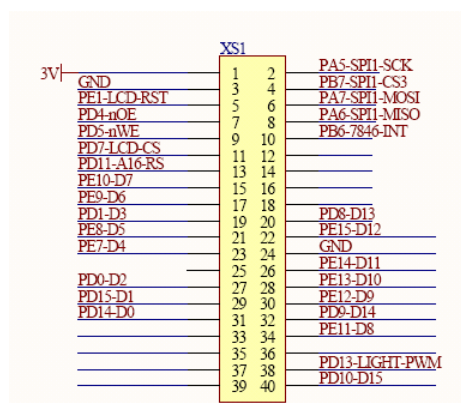
ITEM	Symbol	Min.	Typ.	Max.	Unit	Remark
Power for Circuit Driving	VDD	-0.3	-	3.3	V	
Power for Circuit Logic	VCI	-0.3	-	5.0	V	
LC Operating Voltage *1)	Vop		3.3		V	
LED Forward Voltage	V _f	3.0	-	3.6	V	per LED
LED Forward Current	I _r	-	-	20	mA	per LED
LED Luminance	B _p	-	3000	-	cd/m ²	
Storage Humidity	H _{ST}	10	-	90	%RH	
Storage Temperature	T _{ST}	-40	-	85	°C	At 25±5°C
Operating Ambient Humidity	H _{OP}	10	-	90	%RH	
Operating Ambient temperature	T _{OP}	-20	-	70	°C	

引脚定义

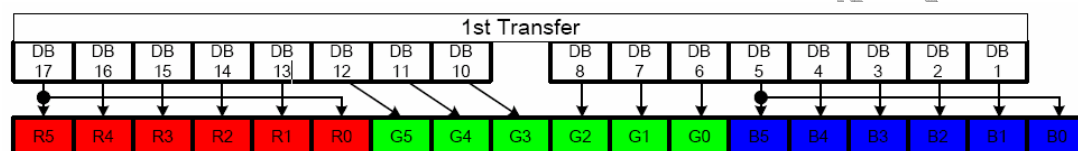
Pin NO.	Symbol	Description
1	NC	NC
2	NC	
3	NC	
4	IM3	IM3:0,16BIT,DB17~DB10,DB8~DB1,DB9&DB0 接地; IM3:1,18BIT,DB17~DB0
5	NC	NC
6	RESET	复位
7	VSYNC	NC
8	HSYNC	
9	DOTCLK	
10	DEN	数据线
11	DB17	
12	DB16	
13	DB15	
14	DB14	
15	DB13	
16	DB12	
17	DB11	
18	DB10	
19	DB9	
20	DB8	
21	DB7	
22	DB6	
23	DB5	
24	DB4	
25	DB3	
26	DB2	
27	DB1	
28	DB0	
29	RD	读信号
30	WR	写信号
31	DC	寄存器/数据选择
32	CS	片选
33	GND	地
34	VCC	电源
35	LED_K	背光负极
36	LED_A	背光正极

2.4 TFT显示屏焊接在奋斗显示转接板上，在屏上贴有触摸屏，通过40芯的接口与V3或者MINI连接。40芯接

口定义如下



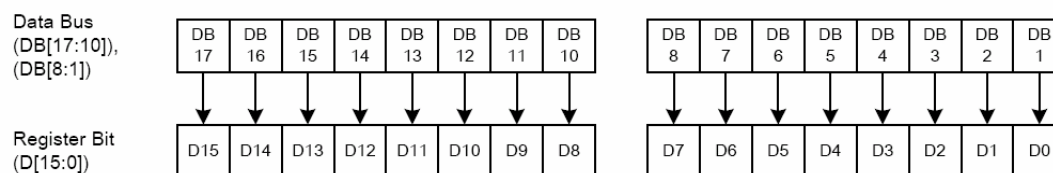
40芯里包含了16位数据线，读写线，命令/数据控制线，片选线，LCD硬件复位线，背光控制线以及触摸控制线。奋斗板V3和MINI就是通过这个接口来控制显示。奋斗板MINI和V3都是选用了具有16位FSMC接口STM32F103VET6作为MCU，FSMC接口也可以称为16位并行接口，时序同I8080接口。按照显示屏驱动电路ILI9325的手册，为了达到色彩与显示效率的平衡，奋斗板采用了16位 64K色接口模式。



在这个模式每个像素用5位红色6位绿色5位蓝色总共16位来表示，根据分辨率，一帧图像占用320*240*2=153600字节。FSMC总线和TFT数据线的连接关系如下

STM32 FSMC	ILI9325	功 能
D15	DB17	数据控制线第15位
D14	DB16	数据控制线第14位
D13	DB15	数据控制线第13位
D12	DB14	数据控制线第12位
D11	DB13	数据控制线第11位
D10	DB12	数据控制线第10位
D9	DB11	数据控制线第9位
D8	DB10	数据控制线第8位
D7	DB8	数据控制线第7位
D6	DB7	数据控制线第6位
D5	DB6	数据控制线第5位
D4	DB5	数据控制线第4位
D3	DB4	数据控制线第3位
D2	DB3	数据控制线第2位
D1	DB2	数据控制线第1位
D0	DB1	数据控制线第0位
A16	RS	指令/数据控制
nWE	nWR	写控制
nOE	nRD	读控制
NE1	nCS	LCD片选控制
PE1	nRESET	LCD复位控制

i80/M68 system 16-bit data bus interface

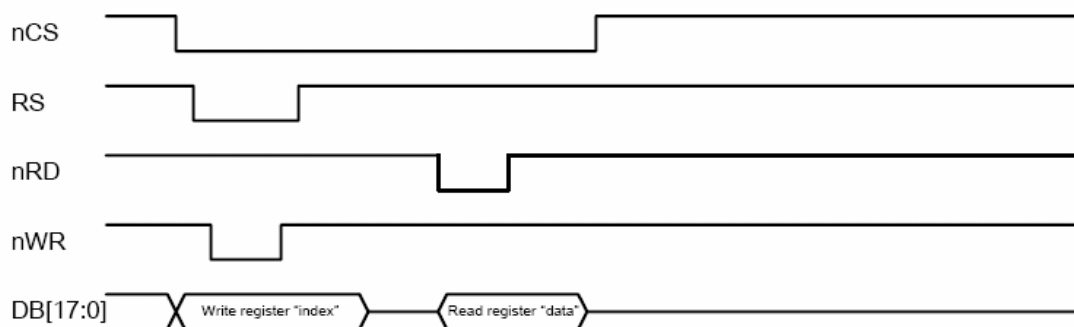


i80 18-/16-bit System Bus Interface Timing

(a) Write to register



(b) Read from register



IL19325的寄存器列表，根据设置这些寄存器，可以灵活进行显示编程。

No.	Registers Name	R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
IR	Index Register	W	0	-	-	-	-	-	-	-	-	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
00h	Driver Code Read	R	1	1	0	0	1	0	0	1	1	0	0	1	0	0	1	0	1
01h	Driver Output Control 1	W	1	0	0	0	0	0	SM	0	SS	0	0	0	0	0	0	0	0
02h	LCD Driving Control	W	1	0	0	0	0	0	0	BC0	EOR	0	0	0	0	0	0	0	0
03h	Entry Mode	W	1	TRI	DFM	0	BGR	0	0	HWM	0	ORG	0	I/D1	I/D0	AM	0	0	0
04h	Resize Control	W	1	0	0	0	0	0	0	RCV1	RCV0	0	0	RCH1	RCH0	0	0	RS21	RS20
07h	Display Control 1	W	1	0	0	PTDE1	PTDE0	0	0	0	BASEE	0	0	GON	DTE	CL	0	D1	D0
08h	Display Control 2	W	1	0	0	0	0	FP3	FP2	FP1	FP0	0	0	0	0	BP3	BP2	BP1	BP0
09h	Display Control 3	W	1	0	0	0	0	0	PTS2	PTS1	PTS0	0	0	PTG1	PTG0	ISC3	ISC2	ISC1	ISC0
0Ah	Display Control 4	W	1	0	0	0	0	0	0	0	0	0	0	0	0	FMARKOE	FM12	FM11	FM10
0Ch	RGB Display Interface Control 1	W	1	0	ENC2	ENC1	ENC0	0	0	0	RM	0	0	DM1	DM0	0	0	RIM1	RIM0
0Dh	Frame Maker Position	W	1	0	0	0	0	0	0	0	FMP8	FMP7	FMP6	FMP5	FMP4	FMP3	FMP2	FMP1	FMP0
0Fh	RGB Display Interface Control 2	W	1	0	0	0	0	0	0	0	0	0	0	0	VSP1	HSP1	0	DPL	EPL
10h	Power Control 1	W	1	0	0	0	SAP	0	BT2	BT1	BT0	APE	AP2	AP1	AP0	0	DSTB	SLP	STB
11h	Power Control 2	W	1	0	0	0	0	0	DC12	DC11	DC10	0	DC02	DC01	DC00	0	VC2	VC1	VC0
12h	Power Control 3	W	1	0	0	0	0	0	0	0	0	VCIRE	0	0	PON	VRH3	VRH2	VRH1	VRH0
13h	Power Control 4	W	1	0	0	0	VDV4	VDV3	VDV2	VDV1	VDV0	0	0	0	0	0	0	0	0
20h	Horizontal GRAM Address Set	W	1	0	0	0	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
21h	Vertical GRAM Address Set	W	1	0	0	0	0	0	0	0	0	AD16	AD15	AD14	AD13	AD12	AD11	AD10	AD9
22h	Write Data to GRAM	W	1	RAM write data (WD17-0) / read data (RD17-0) bits are transferred via different data bus lines according to the selected interfaces.															
29h	Power Control 7	W	1	0	0	0	0	0	0	0	0	0	0	0	VCM5	VCM4	VCM3	VCM2	VCM1
2Bh	Frame Rate and Color Control	W	1	0	0	0	0	0	0	0	0	0	0	0	FRS[3]	FRS[2]	FRS[1]	FRS[0]	
30h	Gamma Control 1	W	1	0	0	0	0	0	KP1[2]	KP1[1]	KP1[0]	0	0	0	0	0	KP0[2]	KP0[1]	KP0[0]
31h	Gamma Control 2	W	1	0	0	0	0	0	KP3[2]	KP3[1]	KP3[0]	0	0	0	0	0	KP2[2]	KP2[1]	KP2[0]
32h	Gamma Control 3	W	1	0	0	0	0	0	KP5[2]	KP5[1]	KP5[0]	0	0	0	0	0	KP4[2]	KP4[1]	KP4[0]
35h	Gamma Control 4	W	1	0	0	0	0	0	RP1[2]	RP1[1]	RP1[0]	0	0	0	0	0	RP0[2]	RP0[1]	RP0[0]
36h	Gamma Control 5	W	1	0	0	0	VRP1[4]	VRP1[3]	VRP1[2]	VRP1[1]	VRP1[0]	0	0	0	0	VRP0[3]	VRP0[2]	VRP0[1]	VRP0[0]
37h	Gamma Control 6	W	1	0	0	0	0	0	KN1[2]	KN1[1]	KN1[0]	0	0	0	0	0	KN0[2]	KN0[1]	KN0[0]
38h	Gamma Control 7	W	1	0	0	0	0	0	KN3[2]	KN3[1]	KN3[0]	0	0	0	0	0	KN2[2]	KN2[1]	KN2[0]
39h	Gamma Control 8	W	1	0	0	0	0	0	KN5[2]	KN5[1]	KN5[0]	0	0	0	0	0	KN4[2]	KN4[1]	KN4[0]
3Ch	Gamma Control 9	W	1	0	0	0	0	0	RN1[2]	RN1[1]	RN1[0]	0	0	0	0	0	RN0[2]	RN0[1]	RN0[0]
3Dh	Gamma Control 10	W	1	0	0	0	VRN1[4]	VRN1[3]	VRN1[2]	VRN1[1]	VRN1[0]	0	0	0	0	VRN0[3]	VRN0[2]	VRN0[1]	VRN0[0]
50h	Horizontal Address Start	W	1	0	0	0	0	0	0	0	0	HSA7	HSA6	HSA5	HSA4	HSA3	HSA2	HSA1	HSA0

No.	Registers Name	R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
	Position																			
51h	Horizontal Address End Position	W	1	0	0	0	0	0	0	0	0	HEA7	HEA6	HEA5	HEA4	HEA3	HEA2	HEA1	HEA0	
52h	Vertical Address Start Position	W	1	0	0	0	0	0	0	0	VSA8	VSA7	VSA6	VSA5	VSA4	VSA3	VSA2	VSA1	VSA0	
53h	Vertical Address End Position	W	1	0	0	0	0	0	0	0	VEA8	VEA7	VEA6	VEA5	VEA4	VEA3	VEA2	VEA1	VEA0	
60h	Driver Output Control 2	W	1	GS	0	NL5	NL4	NL3	NL2	NL1	NL0	0	0	SCN5	SCN4	SCN3	SCN2	SCN1	SCN0	
61h	Base Image Display Control	W	1	0	0	0	0	0	0	0	0	0	0	0	0	0	NDL	VLE	REV	
6Ah	Vertical Scroll Control	W	1	0	0	0	0	0	0	0	VL8	VL7	VL6	VL5	VL4	VL3	VL2	VL1	VL0	
80h	Partial Image 1 Display Position	W	1	0	0	0	0	0	0	0	PTDP08	PTDP07	PTDP06	PTDP05	PTDP04	PTDP03	PTDP02	PTDP01	PTDP00	
81h	Partial Image 1 Area (Start Line)	W	1	0	0	0	0	0	0	0	PTSA08	PTSA07	PTSA06	PTSA05	PTSA04	PTSA03	PTSA02	PTSA01	PTSA00	
82h	Partial Image 1 Area (End Line)	W	1	0	0	0	0	0	0	0	PTEA08	PTEA07	PTEA06	PTEA05	PTEA04	PTEA03	PTEA02	PTEA01	PTEA00	
83h	Partial Image 2 Display Position	W	1	0	0	0	0	0	0	0	PTDP18	PTDP17	PTDP16	PTDP15	PTDP14	PTDP13	PTDP12	PTDP11	PTDP10	
84h	Partial Image 2 Area (Start Line)	W	1	0	0	0	0	0	0	0	PTSA18	PTSA17	PTSA16	PTSA15	PTSA14	PTSA13	PTSA12	PTSA11	PTSA10	
85h	Partial Image 2 Area (End Line)	W	1	0	0	0	0	0	0	0	PTEA18	PTEA17	PTEA16	PTEA15	PTEA14	PTEA13	PTEA12	PTEA11	PTEA10	
90h	Panel Interface Control 1	W	1	0	0	0	0	0	0	0	DIV11	DIV10	0	0	0	RTN13	RTN12	RTN11	RTN10	
92h	Panel Interface Control 2	W	1	0	0	0	0	0	0	0	NOW12	NOW11	0	0	0	0	0	0	0	
95h	Panel Interface Control 4	W	1	0	0	0	0	0	0	0	DIVE1	DIVE0	0	0	RTNE5	RTNE4	RTNE3	RTNE2	RTNE1	
A1h	OTP VCM Programming Control	W	1	0	0	0	0	0	0	0	0	0	0	0	VCM_OTP5	VCM_OTP4	VCM_OTP3	VCM_OTP2	VCM_OTP1	
A2h	OTP VCM Status and Enable	W	1	PGM_CNT1	PGM_CNT0	VCM_D5	VCM_D4	VCM_D3	VCM_D2	VCM_D1	VCM_D0	0	0	0	0	0	0	0	0	VCM_EN
A5h	OTP Programming ID Key	W	1	KEY_15	KEY_14	KEY_13	KEY_12	KEY_11	KEY_10	KEY_9	KEY_8	KEY_7	KEY_6	KEY_5	KEY_4	KEY_3	KEY_2	KEY_1	KEY_0	

2. 灵活的静态存储控制器(FSMC)

奋斗板采用FSMC接口控制显示模块。只有STM32高密度些列芯片上才具有FSMC接口，高密度产品是指闪存容量介于256K字节至512K字节的STM32F101xx和STM32F103xx微控制器。

2.1 FSMC功能描述

FSMC模块能够与同步或异步的存储器和16位的PC存储器卡接口，它的主要作用是：

- 将AHB传输信号转换到适当的外部设备协议
- 满足访问外部设备的时序要求，所有的外部存储器共享控制器输出的地址、数据和控制信号，每个外部设备可以通过一个唯一的片选信号加以区分。FSMC在任一时刻只访问一个外部设备。

FSMC具有下列主要功能：

- 具有静态存储器接口的器件包括：

静态随机存储器 (SRAM)

只读存储器 (ROM)

NOR闪存

PSRAM(4个存储器块)

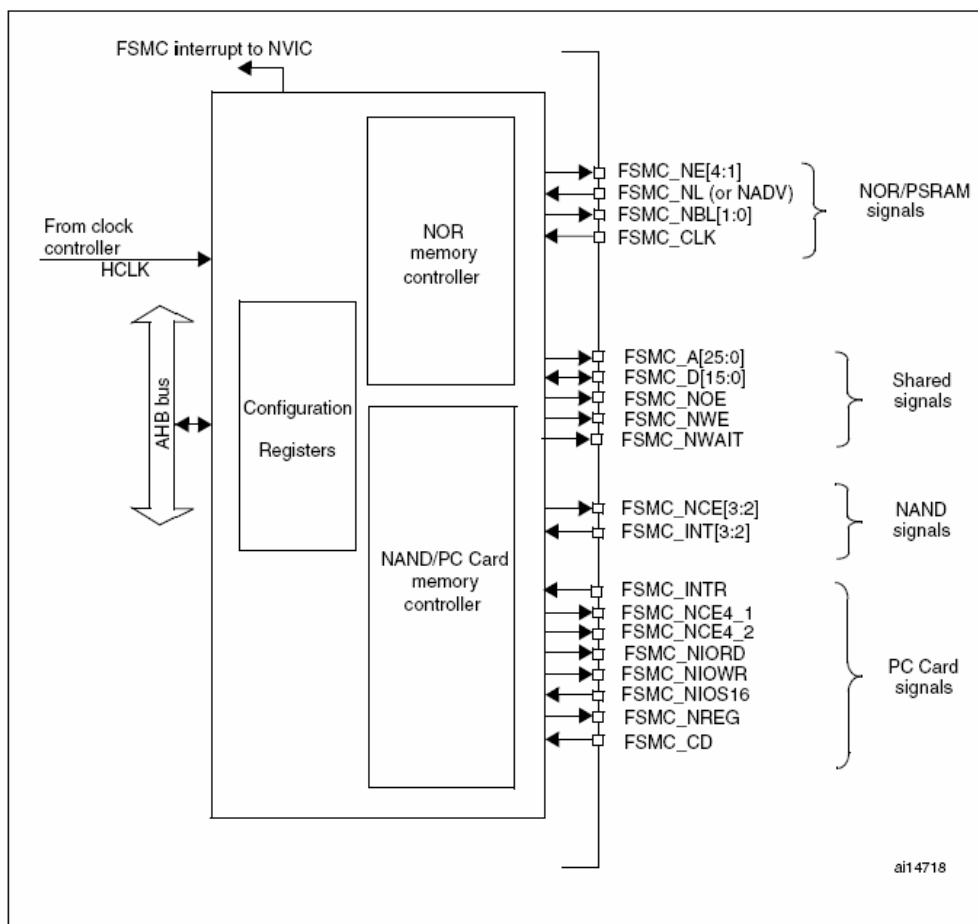
- 两个NAND闪存块，支持硬件ECC并可检测多达8K字节数据
- 16位的PC卡
- 支持对同步器件的成组(Burst)访问模式，如NOR闪存和PSRAM
- 8或16位数据总线
- 每一个存储器块都有独立的片选控制
- 每一个存储器块都可以独立配置
- 时序可编程以支持各种不同的器件：
 - 等待周期可编程(多达15个周期)
 - 总线恢复周期可编程(多达15个周期)
 - 输出使能和写使能延迟可编程(多达15周期)
 - 独立的读写时序和协议，可支持宽范围的存储器和时序
- PSRAM和SRAM器件使用的写使能和字节选择输出
- 将32位的AHB访问请求，转换到连续的16位或8位的，对外部16位或8位器件的访问
- 具有16个字，每个字32位宽的写入FIFO，允许在写入较慢存储器时释放AHB进行其它操作。在开始一次新的FSMC操作前，FIFO要先被清空。通常在系统复位或上电时，应该设置好所有定义外部存储器类型和特性的FSMC寄存器，并保持它们的内容不变；当然，也可以在任何时候改变这些设置。

2.2 框图

FSMC包含四个主要模块：

- AHB接口（包含FSMC配置寄存器）
- NOR闪存和PSRAM控制器
- NAND闪存和PC卡控制器
- 外部设备接口

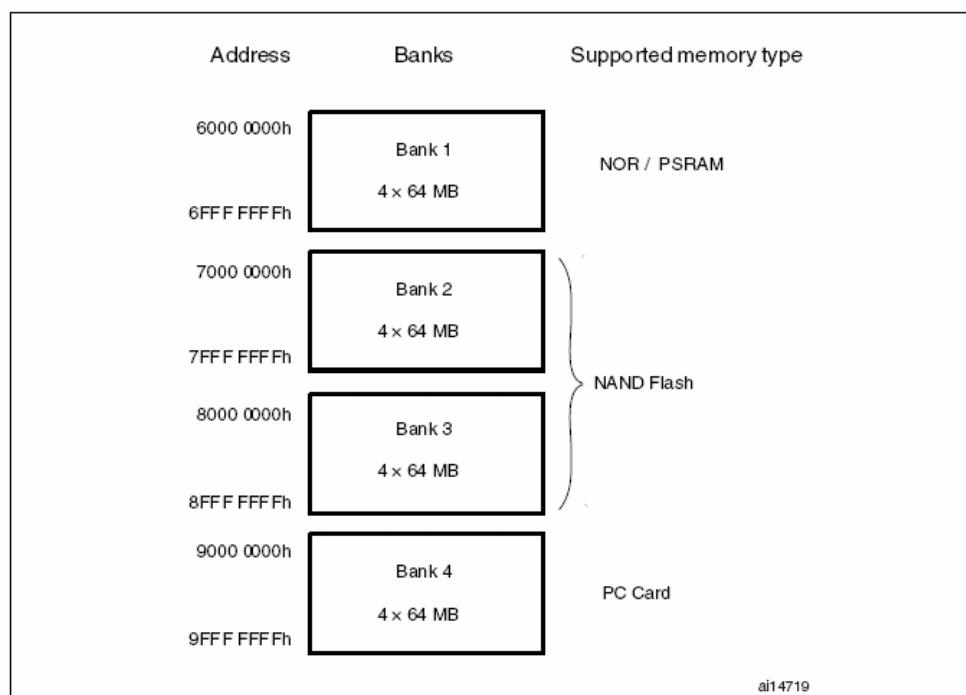
FSMC框图如下：



2.3 外部设备地址映像

从FSMC的角度看，可以把外部存储器划分为固定大小为256M字节的四个存储块。

- 存储块1用于访问最多4个NOR闪存或PSRAM存储设备。这个存储区被划分为4个NOR/PSRAM区并有4个专用的片选。
- 存储块2和3用于访问NAND闪存设备，每个存储块连接一个NAND闪存。
- 存储块4用于访问PC卡设备，每一个存储块上的存储器类型是由用户在配置寄存器中定义的。



3. 应用实例

3.1. 设计要求

在2.4寸TFT屏上显示一副16位色图片，并在图片上透明叠加两个不同显示方向的字符串，该实验学习了2.4寸TFT 16位色显示程序的编制。

3.2 硬件电路连接

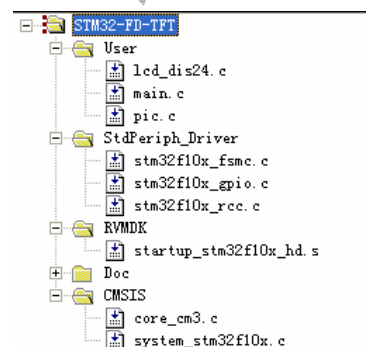
将2.4寸TFT显示模块插到奋斗板上。

3.3 软件程序设计

根据任务要求，程序内容主要包括：

1. 设置系统时钟为72Mhz
2. FSMC接口初始化
3. ILI9325驱动器初始化。
4. 执行显示程序。

整个工程包含4类源文件：



ASM--startup_stm32f10x_hd.s 由于奋斗板采用的是STM32F103大存储器芯片，因此采用STM32标准库自带的大存储器芯片启动代码，这个文件已经配置好了初始状态，以及中断向量表。可以直接在工程里使用，如果你在以后的应用中采用了中存储器或者小存储器STM32芯片，可以将启动代码换为 startup_stm32f10x_md.s 或者 startup_stm32f10x_ld.s。

FWLIB--stm32f10x_gpio.c ST公司的标准库，包含了关于对通用IO口设置的函数。stm32f10x_rcc.c ST公司的标准库，包含了关于对系统时钟及外设设置的函数。

stm32f10x_fsmc.c ST公司的标准库，包含了关于对FSMC接口设置的函数。

CMSYS—是关于CORETEX-M3平台的系统函数及定义

main.c 例程的主函数。

```

//主函数
int main(void)
{
    RCC_Configuration();           //系统时钟配置为72MHz
    FSMC_LCD_Init();              //FSMC总线配置
    LCD_Init();                   //液晶初始化
    while (1)
    {
        LCD_test();
    }
}

```

RCC_Configuration() 是设置系统时钟的函数，将系统时钟通过9倍频为72MHz。保证了系统工作在72MHz时钟下。

```

//设置系统时钟，通过9倍频，将系统时钟设置为72MHz
void RCC_Configuration(void)
{
    SystemInit();
}

```

FSMC_LCD_Init() 是对FSMC接口进行配置。以符合16位 I8080接口时序。用于显示控制。开始部分用于对所用到端口的设置，开启端口复用功能，复用为FSMC接口。

```

//FSMC 接口设置
void FSMC_LCD_Init(void)
{
    FSMC_NORSRAMInitTypeDef FSMC_NORSRAMInitStructure;
    FSMC_NORSRAMTimingInitTypeDef p;
    GPIO_InitTypeDef GPIO_InitStructure;
    //使能FSMC外设时钟
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC |
        RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;           //LCD背光控制
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;           //LCD复位
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    //复用端口为FSMC接口 FSMC-D0--D15
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5 |
        GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | 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);

    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);
    //FSMC NE1 LCD片选
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    //FSMC RS---LCD指令 指令/数据 切换
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    GPIO_SetBits(GPIOD, GPIO_Pin_13);           //LCD背光打开
}

```

后一部分，是对FSMC接口类型进行了配置，

```
//FSMC接口特性配置
p.FSMC_AddressSetupTime = 0x02;
p.FSMC_AddressHoldTime = 0x00;
p.FSMC_DataSetupTime = 0x05;
p.FSMC_BusTurnAroundDuration = 0x00;
p.FSMC_CLKDivision = 0x00;
p.FSMC_DataLatency = 0x00;
p.FSMC_AccessMode = FSMC_AccessMode_B;

FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM1;
FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;
FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_NOR;
FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;
FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode = FSMC_BurstAccessMode_Disable;
FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;
FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState;
FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;
FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;
FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;

FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);
/* Enable FSMC Bank1 SRAM Bank */
FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);
```

LCD_Init是2.4寸显示模块初始化代码。详细定义可以参考ILI9325手册，比对所引用的各寄存器设置。

```
//初始化函数
void LCD_Init(void)
{
    lcd_rst(); //硬件复位

    LCD_WR_CMD(0x00E3, 0x3008); // Set internal timing
    LCD_WR_CMD(0x00E7, 0x0012); // Set internal timing
    LCD_WR_CMD(0x00EF, 0x1231); // Set internal timing
    LCD_WR_CMD(0x0000, 0x0001); // Start Oscillation
    LCD_WR_CMD(0x0001, 0x0100); // set SS and SM bit
    LCD_WR_CMD(0x0002, 0x0700); // set 1 line inversion

    LCD_WR_CMD(0x0003, 0x1018); // set GRAM write direction and BGR=0,
    LCD_WR_CMD(0x0004, 0x0000); // Resize register
    LCD_WR_CMD(0x0008, 0x0202); // set the back porch and front porch
    LCD_WR_CMD(0x0009, 0x0000); // set non-display area refresh cycle
    LCD_WR_CMD(0x000A, 0x0000); // FMARK function
    LCD_WR_CMD(0x000C, 0x0000); // RGB interface setting
    LCD_WR_CMD(0x000D, 0x0000); // Frame marker Position
    LCD_WR_CMD(0x000F, 0x0000); // RGB interface polarity

    //Power On sequence
    LCD_WR_CMD(0x0010, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP, STB
    LCD_WR_CMD(0x0011, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
    LCD_WR_CMD(0x0012, 0x0000); // VREG1OUT voltage
    LCD_WR_CMD(0x0013, 0x0000); // VDV[4:0] for VCOM amplitude
    Delay(200); // Dis-charge capacitor power voltage
    LCD_WR_CMD(0x0010, 0x1690); // SAP, BT[3:0], AP, DSTB, SLP, STB
    LCD_WR_CMD(0x0011, 0x0227); // R11h=0x0221 at VCI=3.3V, DC1[2:0],
    Delay(50); // Delay 50ms
    LCD_WR_CMD(0x0012, 0x001C); // External reference voltage= Vci;
    Delay(50); // Delay 50ms
    LCD_WR_CMD(0x0013, 0x1800); // R13=1200 when R12=009D;VDV[4:0] for
    LCD_WR_CMD(0x0029, 0x001C); // R29=000C when R12=009D;VCM[5:0] for
    LCD_WR_CMD(0x002B, 0x0000); // Frame Rate = 91Hz
    Delay(50); // Delay 50ms
    LCD_WR_CMD(0x0020, 0x0000); // GRAM horizontal Address
    LCD_WR_CMD(0x0021, 0x0000); // GRAM Vertical Address

    // ----- Adjust the Gamma Curve -----//
    LCD_WR_CMD(0x0030, 0x0007);
    LCD_WR_CMD(0x0031, 0x0002);
    LCD_WR_CMD(0x0032, 0x0015);
    LCD_WR_CMD(0x0035, 0x0026);
    LCD_WR_CMD(0x0036, 0x0808);
    LCD_WR_CMD(0x0037, 0x0206);
    LCD_WR_CMD(0x0038, 0x0504);
    LCD_WR_CMD(0x0039, 0x0007);
    LCD_WR_CMD(0x003C, 0x0105);
    LCD_WR_CMD(0x003D, 0x0808);

    // ----- Set GRAM area -----//
    LCD_WR_CMD(0x0050, 0x0000); // Horizontal GRAM Start Address
    LCD_WR_CMD(0x0051, 0x00EF); // Horizontal GRAM End Address
    LCD_WR_CMD(0x0052, 0x0000); // Vertical GRAM Start Address
    LCD_WR_CMD(0x0053, 0x013F); // Vertical GRAM Start Address
    LCD_WR_CMD(0x0060, 0xA700); // Gate Scan Line
    LCD_WR_CMD(0x0061, 0x0001); // NDL,VLE, REV
    LCD_WR_CMD(0x006A, 0x0000); // set scrolling line

    // ----- Partial Display Control -----//
    LCD_WR_CMD(0x0080, 0x0000);
    LCD_WR_CMD(0x0081, 0x0000);
    LCD_WR_CMD(0x0082, 0x0000);
    LCD_WR_CMD(0x0083, 0x0000);
    LCD_WR_CMD(0x0084, 0x0000);
    LCD_WR_CMD(0x0085, 0x0000);

    // ----- Panel Control -----//
    LCD_WR_CMD(0x0090, 0x0010);
    LCD_WR_CMD(0x0092, 0x0000);
    LCD_WR_CMD(0x0093, 0x0003);
    LCD_WR_CMD(0x0095, 0x0110);
    LCD_WR_CMD(0x0097, 0x0000);
    LCD_WR_CMD(0x0098, 0x0000);
    LCD_WR_CMD(0x0007, 0x0133); // 262K color and display ON

    //ini();

    LCD_WR_CMD(32, 0);
    LCD_WR_CMD(33, 0x013F);
    LCD_WR_REG(34); //准备写数据显示区
    for(color1=0;color1<76800;color1++) //用黑色消屏
    {
        LCD_WR_Data(0xffff);
    }
    color1=0;
}
```

根据硬件连接，FSMC接口定义在bank1上，因此，基地址是0x60000000开始的。RS信号接在FSMC A16上，对于16位数据总线，一个实际访问地址是要左移一位的，因此LCD的指令地址和数据地址分别定义如下

```
#define Bank1_LCD_D ((uint32_t)0x60020000) //显示区数据地址
```

```
#define Bank1_LCD_C ((uint32_t)0x60000000) //显示区指令地址
```

ili9325_DrawPicture () 函数，是包含了两种显示方向的图片显示程序。图片采用img2lcd软件取模，取模分辨率控制在320X240之内。

```

/*****
* 名称: void ili9325_DrawPicture(u16 StartX,u16 StartY, u8 Dir, u8 *pic)
* 功能: 在指定坐标范围显示一副图片
* 入口参数: StartX 行起始坐标
*           StartY 列起始坐标
*           Dir 图像显示方向
*           pic 图片头指针
* 出口参数: 无
* 说明: 图片取模格式为水平扫描, 16位颜色模式 取模软件img2LCD
* 调用方法: ili9325_DrawPicture(0,0,0,{u16*}demo);
*****/
void ili9325_DrawPicture(u16 StartX,u16 StartY,u8 Dir,u8 *pic)
{
    u32 i=8, len;
    u16 temp,x,y;

    x=((uint16_t)(pic[2]<<8)+pic[3])-1; //从图像数组里取出图像的长度
    y=((uint16_t)(pic[4]<<8)+pic[5])-1; //从图像数组里取出图像的高度
    if(Dir==0){
        LCD_WR_CMD(0x0003,0x1030); //图像显示方向为左下起 行递增 列递减
        LCD_WR_CMD(0x0050, StartX); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0051, StartX+x); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0052, StartY); //垂直显示区起始地址 0-319
        LCD_WR_CMD(0x0053, StartY+y); //垂直显示区结束地址 0-319

        LCD_WR_CMD(32, StartX); //水平显示区地址
        LCD_WR_CMD(33, StartY); //垂直显示区地址
    }
    else if(Dir==1){
        LCD_WR_CMD(0x0003,0x1018); //图像显示方向为左下起 行递增 列递减
        LCD_WR_CMD(0x0050, StartY); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0051, StartX+y); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0052, 319-(x+StartX)); //垂直显示区起始地址 0-319
        LCD_WR_CMD(0x0053, 319-StartX); //垂直显示区结束地址 0-319

        LCD_WR_CMD(32, StartY); //水平显示区地址
        LCD_WR_CMD(33, 319-StartX); //垂直显示区地址
    }
    LCD_WR_REG(34); //写数据到显示区

    len=2*((uint16_t)(pic[2]<<8)+pic[3])*((uint16_t)(pic[4]<<8)+pic[5]); //计算出图像所占的字节数
    while(i<(len+8)){
        temp=(uint16_t)(pic[i]<<8)+pic[i+1]; //16位总线, 需要一次发送2个字节的数据
        LCD_WR_Data(temp); //将取出的16位像素数据送入显示区
        i=i+2; //取模位置加2, 以为获取下一个像素数据
    }
}

```

lcd_wr_zf () 函数，是包含了两种显示方向字符串显示程序，字符采用ZIM03软件单色取模，字符串取模像素范围320X240之内。

```

/*****
* 名称: lcd_wr_zf(u16 StartX, u16 StartY, u16 X, u16 Y, u16 Color, u8 Dir, u8 *chr)
* 功能: 在指定坐标显示一串字符透明叠加在背景图片上
* 入口参数: StartX 行起始坐标 0-239
*           StartY 列起始坐标 0-319
*           X 长(为8的倍数) 0-320
*           Y 宽 0-240
*           Color 颜色0-65535
*           Dir 图像显示方向
*           chr 字符串指针
* 出口参数: 无
* 说明: 字符取模格式为单色字模, 横向取模, 字节正序 取模软件: ZIM03
* 调用方法: lcd_wr_zf(0,0,100,100,(u16*)demo);
*****/
//++++++LCD写字符串子程序
void lcd_wr_zf(u16 StartX, u16 StartY, u16 X, u16 Y, u16 Color, u8 Dir, u8 *chr)
{
    unsigned int temp=0,num,R_dis_mem=0,Size=0,x=0,y=0,i=0;

    if(Dir==2) LCD_WR_CMD(0x0003,0x1010); //图像显示方向为右下起 行递减 列递增 AM=0 I/D[1:0]=00 <--
    else if(Dir==3) LCD_WR_CMD(0x0003,0x1028); //图像显示方向为右上起 行递减 列递增 AM=1 I/D[1:0]=10 V
    if(Dir==0){
        LCD_WR_CMD(0x0003,0x1030); //图像显示方向为左上起 行递增 列递增 AM=0 I/D[1:0]=11 -->
        LCD_WR_CMD(0x0050, StartX); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0051, StartX+X-1); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0052, StartY); //垂直显示区起始地址 0-319
        LCD_WR_CMD(0x0053, StartY+Y-1); //垂直显示区结束地址 0-319
        LCD_WR_CMD(32, StartX); //水平显示区地址
        LCD_WR_CMD(33, StartY); //垂直显示区地址
        LCD_WR_REG(34); //准备写数据显示区
        Size=X*Y; //字符串或字符占用的像素尺寸
        while(i<Size){
            temp=*chr++; //一个字节代表8个像素, 因此加1代表索引到下8个像素
            for(num=0; num<8; num++){ //数组的每个字节代表了8个像素
                if((temp&0x80)>0){ //对字节的各位进行判断, 为1的用带参数的16位颜色值标示, 写入到像素位置。
                    LCD_WR_Data(Color);
                }
                else{
                    LCD_WR_CMD(32, StartX+x); //水平显示区地址
                    LCD_WR_CMD(33, StartY+y); //垂直显示区地址
                    LCD_WR_REG(34); //准备读数据显示区
                    R_dis_mem=119325_BGR2RGB(LCD_RD_data()); //读取背景色, 为叠加产生透明效果作准备
                    LCD_WR_Data(R_dis_mem); //对字节的各位进行判断, 为0的用当前背景像素16位颜色值标示。
                }
                temp=temp<<1; //字节各位的移出
                x++; //计算像素递增为当前的x和y, 为当前像素读背景颜色做准备
                if(x>=X){x=0; y++;}
                i++;
            }
        }
    }
    else if(Dir==1){
        LCD_WR_CMD(0x0003,0x1018); //图像显示方向为左下起 行递增 列递减 AM=1 I/D[1:0]=01 A
        LCD_WR_CMD(0x0050, StartY); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0051, StartX+Y-1); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0052, 319-(StartX+X-1)); //垂直显示区起始地址 0-319
        LCD_WR_CMD(0x0053, 319-StartX); //垂直显示区结束地址 0-319
        LCD_WR_CMD(32, StartY); //水平显示区地址
        LCD_WR_CMD(33, 319-StartX); //垂直显示区地址
        LCD_WR_REG(34); //准备写数据显示区
        Size=X*Y; //字符串或字符占用的像素尺寸
        while(i<Size){
            temp=*chr++; //一个字节代表8个像素, 因此加1代表索引到下8个像素
            for(num=0; num<8; num++){ //数组的每个字节代表了8个像素
                if((temp&0x80)>0){ //对字节的各位进行判断, 为1的用带参数的16位颜色值标示, 写入到像素位置。
                    LCD_WR_Data(Color);
                }
                else{
                    LCD_WR_CMD(32, StartY+y); //水平显示区地址
                    LCD_WR_CMD(33, 319-(StartX+x)); //垂直显示区地址
                    LCD_WR_REG(34); //准备读数据显示区
                    R_dis_mem=119325_BGR2RGB(LCD_RD_data()); //读取背景色, 为叠加产生透明效果作准备
                    LCD_WR_Data(R_dis_mem); //对字节的各位进行判断, 为0的用当前背景像素16位颜色值标示。
                }
                temp=temp<<1; //字节各位的移出
                x++; //计算像素递增为当前的x和y, 为当前像素读背景颜色做准备
                if(x>=X){x=0; y++;}
                i++;
            }
        }
    }
}

```

LCD_test () 函数演示了图片显示及字符串显示。

```
//演示程序
void LCD_test(void)
{
    unsigned char *p;

    ili9325_DrawPicture(0,0,0,a1);          //在某一指定位置显示图片， 图片的尺寸要在240X320范围内。
    Delay(0xafffff);
    ili9325_DrawPicture(0,0,1,a2);          //在某一指定位置显示图片， 图片的尺寸要在320X240范围内。

    Delay(0xafffff);

    lcd_wr_zf(0,0,280,32,color1,1,&zf3[0]); //显示字符串

    //显示5位的数字， 数值按COLOR1值周期变化
    p=num_pub((color1/10000));
    lcd_wr_zf(50,30,24,32,color1,0,p);

    p=num_pub((color1%10000)/1000);
    lcd_wr_zf(74,30,24,32,color1,0,p);

    p=num_pub(((color1%10000)%1000)/100);
    lcd_wr_zf(98,30,24,32,color1,0,p);


    p=num_pub((((color1%10000)%1000)%100)/10);
    lcd_wr_zf(122,30,24,32,color1,0,p);

    p=num_pub((color1%10));
    lcd_wr_zf(146,30,24,32,color1,0,p);

    //颜色渐变
    color1++;
    if(color1>=65536) color1=0;
    Delay(0xafffff);
}
```

Pic.c里包含了图像取模数据。可以更换为自己的取模数据。

4 运行过程

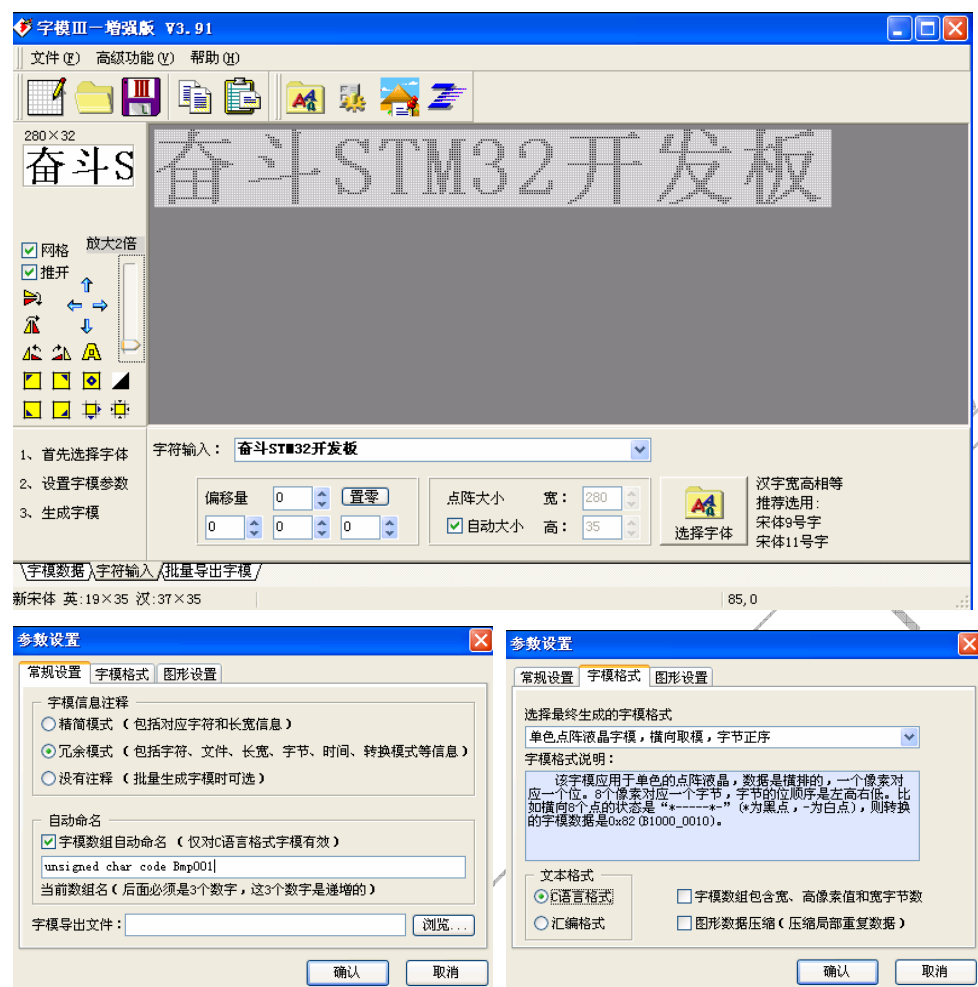
按  编译工程， 完成后会提示如下。

```
Build target 'STM32-FD-IFT'
linking...
Program Size: Code=2844 RO-data=308992 RW-data=4 ZI-data=516
FromELF: creating hex file...
".\Obj\STM32_FD_USART.axf" - 0 Error(s), 0 Warning(s).
```

1. 通过JLINK V8或者串口将代码写入板子，具体的烧写步骤，参考奋斗板文档目录下的《奋斗版STM32开发板JTAG下载步骤》或者《奋斗版STM32开发板串口下载步骤》。
 2. 在2.4寸TFT显示模块会周期性显示图片和字符串。字符串是透明叠加在图片上的。
- Img2lcd取模演示。



ZIM03软件设置界面，需要注意的是，取模的边长要是8的整数倍数。



该例程实验完成。

用户可根据例程编写自己的应用例程。