**STM32 ----小谈FSMC RS选择（转）http://bbs.ednchina.com/BLOG\_ARTICLE\_2134144.HTM**

**STM32 ----小谈FSMC RS选择**发布时间：2011-09-12 09:52:36

STM32 FMSC  LCD难点解析：

以下是网上和自己整理的：感觉应该可以把STM32 ----FSMC LCD中的关键RS说清楚~

----------------------------------------------------------------------------------------------------------------------

第一个角度理解STM32有FSMC（其实其他芯片基本都有类似的总线功能），FSMC的好处就是你一旦设置好之后，WR(写)、RD(读)、DB0-DB15这些控制线和数据线，

都是FSMC自动控制的。打个比方，当你在程序中写到：

\*(volatile unsigned short int \*)(0x60000000)=val;

那么FSMC就会自动执行一个写的操作，其对应的主控芯片的WE、RD这些脚，就会呈现出写的时序出来（即WE=0,RD=1），数据val的值也

会通过DB0-15自动呈现出来(即FSMC-D0:FSMC-D15=val  )。地址0x60000000会被呈现在数据线上（即A0-A25=0，地址线的对应最麻烦，要根据具体情况来，好好看看FSMC手册）。

那么在硬件上面，我们需要做的，仅仅是MCU和LCD控制芯片的连接关系：

WE-WR，均为低电平有效

RD-RD，均为低电平有效

FSMC-D0-15接LCD DB0-15

FSMC\_NE1--CS接PD7

连接好之后，读写时序都会被FSMC自动完成。但是还有一个很关键的问题，就是RS没有接因为在FSMC里面，根本就没有对应RS。怎么办呢？这个时候，有一个好方法，就是用某一根地址线来接RS。比如我们选择了A16这根地址线来接，那么当我们要写寄

存器的时候，我们需要RS，也就是A16(RS为高)置高。软件中怎么做呢？也就是将FSMC要写的地址改成0x60020000，如下：

\*(volatile unsigned short int \*)(0x60020000)=val;

这个时候，A16在执行其他FSMC的同时会被拉高，因为A0-A18要呈现出地址0x60020000。0x60020000里面的Bit17=1，就会导致A16为1。

当要读数据时，地址由0x60020000改为了0x60000000，这个时候A16就为0了。

那么有朋友就会有疑问，第一，为什么地址是0x6xxxxxxx而不是0x0xxxxxxx；第二，CS怎么接；第三，为什么Bit17对应A16？

RS问题：RS为0表示；读写寄存器;RS为1，读写数据RAM；

先来看前两个问题，大家找到STM32的FSMC手册，在FSMC手册里面，我们很容易找到，FSMC将0x60000000-0x6fffffff的地址用作NOR/PRAM（

共256M地址范围）。而这个存储块，又被分成了四部分，每部分64M地址范围。当对其中某个存储块进行读写时，对应的NEx就会置低。这里，

就解决了我们两个问题，第一，LCD的操作时序，和NOR/PRAM是一样的（为什么一样自己找找NOR/PRAM的时序看看），所以我们选择0x6xxxxxxx

这个地址范围（选择这个地址范围，操作这个地址时，FSMC就会呈现出NOR/PRAM的时序）。第二，我们可以将NEx连接到LCD的CS，只要我们操作

的地址是第一个存储块内即可（即0-0x3ffffff地址范围）。

第三个问题再来看一看FSMC手册关于存储器字宽的描述，我们发现，当外部存储器是16位时，硬件管脚A0-A24表示的是地址线A1-A25的值，所以

我们要位移一下，Bit17的值，实际会被反应到A16这根IO来。关于数据宽度及位移的问题，初学的朋友可能会比较疑惑，当你接触了多NOR/PRAM

这样的器件后，你会发现，很多芯片的总线，都是这样设计的，为的是节省地址线。

第二个角度理解:

FSMC总线上看，LCD只有2个地址.

Bank1\_LCD\_C是写寄存器，此时RS=1,告诉LCD我在总线上输出数据的是寄存器的地址

Bank1\_LCD\_D是写数据，此时RS=0,告诉LCD我在总线上输出地数据是寄存器的数据或者GRAM的数据.

写寄存器数据按2步来:

第一步先往Bank1\_LCD\_C (对应RS=1),送寄存器的地址：\*(\_\_IO uint16\_t \*) (Bank1\_LCD\_C)= index;  接着在Bank1\_LCD\_D这个地址（对应RS=0),写入刚指向的寄存器的数据: \*(\_\_IO uint16\_t \*) (Bank1\_LCD\_D)= val;

为什么\*(\_\_IO uint16\_t \*) (Bank1\_LCD\_C)= index; 就是往 LCD 写寄存器呢？

这是一个16位的IO赋值操作,地址是Bank1\_LCD\_C,这个地址就是指向FSMC的 Bank1的NE1对应的地址空间。而LCD片选正是连接到NE1,具体地址要看RS接到哪一根地址线上。当CPU执行到这一条的时候，就会通过FSMC总线控制器在数据总线上进行一个地址为 Bank1\_LCD\_C的数据写操作，此操作自动完成CS信号，

RD信号，WR信号,以及地址总线数据(RS信号)的输出以及数据总线数据的输出.

其他的操作都是这两个操作组合完成。也就是我上面所说的，

"所有的寄存器地址和寄存器数据，以及 GRAM数据都是通过 IO0-IO15完成传输的，而不是FSMC的地址.这是容易搞混的一个地方.LCD的FSMC地址只有一根 ,就是RS."

----------------------------------------------------------------------------------------------------------------------第三个角度理解:

把TFT看做类似SRAM的存储器，只能接在 BANK1上。对应基地址是0x60000000.

而BANK1又有划分为四个片选,分别对应基地址:

NE1 0x600000000

NE2 0x640000000

NE3 0x680000000

NE4 0x6C0000000

所以每个NEx能寻址的空间大小为64M，也就是对应了FSMC的A0到A25 共26根地址线.

假如使用NE4接到为LCD的片选CS上，那么就对应基地址 0x6C000000,

如果RS接到地址线的 A0上，那么当 RS为0时对应的地址就是 LCD\_REG = 0x6C000000,(其实你用0x6CFFFFF0是一样的，因为只用到一根地址线).

RS为1时对应的地址就是 LCD\_RAM =0x6C000001,(0x6CFFFFF1一样对应 LCD\_RAM,因为它一样对应 RS=1).

如果 RS接到其他地址线上，情况是类似的。

比如接到 An上，那么

LCD\_REG= 0x6C000000,

LCD\_RAM= 0x6C000000 | (1<<n)

注意这个地址不是唯一的，只要这个地址能寻址到 BANK1 的 NE4上而且使 RS=0，那么就是 LCD\_REG,使 RS=1，就是LCD\_RAM.

----------------------------------------------------------------------------------------------------------------------

对应Bank1\_LCD\_C 的地址，FSMC总线控制器在RS接的那根地址线输出的是 1，而对应Bank1\_LCD\_D，输出的0.

RS接的可不是GPIO,是FSMC地址总线的一根.FSMC进行读写操作的时候会在地址总线根据要读写的地址输出电平的.

RS接哪一根地址线虽然没有固定要求，但是一旦你确定要接哪一根，那么Bank\_LCD\_C和Bank\_LCD\_D也要随之确定，这可不是“自动的".

虽然没有手动操作GPIO来操作RS,但是你敲代码的时候可是手动指定 Bank1\_LCD\_C 或者 Bank1\_LCD\_D ，从而确定 RS的电平.

所谓的“自动”是指:不是通过操作GPIO来操作RS,而是直接根据地址总线地址的不同来完成操作RS,这两种方法的速度差别是非常大的.

如果是GPIO方式，先要通过操作GPIO 分别输出 RS,CS,等的电平，然后再通过过GPIO操作输出数据，然后还要通过GPIO 再操作RD,WR,CS等的电平。

每操作一个GPIO都要好几个周期，加起来就非常慢了.

而FSMC是在一个FSMC写周期内就完成了这所有的动作。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define Bank1\_LCD\_R    ((uint32\_t)0x60000000)    //disp Reg ADDR

#define Bank1\_LCD\_D    ((uint32\_t)0x60020000)   //disp Data ADDR

从STM32 FMSC系统手册可以看到：

FSMC其实就相当于外部总线存储器和内部AHB总线的接口：而AHB是32位的，当外接NOR/LCD 时，而外部存储器的数据宽度可以选择8位和16位的，这时候就存在一个地址转换的问题即32位和8位或者16位地址转换的问题。解决这个问题STM32采用的HADDR[25:0]，它的作用就是将外部存储器地址转换为AHB地址线。

其中：





 并且无论外部存储器的宽度是多少位,FSMC\_A[0]始终连接到外部存储器的A[0];

NOR闪存/LCD

1：寄存器

2：RAM

FSMC核心控制器

(这里就只介绍使用BANK1的块1：这个块1的起始地址为0x6000 0000)                                                                              

                                       FS  FSMC\_NOE(输出使能)

http://hiphotos.baidu.com/billchian/pic/item/c8863119b051f8197a4aab05dab44aed2f73e7ae.jpg

STM32

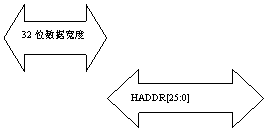
AHB总线

http://hiphotos.baidu.com/billchian/pic/item/3b14e73a0cd7912330eff03bad345982b3b7808d.jpg                                    FSMC\_NE1(片选)

http://hiphotos.baidu.com/billchian/pic/item/94ac8ed5912397dd0976fc4b5982b2b7d1a2878d.jpg

RS?

http://hiphotos.baidu.com/billchian/pic/item/428f3253f8198618cb2847cb4aed2e738ad4e6ae.jpghttp://hiphotos.baidu.com/billchian/pic/item/fd43132197dda144040608fdb2b7d0a20df4868d.jpgFSMC\_NWE(写使能)



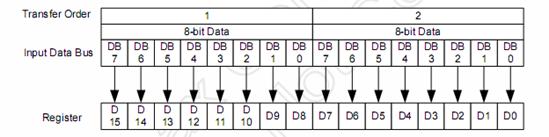
①HADDR[25:0]总共26位线，可以寻址64M的地址空间，而AHB总线是32位宽度，所以能寻址0x0000 0000~0xFFFF FFFF；在这里我们只是指定了FMSC控制器块1的地址为0x6000 0000。寻址空间0x6000 0000 ~0x63ff  ffff;

②LCD 手册相关知识：(9320或者hx8312都一样)

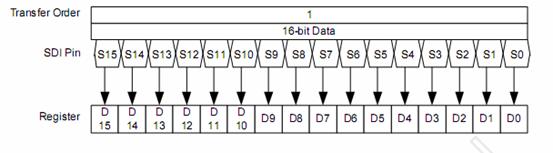
RS引脚决定是寄存器命令还是显示RAM数据。寄存器指令的输入数据是16位，前8位是地址，后8位是数据。

http://hiphotos.baidu.com/billchian/pic/item/098d041a367adab4b5bfb30c8bd4b31c8601e4ae.jpg

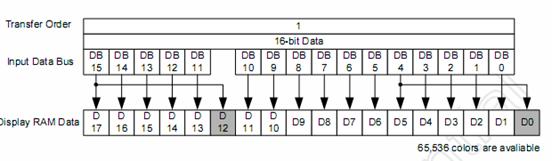
不同的MPU类型寄存器命令和数据总线的格式不同。并且对应输入总线：有一下对应关系：



对于SPI方式，则用以下对应关系：



(2)显示RAM数据格式和系统接口输入总线之间的联系



问题：RS如何选择：

#define Bank1\_LCD\_R    ((uint32\_t)0x60000000)    //disp Reg ADDR

#define Bank1\_LCD\_D    ((uint32\_t)0x60020000)   //disp Data ADDR

这里LCD选取的16位，将RS接在A16，则HADDR[25:1]对应FSMC\_A[24:0];关键在于为什么？？？

从上面可以看出，LCD除了需要数据线之外，额外的地址线是不需要的~~~~~但是在STM32在进行FSMC总线操作时，所有的地址线还是会出现时序的，但是操纵LCD 不需要额外的地址线了，也就是FSMC\_A[16:25]可以解放了，但是要注意一旦配置了FSMC，这些管脚还是会出现时序的；

现在我们向0x60000000这个块地址送出数据，当然这些数据肯定是16位的，因为是16位的LCD，由于RS(A16)为0，所以这个读写寄存器的操作;当向0x60020000写数据时，由于总线时序是要有地址写的，这时bit17就为高了，也就是RS为1了，这时所进行的操作就是读写RAM了！！！！

其中RS的选择可以是任意的，但一般还是选择，不用的地址线为好~~~~~