10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象
华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《CORTEX-M3+UCOS-II 嵌入式系统开发入门与应用》

作者：华清远见



# 第 2 章 Cortex-M3 处理器体系结构

本章简介

Cortex-M3 是一款 32 位的低功耗处理器，它的出现使 ARM 成功进入单片机市场。由于它采用了最新的 ARM v7-M 架构，使其门数更少，性能却更强。另外，Cortex-M3 采用了 Tail-Chaining 中断技术，完全基于硬件进行中断处理，最多可减少 12 个时钟周期数，在实际应用中可减少 70%中断延时时间。相信用不了多久，Cortex-M3 就会在 32 位嵌入式处理器市场中脱颖而出。

本章节将详细介绍 Cortex-M3 的体系结构。

## 2.1 Cortex-M3 综述

Cortex-M3 是一款低功耗处理器，具有门数目少，中断延迟短，调试成本低的特点，是为要求有快速中断响应能力的深度嵌入式应用而设计的。表 2.1 所示为该款处理器所包含的组件及其特点。

表 2.1 Cortex-M3 所包含组件及特点

| 包含组件 | 组件特点 |
|---|---|
| 处理器内核 | ARM v7-M 体系结构：Thumb-2 ISA 指令子集，包含所有基本的 16 位和 32 位 Thumb-2 指令（用于多媒体，SIMD 和加强的 DSP 指令除外） |
| | 具有分组的堆栈指针寄存器 SP |
| | 包含硬件除法指令：SDIV 和 UDIV 指令 |
| | 两种工作模式：处理模式（handler mode）和线程模式（thread mode） |
| | 两种处理器状态：Thumb 状态和调试状态 |
| | 具有可中断-可继续（interruptible-continued）的 LDM/STM 和 PUSH/POP 指令，实现低中断延迟 |
| | 自动保存和恢复处理器状态，可以减少进入和退出中断服务程序（ISR）的系统时间 |
| | 支持 ARM v6 架构的 BE8/LE 和非对齐访问 |

续表

| 包含组件 | 组件特点 |
|---|---|
| 中断向量控制器 NVIC | 外部中断可配置为 1～240 个 |
| | 优先级位可配置为 3～8 位 |
| | 中断优先级可动态地重新配置 |
| | 优先级分组：分为占先中断等级和非占先中断等级 |
| | 支持末尾连锁（tail-chaining）和迟来（late arrival）中断处理机制。这样，在两个中断之间使能背对背中断（back-to-back interrupt）处理，减少中断延时 |
| | 处理器状态在进入中断时自动保存，中断退出时自动恢复，不需要多余的指令 |
| 存储器保护单元 MPU | 8 个存储器区 |
| | 子区禁止功能（SRD），实现对存储器区的有效使用 |
| | 可使能背景区，执行默认的存储器映射属性 |
| 总线接口 | 指令存储区总线：负责对代码存储区的访问 |
| | 系统总线：用于访问内存和外设 |
| | 私有外设总线：负责一部分私有外设的访问，主要包括调试组件的访问 |
| 低成本调试解决方案 | 基于 ARM 全新的 CoreSight 架构,使得当内核在运行、中止、甚至处于复位状态时，通过调试系统,可以对系统中包括 Cortex-M3 寄存器组在内的所有存储器和寄存器进行调试访问 |
| | 使用“调试访问接口（DAP）”代替 JTAG 接口 |
| | 具有 Flash 修补和断点单元（FPB），实现断点和代码修补 |
| | 数据观察点和触发单元（DWT），实现观察点、触发资源和系统分析（systemprofiling） |
| | 仪表跟踪宏单元（ITM），支持对 printf 类型的调试 |
| | 跟踪端口的接口单元（TPIU），用来连接跟踪端口分析仪 |
| | 可选的嵌入式跟踪宏单元（ETM），实现指令跟踪 |

## 2.2 Cortex-M3 编程模式

华清远见

Cortex-M3 处理器采用 ARM v7-M 架构，包括所有的 16 位 Thumb 指令集和基本的 32 位 Thumb-2 指令集。Cortex-M3 只支持 Thumb 指令，不能执行 ARM 指令。

Thumb 指令集是 ARM 指令集的子集，重新被编码为 16 位。它支持较高的代码密度以及 16 位或小于 16 位的存储器数据总线系统。

Thumb-2 在 Thumb 指令集架构（ISA）上进行了大量的改进，它与 Thumb 相比，代码密度更高，并且通过使用 16/32 位指令，提供更高的性能。

### 2.2.1 Cortex-M3 的工作模式和工作状态

Cortex-M3 处理器支持两种工作模式——线程模式和处理（Handler）模式。

- 在复位和异常返回时，处理器进入线程模式。特权级和用户级下的代码均可在线程模式下运行。
- 出现异常时处理器进入处理模式，在处理模式中，只能运行特权级代码。

Cortex-M3 处理器有如下两种工作状态。

- Thumb 状态：这是 16 位和 32 位半字对齐的 Thumb 和 Thumb-2 指令的正常执行状态。
- 调试状态：处理器停机调试时进入该状态。

### 2.2.2 特权访问和用户访问

Cortex-M3 处理器将代码按特权级和用户级分开对待。用户级代码执行时，对一些关键资源的访问受到限制。特权级代码可以访问所有资源。当处理器进入处理模式时，只能运行特权级代码；而线程模式下，用户级代码和特权级代码均可运行。

线程模式在复位之后默认为特权访问，可通过 MSR 指令清零 CONTROL[0]位，将其配置为用户（非特权）访问，只能执行用户级代码。使用用户级代码，具有以下限制。

- 只能运行指令集中的部分指令，例如设置 FAULTMASK 和 PRIMASK 的 CPS 指令，在用户级指令中被禁止。
- 对系统控制空间（SCS）的大部分寄存器的访问是禁止的。

当线程模式从特权访问变为用户访问后，本身不能回到特权访问（即运行特权级代码）。只有当系统发生异常时，处理器才又将以特权级来运行其异常服务程序。异常返回后将回到产生异常之前的特权级继续运行。异常处理程序（例如 SVC）可以通过改变其在退出时使用的 EXC_RETURN 值来改变线程模式使用的堆栈。所有异常均使用主堆栈。堆栈指针 r13 是分组寄存器，在 SP_main 和 SP_process 之间切换。在任何时候，进程堆栈和主堆栈中只有一个是可见的，表示为 r13。

除了使用从处理模式退出时的 EXC_RETURN 的值进行切换外，在线程模式中，使用 MSR 指令对 CONTROL[1]执行写操作也可以从主堆栈切换到进程堆栈。

### 2.2.3 Cortex-M3 的寄存器组织

Cortex-M3 处理器有以下 5 组 32 位寄存器：

- 13 个通用寄存器，r0～r12；
- 分组堆栈指针寄存器 r13，在不同模式下表示为 SP_Process 和 SP_Main；
- 链接寄存器，r14；
- 程序计数器，r15；
- 1 个程序状态寄存器，xPSR。

图 2.1 所示为 Cortex-M3 的寄存器集。

华清远见

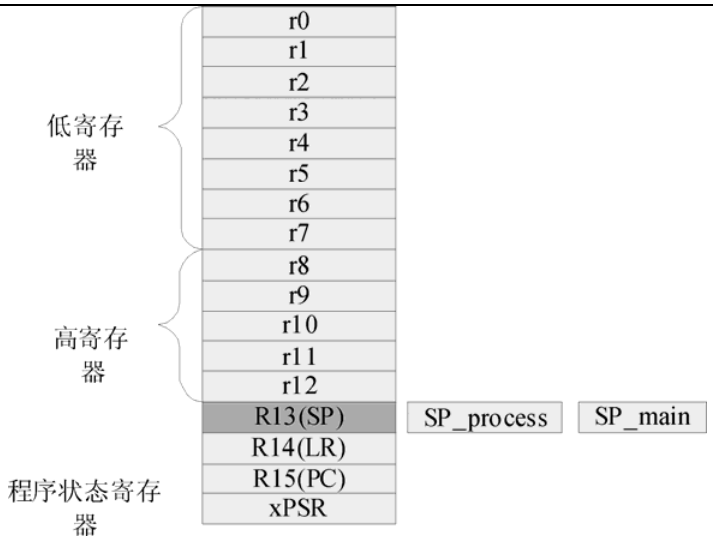



图 2.1　Cortex-M3 的寄存器集

通用寄存器 r0～r12 没有在结构上定义特殊的用法。大多数指定通用寄存器的指令都能够使用 r0～r12。通用寄存器中，r0～r7 被称为“低寄存器”，可以被指定通用寄存器的所有指令访问；r8～r12 被称为“高寄存器”，只能被指定通用寄存器的所有 32 位指令访问，而不能被 16 位指令访问。

寄存器 r13 用作堆栈指针（SP）。由于 SP 忽略了写入值的 bit[1:0]位，因此它自动实现 4 字节边界对齐。处理模式始终使用 SP_main，而线程模式可配置为 SP_main 或 SP_process。

寄存器 r14 是子程序的链接寄存器（LR）。在执行分支（branch）和链接（BL）指令或带有交换的分支和链接指令（BLX）时，LR 用于接收来自 PC 的返回地址。LR 也用于异常返回。其他任何时候都可以将 r14 看作一个通用寄存器。

寄存器 r15 为程序计数器（PC），该寄存器的位 0 始终为 0，因此，指令始终与字或半字边界对齐。

特殊用途程序状态寄存器 xPSR 被分为 3 个：应用状态寄存器 APSR、中断状态寄存器 IPSR 和执行程序状态寄存器 EPSR。它们分别对应处理器的应用、中断、执行 3 种状态。

华清远见

## 2.2.4　Cortex-M3 的数据类型

Cortex-M3 处理器支持以下数据类型：

- 32 位字类型；
- 16 位半字类型；
- 8 位字节类型。

## 2.2.5　Cortex-M3 的存储器格式

Cortex-M3 处理器将存储器看作从 0 开始向上编号的字节的线性集合。例如：

- 字节 0～3 存放第一个被保存的字；
- 字节 4～7 存放第二个被保存的字。

Cortex-M3 处理器能够以“小端格式（little-endian）”或“大端格式（big-endian）”访问存储器中的数据，而访问代码时始终使用小端格式。小端格式是 ARM 处理器默认的存储器格式。在小端格式中，一个字中最低地址的字节为该字的最低有效字节，最高地址的字节为最高有效字节。存储器系统地址 0 的字节与数据线 7-0 相连。在大端格式中，一个字中最低地址的字节为该字的最高有效字节，而最高地址的字节为最低有效字节。存储器系统地址 0 的字节与数据线 31-24 相连。

图 2.2 所示为小端格式和大端格式的区别。

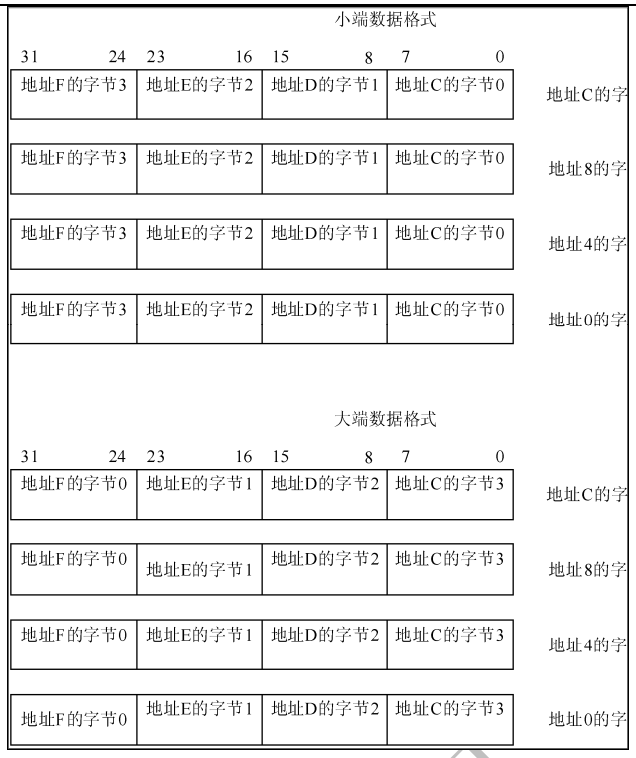


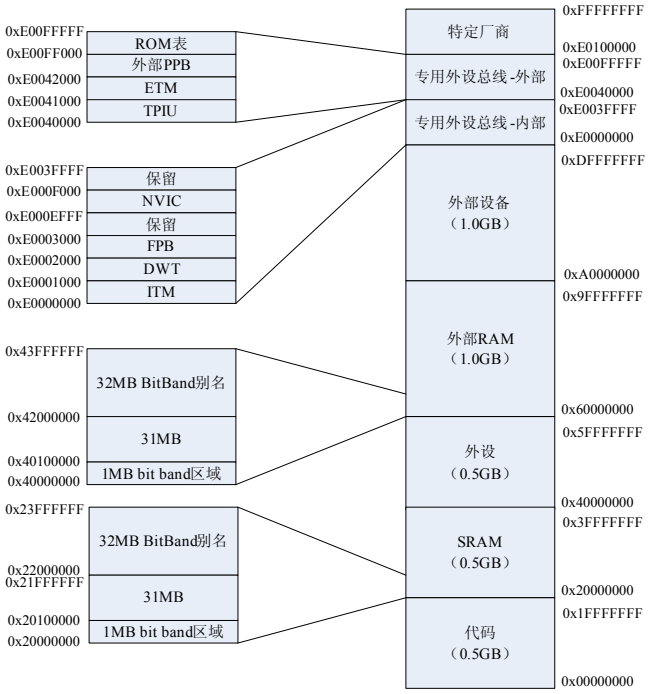

图 2.2　小端格式和大端格式的区别

Cortex-M3 处理器有一个配置管脚 BIGEND，能够配置使用小端格式或 BE-8 大端格式。该管脚在复位时被采样，结束复位后存储器格式不能修改。当系统对系统控制空间（SCS）和 PPB 空间访问时，必须采用小端格式。在非复位的状态下试图改变存储器格式的操作将被忽略。

## 2.3　Cortex-M3 的存储器管理

### 2.3.1　存储器映射

Cortex-M3 具有固定的存储器映射，图 2.3 所示为其映射结构。




华清远见

图 2.3 Cortex-M3 的存储器映射结构

表 2.2 列出了被不同存储器映射区域寻址的处理器接口。

**表 2.2** 存储器接口

| 存储器映射 | 接 口 |
|---|---|
| 代码 | 指令取址在 ICode 总线上执行，数据访问在 DCode 总线上执行 |
| SRAM | 指令取址和数据访问都在系统总线上执行 |
| SRAM_bitband | 别名区域，数据访问是别名。指令访问不是别名 |
| 外设 | 指令取址和数据访问都在系统总线上执行 |
| 外设_bitband | 别名区域，数据访问是别名。指令访问不是别名 |
| 外部 RAM | 指令取址和数据访问都在系统总线上执行 |
| 外部设备 | 指令取址和数据访问都在系统总线上执行 |
| 专用外设总线 | 对 ITM、NVIC、FPB、DWT、MPU 的访问在处理器内部专用外设总线上执行。对 TPIU、ETM 和 PPB 存储器映射的系统区域的访问在外部专用外设总线上执行。该存储区为从不执行（XN），因此指令取址是禁止的。它也不能通过 MPU（如果系统中存在）修改 |
| 系统 | 厂商系统外设的系统部分。该存储区为从不执行（XN），因此指令取址是禁止的。它也不能通过 MPU（如果系统中存在）修改 |

表 2.3 列出了不同存储器区域的用途。

**表 2.3** Cortex-M3 不同存储区的用途

| 名 称 | 区 域 | 区 域 类 型 | XM | 高 速 缓 存 |
|---|---|---|---|---|
| 代码区 | 0x00000000-0x1FFFFFFF | 常规 | - | WT |
| SDAM | 0x20000000-0x3FFFFFFF | 常规 | - | WBWA |
| SRAM_bitband | 0x22000000-0x23FFFFFF | 内部 | - | - |
| 外设 | 0x40000000-0x5FFFFFFF | 设备 | XN | - |
| 外设_bitband | 0x42000000-0x43FFFFFF | 内部 | XN | - |
| 外部 RAM | 0x60000000-0x7FFFFFFF | 常规 | - | WBWA |
| 外部 RAM | 0x80000000-0x9FFFFFFF | 常规 | - | WT |
| 外部设备 | 0xA0000000-0xBFFFFFFF | 设备 | XN | 共用 |
| 专用外设总线 | 0xE0000000-0xE00FFFFF | SO | XN | - |
| 系统 | 0xE0100000-0xFFFFFFFF | 设备 | XN | - |

华清远见

## 2.3.2 Bit-banding 机制

处理器的存储器映射包括两个 Bit-banding 区域，分别为 SRAM 和外设存储区域中的最低的 1MB。这些 Bit-banding 区域将存储器别名区的一个字（Word）映射为 Bit-banding 区的一个位（Bit）。

对应两个 Bit-banding 区域，Cortex-M3 存储器有 2 个 32MB 别名区，分别被映射为两个 1MB 的 Bit-banding 区，其映射关系如下：

- 对 32MB SRAM 别名区的访问映射为对 1MB SRAMBit-banding 区的访问；
- 对 32MB 外设别名区的访问映射为对 1MB 外设 Bit-banding 区的访问。

下面的公式显示了如何将别名区中的字（Word）与 Bit-banding 区中的对应位（Bit）关联。

```
bit_word_offset=(byte_offset×32)+(bit_number×4)
```

```
bit_word_addr=bit_band_base+bit_word_offset
```

其中：

- Bit_word_offset 为 Bit-banding 存储区中的目标位的位置；
- Bit_word_addr 为别名存储区中映射为目标位的字的地址；
- Bit_band_base 为别名区的开始地址；
- Bit_offset 为 Bit-banding 区中包含目标位的字节的编号；
- Bit_number 为目标位的比特位（0～7）。

图 2.4 显示了 SRAM Bit-banding 别名区和 SRAM Bit-banding 区之间的 Bit-banding 映射的例子。

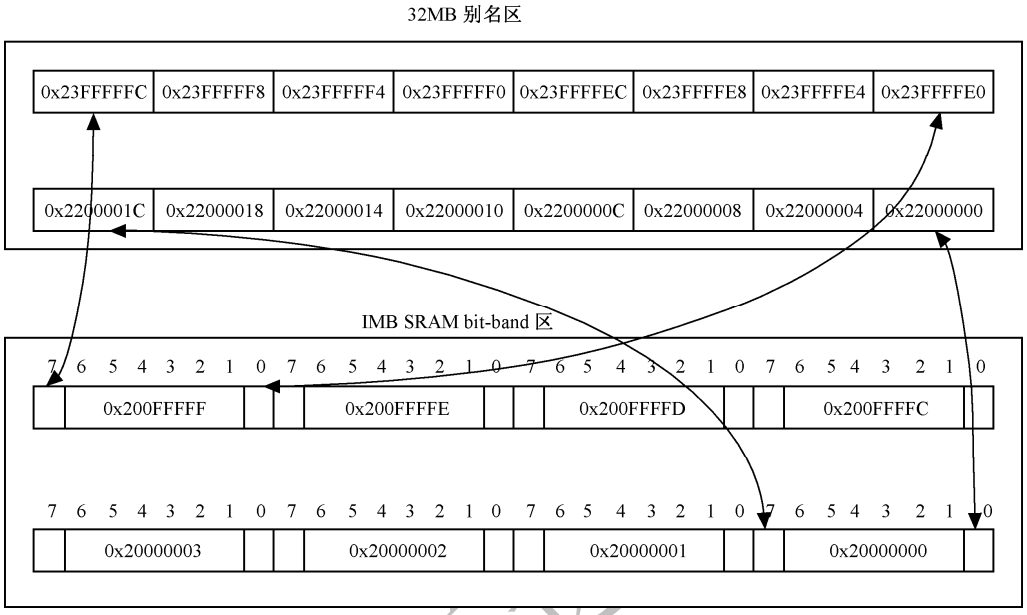



图 2.4　Bit-banding 映射

地址 0x23FFFFE0 的别名字映射为 0x200FFFFC 的 Bit-banding 字节的位 0：

```
0x23FFFFE0=0x22000000+(0xFFFFF*32)+0*4
```

地址 0x23FFFFEC 的别名字映射为 0x200FFFFC 的 bit-band 字节的位 7：

```
0x23FFFFEC=0x22000000+(0xFFFFF*32)+7*4
```

地址 0x22000000 的别名字映射为 0x20000000 的 bit-band 字节的位 0：

```
0x22000000=0x22000000+(0*32)+0*4
```

地址 0x220001C 的别名字映射为 0x20000000 的 bit-band 字节的位 0：

```
0x2200001C=0x22000000+(0*32)+7*4
```

向别名区写入一个字与在 bit-band 区的目标位执行“读-修改-写”操作具有相同的作用。写入别名区的字的 Bit 0 决定了写入 bit-band 区的目标位的值。将 Bit 0 为 1 的值写入别名区表示向 bit-band 位写入 1，将 Bit0 为 0 的值写入别名区，表示向 bit-band 位写入 0。别名字的 Bit[31:1]在 bit-band 位上不起作用，即，写入 0x01 与写入 0xFF 的效果相同；写入 0x00 与写入 0x0E 的效果相同。读别名区的一个字返回 0x01 或 0x00。0x01 表示 bit-band 区中的目标比特位；0x00 表示目标位清零，位[31:1]将为 0。

注：采用大端格式时，对 Bit-banding 别名区的访问必须以字节方式，否则访问结果不可预知。

## 2.4　Cortex-M3 的异常处理

Cortex-M3 在异常处理机制方面有很大的改进，其异常响应时间为 12 个时钟周期。嵌套向量中断控制器（NVIC，Nested Vectored Interrupt Controller）是 CortexM3 处理器的紧耦合部件，它支持 11 种系统异常，外加 240 个外部中断输入。同时，抢占（pre-emption）、尾链（tail-chaining）、迟到（late-arriving）技术的使用，大大缩短了异常事件的响应　　时间。

异常或者中断是处理器响应突发事件的一种机制。当异常发生时，Cortex-M3 通过硬件自动将编程计数器（PC）、编程状态寄存器（xPSR）、链接寄存器（LR）和 R0～R3、R12 等寄存器压栈。在 Dbus（数据总线）保存处理器状态的同时，处理器通过 Ibus（指令总线）从一个可以重新定位的向量表中识别出异常向量，并获取 ISR 函数的地址，也就是保护现场与取异常向量是并行处理的。一旦压栈和取指令完成，中断服务程序或故障处理程序就开始执行。执行完 ISR，硬件进行出栈操作，中断前的程序恢复正常执行。图 2.5 所示为 Cortex-M3 处理器的异常处理流程。

同其他 ARM 芯片相比，Cortex-M3 在异常的分类和优先级上有很大的差别，表 2.4 所示为其支持的异常类型。

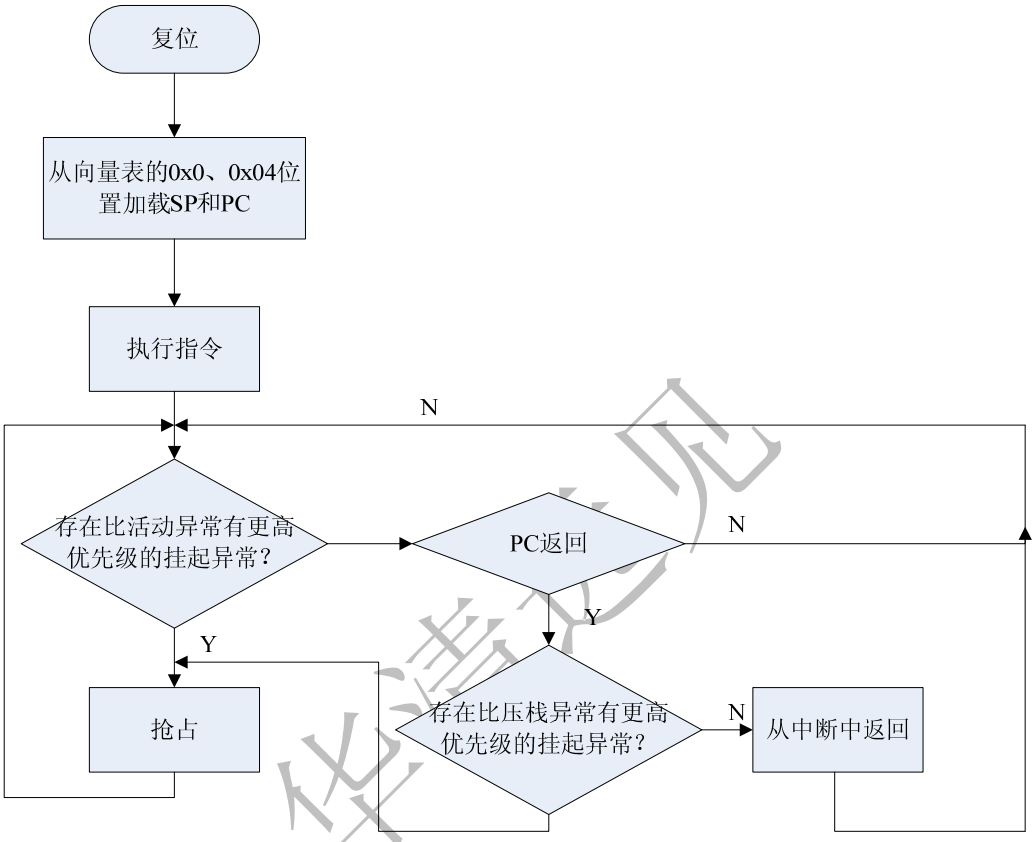



图 2.5　Cortex-M3 异常处理流程

**表 2.4　　Cortex-M3 的异常类型**

| 异 常 类 型 | 位　　置 | 优　先　级 | 描　　述 |
|---|---|---|---|
| - | 0 | - | 在复位时栈顶从向量表的第一个入口加载 |
| 复位 | 1 | −3（最高） | 在上电和热复位（warm reset）时调用；在第一条指令执行时，优先级下降到最低且处理器进入“线程”处理模式 |
| 不可屏蔽的中断 | 2 | −2 | 不能被除复位之外的任何异常停止或抢占，为异步响应异常 |
| 硬故障 | 3 | −1 | 由于优先级的原因不能及时响应的故障或通过配置将故障时能禁止而不能响应的故障，统一归为“硬故障”，该异常为异步响应异常 |
| 存储器管理 | 4 | 可调整 | 存储器管理故障，包括违反访问规范以及数据访问类型不匹配。即使 MPU 被禁止或不存在，也可以用它来支持默认的存储器映射的 XN 区域 |
| 总线故障 | 5 | 可调整 | 预取故障，存储器地址访问故障，以及其他相关的地址/存储故障（预取 Abort 和数据 Abort） |
| 使用故障 | 6 | 可调整 | 使用故障。例如，执行未定义的指令或尝试不合法的状态转换时发生的故障 |

| | | | |
|---|---|---|---|
| - | 7-10 | - | 保留 |
| SVCall | 11 | 可调整 | 利用 SVC 指令调用系统服务 |

续表

| 异常类型 | 位置 | 优先级 | 描述 |
|---|---|---|---|
| 调试监控 | 12 | 可调整 | 调试监控，在处理器没有停止时出现，为同步响应异常，但只有在使能时是有效的。如果它的优先级比当前有效的异常的优先级要低，则不能被激活 |
| - | 13 | - | 保留 |
| PendSV | 14 | 可调整 | 可挂起的系统服务请求。是异步响应的，只能由软件来实现挂起 |
| SysTick | 15 | 可调整 | 系统节拍定时器（tick timer）启动后，为异步响应异常 |
| 外部中断 | 16 及以上 | 可调整 | 在内核的外部产生。INTISR[239:0]，通过 NVIC（设置优先级）输入，均为异步影响中断 |

## 2.4.1 异常优先级

NVIC 支持由软件指定优先级，通过对中断优先级寄存器的 PRI_N 区执行“写”操作，将中断优先级指定为 0～255 中的一级。硬件优先级随着中断号的增加而降低，“0”优先级最高，“255”优先级最低。软件优先级被指定后，硬件优先级无效。如将 INTISR[0]指定优先级为“1”，INTISR[31]指定优先级为“0”，则 INTISR[31]的优先级比 INTISR[0]高。

为了对具有大量中断的系统加强优先级控制，NVIC 支持优先级分组机制。软件可以使用复位控制寄存器中的 PRIGROUP 区来将每个 PRI_N 中的值分为“占先优先级区”和“次优先级区”。占先优先级又被称为“组优先级”。如果有多个挂起异常在同一异常优先级组中，则使用次优先级区来决定同组中的异常的优先级，这就是同组内的次优先级。组优先级和次优先级的结合就是通常所说的“优先级”。如果两个挂起异常具有相同的优先级，则挂起异常的编号越低优先级越高。

表 2.5 显示了如何对 PRIGROUP 执行写操作，并将 8 位 PRI_N 分为占先优先级区（x）和次优先级区（y）。

**表 2.5** 优先级分组

| PRIGROUP[2:0] | 中断优先级区，PRI_N[7：0] | | | | |
|---|---|---|---|---|---|
| | 二进制位置 | 占先区 | 次优先级区 | 占先优先级的数目 | 次优先级的数目 |
| b000 | bxxxxxxx.y | [7:1] | [0] | 128 | 2 |
| b001 | bxxxxxx.yy | [7:2] | [1:0] | 64 | 4 |
| b010 | bxxxxx.yyy | [7:3] | [2:0] | 32 | 8 |
| b011 | bxxxx.yyyy | [7:4] | [3:0] | 16 | 16 |
| b100 | bxxx.yyyyy | [7:5] | [4:0] | 8 | 32 |

续表

| PRIGROUP[2:0] | 中断优先级区，PRI_N[7：0] | | | | |
|---|---|---|---|---|---|
| | 二进制位置 | 占先区 | 次优先级区 | 占先优先级的数目 | 次优先级的数目 |
| b101 | bxx.yyyyyy | [7:6] | [5:0] | 4 | 64 |
| b110 | bx.yyyyyyy | [7] | [6:0] | 2 | 128 |
| b111 | b.yyyyyyyy | 无 | [7:0] | 0 | 256 |

在处理器的异常类型中，优先级决定了处理器如何以及怎样处理异常，系统可以通过软件指定异常优先级，并可以将优先级分组（分为占先优先级和次优先级）。不同的优先级，处理器处理的方式也不一样。表 2.6 列出了系统对不同优先级采取的不同动作。

表 2.6 异常优先级的动作

| 动　作 | 描　述 |
|---|---|
| 占先（Pre-emption Priority） | 占先优先级又被称为“抢占式优先级”。具有高抢占式优先级的中断可以在具有低抢占式优先级的中断处理过程中被响应，即“中断嵌套”。当两个中断源的抢占式优先级相同时，这两个中断将没有嵌套关系，当一个中断到来后，如果正在处理另一个中断，这个后到来的中断就要等到前一个中断处理完之后才能被处理。如果这两个中断同时到达，则中断控制器根据它们的“次优先级”高低来决定先处理哪一个 |
| 末尾连锁（Tail-chain） | 末尾连锁（Tail-chain）是处理器用来加速中断响应的一种机制。在结束 ISR 时，如果存在一个挂起中断，其优先级高于正在返回的 ISR 或线程，那么就会跳过出栈操作，转而将控制权让给新的 ISR |
| 返回 | 在没有挂起（pending）异常或没有比被压栈的 ISR 优先级更高的挂起异常时，处理器执行出栈操作，并返回到被压栈的 ISR 或线程模式。在响应 ISR 之后，处理器通过出栈操作自动将处理器状态恢复为进入 ISR 之前的状态。如果在状态恢复过程中出现一个新的中断，并且该中断的优先级比正在返回的 ISR 或线程更高，则处理器放弃状态恢复操作并将新的中断作为 tail-chain 来处理 |
| 迟来 | 迟来是处理器用来加速占先的一种机制。如果在保存前一个占先的状态时出现一个优先级更高的中断，则处理器转去处理优先级更高的中断，开始该中断的取向量操作。状态保存不会受到迟来的影响。因为被保存的状态对于两个中断都是一样的，状态保存继续执行不会被打断。处理器对迟来中断进行管理，直到 ISR 的第一条指令进入处理器流水线的执行阶段。返回时，采用常规的 tail-chain 技术 |

## 2.4.2 异常处理的堆栈使用

Cortex-M3 处理器支持两个独立的堆栈：进程堆栈（Process Stack）和主堆栈（main stack）。系统运行的任一时刻，进程堆栈或主堆栈只有一个是可见的。复位后进入线程模式默认使用主堆栈，也可以通过软件配置使用进程堆栈。当中断发生，系统进入 ISR 中断服务，程序使用主堆栈，并且后面所有的抢占中断都使用主堆栈。具体堆栈使用规则如下。

- 线程模式使用主堆栈还是进程堆栈取决于 CONTROL 位[1]的值。该位可使用 MSR 或 MRS 指令访问，也可以在退出 ISR 时使用适当的 EXC_RETURN 的值来设置。抢占用户线程的异常将用户线程的状态保存在线程模式正在使用的堆栈中。
- 所有异常使用主堆栈来保存局部变量。

大多数操作系统支持线程模式使用进程堆栈，异常模式使用主堆栈。在操作系统进行任务调度时，内核只需要保存没有被硬件压栈的 8 个寄存器 r4～r11，并将 SP_process 复制到线程控制模块（TCB）中。如果处理器将状态保存在主堆栈中，则内核必须将 16 个寄存器复制到 TCB 中。

系统在线程模式下既可以使用主堆栈也可以线程堆栈，但一般情况下，对于一个受保护的线程模式，用户线程使用进程堆栈，而内核进程和中断服务程序使用主堆栈。

发生异常时，处理器自动将下面 8 个寄存器按以下顺序压栈：

- PC；
- xPSR；
- r0～r3；
- r12；
- LR。

完成压栈之后，SP 减小 8 字节。图 2.6 所示为异常抢占前和异常抢占之后堆栈中的内容。

华清远见

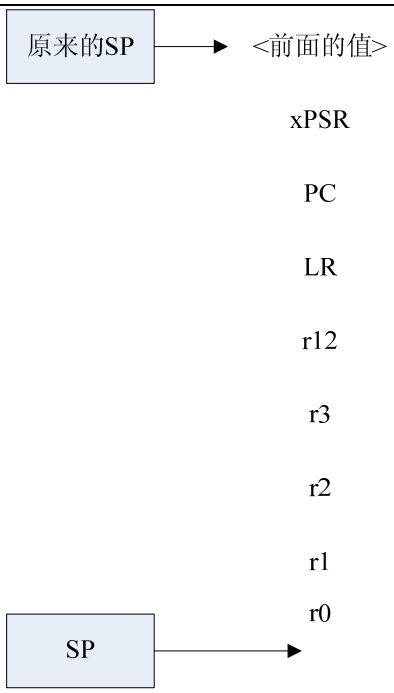



图 2.6　抢占之后堆栈中的内容

从 ISR 返回时，处理器将自动将保存的 8 个寄存器出栈。

表 2.7 所示为 Cortex-M3 处理器进入异常的步骤。

**表 2.7　　Cortex-M3 处理器异常进入步骤**

| 动　　作 | 是否可重启 | 说　　明 |
|---|---|---|
| 8 个寄存器压栈 | 否 | 在所选的堆栈上将 xPSR, PC, r0, r1, r2, r3, r12, LR 压栈 |
| 读向量表 | 是，迟来异常能够引起重启操作 | 读存储器中的向量表，地址为向量表基址 +（异常号 4）。ICode 总线上的读操作能够与 DCode 总线上的寄存器压栈操作同时执行 |
| 从向量表中读 SP | 否 | 只能在复位时，将 SP 更新为向量表中栈顶的值。选择堆栈，压栈和出栈之外的其它异常不能修改 SP |
| 更新 PC | 否 | 利用向量表读出的位置更新 PC。直到第一条指令开始执行时，才能处理迟来异常 |
| 加载流水线 | 是，占先从向量表中读出新的跳转向量，重新加载流水线 | 从向量表指向的位置加载指令。此操作与寄存器压栈操作同时执行 |
| 更新 LR | 否 | LR 设置为 EXC_RETURN，以便从异常中退出。EXC_RETURN 为 ARMv7-M 架构参考手册中定义的 16 个值之一 |

## 2.4.3　Cortex-M3 特有的异常处理机制

### 1．末尾连锁（Tail-chaining）

末尾连锁（Tail-chaining）机制能够在两个中断之间没有多余的状态保存和恢复指令的情况下实现“背对背（back-to-back processing）”处理。在退出 ISR 并进入另一个中断时，处理器略过 8 个寄存器的出栈和压栈操作，因为它对堆栈的内容没有影响。

华清远见

如果当前挂起中断的优先级比所有被压栈的异常的优先级都高，则处理器执行末尾连锁机制。如果挂起中断的优先级比被压栈的异常的最高优先级都高，则省略压栈和出栈操作，处理器立即取出挂起中断的向量。在退出前一个 ISR 之后，开始执行被末尾连锁（Tail-chained）的 ISR。

## 2．迟来（Late-arriving）

如果前一个 ISR 还没有进入执行阶段，并且迟来中断的优先级比前一个中断的优先级高，则迟来中断能够抢占前一个中断。

响应迟来中断时需执行新的取向量地址和 ISR 预取操作。迟来中断不保存状态，因为状态保存已经被最初的中断执行过了，因此不需要重复执行。

图 2.7 所示为一个中断迟来的处理机制。

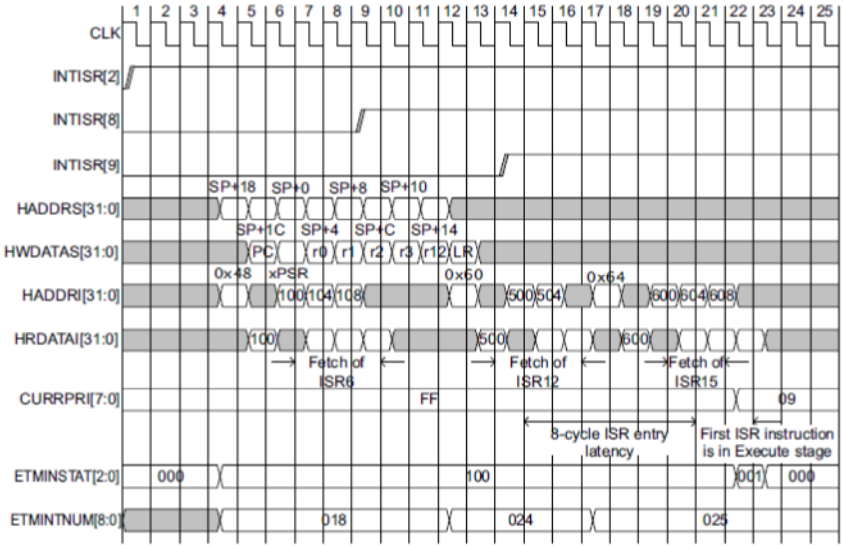



图 2.7　中断迟来处理实例

图 2.6 中，中断 INTISR[8]抢占了中断 INTISR[2]。中断 INTISR[2]的状态保存已经完成，不需要重复。中断 INTISR[8]在 INTISR[2]的 ISR 的第一条指令进入执行阶段之前实现抢占。该抢占点之后的更高优先级的中断将会被当作“占先”来处理。中断 INTISR[9]在 INTISR[8]的 ISR 的第一条指令进入执行阶段之前实现抢占。INTISR[8]的 ISR 取指操作在接收到 INTISR[9]时被中止，然后处理器开始 INTISR[9]的取向量操作。这个点之后的更高优先级的中断将会被当作“抢占”来处理。在 INTISR[9]的 ISR 进入执行阶段的过程中：

- ETMINSTAT[2:0](3`b001)表示已经进入 ISR，该脉冲为 1 个周期；
- CURRPRI[7:0]表示激活中断的优先级，CURRPRI 在整个 ISR 期间保持有效；
- ETMINTNUM[8:0]表示激活中断的数目；
- ETMINTNUM 在整个 ISR 期间保持有效。

## 2.4.4　异常退出

中断服务程序 ISR 的最后一条指令是将进入服务程序时保存的 LR 加载到 PC。该操作实际上是向处理器表明 ISR 已经完成。之后，处理器自动启动异常退出序列。

当从异常返回时，处理器执行下列操作之一。

- 如果挂起异常的优先级比所有被压栈的异常的优先级都高，则处理器会“末尾连锁（Tail-chaining）”到一个挂起异常。
- 如果没有挂起异常，或者如果被压栈的异常的最高优先级比挂起异常的最高优先级要高，则处理器返回到上一个被压栈的 ISR。

- 如果没有挂起中断或被压栈的异常，则处理器返回线程模式。

表 2.8 显示了 Cortex-M3 处理器的异常退出步骤。

表 2.8 Cortex-M3 处理器异常退出步骤

| 动　作 | 说　明 |
|---|---|
| 8 个寄存器出栈 | 如果没有被抢占，则将 PC、xPSR、r0、r1、r2、r3、r12、LR 依次从所选的堆栈中出栈（堆栈由 EXC_RETURN 选择），并调整 SP 寄存器 |
| 加载当前激活的中断号 | 加载来自被压栈的 IPSR 的位[8:0]中保存的当前激活的中断号。处理器用它来跟踪返回到哪个异常以及返回时清除激活位。当位[8:0]等于 0 时，处理器返回线程模式 |
| 选择 SP | 如果返回到异常，堆栈寄存器 SP 为 SP_main，如果返回到线程模式，则 SP 为 SP_main 或 SP_process |

值得注意的是，如果在保存值出栈过程中出现一个优先级更高的中断，则处理器放弃正在进行的出栈操作，堆栈指针退回到执行该次出栈前的状态，并将该异常看作 tail-chain 的情况来响应。

异常返回可以使用下面三种方式之一来实现将保存的 LR（0xFFFFFFFX）加载到 PC：

- POP 操作（包括加载 PC 的 LDM 操作）；
- LDR 操作，将 PC 作为目标寄存器；
- BX 操作，可使用任意寄存器作为操作寄存器。

当系统进入异常服务程序后，LR 的值被自动保存为特殊 EXC_RETURN 值，该值只有 bit[3：0]有意义，其余高 8 为均为 1。当异常服务程序将该值传递给 PC 寄存器时，系统自动启动处理器中断返回序列。表 2.9 显示了 EXC_RETURN 各比特位的具体含义。

因为 LR 的值是由系统自动设置的，所以建议用户不要轻易改动它。

表 2.9 EXC_RETURN 位段含义

| 位　段 | 含　义 |
|---|---|
| [31：4] | EXC_RETURN 标识，全为 1 |
| Bit 3 | 0 = 返回后进入 Handler 模式<br>1 = 返回后进入线程模式 |
| Bit 2 | 0 = 返回 ARM 状态。<br>1 = 返回 Thumb 状态。在 CM3 中必须为 1 |
| Bit 1 | 保留，必须为 0 |
| Bit 0 | 0 = 返回 ARM 状态。<br>1 = 返回 Thumb 状态。在 CM3 中必须为 1 |

如果在线程模式中，一旦将 EXC_RETURN 的值加载到 PC 寄存器，即将该值看作一个地址，而不是特殊的值时，将导致存储器管理故障。

## 2.4.5 复位异常

NVIC 与内核同时复位，并对内核从复位状态释放的行为进行控制。因此，复位的行为是可预测的。表 2.10 显示了 Cortex-M3 处理器的复位行为。

表 2.10 Cortex-M3 处理器复位行为

| 行　为 | 说　明 |
|---|---|
| NVIC 复位，内核保持在复位状态 | NVIC 的大部分寄存器清零。处理器处于线程模式，优先级为特权模式，堆栈设置为主堆栈 |
| NVIC 将内核从复位状态释放 | NVIC 将内核从复位状态释放 |
| 内核设置堆栈 | 内核从向量表偏移 0 中读取最初的 SP，该 SP 为 SP_main |

| | |
|---|---|
| 内核设置 PC 和 LR | 内核从向量表偏移中读取最初的 PC，LR 设置为 0xFFFFFFFF |
| 运行复位程序 | NVIC 的中断被禁止，NMI 和硬故障异常开启 |

位于 0 地址的向量表，从低地址起，依次存放：栈顶地址、复位程序的起始地址、NMI ISR 的起始地址、硬故障处理函数 ISR 的起始地址。如果使用 SVC 指令，还需要指定 SVCall ISR 的位置。

完整的向量表程序清单如下：

```
unsigned int stack_base[STACK_SIZE];
void ResetISR(void);
void NmiISR(void);
…
ISR_VECTOR_TABLE vector_table_at_0
{
stack_base + sizeof(stack_base),
ResetISR,
NmiSR,
FaultISR,
0,                          // 如果使用 MemManage (MPU)，在此添加它的 ISR
0,                          // 如果使用总线故障，在此添加它的 ISR
0,                          // 如果使用“使用故障”，在此添加它的 ISR
0, 0, 0, 0,             // 保留
SVCallISR,
0,                          // 如果使用调试监控，在此添加它的 ISR
0,                          // 保留
0,                          // 如果使用响应请求可挂起功能，在此添加它的 ISR
0,                          // 如果使用 SysTick，在此添加它的 ISR
                            // 外部中断从这里开始
Timer1ISR,
GpioInISR
GpioOutISR,
I2CIsr
};
```

通常一个情况下，复位程序遵循表 2.11 中的步骤进行。C/C ++运行时将执行前三步，然后调用 main()。

**表 2.11　　Cortex-M3 处理器复位启动动作**

| 行　　为 | 说　　明 |
|---|---|
| 初始化变量 | 必须设置所有的全局/静态变量。包括将 BSS（已初始化的变量）清零，并将变量的初值从 ROM 中复制到 RAM 中 |
| 设置堆栈 | 如果使用多个堆栈，另一个分组的 SP 必须进行初始化。当前的 SP 也可以从主堆栈变为进程堆栈 |
| 初始化所有运行时间 | 可选择调用 C/C ++运行时间的注册码，以允许使用堆（heap），浮点运算或其他功能。这通常可通过__main 调用 C/C ++库来完成 |
| 初始化所有外设 | 在中断使能之前设置外设。可以调用它来设置应用中使用的外设 |
| 切换 ISR 向量表 | 可选择将代码区，0 地址中的向量表转换到 SRAM 中。这样做只是为了优化性能或允许动态改变 |

续表

| 行　　为 | 说　　明 |
|---|---|
| 设置可配置的故障 | 使能可配置的故障并设置它们的优先级 |
| 设置中断 | 设置中断优先级和屏蔽位 |
| 使能中断 | 使能中断。使能 NVIC 的中断处理。如果不希望在中断刚使能时产生中断，则可通过 CPS 或 MSR 指令，设置 PRIMASK 寄存器，在准备就绪之前屏蔽中断 |

| | |
|---|---|
| 改变优先级 | 如果有必要，线程模式的特权访问可变为用户访问。该操作通常通过调用 SVCall 处理程序来实现 |
| 循环（loop） | 如果使能退出时进入睡眠功能（sleep-on-exit），则在产生第一个中断/异常之后，控制不会返回。可通过寄存器选择使能/禁止该功能，而 loop 能够处理清除操作和执行的任务。如果选择禁止该睡眠功能，则 loop 能够使用 WFI（现在睡眠）功能 |

复位程序用来使能系统中断，并启动应用程序。下面介绍了 3 种方法，可以在执行完中断处理之后调用复位异常处理程序。

（1）退出中断处理程序时，进入完全睡眠的复位程序。程序操作步骤如下：

```
void reset()
{
// 完成设置工作 (初始化变量，初始化运行时间（根据需要），设置外设等)
nvic[INT_ENA] = 1;        // 使能中断
nvic_regs[NV_SLEEP] |= NVSLEEP_ON_EXIT; // 在第一个异常之后不会正常返回
while (1)
wfi();
}
```

（2）使用 WFI 进入睡眠模式（可选的复位程序）。程序操作步骤如下：

```
void reset()
{
extern volatile unsigned exc_req;
// 完成设置工作 (初始化变量，初始化运行时间（如果需要），设置外设等)
nvic[INT_ENA] = 1;        // 使能中断
while (1)
{
                          // 完成(exc_req = FALSE; exc_req == FALSE; )的部分工作
wfi();                    // 现在进入睡眠状态，等待中断
                          // 完成部分上电自检异常的检验/清除
}
}
```

（3）所选的 Sleep-on-exit 功能被 ISR 取消的复位程序。程序操作步骤如下：

```
void reset()
{
// 完成设置工作 (初始化变量，初始化运行时间（如果需要），设置外设等)
nvic[INT_ENA] = 1;        // 使能中断
while (1)
{
// 处于睡眠状态直到有异常清除 sleep on reset 状态，这样能够处理上电自检/清除。
nvic_regs[NV_SLEEP] |= NVSLEEP_ON_EXIT;
while (nvic_regs[NV_SLEEP] & NVSLEEP_ON_EXIT)
wfi();                    // 现在进入睡眠状态，等待中断来清除
                          // 完成部分上电自检异常的检验/清除
}
}
```

因为可以通过激活 ISR 来改变异常优先级，所以异常的处理程序不必放在复位程序中。这样做可以保证系统对异常优先级改变时的响应速度。另外优先级的改变机制，解决了操作系统中优先级倒置（priority inversion）问题。对于使用线程和特权访问的操作系统来说，用户代码可以使用线程模式进行操作。

### 2.4.6 中止（Abort）异常

Cortex-M3 处理器中，有以下 4 种情况产生中止故障：

- 指令取址或向量表加载时的总线错误；
- 数据访问时的总线错误；
- 内部检测到的错误，例如，出现未定义的指令或使用 BX 指令来改变状态。NVIC 中的故障状态寄存器标识了故障的发生状态；
- 系统出现 MPU 故障。

故障处理程序有以下两种：

① 固定优先级的硬故障；

② 优先级可调整的局部故障。

只有复位和 NMI 能够抢占固定优先级的硬故障。硬故障能够抢占除复位 NMI 或其他硬故障之外的所有异常。

二级总线故障不能升级，因为相同类型的占先故障不能抢占本身。这意味着如果一个被破坏的堆栈引起了故障，尽管压栈操作失败，但故障处理程序仍然执行。故障处理程序能够工作，但堆栈内容被破坏。

局部故障根据引起的原因分类。使能时，局部故障处理程序处理所有常规的故障，当出现下列情况时局部故障升级为硬故障：

- 局部故障处理程序正在响应时触发了相同类型的故障；
- 局部故障处理程序触发了具有相同或更高优先级的故障；
- 异常处理程序触发了具有相同或更高优先级的故障；
- 局部故障未使能。

表 2.12 列出了所有局部故障。

**表 2.12　　Cortex-M3 处理器局部故障**

| 错　误 | 位 名 称 | 处 理 程 序 | 备　注 | 陷进（trap）使能位 |
|---|---|---|---|---|
| 复位 | 复位原因 | 复位 | 任何形式复位 | RESETVCATCH |
| 读向量错误 | VECTTBL | 硬故障 | 读向量表入口时返回的总线错误 | INTERR |
| uCode 压栈错误 | STKERR | 总线故障 | 总线故障使用硬件保存上下文时失败—返回的总线错误 | INTERR |
| uCode 压栈错误 | MSTKERR | 存储器管理 | 使用硬件保存上下文时失败，即违反 MPU 访问规则 | INTERR |
| uCode 出栈错误 | UNSTKERR | 总线故障 | 总线故障使用硬件恢复上下文时失败，即返回的总线错误 | INTERR |
| uCode 出栈错误 | uCode 出栈错误 MUNSKERR | 存储器管理 | 存储器管理使用硬件恢复上下文时失败，即违反 MPU 访问规则 | INTERR |
| 升级为硬故障 | FORCED | 硬故障 | 当局部故障的优先级没有使能，或可配置的故障禁止时，产生的故障以及处理程序与当前的，包括故障内的故障的优先级相等或更高。这包括 SVC，BKPT 和其它类型的故障 | INTERR |
| MPU 不匹配 | DACCVIOL | 存储器管理 | 由于数据访问而产生的 MPU 违反或故障 | MMERR |
| MPU 不匹配 | IACCVIOL | 存储器管理 | 由于指令地址而产生的 MPU 违反/故障 | MMERR |
| 预取错误 | IBUSERR | 总线故障 | 由于指令取指而返回的总线错误。仅在指令进入执行阶段才会发生故障。跳转指令后面的指令若出错，可以被忽略 | BUSERR |
| 精确数据总线错误 | PRECISEERR | 总线故障 | 由于数据访问而返回的总线错误，是精确的，指向指令 | BUSERR |
| 非精确数据总线错 | IMPRECISERR | 用法错误 | 由于数据访问而产生的延迟的总线错误。 | BUSERR |

华清远见

| | | | | |
|---|---|---|---|---|
| 误 | | | 精确的指令不可知。这是被挂起的，不是同步的。它不引起 FORCED | |

续表

| 错　　误 | 位　名　称 | 处理程序 | 备注 | 陷进（trap）使能位 |
|---|---|---|---|---|
| 无协处理器 | NOCP | 用法错误 | 用法错误确实不存在，或不呈现位 | NOCPERR |
| 未定义指令 | UNDEFINSTR | 用法错误 | 未知的指令 | STATERR |
| 试图在无效的 ISA 状态中执行指令 | INVSTATE | 用法错误 | 试图在无效的 EPSR 状态中执行，例如，在 BX 类型指令改变状态之后。这包括从异常中返回之后的状态 | STATERR |
| 在没有使能或带有非法的魔数（magic umber）时返回到 PC = EXC_RETURN | INVPC | 用法错误 | 非法退出，由非法的 EXC_RETURN 值，（EXC_RETURN 与压栈的 EPSR 不匹配），或当前的 EPSR 不包含在当前有效的异常列表中时执行退出而引起的 | STATERR |
| 非法的未对齐加载或存储 | UNALIGNED | 用法错误 | 用法错误当任意的多寄存器加载/存储指令尝试访问一个非字对齐的单元时产生该故障<br>对于任意的与其尺寸不对齐的加载/存储操作，都可以使用 UNALIGN_TRP 位来使能该故障 | CHKERR |
| 除以 0 | DIVBYZERO | 用法错误 | 在执行 SDIV 或 UDIV 时除数为 0，并且 DIV_0_TRP 位置位时，能够使能来产生该故障 | CHKERR |
| SVC | - | SVCall | 系统请求（服务调用） | - |

每个故障都有一个故障状态寄存器，带有用于该故障的标志，分别为：

- 3 个可配置的故障状态寄存器，对应于 3 个可配置的故障处理程序；
- 一个硬故障状态寄存器；
- 一个调试故障状态寄存器。

处理器共有 2 个故障地址寄存器（FAR），分别为：

- 总线故障地址寄存器（BFAR）；
- 存储器故障地址寄存器（MFAR）。

对应故障状态寄存器中的标志位指示了故障状态寄存器中的地址何时是有效的。

表 2.13 所示为故障状态寄存器和两个故障地址寄存器。

**表 2.13　　Cortex-M3 处理器故障状态和故障地址寄存器**

| 状态寄存器名称 | 处 理 程 序 | 地址寄存器名称 | 说　　明 |
|---|---|---|---|
| HFSR | 硬故障 | - | 升级和特殊 |
| MMSR | 存储器管理 | MMAR | MPU 故障 |
| BFSR | 总线故障 | BFAR | 总线故障 |

续表

| 状态寄存器名称 | 处 理 程 序 | 地址寄存器名称 | 说　　明 |
|---|---|---|---|
| UFSR | 用法所谓 | - | 用法错误 |
| DFSR | 调试器监控或停止 | - | 调试陷阱 |

## 2.5　Cortex-M3 的电源管理

华清远见

不同于以往的处理器，Cortex-M3 对电源管理的重视，已经上升到处理器内核的水平上。它提供了两种睡眠模式：立即睡眠（Sleeping）和深度睡眠（Deep Sleep）。在睡眠时，可以停止系统时钟，但可以让 FCLK 继续走，以允许处理器能被 SysTick 异常唤醒。对系统控制寄存器进行写操作，可以控制 Cortex-M3 系统功耗的状态，表 2.14 列出了系统控制寄存器的详细说明。

表 2.14　　系统控制寄存器（地址 0xE000ED10）

| 位　段 | 名　称 | 描　述 |
|---|---|---|
| Bit4 | SEVONPEND | 发生异常悬起时请发送事件，用于在一个新的中断悬起时从 WFE 指令处唤醒。不管这个中断的优先级是否比当前的高，都唤醒。如果没有 WFE 导致睡眠，则下次使用 WFE 时将立即唤醒 |
| Bit3 | 保留 | - |
| Bit2 | SLEEPDEEP | 当进入睡眠模式时，使能外部的 SLEEPDEEP 信号，以允许停止系统时钟 |
| Bit1 | SLEEPONEXIT | 激活“SleepOnExit”功能 |
| Bit0 | 保留 | - |

处理器具有以下信号以指示处理器进入睡眠的具体时间。

- SLEEPING：该信号在立即睡眠或“退出时睡眠”模式下有效，表示处理器时钟可以停止运行。在接收到一个新的中断后，NVIC 会使该信号变无效，使内核退出睡眠。
- SLEEPDEEP：当系统控制寄存器的 SLEEPDEEP 位置位时，该信号在立即睡眠或“退出时睡眠”模式下有效。该信号被传送给时钟管理器，并可以用来门控处理器和包含锁相环（PLL）的系统元件以节省功耗。在接收到新的中断时，嵌套向量中断控制器（NVIC）将 SLEEPDEEP 信号变无效，并在时钟管理器显示时钟稳定时让内核退出睡眠。

## 2.5.1　SLEEPING

图 2.8 给出了如何在低功耗状态利用 SLEEPING 来门控处理器的 HCLK 时钟以减少功耗的实例。如有必要，还可以使用 SLEEPING 来门控其他系统元件。

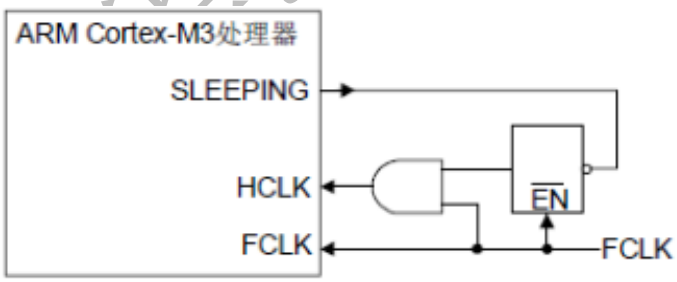



图 2.8　SLEEPING 功耗控制实例

为了探测中断，处理器必须一直接受自由震荡的 FCLK。FCLK 用于对以下元件计时：

- 探测中断的 NVIC 中的少量逻辑电路；
- DWT 和 ITM 模块，这些模块被使能相应功能后可以在睡眠期间产生跟踪包。如果“调试异常与监控寄存器”的 TRCENA 位使能，那么模块的功耗将会降低。在 SLEEPING 信号有效期间可以降低 FCLK 频率。

## 2.5.2　SLEEPDEEP

图 2.9 给出了如何在低功耗状态利用 SLEEPDEEP 来停止时钟控制器以进一步减少功耗的实例。退出低功耗状态时，LOCK 信号指示 PLL 稳定，并且此时使能 Cortex-M3 时钟是安全的，这可以保证处理器不会重启直至时钟稳定。

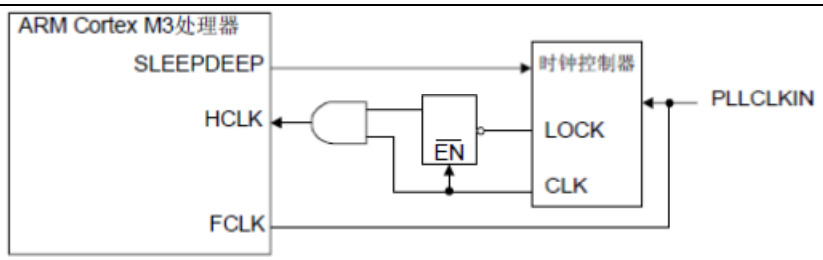



图 2.9　SLEEPDEEP 功耗控制实例

为了检测中断，处理器在低功耗状态下必须接收自由振荡的 FCLK。在 SLEEPDEEP 有效期间可以降低 FCLK 频率。

## 2.6　嵌套向量中断控制器 NVIC

CortexM3 中，NVIC 的存在不仅方便了低延时异常和中断的处理，而且还实现了电源管理和系统的寄存器。NVIC 支持 240 个优先级可动态配置的中断，每个中断的优先级有 256 个选择。低延迟的中断处理可以通过紧耦合的 NVIC 和处理器内核接口来实现，让新进的中断可以得到有效的处理。NVI 通过时刻关注压栈（嵌套）中断来实现中断的末尾连锁（Tail-chaining）。

用户只能在特权模式下完全访问 NVIC，但是如果使能了配置控制寄存器，就可以在用户模式下挂起（pend）中断。其他用户模式的访问会导致总线故障。一般情况下，NVIC 的所有寄存器都可采用字节、半字和字方式进行访问。

不管处理器存储字节的顺序如何，所有 NVIC 寄存器和系统调试寄存器都是采用小端（little endian）字节排列顺序，即低位字节存储在低地址。

NVIC 控制器除了实现异常处理还实现了一些系统控制功能。NVIC 寄存器空间分成以下几部分。

- 0xE000E000 — 0xE000E00F：中断类型寄存器。
- 0xE000E010 — 0xE000E0FF：系统定时器。
- 0xE000E100 — 0xE000ECFF：NVIC。
- 0xE000ED00 — 0xE000ED8F：系统控制模块，包括 CPUID、系统控制、配置和状态、故障报告。
- 0xE000EF00 — 0xE000EF0F：软件触发异常寄存器。
- 0xE000EFD0 — 0xE000EFFF：ID 空间。

表 2.15 所示为 NVIC 寄存器。

**表 2.15　NVIC 寄存器**

| 名　称 | 类　型 | 地　址 | 复 位 值 |
|---|---|---|---|
| 中断控制类型寄存器 | 只读 | 0xE000E004 | |
| 系统时钟节拍（SysTick）控制与状态寄存器 | 读/写 | 0xE000E010 | 0x00000000 |
| 系统时钟节拍（SysTick）重装值寄存器 | 读/写 | 0xE000E014 | 不可预测 |
| 系统时钟节拍（SysTick）当前值寄存器 | 读/写清除 | 0xE000E018 | 不可预测 |
| 系统时钟节拍（SysTick）校准值寄存器 | 只读 | 0xE000E01C | STCALIB |
| Irq0～31 使能设置寄存器 | 读/写 | 0xE000E100 | 0x00000000 |
| Irq224～239 使能设置寄存器 | 读/写 | 0xE000E11C | 0x00000000 |
| Irq0～31 使能清除寄存器 | 读/写 | 0xE000E180 | 0x00000000 |
| Irq224～239 使能清除寄存器 | 读/写 | 0xE000E19C | 0x00000000 |
| Irq0～31 挂起设置寄存器 | 读/写 | 0xE000E200 | 0x00000000 |
| Irq224～239 挂起设置寄存器 | 读/写 | 0xE000E21C | 0x00000000 |

华清远见

| | | | |
|---|---|---|---|
| Irq0～31 挂起清除寄存器 | 读/写 | 0xE000E280 | 0x00000000 |
| Irq224～239 挂起清除寄存器 | 读/写 | 0xE000E29C | 0x00000000 |
| Irq0～31 激活位寄存器 | 只读 | 0xE000E29C<br>0x00000000 | 0x00000000 |
| Irq224～239 激活位寄存器 | 只读 | 0xE000E31C | 0x00000000 |

续表

| 名　　称 | 类　　型 | 地　　址 | 复　位　值 |
|---|---|---|---|
| Irq0～31 优先级寄存器 | 读/写 | 0xE000E400 | 0x00000000 |
| Irq236～239 优先级寄存器 | 读/写 | 0xE000E4F0 | 0x00000000 |
| CPUID 基址寄存器 | 只读 | 0xE000ED00 | 0x410FC230 |
| 中断控制状态寄存器 | 读/写或只读 | 0xE000ED04 | 0x00000000 |
| 向量表偏移寄存器 | 读/写 | 0xE000ED0C | 0x00000000 |
| 应用中断/复位控制寄存器 | 读/写 | 0xE000ED0C | 0x00000000 |
| 系统控制寄存器 | 读/写 | 0xE000ED10 | 0x00000000 |
| 配置控制寄存器 | 读/写 | 0xE000ED14 | 0x00000000 |
| 系统处理器 4-7 优先级寄存器 | 读/写 | 0xE000ED18 | 0x00000000 |
| 系统处理器 8-11 优先级寄存器 | 读/写 | 0xE000ED1C | 0x00000000 |
| 系统处理器 12-15 优先级寄存器 | 读/写 | 0xE000ED20 | 0x00000000 |
| 系统处理器控制与状态寄存器 | 读/写 | 0xE000ED24 | 0x00000000 |
| 可配置故障状态寄存器 | 读/写 | 0xE000ED28 | 0x00000000 |
| 硬故障状态寄存器 | 读/写 | 0xE000ED2C | 0x00000000 |
| 调试故障状态寄存器 | 读/写 | 0xE000ED30 | 0x00000000 |
| 存储器管理地址寄存器 | 读/写 | 0xE000ED34 | 不可预测 |
| 总线故障地址寄存器 | 读/写 | 0xE000ED38 | 不可预测 |
| PFR0:处理器功能寄存器 0 | 只读 | 0xE000ED40 | 0x00000000 |
| PFR1:处理器功能寄存器 1 | 只读 | 0xE000ED44 | 0x00000000 |
| DFR0:调试功能寄存器 0 | 只读 | 0xE000ED48 | 0x00000000 |
| AFR0：辅助功能寄存器 0 | 只读 | 0xE000ED4C | 0x00000000 |
| MMFR0：存储器模型功能寄存器 0 | 只读 | 0xE000ED50 | 0x00000000 |
| MMFR1：存储器模型功能寄存器 1 | 只读 | 0xE000ED54 | 0x00000000 |
| MMFR2：存储器模型功能寄存器 2 | 只读 | 0xE000ED58 | 0x00000000 |
| MMFR3：存储器模型功能寄存器 3 | 只读 | 0xE000ED5C | 0x00000000 |
| ISAR0：ISA 功能寄存器 0 | 只读 | 0xE000ED60 | 0x01141110 |
| ISAR1：ISA 功能寄存器 1 | 只读 | 0xE000ED64 | 0x02111000 |
| ISAR2：ISA 功能寄存器 2 | 只读 | 0xE000ED68 | 0x21112231 |

续表

| 名　　称 | 类　　型 | 地　　址 | 复　位　值 |
|---|---|---|---|
| ISAR3：ISA 功能寄存器 3 | 只读 | 0xE000ED6C | 0x01111110 |
| ISAR4：ISA 功能寄存器 4 | 只读 | 0xE000ED70 | 0x01310102 |
| 软件触发中断寄存器 | 只写 | 0xE000EF00 | - |
| 外设识别寄存器（PERIPHID4） | 只读 | 0xE000EFD0 | 0x04 |

华清远见

| | | | |
|---|---|---|---|
| 外设识别寄存器（PERIPHID5） | 只读 | 0xE000EFD4 | 0x00 |
| 外设识别寄存器（PERIPHID6） | 只读 | 0xE000EFD8 | 0x00 |
| 外设识别寄存器（PERIPHID7） | 只读 | 0xE000EFDC | 0x00 |
| 外设识别寄存器位 7:0（PERIPHID0） | 只读 | 0xE000EFE0 | 0x00 |
| 外设识别寄存器位 15:8（PERIPHID1） | 只读 | 0xE000EFE4 | 0xB0 |
| 外设识别寄存器位 23:16（PERIPHID2） | 只读 | 0xE000EFE8 | 0x0B |
| 外设识别寄存器位 31:24（PERIPHID3） | 只读 | 0xE000EFEC | 0x00 |
| 元件识别寄存器位 7:0（PCELLID0） | 只读 | 0xE000EFF0 | 0x0D |
| 元件识别寄存器位 15:8（PCELLID1） | 只读 | 0xE000EFF4 | 0xE0 |
| 元件识别寄存器位 23:16（PCELLID2） | 只读 | 0xE000EFF8 | 0x05 |
| 元件识别寄存器位 31:24（PCELLID3） | 只读 | 0xE000EFFC | 0xB1 |

关于 NVIC 寄存器的具体定义，请参见 Cortex-M3 处理器参考手册。

Cortex-M3 处理器支持电平中断和脉冲中断。电平中断保持有效，直到访问器件的 ISR 将它清除。脉冲中断是边沿模型的一个变量。边沿不是异步的，相反，它必须在 Cortex-M3 时钟 HCLK 的上升沿被采样。对于电平中断，如果中断程序返回前该信号没有失效，那么中断重新挂起和重新激活。这一点对于 FIFO 和基于缓冲器的器件特别有用，因为它可以保证无需额外的工作，仅通过使用一个 ISR 或重复调用就可将 FIFO 和缓冲器清空。即器件将该信号保持有效，直至器件变空。

脉冲中断在 ISR 过程中可以重新变有效，所以中断可以同时挂起和激活。应用设计必须确保只有在第一个脉冲激活后下一个脉冲才能到达。第二个挂起由于已经挂起所以没有什么用处。但是如果中断在一个或一个以上的周期内保持有效，那么 NVIC 会锁存该挂起位。当 ISR 激活时将挂起位清零。如果在激活的同时中断再次被确定，它可以再次锁存挂起位。脉冲中断大都使用在外部信号、速率或重复信号中。

## 2.7 存储器保护单位 MPU

存储器保护单元（MPU）是用来保护存储器的一个元件。Cortex-M3 处理器支持标准的 ARMv7 PMSA 模型，具有以下功能：

- 保护内存区域；
- 将保护区域重叠；
- 控制访问权限；
- 将存储器属性输出到系统。

内存访问时，如果 MPU 不匹配或越权访问，将发生“内存管理错误（MemManage Fault Handler）”，激活存储器管理故障处理程序。表 2.16 所示为 MPU 相关寄存器，关于寄存器更详细内容，请参见 Cortex-M3 参考手册。

**表 2.16 MPU 寄存器**

| 名　称 | 类　型 | 地　址 | 复 位 值 |
|---|---|---|---|
| MPU 类型寄存器 | 只读 | 0xE000ED90 | 0x00000800 |
| MPU 控制寄存器 | 读/写 | 0xE000ED94 | 0x00000000 |
| MPU 区号寄存器 | 读/写 | 0xE000ED98 | - |
| MPU 区域基址寄存器 | 读/写 | 0xE000ED9C | - |
| MPU 区域属性与大小寄存器 | 读/写 | 0xE000EDA0 | - |
| MPU 别名 1 区基址寄存器 | D9C 重叠 | 0xE000EDA4 | - |
| MPU 别名 1 区属性与大小寄存器 | DA0 重叠 | 0xE000EDA8 | - |

华清远见

| | | | |
|---|---|---|---|
| MPU 别名 2 区基址寄存器 | D9C 重叠 | 0xE000EDAC | - |
| MPU 别名 2 区属性与大小寄存器 | DA0 重叠 | 0xE000EDB0 | - |
| MPU 别名 3 区基址寄存器 | D9C 重叠 | 0xE000EDB4 | - |
| MPU 别名 3 区属性与大小寄存器 | DA0 重叠 | 0xE000EDB8 | - |

对 MPU 区域的编程，可通过对映射到内存的三个字寄存器的编程来实现。三个寄存器相互独立，程序可分开访问。MPU 寄存器相互独立的特性，可以使用户方便地移植现有的 ARMv6、ARMv7 和 CP15 的代码，使 Cortex-M3 很容易的实现向后兼容。当移植 ARMv6、ARMv7 和 CP15 的现有代码时，只需使用 LDRx 和 STRx 操作代替 MRC 和 MCR。

使用 CP15 等效代码更新 MPU 区域的代码实例如下所示：

```
    R1 = region number
    R2 = size/enable
    R3 = attributes
    R4 = address
   MOV R0,#NVIC_BASE
   ADD R0,#MPU_REG_CTRL
   STR R1,[R0,#0] ;
   STR R4,[R0,#4] ;
   STRH R2,[R0,#8] ;
   STRH R3,[R0,#10] ;
```

值得注意的是，如果中断在这期间可以抢占，那么它会受 MPU 区域的影响。即必须禁能、写然后再使能该区域。这对于上下文转换器通常没太大用处，但是如果需要在其他地方进行更新，这就很有必要了。

MPU 可以包含关键的数据。这是因为在更新时得花费 1 个以上的总线处理。通常是 2 个字。结果就不是“线程安全”了。即中断可以将两个字分离，使得区域包含不连续的信息。此时要注意以下两个问题。

- 更新 MPU 通常会产生中断。这不仅是“读-修改-写”的问题，它还会对“保证中断程序不会修改相同区域”的情形造成影响。这是因为编程取决于正写入寄存器的区号，所以它知道要更新哪个区。因而这种情形下每个更新程序周围都必须禁能中断。
- 使用域操作更新 MPU 会产生中断，该中断将使正在更新的区域受到影响，因为只有基址或“大小域”被更新。如果新的大小域发生了改变，但是基址没有变，那么基址 +new_size 可能会在一个被另外区域正常处理的区域内重叠。

但是对于标准的 OS 上下文转换代码，将会改变用户区域，因为这些区域会被预设成用户特权和用户区地址，所以没有风险。也就是说即使是中断也不会引起副作用。因此不需要禁能/使能代码，也不需要禁止中断。

最普通的方法是只从两个位置对 MPU 进行编程：引导代码和上下文转换器。如果以唯一的两个位置进行编程，且上下文转换器仅更新用户区，那么因为上下文转换器已经是一个关键区域且引导代码在禁能中断时运行，所以不需要禁能。

## 2.8 Cortex-M3 开发平台——STM32V100 评估板介绍

STM103V100 是英蓓特公司新推出的一款基于 ST 公司 STM32 系列处理器（Cortex-M3 内核）的全功能评估板。该评估板采用主板与子板组合的工作方式，只需通过换插不同的 CPU 子板，即可实现 STM32 系列多款处理器的代码调试。STM103V100 评估板有 USB、Motor Control、CAN、SD 卡、Smart 卡、UART 等，有助于用户评估 STM32 的性能。

STM103V100 评估板最大特色是板上集成了 uLinkMe 调试电路，用户只需一跟 USB 线即可进行调试，另外 STM103V100 评估板还支持 JTAG 和 SWD 串行调试方式，非常方便。

STM103V100 功能模块图如图 2.10 所示。

STM103V100 评估板主要性能如下。

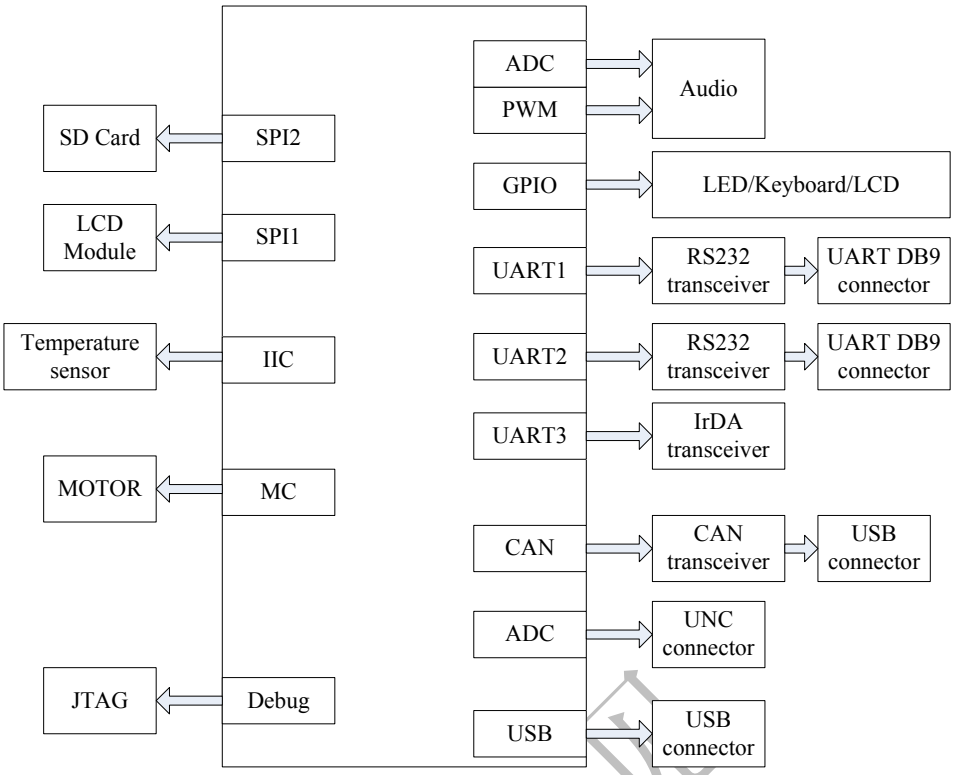



图 2.10　STM103V100 功能模块图

- 处理器：STM32F103VBT6，兼容 STM32F103V8T6（100pins）、STM32F103RBT6、STM32F103R8T6、STM32F103C8T6、STM32F103C6T6。
- RTC。
- 8M byte SPI Flash（预留芯片位，暂未焊接；已提供驱动代码）。
- 启动跳线设置。
- 将所有的引脚引出，通过排针和 Base Board V100 相连。
- 3 路可选电源，分别是外接 5VDC 供电、USB 供电、子板供电。
- 一个 SD 存储卡接口。
- TFT LCD 和 16 × 2 字符型液晶接口。
- 外扩一个温度传感器。
- 一个电机接口。
- 3 种类型调试接口，可选 SWD、20Pin JTAG 接口或 USB uLinkMe 接口调试。
- 2 个 RS-232 串行口，其中一个支持 RTS/CTS 功能。
- 一个 USB Slave B 型接口。
- 一个可以具有控制 4 个方向和确定功能的摇杆手柄。
- 4 个按键分别是 Reset、Wakeup、Temper 和 User 按键。
- 4 个 Led 灯。
- 一路立体声音频输出接口，一路麦克风接口。
- 一个 CAN 总线接口，通过 DB9 接口引出。
- 一个 Smart Card 接口（预留芯片位和卡座位，暂未焊接）。

关于 STM103V100 评估板更详细资料和使用方法请参见英倍特网站及产品参考手册。

# 联系方式

集团官网：www.hqyj.com 嵌入式学院：www.embedu.org 移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn 物联网学院：www.topsight.cn 研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

华清远见