芯达STM32开发板

STM32 入门系列教程

点亮 LCD 液晶屏

Revision 0.01

(2010-04-28)

原想把本期《点亮 LCD 液晶屏》教程放在《GPIO 编程》之后，以提高大家的兴趣，但考虑到可能网友学习 STM32，是想更多地了解 STM32 内部工作机制，因此在之前的教程，我们先介绍了串口、外部中断、定时器等最基本的外设模块，有了这些基础，相信您再来学习 LCD 液晶，已经很轻松了。

我们使用的是芯达 STM32 配套的 2.4 寸 TFT 液晶触摸屏，它是山寨手机上的触摸液晶屏，内部驱动 IC 为 ILI9325。我们操作 LCD，实际上就是在操作 ILI9325。有关该芯片的资料，请参考如下两个网址：

ILI9325 英文datasheet：http://www.arm79.com/read.php?tid=1979

ILI9325 指令说明（中文）：http://www.arm79.com/read.php?tid=1980

考虑到“触摸”涉及到太多的原理，因此把触摸屏单独列出一期教程详细讲解。这里只讲述如何去点亮 LCD 液晶屏，如果您看完本期教程，能理解 LCD 驱动过程，那么笔者心满意足。

要驱动 LCD，分两个部分讲解：

1、CPU 内部模块支持的 LCD 接口（这里使用 FSMC 模块）

2、LCD 控制电路

一、STM32 的FSMC原理

如果是单片机，相信大家再熟悉不过了，直接拿 P0 或者 P1 口用作 LCD 数据总线，再另外拿出几个 IO 口用作控制信号线 —— 一个 LCD 控制电路完成了。STM32 相对于单片机，有啥过人之处呢？

对于 STM32 系列的 CPU 来说，有两种方法给 LCD 总线赋值。第一个方法，就是给对应的 GPIOx_ODR 寄存器赋值 —— 这与单片机一样，单片机也是给 P0-P3 寄存器赋值，使得信号能从对应的 IO 端口输出。而 STM32 的另一种方法就是使用 FSMC。FSMC 全称“静态存储器控制器”。使用 FSMC 控制器后，我们可以把 FSMC 提供的 FSMC_A[25:0]作为地址线，而把 FSMC 提供的 FSMC_D[15:0]作为数据总线。

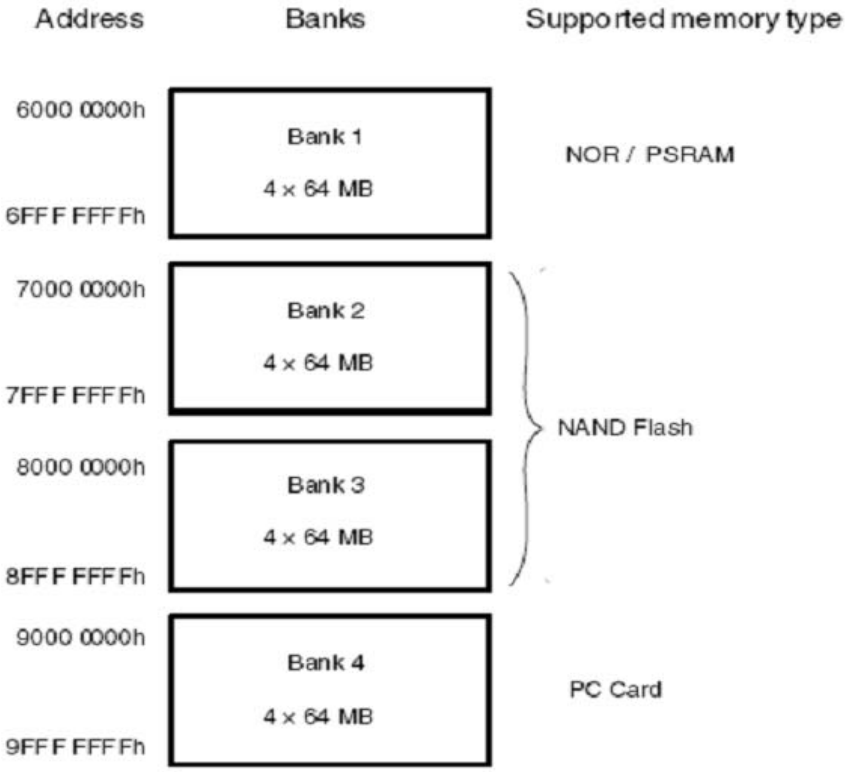
1、 FSMC 包括哪几个部分？

FSMC 包含以下四个模块：

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

要注意的是，FSMC 可以请求 AHB 进行数据宽度的操作。如果 AHB 操作的数据宽度大于外部设备（NOR 或 NAND 或 LCD）的宽度，此时 FSMC 将 AHB 操作分割成几个连续的较小的数据宽度，以适应外部设备的数据宽度。

2、FSMC 对外部设备的地址映像



从上图可以看出，FSMC 对外部设备的地址映像从 0x6000 0000 开始，到 0x9FFF FFFF 结束，共分 4 个地址块，每个地址块 256M 字节。可以看出，每个地址块又分为 4 个分地址块，大小 64M。对 NOR 的地址映像来说，我们可以通过选择 HADDR[27:26]来确定当前使用的是哪个 64M 的分地址块，如下页表格。而这四个分存储块的片选，则使用 NE[4:1]来选择。数据线/地址线/控制线是共享的。

HADDR[27:26](1)	选择的存储块
00	存储块1 NOR/PSRAM 1
01	存储块1 NOR/PSRAM 2
10	存储块1 NOR/PSRAM 3
11	存储块1 NOR/PSRAM 4

这里的 HADDR 是需要转换到外部设备的内部 AHB 地址线，每个地址对应一个字节单元。因此，若外部设备的地址宽度是 8 位的，则 HADDR[25:0]与 STM32 的 CPU 引脚 FSMC_A[25:0]一一对应，最大可以访问 64M 字节的空间。若外部设备的地址宽度是 16 位的，则是 HADDR[25:1]与 STM32 的 CPU 引脚 FSMC_A[24:0]一一对应。在应用的时候，可以将 FSMC_A 总线连接到存储器或其他外设的地址总线引脚上。

二、LCD控制电路设计

1、信号线的连接

STM32F10xxx FSMC 有四个不同的 banks（每个 64M 字节）可支持 NOR 以及其他类似的存储器。这些外部设备的地址线，数据先和控制线是共享的。每个设备的访问通过片选来决定，而每次只能访问一个设备。

FSMC 提供了所有的 LCD 控制器的信号：

FSMC_D[16:0] → 16bit 的数据总线

FSMC NEx：分配给 NOR 的 256M，再分为 4 个区，每个区用来分配一个外设，这四个外设的片选分为是 NE1-NE4，对应的引脚为：PD7—NE1，PG9—NE2，PG10-NE3，PG12—NE4

FSMC NOE：输出使能，连接 LCD 的 RD 脚。

FSMC NWE：写使能，连接 LCD 的 RW 脚。

FSMC Ax：用在 LCD 显示 RAM 和寄存器之间进行选择的地址线，即该线用于选择 LCD 的 RS 脚，该线可用地址线的任意一根线，范围：FSMC_A[25:0]。

注：RS = 0 时，表示读写寄存器；RS = 1 表示读写数据 RAM。

举例 1：选择 NOR 的第一个存储区，并且使用 FSMC_A16 来控制 LCD 的 RS 引脚，则我们访问 LCD 显示 RAM 的基址为 0x6002 0000，访问 LCD 寄存器的地址为：0x6000 0000。

举例 2：选择 NOR 的第四个存储区，使用 FSMC_A0 控制 LCD 的 RS 脚，则访问 LCD 显示 RAM 的基址为 0x6000 0002，访问 LCD 寄存器的地址为：0x6000 0000。

实际上，可用于 LCD 接口的 NOR 存储块信号如下：

FSMC_D[15:0]，连 16bit 数据线

FSMC_NE1，连片选：只有 bank1 可用
FSMC NOE：输出使能
FSMC NEW：FSMC 写使能
FSMC Ax：连接 RS，可用范围 FSMC_A[23:16]

2、时序问题

一般使用模式 B 来做 LCD 的接口控制，不适用外扩模式。并且读写操作的时序一样。此种情况下，我们需要使用三个参数：ADDSET，DATAST，ADDHOLD。这三个参数在位域 FSMC_TCRx 中设置。

当 HCLK 的频率是 72MHZ，使用模式 B，则有如下时序：
地址建立时间：0x1
地址保持时间：0x0
数据建立时间：0x5
好像有点理论化，呵呵，我们来编程看看就理解了。

三、LCD驱动编写

请大家在阅读此部分之前，务必先阅读 LCD 的驱动 IC：ILI9325。查看在本期教程开始，我们给出的两个网址即可。

我们的思路是：既然想使用 STM32 的 FSMC 模块，就首先要使能它的时钟，并初始化这个模块。然后初始化 LCD 启动配置，这时候，我们才可以编写用户程序，来控制 LCD 显示各种字符、图形。

根据这个思路，我们调用函数：

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);
```

来使能 FSMC 模块所使用的时钟。呵呵，STM32 固件库果然给我们提供了超方便的库函数，我们无需了解任何东西，只要知道调用这个函数即可。项目开发进度大大加快。

下面配置 FSMC 初始化部分，采用的函数是 FSMC_LCD_Init();，来看下它的实现吧！

```
void FSMC_LCD_Init(void)
{
  FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
  FSMC_NORSRAMTimingInitTypeDef  FSMC_TimingInitStructure;
```

```
  FSMC_TimingInitStructure.FSMC_AddressSetupTime = 0x02;
  FSMC_TimingInitStructure.FSMC_AddressHoldTime = 0x00;
  FSMC_TimingInitStructure.FSMC_DataSetupTime = 0x05;
  FSMC_TimingInitStructure.FSMC_BusTurnAroundDuration = 0x00;
  FSMC_TimingInitStructure.FSMC_CLKDivision = 0x00;
  FSMC_TimingInitStructure.FSMC_DataLatency = 0x00;
  FSMC_TimingInitStructure.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 = &FSMC_TimingInitStructure;
  FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &FSMC_TimingInitStructure;

  FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);
  FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);
}
```

上面的函数实现字体是小五，如果需要查看完整 FSMC-TFT-LCD 例程，请查看芯达 STM32 的光盘。此部分可作为一个模板，复制到您的项目文件中直接使用。实际上，控制 LCD 关键在于下面的初始化序列：

```
LCD_WriteReg(0x00E3, 0x3008); // Set u16ernal timing
LCD_WriteReg(0x00E7, 0x0012); // Set u16ernal timing
LCD_WriteReg(0x00EF, 0x1231); // Set u16ernal timing
LCD_WriteReg(0x0001, 0x0100); // set SS and SM bit
LCD_WriteReg(0x0002, 0x0700); // set 1 line inversion
LCD_WriteReg(0x0003, 0x1038); // set GRAM write direction and BGR=1.
LCD_WriteReg(0x0004, 0x0000); // Resize register
LCD_WriteReg(0x0008, 0x020E); // set the back porch and front porch
LCD_WriteReg(0x0009, 0x0000); // set non-display area refresh cycle ISC[3:0]
LCD_WriteReg(0x000A, 0x0000); // FMARK function
LCD_WriteReg(0x000C, 0x0000); // RGB u16erface setting
LCD_WriteReg(0x000D, 0x0000); // Frame marker Position
LCD_WriteReg(0x000F, 0x0000); // RGB u16erface polarity
//**************Power On sequence ****************//
LCD_WriteReg(0x0010, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP, STB
LCD_WriteReg(0x0011, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
LCD_WriteReg(0x0012, 0x0000); // VREG1OUT voltage
LCD_WriteReg(0x0013, 0x0000); // VDV[4:0] for VCOM amplitude
Delay(0XAFFFF); // Dis-charge capacitor power voltage
LCD_WriteReg(0x0010, 0x1290); // SAP, BT[3:0], AP, DSTB, SLP, STB
LCD_WriteReg(0x0011, 0x0221); // R11h=0x0221 at VCI=3.3V, DC1[2:0], DC0[2:0],
VC[2:0]
Delay(0XAFFFF); // Delay 50ms
LCD_WriteReg(0x0012, 0x001A); // External reference voltage= Vci;
Delay(0XAFFFF); // Delay 50ms
LCD_WriteReg(0x0013, 0x1600); // R13=0F00 when R12=009E;VDV[4:0] for
VCOM amplitude
LCD_WriteReg(0x0029, 0x0022); // R29=0019 when R12=009E;VCM[5:0] for
VCOMH
```

```
LCD_WriteReg(0x002B, 0x000A); // Frame Rate
Delay(0XAFFFF); // Delay 50ms
LCD_WriteReg(0x0020, 0x0000); // GRAM horizontal Address
LCD_WriteReg(0x0021, 0x0000); // GRAM Vertical Address
// ----------- Adjust the Gamma Curve ----------//
LCD_WriteReg(0x0030, 0x0000);
LCD_WriteReg(0x0031, 0x0302);
LCD_WriteReg(0x0032, 0x0202);
LCD_WriteReg(0x0035, 0x0103);
LCD_WriteReg(0x0036, 0x080C);
LCD_WriteReg(0x0037, 0x0505);
LCD_WriteReg(0x0038, 0x0504);
LCD_WriteReg(0x0039, 0x0707);
LCD_WriteReg(0x003C, 0x0301);
LCD_WriteReg(0x003D, 0x1008);
//------------------ Set GRAM area ---------------//
LCD_WriteReg(0x0050, 0x0000); // Horizontal GRAM Start Address
LCD_WriteReg(0x0051, 0x00EF); // Horizontal GRAM End Address
LCD_WriteReg(0x0052, 0x0000); // Vertical GRAM Start Address
LCD_WriteReg(0x0053, 0x013F); // Vertical GRAM Start Address
LCD_WriteReg(0x0060, 0x2700); // Gate Scan Line
LCD_WriteReg(0x0061, 0x0001); // NDL,VLE, REV
LCD_WriteReg(0x006A, 0x0000); // set scrolling line
//-------------- Partial Display Control ---------//
LCD_WriteReg(0x0080, 0x0000);
LCD_WriteReg(0x0081, 0x0000);
LCD_WriteReg(0x0082, 0x0000);
LCD_WriteReg(0x0083, 0x0000);
LCD_WriteReg(0x0084, 0x0000);
```

```
LCD_WriteReg(0x0085, 0x0000);
//---------- Panel Control ----------//
LCD_WriteReg(0x0090, 0x0010);
LCD_WriteReg(0x0092, 0x0600);
LCD_WriteReg(0x0093, 0x0003);
LCD_WriteReg(0x0095, 0x0110);
LCD_WriteReg(0x0097, 0x0000);
LCD_WriteReg(0x0098, 0x0000);
LCD_WriteReg(0x0007, 0x0133); // 262K color and display ON
```

以上初始化序列代码约有 55 个参数需要配置，每个参数为何配置成这样，由于篇幅有限，这里不一一讲述，详情请参考 http://www.arm79.com/read.php?tid=1980，ILI9325 的中文指令说明。实际上，如果您时间有限，可以直接copy这部分的内容，只需要编写具体的用户实现部分。

当然，在初始化之前，我们要注意 LCD 的复位操作。对于每个 LCD 模块来说，想初始化之前，必须先复位，ILI9325 的复位，是低电平有效。芯达 STM32 开发板根据版本的不同，对复位的操作也不一样。其中一个版本的复位直接采用 STM32 的 CPU 复位，另一个版本的复位采用 PC1 引脚。两者都是可以的。

经过以上步骤初始化之后，现在 LCD 可以显示图片和字符了。为了测试，我们分别编写了字符和图片的测试文件，您可以参考。

如果您对本教程还有不理解的地方，请直接到我们的网站：ARM技术交流网 www.arm79.com，进行讨论。我们将会尽快给您做出答复。

附：

福州芯达工作室简介

福州芯达工作室成立于 2009 年 9 月，我们专注于嵌入式产品的研发与推广，目前芯达产品涉及 ARM9 系列、STM32 系列。

芯达团队成员均硕士研究生毕业，具有一定研发实力。我们的愿景在于把福州芯达打造成国内一流的嵌入式品牌。或许我们现在做的还不够，但是我们真的努力在做，希望通过我们的努力，能够在您学习和使用芯达产品的过程中带来或多或少的帮助。

这是芯达为了配合 STM32 开发板而推出的入门系列教程。如果您在看了我们的教程后，理清了思路，我们都会倍感欣慰！让我们一起学习，共同进步，在征服嵌入式领域的道路上风雨同行！

官方网站：http://www.arm79.com/

官方淘宝：http://shop36353570.taobao.com/