Cortex-M3 技术参考手册

专注的力量成就梦想

目录

前言

前言部分概述了 *Cortex-M3 r0p0 技术参考手册*，包括以下内容：

- 关于本手册
- 信息反馈

关于本手册

本手册是关于 Cortex-M3 处理器的技术参考手册。

产品修订状态

rnpn 标识符表示本手册中所述产品的修订状态。

rn 表示产品的主要修改。

pn 表示产品的细微修改。

目标读者

本手册是为基于 Cortex-M3 处理器来实现片上系统（SoC）器件的系统设计人员，系统整合人员，以及验证工程师而写的。

本手册的使用

本手册按以下章节组织：

第 1 章 概述

本章描述了 Cortex-M3 处理器的组件以及处理器的指令集。

第 2 章 编程模型（programmer’s model）

本章描述了 Cortex-M3 的寄存器集，工作模式，和其它与 Cortex-M3 处理器的编程相关的信息。

第 3 章 系统控制

本章描述了系统控制的寄存器和编程模型。

第 4 章 存储器映射

本章描述了处理器映射和 bit-banding 特性。

第 5 章 异常

本章描述了处理器的异常。

第 6 章 时钟与复位

本章描述了处理器的时钟和复位。

第 7 章 功率管理

本章描述了处理器功率管理和节电技术

第 8 章 嵌套向量中断控制器

本章描述了处理器中断处理和控制

第 9 章　存储器保护单元

本章描述了处理器的存储器保护单元

第 10 章 内核调试

本章描述了对处理器内核的调试和测试处理。

第 11 章 系统调试

本章描述了处理器系统调试组件。

第 12 章 调试端口

本章描述了处理器调试端口，JTAG 调试端口和串行线调试端口。

第 13 章 跟踪端口接口单元

本章描述了处理器的跟踪端口接口单元（TPIU）。

第 14 章 总线接口

本章描述了处理器的总线接口。

第 15 章 嵌入式跟踪宏单元

本章描述了处理器的嵌入式跟踪宏单元（ETM）

第 16 章 嵌入式跟踪宏单元接口

本章描述了处理器的 ETM 接口

第 17 章 指令时序

本章描述了处理器的指令时序和时钟周期

附录 A　信号描述

本章汇总了 Cortex-M3 信号。

第1章 概述

本章简要介绍了 Cortex-M3 处理器和指令集，包含以下内容：

- *关于处理器*
- *处理器的组件*
- *可配置选项*
- *指令集汇总*

1.1 关于处理器

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。该处理器采用 ARMv7-M 架构。

Cortex-M3 处理器整合了以下组件：

- 处理器内核。这款门数目少，中断延迟短的处理器具备以下特性：
  - — ARMv7-M：Thumb-2 ISA 子集，包含所有基本的 16 位和 32 位 Thumb-2 指令，用于多媒体，SIMD，E(DSP)和 ARM 系统访问的模块除外。
  - — 只有分组的 SP
  - — 硬件除法指令，SDIV 和 UDIV（Thumb-2 指令）
  - — 处理模式（handler mode）和线程模式（thread mode）
  - — Thumb 状态和调试状态
  - — 可中断-可继续（interruptible-continued）的 LDM/STM，PUSH/POP，实现低中断延迟。
  - — 自动保存和恢复处理器状态，可以实现低延迟地进入和退出中断服务程序（ISR）。
  - — 支持 ARMv6 架构 BE8/LE
  - — ARMv6 非对齐访问
- 嵌套向量中断控制器（NVIC）。它与处理器内核紧密结合实现低延迟中断处理，并具有以下特性：
  - — 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。这样，在两个中断之间没有多余的状态保存和状态恢复指令的情况下，使能背对背中断（back-to-back interrupt）处理。
  - — 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）。MPU 功能可选，用于对存储器进行保护，它具有以下特性：

— 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

— AHBLite ICode、DCode 和系统总线接口

— APB 专用外设总线（PPB）接口

— Bit band 支持，bit-band 的原子写和读访问。

— 存储器访问对齐

— 写缓冲区，用于缓冲写数据。

- 低成本调试解决方案，具有以下特性：

— 当内核正在运行、被中止、或处于复位状态时，能对系统中包括 Cortex-M3 寄存器组在内的所有存储器和寄存器进行调试访问。

— 串行线（SW-DP）或 JTAG(JTAG-DP)调试访问，或两种都包括。

— 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（system profiling）

— 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。

1.2 处理器的组件

这节内容描述了 Cortex-M3 处理器的组件，系统的主要模块包括：

- 处理器内核
- NVIC
- 总线矩阵
- FPB
- DWT
- ITM
- MPU
- ETM
- TPIU

图 1-1 显示了 Cortex-M3 处理器的结构。

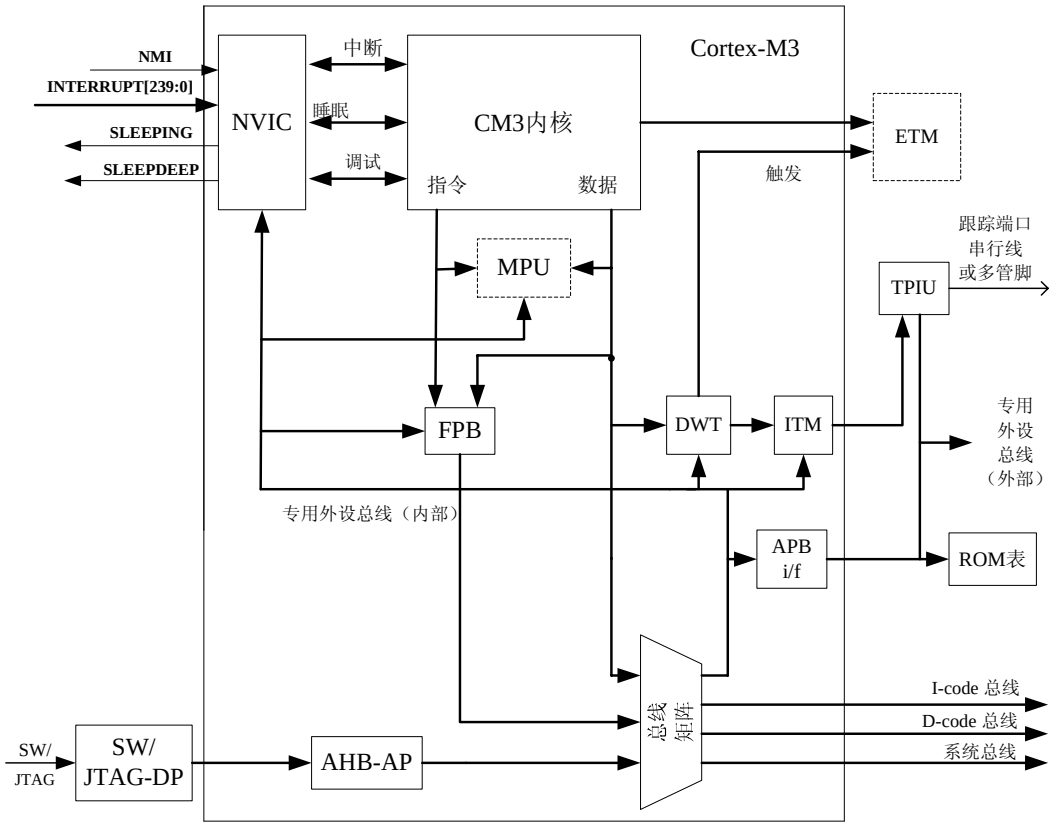


图 1-1 Cortex-M3 模块框图

注：ETM 和 MPU 是可选的，您的系统实现中可以不具有这两个组件。

1.2.1 Cortex-M3 的层次和实现的选项

如图 1-1 所示，Cortex-M3 处理器的组件存在于两个层次等级中。它向我们展示了设计的 RTL 层次。由于 ETM、TPIU、SW/JTAG-DP 和 ROM 表 4 个组件是可选的，或者它们的实现和使用具有灵活性（或者执行和使用它们的方式非常灵活），因此这 4 个组件位于 Cortex-M3 处理器的外面。您的实现可能与图 1-1 中显示的有所不同。可能的系统实现的选项见下面的 3 个部分。

TPIU

TPIU 的实现选项：

- 如果您的系统中有 ETM，则会含有 TPIU 格式程序，否则就不包含该格式程序。
- 一个多内核的实现可使用单个或多个 TPIU 来跟踪。
- ARM TPIU 模块可以用兼容 TPIU 的指定合作伙伴的 CoreSight 取代。
- 在生产设备中，TPIU 可以移除。

注：如果将 TPIU 移除，则 Cortex-M3 不具有跟踪功能。

SW/JTAG-DP

SW/JTAG-DP 的实现选项：

- 您的实现可以含有 SW-DP 或 JTAG-DP 中的任一个或两者都有。
- ARM SW-DP 可以被兼容 SW-DP 的指定合作伙伴的 CoreSight 取代。

- ARM JTAG-DP 可以被兼容 JTAG-DP 的指定合作伙伴的 CoreSight 取代。
- SW-DP 或 JTAG-DP 可以包含指定合作伙伴的测试接口。

ROM 表

如果系统中添加了附加的调试元件，则 ROM 存储器表中的描述需进行修改。

1.2.2 处理器内核

Cortex-M3 处理器内核采用 ARMv7-M 架构，其主要特性如下：

- Thumb-2 指令集架构（ISA）的子集，包含所有基本的 16 位和 32 位 Thumb-2 指令。
- 哈佛处理器架构，在加载/存储数据的同时能够执行指令取指。
- 三级流水线
- 32 位单周期乘法
- 硬件除法
- Thumb 状态和调试状态
- 处理模式和线程模式
- ISR 的低延迟进入和退出
  - 无需多余指令就可实现处理器状态的保存和恢复。在保存状态的同时从存储器中取出异常向量，实现更加快速地进入 ISR。
  - 中断控制器的紧密式耦合接口，能够有效地处理迟来中断。
  - 采用末尾连锁（tail-chaining）中断技术，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，处理背对背中断（back-to-back interrupt）。
- 可中断-可继续（interruptible-continued）的 LDM/STM，PUSH/POP。
- ARMv6 类型 BE8/LE 支持
- ARMv6 非对齐访问

处理器内核的相关内容将在下面各章中进一步描述。

寄存器

Cortex-M3 处理器包含：

- 13 个通用的 32 位寄存器
- 链接寄存器(LR)
- 程序计数器（PC）
- 程序状态寄存器，xPSR
- 两个分组的 SP 寄存器

存储器接口

Cortex-M3 处理器采用哈佛接口，在数据加载/存储的同时能够执行指令取指。存储器访问由下面的部件控制：

- 一个独立的加载存储单元（LSU），与来自 ALU 的加载和存储操作是分离的。
- 一个 3 字的入口预取指单元（entry prefetch unit）。一次取一个字。在取这个字时，

可以使用 2 种 thumb 指令，字对齐的 thumb-2 指令或半字对齐的 thumb-2 指令的高/低半字。所有来自内核的取地址操作都是字对齐的。如果是半字对齐的，则需要两次取指操作才能完成 thumb-2 指令的取指。而 3 字入口的预取指缓冲区确保了只有第一个被取出的半字的 thumb-2 指令才需要一个暂停周期（stall cycle）。

1.2.3 NVIC

NVIC 与处理器内核是紧密耦合的，这样可简化低延迟的异常处理。NVIC 主要特性包括：

- 外部中断可配置为 1～240 个
- 优先级的位可配置为 3～8 位
- 支持电平和边沿中断
- 中断优先级可动态地重新配置
- 优先级分组
- 支持末尾连锁（tail-chaining）中断技术
- 处理器状态在进入中断时自动保存，中断返回时自动恢复，不需要多余的指令。

NVIC 的详细描述见第 8 章*嵌套向量中断控制器*。

1.2.4 总线矩阵

总线矩阵用来将处理器和调试接口与外部总线相连。总线矩阵与下面的外部总线相连：

- ICode 总线，该总线用于从代码空间取指令和向量，是 32 位 AHBLite 总线。
- DCode 总线，该总线用于对代码空间进行数据加载/存储以及调试访问，是 32 位 AHBLite 总线。
- 系统总线，该总线用于对系统空间执行取指令和向量，数据加载/存储以及调试访问，是 32 位 AHBLite 总线。
- PPB，该总线用于对 PPB 空间进行数据加载/存储以及调试访问，是 32 位 APB（v2.0）总线。

总线矩阵还对以下方面进行控制：

- 非对齐访问。总线矩阵将非对齐的处理器访问转换为对齐访问。
- Bit-banding。总线矩阵将 bit-band 别名访问转换为对 bit-band 区的访问。它执行以下功能：

  —对 bit-band 加载进行位域提取

  —对 bit-band 存储进行原子读－修改－写
- 写缓冲。总线矩阵包含一个单入口写缓冲区，该缓冲区使得处理器内核不受到总线延迟的影响。

总线接口的详细描述见第 14 章*总线接口*。

1.2.5 FPB

FPB 单元实现硬件断点以及从代码空间到系统空间的修补访问，FPB 有 8 个比较器：

- 6 个指令比较器，能够单独配置，实现将代码空间的指令取指重新映射到系统空间，或实现硬件断点。
- 两个 literal 比较器，将代码空间的 literal 访问重新映射到系统空间。

FPB 的详细描述见第 11 章*系统调试*。

1.2.6 DWT

DWT 单元集成了以下调试功能：

- DWT 包含 4 个比较器，其中的任一个都可配置为硬件观察点、ETM 触发、PC 采样事件触发、或数据地址采样事件触发。
- 包含几个用于性能分析（performance profiling）的计数器。
- 能够配置为以定义的间隔发送 PC 采样，以及发送中断事件信息。

DWT 的详细描述见第 11 章*系统调试*。

1.2.7 ITM

ITM 是一个应用导向（application driven）的跟踪源，支持对应用事件的跟踪和 *printf* 类型的调试。

ITM 提供以下跟踪信息源：

- 软件跟踪，软件能够直接写入 ITM 激励寄存器（stimulus register），从而完成信息包的发出操作。
- 硬件跟踪，这些信息包由 DWT 产生，并由 ITM 发出。
- 时间戳（time stamping），时间戳要根据信息包来发送。

TIM 的详细描述见第 11 章*系统调试*。

1.2.8 MPU

如果希望向处理器提供存储器保护，则可以使用可选的 MPU。MPU 对访问允许和存储器属性进行检验。它包含 8 个区和一个可选的执行默认存储器映射访问属性的背景区。

MPU 的详细描述见第 9 章*存储器保护单元*。

1.2.9 ETM

ETM 是只支持指令跟踪的低成本跟踪宏单元。

ETM 的详细描述见第 15 章*嵌入式跟踪宏单元*。

1.2.10 TPIU

TPIU 用作来自 ITM 和 ETM（如果存在）的 Cortex-M3 跟踪数据与片外跟踪端口分析仪之间的桥接。TPIU 可配置为支持低成本调试的串行管脚跟踪，或用于更高带宽跟踪的多管脚跟踪。TPIU 与 CoreSight 是兼容的。

TPIU 的详细描述见第 13 章*跟踪端口的接口单元*。

1.2.11 SW/JTAG-DP

Cortex-M3 处理器可配置为具有 SW-DP 或 JTAG-DP 调试端口的接口，或两者都有。这两个调试端口提供对系统中包括处理器寄存器在内的所有寄存器和存储器的调试访问。

SW/JTAG-DP 的详细描述见第 12 章*调试端口*。

1.3 可配置的选项

这节描述处理器的配置选项，若想确认您的实现的配置，请联系您的 implementor。

1.3.1 中断

外部中断可配置为 1～240 个。

中断优先级的位可配置为 3～8 位。

1.3.2 MPU

Cortex-M3 系统可配置为包含一个 MPU。

MPU 的详细描述见第 9 章*存储器保护单元*。

1.3.3 ETM

Cortex-M3 系统可配置为包含一个 ETM。

ETM 的详细描述见第 16 章*嵌入式跟踪宏单元接口*。

1.4 指令集汇总

这节内容包括：

- 处理器 16 位指令汇总
- 处理器 32 位指令汇总

表 1-1 列出了 16 位 Cortex-M3 指令。

表 1-1 16 位 Cortex-M3 指令汇总

操作	汇编指令
寄存器值与寄存器值及 C 标志相加	ADC <Rd>, <Rm>
3 位立即数与寄存器值相加	ADD <Rd>, <Rn>, #<immed_3>
8 位立即数与寄存器值相加	ADD <Rd>, #<immed_8>
低寄存器值与低寄存器值相加	ADD <Rd>, <Rn>, <Rm>
高寄存器值与低或高寄存器值相加	ADD <Rd>, <Rm>
PC 加 4（8 位立即数）	ADD <Rd>, PC, #<immed_8>*4
SP 加 4（8 位立即数）	ADD <Rd>, SP, #<immed_8>*4
SP 加 4（7 位立即数）	ADD <Rd>, SP, #<immed_7>*4 或 ADD SP, SP, #<immed_7>*4
寄存器值按位与	AND <Rd>, <Rm>
算术右移，移位次数取决于立即数值	ASR <Rd>, <Rm>, #<immed_5>
算术右移，移位次数取决于寄存器中的值	ASR <Rd>, <Rs>

续表 1-1

操作	汇编指令
条件分支	B<cond> <target address>
无条件分支	B<tartet address>
位清零	BIC <Rd>, <Rs>
软件断点	BKPT <immed_8>
带链接分支	BL <Rm>
比较结果不为零时分支	CBNZ <Rn>, <label>
比较结果为零时分支	CBZ <Rn>, <Rm>
将寄存器值取反与另一个寄存器值比较	CMN <Rn>, <Rm>
与 8 位立即数比较	CMP <Rn>, #<immed_8>
寄存器比较	CMP <Rn>, <Rm>
高寄存器与高或低寄存器比较	CMP <Rn>, <Rm>
改变处理器状态	CPS <effect>, <iflags>
将高或低寄存器的值复制到另一个高或低寄存器中	CPY <Rd>, <Rm>
寄存器的值按位异或	EOR <Rd>, <Rm>
以下一条指令为条件，以下面两条指令为条件，以下面三条指令为条件，以下面四条指令为条件	IT<cond> IT<x> <cond> IT<x><y> <cond> IT<x><y><z> <cond>
多个连续的存储器字加载	LDMIA <Rn>!, <register>
将基址寄存器与 5 位立即数偏移的和的地址处的数据加载到寄存器中	LDR <Rd>, [<Rn>, #<immed_5*4>]
将基址寄存器与寄存器偏移的和的地址处的数据加载到寄存器中	LDR <Rd>, [<Rn>, <Rm>]
将 PC 与 8 位立即数偏移的和的地址处的数据加载到寄存器中	LDR <Rd>, [PC, #<immed_8>*4]
将 SP 与 8 位立即数偏移的和的地址处的数据加载到寄存器中	LDR <Rd>, [SP, #<immed_8>*4]
将寄存器与 5 位立即数偏移的和的地址处的字节[7:0]加载到寄存器中	LDRB <Rd>, [<Rn>, #<immed_5>]
将寄存器与寄存器偏移的和的地址处的字节[7:0]加载到寄存器中	LDRB <Rd>, [<Rn>, <Rm>]
将寄存器与 5 位立即数偏移的和的地址处的半字[15:0]加载到寄存器中	LDRH <Rd>, [<Rn>, #<immed_5>*2]
将寄存器与寄存器偏移的和的地址处的半字[15:0]加载到寄存器中	LDRH <Rd>, [<Rn>, <Rm>]
将寄存器与寄存器偏移的和的地址处的带符号字节[7:0]加载到寄存器中	LDRSB <Rd>, [<Rn>, <Rm>]
将寄存器与寄存器偏移的和的地址处的带符号半字[15:0]加载到寄存器中	LDRSH <Rd>, [<Rn>, <Rm>]
逻辑左移，移位次数取决于立即数值	LSL <Rd>, <Rm>, #<immed_5>
逻辑左移，移位次数取决于寄存器中的值	LSL <Rd>, <Rs>
逻辑右移，移位次数取决于立即数值	LSR <Rd>, <Rm>, #<immed_5>
逻辑右移，移位次数取决于寄存器中的值	LSR <Rd>, <Rs>
将 8 位立即数传送到目标寄存器	MOV <Rd>, #<immed_8>

续表 1-1

操作	汇编指令
将低寄存器值传送给低目标寄存器	MOV <Rd>, <Rn>
将高或低寄存器值传送给高或低目标寄存器	MOV <Rd>, <Rm>
寄存器值相乘	MUL <Rd>, <Rm>
将寄存器值取反后传送给目标寄存器	MVN <Rd>, <Rm>
将寄存器值取负并保存在目标寄存器中	NEG <Rd>, <Rm>
无操作	NOP <C>
将寄存器值按位作逻辑或操作	ORR <Rd>, <Rm>
寄存器出栈	POP <寄存器>
寄存器和 PC 出栈	POP <寄存器，PC>
寄存器压栈	PUSH <registers>
寄存器和 LR 压栈	PUSH <registers, LR>
将字内的字节逆向（reverse）并复制到寄存器中	REV <Rd>, <Rn>
将两个半字内的字节逆向并复制到寄存器中	REV16 <Rd>, <Rn>
将低半字[15:0]内的字节逆向并将符号位扩展，复制到寄存器中。	REVSH <Rd>, <Rn>
循环右移，移位次数由寄存器中的值标识	ROR <Rd>, <Rs>
寄存器中的值减去寄存器值和C标志	SBC <Rd>, <Rm>
发送事件	SEV <c>
将多个寄存器字保存到连续的存储单元中	STMIA <Rn>!, <registers>
将寄存器字保存到寄存器与5位立即数偏移的和的地址中	STR <Rd>, [<Rn>, #<immed_5>*4]
将寄存器字保存到寄存器地址中	STR <Rd>, [<Rn>, <Rm>]
将寄存器字保存到SP与8位立即数偏移的和的地址中	STR <Rd>, [SP, #<immed_8> * 4]
将寄存器字节[7:0]保存到寄存器与5位立即数偏移的和的地址中	STRB <Rd>, [<Rn>, #<immed_5>]
将寄存器字节[7:0]保存到寄存器地址中	STRB <Rd>, [<Rn>, <Rm>]
将寄存器半字[15:0]保存到寄存器与5位立即数偏移的和的地址中	STRH <Rd>, [<Rn>, #<immed_5> * 2]
将寄存器半字[15:0]保存到寄存器地址中	STRH <Rd>, [<Rn>, #<immed_5> * 2]
寄存器值减去3位立即数	STRH <Rd>, [<Rn>, #<immed_5> * 2]
寄存器值减去8位立即数	SUB <Rd>, #<immed_8>
寄存器值减去寄存器值	SUB <Rd>, <Rn>, <Rm>
SP减4（7位立即数）	SUB SP, #<immed_7> * 4
操作系统服务调用，带8位立即数调用代码	SVC <immed_8>
从寄存器中提取字节[7:0]，传送到寄存器中，并用符号位扩展到32位	SXTB <Rd>, <Rm>
从寄存器中提取半字[15:0]，传送到寄存器中，并用符号位扩展到32位	SXTH <Rd>, <Rm>
将寄存器与另一个寄存器相与，测试寄存器中的置位的位	TST <Rn>, <Rm>
从寄存器中提取字节[7:0]，传送到寄存器中，并用零位扩展到32位	UXTB <Rd>, <Rm>

续表 1-1

操作	汇编指令
从寄存器中提取半字[15:0]，传送到寄存器中，并用零位扩展到32位	UXTH <Rd>, <Rm>
等待事件	WFE <c>
等待中断	WFI <c>

表 1-2 列出了 32 位 Coxtex-M3 指令。

表 1-2 32 位 Coxtex-M3 指令汇总

操作	汇编指令
寄存器值与12位立即数及C位相加	ADC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12>
寄存器值与移位后的寄存器值及C位相加	ADC{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
寄存器值与12位立即数相加	ADD{S}.W <Rd>, <Rn>,#<modify_constant(immed_12)>
寄存器值与移位后的寄存器值相加	ADD{S}.W <Rd>, <Rm>{, <shift>}
寄存器值与12位立即数相加	ADDW.W <Rd>, <Rn>, #<immed_12>
寄存器值与12位立即数按位与	AND{S}.W <Rd>, <Rn>, #<modify_constant(immed_12>
寄存器值与移位后的寄存器值按位与	AND{S}.W <Rd>, <Rn>, Rm>{, <shift>}
算术右移，移位次数取决于寄存器值	ASR{S}.W <Rd>, <Rn>, <Rm>
条件分支	B{cond}.W <label>
位区清零	BFC.W <Rd>, #<lsb>, #<width>
将一个寄存器的位区插入另一个寄存器中	BFI.W <Rd>, <Rn>, #<lsb>, #<width>
12位立即数取反与寄存器值按位与	BIC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
移位后的寄存器值取反与寄存器值按位与	BIC{S}.W <Rd>, <Rn>, {, <shift>}
带链接的分支	BL <label>
带链接的分支（立即数）	BL<c> <label>
无条件分支	B.W <label>
返回寄存器值中零的数目	CLZ.W <Rd>, <Rn>
寄存器值与12位立即数两次取反后的值比较	CMN.W <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值两次取反后的值比较	CMN.W <Rn>, <Rm>{, <shift>}
寄存器值与12位立即数比较	CMP.W <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值比较	CMP.W <Rn>, <Rm>{, <shift>}
数据存储器排序（barrier）	DMB <c>
数据同步排序（barrier）	DSB <c>
寄存器值与12位立即数作异或操作	EOR{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值作异或操作	EOR{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
指令同步排序（barrier）	ISB <c>
多存储器寄存器加载，加载后加1或加载前减1	LDM{IA|DB}.W <Rn>{!}, <registers>
保存寄存器地址与12位立即数偏移的和的地址处的数据字	LDR.W <Rxf>, [<Rn>, #<offset_12>]
将寄存器地址与12位立即数偏移的和的地址处的数据字保存到PC中	LDR.W PC, [<Rn>, #<offset_12>]

续表 1-2

操作	汇编指令
将基址寄存器地址的8位立即数偏移的地址处的数据字保存到PC中，后索引	LDR.W PC, #<+/-<offset_8>
保存基址寄存器地址的8位立即数偏移的地址处的数据字，后索引	LDR.W <Rxf>, [<Rn>], #+/–<offset_8>
保存基址寄存器地址的8位立即数偏移的地址处的数据字，前索引	LDR.W <Rxf>, [<Rn>, #<+/–<offset_8>]!
将基址寄存器地址的8位立即数偏移的地址处的数据字保存到PC中，前索引	LDR.W PC, [<Rn>, #+/–<offset_8>]!
保存寄存器地址左移0，1，2或3个位置后的地址处的数据字	LDR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
将寄存器地址左移0，1，2或3个位置后的地址处的数据字保存到PC中	LDR.W PC, [<Rn>, <Rm>{, LSL #<shift>}]
保存PC地址的12位立即数偏移的地址处的数据字	LDR.W <Rxf>, [PC, #+/–<offset_12>]
将PC地址的12位立即数偏移的地址处的数据字保存到PC中	LDR.W PC, [PC, #+/–<offset_12>]
保存基址寄存器地址与12位立即数偏移的和的地址处的字节[7:0]	LDRB.W <Rxf>, [<Rn>, #<offset_12>]
保存基址寄存器地址的8位立即数偏移的地址处的字节[7:0]，后索引	LDRB.W <Rxf>. [<Rn>], #+/-<offset_8>
保存寄存器地址左移0，1，2或3个位置后的地址处的字节[7:0]	LDRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
保存基址寄存器地址的8位立即数偏移的地址处的字节[7:0]，前索引	LDRB.W <Rxf>, [<Rn>, #<+/–<offset_8>]!
保存PC地址的12位立即数偏移的地址处的字节	LDRB.W <Rxf>, [PC, #+/–<offset_12>]
保存寄存器地址8位偏移4的地址处的双字，前索引	LDRD.W <Rxf>, <Rxf2>, [<Rn>, #+/–<offset_8> * 4]{!}
保存寄存器地址8位偏移4的地址处的双字，后索引	LDRD.W <Rxf>, <Rxf2>, [<Rn>], #+/–<offset_8> * 4
保存基址寄存器地址与12位立即数偏移的和的地址处的半字[15:0]	LDRH.W <Rxf>, [<Rn>, #<offset_12>]
保存基址寄存器地址的8位立即数偏移的地址处的半字[15:0]，前索引	LDRH.W <Rxf>, [<Rn>, #<+/–<offset_8>]!
保存基址寄存器地址的8位立即数偏移的地址处的半字[15:0]，后索引	LDRH.W <Rxf>. [<Rn>], #+/-<offset_8>
保存基址寄存器地址左移0，1，2或3个位置后的地址处的半字[15:0]	LDRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
保存PC地址的12位立即数偏移的地址处的半字	LDRH.W <Rxf>, [PC, #+/–<offset_12>]

续表 1-2

操作	汇编指令
保存基址寄存器地址与12位立即数偏移的和的地址处的带符号字节[7:0]	LDRSB.W <Rxf>, [<Rn>, #<offset_12>]
保存基址寄存器地址的8位立即数偏移的地址处的带符号字节[7:0]，后索引	LDRSB.W <Rxf>. [<Rn>], #+/-<offset_8>
保存基址寄存器地址的8位立即数偏移的地址处的带符号字节[7:0]，前索引	LDRSB.W <Rxf>, [<Rn>, #<+/–<offset_8>]!
保存寄存器地址左移0，1，2或3个位置后的地址处的带符号字节[7:0]	LDRSB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
保存PC地址的12位立即数偏移的地址处的带符号字节	LDRSB.W <Rxf>, [PC, #+/–<offset_12>]
保存基址寄存器地址与12位立即数偏移的和的地址处的带符号半字[15:0]	LDRSH.W <Rxf>, [<Rn>, #<offset_12>]
保存基址寄存器地址的8位立即数偏移的地址处的带符号半字[15:0]，后索引	LDRSH.W <Rxf>. [<Rn>], #+/-<offset_8>
保存基址寄存器地址的8位立即数偏移的地址处的带符号半字[15:0]，前索引	LDRSH.W <Rxf>, [<Rn>, #<+/–<offset_8>]!
保存寄存器地址左移0，1，2或3个位置后的地址处的带符号半字[15:0]	LDRSH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
保存PC地址的12位立即数偏移的地址处的带符号半字	LDRSH.W <Rxf>, [PC, #+/–<offset_12>]
逻辑左移，移位次数由寄存器中的值标识	LSL{S}.W <Rd>, <Rn>, <Rm>
逻辑右移，移位次数由寄存器中的值标识	LSR{S}.W <Rd>, <Rn>, <Rm>
将两个带符号或无符号的寄存器值相乘，并将低32位与寄存器值相加	MLA.W <Rd>, <Rn>, <Rm>, <Racc>
将两个带符号或无符号的寄存器值相乘，并将低32位与寄存器值相减	MLS.W <Rd>, <Rn>, <Rm>, <Racc>
将12位立即数传送到寄存器中	MOV{S}.W <Rd>, #<modify_constant(immed_12)>
将移位后的寄存器值传送到寄存器中	MOV{S}.W <Rd>, <Rm>{, <shift>}
将16位立即数传送到寄存器的高半字[31:16]中	MOVT.W <Rd>, #<immed_16>
将16位立即数传送到寄存器的低半字[15:0]中，并将高半字[31:16]清零	MOVW.W <Rd>, #<immed_16>
将状态传送到寄存器中	MRS<c> <Rd>, <psr>
传送到状态寄存器中	MSR<c> <psr>_<fields>,<Rn>
将两个带符号或不带符号的寄存器值相乘	MUL.W <Rd>, <Rn>, <Rm>
无操作	NOP.W
将寄存器值与12位立即数作逻辑“或非”操作	ORN{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
将寄存器值与移位后的寄存器值作逻辑“或非”操作	ORN[S}.W <Rd>, <Rn>, <Rm>{, <shift>}
将寄存器值与12位立即数作逻辑“或”操作	ORR{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)

续表 1-2

操作	汇编指令
将寄存器值与移位后的寄存器值作逻辑“或”操作	ORR{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
将位顺序逆向	RBIT.W <Rd>, <Rm>
将字内的字节逆向	REV.W <Rd>, <Rm>
将每个半字内的字节逆向	REV16.W <Rd>, <Rn>
将低半字内的字节逆向并用符号扩展	REVSH.W <Rd>, <Rn>
循环右移，移位次数取决于寄存器中的值	ROR{S}.W <Rd>, <Rn>, <Rm>
寄存器值与12位立即数相减	RSB{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值相减	RSB{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
寄存器值与12位立即数及C位相减	SBC{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值及C位相减	SBC{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
将所选的位复制到寄存器中并用符号扩展	SBFX.W <Rd>, <Rn>, #<lsb>, #<width>
带符号除法	SDIV<c> <Rd>,<Rn>,<Rm>
发送事件	SEV<c>
将带符号半字相乘并用符号扩展到2个寄存器值	SMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm>
两个带符号寄存器值相乘	SMULL.W <RdLo>, <RdHi>, <Rn>, <Rm>
带符号饱和操作	SSAT <c> <Rd>, #<imm>, <Rn>{, <shift>}
多个寄存器字保存到连续的存储单元中	STM{IA|DB}.W <Rn>{!}, <registers>
寄存器字保存到寄存器地址与12位立即数偏移的和的地址中	STR.W <Rxf>, [<Rn>, #<offset_12>]
寄存器字保存到寄存器地址的8位立即数偏移的地址中，后索引	STR.W <Rxf>, [<Rn>], #+/–<offset_8>
寄存器字保存到寄存器地址移位0，1，2或3个位置的地址中	STR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
寄存器字保存到寄存器地址的8位立即数偏移的地址中，前索引	STR{T}.W <Rxf>, [<Rn>, #+/–<offset_8>]{!}
寄存器字节[7:0]保存到寄存器地址的8位立即数偏移的地址中，前索引	STRB{T}.W <Rxf>, [<Rn>, #+/–<offset_8>]{!}
寄存器字节[7:0]保存到寄存器地址与12位立即数偏移的和的地址中	STRB.W <Rxf>, [<Rn>, #<offset_12>]
寄存器字节[7:0]保存到寄存器地址的8位立即数偏移的地址中，后索引	STRB.W <Rxf>, [<Rn>], #+/–<offset_8>
寄存器字节保存到寄存器地址移位0，1，2或3个位置的地址中	STRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]
存储双字，前索引	STRD.W <Rxf>, <Rxf2>, [<Rn>, #+/–<offset_8> * 4]{!}
存储双字，后索引	STRD.W <Rxf>, <Rxf2>, [<Rn>], #+/–<offset_8> * 4
寄存器半字[15:0]保存到寄存器地址与12位立即数偏移的和的地址中	STRH.W <Rxf>, [<Rn>, #<offset_12>]
寄存器半字保存到寄存器地址移位0，1，2或3个位置的地址中	STRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}]

续表 1-2

操作	汇编指令
寄存器半字保存到寄存器地址的8位立即数偏移的地址中，前索引	STRH{T}.W <Rxf>, [<Rn>, #+/–<offset_8>]{!}
寄存器半字保存到寄存器地址的8位立即数偏移的地址中，后索引	STRH.W <Rxf>, [<Rn>], #+/–<offset_8>
寄存器值与12位立即数相减	SUB{S}.W <Rd>, <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值相减	SUB{S}.W <Rd>, <Rn>, <Rm>{, <shift>}
寄存器值与12位立即数相减	SUBW.W <Rd>, <Rn>, #<immed_12>
将字节符号扩展到32位	SXTB.W <Rd>, <Rm>{, <rotation>}
将半字符号扩展到32位	SXTH.W <Rd>, <Rm>{, <rotation>}
表格分支字节	TBB [<Rn>, <Rm>]
表格分支半字	TBH [<Rn>, <Rm>, LSL #1]
寄存器值与12位立即数作逻辑“异或”操作	TEQ.W <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值作逻辑“异或”操作	TEQ.W <Rn>, <Rm>{, <shift}
寄存器值与12位立即数作逻辑“与”操作	TST.W <Rn>, #<modify_constant(immed_12)>
寄存器值与移位后的寄存器值作逻辑“与”操作	TST.W <Rn>, <Rm>{, <shift>}
将寄存器的位区复制到寄存器中，并用零扩展到32位	UBFX.W <Rd>, <Rn>, #<lsb>, #<width>
无符号除法	UDIV<c> <Rd>,<Rn>,<Rm>
两个无符号寄存器值相乘并与两个寄存器值相加	UMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm>
两个无符号寄存器值相乘	UMULL.W <RdLo>, <RdHi>, <Rn>, <Rm>
无符号饱和操作	USAT <c> <Rd>, #<imm>, <Rn>{, <shift>}
将无符号字节复制到寄存器中并用零扩展到32位	UXTB.W <Rd>, <Rm>{, <rotation>}
将无符号半字复制到寄存器中并用零扩展到32位	UXTH.W <Rd>, <Rm>{, <rotation>}
等待事件	WFE.W
等待中断	WFI.W

第2章 编程模型（programmer's model）

本章将描述 Cortex-M3 处理器的编程模型，包含以下内容：

- 关于编程模型
- 特权访问和用户访问
- 寄存器
- 数据类型
- 存储器格式
- 指令集

2.1 关于编程模型

Cortex-M3 处理器采用 ARM v7-M 架构。它包括所有的 16 位 thumb 指令集和基本的 32 位 thumb-2 指令集架构。Cortex-M3 处理器不能执行 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。

2.1.1 工作模式

Cortex-M3 处理器支持两种工作模式，线程模式和处理模式：

- 在复位时处理器进入线程模式，异常返回时也会进入该模式。特权和用户（非特权）代码能够在线程模式下运行。
- 出现异常时处理器进入处理模式，在处理模式中，所有代码都是特权访问的。

2.1.2 工作状态

Cortex-M3 处理器有两种工作状态：

- 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 特权访问和用户访问

代码可以是特权执行或非特权执行。非特权执行时对有些资源的访问受到限制或不允许访问。特权执行可以访问所有资源。处理模式始终是特权访问，线程模式可以是特权或非特权访问。

线程模式在复位之后为特权访问，但可通过 MSR 指令清零 CONTROL[0]，将它配置为用户（非特权）访问。用户访问禁止：

- 部分指令的使用，例如设置 FAULTMASK 和 PRIMASK 的 CPS 指令。
- 对系统控制空间（SCS）的大部分寄存器的访问。

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。只有处理操作能够改变线程模式的访问特权。处理模式始终是特权访问的。

2.2.1 主堆栈和进程堆栈

结束复位后，所有代码都使用主堆栈。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。所有异常继续使用主堆栈，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，由 r13 指示。

除了使用从处理模式退出时的 EXC_RETURN 的值外，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。

2.3 寄存器

Cortex-M3 处理器有以下 32 位寄存器：

- 13 个通用寄存器，r0～r12
- 分组的堆栈指针别名，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 显示了 Cortex-M3 的寄存器集。

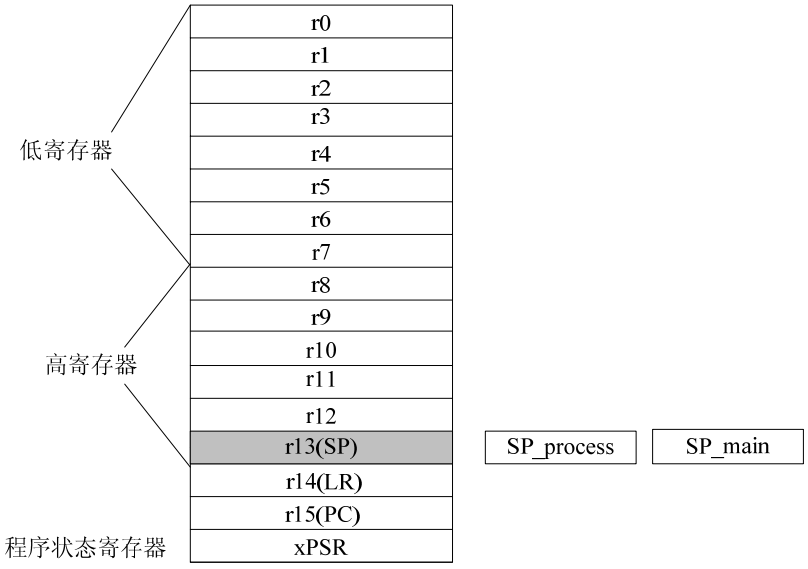


图 2-1 Cortex-M3 的寄存器集

2.3.1 通用寄存器

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。

低寄存器　　寄存器 r0-r7 可以被指定通用寄存器的所有指令访问。

高寄存器　　寄存器 r8-r12 可以被指定通用寄存器的所有 32 位指令访问。

寄存器 r8-r12 不能被 16 位指令访问。

寄存器 r13、r14、r15 具有以下特殊功能：

堆栈指针　寄存器 r13 用作堆栈指针（SP）。由于 SP 忽略了写入位[1:0]的值，因此它自动与字，即 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。

链接寄存器　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。

其它任何时候都可以将 r14 看作一个通用寄存器。

程序计数器　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。

2.3.2 特殊用途的程序状态寄存器（xPSR）

系统级的处理器状态可分为 3 类，因此有 3 个程序状态寄存器。对程序状态寄存器的访问使用 MRS 和 MSR 指令，在访问时可以把它们作为单独的寄存器，3 个中的任两个组合，或 3 个组合。这 3 个寄存器为：

- 应用 PSR
- 中断 PSR
- 执行 PSR

应用 PSR

应用 PSR(APSR)包含条件代码标志。在进入异常之前，Cortex-M3 处理器将条件代码标志保存在堆栈内。您可以使用 MSR(2)和 MRS(2)指令来访问 APSR。

APSR 的位分配如图 2-2 所示。

图 2-2 应用程序状态寄存器的位分配

表 2-1 描述了 APSR 的位分配。

表 2-1 应用程序状态寄存器的位分配

位	名称	定义
[31]	N	负数或小于标志： 1：结果为负数或小于 0：结果为正数或大于
[30]	Z	零标志： 1：结果为 0 0：结果为非 0

续表 2-1

位	名称	定义
[29]	C	进位/借位标志： 1：进位或借位 0：没有进位或借位
[28]	V	溢出标志： 1：溢出 0：没有溢出
[27]	Q	粘着饱和（sticky saturation）标志
[26:0]	-	保留

中断 PSR

中断 PSR（IPSR）包含当前激活的异常的 ISR 编号。

IPSR 的位分配如图 2-3 所示。

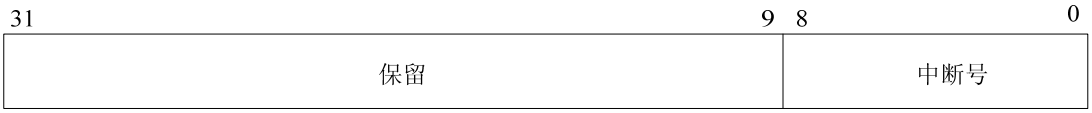


图 2-3 中断程序状态寄存器的位分配

表 2-2 描述了 IPSR 的位分配。

表 2-2 中断程序状态寄存器的位分配

位	名称	定义
[31:9]	-	保留
[8:0]	ISR NUMBER	占先异常的编号： 基础级别=0 NMI=2 SVCall＝11 INTISR[0]=16 INTISR[1]=17 . . . INTISR[15]=31 . . .INTISR[239]=255

执行 PSR

执行 PSR(EPSR)包含两个重叠的区域：

- 可中断-可继续(interruptible-continuable)指令（ICI）区，用于被打断的多寄存器加载和存储指令。

- 用于 If-Then（IT）指令的执行状态区，以及 T 位（Thumb 状态位）。

可中断-可继续指令（ICI）区

多寄存器加载（LDM）和存储（STM）操作是可中断的。EPSR 的 ICI 区用来保存从产生中断的点继续执行多寄存器加载和存储操作时所必需的信息。

If-then 状态区

EPSR 的 IT 区包含了 If-Then 指令的执行状态位。

注：

ICI 区和 IT 区是重叠的，因此，If-Then 模块内的多寄存器加载或存储操作不具有可中断-可继续功能。

EPSR 的位分配如图 2-4 所示。

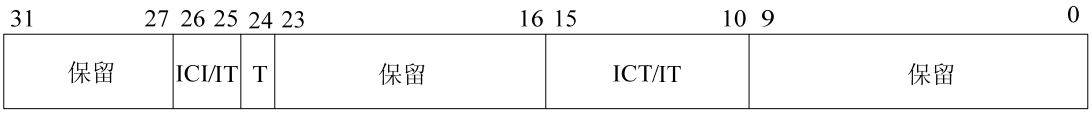


图 2-4 执行程序状态寄存器

不能直接访问 EPSR，若想修改 EPSR 必须发生以下两个事件之一：

- 在执行 LDM 或 STM 指令时产生一次中断
- 执行 If-Then 指令

表 2-3 描述了 EPSR 的位分配。

表 2-3 执行 PSR 的位功能

位	名称	定义
[31:27]	-	保留
[15:12]	ICI	可中断-可继续的指令位。如果在执行 LDM 或 STM 操作时产生一次中断，则 LDM 或 STM 操作暂停。EPSR 使用位[15:12]来保存该操作中下一个寄存器操作数的编号。在中断响应之后，处理器返回由[15:12]指向的寄存器并恢复操作。如果 ICI 区指向的寄存器不在指令的寄存器列表中，则处理器对列表中的下一个寄存器（如果有）继续执行 LDM/STM 操作。
[15:10]:[26:25]	IT	If-Then 位。它们是 If-Then 指令的执行状态位。包含 If-Then 模块的指令数目和它们的执行条件。
[24]	T	T 位使用一条可相互作用的指令来清零，这里写入的 PC 的位 0 为 0。也可以使用异常出栈操作来清零，被压栈的 T 位为 0。 当 T 位为零时执行指令会引起 INVSTATE 异常。
[23:16]	-	保留
[9:0]	-	保留

LDM 和 STM 操作中的基址寄存器更新

以下是 LDM 或 STM 更新基址寄存器的情况：

- 当指令指定了基址寄存器回写操作时，基址寄存器变为更新后的地址。一次中止（abort）能够恢复为原来的基地址。
- 当基址寄存器在 LDM 的寄存器列表中并且不是列表中的最后一个寄存器时，基址寄存器变为被加载的值。

如果出现下列情况，LDM/STM 操作重新开始而不是继续执行：

- LDM/STM 错误
- LDM/STM 指令位于 IT 内

如果 LDM 已经完成基地址加载操作，它会从基地址加载之前的地址继续执行。

保存 xPSR 位

在进入异常时，处理器将 3 个状态寄存器组合的信息保存在堆栈内。

2.4 数据类型

Cortex-M3 处理器支持以下数据类型：

- 32 位字
- 16 位半字
- 8 位字节

注：

存储器系统应该支持所有的数据类型。尤其是要求在不破坏一个字中的相邻字节的情况下支持小于 1 个字（subword）的写操作。

2.5 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或大端格式访问存储器中的数据字，而访问代码时始终使用小端格式。

注：

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。

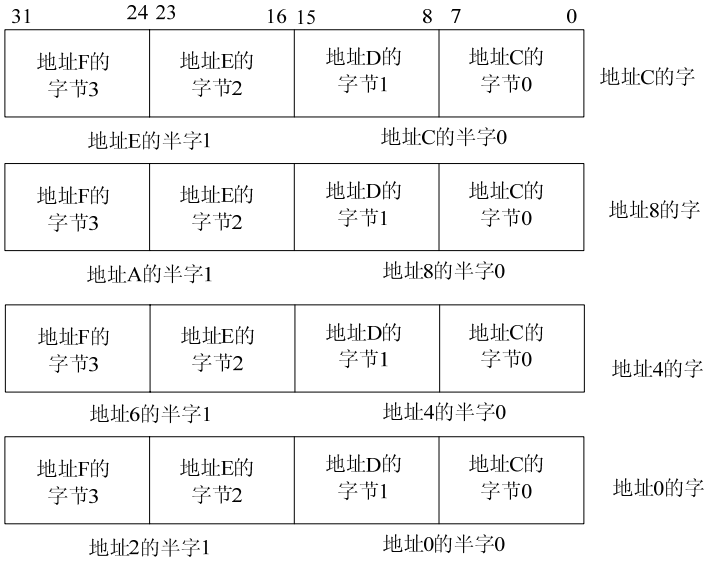
图 2-5 显示了小端格式和大端格式的区别。

Cortex-M3 处理器有一个配置管脚 **BIGEND**，您能够使用它来选择小端格式或 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。

注：

- 对系统控制空间（SCS）的访问始终采用小端格式。
- 在非复位的状态下试图改变存储器格式的操作将被忽略。
- PPB 空间只能为小端格式，**BIGEND** 的设置无效。

小端数据格式



大端数据格式

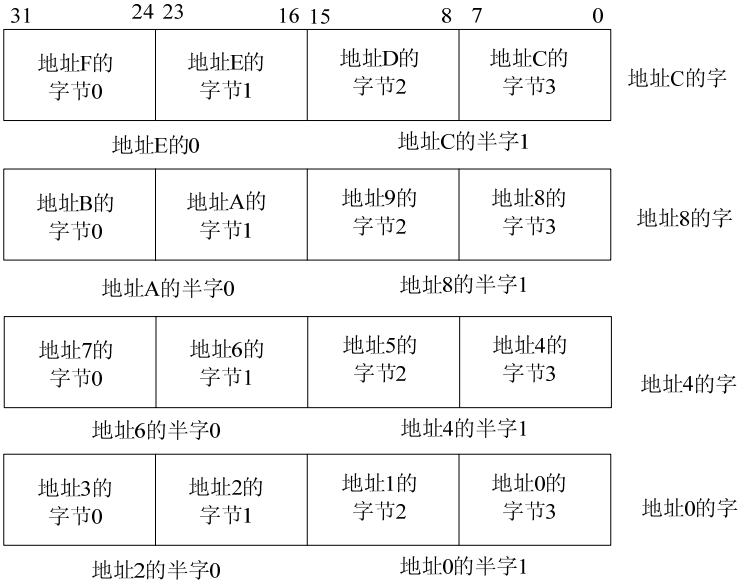


图 2-5 存储器的小端格式和大端格式

2.6 指令集

Cortex-M3 处理器不支持 ARM 指令。

Cortex-M3 处理器支持所有的 ARMv6 Thumb 指令，下表 2-4 中列出的除外。

表 2-4 Cortex-M 不支持的 Thumb 指令

指令	执行时产生的动作
BLX(1) 带链接和交换的分支指令	BLX(1)一直出错
SETEND 设置存储器格式	SETEND 一直出错。使用配置管脚选择 Cortex-M3 的存储器格式。

Cortex-M3 处理器支持表 2-5 中列出的 Thumb-2 指令。

表 2-5 Cortex-M3 支持的 Thumb-2 指令

指令类型	大小	指令
数据操作	16	ADC,ADD,AND,ASR,BIC,CMN,CMP,CPY,EOR,LSL,LSR,MOV,MUL,MVN,NEG,ORR,ROR,SBC,SUB,TST,REV,REVH,REVSH,SXTB,SXTH,UXTB,UXTH
分支	16	B<cond>,B,BL,BX,BLX.。注：不支持带立即数的 BLX 指令
单寄存器加载-存储	16	LDR,LDRB,LDRH,LDRSB,LDRSH,STR,STRB,STRH,T 变量
多寄存器加载-存储	16	LDMIA,POP,PUSH,STMIA
异常产生	16	BKPT，如果调试使能，程序停止，进入调试状态，如果调试禁止，则出错。SVC，出现 SVC 故障则调用 SVCall 处理程序。
带立即数的数据操作	32	ADC{C},ADD{S},CMN,RSB{S},SBC{S},SUB{S},CMP,AND{S},TST,BIC{S},EOR{S},TEQ,ORR{S},MOV{S},ORE{S},MVN{S}
带大立即数的数据操作	32	MOVW,MOVT,ADDW,SUBW。 MOVW 和 MOVT 带有 16 位立即数。这意味着它们能代替来自存储器的 literal 加载。 ADDW 和 SUBW 带有 12 位立即数。这意味着它们能代替多个来自存储器的 literal 加载。
位域操作	32	BFI,BFC,UBFX,SBFX。这些指令都是按位来操作的，使能对位的位置和大小的控制。除了许多比较和一些 AND/OR 赋值表达式之外，它们还都支持 C/C++位区（在 structs 中）。
带 3 个寄存器的数据操作	32	ADC{S},ADD{S},CMN,RSB{S},SBC{S},SUB{S},CMP,AND{S},TST,BIC{S},EOR{S},TEQ,ORR{S},MOV{S},ORN{S},MVN{S} 。 不支持 PKxxx 指令。
移位操作	32	ASR{S},LSL{S},LSR{S},ROR{S}
杂项	32	REV,REVH,REVSH,RBIT,CLZ,SXTB,SXTH,UXTB,UXTH。扩展指令与对应的 v6 16 位指令相同。
表格分支	32	TBB 和 TBH 表格分支，用于 switch/case。这些指令都是带移位的 LDR 操作，然后进行分支。
乘法	32	MUL,MLA,MLS
64 位结果的乘法	32	UMULL,SMULL,UMLAL,SMLAL

续表 2-5

指令类型	大小	指令
加载-存储寻址	32	支持以下格式：PC+/-imm12，Rbase+imm12，Rbase+/-imm8，以及包括移位的调整寄存器。 在特权模式下使用的 T 变量。
单寄存器加载-存储	32	LDR,LDRB,LDRSB,LDRH,LDRSH,STR,STRB,STRH,T 变量。PLD 作为暗示，在没有高速缓存时当作 NOP 指令。
多寄存器加载-存储	32	STM,LDM,LDRD,STRD,LDC,STC
专用的加载-存储（exclusive load-store）	32	LDREX,STREX,LDREXB,LDREXH,STREXB,STREXH,CLREX。 如果没有局部监控程序时，将出现故障，这是 IMP DEF。 这里不包括 DREXD 和 STREXD。
分支	32	B,BL,B<cond>。状态一直改变，因此无 BLX(1)。无 BXJ。
系统	32	用 MSR(2)和 MRS(2)代替 MSR/MRS，但 MSR(2)和 MRS(2)还有其它更多的功能。可以用它们来访问其它堆栈和状态寄存器。 不支持 CPSIE/CPSID 的 32 位格式。 无 RFE 或 SRS。
系统	16	CPSIE 和 CPSID，是 MSR(2)指令的快速版本，使用标准的 Thumb-2 编码，但只允许使用“i”和“f”，不允许使用“a”。
扩展 32	32	NOP（所有格式），协处理器（LDC,MCR,MCR2,MCRR,MRC,MRRC,STC），以及 YIELD（相当于 NOP）。注：无 MRS(1)，MSR(1)，或 SUBS（PC 返回链接）。
组合分支	16	CBZ 和 CBNZ（如果寄存器为 0 或非 0，则进行比较并分支）。
扩展	16	IT 和 NOP。包括 YIELD。
除法	32	SDIV 和 UDIV。带符号和不带符号数的 32/32 除，商也为 32 位，没有余数。可通过减法操作来实现，该操作允许提前退出。
睡眠	16,32	WFI，WFE 和 SEV，相当于 NOP 指令，用来控制睡眠行为。
排序（barrier）	32	ISB，DSB 和 DMB，该类排序指令用来确保在下一条指令执行之前某些动作已经发生。
饱和（Saturation）	32	SSAT 和 USAT，用来对寄存器进行饱和操作。该类指令执行以下操作：使用移位操作将值规格化，测试来自所选位单元（Q 值）的溢出情况。如果溢出，则置位 xPSR 的 Q 位，在检测到溢出时使该值达到饱和。饱和值请参考针对所选大小的最大的无符号数或最大/最小的带符号数。

注：

所有的协处理器指令都产生无 CP 故障。

第3章 系统控制

本章列出了用于对 Cortex-M3 系统进行编程的寄存器，包括以下内容：

- 处理器寄存器汇总

3.1 处理器寄存器汇总

本节汇总了用来控制系统功能的寄存器，包括：

- 嵌套向量中断控制器的寄存器
- 内核调试寄存器
- 系统调试寄存器
- 调试接口的端口寄存器
- 跟踪端口接口单元的寄存器
- 嵌入式跟踪宏单元的寄存器

3.1.1 嵌套向量中断控制器的寄存器

表 3-1 列出了嵌套向量中断控制器（NVIC）的寄存器。NVIC 寄存器的详细描述见第 8 章*嵌入式向量中断控制器*。

表 3-1 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	a
SysTick 控制和状态寄存器	读/写	0xE000E010	0x00000000
SysTick 重装值寄存器	读/写	0xE000E014	不可预知
SysTick 当前值寄存器	读/写 清零	0xE000E018	不可预知
SysTick 校准值寄存器	只读	0xE000E01C	STCALIB
Irq0～31 使能置位寄存器	读/写	0xE000E100	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq224～239 使能置位寄存器	读/写	0xE000E011C	0x00000000
Irq0～31 使能清零寄存器	读/写	0xE000E0180	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq224～239 使能清零寄存器	读/写	0xE000E19C	0x00000000
Irq0～31 挂起置位寄存器	读/写	0xE000E200	0x00000000
.	.	.	.
.	.	.	.

续表 3-1

名称	类型	地址	复位值
.	.	.	.
Irq224～239 挂起置位寄存器	读/写	0xE000E21C	0x00000000
Irq0~31 挂起清零寄存器	读/写	0xE000E280	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq224～239 挂起清零寄存器	读/写	0xE000E29C	0x00000000
Irq0～31 激活位寄存器	只读	0xE000E300	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq224～239 激活位寄存器	只读	0xE000E31C	0x00000000
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	0xE000ED08	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 4-7 优先级寄存器	读/写	0xE000ED18	0x00000000
系统处理 8-11 优先级寄存器	读/写	0xE000ED1C	0x00000000
系统处理 12-15 优先级寄存器	读/写	0xE000ED20	0x00000000
系统处理控制和状态寄存器	读/写	0xE000ED24	0x00000000
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	不可预知
总线故障地址寄存器	读/写	0xE000ED38	不可预知
PFR0：处理器特性寄存器 0	只读	0xE000ED40	0x00000030
PFR1：处理器特性寄存器 1	只读	0xE000ED44	0x00000200
DFR0：调试特性寄存器 0	只读	0xE000ED48	0x00100000
AFR0：辅助特性寄存器 0	只读	0xE000ED4C	0x00000000
MMFR0：存储器模型特性寄存器 0	只读	0xE000ED50	0x00000030
MMFR1：存储器模型特性寄存器 1	只读	0xE000ED54	0x00000000
MMFR2：存储器模型特性寄存器 2	只读	0xE000ED58	0x00000000
MMFR3：存储器模型特性寄存器 3	只读	0xE000ED5C	0x00000000

续表 3-1

名称	类型	地址	复位值
ISAR0：ISA 特性寄存器 0	只读	0xE000ED60	0x01141110
ISAR1：ISA 特性寄存器 1	只读	0xE000ED64	0x02111000
ISAR2：ISA 特性寄存器 2	只读	0xE000ED68	0x21112231
ISAR3：ISA 特性寄存器 3	只读	0xE000ED6C	0x01111110
ISAR4：ISA 特性寄存器 4	只读	0xE000ED70	0x01310102
软件触发中断寄存器	只写	0xE000EF00	-
外设标识寄存器（PERIPHID4）	只读	0xE000EFD0	0x04
外设标识寄存器（PERIPHID5）	只读	0xE000EFD4	0x00
外设标识寄存器（PERIPHID6）	只读	0xE000EFD8	0x00
外设标识寄存器（PERIPHID7）	只读	0xE000EFDC	0x00
外设标识寄存器位 7:0（PERIPHID0）	只读	0xE000EFE0	0x00
外设标识寄存器位 15:8（PERIPHID1）	只读	0xE000EFE4	0xB0
外设标识寄存器位 23:16（PERIPHID2）	只读	0xE000EFE8	0x0B
外设标识寄存器位 31:24（PERIPHID3）	只读	0xE000EFEC	0x00
组件标识寄存器位 7:0（PCELLID0）	只读	0xE000EFF0	0x0D
组件标识寄存器位 15:8（PCELLID1）	只读	0xE000EFF4	0xE0
组件标识寄存器位 23:16（PCELLID2）	只读	0xE000EFF8	0x05
组件标识寄存器位 31:24（PCELLID3）	只读	0xE000EFFC	0xB1

a 复位值取决于定义的中断的数目。

3.1.2 内核调试寄存器

表 3-2 列出了内核调试寄存器。有关内核调试寄存器的详细描述见第 10 章*内核调试*。

表 3-2 内核调试寄存器

名称	类型	地址	复位值
调试中止控制和状态寄存器	读/写	0xE000EDF0	0x00000000[a]
调试内核寄存器的选择器寄存器	只写	0xE000EDF4	-
调试内核寄存器的数据寄存器	读/写	0xE000EDF8	-
调试异常和监控控制寄存器	读/写	0xE000EDFC	0x00000000[b]

a 位 5,3,2,1,0 由 **PORESETn** 复位，位[1]也可由 **SYSRESETn** 以及向应用中断和复位控制寄存器的 **VECTRESET** 位写入 1 来复位。

b 位 16,17,18,19 由 **STSRESETn** 以及向应用中断和复位控制寄存器的 **VECTRESET** 位写入 1 来复位。

3.1.3 系统调试寄存器

这节列出了用于系统调试的寄存器。

Flash 修补和断点寄存器

表 3-3 列出了 Flash 修补和断点（FPB）寄存器。有关 FPB 的详细描述请参考第 11 章*系统调试*。

表 3-3 Flash 修补和断点寄存器

名称	类型	地址	复位值	描述
FP_CTRL	读/写	0xE0002000	位[0]复位为 1’b0	Flash 修补控制寄存器
FP_REMAP	读/写	0xE0002004	-	Flash 修补重映射寄存器
FP_COMP0	读/写	0xE0002008	位[0]复位为 1’b0	Flash 修补比较器寄存器
FP_COMP1	读/写	0xE000200C	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP2	读/写	0xE0002010	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP3	读/写	0xE0002014	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP4	读/写	0xE0002018	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP5	读/写	0xE000201C	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP6	读/写	0xE0002020	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
FP_COMP7	读/写	0xE0002024	位[0]复位为 1’b0	Flash 修补（patch）比较器寄存器
PERIPID4	只读	0xE0002FD0	-	值为 0x04
PERIPID5	只读	0xE0002FD4	-	值为 0x00
PERIPID6	只读	0xE0002FD8	-	值为 0x00
PERIPID7	只读	0xE0002FDC	-	值为 0x00
PERIPID0	只读	0xE0002FE0	-	值为 0x30
PERIPID1	只读	0xE0002FE4	-	值为 0xB0
PERIPID0	只读	0xE0002FE8	-	值为 0x0B
PERIPID1	只读	0xE0002FEC	-	值为 0x00
PCELLID0	只读	0xE0002FF0	-	值为 0x0D
PCELLID1	只读	0xE0002FF4	-	值为 0xE0
PCELLID2	只读	0xE0002FF8	-	值为 0x05
PCELLID3	只读	0xE0002FFC	-	值为 0xB1

数据观察点和触发寄存器

表 3-4 列出了数据观察点和触发(DWT)寄存器。有关 DWT 寄存器的详细描述请参考第 11 章*系统调试*。

表 3-4 DWT 寄存器

名称	类型	地址	复位值	描述
DWT_CTRL	读/写	0xE0001000	0x00000000	DWT 控制寄存器
DWT_CYCCNT	读/写	0xE0001004	0x00000000	DWT 当前 PC 采样周期计数寄存器
DWT_CPICNT	读/写	0xE0001008	-	DWT 当前 CPI 计数寄存器
DWT_EXCCNT	读/写	0xE000100C	-	DWT 当前中断开销计数寄存器
DWT_SLEEPCNT	读/写	0xE0001010	-	DWT 当前睡眠计数寄存器
DWT_LSUCNT	读/写	0xE0001014	-	DWT 当前 LSU 计数寄存器
DWT_FOLDCNT	读/写	0xE0001018	-	DWT 当前折迭（folded）计数寄存器
DWT_COMP0	读/写	0xE0001020	-	DWT 比较器寄存器
DWT_MASK0	读/写	0xE0001024	-	DWT 屏蔽寄存器
DWT_FUNCTION0	读/写	0xE0001028	0x00000000	DWT 功能寄存器
DWT_COMP1	读/写	0xE0001030	-	DWT 比较器寄存器

续表 3-4

名称	类型	地址	复位值	描述
DWT_MASK1	读/写	0xE0001034	-	DWT 屏蔽寄存器
DWT_FUNCTION1	读/写	0xE0001038	0x00000000	DWT 功能寄存器
DWT_COMP2	读/写	0xE0001040	-	DWT 比较器寄存器
DWT_MASK2	读/写	0xE0001044	-	DWT 屏蔽寄存器
DWT_FUNCTION2	读/写	0xE0001048	0x00000000	DWT 功能寄存器
DWT_COMP3	读/写	0xE0001050	-	DWT 比较器寄存器
DWT_MASK3	读/写	0xE0001054	-	DWT 屏蔽寄存器
DWT_FUNCTION3	读/写	0xE0001058	0x00000000	DWT 功能寄存器
PERIPHID4	只读	0xE0001FD0	0x04	值为 0x04
PERIPHID5	只读	0xE0001FD4	0x00	值为 0x00
PERIPHID6	只读	0xE0001FD8	0x00	值为 0x00
PERIPHID7	只读	0xE0001FDC	0x00	值为 0x00
PERIPHID0	只读	0xE0001FE0	0x02	值为 0x02
PERIPHID1	只读	0xE0001FE4	0xB0	值为 0xB0
PERIPHID2	只读	0xE0001FE8	0x0B	值为 0x0B
PERIPHID3	只读	0xE0001FEC	0x00	值为 0x00
PCELLID0	只读	0xE0001FF0	0x0D	值为 0x0D
PCELLID1	只读	0xE0001FF4	0xE0	值为 0xE0
PCELLID2	只读	0xE0001FF8	0x05	值为 0x05
PCELLID3	只读	0xE0001FFC	0xB1	值为 0xB1

仪表跟踪宏单元寄存器

表 3-5 列出了仪表跟踪宏单元（ITM）的寄存器。有关 ITM 寄存器的详细描述请参考第 11 章*系统调试*。

表 3-5 ITM 寄存器

名称	类型	地址	复位值
激励端口 0-31	读/写	0xE0000000-0xE000007C	-
跟踪使能	读/写	0xE0000E00	0x00000000
跟踪特权	读/写	0xE0000E40	0x00000000
控制寄存器	读/写	0xE0000E80	0x00000000
综合写	只写	0xE0000EF8	0x00000000
综合读	只读	0xE0000EFC	0x00000000
综合模式控制	读/写	0xE0000F00	0x00000000
锁定访问寄存器	只写	0xE0000FB0	0x00000000
锁定状态寄存器	只读	0xE0000FB4	0x00000003
PERIPHID4	只读	0xE0001FD0	0x00000004
PERIPHID5	只读	0xE0001FD4	0x00000000
PERIPHID6	只读	0xE0001FD8	0x00000000
PERIPHID7	只读	0xE0001FDC	0x00000000

续表 3-5

名称	类型	地址	复位值
PERIPHID0	只读	0xE0001FE0	0x00000002
PERIPHID1	只读	0xE0001FE4	0x000000B0
PERIPHID2	只读	0xE0001FE8	0x0000000B
PERIPHID3	只读	0xE0001FEC	0x00000000
PCELLID0	只读	0xE0001FF0	0x0000000D
PCELLID1	只读	0xE0001FF4	0x000000E0
PCELLID2	只读	0xE0001FF8	0x00000005
PCELLID3	只读	0xE0001FFC	0x000000B1

AHB-AP 寄存器

表 3-6 列出了先进的高性能总线访问端口（AHB-AP）寄存器。有关 AHB-AP 寄存器的详细描述请参考第 11 章*系统调试*。

表 3-6 AHB-AP 寄存器

名称	类型	地址	复位值	描述
控制和状态字（CSW）	读/写	0x00	见寄存器	AHB-AP 控制和状态字寄存器
传输地址	读/写	0x04	0x00000000	AHB-AP 传输地址寄存器
数据读/写	读/写	0x0C	-	AHB-AP 数据读/写寄存器
分组数据 0（BD0）	读/写	0x10	-	AHB-AP 分组数据寄存器
分组数据 1（BD1）	读/写	0x14	-	AHB-AP 分组数据寄存器
分组数据 2（BD2）	读/写	0x18	-	AHB-AP 分组数据寄存器
分组数据 3（BD3）	读/写	0x1C	-	AHB-AP 分组数据寄存器
调试 ROM 地址	只读	0xF8	0xE000E000	AHB-AP 调试 ROM 地址寄存器
标识寄存器（IDR）	只读	0xFC	0x04770011	AHB-AP ID 寄存器

3.1.4 调试接口的端口寄存器

表 3-7 列出了调试接口的端口寄存器。有关调试接口的端口寄存器的详细描述请参考第 12 章*调试端口*。

表 3-7 调试端口寄存器

名称	描述	JTAG-DP	SW-DP	详细描述所在位置
ABORT	DAP 中止寄存器	是	是	中止寄存器
IDCODE	ID 代码寄存器	是	是	标识代码寄存器
CTRL/STAT	DP 控制/状态寄存器	是	是	控制/状态寄存器
SELECT	选择寄存器	是	是	AP 选择寄存器
RDBUFF	读缓冲区	是	是	读缓冲寄存器
WCR	线控制寄存器	否	是	线控制寄存器
RESEND	读再发寄存器	否	是	读再发寄存器

3.1.5 存储器保护单元的寄存器

表 3-8 列出了存储器保护单元（MPU）的寄存器。有关 MPU 寄存器的详细描述请参考第 9 章*存储器保护单元*。

表 3-8 MPU 寄存器

寄存器名称	类型	地址	复位值
MPU 类型寄存器	只读	0xE000ED90	0x00000000
MPU 控制寄存器	读/写	0xE000ED94	0x00000000
MPU 区域号寄存器	读/写	0xE000ED98	-
MPU 区域基址寄存器	读/写	0xE000ED9C	-
MPU 区域属性和规格寄存器	读/写	0xE000EDA0	-
MPU 别名 1 区域基址寄存器	D9C 的别名	0xE000EDA4	-
MPU 别名 1 区域属性和规格寄存器	DA0 的别名	0xE000EDA8	-
MPU 别名 2 区域基址寄存器	D9C 的别名	0xE000EDAC	-
MPU 别名 2 区域属性和规格寄存器	DA0 的别名	0xE000EDB0	-
MPU 别名 3 区域基址寄存器	D9C 的别名	0xE000EDB4	-
MPU 别名 3 区域属性和规格寄存器	DA0 的别名	0xE000EDB8	-

3.1.6 跟踪端口接口单元的寄存器

表 3-9 列出了跟踪端口接口单元（TPIU）的寄存器。有关 TPIU 寄存器的详细描述请参考第 13 章*跟踪端口接口单元*。

表 3-9 TPIU 寄存器

寄存器名称	类型	地址	复位值
支持的端口规格寄存器	只读	0xE0040000	0bxx0x
当前端口规格寄存器	读/写	0xE0040004	0x01
当前输出速度除数寄存器	读/写	0xE0040010	0x0000
所选的管脚协议寄存器	读/写	0xE00400F0	0x01
格式程序和刷新状态寄存器	只读	0xE0040300	0x08
格式程序和刷新控制寄存器	只读	0xE0040304	0x00 或 0x102
格式程序同步计数器寄存器	只读	0xE0040308	0x00
综合寄存器：ITATBCTR2	只读	0xE0040EF0	0x0
综合寄存器：ITATBCTR0	只读	0xE0040EF8	0x0

3.1.7 嵌入式跟踪宏单元的寄存器

表 3-10 列出了嵌入式跟踪宏单元（ETM）的寄存器。有关 ETM 寄存器的详细描述请参考第 15 章嵌入式跟踪宏单元。

表 3-10 ETM 寄存器

名称	类型	地址	是否存在
ETM 控制	读/写	0xE0041000	是
可配置代码	只读	0xE0041004	是
触发事件	只写	0xE0041008	是
ASIC 控制	只写	0xE004100C	否
ETM 状态	只读或读/写	0xE0041010	是
系统配置	只读	0xE0041014	是
跟踪使能	只写	0xE0041018,0xE004101C	否
跟踪使能事件	只写	0xE0041020	是
跟踪使能控制 1	只写	0xE0041024	是
FIFOFULL 区域	只写	0xE0041028	否
FIFOFULL 级别	只写或读/写	0xE004102C	是
ViewData	只写	0xE0041030-0xE004103C	否
地址比较器	只写	0xE0041040-0xE004113C	否
计数器	只写	0xE0041140-0xE004157C	否
定序器（sequencer）	读/写	0xE0041180-0xE0041194, 0xE0041198	否
外部输出	只写	0xE00411A0-0xE00411AC	否
CID 比较器	只写	0xE00411B0-0xE00411BC	否
特定实现	只写	0xE00411C0-0xE00411DC	否
同步频率	只写	0xE00411E0	否
ETM ID	只读	0xE00411E4	是
配置代码扩展	只读	0xE00411E8	是
扩展的外部输入选择器	只写	0xE00411EC	否
跟踪使能开始/停止嵌入式 ICE	读/写	0xE00411F0	是
嵌入式 ICE 行为控制	只写	0xE00411F4	否
CoreSight 跟踪 ID	读/写	0xE0041200	是
OS 保存/恢复	只写	0xE0041304-0xE0041308	否
ITMISCIN	只读	0xE0041EE0	是
ITTRIGOUT	只写	0xE0041EE8	是
ITATBCTR2	只读	0xE0041EF0	是
ITATBCTR0	只写	0xE0041EF8	是
综合模式控制	读/写	0xE0041F00	是
声明标签（claim tag）	读/写	0xE0041FA0-0xE0041FA4	是
锁定访问	只写	0xE0041FB0-0xE0041FB4	是
鉴别状态	只读	0xE0041FB8	是
设备类型	只读	0xE0041FCC	是

续表 3-10

名称	类型	地址	是否存在
外设 ID4	只读	0xE0041FD0	是
外设 ID5	只读	0xE0041FD4	是
外设 ID6	只读	0xE0041FD8	是
外设 ID7	只读	0xE0041FDC	是
外设 ID0	只读	0xE0041FE0	是
外设 ID1	只读	0xE0041FE4	是
外设 ID2	只读	0xE0041FE8	是
外设 ID3	只读	0xE0041FEC	是
组件 ID0	只读	0xE0041FF0	是
组件 ID1	只读	0xE0041FF4	是
组件 ID2	只读	0xE0041FF8	是
组件 ID3	只读	0xE0041FFC	是

第4章 存储器映射

本章描述了处理器固定的存储器映射以及 bit-banding 特性，包含以下内容：

- 关于存储器映射
- Bit-banding
- ROM 存储器表

4.1 关于存储器映射

图 4-1 显示了固定的 Cortex-M3 存储器映射。

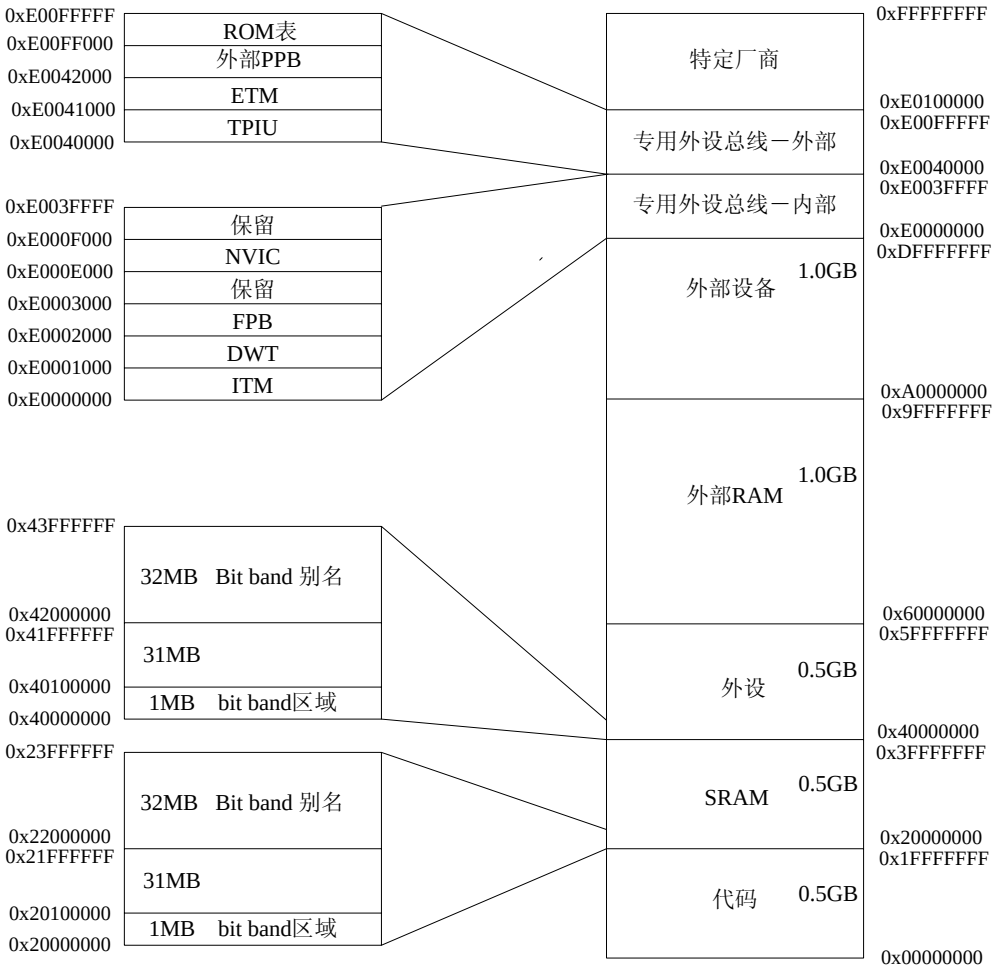


图 4-1 Cortex-M3 存储器映射

表 4-1 列出了被不同的存储器映射区域寻址的处理器接口。

表 4-1 存储器接口

存储器映射	接口
代码	指令取指在 ICode 总线上执行，数据访问在 DCode 总线上执行。
SRAM	指令取指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	指令取指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	指令取指和数据访问都在系统总线上执行。
外部设备	指令取指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），因此指令取指是禁止的。它也不能通过 MPU（如果有）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），因此指令取指是禁止的。它也不能通过 MPU（如果有）修改。

表 4-2 显示了处理器存储区的许可。

表 4-2 存储区的许可

名称	区域	设备类型	XN	高速缓存
代码	0x00000000-0x1FFFFFFF	常规	-	WT
SRAM	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
外部设备	0xC0000000-0xDFFFFFFF	设备	XN	-
专用外设总线	0xE0000000-0xE00FFFFF	SO	XN	-
系统	0xE0100000-0xFFFFFFFF	设备	XN	-

注：

位于 0xE0100000-0xFFFFFFFF 的专用外设总线和系统空间一直是 XN。它不能被存储器保护单元（MPU）覆盖。

有关处理器总线接口的详细描述请参考第 14 章*总线接口*。

4.2 Bit-banding

处理器存储器映射包括两个 bit-banding 区域。它们分别为 SRAM 和外设存储区域中的最低的 1MB。这些 bit-band 区域将存储器别名区的一个字映射为 bit-band 区的一个位。

Cortex-M3 存储器映射有 2 个 32MB 别名区，它们被映射为两个 1MB 的 bit-band 区。

- 对 32MB SRAM 别名区的访问映射为对 1MB SRAMbit-band 区的访问。
- 对 32MB 外设别名区的访问映射为对 1MB 外设 bit-band 区的访问。

映射公式显示如何将别名区中的字与 bit-band 区中的对应位或目标位关联。映射公式如下：

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

bit_word_addr=bit_band_base+bit_word_offset

这里：

- Bit_word_offset 为 bit-band 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 是别名区的开始地址。
- Bit_offset 为 bit-band 区中包含目标位的字节的编号。
- Bit_number 为目标位的位位置（0-7）。

图 4-2 显示了 SRAM bit-band 别名区和 SRAMbit-band 区之间的 bit-band 映射的例子：

地址 0x23FFFFE0 的别名字映射为 0x200FFFFC 的 bit-band 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

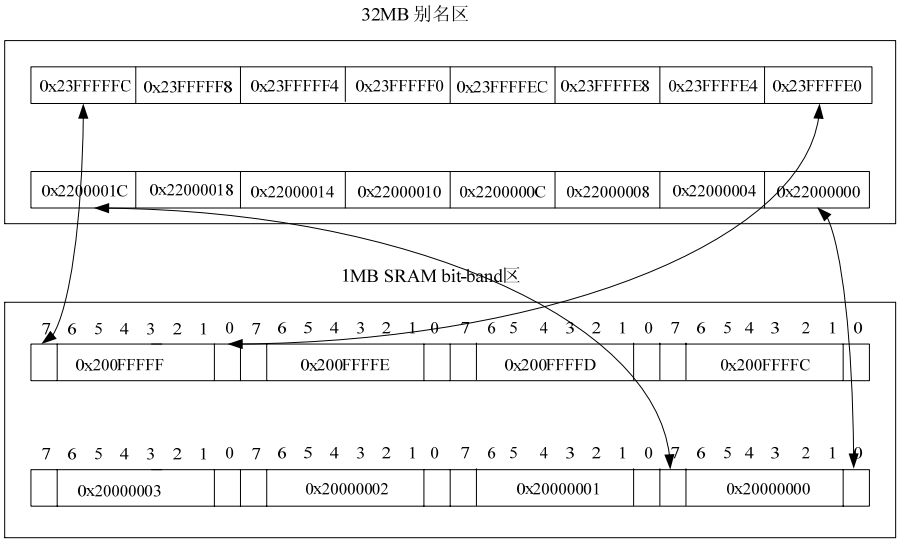


图 4-2 bit-band 映射

4.2.1 直接访问别名区

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。

写入别名区的字的位 0 决定了写入 bit-band 区的目标位的值。将位 0 为 1 的值写入别名区表示向 bit-band 位写入 1，将位 0 为 0 的值写入别名区表示向 bit-band 位写入 0。

别名字的位[31:1]在 bit-band 位上不起作用。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 区中的目标位置位。0x00 表示目标位清零。位[31:1]将为 0。

注：采用大端格式时，对 bit-band 别名区的访问必须以字节方式。否则访问值不可预知。

4.2.2 直接访问 bit-band 区

Bit-band 区能够使用常规的读和写以及写入该区操作进行访问。

4.3 ROM 存储器表

ROM 存储器的描述见表 4-3。

表 4-3 Cortex-M3 的 ROM 表

偏移	值	名称	描述
0x000	0xFFF0F003	NVIC	指向地址为 0xE000E000 的 NVIC
0x004	0xFFF02003	DWT	指向地址为 0xE001000 数据观察点和跟踪模块
0x008	0xFFF03003	FPB	指向地址为 0xE0002000Flash 修补和断点模块
0x00C	0xFFF01003	ITM	指向地址为 0xE0000000 的仪表跟踪模块
0x010	0xFFF41002 或 003	TPIU	指向 TPIU。如果有 TPIU，则值的位 0 设为 1
0x014	0xFFF41002 或 003	ETM	指向 ETM。如果有 ETM，则值的位 0 设为 1。ETM 地址为 0xE0041000
0x018	0	End	屏蔽 ROM 表的末端。如果添加了 CoreSight 组件，则它们从这个位置开始添加，末端屏蔽被移到附加组件之后的下一个位置。
0xFCC	0x1	MEMTYPE	如果为 1，MEMTYPE 区的位 0 定义为“系统存储器访问”，如果为 0，则只调试。
0xFD0	0x0	PID4	-
0xFD4	0x0	PID5	-
0xFD8	0x0	PID6	-
0xFDC	0x0	PID7	-
0xFE0	0x0	PID0	-
0xFE4	0x0	PID1	-
0xFE8	0x0	PID2	-
0xFEC	0x0	PID3	-
0xFF0	0x0D	CID0	-
0xFF4	0x10	CID1	-
0xFF8	0x05	CID2	-
0xFFC	0xB1	CID3	-

第5章 异常

本章描述了处理器的异常模型，包含以下内容：

- 关于异常模型
- 异常类型
- 异常优先级
- 占先(pre-emption)
- 末尾连锁（Tail-chaining）
- 迟来(late-arriving)
- 退出
- 复位
- 异常的控制权转移
- 设置多个堆栈
- 中止（abort）模型
- 激活级别（activation level）
- 流程图

5.1 关于异常模型

Cortex-M3 处理器和嵌套向量中断控制器（NVIC）对所有异常按优先级进行排序并处理。所有异常都在处理模式中操作。出现异常时，自动将处理器状态保存到堆栈中，并在中断服务程序（ISR）结束时自动从堆栈中恢复。在状态保存的同时取出向量快速地进入中断。处理器支持末尾连锁（tail-chaining）中断技术，它能够在没有多余的状态保存和恢复指令的情况下执行背对背中断（back-to-back interrupt）。以下特性可使能高效的低延迟异常处理：

- 自动的状态保存和恢复。处理器在进入 ISR 之前将状态寄存器压栈，退出 ISR 之后将它们出栈，实现上述操作时不需要多余的指令。
- 自动读取代码存储器或 SRAM 中包含 ISR 地址的向量表入口。该操作与状态保存同时执行。
- 支持末尾连锁（tail-chaining），在末尾连锁（tail-chaining）中，处理器在两个 ISR 之间没有对寄存器进行出栈和压栈操作的情况下处理背对背中断。
- 中断优先级可动态重新设置。
- Cortex-M3 与 NVIC 之间采用紧耦合接口，通过该接口可以及早地对中断和高优先级的迟来中断进行处理。
- 中断数目可配置为 1～240。
- 中断优先级的数目可配置为 1～8 位（1～256 级）。
- 处理模式和线程模式具有独立的堆栈和特权等级。
- 使用 C/C++标准的调用规范：*ARM 架构的过程调用标准（PCSAA）*执行 ISR 控制传输。

- 优先级屏蔽支持临界区

注：

中断的数目和中断优先级的位数在实现时配置。软件可以选择只使能已配置中断数目的子集，以及选择所配置的优先级使用多少个位。

5.2 异常类型

Cortex-M3 处理器中存在多种异常类型。异常是指由于执行指令时的一个错误条件而产生的故障。出现故障后可以同步或不同步地向引起故障的指令报告，但通常还是会同步报告。不精确的总线故障是 ARMv7-M 性能分析（performance profiling）支持的一种不同步故障。同步故障总是和引起该故障的指令一同被报告。不同步故障不能保证与引起该故障的指令相关的方式报告。

有关异常的详细信息请参考 *ARMv7-M 架构参考手册*。

表 5-1 显示了异常类型，优先级以及位置。位置是指与向量表开始处的字偏移。在优先级列中，数字越小表示优先级越高。表中还显示了异常类型的激活方式，即是同步的还是异步的。优先级的准确含义和使用见*异常优先级*。

表 5-1 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。在第一条指令上优先级下降到最低（线程模式）。异步的。
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占先。异步的
硬故障	3	-1	由于优先级的原因或可配置的故障处理被禁止而导致不能将故障激活时的所有类型故障。同步的
存储器管理	4	可调整 [a]	MPU 不匹配，包括违反访问规范以及不匹配。是同步的。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 [b]	预取故障，存储器访问故障，以及其它相关的地址/存储故障。精确时是同步，不精确时是异步。
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。是同步的。
-	7-10	-	保留
SVCall	11	可调整	利用 SVC 指令调用系统服务。是同步的。
调试监控	12	可调整	调试监控，在处理器没有停止时出现。是同步的，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）已启动。是异步的。
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，都是异步的。

a　该异常的优先级可修改，见*系统处理器优先级寄存器的位分配*。可调整的范围为 NVIC 的 0～N 优先

级，N 为能够实现的最高优先级。在内部，用户可设置的最高优先级（0）被看作 4。

b　　您可以使能或禁止该故障。见系统处理器控制和状态寄存器的位分配。

5.3 异常优先级

表 5-2 对优先级如何影响处理器处理异常的时间和方式进行了描述，并列出了异常基于优先级而采取的动作。

表 5-2 异常基于优先级的动作

动作	描述
占先	新的异常比当前的异常或线程的优先级更高并打断当前的流程，这是对挂起中断（pended interrupt）的响应。如果挂起中断的优先级比当前的 ISR 或线程的优先级更高，则进入挂起中断的 ISR。如果一个 ISR 抢占了另一个 ISR，则产生了中断嵌套。 在进入异常时，处理器自动保存其状态，将状态压栈。与此同时，取出相应的中断向量。当处理器状态被保存并且 ISR 的第一条指令进入处理器流水线的执行阶段时，开始执行 ISR 的第一条指令。状态保存在系统总线上执行。取向量操作根据向量表所在位置可以在系统总线或 DCode 总线上执行。见*向量表偏移寄存器*。
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。

在处理器的异常模型中，优先级决定了处理器何时以及怎样处理异常，您能够：

- 指定中断的软件优先级
- 将优先级分组，分为占先优先级（pre-emption priorities）和次优先级(subpriorities)。

5.3.1 优先级

NVIC 支持由软件指定的优先级。通过对中断优先级寄存器的 8 位 PRI_N 区执行写操作，来将中断的优先级指定为 0～255，见*中断优先级寄存器*。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。指定软件优先级后，硬件优先级无效。例如，如果您将 **INTISR[0]**指定为优先级 1，**INTISR[31]**指定为优先级 0，则 **INTISR[31]**的优先级比 **INTISR[0]**高。

注：

软件优先级的设置对复位，NMI，和硬故障无效。它们的优先级始终比外部中断要高。

如果两个或更多的中断指定了相同的优先级，则由它们的硬件优先级来决定处理器对它们进行处理时的顺序。例如，如果 **INTISR[0]和 INTISR[1]**优先级都为 1，则 **INTISR[0]**的优先级比 **INTISR[1]**要高。

PRI_N 区的详细信息请参考*中断优先级寄存器*。

5.3.2 优先级分组

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。您可以使用*应用中断和复位控制寄存器*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。我们将占先优先级称为组优先级。如果有多个挂起异常共用相同的组优先级，则需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。这与优先级机制是一致的。

表 5-3 显示了如何对 PRIGROUP 执行写操作，来将 8 位 PRI_N 分为占先优先级区(x)和次优先级区(y)。

表 5-3 优先级分组

PRIGROUP[2:0]	中断优先级区，PRI_N[7:0]				
	二进制点的位置	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
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

注：

- 表 5-3 显示了利用优先级的 8 个位来配置的处理器优先级。
- 如果使用小于 8 的位来配置处理器的优先级，则寄存器的低位始终为 0。例如，如果使用 4 个位来配置优先级，则 PRI_N[7:4]用来配置优先级，而 PRI_N[3:0]为 4`b0000。

如果一个中断想抢占另一个正在处理的中断，则它的占先优先级必须比正在处理的中断的占先优先级要高。

更多关于优先级优化，优先级分组，和优先级屏蔽的信息，请参考 *ARMv7-M 架构参考*

手册。

5.4 特权和堆栈

Cortex-M3 处理器支持两个独立的堆栈：

进程堆栈（process stack）

线程模式可以配置为使用进程堆栈。线程模式在复位后使用主堆栈，SP_process 为进程堆栈的 SP 寄存器。

主堆栈

处理模式使用主堆栈。SP_main 为主堆栈的 SP 寄存器。

在任何时候进程堆栈或主堆栈中只有一个是可见的。在将 8 个寄存器压栈之后，ISR 使用主堆栈，并且后面所有的抢占中断都使用主堆栈。状态保存时的堆栈使用规则如下：

- 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 来访问，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它们自身的局部变量。

当线程模式使用进程堆栈，异常使用主堆栈时支持操作系统(OS)的调度。在重新调度时，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。

注：

MSR/MRS 指令对两个堆栈都具有可见性。

5.4.1 堆栈

堆栈不受到特权模式的约束。即，线程模式可以使用进程堆栈或主堆栈，可以在用户模式或特权模式下。堆栈和特权的 4 种组合都是可能的。对于一个基本的受保护的线程模型，用户线程在使用进程堆栈的线程模式中运行，内核和中断在使用主堆栈的特权模式中运行。

注：

不管是恶意的还是无意的，特权都无法避免对堆栈的破坏。因此，需有一两种形式的存储器保护单元来隔离用户代码，即，必须避免将用户代码写入不属于它的含有其它堆栈的存储器中。

5.4.2 特权

特权用来控制访问的权利，它与 ARMv7-M 中的其它概念是分离的。代码可以是具有完全访问权利的特权访问，或访问权利受到限制的非特权/用户访问。访问权利对以下功能产生影响：

- 使用或不使用某些指令，例如 MSR
- 访问系统控制空间（SCS）的寄存器
- 使用某些协处理器或协处理器寄存器
- 根据系统的设计访问存储器或外设。处理器告诉系统对代码的访问是否是特权访

问，这样，系统能够在非特权访问上加上限制。

- 基于 MPU 的存储单元的访问规则。当配置了 MPU 时，访问限制能够控制哪些存储单元能够读，写以及执行。

只有线程模式可以是非特权访问，所有异常都是特权访问。

5.5 占先

这节描述出现异常时处理器的行为：

- 堆栈
- 迟来
- 末尾连锁（Tail-chain）

5.5.1 堆栈

当处理器调用异常时，它自动将下面的 8 个寄存器按以下顺序压栈：

- PC
- xPSR
- r0~r3
- r12
- LR

在完成压栈之后，SP 减小 8 个字。图 5-1 显示了异常抢占当前的程序流程之后堆栈中的内容。

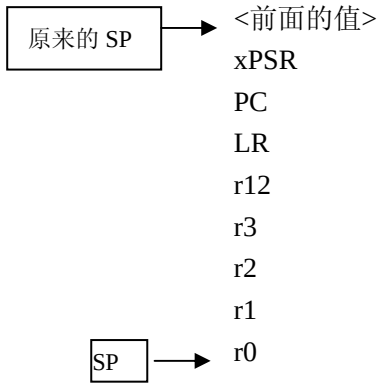


图 5-1 抢占之后堆栈中的内容

注：

图 5-1 显示了压栈时的顺序。

从 ISR 返回之后，处理器自动将 8 个寄存器出栈。根据 LR 中的数据执行中断返回，ISR 函数可以是常规的 C/C++函数，不需要胶合（veneer）。

表 5-4 描述了在进入 ISR 之前 Cortex-M3 处理器采取的步骤。

表 5-4 异常进入步骤

动作	是否可重启	描述
8 个寄存器压栈 [a]	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的新位置重新加载流水线。	从向量表指向的位置加载指令。它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。

a　在使用末尾连锁（tail-chaining）时，该步省略。

图 5-2 显示了异常进入的时序。

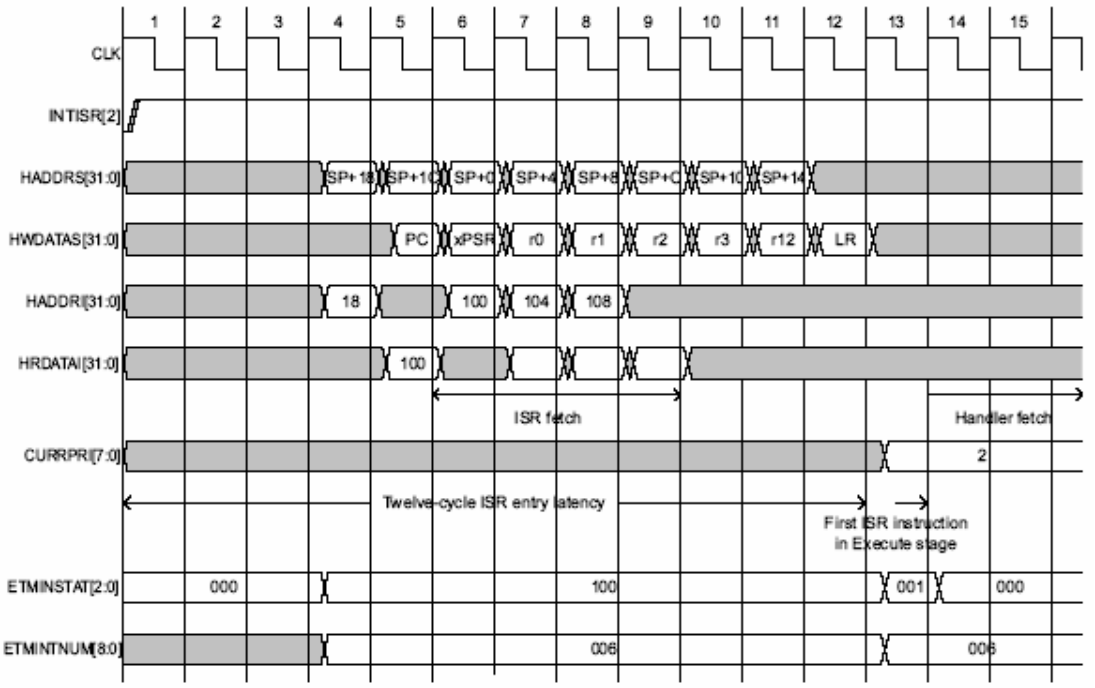


图 5-2 异常进入时序

在接收到 INTISR[2]之后的周期中，NVIC 向处理器内核指示已经接收到中断，处理器在下一个周期开始压栈和取向量操作。

在完成压栈之后，ISR 的第一条指令进入流水线的执行阶段。在 ISR 进入执行阶段的周期中：

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。当 **ETMINTSTAT**(3`b001**)**表示已经进入 ISR 时，**CURRPRI** 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。

当 **ETMINTSTAT(3`b001)**表示已经进入 ISR 时，**ETMINTNUM** 有效。在这之前，它表示正在取出的是哪一个 ISR。

图 5-2 显示从中断有效到 ISR 执行第一条指令，这中间有 12 个周期的延迟。

5.6 末尾连锁（Tail-chaining）

末尾连锁（Tail-chaining）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。

如果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（Tail-chaining）。

图 5-3 显示了一个末尾连锁（Tail-chaining）的实例。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 之后六个周期，开始执行被末尾连锁（tail-chained）的 ISR。

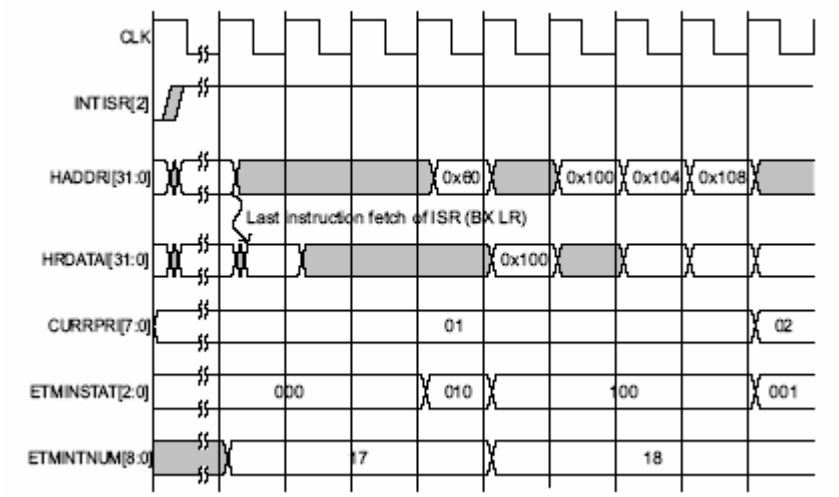


图 5-3 末尾连锁（Tail-chaining）时序

在从上一个 ISR 返回时，**INTISR[2]**的优先级比所有被压栈的 ISR，或其它被挂起的中断都高，因此处理器末尾连锁（tailchain）到与 **INTISR[2]**对应的 ISR。在 **INTISR[2]**进入执行阶段的周期中：

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。

图 5-3 显示从上一个 ISR 返回到执行新的 ISR，这中间有 6 个周期的延迟。

5.7 迟来

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要高，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 5-4 显示了一个迟来中断的实例。

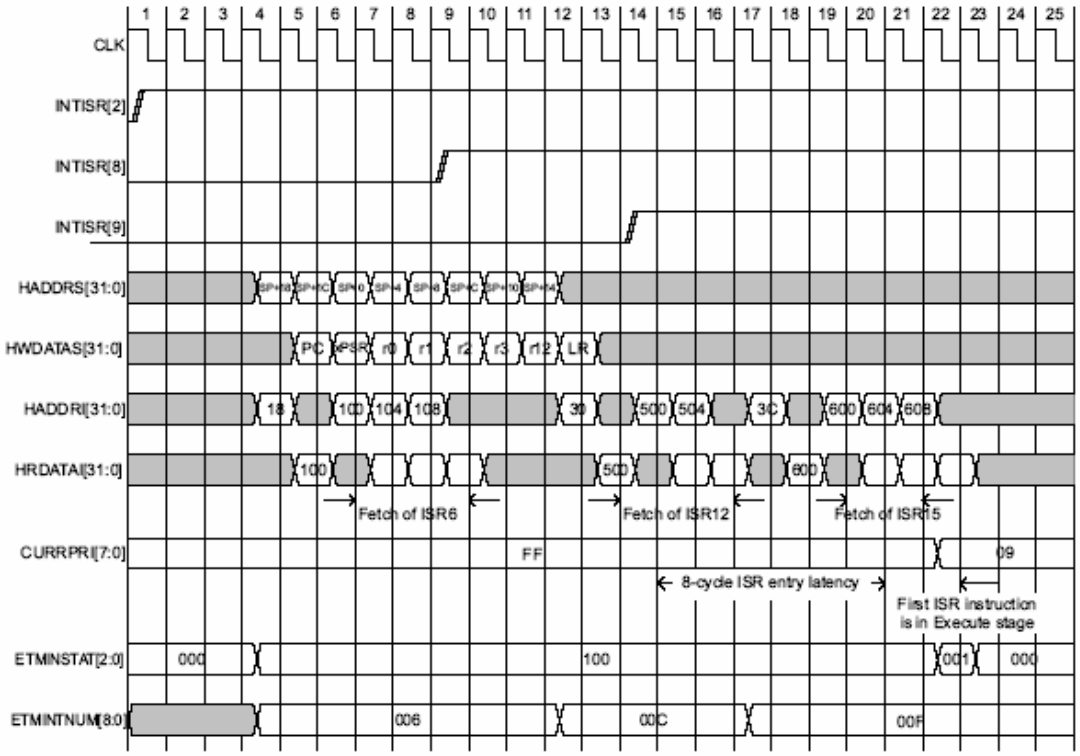


图 5-4 迟来异常时序

在图 5-4 中，**INTISR[8]**抢占了 **INTISR[2]**。**INTISR[2]**的状态保存已经完成，不需要重复。图 5-4 显示了 **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的最迟的点。这个点之后的更高优先级的中断将会被当作抢占来处理。

图 5-4 显示了 **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的最迟的点。**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 进入执行阶段的周期中：

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。

5.8 退出

ISR 的最后一条指令是将进入异常时的 LR（0xFFFFFFFX）加载到 PC 中。该操作向处理器指示 ISR 已完成，处理器启动异常退出序列。处理器从 ISR 返回时使用的指令请参考*处理器从 ISR 返回*。

5.8.1 异常退出

在从异常返回时，处理器将执行下列操作之一：

- 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 5-5 描述了后同步序列。

表 5-5 异常退出步骤

动作	描述
8 个寄存器出栈	如果没有被抢占，则将 PC,xPSR,r0,r1,r2,r3,r12,LR 从所选的堆栈中出栈（堆栈由 EXC_RETURN 选择），并调整 SP。
加载当前激活的中断号	加载来自被压栈的 IPSR 的位[8:0]中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]为 0 时，处理器返回线程模式。
选择 SP	如果返回到异常，SP 为 SP_main，如果返回到线程模式，则 SP 为 SP_main 或 SP_process。

a 由于优先级可动态改变，NVIC 使用中断号代替中断优先级来决定当前的 ISR 是哪一个。

图 5-5 显示了异常退出时序。

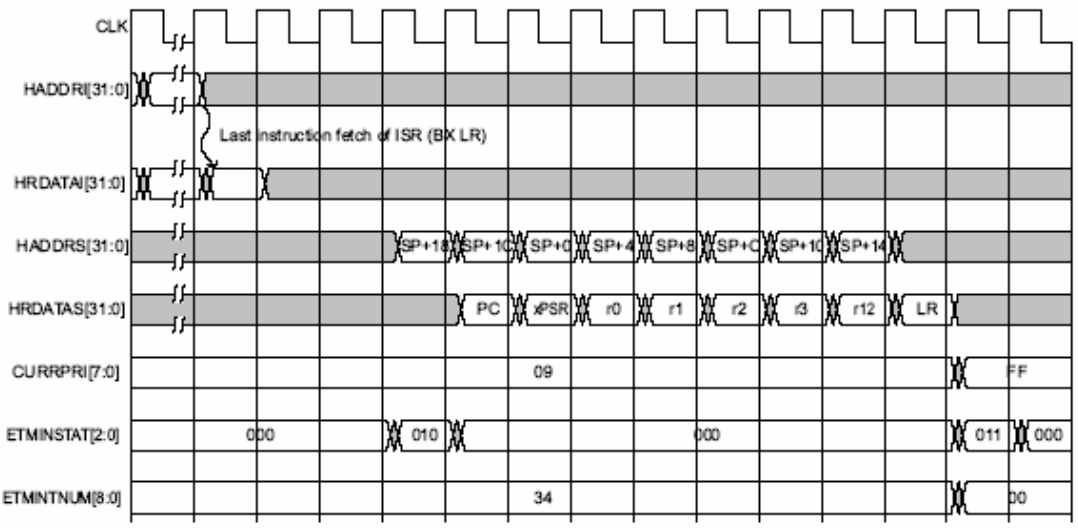


图 5-5 异常退出时序

对于 **ETMINSTAT**：

- **ETMINSTAT** 为 3`b010 表示已经退出 ISR。**ETMINTNUM** 显示退出的 ISR 的中断号。
- 如果正在返回前一个被压栈的 ISR，则中断退出之后的周期中 **ETMINSTAT** 为 3`b011。**ETMINTNUM** 显示正在返回的中断的中断号。

注：

如果在出栈过程中出现一个优先级更高的中断，则处理器放弃出栈操作，堆栈指针退回去，并将该异常看作 tail-chain 的情况来响应。

5.8.2 处理器从 ISR 中返回

当使用下面的其中一条指令将 0xFFFFFFFX 的值装入 PC 时，执行异常返回：

- POP/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

当以上述方式返回时，写入 PC 的值被截取，作为 **EXC_RETURN** 的值。**EXC_RETURN[3:0]**用来提供返回信息，见表 5-6 中的定义。

表 5-6 异常返回的行为

EXC_RETURN[3:0]	
0bxxx0	保留
0b0001	返回处理模式 异常返回，获得来自主堆栈的状态。 在返回时指令执行使用主堆栈
0b0011	保留
0b01x1	保留
0b1001	返回线程模式 异常返回，获得来自主堆栈的状态 返回时指令执行使用主堆栈
0b1101	返回线程模式 异常返回，获得来自进程堆栈的状态 返回时指令执行使用进程堆栈
0b1x11	保留

如果 EXC_RETURN[3:0]的值为该表中的保留值，则将导致一个称作使用故障的连锁（chained）异常。

如果在线程模式中，或从向量表，或通过任何其它的指令将 EXC_RETURN 的值加载到 PC 时，该值看作一个地址，而不是特殊的值。这个地址范围被定义为具有永不执行（XN）许可，并将导致存储器管理故障。

5.9 复位

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 5-7 显示了复位的行为。

表 5-7 复位动作

动作	描述
NVIC 复位，内核保持在复位状态	NVIC 对它的大部分寄存器进行清零。处理器位于线程模式，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，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 和硬故障未被禁止。

有关复位的详细信息请参考第 6 章*时钟和复位*。

5.9.1 向量表和复位

位于地址 0 的向量表只需要有 4 个值：

- 栈顶地址
- 复位程序的位置
- NMI ISR 的位置
- 硬故障 ISR 的位置

当中断使能时，不管向量表的位置在哪，它都指向所有使能屏蔽的异常。并且如果使用 SVC 指令，还需要指定 SVCall ISR 的位置。

一个完整的向量表：

```
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
    };
```

5.9.2 预期的启动顺序（boot up sequence）

一个正常的复位程序遵循表 5-8 中的步骤。C/C++运行时间将执行前三步，然后调用 main()。

表 5-8 复位启动时的动作

动作	描述
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。
初始化所有的运行时间	可选择调用 C/C++运行时间的注册码，以允许使用堆(heap)，浮点运算或其它功能。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 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。
[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	设置优先级和屏蔽
使能中断	使能中断。使能 NVIC 中的中断处理。如果不希望在中断刚使能时产生中断，如果中断多于 32 个，则需要多个设置使能寄存器。通过 CPS 或 MSR，能够使用 **PRIMASK** 在准备就绪之前屏蔽中断。
[改变优先级]	[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。如果 sleep-on-exit 可选择使能/禁止，则 loop 能够处理清除操作和执行的任务。如果不使用 sleep-on-exit，则 loop 能够做想做的事并且能够使用 WFI（现在睡眠）。

复位程序实例

复位程序用来启动应用程序，以及使能中断。有 3 种方法可以在执行完中断处理之后调用复位 ISR。这称作复位 ISR 的“主循环”部分，这 3 种方法如下所示：

例 5-1 退出时进入完全睡眠的复位程序（复位程序不会执行主循环）

```
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();
}
```

例 5-2 使用 WFI 使睡眠模式可选的复位程序

```
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
}
}
```

例 5-3 所选的 Sleep-on-exit 功能被 ISR 取消的复位程序，我们要留意这个 ISR

```
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
}
}
```

注：

因为可以通过激活 ISR 来改变优先级，所以执行代码不必放在复位程序中。这确保了对加载量的改变作出快速响应，以及使用优先级提升（priority boosting）机制，来解决优先级倒置(priority inversion)和确保精细的微量支持。对于使用线程和特权访问的 RTOS 模型，用户代码使用线程模式。

5.10 异常的控制权转移

处理器按照表 5-9 中的规则将控制权转给 ISR。

表 5-9 转向异常处理

异常有效时的处理器动作	转向异常处理
无存储器指令	在下一条指令之前，这个周期结束时获取异常
单寄存器加载/存储	完成或放弃，由总线状态决定。根据总线等待状态，下一个周期获取异常。
多寄存器加载/存储	完成或放弃当前的寄存器并在 EPSR 中设置下一个寄存器。根据总线许可以及可中断-可继续指令（ICI）规则，在下一个周期获取异常。ICI 规则的详细信息请参考 *ARMv7-M 架构参考手册*。
异常进入	这是一个迟来异常，如果它的优先级比正在进入的异常的优先级要高，则处理器取消异常的进入行为并获取迟来异常。迟来导致了中断处理时间上的决策改变（向量表）。在进入一个新的处理程序时，它是第一条 ISR 指令，应用通常的占先规则，不再看作是迟来。
Tail-chaining	如果迟来异常的优先级比正在被末尾连锁（tail-chained）的异常的优先级要高，则处理器取消前导码（preamble）并捕获迟来异常。
异常后同步（postamble）	如果新的异常的优先级比处理器即将返回的被压栈异常的优先级要高，则处理器末尾连锁（tail-chain）到新的异常。

5.11 设置多个堆栈

为实现多个堆栈，应用程序必须完成以下操作：

- 使用 MSR 指令来设置 process_SP 寄存器
- 如果使用 MPU，则适当地保护堆栈
- 将中断与堆栈关联
- 初始化线程模式的堆栈和特权模式

如果线程模式从特权访问变为用户访问，则只有另一个 ISR，例如 SVCall 能够将其从用户访问返回特权访问。

线程模式中的堆栈能够从主堆栈变为进程堆栈或从进程堆栈变为主堆栈，但这样做会影

响对线程模式自身的局部变量的访问。最好让另一个 ISR 来改变线程模式使用的堆栈。下面是一个引导序列（boot sequence）的实例：

1. 调用 setup 程序，完成下列操作：
    a. 使用 MSR 设置其它堆栈
    b. 如果有 MPU 则使能，以支持基址区域（base region）
    c. 调用所有的引导程序（boot routine）
    d. 从 setup 程序返回
2. 将线程模式变为非特权访问
3. 使用 SVC 来调用内核（kernel），然后内核开始执行以下操作：
    a. 启动线程
    b. 使用 MRS 来读当前用户线程的 SP，并将它保存在自身的 TCB 中。
    c. 使用 MSR 来设置下一个线程的 SP，这是通常的 SP_process
    d. 如果有必要，设置当前最新的线程的 MPU
    e. 返回当前最新的线程。

例 5-4 为在 ISR 中修改 EXC_RETURN 的值来实现使用 PSP 返回的例子。

例 5-4 在 ISR 中修改 EXC_RETURN

```
;第一次使用 PSP，从 RETTOBASE == 1 的处理模式中运行
LDR r0, PSPValue ; 获得新的进程堆栈的值
MSR PSP, r0 ; 设置进程堆栈的值
ORR lr, lr, #4 改变 EXC_RETURN 的值，实现在 PSP 上返回
BX lr ; 从处理模式返回线程模式
```

例 5-5 显示了在切换到 PSP 上的线程之后，如何实现一个简单的上下文切换。

例 5-5 实现一个简单的上下文切换

```
; 上下文切换举例 (假设线程已经在 PSP 上)
MRS r12, PSP ; 将 PSP 复制到 R12 中
STMDB r12!, {r4-r11} ;将非堆栈的寄存器压栈
LDR r0, =OldPSPValue ; 指针指向原来的线程控制模块
STR r12, [r0] ;SP 保存在线程控制模块中
LDR r0, =NewPSPValue ; 指针指向新的线程控制模块
LDR r12, [r0] ; 获得新的进程的 SP
LDMIA r12!, {r4-r11} ; 恢复非堆栈的寄存器
MSR PSP, r12 ; 将 R12 写入 PSP
BX lr ; 返回线程
```

注：

在例 5-4 和 5-5 中，仅当只有一个 ISR/处理程序（Handler）活动时，才能将线程从 MSP

移到 PSP，或者才能保护非堆栈的寄存器不会被压栈的处理程序修改。

5.12 中止(abort)模型

能够产生中止故障的 4 个事件包括：

- 指令取指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，未定义的指令或试图用 BX 指令来改变状态。NVIC 中的故障状态寄存器指示了是否引起故障。
- 由于违反了特权模式或未管理的区域而引起的 MPU 故障

故障处理程序有两种：

- 固定优先级的硬故障
- 优先级可调整的局部故障

5.12.1 硬故障

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 或其它硬故障之外的所有异常。

注：

将使用 **FAULTMASK** 的代码看作硬故障，并遵循与硬故障相同的规则。

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。

5.12.2 局部故障和升级

局部故障根据引起的原因来分类，见 5-10。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引起了相同类型的故障。
- 局部故障处理程序引起了具有相同或更高优先级的故障。
- 异常处理程序引起了具有相同或更高优先级的故障。
- 局部故障未使能。

表 5-10 列出了局部故障。

表 5-10 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	任何形式的复位	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	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－违反 MPU 访问规则	INTERR
uCode 出栈错误	UNSTKERR	总线故障	使用硬件恢复上下文时失败－返回的总线错误	INTERR
uCode 出栈错误	MUNSKERR	存储器管理	使用硬件恢复上下文时失败－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
非精确数据总线错误	IMPRECISERR	用法错误	由于数据访问而产生的延迟的总线错误。精确的指令不可知。这是被挂起的，不是同步的。它不引起 FORCED。	BUSERR
无协处理器	NOCP	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。例如，不是 thumb	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

续表 5-10

错误	位名称	处理程序	备注	陷进（trap）使能位
在没有使能或带有非法的魔数（magic number）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	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。 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）	-

表 5-11 显示了调试故障。

表 5-11 调试故障

故障	标志	备注	陷进（Trap）使能位
内部中止请求	HALTED	来自单步，内核中止等的 NVIC 请求	-
断点	BKPT	来自修补指令或 FPB 的 SW 断点	-
观察点	DWTTRAP	DWT 中的观察点匹配	-
除以 0	DIVBYZERO	当使能以用于 trap 时，除以 0	-
不对齐访问	UNALIGNED	当使能以用于 trap 时，非对齐访问	-
外部	EXTERNAL	EDBGRQ 线有效	-

5.12.3 故障状态寄存器和故障地址寄存器

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：

- 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。

根据故障的原因，5 个状态寄存器的其中一个寄存器中有一个位置位。

有 2 个故障地址寄存器（FAR）：

- 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。

注：BFAR 和 MMFAR 是同一个物理寄存器，因此，BFARVALID 和 MMFARVALID 位是相互排斥的。

表 5-12 显示了故障状态寄存器和两个故障地址寄存器。

表 5-12 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	描述
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
UFSR	用法错误	-	用法错误
DFSR	调试器监控或停止	-	调试陷阱

5.13 激活等级(activation level)

在无激活的异常时，处理器处于线程模式。当 ISR 或故障处理程序有效时处理器进入处理模式。表 5-13 列出了激活等级的特权模式和堆栈。

表 5-13 不同激活等级的特权模式和堆栈

有效的异常	激活等级	特权模式	堆栈
无	线程模式	特权访问或用户访问	主堆栈或进程堆栈
ISR 有效	异步占先级	特权访问	主堆栈
故障处理程序有效	同步占先级	特权访问	主堆栈
复位	线程模式	特权访问	主堆栈

表 5-14 总结了所有异常类型的转换规则以及它们与访问规则和堆栈的关联。

表 5-14 异常转换

有效的异常	触发事件	转换类型	特权模式	堆栈
复位	复位信号	线程	特权访问或 用户访问	主堆栈或 进程堆栈
ISR[a] 或 NMI[b]	设置挂起的软件指令或硬件信号	异步占先	特权访问	主堆栈
故障： 硬故障 总线故障 无 CP[c] 故障 未定义指令故障	 升级 存储器访问错误 访问不存在的 CP 为定义的指令	同步占先	特权访问	主堆栈
调试监控	停止未使能时的调试事件	同步	特权访问	主堆栈
SVC[d]	SVC 指令			
外部中断				

a　中断服务程序

b　不可屏蔽的中断

c　协处理器

d　软中断

表 5-15 显示了异常子类型的转换。

表 5-15 异常子类型转换

预期的激活子类型	触发事件	激活	优先级的影响
线程	复位信号	异步	立即，线程是最低的
ISR/NMI	HW 信号或设置挂起	异步	占先或 tail-chain，根据优先级
监控	调试事件 [a]	同步	如果优先级小于或等于当前异常，硬故障
SVCall	SVC 指令	同步	如果优先级小于或等于当前异常，硬故障
PendSV	软件挂起请求	连锁（chain）	占先或 tail-chain，根据优先级
使用错误	未定义指令	同步	如果优先级大于或等于当前异常，硬故障
无 CP 故障	访问不存在的 CP	同步	如果优先级大于或等于当前异常，硬故障
总线故障	存储器访问错误	同步	如果优先级大于或等于当前异常，硬故障
存储器管理	MPU 不匹配	同步	如果优先级大于或等于当前异常，硬故障
硬故障	升级	同步	高于所以的总线 NMI
故障升级	来自可配置故障处理程序的升级请求	连锁（chain）	局部处理程序的优先级提升为与硬故障相同，这样它能够返回并连锁（chain）到可配置的故障处理程序。

a　在停止并且没有使能时

5.14 流程图

这节总结了以下中断流程：

- 中断处理
- 占先
- 返回

5.14.1 中断处理

图 5-6 显示了指令的执行过程，直到被一个优先级更高的中断抢占。

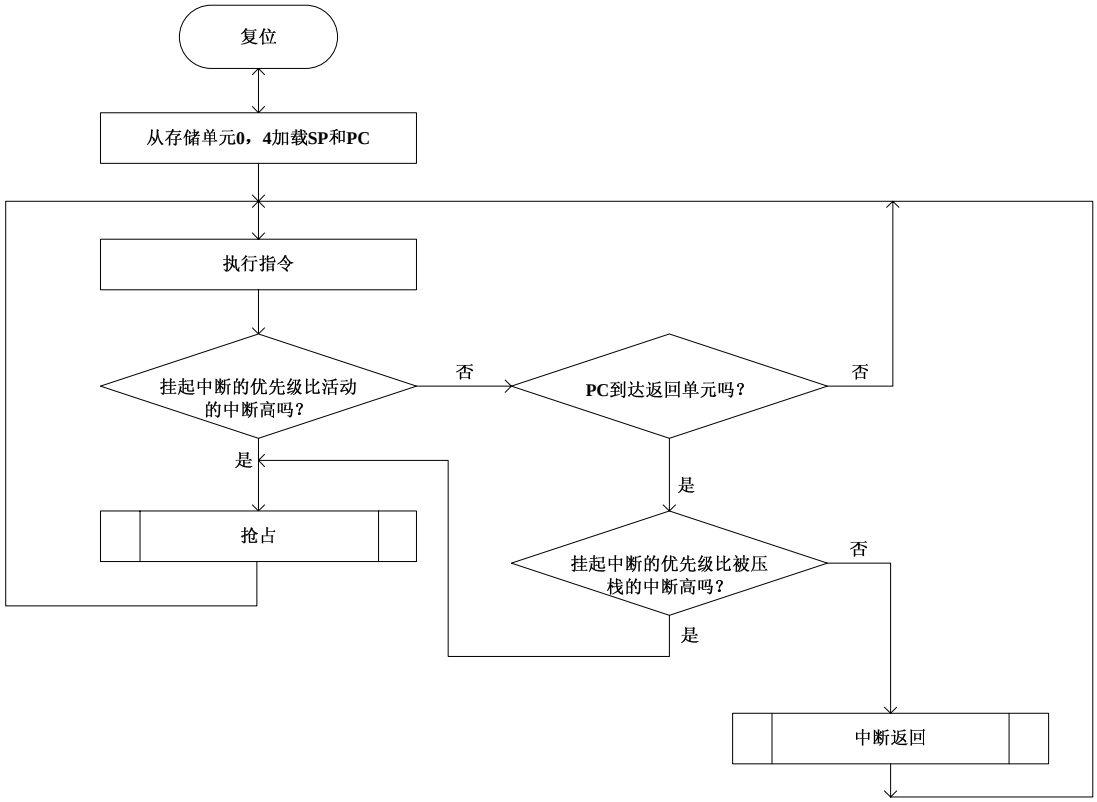


图 5-6 中断处理流程图

5.14.2 占先

图 5-7 显示了当异常抢占了当前的 ISR 时执行的操作。

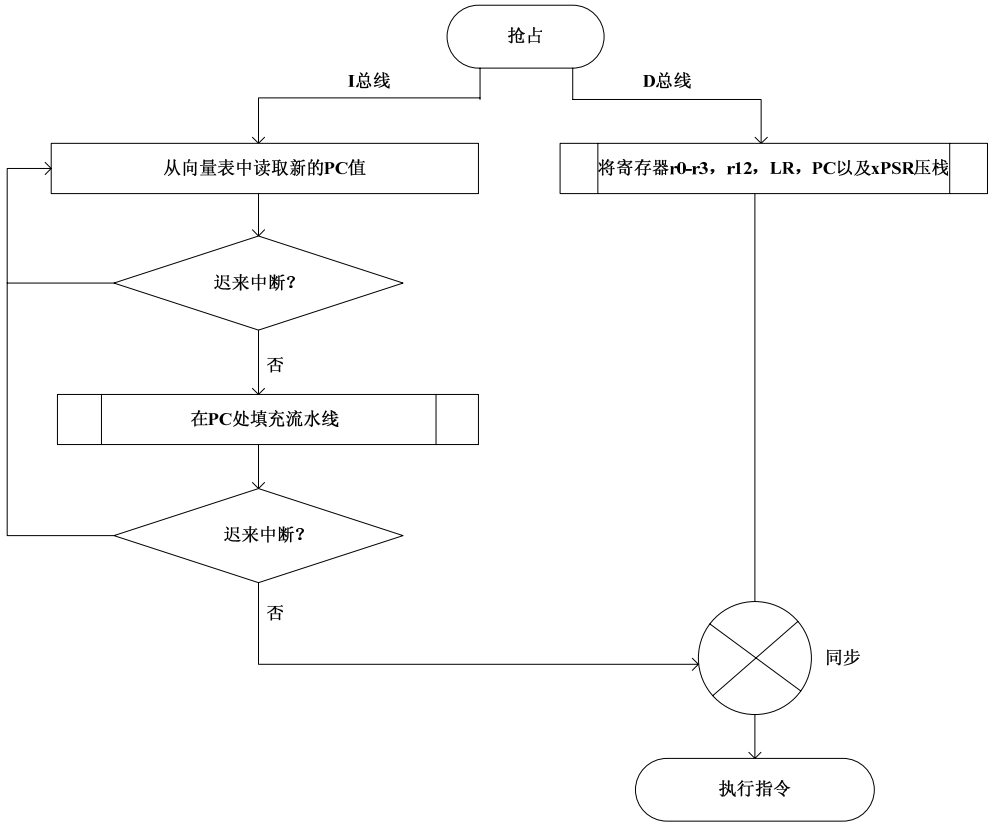


图 5-7 占先流程

5.14.3 返回

图 5-8 显示了处理器如何恢复被压栈的 ISR 或末尾连锁到优先级比被压栈的 ISR 更高的迟来中断。

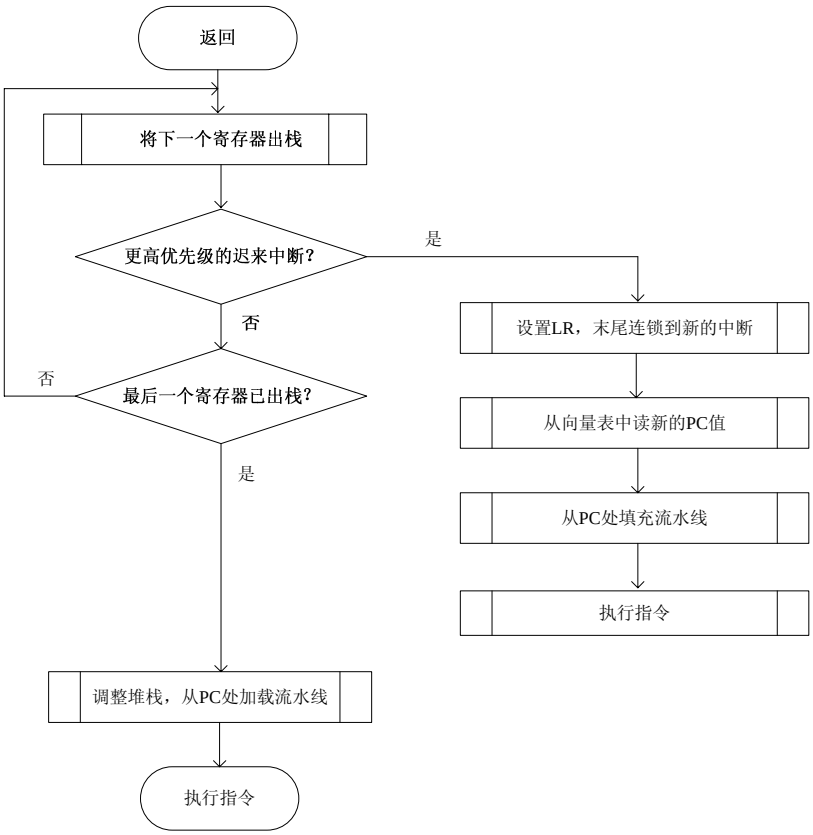

返回
将下一个寄存器出栈
更高优先级的迟来中断?
是
否
设置LR，末尾连锁到新的中断
从向量表中读新的PC值
从PC处填充流水线
执行指令
否
最后一个寄存器已出栈?
是
调整堆栈，从PC处加载流水线
执行指令

图 5-8　中断返回流程图

第6章 时钟和复位

本章介绍了处理器的时钟和复位。包括以下小节：

- *Cortex-M3 时钟*
- *Cortex-M3 复位*
- *Cortex-M3 复位方式*

6.1 Cortex-M3 时钟

处理器含 3 个功能时钟输入。如表 6-1 所示。

表 6-1　Cortex-M3 处理器时钟

时钟	域	描述
FCLK	处理器	自由振荡的处理器时钟，用来采样中断和为调试模块计时。在处理器休眠时，通过 FCLK 保证可以采样到中断和跟踪休眠事件。
HCLK	处理器	处理器时钟
DAPCLK	处理器	调试端口（AHB-AP）时钟

FCLK 和 **HCLK** 互相同步。**FCLK** 是一个自由振荡的 **HCLK**。要了解更多信息，请查看第七章*“电源管理”*。**FCLK** 和 **HCLK** 应该互相平衡，保证进入 Cortex-M3 时的延迟相同。

处理器集成了供调试和跟踪使用的元件。宏单元可以包含表 6-2 所示的一些或全部时钟。

表 6-2　Cortex-M3 宏单元时钟

时钟	域	描述
SWCLK	SW-DP	串行线时钟
TRACECLKIN	TPIU	用于为 TPIUD 的输出计时
TCK	JTAG-DP	TAP 时钟

SWCLK 是串行线时钟，用来对*串行调试端口*（SW-DP）的 SWDIN 输入进行计时。SWCLK 与其他所有时钟异步。

TCK 是*跟踪访问端口（TAP）*的时钟。它对 JTAG-DP TAP 进行计时。TCK 也与其他所有时钟异步。

TRACECLKIN 是*跟踪端口接口单元*（TPIU）的参考时钟。它与其他所有时钟异步。

注：

TCK，**SWCLK** 和 **TRACECLKIN** 都只是在设备分别含 JTAG-DP，SW-DP 和 TPIU 模块时才必须驱动。否则，必须断开（tied off）这些时钟输入。

注：

Cortex-M3 还包含一个 **STCLK** 输入。这个端口不是时钟。它是 SysTick 计数器的一个参考输入，其频率必须小于 **FCLK**/2。处理器将 **STCLK** 变成与 **FCLK** 内部同步。

6.2　Cortex-M3 复位

Cortex-M3 处理器含 3 个复位输入。如表 6-3 所示。

表 6-3　复位输入

复位输入	描述
PORESETn	复位整个处理器系统，JTAG-DP 除外
SYSRESETn	复位整个处理器系统，NVIC 中的调试逻辑、FPB、DWT、ITM 以及 AHB-AP 除外
nTRST	复位 JTAG-DP

注：

nTRST 复位 JTAG-DP。如果设备不含 JTAG-DP，那么必须断开（tied off）该复位端。

6.3　Cortex-M3 复位方式

通过处理器设计中出现的复位信号，用户可以单独对设计中的不同元件进行复位。表 6-4 列出了这些复位信号和他们的组合形式，以及可能的应用。

表 6-4　复位方式

复位方式	SYSRESETn	nTRST	PORESETn	应用
上电复位	x	0	0	接通电源后复位，复位整个系统。冷复位
系统复位	0	x	1	复位处理器内核和系统元件，调试元件除外
JTAG-DP 复位	1	0	1	复位 JTAG-DP 逻辑

注：

PORESETn 复位 **SYSRESETn** 逻辑的超集（superset）。

6.3.1　上电复位

图 6-1 显示了供宏单元使用的复位信号。

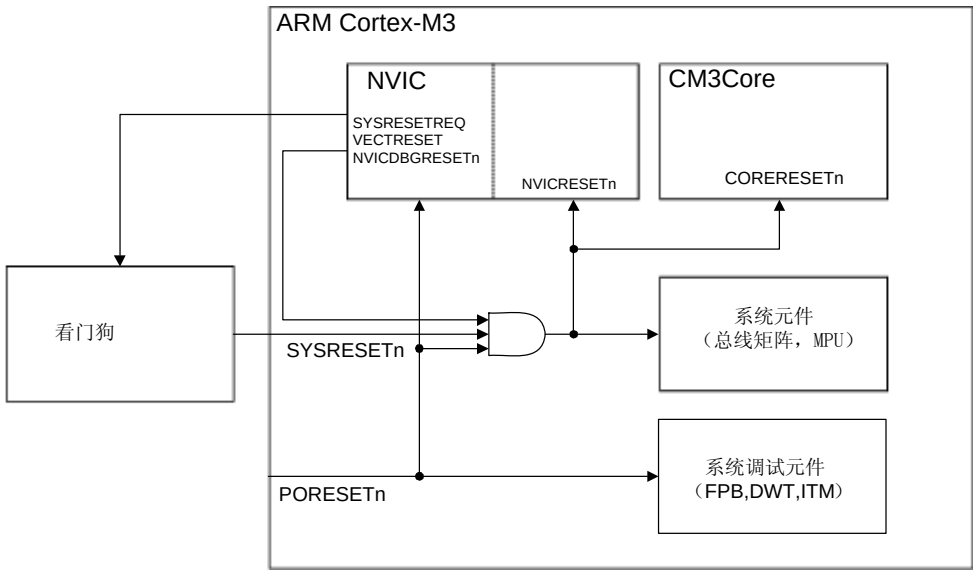


图 6-1　复位信号

在第一次给系统加电时，必须对处理器上电或*冷*复位。上电复位时，复位信号 PORESETn 的下降沿不必与 HCLK 同步。因为 PORESETn 在处理器内已经被同步了，所以不必再使它同步。图 6-2 显示了上电复位的应用情况。图 6-3 显示了处理器内的复位同步器。

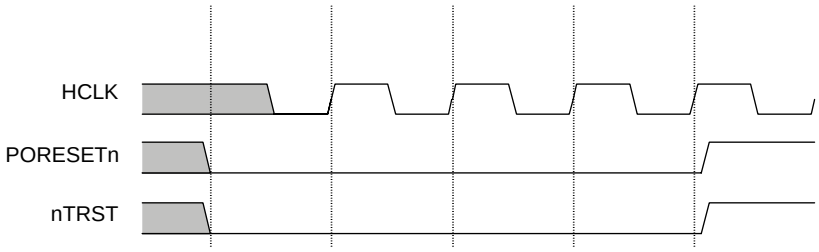


图 6-2　上电复位

为了保证正确的复位，建议复位信号至少保持 3 个 **HCLK** 周期有效。图 6-3 显示了内部复位的同步情况。

注：

在没有连接外部调试器时，所有外部看门狗电路都应该考虑使用 Cortex-M3 系统的 LOCKUP。

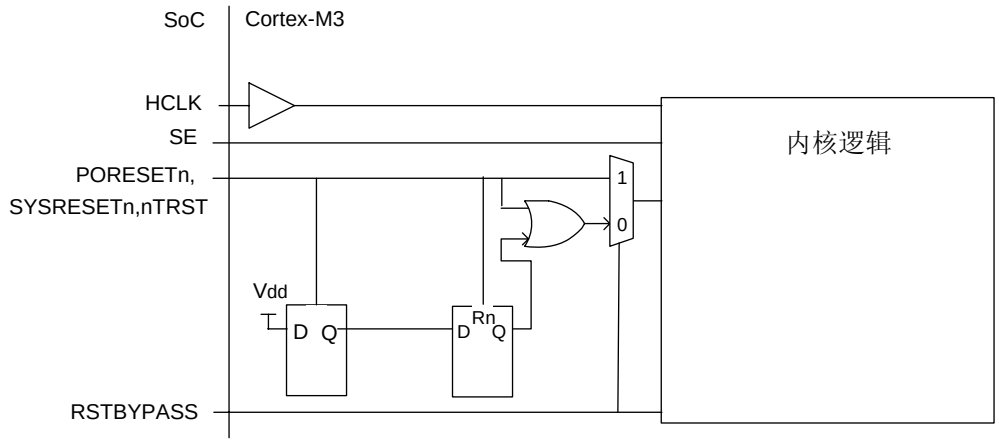


图 6-3　内部复位同步

6.3.2　系统复位

系统复位或*热*复位对宏单元中的大部分元件进行初始化，NVIC 调试逻辑、*Flash 修补与断点*（FPB）、*数据观察点与触发*（DWT）以及*仪表跟踪宏单元*（ITM）除外。系统复位主要用来复位已经工作一段时间的系统，比如看门狗复位。

由于 **SYSRESETn** 信号在处理器内已经被同步，所以不必使之再同步。图 6-3 显示了复位的同步情况。

Cortex-M3 输出一个 **SYSRESETREQ** 信号，该信号在“应用中断与复位控制寄存器”的 **SYSRESETREQ** 位置位时有效。用户可以使用该信号，例如将它用作图 6-1 所示的看门狗定时器的输入。

6.3.3 JTAG-DP 复位

nTRST 复位对 JTAG-DP 控制器的状态初始化。**nTRST** 复位通常被 RealView™ ICE 模块用作调试器与系统的热插拔连接。

nTRST 可以在不影响处理器正常工作的情况下对 JTAG-DP 控制器进行初始化。

由于 nTRST 在处理器内已经被同步了，所以不必使之再同步。

6.3.4 SW-DP 复位

SW-DP 是通过 **SWRSTn** 来复位的。该复位必须与 **SWCLK** 同步。

6.3.5 正常工作

在正常工作期间，不产生处理器复位和上电复位。如果没有使用 JTAG-DP 端口，**nTRST** 的值就变得不紧要了。

第7章 电源管理

本章介绍了处理器的电源管理功能。包括以下小节：

- *电源管理概述*
- *系统电源管理*

7.1 电源管理概述

处理器广泛地利用门时钟来禁能那些未用的功能和未用功能块的输入，因此只有正在有效使用中的逻辑才会消耗动态功率。

ARMv7-M 架构支持为减少功耗而让 Cortex-M3 和系统时钟停止运行的系统睡眠模式。详细情况在*“系统电源管理”*一节中作介绍。

7.2 系统电源管理

对系统控制寄存器进行写操作（见“*系统控制寄存器*”）可以控制 Cortex-M3 系统功耗的状态，表 7-1 列出了支持的睡眠模式。

表 7-1　支持的睡眠模式

睡眠机制	描述
立即睡眠	*等待中断*（WFI）或*等待事件*（WFE）指令请求立即睡眠模式。这些指令使得 NVIC 让处理器进入挂起其他异常事件的低功耗状态。
退出时睡眠	当系统控制寄存器的 SLEEPONEXIT 位置位时，一旦处理器退出最低优先级的 ISR，它就进入低功耗状态。处理器无需将寄存器出栈，就可进入低功耗状态，并且无需让寄存器压栈就可以产生下一个异常事件。内核一直处于睡眠状态直至别的异常被挂起。这是一个自动的 WFI 模式。 **注：** 在如调试等各种情况下，“退出时睡眠”可以返回到基址处。因而必须提供基址代码，如空闲循环（idle loop）或空闲线程。
深度睡眠	深度睡眠与立即睡眠和“退出时睡眠”机制共用。当系统控制寄存器的 SLEEPDEEP 位置位时，处理器指示系统可以进入深度睡眠。

a. 即使没有异常被激活也可以执行 WFI 指令。不要使用 WFI 指令来探测异常是否发生。WFI 通常使用在线程模式下的空闲循环中。要了解更多有关 WFI、WFE、BASEPRI 以及 PRIMASK 的信息，请参考“*ARMv7-M 架构参考指南*”。

处理器导出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。有关 **SLEEPING** 的用法实例见*“SLEEPING”*。

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。有关 SLEEPDEEP 的用法实例见*“SLEEPDEEP ”*。

7.2.1 SLEEPING

图 7-1 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。

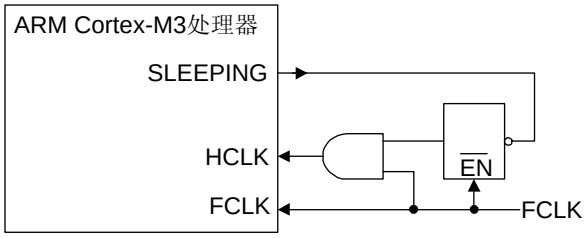


图 7-1　SLEEPING 功耗控制实例

为了探测中断，处理器必须一直接收自由振荡的 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，那些模块的功耗将会降低。见“*调试异常与监控寄存器*”。

在 SLEEPING 信号有效期间可以降低 FCLK 频率。

7.2.2 SLEEPDEEP

图 7-2 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。

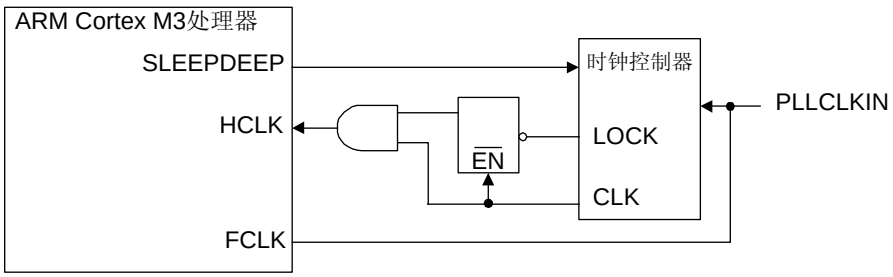


图 7-2　SLEEPDEEP 功耗控制实例

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。

第8章 嵌套向量中断控制器

本章介绍了*嵌套向量中断控制器*（NVIC）。包括以下小节：

- *NVIC 概述*
- *NVIC 编程器模型*
- *电平中断与脉冲中断*

8.1 NVIC 概述

NVIC：

- 便于进行低延迟的异常和中断处理
- 控制电源管理
- 执行系统控制寄存器

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（见*“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。

除非特别说明，否则所有的 NVIC 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。

处理器异常处理在第五章*“异常”*中介绍。

8.2 NVIC 编程器模型

本章列出和介绍了 NVIC 寄存器。包括以下内容：

- *NVIC 寄存器映射*
- *NVIC 寄存器描述*

8.2.1 NVIC 寄存器映射

表 8-1 列出了 NVIC 寄存器。NVIC 空间还用来实现系统控制寄存器。NVIC 空间分成以下部分：

- 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 8-1　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	[a]
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
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
Irq 0～31 激活位寄存器	只读	0xE000E29C	0x00000000
.	.	.	.
.	.	.	.
.	.	.	.
Irq 224～239 激活位寄存器	只读	0xE000E31C	0x00000000
Irq 0～31 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

续上表…

名称	类型	地址	复位值
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	0xE000ED08	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
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
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

a　复位值取决于被定义的中断数目

8.2.2 NVIC 寄存器描述

本节介绍了 NVIC 寄存器的使用方法。

注：

*存储保护单元（MPU）*寄存器和调试寄存器分别在第九章“*存储保护单元*”和第十章“*内核调试*”中介绍。

中断控制器类型寄存器

要了解 NVIC 支持的中断线数目，请查看“中断控制器类型寄存器”。

寄存器地址、访问类型和复位状态：

访问类型　　只读

地址　　0xE000E004

复位状态　　取决于在 Cortex-M3 实现中定义的中断数目。

中断控制器类型寄存器的位分配如图 8-1 所示。

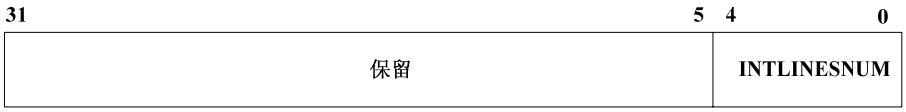


图 8-1　中断控制器类型寄存器的位分配

表 8-2 描述了中断控制器类型寄存器的各个位。

表 8-2　中断控制器类型寄存器的位分配

域	名称	定义
[31:5]	-	保留
[4:0]	INTLINESNUM	中断线总数（32 条一组）： b00000= 0…32* b00001= 33…64 b00010= 65…96 b00011= 97…128 b00100= 129…160 b00101= 161…192 b00110= 193…224 b00111= 225…256* * Cortex-M3 处理器仅支持 1 到 240 个外部中断。

系统时钟节拍（SysTick）控制与状态寄存器

使用系统时钟节拍（SysTick）控制与状态寄存器来使能 SysTick 功能。

寄存器地址、访问类型和复位状态：

地址　　0xE000E010

访问类型　　读/写

复位状态　　0x00000000

系统时钟节拍（SysTick）控制与状态寄存器的位分配如图 8-2 所示。

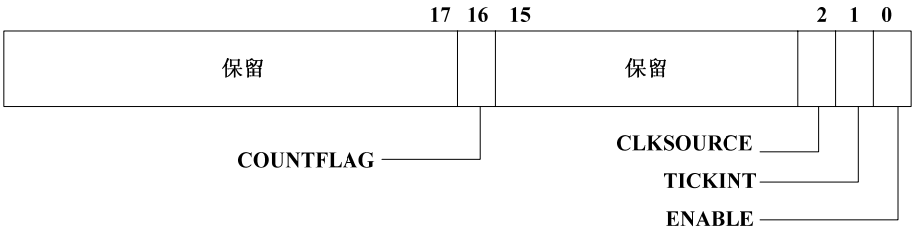


图 8-2　SysTick 控制与状态寄存器的位分配

表 8-3 描述了 SysTick 控制与状态寄存器的各个位。

表 8-3　SysTick 控制与状态寄存器的位分配

域	名称	定义
[16]	COUNTFLAG	从上次读取定时器器开始，如果定时器计数到 0，则返回 1。读取时清零。
[2]	CLKSOURCE	0= 外部参考时钟 1= 内核时钟 如果没有提供参考时钟，那么该位保持为 1，并且因此赋予和内核时钟一样多的时间。内核时钟比参考时钟至少要快 2.5 倍。否则计数值将不可预测。
[1]	TICKINT	1= 向下计数至 0 会导致挂起 SysTick 处理器 0= 向下计数至 0 不会导致挂起 SysTick 处理器。软件可以使用 COUNTFLAG 来判断是否计数到 0。
[0]	ENABLE	1= 计数器工作在连拍模式（multi-shot）。即计数器装载重装值后接着开始往下计数。到计数到 0 时将 COUNTFLAG 设为 1，此时根据 TICKINT 的值可以选择是否挂起 SysTick 处理器。接着又再次装载重装值，并重新开始计数。 0= 禁能计数器

系统时钟节拍（SysTick）重装值寄存器

在计数器到达 0 时，使用 SysTick 重装值寄存器来指定载入“当前值寄存器”的初始值。初始值可以是 1 到 0x00FFFFFF 之间的任何值。初始值也可以为 0，但因为从 1 计数到 0 时会将 SysTick 中断和 COUNTFLAG 激活，所以没有什么用处。

因此，作为一个连拍式（multi-shot）定时器，它每 N+1 个时钟脉冲就触发一次，周而复始，此处 N 为 1 到 0x00FFFFFF 之间的任意值。所以，如果每 100 个时钟脉冲就请求一次时钟中断（tick interrupt），那么必须向 RELOAD 载入 99。如果每次时钟中断后都写入一个新值，那么可以看作单拍（single shot）模式，因而必须写入实际的倒计数值。例如，如果在 400 个时钟脉冲后想请求一个时钟中断（tick），那么必须向 RELOAD 写入 400。

寄存器地址、访问类型和复位状态：

地址　　0xE000E014

访问类型　　读/写

复位状态　　不可预测

系统时钟节拍（SysTick）重装值寄存器的位分配如图 8-3 所示。

图 8-3　SysTick 重装值寄存器的位分配

表 8-4 描述了 SysTick 重装值寄存器的各个位。

表 8-4　SysTick 重装值寄存器的位分配

域	名称	定义
[23:0]	RELOAD	当计数器到达 0 时装载“当前值寄存器”的值

系统时钟节拍（SysTick）当前值寄存器

使用 SysTick 当前值寄存器来查找寄存器中的当前值。

寄存器地址、访问类型和复位状态：

地址　　0xE000E018

访问类型　　读/写清除

复位状态　　不可预测

SysTick 当前值寄存器的位分配如图 8-4 所示。

图 8-4　SysTick 当前值寄存器的位分配

表 8-5 描述了 SysTick 当前值寄存器的各个位。

表 8-5　SysTick 当前值寄存器的位分配

域	名称	定义
[23:0]	CURRENT	访问寄存器时的当前值。没有提供读-修改-写保护，所以在修改时要特别小心。该寄存器是写-清除。向该寄存器写入任意值都可以将其清除变为 0。清零该寄存器还会导致“SysTick 控制与状态寄存器”的 COUNTFLAG 位清零。

系统时钟节拍（SysTick）校准值寄存器

使用系统时钟节拍（SysTick）校准值寄存器通过乘法和除法运算可以将寄存器调节成任意所需的时钟速率。

寄存器地址、访问类型和复位状态：

地址　　0xE000E01C

访问类型　　读

复位状态　　STCALIB

SysTick 校验值寄存器的位分配如图 8-5 所示。

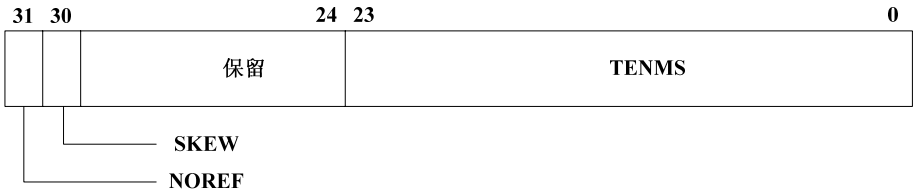


图 8-5　SysTick 校验值寄存器的位分配

表 8-6 描述了 SysTick 校验值寄存器的各个位。

表 8-6　SysTick 校验值寄存器的位分配

域	名称	定义
[31]	NOREF	1= 没有提供参考时钟
[30]	SKEW	1= 由于时钟频率的原因，校验值不是精确的 10ms。这会影响其作为一个软件实时时钟的适合性。
[23:0]	TENMS	该值是用于 10ms 定时的重装值。其值取决于 SKEW，它可以是精确的 10ms，也可以是最接近 10ms 的值。 如果为 0，那么检验值就未知。这很可能是因为参考时钟是系统的一个未知输入或者因为参考时钟可以动态调节。

中断使能设置寄存器

中断使能设置寄存器用于：

- 使能中断
- 决定当前使能的是哪个中断。

该寄存器的一个位对应一个中断（共 32 个中断）。置位中断使能设置寄存器的位可以使能相应的中断。

当挂起（pending）中断的使能位置位时，处理器会根据其优先级将其激活。使能位清零时，虽然其中断信号有效，可以将中断挂起，但不管其优先级如何，该中断都不能被激活。因此被禁能的中断可以当作一个被锁存的 GPIO 位。用户无需调用中断就可以直接对它进行读取和清零操作。

通过写 1 到中断使能清除寄存器（见 *“中断使能清除寄存器”*）的相应位可以清零中断使能设置寄存器的位。

注：

清零一个中断使能设置寄存器的位并不会影响当前运行的中断。它只是阻止激活新的中断。

寄存器地址、访问类型和复位状态：

地址　　0xE000E100-0xE000E11C

访问类型　　读/写

复位状态　　0x00000000

表 8-7 描述了中断使能设置寄存器的各个位。

表 8-7　中断使能设置寄存器的位功能

域	名称	定义
[31:0]	SETENA	中断使能设置位： 1= 使能中断 0= 禁能中断 写 0 到 SETENA 位没有什么用处。读取该位返回其当前状态。复位会将 SETENA 清零。

中断使能清除寄存器

中断使能清除寄存器用于：

- 禁能中断
- 决定当前被禁能的中断

该寄存器的一个位对应一个中断（共 32 个中断）。置位中断使能清除寄存器的位可以禁能相应的中断。

寄存器地址、访问类型和复位状态：

地址　　0xE000E180-0xE000E19C

访问类型　　读/写

复位状态　　0x00000000

表 8-8 描述了中断使能清除寄存器的各个位。

表 8-8　中断使能清除寄存器的位功能

域	名称	定义
[31:0]	CLRENA	中断使能清除位： 1= 禁能中断 0= 使能中断 写 0 到 CLRENA 位没有什么用处。读取该位返回其当前状态。

中断挂起（pend）设置寄存器

中断挂起设置寄存器用于：

- 将中断强制挂起
- 决定当前被挂起的中断

该寄存器的一个位对应一个中断（共 32 个中断）。置位中断挂起设置寄存器的位可以挂起相应的中断。

通过写 1 到中断挂起清除寄存器（见“*中断挂起清除寄存器*”）的相应位可以清零中断挂起设置寄存器的位。清零中断挂起设置寄存器的位不会将中断挂起。

注：

写中断挂起设置寄存器操作对已经挂起或已经被禁能的中断没有影响。

寄存器地址、访问类型和复位状态：

地址　　0xE000E200－0xE000E21C

访问类型　　读/写

复位状态　　0x00000000

表 8-9 描述了中断挂起设置寄存器的各个位。

表 8-9　中断挂起设置寄存器的位功能

域	名称	定义
[31:0]	SETPEND	中断挂起设置位： 1= 挂起相应的中断 0= 不挂起相应的中断 写 0 到 SETPEND 位没有什么用处。读取该位返回其当前状态。

中断挂起清除寄存器

中断挂起清除寄存器用于：

- 清除挂起中断
- 决定当前正在挂起哪个中断

该寄存器的一个位对应一个中断（共 32 个中断）。置位中断挂起清除寄存器的位可以让相应的挂起中断变为不激活状态。

注：

写中断挂起清除寄存器操作对那些已经激活的中断没有影响，除非这些中断也正处于挂起状态。

寄存器地址、访问类型和复位状态：

地址　　0xE000E280-0xE000E29C

访问类型　　读/写

复位状态　　0x00000000

表 8-10 描述了中断挂起清除寄存器的各个位。

表 8-10　中断挂起清除寄存器的位功能

域	名称	定义
[31:0]	CLRPEND	中断挂起清除位： 1= 清除挂起中断 0= 不清除挂起中断 向 CLRPEND 位写入 0 没什么用处。读取该位返回其当前状态。

激活位寄存器

通过读取激活位寄存器来判断激活哪个中断。寄存器的一个标志对应一个中断（共 32 个中断）。

寄存器地址、访问类型和复位状态：

地址　　0xE000E300－0xE000E31C

访问类型　　读/写

复位状态　　0x00000000

表 8-11 描述了激活位寄存器的各个位。

表 8-11　激活位寄存器的位功能

域	名称	定义
[31:0]	ACTIVE	中断激活标志： 1= 中断被激活或者被抢占和压栈 0= 中断不被激活或中断未被压栈

中断优先级寄存器

使用中断优先寄存器将 0 到 255 个优先级分别分配给各个中断。0 代表最高优先级，255 则代表最低优先级。

优先级寄存器首先存放最高位（MSB）。即当优先级值为 4 位时，存放在字节的位[7:4]中。优先级值为 3 位时，存放在字节的位[7:5]中。这也意味着某个应用即使不知道可能含有多少个优先级也可以正常工作。

寄存器地址、访问类型和复位状态：

地址　　0xE000E400-0xE000E41C

访问类型　　读/写

复位状态　　0x00000000

图 8-6 描述了中断优先级寄存器 0-7 的各个位。

	31 24	23 16	15 8	7 0
E000E400	PRI_3	PRI_2	PRI_1	PRI_0
E000E404	PRI_7	PRI_6	PRI_5	PRI_4
E000E408	PRI_11	PRI_10	PRI_9	PRI_8
E000E40C	PRI_15	PRI_14	PRI_13	PRI_12
E000E410	PRI_19	PRI_18	PRI_17	PRI_16
E000E414	PRI_23	PRI_22	PRI_21	PRI_20
E000E418	PRI_27	PRI_26	PRI_25	PRI_24
E000E41C	PRI_31	PRI_30	PRI_29	PRI_28

图 8-6　中断优先级寄存器 0-31 的位分配

PRI_*n* 的低位可以为优先级分组指定子优先级。见 *“异常优先级”*。

表 8-12 描述了中断优先级寄存器的各个位。

表 8-12　中断优先级寄存器 0-31 的位分配

域	名称	定义
[7:0]	PRI_*n*	中断 *n* 的优先级

CPU ID 基址寄存器

读取 CPU ID 基址寄存器的值来决定：

- Cortex-M3 内核的 ID 号
- Cortex-M3 内核的版本号
- 内核的详细执行情况

寄存器地址、访问类型和复位状态：

地址　　0xE000ED00

访问类型　　只读

复位状态　　0x410FC230

CPUID 基址寄存器的位分配如图 8-7 所示。

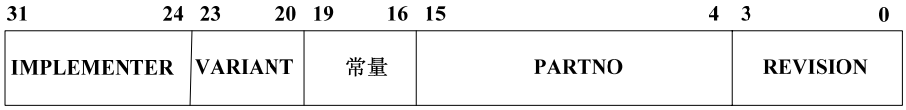


图 8-7　CPUID 基址寄存器的位分配

表 8-13 描述了 CPUID 基址寄存器的各个位。

表 8-13　CPUID 基址寄存器的位分配

域	名称	定义
[31:24]	IMPLEMENTER	实现者代码。ARM 为 0x41
[23:20]	VARIANT	设备已定义的变量号
[19:16]	常量	读作 0xF
[15:4]	PARTNO	系列处理器的编号： [11:10] b11 = Cortex 系列 [9:8] b00 = 版本 [7:6] b00 = 保留 [5:4] b10= M（v7-M） [3:0]X = 系列成员。Cortex-M3 为 b0011。
[3:0]	REVISION	设备已定义的版本号

中断控制状态寄存器

中断控制状态寄存器用于：

- 设置一个挂起（pending）NMI
- 设置或清除一个挂起 SVC

- 设置或清除一个挂起 SysTick
- 查找挂起异常
- 查找最高优先级挂起异常的向量号
- 查找激活异常的向量号

寄存器地址、访问类型和复位状态：

地址　　0xE000ED04

访问类型　　读/写或只读

复位状态　　0x00000000

中断控制状态寄存器的位分配如图 8-8 所示。

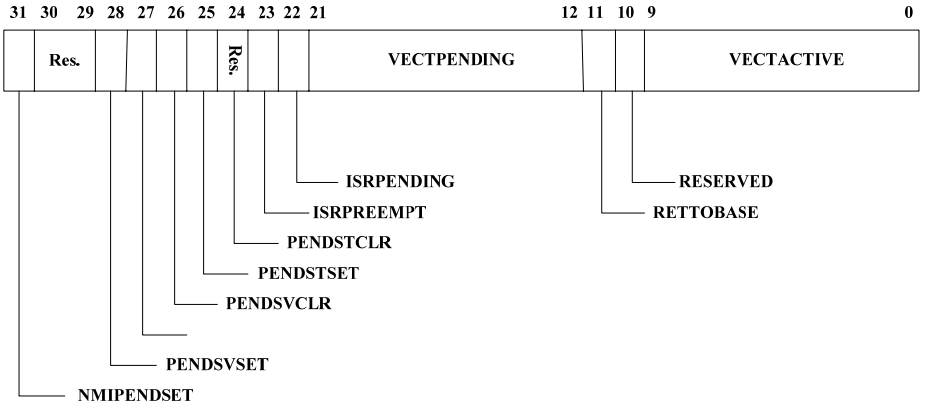


图 8-8　中断控制状态寄存器的位分配

表 8-14 描述了中断控制状态寄存器的各个位。

表 8-14　中断控制状态寄存器的位分配

域	名称	类型	定义
[31]	NMIPENDSET	读/写	设置挂起（pending）NMI 位： 1= 设置挂起 NMI 0= 不设置挂起 NMI NMIPENDSET 挂起并激活一个 NMI。因为 NMI 是优先级最高的中断，所以只要它符合上述值就可以生效。
[30:29]	-	-	保留
[28]	PENDSVSET	读/写	设置挂起 pendSV 位： 1= 设置挂起 pendSV 0= 不设置挂起 pendSV
[27]	PENDSVCLR	只写	清除挂起 pendSV 位 1= 清除挂起 pendSV 0= 不清除挂起 pendSV
[26]	PENDSTSET	读/写	设置挂起 SysTick 位 1= 设置挂起 SysTick 0= 不设置挂起 SysTick

续上表…

域	名称	类型	定义
[25]	PENDSTCLR	只写	清除挂起 SysTick 位 1= 清除挂起 SysTick 0= 不清除挂起 SysTick
[23]	ISRPREEMPT	只读	必须仅在调试时使用。它表示挂起中断将在下一个运行周期有效。如果 C_MASKINTS 在“调试停止控制与状态寄存器”中清零，则开始中断处理。
[22]	ISRPENDING	只读	中断挂起标志。NMI 和故障（Faults）除外： 1= 中断挂起 0= 中断不挂起
[21:12]	VECTPENDING	只读	挂起 ISR 的号码域。VECTPENDING 包含最高优先级的挂起 ISR 的中断号。
[10]	-	-	保留
[11]	RETTOBASE	只读	如果没有其他异常处于挂起状态，那么当“从异常返回”返回至激活的基本级（base level）时，该位为 1。如果处于线程状态，或处于基本级（Base）包含超过 1 个激活级别的处理状态（in a Handler more than one level of activation from Base），抑或处于未标志为激活（返回故障）的处理状态时，该位为 0。
[9:0]	VECTACIVE	只读	激活 ISR 的号码域。VECTACTIVE 包含当前正在运行的 ISR 的中断号，包括 NMI 和硬故障（Hard Fault）。共用的处理器可以使用 VECTACTIVE 来判断自己被哪个中断调用。可以用 VECTACTIVE 域减去 16 来指向中断使能清除/设置寄存器、中断挂起清除/设置寄存器和中断优先级寄存器。**INTISR[0]**向量号为 16。复位会将 VECTACTIVE 域清除。

向量表偏移寄存器

向量表偏移寄存器用来决定：

- 向量表是位于 RAM 还是程序存储器中
- 向量表的偏移量

寄存器地址、访问类型和复位状态：

地址　　0xE000ED08

访问类型　　读/写

复位状态　　0x00000000

向量表偏移寄存器的位分配如图 8-9 所示。

图 8-9　向量表偏移寄存器的位分配

表 8-15 描述了向量表偏移寄存器的各个位。

表 8-15　向量表偏移寄存器的位分配

域	名称	定义
[31:30]	-	保留
[29]	TBLBASE	向量表基址位于 Code（0）或 RAM（1）处
[28:7]	TBLOFF	向量表的基址偏移域。包括向量表的基址与 SRAM 或 CODE 空间的底部的偏移量。
[6:0]	-	保留

向量表偏移寄存器将向量表定位在 CODE 或 SRAM 中。默认情况下复位时为 0（CODE 空间）。定位时，偏移量必须根据表中异常的数目来对齐。即最小的对齐是 32 字对齐，可供 16 个中断使用。但当多于 16 个中断时，你必须通过“四舍五入（round up）”将其对齐调节成 2 的下一个乘幂。例如，如果你需要 21 个中断，而由于表的大小是 37 个字，2 的下一个乘幂（$2^{(5+1)}$）为 64，因此必须在 64 字边界上实现对齐。

应用中断与复位控制寄存器

应用中断与复位控制寄存器用于：

- 决定数据的字节顺序
- 清除所有有效的状态信息，以便进行调试或从硬故障中恢复
- 执行系统复位
- 改变优先级分组位置（二进制小数点）

寄存器地址、访问类型和复位状态：

地址　　0xE000ED0C

访问类型　　读/写

复位状态　　0x00000000

应用中断与复位控制寄存器的位分配如图 8-10 所示。

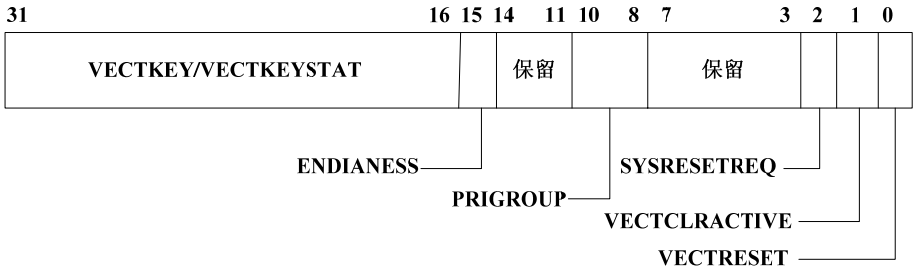


图 8-10　应用中断与复位控制寄存器的位分配

表 8-16 描述了应用中断与复位控制寄存器的各个位。

表 8-16　应用中断与复位控制寄存器的位分配

域	名称	定义
[31:16]	VECTKEY	注册码（register key）。对寄存器进行写操作时要求在 VECTKEY 域中写入 0x5FA。否则写入值被忽略。
[31:16]	VECTKEYSTAT	读取时为 0xFA05
[15]	ENDIANESS	数据的字节顺序位： 1= 大端（高位在前） 0= 小端（低位在前） ENDIANESS 在复位期间从 BIGEND 输入端采样。只有复位外才能修改 ENDIANESS
[14:11]	-	保留
[10:8]	PRIGROUP	中断优先级分组域： PRIGROUP　从子优先级中拆分强占式优先级 0　7.1 表示 7 位抢占式优先级，1 位子优先级 1　6.2 表示 6 位抢占式优先级，2 位子优先级 2　5.3 表示 5 位抢占式优先级，3 位子优先级 3　4.4 表示 4 位抢占式优先级，4 位子优先级 4　3.5 表示 3 位抢占式优先级，5 位子优先级 5　2.6 表示 2 位抢占式优先级，6 位子优先级 6　1.7 表示 1 位抢占式优先级，7 位子优先级 7　0.8 表示 0 位抢占式优先级，8 位子优先级 PRIGROUP 域是一个二进制小数点定位指示器，用于为共用同一抢占级别的异常创建优先级。它将中断优先级的 PRI_*n* 域分成抢占式优先级和子优先级。二进制小数点是一个偏左值。即 PRIGROUP 值代表一个从 LSB 左边开始的小数值。这是 7:0 的位 0。 最低的值不能为 0，这取决于为优先级分配的位数以及设备的选择
[7:3］	-	保留
[2]	SYSRESETREQ	让信号在外部系统有效，表示请求复位。试图强制对调试元件之外的所有大型元件进行大的系统复位。该位置位并不会阻止“停止调试（Halting Debug）”运行。
[1]	VECTCLRACTIVE	清除有效向量位： 1= 清除活动 NMI、故障和中断的所有状态信息 0= 不清除 重新初始化堆栈是应用程序的职责。VECTCLRACTIVE 位供调试过程返回至已知状态使用，它会自己清零。 此操作不会清除 IPSR。因此如果应用程序使用了 IPSR，那么 IPSR 必须只能使用在激活基本级（base level）或活动位能够置位的系统处理器内。
[0]	VECTRESET	系统复位位。将系统复位，调试元件除外： 1= 复位系统 0= 不复位系统 VECTRESET 位自己清零。复位会导致 VECTRESET 位清零。对于调试，仅在内核停止运行时才对这个位进行写操作。

系统控制寄存器

将系统控制寄存器用于电源管理功能：

- 当 Cortex-M3 处理器可以进入低功耗状态时发信号给系统
- 对处理器进入和退出低功耗状态的方式进行控制

寄存器地址、访问类型和复位状态：

地址　　0xE000ED10

访问类型　　读/写

复位状态　　0x00000000

系统控制寄存器的位分配如图 8-11 所示。

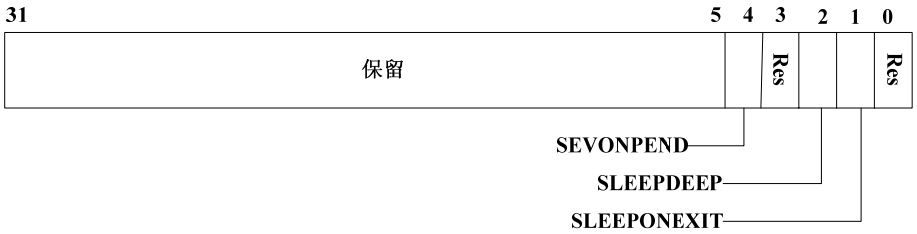


图 8-11　系统控制寄存器的位分配

表 8-17 描述了系统控制寄存器的各个位。

表 8-17　系统控制寄存器的位分配

域	名称	定义
[31:5]	-	保留
[4]	SEVONPEND	使能时，在中断从没有挂起到挂起时 SEVONPEND 可以唤醒 WFE。否则 WFE 只能通过事件信号、外部和产生的 SEV 指令来唤醒。事件输入 **RXEV**，即使在不等待事件时也会被记录，同样也会影响下一个 WFE。
[2]	SLEEPDEEP	深度睡眠位： 1= 向系统指示可以停止 Cortex-M3 时钟。置位该位会使得 **SLEEPDEEP** 端口在可以停止处理器时变为有效。 0= 不适合将系统时钟关闭。 要了解更多有关 **SLEEPDEEP** 的用法信息，请参考第七章“电源管理”。
[1]	SLEEPONEXIT	当从处理器模式返回到线程模式时，开始“退出时睡眠” 1= 退出 ISR 时开始睡眠 0= 返回到线程模式时不睡眠 使得中断驱动应用程序可以避免返回到空的主应用程序中。
[0]	-	保留

配置控制寄存器

配置控制寄存器用于：

- 让 NMI、硬故障和 FAULTMASK 忽略总线故障
- 捕获除 0 和不对齐访问
- 让用户可以访问软件触发异常寄存器
- 控制线程模式（Thread mode）的进入

寄存器地址、访问类型和复位状态：

地址　　0xE000ED14

访问类型　　读/写

复位状态　　0x00000000

配置控制寄存器的位分配如图 8-12 所示。

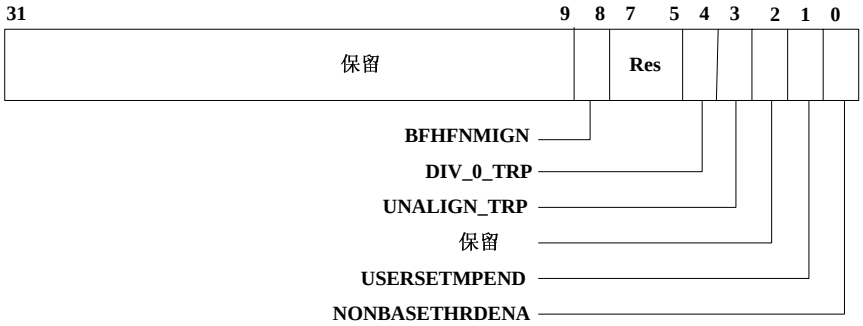


图 8-12　配置控制寄存器的位分配

表 8-18 描述了配置控制寄存器的各个位。

表 8-18　配置控制寄存器的位分配

域	名称	定义
[8]	BFHFNMIGN	使能时，使得以优先级-1 和-2（硬故障，NMI 和 FAULTMASK 升级处理器）运行的处理器忽略装载和存储指令引起的数据总线故障。禁能时，这些总线故障会引起锁存。在使用该使能位时要特别小心。忽略所有数据总线故障——必须仅在处理器和其数据处于完全安全的存储器时使用。一般将它用于探测系统设备和桥接器以检测并纠正控制信道问题。
[4]	DIV_0_TRP	除 0 陷阱。在试图进行除 0 操作时，会导致故障或停止。相关的“使用故障状态寄存器”位是 DIVBYZERO，见 *“使用故障状态寄存器”*。
[3]	UNALIGN_TRP	不对齐访问陷阱。在出现任何不对齐的半字或全字访问时会导致故障或停止。不对齐的多寄存器装载－存储总是出错。相关的 *“使用故障状态寄存器”*位是 UNALIGNED，见 *“使用故障状态寄存器”*。
[1]	USERSETMPEND	如果写成 1，那么用户代码可以写软件触发中断寄存器以触发（挂起）一个主异常，该异常是和主堆栈指针相联系的。
[0]	NONEBASETHRDENA	为 0 时，默认下在从上次异常返回时只能进入线程模式。为 0 时，通过受控的返回值可以从处理器模式的任意级别进入线程模式。

系统处理器优先级寄存器

通过 3 个处理器优先级寄存器为以下系统处理器排列优先级：

- 存储器管理
- 总线故障
- 使用故障
- 调试监控
- SVC
- SysTick
- PendSV

系统处理器是一种特殊的异常处理器，它可以将自己的优先级设置成任意级别。大多数系统处理器都可以打开屏蔽（使能）或关闭屏蔽（禁能）。禁能时，故障总是被当作硬故障。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED18，0xE000ED1C，0xE000ED20

访问类型　　读/写

复位状态　　0x00000000

系统处理器优先级寄存器的位分配如图 8-13 所示。

	31 24	23 16	15 8	7 0
E000ED18	PRI_7	PRI_6	PRI_5	PRI_4
E000ED1C	PRI_11	PRI_10	PRI_9	PRI_8
E000ED20	PRI_15	PRI_14	PRI_13	PRI_12

图 8-13　系统处理器优先级寄存器的位分配

表 8-19 描述了系统处理器优先级寄存器的各个位。

表 8-19　系统处理器优先级寄存器的位分配

域	名称	定义
[31:24]	PRI_N3	系统处理器 7，11 和 15 的优先级。分别对应于：保留，SVCall 和 SysTick
[23:16]	PRI_N2	系统处理器 6，10 和 14 的优先级。分别对应于：使用故障，保留和 PendSV
[15:8]	PRI_N1	系统处理器 5，9 和 3 的优先级。分别对应于：总线故障，保留和保留
[7:0]	PRI_N	系统处理器 4，8 和 12 的优先级。分别对应于：存储器管理，保留和调试监控

系统处理器控制与状态寄存器

系统处理器控制与状态寄存器用于：

- 使能或禁能系统处理器
- 决定总线故障、存储器管理故障以及 SVC 的挂起（pending）状态
- 决定系统处理器的激活状态

如果在故障处理器被禁能时发生故障条件，那么故障将升级为硬故障。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED24

访问类型　　读/写

复位状态　　0x00000000

系统处理器控制与状态寄存器的位分配如图 8-14 所示。

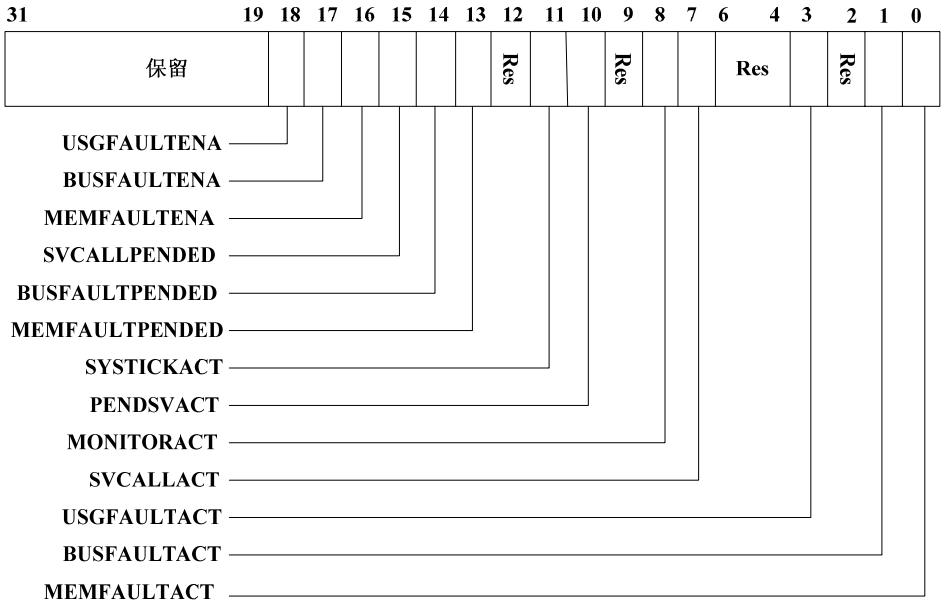


图 8-14　系统处理器控制与状态寄存器的位分配

表 8-20 描述了系统处理器控制寄存器的各个位。

表 8-20　系统处理器控制与状态寄存器的位分配

域	名称	定义
[31:19]	-	保留
[18]	USGFAULTENA	设为 0 时禁能，设为 1 时使能
[17]	BUSFAULTENA	设为 0 时禁能，设为 1 时使能
[16]	MEMFAULTENA	设为 0 时禁能，设为 1 时使能
[15]	SVCALLPENDED	如果在开始调用 SVCall 时被一个更高优先级的中断取代而导致 SVCall 被挂起，那么为 1。
[14]	BUSFAULTPENDED	如果在开始调用 BusFault 时被一个更高优先级的中断取代而导致 BusFault 被挂起，那么为 1。
[13]	MEMFAULTPENDED	如果在开始调用 MemManage 时被一个更高优先级的中断取代而导致 MemManage 被挂起，那么为 1。
[12]	-	保留
[11]	SYSTICKACT	如果 SysTick 已激活，那么为 1
[10]	PENDSVACT	如果 PendSV 已激活，那么为 1

续上表…

域	名称	定义
[9]	-	保留
[8]	MONITORACT	如果监控器已激活，那么为 1
[7]	SVCALLACT	如果 SVCall 已激活，那么为 1
[6:4]	-	保留
[3]	USGFAULTACT	如果 UsageFault 已激活，那么为 1
[2]	-	保留
[1]	BUSFAULTACT	如果 BusFault 已激活，那么为 1
[0]	MEMFAULTACT	如果 MemManage 已激活，那么为 1

激活位表示如果任意系统处理器被激活，则会立即运行或者由于占先而被压栈。这可以用于调试和应用处理器中。挂起位仅在以后不会再发生的故障由于出现更高优先级的迟来中断而被延迟的情况下才置位。

注：

有效位可以被写、清零或置位，但是使用时要必须特别小心。清零和置位不会修复堆栈内容也不会清除其他数据结构。即使位于故障处理器中，清零和置位也可以被上下文转换器（context switcher）用来保存线程的上下文。最普遍的做法是保存位于 SVCall 处理器或 UsageFault 处理器的线程的上下文，供未定义指令和协处理器仿真使用。进行这些操作的模型是为了保存当前状态，从含有处理器上下文的堆栈中退出，装载新线程的状态，转入新线程的堆栈，然后返回至线程。由于 IPSR 没有改变，没有反映这些情况，所以当前处理器的激活位不能清零。该激活位只能用来修改被压栈的激活处理器。

如上文所述，SVCALLPENDED 和 BUSFAULTPENDED 位在相应的处理器被迟来的中断推迟时置位，直到优先的处理器真正被激活才清零。即如果在 SVCall 或 BusFault 处理器启动前发生堆栈错误或向量读错误，那么这些位不清零。这就使得入栈错误或向量读错误处理器可以选择将他们清除或再试。

可配置故障状态寄存器

使用 3 个配置故障状态寄存器来获取有关局部错误的信息。这些寄存器包括：

- *存储器管理故障状态寄存器*
- *总线故障状态寄存器*
- *使用故障状态寄存器*

这些寄存器中的标志表示引起本地故障的原因。如果发生一个以上的故障，那么可以置位多个标志。这些寄存器可以读/写-清除。即他们可以被正常地读，但是向任意位写 1 会将该位清除。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED28 存储器管理故障状态寄存器

0xE000ED29 总线故障状态寄存器

0xE000ED28 使用故障状态寄存器

访问类型　　读/写清除

复位状态　　0x00000000

可配置故障状态寄存器的位分配如图 8-15 所示。

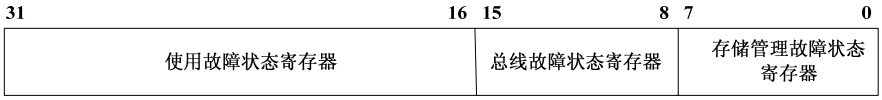


图 8-15　局部故障状态寄存器的位分配

存储器管理故障状态寄存器

存储器管理故障状态寄存器中的标志位表示引起存储器访问故障的原因。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED28

访问类型　　读/写清除

复位状态　　0x00000000

存储器管理故障状态寄存器的位分配如图 8-16 所示。

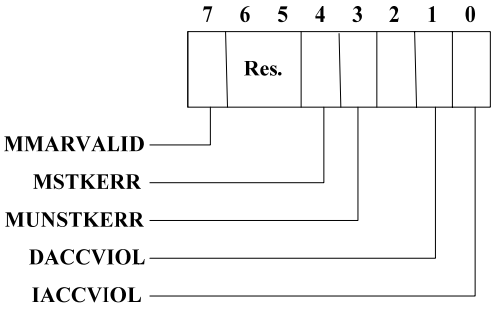


图 8-16　存储器管理故障寄存器的位分配

表 8-21 描述了存储器管理故障状态寄存器的各个位。

表 8-21　存储器管理故障状态寄存器的位分配

域	名称	定义
[7]	MMARVALID	*存储器管理地址寄存器*（MMAR）地址有效标志： 1= MMAR 中有效的故障地址。迟来的故障，如总线故障，可以将存储器管理故障清除 0= MMAR 中没有有效的故障地址 如果发生 MemManage 故障，并由于优先级的原因升级成一个硬故障，那么硬故障处理器必须清除该位。这样，避免在返回到一个 MMAR 值已被覆写的被压栈的活动 MemManage 处理器时出现问题。
[4]	MSTKERR	进入异常时的压栈操作引起了 1 个或 1 个以上的访问违犯。依然要对 SP 进行调节，并且堆栈的上下文域的值可能不正确。没有写 MMAR。
[3]	MUNSTKERR	异常返回时的出栈操作引起了 1 个或 1 个以上的访问违犯。它与处理器相连，所以原始的返回堆栈仍然存在。返回失败不能对 SP 进行调节，并且不会执行新的保存操作。没有写 MMAR。

续上表…

域	名称	定义
[1]	DACCVIOL	数据访问违犯标志。试图在不允许执行加载或存储的单元上进行加载和存储将导致 DACCVIOL 置位。返回的 PC 指向出错指令。该故障在 MMAR 中装载目标访问的地址。
[0]	IACCVIOL	指令访问违犯标志。试图在不允许执行取指操作的的单元上取指将导致 IACCVIOL 标志置位。即使 MPU 被禁能或不存在，这些操作也会在每次访问 XN 区时发生。返回的 PC 指向出错指令。没有写 MMAR。

总线故障状态寄存器

总线故障状态寄存器的标志表示引起总线访问故障的原因。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED29

访问类型　　读/写清除

复位状态　　0x00000000

总线故障状态寄存器的位分配如图 8-17 所示。

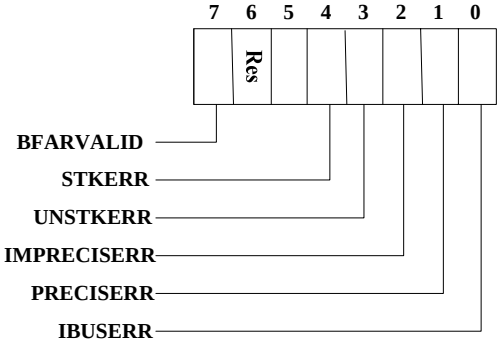


图 8-17　总线故障状态寄存器的位分配

表 8-22 描述了总线故障状态寄存器的各个位。

表 8-22　总线故障状态寄存器的位分配

域	名称	定义
[7]	BFARVALID	如果*总线故障地址寄存器*（BFAR）包含有效的地址，那么该位置位。这在地址已知的总线故障后是正确的。该位可以被其他故障，如之后发生的 Mem Manage 故障清零。如果发生总线故障，并升级成一个硬故障，那么硬故障处理器必须清除该位。当返回至一个 BFAR 值已被覆写的被压栈的活动总线故障处理器时，这样做可以阻止问题的出现。
[6:4]	-	保留
[4]	STKERR	进入异常时的压栈操作引起了 1 个或 1 个以上的总线故障。依然要对 SP 进行调节，并且堆栈的上下文域的值可能不正确。没有写 BFAR。
[3]	UNSTKERR	异常返回时的出栈操作引起了 1 个或 1 个以上的总线故障。它与处理器相连，所以原始的返回堆栈仍然存在。返回失败不能对 SP 进行调节，并且不会执行新的保存操作。没有写 BFAR。

续上表…

域	名称	定义
[2]	IMPRECISERR	不精确的数据总线错误。它是 BusFault，但是返回 PC 与出错指令无关。它不是同步故障。因此如果在当前激活的优先级高于总线故障时进行检测到该故障，那么只能挂起。总线故障在返回至一个更低优先级的激活时开始激活。如果在返回至一个更低优先级的异常前发生精确的故障，那么处理器同时对 IMPRECISERR 和其中的一个精确故障状态位进行探测，判断它们是否置位。没有写 BFAR。
[1]	PRECISERR	精确的数据总线错误返回
[0]	IBUSERR	指令总线错误标志： 1= 指令总线错误 0= 无指令总线错误 IBUSERR 标志在发生预取错误时置位。故障在执行完该指令后才发生，所以如果在分支指令后发生总线错误，则不会发生故障。没有写 BFAR。

使用故障状态寄存器

使用故障状态寄存器的标志表示下列错误：

- EPSR 和指令的非法组合
- 非法的 PC 装载
- 非法的处理器状态
- 指令译码错误
- 试图使用协处理器指令
- 非法的不对齐访问

寄存器地址、访问类型和复位状态：

地址　　0xE000ED2B

访问类型　　读/写清除

复位状态　　0x00000000

使用故障状态寄存器的位分配如图 8-18 所示。

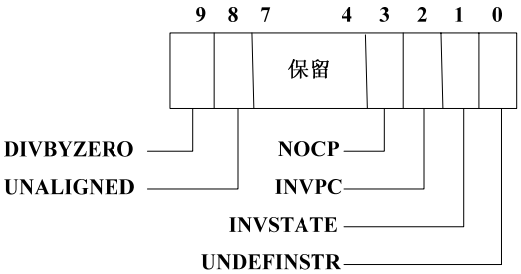


图 8-18　使用故障状态寄存器的位分配

表 8-23 描述了使用故障状态寄存器的各个位。

表 8-23　使用故障状态寄存器的位分配

域	名称	定义
[9]	DIVBYZERO	当 DIV_0_TRP（见 *“配置控制寄存器”*）使能且 SDIV 或 UDIV 指令的除数为 0 时，该故障产生。执行指令且返回 PC 指向该指令。若 DIV_0_TRP 没有置位，除法则返回商（0）。
[8]	UNALIGNED	当 UNALIGN_TRP 使能（见 *“配置控制寄存器”*）且存在试图进行不对齐的存储器访问时，该故障产生。不对齐的 LDM/STM/LDRD/STRD/LDC/STC 指令总是出错，而与 UNALIGN_TRP 的设置无关。
[7:4]	-	保留
[3]	NOCP	试图使用协处理器指令。处理器不支持协处理器指令
[2]	INVPC	试图将 EXC_RETURN 非法载入 PC。无效的指令、无效的上下文、无效的值。返回 PC 指向试图设置 PC 的指令。
[1]	INVSTATE	EPSR 和指令的非法组合，由未定义指令引发的除外。返回 PC 指向出错指令（无效状态）
[0]	UNDEFINSTR	在处理器试图执行一个未定义的指令时置位 UNDEFINSTR 标志。这是一条不能被译码的指令。返回 PC 指向未定义的指令。

注：

如果在清除寄存器前发生了 1 个以上的故障，那么故障位是累加的。

硬故障状态寄存器

使用硬故障状态寄存器来获取激活硬故障处理器的事件的相关信息。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED2C

访问类型　　读/写清除

复位状态　　0x00000000

HFSR 是一个写-清除寄存器。即向该位写 1 可以将其清除。

硬件故障状态寄存器的位分配如图 8-19 所示。

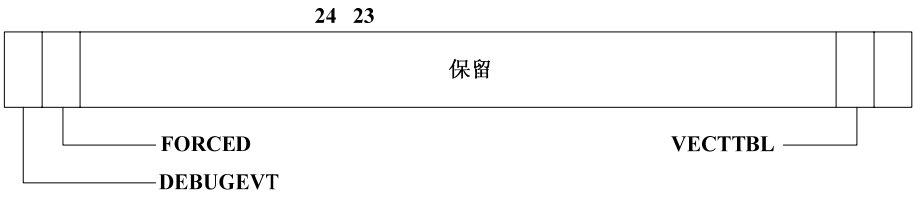


图 8-19　硬故障状态寄存器的位分配

表 8-24 描述了硬故障状态寄存器的各个位。

表 8-24　硬故障状态寄存器的位分配

域	名称	定义
[31]	DEBUGEVT	如果出现与调试有关的故障，那么该位置位。 DEBUGEVT 仅在没有使能“停止调试”的情况下才可能置位。对于监控使能调试，在当前优先级高于监控程序时只有 BKPT 会发生；当停止调试和监控调试都被禁能时，只有没被忽略的调试事件（最小的，BKPT）才会发生。更新调试故障状态寄存器。
[30]	FORCED	硬故障在接收到可配置故障时激活，由于优先级或因为可配置故障被禁能而不能激活。 所以为了判断具体原因，硬故障处理器必须读其他故障状态寄存器
[29:2]	-	保留
[1]	VECTTBL	因为在异常处理（总线故障）过程中读向量表而导致发生故障，此时该位置位。这种情况下通常都是硬故障。返回 PC 指向占先指令。

调试故障状态寄存器

调试故障状态寄存器用于监控：

- 外部调试请求
- 向量捕获
- 数据观察点匹配
- BKPT 指令执行
- 中止请求

当产生多个故障条件时，可以在调试故障状态寄存器中设置多个标志。该寄存器是读/写清除。即可以正常地对它进行读操作。向该位写 1 可以将其清除。

注：

除非捕获到了事件，否则这些位不会置位。即它会引起某些类型的停止。如果停止调试使能，那么这些事件会让处理器停止调试。如果调试禁能而调试监控使能，那么只要优先级允许，它就会开始调用调试监控处理器。如果调试和监控都被禁能，那么其中某些事件是硬故障，且 DBGEVT 位在硬故障状态寄存器中置位，而有些事件没有什么用处。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED30

访问类型　　读/写清除

复位状态　　0x00000000

调试故障状态寄存器的位分配如图 8-20 所示。

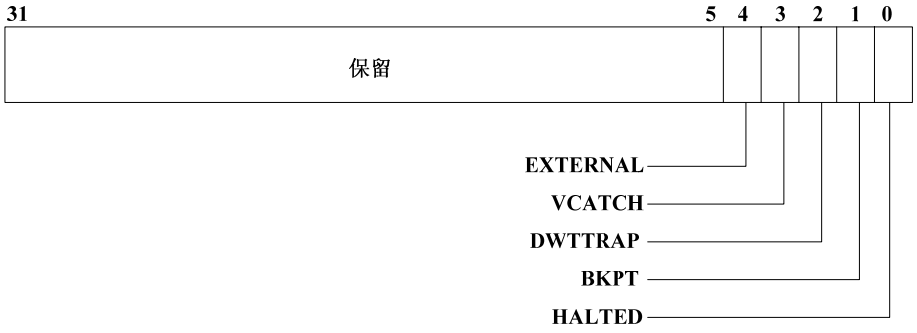


图 8-20　调试故障状态寄存器的位分配

表 8-25 描述了调试故障状态寄存器的各个位。

表 8-25　调试故障状态寄存器的位分配

域	名称	定义
[31:5]	-	保留
[4]	EXTERNAL	外部调试请求标志： 1= **EDBGRQ** 信号有效 0= **EDBGRQ** 信号无效 处理器停在下一个指令边界
[3]	VCATCH	向量捕获标志： 1= 向量捕获发生 0= 没有向量捕获发生 在 VCATCH 标志置位时，将其中一个局部故障状态寄存器中的标志置位，以指示故障类型。
[2]	DWTIRAP	数据观察点与跟踪（DWT）标志： 1= DWT 匹配 0= 没有 DWT 匹配 处理器停在当前指令或下一条指令
[1]	BKPT	BKPT 标志： 1= 执行 BKPT 指令 0= 没有执行 BKPT 指令 BKPT 标志由 flash 修补代码中的 BKPT 指令设置，也可以由正常代码设置。返回 PC 指向包含指令的断点。
[0]	HALTED	中止请求标志： 1= 由 NVIC 发出的中止请求，包括单步调试。处理器在下一条指令被中止 0= 没有中止请求

存储器管理故障地址寄存器

使用存储器管理故障地址寄存器来读取引起存储器管理故障的单元地址。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED34

访问类型　　读/写

复位状态　　不可预测

表 8-26 描述了存储器管理故障地址寄存器的各个位。

表 8-26　存储器管理故障地址寄存器的位功能

域	名称	定义
[31:0]	ADDRESS	Mem Manage 故障的地址域。ADDRESS 是错误的加载或存储操作的数据地址。当发生不对齐访问故障时，该地址是出错的实际地址。因为访问可以分成多个部分，每个都对齐，所以该地址可以是所需大小范围内的任意偏移量。存储器管理故障状态寄存器的标志表示引起故障的原因。见 *“存储器管理故障状态寄存器”*。

总线故障地址寄存器

使用总线故障地址寄存器来读取产生总线故障的单元地址。

寄存器地址、访问类型和复位状态：

地址　　0xE000ED38

访问类型　　读/写

复位状态　　不可预测

表 8-27 描述了总线故障地址寄存器的各个位。

表 8-27　总线故障地址寄存器的位功能

域	名称	定义
[31:0]	ADDRESS	总线故障的地址域。ADDRESS 是装载或保存出错的地址。当发生不对齐访问故障时，即使该地址不是出错的地址，它也是指令请求的地址。总线故障状态寄存器的标志表示引起故障的原因。见 *“总线故障状态寄存器”*。

软件触发中断寄存器

使用软件触发中断寄存器来挂起一个将要触发的中断。

寄存器地址、访问类型和复位状态：

地址　　0xE000EF00

访问类型　　只写

复位状态　　0x00000000

软件触发中断寄存器的位分配如图 8-21 所示。

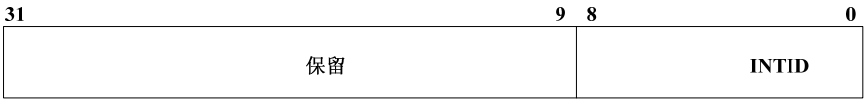


图 8-21　软件触发中断寄存器的位分配

表 8-28 描述了软件触发中断寄存器的各个位。

表 8-28　软件触发中断寄存器的位分配

域	名称	定义
[31:9]	-	保留
[8:0]	INTID	中断 ID 域。写值到 INTID 域和在中断挂起设置寄存器里设置相应的中断位将中断手动挂起所达到的效果相同。

8.3 电平中断与脉冲中断

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。

第9章 存储器保护单元

本章介绍了处理器“*存储器保护单元*”（MPU）。包括以下内容：

- *MPU 概述*
- *MPU 编程器模型*
- *中断和更新 MPU*
- *MPU 访问权限*
- *MPU 异常中止*
- *更新 MPU 区*

9.1 MPU 概述

存储器保护单元（MPU）是用来保护存储器的一个元件。处理器支持标准的 ARMv7“*受保护的存储器系统结构*”（PMSA）模型。MPU 为以下操作提供完整的支持：

- 保护区域
- 重叠保护区区域
- 访问权限
- 将存储器属性输出到系统

MPU 不匹配和许可违犯调用优先级可设定的 MemManage 故障处理器。要了解更多信息，请参考“*存储器管理故障地址寄存器*”。

MPU 可以用于：

- 强制执行特权原则
- 分离程序
- 强制执行访问原则

9.2 MPU 编程器模型

本节介绍了控制 MPU 的寄存器。包括以下内容：

- *MPU 寄存器汇总*
- *MPU 寄存器描述*

9.2.1 MPU 寄存器纵览

表 9-1 汇总了 MPU 寄存器。

表 9-1　MPU 寄存器

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	-

续上表…

寄存器名称	类型	地址	复位值
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	-
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	-

9.2.2　描述 MPU 寄存器

本小节描述了 MPU 寄存器。

MPU 类型寄存器

使用 MPU 类型寄存器查看 MPU 支持的区域数。读取位[15:8]的内容以判断是否存在 MPU。

寄存器地址，访问类型和复位状态：

地址　　0xE000ED90

访问类型　　只读

复位状态　　0x00000800

MPU 类型寄存器的位分配如图 9-1 所示。

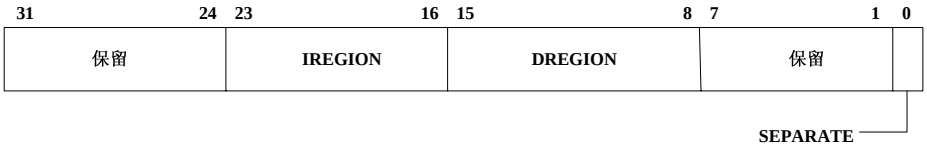


图 9-1　MPU 类型寄存器的位分配

表 9-2 描述了 MPU 类型寄存器的各个位。

表 9-2　MPU 类型寄存器的位分配

域	名称	描述
[31:24]	-	保留
[23:16]	IREGION	因为处理器内核只使用统一的 MPU，所以 IREGION 总是包含 0x00
[15:8]	DREGION	支持的 MPU 区域数。如果实现包含一个表明有 8 个 MPU 区的 MPU，那么 DREGION 包含 0x08，否则包含 0x00。
[7:0]	-	保留
[0]	SEPARATE	因为处理器内核只使用统一的 MPU，所以 SEPARATE 总是为 0

MPU 控制寄存器

MPU 控制寄存器用于：

- 使能 MPU
- 使能缺省的存储器映射（背景区域）
- 在处于硬故障、NMI 和 FAULTMASK 升级处理器时使能 MPU

在使能 MPU 时，为了运行 MPU，除非 PRIVDEFENA 位置位，否则必须至少使能一个存储器映射区。如果 PRIVDEFENA 位置位而没有区域被使能，那么只能运行特权代码。

MPU 禁能时，使用缺省的地址映射，相当于没有 MPU。

MPU 使能时，只有系统分区和向量表装载总是可访问。其他区必须根据区域以及 PRIVDEFENA 是否使能才能决定其是否可访问。

除非 HFNMIENA 置位，否则 MPU 在异常优先级为-1 或-2 时不能被使能。这些优先级仅在处于硬故障、NMI 或当 FAULTMASK 使能时才存在。当 HFNMIENA 位和这两个优先级一起工作时，它用于使能 MPU。

寄存器地址，访问类型和复位状态：

地址	0xE000ED94
访问类型	读/写
复位状态	0x00000000

MPU 控制寄存器的位分配如图 9-2 所示。

图 9-2　MPU 控制寄存器的位分配

表 9-3 描述了 MPU 控制寄存器的各个位。

表 9-3　MPU 控制寄存器的位分配

域	名称	定义
[31:2]	-	保留
[2]	PRIVDEFENA	当使能 MPU 时，该位使能供特权访问使用的缺省存储器映射，将其作为背景区域。背景区域在任意可设置区域前都表现得像区号为 1。设置的任意区域与这个缺省的映射重叠，并将其替换。如果位=0，那么缺省的存储器映射被禁能，且区域错误不会覆盖存储器。 当使能 MPU 和 PRIVDEFENA 时，缺省的存储器映射如第四章“*存储器映射*”所述。这适用于存储器类型、XN、高速缓存和可共享原则。但也仅适用于特权模式（读取和数据访问）。除非已经为其代码和数据建立了区域，否则用户模式代码会弄错。 禁能 MPU 时，缺省的映射对特权模式和用户模式的代码都起作用。不管该使能位是否置位，SN 和 SO 原则总是适用于系统分区。如果 MPU 被禁能，那么就忽略该位。 复位将 PRIVDEFENA 位清零。
[1]	HFNMIENA	在处于硬故障、NMI 和 FAULTMASK 升级处理器时，该位使能 MPU。如果 HFNMIENA=1 且 ENABLE=1，那么在处于这些处理器时 MPU 被使能。如果 HFNMIENA=0，那么在处于这些处理器时 MPU 被禁能，但不管 ENABLE 的值如何，只要 HFNMIENA=1 且 ENABLE=0，那么行为将变得不可预测。 复位将 HFNMIENA 位清零。
[0]	ENABLE	MPU 使能位： 1= 使能 MPU 0= 禁能 MPU 复位将 ENABLE 位清零。

MPU 区号寄存器

MPU 区号寄存器用于选择进行访问的保护区。然后写 MPU 区域基址寄存器或 MPU 属性与大小寄存器以对保护区的特性进行配置。

寄存器地址，访问类型和复位状态：

地址　　0xE000ED98

访问类型　　读/写

复位状态　　不可预测

MPU 区号寄存器的位分配如图 9-3 所示。

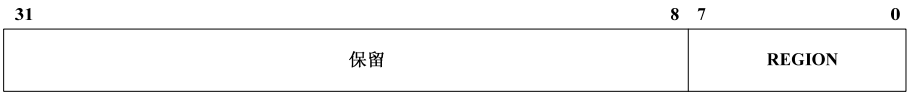


图 9-3　MPU 区号寄存器的位分配

表 9-4 描述了 MPU 区号寄存器的各个位。

表 9-4　MPU 区号寄存器的位分配

域	名称	定义
[31:8]	-	保留
[7:0]	REGION	区域选择域。在使用“区域属性与大小寄存器”和“区域基址寄存器”时选择进行操作的域。首先必须写 REGION，但在对地址 VALID＋REGION 域进行写操作时除外，它覆盖了 REGION。

MPU 区域基址寄存器

MPU 区域基址寄存器用于写区域的基址。区域基址寄存器还含有 REGION 域，如果 VALID 置位，你可以将它用于覆盖 MPU 区号寄存器中的 REGION 域。

区域基址寄存器为区域设置基址，按照大小对齐。那么一个 64KB 大小的区域必须按 64KB 的倍数对齐，例如，0x00010000，0x00020000 等等。

Region 读回的结果总是当前 MPU 的区号。Valid 总是当作 0 读回。将 VALID 和 REGION 分别设成 VALID=1 和 REGION=n 时，区号将变为 n。这是写 MPU 区号寄存器一个最快捷的方法。

如果不是按字访问，那么寄存器将不可预测。

寄存器地址，访问类型和复位状态：

地址　　0xE000ED9C

访问类型　　读/写

复位状态　　不可预测

MPU 区域基址寄存器的位分配如图 9-4 所示。

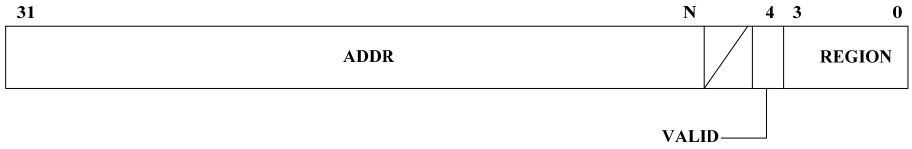


图 9-4　MPU 区域基址寄存器的位分配

表 9-5 描述了 MPU 区域基址寄存器的各个位。

表 9-5　MPU 区域基址寄存器的位分配

域	名称	定义
[31:N]	ADDR	区域基址域。N 的值取决于区域的大小，所以基址是按照大小的偶数倍来对齐的。由 MPU 区域属性与大小寄存器的 SZENABLE 域指定的 2 的乘幂决定了使用的基址位数
[4]	VALID	MPU 区号有效位： 1= MPU 区号寄存器被位 3:0（REGION 值）覆写。 0= MPU 区号寄存器保持不变并被解释。
[3:0]	REGION	MPU 区域覆盖域

MPU 区域属性与大小寄存器

MPU 区域属性与大小寄存器用于控制 MPU 的访问权限。寄存器由两个局部寄存器组成，每个都是半字大小。可以使用单独的长度对这些寄存器进行访问，也可以使用字操作同时对它们进行访问。

子区域禁能位在区域大小为 32 字节、64 字节和 128 字节时是不可预测的。

寄存器地址，访问类型和复位状态：

地址　　0xE000ED9C

访问类型　　读/写

复位状态　　不可预测

MPU 区域属性与大小寄存器的位分配如图 9-5 所示。

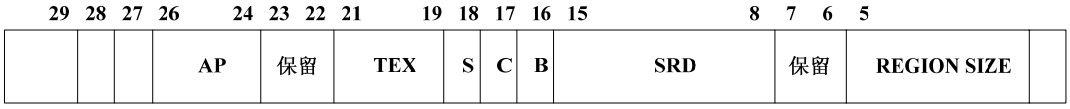


图 9-5　MPU 区域属性与大小寄存器的位分配

表 9-6 描述了 MPU 区域属性与大小寄存器的各个位。要了解更多信息，请参考 *“MPU 访问权限”*。

表 9-6　MPU 区域属性与大小寄存器的位分配

域	名称	定义
[31:29]	-	保留
[28]	XN	指令访问禁止位： 1= 禁止取指 0= 使能取指
[27]	-	保留
[26:24]	AP	数据访问许可域： 值 / 特权许可 / 用户许可 b000 / 不可访问 / 不可访问 b001 / 读/写 / 不可访问 b010 / 读/写 / 只读 b011 / 读/写 / 读/写 b100 / 保留 / 保留 b101 / 只读 / 不可访问 b110 / 只读 / 只读 b111 / 只读 / 只读
[23:22]	-	保留
[21:19]	TEX	类型扩展域
[18]	S	可共享位： 1= 可共享 0= 不可共享

续上表…

域	名称	定义
[17]	C	可高速缓存的位： 1= 可高速缓存 0= 不可高速缓存
[16]	B	可缓冲的位： 1= 可缓冲 0= 不可缓冲
[15:8]	SRD	子区域禁能域。SRD 位置位以禁能相应的子区域。区域被分成 8 个同等大小的子区域。不支持 128 字节及更小的子区域。要了解更多信息，请参考 “*子区域*”。
[7:6]	-	保留
[5:1]	REGION SIZE	MPU 保护区大小域。见表 9-7
[0]	SZENABLE	区域使能位

要了解更多有关访问权限的信息，请参考 *“MPU 访问权限”*。

表 9-7　MPU 保护区域大小域

区域	大小
b00000	保留
b00001	保留
b00010	保留
b00011	保留
b00100	32B
b00101	64B
b00110	128B
b00111	256B
b01000	512B
b01001	1KB
b01010	2KB
b01011	4KB
b01100	8KB
b01101	16KB
b01110	32KB
b01111	64KB
b10000	128KB
b10001	256KB
b10010	512KB
b10011	1MB
b10100	2MB
b10101	4MB
b10110	8MB
b10111	16MB
b11000	32MB

续上表…

区域	大小
b11001	64MB
b11010	128MB
b11011	256MB
b11100	512MB
b11101	1GB
b11110	2GB
b11111	4GB

9.2.3　使用重叠寄存器访问 MPU

用户可以使用寄存器地址重叠来优化 MPU 寄存器的装载速度。有三组 NVIC 重叠寄存器。这些寄存器在“NVIC 寄存器描述”中作了介绍。

别名（alias）使用相同的方式访问寄存器，并且可以让“顺序写（STM）”在 1 到 4 个区之间更新。这在不需要禁能/修改/使能时使用。

此外，因为必须写区号，所以不可以使用这些别名（alias）来读取区域的内容。

以下便是用来更新 4 个区的代码序列的一个实例：

```
; R1 = 4 region pairs from process control block (8 words)
MOV R0, #NVIC_BASE
ADD R0, #MPU_REG_CTRL
LDM R1, [R2-R9] ; 为 4 个区装载区域信息
STM R0, [R2-R9] ; 立即对 4 个区进行全部更新
```

注：

你可以正常使用该序列的 C/C++编译器的 memcpy()函数。但必须对编译器是否使用字传输进行校验。

9.2.4 子区域

使用区域属性与大小寄存器的 8 个子区域禁能（SRD）位将区域分成 8 个基于区域大小且大小相同的单元。这样可以选择禁能一些 1/8 的子区域。最低位影响第一个 1/8 子区域，最高位影响最后的 1/8 子区域。被禁能的子区域可以让范围匹配的任何其他区域重叠来替代。如果没有其他区域将该禁能的子区域重叠，那么使用默认的行为，不匹配——故障。子区域不能和 3 个小长度（32，64 和 128）的区域共用。如果使用了这些子区域，结果将不可预测。

SRD 使用实例

基址相同的两个区域相互重叠。一个区域为 64KB，另外一个区域为 512KB。512KB 区域的底部 64KB 是禁能的，所以可以采用 64KB 的属性。这可以通过将 512KB 区域的 SRD 设置成 b11111110 来实现。

9.3　MPU 访问权限

本小节描述了 MPU 的访问权限。访问权限位 TEX——区域访问控制寄存器（见“*MPU 区域属性与大小寄存器*”）的 TEX、C、B、AP 和 XN 位，控制对相应存储区的访问。如果未经许可就对存储区进行访问，将引发许可故障。

表 9-8 描述了 TEX、C 和 B 编码。

表 9-8　TEX，C，B 编码

TEX	C	B	描述	存储器类型	区域的共享性
b000	0	0	非常有序	非常有序	可共享
b000	0	1	共享器件	器件	可共享
b000	1	0	外部和内部直写，无写分配	正常	S
b000	1	1	外部和内部写回，无写分配	正常	S
b001	0	0	外部和内部不可高速缓存	正常	S
b001	0	1	保留	保留	保留
b001	1	0	已定义的执行		
b001	1	1	外部和内部写回，写和读分配	正常	S
b010	1	X	不共享器件	器件	不能共享
b010	0	1	保留	保留	保留
b010	1	X	保留	保留	保留
b1BB	A	A	高速缓存 **BB**= 外部政策。**AA**= 内部政策	正常	S

注：

表 9-8 中的‘S’是 MPU 区域属性与大小寄存器的 S bit[2]。

表 9-9 描述了存储器属性编码的高速缓存政策。

表 9-9　存储器属性编码的高速缓存政策

存储器属性编码（AA 和 BB）	高速缓存政策
00	非高速缓存
01	写回、写和读分配
10	写通过，无写分配
11	写回、无写分配

表 9-10 描述了 AP 编码。

表 9-10　AP 编码

AP[2:0]	特权许可	用户许可	描述
000	不可访问	不可访问	所有的访问都会产生许可故障
001	读/写	不可访问	只可进行特权访问
010	读/写	只读	在用户模式进行写操作时会产生许可故障
011	读/写	读/写	完全访问
100	不可预测	不可预测	保留
101	只读	不可访问	只能进行特权读操作
110	只读	只读	只能进行特权/用户读操作
111	只读	只读	只能进行特权/用户读操作

表 9-11 描述了 XN 编码。

表 9-11 XN 编码

XN	描述
0	使能所有取指
1	没有使能取指

9.4　MPU 异常中止

要了解有关“MPU 异常中止”的信息，请参考“*存储器管理故障地址寄存器*”。

9.5 更新 MPU 区域

有 3 个包含存储器映射字的寄存器，用于编程 MPU 区域。这些都是局部寄存器，可以被编程和单独访问。即可以移植现存的 ARMv6、ARMv7 和 CP15 代码。使用 **LDRx** 和 **STRx** 操作替代 **MRC** 和 **MCR**。

9.5.1　使用 CP15 等效代码更新 MPU 区域

使用 CP15 等效代码：

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

注：

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。

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0] ;区号
BIC R2,R2, #1 ;禁能
STRH R2,[R0,#8] ;大小与使能
STR R4,[R0,#4] ;地址
STRH R3,[R0,#10] ;属性
ORR R2,#1 ;使能
STRH R2,[R0,#8] ;大小与使能
```

DMB/DSB 不是必需的，因为专用的外设总线是非常有序的存储区。但是，在发生 MPU 效应前 DSB 却是必需的。如上下文转换器的末端。

如果使用分支或调用进入用于对 MPU 区域进行编程的代码，那么 ISB 就是必需的。如果使用一个从异常返回或通过异常来进入代码，那就不需要 ISB 了。

9.5.2 使用两个或三个字来更新 MPU 区域

我们可以使用两个或三个字来直接编程，这取决于分离信息所采取的方式：

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#0] ;区号
STR R2,[R0,#4] ;地址
STR R3,[R0,#8] ;大小，属性
```

可以使用 STM 来优化：

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
MOV R0,#NVIC_BASE
```

```
ADD R0,#MPU_REG_CTRL
STM R0,{R1-R3} ;区号,地址,大小和属性
```

这可以通过预打包信息的 2 个字来完成。即基址寄存器除含有区域有效位之外，还包含区号。这在静态打包数据时非常有用，例如在引导列表或 PCB 中。

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STR R1,[R0,#4] ;地址和区号
STR R2,[R0,#8] ;大小和属性
```

可以使用一个 STM 来优化：

```
; R1 = address and region number in one
; R2 = size and attributes in one
MOV R0,#NVIC_BASE
ADD R0,#MPU_REG_CTRL
STM R0,{R1-R2} ;地址，区号和大小
```

要了解更多有关中断和更新 MPU 的信息，请参考 *“中断和更新 MPU”*。

9.6　中断和更新 MPU

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。此时要注意两个问题：

- 会产生中断，通常会更新 MPU。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。
- 会产生中断，该中断将使用正在更新的区域或者将受到影响。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。如果那些是唯一的 2 个位置，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。

第10章 内核调试

本章描述了如何对处理器进行调试和测试，包含以下内容：

- 关于内核调试
- 内核调试寄存器
- 内核调试访问实例
- 在内核调试中使用应用寄存器

10.1 关于内核调试

内核调试通过内核调试寄存器进行访问。而调试访问这些寄存器时需通过 AHB-AP 端口，详细信息请参考 *AHB 访问端口*。处理器能够在内部的专用外设总线(PPB)上直接访问这些寄存器。

表 10-1 显示了内核调试寄存器。

表 10-1 内核调试寄存器

地址	类型	复位值	描述
0xE000EDF0	读/写	0x00000000[a]	调试停止控制和状态寄存器
0xE000EDF4	只写	-	调试内核寄存器的选择器寄存器
0xE000EDF8	读/写	-	调试内核寄存器的数据寄存器
0xE000EDFC	读/写	0x00000000[b]	调试异常和监控控制寄存器

a　位 5，3，2，1，0 由 **PORESETn** 复位。位[1]也可由 **SYSRESETn** 以及向应用中断和复位控制寄存器的 **VECTRESET** 位写入 1 来复位。

b　位 16，17，18，19 也可由 **SYSRESETn** 以及向应用中断和复位控制寄存器的 **VECTRESET** 位写入 1 来复位。

在调试内核时还用到了调试故障状态寄存器，有关它的详细信息请参考 *调试故障状态寄存器*。

10.1.1 停止模式调试

调试器通过置位调试停止控制和状态寄存器中的 **C_DEBUGEN** 和 **C_HALT** 位来将内核停止。一旦调试停止控制和状态寄存器中的 **S_HALT** 位置位使内核停止时，内核将立即接受。

通过将内核停止，置位 **C_STEP**，然后清零 **C_HALT** 位可单步调试内核。通过置位调试停止控制和状态寄存器中的 **S_HALT** 位，内核将确认单步操作完成并重新停止内核。

10.1.2 退出内核调试

通过清零调试停止和状态寄存器中的 **C_DEBUGEN** 位可以让内核退出调试。

10.2 内核调试寄存器

提供调试操作的寄存器如下：

- 调试停止控制和状态寄存器
- 内核调试寄存器
- 调试内核寄存器的数据寄存器
- 调试异常和监控控制寄存器

10.2.1 调试停止控制和状态寄存器

调试停止控制和状态寄存器(DHCSR)的用途：

- 提供有关处理器状态的信息
- 使能内核调试
- 实现处理器停止和单步操作

DHCSR：

- 32 位读/写寄存器
- 地址为 0xE000EDF0

注：

只能通过系统复位（包括上电）来将 DHCSR 复位。DHCSR 的位 16 在复位时是不可预知的。

图 10-1 显示了寄存器的位分配。

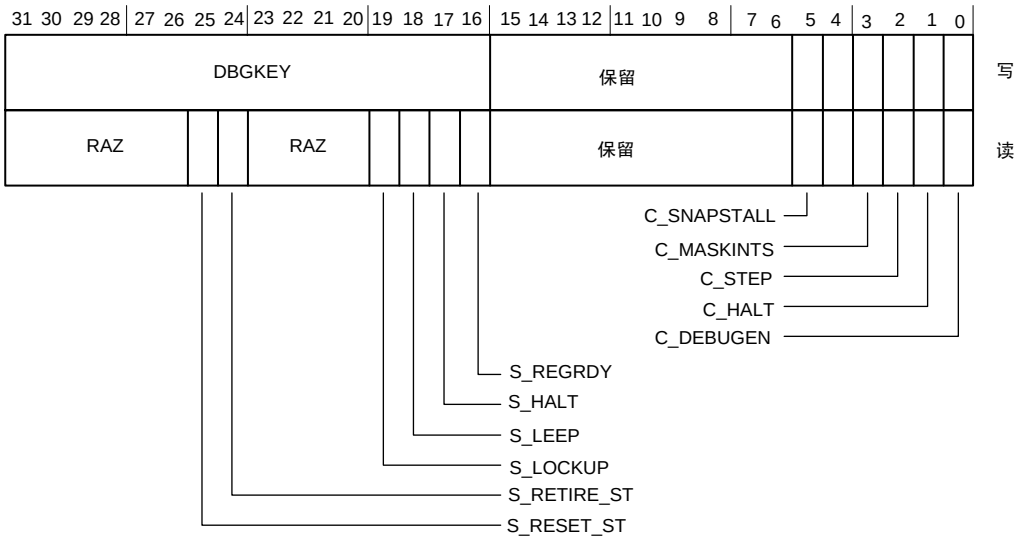


图 10-1 调试停止控制和状态寄存器的格式

表 10-2 显示了调试停止控制和状态寄存器的位功能。

表 10-2 调试停止控制和状态寄存器的位功能

位范围	R/W	区域名称	功能
[31:16]	W	DBGKEY	调试按键。任何时候在对该寄存器执行写操作时必须向位[31:16]写入 0xA05F。读回时作为状态位[25:16]。如果不作为按键写入，则写操作被忽略，不向寄存器写入任何位。
[31:26]	-	-	保留，RAZ
[25]	R	S_RESET_ST	表示自从上一次读该位之后，内核已经复位，或正在复位。这是一个粘着（sticky）位，在读操作时清零。因此，读两次得到 1 然后是 0 表示它已经复位。读两次都得到 1 表示它正在复位（仍然保持在复位状态）。
[24]	R	S_RETIRE_ST	表示自从上一次读操作之后，已经执行完一条指令。这是一个粘着（sticky）位，在读操作时清零。该位用来确定是否内核在加载/存储或取指操作时被暂停。
[23:20]	-	-	保留，RAZ
[19]	R	S_LOCKUP	如果内核正在运行（没有停止）并且处于锁定（lockup）状态，则该位读作 1。
[18]	R	S_SLEEP	表示内核正处于睡眠状态（WFI,WFE 或 SLEEP-ON-EXIT）。必须使用 C_HALT 来获得控制或等待中断来唤醒内核。SLEEP-ON-EXIT 的详细信息请参考表 7-1。
[17]	R	S_HALT	表示内核处于停止状态
[16]		S_REGRDY	
[15]	-	-	保留
[5]	R/W	C_SNAPSTALL	如果内核在加载/存储操作时停止，则停止操作无效，指令被强制完成。该操作使能停止调试来获得对内核的控制。如果出现下列情况，该位只能置位： **C_DEBUGEN**=1 **C_HALT**=1 **S_RETIRE_ST** 由内核读作 0。这表示没有指令超前，可避免误用。 使用该位时，总线状态是不可预知的。 能够使用 **S_RETIRE** 来检测内核在加载/存储操作上停止。
[4]	-	-	保留
[3]	R/W	C_MASKINTS	当在处于调试状态的内核中执行单步操作或运行时屏蔽中断。不影响 NMI，因为它是不可屏蔽的。当处理器停止时（S_HALT==1），该位只能执行修改操作。
[2]	R/W	C_STEP	在调试状态中对内核进行单步操作。当 **C_DEBUGEN**=0 时，该位无效。当处理器停止（S_HALT=＝1）时，该位只能执行修改操作。
[1]	R/W	C_HALT	停止内核。当内核停止时该位自动置位。例如断点。该位在内核复位时清零。如果 **C_DEBUGEN**=1，该位只能执行写操作，否则忽略。当将该位置位时，**C_DEBUGEN** 也必须写入 1（[1:0]为 2’b11）。

续表 10-2

位范围	R/W	区域名称	功能
[0]	R/W	C_DEBUGEN	使能调试。该位只能通过 AHP-AP 写入，不能由内核写入。当通过内核写入时，将忽略操作，不能置位也不能清零。 当内核对 C_HALT 执行写操作将自身停止时，内核必须向该位写入 1。

该寄存器在系统复位时不执行复位操作，在上电时复位。但 **C_HALT** 位在系统复位时一直为 0。

为实现在复位时停止，以下位必须使能：

调试异常和监控控制寄存器的位[0]，**VC_CORERESET**

调试停止控制和状态寄存器的位[0]，**C_DEBUGEN**

注：

以除了字之外的方式写该寄存器时，结果是不可预知的。而读操作时任何尺寸都是可以的，这可用来避免或故意修改 sticky 位。

10.2.2 调试内核选择寄存器

调试内核选择寄存器（DCSR）用来选择进行数据输入输出操作的处理器寄存器。

DCSR：

- 17 位只写寄存器
- 地址为 0xE000EDF4

图 10-2 显示了该寄存器的位分配。

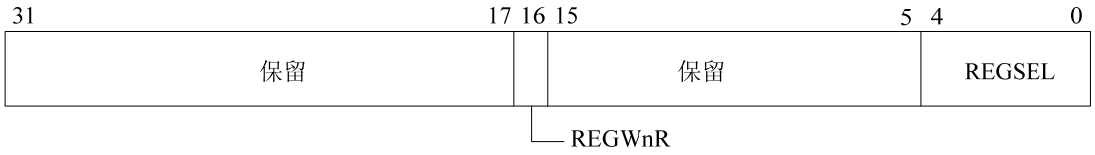


图 10-2 调试内核选择寄存器的格式

表 10-3 显示了调试内核选择寄存器的位功能。

表 10-3 调试内核选择寄存器

位范围	R/W	区域名称	功能
[31:17]	-	-	保留
[16]	W	REGWnR	写操作为 1 读操作为 0
[15:5]	-	-	-

续表 10-3

位范围	R/W	区域名称	功能
[4:0]	W	REGSEL	0b00000=R0 0b00001=R1 … 0b01111=R15 0b10000=xPSR/Flags 0b10001=MSP(主堆栈) 0b10010=PSP(进程堆栈) 0b10011=RAZ/WI 0b10100=CONTROL/FAULTMASK/BASEPRI/PRIMASK(组成字的 4 个字节，CONTROL 为 MSB(31:24))。 0b1xxxx=保留

这是一个只写寄存器，用来产生与内核的握手机制，实现与调试内核寄存器的数据寄存器以及所选的寄存器进行数据传输。在内核处理完成之前，DHSCR 的位 16，即 **S_REGRDY** 一直为 0。

注：

- 以除了字以外的方式写该寄存器时，结果是不可预知的。
- PSR 寄存器在此情况下是可完全访问的，然而在使用 MRS 指令执行读操作时有些位读作 0。
- 所有位都可写入，但当执行过程恢复时，有些位的组合会引起错误。
- 可以向 IT 写入，其操作就好像是在 IT 模块中。
- 可以向 ICI 写入，虽然在异常返回时，写入的无效的值或在 LDM/STM 上不使用的值会引起故障。将 ICI 的值变为 0 可启动下面的 LDM/STM 操作，不继续。

10.2.3 调试内核寄存器的数据寄存器

调试内核寄存器的数据寄存器（DCRDR）用来保存处理器对寄存器执行读和写操作时的数据。

DCRDR：

- 32 位读/写寄存器
- 地址为 0xE000EDF8

该寄存器中包含写入寄存器的数据，被写入的寄存器由调试寄存器的选择寄存器选择。

当处理器接收到来自调试内核寄存器的选择器的请求时，处理器使用常规的单次加载-存储操作读或写该寄存器。

如果没有执行内核寄存器的传输，则基于软件的调试监控程序能够将这个寄存器用在没有停止的调试状态中进行通信。例如，OS RSD 和 Real View 监控程序。这样，能够使用标志和位来确认状态以及指示是否已经接收到命令，已经应用，或接收并应用。

10.2.4 调试异常和监控控制寄存器

调试异常和监控控制寄存器（DEMCR）的用途：

- 捕获向量。当一个指定的向量被提交用于执行时，该寄存器可用来产生调试入口。
- 调试监控控制

DEMCR：

- 32 位读/写寄存器
- 地址为 0xE000EDFC

图 10-2 显示了该寄存器的位分配。

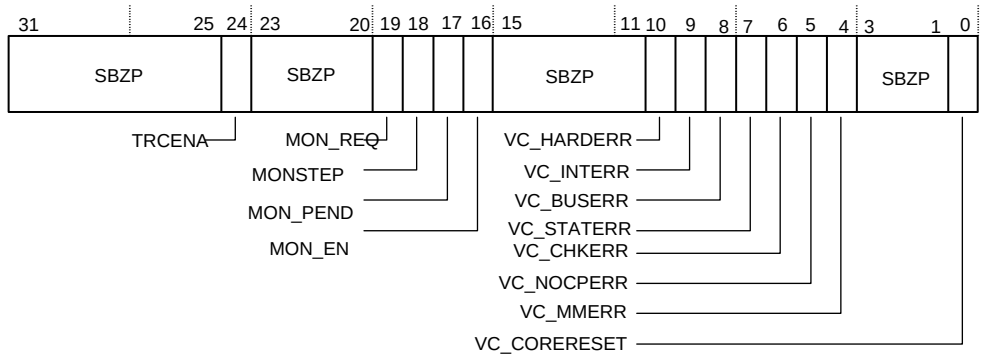


图 10-3 调试异常和监控控制寄存器的格式

表 10-4 显示了调试异常和监控控制寄存器的位功能。

表 10-4 调试异常和监控控制寄存器的位功能

位	R/W	名称	功能
[31:25]	-	-	保留，SBZP
[24]	R/W	TRCENA	该位必须为 1，使能跟踪和调试模块的使用： ● 数据观察点和跟踪（DWT） ● 指令跟踪宏单元（ITM） ● 嵌入式跟踪宏单元（ETM） ● 跟踪端口的接口单元（TPIU） 在没有使用跟踪时，该位使能对功率使用的控制。它能够由应用程序或调试器使能，供 ITM 使用。
[23:20]	-	-	保留，SBZP
[19]	R/W	MON_REQ[a]	该位使能监控程序来确认如何唤醒： 1：由 MON_PEND 唤醒 0：由调试异常唤醒
[18]	R/W	MONSTEP[a]	当 **MON_EN**=1 时，使用该位来单步调试内核。当 **MON_EN**=0 时，该位忽略，它等同于 **C_STEP**。根据监控程序的优先级和 PRIMASK，FAULTMSK，BASEPRI 的设置，中断只能单步调试。

续表 10-4

位	R/W	名称	功能
[17]	R/W	MON_PEND[a]	挂起监控程序，以便在优先级允许时将其激活。能够使用该位通过 AHB-AP 端口唤醒监控程序。在监控调试时它等同于 **C_HALT**。 该寄存器在系统复位时不执行复位，只在上电时复位。调试监控必须在复位处理程序中或后面由软件使能，或通过 DAP 使能。
[16]	R/W	MON_EN[a]	使能调试监控，在使能时，系统处理器优先级寄存器控制其优先级。如果禁止，则所有的调试事件都变为硬故障，调试停止控制和状态寄存器中的 **C_DEBUGEN** 覆盖了该位。 向量捕获操作是半同步的。当事件匹配时，产生停止请求。处理器只能在指令边界上停止，因此它必须等待直到下一个指令边界。最终，它在异常处理程序的第 1 条指令上停止。如果已经触发了向量捕获，则存在两个特殊的情况： ● 如果在向量捕获，读向量或压栈错误时出现故障，则由于向量错误或压栈，对应的故障处理程序停止。 ● 如果在捕获向量时出现迟来中断，则不捕获。即，支持迟来中断优化的实现在此情况下必须能够抑制向量捕获。
[15:11]	-	-	保留，SBZP
[10]	R/W	VC_HARDERR[b]	硬故障上的调试陷阱（见 8.3 节）
[9]	R/W	VC_INTERR[b]	中断/异常程序错误上的调试陷阱（见 8.3）它们是 BUSERR 或 HARDERR 之前的其它故障和捕获到的异常的子集。
[8]	R/W	VC_BUSERR[b]	常规总线错误上的调试陷阱
[7]	R/W	VC_STATERR[b]	使用故障状态错误上的调试陷阱。
[6]	R/W	VC_CHKERR[b]	使能来检验错误的使用故障上的调试陷阱。
[5]	R/W	VC_NOCPERR[b]	在使用故障访问不存在或被标记为在 CAR 寄存器中不存在的协处理器时的调试陷阱
[4]	R/W	VC_MMERR[b]	存储器管理故障上的调试陷阱（见 8.3）
[3:1]	-	-	保留，SBZP
[0]	R/W	VC_CORERESET[a]	捕获复位向量。如果出现内核复位则停止运行。

a.　该位在内核复位时清零。

b.　只有当 C_DEBUGEN=1 时是可用的。

该寄存器用来管理调试状态下的异常行为。

向量捕获只能在调试状态中进行。该寄存器的高半字用于监控控制，低半字用于支持出错的异常。

该寄存器在系统复位时不执行复位。

该寄存器在上电时复位。在内核复位时，位[19:16]始终为零。调试监控程序在复位处理程序中或随后由软件使能，或由 AHB-AP 端口使能。

向量捕获操作是半同步的。当事件匹配时，产生停止请求。处理器只能在指令边界上停止，因此它必须等待直到下一个指令边界。最终，它在异常处理程序的第 1 条指令上停止。如果已经触发了向量捕获，则存在两个特殊的情况：

1. 如果在向量捕获，读向量或压栈错误时出现故障，则由于向量错误或压栈，对应的故障处理程序停止。
2. 如果在捕获向量时出现迟来中断，则不捕获。即，支持迟来中断优化的实现在此情况下必须能够抑制向量捕获。

10.3 内核调试访问实例

如果您想停止处理器并向其中一个寄存器执行写操作，则按以下顺序执行：

1. 向调试停止控制和状态寄存器中写入 0xA05F0003，该操作使能调试并停止内核。
2. 等待调试停止和状态寄存器中的 S_HALT 位置位。如果置位表示内核停止。
3. 向调试内核寄存器的数据寄存器写入您想写入的值。
4. 向调试内核寄存器的选择器寄存器写入您想写入的寄存器编号。

10.4 在内核调试中使用应用寄存器

您也可以将应用寄存器用于状态访问以及在系统上实现改变。

如果您想将应用寄存器用于内核调试，则要知道：

- 如果 AHB-AP 和应用程序正在修改这些寄存器，则就存在读-修改-写的问题。
- 对于 PENDSET 和 PENDCLR 这样的写寄存器，它们不能先读，因此也存在读-修改-写的问题。
- 对于包含优先级的寄存器和其它读-写寄存器，在执行读-修改-写操作时，寄存器能够在读和写之间改变。在某些情况下，寄存器使能字节访问来缓解这个问题。处理器正在运行时，调试器必须意识到这些问题。

表 10-5 显示了在内核调试中非常有用的应用寄存器和寄存器位。应用寄存器的完整列表见 *ARMv7-M 架构参考手册*。

表 10-5 在内核调试中使用的应用寄存器

寄存器	在内核调试中使用的位或区域
中断控制状态	ISRPREEMPT ISRPENDING VECTPENDING
向量表偏移	找出向量表
应用中断/复位控制	VECTCLRACTIVE ENDIANESS
配置控制	DIV_0_TRP UNALIGN_TRP
系统处理控制和状态	ACTIVE PENDED

第11章 系统调试

本章描述处理器的系统调试，包含以下内容：

- 关于系统调试
- 系统调试访问
- 系统调试的编程模型
- Flash 修补和断点
- 数据观察点和跟踪
- 指令跟踪宏单元
- AHB 访问端口

11.1 关于系统调试

Cortex-M3 处理器包含几个系统调试组件，可实现：

- 低成本调试
- 跟踪和 profiling
- 断点
- 观察点
- 代码修补

包含的系统调试组件有：

- Flash 修补和断点（FPB）单元，实现断点和代码修补。
- 数据观察点和触发（DWT）单元，实现观察点，触发资源和系统 profiling。
- 指令跟踪宏单元（ITM），用于支持 printf 类型调试的应用导向（application-driven）的跟踪源。
- 嵌入式跟踪宏单元（ETM），用于指令跟踪。各个处理器版本可以支持或不支持 ETM。
- 所有调试组件都在内部专用外设总线（PPB）上，能够使用特权代码来访问。

注：

内核调试的描述见第 10 章*内核调试*。

11.2 系统调试访问

调试控制和数据访问通过 AHB-AP 接口实现。该接口由 SW-DP 或 JTAG-DP 组件驱动。有关 SW-DP 和 JTAG-DP 组件的详细信息请参考第 12 章 *调试端口*。在执行系统调试访问时，可对以下总线进行访问：

- 内部专用外设总线（PPB）。调试器能够通过该总线访问以下 Cortex-M3 组件：
  - 嵌套向量中断控制器（NVIC）。通过 NVIC 可实现对处理器的调试访问。详细信息见第 10 章 *内核调试*。
  - 数据观察点和跟踪（DWT）
  - Flash 修补和断点（FPB）单元
  - 指令跟踪宏单元（ITM）
  - 存储器保护单元（MPU）
- 外部专用外设总线。通过该总线能够调试访问以下组件：
  - 嵌入式跟踪宏单元（ETM）。它是一个只支持指令跟踪的低成本跟踪宏单元。详细信息见第 15 章 *嵌入式跟踪宏单元*。
  - 跟踪端口接口单元（TPIU）。该组件作为 Cortex-M3 跟踪数据（来自 ITM,ETM）和片外跟踪端口分析仪之间的桥接。详细信息见 13 章 *跟踪端口接口单元（TPIU）*。
  - ROM 表
- DCode 总线。调试器能够通过该总线访问位于代码空间的存储器。
- 系统总线。通过该总线能够访问位于系统总线空间的存储器和外设。

图 11-1 显示了系统调试访问的结构并显示了如何使用 AHB-AP 对每个系统组件和外部总线进行访问。

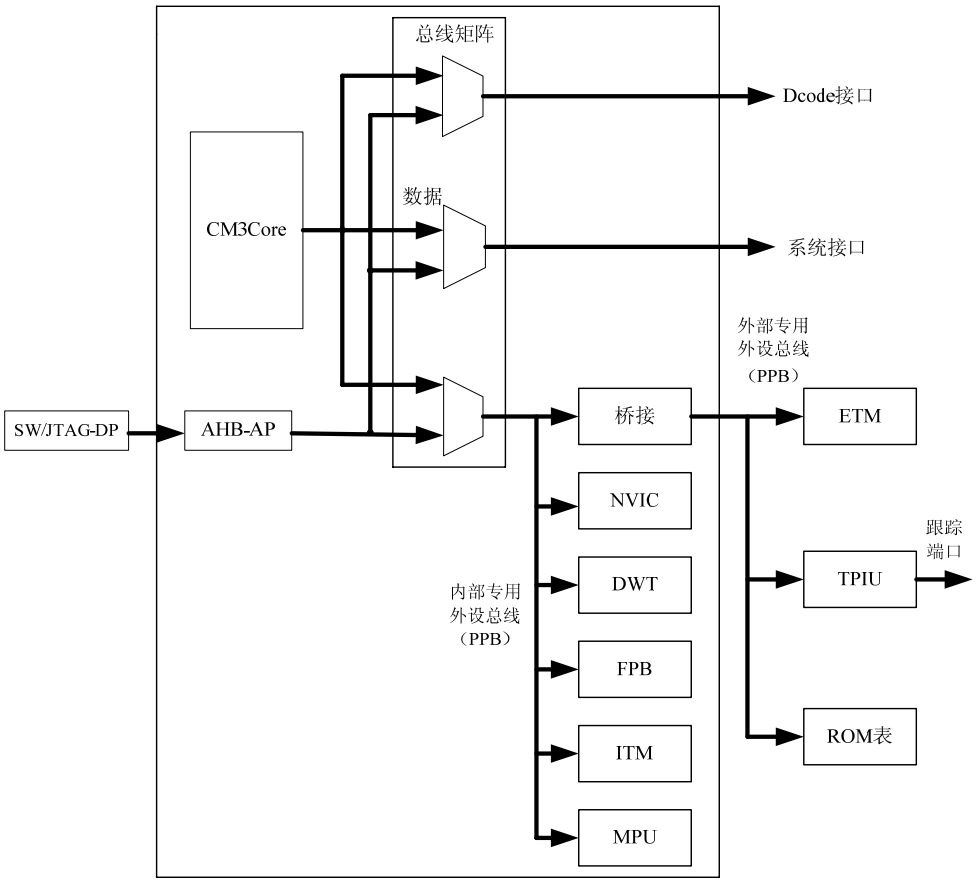


图 11-1 系统调试访问模块框图

11.3 系统调试的编程模型

这节列出并描述了用于所有系统调试组件的调试寄存器，包括：

- Flash 修补和断点
- 数据观察点和跟踪
- 指令跟踪宏单元
- AHB 访问端口

注：

- 内核调试寄存器的描述见*内核调试寄存器*。
- JTAG-DP 和 SW-DP 寄存器的描述见第 12 章*调试端口*。
- TPIU 的描述见第 13 章*跟踪端口接口单元*。

11.4 Flash 修补和断点

Flash 修补和断点单元（FPB）：

- 实现硬件断点
- 从代码空间到系统空间的代码和数据修补

FPB 单元包括：

- 2 个 literal 比较器，与来自代码空间的 literal 加载进行比较并重新映射到系统空间的对应区域。
- 6 个指令比较器。与来自代码空间的指令取指进行比较并重新映射到系统空间的对应区域。另外，比较器可单独配置为在匹配时向处理器内核返回一个断点指令（BKPT），因此可提供硬件断点功能。

FPB 包含一个全局使能，还包含 8 个比较器的单独使能。如果入口的比较匹配时，地址被重映射为重映射寄存器中设置的地址加上与产生匹配的比较器对应的偏移量，或重映射为 BKPT 指令（如果该特性使能）。比较操作在空闲时发生，但比较结果太晚而不能停止原来的来自代码空间的指令取指或 literal 加载。处理器忽略这个处理，而只使用被重映射的处理。

如果存在 MPU，则对原来的地址而不是重映射的地址执行 MPU 锁定。

注：

非对齐的 literal 访问没有重映射。在此情况下执行原来对 DCode 总线的访问。

注：

专用加载对于 FPB 来说是不可预知的。地址被重映射但访问不作为专用加载来执行。

注：

对 bit-band 别名的重映射直接访问别名地址，不重映射为 bit-band 区域。

11.4.1 FPB 的编程模型

表 11-1 列出了 Flash 修补寄存器。

表 11-1 Flash 修补寄存器

名称	类型	地址	描述
FP_CTRL	读/写	0xE0002000	Flash 修补控制寄存器
FP_REMAP	读/写	0xE0002004	Flash 修补重映射寄存器
FP_COMP0	读/写	0xE0002008	Flash 修补比较器寄存器
FP_COMP1	读/写	0xE000200C	Flash 修补比较器寄存器
FP_COMP2	读/写	0xE0002010	Flash 修补比较器寄存器
FP_COMP3	读/写	0xE0002014	Flash 修补比较器寄存器
FP_COMP4	读/写	0xE0002018	Flash 修补比较器寄存器
FP_COMP5	读/写	0xE000201C	Flash 修补比较器寄存器
FP_COMP6	读/写	0xE0002020	Flash 修补比较器寄存器
FP_COMP7	读/写	0xE0002024	Flash 修补比较器寄存器
PERIPID4	只读	0xE0002FD0	值为 0x04

续表 11-1

名称	类型	地址	描述
PERIPID5	只读	0xE0002FD4	值为 0x00
PERIPID6	只读	0xE0002FD8	值为 0x00
PERIPID7	只读	0xE0002FDC	值为 0x00
PERIPID0	只读	0xE0002FE0	值为 0x30
PERIPID1	只读	0xE0002FE4	值为 0xB0
PERIPID0	只读	0xE0002FE8	值为 0x0B
PERIPID1	只读	0xE0002FEC	值为 0x00
PCELLID0	只读	0xE0002FF0	值为 0x0D
PCELLID1	只读	0xE0002FF4	值为 0xE0
PCELLID2	只读	0xE0002FF8	值为 0x05
PCELLID3	只读	0xE0002FFC	值为 0xB1

Flash 修补控制寄存器

使用 Flash 修补控制寄存器来使能 Flash 修补模块。

寄存器地址，访问类型和复位状态：

地址　　0xE00020

访问　　读/写

复位状态　　位[0](ENABLE)复位值为 1’b0。

图 11-2 显示了 Flash 修补控制寄存器的位分配。

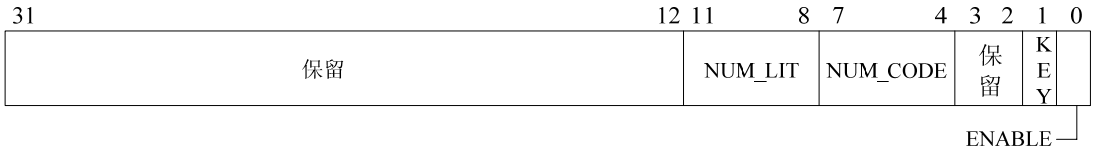


图 11-2 Flash 修补控制寄存器的位分配。

表 11-2 描述了 Flash 修补控制寄存器的位分配。

表 11-2 Flash 修补控制寄存器的位分配。

位	名称	定义
[31:12]	-	保留，读作 0，写操作时无效。
[11:8]	NUM_LIT	literal 插槽的数目。该只读区域的值为 b0010，表示有 2 个 literal 插槽。
[7:4]	NUM_CODE	代码插槽的数目。该只读区域的值为 b0110，表示有 6 个代码插槽。
[3:2]	-	保留
[1]	KEY	按键区。该只写位必须写入 1，才能对 Flash 修补控制寄存器执行写操作。

续表 11-2

位	名称	定义
[0]	ENABLE	Flash 修补单元的使能位： 1：Flash 修补单元使能 0：Flash 修补单元禁止 复位时将清零 ENABLE 位

Flash 修补重映射寄存器

Flash 修补重映射寄存器用来提供系统空间中的单元，匹配的地址重映射到该单元中。REMAP 地址是 8 字对齐的，8 个字分别分配给 8 个 FPB 比较器。

比较匹配时重映射为：

{3’b001，REMAP，COMP[2:0]，HADDR[1:0]}

这里：

- 3’b001 将重映射的访问硬连接到系统空间
- REMAP 为 24 位、8 字对齐的重映射地址
- COMP 为匹配比较器，见表 11-3。

表 11-3 COMP 映射

COMP[2:0]	比较器	描述
000	FP_COMP0	指令比较器
001	FP_COMP1	指令比较器
010	FP_COMP2	指令比较器
011	FP_COMP3	指令比较器
100	FP_COMP4	指令比较器
101	FP_COMP5	指令比较器
110	FP_COMP6	literal 比较器
111	FP_COMP7	literal 比较器

- HADDR[1:0]为原始地址的低两位。HADDR[1:0]始终为 2’b00，用于指令取指。

寄存器地址，访问类型和复位状态：

地址　　0xE0002004

访问　　读/写

复位状态　　该寄存器不复位

图 11-3 显示了 Flash 修补重映射寄存器的位分配。

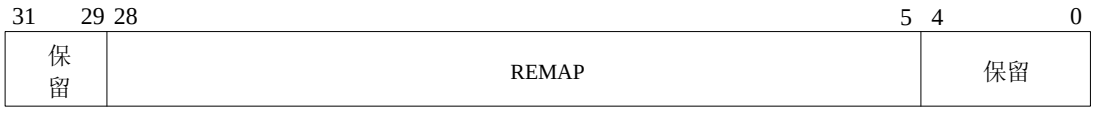


图 11-3 Flash 修补重映射寄存器的位分配

表 11-4 描述了 Flash 修补重映射寄存器的位分配。

表 11-4 Flash 修补重映射寄存器的位分配

位	名称	定义
[31:29]	-	保留，读作 b001，将重映射硬连接到系统空间。
[28:5]	REMAP	重映射基地址
[4:0]	-	保留，读作 0，写操作时无效。

Flash 修补比较器寄存器

Flash 修补比较器寄存器用来存储与 PC 地址进行比较的值。

寄存器地址，访问类型和复位状态：

访问　　读/写

地址　　0xE0002008， 0xE000200C， 0xE0002010， 0xE0002014， 0xE0002018, 0xE000201C，0xE0002020，0xE0002024

复位状态　　位[0](ENABLE)复位值为 1’b0。

图 11-4 显示了 Flash 修补比较器寄存器的位分配。

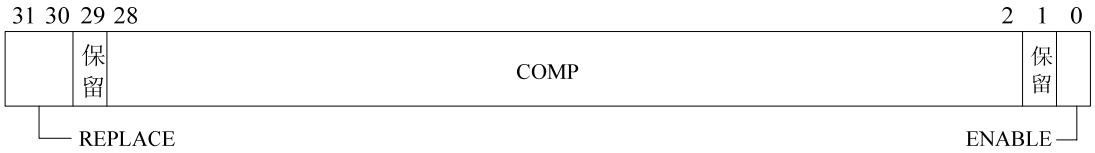


图 11-4 Flash 修补比较器寄存器的位分配

表 11-5 描述了 Flash 修补比较器寄存器的位分配。

表 11-5 Flash 修补比较器寄存器的位分配

位	名称	定义
[31:30]	REPLACE	这两个位用来选择当 COMP 地址匹配时产生的动作： b00：重映射到重映射地址。见 FP_REMAP b01：在低半字上设置 BKPT，高半字不受影响。 b10：在高半字上设置 BKPT，低半字不受影响。 b11：在高低半字上设置 BKPT。 b00 之外的设置只在用于指令比较器时是有效的。literal 比较器忽略非 b00 的设置。 只在 b00 设置时才产生地址重映射。
[29]	-	保留
[28:2]	COMP	比较地址
[1]	-	保留，读作 0，写操作时忽略。
[0]	ENABLE	该位为 1 时使能 Flash 修补比较器寄存器 *n* 的比较和重映射，为 0 时禁止。 FP_CTRL 的 ENABLE 位也必须置位来使能比较。 复位时清零 ENABLE 位。

11.5 数据观察点和跟踪

数据观察点和跟踪（DWT）单元执行以下调试功能：

- DWT 包含 4 个比较器，每个比较器都可配置为硬件观察点，ETM 触发，PC 采样器事件触发，或数据地址采样器事件触发。第一个比较器 DWT_COMP0 还可与时钟周期计数器（CYCCNT）进行比较。
- DWT 还含有几个计数器，可对以下各项进行计数：
  - 时钟周期（CYCCNT）
  - 折迭（folded）指令
  - LSU 操作
  - 睡眠周期
  - CPI（除第 1 个周期之外的所有指令周期）
  - 中断开销（interrupt overhead）

注：

每次计数器溢出时，发出一个事件。

- DWT 能够配置为以定义的间隔发出 PC 采样，并发出中断事件信息。

11.5.1 DWT 寄存器总结及描述

表 11-6 列出了 DWT 寄存器。

表 11-6 DWT 寄存器

名称	类型	地址	复位值	描述
DWT_CTRL	读/写	0xE0001000	0x00000000	见 DWT 控制寄存器
DWT_CYCCNT	读/写	0xE0001004	0x00000000	见 DWT 当前 PC 采样器周期计数寄存器
DWT_CPICNT	读/写	0xE0001008	-	见 DWT CPI 计数寄存器
DWT_EXCCNT	读/写	0xE000100C	-	见 DWT 异常开销计数寄存器
DWT_SLEEPCNT	读/写	0xE0001010	-	见 DWT 睡眠计数寄存器
DWT_LSUCNT	读/写	0xE0001014	-	见 DWT LSU 计数寄存器
DWT_FOLDCNT	读/写	0xE0001018	-	见 DWT 折迭（folded）计数寄存器
DWT_COMP0	读/写	0xE0001020	-	见 DWT 比较器寄存器
DWT_MASK0	读/写	0xE0001024	-	见 DWT 屏蔽寄存器 0-3
DWT_FUNCTION0	读/写	0xE0001028	0x00000000	见 DWT 功能寄存器 0-3
DWT_COMP1	读/写	0xE0001030	-	见 DWT 比较器寄存器
DWT_MASK1	读/写	0xE0001034	-	见 DWT 屏蔽寄存器 0-3
DWT_FUNCTION1	读/写	0xE0001038	0x00000000	见 DWT 功能寄存器 0-3
DWT_COMP2	读/写	0xE0001040	-	见 DWT 比较器寄存器
DWT_MASK2	读/写	0xE0001044	-	见 DWT 屏蔽寄存器 0-3
DWT_FUNCTION2	读/写	0xE0001048	0x00000000	见 DWT 功能寄存器 0-3

续表 11-6

名称	类型	地址	复位值	描述
DWT_COMP3	读/写	0xE0001050	-	见 DWT 比较器寄存器
DWT_MASK3	读/写	0xE0001054	-	见 DWT 屏蔽寄存器 0-3
DWT_FUNCTION3	读/写	0xE0001058	0x00000000	见 DWT 功能寄存器 0-3
PERIPHID4	只读	0xE0001FD0	0x04	值为 0x04
PERIPHID5	只读	0xE0001FD4	0x00	值为 0x00
PERIPHID6	只读	0xE0001FD8	0x00	值为 0x00
PERIPHID7	只读	0xE0001FDC	0x00	值为 0x00
PERIPHID0	只读	0xE0001FE0	0x02	值为 0x02
PERIPHID1	只读	0xE0001FE4	0xB0	值为 0xB0
PERIPHID2	只读	0xE0001FE8	0x0B	值为 0x0B
PERIPHID3	只读	0xE0001FEC	0x00	值为 0x00
PCELLID0	只读	0xE0001FF0	0x0D	值为 0x0D
PCELLID1	只读	0xE0001FF4	0xE0	值为 0xE0
PCELLID2	只读	0xE0001FF8	0x05	值为 0x05
PCELLID3	只读	0xE0001FFC	0xB1	值为 0xB1

DWT 控制寄存器

DWT 控制寄存器用来使能 DWT 模块。

寄存器的地址，访问类型和复位状态：

地址　　0xE0001000

访问　　读/写

复位状态　　0x40000000

图 11-5 显示了 DWT 控制寄存器的位分配。

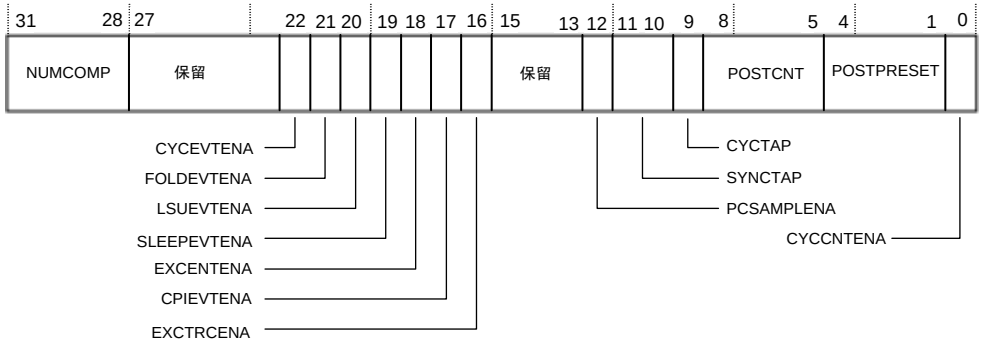


图 11-5 DWT 控制寄存器的位分配

表 11-7 描述了 DWT 控制寄存器的位分配。

表 11-7 DWT 控制寄存器的位分配

位	名称	定义
[31:28]	NUMCOMP	比较器的数目。该只读区域的值为 b0100，表示有 4 个比较器。
[27:23]	-	保留，读作 0，写操作无效
[22]	CYCEVTEN	使能周期计数事件，当 POSTCNT 计数器触发周期计数时，发出一个事件，详细信息见 CYCTAP(位[9])和 POSTPRESET(位[4:1])。
[21]	FOLDEVTENA	使能折迭（folded）指令计数事件。当 DWT_FOLDCNT 溢出（折迭（folded）指令的每 256 个周期）时，发出一个事件。折迭（folded）指令在执行时甚至不需要一个周期（例如 IT 指令，它属于折迭（folded）指令，不需要用完一个周期。） 1：折迭（folded）指令计数事件使能 0：折迭（folded）指令计数事件禁止 复位时 FOLDEVTENA 位清零。
[20]	LSUEVTENA	使能 LSU 计数事件。当 DWT_LSUCNT 溢出时（LSU 操作的每 256 个周期），发出一个事件。LSU 计数包括指令的初始周期之后的所有 LSU 开销。 1：LSU 计数事件使能 0：LSU 计数事件禁止 复位时 LSUEVTENA 位清零。
[19]	SLEEPEVTENA	使能睡眠计数事件。当 DWT_SLEEPCNT 溢出时（处理器处于睡眠状态的每 256 个周期），发出一个事件。 1：睡眠计数事件使能 0：睡眠计数事件禁止 复位时 SLEEPEVTENA 位清零。
[18]	EXCEVTENA	使能中断开销事件。当 DWT_EXCCNT 溢出（中断开销的每 256 个周期），发出一个事件。 1：中断开销事件使能 0：中断开销事件禁止。 复位时 EXCEVTENA 位清零。
[17]	CPIEVTENA	使能 CPI 计数事件。当 DWT_CPICNT 溢出时（多周期指令的每 256 个周期），发出一个事件。 1：CPI 计数器事件使能 0：CPI 计数器事件禁止 复位时 CPIEVTENA 位清零。
[16]	EXCTRCENA	使能中断事件跟踪： 1：中断事件跟踪使能 0：中断事件跟踪禁止 复位时 EXCEVTENA 位清零。
[15:13]	-	保留

续表 11-7

位	名称	定义
[12]	PCSAMPLEENA	使能 PC 采样计数事件。当 POSTCNT 计数器触发 PC 采样时，发出一个 PC 采样事件。详细信息见 CYCTAP(位[9])和 POSTPRESET(位[4:1])。该位使能时将使 CYCEVTENA(位[20])无效。 1：PC 采样事件使能 0：PC 采样事件禁止 复位时 PCSAMPLENA 位清零。
[11:10]	SYNCTAP	用来向 ITM SYNCEN 控制输入一个同步脉冲。这里所选的值通过在 DWT_CYCCNT 寄存器上选择一个“节拍”来选择速率（大约为 1/s 或更小）。为使用同步（heartbeat 与热连接（hot-connect）同步），CYCCNTENA 必须设为 1，SYNCTAP 必须设置为下列值中的一个，SYNCEN 必须设为 1。 0b00 禁止。无同步计数 0b01 在 CYCCNT 位 24 的节拍 0b10 在 CYCCNT 位 26 的节拍 0b11 在 CYCCNT 位 28 的节拍
[9]	CYCTAP	在 DWT_CYCCNT 寄存器上选择一个节拍。可以是位[6]和位[10]。CYCTAP=0 时选择位[6]作为节拍，CYCTAP=1 时选择位[10]作为节拍。当 CYCCNT 寄存器中所选的位从 0 变为 1 或从 1 变为 0 时，它向 POSTCNT(位[8:5])后标量计数器发出信息。该计数器递减计数。当后标量为 0 时，位的改变将触发 PC 采样或 CYCEVTCNT 事件。
[8:5]	POSTCNT	CYCTAP 的后标量计数器。当所选的节拍位从 0 变为 1 或从 1 变为 0 时，后标量计数器在不为 0 时递减计数。如果为 0，则触发 PCSAMPLENA 或 CYCEVTENA 事件。它还重装来自 POSTPRESET(位[4:1])的值。
[4:1]	POSTPRESET	POSTCNT(位[8:5])后标量计数器的重装值。如果该值为 0，事件在每个节拍改变时触发（2 次幂，例如 1<<6 或 1<<10）。如果该区域的值非零，则将形成一个递减的值（每次计数器到达 0 时该值重装入 POSTCNT）。例如，该计数器中的值为 1 表示每隔一个节拍改变时，形成一个事件。
[0]	CYCCNTENA	使能 DWT_CYCCNT 计数器。如果不使能，则计数器不执行计数操作，因此不会产生 PC 采样或 CYCCNTENA 事件。在正常使用时，CYCCNT 计数器应由调试器初始化为 0。

注：

在 DWT 能够使用之前，调试异常和监控控制寄存器的 TRCENA 位必须置位。见*调试异常和监控控制寄存器*。

注：

DWT 从 ITM 中独立地使能。如果使能 DWT 来发出事件，则 ITM 也必须使能。

DWT 当前 PC 采样器周期计数寄存器

该寄存器用来对内核周期进行计数。计数值用来测量消耗的执行时间。

寄存器的地址，访问类型和复位状态：

地址　　0xE0001004

访问　　只读

复位状态　　0x00000000

表 11-8 描述了 DWT 当前 PC 采样器周期计数寄存器的位分配。

表 11-8 DWT 当前 PC 采样器周期计数寄存器的位分配

位	名称	定义
[31:0]	CYCCNT	当前 PC 采样器周期计数器的计数值。使能时，该计数器对内核周期进行计数，内核停止时除外。 DWT_CYCCNT 是一个自由运行的计数器，递增计数。在溢出时返回到 0。在首次使能时，调试器应将它初始化为零。

这是一个自由运行的计数器，有 3 个功能：

- 如果 PCSAMPLENA 置位，当所选的节拍位改变（0 到 1 或 1 到 0）并且后标量值计作 0 时，对 PC 进行采样并发出采样事件。
- 如果 CYCEVTENA 置位（并且 PCSAMPLENA 清零），当所选的节拍位改变（0 到 1 或 1 到 0）并且后标量值计作 0 时，发出一个事件。
- 应用程序和调试器能用它来测量消耗的执行时间。通过减去起始和结束时间，应用程序能够测量两个内核时钟（内核停止处于调试状态时除外）之间的时间。这对于 2^{32} 内核时钟周期是有效的（例如，在 50MHz 时大约为 82s）。

DWT CPI 计数寄存器

DWT CPI 计数寄存器用来计算除第一个周期之外的指令周期的总数目。

寄存器地址，访问类型和复位状态：

地址　　0xE0001008

访问　　读/写

复位状态　　-

图 11-6 显示了 DWT CPI 计数寄存器的位分配。

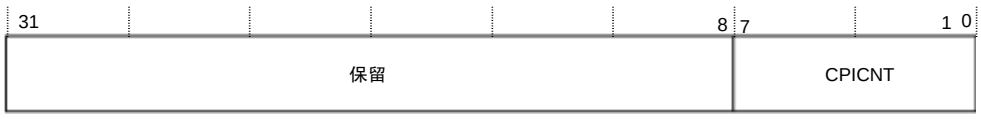


图 11-6 DWT CPI 计数寄存器的位分配。

表 11-9 描述了 DWT CPI 计数寄存器的位分配。

表 11-9 DWT CPI 计数寄存器的位分配

位	名称	定义
[31:8]	-	保留
[7:0]	CPICNT	当前 CPI 计数器的值。在执行除 DWT_LSUCNT 记录的指令之外的所有指令时，对所消耗的周期（第一个周期不计）进行递增计数。 在所有的指令取指暂停时该计数器仍递增计数。 如果 CPIEVTE 置位，则当计数器溢出时，发出一个事件。 在使能时清零。

DWT 异常开销计数寄存器

DWT 异常开销计数寄存器用来计算中断处理过程中消耗的总周期数。

寄存器地址，访问类型和复位状态：

地址　　0xE000100C

访问类型　　读/写

复位状态　　-

图 11-7 显示了 DWT 异常开销计数寄存器的位分配。

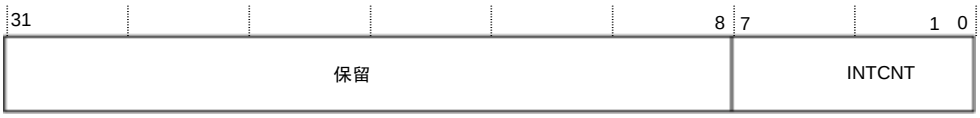


图 11-7 DWT 异常开销计数寄存器的位分配

表 11-10 描述了 DWT 异常开销计数寄存器的位分配

表 11-10 DWT 异常开销计数寄存器的位分配

位	名称	定义
[31:8]	-	保留
[7:0]	EXCCNT	当前中断开销计数器的值。计算中断处理过程中消耗的总周期数。（例如，入口堆栈，出栈返回，占先）。当计数器溢出（每 256 个周期）时，发出一个事件。该计数器在使能时初始化为零。 使能时清零。

DWT 睡眠计数寄存器

DWT 睡眠计数寄存器用来计算处理器处于睡眠状态的总周期数。

寄存器地址，访问类型和复位状态：

地址　　0xE0001010

访问类型　　读/写

复位状态　　-

图 11-8 显示了 DWT 睡眠计数寄存器的位分配。

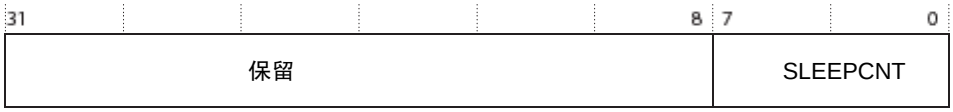


图 11-8 DWT 睡眠计数寄存器的位分配

表 11-11 描述了 DWT 睡眠计数寄存器的位分配。

表 11-11 DWT 睡眠计数寄存器的位分配

位	名称	定义
[31:8]	-	保留
[7:0]	SLEEPCNT	睡眠计数器。计算处理器处于睡眠状态的周期数。当计数器溢出时（每 256 个周期），发出一个事件。该计数器在使能时初始化为零。

注：

SLEEPCNT 使用 **FCLK** 作为时钟。当处理器处于睡眠状态时，可以减小 **FCLK** 的频率来降低功耗。这意味着在睡眠状态下，睡眠持续时间必须按 **FCLK** 频率来计算。

DWT LSU 计数寄存器

DWT LSU 计数寄存器用来计算当处理器处理 LSU 操作时的总周期数，第一个周期除外。

寄存器地址，访问类型和复位状态：

地址　　0xE0001014

访问类型　　读/写

复位状态　　-

图 11-9 显示了 DWT LSU 计数寄存器的位分配。

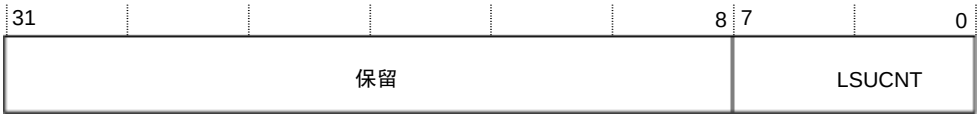


图 11-9 DWT LSU 计数寄存器的位分配

表 11-12 描述了 DWT LSU 计数寄存器的位分配。

表 11-12 DWT LSU 计数寄存器的位分配

位	名称	定义
[31:8]	-	保留
[7:0]	LSUCNT	LSU 计数器。用它来计算处理器处理一个 LSU 操作时需要的总周期数。最初执行指令时的开销不计算在内。 例如，对于需要 2 个周期才能完成的 LDR 操作，计数器的值只增加一个周期。同样的，对于暂停 2 个周期的 LDR 操作（共需 4 个周期），计数器的值加 3。当计数器溢出（每 256 个周期）时，发出一个事件。 使能时清零。

DWT 折迭（folded）计数寄存器

DWT 折迭（folded）计数寄存器用来计算折迭（folded）指令的总数目。当指令不需要任何周期时该计数器计作 1。

寄存器地址，访问类型和复位状态：

地址　　0xE0001018

访问类型　　读/写

复位状态　　-

图 11-10 显示了 DWT 折迭（folded）计数寄存器的位分配。

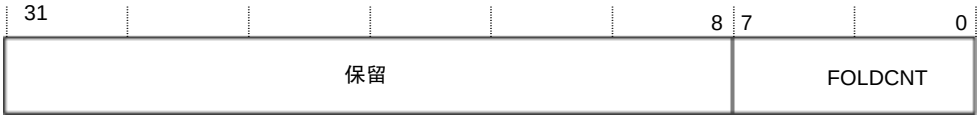


图 11-10 DWT 折迭（folded）计数寄存器的位分配

表 11-13 描述了 DWT 折迭（folded）计数寄存器的位分配

表 11-13 DWT 折迭（folded）计数寄存器的位分配

位	名称	定义
[31:8]	-	保留
[7:0]	FOLDCNT	计算折迭（folded）指令的总数目。该计数器在使能时初始化为零。

DWT 比较器寄存器

DWT 比较器寄存器 0-3 用来存放触发观察点事件写入的值。

寄存器地址，访问类型和复位状态：

地址　　0xE0001020，0xE00010230，0xE0001040，0xE0001050

访问类型　　读/写

复位状态　　-

表 11-14 描述了 DWT 比较器寄存器的位分配。

表 11-14 DWT 比较器寄存器的位分配

位	名称	定义
[31:0]	COMP	进行比较的数据。如果 CMATCH 置位，则 PC 与该值进行比较，否则与数据地址进行比较。 DWT_COMP0 也可与 PC 采样器计数器的值（DWT_CYCCNT）进行比较。

DWT 屏蔽寄存器 0-3

当 COMP 匹配时，DWT 屏蔽寄存器 0-3 用来对数据地址进行屏蔽。

寄存器地址，访问类型和复位状态：

地址　　0xE0001024，0xE00010234，0xE0001044，0xE0001054

访问类型　　读/写

复位状态　　-

图 11-11 显示了 DWT 屏蔽寄存器的位分配。

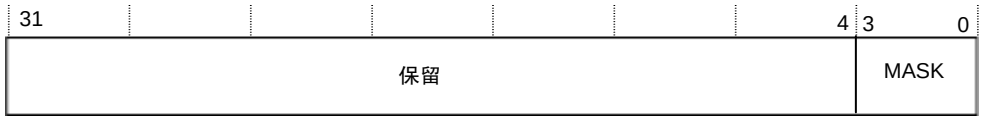


图 11-11 DWT 屏蔽寄存器的位分配

表 11-15 描述了 DWT 屏蔽寄存器的位分配。

图 11-11 DWT 屏蔽寄存器的位分配

位	名称	定义
[31:4]	-	保留
[3:0]	MASK	在与 COMP 相比时，对数据地址进行屏蔽。[3:0]的值为忽略屏蔽操作的位。因此，~0<<MASK 形成了对地址的屏蔽。即，DWT 匹配按下面的等式执行：（ADDR&(~0<<MASK)）==COMP 但是，为了使能与出现在总线上任何地方的地址进行匹配，实际的比较稍微复杂一些。例如，如果 COMP 为 3，则它与地址为 0 的字访问进行比较，因为 3 在一个字内。

DWT 功能寄存器 0-3

DWT 功能寄存器 0-3 用来控制比较器的操作。每个比较器能够：

- 与 PC 或数据地址相比。这是由 CYCMATCH 控制的。该功能只适用于比较器 0（DWT_COMP0）。
- 根据 FUNCTION 定义的操作，可发出几个数据或 PC，触发 ETM，或生成观察点。

寄存器地址，访问类型和复位状态：

地址　　0xE0001028，0xE00010238，0xE0001048，0xE0001058

访问　　读/写

复位状态　　0x00000000

图 11-12 显示了 DWT 功能寄存器 0-3 的位分配。

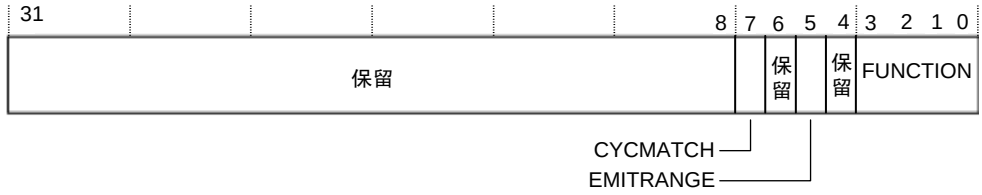


图 11-12 DWT 功能寄存器 0-3 的位分配

表 11-16 描述了 DWT 功能寄存器 0-3 的位功能。

表 11-16 DWT 功能寄存器 0-3 的位功能

位	名称	定义
[31:8]	-	保留
[7]	CYCMATCH	只适用于比较器 0。该位置位时，比较器 0 与 PC 采样器计数器进行比较。
[6]	-	保留
[5]	EMITRANGE	发出范围。被保留以允许当范围匹配时发出偏移量。复位时清零 EMITRANGE 位。当 EMITRANGE 使能时，不支持 PC 采样。 EMITRANGE 只适用于： FUNCTION=b0001，b0010，b0011。
[4]	-	保留
[3:0]	FUNCTION	功能设置见表 11-17。

表 11-17 DWT 功能寄存器的设置

值	功能
b0000	禁止
b0001	EMITRANGE=0，通过 ITM 对 PC 采样并发出 PC 采样事件。 EMITRANGE=1，通过 ITM 发出地址偏移量。
b0010	EMITRANGE=0，在读和写操作时通过 ITM 发出数据。 EMITRANGE=1，在读和写操作时通过 ITM 发出数据和地址偏移量
b0011	EMITRANGE=0，在读或写操作时通过 ITM 对 PC 和数据进行采样 EMITRANGE=1，在读或写操作时通过 ITM 发出地址偏移量和数据
b0100	PC 匹配时的观察点
b0101	读操作时的观察点
b0110	写操作时的观察点
b0111	读或写操作时的观察点
b1000	PC 匹配时的 ETM 触发
b1001	读操作时的 ETM 触发
b1010	写操作时的 ETM 触发
b1011	读或写操作时的 ETM 触发
b1100	保留
b1101	保留
b1110	保留
b1111	保留

注：

- 如果没有 ETM，则不可能有 ETM 触发。
- 数据采样只针对不产生故障（MPU 或总线故障）的访问。而 PC 采样不考虑任何故障。PC 采样只针对突发操作的第一个地址。
- 不建议在 PC 匹配时生成观察点，因为它在指令之后停止。它主要用于保护和 ETM 触发。

11.6 仪表跟踪宏单元

仪表跟踪宏单元(ITM)是应用导向的跟踪源，它支持 printf 类型调试，用于跟踪 OS 和应用事件并发出诊断系统信息。跟踪信息作为信息包从 ITM 发出，信息包能够由 3 个跟踪源产生。如果多个跟踪源同时产生信息包，则 ITM 对信息包输出的顺序进行仲裁，以下便是这 3 个跟踪源，其优先级按递减顺序排列：

- 软件跟踪。软件能够直接写入 ITM 激励寄存器（stimulus register）。该操作能够使信息包发出去。
- 硬件跟踪。信息包由 DWT 产生，由 ITM 发出。
- 时间戳（timestamp）。信息包根据时间戳来发送。ITM 含有一个 21 位的计数器，用来产生时间戳。该计数器使用 Cortex-M3 时钟或 SWV 输出的位时钟率。

11.6.1 ITM 寄存器总结和描述

注：

在对 ITM 进行编程或使用 ITM 之前，调试异常和监控控制寄存器的 TRCENA 位必须使能（见调试异常和监控控制寄存器）。

表 11-8 列出了 ITM 的寄存器。

表 11-18 ITM 寄存器

名称	类型	地址	复位值	描述
激励端口 0-31	读/写	0xE0000000- 0xE000007C	-	见 ITM 激励端口 0-31
跟踪使能	读/写	0xE0000E00	0x00000000	见 ITM 跟踪使能寄存器
跟踪特权	读/写	0xE0000E40	0x00000000	见 ITM 跟踪特权寄存器
控制寄存器	读/写	0xE0000E80	0x00000000	见 ITM 控制寄存器
综合写	只写	0xE0000EF8	0x00000000	见 ITM 综合写寄存器
综合读	只读	0xE0000EFC	0x00000000	见 ITM 综合读寄存器
综合模式控制	读/写	0xE0000F00	0x00000000	见 ITM 综合模式控制寄存器
锁定访问寄存器	只写	0xE0000FB0	0x00000000	见 ITM 锁定访问寄存器
锁定状态寄存器	只读	0xE0000FB4	0x00000003	见 ITM 锁定状态寄存器
PERIPHID4	只读	0xE0001FD0	0x00000004	值为 0x04
PERIPHID5	只读	0xE0001FD4	0x00000000	值为 0x00
PERIPHID6	只读	0xE0001FD8	0x00000000	值为 0x00
PERIPHID7	只读	0xE0001FDC	0x00000000	值为 0x00
PERIPHID0	只读	0xE0001FE0	0x00000002	值为 0x01
PERIPHID1	只读	0xE0001FE4	0x000000B0	值为 0xB0
PERIPHID2	只读	0xE0001FE8	0x0000000B	值为 0x0B
PERIPHID3	只读	0xE0001FEC	0x00000000	值为 0x00
PCELLID0	只读	0xE0001FF0	0x0000000D	值为 0x0D
PCELLID1	只读	0xE0001FF4	0x000000E0	值为 0xE0
PCELLID2	只读	0xE0001FF8	0x00000005	值为 0x05
PCELLID3	只读	0xE0001FFC	0x000000B1	值为 0xB1

注：

ITM 寄存器在特权模式下是可完全访问的。在用户模式下，所有寄存器都可读，但只有激励寄存器和跟踪使能寄存器能够执行写访问，并且只有在对应的跟踪特权寄存器位置位时。用户模式下无效的 ITM 寄存器写操作将被放弃。

ITM 激励端口 0-31

32 个激励端口中每一个都有自己的地址。如果跟踪使能寄存器中对应的位置位，则对其中一个地址执行写操作将使数据写入 FIFO。读任一个激励端口将返回位 0 的 FIFO 状态（0 为满，1 未满）。

可查询的 FIFO 接口不提供原子（atomic）读-修改-写操作，因此，如果中断和其它线程同时使用可查询的 printf 与 ITM 用法，则必须使用 Cortex-M3 专用监控程序。下面的查询代码能够保证在 ITM 的查询访问时，激励没有丢失。

```
;Cortex-M3 专用监控程序，被中断跳过
; R0 = FIFO 满/专用状态
; R1 = ITM 激励端口的基地址
; R2 =  写入的值
retry
LDREX R0, [R1, #??]        ; 读 FIFO 状态并请求专用锁定（excl lock）
CBZEQ R0, retry            ; FIFO 没有准备好，再试
STREX R0, R2, [R1, #??]    ; 如果 FIFO 满并且专用锁定，则保存
CZBNE R0, retry            ; 专用锁定失败，再试
```

ITM 跟踪使能寄存器

通过写对应的激励端口，可使用跟踪使能寄存器来产生跟踪数据。

寄存器地址，访问类型，复位状态：

访问　　读/写

地址　　0xE0000E00

复位状态　　0x00000000

表 11-19 描述了 ITM 跟踪使能寄存器的位分配。

表 11-19 ITM 跟踪使能寄存器的位功能

位	名称	定义
[31:0]	STIMULUS MASK	位屏蔽，使能对 ITM 激励端口的跟踪，每个激励端口对应一个位。

注：

如果 ITMEN 位置位，该寄存器可接受特权写访问。如果 ITMEN 置位并且适当的特权屏蔽清零，该寄存器可接受用户写访问。激励端口的特权访问使得 RTOS 内核保证能够满足仪表插槽或带宽要求。

ITM 跟踪特权寄存器

ITM 跟踪特权寄存器使操作系统能够控制哪些激励端口是用户代码可以访问的。

注：

只能在特权模式下写该寄存器。

寄存器地址，访问类型和复位状态：

访问类型　　读/写

地址　　0xE0000E40

复位　　0x00000000

图 11-13 显示了 ITM 跟踪特权寄存器的位分配。

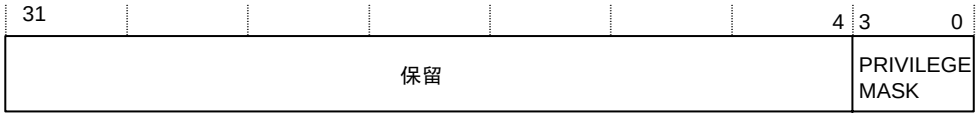


图 11-13 ITM 跟踪特权寄存器的位分配

表 11-20 描述 ITM 跟踪特权寄存器的位分配。

表 11-20 ITM 跟踪特权寄存器的位功能

位	名称	定义
[31:4]	-	保留
[3:0]	Privilege Mask	位屏蔽，使能对 ITM 激励端口的跟踪： 位[0]：激励端口[7:0] 位[1]：激励端口[15:8] 位[2]：激励端口[23:16] 位[3]：激励端口[31:24]

ITM 控制寄存器

该寄存器用来配置和控制 ITM 传输。

注：

只能在特权模式下写该寄存器。

寄存器地址，访问类型和复位状态：

访问　　读/写

地址　　0xE0000E80

复位　　0x00000000

图 11-14 显示了 ITM 控制寄存器的位分配。

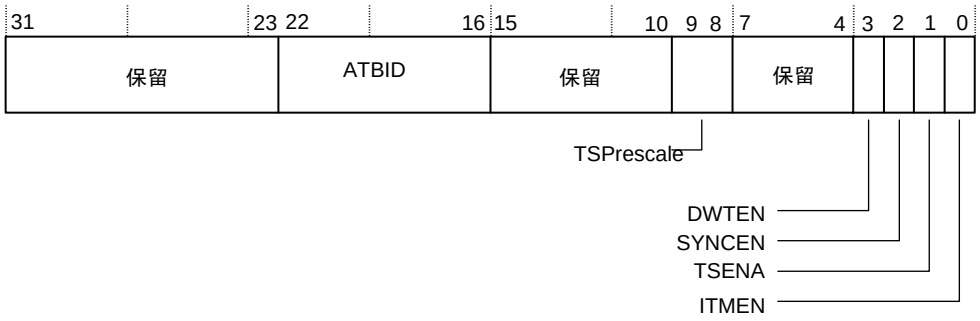


图 11-14 ITM 控制寄存器的位分配

表 11-21 描述了 ITM 控制寄存器的位分配。

表 11-21 ITM 控制寄存器的位功能

位	名称	定义
[31:23]	-	0b00000000
[22:16]	ATBID	CoreSight 系统的 ATB ID
[15:10]	-	0b000000
[9:8]	TSPrescale	时间戳预分频器 0b00：不分频 0b01：4 分频 0b10：16 分频 0b11：64 分频
[7:4]	-	保留
[3]	DWTEN	使能 DWT 激励
[2]	SYNCEN	使能用于 TPIU 的同步信息包
[1]	TSENA	使能差分时间戳。当时间戳计数器为非零时向 FIFO 写入信息包，以及当时间戳计数器溢出时，发出差分时间戳。在一个固定周期数之后的空闲周期内，发出时间戳。这为信息包和包内部的间隙提供了一个时间参考。
[0]	ITMEN	使能 ITM。这是 ITM 主机使能，必须在 ITM 激励和跟踪使能寄存器能够写入之前使能。

注：

DWT 在 ITM 模块中不使能，但进入 FIFO 的 DWT 激励入口由 DWTEN 控制。如果 DWT 需要时间戳，TSSEN 位必须置位。

ITM 综合写寄存器

该寄存器用来确定 ATVALIDM 位的行为。

图 11-15 显示了 ITM 综合写寄存器的位分配。

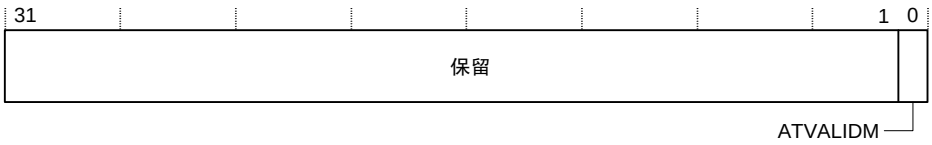


图 11-15 ITM 综合写寄存器的位分配

表 11-22 描述了 ITM 综合写寄存器的位分配。

表 11-22 ITM 综合写寄存器的位功能

位	名称	定义
[31:1]	-	保留
[0]	ATVALIDM	当设置了综合模式时： 0：ATVALIDM 清零 1：ATVALIDM 置位

注：

当设置了综合模式时，ATVALIDM 由位 0 驱动。

ITM 综合读寄存器

该寄存器用来读 ATREADYM 上的值。

图 11-16 显示了 ITM 综合读寄存器的位分配。

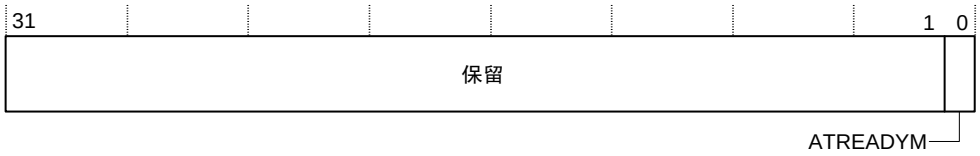


图 11-16 ITM 综合读寄存器的位分配

表 11-23 描述了 ITM 综合读寄存器的位分配。

表 11-23 ITM 综合读寄存器的位功能

位	名称	定义
[31:1]	-	保留
[0]	ATREADYM	ATREADYM 的值

ITM 综合模式控制寄存器

该寄存器用来使能对控制寄存器的写访问。

图 11-17 显示了 ITM 综合模式控制寄存器的位分配。

图 11-17 ITM 综合模式控制寄存器的位分配

表 11-24 描述了 ITM 综合模式控制寄存器的位分配。

表 11-24 ITM 综合模式控制寄存器的位功能

位	名称	定义
[31:1]	-	保留
[0]	INTEGRATION	0：常规的 ATVALIDM 1：综合写寄存器驱动的 ATVALIDM

ITM 锁定访问寄存器

该寄存器用来阻止对控制寄存器的写访问。

表 11-25 描述了 ITM 锁定访问寄存器的位分配。

表 11-25 ITM 锁定访问寄存器的位功能

位	名称	定义
[31:0]	Lock Access	在特权模式下对该寄存器写入 0xC5ACCE55 可使能对控制寄存器 0xE00::0xFFC 的进一步写访问。 一次无效的写操作将阻止写访问。

ITM 锁定状态寄存器

该寄存器用来使能对控制寄存器的写访问。

图 11-18 显示了 ITM 锁定状态寄存器的位分配。

图 11-18 ITM 锁定状态寄存器的位分配

表 11-26 描述了 ITM 锁定状态寄存器的位分配。

表 11-26 ITM 锁定状态寄存器的位功能

位	名称	定义
[31:3]	-	保留
[2]	ByteAcc	不能实现 8 位锁定访问
[1]	Access	组件的写访问被阻止。忽略所有写操作，允许读操作。
[0]	Present	表示该组件具有锁定机制。

11.7 AHB 访问端口

先进的高性能总线访问端口（AHB-AP）是进入 Cortex-M3 的调试访问端口，并提供对系统中的所有存储器和包括处理器寄存器（通过 NVIC）在内的寄存器进行访问。系统访问独立于处理器的状态。AHB-AP 通过 SW-DP 或 JTAG-DP 来访问。

AHB-AP 是进入总线矩阵的主机。使用 AHB-AP 的编程模型可以进行事务处理。详细信息见 AHB-AP 寄存器的总结和描述，这些寄存器生成进入总线矩阵的 AHBLite 事务处理。

11.7.1 AHB-AP 处理类型

AHB-AP 不支持总线上的背对背处理，因此所有的处理都是非连续的。AHB-AP 支持非对齐和 bit-band 处理，它们都由总线矩阵来操作。AHB-AP 处理不受到 MPU 锁定的影响。AHB-AP 处理将旁路 FPB，因此由 FPB 实现 AHB-AP 处理的重映射是不可能的。

AHB-AP 支持由 JTAG/SW-DP 启动的处理中止来驱动边带信号，HABORT。HABORT 进入总线矩阵，复位总线矩阵的状态，因此，对于“陷入绝境（last ditch）”的调试（读/停止/复位内核），专用外设总线能够通过 AHB-AP 来访问。

AHB-AP 处理采用小端格式。

11.7.2 AHB-AP 寄存器总结和描述

表 11-27 列出了 AHB-AP 寄存器。

表 11-27 AHB-AP 寄存器

名称	类型	地址	复位值	描述
控制和状态字（CSW）	读/写	0x00	见寄存器	见 AHB-AP 控制和状态字寄存器
传输地址（TAR）	读/写	0x04	0x00000000	见 AHB-AP 传输地址寄存器
数据读/写（DRW）	读/写	0x0C	-	见 AHB-AP 数据读/写寄存器
分组数据 0（BD0）	读/写	0x10	-	见 AHB-AP 分组数据寄存器 0-3
分组数据 1（BD1）	读/写	0x14	-	见 AHB-AP 分组数据寄存器 0-3
分组数据 2（BD2）	读/写	0x18	-	见 AHB-AP 分组数据寄存器 0-3
分组数据 3（BD3）	读/写	0x1C	-	见 AHB-AP 分组数据寄存器 0-3
调试 ROM 地址	只读	0xF8	0xE000E000	见 AHB-AP 调试 ROM 地址寄存器
标识寄存器（IDR）	只读	0xFC	0x04770011	见 AHB-AP ID 寄存器

AHB-AP 控制和状态字寄存器

该寄存器用来配置和控制 AHB 接口的传输。

图 11-19 显示了 AHB-AP 控制和状态字寄存器的位分配。

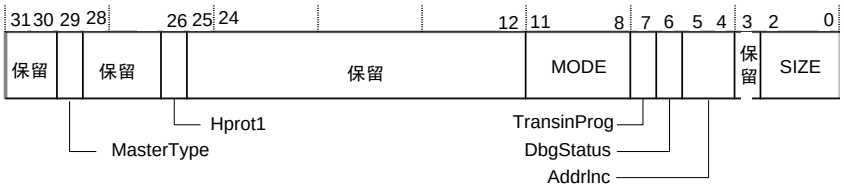


图 11-19 AHB-AP 控制和状态字寄存器

表 11-28 描述了 AHB-AP 控制和状态字寄存器。

表 11-28 AHB-AP 控制和状态字寄存器的位功能

位	名称	定义
[31:30]	-	保留，读作 b010
[29]	MasterType[a]	0：内核 1：调试 如果 COREACCEN=0，该位不能清零。读回该位的值来确认是否被接受。如果处理还未完成，该位不能改变（调试器必须先检验 TransinProg）。 复位值 0b1
[28:26]	-	保留，0b000
[25]	Hprot1	用户/特权控制-HPROT[1] 复位值 0b1
[24]	-	保留，0b1
[23:12]	-	保留，0x000
[11:8]	Mode	操作模式位： b0000：常规的下载/上传模式 b0001-b1111 保留 复位值 0b0000
[7]	TransINProg	传输正在处理。该位表示 APB 主端口上是否正在处理一次传输。
[6]	DbgStatus	表示 DBGEN 端口的状态。如果 DbgStatus 为低，没有执行 AHB 传输。 1：允许 AHB 传输 0：不允许 AHB 传输
[5:4]	AddrInc	数据读或写访问上的自动地址增加和装满模式。当前的处理在无错情况下完成时才执行增加操作。 在访问分组数据寄存器 0x10-0x1C 时，不执行自动地址增加和装满传输操作。此时，忽略这两个位的状态。 在 4KB 地址边界内递增并在该范围内重叠操作，例如以字方式从 0x1000 增加到 0x1FFC 时，如果开始地址为 0x14A0，计数器增加到 0x1FFC，回到 0x1000，然后继续增加到 0x149C。 0b00：自动增加关闭 0b01：单次增加。来自对应字节通道的单次增加。 0b10：增加操作已满 0b11：保留，不传输 地址增加的大小由位[2:0]定义。 复位值：0b00
[3]	-	保留
[2:0]	SIZE	访问大小： b000：8 位 b001：16 位 b010：32 位 b011-111 保留 复位值：b000

a 清零时，该位将禁止调试器置位调试停止控制和状态寄存器中的 C_DEBUGEN 位，从而避免调试器

将内核停止。

AHB-AP 转换地址寄存器

该寄存器用来对当前传输的地址进行编程。

表 11-29 描述了 AHB-AP 传输地址寄存器的位分配。

表 11-29 AHB-AP 传输地址寄存器的位功能

位	名称	定义
[31:0]	ADDRESS	当前传输的地址 复位值：0x00000000

AHB-AP 数据读/写寄存器

该寄存器用来读和写当前传输的数据。

表 11-30 描述了 AHB-AP 数据读/写寄存器的位分配。

表 11-30 AHB-AP 数据读/写寄存器的位功能

位	名称	定义
[31:0]	DATA	写模式：当前传输写入的数据。 读模式：当前传输读出的数据。 复位值：0x00000000

AHB-AP 分组数据寄存器 0-3

使用这些寄存器直接将 AHB-AP 访问映射为 AHB 传输，无需重写 AHB-AP 传输地址寄存器（TAR）。

表 11-31 描述了 AHB-AP 分组数据寄存器的位分配。

表 11-31 AHB-AP 分组数据寄存器的位功能

位	名称	定义
[31:0]	DATA	BD0-BD3 提供一个直接将 DAP 访问映射为 AHB 传输的机制，无需重写 4 字节边界的传输地址寄存器（TAR），即 BD0 对 TAR 执行读/写操作，BD1 对 TAR+4 执行读/写操作，等等。 如果 **DAPADDR[7:4]**==0x0001，则访问 0x10-0x1C 范围内的 AHB-AP 寄存器，然后获得的 **HADDR[31:0]**执行如下操作： 写模式：针对当前传输，写入外部地址 TAR[31:4]+**DAPADDR[3:0]** 的数据。 读模式：从外部地址 TAR[31:4]+**DAPADDR[3:0]**读出的当前传输的数据。 在 DAP 访问 BD0-BD3 时不执行自动地址增加操作。 只有字方式传输支持分组传输。非字的分组传输在当前忽略，假定为字访问。 复位值：0x00000000。

AHB-AP 调试 ROM 地址寄存器

该寄存器指定了调试接口的基地址，为只读寄存器。

表 11-32 描述了 AHB-AP 调试 ROM 地址寄存器的位分配。

表 11-32 AHB-AP 调试 ROM 地址寄存器的位功能

位	名称	定义
[31:0]	调试 ROM 地址	调试接口的基地址

AHB-AP ID 寄存器

该寄存器定义了访问端口上的外部接口。

图 11-20 显示了 AHB-AP ID 寄存器的位分配。

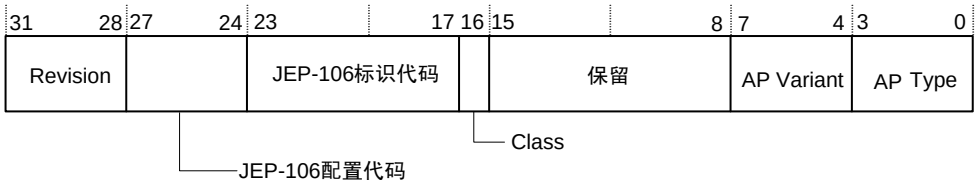


图 11-20 AHB-AP ID 寄存器

表 11-33 描述了 AHB-AP ID 寄存器的位分配。

表 11-33 AHB-AP ID 寄存器的位功能

位	名称	定义
[31:28]	Revision	第一次执行 AP 设计时该位为零。在对设计每作一次较大的修改时，对该位进行更新。
[27:24]	JEP-106 continuation code	对于一个采用 ARM 设计的 AP，[27:24]值为 0b0100，0x4。
[23:17]	JEP-106 identify code	对于一个采用 ARM 设计的 AP，[23:17]值为 0b0111011，0x3B
[16]	Class	0b1，这个 AP 是存储器访问端口。
[15:8]	-	保留，SBZ
[7:4]	AP Variant	0x1：Cortex-M3 变量
[3:0]	AP Type	0x1：AMBA AHB 总线

第12章 调试端口

本章描述了 Cortex-M3 处理器的调试端口。包含以下内容：

- 关于调试端口
- JTAG-DP
- SW-DP
- 通用调试端口（DP）的特性
- 调试端口的编程模型

12.1 关于调试端口

Cortex-M3 处理器包含一个用于调试访问的 AHB-AP 接口。该接口通过调试(DP)端口组件从外部访问处理器。Cortex-M3 支持两种可能的 DP 实现：

- JTAG 调试端口（JTAG-DP）。JTAG-DP 基于 IEEE 1149.1 测试访问端口（TAP）以及边界扫描结构，广义上称作 JTAG，它向 AHB-AP 端口提供 JTAG 接口。详细信息见 *JTAG-DP*。
- 串行线调试端口（SW-DP）。SW-DP 向 AHB-AP 端口提供两个脚的接口（时钟和数据）。详细信息见 *AHB 访问端口*。

这两个 DP 实现在调试访问 Cortex-M3 时提供不同的机制。您的实现可以包含这两个组件中的任一个，或两个都有。

注：

- 一次只能使用一个 DP，只有当两个 DP 都没有使用时才能在这两个调试端口间进行切换。
- 除了 SW-DP 或 JTAG-DP，您的实现可以含有一个针对特定 implementor 的 DP。详细信息见您的 implementor。

AHB-AP 的详细信息见 AHB-AP 访问端口。

DP 和 AP 一起称作调试访问端口（DAP）。

12.2 JTAG-DP

JTAG-DP 含有一个 DP 状态机（JTAG），用来控制 JTAG-DP 的操作，包括控制扫描链接口（扫描链接口提供与 JTAG-DP 连接的外部物理接口）。它是严格基于 JTAG TAP 状态机的，详细信息见 *IEEE Std 1149.1-1990*。本节将描述 JTAG-DP 及其扫描链接口。

图 12-1 显示了带有 JTAG-DP 的 ARM 调试接口，包括扫描链接口的操作。

这节内容包括：

- 扫描链接口
- 扫描链和 IR 指令
- DR 扫描链和 DR 寄存器

12.2.1 扫描链接口

JTAG-DP 组件包括：

- DP 状态机（JTAG）
- 指令寄存器（IR）和相关的 IR 扫描链，用来控制 JTAG 和当前所选的数据寄存器的行为。
- 众多数据寄存器和相关的 DR 扫描链，它们与 JTAG-DP 中的寄存器进行连接。

JTAG-DP 的物理连接

表 12-1 列出了 JTAG-DP 的 JTAG 连接。

表 12-1 JTAG-DP 的信号连接

端口	方向	描述
TDI	输入	调试数据输入
TDO	输出	调试数据输出
TCK	输入	调试时钟
TMS	输入	调试模式选择
nTRST	输入	调试复位

注：

JTAG 的 nTRST 输入能够通过上电复位驱动，并且不需要与专用管脚进行连接。

建议采用的 JTAG-DP 物理连接如图 12-1 所示。

图 12-1 JTAG-DP 物理连接

DAP 状态机（JTAG）

图 12-2 显示了 JTAG 状态机。

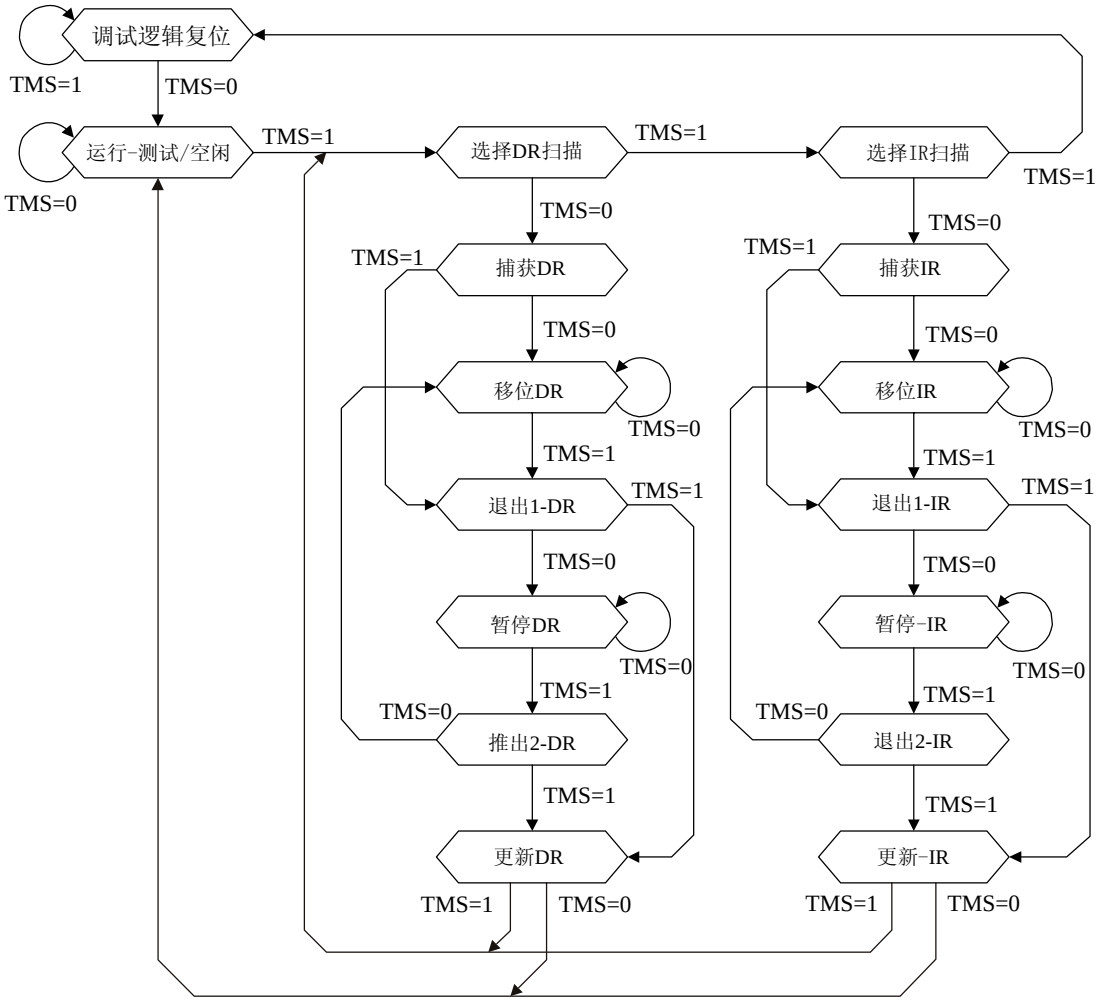


图 12-2 DAP 状态机（JTAG）

在使用 ARM 调试接口时，为了使调试处理能够正确工作，系统不可以在调试期间将电源从 JTAG-DP 中移除。如果移除了电源，DAP 控制器的状态将丢失。而在保持 JTAG-DP 有电时，DAP 能够使能 DAP 余下的部分并让内核掉电进入调试状态。

JTAG-DP 的基本操作

DAP 的 TDI 信号表示扫描链开始，TDO 信号输出表示扫描链结束。

参考图 12-2 中的 DAP 状态机（JTAG）。

- 当 JTAG 进入捕获-IR 状态时，有一个值传输到指令寄存器（IR）的扫描链上。该 IR 扫描链在 TDI 和 TDO 之间。
- 当 JTAG 处于移位-IR 状态时，对于从捕获-IR 到移位-IR 的处理，每一个 TCK 时钟，IR 扫描链就前进一位。这意味着在第一个时钟上，IR 的 LSB 在 TDO 上输出，IR 的位[1]传输到位[0]，位[2]传输到位[1]，等等。IR 的 MSR 用 TDI 的值取代。
- 当 JTAG 进入更新-DP 状态时，扫描到扫描链的值传输到指令寄存器中。

- 当 JTAG 进入捕获-DR 状态时，从其中一个数据寄存器(DR)中传输一个值到其中一条数据寄存器扫描链，该扫描链位于 TDI 和 TDO 之间。

  该数据然后在 JTAG 处于移位 DR 状态时进行移位，其方式与移位 IR 状态中的 IR 移位相同。

- 当 JTAG 进入更新-DR 状态时，扫描到扫描链中的值传输到数据寄存器中。
- 当 JTAG 在运行-测试/空闲状态时，不会出现特殊的操作。调试器可将该状态作为真正的睡眠(resting)状态。

注：

这与先前版本的基于 IEEE JTAG 标准的 ARM 调试接口相比，操作有所改变。对于 ARM 调试接口 v5，调试器不需要对 DAP 时钟进行门控，来获得真正的睡眠(resting)状态。

IR 和 DR 扫描链的操作在 *IR 扫描链和 IR 指令*以及 *DR 扫描链和 DR 寄存器*中详细描述。

nTRST 信号只对 JTAG 状态机逻辑进行复位。该信号通过异步操作，让 JTAG 状态机的逻辑进入调试-逻辑-复位状态。如图 12－2 所示，在 **TMS** 为高电平时通过 5 个 **TCK** 周期的序列，也可以从任意状态同步进入调试-逻辑-复位状态。但根据 JTAG 的初始状态，该操作可能使状态机经过其中一个更新状态，因而，产生了副作用。

在 DAP 内：

- DP 寄存器仅在上电复位时复位。

12.2.2 IR 扫描链和 IR 指令

本小节描述了 JTAG-DP 的指令寄存器（IR），它通过 IR 扫描链进行访问。

JTAG 指令寄存器(IR)

用途

保存当前的 DAP 控制器指令。

长度

4 位。

操作模式

在移位 IR 状态时，选择 IR 的移位区域作为 **TDI** 和 **TDO** 之间的串行路径。在捕获 **IR** 状态中，将二进制值 b0001 装入该移位区域。在移位 IR 过程中，以先移最低位的方式将移位区域中的值移出去。同时，以先移最低位的方式将新的指令移入。在更新 IR 状态时，移位区域的值装入 IR，因此变为当前指令。

在调试逻辑复位时，IDCODE 变成当前指令，请见 JTAG 器件 ID 代码寄存器(IDCODE)。

顺序

图 12-3 所示为指令寄存器的位顺序。

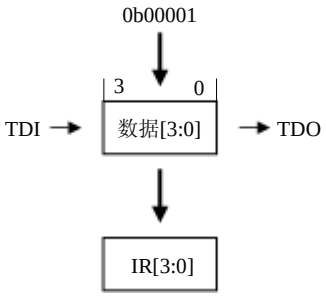


图 12-3　JTAG 指令寄存器位顺序

该寄存器强制遵循 IEEE 1149.1 标准。

IR 指令

对 JTAG 指令寄存器的描述显示了如何将一个 4 位指令传输到 IF 中。该指令决定了 JTAG 数据寄存器映射到的物理数据寄存器，如 DR 扫描链和 DR 寄存器所述。标准的 IR 指令在表 12-2 中列出，且推荐的自定义实现对该指令集的扩展，在*自定义实现对 IR 指令集的扩展中*描述。

不使用的 IR 指令值用来选择旁路寄存器(Bypass register)，详细描述见 JTAG 旁路寄存器。

表 12-2　标准 IR 指令

IR 指令值	JTAG-DP 寄存器	DR 扫描宽度	参考的章节
b0xxx	-	-	自定义实现对 IR 指令集的扩展
b1000	ABORT	35	JTAG-DP 中止寄存器(ABORT)
b1001	-	-	
b1010	DPACC	35	JTAG DP/AP 访问寄存器(DPACC/APACC)
b1011	APACC	35	
b110x	-	-	
b1110	IDCODE	32	JTAG 器件 ID 代码寄存器(IDCODE)
b1111	BYPASS	1	JTAG 旁路寄存器(BYPASS)

自定义实现对 IR 指令集的扩展

8 个 IR 指令 b0000 到 b0111 被保留，供对 JTAG-DP 进行扩展的自定义实现使用。这些寄存器可用于访问边界扫描寄存器（遵循 IEEE1149.1）。需要的指令在表 12-3 中列出。

注：

- ARM 公司建议将单独的 JTAG TAP 用于边界扫描和调试。
- 如果将 IR 寄存器设为不能执行或保留的 IR 指令值，那么选择了旁路寄存器。

表 12-3 建议的自定义实现的 IR 指令（遵循 IEEE 1149.1）

IR 指令值	指令	是 IEEE 1149.1 要求的吗？
b0000	EXTEST	是
b0001	SAMPLE	是
b0010	PRELOAD	是

续表 12-3

IR 指令值	指令	是 IEEE 1149.1 要求的吗？
b0011	保留	-
b0100	INTEST[a]	否
b0101	CLAMP[a]	否
b0110	HIGHZ[a]	否
b0111	CLAMPZ[a]	否，见 CLAMPZ 指令。

a. 如果不执行，则该指令保留。

如果你需要一个边界扫描实现，则你必须在到 JTAG-DP 的封包（wrapper）中执行 IEEE1149.1 要求的指令。表 12-3 中列出的其它 IR 指令值被保留，如果在边界扫描中执行这些指令，则需对这些保留值进行编码。如果执行，则这些指令必须根据 IEEE.1149.1 规范进行动作。如果不执行，则这些保留值选择旁路寄存器。

IEEE 1149.1 规范也要求 IDCODE 和 BYPASS 指令。而这些指令包含在表 12-2 中。

CLAMPZ 指令

CLAMPZ 不是 IEEE 1149.1 指令。

如果执行，则在选择 CLAMPZ 指令时所有的三态输出都置于其无效状态，但供输出的数据从扫描单元中获得。

能够执行 CLAMPZ 以便确保在生产测试过程中，每个输出在其值为 0 或 1 时禁止。如果需使用该功能，则应对它进行编码。

JTAG 旁路寄存器(BYPASS)

用途

通过在 **TDI** 和 **TDO** 之间提供直接路径来将器件旁路。

长度

1 位

操作模式

当 BYPASS 指令是 IR 中的当前指令时：

- 在移位-DR 状态中，数据从 **TDI** 传输到 **TDO**，带一个 TCK 周期延时。
- 在捕获-DR 状态中，向该寄存器加载逻辑 0。
- 在更新-DR 状态中不发生任何操作。

顺序

图 12-4 所示为旁路寄存器的操作。

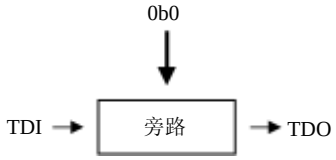


图 12-4　JTAG 旁路寄存器操作

该寄存器强制遵循 IEEE 1149.1 标准。

12.2.3 DR 扫描链和 DR 寄存器

下面有 5 个物理 DR 寄存器：

- BYPASS 和 IDCODE 寄存器，如 IEEE 1149.1 标准所定义。
- DPACC 和 APACC 访问寄存器
- ABORT 寄存器，用来中止传输

有一条扫描链与上述每个寄存器关联。正如 IR 扫描链和 IR 指令中所述，IR 寄存器中的值确定了哪条扫描链与 **TDI** 和 **TDO** 信号相连。

JTAG 器件 ID 代码寄存器（IDCODE）

用途

器件标识。器件 ID 代码值使调试器能够识别与之相连的调试端口。不同的调试端口有不同的器件 ID 代码，因此调试器可将其区分开来。

这是标识代码寄存器的 JTAG-DP 实现，请见标识代码寄存器(IDCODE)。

长度

32 位。

操作模式

当 IDCODE 指令是 IR 中的当前指令时，器件 ID 代码寄存器中的移位区域被选择作为 **TDI** 和 **TDO** 之间的串行路径。

- 在捕获-DR 状态中，将 32 位器件 ID 代码装入该移位区域
- 在移位-DR 寄存器状态中，以先移最低位的方式将该数据移出。
- 在更新-DR 状态中忽略移入的数据。

顺序

图 12-5 所示为器件 ID 代码寄存器的位顺序。

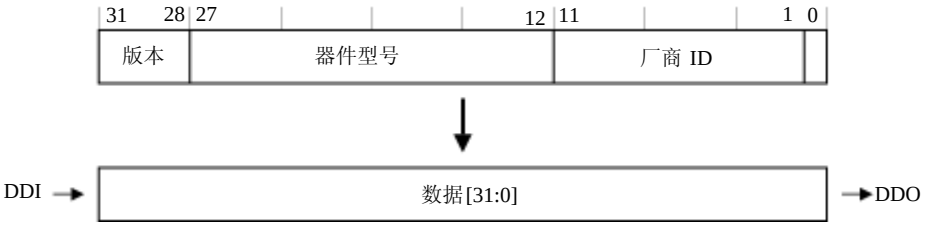


图 12-5　JTAG 设备 ID 代码寄存器位顺序

JTAG DP/AP 访问寄存器（DPACC/APACC）

DPACC 和 APACC 扫描链具有相同的格式。

用途

启动 DP 或 AP 访问，来访问 DP 或 AP 寄存器。DPACC 和 APACC 用作对寄存器的读和写访问。

DPACC 用于访问 CTRL/STAT，SELECT 和 RDBUFF 寄存器，请见 JTAG-DP 寄存器映射。

APACC 用于访问所有的 AP 寄存器，有关访问 AHB-AP 寄存器的详细信息请见 AHB-AP 寄存器的汇总和描述，访问 JTAG-AP 寄存器的详细信息请见 JTAG-DP 寄存器。

长度

35 位。

操作模式

当 DPACC 或 APACC 指令是 IR 中的当前指令时，选择 DP 访问寄存器或 AP 访问寄存器的移位区域作为 **TDI** 和 **TDO** 之间的串行路径：

- 在捕获-DR 状态中，返回先前处理的结果（如果有）以及 3 位 ACK 响应。仅执行两个 ACK 响应，见表 12-4 中的总述。

表 12-4　DPACC 和 APACC 的 ACK 响应

响应	ACK[2:0]编码	参考
OK/FAULT	b010	DPACC 或 APACC 访问的 OK/FAULT 响应
WAIT	b001	DPACC 或 APACC 访问的 WAIT 响应

其它所有的 ACC 编码被保留。

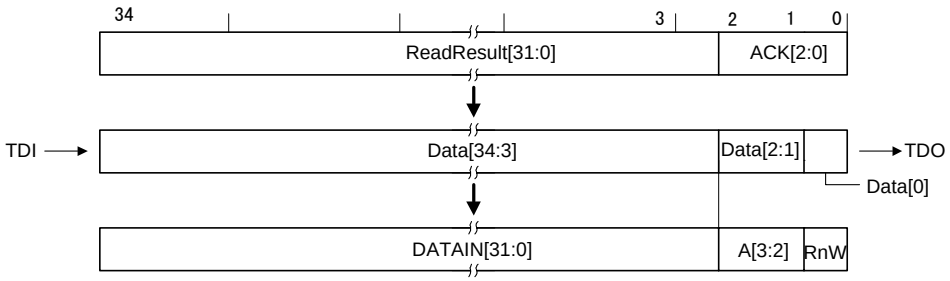
- 在移位-DR 状态中，以先移最低位的方式将该数据移出。如图 12-6 所示，移出数据的前 3 位为 ACK[2:0]，因此你可以在无需移出所有返回的数据时检查 ACK 响应，请见 DPACC 或 APACC 访问的 WAIT 响应。

  将返回的数据移出到 **TDO** 时，新的数据从 **TDI** 中移入。请见 DPACC 或 APACC 访问的 OK/FAULT 响应。

- 更新-DR 状态中的操作取决于 ACK[2:0]的响应为 OK/FAULT 还是 WAIT。这两种情况在以下部分描述：
  - 紧随 OK/FAULT 响应的更新-DR 操作
  - 紧随 WAIT 响应的更新-DR 操作

顺序

图 12-6 所示为 DP 和 AP 访问寄存器的位顺序。



12-6 JTAG DP 和 AP 访问寄存器的位顺序

DPACC 或 APACC 访问的 OK/FAULT 响应

如果 ACK[2:0]指示的响应为 OK/FAULT，则先前的操作已完成。响应代码不显示传输是成功完成还是传输出错。你必须读 CTRL/STAT 寄存器来判断传输是否成功，请见控制/状态寄存器（CTRL/STAT）。

- 如果先前的处理是读操作并已成功完成，则捕获的 ReadResult[31:0]为请求的寄存

器值。该结果作为数据[34:3]移出。

- 如果先前的处理是写操作或没有成功完成的读操作，则捕获的 ReadResult[31:0]不可预知，如果数据[34:3]被移出，则它必须丢弃。

紧随 OK/FAULT 响应的更新-DR 操作

移入扫描链的值构成对寄存器的读或写请求：

- 如果当前 IR 指令是 DPACC，则将 TDI 和 TDO 连接到 DPACC 扫描链，并且请求读或写 DP 寄存器。
- 如果当前 IR 指令是 APACC，则将 TDI 和 TDO 连接到 APACC 扫描链，并且请求读或写 AP 寄存器。

在任何一种情况下：

- 如果 RnW 移入 0，则请求将 DATAIN[31:0]中的值写入被寻址的寄存器。
- 如果 RnW 移入 1，则请求读被寻址的寄存器的值。DATAIN[31:0]中的值被忽略。你必须再次读扫描链来获得寄存器中读出的值。

寻址需要的寄存器：

- 在 DPACC 访问的情况下，通过移入 A[3:2]的值来读 DP 寄存器。有关寻址的详细内容请见 JTAG-DP 寄存器映射。
- 在 APACC 访问的情况下，通过结合以下情况来读 AP 寄存器：
  - 移入 A[3:2]的值
  - DP 中 SELECT 寄存器的当前值，请见 AP 选择寄存器，SELECT。
  - 如果你想访问 AHB-AP 寄存器，则有关寄存器寻址的详细信息见表 11-27。

一次 DPACC 或 APACC 扫描能够在请求另一次寄存器访问的同时，返回前一次读操作的结果，因此，寄存器访问能够采用流水线技术来处理。在具有流水线属性的寄存器读操作序列结束时，你可以读 DP RDBUFF 寄存器来返回最后一次读操作的结果。DP RDBUFF 寄存器的读操作不会在 JTAG 的操作上产生影响，请见读缓冲寄存器，RDBUFF。更多关于一次 DPACC 或 APACC 扫描如何从前一次扫描中返回结果的信息请见目标响应总结部分。

如果当前 IR 指令为 APACC，引起 APACC 访问：

- 如果 DP CTRL/STAT 寄存器中的任意 sticky 标志置位，则放弃处理。下一次扫描立即返回 OK/FAULT 响应。详细信息见 *sticky 标志和 DP 错误响应*，以及*控制/状态寄存器，CTRL/STAT*。
- 如果推动比较或推动验证(pushed compare or pushed verify)操作使能，则 RnW 的扫描输入值必须为 0，否则操作不可预知。在更新-DR 状态时，发出读请求，返回的值与 DATAIN[31:0]进行比较，DP CTRL/STAT 寄存器中的 STICKYCMP 标志根据这个比较值来更新。详细信息见*推动比较和推动验证操作*。推动操作通过 DP CTRL/STAT 寄存器中的 TRNMODE 区域来使能，详细信息见*控制/状态寄存器，CTRL/STAT*。
- AP 访问直到 AP 通知它已完成时才结束。例如，如果你访问存储器访问端口（AHB-AP），该操作可能引起对连接到 AHB-AP 的存储器系统的访问。此时，直到存储器系统向 AHB-AP 告知存储器访问已完成时，该访问操作才结束。

DPACC 或 APACC 访问的 WAIT 响应

WAIT 响应表示前面的处理没有完成。主机应再次执行 DPACC 或 APACC 访问。

注：

前面的处理可以是 DP 或 AP 访问。通过返回 WAIT，DP 访问停止，直到前面的所有 AP 处理完成。

通常，如果软件检测到 WAIT 响应，则它会重试相同的传输操作。这使协议能够尽可能快地处理数据。但如果软件已对该传输重试了许多次，给较慢的互连和存储器系统留了足够的时间来响应，则它可能会写 ABORT 寄存器来取消操作。这将通知活动的 AP，告知 AP 应终止当前正在尝试的传输，并允许访问调试系统的其它部分。AP 可能无法终止在其 ASIC 接口上的传输。但在接收到 ABORT 时，AP 应释放其 JTAG 接口。

紧随 WAIT 响应的更新-DR 操作

在更新-DR 状态时不产生任何请求并放弃移入的数据。捕获的 ReadResult[31:0]不可预知。

注：

你可以在不移出整个 DP 或 AP 访问寄存器的情况下检测 WAIT 响应，详细信息见表 12-4 的响应。

DPACC 或 APACC 访问的 sticky 溢出行为

在捕获-DR 状态时，如果前面的处理没有完成，则产生 WAIT 响应。此时，如果溢出检测标志置位，则 Sticky Overrun 标志，即 STICKYORUN 置位。有关 Overrun Detect 和 Sticky Overrun 标志的详细信息请见*控制/状态寄存器，CTRL/STAT*。

当前面的处理保持没有完成状态时，后面的扫描也接收到 WAIT 响应。

一旦前面的处理完成，其它的 APACC 处理被放弃，对扫描立即给出 OK/FAULT 响应。但能够访问 DP 寄存器。特别是能够访问 CTRL/STAT 寄存器，来确认 Sticky Overrun 标志是否置位，并在收集了所有关于溢出条件的所需信息之后将该标志清零。详细信息见*溢出检测*。

最小响应时间

如在 *DPACC 或 APACC 访问的 OK/FAULT 响应*中所阐述的，在一次 DPACC 或 APACC 访问的更新-DR 状态中启动 DP 或 AP 寄存器访问，访问结果在下一次 DPACC 或 APACC 访问的捕获-DR 状态时返回。但如果请求的寄存器访问没有完成，则第二次访问返回 WAIT 响应。

JTAG 时钟 **TCK** 与进行调试的系统的内部时钟是异步的，完成一次访问所需的时间由这两种时钟域来计时。但更新-DR 状态和捕获-DR 状态之间的时序只包含 **TCK** 周期。参考图 12-2，有两条路径可从更新-DR 状态（在该状态中启动寄存器访问）到达捕获-DR 状态（在该状态中捕获响应）：

- 通过选择-DR-扫描状态的直接路径
- 通过运行-测试/空闲和选择-DR-扫描状态的路径

如果采用第二条路径，则状态机能够在运行-测试/空闲状态中消耗任意个 **TCK** 周期。这意味着可以在更新-DR 和捕获-DR 状态之间变化 **TCK** 的周期数。

JTAG 实现可以在更新-DR 状态和捕获-DR 状态之间的 **TCK** 周期数上，降低对自定义

实现的限制。并且如果在达到该限制之前进入捕获-DR 状态，则始终产生一个立即的 WAIT 响应。虽然任何调试器都必须能够成功地从任意的 WAIT 响应中恢复，但 ARM 公司仍建议调试器应能够适应所有自定义实现的限制。

另外，在访问 AP 寄存器或通过 AP 访问相连的一个器件时，系统内可能有其它可变的响应延迟。一个能适应这些延迟，避免多余的 WAIT 扫描的调试器能够更有效地工作并提供更高的最大数据吞吐量。

目标响应总结

如在 *DPACC 或 APACC 访问的 OK/FAULT 响应*以及*最小响应时间*中所描述的，在一次 DPACC 或 APACC 访问的更新-DR 状态中启动 DP 或 AP 寄存器访问，在下一次 DPACC 或 APACC 访问的捕获-DR 状态中返回结果。表 12-5 针对前一次扫描中的每一个可能的 DPACC 或 APACC 访问，总结了捕获-DR 状态时的目标响应。

注：

表 12-5 中显示的目标响应与当前 DPACC 或 APACC 扫描中正在执行的操作是独立的。在表中，*读结果*为移出的数据 Data[34:3]，ACK 根据移出的数据 Data[2:0]来译码。

表 12-5 JTAG 目标响应总结

前一次扫描，在更新-DR 状态 [a]				当前扫描，在捕获-DR 状态			备注
R/W	**IR**	**ADDR[b]**	**Sticky[c]**	**AP 状态 [d]**	**读结果**	**ACK**	
X	X	bxx	x	忙	UNP[e]	WAIT	可引起 Sticky Oerrun 标志置位 [f]
R	DPACC	b01	x	不忙	CTRL/STAT	OK/FAULT	返回 CTRL/STAT 的值
		b10			SELECT		返回 SELECT 的值
		b00 或 b11			0x00000000		在地址 b00 和 b11，没有可读的 DP 寄存器
W	DPACC	b01	x	不忙	UNP[e]	OK/FAULT	写 CTRL/STAT 寄存器
		b10					写 SELECT 寄存器
		b00 或 b11					写操作忽略
R	APACC	bxx	否	准备	见备注	OK/FAULT	见脚注 [g]
				错误	UNP[e]		Sticky Error 标志置位
W	APACC	bxx	否	准备	UNP[e]	OK/FAULT	见脚注 [h]
				错误	UNP[e]		Sticky Error 标志置位
X	APACC	bxx	是	x	UNP[e]	OK/FAULT	放弃前一次处理

a　对于在捕获-DR 状态时 ACK 响应为 OK/FAULT 的扫描来说，前面的扫描为最近的扫描。进入更新状态使得后面的 WAIT 响应被放弃。

b　DPACC 或 APACC 访问中的 ADDR[3:2]

c　sticky 列表示在 DP CTRL/STAT 寄存器中是否有 sticky 标志置位。请*见控制/状态寄存器，CTRL/STAT*。

d　在当前扫描到达捕获-DR 状态时的 AP 状态，或在该时刻来自 AP 的响应。

e　UNP 为不可预知

f　如果溢出检测标志置位，则该访问/响应序列将导致 Sticky Overrun 标志置位。请*见控制/状态寄*

存器，CTRL/STAT。

g　如果推动验证或推动比较使能，则操作是不可预知的，否则返回前一次扫描时寻址的 AP 寄存器的值。

h　如果推动验证或推动比较使能，则前一次处理执行所需的推动操作，该操作可能具有置位的 Sticky 比较标志，见*推动比较和推动验证操作*。否则，前一次扫描捕获的值已写入请求的 AP 寄存器。

主机响应总结

在表 12-5 中，“当前扫描，在捕获－DR 状态”部分的“ACK 列”显示了在启动一次 DPACC 或 APACC 访问之后主机可能接收到的响应。

表 12-6 JTAG 主机响应总结

JTAG 访问类型	来自目标的 ACK	对应 ACK，可能的主机动作
读	OK/FAULT	捕获读数据
写	OK/FAULT	没有更多必需的动作
读或写	WAIT	重复相同的访问操作，直到接收到 OK/FAULT ACK 或到达等待超时 如果有必要，使用 DAP ABORT 寄存器来使能对 AP 的访问。
读或写	无效的 ACK	假设产生目标或线错误，并视为重大错误。

JTAG-DP 中止寄存器（ABORT）

用途

访问 DP 中止寄存器来强制 DAP 中止。

这是中止寄存器的 JTAG-DP 实现，见*中止寄存器，ABORT*。

长度

35 位

操作模式

当 ABORT 指令为 IR 中的当前指令时，**TDI** 和 **TDO** 之间的串行路径连接到一条用来访问中止寄存器的 35 位的扫描链上。

调试器必须将值 0x00000008 扫描到扫描链。该值：

- 将 RnW 位写作 0
- A[3:2]写作 b00
- 中止寄存器的位 1，即 DAPABORT 位写入 1

注意：

向该扫描链写入任何其它值，其影响不可预知。

顺序

图 12-7 显示了 ABORT 扫描链的位顺序。

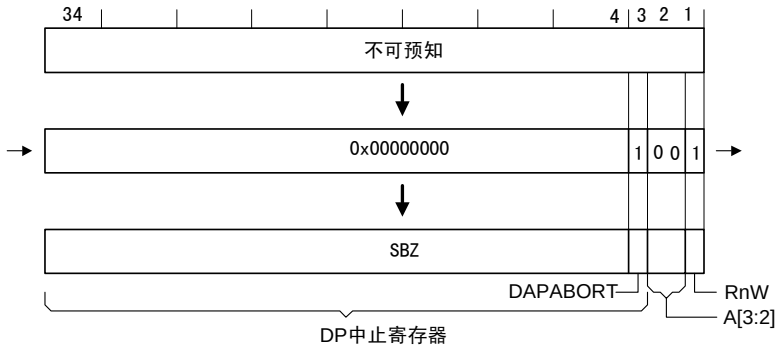


图 12-7 JTAG-DP ABORT 扫描链的位顺序

12.3 SW-DP

本节针对 ARM 串行线调试（SW-DP）接口给出了结构上的描述。特别描述了串行线调试（SWD）协议，以及该协议如何向 DP 寄存器提供访问。DP 寄存器将在*调试端口的编程模型*部分详细描述。

SW-DP 利用同步的串行接口来操作。该接口使用一个双向数据信号和时钟信号。

线上的每个操作序列都由以下 2 个或 3 个阶段组成：

数据包请求

外部主机调试器向 DP 发出请求。DP 是请求的目标。

应答

目标向主机发送应答。

数据传输阶段

只有当数据读或数据写请求后跟随一个有效的（OK）应答时，或 CTRL/STAT 寄存器中的 ORUNDETECT 标志置位时才呈现该阶段。请*见控制/状态寄存器，CTRL.STAT*。

数据传输可以从目标到主机（遵循读请求），或主机到目标（遵循写请求）。

注：

如果 DR CTRL/STAT 寄存器中的溢出检测位置位，则所有响应上都需要数据传输阶段，包括 WAIT 和 FAULT。有关 CTRL/STAT 寄存器的详细信息请见*控制/状态寄存器，CTRL/STAT*。

12.3.1 时钟

SW-DP 的时钟 **DBGCLK** 能够与 Cortex-M3 时钟异步。当调试端口空闲时，**DBGCLK** 能够停止。在最后一个数据位已传输到线上时，必须再计时两个 **DBGCLK** 时钟。

12.3.2 调试接口概述

本小节将简要介绍 SW-DP 使用的物理接口。

线接口

SW-DP 将一条串行线用于主机和目标信号。主机仿真器驱动协议的时序—只有主机仿真器产生数据包的报头（packet header）。

SW-DP 在同步模式下工作，需要一个时钟管脚和数据管脚。

同步模式使用时钟参考信号，该信号能够由片内的信号源产生并输出，或由主机器件提供。然后主机将该时钟用作数据生成和采样时的参考信号，这样，目标就不必执行任何过采样（oversampling）。

目标和主机都能够驱动总线的高和低电平，或将其置于三态。端口必须能够承受短时间的竞争现象来允许不同步。

线上拉

主机和目标都可以将线驱动为高或低电平。因此，通过提供无驱动（undriven）的时间片（time solt）作为切换（handover）的一部分来确保不出现竞争现象，这一点非常重要。这样，当主机和目标都没有驱动线时，能够将线假定为处于一个已知状态。目标端需要一个 100kΩ的上拉电阻，而只能依靠它来保持线的状态。如果将线驱动为低电平并释放，则上拉电阻最终将线拉为高电平，但这需消耗多个位周期。

在没有连接主机时，使用上拉电阻来防止信号的错误检测。当主机将线拉低时，该上拉必须采用大电阻以减小来自目标端的 IDLE 状态的电流消耗。

注：

任何时候只要将线驱动为低电平，都将使目标端的电流消耗非常低。如果该接口在目标必须使用低功耗模式时保持连接状态并持续所需的周期，则线必须由主机来保持高电平或复位状态直到接口必须被激活。

线掉转（turn-round）

为避免竞争，当驱动线的器件变化时，必需有一个掉转（turnaround）周期。

空闲和复位

在两次传输之间，主机必须将线驱动为低电平进入 IDLE 状态，或立即发送一次新传输的起始位继续传输。在一次数据包传输之后，主机也可以空闲，使线保持为高电平（可以驱动为高电平或三态）。这样可减小静态电流消耗，但如果在自由运行的时钟下使用该方法，则最少必须使用 50 个时钟周期，后面跟随一个 READ-ID，作为重新连接的新序列。

该协议没有明确的复位信号。在没有看到预期的协议时，主机或目标对复位进行检测。在协议能够以一个训练序列（training sequence）重新开始之前，链接的两端变为复位信号，这一点非常重要。通过保持线为高电平或三态并持续 50 个时钟周期，之后跟随一个读 ID 请求，可使检测到协议错误之后或复位之后的重新同步能够成功。如果 SW-DP 检测到同步已丢失，例如，没有看到预期的停止位，则它让线保持为无驱动，并等待主机以一个新的报头重新开始，或通过不对线进行驱动来通知复位。如果 SW-DP 在一行中检测到两个错误的数据序列，则它将数据挡在外面，直到出现 DBGDI 为高电平并持续 50 个时钟周期的复位序列。

如果主机没有看到来自 SW-DP 的预期的响应，则它必须给 SW-DP 一定的时间来返回一个数据的有效载荷。然后主机能够读 SW-DP ID 代码寄存器来重试。如果不成功，则主机必须尝试复位。

12.3.3 协议操作概述

本小节简要介绍协议的双向操作。其中阐述了 SW-DP 接口数据连接上每一个可能的操作序列。

操作序列在以下部分阐述：

- 成功的写操作（OK 响应）
- 成功的读操作（OK 响应）
- 读或写操作的 WAIT 响应
- 读或写操作的 FAULT 响应
- 协议错误序列

阐述过程中用到的术语在*阐述操作时的关键词*部分描述。

阐述操作时的关键词

在阐述可能的不同操作时使用以下术语：

Start　　单个开始位，值为 1

DPnAP　　单个位，表示访问的是 DP 访问寄存器还是 AP 访问寄存器。该位为 0 表示 APACC 访问，为 1 表示 DPACC 访问。

RnW　　单个位，表示是读访问还是写访问。该位为 0 表示写访问，为 1 表示读访问。

ADDR[2:3]　　两位，给出 DP 或 AP 寄存器地址的 ADDR[3:2]地址区域。

- 对于 APACC 访问，正在寻址的寄存器取决于 ADDR[3:2]的值以及 SELECT 寄存器中的值。有关寻址的详细信息见：
  - 表 11-27，如果想访问 AHB-AP 寄存器。

    有关 SELECT 寄存器的详细信息见 *AP 选择寄存器，SELECT*。
- 对于 DPACC 访问，A[3:2]决定了访问的是哪个寄存器，见表 12-13。

注：

A[3:2]的值在线上以 LSB 在先的方式发送。

Parity　　为前面的数据包提供单一的奇偶校验位

注：

- 奇偶校验位用来对数据包的报头部分和数据部分分别进行保护。ACK 位已进行冗余编码，并且不使用其它任何附加的保护。
- 为推测报头的奇偶校验位，需计算 **DP/AP**，**RnW** 以及 **ADDR[0:1]**中为 1 的位数目。如果该值为奇数（1 或 3），则奇偶校验位置位，使 1 的总数目为偶数。为推测数据区的奇偶校验位，计算 32 个数据位中为 1 的位数目。如果该值为奇数，则奇偶校验位应置位，使 1 的总数目为偶数。

Stop　　单一的停止位，在同步 SWD 协议中，该值始终为 0。

Park　　单一位。在传输该位时，主机不对线进行驱动，该串行线由 SWD 接口硬件拉高。目标将该位读作 1。

Trn　　掉转周期（turnaround）。在该周期中，不对线进行驱动，并且线的状态未定义。掉转周期的长度由线控制寄存器中的 TURNROUND 区域控制，见*线控制寄存器，WCR（只用于 SW-DP）*。掉转周期的默认设置为一个时钟周期。

注：

本章给出的所有例子指示一个时钟的默认掉转周期。

在异步 SWD 协议中没有附加的掉转周期。

Ack　　一个 3 位的目标到主机响应。这些位在线上以 LSB 在先的方式出现。

WDATA[0:31]　　32 位写数据，从主机到目标。

注：

WDATA[0:31]的值在线上以 LSB 在先的方式发送。

RDATA[0:31]　　32 位读数据，从目标到主机。

注：

RDATA[0:31]的值在线上以 LSB 在先的方式发送。

成功的写操作（OK 响应）

一次成功的写操作由以下 3 个阶段组成：

- 一个 8 位的写数据包请求，从主机到目标
- 一个 3 位的 OK 应答，从目标到主机
- 一个 33 位的数据写阶段，从主机到目标。

默认情况下，每两个阶段之间都有一个时钟的掉转周期。详细信息见*阐述操作时的关键词*中对 **Trn** 的描述。

一次成功的写操作如图 12-8 所示。

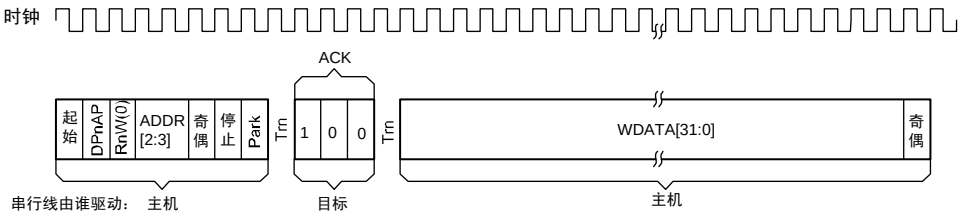


图 12-8 串行线调试时成功的写操作

成功的读操作（OK 响应）

一次成功的读操作由以下 3 个阶段组成：

- 一个 8 位的读数据包请求，从主机到目标
- 一个 3 位的 OK 应答，从目标到主机
- 一个 33 位的数据读阶段，在该阶段，数据从目标传输到主机。

默认情况下，在第一和第二个阶段之间以及第三阶段之后有一个时钟的掉转周期。详细信息见*阐述操作时的关键词*中对 **Trn** 的描述。而在第二和第三阶段之间没有掉转周期。

一次成功的读操作如图 12-9 所示。

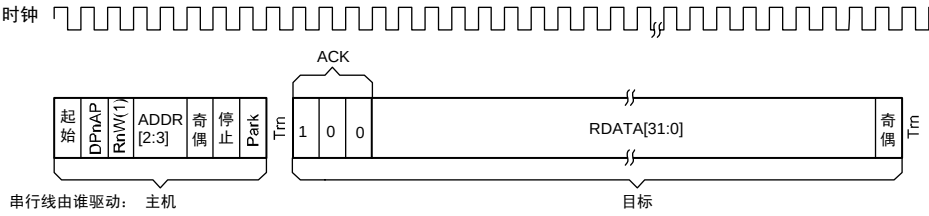


图 12-9 串行线调试时成功的读操作

读或写操作的 WAIT 响应

读或写数据包请求的 WAIT 响应由以下两个阶段组成：

- 一个 8 位的读或写数据包请求，从主机到目标
- 一个 3 位的 WAIT 应答，从目标到主机

默认情况下，这两个阶段之间以及第二阶段之后有一个时钟的掉转周期，详细信息见*阐述操作时的关键词*中对 **Trn** 的描述。

读或写数据包请求的 WAIT 响应如图 12-10 所示。

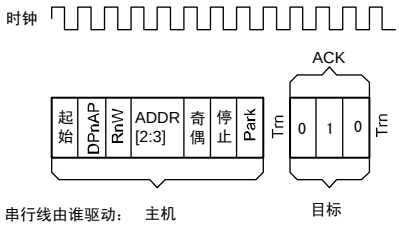


图 12-10 串行线调试时数据包请求的 WAIT 响应

注：

如果溢出检测使能，则在 WAIT 响应上必须有数据阶段。详细信息见 Sticky 溢出行为。

读或写操作的 FAULT 响应

读或写数据包请求的 FAULT 响应由以下两个阶段组成：

- 一个 8 位的读或写数据包请求，从主机到目标
- 一个 3 位的 FAULT 应答，从目标到主机

默认情况下，这两个阶段之间以及第二阶段之后有一个时钟的掉转周期，详细信息见*阐述操作时的关键词*中对 **Trn** 的描述。

读或写数据包请求的 FAULT 响应如图 12-11 所示。

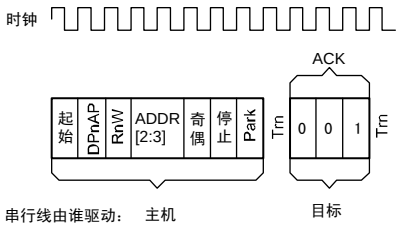


图 12-11 串行线调试时数据包请求的 FAULT 响应

注：

如果溢出检测使能，则在 FAULT 响应上必须有数据阶段。详细信息见 Sticky 溢出行为。

协议错误序列

当主机发出数据包请求但目标没有返回任何应答时，出现协议错误。这一过程由图 12-12 来阐述。

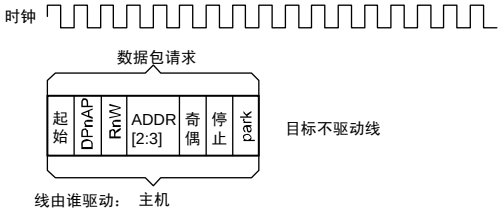


图 12-12 串行线调试时数据包请求之后的协议错误

12.3.4 协议描述

除了在*协议操作概述*部分描述的之外，本小节将提供其它有关 DAP 串行线调试操作的信息。

连接序列

连接序列用来确保热插拔不会导致意想不到的传输。为确保 SW-DP 能够与用来通知连接的报头准确同步，必须先发送 50 个时钟周期的串行线复位序列。在向 SW-DP 传输复位序列之后，主机接收到读 ID 寄存器的有效的报头请求，然后主机返回常规的响应和数据。让主机读 ID 代码寄存器来退出训练状态（training state），这样极大地确保了数据帧能够得到准确对齐。

OK 响应

当接收到来自调试主机的数据包请求时，SW-DP 必须立即响应。如果需要一次传输，并且 SW-DP 已准备好传输的数据阶段，则发出 OK 响应，由应答阶段为 b001 来指示。

注：

- 如在*协议操作概述*部分所示，在来自主机的数据包请求的结束和来自 SW-DP 目标的应答的开始之间，始终有一个掉转周期。默认掉转周期的精确值为一个串行时钟周期，详细信息见*阐述操作时的关键词*中对 **Trn** 的描述。
- 不管在串行 SWD 连接上数据传输的方向何时改变，都会有一个掉转周期。如果需改变数据传输方向的操作为立即操作，则该操作必须在掉转周期之后立即开始。

- 所有的 SWD 传输都是 LSB 在先。因此，由 b001 指示的 OK 响应在串行线上先出现 1，然后是 0，再 0，如图 12-8 和 12-9 所示。

如果主机请求一次写访问，则它必须在接收到来自目标的应答之后立即启动写传输。不管是写入 DP 还是 AP，操作都是一样的，只是 SW-DP 能够对 AP 写操作进行缓冲，见*访问端口写缓冲*部分。

如果主机请求对 DP 执行一次读访问，则 SW-DP 在应答之后立即发送读数据。因为在应答和读数据之间数据传输方向没有改变，所以这两个阶段之间没有掉转周期，如图 12-9 所示。

主机对 AP 的读访问“被邮寄（posted）”。这意味着访问结果在下一次传输时返回。如果下一次访问不是读 AP 操作，则必须插入读 DP RDBUFF 寄存器操作来获得被邮寄的结果，见*读缓冲寄存器，RDBUFF*。

当你必须执行连续的 AP 读访问时，你只需插入一次读 RDBUFF 寄存器操作：

- 在第一次 AP 读访问时，返回的读数据未定义。你必须丢弃这个结果。
- 如果你立即执行另一次 AP 读访问，则返回前一次 AP 读访问的结果。
- 针对任意次的 AP 读访问，你可以重复上述操作。
- 发送最后一次 AP 读数据包请求，返回上一次 AP 读结果。
- 然后你必须读 DP RDBUFF 寄存器来获得最后一次 AP 读结果。

READOK 标志的操作和使用

SW-DP CTRL/STAT 寄存器含有一个 READOK 标志，位于位[6]。该寄存器的描述见*控制/状态寄存器，CTRL/STAT*。

READOK 标志在每一次 AP 读访问以及每次 RDBUFF 读访问上都会更新。当 SW-DP 启动 AP 访问时，READOK 标志清零。当 SW-DP 目标针对读请求给出 OK 响应时，READOK 标志置位。

这意味着如果主机接收到针对 AP 或 RDBUFF 读请求的一个讹误的 ACK 响应，它能够检测读操作实际上是否正确结束。主机能够读 DP CTRL/STAT 寄存器来找出 READOK 标志的值。

- 如果该标志为 1，则读操作正确完成。主机能使用 RESEND 请求来获得读结果。见*读再发寄存器，RESEND(只用于 SW-DP)*。
- 如果该标志为 0，则读操作没有成功。主机必须重试原来的 AP 或 RDBUFF 读请求。

WAIT 响应

如果 SW-DP 不能立即处理来自调试器的请求，则发出 WAIT 响应。但不可以向下面的请求发送 WAIT 响应。SW-DP 必须始终能够立即处理以下 3 个请求：

- IDCODE 寄存器的读操作，见*标识代码寄存器，IDCODE*。
- CTRL/STAT 寄存器读操作，见*控制/状态寄存器，CTRL/STAT*。
- ABORT 寄存器写操作，见*中止寄存器，ABORT*。

对于除了上面列出的之外的请求，如果 SW-DP 不能处理该请求，则发出 WAIT 响应，没有数据阶段。出现下列情况时发送 WAIT 响应：

- 如果前一次 AP 或 DP 访问未完成

- 如果新的请求为 AP 读请求，并且前一次 AP 读访问的结果还不可用

注：

当溢出检测使能，WAIT 响应必须含有数据阶段。详细信息见 sticky 溢出行为。

通常，当调试器接收到 WAIT 响应时，它将重试相同的操作。这使它能尽可能快地处理数据。但如果已重试了几次，并且给出了时间允许较慢的互连和存储器系统来响应操作，如果仍不成功，调试器可能会写 ABORT 寄存器。该操作通知活动的 AP，告知它必须终止当前正在尝试的传输。AP 实现可能不能终止其 ASIC 接口上的传输。但在接收到 ABORT 请求时，AP 必须释放串行线调试接口。

在接收到 WAIT 响应之后写 ABORT 寄存器，这意味着此后调试器能够访问调试系统的其它部分。

FAULT 响应

SW-DP 不能对 IDCODE，CTRL/STAT 或 ABORT 寄存器访问发出 FAULT 响应。对于其它任何访问，如果 CTRL/STAT 寄存器中的任意 sticky 标志置位，则 SW-DP 发出 FAULT 响应，见*控制/状态寄存器，CTRL/STAT*。sticky 溢出标志的详细信息见 sticky 溢出行为。

FAULT 响应的使用能够使协议保持同步。调试器可以发送一个数据块，然后在数据块结束时检测 CTRL/STAT 寄存器。

对 ABORT 寄存器中的位执行写操作可将 Sticky 错误标志清零，见*中止寄存器，ABORT*。

Sticky 溢出行为

如果 SW-DP 在前一次处理还没有成功完成时接收到一个处理请求，则它产生 WAIT 响应。如果 CTRL/STAT 寄存器中的溢出检测使能，则该寄存器中的 STICKYORUN 标志置位。详细信息见*控制/状态寄存器，CTRL/STAT*。由于 sticky 标志置位，随后的处理产生 FAULT 响应。

当溢出检测使能时，请求数据阶段的 WAIT 和 FAULT 响应为：

- 如果该请求为读数据阶段的数据，则响应为结果不可预知。
- 如果该请求为写数据阶段，则响应为忽略请求。

图 12-13 显示了在溢出检测使能时，针对读操作的 WAIT 或 FAULT 响应。图 12-14 显示了在溢出检测使能时，针对写操作的响应。

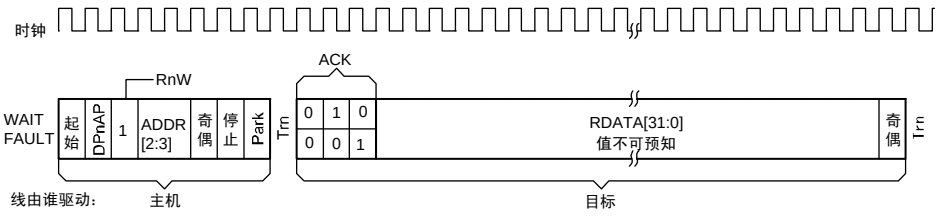


图 12-13 溢出检测使能时，串行线针对读操作的 WAIT 或 FAULT 响应

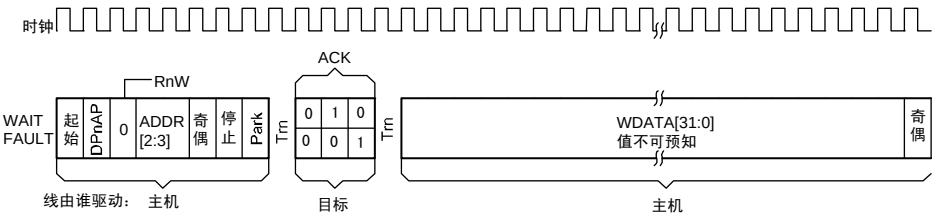


图 12-14 溢出检测使能时，串行线针对写操作的 WAIT 或 FAULT 响应

协议错误响应

如果 SW-DP 检测到数据包请求中的奇偶校验错误，则它不对该请求进行应答。

如果 SW-DP 没有对齐数据包中的帧并且将要返回读数据，则主机接收到其请求的无应答时必须往后退（back-off）。后退周期应与将完成的读处理所需的周期对应。在这之后，主机能够发出一个新的传输请求。此时，它必须读 IDCODE 寄存器，见*标识代码寄存器，IDCODE*。这个操作是该规范所要求的，因为一次成功的 IDCODE 寄存器读操作能够确认目标是可操作的。

如果第二次尝试没有回应，则调试器应强制将线复位，然后启动再训练（retraining）。这是必要的，因为 SW-DP 处于只响应调试请求的状态。如果导致最初的协议错误的传输为写操作，则你可以安全地假设没有发生写操作。如果最初的传输为读操作，则可以向 AP 发出读访问。虽然不一定成功，但可能性也是非常大的。因为读操作具有流水线属性并且 AP 能够实现 FIFO 功能。

访问端口写缓冲

SW-DP 具有写缓冲区，这使它能够在其它处理仍未完成时接收写操作。如果 DP 能够接受向其写缓冲区执行写操作的请求，则它对该写请求发出 OK 响应。这意味着写请求的 OK 响应不同于 DP ABORT 寄存器的写请求，它只表示写操作已被 DP 接收，不表示之前的所有处理已完成。

如果接受缓冲区写请求但随后放弃，则 CTRL/STAT 寄存器中的 WDATAERR 标志置位。见*控制/状态寄存器，CTRL/STAT*。出现下列情况时，缓冲的写访问被放弃：

- Sticky 标志由于前一次传输而置位
- DP 读 IDCODE 或 CTRL/STAT 寄存器请求。因为不允许 DP 停止这些寄存器的读请求，所以它必须：
  - — 立即执行 IDCODE 或 CTRL/STAT 寄存器访问
  - — 放弃任何缓冲的写操作，因为不能按顺序执行这些操作
- DP 写 ABORT 寄存器请求。同样是由于 DP 不能停止 ABORT 寄存器访问。

这意味着如果请求连续的 AP 写处理，则可能无法从检查 ACK 响应来确定哪个处理失败。但可以使用其它请求来找出哪个写处理是失败的。例如，如果你正在使用存储器访问端口的自动地址增加（AddrInc）特性，则你可以读传输地址寄存器来找出哪个是最后成功的写处理。详细信息见 *AHB-AP 传输地址寄存器和 AHB-AP 寄存器总结及描述*。

写缓冲区在执行后面的操作之前必须是空的：

- 任何 AP 读操作
- 读 IDCODE 或 CTRL/STAT 寄存器，写 ABORT 寄存器之外的任何 DP 操作

尝试上述操作将导致来自 DP 的 WAIT 响应，直到写缓冲区为空。

注：

如果推动验证或推动比较使能，则 AP 写处理转换为 AP 读处理。然后，象对其他 AP 读操作一样对他们进行处理。上述操作的详细信息见*推动比较和推动验证操作*。

如果必须执行 SW-DP 读 IDCODE 或 CTRL/STAT 寄存器操作，或在 AP 写序列之后立即执行 SW-DP 写 ABORT 寄存器操作，则必须先执行 SW-DP 能够停止的访问。这样，能够在执行 SW-DP 寄存器访问之前检验写缓冲区是否可用。

目标响应总结

表 12-7 总结了针对所有可能的调试器 DP 读操作请求，目标 SW-DP 的响应。

表 12-8 总结了针对所有可能的调试器 AP 读操作请求，目标 SW-DP 的响应。

表 12-9 总结了针对所有可能的调试器 DP 写操作请求（假定 WDATA 奇偶校验正确），目标 SW-DP 的响应。

表 12-10 总结了针对所有可能的调试器 AP 读操作请求（假定 WDATA 奇偶校验正确），目标 SW-DP 的响应。

上述两个表中没有显示的错误条件在*目标响应表格中没有包含的错误条件*部分描述。

表 12-7 DP 读处理请求的目标响应总结

ADDR[3:2]	Sticky 标志置位？	AP 准备好？	SW-DP(目标)响应	
			ACK	动作
b00	X	X	OK	以 IDCODE 值来响应
b01	X	X	OK	以 CRTL/STAT 或 WCR 值来响应
b10	否	是	OK	以来自前一次 AP 读操作的 RESEND 值来响应
b11	否	是	OK	以来自前一次 AP 读操作的 RDBUF 值来响应，并将 CTRL/STAT 寄存器中的 READOK 标志置位。
b10	否	否	WAIT	无数据阶段，除非溢出检测使能 [b]
b10	是	X	FAULT	无数据阶段，除非溢出检测使能 [b]
b11	否	否	WAIT	无数据阶段，除非溢出检测使能 [b]。将 CTRL/STAT 寄存器中的 READOK 标志清零。
b11	是	X	FAULT	无数据阶段，除非溢出检测使能 [b]。将 CTRL/STAT 寄存器中的 READOK 标志清零。

a　返回哪个值取决于 DP 中 SELECT 寄存器的 CTRLSEL 位的值。见 *AP 选择寄存器，SELECT*。

b　溢出检测使能时数据阶段的详细信息见 *sticky 溢出行为*。

表 12-8 AP 读处理请求的目标响应总结

ADDR[3:2]	Sticky 标志置位？	AP 准备好？	SW-DP(目标)响应	
			ACK	动作
bxx	否	是	OK	通常情况下[a]，返回来自前一次 AP 读操作的值，并将 CTRL/STAT 寄存器中的 READOK 标志置位。启动被寻址寄存器的 AP 读访问[c]。
bxx	否	否	WAIT	无数据阶段，除非溢出检测使能[b]。将 CTRL/STAT 寄存器中的 READOK 标志清零
bxx	是	X	FAULT	无数据阶段，除非溢出检测使能[b]。将 CTRL/STAT 寄存器中的 READOK 标志清零

a　如果推动验证或推动比较使能，则操作不可预知。

b　在 AP 读序列的第一个读请求上，数据阶段返回的值不可预知。

c　AP 寄存器由 A[3:2]的值结合 SELECT 寄存器中的 APBANKSEL 区域的值来寻址。见 *AP 选择寄存器，SELECT*。

d　溢出检测使能时数据阶段的详细信息见 *sticky 溢出行为*。

表 12-9 DP 写处理请求的目标响应总结

ADDR[3:2]	Sticky 标志置位？	AP 准备好？	SW-DP(目标)响应	
			ACK	动作
b00	X	X	OK	将 WDATA 的值写入 ABORT 寄存器
不是 b00	否	是[a]	OK	将 WDATA 的值写入由 ADDR[3:2]指示的 DP 寄存器。
不是 b00	否	否	WAIT	无数据阶段，除非溢出检测使能[b]
不是 b00	是	X	FAULT	无数据阶段，除非溢出检测使能[b]

a　当其它处理仍未完成时可以接受写操作。这些写操作随后可能被放弃。详细信息见*访问端口写缓冲*。

b　溢出检测使能时数据阶段的详细信息见 *sticky 溢出行为*。

表 12-10 AP 写处理请求的目标响应总结

ADDR[3:2]	Sticky 标志置位？	AP 准备好？	SW-DP(目标)响应	
			ACK	动作
bxx	否	是	OK	通常情况下[a]，将 WDATA 的值写入 ABORT 寄存器
bxx	否	否	WAIT	无数据阶段，除非溢出检测使能[d]
bxx	是	X	FAULT	无数据阶段，除非溢出检测使能[d]

a　当其它处理仍未完成时可以接受写操作。这些写操作随后可能被放弃。详细信息见*访问端口写缓冲*。

b　如果推动验证或推动比较使能，则对被寻址 AP 寄存器的写处理转换为读处理。并且该读处理返回的值与提供给 WDATA 的值进行比较，详细信息见推动比较和推动验证操作。对 AP 寄存器如何寻址的概述见本表格的脚注[c]。

c　AP 寄存器由 A[3:2]的值结合 SELECT 寄存器中的 APBANKSEL 区域的值来寻址。见 *AP 选择寄*

存器，SELECT。

d　溢出检测使能时数据阶段的详细信息见 *sticky 溢出行为*。

目标响应表格中没有包含的错误条件

有两个错误条件，没有在表 12-7 和 12-9 可能的操作请求中列出。

协议错误

如果操作请求中存在协议错误，则目标根本不响应该请求。这意味着在主机期待 ACK 响应时，发现线没有被驱动。

WDATA 的奇偶校验错误（只用于写操作）

DP 的 ACK 响应在执行奇偶校验之前发送出去，从表 12-9 中可以查看响应。在执行奇偶校验并失败时，CTRL/STAT 寄存器中的 WDATAERR 标志置位，见*控制/状态寄存器，CTRL/STAT*。

主机响应总结

调试器对 SW-DP 的每次访问都以操作请求开始。*目标响应总结*部分列出了来自调试器的所有可能的请求，并总结了 SW-DP 如何响应每次请求。

任何时候当调试器向 SW-DP 发出一个操作请求时，它期待接收到 3 位的应答，如表 12-7 的 ACK 栏所列。本部分总结了针对所有可能的情况，调试器如何响应该应答。

表 12-11 针对 SW-DP 应答的主机（调试器）响应总结

请求的操作	接收到的 ACK	主机响应	
		数据阶段	其它动作
R	OK	捕获来自目标的 RDATA 并检验有效的奇偶校验和协议	如果奇偶校验或协议错误并且不能将数据标记为无效，则可能必须重新发送原来的读请求 [a]。
W	OK	发送 DATA	这次传输的有效性在下一次访问时确认。
X	WAIT	无数据阶段，除非溢出检测使能 [b]	通常重复原来的操作请求。详细信息见 *WAIT 响应*。
X	FAULT	无数据阶段，除非溢出检测使能 [b]	能够发送新的报头，但只有访问地址为 b0x 的 DP 寄存器时才给出有效的响应。
X	无 ACK	考虑到可能的数据阶段，主机需退回（back off）	可尝试读 IDCODE 寄存器，另外将连线复位并启动再训练（retrain），见*协议错误响应*。
R	无效的 ACK	考虑到可能的数据阶段，主机需退回（back off）	可检查 CTRL/STAT 寄存器，察看发送的是否是 OK 响应。
W	无效的 ACK	退回去以确保目标不捕获作为 WDATA 的下一个报头。	重复写访问。如果发送的第一个响应为 OK，但没有被调试器承认为有效，则这次响应可能是 FAULT 响应。后面的写操作不会受到第一次读错的响应的影响。

a　主机调试器可能支持讹误的读操作，或者可能必须重试这次传输。

b　如果溢出检测使能，则必须要求有数据阶段。在读操作时，RDATA 的值不可预知，调试器必须捕获并放弃该数据。在写操作时，调试器必须发送 WDATA 数据包，但目标将忽略该数据包。

12.3.5 传输时序

本节描述串行线接口上的处理时序与 DAP 内部总线传输之间的相互作用。其中显示了目标给出 WAIT 应答的时间。

图 12-15 显示了发送 WAIT 应答在数据包长度上的影响。

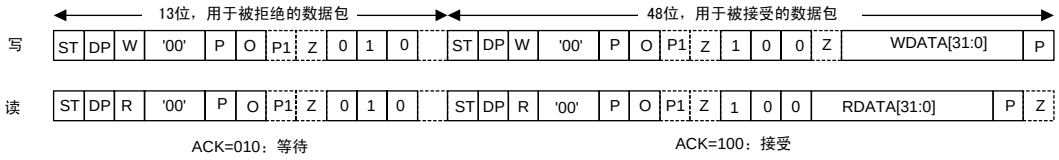


图 12-15 SW-DP 应答时序

对 AP 进行访问将导致在 DAP 内部总线上产生一次传输。这些传输含有地址阶段和数据阶段。如果该访问需要更多时间来处理事务，例如，如果它必须向系统总线执行 AHB 访问来读数据，则可将数据阶段延长。

表 12-12 显示了图 12-16～18 中用到的术语。

表 12-12 SW-DP 时序中用到的术语

术语	描述
W.APACC	写 DAP 访问端口寄存器
R.APACC	读 DAP 访问端口寄存器
xxPACC	对调试端口或访问端口寄存器执行读或写操作
WD[0]	第一次写数据包的数据
WD[-1]	前一次写数据包的数据。在图中时间帧之前发生的处理
WD[1]	第二次写数据包的数据
RD[0]	第一次读数据包的数据
RD[0]	第二从读数据包的数据

图 12-16 显示了写传输序列。它表示的是在前一次写传输 WD[0]完成时，能够被串行引擎接受的一次新传输 WD[1]。后面的任何传输都必须停止，直到第一次传输结束。

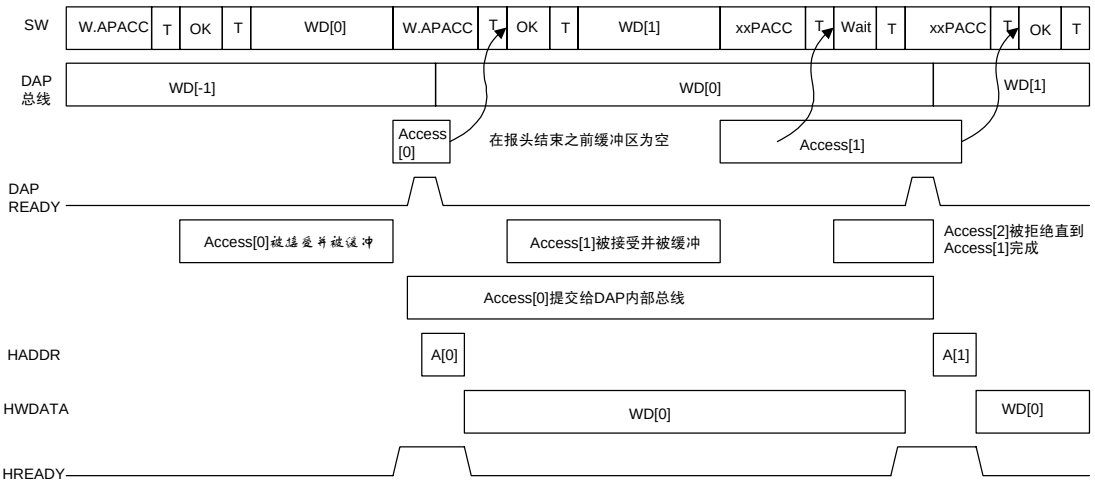


图 12-16 写操作时 SW-DP 到 DAP 总线的时序

图 12-17 显示了读传输序列。它表示的是一次访问端口读传输的有效载荷为前一次读请求提供数据。读传输只有在前一次传输没有完成时停止。因此，第一次读传输返回未定义的数据。为使协议时序保持可预测，第一次传输也必须要返回数据。

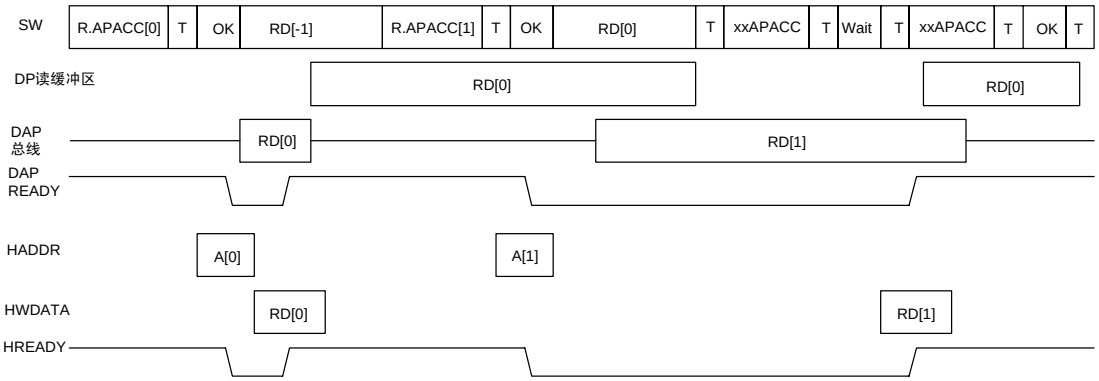


图 12-17 读操作时 SW-DP 到 DAP 总线的时序

图 12-18 显示了用 IDLE 周期分开的传输序列。图中显示，在任何传输之后始终将串行线退还给主机。

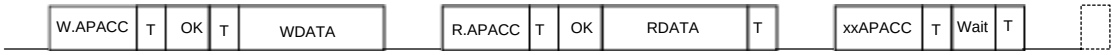


图 12-18 SW-DP 空闲时序

在数据包的最后一位之后，串行线可以是低电平或空闲，并可持续长于 1 个位的任意周期，来确保能够检测到背对背处理（back-to-back transaction）的起始位。

12.4 调试端口（DP）的通用特性

本节描述 SW-DP 和 JTAG-DP 都具有的特性。这些通用特性影响调试器利用 DAP 执行处理的方式。本节包含以下内容：

- Sticky 标志和 DP 错误响应

12.4.1 Sticky 标志和 DP 错误响应

在 SW-DP 和 JTAG-DP 内，sticky 标志用来表示错误条件，并报告推动比较和推动验证操作的结果。不同的 sticky 标志在以下部分描述：

- 读和写错误
- 溢出检测
- 协议错误，只用于 SW-DP
- 推动比较和推动验证操作

注：

Sticky 标志在置位时保持置位状态直到被明确地清零。尽管引起该标志置位的条件已不再应用，但该标志仍保持置位，直到调试器将其清零。将 sticky 标志清零的方法对于 JTAG-DP 和 SW-DP 来说是不同的，关于该标志如何清零的详细信息见*控制/状态寄存器，CTRL/STAT*。

错误可由 DP 本身返回，或者也可来自对调试寄存器的访问，例如，当处理器掉电时访

问调试寄存器将产生错误。

在调试端口内，错误由 DP 控制/状态寄存器（CTRL/STAT）中的 sticky 标志来标识。当存在错误标识时，完成当前的处理，并放弃其它的 APACC(AP 访问)处理，直到 sticky 标志清零。

DP 对错误条件的响应可能是：

- 立即发出错误响应。针对 SW-DP。
- 立即放弃所有处理，当作处理完成。针对 JTAG-DP。

这意味着在执行连续的 APACC 处理之后，调试器必须检验控制/状态寄存器，查看是否有错误发生。如果 sticky 标志置位，调试器能够将该标志清零，然后，如果有必要，启动更多的 APACC 处理来找出 sticky 标志置位的原因。因为是 sticky 标志，所以没必要在每次处理之后检查这些标志。调试器只需要定期地检查控制/状态寄存器，这样可减小错误检查的开销。

12.4.2 读和写错误

读或写错误可能在 DP 内发生，也可能来自正在调试的系统，作为响应 AP 请求的一次（AHB-AB）访问的结果。在任一种情况下，当检测到错误时，控制/状态寄存器中的 sticky Error 标志 STICKYERR 将置位。

如果调试器在调试器电源域掉电时产生 AP 处理请求，则也会发生读/写错误。

12.4.3 溢出检测

调试端口支持溢出检测模式。该模式使仿真器能够以长延迟，高吞吐量连接的方式发送命令块。发送这些命令块时需具有足够的线上延迟，来确保不出现溢出错误。但如果出现溢出错误，DP 将检测并通过将控制/状态寄存器中的标志置位来标记这些溢出错误。在溢出检测模式中，调试器必须在每个 APACC 处理序列之后，通过检验控制/状态寄存器中的 sticky 溢出标志来检查溢出错误。对于仿真器来说，没必要对溢出状态立即作出反应。

溢出检测模式通过将 DP 控制/状态寄存器中的溢出检测位 ORUNDETECT 置位来使能。当该位置位时，所有处理唯一允许的响应为：

- JTAG-DP 上为 OK/FAULT 响应
- SW-DP 上为 OK 响应

在溢出检测模式中，任意点的任何其它处理都被看作是错误，并使 DP 控制/状态寄存器中的 sticky 溢出标志 STICKYORUN 置位。Sticky 错误标志 STICKYERR 不置位。

调试器必须通过将 STICKYORUN 清零来恢复处理。

详细信息见*控制/状态寄存器，CTRL/STAT*。

sticky 溢出标志置位时，调试端口的操作由调试端口定义。

如果在之前的处理完成之前尝试新的处理并导致溢出错误，则第一次处理仍正常完成。其它 sticky 标志可能在第一次处理完成时置位。

如果在 STICKYORUN 置位时通过将 ORUNDETECT 标志清零来禁止溢出检测模式，则后面的 STICKYORUN 的值不可预知。为保持溢出检测模式，调试器必须：

- 检查控制/状态寄存器中 STICKYORUN 位的值
- 如果 STICKYORUN 位置位，则将其清零

- 通过将 ORUNDETECT 位清零来停止溢出检测模式

12.4.4 协议错误，只用于 SW-DP

虽然协议错误只能利用 SW-DP 才能检测到，但在这里描述这部分内容，是因为它们是 SW-DP 上的 sticky 标志错误处理机制的一部分。

在串行线调试接口上，只在线路电平出现错误时才出现协议错误。这些错误可由数据上的奇偶校验来检测。

如果 SW-DP 检测到消息报头的奇偶错误，则调试端口不响应该消息。调试器必须意识到这个可能性。如果它没有接收到针对该消息的响应，则调试器必须后退（back off）。然后，调试器必须请求读 IDCODE 寄存器来确保在重试原来的访问之前调试端口是可作出响应的。

如果 SW-DP 检测到写处理的数据阶段的奇偶错误，则它将控制/状态（CTRL/STAT）寄存器中的写数据错误标志 WDATAERR 置位。后面来自调试器的访问，除了 IDCODE，CTRL/STAT，ABORT 之外，都得到 FAULT 响应。CTRL/STAT 寄存器的详细信息见*控制/状态寄存器，CTRL/STAT*。

在接收到来自 SW-DP 的 FAULT 响应时，调试器必须读 CTRL/STAT 寄存器并检查 sticky 标志的值。WDATAERR 标志通过向中止寄存器的 WDERRCLR 写入 b1 来清零，见*中止寄存器，ABORT*。

12.4.5 推动比较和推动验证操作

SW-DP 和 JTAG-DP 调试端口都支持推动操作，在该操作中，AP 写处理的值在 DP 级用来与目标读处理的值进行比较：

- 调试器执行 AP 写处理
- DP 执行来自 AP 的读操作

DP 将这两个值进行比较并根据比较结果更新 DP 控制/状态寄存器中的 sticky 比较标志，STICKYCMP。

- 如果这两个值相等，推动比较将 STICKCMP 设为 b1
- 如果这两个值不相等，推动验证将 STICKCMP 设为 b1

在检测到一次有效的比较操作时，任何时候只要 STICKYCMP 位置位，所有未完成的重复处理都被取消。

详细信息见*控制/状态寄存器，CTRL/STAT*。

DP 含有一个字节通道屏蔽，能够将比较操作限制在字中的特定字节。该屏蔽操作使用控制/状态寄存器中的 MASKLANE 位来设置。有关屏蔽的详细信息见*推动比较和推动验证操作中的 MASKLANE 和位屏蔽*。

图 12-19 简要描述了推动操作。

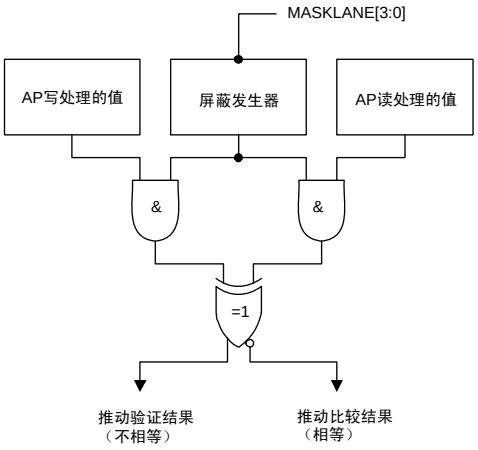


图 12-19 推动操作概述

推动操作可提高性能，在该操作中，写操作可能比读操作要快。它们用作线上测试的一部分。例如 Flash ROM 编程以及通信监控。推动操作使用 DP 控制/状态寄存器中的处理模式位 TRNMODE 来使能，见*控制/状态寄存器，CTRL/STAT*。

考虑到在理解推动操作如何执行时，通过对特定 AP 上的推动操作进行描述会更易懂，因此作如下描述：在 AHB-AP 上，如果你向数据读/写寄存器（DRW）或其中一个分组数据寄存器（BD0~BD3）执行 AP 写处理，则不管是推动比较还是推动验证：

- DP 将来自 AP 写处理的数据保存在推动比较逻辑中
- AP 读取来自由 AP 传输地址寄存器（TAR）指示的地址中的值
- 该读操作返回的值与保存在推动比较逻辑中的值进行比较，并根据比较结果将 STICKYCMP 置位。比较操作根据 MASKLANE 位进行屏蔽。详细信息见*控制/状态寄存器，CTRL/STAT*。

如上所述，在推动比较或推动验证有效时执行 AP 写处理，实际的 AP 访问为读操作，而不是写操作。

注：

在推动比较或推动验证有效时执行 AP 读处理将导致不可预知的行为。

在 SW-DP 上，推动比较或推动验证有效时执行 AP 读处理将返回一个值。这意味着线路电平协议保持一致。但返回的值不可预知，且读操作具有不可预知的副作用。

推动验证操作在 AHB-AP 上的使用举例

你可以使用推动验证操作来验证系统存储器的内容。

- 确保 AHB-AP 控制/状态字（CSW）设置为每次访问之后将传输地址寄存器（TAP）的值加 1。见*控制/状态寄存器，CTRL/STAT*。
- 写传输地址寄存器（TAR）来指示即将进行验证的调试寄存器区域的起始地址，见*AHB-AP 传输地址寄存器*。
- 连续写入预期的作为 AP 处理的值，在每次写处理时，DP 发出 AP 读访问，将读访问的值与 AP 写处理中提供的值进行比较，如果两个值不相等，则将 CTRL/STAT 寄存器中的 STICKYCMP 位置位，见*控制/状态寄存器，CTRL/STAT*。

每次处理时 TAR 的值加 1。

这样，将连续提供的值与 AP 单元的内容进行比较，并在两个值不相等时将 STICKYCMP 置位。

推动查找操作在 AHB-AP 上的使用举例

你可以使用推动查找来搜索系统存储器中的一个特定的字。如果使用带有字节通道屏蔽的推动查找功能，则你可以搜索一个或多个字节。

- 确保 AHB-AP 控制/状态字（CSW）设置为每次访问之后将传输地址寄存器（TAP）的值加 1。见*控制/状态寄存器，CTRL/STAT*。
- 写传输地址寄存器（TAR）来指示即将进行查找的调试寄存器区域的起始地址，见 *AHB-AP 传输地址寄存器*。
- 写入进行查找的值，作为一次 AP 写处理。DP 反复地读由 TAP 指示的单元。在每次 DP 读操作上：
  - 返回的值与 AP 写处理中提供的值进行比较。如果相等，STICKYCMP 标志置位。
  - TAP 的值加 1

该操作继续进行，直到 STICKYCMP 置位，或用 ABORT 来终止查找。

你也可以使用没有地址加 1 功能的推动查找来查询单个的单元，例如来测试一次操作完成时即将置位的标志。

12.5 调试端口的编程模型

每个 Cortex-M3 系统都包含以下两种调试端口中的一种或两种都有：

- JTAG 调试端口（JTAG-DP）
- 串行线调试端口（SW-DP）

本节内容包括：

- JTAG-DP 寄存器。含有对 JTAG-DP 寄存器的总结。
- SW-DP 寄存器。含有对 SW-DP 寄存器的总结。
- 调试端口（DP）的寄存器描述。含有对 DP 寄存器的详细描述，并描述了 SW-DP 和 JTAG-DP 寄存器之间的实现的不同。

12.5.1 JTAG-DP 寄存器

所访问的 JTAG-DP 寄存器取决于：

- 针对 DAP 访问的指令寄存器的值
- DAP 访问的地址区

详细信息见*访问 JTAG-DP 寄存器*部分。

表 12-13 显示了 JTAG-DP 的寄存器映射。

表 12-13 JTAG-DP 寄存器映射

IR 内容	描述	地址	访问	参考	备注
IDCODE	ID 代码寄存器	-[a]	RO	标识代码寄存器，IDCODE	-
DPACC	-	0x0	RAZ/WI	-	保留，读作 0，写操作忽略。
DPACC	DP 控制/状态寄存器	0x4	R/W	控制/状态寄存器，CTRL/STAT	-
DPACC	选择寄存器	0x8	R/W	AP 选择寄存器，SELECT	-
DPACC	读缓冲	0xC	RAZ/WI	读缓冲，RDBUFF	-
ABORT	DAP 中止寄存器	0x0	WO[b]	中止寄存器，ABORT	-
ABORT	-	0x4-0xC	-	-	-b

a　没有与 IDCODE 访问相关的地址。见 *访问 JTAG-DP 寄存器*。

b　ABORT 扫描链上读出的值不可预知。访问地址区没有设置为 0x0 的 ABORT 扫描链时，结果不可预知。

访问 JTAG-DP 寄存器

只有当用于 DAP 访问的指令寄存器（IR）包含 IDCODE，DPACC，ABORT 指令时才能访问 JTAG-DP 寄存器。每条指令的寄存器访问的详细描述如下：

IDCODE　IDCODE 扫描链没有地址区，访问 IDCODE 寄存器。

DPACC　DPACC 扫描链访问地址为 0x0～0xC 的寄存器。

ABORT　对于地址为 0x0 的写访问，ABORT 扫描链访问 ABORT 寄存器。

对于地址为 0x0 的读访问，以及地址为 0x4～0xC 的任何访问，ABORT 扫描链的行为都是不可预知的。

12.5.2 SW-DP 寄存器

对于 SW-DP 上的大多数寄存器地址，读和写访问时将寻址到不同的寄存器。另外，选择寄存器中的 CTRLSEL 位用来改变访问地址 0b01 时的寄存器。

表 12-14 显示了 SW-DP 的寄存器映射。

表 12-14 SW-DP 寄存器映射

地址	CTRLSEL[a]	描述	访问[b]	参考
b00	X	ID 代码寄存器	R	标识代码寄存器，IDCODE
		中止寄存器	W	中止寄存器，ABORT
b01	b0	DP 控制/状态寄存器	R/W	控制/状态寄存器，CTRL/STAT
	b1	线控制寄存器	R/W	线控制寄存器，WCR(只用于 SW-DP)
b10	X	读再发寄存器	R	读再发寄存器，RESEND（只用于 SW-DP）
		选择寄存器	W	AP 选择寄存器，SELECT
b11	X	读缓冲	R	读缓冲，RDBUFF
		-	-	-

a　SELECT 寄存器中的 CTRLSEL 位，见 *AP 选择寄存器，SELECT*。

b　“访问”列中的访问权指的是 SWD 协议对给定地址执行的是读还是写访问。

12.5.3 调试端口（DP）的寄存器描述

本节详细描述了所有的 DP 寄存器。每个描述都声明该寄存器是在 JTAG-DP 还是在 SW-DP 中实现，以及在实现中的任何不同。

中止寄存器，ABORT

在所有 DP 实现上都始终具有中止寄存器。其主要用途是用来强制 DAP 中止，在 SW-DP 上，它还用来清除错误和 sticky 标志条件。

JTAG-DP　当指令寄存器（IR）含有 ABORT 时，JTAG-DP 的中止寄存器位于地址 0x0。

SW-DP　当 DPnAP 位为 1 时，SW-DP 的中止寄存器在写操作时位于地址 0b00，见 *阐述操作时的关键词*部分。对中止寄存器的访问不会受到选择寄存器中的 CTRLSEL 位的影响。

该寄存器是：

- 只写寄存器
- 始终可访问，如果接收到有效的处理则返回 OK 响应。
- 中止寄存器的访问始终在第一次尝试时完成。

图 12-20 显示了该寄存器的位分配。

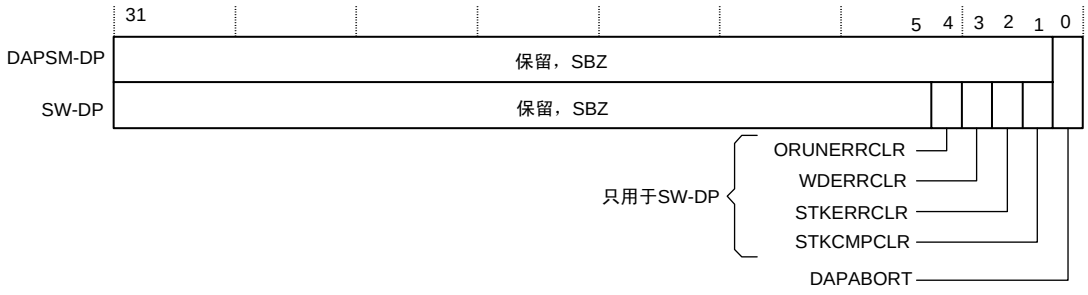


图 12-20 中止寄存器的位分配

表 12-15 列出了中止寄存器的位功能。

表 12-15 中止寄存器的位功能

位	功能	描述
[31:5]	-	保留，SBZ
[4][a]	ORUNERRCLR [a]	向该位写 1，将 STICKYORUN 溢出错误标志清零 [b]
[3] [a]	WDERRCLR [a]	向该位写 1，将 WDATAERR 写数据错误标志清零 [b]
[2] [a]	STKERRCLR [a]	向该位写 1，将 STICKYERR sticky 错误标志清零 [b]
[1] [a]	STKCMPCLR [a]	向该位写 1，将 STICKYCMP sticky 比较标志清零 [b]
[0]	DAPABORT	向该位写 1 来产生 DAP 中止。它将中止当前的 AP 处理。该操作只有当处理器在扩展周期上接收到 WAIT 响应时才执行。

a　只在 SW-DP 上实现，在 JTAG-DP 上该位保留，SBZ。

b　在控制/状态寄存器中。

DP 中止

向中止寄存器的位[0]写 1 产生 DP 中止，使得当前的 AP 处理停止。如果处理计数器有效，该操作也会使它停止。

从软件角度来看，这是一个重大的操作。它放弃所有未完成和挂起的处理，并让 AP 保持在未知状态。但在 SW-DP 上，sticky 错误位没有被清零。

只有在无计可施时才使用该功能。此时，调试主机软件让目标硬件在一个预期的时间内保持停止。停止的目标硬件由 WAIT 响应来指示。

在请求 DP 中止之后，DP 能够接受新的处理。但对被中止的 AP 进行 AP 访问将导致更多的 WAIT 响应。其它 AP 可访问，但系统的状态可能使调试无法继续进行。

注意：

JTAG-DP 上的中止寄存器：

- 位[0]DAPABORT 为唯一已定义的位。
- 向该寄存器写入除 0x00000001 之外的任意值时，结果不可预知。

清除错误和 sticky 比较标志，只用于 SW-DP

当与 SW-DP 相连的调试器检查控制/状态寄存器，并发现一个错误标志或 sticky 比较标志置位时，调试器必须写中止寄存器来将错误或 sticky 比较标志清零。表 12-15 列出了控制/状态寄存器中可能置位的标志，并显示了中止寄存器中的哪个位用来清除哪个标志。如果有必要，你可以使用中止寄存器的一次写操作来将多个标志清零。

在将标志清零之后，你可能必须访问 DP 和 AP 寄存器来找出引起标志置位的原因。通常：

- 对于 STICKYCMP 或 STICKERR 标志，你必须找出访问哪个单元时引起了标志置位。
- 对于 WDATAERR 标志，在将它清零之后，你必须再发送被破坏的数据。
- 对于 STICKYORUN 标志，你必须找出哪个 DP 或 AP 处理引起了溢出。然后，你必须从该点重复处理。

标识代码寄存器，IDCODE

所有 DP 实现上都始终具有标识代码寄存器。它提供有关 ARM 调试接口的标识信息。

JTAG-DP　该寄存器使用自己的扫描链来访问。

SW-DP　当 DPnAP 位为 1 时，该寄存器在读操作时位于地址 0b00。对标识代码寄存器的访问不受到选择寄存器中的 CTRLSEL 位的影响。

该寄存器是：

- 只读寄存器
- 始终可访问

图 12-21 显示了该寄存器的位分配

图 12-21 标识代码寄存器的位分配

表 12-16 列出了标识代码寄存器的位功能。

表 12-16 标识寄存器的位功能

位	功能	描述
[31:28]	Version	版本代码。该区域的含义由实现定义。
[27:12]	PARTNO	DP 的元件型号。该值由调试端口的设计人员提供，并且不可以改变。当前 ARM 设计的 DP 的 PARTNO 值如下： **JTAG-DP**　0xBA00 **SW-DP**　0xBA10
[11:1]	MANUFACTURER	JEDEC 厂商 ID，这个 11 位的 JEDEC 代码用来标识器件的厂商，见 *JEDEC 厂商 ID*。该区域的 ARM 默认值如图 12-21 所示，为 0x23B。
[0]	-	始终为 0b1

JEDEC 厂商 ID

该代码也被看作 JEP-106 厂商标识代码，它能够再分为两个区域，如表 12-17 所示。

表 12-17 JEDEC JEP-106 厂商 ID 代码，带有 ARM 公司注册的值

JEP-106 区域	位 [a]	ARM 公司注册的值
扩展代码	4 位，[11:8]	b0100，0x4
标识代码	7 位，[7:1]	b0111011，0x3B

a　区域宽度，以位来表示，与标识代码寄存器中的位对应。

JDEC 代码由 JEDEC 固体技术协会（JEDEC Solid State Technology Association）来分配，见 *JEP106M，标准厂商标识代码*。

控制/状态寄存器，CTRL/STAT

所有 DP 实现上都始终具有控制/状态寄存器。它提供对 DP 的控制以及有关 DP 的状态信息。

JTAG-DP　当指令寄存器（IR）包含 DPACC 时，该寄存器位于地址 0x4。

SW-DP　当 DPnAP 位为 1 时，该寄存器在读和写操作时都位于地址 0b01，并且选择寄存器中的 CTRLSEL 位设置为 b0。有关 CTRLSEL 位的详细信息见 *AP 选择寄存器，SELECT*。

它是一个读写寄存器，寄存器中的位具有不同的访问权。是否支持该寄存器中的某些区域是由实现定义的。表 12-18 显示在所有的实现中哪些区域是必需的。

图 12-22 显示了寄存器的位分配。

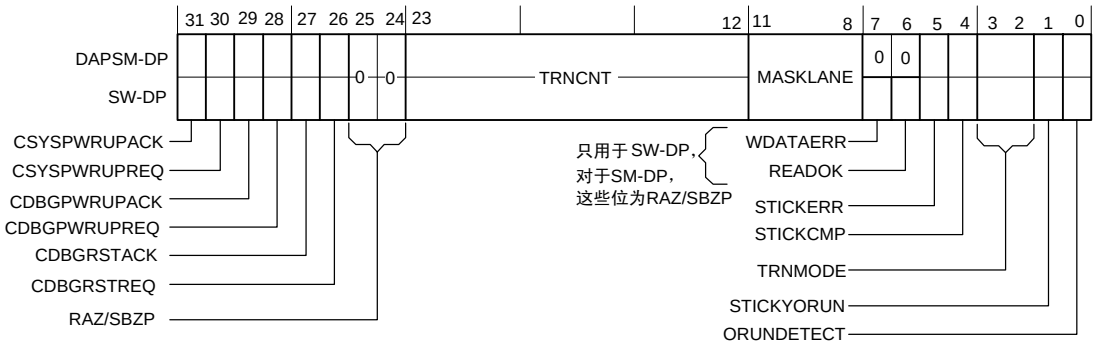


图 12-22 控制/状态寄存器的位分配

表 12-18 列出了控制/状态寄存器的位功能。

表 12-18 控制/状态寄存器的位功能

位	访问	功能	描述	是否必需
[31]	RO	CSYSPWRUPACK	系统掉电确认	否
[30]	R/W	CSYSPWRUPREQ	系统上电请求 复位之后，该位为低电平（0）。	否
[29]	RO	CDBGPWRUPACK	调试上电确认	否
[28]	R/W	CDBGPWRUPREQ	调试上电请求 复位之后，该位为低电平（0）。	否
[27]	RO	CDBGRSTACK	调试复位确认	是
[26]	R/W	CDBGRSTREQ	调试复位请求 复位之后，该位为低电平（0）	是
[25:24]	-	-	保留，RAZ/SBZP	-
[21:12]	R/W	TRNCNT	处理计数器 复位之后，该区域的值不可预知。	是
[11:8]	R/W	MASKLANE	表示推动比较和推动验证操作中被屏蔽的字节。见*推动比较和推动验证操作的 MASKLANE 和位屏蔽*部分。 复位之后，该区域的值不可预知	是
[7]	RO[b]	WDATAERR[a]	如果出现写数据错误，该位置位。它在以下情况下置位： ● 写操作的数据阶段出现数据帧的奇偶校验错误 ● 已经被 DP 接受的写操作然后在没有提交给 AP 的情况下放弃。 该位可通过向中止寄存器的 WDERRCLR 写 1 来清零。见*中止寄存器，ABORT*。 复位之后，该位为低电平（0）。	是 [a]

续表 12-18

位	访问	功能	描述	是否必需
[6]	RO[b]	READOK[a]	如果前面的 AP 或 RDBUFF 得到 OK 响应，则该位置位。如果不是 OK 响应，该位清零。该标志始终表示对上一次 AP 读访问的响应。 复位之后，该位为低电平（0）。	是 [a]
[5]	RO[b]	STICKYERR	如果 AP 处理返回一个错误，则该位置位。为了将其清零： 在 **JTAG-DP** 上：向该寄存器的这个位写 1。 在 **SW-DP** 上：向中止寄存器的 STKERRCLR 写 1，见*中止寄存器，ABORT*。 复位之后，该位为低电平（0）。	是
[4]	RO[b]	STICKYCMP	如果推动比较或推动验证操作时出现相等的情况，该位置位。为了将其清零： 在 **JTAG-DP** 上：向该寄存器的这个位写 1。 在 **SW-DP** 上：向中止寄存器的 STKCMPCLR 写 1，见*中止寄存器，ABORT*。 复位之后，该位为低电平（0）。	是
[3:2]	R/W	TRNMODE	该区域用来设置 AP 操作的传输模式，见*传输模式（TRNMODE），位[3:2]*。 复位之后，该位为低电平（0）。	是
[1]	RO[b]	STICKYORUN	如果溢出检测使能（见该寄存器的位[0]），则在出现溢出时该位置位。为了将其清零： 在 **JTAG-DP** 上：向该寄存器的这个位写 1。 在 **SW-DP** 上：向中止寄存器的 ORUNERRCLR 写 1，见*中止寄存器，ABORT*。 复位之后，该位为低电平（0）。	是
[0]	R/W	ORUNDETECT	该位设为 1 来使能溢出检测。 复位之后，该位为低电平（0）。	是

a　只在 SW-DP 上实现。在 JTAG-DP 上，该位保留，RAZ/SBZP。

b　在 SW-DP 上为 RO，在 JTAG-DP 上，该位能够执行常规的读操作，也可以写 1 将其清零。

推动比较和推动验证操作的 MASKLANE 和位屏蔽

只有在传输模式设置为推动验证或推动比较操作时，CTRL/STAT 寄存器的位[11:8]MASKLEAN 才有用。见*传输模式（TRNMODE），位[3:2]*。

在推动操作中，AP 写处理中提供的字与目标 AP 地址的当前值进行比较。MASKLANE 允许用户指定该比较操作只针对字中的特定字节。MASKLANE 中的每个位与 AP 值的一个字节对应。因此，一个位控制比较操作的一个字节通道。

表 12-19 显示了 MASKLANE 的位对比较屏蔽操作的控制。

表 12-19 MASKLANE 对推动比较操作的控制

MASKLANE[a]	含义	用于比较操作的屏蔽结果 [b]
b1xxx	比较中包含字节通道 3	0xFF- - - - - -
bx1xx	比较中包含字节通道 2	0x- - FF- - - -
bxx1x	比较中包含字节通道 1	0x- - - - FF- -
bxxx1	比较中包含字节通道 0	0x- - - - - - FF

a　CTRL/STAT 寄存器的位[11:8]

b　屏蔽结果显示为- -的字节由 MASKLANE 的其它位决定。

如果 MASKLANE 设为 b0000 或 b1111，则对整个字进行比较操作。此时，屏蔽结果为 0xFFFFFFFF。

传输模式（TRNMODE，位[3:2]）

该区域用来设置 AP 操作的传输模式。表 12-20 列出了传输模式允许的值及含义。

表 12-20 传输模式 TRNMODE 的位定义

TRNMODE[a]	AP 传输模式
b00	常规操作
b01	推动验证操作
b10	推动比较操作
b11	保留

a　CTRL/STAT 寄存器的位[3:2]

在常规操作中，AP 处理被传送到 AP 中进行处理。

在推动验证和推动比较操作中，DP 将 AP 处理中提供的值与目标 AP 地址中保存的值进行比较。

AP 选择寄存器，SELECT

所有 DP 实现上都始终具有 AP 选择寄存器。其主要用途是用来选择当前的访问端口（AP）以及该 AP 中有效的 4 字寄存器窗口。在 SW-DP 上，它还用来选择调试端口的地址分组。

JTAG-DP　当指令寄存器（IR）含有 DPACC 时，该寄存器位于地址 0x8，是一个读写寄存器。

SW-DP　当 DPnAP 位为 1 时，该寄存器在写操作时位于地址 0b10，是一个只写寄存器。对 AP 选择寄存器进行访问不会受到 CTRSEL 位的影响。

图 12-23 显示了该寄存器的位分配。

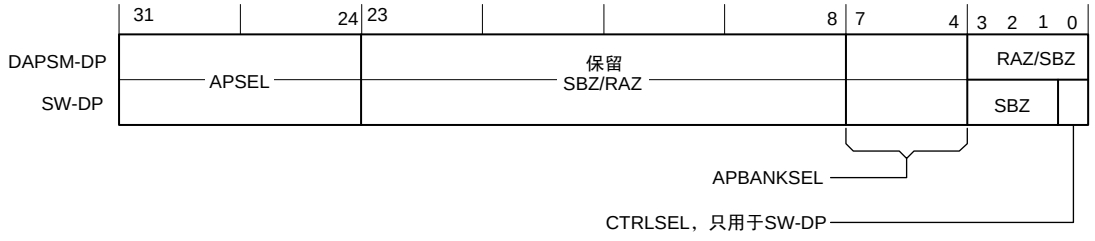


图 12-23 AP 选择寄存器的位分配

表 12-21 列出了 AP 选择寄存器的位功能。

表 12-21 AP 选择寄存器的位功能

位	功能	描述
[31:24]	APSEL	选择当前 AP 该区域的复位值不可预知。[a]
[23:8]	-	保留，SBZ/RAZ[a]
[7:4]	APBANKSEL	在当前的 AP 上选择有效的 4 字寄存器窗口。 该区域的复位值不可预知。[a]
[3:1]	-	保留，SBZ/RAZ[a]
[0][a]	CTRLSEL[b]	SW-DP 调试端口的地址组选择，见 *CTRL/STAT，只用于 SW-DP*。 复位之后，该位为 b0。AP 选择寄存器是只写的，因此，不能读出这个值。

a　在 SW-DP 上，AP 选择寄存器为只写的，因此，不能读出这个区域的值。

b　只在 SW-DP 上实现。在 JTAG-DP 上，该位保留，SBZ/RAZ。

如果将 APSEL 设置为不存在的 AP，则读操作时所有 AP 处理返回零，写操作忽略。

注：

每个 ARM 调试接口必须至少包含一个 AP。

CTRLSEL，只用于 SW-DP

AP 选择寄存器的位[0]，CTRLSEL 用来控制在 SW-DP 的地址 b01 处选择哪个 DP 寄存器。表 12-22 显示了 CTRLSEL 的不同值的含义。

表 12-22 CTRLSEL 位定义

CTRLSEL[a]	地址 b01 的 DP 寄存器
0	CTRL/STAT，见*控制/状态寄存器，CTRL/STAT*。
1	WCR，见*线控制寄存器，WCR（只用于 SW-DP）*。

a　SELECT 寄存器的位[0]

读缓冲，RDBUFF

所有 DP 实现上都始终具有 32 位的读缓冲寄存器。但在 JTAG 和 SW 调试端口中，读缓冲实现有着非常大的不同。

JTAG-DP　当指令寄存器（IR）含有 DPACC 时，读缓冲寄存器位于地址 0xC，读操作时结果为 0，写操作忽略（RAZ/WI）。

SW-DP　当 DPnAP 位为 1 时，读缓冲寄存器在读操作上位于地址 0b11，并且为只读寄存器。对读缓冲的访问不受到 SELECT 寄存器的 CTRLSEL 位的影响。

JTAG-DP 上的读缓冲实现及使用

在 JTAG-DP 上，读缓冲的读操作结果始终为 0，读缓冲地址的写操作忽略。

读缓冲在结构上被定义为提供没有任何副作用的 DP 读操作。这意味着调试器能够在操

作序列的末尾插入一个读缓冲的 DP 读操作，来返回最后一次传输的读结果以及 ACK 的值。

SW-DP 上的读缓冲实现及使用

在 SW-DP 上，对读缓冲执行读操作能够实现在不启动一次新的 AP 处理的情况下，从 AP 中捕获数据作为前一次读操作的结果。这意味着对读缓冲执行读操作能够返回上一次 AP 读访问的结果，而无需产生一次新的 AP 访问。

在完成读缓冲的读操作之后，读缓冲中的内容不再有效。读缓冲的第二次读操作结果不可预知。

如果你需要从 AP 寄存器读操作中获得值，则该读操作后面必须执行以下两个选项之一：

- 另一次 AP 寄存器读操作，如果想确保这次读操作没有副作用，你可以读控制/状态寄存器，CSW。
- DP 读缓冲的读操作

第二次访问针对的是 AP 还是 DP 取决于你使用的是哪个选项，这次访问将延长到原来的 AP 读操作的结果可用。

线控制寄存器，WCR（只用于 SW-DP）

所有 SW-DP 实现上都始终具有线控制寄存器。其用途是用来选择与 SW-DP 相连的物理串行端口的操作模式。

当选择寄存器中的 CTRLSEL 位设为 1 时，该读写寄存器在读和写操作上位于地址 0b01。有关 CTRLSEL 位的详细信息见 *AP 选择寄存器，SELECT*。

注：

当 CTRLSEL 位设为 1 使能对 WCR 的访问时，DP 控制/状态寄存器不可访问。

线控制寄存器的众多特性都是实现自定义的。

图 12-24 显示了该寄存器的位分配。

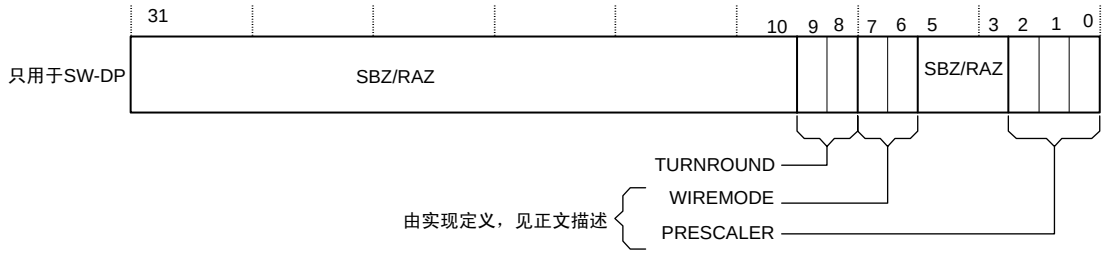


图 12-24 线控制寄存器（只用于 SW-DP）的位分配。

表 12-23 列出了线控制寄存器的位功能。

表 12-23 线控制寄存器（只用于 SW-DP）的位功能

位	功能	描述
[31:10]	-	保留，SBZ/RAZ。
[9:8]	TURNROUND	掉转三态周期，见*掉转三态周期，TURNROUND，位[9:8]*。 复位之后，该区域为 b00。
[7:6]	WIREMODE	表示与 DP 相连的串行线的操作模式。见*线操作模式，WIREMODE，位[7:6]*。 复位之后，该区域为 b01。
[5:3]	-	保留，SBZ/RAZ。
[2:0]	保留	-

掉转三态周期，TURNROUND，位[9:8]

TURNROUND 定义了掉转三态周期。该掉转周期允许在使用高采样时钟频率时，焊盘上出现延迟。表 12-24 列出了掉转周期允许的值及其含义。

表 12-24 掉转周期 TURNROUND 的位定义

TURNROUND[a]	掉转周期
b00	1 个采样周期
b01	2 个采样周期
b10	3 个采样周期
b11	4 个采样周期

a　WCR 寄存器的位[9:8]的值。

线操作模式，WIREMODE，位[7:6]

线操作模式将 SW-DP 确定为只在同步模式下操作。

线操作模式是必需的，表 12-25 列出了其允许的值及含义。

表 12-25 线操作模式 WIREMODE 的位定义

WIREMODE[a]	线操作模式
b00	保留
b01	同步（无过采样（oversampling））
b1x	保留

a　WCR 寄存器的位[7:6]的值。

读再发寄存器，RESEND（只用于 SW-DP）

所有 SW-DP 实现上都始终具有读再发寄存器。其用途是在无需重复原来的 AP 传输的情况下，使读数据能够从被破坏的调试器传输中恢复。

它是一个 32 位只读寄存器，读操作时位于地址 0b10。对读再发寄存器的访问不受到 SELECT 寄存器中的 CTRLSEL 位的影响。

对 RESEND 寄存器执行读操作并没有从 AP 中捕获到新的数据。它返回一个值，该值是上次 AP 读或 DP RDBUFF 读返回的值。

对 RESEND 寄存器执行读操作使读数据能够从被破坏的传输中恢复，而无需再次发送原来的读请求或产生新的 DAP 或系统级访问。

RESEND 寄存器能够执行多次访问。在向 DP RDBUFF 寄存器或 AP 寄存器执行一次新的访问之前，它始终返回相同的值。

第13章 跟踪端口的接口单元

本章介绍了跟踪端口的接口单元（TPIU）。包括以下小节：

- 关于跟踪端口的接口单元
- TPIU 寄存器

13.1 关于跟踪端口的接口单元

跟踪端口的接口单元充当嵌入式跟踪宏单元（ETM）和仪表跟踪宏单元（ITM）与跟踪端口分析仪（TPA）之间的桥，跟踪数据从 ETM 和 ITM 流入 TPIU，并从 TPIU 流向跟踪端口，ETM 和 ITM 具有独立的 ID，并在需要的地方将 ID 封包，然后由跟踪端口分析仪（TPA）捕获。

TPIU 专为低成本调试而设计。它是 CoreSight TPIU 的特殊版本，并且如果系统还需要使用 CoreSight TPIU 的其他特性，它可以使用 CoreSight 组件来代替 TPIU。

TPIU 有两种配置：

- 支持 ITM 调试跟踪
- 支持 ITM 和 ETM 调试跟踪

注：

如果 Cortex-M3 系统使用可选的 ETM 组件，那么用户必须使用支持 ITM 和 ETM 调试跟踪的 TPIU 配置。详见第 15 章“嵌入式跟踪宏单元”。

13.1.1 TPIU 方框图

图 13-1 和图 13-2 显示了两种配置下 TPIU 组件的布局。

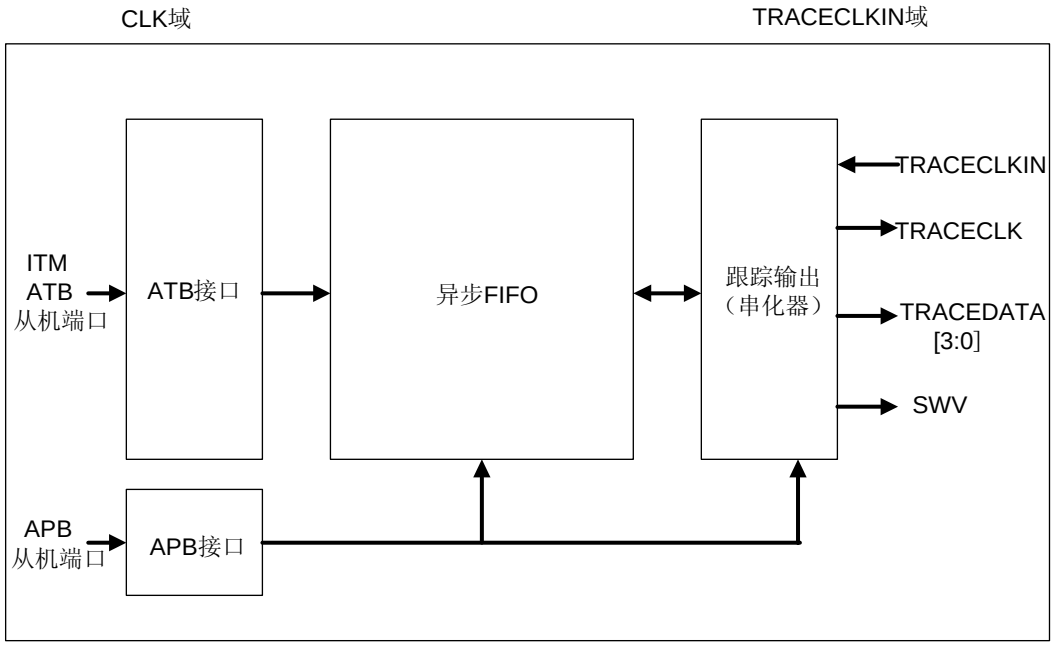


图 13-1　TPIU 的方框图（无 ETM 版本）

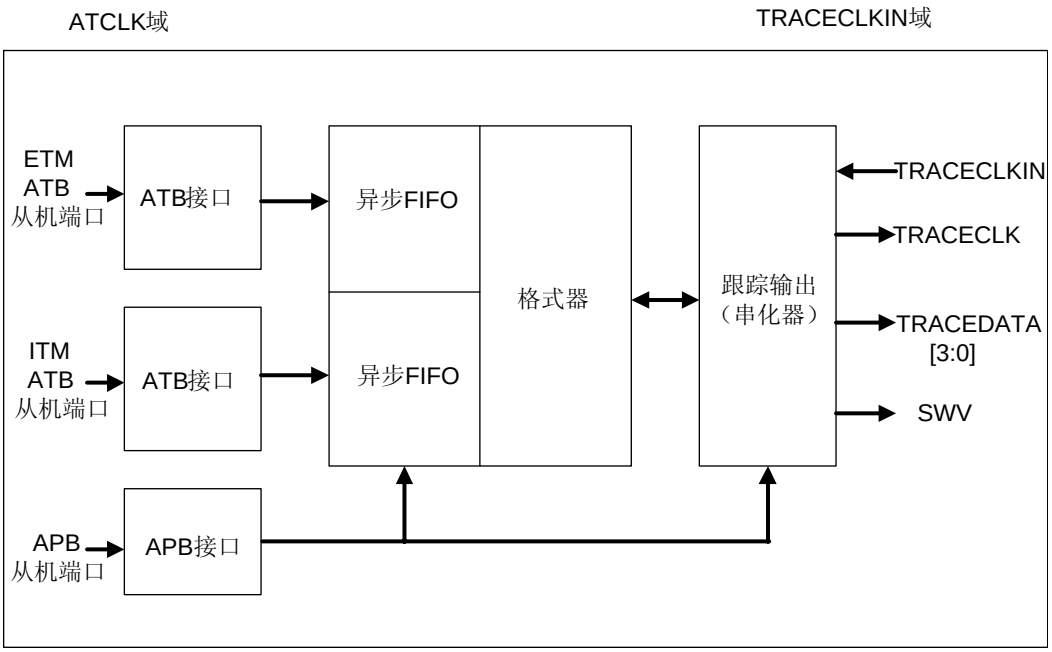


图 13-2　TPIU 方框图（含 ETM 的版本）

13.1.2　TPIU 组件

以下小节将会对 TPIU 的主要组件进行描述。包括：

- *异步 FIFO*
- *格式器*
- *跟踪输出*
- *ATB 接口*
- *APB 接口*

异步 FIFO

通过异步 FIFO 可以以任意速度驱动跟踪数据输出，而不会受内核时钟速率的影响。

格式器

格式器将源 ID 信号插入数据包流，这样跟踪数据就可以和它的跟踪源重新联合。格式器仅存在于含 ETM 的 TPIU 版本中。

跟踪输出

跟踪输出模块在已格式化的数据离开芯片前将其串行化。

ATB 接口

TPIU 可以直接从跟踪源（ETM 或 ITM）或者通过跟踪漏斗（Trace Funnel）来接收来自跟踪源的跟踪数据。详见“ATB 接口”

APB 接口

APB 接口是 TPIU 的编程接口。详见“APB 接口”。

13.1.3　TPIU 输入和输出

本节描述了 TPIU 输入和输出。包括以下内容：

- *跟踪输出端口*
- *ATB 接口*
- *其他配置输入*

跟踪输出端口

表 13-1 描述了跟踪输出端口信号。

表 13-1　跟踪输出端口信号

名称	类型	描述
TRACECLKIN	输入	从 ATB 处断开的时钟信号，用于简单控制跟踪端口的速率。该信号通常由可控制的片内时钟源提供，但是如果使用了高速管脚，则可以由外部时钟发生器驱动。数据仅在上升沿发生变化。
TRESETn	输入	这是 **TRACECLKIN** 域的复位信号。该信号主要通过上电复位来驱动，并且必须和 **TRACECLKIN** 信号同步。
TRACECLK	输出	**TRACEDATA** 可以在 TRACECLK 的上升沿和下降沿发生变化
TRACEDATA[3:0]	输出	计时模式下的输出数据
SWV	输出	异步模式下的输出数据

ATB 接口

ATB 接口可以为 1 个或 2 个，具体取决于 TPIU 的配置。表 13-2 对 ATB 端口信号进行了描述。若将 TPIU 配置成仅含一个 ATB 接口，则不使用端口 2 的信号。

表 13-2　ATB 端口信号

名称	类型	描述
CLK	输入	跟踪总线和 APB 接口的时钟信号
nRESET	输入	**CLK** 域（ATB/APB 接口）的复位信号
CLKEN	输入	**CLK** 域的时钟使能信号
ATVALID1S	输入	来自跟踪源 1 的数据在该周期有效
ATREADY1S	输出	如果该信号有效（**ATVALID** 为高电平），那么该周期从跟踪源 1 接收数据
ATDATA1S[7:0]	输入	跟踪源 1 的跟踪数据输入
ATID1S[6:0]	输入	跟踪源 1 的 ID。它不会动态改变
ATVALID2S	输入	来自跟踪源 2 的数据在该周期有效
ATREADY2	输出	如果该信号有效（**ATVALID** 为高电平），那么该周期从跟踪源 2 接收数据
ATDATA2S[7:0]	输入	跟踪源 2 的跟踪数据输入
ATID2S[6:0]	输入	跟踪源 2 的 ID。它不会动态改变

其他配置输入

表 13-3 描述了其他配置输入。

表 13-3　其他配置输入

名称	类型	描述
MAXPORTSIZE[1:0]	输入	定义同步跟踪输出的最大管脚数
SyncReq	输入	全局跟踪同步触发信号。用来将同步数据包插入到格式化数据流中。不用于无 ETM 的配置。

13.2 TPIU 寄存器

本小节对 TPIU 寄存器进行了描述。包括以下内容：

- *TPIU 寄存器汇总*
- *TPIU 寄存器描述*

13.2.1　TPIU 寄存器汇总

表 13-4 汇总了 TPIU 寄存器。

表 13-4　TPIU 寄存器

名称	类型	地址	复位值
支持的端口大小寄存器	只读	0xE0040000	0bxx0x
当前端口大小寄存器	读/写	0xE0040004	0x01
当前输出速率因子寄存器	读/写	0xE0040010	0x0000
所选管脚的协议寄存器	读/写	0xE0040F0	0x01
格式器和刷新状态寄存器	读/写	0xE0040300	0x08
格式器和刷新控制寄存器	只读	0xE0040304	0x00 或 0x102
格式器同步计数器寄存器	只读	0xE0040308	0x00
整合寄存器：ITATBCTR2	只读	0xE0040EF0	0x0
整合寄存器：ITATBCTR0	只读	0xE0040EF8	0x0

13.2.2 TPIU 寄存器描述

以下是对 TPIU 寄存器的一些描述。

支持的端口大小寄存器

该寄存器可读/写。每个位单元表示器件支持的一个端口大小，即支持 4，2 或 1，分别与位[3:0]对应。如果相应的位置位，则表示允许该端口大小。默认下，RTL 被设计成支持所有端口大小，即设为 0x0000000B。该寄存器还要受输入 tie-off－**MAXPORTSIZE** 的限制。外部 tie-off－**MAXPORTSIZE** 必须在 ASIC 最终定案时设置，以反映与物理管脚相连的 TRACEDATA 信号的实际数目。这样可以保证工具不会选择一个不能被 TPA 捕获的端口大小。**MAXPORTSIZE** 的值促使代表更宽宽度的“支持的端口大小寄存器”的位清零，即变成不被支持。

其位分配如图 13-3 所示。

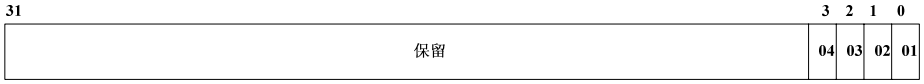


图 13－3　支持的端口大小寄存器的位分配

当前端口大小寄存器

该寄存器可读/写。当前端口大小寄存器的格式与支持的端口大小寄存器相同，但是只有一个位置位，且所有其他的位都必须为 0。在写该寄存器时，如果使得多个位置位或者让不被支持的位置位，那么该次写入的值无效，并且还会引起不可预测的行为。

当前端口大小寄存器如果使用与支持的端口大小寄存器相同的格式会更加方便，因为以后在器件中不用再对大小进行译码，并且还保持了其他寄存器组的格式，以对有效分配进行检验。

复位时该寄存器默认为最小端口大小，即 1 位，因此在读取时其值为 0x00000001。

当前输出速率因子寄存器

使用当前输出速率因子寄存器来调整异步输出的波特率。

当前输出速率因子寄存器的位分配如图 13-4 所示。

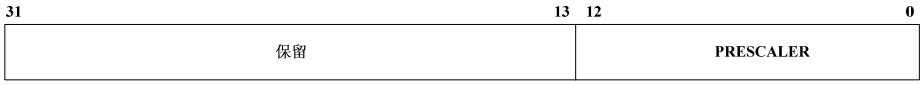


图 13-4　当前输出速率因子寄存器

表 13-5 描述了当前输出速率因子寄存器的各个位。

表 13-5　当前输出速率分配寄存器的位分配

域	名称	定义
[31:13]	-	保留。RAZ/SBZP
[12:0]	PRESCALER	TRACECLKIN 的因子是预分频值+1

所选管脚的协议寄存器

使用所选管脚的协议寄存器来选择用于跟踪输出的协议。

寄存器地址，访问类型和复位状态：

地址　　0xE00400F0

访问类型　　读/写

复位状态　　0x01

所选管脚的协议寄存器的位分配如图 13-5 所示。

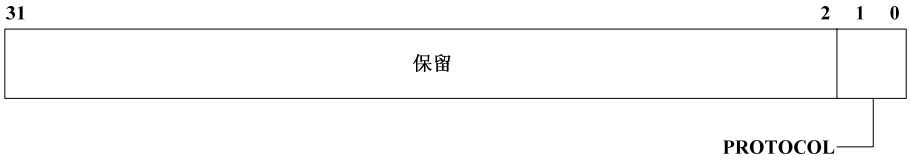


图 13-5　所选管脚的协议寄存器的位分配

表 13-6 描述了所选管脚的协议寄存器的各个位。

表 13-6　所选管脚的协议寄存器的位分配

域	名称	定义
[31:2]	-	保留
[1:0]	PROTOCOL	00 – 跟踪端口模式 01 – 串行写输出（曼彻斯特）。它为复位值 10 – 串行写输出（NRZ） 11 – 保留

注：

如果该寄存器的值在跟踪数据输出时发生变化，那么将会破坏数据。

格式器和刷新状态寄存器

使用“格式器和刷新状态寄存器”来读取 TPIU 格式器的状态。

寄存器地址，访问类型和复位状态：

地址　　0xE0040300

访问类型　　只读

复位状态　　0x08

格式器和刷新状态寄存器的位分配如图 13-6 所示。

图 13-6　格式器和刷新状态寄存器的位分配

表 13-7 描述了格式器和刷新状态寄存器的各个位。

表 13-7　格式器和刷新状态寄存器的位分配

域	名称	定义
[31:4]	-	保留
[3]	FtNonStop	格式器不能停止
[2]	TCPresent	读取该位时总是为 0
[1]	FtStopped	读取该位时总是为 0
[0]	FIInProg	读取该位时总是为 0

格式器和刷新控制寄存器

通过格式器和刷新控制寄存器来读取格式器是否存在。因为不能动态控制格式器，所以该寄存器为只读寄存器。如果格式器存在，那么读取该寄存器时为 0x102，表示格式器使能并且处于连续模式，此时在 **TRIGIN** 有效时指示开始触发。

寄存器地址，访问类型和复位状态：

地址　　0xE0040304

访问类型　　只读

复位状态　　0x00 或 0x102

格式器同步计数器寄存器

全局同步触发器由 PC 采样器模块产生。这意味着 TPIU 中不含同步计数器。

寄存器地址，访问类型和复位状态：

地址　　0xE0040308

访问类型　　只读

复位状态　　0x00

整合测试寄存器

使用整合测试寄存器来对 TPIU 和 Cortex-M3 系统中的其他器件进行布局检测。这些寄存器可以直接对输出进行控制，也可以读取输入值。处理器提供两个整合测试寄存器：

- 整合测试寄存器－ ITATBCTR2
- 整合测试寄存器－ ITATBCTR0

整合测试寄存器－ITATBCTR2

寄存器地址，访问类型和复位状态：

地址　　0xE0040EF0

访问类型　　只读

复位状态　　0x0

整合测试寄存器的位分配如图 13-7 所示。

图 13-7　整合测试寄存器的位分配

表 13-8 描述了整合测试寄存器的各个位。

表 13-8　整合测试寄存器的位分配

域	名称	定义
[31:1]	-	保留
[0]	ATREADY1	该位读取或设置 ATREADYS1 和 ATREADYS2 的值

整合测试寄存器—ITATBCTR0

寄存器地址，访问类型和复位状态：

地址　　0xE0040EF8

访问类型　　只读

复位状态　　0x0

整合测试寄存器的位分配如图 13-8 所示。

图 13-8　整合测试寄存器的位分配

表 13-9 描述了整合测试寄存器的各个位。

表 13-9　整合测试寄存器的位分配

域	名称	定义
[31:1]	-	保留
[0]	ATVALID1，ATVALID2	该位读取或设置 ATVALIDS1 和 ATVALIDS2 的或值

第14章 总线接口

本章介绍了处理器总线接口。包括以下小节：

- *关于总线接口*
- *ICode 总线接口*
- *DCode 总线接口*
- *系统接口*
- *外部专用外设接口*
- *对齐访问*
- *横跨区域的不对齐访问*
- *Bit-band 访问*
- 写缓冲器

14.1 关于总线接口

处理器包含 4 个总线接口：

- ICode 存储器接口。从 Code 存储器空间（0x0000000 – 0x1FFFFFFF）的取指都在这条 32 位 AHBLite 总线上执行。详见“ICode 总线接口”。
- DCode 存储器接口。对 Code 存储器空间（0x0000000 – 0x1FFFFFFF）进行数据和调试访问都在这条 32 位 AHBLite 总线上执行。详见“DCode 总线接口”。
- 系统接口。对系统空间（0x20000000 – 0xDFFFFFFF）进行取指、数据和调试访问都在这条 32 位 AHBLite 总线上执行。详见“系统接口”。
- 外部专用外设总线（PPB）。对外部 PPB 空间（oxE0040000 – 0xE00FFFFF）进行数据和调试访问都在这条 32 位 APB 总线（AMBA v2.0）上执行。跟踪端口接口单元（TPIU）和厂商特定的外围器件都在这条总线上。详见“*外部专用外设接口*”。

注：

处理器包含一条内部专用外设总线，用来访问嵌套向量中断控制器（NVIC）、数据观察点和触发（DWT）、Flash 修补和断点（FPB），以及存储器保护单元（MPU）。

14.2 ICode 总线接口

ICode 接口是一个 32 位的 AHBLite 总线接口。从程序存储器空间（0x00000000 – 0x1FFFFFFF）取指和取向量都在这条总线上执行。

只有 CM3Core 取指总线可以访问 ICode 接口，以获取最佳的代码读取性能。所有取指都是按字来操作。每个字的取指数目取决于运行的代码和存储器中代码的对齐情况。如表 14-1 所述。

表 14-1　取指

32 位取指[31:16]	32 位取指[15:0]	描述
Thumb[15:0]	Thumb[15:0]	所有 Thumb 指令在存储器中都是半字对齐，所以一次取指两条 Thumb 指令。对于连续的代码，每隔一个周期进行一次取指。如果出现中断或分支，取指可以在背对背（back-to-back）周期进行。
Thumb-2[31:16]	Thumb-2[15:0]	如果 Thumb-2 代码在存储器中是字对齐，那么每个周期取指一条完整的 Thumb-2 指令。
Thumb-2[15:0]	Thumb-2[31:16]	如果 Thumb-2 代码是半字对齐，那么第一次 32 位取指仅返回 Thumb-2 指令的第一个半字。要想读取第二个半字，必须进行第二次取指。这种情况会根据执行的指令产生一个等待周期（CM3Core 在该周期不能执行指令）。其他的周期延迟仅在第一个半字对齐的 Thumb-2 取指时发生。CM3Core 包含一个 3 个入口的取指冲区，并且因此半字对齐的 Thumb-2 指令的高半字存在于取指缓冲区中，供接下来的连续 Thumb-2 指令使用。

所有 ICode 取指都被标记为可高速缓存和缓冲（**HPROTI[3:2]**＝2’b11）以及不可分配和不可共用（**MEMATTRI**＝2’b00）。这些属性都是硬连线。如果含有 MPU，ICode 总线将忽略 MPU 区域属性。

HPROTI[0]表示正在读取的是：

- 0－ 取指
- 1－ 取向量

所有 ICode 操作都是不连续的。

14.2.1　分支状态信号

分支状态信号（**BRCHSTAT**）从 ETM 接口上输出，该信号表示在流水线上是否存在分支。这个可以使用在比如预取器中，在即将获取分支时阻止预取进行。要详细了解分支状态信号，请参考第 16 章“嵌入式跟踪宏单元接口”。

14.3　DCode 总线接口

DCode 接口是 32 位的 AHBLite 总线。对程序存储空间（0x00000000 – 0x1FFFFFFF）的数据和调试访问都在这条总线上执行。内核数据访问的优先级比调试访问要高。因而当总线上同时出现内核和调试访问时，必须在内核访问结束后才开始调试访问。

该接口的控制逻辑将不对齐的数据和调试访问转入两个或三个（这取决于不对齐访问的大小和对齐情况）对齐访问。这样就会停止接下来的所有数据或调试访问，直至不对齐访问结束。

要详细了解不对齐访问，请参考“访问对齐情况”。

14.3.1　专用

DCode 总线支持独占访问。这通过使用两个边带（sideband）信号——**EXREQD** 和 **EXRESPD** 来实现。详见“DCode 接口”。

14.3.2　存储器属性

处理器通过使用一条称作 **MEMATTRD** 的边带总线导出 DCode 总线上的存储器属性。详见“存储器属性”。

14.4　系统接口

系统接口是 32 位的 AHBLite 总线。对系统存储空间（0x20000000 – 0xDFFFFFFF，0xE0100000 – 0xFFFFFFFF）的取指、取向量以及数据和调试访问都在这条总线上执行。

当该总线上同时出现上述访问，仲裁顺序（按递减优先级）应该是：

- 数据访问
- 取指和取向量
- 调试

系统总线接口包含处理不对齐访问、FPB 重新映射访问、bit-band 访问，以及流水线取指的控制逻辑。

14.4.1　不对齐访问

不对齐的数据和调试访问都被转换成 2 个或 3 个（这取决于不对齐访问的大小和对齐情况）对齐访问。这样就会停止接下来的所有数据或调试访问，直至不对齐访问结束。要详细了解不对齐访问，请参考“访问的对齐情况”。

14.4.2　Bit-band 访问

对 bit-band 别名区域的访问被转换成访问 bit-band 区域。Bie-band 写操作要花费 2 个周期（它们都被转换成读－修改－写操作），并且因此 bit-band 写访问会停止接下来的所有访问，直至 bit-band 访问结束。要详细了解 bit-band 访问，请参考“bit-band 访问”。

14.4.3　Flash 修补重新映射

在访问被重新映射到系统存储空间的程序存储空间时，重新映射操作会延迟一个周期，这样一来便会停止接下来的所有访问，直至 Flash 修补访问结束。要详细了解 Flash 修补，请参考“Flash 修补和断点”。

14.4.4　独占访问（exclusive access）

系统总线支持独占访问。它通过两个边带信号——**EXREQS** 和 **EXRESPS** 来执行。详见“系统总线接口”。

14.4.5　存储器属性

处理器通过使用一条称作 **MEMATTRS** 的边带总线将系统总线上的存储器属性导出。详见“存储器属性”。

14.4.6　流水线式取指

为了在系统总线上提供一个清除时序的接口，对该总线的取指和取向量请求都被记录。这导致额外的延迟周期，因为从系统总线上取指要花费 2 个周期。这同样意味着从系统总线进行背对背取指是不可能的。

注：

不记录对 ICode 总线的取指请求。性能临界（performance critical）代码必须从 ICode 接口运行。

14.5　外部专用外设接口

外部专用外设接口是高级外设总线（APB）（AMBA v2.0）总线。对外部外设存储空间（oxE0040000 – 0xE00FFFFF）的数据和调试访问都在这条总线上执行。该总线不支持等待状态。TPIU 和所有厂商特定的元件都在这条总线上。内核数据访问的优先级比调试访问要高，因此当该总线同时出现内核和调试访问，那么调试访问必须等到内核访问结束后才能进行。该接口只支持译码外部 PPB 空间所必需的地址位。这些地址位是 **PADDR** 的位[19:2]。

PADDR31 在这条总线上被当作边带信号驱动。当 **PADDR31** 信号为高电平时，表示 AHB-AP 调试是请求主机。当该信号为低电平时，表示内核是请求主机。

对该总线的不对齐访问在结构上是不可预测的，且不被支持。处理器将来自内核的原始的 **HADDR[1:0]**请求逐出，并且不会将该请求转换成多个对齐访问。

14.6　访问的对齐情况

Cortex-M3 处理器使用 ARMv6 模型时支持不对齐的数据访问。DCode 和系统总线接口包含将不对齐访问转换成对齐访问的逻辑电路。

表 14-2 描述了不对齐的数据访问。该表在第一列列出了不对齐访问，其他列表示访问转换后的类型。根据不对齐访问的大小和对齐情况，不对齐的数据访问被转换成 2 个或 3 个对齐访问。

表 14-2　不对齐访问的总线映射表

不对齐访问		对齐访问					
		周期 1		周期 2		周期 3	
大小	ADDR [1:0]	HSIZE	HADDR [1:0]	HSIZE	HADDR[1:0]	HSIZE	HADDR[1:0]
半字	00	半字	00	-	-	-	-
半字	01	字节	01	字节	10	-	-
半字	10	半字	10	-	-	-	-
半字	11	字节	11	字节	{（Addr+4）[31:2],2b00}	-	-
字	00	字	00	-	-	-	-
字	01	字节	01	半字	10	字节	{（Addr+4）[31:2],2b00}
字	10	半字	10	半字	{（Addr+4）[31:2],2b00}	-	-
字	11	字节	11	半字	{（Addr+4）[31:2],2b00}	字节	{（Addr+4）[31:2],2b00}

注：

没有将跨入 bit-band 别名区域的不对齐访问当作 bit-band 请求来对待，并且访问没有被重新映射到 bit-band 区域。相反，它们被当作对 bit-band 别名区域的半字或字节访问。

14.7　横跨区域的不对齐访问

CM3Core 支持 ARMV6 的不对齐访问。所有访问都被 CM3Core 当作单个不对齐访问来执行，并且都被 DCode 和系统总线接口转换成 2 个或 3 个对齐访问。

注：

所有的 Cortex-M3 外部访问都是对齐的。

只有单寄存器 Load/Store（LDR，STR）支持不对齐访问。双寄存器 Load/Store 已经支持字对齐访问，但是不允许其他不对齐访问，否则会发生错误。

横跨存储器映射边界的不对齐访问在结构上是不可预测的。处理器的行为与边界有关，如下所示：

- DCode 访问限制（wrap）在区域内。例如，对代码空间最后一个字节（0x1FFFFFFF）的不对齐半字访问被 DCode 接口转换成先对 0x1FFFFFFF 进行字节访问，接着对 0x00000000 进行字节访问。
- 跨入 PPB 空间的系统访问不会限制在系统空间内。例如，对系统空间最后一个字节（0xDFFFFFFF）的不对齐半字访问被系统接口转换成先对 0xDFFFFFFF 进行字节访问，接着对 0xE0000000 进行字节访问。0xE0000000 不是系统总线上的有效地址。
- 跨入程序空间的系统访问不会限制在系统空间内。例如，对系统空间最后一个字节（0xFFFFFFFF）的不对齐半字访问被系统接口转换成先对 0xFFFFFFFF 进行字节访问，接着对 0x00000000 进行字节访问。0x00000000 不是系统总线上的有效地址。
- PPB 空间不支持不对齐访问，因此在访问 PPB 时不会出现跨越边界的情况。

跨入 bit-band 别名区域的不对齐访问同样在结构上是不可预测的。处理器访问 bit-band 别名地址，但是这样做并不会导致发生 bit-band 操作。例如，对 0x21FFFFFF 执行不对齐半字访问可以当作先对 0x21FFFFFF 进行字节访问，接着对 0x22000000（bit-band 的第一个字节）进行字节访问。

可以与 FPB 中的 literal 比较器相媲美的不对齐装载不会被重新映射。FPB 只重新映射对齐地址。

14.8　Bit-band 访问

系统总线接口包含控制 bit-band 访问的逻辑电路，如下所示：

- 它将 bit-band 别名地址重新映射到 bit-band 区域
- 在读操作时，它从读字节中取出请求位，并将返回的读数据的 LSB 中的这个位返回给内核。
- 在写操作时，它将写操作转换成原子的读－修改－写操作。

要详细了解 bit-banding，请参考“bit-banding”。

注：

- CM3Core 在 bit-band 操作过程不会停止运行，除非它在正执行 bit-band 操作时试图访问系统总线。

- 对 bit-band 别名区域的大端访问必须为字节大小。否则，访问是不可预测的。

14.9　写缓冲区

在数据存储过程中，为了防止总线等待周期阻止 Cortex-M3 处理器运行，DCode 和系统总线带缓冲的存储通过含一个入口的写缓冲区。如果写缓冲区是满的，那么接下来的总线访问都将停止，直至写缓冲区被清空。写缓冲区只在总线等待带缓冲的存储操作的数据阶段时才使用，否则处理将在总线上结束。

DMB 和 DSB 指令在完成前一直等待写缓冲区清空。如果在 DMB/DSB 等待写缓冲区清空时发生中断，那么在中断结束时返回到 DMB/DSB 后面的操作码。这是因为中断处理是一个内存排序（memory barrier）操作。

14.10　存储器属性

处理器通过额外的边带总线 MEMATTR 导出 DCode 和系统总线上的存储器属性。

表 14-3 列出了 **MEMATTR[0]**和 **HPROT[3:2]**之间的关系。

表 14-3　存储器属性

MEMATTR[0]	HPROT[3]	HPROT[2]	描述
0	0	0	非常整齐
0	0	1	器件
0	1	0	L1 可高速缓存，L2 不可高速缓存
1	0	0	无效
1	0	1	无效
1	1	0	Cache WT，读取时分配
0	1	1	Cache WB，读写时分配
1	1	1	Cache WB，读取时分配

第15章 嵌入式跟踪宏单元

本章描述了嵌入式跟踪宏单元（ETM）。包括以下内容：

- *ETM 概述*
- *数据跟踪*
- *ETM 资源*
- *跟踪输出*
- *ETM 结构*
- *ETM 编程器模型*

15.1 ETM 概述

ETM 是可选的调试元件，它可以重构程序执行。ETM 是高速低功耗的调试工具，仅支持指令跟踪。因而可以保证将占用空间减到最少，同时还简化了门数。

15.1.1 ETM 框图

图 15-1 显示了 ETM 的方框图，并描述了 ETM 与*跟踪端口接口单元（TPIU）*块接口的方法。

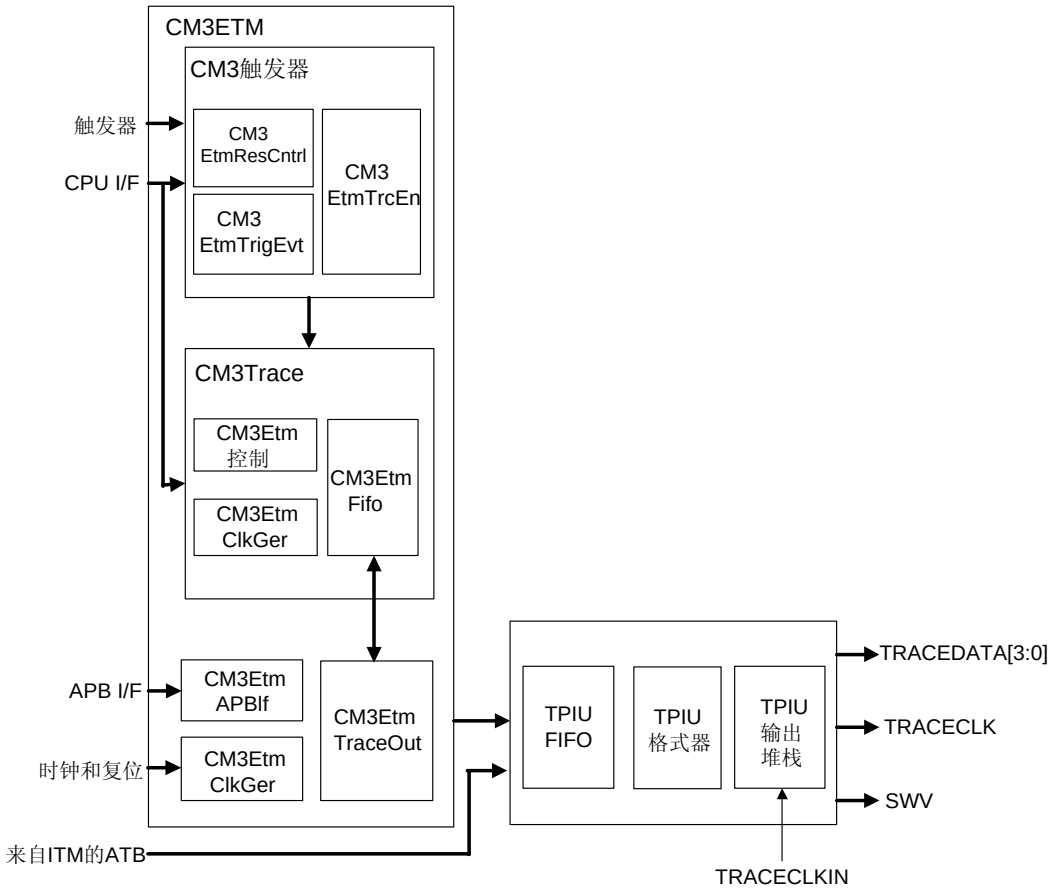


图 15-1　ETM 方框图

15.1.2　ETM 资源

表 15-1 列出了 Cortex-M3 资源。

表 15-1　Cortex-M3 资源

特性	Cortex-M3 ETM 包含的组件
结构版本	ETMv3.4
地址比较器对	0
数据比较器	0
上下文 ID 比较器	0
MMD	0
计数器	0
定序器（sequencer）	无
开始/停止块	有
嵌入式 ICE 比较器	4
外部输入	2
外部输出	0
扩展的外部输入	0
扩展的外部输入选择器	0
FIFOFULL	有
FIFOFULL 级别（level）设置	有
分支广播	有
ASIC 控制寄存器	无
数据抑制（suppression）	无
对寄存器的软件访问	有
可读寄存器	有
FIFO 大小	16 字节
最小的端口大小	8 字节
最大的端口大小	8 字节
正常的端口模式	-
正常的半速率计时（half-rate clocking）/1:1	有—异步
解复用（demux）的端口模式	-
解复用（demux）的半速率计时/1:2	无
复用（mux）的端口模式/2:1	无
1:4 端口模式	无
动态端口模式（包括停止）	无，异步端口模式支持
CPRT 数据	无
首先装载 PC	无
取指比较	无
装载跟踪的数据	无

15.2　数据跟踪

Cortex-M3 系统使用“*数据观察点与跟踪*”（DWT）和“*仪表跟踪宏单元（ITM）*”元件可以执行低带宽数据跟踪。为了使用较少的管脚实现指令跟踪，ETM 不包含数据跟踪。因为这样可以简化触发资源，从而大大节省了 ETM 的门数。

当处理器包含 ETM 时，两个跟踪源（ITM 和 ETM）都将数据馈送入*跟踪端口接口单元*（TPIU），在 TPIU 那里它们联合在一起，然后通常通过跟踪端口输出。DWT 可以提供聚焦的（focused）数据跟踪，或者全局数据跟踪（遭受 FIFO 溢出问题）。TPIU 应单内核 Cortex-M3 系统的要求而优化。

15.3　ETM 资源

因为 ETM 不会产生数据跟踪信息，所以低带宽减少了复杂触发能力的要求。即 ETM 不包括以下器件：

- 内部比较器
- 计数器
- 定序器（sequencer）

15.3.1　周期性同步（periodic synchronization）

嵌套向量中断控制器（NVIC）含有一个 12 位计数器，它可以将 PC 值写入 DWT。计数器用来为 ETM 创建周期性的同步数据包（packet）。

15.3.2　数据和指令地址比较资源

DWT 在数据总线上提供 4 个地址比较器，这些地址比较器用于提供调试功能。在 DWT 单元内，可以指定通过匹配来触发的功能，并且其中一个功能将产生一个 ETM 匹配输入。这些输入作为嵌入式 ICE 比较器的输入提交给 ETM。

单个 DWT 资源可以触发一个 ETM 事件，还可以直接从相同事件产生仪表跟踪（instrumentation trace）。

4 个 DWT 比较器还可以单独配置成与执行 PC 比较，以便允许 ETM 访问 PC 比较资源。这些输入都作为嵌入式 ICE 比较器输入提交给 ETM。

注：

将 DWT 比较器用作 PC 比较器，这样可以减少数据地址比较的数目。

要了解更多有关 DWT 单元的信息，请参考“*数据观察点和跟踪*”。

外部输入

两个外部输入 ETMEXTIN[1:0]可以使额外的片内 IP 为 ETM 产生触发/使能信号。

开始/停止块

开始/停止块通过使用 ETM 的嵌入式 ICE 输入来控制开始/停止行为。这些输入都由 DWT 控制。

15.3.3　FIFO 功能

FIFO 为 16 字节大小。

处理器内核提供 FIFOFULL 输出，但是不支持将 FIFO 用作输入。如果使用了 FIFOFULL 资源，那么为了阻止内核运行，还需要一个外部机制。虽然在典型的应用中阻止内核运行的做法不太被接受，但它却为 100%跟踪提供了一种机制，这种机制可以与为不停止运行提供的局部跟踪相媲美。

15.4　跟踪输出

ETM 以内核时钟速率一次输出 8 位数据。它不支持不同的跟踪端口大小和跟踪端口模式。TPIU 用于在片外导出跟踪输出。该输出和 ATB 协议兼容。

因为不支持 AFVALID 功能，所以跟踪端口不能刷新来自 ETM FIFO 的数据。但是由于 8 位 ATB 端口的原因，FIFO 总是清空，这使得 AFVALID 变得不再必要。

Cortex-M3 系统配有一个优化的 TPIU，TPIU 被设计成与 ETM 和 ITM 共用。这个 TPIU 不支持额外的跟踪源。但是如果 TPIU 被更复杂的版本和更多的跟踪基础结构替代，那么可以加入其他跟踪源。

注：

为使用多个跟踪源的系统提供跟踪 ID 寄存器和输出。

TPIU 使用“格式化”跟踪输出协议。即 **TRACECTL** 信号无需额外的管脚。

ETM 的跟踪输出与内核时钟同步。跟踪端口接口包含一个异步 FIFO。如果想要将 ETM 集成进多核系统，你可能必须使用一个异步的 ATB 桥。

15.5　ETM 结构

ETM 只跟踪指令，它执行 ARM ETM 结构 v3.4，并基于 ARM ETM 结构规范。详见“*ARM 嵌入式跟踪宏结构规范*”。

所有 Thumb-2 指令都被当作单个指令跟踪。IT 指令之后的指令被当作正常的条件指令来跟踪。减压器无需涉及（refer to）IT 指令。

15.5.1　可重新开始的指令

ARMv7-M 结构可以重新开始被异常中断的 LSM 指令。ETM 通过指示异常已被取消来跟踪已经被异常中断的指令。在从异常返回时，ETM 再次跟踪相同的指令，而不管指令是否被重新开始或恢复。

15.5.2　异常返回

ETM 清楚地在跟踪流中指示了异常返回。这是因为异常返回泛函性（functionality）以数据独立的方式编码，并且异常返回表现得和简单的分支不同。

用于指示从异常返回的数据包编码如图 15-2 所示。

7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0

图 15-2　异常返回数据包编码

如果一个新的、优先级更高的异常抢占了堆栈出栈。跳到异常处理器的分支必须指示取消最后一条指令，但是却不取消“从异常指令返回”。如果出现“从异常数据包返回”，那么这就意味着先前的指令结束。

15.5.3 异常跟踪

为了跟踪异常，必须在分支数据包中添加一个可选域。这个额外的域指定了异常信息。当异常分支为以下情况时，正常的分支数据包在 1 到 5 字节跟踪数据中编码：

- 2-5 字节地址
- 1-2 字节异常

通过异常映射可以在一个周期内对最频繁的异常进行编码。ETM 异常跟踪映射在表 15-2 中作了描述。

表 15-2　异常跟踪映射

字节数	异常	当前中断	跟踪值
1 字节异常	无	0	0
1 字节异常	IRQ1	17	1
1 字节异常	IRQ2	18	2
1 字节异常	IRQ3	19	3
1 字节异常	IRQ4	20	4
1 字节异常	IRQ5	21	5
1 字节异常	IRQ6	22	6
1 字节异常	IRQ7	23	7
1 字节异常	IRQ0	16	8
1 字节异常	使用故障	6	9
1 字节异常	NMI	2	10
1 字节异常	SVC	11	11
1 字节异常	DebugMon	12	12
1 字节异常	MemManage	4	13
1 字节异常	PendSV	14	14
2 字节异常	SysTick	15	15
2 字节异常	保留	8	16
2 字节异常	复位	1	17
2 字节异常	保留	10	18
2 字节异常	硬故障	3	19
2 字节异常	保留	9	20
2 字节异常	总线故障	5	21
2 字节异常	保留	7	22
2 字节异常	保留	13	23
2 字节异常	IRQ8	24	24
2 字节异常	IRQ9	25	25
2 字节异常	IRQ10	26	26

续上表…

字节数	异常	当前中断	跟踪值
-	-	-	-
-	-	-	-
-	-	-	-
2 字节异常	IRQ239	512	512

图 15-3 显示了带有异常包的全部分支。

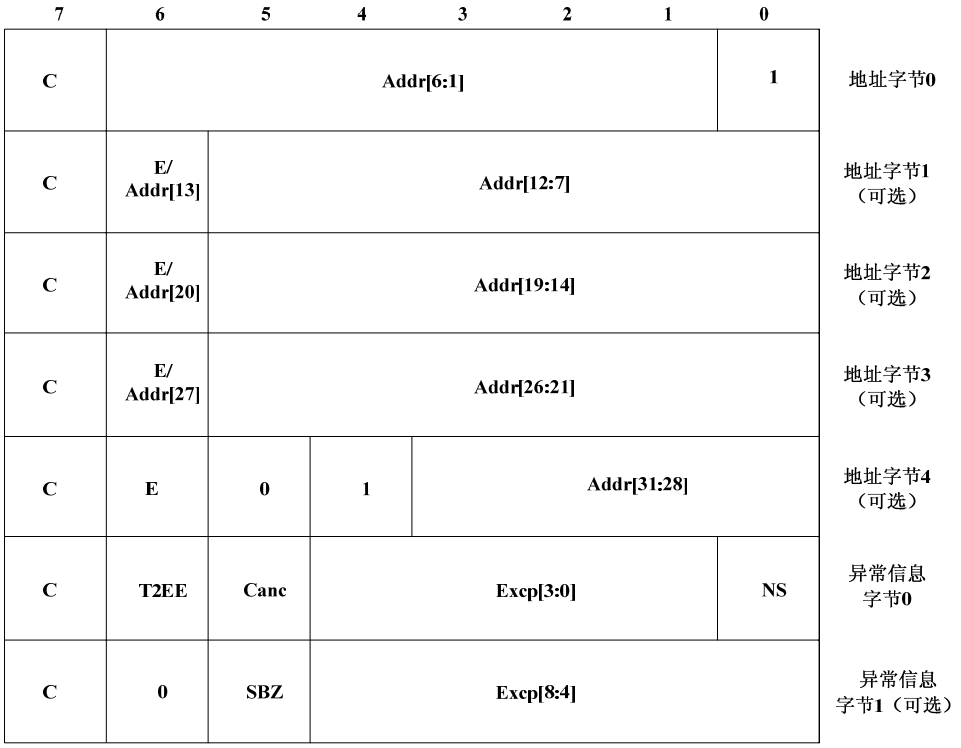


图 15-3　分支数据包的异常编码

最后的地址字节使用被设为 0b01 的位[7:6]来指示地址域的末端。异常数据紧跟着该域之后。如果再发生一个异常，那么异常字节 0 将位[7]设为 1。如果没有异常出现，并且仅地址位[6:1]发生改变，则使用一个单字节。如果出现异常，那么至少要使用两个字节来发信号给地址。

在进入异常处理器前立即关闭跟踪时，ETM 保持使能状态直到异常发生。因而 ETM 可以跟踪分支地址、异常类型和恢复信息。

15.6　ETM 编程器模型

有关 ETM 编程器模型的信息详见“ARM 嵌入式跟踪宏单元架构规范”。本小节定义了 ETM 编程器模型的实现特性。

15.6.1　APB 接口

ETM 包含先进的外设总线（APB）从机接口，可以对 ETM 寄存器进行读写操作。该接口与处理器时钟同步，内核和外部调试接口可以通过 SW-DP/JTAG-DP 对其进行访问。

15.6.2　ETM 寄存器列表

ETM 寄存器在表 15-3 中列出。详见“ARM 嵌入式跟踪宏单元架构规范”。

表 15-3　ETM 寄存器

名称	类型	地址	存在	描述
ETM 控制	R/W	0xE0041000	是	
配置代码	RO	0xE0041004	是	
触发事件	WO	0xE0041008	是	定义控制触发的事件
ASIC 控制	WO	0xE004100C	否	-
ETM 状态	RO 或 R/W	0xE0041010	是	提供跟踪和触发逻辑的当前状态的相关信息
系统配置	RO	0xE0041014	是	
跟踪使能（TraceEnable）	WO	0xE0041018,0xE004101C	否	-
跟踪使能事件	WO	0xE0041020	是	描述 TraceEnable 使能事件
跟踪使能（TraceEnable）控制 1	WO	0xE0041024	是	
FIFOFULL 区	WO	0xE0041028	否	-
FIFOFULL 级别（level）	WO 或 R/W	0xE004102C	是	其级别保持低于 FIFO 满时对应的级别
观察数据（ViewData）	WO	0xE0041030~0xE004103C	否	-
地址比较器	WO	0xE0041040~0xE004113C	否	-
计数器	WO	0xE0041140~0xE004157C	否	-
定序器（sequencer）	R/W	0xE0041180~0xE0041194, 0xE0041198	否	-
外部输出	WO	0xE00411A0~0xE00411AC	否	-
CID 比较器	WO	0xE00411B0~0xE00411BC	否	-
实现规范	WO	0xE00411C0~0xE00411DC	否	所有 RAZ。忽略写
同步频率	WO	0xE00411E0	否	从 DWT 产生同步
ETM ID	RO	0xE00411E4	是	
配置代码扩展	RO	0xE00411E8	是	
扩展的外部输入选择器	WO	0xE00411EC	否	没有实现扩展的外部输入
跟踪使能（TraceEnable）开始/停止嵌入式 ICE	R/W	0xE00411F0	是	位 19:16 将 E-ICE 输入配置成用作停止资源。位 3:0 将 E-ICE 输入配置成用作开始资源
嵌入式 ICE 行为控制	WO	0xE00411F4	否	嵌入式 ICE 输入使用默认的行为
CoreSight 跟踪 ID	R/W	0xE0041200	是	正常实现
OS 保存/恢复	WO	0xE0041304~0xE0041308	否	没有实现 OS 保存/恢复。RAZ，忽略写
ITMISCIN	RO	0xE0041EE0	是	将[1:0]设成 EXTIN[1:0]，将[4]设成 COREHALT
ITTRIGOUT	WO	0xE0041EE8	是	将[0]设成 TRIGOUT

续上表…

名称	类型	地址	存在	描述
ITATBCTR2	RO	0xE0041EF0	是	将[0]设成 ATREADY
ITATBCTR0	WO	0xE0041EF8	是	将[0]设成 ATVALID
整合模式控制	R/W	0xE0041F00	是	正常执行
声明标签（claim tag）	R/W	0xE0041FA0~0xE0041FA4	是	实现 4 位声明标签
锁定访问	WO	0xE0041FB0~0xE0041FB4	是	正常实现
鉴别状态	RO	0xE0041FB8	是	正常实现
器件类型	RO	0xE0041FCC	是	复位值：0x13
外设 ID4	RO	0xE0041FD0	是	0x04
外设 ID5	RO	0xE0041FD4	是	0x00
外设 ID6	RO	0xE0041FD8	是	0x00
外设 ID7	RO	0xE0041FDC	是	0x00
外设 ID0	RO	0xE0041FE0	是	0x24
外设 ID1	RO	0xE0041FE4	是	0xb9
外设 ID2	RO	0xE0041FE8	是	0x0b
外设 ID3	RO	0xE0041FEC	是	0x00
元件 ID0	RO	0xE0041FF0	是	0x0d
元件 ID1	RO	0xE0041FF4	是	0x90
元件 ID2	RO	0xE0041FF8	是	0x05
元件 ID3	RO	0xE0041FFC	是	0xb1

15.6.3　描述 ETM 寄存器

下节将进一步描述 ETM 寄存器。详见“*ARM 嵌入式跟踪宏单元架构规范*”。

ETM 控制寄存器

ETM 控制寄存器控制 ETM 的一般操作，例如是否使能跟踪。

复位值：0x00002411

实现位：21，17:16，13，11:4，0

所有其他位 RAZ，忽略写。

配置代码寄存器

ETM 配置代码寄存器让调试器可以读取特定实现的 ETM 配置。

复位值：0x8C800000

位 22:20 固定为 0，并且不是由 ASIC 提供。位 18:17 由 MAXEXTIN[1:0]输入总线提供，并读取 MAXEXTIN 的低位值和 2（EXTIN 的数目）号值。这表明：

- 支持软件访问
- 跟踪现存的开始/停止块
- 无 CID 比较器
- 无外部输出

- 0~2 外部输入（由 MAXEXTIN 控制）
- 无定序器（sequencer）
- 无计数器
- 无 MMD
- 无数据比较器
- 无地址比较器对

系统配置寄存器

系统配置寄存器显示 ASIC 支持的 ETM 特性。

复位值：0x00020D09

位 11:10 按正常情况实现。位 9，2:0 固定为 4’b0001。

跟踪使能（TraceEnable）控制 1 寄存器

跟踪使能（TraceEnable）控制 1 寄存器是其中一个用来配置 TraceEnable 的寄存器。只执行位 25。它对控制跟踪的开始/停止资源进行控制。

ETM ID 寄存器

ETM ID 寄存器保持 ETM 结构的派生（variant），并精确定义了 ETM 的编程器模型。

复位值：0x4114F240

这表明：

- ARM 实现者（implementor）
- 特殊的分支编码。影响每个字节的位 7:6
- 支持 Thumb-2
- 在其他地方发现内核系列
- ETMv3.4
- 实现版本 0

配置代码扩展寄存器

配置代码扩展寄存器保存 ETM 配置代码的其他位。它描述了扩展的外部输入。

复位值：0x00018800

该寄存器表明：

- 开始/停止模块使用 E-ICE 输入
- 4 个嵌入式 ICE 输入
- 不支持数据比较
- 所有寄存器都可读

不支持扩展的外部输入

第16章 嵌入式跟踪宏单元的接口

本章描述了“*嵌入式跟踪宏单元（ETM）*”接口。包含以下小节：

- *ETM 接口概述*
- *CPU ETM 接口的端口描述*
- *分支状态接口*

16.1 ETM 接口概述

ETM 接口可以轻易地实现 ETM 与处理器的连接。它为 ETM 提供了一个用来跟踪指令的通道。

16.2 CPU ETM 接口端口描述

处理器包含一个端口，该端口可以使 ETM 确定指令的执行序列。这些端口如表 16-1 所述。

表 16-1　ETM 接口端口

端口名称	方向	由谁限定	描述
ETMVALID	输出	-	指令执行有效。标志操作码已进入执行的第一个周期
ETMIBRANCH	输出	ETMIVALID	操作码是一个目标分支。标志当前代码是修改事件（分支，中断处理）的 PC 的目标
ETMINDBR	输出	ETMIBRANCH	操作码跳转目标是间接的。标志当前操作码是跳转目标，且该跳转目标的终点不能从 PC 内容推断出来。例如，LSU，寄存器移位或中断处理
ETMVALID	输出	无限定器	表明 DWT 观察到的当前数据地址在本周期有效
ETMICCFAIL	输出	ETMIVALID	操作码条件代码失败或通过。标志当前操作码通过或没通过其条件执行的检查。如果操作码是一个条件分支，或者是在 IT 块中发现的其他操作码，那么就有条件地执行该操作码。
ETMINTSTAT[2:0]	输出	无限定器	输入状态。标志当前周期的中断状态： 000　无状态 001　中断进入 010　中断退出 011　中断返回 100－向量读取和堆栈入栈 ETMINTSTAT 进入/返回在新中断上下文的第一个周期有效。无需 ETMIVALID 便可退出。
ETMINTNUM[8:0]	输出	ETMINTSTAT	中断号。标志当前执行上下文的中断号

续上表…

端口名称	方向	由谁限定	描述
ETMIA[31:1]	输出	-	指令地址。指示正在执行或最后执行的操作码的当前取指地址。可以通过检验以下端口来判断上下文： ETMIVALID HALTED SLEEPING ETM 在 ETMIVALID 和 ETMIBRANCH 有效时对 ETMIA[31:1]进行检验。DWT 对 ETMIA[31:1]进行检验，以实现 PC 采样和总线观察。
ETMFOLD	输出	ETMIVALID	操作码关闭。表明在这周期已将 IT 或 NOP 操作码关闭。PC 提前通过当前（16 位）操作码和 IT/NOP 指令（16 位）。这样会影响 ETMIA。
ETMFLUSH	输出	-	PC 事件的刷新标记。PC 变更操作码已经执行或者中断已经开始进栈/出栈。ETM 可以使用该控制来结束未完的数据包，为 ETMIBRANCH 事件作准备。
ETMFINDBR	输出	ETMFLUSH	刷新是间接的。标明“刷新提示目标不能从 PC 中推断出来”。
ETMCANCEL	输出	-	执行的当前操作码被取消。被中断的操作码在返回执行上下文时重新开始或继续执行。这些包括： LDR/STR LDRD/STRD LDM/STM U/SMULL MLA U/SDIV MSR CPSID

16.3　分支状态接口

分支状态信号 **BRCHSTAT**，提供与译码和接下来执行的操作码有关的取指时间信息。

图 16-1 和图 16-2 显示后跳条件分支未发生或已发生的情形。分支在操作码译码阶段随机发生。分支目标是半字不对齐的 16 位操作码。

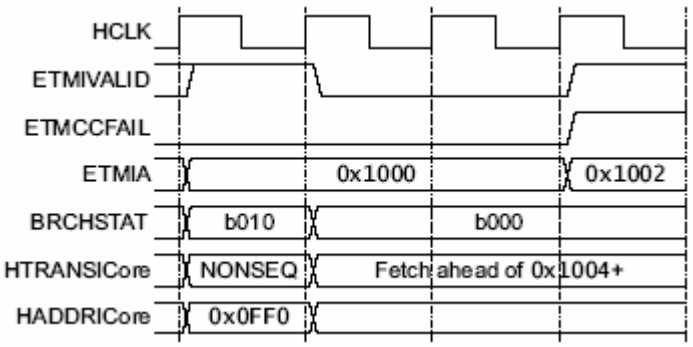


图 16-1　后跳条件分支未发生

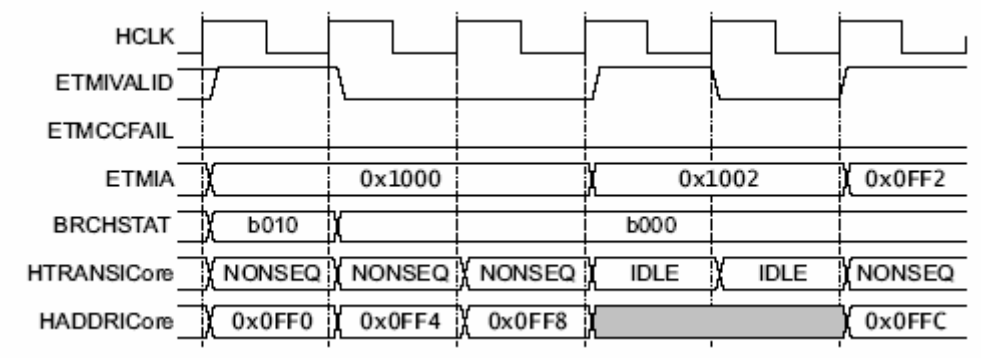


图 16-2　后跳条件分支已发生

注：

HADDRICore 和 **HTRANSICore** 是处理器的地址和处理请求信号，它不是外部 Cortex-M3 接口的信号。

图 16-3 和图 16-4 显示了前跳条件分支未发生和已发生的情形。分支在操作码的译码阶段随机产生。分支目标是半字不对齐的 16 位操作码。

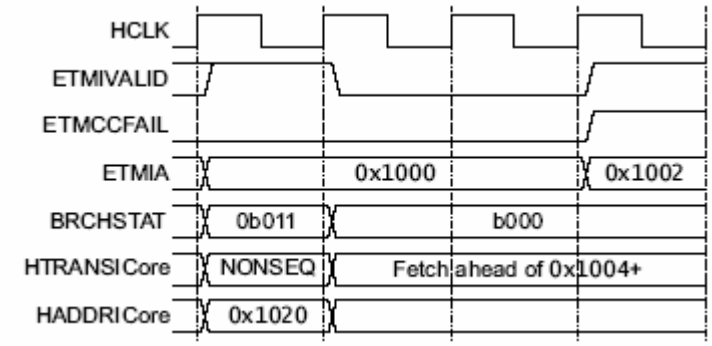


图 16-3　前跳条件分支未发生

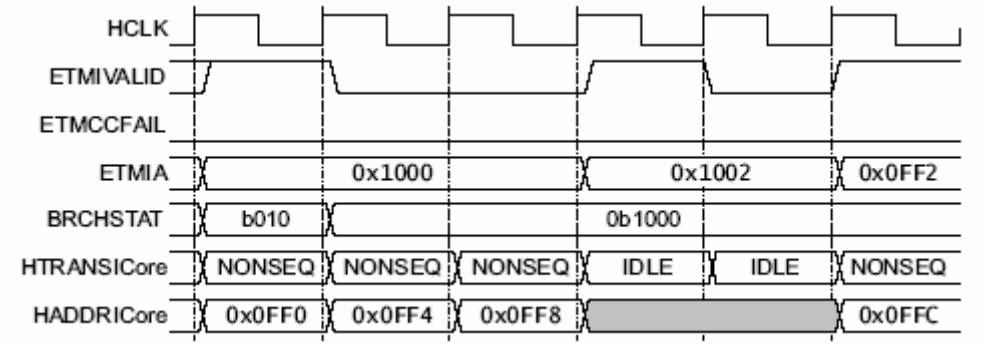


图 16-4　前跳条件分支已发生

图 16-5 和图 16-6 显示了这周期，在前一条操作码的执行阶段中，含流水线延迟和不含流水线延迟的无条件分支。分支在操作码译码阶段产生。分支目标是一个不对齐的 32 位操作码。

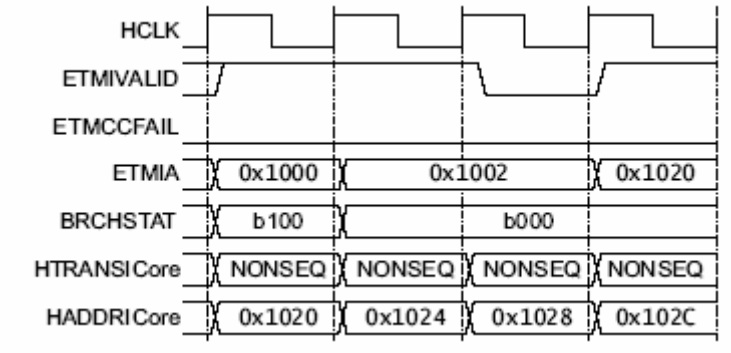


图 16-5　无流水线延迟的无条件分支

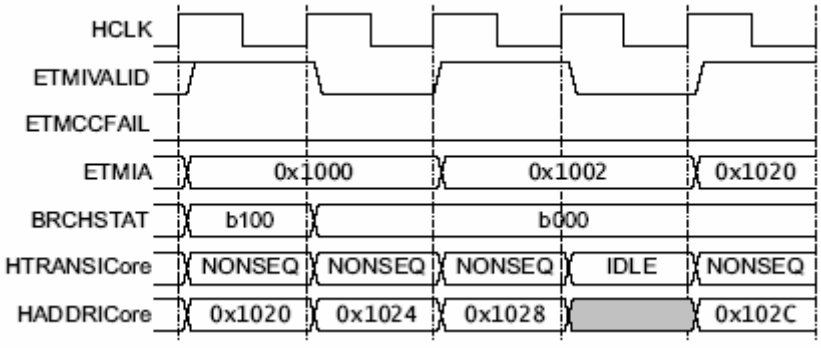


图 16-6　含流水线延迟的无条件分支

图 16-7 和图 16-8 显示了下一操作码的无条件分支。分支在执行操作码时发生。分支目标是一个对齐的和不对齐的 32 位 ALU 操作码。

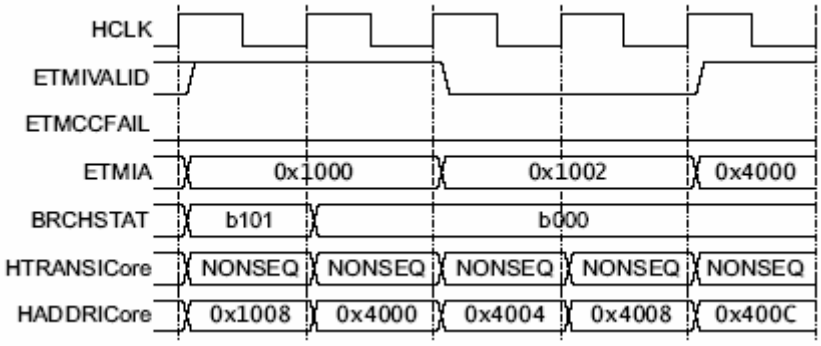


图 16-7　执行时的无条件分支（对齐的）

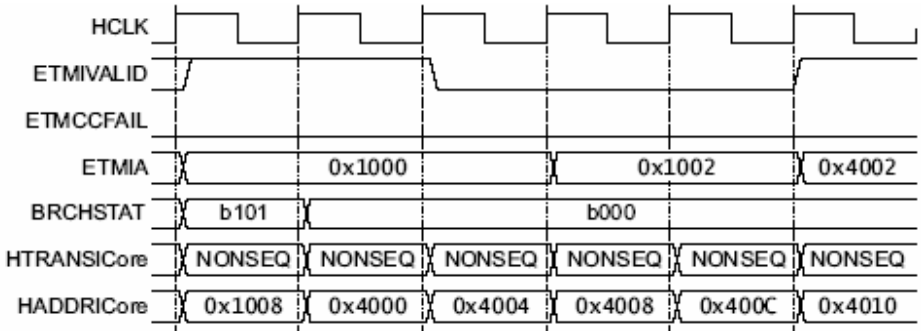


图 16-8　执行时的无条件分支（不对齐的）

第17章 指令周期定时

本章介绍了处理器的指令周期定时。包括以下小节：

- 关于指令周期定时
- 处理器的指令周期定时
- 加载/存储（Load-Store）指令周期定时

17.1 关于指令周期定时

本章提供的定时信息涵盖每一条指令和各条指令之间的相互作用。其中还包括影响定时的因素的相关信息。

在查看定时时要特别注意理解系统结构所扮演的角色。必须对每条指令取指，并且每条加载/存储（load/store）指令都必须传到系统那里。本章将对这些因素连同未来的系统设计（以及时序的推断）一起讨论。

17.2 处理器的指令周期定时

表 17-1 描述了 ARMv7-M 架构支持的 Thumb-2 子集。它提供了包括注释（解释指令流之间的相互作用如何影响时序）在内的周期信息。同时还考虑到了系统的影响，如从较慢存储器运行代码。

表 17-1　指令周期定时

指令类型	大小	周期数	描述
数据操作	16	1(+P^1 如果 PC 是目标)	ADC, ADD, AND, ASR, BIC, CMN, CMP, CPY, EOR, LSL, LSR, MOV, MUL, MVN, NEG, ORR, ROR, SBC, SUB, TST, REV, REVH, REVSH, SXTB, SXTH, UXTB 和 UXTH。MUL 是单周期指令。
跳转	16	1+P^1	B<cond>, B, BL, BX 和 BLX。注：BLX 指令不带立即数。如果发生跳转，流水线重载（增加了 2 个周期）
单寄存器加载/存储	16	2^2(+P^1 如果 PC 是目标)	LDR, LDRB, LDRH, LDRSB, LDRSH, STR, STRB 和 STRH,以及 "T" 变量.
多寄存器加载/存储	16	1+N^2(+P^1 如果 PC 被加载)	LDMIA, POP, PUSH 和 STMIA
异常发生	16	-	BKPT，在调试使能时程序停止运行，进入调试模式，如果调试被禁能则会产生故障。 SVC，产生 SVCall 异常，进入 SVCall 处理程序（详见“ARMv7 架构规范”）
带立即数的数据操作	32	1(+P^1 如果 PC 是目标)	ADC{S}. ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}和 MVN{S}。

续上表…

指令类型	大小	周期数	描述
带大立即数的数据操作	32	1	MOVW, MOVT, ADDW 和 SUBW。MOVW 和 MOVT 含一个 16 位立即数（所以可以取代存储器的 literal 加载操作）。 ADDW 和 SUBW 含一个 12 位立即数（所以同样可以取代许多存储器 literal 加载操作）
位操作	32	1	BFI, BFC, UBFX 和 SBFX。这些都是按位操作。允许对位的位置和大小进行控制。这些指令都支持 C/C++位域（在 structs 内）和许多比较赋值表达式和某些 AND/OR 赋值表达式。
带 3 个寄存器的数据操作	32	1(+P^1 如果 PC 是目标)	ADC{S}. ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}和 MVN{S}。 无 PKxxx 指令。
移位操作	32	1	ASR{S}, LSL{S}, LSR{S}和 ROR {S}
杂项(miscellaneous)	32	1	REV, REVH, REVSH, RBIT, CLZ, SXTB, SXTH, UXTB和 UXTH。扩展指令和相应的ARM v6的16位指令相同。
表格跳转		4+P^1	表格跳转指令，用于“Switch/case”语句。这些都是带移位的LDR操作，然后进行跳转。
乘法		1 或 2	MUL, MLA和 MLS。MUL是单周期指令，而MLA和 MLS是双周期指令。
运算结果为 64 位的乘法	16	3-7[3]	UMULL, SMULL, UMLAL和 SMLAL。周期数基于输入大小。即 ABS(输入) < 64K将及早结束。
加载/存储寻址		-	支持格式PC+/-imm12, Rbase+imm12, Rbase+/-imm8 和包括移位在内的已调节寄存器。"T"变量在特权模式时使用。
单寄存器加载/存储	32	2[2] (+P_1 如果PC是目标)	LDR, LDRB, LDRSB, LDRH, LDRSH, STR, STRB 和 STRH, 以及 "T" 变量。PLD是一个提示，所以如果没有缓存就将它当作NOP指令。
多寄存器加载/存储	32	1+N^2 (+P^1 如果PC被加载)	STM, LDM, LDRD, STRD, LDC 和 STC。
跳转	32	1+N^2	LDREX, STREX, LDREXB, LDREXH, STREXB, STREXH, CLREX。若没有局部监控程序（IMP DEF）将出错。LDREXD 和 STREXD 指令都没有包括在该类指令中。
特殊的加载/存储	32	1+P^1	B, BL 和 B<cond>。由于状态总是发生变化，所以无 BLX (1)。无BXJ。
系统	32	1	MSR(2) 和 MRS(2) 取代 MSR/MRS，但还有更多用处。这些指令都用来访问其他堆栈和状态寄存器。不支持CPSIE/CPSID的32位形式。 无RFE 或 SRS。

续上表…

指令类型	大小	周期数	描述
系统	16	1	CPSIE 和 CPSID 是 MSR(2)指令的快版本，使用标准的Thumb-2 译码技术，但是只允许使用 "i"和"f"，不允许使用"a"。
扩展的 32 位指令（Extended32）	32	1	NOP, 协处理器指令 (LDC, MCR, MCR2, MCRRMRC, MRC2, MRRC 和 STC) 和YIELD (看作NOP)。注：无 MRS (1), MSR (1) 和 SUBS (PC 返回链接)。
组合分支	16	1+P[1]	CBZ
扩展	16	0-1[4]	IT 和 NOP (包括 YIELD)。
除法	32	2-8[5]	SDIV 和 UDIV。 32/32 除带符号和无符号数，结果为32位的商(没有余数,该指令可以由减法指令实现)。当被除数和除数大小相近时，该指令将提前结束（early out）。
睡眠	32	1+W[6]	WFI, WFE 和 SEV 都属于用来控制睡眠行为的"可当作NOP"指令。
排序（barrier）	16	1+B[7]	ISB, DSB 和 DMB都是排序（barrier）类指令，这类指令确保某些动作在执行下一指令前已经发生。
饱和运算	32	1	SSAT 和 USAT 在一个寄存器上执行饱和运算。它们执行3个任务：通过移位操作将值规格化；检测所选位单元（Q值）的溢出情况，并且如果溢出，将置位xPSR Q位；如果检测到溢出，就使该值饱和。饱和运算与所选位数的最大无符号数或最大/最小有符号数有关。

周期数的相关信息：

- P=流水线重载
- N=元素（element）的数量
- W=睡眠等待
- W=睡眠等待

通常，每条指令的执行都要花费一个周期（一个内核周期），如上文所述。由于取指延迟所以还要多花费几个周期。

5. 分支指令需一个周期来执行指令，然后为目标指令进行流水线重载。不执行分支总共为 1 个周期。带立即数的执行分支进行流水线重载一般要一个周期(共 2 个周期)。而带有寄存器操作数的执行分支进行流水线重载一般要 2 个周期（共 3 个周期）。流水线重载在跳转到不对齐的 32 位指令，或者在访问较慢存储器时所要的时间更长。分支提示被发送到代码总线，见 14.2.1 小节，它允许较低系统进行预加载；这样可以减少分支目标为较慢存储器的损失（penalty），但是永远不会比此处给出的少。
6. 通常，加载/存储指令在进行第一次访问时要 2 个周期，额外的访问要一个周期。
7. UMULL/SMULL/UMLAL/SMLAL 根据源值的大小来使用提早中止。这些都是可中断的（被丢弃/重新开始），出现最坏情况时延迟为一个周期。MLAL 类要花费 4 到

7 个周期，而 MULL 类要花费 3 到 5 个周期。在这两种情况下，带符号类指令要比无符号类指令多花费 1 个周期。

8. DIV 的时序取决于被除数和除数。DIV 是可中断的（被丢弃/重新开始），出现最坏情况时延迟为一个周期。当被除数和除数大小相近时，除法操作将提早结束。当除数大于被除数以及除数为 0 时所花费的时间最短。尽管调试陷阱可以捕获这种情形，但除数为 0 时将返回 0（不是出错）。
9. 睡眠指令的执行需 1 个周期，再加上相应的几个睡眠周期。WFE 在事件已经发生时只使用一个周期。WFI 指令执行通常大于一个周期，除非在进入 WFI 时产生中断，刚好将其挂起。
10. ISB 为单周期指令（当作分支）。DMB 和 DSB 将花费一个周期，除非数据在写缓冲区或 LSU 中挂起。如果在排序（barrier）过程中发生中断，它将被丢弃或重新开始。

17.3 加载/存储（Load-store）执行时序

本小节将描述如何最佳地将指令配对以进一步精简时序。

- STR Rx,[Ry,#imm]执行总要一个周期。这是因为地址的产生发生在最初的周期内，而数据的存储正好在执行下条指令时发生。如果存储发生在存储缓冲区，但存储缓冲区已满，那么下条指令将被延迟，直至完成存储操作。如果存储不是发生在存储缓冲区（如发生在代码段），并且该执行将停止，那么只有当另外的加载或存储操作在结束前执行才能感觉到其对时序的影响。
- LDR Rx!,[any]不执行常规的流水线操作。即基址更新加载（base update load）操作通常至少要 2 个周期（如果被延迟则相应的增加）。但是如果下条指令不需要从寄存器中读取时，加载操作将减少至一个周期。非寄存器读取指令包括 CMP，TST，NOP，不执行 IT 控制指令，等等。
- LDR PC,[any]一直是一个分块操作。即加载至少要 2 个周期，流水线重载至少要 3 个周期。所以至少要 5 个周期（如果加载或取指被中止则相应的增加）。
- 由于存在取指单元争用情况，所以 LDR Rx,[PC,#imm]可能要增加一个周期。
- TBB 和 TBH 也是分块操作。加载至少要 2 个周期，加法至少要 1 个周期，流水线重载至少要 3 个周期。即至少要 6 个周期（如果加载或取指被中止则相应的增加）。
- LDR any 如有可能也可以执行流水线操作。即如果下条指令是 LDR 或非基址更新（non-base updating）的 STR，且第一条 LDR 的目标不用于计算下条指令的地址，那么下条指令可以少一个周期。因此，LDR 之后可以跟一条 STR 指令，这样 STR 将 LDR 加载的内容写出来（write out）。此外，多条 LDR 可以一起执行流水线操作。以下便是一些经过优化的实例：
  - LDR R0,[R1]; LDR R1,[R2]　通常总共为 3 个周期
  - LDR R0,[R1,R2]; STR R0,[R3,#20]　　通常总共为 3 个周期
  - LDR R0,[R1,R2]; STR R1,[R3,R2]　　通常总共为 3 个周期
  - LDR R0,[R1,R5]; LDR R1,[R2]; LDR R2,[R3,#4]　　通常总共为 4 个周期
- 后面的 STR any 可能不知执行流水线操作。只有当 STR 在 LDR 指令之后时，STR 才能执行流水线操作，但是在存储后可能所有指令都不执行流水线操作。因为存储缓冲区（用于 bit band，数据段和不对齐）的原因，即使是一条被延迟的 STR 它一

般也要花费 2 个周期。此外，如果 STR 之后紧跟着一个 ALU 操作（不是加载或存储），那么处理器将立即清空加载操作。因此如果可能，在 STR 指令后应该尽量避免执行加载/存储（Load/store）操作。

- LDREX 和 STREX 和 LDR 一样可以执行流水线操作。因为 STREX 更象 LDR，所以它可以象 LDR 一样执行流水线操作。同样地，LDREX 也象 LDR，因此也可以执行流水线操作。
- LDM，STM，LDC，STC 不能和前一条或下一条指令一起执行流水线操作。但是，第一个元素后面的所有元素都一起执行流水线操作。因此，一条 3 元素的 LDM 如果没有被延迟，那么将要花费 2+1+1 或 5 个周期。当被中断时，LDM 和 STM 指令在返回时将从停止的地方继续。继续操作在开始第一个元素时将增加 1 个或 2 个周期。
- “不对齐的字或半字加载/存储”将增加代价周期（penalty cycle）。字节对齐的半字加载或存储操作将额外增加一个周期，当作两个字节来执行。半字对齐的字加载/存储操作将额外增加一个周期，当作两个半字来执行。字节对齐的字加载/存储将额外增加 2 个周期，当作 1 个字节、1 个半字以及 1 个字节来执行。如果发生存储器延迟，这些值都将增加。因为存储缓冲区的关系，STR 或 STRH 都不会延迟处理器。

附录A 信号描述

本附录列出并描述了处理器接口信号。包括以下小节：

- *时钟*
- *复位*
- *杂项（miscellaneous）*
- *中断接口*
- *ICode 接口*
- *DCode 接口*
- *系统总线接口*
- *专用外设总线接口*
- *ITM 接口*
- *AHB-AP 即可*
- *ETM 接口*
- *测试接口*

A.1 时钟

表 A-1 列出了时钟信号。

表 A-1　时钟信号

名称	方向	描述
HCLK	输入	Cortex-M3 主时钟
FCLK	输入	Cortex-M3 自由振荡时钟
DAPCLK	输入	AHB-AP 时钟

A.2 复位

表 A-2 列出了复位信号。

名称	方向	描述
PORESETn	输入	上电复位。复位整个 Cortex-M3 系统
SYSRESETn	输入	系统复位。复位处理器、NVIC 的非调试部分、总线矩阵和 MPU。调试元件除外
SYSRESETREQ	输出	系统复位请求信号
DAPRESETn	输入	AHB-AP 复位

A.3 杂项

表 A-3 列出了剩余的其他信号。

表 A-3　其他信号

名称	方向	描述
LOCKUP	输出	指示内核被加锁
SLEEPDEEP	输出	指示 Cortex-M3 时钟可以停止了
SLEEPING	输出	指示 Cortex-M3 时钟可以停止了
CURRPRI[7:0]	输出	指示当前使用的是什么优先级中断。**CURRPRI** 表示占先优先级，不表示次级优先级
HALTED	输出	处于“停止调试”模式。内核处于调试状态时 **HALTED** 保持有效
TXEV	输出	由于执行 SEV 指令而发送事件。这是单周期脉冲
TRCENA	输出	跟踪使能。该信号反映“调试异常与监控寄存器”的位[24]的设置。该信号可以用来对 TPIU 和 ETM 块的时钟进行门控，以便在禁止跟踪时减少功耗
BIGEND	输入	静态的字节排列顺序选择： 1＝ 大端（高位在前） 0＝ 小端（低位在前） 该信号复位时被采样，并且在复位失效时不能变动
EDBGRQ	输入	外部调试请求
PPBLOCK[5;0]	输入	保留。必须与 b000000 相连
STCLK	输入	系统时钟节拍（Tick）
STCALIB[25:0]	输入	系统时钟节拍（Tick）校准
RXEV	输入	通过 WFE 指令引起唤醒
VECTADDR[9:0]	输入保留	必须与 b0000000000 相连
VECTADDREN	输入保留	必须与 b0 相连

A.4 中断接口

表 A-4 列出了外部中断接口的信号。

表 A-4　中断接口

名称	方向	描述
INTISR[239:0]	输入	外部中断信号
INTNMI	输入	非屏蔽中断

A.5 ICode 接口

表 A-5 列出了 ICode 接口的信号。

表 A-5　ICode 接口

名称	方向	描述
HADDRI[31:0]	输出	32 位指令地址总线
HTRANSI[1:0]	输出	指示当前传输的是空闲（IDLE）的还是非连续的（NONSEQUENTIAL）
HSIZEI[2:0]	输出	指示取指的长度。Cortex-M3 中所有取指都是 32 位操作
HBURSTI[2:0]	输出	指示传输是否为突发传输的一部分。Cortex-M3 将所有取指和 literal 装载都看作单次（SINGLE）操作
HPROTI[3:0]	输出	提供有关访问信息。总是指示该总线可高速缓存和可缓冲。 HPROTI[0]＝ 0 表示取指 HPROTI[0]＝ 1 表示取向量
MEMATTRI[1:0]	输出	存储器属性。对于 ICode 总线，这两个位的值总是为 0（不分配，不共享）
BRCHSTAT[3:0]	输出	就现存或即将到来的 AHB 取指提供提示信息。条件操作码可以是随机的，也可以随后丢弃。 0000　无提示 0001　译码时后跳的条件分支 0010　译码时的条件分支 0011　执行时的条件分支 0100　译码时的无条件分支 0101　执行时的无条件分支 0110　保留 0111　保留 1000　译码发生时的条件分支（**IHTRANS** 之后的周期） 1001…1111　保留
HRDATA[31:0]	输入	指令读总线
HREADYI	输入	HIGN 表示传输在总线上已经结束。该信号被驱至低电平（LOW）以扩展传输
HRESPI[1:0]	输入	传输响应状态。OKAY 或 ERROR

A.6 DCode 接口

表 A-6 列出了 DCode 接口的信号。

表 A-6　DCode 接口

名称	方向	描述
HADDRD[31:0]	输出	32 位数据地址总线
HTRANSD[1:0]	输出	指示当前传输是空闲的（IDLE），还是非连续的（NONSEQUENTIAL）或者是连续的（SEQUENTIAL）。
HWRITED	输出	写，不是读
HSIZED[2:0]	输出	指示访问的长度。可以是 8 位、16 位或 32 位
HBURSTD[2:0]	输出	指示传输是否为突发传输的一部分。在 Cortex-M3 中数据访问被当作 INCR 执行
HPROTD[3:0]	输出	提供有关访问的信息
EXREQD	输出	专门请求

续上表…

名称	方向	描述
MEMATTRD[1:0]	输出	存储器属性。 位 0＝ 分配 位 1＝ 可共享
HWDATAD[31:0]	输入	数据读总线
HREADYD	输入	HIGH 表示传输在总线上已结束。该信号被驱至低电平以扩展传输
HRESPD[1:0]	输入	传输响应状态。正确（OKAY）或出错（ERROR）
HRDATAD[31:0]	输入	读数据
EXRESPD	输入	专门响应

A.7 系统总线接口

表 A-7 列出了系统总线接口的信号。

表 A-7　系统总线接口

名称	方向	描述
HADDRS[31:0]	输出	32 位地址
HTRANSS[1:0]	输出	指示当前传输的类型。可以是空闲的（IDLE），还是非连续的（NONSEQUENTIAL）或连续的（SEQUENTIAL）。
HSIZES[2:0]	输出	指示访问的长度。可以是 8 位、16 位或 32 位
HBURST[2:0]	输出	指示传输是否为突发传输的一部分。
HPROTS[3:0]	输出	提供有关访问的信息
HWDATAS[31:0]	输出	32 位写数据总线
HWRITES	输出	写，不是读
HMASTLOCKS	输出	表明总线上的处理必须是原子操作。这仅用于 bit-band 写操作（当作读-修改-写执行）
EXREQS	输出	专门请求
MEMATTRS[1:0]	输出	存储器属性。 位 0＝ 分配 位 1＝ 可共享
HRDATAS[31:0]	输入	读数据总线
HREADYS	输入	HIGH 表示传输在总线上已结束。该信号被驱至低电平以扩展传输
HRESPS[1:0]	输入	传输响应状态。OKAY 或 ERROR
EXRESPS	输入	专门响应

A.8 专用外设总线接口

表 A-8 列出了 DCode 接口的信号。

表 A-8　专用外设总线接口

名称	方向	描述
PADDR[19:2]	输出	17 位地址。只驱动与外部专用外设总线有关联的位。
PADDR31	输出	该信号在 AHB-AP 为请求主设备时被驱至高电平（HIGH）。当 DCore 为请求主设备时被驱至低电平（LOW）
PSEL	输出	指示请求数据传输
PENABLE	输出	通过选通为所有访问操作计时。用来指示 APB 传输的第二个周期
PWDATA[31:0]	输出	32 位写数据总线
PWRITE	输出	写，不是读
PRDATA[31:0]	输入	读数据总线

A.9 ITM 接口

表 A-9 列出了 ITM 接口的信号。

表 A-9　ITM 接口

名称	方向	描述
ATVALID	输出	ATB 有效
AFREADY	输出	ATB 刷新
ATDATA[7:0]	输出	ATB 数据
ATIDITM[6:0]	输出	TPIU 的 ITM ID
ATREADY	输入	ATB 准备就绪

A.10 AHB-AP 接口

表 A-10 列出了 AHB-AP 接口的信号。

表 A-10　AHB-AP 接口

名称	方向	描述
DAPRDATA[31:0]	输出	在读周期过程（当 DAPWRITE 为低电平时）读总线被所选的 AHB-AP 驱动
DAPREADY	输出	AHB-AP 使用该信号来扩展 DAP 传输
DAPSLVERR	输出	错误响应是因为： ● 主机端口产生错误响应，或由于 DAPEN 阻止传输，而使得传输没有启动 ● 在 DAPABORT 操作后不接受对 AP 寄存器的访问
DAPCLKEN	输入	DAP 时钟使能（节电）
DAPEN	输入	AHB-AP 使能
DAPADDR[31:0]	输入	DAP 地址总线
DAPSEL	输入	选择从 DAP 译码器产生的信号，并提供给 AP。该信号指示是否选择了从机设备，并且是否需要数据传输。每个从机都有一个 **DAPSEL** 信号。该信号是由驱动的 DP 产生。译码器监控地址总线，并使相应的 **DAPSEL** 变为有效
DAPENABLE	输入	该信号用来指示从 DP 到 AHB-AP 的 DAP 传输的第二个和之后的周期
DAPWRITE	输入	HIGH 表示从 DP 到 AHB-AP 的 DAP 写访问。LOW 表示读访问
DAPWDATA[31:0]	输入	写总线在写周期（当 **DAPWRITE** 为 HIGH 时）被 DP 块驱动

续上表…

名称	方向	描述
DAPABORT	输入	中止当前的传输。AHB-AP 让返回高电平（HIGH）的 **DAPREADY**，而不会影响 AHB 主机端口中正在进行的传输的状态

A.11 ETM 接口

表 A-11 列出了 ETM 接口的信号

名称	方向	描述
ETMTRIGGER[3:0]	输出	DWT 的触发信号。4 个 DWT 比较器每个比较器一位
ETMTRIGINOTD[3:0]	输出	指示 ETM 是在指令匹配还是数据匹配时触发
ETMIVALID	输出	指令有效
ETMIA[31:1]	输出	正在执行的指令的 PC
ETMICCFAIL	输出	条件代码失败。指示当前指令是通过还是未通过其条件执行检验
ETMIBRANCH	输出	操作码是分支目标
ETMIINDBR	输出	操作码是一个间接的分支目标
ETMINTSTAT[2:0]	输出	中断状态。指示当前周期的中断状态。 000　无状态 001　中断进入 010　中断退出 011　中断返回 100　向量读取和压栈 **ETMINTSTAT** 进入/返回在新中断上下文的第一个周期是有效的。无需 **ETMIVALID** 信号就可时发生。
ETMINTNUM[8:0]	输出	指示当前执行上下文的中断号
ETMFLUSH	输出	PC 修改操作码已经执行，或者中断入栈/出栈已经开始
ETMPWRUP	输出	ETM 被使能
ETMDVALID	输出	数据有效
ETMCANCEL	输出	取消的指令
ETMFINDBR	输入	刷新是间接的。指示刷新提示目标（flush hint destination）不能从 PC 推断出来
ETMFOLD		操作码关闭。一个 IT 或 NOP 操作码在本周期已被关闭。PC 提前通过当前（16 位）操作码和 IT/NOP 指令（16 位）。这在 ETMIA 中反映
DSYNC		DWT 的同步脉冲

A.12 测试接口

表 A-12 列出了测试接口的信号。

表 A-12　测试接口

名称	方向	描述
SE	输入	扫描使能
RSTBYPASS	输入	扫描测试的复位旁路。**PORESETn** 是扫描测试过程使用的唯一一个复位

附录B 术语表

这部分内容描述了 ARM 公司的技术文档中用到的部分术语。

中止（Abort）

向内核指示所尝试的存储器访问无效或不允许，或者存储器访问返回的数据无效的机制。中止能够由外部或内部存储器系统产生，作为试图对无效或受保护的指令或数据存储器进行访问的结果。

也请参考数据中止、外部中止和预取中止。

寻址方式

由多条不同的指令共享，用于产生供指令使用的值的各种机制。

先进的高性能总线（AHB）

AHB 是在地址/控制阶段和数据阶段之间具有固定流水线的总线协议。它只支持由 AMBA AXI 协议提供的功能性的子集。完全的 AMBA AHB 协议规范包括许多对于主机和从机 IP 开发来说不是必需的属性，因此 ARM 公司建议通常只使用该协议的子集。这个子集定义为 AMBA AHB-Lite 协议。

也请参考先进的微控制器总线架构和 AHB-Lite。

先进的微控制器总线架构（AMBA）

AMBA 是描述互连策略的协议规范系列。它是 ARM 公开的片内总线标准，该片内总线规范详述了互连策略以及对组成片上系统（SoC）的功能模块的管理。它协助带有一个或多个 CPU 或信号处理器和多个外设的嵌入式处理器的开发。AMBA 通过定义一个用于 SoC 模块的通用骨干（backbone）来对可重复使用的设计方法作了补充。

先进的外设总线（APB）

APB 是一个比 AXI 和 AHB 相对简单的总线协议。它是为配件，或诸如定时器、中断控制器、UART、I/O 口等通用外设而设计的。它与主要系统总线的连接是通过系统到外设总线桥来实现的，这样有助于降低系统功耗。

AHB

见先进的高性能外设总线

AHB 访问端口（AHB-AP）

AHB-AP 是 DAP 的一个可选组件，提供到 SoC 的 AHB 接口。

AHB-AP

见 AHB 访问端口

AHB-Lite

AHB-Lite 是完全的 AMBA AHB 协议规范的子集。它提供大多数 AMBA AHB 从机和主机设计所必需的所有基本功能，尤其是与多层 AMBA 互连一起使用时。在大多数情况下，

通过使用 AMBA AXI 协议接口可以高效地实现由完全 AMBA AHB 接口提供的额外的便利。

对齐

保存在能够被定义数据大小的字节数整除的地址中的数据项（data item）称作是对齐的。对齐的字和半字其地址能够分别被 4 和 2 整除。因此，术语字对齐和半字对齐规定，其地址能够分别被 4 和 2 整除。

AMBA

见先进的微控制器总线架构

先进的跟踪总线（ATB）

跟踪设备用来共享 CoreSight 捕获资源的总线。

APB

见先进的外设总线

专用集成电路（ASIC）

ASIC 是指用来执行一个特定应用功能的集成电路。它可以是定制的或量产的。

专用标准器件/产品（ASSP）

ASSP 是指用来执行一个特定应用功能的集成电路。它通常由两个或两个以上分立电路组成，其功能结合作为一个构建模块（building block），适合一个或多个特定应用市场中的大部分产品使用。

架构

架构是描述处理器及其附属组件特性的硬件和/或软件构造，并且在描述带有类似特性的器件的行为时将这些器件集合在一起。例如，哈佛架构，指令集架构，ARMv7-M 架构。

ARM 指令

ARM *指令集架构*（ISA）的指令。Cortex-M3 不能执行 ARM 指令。

ARM 状态

处理器执行 ARM ISA 指令时所处的处理器状态。处理器只在 Thumb 状态下，而不在 ARM 状态下工作。

ASIC

见专用集成电路

ASSP

见专用标准产品

ATB

见先进的跟踪总线

ATB 桥

同步 ATB 桥提供寄存器片（register slice）来简化通过增加的流水线等级时的时序收敛（timing closure）。它还在两个同步的 ATB 域之间提供单向链接。

异步 ATB 桥在含异步时钟的两个 ATB 域之间提供单向链接。其目的是使得含存在于不同时钟域的 ATB 端口的组件能够连接。

基址寄存器

基址寄存器是指由加载或存储指令指定，用来保存进行指令地址计算的基址值的寄存器。根据指令及其寻址方式，在基址寄存器值上加上或减去偏移量来形成发送到存储器的地址。

基址寄存器写回

对在指令目标地址计算中使用的基址寄存器的内容进行更新。这样，修改后的地址变为存储器中的下一个更高或更低的连续地址。这意味着没必要为连续的指令传输取目标地址，从而能够更快地对连续存储器进行突发访问。

节拍（Beat）

用于突发传输中的一次数据传输的另一个字。例如，INCR4 突发由 4 个节拍组成。

BE-8

字节不变（byte-invariant）系统中存储器的大端视图。

也请参考 BE-32，LE，字节不变（byte-invariant）和字不变（Word-invariant）。

BE-32

字不变（Word-invariant）系统中存储器的大端视图。也请参考 BE-8，LE，字节不变和字不变。

大端（big-endian）

字节定序机制，在大端格式中，一个数据字中的递减有效字节存放在存储器的递增地址中。

也请参考小端（little-endian）和字节顺序（endianness）

大端存储器（big-endian memory）

在大端存储器中：

- 字对齐地址中的字节或半字为该地址的字中的最高有效字节或半字。
- 半字对齐地址中的字节为该地址的半字中的最高有效子字节。

也请参考小端存储器（little-endian memory）

边界扫描链

边界扫描链由使用标准 JTAG TAP 接口实现边界扫描技术的串行连接的器件组成。每个器件至少含有一个包含移位寄存器的 TAP 控制器，这些移位寄存器形成一条链，连接在 **TDI** 和 **TDO** 之间，通过它将测试数据移位。处理器可包含几个移位寄存器，使你能够对所选的

器件部分进行访问。

分支消除（branch folding）

在分支消除技术中，从呈现给执行流水线的指令流中将分支指令完全移除。

断点

断点是由调试器提供，来识别程序的执行将被停止的指令的一种机制。断点由程序员插入来使能程序执行过程中寄存器内容，存储单元，固定点的变量值的检查以便测试程序是否正确运行。断点在程序成功测试之后移除。

也请参考观察点。

突发（burst）

对连续地址的一组传输。因为地址是连续的，所以不需要为第一次传输之后的任意传输提供地址。这可以增加进行该组传输的速度。通过使用信号来指示突发长度以及地址的递增方式，来对 AMBA 上的突发传输进行控制。

也请参考节拍（beat）

字节

一个 8 位的数据项。

字节不变（byte-invariant）

对于字节不变的系统，当在小端和大端操作之间进行切换时，存储器每个字节的地址保持不变。当执行加载和存储操作的数据项大于 1 个字节时，根据存储器访问的字节顺序（endianness）以正确的顺序对该数据项执行字节屏蔽。

在 ARMv6 和后面的版本中，ARM 架构支持字节不变的系统。当选择字节不变支持时，非对齐的半字和字存储器访问也支持。多字访问要求是字对齐的。

也请参考字不变。

时钟门控

利用一个控制信号对宏单元的时钟信号进行门控，并使用修改后时钟对宏单元的工作状态进行控制。

每条指令的时钟数（CPI）

见每条指令的周期数（CPI）

冷复位

也称为上电复位。

上下文

在多任务操作系统中，每个处理执行时的环境。

也请参考快速上下文切换。

内核

内核是处理器的一部分，包含 ALU，数据路径，通用寄存器，程序计数器以及指令译码和控制电路。

内核复位

见热复位

CoreSight

对一个完整的片内系统进行监控，跟踪，和调试的基础结构。

CPI

见每条指令的周期数。

每条指令的周期数（CPI）

每条指令的周期数是一个测量单位，表示在一个时钟周期内能够执行的计算机指令的数目。这个数字能用来对执行相同指令集的不同 CPU 的性能进行比较。值越低，性能越好。

数据中止

数据中止为从存储器系统到内核的一个指示，表示试图访问一个非法数据存储单元。如果处理器试图使用引起中止的这个数据，则必定会出现异常。

也请参考中止。

DCode 存储器

0x00000000~0x1FFFFFFF 的存储器空间。

调试访问端口（DAP）

DAP 是一个 TAP 模块，担当 AMBA、AHB 或 AHB-Lite、访问系统总线的主机的角色。DAP 是一个专用名词，包含支持系统广泛调试的组合模块（modular block）集。DAP 是一个组合元件，目的是可以扩展以便通过单一的调试接口实现对多个系统（例如，映射到 AHB 和 CoreSight APB 的存储器）的可选访问的支持。

调试器

调试器是含有程序的调试系统，连同支持软件调试的自定义硬件一起，用来检测、定位以及修正软件故障。

嵌入式跟踪宏单元（ETM）

ETM 是一个硬件宏单元，当与处理器内核相连时，在跟踪端口上输出指令跟踪信息。

字节顺序（endianness）

字节定序。该机制用来确定存储在存储器中的一个数据字的连续字节的顺序，是系统存储器映射的一个方面。

也请参考小端和大端。

ETM

见嵌入式跟踪宏单元

异常

异常是一个错误或事件，能够使处理器挂起当前正在执行的指令流，并执行一个指定的异常处理程序或中断服务程序。异常可以是一个外部中断或 NMI，或者可以是一个相当严重，需要打断程序执行过程的故障或错误事件。例如，试图执行一次无效的存储器访问、外部中断、以及未定义指令。当发生异常时，正常的程序流程被打断，并在对应的异常向量处恢复执行。异常向量包含进行异常处理的中断服务程序的第一条指令。

异常处理程序

见中断服务程序

异常向量

见中断向量

外部 PPB

0xE0040000~0xE00FFFFF 的 PPB 存储器空间

Flash 修补和断点单元（FPB）

FPB 是一组地址匹配标签（tag）。它将对 flash 的访问变为对 SRAM 的特定部分的访问。这样可以为断点和快速地固定或改变来修补 flash 单元。

格式程序（formatter）

格式程序是 ETB 和 TPIU 的内部输入模块，它将跟踪源 ID 嵌入到数据中来创建一个跟踪流。

半字

一个 16 位的数据项。

停止模式

两种相互排斥的调试模式中的一种。在停止模式中，当遇到断点或观察点时，所有的处理器操作停止。所有的处理器状态、协处理器状态、存储器以及输入/输出单元都能够通过 JTAG 接口来检查和改变。

也请参考监控调试模式.

主机

向另一个计算机提供数据和其它服务的计算机。尤其是向正在被调试的目标提供调试服务的计算机。

ICode 存储器

0x00000000~0x1FFFFFFF 的存储器空间。

非法指令

结构上未定义的指令。

实现定义的（implementation-defined）

行为不是在结构上定义的，而是由单独的实现定义和证明的。

实现特定的（implementation-specific）

行为不是在结构上定义的，并且不需要由单独的实现来证明。当有多个实现选项可供选择，且所选选项不会影响软件的兼容性时，我们就使用该“实现特定的（implementation-specific）”。

指令周期数

一条指令占用流水线的执行阶段所消耗的周期数。

仪表跟踪

通过一个简单的存储器映射跟踪接口来对实时系统进行调试的一个组件，提供 printf 类型调试。

智能能量管理（IEM）

使用动态电压调整和可变的时钟频率来降低器件内的功耗的一种技术。

内部 PPB

0xE0000000~0xE003FFFF 的 PPB 存储器空间。

中断服务程序

是一个程序，当中断产生时，处理器的控制传递到这个程序。

中断向量

存储器低地址空间中的一个固定地址，含有对应中断服务程序的第一条指令。

联合测试行动组（JTAG）

联合测试行动组是发展标准 IEEE 1149.1.的机构的名称。该标准定义了用于集成电路器件的在线测试的边界扫描架构。通常采用其首字母 JTAG 来称呼。

JTAG

见联合测试行动组。

JTAG 调试端口（JTAG-DP）

JTAG-DP 是 DAP 的一个可选的外部接口，为调试访问提供一个标准的 JTAG 接口。

JTAG-DP

见 JTAG 调试端口。

LE

字节不变和字不变系统中的存储器小端视图。也请参考字节不变、字不变。

小端

一种字节定序机制。在小端格式中，一个数据字中的递增有效的字节存放在存储器的递增地址中。

也请参考大端和字节顺序

小端存储器

在小端存储器中：

- 字对齐地址中的字节或半字为该地址的字中的最低有效字节或半字。
- 半字对齐地址中的字节为该地址的半字中的最低有效字节。

也请参考大端存储器。

加载/存储架构

一种处理器架构，在加载/存储架构中，数据处理只能对寄存器内容进行操作，不能直接对存储器内容进行操作。

加载存储单元（LSU）

LSU 是处理器的一部分，用来处理加载和存储传输。

LSU

见加载存储单元。

宏单元

宏单元是具有自定义接口和行为的复杂逻辑模块。一个典型的 VLSI 系统就是由几个宏单元（例如处理器、ETM、存储器模块）和针对特定应用的逻辑组成的。

存储器一致性（memory coherency）

如果数据读操作的值或指令取指的值为最近写入该单元的值，那么存储器是一致的。当涉及到多个可能的物理单元时，例如具有主存储器、写缓冲区和高速缓存的系统，存储器一致性的实现会变得更加困难。

存储器保护单元（MPU）

MPU 是对存储器模块的访问权限进行控制的硬件。与 MMU 不同的是，MPU 不修改地址。

微处理器

见处理器

监控调试模式

两种相互排斥的调试模式中的一种。在监控调试模式中，处理器使能由调试监控或操作系统测试任务提供的软件中止处理器。在遇到断点或观察点时，它能够在正常的程序执行被

挂起时使至关重要的系统中断能够继续得到响应。

也请参考停止模式。

MPU

见存储器保护单元。

多层

多层是与交叉矩阵（cross-bar）类似的一种互连策略。互连上的每个主机都有一条能够到每个从机的直接链接。该链接不与其它主机共享。这使得每个主机能够与其它主机并行处理传输。竞争现象只在有效载荷目标端的一个多层互连中发生，通常是从机。

损失（penalty）

由于指令流与假定的或预测的不同，而使得流水线的执行阶段没有产生有用活动的周期数。

上电复位

见冷复位。

PPB

见专用外设总线。

预取

预取是指在具有流水线的处理器中，先前的指令已完成执行之前，从存储器中取指来填充流水线的处理。预取指令并不表示指令必须执行。

预取中止

存储器系统向内核指示，指令已从非法的存储单元取出。如果处理器试图执行该指令，则必定出现异常。预取中止能够由外部或内部存储器系统引起，作为试图访问无效的指令存储器的结果。

也请参考数据中止、中止。

专用外设总线

0xE0000000~0xE00FFFFF 的存储器空间。

处理器

处理器是计算机系统中使用计算机指令处理数据的一个必需电路。它是微处理器的缩写。在创建一个完全可工作的最小计算机系统时，还必须有时钟源，电源，和主存储器。

RealView ICE

RealView ICE 为使用 JTAG 接口调试嵌入式处理器内核的一个系统。

保留

如果控制寄存器或指令格式中的区域将由实现来定义，或者如果该区域的内容非零时产生不可预测的结果，则该区域被保留。这些保留区域可供将来的架构扩展使用，或者供特定

实现使用。实现没有使用的所有保留位必须写 0 或读作 0。

SBO

见应为 1

SBZ

见应为 0

SBZP

见应为 0 或被保护。

扫描链

扫描链由使用标准 JTAG TAP 接口执行边界扫描技术的串行连接的器件组成。每个器件至少含有一个 TAP 控制器，该控制器中的移位寄存器形成连接在 **TDI** 和 **TDO** 之间的扫描链。测试数据能够通过该扫描链来移位。处理器可包含几个移位寄存器，从而能对所选的器件部分进行访问。

应为 1（SBO）

应该由软件写 1（或者位区域全为 1）。写 0 将产生不可预知的结果。

应为 0（SBZ）

应该由软件写 0（或者位区域全为 0）。写 1 将产生不可预知的结果。

应为 0 或被保护（SBZP）

应该由软件写 0（或者位区域全为 0），或者将相同处理器的某个区域上先前读出的值写回去来保护该值。

串行线调试端口

DAP 的一个可选的外部接口，提供双向的串行线调试接口。

SW-DP

见串行线调试端口。

同步原始指令（synchronization primitive）

存储器同步的原始指令是指用来确保存储器同步的指令。即，LDREX 和 STREX 指令。

系统存储器

0x20000000~0xFFFFFFFF 的存储器空间，包括 PPB 空间（0xE0000000~0xE00FFFFF）。

TAP

见测试访问端口

测试访问端口（TAP）

TAP 是 4 个强制终端和一个可选终端的集合，形成到 JTAG 边界扫描架构的输入/输出

和控制接口。强制的终端为 **TDI**、**TDO**、**TMS**、**TCK**。可选的终端为 **TRST**。RAM 内核中必须具有 TRST 信号，因为要用它来将调试逻辑复位。

Thumb 指令

规定 Thumb 状态中的 ARM 处理器怎样执行一个操作的半字。Thumb 指令必须是半字对齐的。

Thumb 状态

正在执行 thumb（16 位）半字对齐指令的处理器运行时所在的状态即为 Thumb 状态。

TPA

见跟踪端口分析仪。

TPIU

见跟踪端口接口单元。

跟踪端口接口单元（TPIU）

输出跟踪数据并作为片内跟踪数据和 TPA 捕获的数据流之间的桥。

非对齐

存储在不能被定义数据大小的字节数整除的地址中的数据项称作非对齐。例如，存储在不能被 4 整除的地址中的一个字。

UNP

见不可预知。

不可预知

对于读操作，在读这个单元时返回的数据不可预知。它可以有任意的值。对于写操作，写该单元时引起不可预知的行为，或器件配置中的不可预知的改变。不可预知的指令不可以将处理器或系统的任何部分停止或挂起。

热复位

热复位也称作内核复位。它对处理器的大部分元件进行初始化，调试控制器和调试内核除外。如果你正在使用处理器的调试特性，则这种类型的复位将非常有用。

观察点

观察点是由调试器提供的，当特定存储器地址包含的数据被改变时使程序执行停止的一种机制。当对存储器执行写操作来测试程序是否正确运行时，由程序员插入观察点来使能对寄存器内容、存储单元、以及变量值的检查。程序成功测试之后，观察点被移除。也请参考断点。

字

一个 32 位的数据项。

字不变

在字不变系统中，当在小端和大端操作之间进行切换时，存储器每个字节的地址在改变。

这样，在一种字节顺序（endianness）中地址 A 的字节，在另一种顺序（endianness）中其地址为 A 与 3 异或。结果，不管字节顺序（endianness）如何，存储器中每个对齐的字都由相同顺序中的 4 个相同的字节组成。字节顺序（endianness）的改变是由于字节地址的改变而产生的，而不是由于字节的排列顺序改变而产生。

在 ARMv3 及后面的版本中，ARM 架构支持字不变系统。当选择字不变支持时，给出非对齐地址的加载或存储指令的行为是针对特定指令的，通常都不是非对齐访问预期的行为。建议字不变系统一直使用产生所需字节地址的字节顺序（endianness），但在它们建立字节顺序（endianness）之前，处于复位处理器的早期阶段时除外。在复位处理器的早期阶段时必须只使用字对齐的存储器访问。

也请参考字节不变。

写缓冲

写缓冲是一个流水线级，用来对写数据进行缓冲以避免总线延迟而使处理器停止。