**一．通讯协议：**

DMA:Direct Memory Access,直接存储器访问。

DMA传输将数据从一个地址空间复制到另一个地址空间，提供在外设和存储器之间或者存储器和存储器之间的高速数据传输。当CPU初始化这个传输动作，传输动作本身是由DMA控制器来实现和完成的。DMA传输方式无需CPU直接控制传输，也没有中断处理方式那样保留现场和恢复现场过程，通过硬件为RAM和IO设备开辟一条直接传输数据的通道，使得CPU的效率大大提高。

**特征：**

·每个通道都直接连接专用的硬件DMA请求，每个通道都同样支持软件触发，这些功能通过软件来配置。

·在同一个DMA模块上，多个请求间的优先权可以通过软件编程设置（共有四级：很高、高、中等和低），优先权设置相等时由硬件决定（请求0优先于请求1，依此类推）。

·独立数据源和目标数据区的传输宽度（字节、半字、全字），模拟打包和拆包的过程。源和目标地址必须按数据传输宽度对齐。

·支持循环的缓冲器管理。

·每个通道都有3个事件标志（DMA半传输、DMA传输完成和DMA传输出错），这3个事件标志逻辑或成为一个单独的中断请求。

·存储器和存储器间的传输、外设和存储器、存储器和外设之间的传输。

·闪存、SRAM、外设的SRAM、APB1、APB2和AHB外设均可作为访问的源和目标。

·可编程的数据传输数目：最大为65535（0xFFFF）。

串行通信接口



SPI： Serial Perripheral Interface, 串行外围设备接口，支持高速、全双工的同步（同步通信指的是一种比特同步通信技术，要求发收双方具有同频同相的同步时钟信号，SPI是通过CLK和相位实现这一点的）串行方式，SPI 主要应用在 EEPROM, Flash, 实时时钟(RTC), 数模转换器(ADC), 数字信号处理器(DSP) 以及数字信号解码器之间. 它在芯片中只占用四根管脚 (Pin) 用来控制以及数据传输。

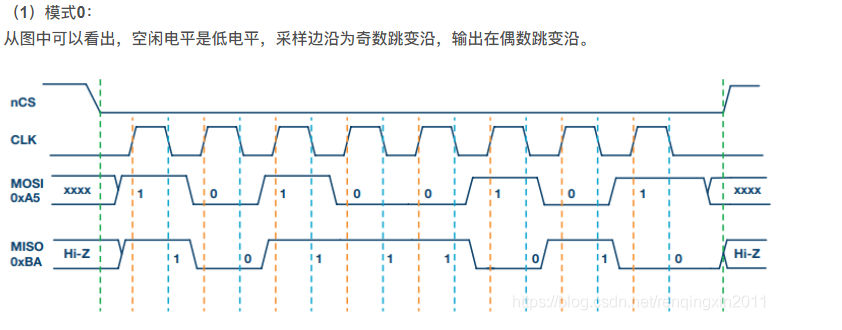
接口定义：4根信号线，设备选择线、时钟线、串行输出数据线、串行输入数据线

（1）MOSI：主设备数据输出，从设备数据输入 master out slave in  
（2）MISO：主设备数据输入，从设备数据输出 master in slave out  
（3）SCLK ：时钟信号，由主器件产生  
（4）/SS：片选，从器件使能信号，由主器件控制（主从模式选择线）

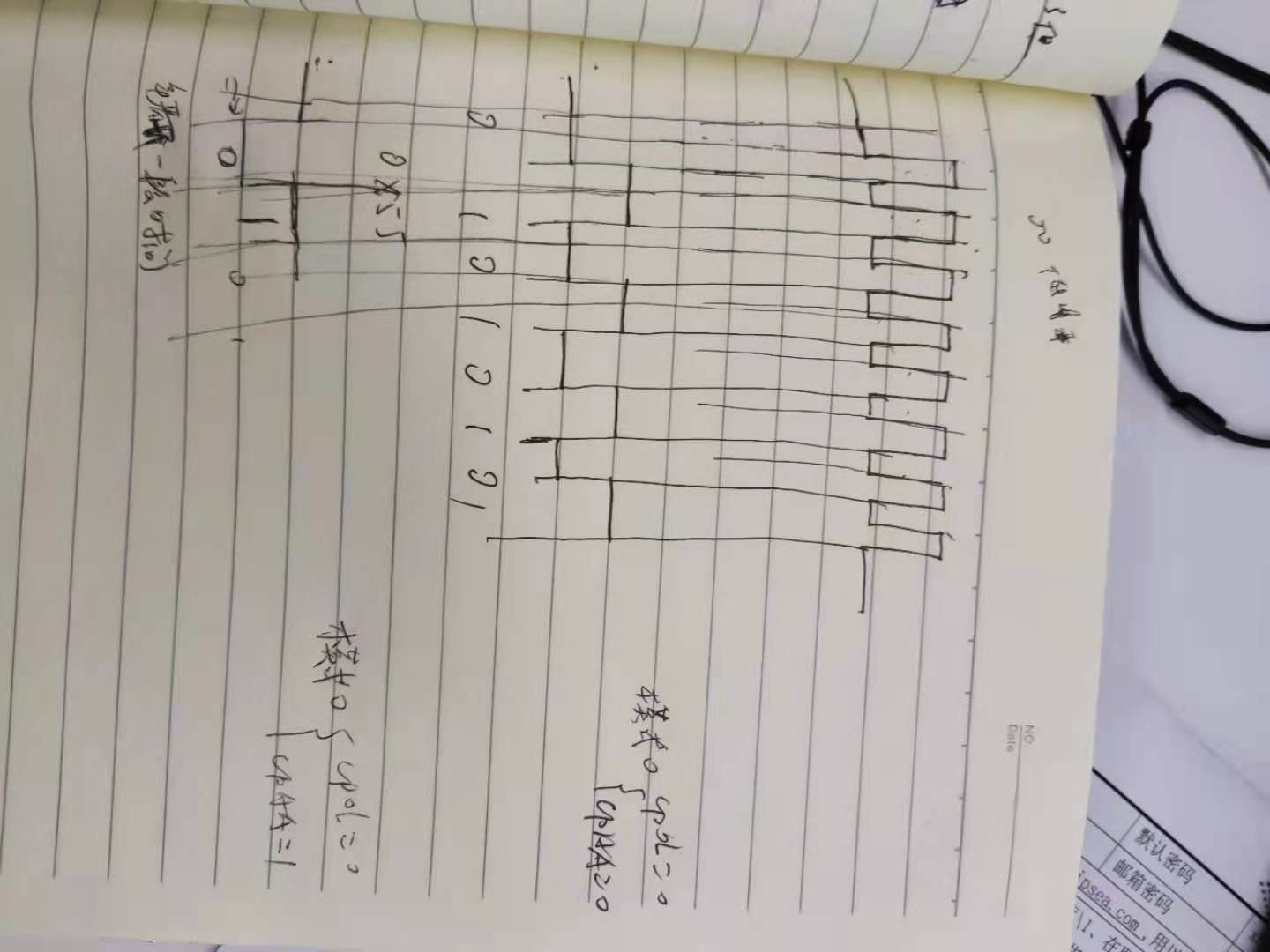
注：（1）SPI会有主从机（或者对于MCU来说会有主模式，从模式）之分，它的区分依据是SCLK同步时钟和SS片选是由谁输出，输出方就会被定义为工作在主机（主模式）状态下。（2）如果在一个系统中只含有一个采用SPI通信的外设，那么我们可以将外设的SS引脚直接连接板子的GND引脚，这样使得外设始终被选中，从而节省了连接线（三线spi）。

SPI协议时序或者工作模式：

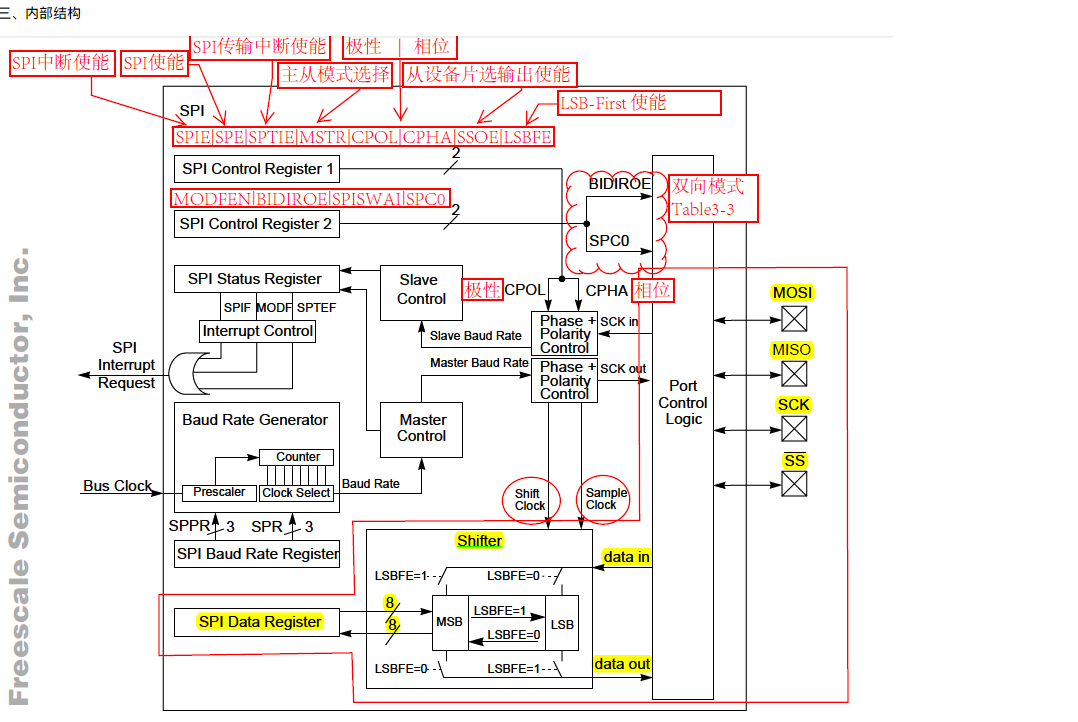




空闲电平，MOSI和MISO上面的数据可以变，非空闲的时钟电平才是有效电平。上图中的MOSI和MISO信号没有啥联系，主机的MOSI向从机发了一段10100101的数据而已，每个时钟周期发一位数据。首先在上升沿的时候1被采集（应该此时到了主机数据寄存器当中），然后在下降沿的时候把bit1送出去给从机。



具体硬件内部结构：



主要寄存器有：SPCR,控制寄存器，spi的大部分设置

SPSR（status register）:标志寄存器，（查看一下标志，SPIF中断位）

SPDR：数据寄存器

工作过程：

设置好相应的配置后就可以发送或接收数据，当放送或接收完成后就会是SPIF置位，此时如果开了中断就会就入中断服务程序。如果没开中断就可以通过查询该标志位来做相应的事。进入中段服务程序后，或者读（写）SPDR寄存器都可以清零SPIF标志位。从而可以开始下一次的传输

一般的外设应该都是满足这种工作流程。

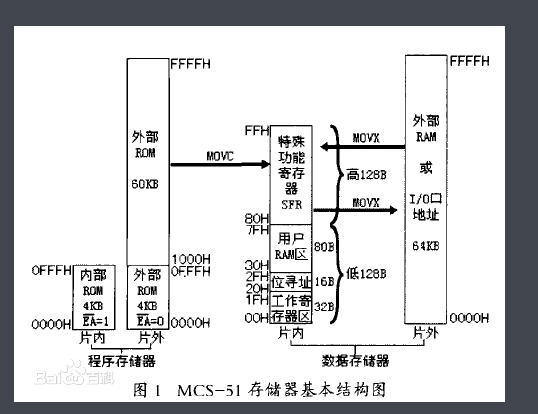
**二．芯片通讯电平**：

TTL[电平信号](https://baike.baidu.com/item/%E7%94%B5%E5%B9%B3%E4%BF%A1%E5%8F%B7)规定，+5V等价于逻辑“1”，0V等价于逻辑“0”(采用二进制来表示[数据](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE/5947370)时)。这样的数据通信及电平规定方式，被称做TTL（晶体管-晶体管逻辑电平）[信号系统](https://baike.baidu.com/item/%E4%BF%A1%E5%8F%B7%E7%B3%BB%E7%BB%9F/22324481)。这是[计算机处理器](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%A4%84%E7%90%86%E5%99%A8/10313260)控制的设备内部各部分之间通信的标准技术。

还有cmos电平和232电平等

TTL电平,TTL的电源工作电压是5V，所以TTL的电平是根据电源电压5V来定的。CMOS电平，CMOS的电源工作电压是3V - 18V，CMOS的电源工作电压范围宽，如果你的CMOS的电源工作电压是12V，那么这个CMOS的输入输出电平电压要适合12V的输入输出要求。即CMOS的电平，要看你用的电源工作电压是多少，3v - 18V，都在CMOS的电源工作电压范围内，具体数值，看你加在[CMOS芯片](https://baike.baidu.com/item/CMOS%E8%8A%AF%E7%89%87/10778407)上的电源工作电压是多少。

**三．单片机存储结构**：80C51采用哈佛结构，程序存储器和数据存储器分开。



**程序存储器（ROM）**：MCS-51中,程序存储器通过16位程序计数器(PC)寻址,具有64KB寻址能力,也即可以在64KB的地址空间任意寻址。其中,具有4KB片内程序存储器空间,地址为000H-0FFFH(注:8031无片内程序存储器);片外程序存储器空间最大可扩展到64KB,地址为0000H-FFFFH,片内、外统一编址。

0000H单元为系统启动地址。MCS-51单片机启动复位后,程序计数器(PC)的内容为0000H,所以系统将从0000H单元取指令,并开始执行程序。程序设计时一般在该地址存放一条绝对跳转指令,转入主程序的入口地址。

另外0003H一002AH日被均匀地分成5段,用于5个中断服务程序的入口。

其中0003H、000BH、0013H、O01BH、0023H为5个中断源的中断服务程序的入口地址。

中断源的中断服务程序的入口地址,即当中断产生相应的中断服务程序的起始地址被装入程序计数器(PC),系统将从该地址取指令,并执行程序[1]  。

**数据存储器(RAM):**

8051有256个单元的内片数据存储器，其中00H-7FH为片内随机存储器RAM，也叫低128B；80H-FFH为特殊功能寄存器，也叫高128B。低128B又分为工作寄存器区、位寻址区、用户RAM区。

1.工作寄存器区(00H-1FH)

在00H-1FH共32个单元，被均匀地分为四组工作寄存器堆：RB0、RB1、RB2、RB3，每组寄存器堆包含8个工作寄存器，均以R0-R7来命名，这些寄存器被称为通用寄存器。工作寄存器用于临时存放8位信息。在使用时，由程序状态字寄存器中的RS0、RS1来选择工作寄存器堆。

2.位寻址区(20H-2FH)

片内数据存储器的20H-2FH存储区为位寻址区，既可作为一般单元按字节寻址，也可按位进行寻址。位寻址区有16个字节，位地址为00H-7FH。

3.用户RAM区(30H-7FH)

用户RAM区主要用作数据缓冲区和堆栈。这个存储区只能按字节寻址，用作存放数据及作为堆栈区。通常堆栈区被设置在这块存储区，由堆栈寄存器SP指定，CPU复位时SP=07H，使得堆栈实际上是从08H开始的，但08H-1FH属于工作寄存器区[1]  。

**四．GPIO通用型输入输出：如8051的P0-P3，stm32的GpioA,GPIOB等，是一个外设，其引脚供使用者由程序控制使用。**

GPIO的优点(端口扩展器)

低功耗：GPIO具有更低的功率损耗(大约1μA，μC的工作电流则为100μA)。

集成IIC从机接口：GPIO内置IIC从机接口，即使在待机模式下也能够全速工作。

小封装：GPIO器件提供最小的封装尺寸 ― 3mm x 3mm QFN!

低成本：您不用为没有使用的功能买单。

快速上市：不需要编写额外的代码、文档，不需要任何维护工作。

灵活的灯光控制：内置多路高分辨率的PWM输出。

可预先确定响应时间：缩短或确定外部事件与中断之间的响应时间。

更好的灯光效果：匹配的电流输出确保均匀的显示亮度。

布线简单：仅需使用2条就可以组成IIC总线或3条组成SPI总线。

与ARM 的几组GPIO[引脚](https://baike.baidu.com/item/%E5%BC%95%E8%84%9A)，功能相似，GPxCON 控制引脚功能，GPxDAT用于读写引脚数据。另外，GPxUP用于确定是否使用上拉电阻。 x为A,B,,H/J,

GPAUP 没有上拉电阻。

**GPIO的使用：**既然一个[引脚](https://baike.baidu.com/item/%E5%BC%95%E8%84%9A/10879873)可以用于输入、输出或其他特殊功能，那么一定有[寄存器](https://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8/187682)用来选择这些功能。对于输入，一定可以通过读取某个寄存器来确定引脚电位的高低；对于输出，一定可以通过写入某个寄存器来让这个引脚输出高电位或者低电位；对于其他特殊功能，则有另外的寄存器来控制它们。

* 4种输入模式：
  + 输入浮空IN\_FLOATING
  + 输入上拉IPD
  + 输入下拉IPU
  + 模拟输入AIN
* 4种输出模式：
  + 开漏输出
  + 开漏复用功能
  + 推挽式输出
  + 推挽式复用功能
* 3种最大输出速度（看具体芯片型号）
  + 2MHZ
  + 10MHZ
  + 50MHZ

**五．中断系统包括中断装置和中断处理函数（中断服务函数）**。中断系统是计算机的重要组成部分。实时控制、故障自动处理、计算机与外围设备间的数据传送往往采用中断系统。中断系统的应用大大提高了计算机效率。

不同的计算机其硬件结构和软件指令是不完全相同的，因此，中断系统也是不相同的。计算机的中断系统能够加强CPU对多任务事件的处理能力。[中断机制](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%9C%BA%E5%88%B6)是现代计算机系统中的基础设施之一，它在系统中起着通信网络作用，以协调系统对各种外部事件的响应和处理。中断是实现[多道程序设计](https://baike.baidu.com/item/%E5%A4%9A%E9%81%93%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1)的必要条件。 中断是CPU对系统发生的某个事件作出的一种反应。 引起中断的事件称为[**中断源**](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%BA%90)。中断源向CPU提出处理的请求称为[**中断请求**](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E8%AF%B7%E6%B1%82)。发生中断时被打断程序的暂停点成为[**断点**](https://baike.baidu.com/item/%E6%96%AD%E7%82%B9)。CPU暂停现行程序而转为响应[中断请求](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E8%AF%B7%E6%B1%82)的过程称为[**中断响应**](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%93%8D%E5%BA%94)。处理中断源的程序称为[**中断处理**](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%A4%84%E7%90%86)**程序**。CPU执行有关的中断处理程序称为[**中断处理**](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%A4%84%E7%90%86)。而返回[断点](https://baike.baidu.com/item/%E6%96%AD%E7%82%B9)的过程称为**中断返回**。中断的实现由实行软件和硬件综合完成，硬件部分叫做硬件装置，软件部分称为软件处理程序。

**Cpu中断响应过程：**首先将断点处的PC值推入堆栈保留下来，实行断点保护，由硬件自动执行。然后，将有关的[寄存器](https://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8)内容和标志位状态推入堆栈保留下来，这称为保护现场，由用户自己编程完成。保护断点和现场后即可执行[中断服务程序](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%9C%8D%E5%8A%A1%E7%A8%8B%E5%BA%8F)，执行完毕，CPU由中断服务程序返回主程序，中断返回过程如下：首先恢复原保留寄存器的内容和标志位的状态，这称为[恢复现场](https://baike.baidu.com/item/%E6%81%A2%E5%A4%8D%E7%8E%B0%E5%9C%BA)，由用户编程完成。然后，再加返回指令RETI，RETI指令的功能是恢复PC值，使CPU返回[断点](https://baike.baidu.com/item/%E6%96%AD%E7%82%B9)，这称为恢复断点。恢复现场和断点后，CPU将继续执行原主程序，[中断响应](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%93%8D%E5%BA%94)过程到此为止。

**优先级排队：**通常，系统中有多个[中断源](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%BA%90)，当有多个中断源同时发出[中断请求](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E8%AF%B7%E6%B1%82)时，要求计算机能确定哪个中断更紧迫，以便首先响应。为此，计算机给每个中断源规定了优先级别，称为**优先权**。这样，当多个中断源同时发出中断请求时，优先权高的中断能先被响应，只有优先权高的[中断处理](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%A4%84%E7%90%86)结束后才能响应优先权低的中断。计算机按中断源优先权高低逐次响应的过程称**优先权排队**，这个过程可通过硬件电路来实现，亦可通过软件查询来实现。

**中断嵌套：**当CPU响应某一中断时，若有优先权高的[中断源](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%BA%90)发出[中断请求](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E8%AF%B7%E6%B1%82)，则CPU能中断正在进行的[中断服务程序](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%9C%8D%E5%8A%A1%E7%A8%8B%E5%BA%8F)，并保留这个程序的[断点](https://baike.baidu.com/item/%E6%96%AD%E7%82%B9)（类似于[子程序](https://baike.baidu.com/item/%E5%AD%90%E7%A8%8B%E5%BA%8F)嵌套），响应高级中断，高级[中断处理](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E5%A4%84%E7%90%86)结束以后，再继续进行被中断的中断服务程序，这个过程称为**中断嵌套**。如果发出新的中断请求的中断源的优先权级别与正在处理的中断源同级或更低时，CPU不会响应这个中断请求，直至正在处理的中断服务程序执行完以后才能去处理新的中断请求。

**六．定时器：用于定时、计数或者波形的其他处理。**

对于80C51单片机内集成了两个可编程的定时/计数器：T0和T1，它们既可以工作在定时模式也可以工作在外部事件计数模式。T1还是串行口的波特率发生器。这些都是由两个特殊的寄存器TMOD和TCON来配置控制的。

对于STM32F1（contex-m3内核）系列除了互联型的产品，共有 8 个定时器，分为基本定时器，通用定时器和高级定时器。基本定时器 [TI](http://www.elecfans.com/tags/%E5%BE%B7%E5%B7%9E%E4%BB%AA%E5%99%A8/)M6 和 [TI](http://bbs.elecfans.com/zhuti_715_1.html)M7 是一个 16 位的只能向上计数的定时器，只能定时，没有外部 IO。通用定时器 [TI](http://bbs.elecfans.com/zhuti_715_1.html)M2/3/4/5 是一个 16 位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，每个定时器有四个外部 IO。高级定时器 [TI](http://bbs.elecfans.com/zhuti_715_1.html)M1/8是一个 16 位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，还可以有[三相电](http://m.elecfans.com/article/630779.html)机互补输出信号（pwm输出），每个定时器有 8 个外部 IO。

同样可以配置相应的寄存器来选择使用定时器，最重要是确定定时器的时基，即时钟来源。

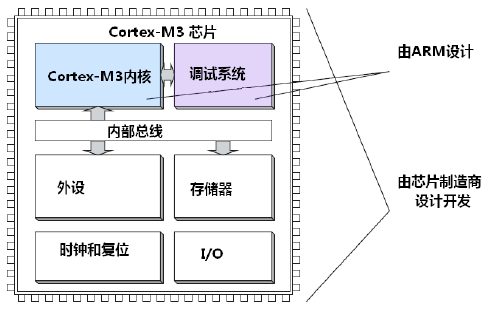
**七．RTOS实时操作系统(包括一个实时任务调度器)**, 会按照排序运行、管理系统资源，并为开发应用程序提供一致的基础。实时操作系统与一般的操作系统相比，最大的特色就是“实时性”，如果有一个任务需要执行，实时操作系统会马上（在较短时间内）执行该任务，不会有较长的延时。这种特性保证了各个任务的及时执行。单片机裸机开发可能会带来很多问题：1.并发性：程序并发工作效率低，裸机软件一般在主程序中用一个巨大的while循环包括所有业务逻辑。而里面又包含了许多delay延时循环等待过程，使得每个业务逻辑几乎是串行起来工作。Cpu会浪费很多时间在空转，使得效率低下。2.模块化不好，所有业务都在一个while循环中，各模块耦合严重。特别是大项目。

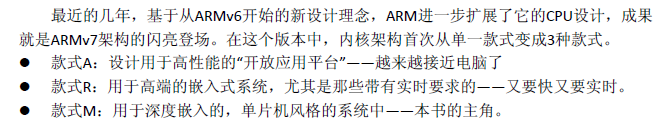
3.生态不好，不能使用很多依赖操作系统实现的高级软件组件（SDk）.4.实时性无法保证，特别是在工控等场景。5软件重用性低。

RTOS的优势：线程方式的并发任务处理，解决模块化问题，同时保证实时性。常用的有uc/os,FreeRTOS,RT-Thread.

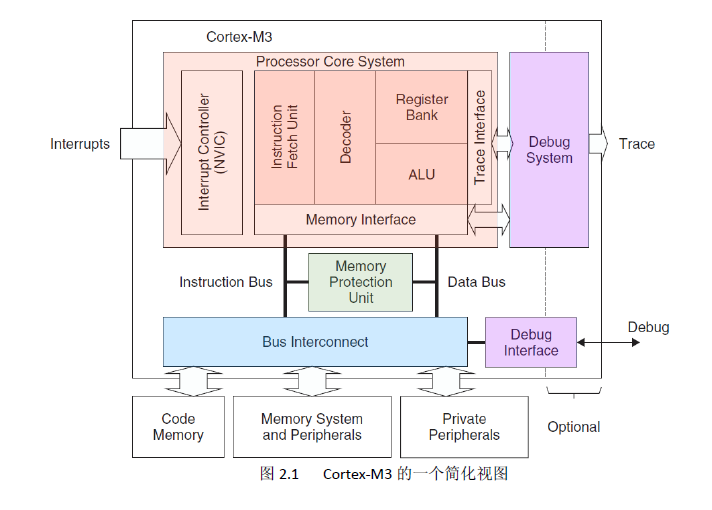
Cortex-M3处理器内核和基于Cortex-M3的MCU。

Cortex‐M3处理器内核是单片机的中央处理单元（CPU）。完整的基于CM3的MCU还需要很多其它组件。在芯片制造商得到CM3处理器内核的使用授权后，它们就可以把CM3内核用在自己的硅片设计中，添加存储器，外设，I/O以及其它功能块。不同厂家设计出的单片机会有不同的配置，包括存储器容量、类型、外设等都各具特色。本书主讲处理器内核本身。如果想要了解某个具体型号的处理器，还需查阅相关厂家提供的文档。



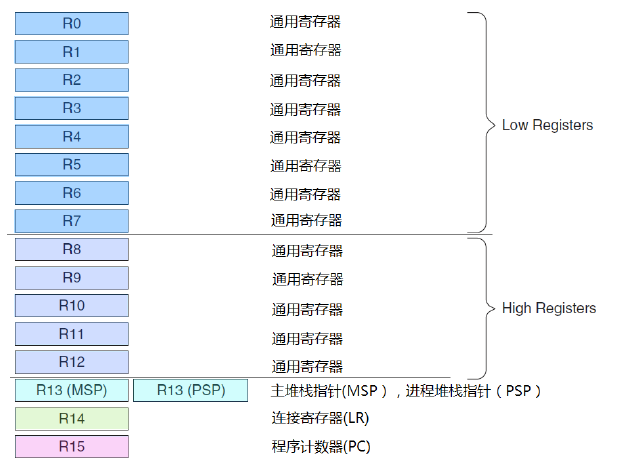


CM3内核的结构：

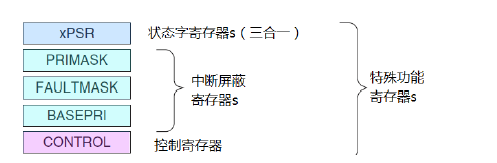


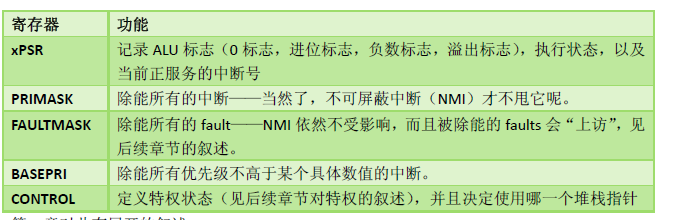
Cortex‐M3 是一个32 位处理器内核。内部的数据路径是32 位的，寄存器是32 位的，存储器接口也是32 位的。CM3 采用了哈佛结构，拥有独立的指令总线和数据总线，可以让取指与数据访问并行不悖。这样一来数据访问不再占用指令总线，从而提升了性能。为实现这个特性， CM3 内部含有好几条总线接口，每条都为自己的应用场合优化过，并且它们可以并行工作。但是另一方面，指令总线和数据总线共享同一个存储器空间（一个统一的存储器系统）。换句话说，不是因为有两条总线，可寻址空间就变成8GB 了。

**寄存器组：**CM3有16个32位寄存器组，其中R13 作为堆栈指针SP。SP 有两个，但在同一时刻只能有一个可以看到，这也就是所谓的“banked”寄存器。



除此之外还有若干个特殊功能寄存器





**嵌套向量中断控制器NVIC（Nested Vectored Interrupt Controller）**

Cortex‐M3 在内核水平上搭载了一颗中断控制器——嵌套向量中断控制器NVIC(Nested VectoredInterrupt Controller)。它与内核有很深的“私交”——与内核是紧耦合的。NVIC 提供如下的功能：

**可嵌套的中断支持**

**向量中断支持**

**动态优先级调整支持**

**中断延迟大大缩短**

**中断可屏蔽**

可嵌套中断支持

可嵌套中断支持的作用范围很广，覆盖了所有的外部中断和绝大多数系统异常。外在表现是，这些异常都可以被赋予不同的优先级。当前优先级被存储在xPSR 的专用字段中。当一个异常发生时，硬件会自动比较该异常的优先级是否比当前的异常优先级更高。如果发现来了更高优先级的异常，处理器就会中断当前的中断服务例程（或者是普通程序），而服务新来的异常——即立即抢占。

向量中断支持

当开始响应一个中断后，CM3 会自动定位一张向量表，并且根据中断号从表中找出ISR 的入口地址，然后跳转过去执行。不需要像以前的ARM 那样，由软件来分辨到底是哪个中断发了，也无需半导体厂商提供私有的中断控制器来完成这种工作。这么一来，中断延迟时间大为缩短。

动态优先级调整支持

软件可以在运行时期更改中断的优先级。如果在某ISR 中修改了自己所对应中断的优先级，而且这个中断又有新的实例处于悬起中（pending），也不会自己打断自己，从而没有重入(reentry)风险。

中断延迟大大缩短

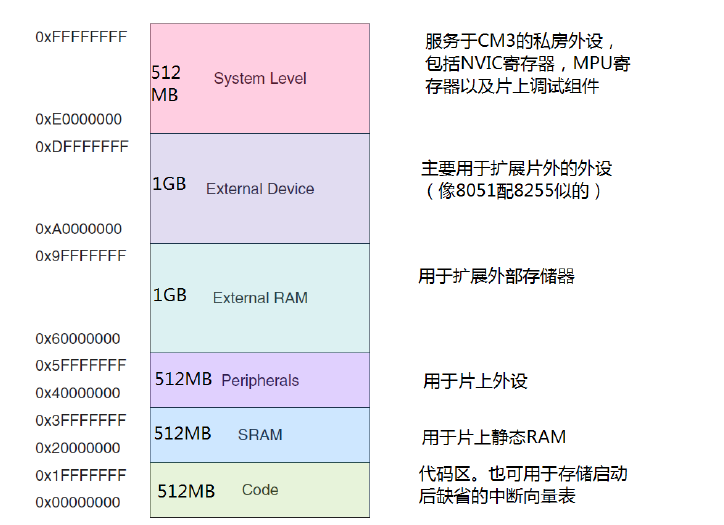
Cortex‐M3 为了缩短中断延迟，引入了好几个新特性。包括自动的现场保护和恢复，以及其它的措施，用于缩短中断嵌套时的ISR 间延迟。详情请见后面关于“咬尾中断”和“晚到中断”的讲述。

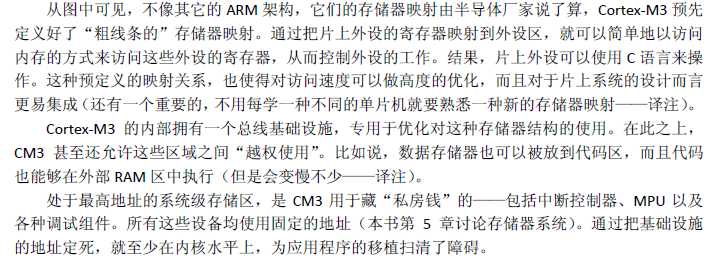
中断可屏蔽

既可以屏蔽优先级低于某个阈值的中断/异常［译注8］(设置BASEPRI 寄存器)，也可以全体封杀(设置PRIMASK 和FAULTMASK 寄存器)。这是为了让时间关键（time‐critical）的任务能在死线(deadline，或曰最后期限)到来前完成，而不被干扰

**存储器映射**

CM3支持4GB的存储空间，被划分为几个区域如下：





**总线接口：**

Cortex‐M3 内部有若干个总线接口，以使CM3 能同时取址和访内（访问内存），它们是：

指令存储区总线（两条）

系统总线

私有外设总线

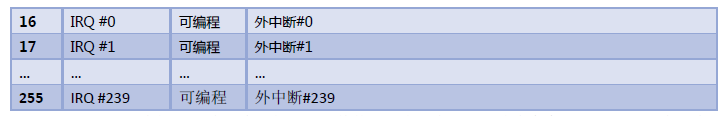
有两条代码存储区总线负责对代码存储区的访问，分别是I‐Code 总线和D‐Code 总线。前者用于取指，后者用于查表等操作，它们按最佳执行速度进行优化。系统总线用于访问内存和外设，覆盖的区域包括SRAM，片上外设，片外RAM，片外扩展设备，以及系统级存储区的部分空间。

**存储器保护单元（MPU）可选**

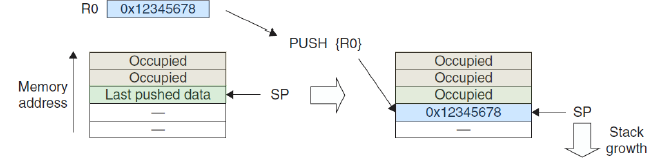
**指令集使用Thumb-2指令集。有32位和16位指令并用。兼顾代码密度和处理速度**

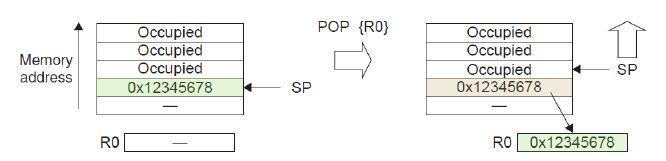
**中断和异常：**CM3所有中断机制都由NVIC实现，支持240条外部中断（由外设发出的中断请求）具体多少由芯片厂商决定和11个内部中断（硬件或者运算出错引发）





**CM3堆栈的实现：**使用“向下生长的满栈”模型。堆栈指针sp指向最后一个被压入栈的32位数值。在下一次压栈push时，sp先自减4，再存入新的值。pop数据时，先从sp指针处读出值，再自增4.



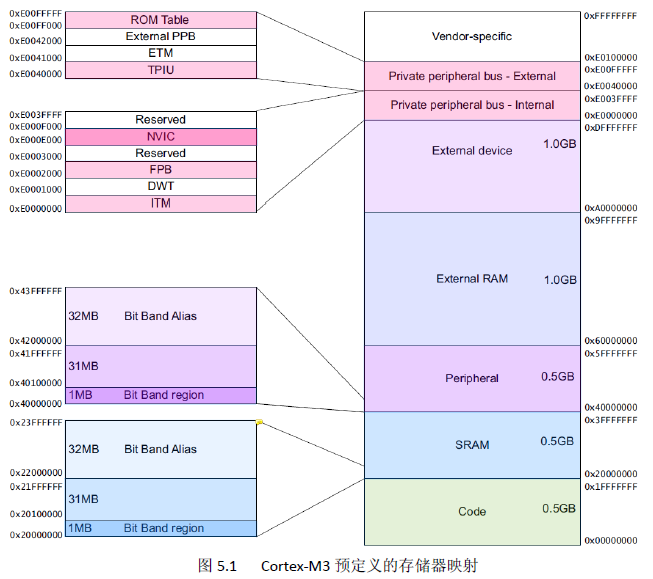


大端存储模式：数据的高字节保存在低地址，低字节保存在高地址。小端相反。

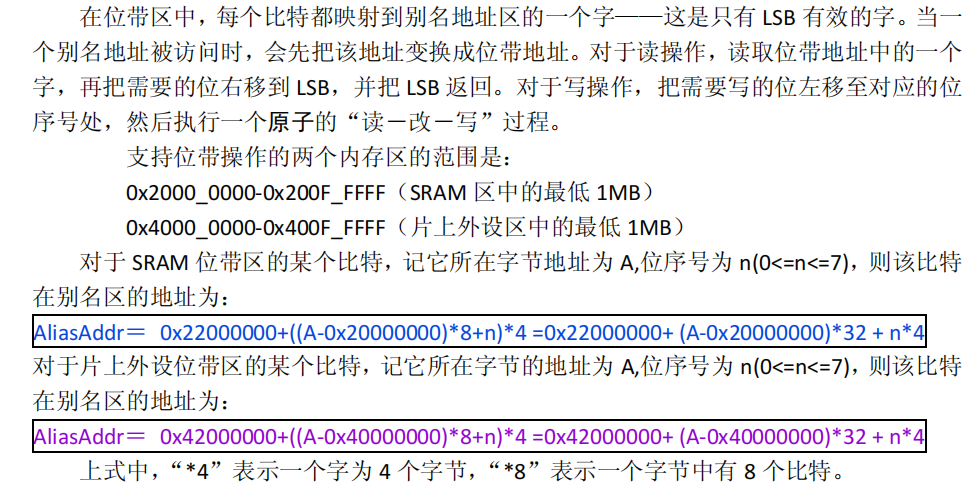
# 存储系统

CM3只有一个单一固定的存储器映射，将内核外的存储器和外设以及核内的外设都映射到固定的位置。但在这些固定的空间内，芯片制造商可以灵活分配存储器空间，为各外设布设寄存器。

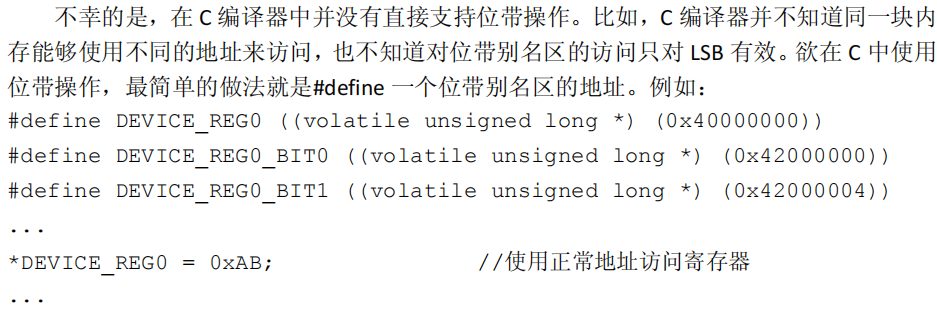
这4GB空间的粗线条划分为：



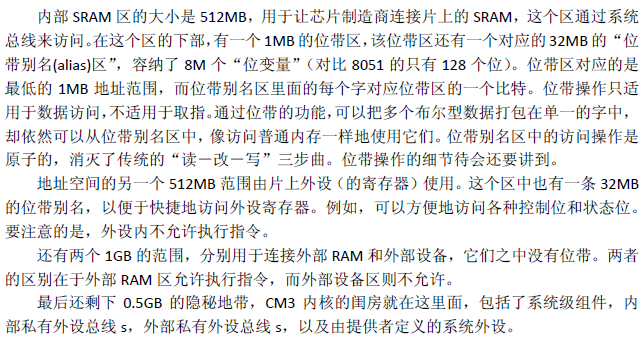
位带别名区里面的每个字对应位带区的一个比特。位带别名区里只有LSB位有效，通过访问微带别名区里这些字，就可以达到访问原始比特的目的。它们之间有一个地址映射关系



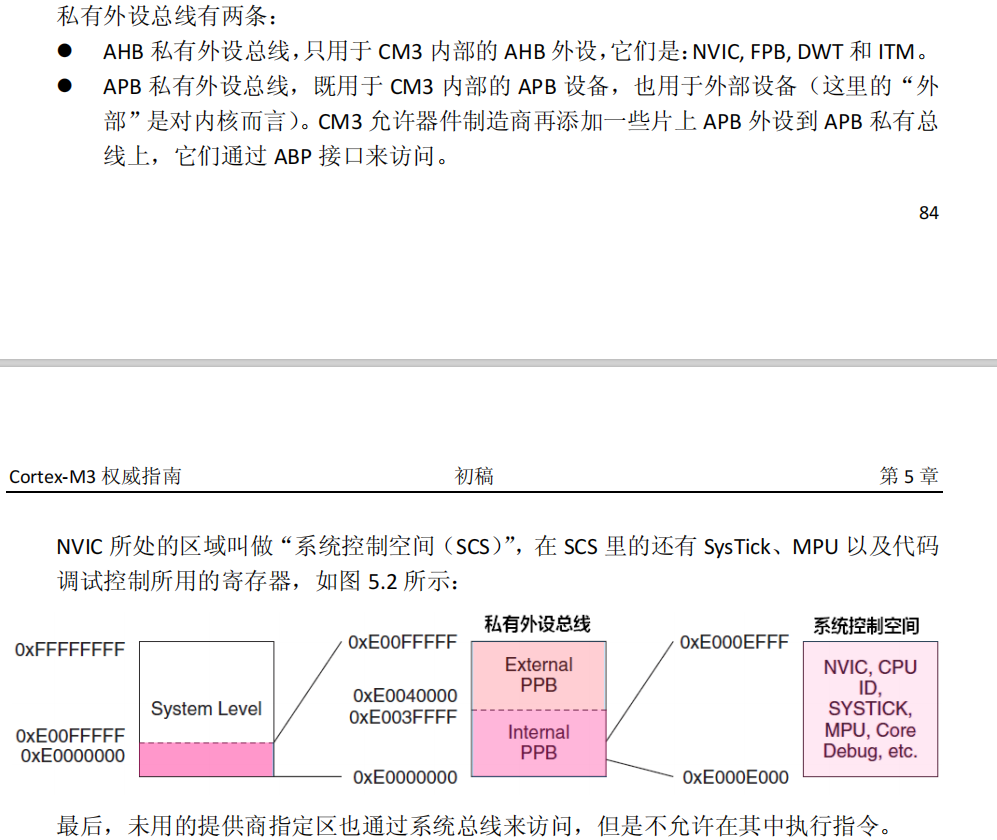
C语言中的位带操作，因为C编译器不直接支持位带操作，比如说C编译器不知道同一块内存能够使用不同的地址访问（这两块地址应该是内核设计时已经安排好了其运作流程）。

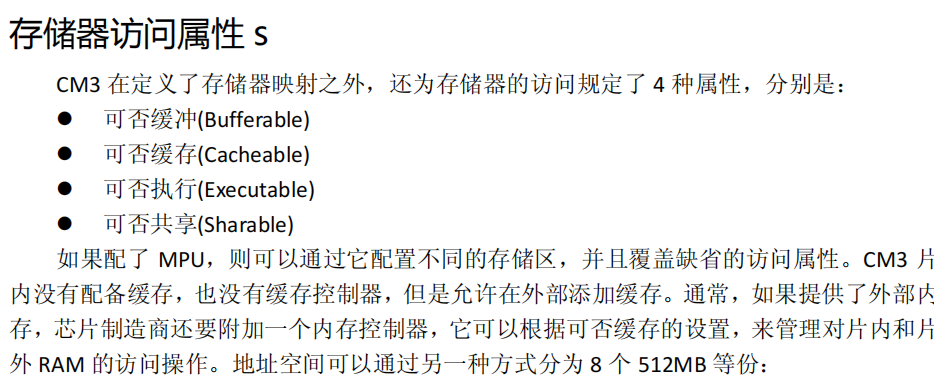




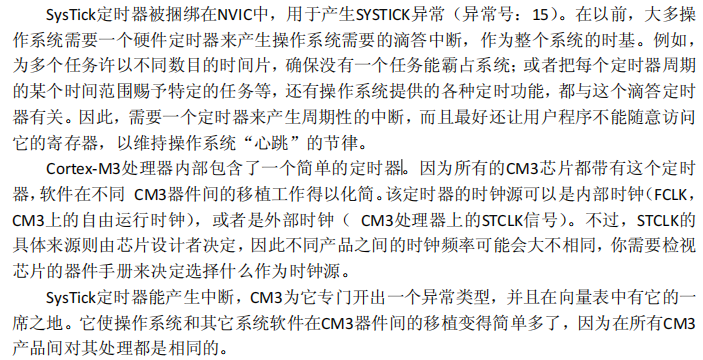


私有外设总线：AHB私有外设总线APB私有外设总线。

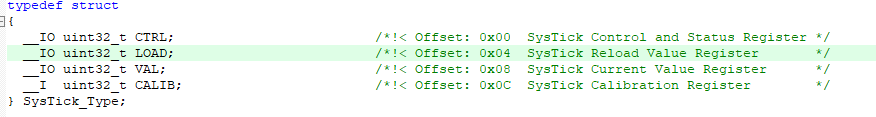




SysTick定时器：



四个寄存器控制SysTick定时器：





可以利用该定时器（倒计时计数到0时产生中断）产生延迟函数：先确定定时器的时钟源，然后计算定时器计时1us需要多少个时钟周期，接着就可以重装载相应的数值去计时。

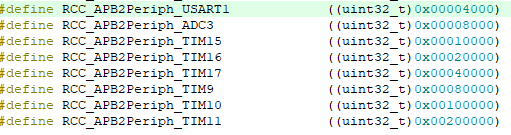
RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1|RCC\_APB2Periph\_GPIOA

|RCC\_APB2Periph\_AFIO, ENABLE); //使能 USART1，GPIOA 时钟

//以及复用功能时钟

之所以可以用|,使能一连串外设，是因为它们都挂在APB2这个总线上，而且它们都对应一个独立地址和它们都只对应一个32位寄存器里面的一位而已。

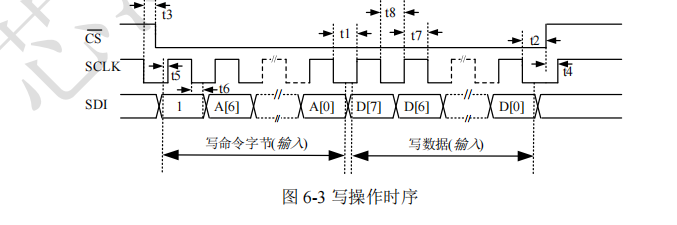
void RCC\_APB2PeriphClockCmd(uint32\_t RCC\_APB2Periph, FunctionalState NewState)



在固件库 V3.5 中，系统在启动的时候会调用 system\_stm32f10x.c 中的函数 SystemInit()对系统时钟进行初始化，在时钟初始化完毕之后会调用 main()函数。 所以我 们不需要再在 main()函数中调用 SystemInit()函数。当然如果有需要重新设置时钟系统，可以写 自己的时钟设置代码，SystemInit()只是将时钟系统初始化为默认状态。

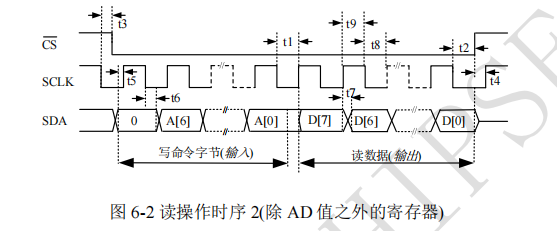
ADC：对模拟信号采样保持以后，再进行量化和编码。这两个过程在转化时同时实现。

1. 分辨率：说明AD对输入信号的分辨能力，及数值部分的精度。一般模拟采样中使用16位还是24位的AD芯片说的就是分辨率.**例如**：输入模拟电压的变化范围为0～3.3 V，输出16位二进制数可以分辨的最小模拟电压为3.3V / 216＝0.05 mV；
2. **转化误差**-表示AD实际电压与理论电压的偏差，一般用最低有效位来表示，单位LSB，通常以相对误差的形式出现，比如相对误差≤±LSB/2，表明实际输出的数字量和理论量误差小于最低位的一半。
3. **转换精度：**
4. **转化时间：输入到稳定输出的时间**
5. 工作电压和基准电压（内部或者外部基准）：工作电压是AD芯片工作的额定电压，关键的是基准电压，又叫参考电压，可以来之芯片内部又或者外部接入，其决定了AD的分辨率
6. 主机向从机写数据，数据传输先高位后低，先准备好数据，等待时钟来就把数据写进去



1. for(index=0;index<8;index++)
2. {
3. if(byteData&0x80) //取最高位先传输
4. {
5. fly\_gpio\_set(CS125X\_PIN\_SDA,Bit\_SET); // SDA 1
6. }
7. else
8. {
9. fly\_gpio\_set(CS125X\_PIN\_SDA,Bit\_RESET); // SDA 0
10. }
11. delay\_us(5);
12. fly\_gpio\_set(CS125X\_PIN\_SCL,Bit\_SET); // SCK 1
13. delay\_us(5);
14. fly\_gpio\_set(CS125X\_PIN\_SCL,Bit\_RESET); // SCK 0
15. byteData<<=1; //取下一位
16. }

主机向从机读数据，先读高位，主机向从机发生好时钟时序再读数据：



for(index=0;index<8;index++)

{

read\_data<<=1;//先取最高位

fly\_gpio\_set(CS125X\_PIN\_SCL,Bit\_SET); // SCK 1

delay\_us(5);

fly\_gpio\_set(CS125X\_PIN\_SCL,Bit\_RESET); // SCK 0

delay\_us(5);

if(HalGpioGet(CS125X\_PIN\_SDA) == TRUE)

{

read\_data |= 0x01;//读数据

}

}

无论是读还是写，先引脚初始化，配置相应的输入输出模式，然后建立好时序，再传输数据。

1. ，所有基准电压一定要稳