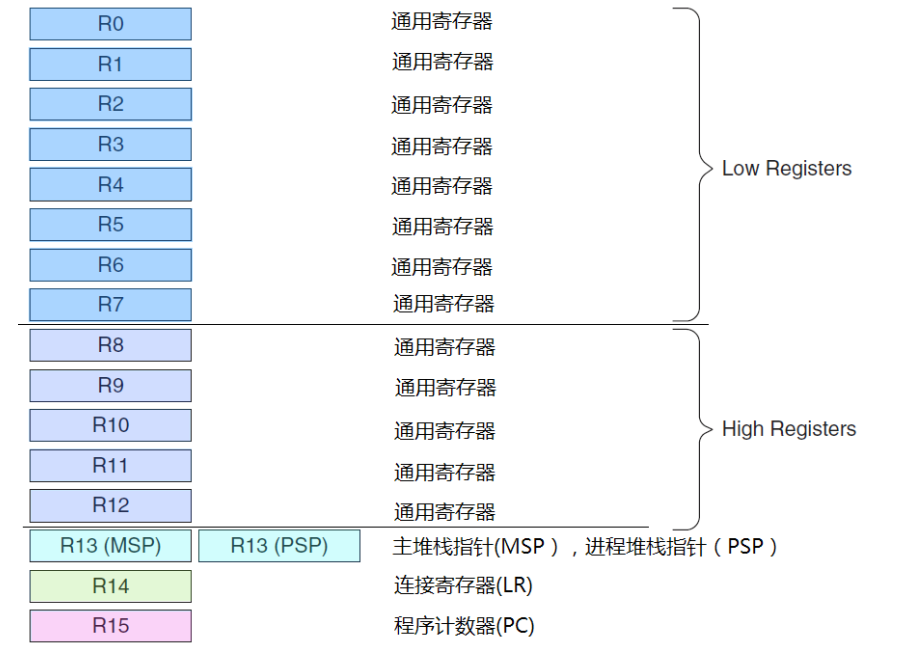
# CH3 CM3基础

## ARM分类：

ARM进一步扩展了它的CPU设计, 内核架构首次从单一款式变成3种款式:

(1) 款式A：设计用于高性能的“开放应用平台”——越来越接近电脑了  
(2) 款式R：用于高端的嵌入式系统，尤其是那些带有实时要求的——又要快又要实时。  
(3) 款式M：用于深度嵌入的，单片机风格的系统中——本书的主角

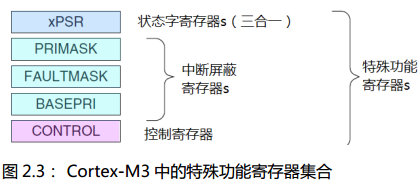


MSP(亦作SP\_main)，这是复位后缺省使用堆栈指针，服务于操作系统内核和异常服务例程；而 PSP(亦作SP\_process)，典型地用于普通的用户线程中。

LR 用于在调用子程序时存储返回地址。

## CM3特殊功能寄存器：

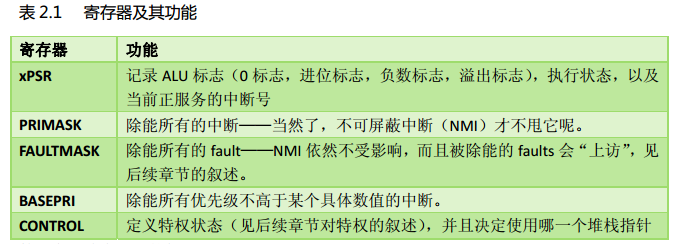
Cortex‐M3 还在内核水平上搭载了若干特殊功能寄存器，包括

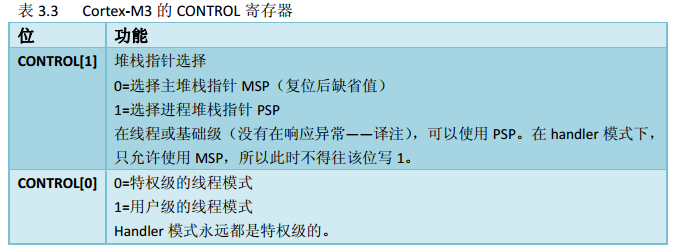


（1）程序状态字寄存器组（ PSRs）

（2）中断屏蔽寄存器组（ PRIMASK, FAULTMASK, BASEPRI，只能在特权级下访问这三个寄存器）

（3）控制寄存器（ CONTROL，只能在特权级下访问这个寄存器，还用于选择当前使用哪个堆栈指针）





※ CONTROL[1]：仅当处于特权级的线程模式下(复位后的状态，见图2.5)，此位才可写，其它场合下禁止写此位。改变处理器的模式也有其它的方式：在异常返回时，通过修改 LR 的位 2，也能实现模式切换；

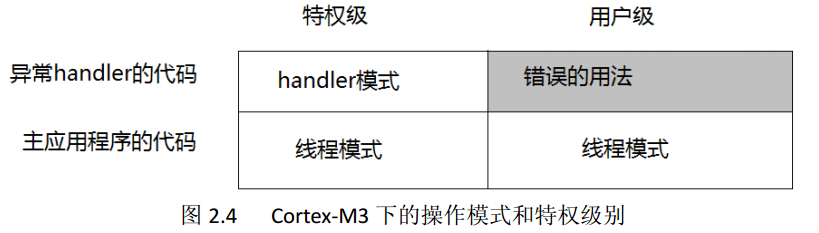
※ CONTROL[0]： 仅当在特权级下操作时才允许写该位。一旦进入了用户级，唯一返回特权级的途径，就是触发一个（软）中断，再由服务例程改写该位;

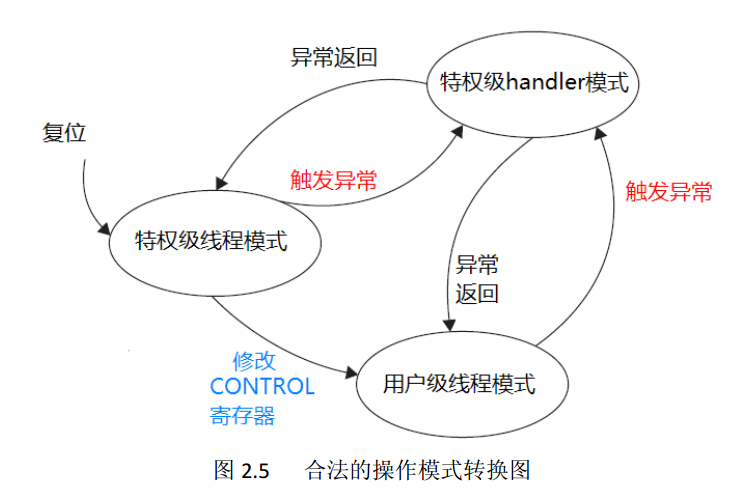
## CM3处理器的模式和级别：

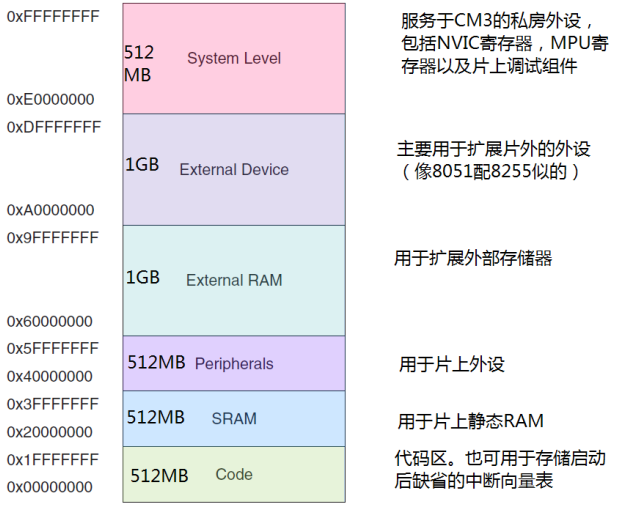
Cortex‐M3 处理器支持两种处理器的操作模式和两级特权操作：

两种操作模式分别为： 处理者模式(handler mode)和线程模式（thread mode），用于区别普通应用程序的代码和异常服务例程的代码（包括中断服务例程的代码）；

Cortex‐M3 的另一个侧面则是特权的分级：特权级和用户级，这可以提供一种存储器访问的保护机制。



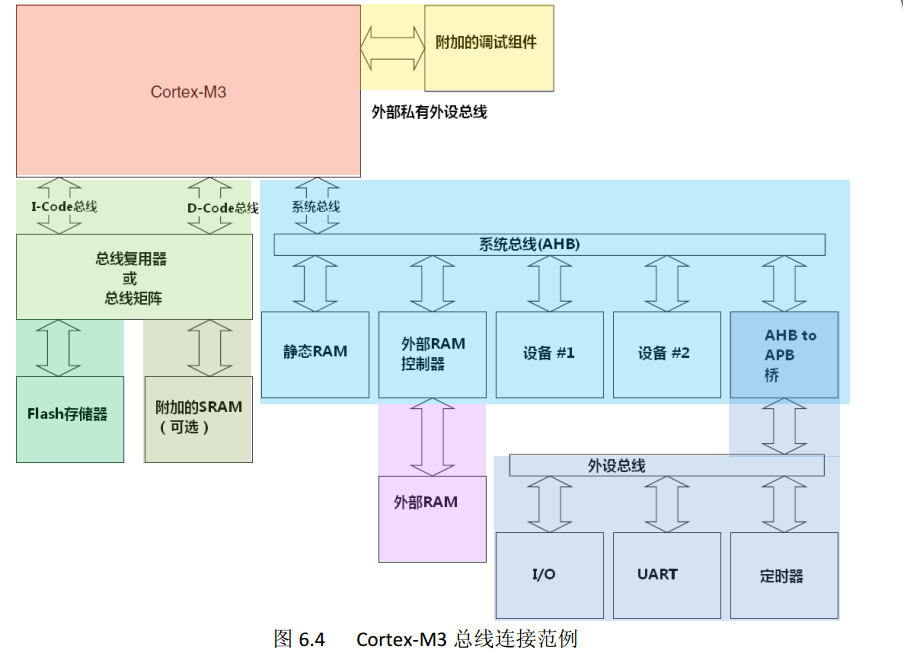




## CM3总线接口：

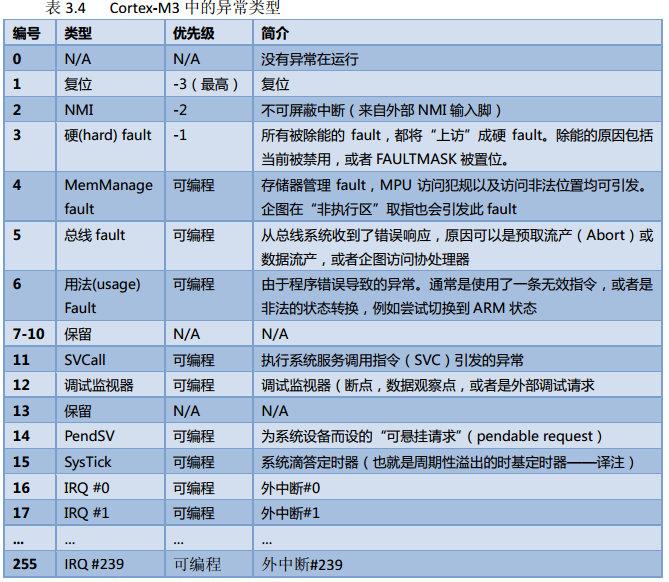
Cortex‐M3 内部有若干个总线接口，以使 CM3 能同时取址和访内（访问内存），它们是：

1. 指令存储区总线（两条, 负责对代码存储区的访问，分别是 I‐Code 总线和 D‐Code， 前者用于取指，后者用于查表等操作）
2. 系统总线( 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 )
3. 私有外设总线 （主要就是访问调试组件，它们也在系统级存储区）



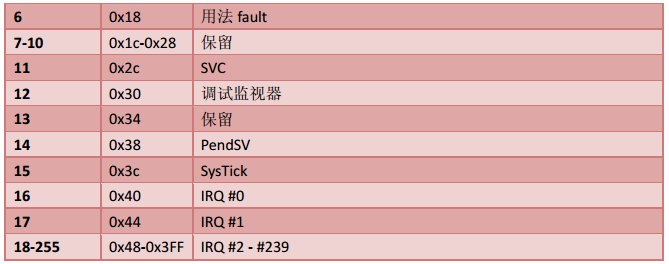
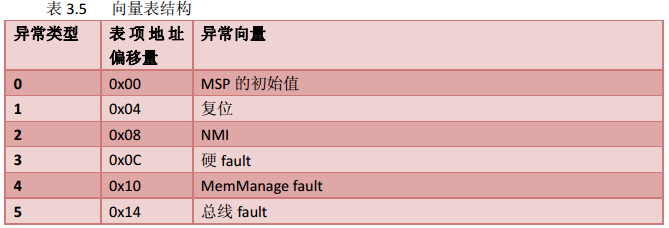
## CM3异常和中断：

Cortex‐M3 支持大量异常，包括 16‐4‐1=11 个系统异常（5个保留），和最多 240 个外部中断（由芯片制造商决定）。由外设产生的中断信号，除了 SysTick 的之外，全都连接到 NVIC 的中断输入信号线。



## 向量表：

当一个发生的异常被 CM3 内核接受，对应的异常 handler 就会执行。为了决定 handler 的入口地址， CM3 使用了“向量表查表机制”。



* 向量表其实是一个 WORD（ 32 位整数）数组，每个下标对应一种异常，该下标元素的值则是该异常 handler 的入口地址；
* 向量表的存储位置是可以设置的，通过 NVIC 中的一个重定位寄存器来指出向量表的地址；

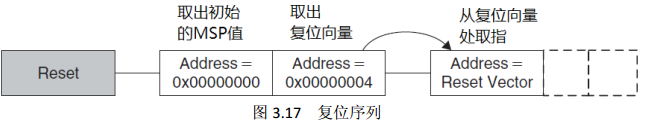
## 栈内存操作：

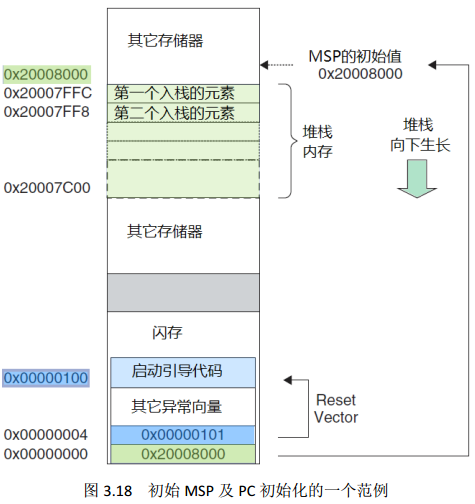
笼统地讲，堆栈操作就是对内存的读写操作，但是其地址由 SP 给出。堆栈的功能就是把寄存器的数据放入内存，以便将来能恢复之——当一个任务或一段子程序执行完毕后恢复。（正常情况下， PUSH 与 POP 必须成对使用，而且参与的寄存器，不论是身份还是先后顺序都必须完全一致。当 PUSH/POP 指令执行时， SP 指针的值也根着自减/自增）

## 复位序列：

在离开复位状态后， CM3 做的第一件事就是读取下列两个 32 位整数的值：

1. 从地址 0x0000,0000 处取出 MSP 的初始值。
2. 从地址 0x0000,0004 处取出 PC 的初始值——这个值是复位向量， LSB 必须是 1（表明CM3是在Thumb态下执行）。然后从这个值所对应的地址处取指。



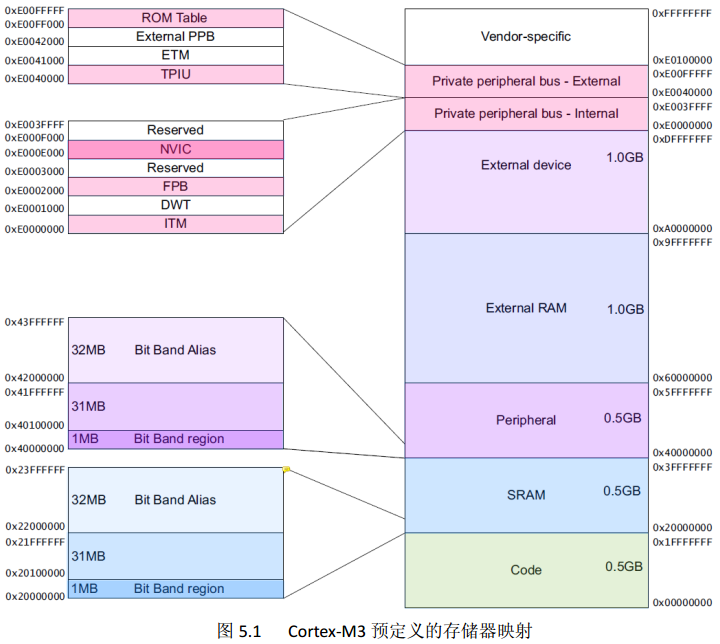


# CH5 存储器系统

## 存储器映射：

CM3 的存储器系统与从传统 ARM 架构的相比，有了如下变化：

1. 它的存储器映射是预定义的， 并且还规定好了哪个位置使用哪条总线;
2. CM3 的存储器系统支持所谓的“位带”(bit‐band)操作。通过它，实现了对单一比特的原子操作(位带操作只适用于数据访问，不适用于取指);
3. CM3 的存储器系统支持非对齐访问和互斥访问;



## 存储器访问属性:

CM3 在定义了存储器映射之外，还为存储器的访问规定了 4 种属性:

(1) 可否缓冲(Bufferable);

(2) 可否缓存(Cacheable);

(3) 可否执行(Executable);

(4) 可否共享(Sharable);

地址空间可以通过另一种方式分为 8 个 512MB 等份：

1. 代码区（ 0x0000\_0000‐ 0x1FFF\_FFFF）: 该区可执行指令，不可缓存, 写操作缓冲;
2. SRAM 区（ 0x2000\_0000 – 0x3FFF\_FFFF）: 该区可执行指令, 写操作缓冲;
3. 片上外设区( 0x4000\_0000 – 0x5FFF\_FFFF ) : 该区不可执行指令，不可缓存；
4. 外部 RAM 区的前半段（ 0x6000\_0000 ‐ 0x7FFF\_FFFF）: 该区是可执行指令, 可缓存;
5. 外部 RAM 区的后半段（ 0x8000\_0000 – 0x9F FF\_FFFF）: 该区是可执行指令, 不可缓存;
6. 外部外设区的前半段( 0xA000\_0000 – 0xBFFF\_FFFF ) : 用于片外外设的寄存器, 也用于多核系统中的共享内存, 该区不可缓冲, 不可执行指令。
7. 外部外设区的后半段( 0xC000\_0000 – 0xDFFF\_FFFF )：目前与前半段的功能完全一致。
8. 系统区( 0xE000\_0000 – 0xFFFF\_FFFF ): 此区是私有外设和供应商指定功能区。此区不可执行代码, 系统区涉及到很多关键部位，因此访问都是严格序列化的（不可缓存，不可缓冲）。而供应商指定功能区则是可以缓存和缓冲的



## 位带操作：

使用普通的加载/存储指令来对单一的比特进行读写。在 CM3中，有两个区中实现了位带。其中一个是 SRAM 区的最低 1MB 范围，第二个则是片内外设区的最低 1MB 范围。位带操作的优越性体现在了可以直接设置GPIO的管脚来控制LED灯的点亮和熄灭，位带操作还能用来简化跳转的判断，如下是新旧操作的对比：

* 读取整个寄存器；
* 掩蔽不需要的位；
* 比较并跳转；

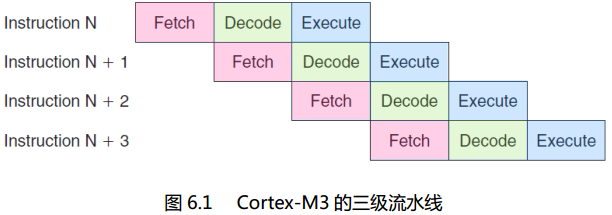
而现在只需要：

* 从位带别名区读取状态位；
* 比较并跳转；

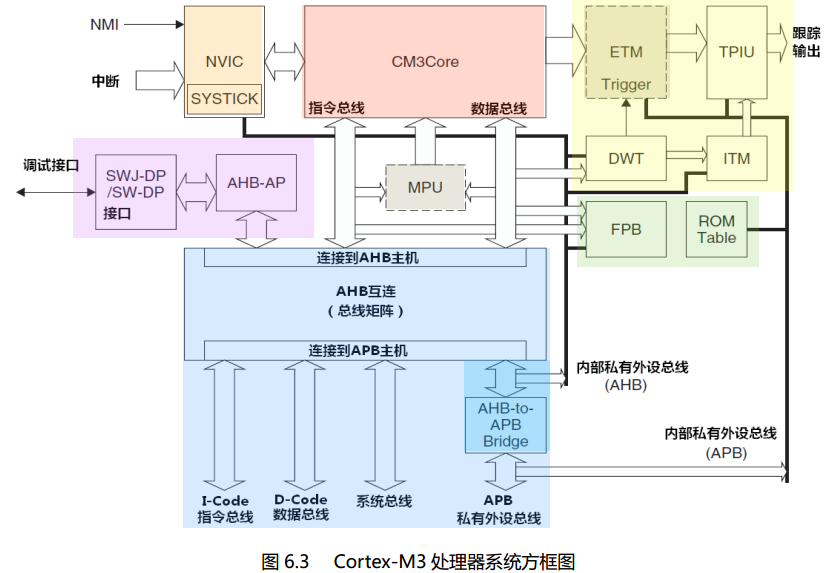
# CH6 实现Cortex-M3的整体风景

## 流水线：

Cortex‐M3 处理器使用一个 3 级流水线。 流水线的 3 级分别是： 取指， 解码和执行。

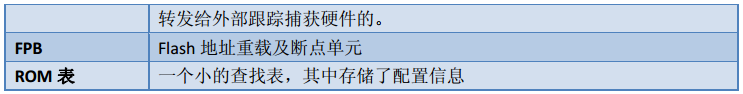
****

## 详细的框图：



虚线框住的 MPU 和 ETM 是可选组件，不一定会包含在每一个 CM3 的 MCU 中。





Cortex‐M3 处理器是以一个“处理器子系统”呈现的，其 CPU 内核本身与 NVIC和一系列调试块都亲密耦合：

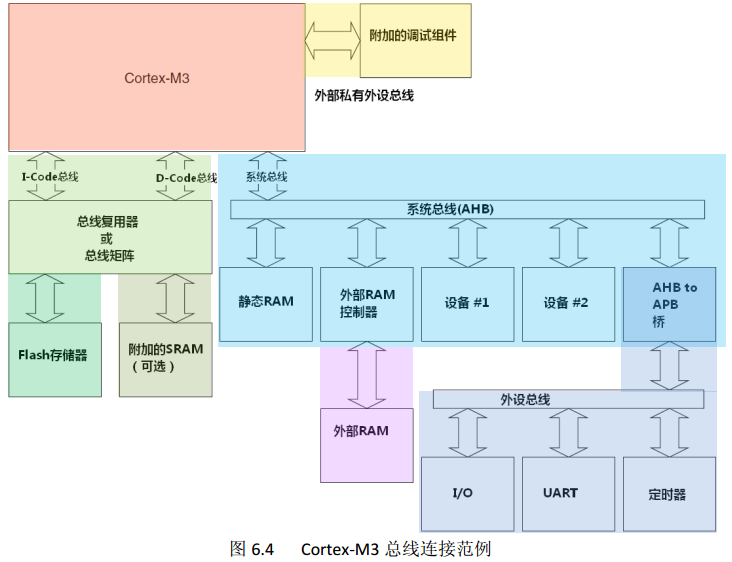
1. CM3Core: Cortex‐M3 处理器的中央处理核心;
2. NVIC(嵌套向量中断控制器): CM3 中内建的中断控制器, 当中断发生时，它会自动取出对应的服务例程入口地址，并直接调用，无需软件判定中断源，缩短了中断延时。
3. SysTick 定时器: 系统滴答定时器是一个非常基本的倒计时定时器，用于在每隔一定的时间产生一个中断，即使是系统在睡眠模式下也能工作。
4. MPU(存储器保护单元): MPU 是一个选配的单元，有些 CM3 芯片可能没有配备此组件。它可以把存储器分成一些 regions，并分别予以保护。(让某些 regions 在用户级下变成只读，从而阻止了一些用户程序破坏关键数据)。
5. BusMatrix: BusMatrix 是 CM3 内部总线系统的核心。它是一个 AHB 互连的网络，通过它可以让数据在不同的总线之间并行传送——只要两个总线主机不试图访问同一块内存区域。
6. AHB to APB: 它是一个总线桥，用于把若干个 APB 设备连接到 CM3 处理器的私有外设总线上（内部的和外部的）。

## Cortex-M3 的总线接口：

1. I-Code 总线: I‐Code 总线是一条基于 AHB‐Lite 总线协议的 32 位总线，负责在 0x0000\_0000 –0x1FFF\_FFFF 之间的取指操作。
2. D-Code 总线: D‐Code 总线也是一条基于 AHB‐Lite 总线协议的 32 位总线，负责在 0x0000\_0000 –0x1FFF\_FFFF 之间的数据访问操作。
3. 系统总线: 系统总线也是一条基于 AHB‐Lite 总线协议的 32 位总线，负责在 0x2000\_0000 –0xDFFF\_FFFF 和 0xE010\_0000 – 0xFFFF\_FFFF 之间的所有数据传送，取指和数据访问都算上。和 D‐Code 总线一样，所有的数据传送都是对齐的。
4. 外部私有外设总线: 系统总线也是一条基于 AHB‐Lite 总线协议的 32 位总线，负责在 0x2000\_0000 –0xDFFF\_FFFF 和 0xE010\_0000 – 0xFFFF\_FFFF 之间的所有数据传送，取指和数据访问都算上。和 D‐Code 总线一样，所有的数据传送都是对齐的。
5. 调试访问端口总线: 调试访问端口总线接口是一条基于“增强型 APB 规格”的 32 位总线，它专用于挂接调试接口，例如 SWJ‐DP 和 SW‐DP。
6. Cortex-M3 的其它接口: 除了总线接口之外， CM3 还有若干个用于其它目的的接口，这些接口的信号都不大可能会引出到引脚上，而只用于连接 SoC 不同的部分，或者干脆就没有使用。



## CM3总线典型的连接方式：



**CM3中断／异常的响应序列：**

当CM3开始响应一个中断时，会在它看不见的体内奔涌起三股暗流：

入栈 🡺 取向量 🡺 更新寄存器

(1) 入栈： 把8个寄存器的值压入栈

(2) 取向量：从向量表中找出对应的服务程序入口地址

(3) 更新寄存器 : 选择堆栈指针MSP/PSP，更新堆栈指针SP，更新连接寄存器LR，更新程序计数器PC



表 3.3 Cortex‐M3 的 CONTROL 寄存器

位 功能

CONTROL[1] 堆栈指针选择

0=选择主堆栈指针 MSP（复位后缺省值）

1=选择进程堆栈指针 PSP

在线程或基础级（没有在响应异常——译注），可以使用 PSP。在 handler 模式下，

只允许使用 MSP，所以此时不得往该位写 1。

CONTROL[0]

0=特权级的线程模式

1=用户级的线程模式

Handler 模式永远都是特权级的

程序状态寄存器（PSRs 或曰 PSR）

程序状态寄存器在其内部又被分为三个子状态寄存器：

z 应用程序 PSR（ APSR）

z 中断号 PSR（ IPSR）

z 执行 PSR（ EPSR）

===============================================================

表 4.2 16 位数据操作指令

名字 功能

ADC 带进位加法

ADD 加法

AND 按位与（原文为逻辑与，有误——译注）。这里的按位与和 C 的”&”功能相同

ASR 算术右移

BIC 按位清 0（把一个数跟另一个无符号数的反码按位与）

CMN 负向比较（把一个数跟另一个数据的二进制补码相比较）

CMP 比较（比较两个数并且更新标志）

CPY 把一个寄存器的值拷贝到另一个寄存器中

EOR 近位异或

LSL 逻辑左移（如无其它说明，所有移位操作都可以一次移动多格——译注）

LSR 逻辑右移

MOV 寄存器加载数据，既能用于寄存器间的传输，也能用于加载立即数

MUL 乘法

MVN 加载一个数的 NOT 值（取到逻辑反的值）

NEG 取二进制补码

ORR 按位或（原文为逻辑或，有误——译注）

ROR 圆圈右移

SBC 带借位的减法

SUB 减法

TST 测试（执行按位与操作，并且根据结果更新 Z）

REV 在一个 32 位寄存器中反转字节序

REVH 把一个 32 位寄存器分成两个 16 位数，在每个 16 位数中反转字节序

REVSH 把一个 32 位寄存器的低 16 位半字进行字节反转，然后带符号扩展到 32 位

SXTB 带符号扩展一个字节到 32 位

SXTH 带符号扩展一个半字到 32 位

UXTB 无符号扩展一个字节到 32 位

UXTH 无符号扩展一个半字到 32 位

表 4.3 16 位转移指令

名字 功能

B 无条件转移

B<cond> 条件转移

BL 转移并连接。用于呼叫一个子程序，返回地址被存储在 LR 中

CBZ 比较，如果结果为 0 就转移（只能跳到后面的指令——译注）

CBNZ 比较，如果结果非 0 就转移（只能跳到后面的指令——译注）

IT If‐Then

表 4.4 16 位存储器数据传送指令

名字 功能

LDR 从存储器中加载字到一个寄存器中

LDRH 从存储器中加载半字到一个寄存器中

LDRB 从存储器中加载字节到一个寄存器中

LDRSH 从存储器中加载半字，再经过带符号扩展后存储一个寄存器中

LDRSB 从存储器中加载字节，再经过带符号扩展后存储一个寄存器中

STR 把一个寄存器按字存储到存储器中

STRH 把一个寄存器存器的低半字存储到存储器中

STRB 把一个寄存器的低字节存储到存储器中

LDMIA 加载多个字，并且在加载后自增基址寄存器

STMIA 加载多个字，并且在加载后自增基址寄存器

PUSH 压入多个寄存器到栈中

POP 从栈中弹出多个值到寄存器中

表 4.5 其它 16 位指令

名字 功能

SVC 系统服务调用

BKPT 断点指令。如果调试被使能，则进入调试状态（停机）。或者如果调试监视器异常被

使能，则调用一个调试异常，否则调用一个 fault 异常

NOP 无操作

CPSIE 使能 PRIMASK(CPSIE i)/ FAULTMASK(CPSIE f)——清 0 相应的位

CPSID 除能 PRIMASK(CPSID i)/ FAULTMASK(CPSID f)——置位相应的位

表 4.6 32 位数据操作指令

名字 功能

ADC 带进位加法

ADD 加法

ADDW 宽加法（可以加 12 位立即数）

AND 按位与（原文是逻辑与，有误——译注）

ASR 算术右移

BIC 位清零（把一个数按位取反后，与另一个数逻辑与）

BFC 位段清零

BFI 位段插入

CMN 负向比较（把一个数和另一个数的二进制补码比较，并更新标志位）

CMP 比较两个数并更新标志位

CLZ 计算前导零的数目

EOR 按位异或

LSL 逻辑左移

LSR 逻辑右移

MLA 乘加

MLS 乘减

MOVW 把 16 位立即数放到寄存器的底 16 位， 高 16 位清 0

MOV 加载 16 位立即数到寄存器（其实汇编器会产生 MOVW——译注）

MOVT 把 16 位立即数放到寄存器的高 16 位， 低 16 位不影响

MVN 移动一个数的补码

MUL 乘法

ORR 按位或（原文为逻辑或，有误——译注）

ORN 把源操作数按位取反后，再执行按位或（原文为逻辑或，有误——译注）

RBIT 位反转（把一个 32 位整数先用 2 进制表达，再旋转 180 度——译注）

REV 对一个 32 位整数做按字节反转

REVH/REV16 对一个 32 位整数的高低半字都执行字节反转

REVSH 对一个 32 位整数的低半字执行字节反转，再带符号扩展成 32 位数

ROR 圆圈右移

RRX 带进位的逻辑右移一格（最高位用 C 填充，且不影响 C 的值——译注）

SFBX 从一个 32 位整数中提取任意的位段，并且带符号扩展成 32 位整数

SDIV 带符号除法

SMLAL 带符号长乘加（两个带符号的 32 位整数相乘得到 64 位的带符号积，再把积加到另一个带符号 64 位整数中）

SMULL 带符号长乘法（两个带符号的 32 位整数相乘得到 64 位的带符号积）

SSAT 带符号的饱和运算

SBC 带借位的减法

SUB 减法

SUBW 宽减法，可以减 12 位立即数

SXTB 字节带符号扩展到 32 位数

TEQ 测试是否相等（对两个数执行异或，更新标志但不存储结果）

TST 测试（对两个数执行按位与，更新 Z 标志但不存储结果）

UBFX 无符号位段提取

UDIV 无符号除法

UMLAL 无符号长乘加（两个无符号的 32 位整数相乘得到 64 位的无符号积，再把积加到另一

个无符号 64 位整数中）

UMULL 无符号长乘法（两个无符号的 32 位整数相乘得到 64 位的无符号积）

USAT 无符号饱和操作（但是源操作数是带符号的——译注）

UXTB 字节被无符号扩展到 32 位（高 24 位清 0——译注）

UXTH 半字被无符号扩展到 32 位（高 16 位清 0——译注）

表 4.7 32 位存储器数据传送指令

名字 功能

LDR 加载字到寄存器

LDRB 加载字节到寄存器

LDRH 加载半字到寄存器

LDRSH 加载半字到寄存器，再带符号扩展到 32 位

LDM 从一片连续的地址空间中加载多个字到若干寄存器

LDRD 从连续的地址空间加载双字（ 64 位整数）到 2 个寄存器

STR 存储寄存器中的字

STRB 存储寄存器中的低字节

STRH 存储寄存器中的低半字

STM 存储若干寄存器中的字到一片连续的地址空间中

STRD 存储 2 个寄存器组成的双字到连续的地址空间中

PUSH 把若干寄存器的值压入堆栈中

POP 从堆栈中弹出若干的寄存器的值

表 4.8 32 位转移指令

名字 功能

B 无条件转移

BL 转移并连接（呼叫子程序）

TBB 以字节为单位的查表转移。从一个字节数组中选一个 8 位前向跳转地址并转移

TBH 以半字为单位的查表转移。从一个半字数组中选一个 16 位前向跳转的地址并转移

表 4.9 其它 32 位指令

LDREX 加载字到寄存器，并且在内核中标明一段地址进入了互斥访问状态

LDREXH 加载半字到寄存器，并且在内核中标明一段地址进入了互斥访问状态

LDREXB 加载字节到寄存器，并且在内核中标明一段地址进入了互斥访问状态

STREX 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的字

STREXH 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的半字

STREXB 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的字节

CLREX 在本地的处理上清除互斥访问状态的标记（先前由 LDREX/LDREXH/LDREXB 做的标记）

MRS 加载特殊功能寄存器的值到通用寄存器

MSR 存储通用寄存器的值到特殊功能寄存器

NOP 无操作

SEV 发送事件

WFE 休眠并且在发生事件时被唤醒

WFI 休眠并且在发生中断时被唤醒

ISB 指令同步隔离（与流水线和 MPU 等有关——译注）

DSB 数据同步隔离（与流水线、 MPU 和 cache 等有关——译注）

DMB 数据存储隔离（与流水线、 MPU 和 cache 等有关——译注）

0xE0000000 至 0xE00FFFFF

