

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3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3寸TFT 16位色显示程序的编制。

预先需要掌握的知识

1. 3寸TFT显示模块。

3寸TFT显示器：（关于3寸TFT的详细资料请参考光盘奋斗板文档目录下\奋斗开发板各种配件的硬件文档\奋斗板配3寸显示模块文档\下的SPFD5420A手册.pdf和3寸屏（240X400）规格书.pdf），3显示模块采用的是基于LGDP5420驱动的3寸 TFT显示器（400X240），规格如下：

Item		Specifications	Unit
External shape dimensions		*45.04 (W) x77(H) x 3.4MAX)	mm
Main LCD	Pixel Pitch	54x162	um
	Active Area	38.88(W) x 64.8(H)	mm
Weight		TBD	g

Item	Symbol	Min.	Max.	Unit	Remarks
Power Supply for Logic	VCC-GND	-0.3	+4.6	V	
Power Supply for LED backlight	LEDA-LEDK	-0.5	+5	V	
Input voltage	VIN	-0.5	VCC+0.5	V	

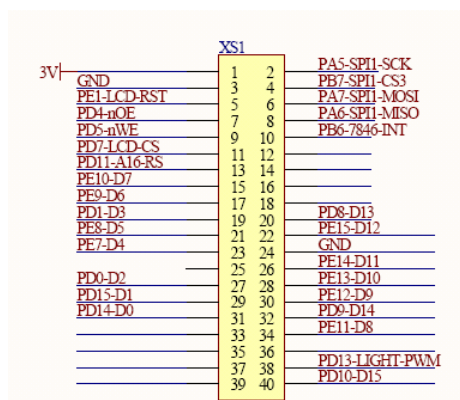
7.2.2 Environment

Item	Specification	Remarks
Storage temperature	Max. +70 °C, Min. -30°C	Note 1: Non-condensing
Operating temperature	Max. +60 °C, Min. -20°C	Note 1: Non-condensing

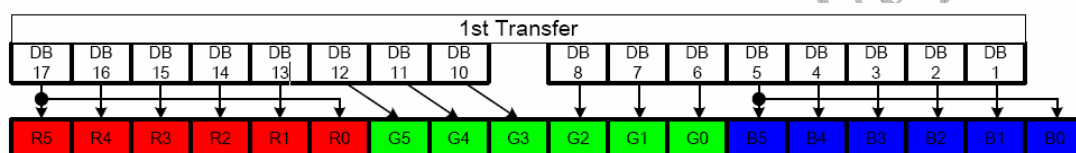
引脚定义

Item	Terminal	Functions
1	LCM_ID	NC
2	XL	Touch Panel Onput PIN
3	YU	Touch Panel Onput PIN
4	XR	Touch Panel Onput PIN
5	YD	Touch Panel Onput PIN
6	GND	Ground
7	IOVCC	Power input(1.8V/2.8V)
8	VCC/VCI	Power input(2.8V)
9	FARMK	NC
10	CS	Chip Select Input PIN
11	RS	Register Select Input PIN
12	WR	Write Data Select Input PIN
13	RD	Read Data Select Input PIN
14-29	DB0-DB15	Input/output data pin 16 bit: DB0---DB15 8 bit: DB8—DB15 //高 8 位
30	RESET	Chip reset Select Input PIN
31	GND	Ground
32	A	B/L Power input PIN anode
33-36	K1-K4	B/L Power input PIN negative
37	NC	NC

3TFT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



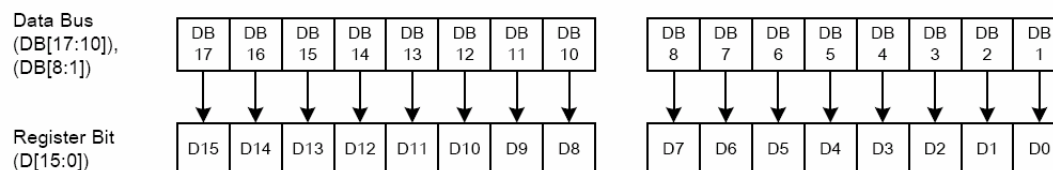
40芯里包含了16位数据线，读写线，命令/数据控制线，片选线，LCD硬件复位线，背光控制线以及触摸控制线。奋斗板V3和MINI就是通过这个接口来控制显示。奋斗板MINI和V3都是选用了具有16位FSMC接口STM32F103VET6作为MCU，FSMC接口也可以称为16位并行接口，时序同I8080接口。按照显示屏驱动电路LGDP5420的手册，为了达到色彩与显示效率的平衡，奋斗板采用了16位 64K色接口模式。



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 $400 \times 240 \times 2 = 192000$ 字节。FSMC总线和TFT数据线的连接关系如下

STM32 FSMC	LGDP5420A	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

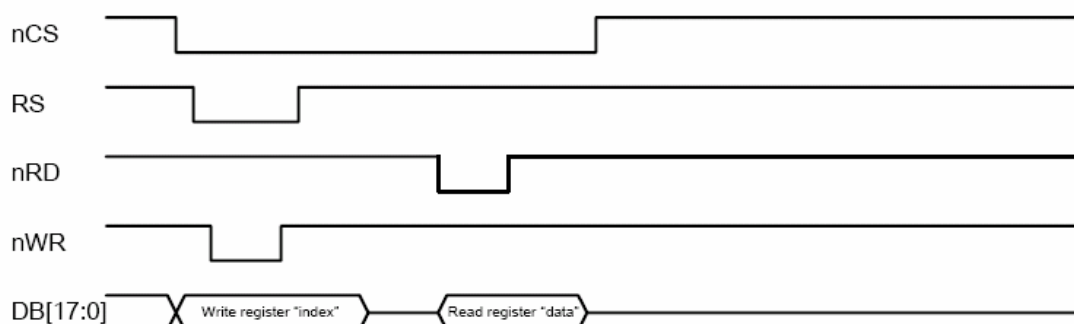


i80 18-/16-bit System Bus Interface Timing

(a) Write to register



(b) Read from register



LGDP5420A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。

Register No	Register	Upper 8-bit								Lower 8-bit							
		CB15	CB14	CB13	CB12	CB11	CB10	CB9	CB8	CB7	CB6	CB5	CB4	CB3	CB2	CB1	CB0
000h	ID Read	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0
001h	Driver Output Control	0	0	0	0	0	SM (0)	0	SS (0)	0	0	0	0	0	0	0	0
002h	LCD Drive Waveform Control	0	0	0	0	0	0	0	B/C (0)	0	0	0	0	0	0	NW1 (0)	NW0 (0)
003h	Entry Mode	TRI REG (0)	DFM (0)	0	BGR (0)	0	0	HWM (0)	0	ORG (0)	0	I/D1 (1)	I/D0 (1)	AM (0)	0	EPF1 (0)	EPF0 (0)
004h-006h	Setting disabled																
007h	Display Control (1)	0	0	PTDE1 (0)	PTDE0 (0)	0	0	0	BASEE (0)	0	VON (0)	GON (0)	DTE (0)	0	0	D1 (0)	D0 (0)
008h	Display Control (2)	0	0	0	0	FP3 (1)	FP2 (0)	FP1 (0)	FP0 (0)	0	0	0	0	BP3 (1)	BP2 (0)	BP1 (0)	BP0 (0)
009h	Low Power Control (1)	0	0	0	0	PTV (0)	PTS2 (0)	PTS1 (0)	PTS0 (0)	0	0	PTG1 (0)	PTG0 (0)	ISC3 (0)	ISC2 (0)	ISC1 (0)	ISC0 (0)
00Ah	Setting Disabled																
00Bh	Low Power Control (2)	0	0	0	0	0	0	0	0	0	0	0	VEM0 (0)	0	0	0	COL (0)
00Ch	External Display Control (1)	0	ENC2 (0)	ENC1 (0)	ENC0 (0)	0	0	0	RM (0)	0	0	DM1 (0)	DM0 (0)	0	0	RIM1 (0)	RIM0 (0)
00Dh-00Eh	Setting Disabled																
00Fh	External Display Control (2)	0	0	0	0	0	0	0	0	0	0	0	VSPL (0)	HSPL (0)	0	EPL (0)	DPL (0)
010h	Panel interface Control 1	0	0	0	0	0	0	DIV11 (0)	DIV10 (0)	0	0	0	RTN14 (1)	RTN13 (0)	RTN12 (1)	RTN11 (1)	RTN10 (1)
011h	Panel interface Control 2	0	0	0	0	0	NOW12 (0)	NOW11 (0)	NOW10 (0)	0	0	0	0	0	SDT12 (0)	SDT11 (0)	SDT10 (0)
012h	Panel interface Control 3	0	0	0	0	0	0	VEQW11 (0)	VEQW10 (0)	0	0	0	0	0	0	0	0
013-01Fh	Setting Disabled																
020h	Panel Interface Control 4	0	0	0	0	0	0	DIVE1 (0)	DIVE0 (0)	0	RTNE6 (0)	RTNE5 (0)	RTNE4 (1)	RTNE3 (1)	RTNE2 (1)	RTNE1 (1)	RTNE0 (0)
021h	Panel Interface Control 5	0	0	0	0	NOWE3 (0)	NOWE2 (0)	NOWE1 (0)	NOWE0 (0)	0	0	0	0	SDTE3 (0)	SDTE2 (0)	SDTE1 (0)	SDTE0 (0)
022h	Panel Interface Control 6	0	0	0	0	0	VEQWE2 (0)	VEQWE1 (0)	VEQWE0 (0)	0	0	0	0	0	0	0	0
023h-08Fh	Setting Disabled																
090h	Frame Marker Control	FMKM (0)	FMI2 (0)	FMI1 (0)	FMI0 (0)	0	0	0	FMP8 (0)	FMP7 (0)	FMP6 (0)	FMP5 (0)	FMP4 (0)	FMP3 (0)	FMP2 (0)	FMP1 (0)	FMP0 (0)
091h-0FFh	Setting disabled																
100h	Power Control (1)	0	0	0	SAP (0)	0	BT2 (0)	BT1 (0)	BT0 (0)	APE (0)	0	AP1 (0)	AP0 (0)	0	DSTB (0)	SLP (0)	0
101h	Power Control (2)	0	0	0	0	0	DC12 (0)	DC11 (0)	DC10 (0)	0	DC02 (0)	DC01 (0)	DC00 (0)	0	VC2 (0)	VC1 (0)	VC0 (0)
102h	Power Control (3)	0	0	0	0	0	0	0	VCMR0 (0)	VREG1R (0)	0	PSON (0)	PON (0)	VRH3 (0)	VRH2 (0)	VRH1 (0)	VRH0 (0)
103h	Power Control (4)	0	0	VCOMG (0)	VDV4 (0)	VDV3 (0)	VDV2 (0)	VDV1 (0)	VDV0 (0)	0	0	0	0	0	0	0	0
104h-106h	Setting disabled																
107h	Power Control (5)	0	0	0	0	0	0	0	0	0	0	0	DCM0 (0)	DCT3 (0)	DCT2 (0)	DCT1 (0)	DCT0 (0)
108-10Fh	Setting disabled																
110h	Power Control(6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PSE (0)
111-1ffh	Setting disabled																
200h	GRAM address Set Horizontal Address	0	0	0	0	0	0	0	0	AD7 (0)	AD6 (0)	AD5 (0)	AD4 (0)	AD3 (0)	AD2 (0)	AD1 (0)	AD0 (0)
201h	GRAM address Set Vertical Address	0	0	0	0	0	0	0	0	AD16 (0)	AD15 (0)	AD14 (0)	AD13 (0)	AD12 (0)	AD11 (0)	AD10 (0)	AD9 (0)
202h	Write Data to GRAM Read Data from GRAM	Data format is varied according to "interface".															
203-20Fh	Setting disabled																
210h	Window Horizontal RAM Address Start	0	0	0	0	0	0	0	0	HSA7 (0)	HSA6 (0)	HSA5 (0)	HSA4 (0)	HSA3 (0)	HSA2 (0)	HSA1 (0)	HSA0 (0)
211h	Window Horizontal RAM Address End	0	0	0	0	0	0	0	0	HEA7 (1)	HEA6 (1)	HEA5 (1)	HEA4 (0)	HEA3 (1)	HEA2 (1)	HEA1 (1)	HEA0 (1)
212h	Window Vertical RAM Address Start	0	0	0	0	0	0	0	0	VSA8 (0)	VSA7 (0)	VSA6 (0)	VSA5 (0)	VSA4 (0)	VSA3 (0)	VSA2 (0)	VSA1 (0)
213h	Window Vertical RAM Address End	0	0	0	0	0	0	0	0	VEA8 (1)	VEA7 (0)	VEA6 (0)	VEA5 (1)	VEA4 (1)	VEA3 (1)	VEA2 (1)	VEA1 (1)
214-27Fh	Setting Disabled																
280h	NVM Write/Read	0	0	0	0	0	0	0	0	0	0	0	0	UID3 (0)	UID2 (0)	UID1 (0)	UID0 (0)
281h	VCom high voltage 1	0	0	0	0	0	0	0	0	0	0	0	0	VCM14 (0)	VCM13 (0)	VCM12 (0)	VCM11 (0)
282h	VCom high voltage 2	0	0	0	0	0	0	0	0	VCMSEL (0)	0	0	0	VCM24 (0)	VCM23 (0)	VCM22 (0)	VCM21 (0)
283-2FFh	Setting disabled																
300h	γ Control (1)	0	0	0	V1RP4	V1RP3	V1RP2	V1RP1	V1RP0	0	0	0	0	V6RN4	V6RN3	V6RN2	V6RN1

301h	γ Control (2)	0	0	V2RP5	V2RP4	V2RP3	V2RP2	V2RP1	V2RP0	0	0	V5RN5	V5RN4	V5RN3	V5RN2	V5RN1	V5RN0
302h	γ Control (3)	0	0	V3RP5	V3RP4	V3RP3	V3RP2	V3RP1	V3RP0	0	0	V4RN5	V4RN4	V4RN3	V4RN2	V4RN1	V4RN0
303h	γ Control (4)	0	0	V4RP5	V4RP4	V4RP3	V4RP2	V4RP1	V4RP0	0	0	V3RN5	V3RN4	V3RN3	V3RN2	V3RN1	V3RN0
304h	γ Control (5)	0	0	V5RP5	V5RP4	V5RP3	V5RP2	V5RP1	V5RP0	0	0	V2RN5	V2RN4	V2RN3	V2RN2	V2RN1	V2RN0
305h	γ Control (6)	0	0	0	V6RP4	V6RP3	V6RP2	V6RP1	V6RP0	0	0	0	V1RN4	V1RN3	V1RN2	V1RN1	V1RN0
306h	γ Control (7)	0	0	0	V7RP4	V7RP3	V7RP2	V7RP1	V7RP0	0	0	0	V8RN4	V8RN3	V8RN2	V8RN1	V8RN0
307h	γ Control (8)	0	0	0	V8RP4	V8RP3	V8RP2	V8RP1	V8RP0	0	0	0	V7RN4	V7RN3	V7RN2	V7RN1	V7RN0
308h	γ Control (9)	0	0	0	0	V9RP3	V9RP2	V9RP1	V9RP0	0	0	0	0	V16RN3	V16RN2	V16RN1	V16RN0
309h	γ Control (10)	0	0	0	0	V10RP3	V10RP2	V10RP1	V10RP0	0	0	0	0	V15RN3	V15RN2	V15RN1	V15RN0
30Ah	γ Control (11)	0	0	0	0	V11RP3	V11RP2	V11RP1	V11RP0	0	0	0	0	V14RN3	V14RN2	V14RN1	V14RN0
30Bh	γ Control (12)	0	0	0	0	V12RP3	V12RP2	V12RP1	V12RP0	0	0	0	0	V13RN3	V13RN2	V13RN1	V13RN0
30Ch	γ Control (13)	0	0	0	0	V13RP3	V13RP2	V13RP1	V13RP0	0	0	0	0	V12RN3	V12RN2	V12RN1	V12RN0
30Dh	γ Control (14)	0	0	0	0	V14RP3	V14RP2	V14RP1	V14RP0	0	0	0	0	V11RN3	V11RN2	V11RN1	V11RN0
30Eh	γ Control (15)	0	0	0	0	V15RP3	V15RP2	V15RP1	V15RP0	0	0	0	0	V10RN3	V10RN2	V10RN1	V10RN0
30Fh	γ Control (16)	0	0	0	0	V16RP3	V16RP2	V16RP1	V16RP0	0	0	0	0	V9RN3	V9RN2	V9RN1	V9RN0
310-3FFh	Setting disabled																
400h	Size of base image	GS (0)	0	NL5 (0)	NL4 (0)	NL3 (0)	NL2 (0)	NL1 (0)	NL0 (0)	0	0	SCN5 (0)	SCN4 (0)	SCN3 (0)	SCN2 (0)	SCN1 (0)	SCN0 (0)
401h	Base image display control	0	0	0	0	0	0	0	0	0	0	0	0	0	NDL (0)	VLE (0)	REV (0)
402-403h	Setting disabled																
404h	Vertical Scroll Control	0	0	0	0	0	0	0	VL8 (0)	VL7 (0)	VL6 (0)	VL5 (0)	VL4 (0)	VL3 (0)	VL2 (0)	VL1 (0)	VL0 (0)
405-4FFh	Setting disabled																
500h	Display Position - Partial Display 1	0	0	0	0	0	0	0	PTDP08 (0)	PTDP07 (0)	PTDP06 (0)	PTDP05 (0)	PTDP04 (0)	PTDP03 (0)	PTDP02 (0)	PTDP01 (0)	PTDP00 (0)
501h	RAM Address Start - Partial Display 1	0	0	0	0	0	0	0	PTSA08 (0)	PTSA07 (0)	PTSA06 (0)	PTSA05 (0)	PTSA04 (0)	PTSA03 (0)	PTSA02 (0)	PTSA01 (0)	PTSA00 (0)
502h	RAM Address End - Partial Display 1	0	0	0	0	0	0	0	PTEA08 (0)	PTEA07 (0)	PTEA06 (0)	PTEA05 (0)	PTEA04 (0)	PTEA03 (0)	PTEA02 (0)	PTEA01 (0)	PTEA00 (0)
503h	Display Position - Partial Display 2	0	0	0	0	0	0	0	PTDP18 (0)	PTDP17 (0)	PTDP16 (0)	PTDP15 (0)	PTDP14 (0)	PTDP13 (0)	PTDP12 (0)	PTDP11 (0)	PTDP10 (0)
504h	RAM Address Start - Partial Display 2	0	0	0	0	0	0	0	PTSA18 (0)	PTSA17 (0)	PTSA16 (0)	PTSA15 (0)	PTSA14 (0)	PTSA13 (0)	PTSA12 (0)	PTSA11 (0)	PTSA10 (0)
505h	RAM Address End - Partial Display 2	0	0	0	0	0	0	0	PTEA18 (0)	PTEA17 (0)	PTEA16 (0)	PTEA15 (0)	PTEA14 (0)	PTEA13 (0)	PTEA12 (0)	PTEA11 (0)	PTEA10 (0)
506-605h	Setting Disabled																
606h	i80-IF Endian Control	0	0	0	0	0	0	0	TCREV1 (0)	0	0	0	0	0	0	0	TCREV0 (0)
607-6EFh	Setting disabled																
6F0h	NVM access control	0	0	0	0	0	0	0	TE (0)	0	0	EOP1 (0)	EOP0 (0)	0	0	EAD1 (0)	EAD0 (0)
6F1-FFFh	Setting disabled																

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

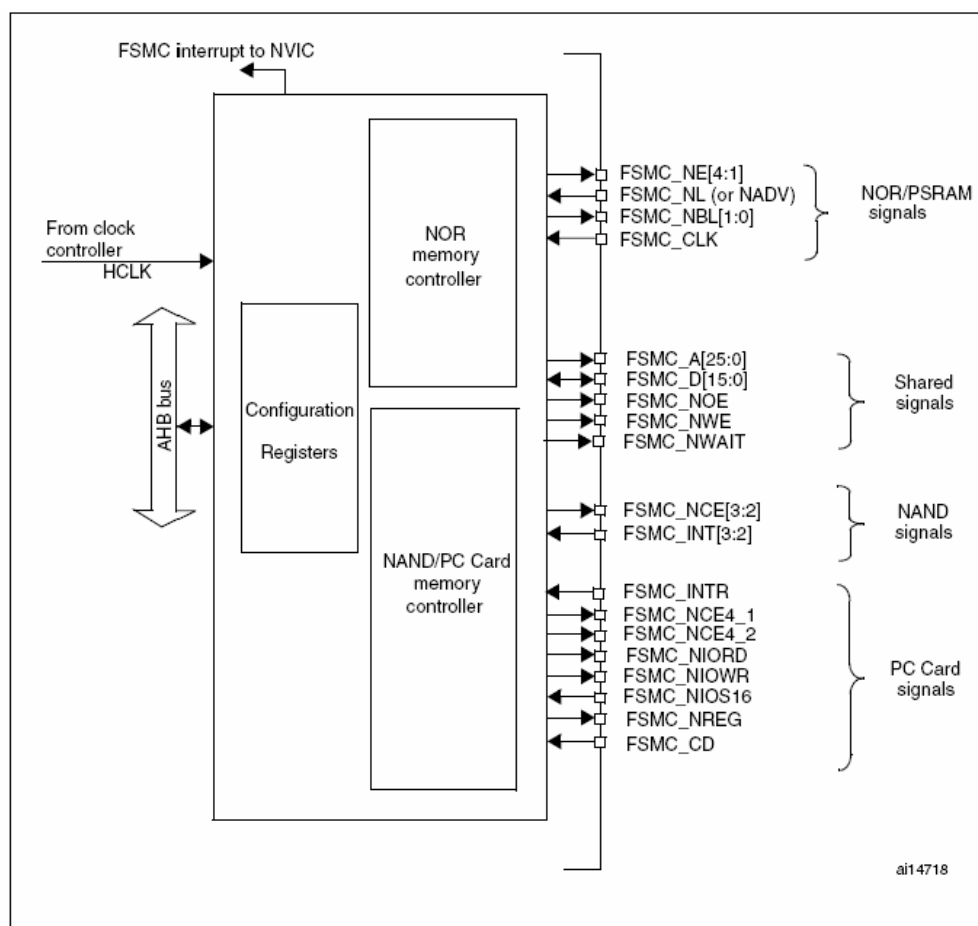
- 16位的PC卡
- 支持对同步器件的成组 (Burst) 访问模式，如NOR闪存和PSRAM
- 8或16位数据总线
- 每一个存储器块都有独立的片选控制
- 每一个存储器块都可以独立配置
- 时序可编程以支持各种不同的器件：
 - 等待周期可编程 (多达15个周期)
 - 总线恢复周期可编程 (多达15个周期)
 - 输出使能和写使能延迟可编程 (多达15周期)
 - 独立的读写时序和协议，可支持宽范围的存储器和时序
- PSRAM和SRAM器件使用的写使能和字节选择输出
- 将32位的AHB访问请求，转换到连续的16位或8位的，对外部16位或8位器件的访问
- 具有16个字，每个字32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

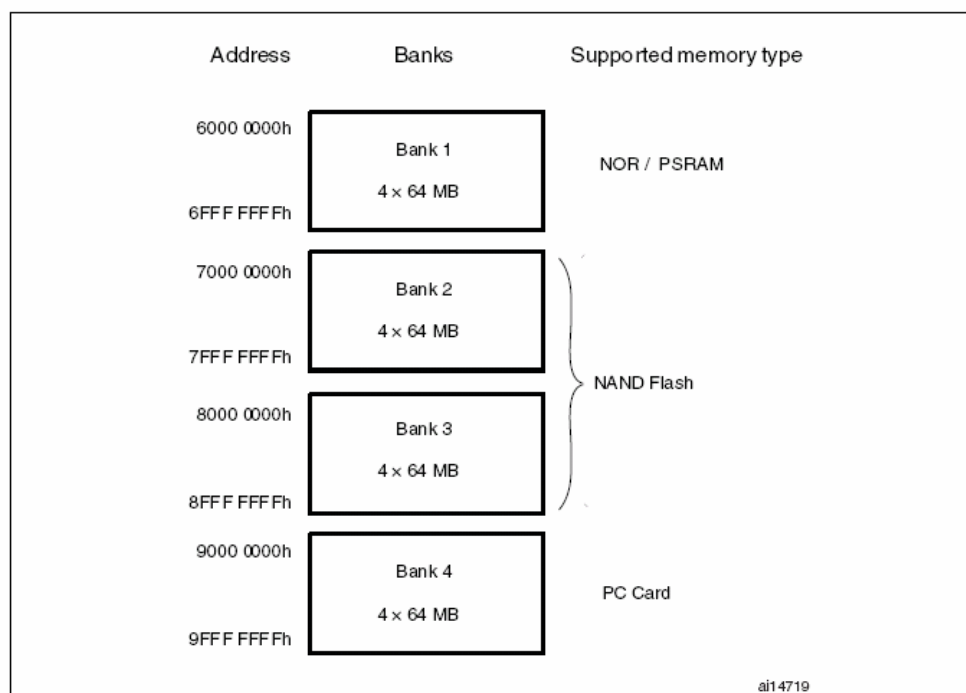
FSMC框图如下：



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3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3寸TFT 16位色显示程序的编制。

3.2 硬件电路连接

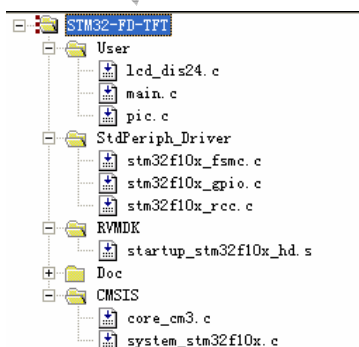
将3寸TFT显示模块插到奋斗板上。

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. LGDP5420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LGDP5420A手册，比对所引用的各寄存器设置。

```

//初始化函数
void LCD_Init(void)
{
    unsigned int i;
    GPIO_ResetBits(GPIOE, GPIO_Pin_1);    //硬件复位
    Delay(0x1AFFF);
    GPIO_SetBits(GPIOE, GPIO_Pin_1);
    Delay(0x1AFFF);

    LCD_WR_CMD(0x0000, 0x0000);
    LCD_WR_CMD(0x0001, 0x0100);
    LCD_WR_CMD(0x0002, 0x0100);

    /* R003H 扫描方向寄存器 决定了显示方向 */
    LCD_WR_CMD(0x0003, 0x1030);

    LCD_WR_CMD(0x0008, 0x0808);
    LCD_WR_CMD(0x0009, 0x0001);
    LCD_WR_CMD(0x000B, 0x0010);
    LCD_WR_CMD(0x000C, 0x0000);
    LCD_WR_CMD(0x000F, 0x0000);
    LCD_WR_CMD(0x0007, 0x0001);
    LCD_WR_CMD(0x0010, 0x0013);
    LCD_WR_CMD(0x0011, 0x0501);
    LCD_WR_CMD(0x0012, 0x0300);
    LCD_WR_CMD(0x0020, 0x021E);
    LCD_WR_CMD(0x0021, 0x0202);
    LCD_WR_CMD(0x0090, 0x8000);
    LCD_WR_CMD(0x0100, 0x17B0);
    LCD_WR_CMD(0x0101, 0x0147);
    LCD_WR_CMD(0x0102, 0x0135);
    LCD_WR_CMD(0x0103, 0x0700);
    LCD_WR_CMD(0x0107, 0x0000);
    LCD_WR_CMD(0x0110, 0x0001);
    LCD_WR_CMD(0x0210, 0x0000);
    LCD_WR_CMD(0x0211, 0x00EF);
    LCD_WR_CMD(0x0212, 0x0000);
    LCD_WR_CMD(0x0213, 0x018F);
    LCD_WR_CMD(0x0280, 0x0000);
    //LCD_WR_CMD(0x0281, 0x0004);
    //LCD_WR_CMD(0x0282, 0x0000);
    /* 改变这2个寄存器可以调整显示的对比度 */
    LCD_WR_CMD(0x0281, 0x000F);
    LCD_WR_CMD(0x0282, 0x0000);

    LCD_WR_CMD(0x0300, 0x0101);
    LCD_WR_CMD(0x0301, 0x0B2C);
    LCD_WR_CMD(0x0302, 0x1030);
    LCD_WR_CMD(0x0303, 0x3010);
    LCD_WR_CMD(0x0304, 0x2C0B);
    LCD_WR_CMD(0x0305, 0x0101);
    LCD_WR_CMD(0x0306, 0x0807);
    LCD_WR_CMD(0x0307, 0x0708);
    LCD_WR_CMD(0x0308, 0x0107);
    LCD_WR_CMD(0x0309, 0x0105);
    LCD_WR_CMD(0x030A, 0x0F04);
    LCD_WR_CMD(0x030B, 0x0F00);
    LCD_WR_CMD(0x030C, 0x000F);
    LCD_WR_CMD(0x030D, 0x040F);
    LCD_WR_CMD(0x030E, 0x0300);
    LCD_WR_CMD(0x030F, 0x0701);
    LCD_WR_CMD(0x0400, 0x3500);
    LCD_WR_CMD(0x0401, 0x0001);
    LCD_WR_CMD(0x0404, 0x0000);
    LCD_WR_CMD(0x0500, 0x0000);
    LCD_WR_CMD(0x0501, 0x0000);
    LCD_WR_CMD(0x0502, 0x0000);
    LCD_WR_CMD(0x0503, 0x0000);
    LCD_WR_CMD(0x0504, 0x0000);
    LCD_WR_CMD(0x0505, 0x0000);
    LCD_WR_CMD(0x0600, 0x0000);
    LCD_WR_CMD(0x0606, 0x0000);
    LCD_WR_CMD(0x06F0, 0x0000);
    LCD_WR_CMD(0x07F0, 0x5420);
    LCD_WR_CMD(0x07DE, 0x0000);
    LCD_WR_CMD(0x07F2, 0x00DF);
    LCD_WR_CMD(0x07F3, 0x0810);
    LCD_WR_CMD(0x07F4, 0x0077);
    LCD_WR_CMD(0x07F5, 0x0021);
    LCD_WR_CMD(0x07F0, 0x0000);
    LCD_WR_CMD(0x0007, 0x0173);

    /* 设置显示窗口 WINDOWS */
    LCD_WR_CMD(0x0210, 0);          /* 水平起始地址 */
    LCD_WR_CMD(0x0211, 239);        /* 水平结束坐标 */
    LCD_WR_CMD(0x0212, 0);          /* 垂直起始地址 */
    LCD_WR_CMD(0x0213, 399);        /* 垂直结束地址 */

    LCD_WR_CMD(0x0200, 0);
    LCD_WR_CMD(0x0201, 0);
    /* IO uint16_t */ (Bank1_LCD_C) = 0x0202;
    for(i=0; i<96000; i++)
    {
        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400X240之内。

```

/*****
* 名称: void ili9325_DrawPicture(u16 StartX,u16 StartY, u8 Dir, u8 *pic)
* 功能: 在指定座标范围显示一副图片
* 入口参数: StartX 行起始座标
*           StartY 列起始座标
*           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(0x0210, StartX); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0211, StartX+x); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0212, StartY); //垂直显示区起始地址 0-399
        LCD_WR_CMD(0x0213, StartY+y); //垂直显示区结束地址 0-399

        LCD_WR_CMD(0x0200, StartX); //水平显示区地址
        LCD_WR_CMD(0x0201, 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(0x0210, StartY); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0211, StartX+y); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0212, 399-(x+StartX)); //垂直显示区起始地址 0-399
        LCD_WR_CMD(0x0213, 399-StartX); //垂直显示区结束地址 0-399

        LCD_WR_CMD(0x0200, StartY); //水平显示区地址
        LCD_WR_CMD(0x0201, 399-StartX); //垂直显示区地址
    }
    LCD_WR_REG(0x0202); //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)){ //从图像数组的第9位开始递增
        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_wrfz () 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400X240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 0-399
*           X 长(为8的倍数) 0-400
*           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: ZIMO3
* 调用方法: lcd_wr_zf(0,0,100,100,(u16*)demo);
*****/
//+++++LCD写字程序
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(0x0210, StartX); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0211, StartX+X-1); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0212, StartY); //垂直显示区起始地址 0-399
        LCD_WR_CMD(0x0213, StartY+Y-1); //垂直显示区结束地址 0-399
        LCD_WR_CMD(0x0200, StartX); //水平显示区地址
        LCD_WR_CMD(0x0201, StartY); //垂直显示区地址
        LCD_WR_REG(0x0202); //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(0x0200, StartX+x); //水平显示区地址
                    LCD_WR_CMD(0x0201, StartY+y); //垂直显示区地址
                    LCD_WR_REG(0x0202); //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++;
                if(x>=X){x=0; y++;} //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
                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(0x0210, StartY); //水平显示区起始地址 0-239
        LCD_WR_CMD(0x0211, StartX+Y-1); //水平显示区结束地址 0-239
        LCD_WR_CMD(0x0212, 399-(StartX+X-1)); //垂直显示区起始地址 0-399
        LCD_WR_CMD(0x0213, 399-StartX); //垂直显示区结束地址 0-399
        LCD_WR_CMD(0x0200, StartY); //水平显示区地址
        LCD_WR_CMD(0x0201, 399-StartX); //垂直显示区地址
        LCD_WR_REG(0x0202); //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(0x0200, StartY+y); //水平显示区地址
                    LCD_WR_CMD(0x0201, 399-(StartX+x)); //垂直显示区地址
                    LCD_WR_REG(0x0202); //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++;
                if(x>=X){x=0; y++;} //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
                i++;
            }
        }
    }
}

```

LCD_test () 函数演示了图片显示及字符串显示。

```
void LCD_test(void)
{
    unsigned char *p;
    ili9325_DrawPicture(0,0,0,a1);           //在某指定位置显示图片， 图片的尺寸要在240X400范围内。
    Delay(0x1afffff);
    ili9325_DrawPicture(0,0,1,a2);           //在某指定位置显示图片， 图片的尺寸要在400X240范围内。
    Delay(0x1afffff);
    lcd_wr_zf(0,0,280,32,color1,1,szf3[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。

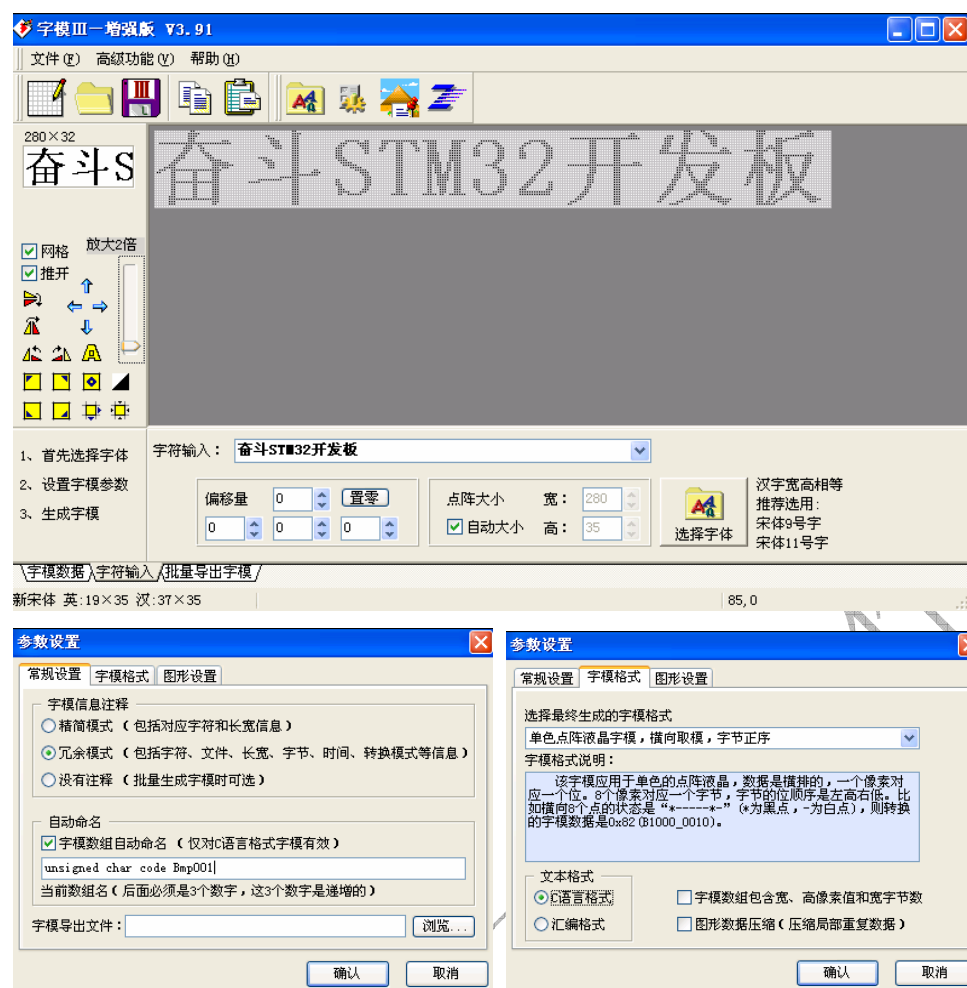
```
Build target 'STM32-FD-TFT'
linking...
Program Size: Code=2844 RO-data=308992 RW-data=4 ZI-data=516
FromELF: creating hex file...
".\Obj\STM32_FD_USART.axf" - 0 Error(s), 0 Warning(s).
```

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3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。要注意的是转换后的实际尺寸大小，就是界面最下一行的 输出图像那里。



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。