ARM7TDMI-S(Rev 4)技术参考手册

第 1 章 介绍

这一章介绍 ARMTDMI-S 处理器。包含以下小节：

- 关于 ARM7TDMI-S 处理器
- ARM7TDMI-S 结构
- ARM7TDMI-S 模块、内核和功能框图
- ARM7TDMI-S 指令集汇总
- Rev 3a 和 Rev 4 之间的差异

1.1 关于 ARM7TDMI-S 处理器

ARM7TDMI-S 处理器是 ARM 通用 32 位微处理器家族的成员之一。ARM 处理器具有优异的性能，但功耗却很低，使用门的数量也很少。ARM 结构是基于精简指令集计算机(RISC)原理而设计的。指令集和相关的译码机制比复杂指令集计算机要简单得多。这样的简化实现了：

- 高的指令吞吐量
- 出色的实时中断响应
- 小的、高性价比的处理器宏单元

1. 1. 1 指令流水线

ARM7TDMI-S 处理器使用流水线来增加处理器指令流的速度。这样可使几个操作同时进行，并使处理和存储器系统连续操作。

流水线使用 3 个阶段，因此指令分 3 个阶段执行：

- 取指
- 译码
- 执行

3 阶段流水线如图 1-1 所示。

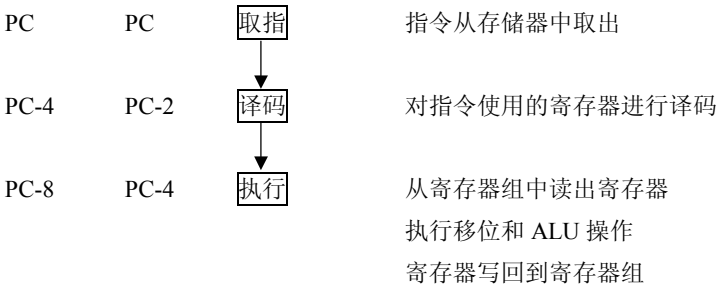

图 1-1 指令流水线

注：程序计数器(PC)指向被取指的指令，而不是指向正在执行的指令。

在正常操作过程中，在执行一条指令的同时对下一条指令进行译码，并将第三条指令从存储器中取出。

1.1.2 存储器访问

ARM7TDMI-S 处理器使用了冯·诺依曼（Von Neumann）结构，指令和数据共用一条 32 位总线。只有装载、存储和交换指令可以对存储器中的数据进行访问。

数据可以是 8 位字节、16 位半字或者 32 位字。字必须分配为占用 4 字节，而半字必须分配为占用 2

字节。

1.1.3 存储器接口

ARM7TDMI-S 处理器的存储器接口可以使潜在的性能得到实现，这样减少了存储器的使用。对速度有严格要求的控制信号使用流水线，这样使系统控制功能以标准的低功耗逻辑实现。这些控制信号使许多片内和片外存储器技术所支持的“快速突发访问模式”得到充分利用。

ARM7TDMI-S 处理器的存储器周期有 4 种基本类型：

- 内部周期
- 非连续的周期
- 连续的周期
- 协处理器寄存器传输周期

1.2 ARM7TDMI-S 的结构

ARM7TDMI-S 处理器有两个指令集：

- 32 位 ARM 指令集
- 16 位 Thumb 指令集

ARM7TDMI-S 处理器使用了 ARM 结构 v4T。关于 ARM 和 Thumb 指令集的详细信息请参阅《ARM 体系结构参考手册》。

1.2.1 指令压缩

传统的微处理器结构对于指令和数据有相同的带宽。因此，和 16 位结构相比，32 位结构处理 32 位数据具有更高的性能，并且在寻址更大的地址空间时要有效得多。

16 位结构比 32 位结构具有更高的代码密度，并且超过 32 位结构 50%的性能。Thumb 在 32 位结构上实现了 16 位的指令集，这样可提供：

- 比 16 位结构更高的性能
- 比 32 位结构更高的代码密度

1.2.2 Thumb 指令集

Thumb 指令集是最通用的 ARM 指令的子集。Thumb 指令长度为 16 位，每条指令都对应一条 32 位 ARM 指令，它对处理器模型有相同的效果。

Thumb 指令使用标准的 ARM 寄存器配置进行操作，这样 ARM 和 Thumb 状态之间具有极好的互用性。在执行方面，Thumb 具有 32 位内核所有的优点：

- 32 位地址空间
- 32 位寄存器
- 32 位移位器和算术逻辑单元(ALU)
- 32 位存储器传输

Thumb 因此提供了长的分支范围、强大的算术操作和巨大的地址空间。

Thumb 代码仅为 ARM 代码规模的 65%，但其性能却相当于连接到 16 位存储器系统的 ARM 处理器性能的 160%。因此 Thumb 使 ARM7TDMI-S 处理器非常适用于那些只有有限的存储器带宽并且代码密度很高的嵌入式应用。

16 位 Thumb 和 32 位 ARM 指令集使设计者极大的灵活性，使他们可以根据各自应用的需求，在子程序一级上实现对性能或者代码规模的优化。例如，应用中的快速中断和 DSP 算法可使用完全的 ARM 指令集编写并使用 Thumb 代码连接。

1.3 ARM7TDMI-S 模块、内核和功能框图

ARM7TDMI-S 处理器结构、内核和功能框图见下：

- ARM7TDMI-S 模块，见图 1-2
- ARM7TDMI-S 内核，见图 1-3
- ARM7TDMI-S 功能框图，见图 1-4

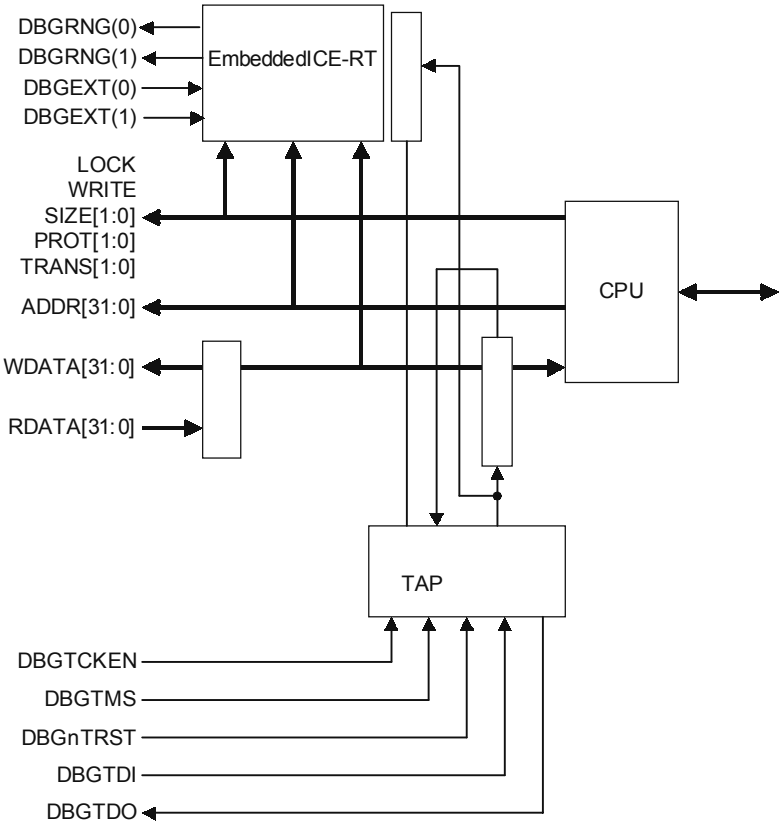

图 1-2 ARM7TDMI-S 模块

注：数据总线上没有双向路径。图 1-2 对这些作了简化。

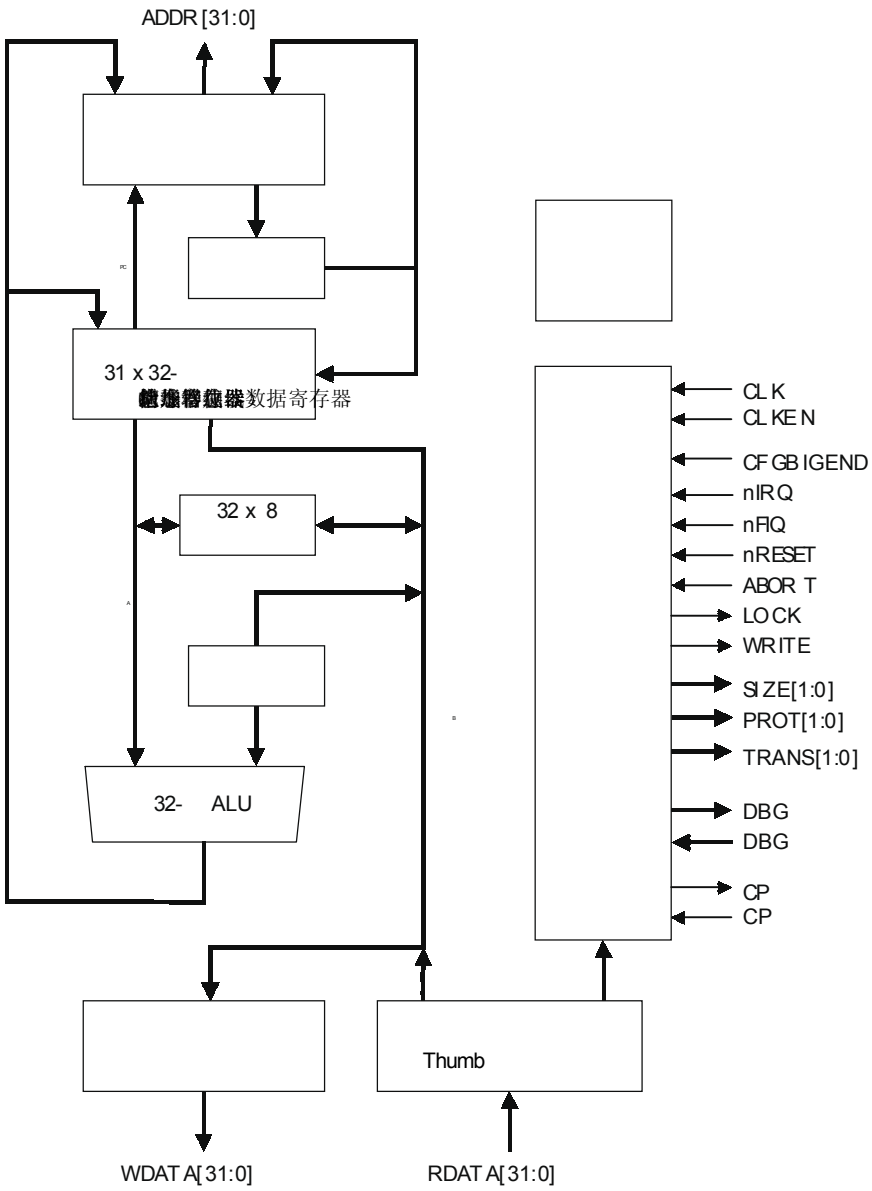

图 1-3 ARM7TDMI-S 内核

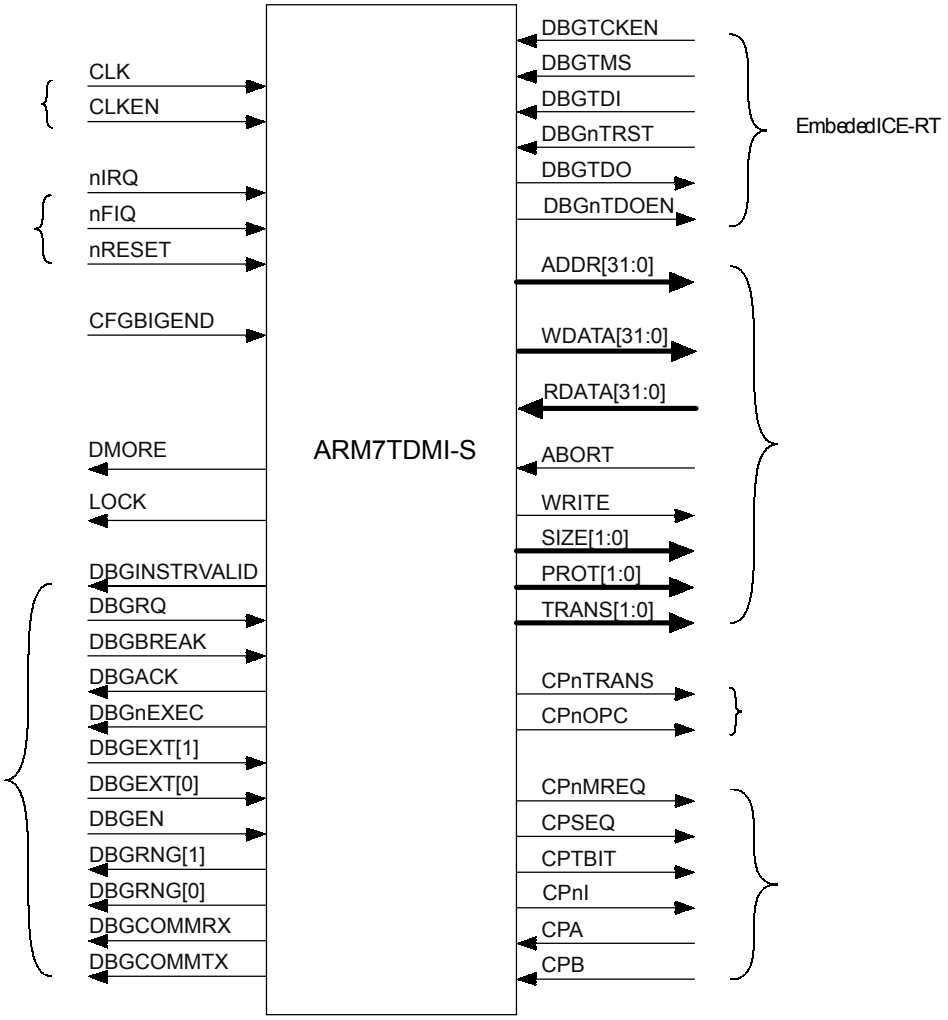

图 1-4 ARM7TDMI-S 功能框图

1.4 ARM7TDMI-S 指令集汇总

该节提供了 ARM 和 Thumb 指令集的汇总：

- ARM 指令汇总
- Thumb 指令汇总

指令集详见 *ARM 体系结构参考手册*。

1.5 Rev 3a 和 Rev 4 之间的差异

ARM7TDMI-S(Rev 4)的变更见下面的章节：

- 增加的 EmbeddedICE-RT 逻辑
- 改进的调试通信通道(DCC)带宽
- 通过 JTAG 对 DCC 进行访问
- TAP 控制器 ID 寄存器
- 更加有效的多路传输

1.5.1 增加的 EmbeddedICE-RT 逻辑

EmbeddedICE-RT 对 ARM7TDMI-S(Rev 3)当中的 EmbeddedICE 逻辑作了改进。EmbeddedICE-RT 可以使您在监控模式下执行调试。在监控模式下，内核在遇到断点或观察点时执行异常处理，并不像在暂停模式中那样进入调试状态。

如果内核在遇到断点或观察点时不进入调试状态，它仍然可以像正常情况下一样响应硬件中断请求。如果内核构成了机械系统反馈环的一部分，那么在监控模式下进行调试非常有用，因为如果停止内核会导致系统运转中断。

更详细的信息请查阅*第 5 章 调试您的系统*。

节电

当 DBGEN 被拉低时，大部分的 EmbeddedICE-RT 逻辑都被禁止以实现最低功耗。

程序员模式的改变

程序员模式的改变如下：

调试控制寄存器

增加了两个新的位：

Bit4 监控模式使能。使用该位来控制器件对断点或观察点的反应。

- 置位时，内核执行指令或数据的异常中止。
- 当清零时，内核进入调试状态。

Bit5 EmbeddedICE-RT 禁止。 在更改观察点和断点时使用该位。

- 置位时，该位禁止断点和观察点，断点或观察点寄存器可以编程为新的值。
- 清零时，新的断点或观察点值生效。

协处理器寄存器映射

协处理器寄存器映射中的一个新的寄存器（R2）指示处理器是因为一个真实的中止，还是因为断点或观察点而进入预取指或数据异常中止。更详细的信息请参阅第 5 章的*中止状态寄存器*。

1.5.2 改进的调试通信通道(DCC)带宽

在 ARM7TDMI-S 处理器(Rev 3)中，读取 DCC 数据需要对扫描链 2 进行两次访问。第一次访问状态位，第二次访问数据本身。

为了改进 DCC 带宽，在 ARM7TDMI-S 处理器(Rev 4)中，读取数据和状态位只需要一次访问。状态位包含在扫描链所读取的地址区域的最低位当中。

DCC 控制寄存器中的状态位保持不变以确保向下兼容性。更多信息参考第 5 章的*调试通信通道*一节。

1.5.3 通过 JTAG 访问 DCC

DCC 控制寄存器可通过 ARM7TDMI-S 处理器(Rev 4)中的 JTAG 接口进行控制。处理器写操作清零 bit0 —数据读控制位。更多信息请参考第 5 章的*调试通信通道*一节。

1.5.4 TAP 控制器 ID 寄存器

TAP 控制器 ID 寄存器值为 0x7F1F0F0F。更多信息请参考第 5 章的 *ARM7TDMI-S 器件标识(ID)代码寄存器*一节。

第 2 章 编程模型

这一章讲述 ARM7TDMI-S 处理器的编程模型。包含以下小节：

- 关于编程模型
- 处理器操作状态
- 存储器格式
- 指令长度
- 数据类型
- 操作模式
- 寄存器
- 程序状态寄存器
- 异常
- 状态延迟
- 复位

2.1 关于编程模型

ARM7TDMI-S 处理器内核使用 ARM v4T 结构实现，该结构包含 32 位 ARM 指令集和 16 位 Thumb 指令集。在 *ARM 体系结构参考手册*中详细讲述了编程模型。

2.2 处理器操作状态

ARM7TDMI-S 处理器有两种操作状态：

ARM 状态 32 位，这种状态下执行的是字方式的 ARM 指令

Thumb 状态 16 位，半字方式的 Thumb 指令

在 Thumb 状态中，程序计数器（PC）使用 bit1 来选择切换半字。

注：ARM 和 Thumb 状态间的切换并不影响处理器模式或寄存器内容。

2.2.1 状态切换

您可以使用 BX 指令将 ARM7TDMI-S 内核的操作状态在 ARM 状态和 Thumb 状态之间进行切换。详见 *ARM 体系结构参考手册*。

所有的异常处理都在 ARM 状态中执行。如果异常发生在 Thumb 状态中，处理器会返回 ARM 状态。在异常处理返回时自动切换回 Thumb 状态。

2.3 存储器格式

ARM7TDMI-S 处理器将存储器看作是一个从 0 开始的线性递增的字节集合：

- 字节 0 到 3 保存第 1 个存储的字
- 字节 4 到 7 保存第 2 个存储的字
- 字节 8 到 11 保存第 3 个存储的字

ARM7TDMI-S 处理器可以将存储器中的字以下列格式存储：

- 大端（Big-endian）格式
- 小端（Little-endian）格式

2.3.1　大端格式

在大端格式中，ARM7TDMI-S 处理器将最高位字节保存在最低地址字节，最低位字节保存在最高地址字节。因此存储器系统字节 0 连接到数据线 31～24。

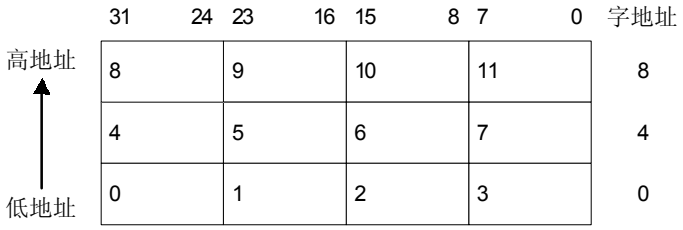

图 2-1　字内字节的大端地址

2.3.2　小端格式

在小端格式中，一个字当中最低地址的字节被看作是最低位字节，最高地址字节被看作是最高位字节。因此存储器系统字节 0 连接到数据线 7～0。如图 2-2 所示。

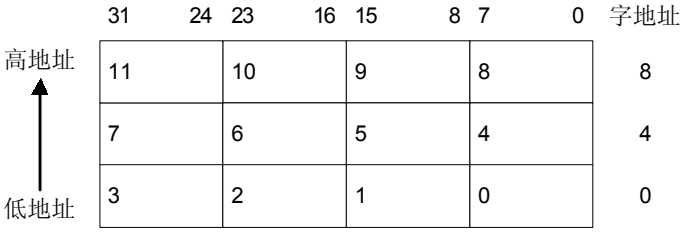

图 2-2　字内字节的小端地址

2.4　指令长度

指令长度为下面两种之一：

- 32 位长度（在 ARM 状态中）
- 16 位长度（在 Thumb 状态中）

2.5　数据类型

ARM7TDMI-S 处理器支持下列数据类型：

- 字（32 位）
- 半字（16 位）
- 字节（8 位）

您必须这样进行分配：

- 字量必须分配为占用 4 个字节
- 半字量必须分配为占用 2 个字节
- 字节量可放置在任何一个字节内

2.6　操作模式

ARM7TDMI-S 处理器具有 7 种操作模式：

- 用户模式　这是 ARM 程序通常执行的状态，用于执行大多数应用程序。
- 快速中断（FIQ）模式　支持数据传输或通道处理。
- 中断（IRQ）模式　用于通用中断处理。

- 超级用户模式 是操作系统一种受保护的模式。
- 中止模式 在数据或指令预取指中止时进入该模式。
- 系统模式 是操作系统一种特许的用户模式。
- 未定义模式 当执行未定义的指令时进入该模式。

除了用户模式之外，其它模式都被归为特权模式。特权模式用于服务中断、异常或者访问受保护的资源。

2.7 寄存器

ARM7TDMI-S 处理器总共有 37 个寄存器：

- 31 个通用 32 位寄存器
- 6 个状态寄存器

这些寄存器并不是在同一时间全都可以被访问的。处理器状态和操作模式决定了程序员可以访问哪些寄存器。

2.7.1 ARM 状态寄存器集

在 ARM 状态中，16 个通用寄存器和 1 个或 2 个状态寄存器可在任何时候同时被访问。在特权模式中，与模式相关的分组寄存器可以被访问。图 2-3 所示为每种模式所能访问的寄存器。

ARM 状态寄存器集包含 16 个可直接访问的寄存器 r0～r15。一个附加的寄存器*当前程序状态寄存器*（CPSR）包含条件代码标志和当前模式位。寄存器 r0～r13 为保存数据或地址值的通用寄存器。寄存器 r14 和 r15 具有下面的特殊功能：

连接寄存器 寄存器 14 作为一个子程序*连接寄存器*（LR）。当执行*连接分支*（BL）指令时，r14 接收 r15 的备份。

在其它时候可将 r14 当成一个通用寄存器。对应的分组寄存器 r14_svc, r14_irq, r14_fiq, r14_abt 和 r14_und 与之相似，当发生中断和异常，或者当中断或异常子程序中的 BL 指令执行时，用于保存 r15 的返回值。

程序计数器 寄存器 15 用于保存*程序计数器*（PC）。

在 ARM 状态中，r15 中的 bits[1:0]为 0。bits[31:2]包含 PC 值。在 Thumb 状态中，bit[0]为 0。bits[31:1]包含 PC 值。

在特权模式中，另外一个寄存器*被保存的程序状态寄存器*（SPSP）可以被访问。它包含了条件代码标志和作为异常的结果所保存的模式位，此异常导致进入当前模式。关于程序状态寄存器的描述见后面的章节。

分组寄存器有一个模式标识符，用于指示它们被映射到哪个用户模式寄存器。这些模式标识符如表 2-1 所示：

表 2-1 寄存器模式标识符

模式	模式标识符
用户	usr
快速中断	fiq
中断	irq
超级用户	svc
中止	abt
系统	sys
未定义	und

FIQ 模式有 7 个分组寄存器，分别映射到 r8～r14（r8_fiq～r14_fiq）。

在 ARM 状态中，大多数 FIQ 处理程序都不必保存任何寄存器。用户、IRQ、超级用户、中止和未定义模式各有 2 个分组寄存器，分别映射到 r13 和 r14，每种模式允许有一个专用的堆栈指针和 LR。

图 2-3 所示为 ARM 状态寄存器。

图 2-3 ARM 状态中的寄存器结构

2.7.2 Thumb 状态寄存器集

Thumb 状态寄存器集时 ARM 状态集的子集。程序员可直接访问：

- 8 个通用寄存器 r0～r7
- PC
- 堆栈指针（SP）
- 连接寄存器（LR）
- CPSR

每个特权模式都有分组的 SP、LR 和 SPSR。该寄存器集如图 2-4 所示。

图 2-4 Thumb 状态中的寄存器结构

2.7.3 ARM 状态寄存器和 Thumb 状态寄存器之间的关系

Thumb 状态寄存器与 ARM 状态寄存器有如下的关系：

- Thumb 状态 r0～r7 与 ARM 状态 r0～r7 相同
- Thumb 状态 CPSR 和 SPSR 与 ARM 状态 CPSR 和 SPSR 相同
- Thumb 状态 SP 映射到 ARM 状态 r13
- Thumb 状态 LR 映射到 ARM 状态 r14
- Thumb 状态 PC 映射到 ARM 状态 PC(r15)

这些关系如图 2-5 所示。

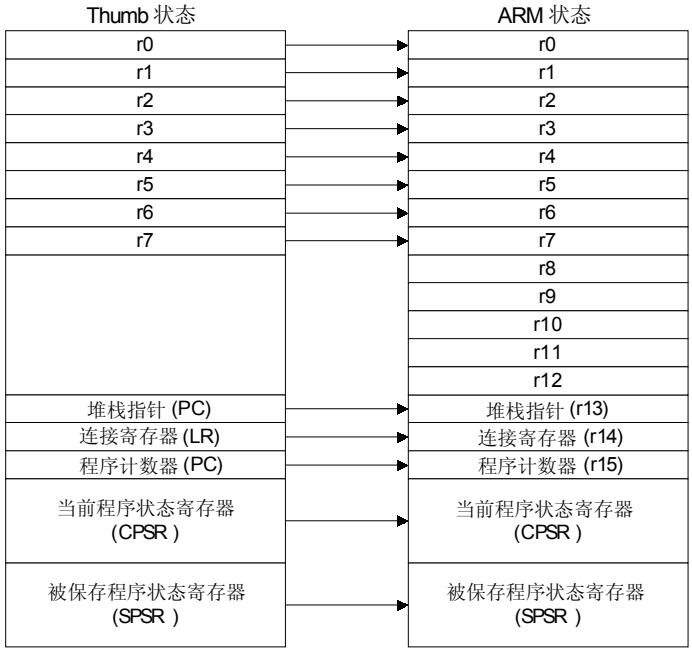

图 2-5 Thumb 寄存器在 ARM 状态寄存器上的映射

注：寄存器 r0～r7 为低寄存器。寄存器 r8～r15 为高寄存器。

2.7.4 在 Thumb 状态中访问高寄存器

在 Thumb 状态中，高寄存器（r8～r15）不是标准寄存器集的一部分。汇编语言程序员对它们的访问受到限制，但可以将它们用于快速暂存。

可以使用 MOV 指令的特殊变量将一个值从低寄存器（r0～r7）转移到高寄存器，或者从高寄存器到低寄存器。CMP 指令可用于比较高寄存器和低寄存器的值。ADD 指令可用于将高寄存器的值与低寄存器的值相加。详细信息请参考 *ARM 体系结构参考手册*。

2.8 程序状态寄存器

ARM7TDMI-S 内核包含 1 个 CPSR 和 5 个 SPSR 供异常处理程序使用。程序状态寄存器：

- 保持条件代码标志
- 控制中断的使能和禁止
- 设置处理器操作模式

位的分配如图 2-6 所示。

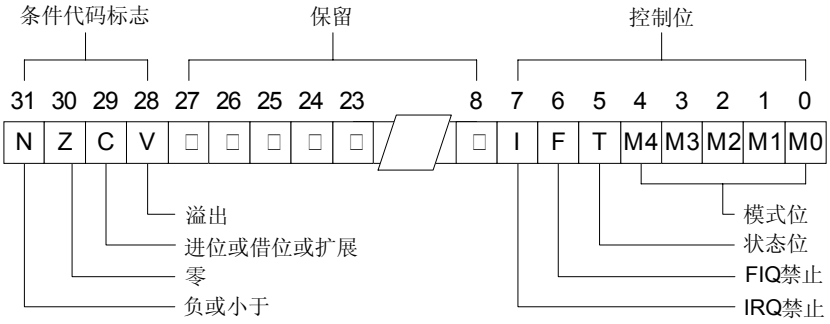

图 2-6 程序状态寄存器格式

注：为了保持与将来的 ARM 处理器兼容，并且作为一种良好的习惯，在更改 CPSR 时，我们强烈建议您使用读－写－修改的方法。

2.8.1 条件代码标志

N, Z, C 和 V 位都是条件代码标志。可以通过算术和逻辑操作来设置这些位。这些标志还可通过 MSR 和 LDM 指令进行设置。ARM7TDMI-S 处理器对这些位进行测试以决定是否执行一条指令。

在 ARM 状态中，所有指令都可按条件来执行。在 Thumb 状态中，只有分支指令可条件执行。更详细的信息请参考 *ARM 体系结构参考手册*。

2.8.2 控制位

PSR 的最低 8 位为控制位。它们分别是：

- 中断禁止位
- T 位
- 模式位

当发生异常时，控制位改变。当处理器在一个特权模式下操作时，可用软件操作这些位。

中断禁止位

I 和 F 位都是中断禁止位：

- 当 I 位置位时，IRQ 中断被禁止
- 当 F 位置位时，FIQ 中断被禁止

T 位

T 位反映了正在操作的状态：

- 当 T 位置位时，处理器正在 Thumb 状态下运行
- 当 T 位清零时，处理器正在 ARM 状态下运行

操作状态通过 **CPTBIT** 外部信号反映。

警告：绝对不要强制改变 CPSR 寄存器中的 T 位。如果这样做，处理器会进入一个无法预知的状态。

模式位

M4, M3, M2, M1 和 M0 位（M[4:0]）都是模式位。这些位决定处理器的操作模式，见表 2-2。不是所有模式位的组合都定义了有效的处理器模式，因此请小心不要使用表中所没有列出的组合。

表 2-2 PSR 模式位值

M[4:0]	模式	可见的 Thumb 状态寄存器	可见的 ARM 状态寄存器
10000	用户	r0~r7, SP, LR, PC, CPSR	r0~r14,PC, CPSR
10001	FIQ	r0~r7,SP_fiq,LR_fiq,PC,CPSR, SPSR_fiq	r0~r7,r8_fiq~r14_fiq,PC,CPSR, SPSR_fiq
10010	IRQ	r0~r7,SP_irq,LR_irq,PC,CPSR, SPSR_fiq	r0~r12,r13_irq,r14_irq,PC,CPSR, SPSR_irq
10011	超级用户	r0~r7,SP_svc,LR_svc,PC,CPSR, SPSR_svc	r0~r12,r13_svc,r14_svc,PC,CPSR, SPSR_svc
10111	中止	r0~r7,SP_abt,LR_abt,PC,CPSR, SPSR_abt	r0~r12,r13_abt,r14_abt,PC,CPSR, SPSR_abt
11011	未定义	r0~r7,SP_und,LR_und,PC,CPSR, SPSR_und	r0~r12,r13_und,r14_und,PC,CPSR, SPSR_und
11111	系统	r0~r7,SP,LR,PC,CPSR	

注：如果将非法值写入 M[4:0]中，处理器将进入一个无法恢复的模式。

2.8.3 保留位

PSR 中的保留位被保留将来使用。当改变 PSR 标志和控制位时，请确认没有改变这些保留位。另外，请确保您的程序不依赖于包含特定值的保留位，因为将来的处理器可能会将这些位设置为 1 或者 0。

2.9 异常

只要正常的程序流被暂时中止，处理器就进入异常模式。例如响应一个来自外设的中断。在处理异常之前，ARM7TDMI-S 内核保存当前的处理器状态，这样当处理程序结束时可以恢复执行原来的程序。

如果同时发生两个或更多异常，那么将按照固定的顺序来处理异常，见*异常优先级*一节。

该节将会详细讲述 ARM7TDMI-S 处理器的异常处理：

- 异常入口/出口汇总
- 进入异常
- 退出异常

2.9.1 异常入口/出口汇总

表 2-3 所示为异常入口处变量 r14 所保存的 PC 值以及退出异常处理程序所推荐使用的指令。

表 2-3 异常入口/出口

异常或入口	返回指令	之前的状态		备注
		ARM r14_x	**Thumb r14_x**	
BL	MOV PC,R14	PC+4	PC+2	此处 PC 为 BL, SWI, 未定义的指令取指或者预取中止指令的地址。
SWI	MOVS PC,R14_svc	PC+4	PC+2	
未定义的指令	MOVS PC,R14_und	PC+4	PC+2	
预取指中止	SUBS PC,R14_abt,#4	PC+4	PC+4	
FIQ	SUBS PC,R14_fiq,#4	PC+4	PC+4	此处 PC 为由于 FIQ 或 IRQ 占先而没有被执行的指令的地址
IRQ	SUBS PC,R14_irq,#4	PC+4	PC+4	
数据中止	SUBS PC,R14_abt,#4	PC+8	PC+8	此处 PC 为产生数据中止的装载或保存指令的地址。
复位	无	—	—	复位时保存在 r14_svc 中的值不可预知。

2.9.2 进入异常

当处理异常时，ARM7TDMI-S 内核会：

1. 在适当的 LR 中保存下一条指令的地址。当异常入口来自：
   - ARM 状态，ARM7TDMI-S 将下一条指令的地址复制到 LR 中（当前 PC+4，或 PC+8，取决于异常的类型）
   - Thumb 状态，ARM7TDMI-S 将 PC 加偏移值（PC+4 或 PC+8，取决于异常的类型）写入 LR

   当进入异常时，异常处理程序不必确定状态。例如在 SWI 情况下，MOVS PC,r14_svc 总是返回到下一条指令，而不管 SWI 是在 ARM 还是在 Thumb 状态下执行。
2. 将 CPSR 复制到适当的 SPSR。
3. 根据异常将 CPSR 模式强制设为某一值。
4. 强制 PC 从相关的异常向量处对下一条指令取指。

ARM7TDMI-S 内核在中断异常时置位中断禁止标志，这样可防止不受控制的异常嵌套。

注：异常总是在 ARM 状态中进行处理。当处理器处于 Thumb 状态时发生了异常，在异常向量地址装入 PC 时，会自动切换到 ARM 状态。

2.9.3 退出异常

当异常结束时，异常处理程序必须：

1. 将 LR 中的值减去偏移量后移入 PC。偏移量根据异常的类型而有所不同，见表 2-3。
2. 将 SPSR 的值复制回 CPSR。
3. 清零在入口置位的中断禁止标志。

注：恢复 CPSR 的动作会将 T, F 和 I 位自动恢复为异常发生前的值。

2.9.4 快速中断请求

快速中断请求(FIQ)异常支持数据转移或通道处理。在 ARM 状态中，FIQ 模式有 8 个专用的寄存器可用来满足寄存器保护的需要（这是上下文切换的最小开销）。

将 nFIQ 信号拉低可实现外部产生 FIQ。

不管异常入口是来自 ARM 状态还是 Thumb 状态，FIQ 处理程序都会通过执行下面的指令从中断返回：

SUBS PC,R14_fiq,#4

在一个特权模式中，可通过置位 CPSR 中的 F 标志来禁止 FIQ 异常。当 F 标志清零时，ARM7TDMI-S 在每条指令结束时检测 FIQ 同步器输出端的低电平。

2.9.5 中断请求

中断请求（IRQ）异常是一个由 nIRQ 输入端的低电平所产生的正常中断。IRQ 的优先级低于 FIQ。对于 FIQ 序列它是被屏蔽的。任何时候在一个特权模式下，都可通过置位 CPSR 中的 I 位来禁止 IRQ,。

不管异常入口是来自 ARM 状态还是 Thumb 状态，IRQ 处理程序都会通过执行下面的指令从中断返回：

SUBS PC,R14_irq,#4

2.9.6 中止

中止表示当前存储器访问不能被完成。这是通过外部 ABORT 输入指示的。不管异常入口是来自 ARM 状态还是 Thumb 状态，FIQ 处理程序都会通过执行下面的指令从中断返回：

SUBS PC,R14_fiq,#4 在存储器访问周期结束时检测中止异常。

有两种类型的中止：

- 预取指中止 发生在指令预取指过程中
- 数据中止 发生在对数据访问时。

预取指中止

当发生预取指中止时，ARM7TDMI-S 内核将预取的指令标记为无效，但在指令到达流水线的执行阶段时才进入异常。如果指令在流水线中因为发生分支而没有被执行，中止将不会发生。

在处理中止的原因之后，不管处于哪种处理器操作状态，处理程序都会执行下面的指令：

SUBS PC,R14_abt,#4

这个动作恢复了 PC 和 CPSR 并重试被中止的指令。

数据中止

当发生数据中止时，根据指令的类型产生不同的动作：

- 数据转移指令（LDR,STR）回写到被修改的基址寄存器。中止处理程序必须注意这一点。
- 交还指令（SWP）中止好像没有被执行过一样（中止必须发生在 SWP 指令进行读访问时）
- 块数据转移指令（LDM,STM）完成。 当回写被设置时，基址寄存器被更新。在指示出现中止后，ARM7TDMI-S 内核防止所有寄存器被覆盖。这意味着 ARM7TDMI-S 内核总是会保护被中止的 LDM 指令中的 r15（总是最后一个被转移的寄存器）。

中止的机制使指令分页的虚拟存储器系统能够被实现。在这样一个系统中，处理器允许产生仲裁地址。当某一地址的数据无法访问时，*存储器管理单元*（MMU）通知产生了中止。中止处理程序必须找出中止的原因，使请求的数据可以被访问并重新执行被中止的指令。应用程序不必知道可用存储器的数量，也不必知道它的被中止时所处的状态。

在修复产生中止的原因后，不管处于哪种处理器操作状态，处理程序都必须执行下面的返回指令：

SUBS PC,R14_abt,#8

这个动作恢复了 PC 和 CPSR 并重试被中止的指令。

2.9.7 软件中断指令

软件中断(SWI)用于进入超级用户模式，通常用于请求一个特定的超级用户函数。SWI 处理程序通过执行下面的指令返回：

MOVS PC,R14_svc

这个动作恢复了 PC 和 CPSR 并返回到 SWI 之后的指令。SWI 处理程序读取操作码以提取 SWI 函数编号。

2.9.8 未定义的指令

当 ARM7TDMI-S 处理器遇到一条系统内任何协处理器都无法处理的指令时，ARM7TDMI-S 内核执行

未定义指令陷阱。软件可使用这一机制通过仿真未定义的协处理器指令来扩展 ARM 指令集。

注：ARM7TDMI-S 处理器完全遵循 ARM 结构 v4T，可以捕获所有分类未被定义的指令位格式。

在防止失败的指令后，捕获处理器执行下面的指令：

MOVS PC,R14_und

这个动作恢复了 PC 和 CPSR 并返回到未定义指令之后的指令。

关于未定义指令更详细的信息请参考 *ARM 体系结构参考手册*。

2.9.9 异常向量

表 2-4 所示位异常向量地址。在表中，I 和 F 表示先前的值。

表 2-4 异常向量

地址	异常	进入时的模式	进入时 I 的状态	进入时 F 的状态
0x00000000	复位	超级用户	禁止	禁止
0x00000004	未定义指令	未定义	I	F
0x00000008	软件中断	超级用户	禁止	F
0x0000000C	中止（预取指）	中止	I	F
0x00000010	中止（数据）	中止	I	F
0x00000014	保留	保留	—	—
0x00000018	IRQ	IRQ	禁止	F
0x0000001C	FIQ	FIQ	禁止	禁止

2.9.10 异常优先级

当多个异常同时发生时，一个固定的优先级系统决定它们被处理的顺序：

1. 复位（最高优先级）
2. 数据中止
3. FIQ
4. IRQ
5. 预取指中止
6. 未定义指令
7. SWI（最低优先级）

有些异常不能一起发生：

- 未定义的指令和 SWI 异常互斥。它们分别对应于当前指令的一个特定（非重叠）译码。
- 当 FIQ 使能，并且在发生 FIQ 的同时产生了一个数据中止，ARM7TDMI-S 内核进入数据中止处理程序，然后立即转到 FIQ 向量。从 FIQ 的正常返回使数据中止处理程序恢复执行。数据中止的优先级必须高于 FIQ 以确保数据转移错误不会被漏过。必须将异常入口的时间增加到系统中最坏情况下 FIQ 的延迟时间。

2.10 中断延迟

中断延迟被描述为：

- 最大中断延迟
- 最小中断延迟

2.10.1 最大中断延迟

当 FIQ 使能时，最坏情况下 FIQ 的延迟时间包含：

- Tsyncmax，请求通过同步器的最长时间。Tsyncmax 为 2 个处理器周期。
- Tldm，最长的指令执行需要的时间（最长的指令是装载包括 PC 在内所有寄存器的 LDM 指令）。Tldm 在零等待状态系统中的执行时间为 20 个周期。
- Texc，数据中止入口的时间。Texc 为 3 个周期。
- Tfiq，FIQ 入口的时间。Tfiq 为 2 个周期。

因此总的延迟时间为 27 个周期，在系统使用 40MHz 处理器时钟时，略微小于 0.7 微妙。在此时间结束后，ARM7TDMI-S 执行位于 0x1c 处的指令。

最大的 IRQ 延迟时间与之相似，但必须考虑到这样一个事实，即有更高优先级的 FIQ 可能会因为仲裁的时间而延迟 IRQ 处理程序的进入。

2.10.2 最小中断延迟

FIQ 或 IRQ 的最小中断延迟是请求通过同步器的时间 Tsyncmin 加上 Tfiq（4 个处理器周期）。

2.11 复位

当 nRESET 信号被拉低时，ARM7TDMI-S 处理器放弃正在执行的指令。

当 nRESET 信号再次变为高电平时，nRESET 处理器：

1. 强制 M[4:0]变为 b10011（超级用户模式）
2. 置位 CPSR 中的 I 和 F 位
3. 清零 CPSR 中的 T 位
4. 强制 PC 从地址 0x00 开始对下一条指令进行取指。
5. 返回到 ARM 状态并恢复执行

在复位后，除 PC 和 CPSR 之外的所有寄存器的值都不确定。

第 3 章 存储器接口

这一章讲述 ARM7TDMI-S 处理器的存储器接口。包含以下内容：

- 关于存储器接口
- 总线接口信号
- 总线周期类型
- 寻址信号
- 数据定时信号
- 使用 CLKEN 控制总线周期

3.1 关于存储器接口

ARM7TDMI-S 处理器采用冯·诺曼（Von Neumann）结构，指令和数据共用一条 32 位数据总线。只有装载、保存和交换指令可访问存储器中的数据。

ARM7TDMI-S 处理器支持 4 种基本类型的存储器周期：

- 非连续
- 连续
- 内部
- 协处理器寄存器传递

3.2 总线接口信号

ARM7TDMI-S 处理器总线接口的信号可分成 4 类：

- 时钟和时钟控制
- 地址分类信号
- 存储器请求信号
- 数据定时信号

时钟和时钟控制信号为：

- CLK
- CLKEN
- nRESET

地址分类信号：

- ADDR[31:0]
- WRITE
- SIZE[1:0]
- PROT[1:0]
- LOCK

存储器请求信号为：

- TRANS[1:0]

数据定时信号为：

- WDATA[31:0]
- RDATA[31:0]
- ABORT

每个信号组都共用相同的总线接口周期时序关系。ARM7TDMI-S 处理器总线接口中的所有信号都产生或采样自 CLK 的上升沿。

总线周期可通过使用 CLKEN 信号进行延长。该信号的介绍见*使用 CLKEN 控制总线周期*一节。这一章的所有其它节所描述的都是一个 CLKEN 永远为高电平的简单系统。

3.3 总线周期类型

ARM7TDMI-S 处理器总线接口采用流水线结构，因此地址分类信号和存储器请求信号都在前一个总线周期内广播。这样可使一个存储器周期用最大时间对地址进行译码并响应访问请求。

单个的存储器周期如图 3-1 所示。

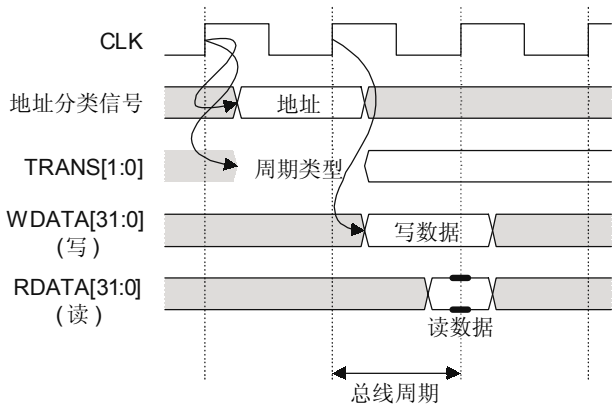

图 3-1 单个存储器周期

ARM7TDMI-S 处理器总线接口可执行 4 种不同类型的存储器周期。这些类型通过 TRANS[1:0]信号指示。存储器周期类型在 TRANS[1:0]信号上的编码见表 3-1。

表 3-1 周期类型

TRANS[1:0]	**周期类型**	**描述**
00	I 周期	内部周期
01	C 周期	协处理器寄存器传输周期
10	N 周期	非连续周期
11	S 周期	连续周期

ARM7TDMI-S 处理器的存储器控制器只在 N 周期或 S 周期负责存储器的访问。ARM7TDMI-S 处理器有 4 种基本类型的处理器周期：

非连续周期

在此周期中，ARM7TDMI-S 内核请求与一个地址进行数据传送，该地址与前一个周期所使用的地址无关。

连续周期

在此周期中，ARM7TDMI-S 内核请求与一个地址进行数据传送，该地址比前一个周期所使用的地址大一个字或半个字。

内部周期

在此周期中，ARM7TDMI-S 内核不请求传送，因为它正在执行一个内部功能，这时没有有用的预取指能够被执行。

协处理器传输周期

在此周期中，ARM7TDMI-S 内核使用数据总线与协处理器进行通信，但不请求存储器的任何动作。

3.3.1 非连续周期

非连续周期是 ARM7TDMI-S 处理器总线周期的最简格式，当 ARM7TDMI-S 内核请求与一个地址进行数据传送，而该地址与前一个周期所使用的地址无关时，就产生了非连续周期。存储器控制器必须启动一个存储器访问以满足该请求。

地址分类信号和 TRANS[1:0]=N 周期都在总线上广播。在下一个总线周期结束时，数据在 CPU 和存储器之间进行传输。如图 3-2 所示。

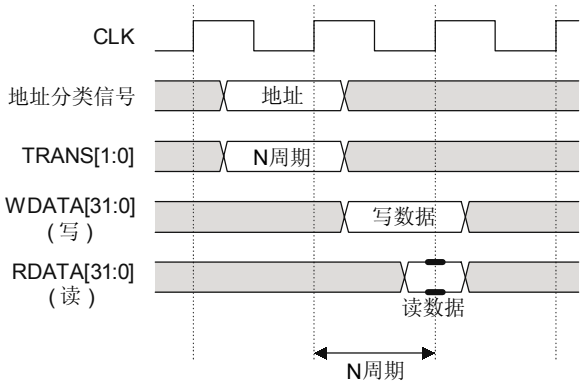

图 3-2 非连续存储器周期

ARM7TDMI-S 处理器可执行背靠背的非连续存储器周期。例如，执行一个 STR 指令，见图 3-3。如果您正在设计一个 ARM7TDMI-S 处理器的存储器控制器，而您的存储器系统无法应付这种情况，那么您必须使用 CLKEN 信号延长总线周期，以使存储器系统有足够多的周期可用。

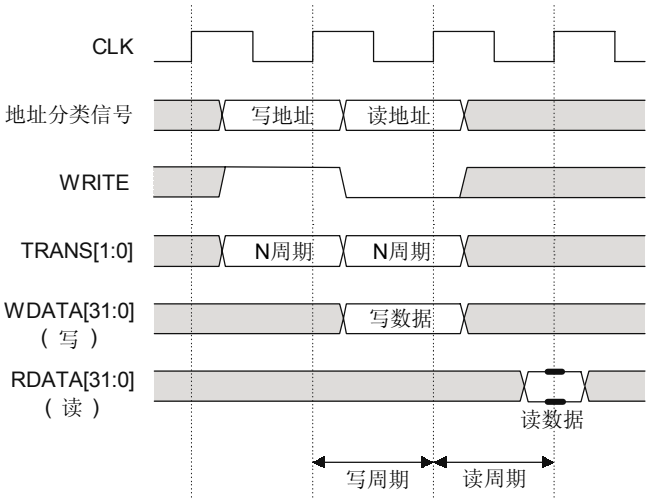

图 3-3 背靠背的存储器周期

3.3.2 连续周期

连续周期执行总线上的突发串传输。可以使用此信息优化存储器控制器与突发串存储器件间的接口设计，例如，DRAM。

在一个连续周期中，ARM7TDMI-S 处理器请求一个存储器位置，该位置是连续突发串的一部分。如果是突发串的第一部分，那么地址与前一个内部周期相同。否则，地址根据前一个周期增加：

- 对于一个字突发串的访问，地址加 4
- 对于一个半字突发串的访问，地址加 2

字节突发串的访问无法实现。

一个突发串总是以一个 N 周期或一个合并的 I-S 周期开始，接下来是 S 周期。一个突发串由相同类型的传输组成。ADDR[31:0]信号在突发串中增加。其它地址分类信号在整个突发串中保持不变。

突发串的类型见表 3-2。

表 3-2 突发串类型

突发串类型	地址增量	原因
字读	4 字节	ARM7TDMI-S 代码取指或 LDM 指令
字写	4 字节	STM 指令
半字读	2 字节	Thumb 代码取指

突发串中的所有访问都具有相同的宽度、方向和保护类型。

图 3-4 所示为突发串访问的一个例子。

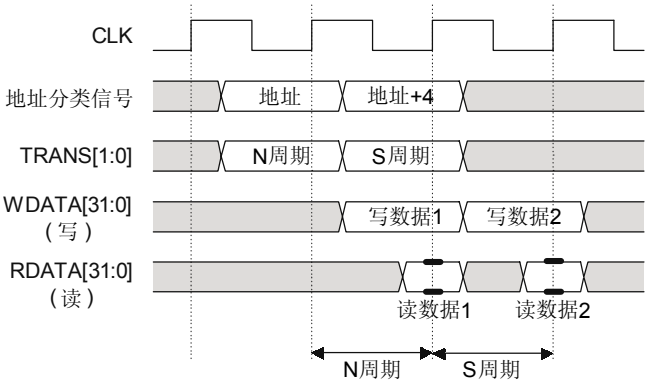

图 3-4 连续访问周期

3.3.3 内部周期

在一个内部周期中，ARM7TDMI-S 处理器不请求对存储器进行访问，由于正在执行一个内部功能，这时没有有用的预取指能够被执行。

ARM7TDMI-S 处理器有可能广播下一次访问的地址，这样使译码能够开始执行，但存储器控制器不能提交存储器访问。见下一节的描述。

3.3.4 合并的 I-S 周期

ARM7TDMI-S 处理器可以执行对总线的优化，这样可为存储器译码增加额外的时间。这样做的时候，下一个存储器周期的地址在总线的内部周期中广播。这使存储器控制器对地址进行译码，但在这个周期决不会启动存储器的访问。在合并的 I-S 周期中，下一个周期是同一个存储器位置的连续周期。这使访问被提交，而存储器控制器必须启动对存储器的访问。见图 3-5。

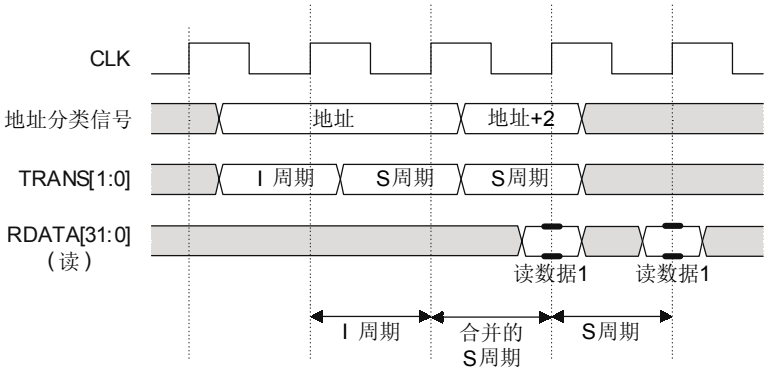

图 3-5 合并的 I-S 周期

注：当设计一个存储器控制器时，请确保在指向一个不同地址的 N 周期跟在 I 周期之后时，设计也能工作。在发生异常或写 PC 时可能会出现这种时序。很重要的一点就是，存储器控制器在 I 周期内不提交存储器周期。

3.3.5 协处理器寄存器传输周期

在一个协处理器寄存器传输周期中，ARM7TDMI-S 处理器使用数据总线与协处理器进行数据传输。这时不需要存储器周期，存储器控制器不会启动处理。

协处理器接口在第 4 章有详细描述。

3.4 寻址信号

地址分类信号分为：

- ADDR[31:0]
- WRITE
- SIZE[1:0]
- PROT[1:0]
- LOCK
- CPTBIT

3.4.1 ADDR[31:0]

ADDR[31:0]是 32 位地址总线，它指定传输的地址。所有地址都为字节地址，因此一个字突发串访问使地址总线每个周期加 4。

地址总线提供 4GB 的线性地址空间。当被通知进行字访问时，存储器系统必须忽略最低两位 **ADDR[1:0]**，当被通知进行半字访问时，存储器系统必须忽略最低位 **ADDR[0]**。

3.4.2 WRITE

WRITE 指定传输的方向。WRITE 为高电平时指示写周期，为低电平时指示读周期。S 周期的突发串总是为读突发串或写突发串。在突发串的中间不能改变方向。

3.4.3 SIZE[1:0]

SIZE[1:0]对传输规格进行编码。ARM7TDMI-S 处理器可传输字、半字和字节量。见表 3-3 所示的 **SIZE[1:0]**编码。

表 3-3 传输宽度

SIZE[1:0]	**传输宽度**
00	字节
01	半字
10	字
11	保留

传输规格在 S 周期的突发串中不能改变。

注：为 ARM7TDMI-S 处理器提供的可写存储器系统必须具有单个的字节写使能。C 编译器和 ARM 调试工具链（例如 Multi-ICE）都假设存储器中的仲裁字节可写。如果不提供单个的字节写功能，可能无法使用上述两种工具。

3.4.4 PORT[1:0]

PORT[1:0]对关于传输的信息进行编码。存储器管理单元使用该信号确定访问是否来自一个特权模式，是操作码还是数据取指。因此可用它来实现访问允许机制。**PORT[1:0]**的编码如表 3-4 所示。

表 3-4 PORT[1:0]编码

PORT[1:0]	模式	操作码或数据
00	用户	操作码
01	用户	数据
10	特权	操作码
11	特权	数据

3.4.5 LOCK

LOCK 向仲裁器指出在总线上正在执行一个微小的操作。**LOCK** 通常为低电平，但在执行 **SWP** 或 **SWPB** 指令时置为高电平。这些指令执行一个微小的读/写操作并且可以用于实现信号量。

3.4.6 CPTBIT

CPTBIT 指示 ARM7TDMI-S 处理器的操作状态：

- 在 ARM 状态中，**CPTBIT** 信号为低电平
- 在 Thumb 状态中，**CPTBIT** 信号为高电平

3.5 数据定时信号

数据定时信号在下面的小节中描述：

- **WDATA[31:0]**
- **RDATA[31:0]**
- **ABORT**

3.5.1 WDATA[31:0]

WDATA[31:0]是写数据总线。所有 **WDATA[31:0]**处理器的写数据都在该总线上广播。**WDATA[31:0]** 内核到协处理器的数据传输在 C 周期中也使用该总线。在正常环境下，存储器系统必须在写总线周期结束时的 **CLK** 上升沿对 **WDATA[31:0]**总线进行采样。**WDATA[31:0]**的值只在写周期内有效。

3.5.2 RDATA[31:0]

RDATA[31:0]是读数据总线，它供 ARM7TDMI-S 内核读取操作码和数据之用。**RDATA[31:0]**信号在总线周期结束时的 **CLK** 上升沿被采样。在 C 周期中，该总线也用于从协处理器向 ARM7TDMI-S 内核传输数据。

3.5.3 ABORT

ABORT 指示存储器处理器失败。**ABORT** 在有效的存储器周期（S 周期和 N 周期）内的总线周期结束时被采样。

如果 **ABORT** 在数据访问时被声明，它将使 ARM7TDMI-S 处理器执行数据中止陷阱。如果在操作码取指时声明，中止将沿着流水线被跟踪。如果指令已被执行，则执行预取指中止陷阱。

ABORT 可被存储器管理系统用来实现一个基本的存储器保护机制或一个指令分页的虚拟存储器系统。

3.5.4 字节和半字访问

ARM7TDMI-S 处理器通过 SIZE[1:0]信号指示传输数据的规格。其编码见表 3-5：

表 3-5 传输规格编码

SIZE[1:0]	传输宽度
00	字节
01	半字
10	字
11	保留

为了能够使用 C 编译器和 ARM 调试工具链（例如 Multi-ICE），基于 ARM7TDMI-S 处理器系统中所有的可写存储器都支持对单个字节的写操作。

ARM7TDMI-S 处理器所产生的地址总是字节地址。但是存储器系统会忽略地址中无意义的位。有用的位如表 3-6 所示：

SIZE[1:0]	宽度	有用的地址位
00	字节	**ADDR[31:0]**
01	半字	**ADDR[31:1]**
10	字	**ADDR[31:2]**

当执行字节或半字读操作时，32 位存储器系统可返回完整的 32 位字，ARM7TDMI-S 处理器从中提取有效的半字或字节部分。这部分根据 **CFGBIGEND** 信号的状态进行提取，**CFGBIGEND** 信号决定系统的使用大端还是小端格式。

被 ARM7TDMI-S 处理器提取出来的部分见表 3-7。

表 3-7 字访问

SIZE[1:0]	ADDR[1:0]	小端 CFGBIGEND=0	大端 CFGBIGEND=1
10	XX	**RDATA[31:0]**	**RDATA[31:0]**

将 8 位到 16 位存储器系统连接到 ARM7TDMI-S 处理器时，请确保数据出现在 ARM7TDMI-S 处理器正确的字节位置上。见表 3-8 和 3-9。

表 3-8 半字访问

SIZE[1:0]	ADDR[1:0]	小端 CFGBIGEND=0	大端 CFGBIGEND=1
01	0X	**RDATA[15:0]**	**RDATA[31:16]**
01	1X	**RDATA[31:16]**	**RDATA[15:0]**

表 3-9 字节访问

SIZE[1:0]	ADDR[1:0]	小端 CFGBIGEND=0	大端 CFGBIGEND=1
00	00	**RDATA[7:0]**	**RDATA[31:24]**
00	01	**RDATA[15:8]**	**RDATA[23:16]**
00	10	**RDATA[23:16]**	**RDATA[15:8]**
00	11	**RDATA[31:24]**	**RDATA[7:0]**

写

当 ARM7TDMI-S 处理器执行字节或半字写操作时，被写的数据在总线上被复制，见图 3-6。存储器系统可使用最便捷的数据复制。一个可写的存储器系统必须能够执行对存储器中任何单个字节的写操作。这一功能是 ARM 的 C 编译器和调试工具链所需要的。

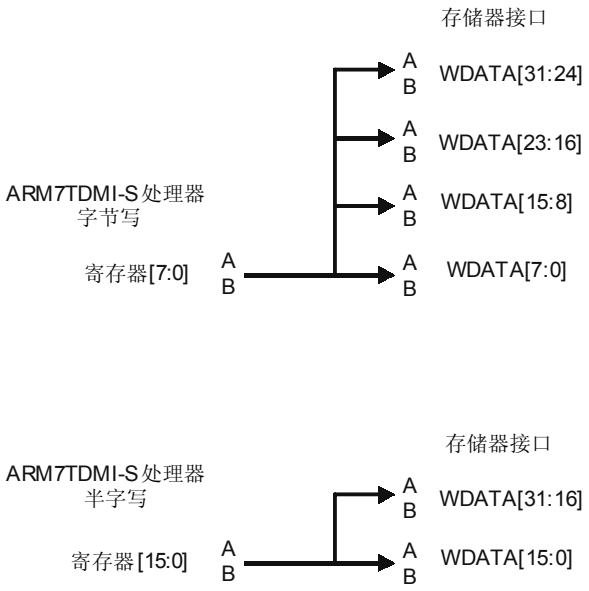

图 3-6 数据复制

3.6 使用 CLKEN 控制总线周期

ARM7TDMI-S 处理器总线接口的流水线特性意味着时钟周期和总线周期存在着区别。CLKEN 可用于延长总线周期，使其可以持续许多个时钟周期。CLKEN 输入通过增加完整的 CLK 周期使总线周期时序得到扩展：

- 当 CLKEN 在 CLK 上升沿时为高电平，总线周期结束。
- 当 CLKEN 被采样为低电平时，总线周期扩展。

在流水线中，地址分类信号和存储器请求信号超前数据传输一个总线周期。在一个使用 CLKEN 的系统中，这个总线周期可以大于一个 CLK 周期。图 3-7 所示为使用 CLKEN 扩展一个非连续周期。在该例中，第一个 N 周期之后的 N 周期指向一个无关的地址，第二次访问的地址在第一次访问结束之前广播。

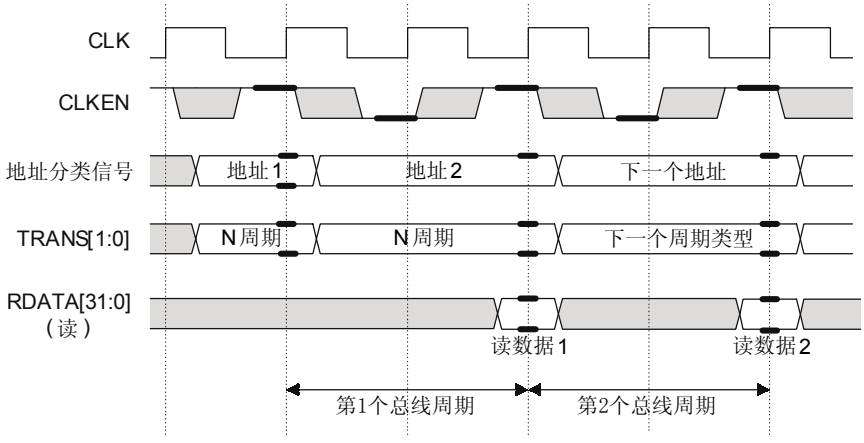

图 3-7 CLKEN 的使用

注：当设计一个存储器控制器时，强烈建议您只在 CLKEN 为高电平时对 TRANS[1:0]和地址分类信号的值进行采样。这样可以保证存储器控制器的状态不会在一个总线周期内意外更新。

第 4 章 协处理器接口

这一章讲述 ARM7TDMI-S 协处理器接口。包含以下部分：

- 关于协处理器
- 协处理器接口信号
- 协处理器接口握手
- 连接协处理器
- 不使用外部协处理器
- 未定义的指令
- 特权指令

4.1 关于协处理器

ARM7TDMI-S 处理器指令集使您可以通过协处理器来实现特殊的附加指令。这些协处理器是与 ARM7TDMI-S 内核相结合的单独的处理单元。一个典型的协处理器包括：

- 指令流水线
- 指令译码逻辑
- 寄存器分组
- 带独立数据通路的特殊处理逻辑

协处理器和 ARM7TDMI-S 处理器连接到同一个数据总线，这意味着协处理器可以对指令流中的指令进行译码并执行那些它所支持的指令。每条指令的处理都沿着 ARM7TDMI-S 处理器流水线和协处理器流水线同时进行。

指令的执行由 ARM7TDMI-S 内核与协处理器共同实现。

ARM7TDMI-S 内核：

1. 求出条件代码的值以确定指令是否必须由协处理器执行，然后使用 CPnI 通知系统中的所有协处理器。
2. 产生指令所要求的地址（包括下一条指令的预取指）来填充流水线。
3. 如果出现协处理器不接受的指令，则执行未定义指令陷阱。

协处理器：

1. 对指令进行译码以确定是否接受。
2. 通过 CPA 和 CPB 指示它是否接受这一指令。
3. 从自身的寄存器组当中取出任何需要的值。
4. 执行指令所要求的操作。

如果协处理器无法执行某条指令，则执行未定义指令陷阱。您可以选择在软件中仿真协处理器功能或设计一个专用的协处理器。

4.1.1 可用的协处理器

一个系统中最多可连接 16 个协处理器，每个协处理器都通过唯一的 ID 号识别。ARM7TDMI-S 处理器包含两个内部协处理器：

- CP14 通信通道协处理器
- CP15 为 cache 和 MMU 功能提供的系统控制协处理器

因此，您不能将外部协处理器的编号分配为 14 和 15。ARM 还保留了其它的协处理器编号，见表 4-1。

表 4-1 可用的协处理器

协处理器编号	分配
15	系统控制
14	调试控制器
13:8	保留
7:4	可供用户使用
3:0	保留

如果您打算设计一个协处理器，请将标题为 coprocessor 的邮件发送到 info@arm.com 获取已经分配的协处理器编号的最新信息。

4.2 协处理器接口信号

此信号用于连接 ARM7TDMI-S 内核与协处理器，分为 4 类：

时钟和时钟控制信号：

- CLK
- CLKEN
- nRESET

流水线跟随信号：

- CPnI
- CPA
- CPB

握手信号：

- WDATA[31:0]
- RDATA[31:0]

这些信号和它们的用途见下面的章节：

- 流水线跟随信号
- 协处理器接口的握手
- 连接协处理器
- 不使用外部协处理器
- 未定义的指令
- 特权指令

4.3 流水线跟随信号

每个系统中的协处理器都必须包含一个流水线跟随器，用它对 ARM7TDMI-S 内核流水线中执行的指令进行跟踪。协处理器连接到 ARM7TDMI-S 处理器的输入数据总线 RDATA[31:0]（指令通过其被取指）和 CLK 以及 CLKEN。

在任何时候两条流水线都必须保持步调一致。为协处理器设计一个流水线跟随器时，必须遵循以下原则：

- 在复位时（nRESET），流水线必须被标记为无效，或者填充不会被协处理器译码成有效指令的指令。
- 协处理器状态必须只在 CLKEN 为高电平时改变（复位除外）
- 指令必须在 CLK 的上升沿，并且前一个总线周期的 CPnOPC, CPnMREQ 和 CPTBIT 全都为低电平，装入流水线。这些条件指示了该周期时 ARM 状态操作码取指，因此必须将新的操作码采样到流水线。

- 流水线必须在 CLK 的上升沿并且当前总线周期内的 CPnOPC, CPnMREQ 和 CPTBIT 全都为低电平时超前。这些条件指示了当前指令将要执行完毕，因为任何指令的第一个动作是执行指令预取指来重新填充流水线。

ARM7TDMI-S 处理器流水线上的任何指令在它们进入执行时都不会在 CPnI 上输出信号，因此它们根据所要求的预取指自动填充流水线。

在 Thumb 指令集当中没有协处理器指令，因此协处理器必须监视 CPTBIT 信号的状态以确保它们不会将 Thumb 指令对像 ARM 指令一样译码。

4.4 协处理器接口的握手

ARM7TDMI-S 内核与系统中的任何协处理器通过信号执行握手，见表 4-2。

表 4-2 握手信号

信号	方向	含义
CPnI	ARM7TDMI-S 到协处理器	非协处理器指令
CPA	协处理器到 ARM7TDMI-S	协处理器缺席
CPB	协处理器到 ARM7TDMI-S	协处理器忙

这些信号在后面的章节会详细讲述。

4.4.1 协处理器

协处理器对其流水线译码阶段中的当前指令进行译码，并检测指令是否为协处理器指令。协处理器指令具有一个与协处理器 ID 相匹配的编号。

如果译码阶段当前的指令时一个协处理器指令：

1. 协处理器试图执行该指令。
2. 协处理器使用 CPA 和 CPB 向 ARM7TDMI-S 内核发出信号。

4.4.2 ARM7TDMI-S 内核

协处理器指令沿着 ARM7TDMI-S 处理器流水线运行，与协处理器流水线保持步调一致。协处理器指令在下面的条件为真时执行：

1. 协处理器指令到达流水线的执行阶段（如果被一个分支抢先，则有可能不是）。
2. 指令已经通过了条件执行测试。
3. 系统中的协处理器已经 CPA 和 CPB 上发出信号表明它能够接受该指令。

如果满足所有这些条件，ARM7TDMI-S 内核通过将 CPnI 拉低来通知协处理器可以执行该指令。

4.4.3 协处理器发信号

协处理器所发信号如下：

协处理器缺席

如果协处理器不能接受当前译码阶段的指令，它必须使 CPA 和 CPB 保持高电平。

协处理器出席

如果协处理器可以接受一个指令并且可以立即启动该指令，它必须通过拉低 CPA 和 CPB 来通知 ARM7TDMI-S 内核。

协处理器忙（忙─等待）

如果协处理器可以接受一条指令，但它不能够立即处理该请求，它可以通过声明“忙-等待”使 ARM7TDMI-S 内核暂停。这通过拉低 CPA，保持 CPB 高电平来声明。当协处理器就绪时将 CPB 拉低。如图 4-1 所示。

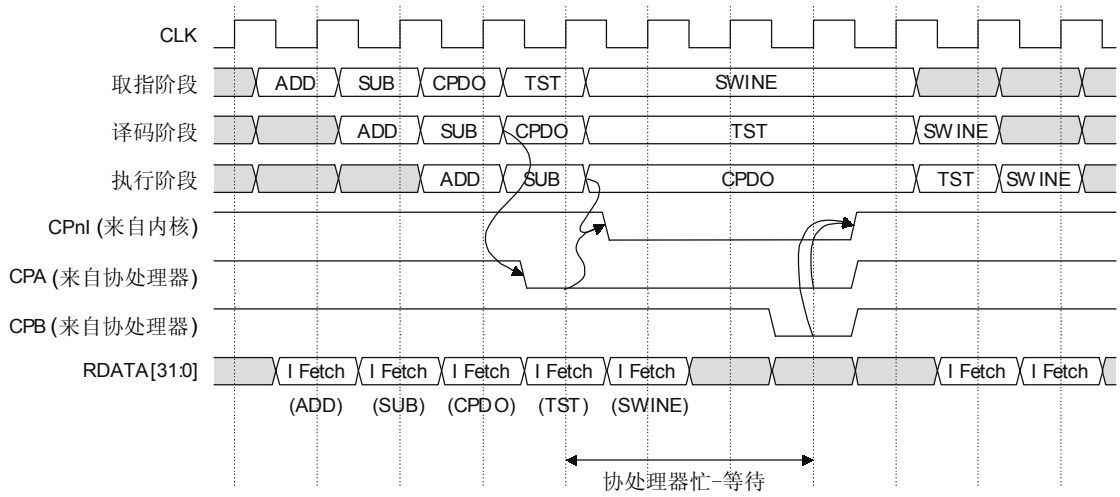

图 4-1 协处理器“忙-等待”时序

4.4.4 忙-等待的后果

一个遇忙等待的协处理器指令可以被中断。如果发生一个有效的 FIQ 和 IRQ（CSPR 中的适当位清零），ARM7TDMI-S 内核放弃协处理器指令，并通过将 CPnI 置为高电平来通知协处理器。协处理器必须监视 CPnI 信号以检测到这一状况。当 ARM7TDMI-S 内核放弃协处理器时，协处理器也放弃这条指令并继续跟踪 ARM7TDMI-S 处理器流水线。

警告：

协处理器所执行的任何动作都不能破坏协处理器的状态，并且可重复地得到相同的结果。协处理器只有在指令执行完成之后才能改变它自身的状态。

4.4.5 协处理器寄存器传输指令

协处理器寄存器传输指令 MCR 和 MRC 在 ARM7TDMI-S 处理器寄存器组的一个寄存器和协处理器寄存器组的一个寄存器之间传递数据。图 4-2 所示为协处理器寄存器传输的时序。

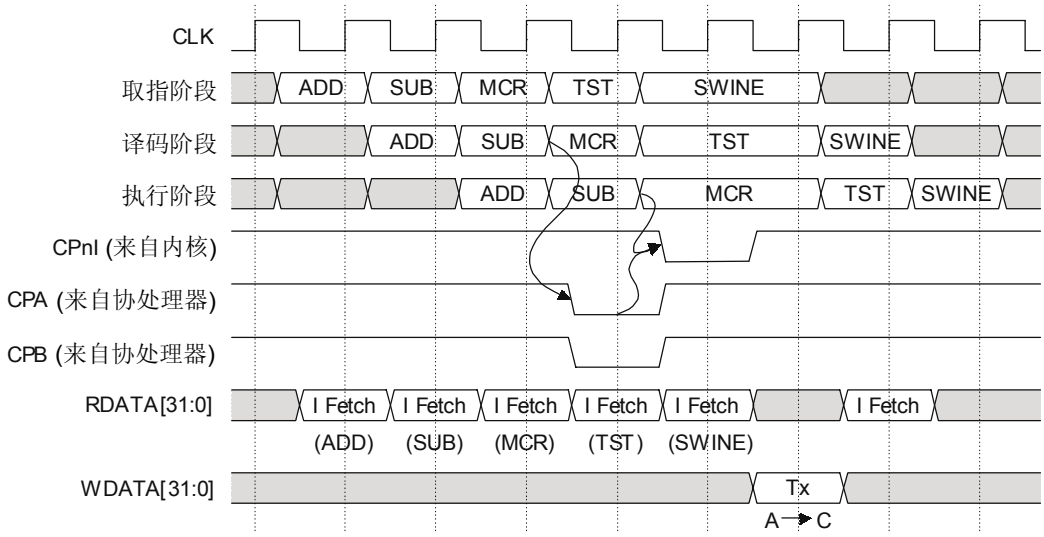

图 4-2 协处理器寄存器传输时序

4.4.6 协处理器数据操作

协处理器数据操作 CDP 指令对保存在协处理器寄存器组当中的数据进行处理。该操作不会使

ARM7TDMI-S 内核与协处理器之间产生信息传递。图 4-3 所示为操作时序。

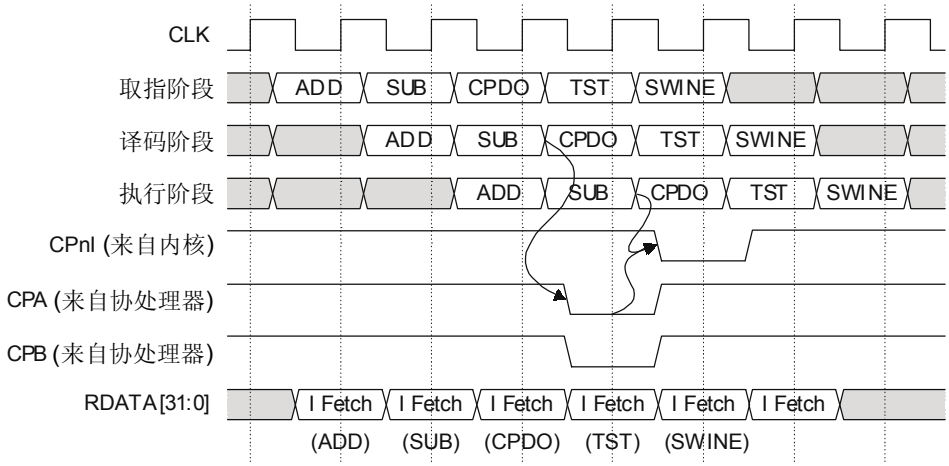

图 4-3 协处理器数据操作时序

4.4.7 协处理器装载和保存操作

协处理器装载和保存指令用于在协处理器和存储器之间传输数据。它们可用于传输单个字数据或一些协处理器寄存器数据。单个 LDC 和 STC 指令对传输的数据字的数目没有限制，但是按照惯例，协处理器单个指令传输的数据字的数目必须大于 16。图 4-4 所示为装载时序。

注：如果在单个指令中传输的数据大于 16 个字，最坏情况下的 ARM7TDMI-S 内核的中断延迟时间将会增加。

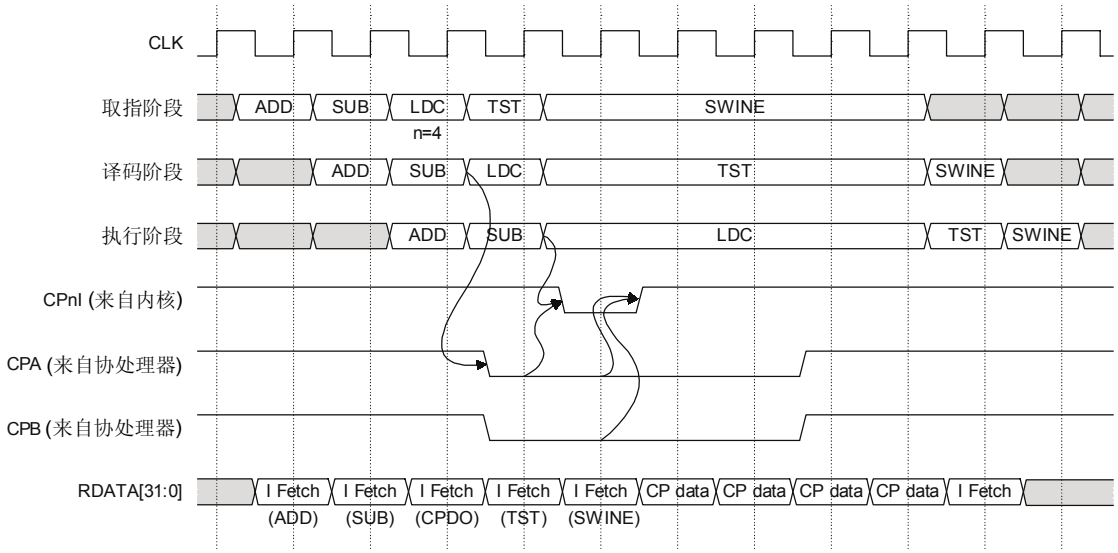

图 4-4 协处理器装载时序

4.5 连接协处理器

基于 ARM7TDMI-S 处理器的系统中的协处理器必须具有如下的 32 位连接：

- 从存储器读取数据（指令流和 LDC）
- ARM7TDMI-S（MCR）写数据
- ARM7TDMI-S（MRC）读数据

4.5.1 连接单个协处理器

图 4-5 所示为将一个协处理器连接到基于 ARM7TDMI-S 处理器的系统。

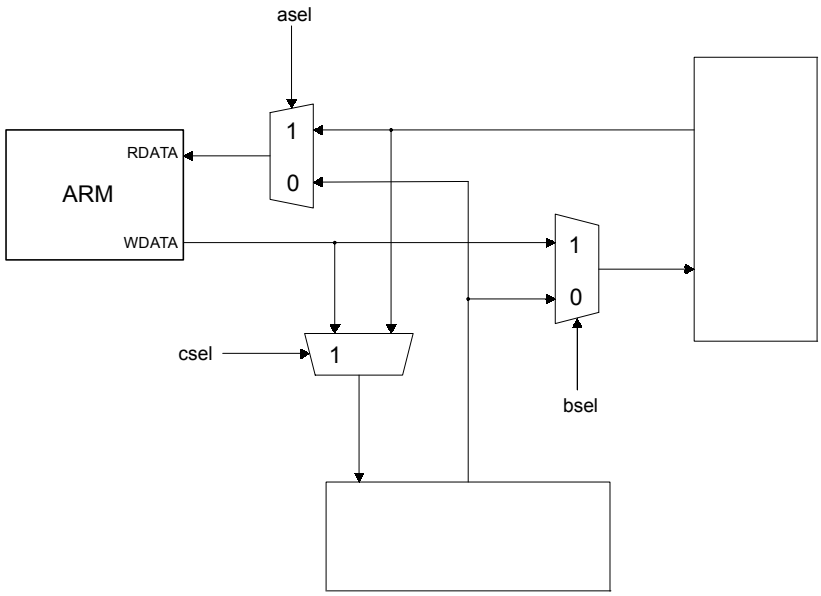

图 4-5 协处理器连接

这一节讲述寄存器逻辑从相关的 ARM7TDMI-S 处理器或 ARM7TDMI-S 处理器管脚取得 asel, bsel 和 csel 的 Verilog 片断。

asel, bsel 和 csel 的逻辑如下：

```
assign asel = ~(cprt | (cpdt & nRW_r));
assign bsel = ~cpdt;
assign csel = cprt;
assign cpdt = ~nMREQ_r & ~CPA_r2 & nOPC_r;
assign cprt = nMREQ_r & SEQ_r;
```

注：cpdt 表示由于一个 LDC 或 STC 指令，因此当前周期是一个装载或保存周期。

cprt 表示当前周期是一个协处理器寄存器传输周期。

其它信号如下所示：

```
always @(posedge CLK)
if (CLKEN)
    begin
        nMREQ_r <= CPnMREQ;     // Output from ARM7TDMI-S
            SEQ_r <= CPSEQ;     // Output from ARM7TDMI-S
            nOPC_r <= CPnOPC;   // Output from ARM7TDMI-S
            nRW_r <= WRITE;     // Output from ARM7TDMI-S
            CPA_r <= CPA;       // Input to ARM7TDMI-S
            CPA_r2 <= CPA_r;
        end
```

注： 如果您使用 ETM 和 ARM7TDMI-S 处理器构造一个系统，必须将 ETM7 RDATA[31:0]和 WDATA[31:0]直接连接到 ARM7TDMI-S 的 RDTA[31:0]和 WDATA[31:0]总线。这样使 ETM 可以直接跟踪协处理器指令。

4.5.2 连接多个协处理器

如果在系统中有多个协处理器，按照表 4-3 所示连接握手信号。

表 4-3 握手信号的连接

信号	连接
CPnI	将该信号连接到系统中所有的协处理器
CPA 和 CPB	每个协处理器独立的 CPA 和 CPB 必须相与然后连接到 ARM7TDMI-S 处理器的 CPA 和 CPB。

协处理器的输出数据也必须多路传输。

4.6 不使用外部协处理器

如果您实现的系统不包含任何外部协处理器，那么必须将 CPA 和 CPB 都固定为高电平。这样就表示系统中没有外部协处理器。如果接收到任何协处理器指令，则执行未定义指令陷阱，这样可以在需要的时候对它们进行软件仿真。

与 ARM7TDMI-S 协处理器相关的输出必须悬空：

- **CPnMREQ**
- **CPSEQ**
- **CPnTRANS**
- **CPnOPC**
- **CPnI**
- **CPTBIT**

4.7 未定义的指令

ARM7TDMI-S 处理器执行完全的 ARM 结构 v4T 未定义指令的处理。这意味着 ARM 体系结构参考手册中定义为 UNDEFINED 的任何指令都会使 ARM7TDMI-S 处理器执行未定义指令陷阱。任何一个不被协处理器接受的指令也会使 ARM7TDMI-S 处理器执行未定义指令陷阱。

4.8 特权指令

CPnTRANS 输出信号使能只有在特权模式下才能访问的协处理器指令。信号的含义如表 4-4 所示：

表 4-4 CPnTRANS 信号含义

CPnTRANS	含义
低	用户模式指令
高	特权模式指令

CPnTRANS 信号与指令同时被采样，并进入协处理器流水线译码阶段。

注：如果用户模式处理（CPnTRANS 为低）试图访问只能在特权模式下才能执行的协处理器指令，协处理器必须将 CPA 和 CPB 置高作为响应。这样使 ARM7TDMI-S 处理器执行未定义指令陷阱。

第 5 章 调试您的系统

这一章讲述 ARM7TDMI-S 处理器的调试特性。包含下列小节：

- 关于调试系统
- 控制调试
- 进入调试状态
- 调试接口
- ARM7TDMI-S 内核时钟域
- Embedded-RT 宏单元
- 禁止 Embedded-RT
- 调试通信通道
- 扫描链和 JTAG 接口
- 复位 TAP 控制器
- 公共 JTAG 指令
- 测试数据寄存器
- 扫描时序
- 在调试状态中检查内核和系统
- 调试当中的程序计数器
- 优先级和异常
- 观察点单元寄存器
- 编程断点
- 编程观察点
- 中止状态寄存器
- 调试控制寄存器
- 调试状态寄存器
- 结合断点和观察点
- Embedded-RT 时序

5.1 关于调试系统

ARM7TDMI-S(Rev 4)处理器的高级调试特性使应用程序、操作系统和硬件的开发变得更加容易。

5.2 一个典型的调试系统

ARM7TDMI-S 处理器构成了调试系统的一个部件，它作为您执行的高级调试与 ARM7TDMI-S 所支持的低级调试之间的接口。图 5-1 所示为一个典型的调试系统。

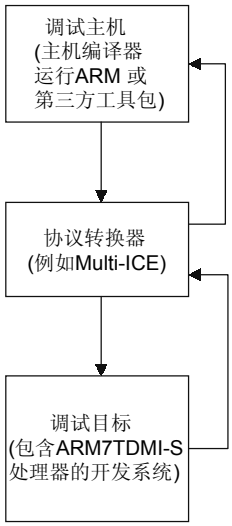

图 5-1 典型的调试系统

一个调试系统通常具有 3 个部分：

调试主机

一台运行调试软件（例如 ARM 的 Windows 版调试器 ADW）的计算机。调试主机使您可以使用设置断点或检查存储器内容这些高级命令。

协议转换器

调试主机发出的高级命令与 ARM7TDMI-S 处理器 JTAG 接口的低级命令之间的接口。典型地，它通过一个接口（例如增强型并口）与主机相连。

调试目标

ARM7TDMI-S 处理器具有便于进行底层调试的硬件扩展。这些扩展可使您：

- 暂停程序的执行
- 检查和修改内核的内部状态
- 检查存储器系统的状态
- 执行中止异常，允许实时监控内核
- 恢复程序执行

调试主机和协议转换器与系统有关。

5.2 控制调试

ARM7TDMI-S 处理器的主要模块分别为：

ARM CPU 内核 它具有支持调试的硬件

Embedded-RT 宏单元 使用一套寄存器和比较器产生调试异常（例如断点）。该单元将在 *Embedded-RT 宏单元*一节中详细讲述。

TAP 控制器 使用 JTAG 串行接口控制扫描链的动作，见 *TAP 控制器*一节。

这些模块如图 5-2 所示。

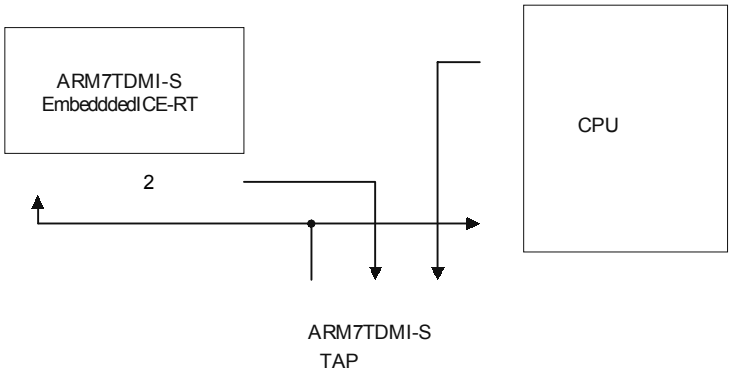

图 5-2 ARM7TDMI-S 框图

5.2.1 调试模式

您可以在下列任意一种模式下执行调试：

暂停模式

当系统处于暂停模式时，内核在遇到断点或观察点时进入调试状态。在调试状态下，内核停止工作并与系统其它部分隔离。当调试结束时，调试主机恢复内核和系统状态，并恢复程序的执行。详见*进入调试状态*一节。

监控模式

当系统处于监控模式时，内核在遇到断点或观察点时不进入调试状态，而是产生一个指令中止或数据中止，内核像正常情况下一样接收和服务中断。可以使用中止状态寄存器来确定异常是由断点或观察点或是真正的存储器中止而造成的。详见*监控模式调试*一节。

5.2.2 在调试中检查系统状态

在暂停模式和监控模式下，可以通过 JTAG 接口检查内核的内部状态和系统的外部状态（在系统处于活动状态时）。

在暂停模式下，这使指令可以在不使用外部数据总线的情况下，串行插入内核的流水线当中。例如，当处于调试模式时，STM 指令可插入到指令流水线当中，从而输出 ARM7TDMI-S 处理器的寄存器内容。这些数据可以串行移出，不会影响系统的其它部分。详见*在调试状态下检查内核与系统*一节。

在监控模式下，JTAG 接口用于传输在调试器与 ARM7TDMI-S 内核上运行的简单监控程序之间的数据。详见*扫描链与 JTAG 接口*一节。

5.3 进入调试状态

如果系统处于暂停模式，下面任何一种类型的中断都会使处理器进入调试状态：

- 断点（给定的指令取指）
- 观察点（数据访问）
- 外部调试请求

注：在监控模式下，处理器继续实时执行指令，并执行一个中止异常。中止状态寄存器可用于确定异常是由断点或观察点或是真正的存储器中止而造成的。

您还可以使用 EmbeddedICE-RT 逻辑编程在什么条件下会产生断点或观察点。另外，您可以使用 DBGBREAK 信号使能外部逻辑对断点或观测点进行标记，并监控：

- 地址总线

- 数据总线
- 控制信号

外部产生的断点和观察点的时序都相同。数据必须在 CLK 的上升沿有效。当数据是断点所在的指令时，DBGBREAK 信号必须在 CLK 上升沿保持高电平。相似地，当数据被装载或储存时，在 CLK 上升沿声明 DBGBREAK 会把数据看作是观察点所在的数据。

当产生断点或观察点时，在 ARM7TDMI-S 内核进入调试状态之前可能有一个延迟。当内核进入调试状态时，DBGACK 信号声明。外部产生的断点时序如图 5-3 所示。

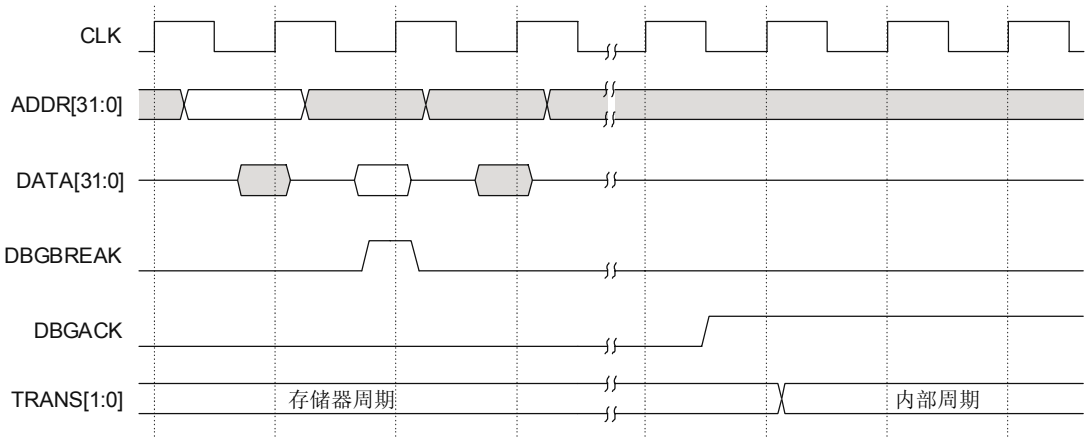

图 5-3 进入调试状态

5.3.1 在断点处进入调试状态

ARM7TDMI-S 处理器在指令进入流水线时，将指令标示为断点，但是在指令到达执行阶段之前，内核不会进入调试状态。

标示断点的指令不会被执行，而是使 ARM7TDMI-S 内核进入调试状态。当检查内部状态时，在断点指令之前观察状态。当检查完成之后，将断点移去，程序从标示断点的指令处开始恢复执行。

当一个条件断点指令到达流水线的执行阶段时，如果系统处于暂停模式，断点总是会被提取出来。ARM7TDMI-S 内核不管是否满足指令条件都会进入调试状态。

当发生下列情况时，标示断点的指令不会使 ARM7TDMI-S 内核进入调试状态：

- 在标示断点的指令之前有分支或写 PC 的指令。这种情况下，执行分支时，ARM7TDMI-S 处理器会清洗指令流水线，并因此而取消断点。
- 当发生异常时，会导致 ARM7TDMI-S 处理器清洗指令流水线，并取消断点。在正常环境下，从异常退出时，ARM7TDMI-S 内核会跳转到发生异常的前一条指令。这种情况下，流水线被重新填充，而断点也被重新标记。

5.3.2 在观察点处进入调试状态

在访问数据时产生观察点。在暂停模式下，内核的处理被停止。在监控模式下，会执行中止异常。观察点总是会被取出，但在暂停模式下，内核可能不会立即进入调试状态，因为要等待当前指令执行完毕。如果当前指令是一个多字装载或存储指令（LDM 或 STM），需要等待许多个周期才会将观察点取出。

处于观察点时会发生下列动作：

- 当前指令结束。
- 产生所有内核状态的变化。
- 装载数据写入目标寄存器。
- 执行基址读回。

注：观察点与数据中止相似。区别在于，当发生数据中止时，即使指令结束，ARM7TDMI-S 内核仍然

防止处理器状态出现任何改变。这样使异常处理程序能够消除引起异常的原因，使指令得以恢复执行。

如果在异常挂起的时候产生一个观察点，内核进入调试状态的模式与进入异常的模式相同。

5.3.3 通过调试请求进入调试状态

暂停模式下的 ARM7TDMI-S 内核可通过调试请求强制进入调试状态，方式有两种：

- 通过 EmbeddedICE-RT 编程
- 通过声明 DBGRQ 管脚

当 DBGRQ 管脚被声明后，内核在当前指令结束时正常进入调试状态。但如果当前指令访问协处理器时处于忙-等待，指令会被终止，ARM7TDMI-S 内核立即进入调试状态。这和 nIRQ 和 nFIQ 的动作相似。

5.3.4 ARM7TDMI-S 在调试状态下的动作

当 ARM7TDMI-S 处理器进入调试状态时，内核强制 TRANS[1:0]指示内部周期。这一动作使存储器系统忽略 ARM7TDMI-S 内核并像正常情况下一样工作。由于系统的其余部分继续工作，ARM7TDMI-S 内核被强制忽略中止和中断。

警告：在调试过程中不要复位内核，否则调试程序会失去对内核的跟踪。

注：系统在调试时不能改变 CFGBIGEND 信号。从程序员的角度出发，如果 CFGBIGEND 改变了，ARM7TDMI-S 处理器改变了，而调试程序不知道内核已经复位。必须确保调试过程中 nRESET 处于稳定的低电平。当系统将 ARM7TDMI-S 处理器复位时（即，将 nRESET 驱动为低电平），处理器状态改变，而调试程序不知道内核已经复位。

5.3.5 时钟

系统和测试时钟必须与宏单元外部同步。ARM Multi-ICE 调试工具在一个 ASIC 设计中直接支持一个或多个内核。要使片外调试时钟与 ARM7TDMI-S 宏单元同步，需要一个 3 阶同步器。片外设备（例如 Multi-ICE）提供一个 TCK 信号并等待 RTCK（返回的 TCK）信号返回。由于片外设备只有接收到 RTCK 才前进到下一个 TCK，因此使同步得以维持，见图 5-4。

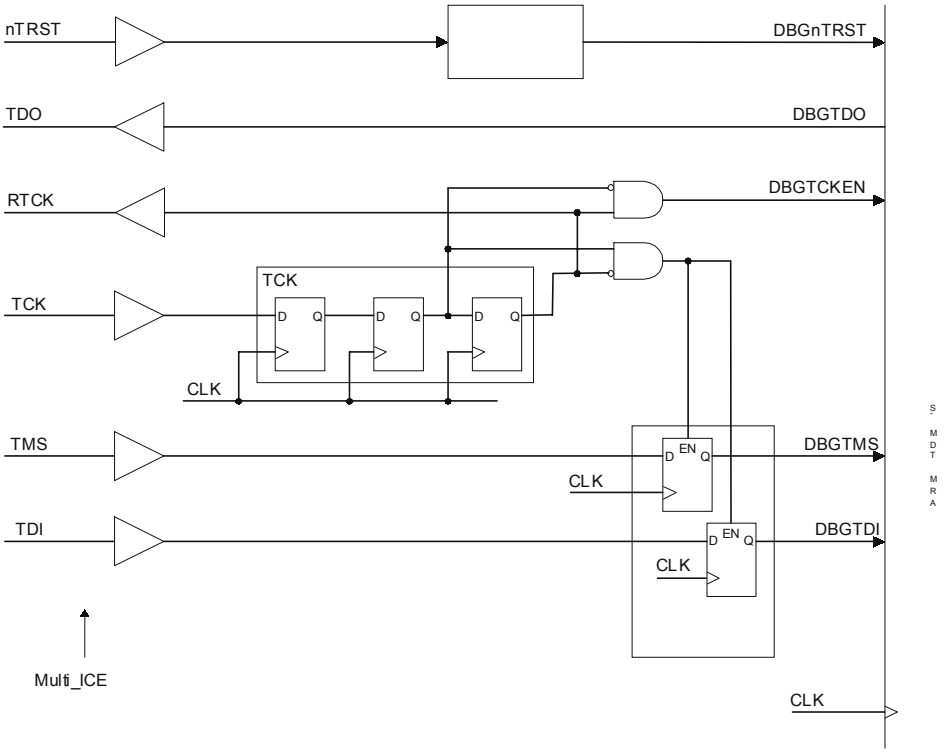

图 5-4 时钟同步

注：图 5-4 中所示所有 D 型都由 DBGnTRST 复位。

5.4 调试接口

ARM7TDMI-S 处理器的调试接口是建立在 IEEE 1149.1-1990 标准、*标准测试访问端口和边界扫描结构*基础之上的。这一章中所使用术语的详细解释请参考上述标准和 TAP 控制器状态的描述。

5.4.1 调试接口信号

与调试接口有关的主要外部信号有：

- **DBGBREAK** 和 **DBGRQ** 是 ARM7TDMI-S 内核进入调试状态的系统请求。
- **DBGACK** 被 ARM7TDMI-S 所使用，用来通知系统已经处于调试状态。

5.5 ARM7TDMI-S 内核时钟域

ARM7TDMI-S 处理器有一个单时钟 CLK，它被下面两个时钟使能所限制：

- CLKEN 控制访问存储器系统
- DBGTCKEN 控制调试操作

在正常操作过程中，CLKEN 控制 CLK 为内核提供时钟。当 ARM7TDMI-S 处理器处于调试状态时，DBGTCKEN 控制 CLK 为内核提供时钟。

5.6 EmbeddedICE-RT 宏单元

ARM7TDMI-S 处理器 EmbeddedICE-RT 宏单元模块为 ARM7TDMI-S 内核提供集成的片内调试支持。EmbeddedICE-RT 通过 ARM7TDMI-S 处理器 TAP 控制器串行编程。图 5-5 所示为内核、EmbeddedICE-RT 与 TAP 控制器之间的关系，图中只显示了与 EmbeddedICE-RT 有关的信号。

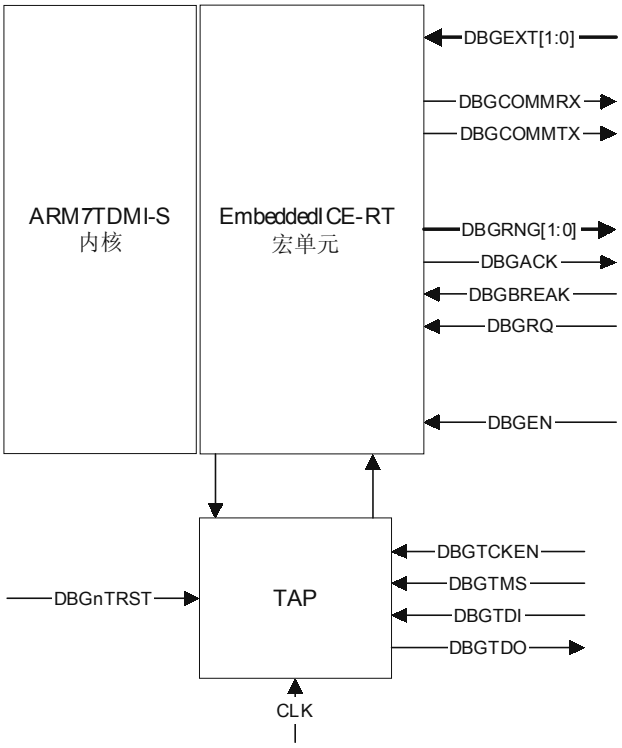

图 5-5 ARM7TDMI-S 内核、TAP 控制器和 EmbeddedICE-RT 宏单元

EmbeddedICE-RT 逻辑包含下面这些部分：

两个实时观察点单元

可以编程这两个观察点或其中一个使内核暂停指令的执行。当编程到 EmbeddedICE-RT 的值与地址总线、数据总线和不同的控制信号上出现的值相匹配时，指令的执行会被暂停。可以屏蔽其中任何一位使它的值不影响比较。

每个观察点单元都可配置为观察点（监视数据的访问）或断点（监视指令取指）。详见*观察点单元寄存器*一节。

中止状态寄存器

该寄存器用于识别产生异常中止的原因。详见*中止状态寄存器*一节。

调试通信通道（DCC）

DCC 在目标系统与主机调试程序之间传递信息。详见*调试通信通道*一节。

此外还有两个独立的寄存器提供对 EmbeddedICE-RT 操作的全局控制：

- 调试控制寄存器
- 调试状态寄存器

EmbeddedICE-RT 寄存器的位置见表 5-1。

5.7 禁止 EmbeddedICE-RT

可以使用以下两种方式禁止 EmbeddedICE-RT：

永久

通过将 DBGEN 输入固定为低电平，当 DBGEN 为低电平时：

- DBGBREAK 和 DBGRQ 被内核忽略
- DBGACK 被 ARM7TDMI-S 内核强制为低电平
- 输入到处理器的中断未被禁止
- EmbeddedICE-RT 逻辑进入低功耗模式

警告：将 DBGEN 输入端固定为低电平会永久禁止调试访问。但是您不能依赖这一点来保证系统的安全性。

暂时

通过置位调试控制寄存器的 bit5。bit5 是 EmbeddedICE-RT 的禁止位。

在执行下面的动作之前必须置位 bit5：

- 编程断点或观察点寄存器。
- 改变调试控制寄存器的 bit4。

5.8 EmbeddedICE-RT 寄存器映射

EmbeddedICE-RT 寄存器的位置如表 5-1 所示：

表 5-1 EmbeddedICE-RT 寄存器的功能和地址

地址	宽度	功能
b00000	6	调试控制
b00001	5	调试状态
b00100	32	调试通信通道（DCC）控制寄存器
b00101	32	调试通信通道（DCC）数据寄存器
b01000	32	观察点 0 地址值

b01001	32	观察点 0 地址屏蔽
b01010	32	观察点 0 数据值
b01011	32	观察点 0 数据屏蔽
b01100	9	观察点 0 控制值
b01101	8	观察点 0 控制屏蔽
b10000	32	观察点 1 地址值
b10001	32	观察点 1 地址屏蔽
b10010	32	观察点 1 数据值
b10011	32	观察点 1 数据屏蔽
b10100	9	观察点 1 控制值
b10101	8	观察点 1 控制屏蔽

5.9 监控模式调试

ARM7TDMI-S(Rev 4)处理器包含的逻辑可使能在不完全停止内核的情况下对系统进行调试。这意味着内核正在被调试程序询问时也能继续服务重要的中断程序。

5.9.1 使能监控模式

调试模式由调试控制寄存器的 bit4 所控制。bit4 是监控模式使能位：

bit4 置位

使能 ARM7TDMI-S 处理器的监控模式特性。当该位置位时，EmbeddedICE-RT 逻辑配置为断点或观察点可使 ARM7TDMI-S 内核进入中止模式，分别提取预取指或数据中止向量。

bit4 清零

监控模式调试被禁止，而系统进入暂停模式。在暂停模式下，内核在遇到断点或观察点时进入调试状态。

5.9.2 监控模式调试的限制

当 ARM 内核配置为监控模式调试时，您必须注意以下几点限制：

- 断点和观察点在监控模式下不能由数据决定。不支持测距功能的使用。断点和观察点只支持下列条件：
  - 指令或数据地址
  - 外部观察点条件（DBGEXT[0]或 DBGEXT[1]）
  - 用户或特权模式访问（CPnTRANS）
  - 对观察点的读/写访问（WRITE）
  - 访问规格（观察点 SIZE[1:0]）
- 不支持外部断点和观察点
- 不支持混合的暂停模式和监控模式

由监控模式产生的一个中止事件记录在协处理器 14 的中止状态寄存器当中（见*中止状态寄存器*一节）。

监控模式使能位不会使 ARM7TDMI-S 处理器进入调试状态。因此在访问外部存储器时，有必要改变观察点寄存器的内容，而不是在内核中止时的调试状态中改变它们。

如果在改变观察点寄存器的过程中有可能产生错误匹配（由一些寄存器中的旧数据和另一些寄存器中的新数据所引起），那么您必须：

1. 通过置位调试控制寄存器中的 BIT5（也称 EmbeddedICE-RT 禁止位）禁止观察点单元；
2. 轮询调试控制寄存器，直到 EmbeddedICE-RT 禁止位读出为 1 为止；

3. 改变其它寄存器；
4. 通过清零 EmbeddedICE-RT 禁止位，重新使能观察点单元。

5.10 调试通信通道

ARM7TDMI-S (Rev 4) EmbeddedICE-RT 包含一个调试通信通道（DCC），它用于在目标板和主机调试器之间传递信息。这由协处理器 14 实现。

DCC 包含两个寄存器：

DCC 控制寄存器

32 位寄存器，用于处理器和一般调试器之间的同步握手。详见 *DCC 控制寄存器*一节。

DCC 数据寄存器

32 位寄存器，用于调试器和处理器之间的数据传输。详见*通过 DCC 进行通信*一节。

这些寄存器占用 EmbeddedICE-RT 存储器映射中的固定位置，如表 5－1 所示。处理器使用对协处理器 14 的 MCR 和 MRC 指令对它们进行访问。

寄存器按照下面的方式进行访问：

被调试器访问： 按照通常的方式，通过扫描链 2 访问。

被处理器访问：通过协处理器寄存器传输指令访问。

5.10.1 DCC 控制寄存器

DCC 控制寄存器只读，它使能处理器和调试器之间的同步握手。寄存器的格式如图 5-6 所示：

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0	0	0	1		W	R

图 5-6 DCC 控制寄存器

DCC 寄存器位的分配见表 5-2

表 5-2 DCC 控制寄存器位分配

位	功能
31：28	包含固定格式，用于指明 EmbeddedICE-RT 的版本号，此处为 b0001。
27：2	保留
1	写控制位 如果该位清零，DCC 数据写寄存器准备接收来自处理器的数据 如果该位置位，表示 DCC 数据写寄存器中有数据，调试器可将其扫描读出。
0	读控制位 如果该位清零，DCC 数据读寄存器准备接收来自调试器的数据； 如果该位置位，表示 DCC 数据读寄存器包含处理器还没有读出的新数据，调试器必须等待。

注：如果执行被暂停，bit0 可能会保持置位状态，调试器可通过写 DCC 控制寄存器将其清零。只有很少的时候会需要写 DCC 控制寄存器，因为在正常的操作中，处理器会在读取 之后将 bit0 清零。

指令

必须使用下面的指令：

MRC CP14,0,Rd,C0,C0

将 DCC 控制寄存器中的值返回到目标寄存器 Rd。

MCR CP14,0,Rn,C1,C0

将源寄存器 Rn 中的值写入 DCC 数据写寄存器。

MRC CP14,0,Rd,C1,C0

将 DCC 数据读寄存器中的值返回到源寄存器 Rd 中。

注：Thumb 指令集并不包含协处理器指令，因此在 Thumb 状态下推荐使用 SWI 指令进行寻址。

5.10.2 通过 DCC 进行通信

可以通过 DCC 发送和接收消息。

将消息发送到调试器

当处理器希望将一个消息发送到调试器时，它必须通过检查 DCC 控制器中的 W 位是否清零来确定 DCC 数据写寄存器是否处于空闲状态。

处理器读取 DCC 控制器来检查 W 位的状态：

- 如果 W 位清零，表示 DCC 数据写寄存器清零。
- 如果 W 位置位，表示调试器还没有读出先前写入的数据。处理器必须继续查询控制寄存器，直到 W 位清零为止。

当 W 位清零时，写入的数据通过寄存器传送到协处理器 14。当数据从处理器传送到 DCC 数据写寄存器时，W 位置位。

调试器通过 JTAG 接口查询 DCC 控制寄存器时，可同时看到 R 和 W 位。它可以读取通信数据写寄存器并扫描数据输出。读该数据寄存器的动作将调试通信控制寄存器 W 位清零。此时通信处理可以再次开始。

从调试器接收消息

将消息从调试器发送到处理器与上面提到的将消息发送到调试器相似。此时，调试器查询调试通信控制寄存器的 R 位：

- 如果 R 位为 0，表示通信数据读寄存器为空闲状态，数据可存在该寄存器供处理器读取。
- 如果 R 位置位，表示先前存放的数据还没被读取，因此调试器必须等待。

当通信数据读寄存器空闲时，数据通过 JTAG 接口写入。写入的动作使调试通信控制寄存器中的 R 位置位。

处理器轮询调试通信控制寄存器，如果 R 位置位，可以饲养 MRC 指令将数据装入协处理器 14。此装载动作使 R 位清零。当调试器轮询该寄存器并看见 R 位清零，表示数据已经被取走，处理器可以重复执行。

5.11 扫描链和 JTAG 接口

在 ARM7TDMI-S 处理器中有两个扫描链用于实现调试和 EmbeddedICE-RT 编程。JTAG 类型的测试访问端口（TAP）控制器控制扫描链。关于 JTAG 规范的详细信息请参阅 IEEE 标准 1149.1-1990。

5.11.1 扫描链的实现

两个扫描路径分别指扫描链 1 和扫描链 2，如图 5-7 所示。ARM7TDMI-S 处理器没有实现扫描链 0。

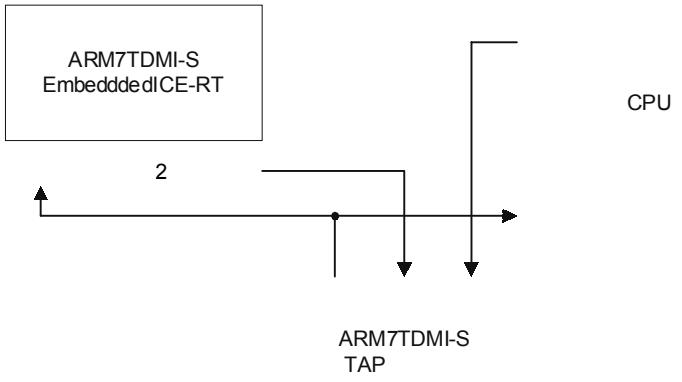

图 5-7 ARM7TDMI-S 扫描链排列

扫描链 1

扫描链 1 提供对内核数据总线 RDATA/WDATA 和 DBGBREAK 信号的串行访问。

在该扫描链当中有 33 个位，顺序如下（从串行数据输入到输出）：

- 数据总线位 0 到 31
- DBGBREAK 位（最先移出）

扫描链 2

扫描链 2 使能对 EmbeddedICE-RT 寄存器的访问。见*测试数据寄存器*一节。

5.11.2 控制 JTAG 接口

JTAG 接口由指令寄存器中当前装载的指令驱动（见*指令寄存器*一节）。装载的指令由 TAP 控制器控制。

5.12 TAP 控制器

TAP 控制器是决定 ARMYTDMI-S 边界扫描测试信号 **DBGTDI** 和 **DBGTDO** 状态的状态机。图 8 所示为发生在 TAP 控制器当中的状态转换。

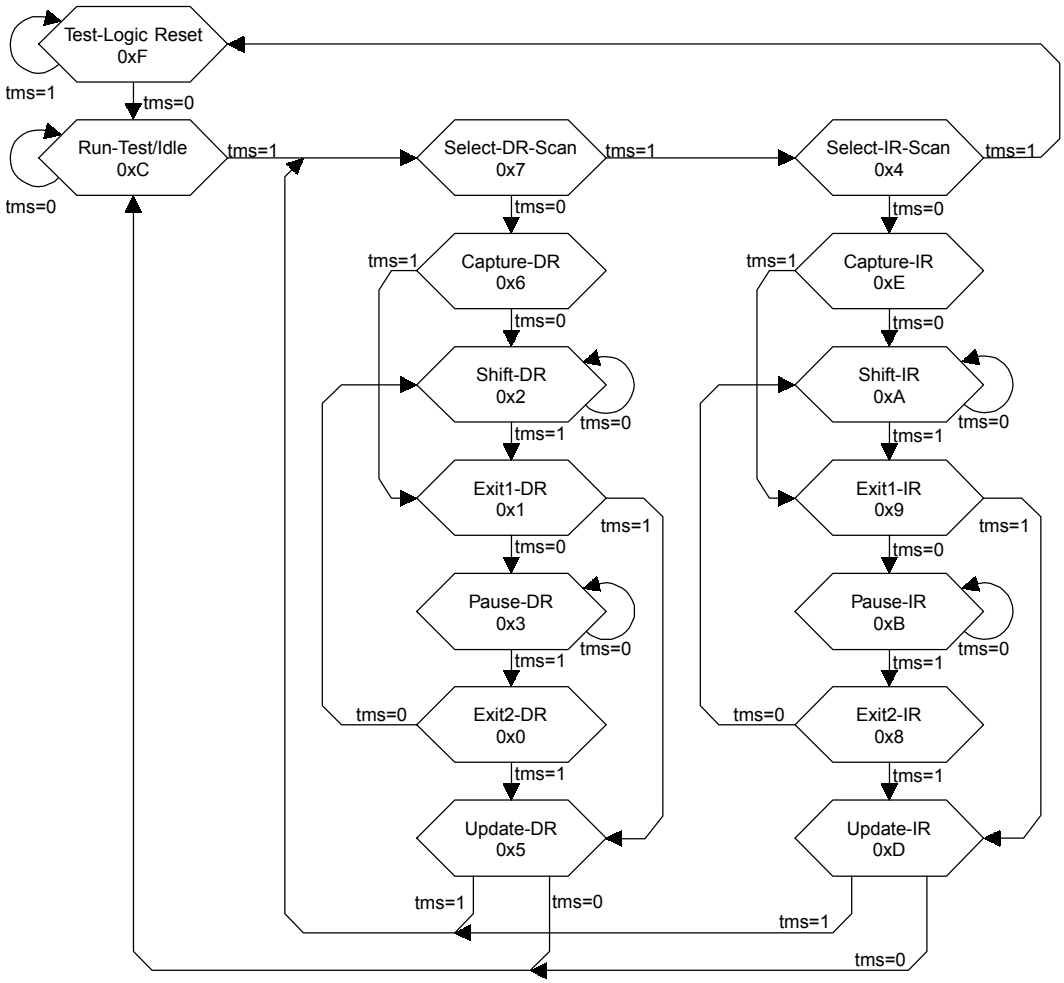

图 8 测试访问端口控制器状态转换

取自 IEEE Std 1149.1-1990, 2001 IEEE 版权所有

5.12.1 复位 TAP 控制器

要强制使 TAP 控制器在复位后进入正确的状态，您必须向 **DBGnTRST** 信号端提供一个复位脉冲。

- 当使用边界扫描接口是，**DBGnTRST** 必须驱动为低，然后恢复为高电平。
- 当没有使用边界扫描接口时，可将 **DBGnTRST** 输入端固定为低电平。

注：复位芯片时，**CLK** 上的时钟（**DBGTCKEN** 为高电平）是不需要的。

复位的动作如下：

1. 选择系统模式。这意味着边界扫描单元不截取外部系统和内核之间传递的任何信号。
2. 选择 IDCODE 指令。

当 TAP 控制器进入 SHIFT-DR 状态，并且 **DBGTCKEN** 使能提供 **CLK** 信号时，ID 寄存器的内容从 **DBGTDO** 输出。

5.13 公共 JTAG 指令

表 5-3 所示为公共 JTAG 指令。

表 5-3 公共 JTAG 指令

指令	二进制代码

SCAN_N	0010
INTEST	1100
IDCODE	1110
BYPASS	1111
RESTART	0100

在下面的描述中，ARM7TDMI-S 处理在 CLK 上升沿并且 DBGTCKEN 为高时对 DBGTDI 和 DBGTMS 进行采样。TAP 控制器状态如图 5-8 所示。

5.13.1 SCAN_N(0010)

SCAN_N 指令连接 DBGTDI 阿 DBGTDO 之间的扫描路径选择寄存器：

- 在 CAPTURE-DR 状态下，将固定值 1000 装入寄存器。
- 在 SHIFT-DR 状态下，将选择的扫描路径的 ID 号装入寄存器。
- 在 UPDATE-DR 状态下，被选择扫描链的扫描寄存器连接 **DBGTDI** 和 **DBGTDO**，并一直保持连接直到执行下一条 SCAN_N 指令为止。
- 复位时，默认选择扫描链 0。

此时扫描路径选择寄存器长度为 4 位，即使没有指定长度。

5.13.2 INTEST (1100)

INTEST 指令使被选择的扫描链进入测试模式：

- INTEST 指令连接 **DBGTDI** 和 **DBGTDO** 之间被选择的扫描链
- 当 INTEST 指令装入指令寄存器时，所有扫描单元都进入各自的测试模式。
- 在 CAPTURE-DR 状态下，数据值由内核逻辑提供给输出扫描单元，而系统逻辑提供给输入扫描单元的数据值被捕获。
- 在 SHIFT-DR 状态下，之前被捕获的测试数据通过 **DBGTDO** 管脚从扫描链移出，而新的测试数据通过 **DBGTDI** 管脚移入。

使用 INTEST 指令可实现内核的单步操作。

5.13.3 IDCODE (1110)

IDCODE 指令连接 **DBGTDI** 和 **DBGTDO** 之间的器件标识代码寄存器（ID 寄存器）。ID 寄存器是一个 32 位寄存器，可通过 TAP 从中读出制造商、产品编号和部件版本。详见 *ID 寄存器*一节。

当 IDCODE 指令装入指令寄存器时，所有扫描单元都进入各自的正常（系统）操作模式：

- 在 CAPTURE-DR 状态下，器件标识代码被捕获到 ID 寄存器中
- 在 SHIFT-DR 状态下，之前捕获的器件标识代码通过 **DBGTDO** 管脚从 ID 寄存器移出，而新的数据通过 **DBGTDI** 管脚移入 ID 寄存器。
- 在 UPDATE-DR 状态下，ID 寄存器不受影响。

5.13.4 BYPASS (1111)

BYPASS 指令连接 **DBGTDI** 和 **DBGTDO** 之间的 1 位移位寄存器（旁路寄存器）。

当 BYPASS 指令装入指令寄存器时，PYPASS 指令对系统管脚无影响。

- 在 CAPTURE-DR 状态下，逻辑 0 捕获到旁路寄存器。
- 在 SHIFT-DR 状态下，测试数据通过 **DBGTDI** 移入旁路寄存器，并在延迟一个 CLK 周期后从 **DBGTDO** 移出。第一个移出的位为 0。
- 旁路寄存器在 UPDATE-DR 状态下不受影响。

5.13.5 RESTART (0100)

RESTART 指令重新启动处理器使其退出调试模式。RESTART 指令连接 DBGTDI 和 DBGTDO 之间的旁路寄存器。TAP 控制器的表现好像装载了 BYPASS 指令。当进入 RUN-TEST/IDLE 状态时，处理器退出调试状态。详见*退出调试状态*一节

5.14 测试数据寄存器

可连接在 DBGTDI 和 DBGTDO 之间的 6 个测试数据寄存器在下面的章节描述：

- 旁路寄存器
- ARM7TDMI-S 器件标识（ID）代码寄存器
- 指令寄存器
- 扫描路径选择寄存器
- 扫描链 1
- 扫描链 2

在下面的描述中，当 DBGTCKEN 使能信号为高电平时，数据在每个 CLK 周期移出。

5.14.1 旁路寄存器

用途　通过提供一个 DBGTDI 和 DBGTDO 之间的路径，在扫描测试时旁路器件。

长度　1 位

操作模式　当 PYPASS 指令为指令寄存器的当前指令时，在 SHIFT-DR 状态下，串行数据从 **DBGTDI** 输入，并在延迟一个 **CLK** 周期（**DBGTCKEN** 使能）后从 **DBGTDO** 输出。

旁路寄存器无并行输出。

在 CAPTURE-DR 状态下，逻辑 0 装入旁路寄存器的并行输入端。

5.14.2 ARM7TDMI-S 器件标识（ID）代码寄存器

用途　读取 32 位器件标识代码，不提供可编程的补充标识代码。

长度　32 位。ID 代码寄存器的格式见图 5-9。

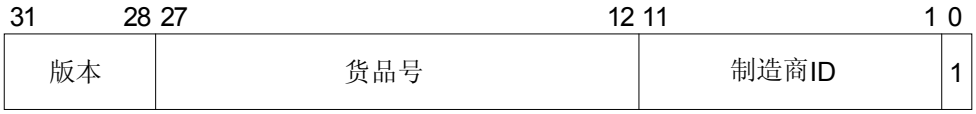

图 5-9　ID 代码寄存器格式

操作模式　当 IDCODE 指令为指令寄存器的当前指令时，ID 寄存器被选择作为 **DBGTDI** 和 **DBGTDO** 之间的串行通路。

ID 寄存器无并行输出。

在 CAPTURE-DR 状态下，32 位器件标识代码装入 ID 寄存器的并行输入端。

5.14.3 指令寄存器

用途　改变当前 TAP 指令。

长度　4 位

操作模式　在 SHIFT-DR 状态下，指令寄存器被选择作为 **DBGTDI** 和 **DBGTDO** 之间的串行通路。

旁路寄存器无并行输出。

在 CAPTURE-DR 状态下，二进制值 0001 装入该寄存器。该值在 SHIFT-IR 状态时移出（最低位在前），而新的指令移入（最低位在前）。

在 UPDATE-IR 状态下，指令寄存器中的值成为当前指令。

在复位时，IDCODE 成为当前指令。

无奇偶位。

5.14.4 扫描路径选择寄存器

用途 改变当前有效的扫描链。

长度 4 位

操作模式 当 SCAN_N 指令为指令寄存器的当前指令时，在 SHIFT-DR 状态下，扫描路径选择寄存器被选择作为 **DBGTDI** 和 **DBGTDO** 之间的串行通路。

在 CAPTURE-DR 状态下，二进制值 1000 装入该寄存器。该值在 SHIFT-DR 状态时移出（最低位在前），而新的值移入（最低位在前）。在 UPDATE-DR 状态下，寄存器中的值选择一个扫描链成为当前有效的扫描链。所有附加的指令，例如 INTEST 将应用于该扫描链。

当前被选择的扫描链只有在执行 SCAN_N 指令或者发生复位时才会被改变。

在复位时，选择扫描链 0 作为有效扫描链。

5.14.5 扫描链 1 和 2

扫描链允许对内核逻辑和 EmbeddedICE-RT 硬件进行串行访问。每个扫描链都很简单，它包含一个串行寄存器和一个多路开关。

扫描单元执行 3 种基本功能：

- 捕获
- 移位
- 更新

对于输入单元，捕获阶段将输入内核的系统值复制到串行寄存器。在移位时，该值串行输出。在多路开关的控制下，输入单元提供给内核的值可以是系统输入，或者是并行寄存器的值。

对于输出单元，捕获阶段将内核输出值放入串行寄存器。在移位时该值串行移出。输出单元提供给系统的值是内核的输出或者是串行寄存器的值。

扫描单元的所有控制信号都由 TAP 控制器内部产生。TAP 控制器的动作由当前指令和 TAP 状态机的状态决定。

扫描链 1

用途 扫描链 1 用于调试器和 ARM7TDMI-S 内核之间的通信。它对数据进行读取/写入，并将指令扫描到流水线。SCAN_N TAP 指令可用于选择扫描链 1。

长度 33 位，其中 32 位为数据值，1 位为 **DBGBREAK** 内核输入。

扫描链顺序 从 **DBGTDI** 到 **DBGTDO**，ARM7TDMI-S 处理器数据位 0~31，接下来是第 33 位 **DBGBREAK** 扫描单元。

扫描链 1 的第 33 位有以下 3 种用途：

- 在正常 INTEST 测试条件下，它使能将一个已知的值扫描到 **DBGBREAK** 输入端。
- 在调试过程中，放入第 33 位的值决定 ARM7TDMI-S 内核在执行指令前是否与系统速度同步。
- 在 ARM7TDMI-S 内核进入调试状态后，被捕获和扫描的第 33 位值告诉调试器内核是从断点（位 33 为低）还是从观察点（位 33 为高）进入调试状态。

扫描链 2

用途 扫描链 2 提供对 EmbeddedICE-RT 寄存器的访问。为了实现这一点，扫描链 2 必须通过 SCAN_N TAP 控制器指令选择，而且 TAP 控制器必须进入 INTEST 模式。

长度 38 位。

扫描链顺序 从 **DBGTDI** 到 **DBGTDO**，读/写位，寄存器地址位，bit4~0，然后是数据位 bit0~31。

在 CAPTURE-DR 状态下不发生任何动作。

在 SHIFT-DR 状态下，数据值移入串行寄存器。bit32~36 指定所要访问的 EmbeddedICE-RT 寄存器的地址。

在 UPDATE-DR 状态下，该寄存器处于读还是写状态取决于 bit37 的值（0＝读，1＝写），见图 5-12。

5-15 扫描时序

图 5-10 提供了通用的扫描时序信息。

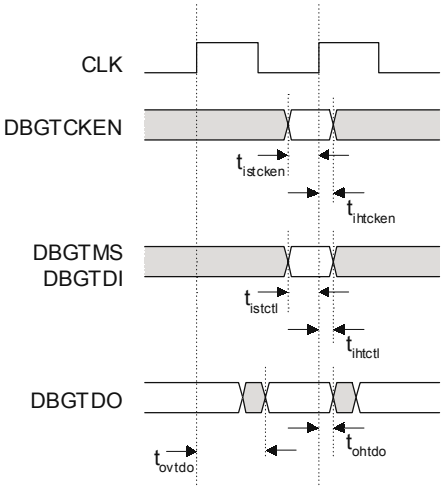

图 5-10 扫描时序

5.15.1 扫描链 1 单元

ARM7TDMI-S 处理器为扫描链 1 提供的数据见表 5-5。

表 5-5 扫描链 1 单元

No.	信号	类型
1	**DATA[0]**	输入/输出
2	**DATA[1]**	输入/输出
3	**DATA[2]**	输入/输出
4	**DATA[3]**	输入/输出
5	**DATA[4]**	输入/输出
6	**DATA[5]**	输入/输出
7	**DATA[6]**	输入/输出
8	**DATA[7]**	输入/输出
9	**DATA[8]**	输入/输出
10	**DATA[9]**	输入/输出
11	**DATA[10]**	输入/输出
12	**DATA[11]**	输入/输出
13	**DATA[12]**	输入/输出
14	**DATA[13]**	输入/输出
15	**DATA[14]**	输入/输出
16	**DATA[15]**	输入/输出
17	**DATA[16]**	输入/输出
18	**DATA[17]**	输入/输出

19	**DATA[18]**	输入/输出
20	**DATA[19]**	输入/输出
21	**DATA[20]**	输入/输出
22	**DATA[21]**	输入/输出
23	**DATA[22]**	输入/输出
24	**DATA[23]**	输入/输出
25	**DATA[24]**	输入/输出
26	**DATA[25]**	输入/输出
27	**DATA[26]**	输入/输出
28	**DATA[27]**	输入/输出
29	**DATA[28]**	输入/输出
30	**DATA[29]**	输入/输出
31	**DATA[30]**	输入/输出
32	**DATA[31]**	输入/输出
33	**DBGBREAK**	输入

5.16 在调试状态下检查内核和系统

当 ARM7TDMI-S 处理器处于调试状态下时，您可以通过对指令流水线强制执行多路装载和保存来检查内核与系统状态。

在内核与系统状态能够被检查之前，调试器必须通过检查 EmbeddedICE-RT 调试状态寄存器的 bit4 来确定处理器是从 Thumb 状态还是从 ARM 状态进入调试状态：

- bit4 为高，表示内核从 Thumb 状态进入调试状态；
- bit4 为低，标识内核从 ARM 状态进入调试状态。

5.16.1 决定内核状态

当处理器从 Thumb 状态进入调试状态后，调试器执行的最简单的动作是使内核返回到 ARM 状态。调试器可以接着执行同样的指令序列来决定处理器的状态。

为了使处理器进入 ARM 状态，对内核执行下列的 Thumb 指令序列：

```
STR   R0,[R0 ]   ;在使用前保存 R0
MOV R0,PC        ;将 PC 复制到 R0
STR   R0,[R0 ]   ;将 PC 存入 R0
BX    PC         ;跳入 ARM 状态
MOV R8,R8        ;空操作
MOV R8,R8        ;空操作
```

注：

由于所有 Thumb 指令都只有 16 位长度，您在移位扫描链 1 时可以重复执行指令。例如，BX R0 的编码为 0x4700，将 0x4700 移入扫描链 1 时，调试器不必跟踪处理器执行读数据的那一半总线。

您可以使用下面的 ARM 指令序列决定处理器的状态。

处于 ARM 状态下的处理器通常执行的第一条指令是：

```
STM  R0,{R0-R15}
```

该指令使寄存器的内容出现在数据总线上。接下来就可对这些值进行采样和移出。

注：

使用 R0 作为 STM 指令的基址寄存器只是用来作为演示，上面的指令可以使用任何寄存器。

在决定当前寄存器组的值之后，您可能想要访问分组的寄存器，那么您必须改变模式。通常，模式的改变只能在内核处于特权模式时才会发生。不过当处于调试状态时，一个模式可以改变到任何其它模式。

调试器在退出调试状态时必须恢复初始的模式。例如，如果要求调试器返回用户模式寄存器和 FIQ 模式寄存器的状态，并在超级用户模式下进入调试状态，指令序列如下：

```
STM  R0,{R0-R15}   ;保存当前寄存器
MRS  R0,CPSR
STR  R0,R0         ;保存 CPSR 以决定当前模式
BIC  R0,0x1F       ;清零模式位
ORR  R0,0x10       ;选择用户模式
MSR  CPSR,R0       ;进入用户模式
STM  R0,{R13,R14}  ;保存之前不可见的寄存器
ORR  R0,0x01       ;选择 FIQ 模式
MSR  CPSR,R0       ;进入 FIQ 模式
STM  R0,{R8-R14}   ;保存分组 FIQ 寄存器
```

所有这些指令都以调试速度执行。调试速度远远低于系统速度，这是因为在每个内核时钟之间产生了 33 个时钟来移位指令或数据。对于访问内核状态来说，这样低的访问速度是可以接受的，因为 ARM7TDMI-S 处理器是全静态的。但是您不能使用该模式决定系统复位的状态。

当处于调试状态时，只有下面的指令才能被扫描到指令流水线执行：

- 所有的数据处理操作
- 所有装载，保存，多路装载和多路保存指令
- MSR 和 MRS

5.16.2 决定系统状态

为了符合存储器系统的动态时序要求，任何对系统状态的访问都必须在 **CLKEN** 所限定的时钟下产生。为了执行存储器访问，**CLKEN** 必须强制 ARM7TDMI-S 处理器运行在正常操作模式下。这由扫描链 1 的 bit33 控制。

当扫描链 1 的 bit33 位 **DBGBREAK** 为低时，放置在扫描链 1 当中的指令以调试速度执行。要使指令以系统速度执行，必须在 **DBGBREAK** 为高时将指令扫描到扫描链 1 当中。

在系统速度指令被扫描到数据总线并进入流水线之后，必须将 RESTART 指令装入 TAP 控制器。RESTART 使 ARM7TDMI-S 处理器产生下列动作：

1. 自动切换到 CLKEN 控制。
2. 以系统速度执行指令。
3. 重入调试状态。

当执行执行完毕时，**DBGACK** 为高，内核返回到 **DBGTCKEN** 控制。现在可以选择 TAP 控制器中的 INTEST 并恢复调试。

调试器必须观察 **DBGACK** 和 **TRANS[1:0]**以确定系统速度指令是否执行完毕。为了访问存储器，ARM7TDMI-S 内核在它与系统速度取得同步之后将 **TRANS[1:0]**都置低，这一跳变供存储器控制器仲裁 ARM7TDMI-S 内核在下一个周期是否可以拥有总线。如果总线不可用，ARM7TDMI-S 处理器可以无限停止其时钟。确定存储器访问是否结束的唯一方法是检查 **TRANS[1:0]**和 **DBGACK** 的状态。当它们都为高电平时表示访问已经结束。

调试器通常使用 EmbeddedICE-RT 来控制调试，这样 **TRANS[1:0]**和 **DBGACK** 的状态可以通过读取 EmbeddedICE-RT 状态寄存器来确定。详见*调试状态寄存器*一节。

系统存储器的状态可以通过使用系统速度多路装载和调试速度多路存储反馈到调试主机。

对在 bit33 置位时可以执行那些指令有一些限制。下面是在置位 bit33 时的有效指令：

- 装载
- 保存
- 多路装载
- 多路保存

另见*退出调试状态*一节。

当 ARM7TDMI-S 处理器在一次系统速度访问之后返回调试状态时，扫描链 1 的 bit33 为高电平。bit33 的状态为调试器提供了这样一个信息，即为什么在第一次读取该扫描链时，内核进入调试状态。

5.17 退出调试状态

退出调试状态包含下列动作：

- 恢复 ARM7TDMI-S 处理器的内部状态
- 下一条指令执行分支指令
- 返回正常状态

在恢复内部状态之后，必须将一条分支指令装入流水线。详见*调试状态中的程序计数器*一节。

扫描链 1 的 bit33 强制 ARM7TDMI-S 处理器与 **CLKEN** 重新同步。调试指令序列的倒数第二条指令在 bit33 为高时被扫描。调试序列的最后一条指令是分支指令，它在 bit33 为低时被扫描。内核接着将分支指令装入流水线并且选择 TAP 控制器指定 RESTART 指令。

当状态机进入 RUN-TEST/IDLE 状态时，扫描链返回到系统模式。然后 ARM7TDMI-S 处理器恢复正常操作，从存储器中取指。这一延迟（直到状态机处于 RUN-TEST/IDLE 状态为止）使多处理器系统中的其它器件中的条件建立，但不立即生效。当状态机进入 RUN-TEST/IDLE 状态时，所有处理器同时恢复操作。

DBGACK 在 ARM7TDMI-S 处理器处于调试状态时通知系统的其余部分。该信息可用于禁止具有实时特性的外设，例如看门狗定时器。**DBGACK** 也可屏蔽由调试处理所引起的存储器访问。

例如，当 ARM7TDMI-S 处理器在一个断点后进入调试状态，指令流水线包含断点指令和其它两个被预取指的指令。在进入调试状态的入口，流水线被清洗。在退出调试状态时，流水线必须恢复之前的状态。

由于调试的处理，对存储器的访问多于正常情况，**DBGACK** 可禁止任何可能对存储器访问次数敏感的外设。例如，不管程序运行在调试或者无调试状态下，一个对存储器访问进行计数的外设都必须返回相同的值。图 5-11 所示为 ARM7TDMI-S 处理器在退出调试状态时的状态。

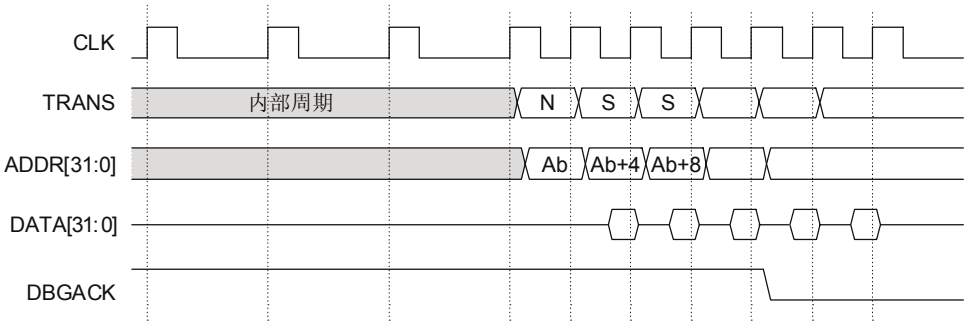

图 5-11 退出调试的时序

图 5-3 所示为在 **DBGACK** 变为高电平后的最后一个周期内对存储器的访问。此时周期计数器必须被禁止。图 5-11 所示为 **DBGACK** 变为低之后的周期中发生的周期计数器之前没有看到的第一个存储器访问。此时计数器被重新使能。

注：从调试状态中产生一个系统速度访问时，ARM7TDMI-S 处理器暂时离开调试状态以使 **DBGACK** 能够变为低电平。如果有外设对存储器访问次数敏感，必须使它们认为 ARM7TDMI-S 处理器仍然处于调试状态。可以编程 EmbeddedICE-RT 控制器寄存器使 **DBGACK** 强制为高来实现这一点。详见*调试状态寄*

*存器*一节。

5.18 调试中的程序计数器

调试器必须对 PC 进行跟踪，这样 ARM7TDMI-S 内核才能通过分支跳转到由于调试而中断的程序位置。程序流可以被下列事件中断：

- 断点
- 观察点
- 带有其它异常的观察点
- 调试请求
- 系统速度访问

5.18.1 断点

断点的调试状态入口占用了 PC 4 个地址或 16 个字节。调试状态下执行的每个指令占用 PC 1 个地址或 4 个字节。

通常在断点后退出调试状态的方法是撤消断点并通过分支跳转回之前发生断点的地址。

例如，如果 ARM7TDMI-S 处理器在一个给定地址处设定的断点进入调试状态，并且执行了两条调试速度的指令，那么必须产生一个-7 地址的分支指令（4 个调试入口，加上 2 个指令，再加 1 个最后的分支指令）。

下面的指令序列所示为扫描到扫描链 1 的数据，最高位在前。第一个数据位的值存入 DBGBREAK，指令数据进入扫描链 1 的其余部分：

```
0 E0802000      ;ADD   R2,R0,R0
1 E1826001      ;ORR   R6,R2,R1
0 EAFFFFF9      ;B -7 (2 的补码)
```

在 ARM7TDMI-S 处理器进入调试状态后，它在执行分支指令之前必须至少执行两条指令，即使都为 NOP 指令（MOV R0,R0）。对于小的分支，您可以将 PC 作为目的数执行减法来替换最后的分支指令（在上面的例子中为 SUB PC,PC,#28）。

5.18.2 观察点

在进入观察点后返回程序执行的处理与断点所采用的方法相同。

调试入口使 PC 增加了 4 个地址，每个指令增加 1 个地址。与断点的区别在于：产生观察点的指令已经执行，程序必须返回到下一条指令。

5.18.3 具有另外异常的观察点

如果对观察点的访问同时导致了一次数据中止，ARM7TDMI-S 处理器进入中止模式的调试状态。进入调试的入口被拖延到内核进入中止模式并从中止向量取指为止。

当一个中断或其它异常发生在观察点存储器访问时也遵循相似的过程。ARM7TDMI-S 处理器在异常模式中进入调试状态。调试器必须通过检查当前和先前的模式（在 CPSR 和 SPSR 中）以及 PC 的值来确定是否产生了异常。当发生了异常时，您可以选择在调试之前服务异常。

调试状态的入口在发生异常时使 PC 增加 3 个指令而不是 4 个，在退出调试状态时进行返回分支计算时必须考虑到这一点。例如，假设一个异常发生在观察点访问时，并且执行了 10 条指令来确定这一偶然事件，那么您可以使用下面的指令序列返回程序执行：

```
0   E1A00000;   MOV R0,R0
1   E1A00000;   MOV R0,R0
```

0 EAFFFFF0; B -16

上面的代码强制执行一个分支返回到异常向量处，使该位置的指令被重新取指和执行。

注：

在中止服务程序之后，导致异常和观察点的指令被重新取指和执行。这将再次触发观察点并使 ARM7TDMI-S 处理器重入调试状态。

5.18.4 调试请求

通过调试请求进入调试状态的入口与断点相似，但和断点不同的是，在最后一条指令已经执行完毕并且在退出调试状态时不能再重新对其取指。因此，您可以假设调试状态的入口使 PC 增加了 3 个地址，而调试状态下的每条指令增加 1 个地址。

例如，假设您调用了一个调试请求，并且决定直接返回程序的执行，那么您可以使用下面的代码序列：

```
0 E1A00000 ;MOV R0,R0
1 E1A00000 ;MOV R0,R0
0 EAFFFFFA ;B -6
```

该代码将 PC 恢复到下一条指令并重新启动程序执行。

5.18.5 系统速度访问

在调试状态下执行系统速度访问时，PC 的值加 3 个地址。系统速度指令访问存储器系统并有可能产生异常。如果异常发生在系统速度访问时，ARM7TDMI-S 处理器在返回调试状态之前进入异常模式。

此情形与一个出现异常的观察点相似，但问题更难解决，因为异常并不是由主程序中的指令所引起的，因此 PC 并不指向导致出现异常的指令。异常处理器通常观察 PC 来确定导致异常的指令和异常地址。在这种情况下，PC 的值无效。但由于调试器可以决定对哪个位置进行访问，因此可以对调试器执行写操作以帮助异常处理器修复存储器系统。

5.18.6 返回地址计算汇总

分支返回地址的计算如下：

- 对于正常的断点和观察点，分支为：

  - (4 + N + 3S)

- 对于通过调试请求或者带有异常的观察点入口，分支为：

  - (3 + N + 3S)

此处，N 为执行的调试速度指令（包括最后的分支指令），S 为执行的系统速度指令。

5.19 优先级和异常

当一个断点或调试请求产生时，正常的程序流被打断。因此，调试可以看作是另一种类型的异常。调试器与其它异常的交互作用详见*调试中的程序计数器*一节。该节包含下面的优先级：

- 带预取指中止的断点
- 中断
- 数据中止

5.19.1 带预取指中止的断点

当一个断点指令取指导致预取指中止是，执行中止处理并忽略断点。通常预取指中止发生在，例如对一个物理上并不存在的虚拟地址进行访问，因此返回的值无效。在该情况下，操作系统的正常动作是交换存储器页并返回之前无效的地址。这时当指令被取指并且断点有效时，ARM7TDMI-S 处理器进入调试状态。

因此预取指中止的优先级高于断点。

5.19.2 中断

当 ARM7TDMI-S 处理器进入调试状态时，中断被自动禁止。如果中断在进入调试状态之前的指令过程中被挂起，ARM7TDMI-S 处理器进入中断模式的调试状态。在调试状态的入口，调试器不能假设 ARM7TDMI-S 处理器处于用户程序所期望的模式中。ARM7TDMI-S 内核必须检查 PC、CPSR 和 SPSR 来确定发生异常的准确原因。

因此调试的优先级高于中断，但 ARM7TDMI-S 处理器不会记忆已经发生的中断。

5.19.3 数据中止

当数据中止发生在观测点访问时，ARM7TDMI-S 处理器进入中止模式的调试状态。因此观察点的优先级高于数据中止，但 ARM7TDMI-S 处理器会记忆所发生的中止。

5.20 观察点单元寄存器

两个观察点单元为观察点 0 和观察点 1，各自包含 3 对寄存器：

- 地址值和地址屏蔽
- 数据值和数据屏蔽
- 控制值和控制屏蔽

每个寄存器都独立可编程并具有唯一的地址。寄存器的功能和映射如表 5-1 所示。

5.20.1 编程和读取观察点寄存器

观察点寄存器由移位到 EmbeddedICE-RT 扫描链（扫描链 2）中的数据编程。扫描链为 38 为移位寄存器，包含：

- 32 位数据域
- 5 位地址域
- 1 个读/写位

这一设置见图 5-12。

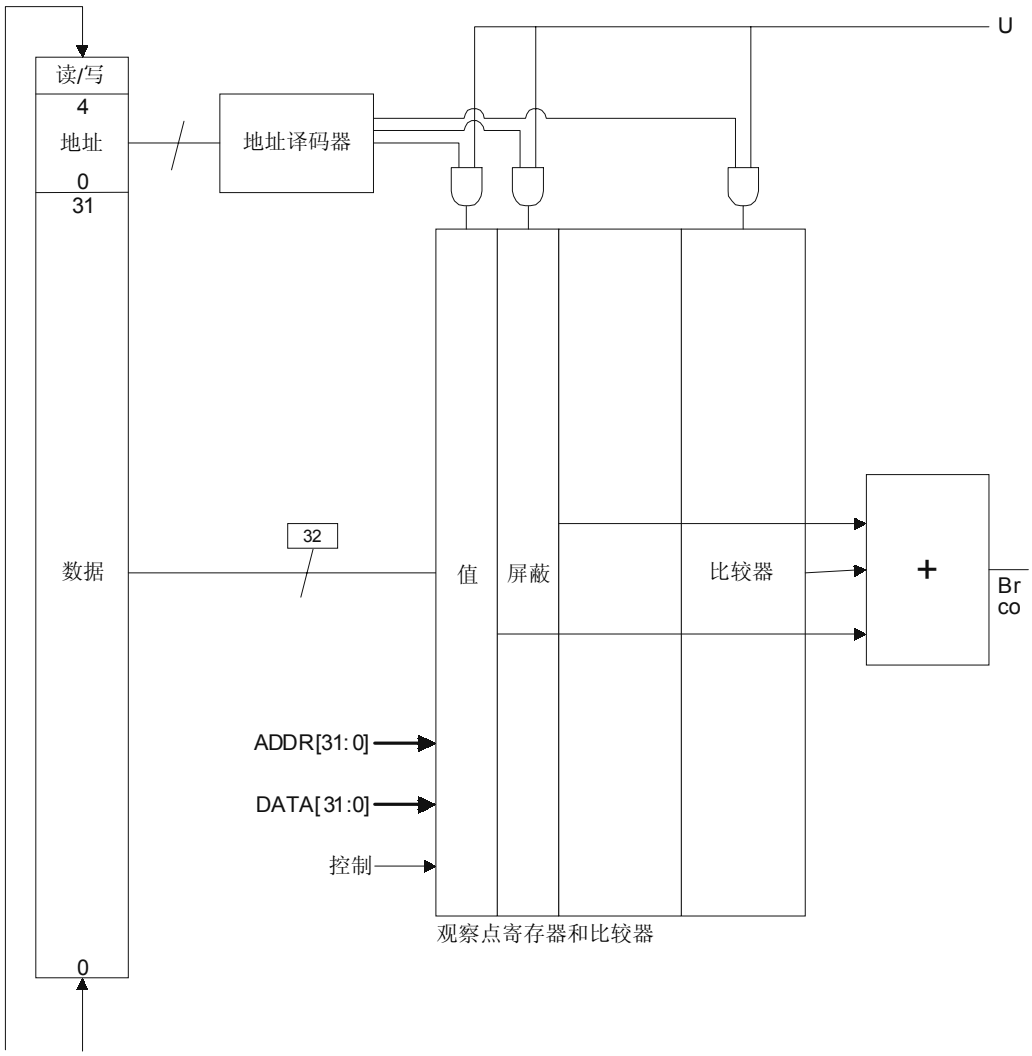

图 5-12 EmbeddedICE-RT 方框图

被写入的数据移入 32 位数据域，寄存器地址移入 5 位地址域，并设置读/写位。

寄存器通过将地址移入地址域并将 0 移入读/写位来读出。32 位数据域忽略。

寄存器地址见表 5-1。

注：

当 TAP 控制器进入 UPDATE-DR 状态时会发生实际的读或写。

5.20.2 使用数据和地址屏蔽寄存器

每个寄存器对中的值寄存器都有一个相同格式的屏蔽寄存器相对应。将屏蔽寄存器中的位置 1 使值寄存器中的对应位在比较时被忽略。

例如，当要求观察点处于特定的存储器位置时，但数据值无关，数据屏蔽寄存器可以编程为 0xffffffff（所有位都为 1）使整个数据总线域被忽略。

注：

屏蔽是 XNOR 屏蔽，而不是传统的 AND 屏蔽。当屏蔽位设置为 1 时，比较器在该位置总是相匹配，而不管值寄存器或输入的值为多少。

设置屏蔽位为 0 表示只有输入值与值寄存器的值相匹配时，比较器才会匹配。

5.20.3 控制寄存器

控制值和控制屏蔽寄存器的低 8 位映射相同，见图 5-13。

8	7	6	5	4	3	2	1	0
ENABLE	RANGE	CHAIN	DBGEXT	PROT[1]	PROT[0]	SIZE[1]	SIZE[0]	WRITE

图 5-13 观察点控制器值和屏蔽格式

控制值寄存器的 bit8 为 ENABLE 位，不能被屏蔽。

这些位具有下面的功能：

WRITE 与内核的写信号比较以检测总线活动的方向。**WRITE** 为 0 表示读周期，1 表示写周期。

SIZE[1:0] 与内核的 **SIZE[1:0]**信号比较以检测总线活动的规模。编码如下：

表 5-6 **SIZE[1:0]**信号编码

bit1	bit0	数据规格
0	0	字节
0	1	半字
1	0	字
1	1	保留

PROT[0] 用于检测当前周期是指令取指（**PROT[0]**=0）还是数据访问（**PROT[0]**=1）周期。

PROT[1] 与内核的非译码信号比较以区分用户模式（**PROT[1]**=0）还是非用户模式（**PROT[1]**=1）的访问。

DBGEXT[1:0] EmbeddedICE-RT 逻辑的外部输入，它根据某个外部条件使能观察点。

观察点 0 的 **DBGEXT** 输入标记为 **DBGEXT[0]**。

观察点 1 的 **DBGEXT** 输入标记为 **DBGEXT[1]**。

CHAIN 可以连接到其它观察点的链输出，用于实现例如下面形式的调试器请求：breakpoint on address YYY only when in process XXX。

在 ARM7TDMI-S 处理器 EmbeddedICE-RT 宏单元中，观察点 1 的 **CHAINOUT** 输出连接到观察点 0 的 **CHAIN** 输入。

CHIANOUT 输出源自一个寄存器。地址/控制域比较器驱动寄存器的写使能。寄存器的输入为比较器数据域的值。

CHIANOUT 寄存器在写入控制值寄存器或 **DBGnTRST** 为低时清零。

RANGE 在 ARM7TDMI-S 处理器 EmbeddedICE-RT 逻辑中，观察点 1 的 **RANGEOUT** 输出连接到观察点 0 的 **RANGE** 输入。该连接使两个观察点共同检测同时发生的条件，例如范围检测。

ENABLE] 当观察点发生匹配时，内部 **DBGRREAK** 信号只有在 **ENABLE** 位置位时才声明。该位只存在于值寄存器当中，不能被屏蔽。

对于控制值寄存器中的每个位[7:0]，在控制屏蔽寄存器中都有一个对应的位。这些位消除了特定信号的相关性。

5.21 编程断点

断点可以分为硬件断点和软件断点：

- 硬件断点通常监视地址值并可以在任何代码中设置，即使代码存在于 ROM 中或是自修改代码。详见*硬件断点*一节。
- 软件断点监视从任何地址取出的特定位格式。一个 EmbeddedICE-RT 观察点可以用于支持任意数目的软件断点。详见*软件断点*一节。软件断点只能在 RAM 中正常设置，因为特定的位格式选择使软件断点必须替换指令。

5.21.1 硬件断点

为了使一个观察点单元产生硬件断点（在指令取指时），必须执行以下动作：

1. 用发生断点的指令地址编程地址值寄存器。
2. 对于 ARM 状态的断点，将地址屏蔽寄存器的 bit[1:0]编程为 11。对于 Thumb 状态断点，将地址屏蔽寄存器的 bit[1:0]编程为 01。
3. 只有当您需要一个数据相关的断点时，才对数据值寄存器编程，也就是说必须匹配所取指的实际指令代码和地址。如果数据值无关，将数据屏蔽寄存器编程为 0xffffffff（所有位都为 1）。否则编程为 0x00000000。
4. 将控制值寄存器编程为 **PROT[0]**=0。
5. 将控制屏蔽寄存器编程为 **PROT[0]**=0。
6. 当您必须区别用户和非用户模式的指令取指时，将 **PROT[1]**和屏蔽位编程为适当的值。
7. 如果需要，以同样的方式编程 **DBGEXT**、**RANGE** 和 **CHAIN**。
8. 将所有未用的控制值编程为 1。

5.21.2 软件断点

为了使一个观察点单元产生软件断点（在一个特定位格式的指令取指时），必须执行以下动作：

1. 将地址屏蔽寄存器编程为 0xffffffff（所有位都为 1），这样可将地址忽略。
2. 使用特定的位格式编程数据值寄存器，该特定的位格式代表一个软件断点。
   如果您编程一个 Thumb 软件断点，在数据值寄存器的两半都编程为相同的 16 位格式。例如，如果位格式为 0xdfff，则编程为 0xdfffdfff。对 16 位指令取指时，EmbeddedICE-RT 只将数据总线有效的一半与数据值寄存器的内容相比较。采用这种方法，您可以用单个观察点寄存器同时捕获数据总线上半部分和下半部分的软件断点。
3. 将数据屏蔽寄存器编程为 0x00000000。
4. 将控制值寄存器编程为 **PROT[0]**=0。
5. 将控制屏蔽寄存器编程为 **PROT[0]**=0，其它位都编程为 1。
6. 当您必须区别用户和非用户模式的指令取指时，分别编程控制值和控制屏蔽寄存器中的 **PROT[1]**。
7. 如果需要，以同样的方式编程 **DBGEXT**、**RANGE** 和 **CHAIN**。
8. 将所有未用的控制值编程为 1。

注：

您不必编程地址值寄存器。

设置断点

要设置断点：

1. 读取指定地址的指令并将其保存。
2. 写特定的位格式，用于表示一个软件断点。

清除断点

要清除断点，将指令恢复到地址。

5.22 编程观察点

为了使一个观察点单元产生观察点（在数据访问时），必须执行以下动作：

1. 将要观察的数据访问的地址编程到观察点单元的地址值寄存器。

2. 将地址屏蔽寄存器编程为 0x00000000。
3. 只有当您需要一个数据相关的观察点时，才对数据值寄存器编程，也就是说必须匹配实际读出或写入的数据值和地址。如果数据值无关，将数据屏蔽寄存器编程为 0xffffffff（所有位都为 1）。否则编程为 0x00000000。
4. 将控制值寄存器编程为 **PROT[0]**=1，**WRITE**=0（读）或 **WRITE**=1（写），将 **SIZE[1:0]**编程为适当的数据类型。
5. 将控制屏蔽寄存器编程为 **PROT[0]**=0、**WRITE**=0、**SIZE[1:0]**＝0，其它位都编程为 1。当具有读和写操作或者数据类型将被分别观察时，可将 **WRITE** 和 **SIZE[1:0]**设置为 1
6. 当您必须区别用户和非用户模式的指令取指时，分别编程控制值和控制屏蔽寄存器中的 **PROT[1]**。
7. 如果需要，以同样的方式编程 **DBGEXT**、**RANGE** 和 **CHAIN**。

注：

上面是一个如果编程观察点寄存器以产生断点和观察点的例子。也可使用其它的方式编程寄存器。例如，可以通过设置一个或多个地址屏蔽位提供简单范围的断点。

5.23 中止状态寄存器

该 32 位读/写寄存器只有 bit0 可用。它用于确定一个中止异常入口是由断点、观察点还是真实的中止所引起的。格式如图 5-14 所示：

31：1	0
SBZ/RAZ	DbgAbt

图 5-14 调试中止状态寄存器

当 ARM7TDMI-S 内核将预取指或数据中止作为断点或观察点的结果时，该位置位。如果在特定指令或数据取指时，调试中止和外部中止信号声明，那么外部中止优先，而 DbgAbt 位不置位。DbgAbt 一旦置位，将一直保持到用户将其复位为止。该寄存器通过 MRC 和 MCR 指令进行访问。

5.24 调试控制寄存器

调试控制寄存器宽度为 6 位。对调试控制寄存器的写操作发生在对观察点单元寄存器进行写操作之时。而对调试控制寄存器的读操作则发生在对观察点单元寄存器进行读操作之时。详见*观察点单元寄存器*一节。

图 5-15 所示为调试控制寄存器中每一位的功能。

5	4	3	2	1	0
EmbeddedICE-RT 禁止	监视模式使能	SBZ/RAZ	**INTDIS**	**DBGRQ**	**DBGACK**

图 5-57 调试控制寄存器格式

调试控制寄存器位分配如表 5-7 所示。

表 5-7 调试控制寄存器位分配

位	功能
5	当观察点和断点寄存器被编程时，禁止 EmbeddedICE-RT 比较器输出。该位可通过 JTAG 接口读写。 下列情况下置位 bit5： • 编程断点或观察点寄存器 • 改变调试控制寄存器的 bit4。
4	当到达断点或观察点时，用于确定内核的状态： • 如果清零，当到达断点或观察点时，内核进入调试状态。 • 如果置位，当到达断点或观察点时，内核执行中止异常处理。 该位可通过 JTAG 读写。
3	该位必须清零。
2	用于禁止中断： • 如果置位，内核（IFEN）的中断使能信号被强制为低。IFEN 信号信号的功能见表 5-8。 • 如果清零，中断使能。
1	用于强制 DBGRQ 上的信号。
0	用于强制 DBGACK 上的信号。

5.24.1 禁止中断

IRQs 和 FIQs 在下列条件下禁止：

- 调试过程中（DBGACK 为高）
- INTDIS 位为高

IFEN 信号功能如表 5-8 所示。

表 5-8 中断信号控制

DBGACK	INTDIS	IFEN	中断
0	0	1	允许
1	x	0	禁止
x	1	0	禁止

5.24.2 强制 DBGRQ

图 5-17 显示了保存在调试控制寄存器 bit1 中的值被同步并在输入到处理器之前与 DBGRQ 相或。或门的输出为信号 DBGRQI。

调试控制寄存器 bit1 与 DBGRQI 之间的同步辅助实现多处理器环境。同步锁存只有在 TAP 控制器状态机处于 RUN-TEST-IDLE 状态时才打开。这使得系统中的所有处理器在运行时建立了一个进入调试的条件。当所有处理器都建立了条件时，它们可同时进入 RUN-TEST-IDLE 状态。

5.24.3 强制 DBGACK

在图 5-17 当中，内部信号 DBGACKI 的值与调试控制器的 bit0 相或产生了 ARM7TDMI-S 内核的外设可见的 DBGACK 外部值。这使得调试系统可以通知系统的其余部分内核仍然处于调试状态，即使在执行系统速度的访问（当内部 DBGACK 信号为低时）。

5.25 调试状态寄存器

调试状态寄存器宽度为 5 位。如果读/写位置位，对其的访问为写操作，如果读/写位清零，对其的访问

为读操作。调试状态寄存器的格式见图 5-16。

4	3	2	1	0
TBIT	TRANS[1]	IFEN	DBGRQ	DBGACK

图 5-16 调试状态寄存器的格式

该寄存器每一位的功能如下：

位	功能
4	使能读取 **TBIT**，使调试器能够确定处理器的状态并因此执行相关的指令。
3	使能读取内核的 **TRANS[1]**信号，这使调试器能够确定调试状态下的存储器访问是否结束。
2	使能读取内核中断使能信号 **IFEN** 的状态。
1:0	使能读取 DBGRQ 和 DBGACK 的同步值。

调试控制和状态寄存器的结构如图 5-17 所示。

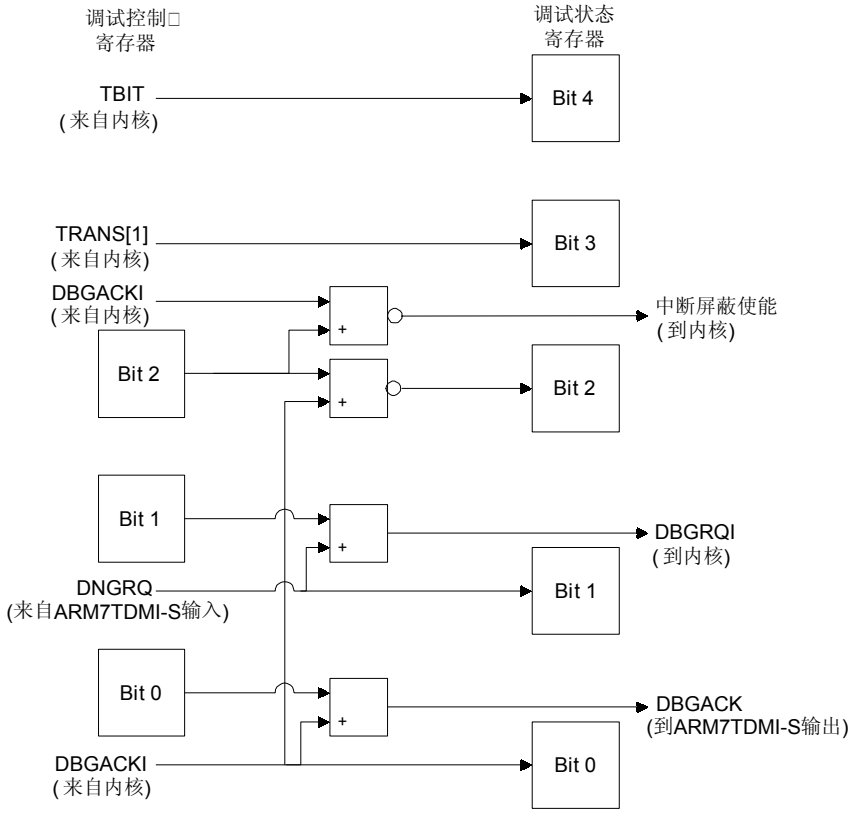

图 5-17 调试控制和状态寄存器结构

5.26 结合断点和观察点

使用 **CHAIN** 和 **RANGE** 输入可以使观察点单元 1 和 0 结合到一起。**CHAIN** 使观察点 0 只有在观察点 1 发生匹配的情况下才被触发。**RANGE** 则通过组合两个观察点的输出使能执行简单的范围检测。

5.26.1 断点和观察点结合实例

Av [31:0] Be the value in the address value register

Am [31:0] Be the value in the address mask register

A [31:0] Be the address bus from the ARM7TDMI-S processor

Dv [31:0] Be the value in the data value register

Dm [31:0]	Be the value in the data mask register
D [31:0]	Be the data bus from the ARM7TDMI-S processor
Cv [8:0]	Be the value in the control value register
Cm [7:0]	Be the value in the control mask register
C [9:0]	Be the combined control bus from the ARM7TDMI-S core, other watchpoint registers, and the **DBGEXT** signal.

CHAINOUT 信号

CHAINOUT 信号来自：

WHEN (({Av [31:0],Cv [4:0]}XNOR {A [31:0],C [4:0]})OR {Am [31:0],Cm [4:0]}==0xFFFFFFFF)

CHAINOUT=((({Dv [31:0],Cv [6:4]}XNOR {D [31:0],C [7:5]})OR {Dm [31:0],Cm [7:5]})==0x7FFFFFFF)

观察点寄存器 1 的 **CHAINOUT** 输出向观察点 0 提供 **CHAIN** 输出。该 **CHAIN** 输入使您可以使用配置相当复杂的断点和观察点。

注：

观察点 1 无 **CHAIN** 输入，而观察点 0 无 **CHAIN** 输出。

例如，调试器请求一个位于地址 YYY，并且在多处理系统中运行处理 XXX 时的断点。如果当前处理 ID 保存在存储器当中，那么可以使用观察点和断点的结合来实现上述功能。观察点地址指向一个已知的包含当前处理 ID 的存储器位置，观察点数据指向所请求的处理 ID 并且 **ENABLE** 位清零。

观察点的地址比较器输出用于驱动 **CHAINOUT** 锁存的写使能。锁存的输入是同一个观察点数据比较器的输出。锁存输出驱动断点比较器的 **CHAIN** 输入。地址 YYY 保存在断点寄存器当中，当 **CHAIN** 输入声明时，断点地址匹配并正确触发断点。

5.26.2 DBGRNG 信号

DBGRNG 信号来自：

DBGRNG=((({Av [31:0],Cv [4:0]}XNOR{A[31:0],C [4:0]}) OR {Am[31:0],Cm[4:0]})==0xFFFFFFFF) AND ((({Dv [31:0],Cv [7:5]}XNOR {D [31:0],C [7:5]}) OR Dm [31:0],Cm [7:5]))==0x7FFFFFFF)

观察点寄存器 1 的 **DBGRNG** 输出为观察点寄存器 0 提供 **RANGE** 输入。该 **RANGE** 输入将两个断点结合到一起构成范围断点。

可选择的范围限制为 2 的幂。例如，如果一个断点发生的地址在存储器的前 256 个字节内，但不在前 32 字节内，那么对观察点寄存器进行如下编程：

对于观察点 1：

1. 使用地址值 0x00000000 和地址屏蔽值 0x0000001f 编程观察点 1。
2. 清零 **ENABLE** 位。
3. 将所有其它观察点 1 寄存器按照正常的断点编程。前 32 字节内的地址使 RANGE 输出为高，但不会触发断点。

对于观察点 0：

1. 使用地址值 0x00000000 和地址屏蔽值 0x000000ff 编程观察点 1。
2. 置位 **ENABLE** 位。
3. 将 RANGE 位编程为匹配 0。
4. 将所有其它观察点 0 寄存器按照正常的断点编程。

如果观察点 0 匹配但观察点 1 不匹配（也就是说观察点 0 的 RANGE 输入为 0），断点被触发。

5.27 EmbeddedICE-RT 时序

EmbeddedICE-RT 在 **CLK** 的上升沿对 **DBGEXT[1]**和 **DBGEXT[0]**采样。

所需要的建立和保持时间详见第 8 章 *AC 参数*。

第 6 章 ETM 接口

这一章描述了 ARM7TDMI-S 处理器所提供的 ETM 接口。包含以下小节：

- 关于 ETM 接口
- 使能和禁止 ETM7 接口
- ETM7 与 ARM7TDMI-S 的连接
- 时钟和复位
- 调试请求接线

6.1 关于 ETM 接口

您可以将一个外部嵌入式跟踪宏单元（ETM）连接到 ARM7TDMI-S 处理器，这样就能够实现对正在执行的处理器进行代码的实时跟踪。

注：

如果在您的系统中有超过一个的 ARM 处理器，每个处理器都必须有一个专用的 ETM。

通常来说，需要使用小的或无胶合的逻辑将 ETM7 连接到 ARM7TDMI-S（Rev 4）。通过 JTAG 接口对 ETM 进行编程。该接口是 ARM TAP 控制器的扩展，被分配为扫描链 6。

6.2 使能和禁止 ETM7 接口

在 ARM 调试工具的控制下，ETM7 **PWRDOWN** 输出用于使能和禁止 ETM。当 **PWRDOWN** 为高时，表示 ETM 被禁止。这样您可以停止 **CLK** 输入并使其它 ETM 信号保持稳定。这样可在您不执行跟踪的时候降低功耗。

当 TAP 复位（**DBGnTRST**）发生时，**PWRDOWN** 被强制为高，直到 ETM7 控制寄存器被编程为止（详见*嵌入式跟踪宏单元规范*）。

PWRDOWN 在调试会话开始时自动清零。

6.3 ETM7 与 ARM7TDMI-S（Rev 4）的连接

ETM7 接口名称时 ARM7TDMI 与 ARM7TDMI-S 宏单元的混合。表 6-1 所示为 ARM7TDMI-S 处理器与 ETM7 之间必须的连接。

表 6-1 ETM7 和 ARM7TDMI-S (Rev 4)管脚连接

ETM7 信号名称	ARM7TDMI-S (Rev 4) 信号名称
A[31:0]	**ADDR[31:0]**
ABORT	**ABORT**
ARMTDO	**DBGTDO**
BIGEND	**CFGBIGEND**
CLK [a]	**CLK** [a]
CLKEN	**CLKEN**
CPA	**CPA**
CPB	**CPB**
DBGACK	**DBGACK**

DBGRQ b	**DBGRQ** b
nMREQ	**CPnMREQ**
SEQ	**CPSEQ**
MAS[1:0]	**SIZE[1:0]**
nCPI	**CPnI**
nEXEC	**DBGnEXEC**
nOPC	**CPnOPC**
nRESET	**nRESET**
nRW	**WRITE**
nTRST [a]	**DBGnTRST** [a]
PROCID[31:0] [c]	-
PROCIDWR [c]	-
RANGEOUT[0]	**DBGRNG[0]**
RANGEOUT[1]	**DBGRNG[1]**
RDATA[31:0]	**RDATA[31:0]**
TBIT	**CPTBIT**
TCK [a]	**CLK** [a]
TCKEN	**DBGTCKEN**
TDI	**DBGTDI**
TDO	**DBGTDO**
TMS	**DBGTMS**
WDATA[31:0]	**WDATA[31:0]**
INSTRVALID	**DBGINSTRVALID**

a. 见时钟和复位一节。

b. 见调试请求接线一节。

c. ARM7TDMI-S 处理器不提高 PROCID[31:0]或 PROCIDWR 信号。您必须将这些 ETM 输入连接到低电平。

6.4 时钟和复位

ARM7TDMI-S（Rev 4）处理器使用信号时钟 **CLK** 作为主系统时钟和 JTAG 时钟。您必须将处理器时钟连接到 ETM 的 **CLK** 和 **TCK**。您可以使用 **TCKEN** 来控制 JTAG 接口。

要在 ARM7TDMI-S 处理器热启动对其进行跟踪，应使用 TAP 复位（**nTRST** 连接到 **DBGnTRST**）来复位 ETM7 状态。

有关 ETM7 时钟和复位的详细信息请参阅 *ETM7 技术参考手册*。

6.5 调试请求接线

建议您将 ETM7 的 **DBGRQ** 输出连接到 ARM7TDMI-S 处理器的 **DBGRQ** 输入端。如果该输入已经被使用，那么可以和 **DBGRQ** 输入相或。详见 *ETM7 技术参考手册*。

第 7 章 指令周期时序

这一章讲述 ARM7TDMI-S 处理器的指令周期时序。包含下列小节：

- 关于指令周期时序
- 指令周期计数汇总
- 分支和带链接的 ARM 分支
- 带链接的 Thumb 分支
- 分支和交换
- 数据操作
- 乘法和乘法累加
- 装载寄存器
- 保存寄存器
- 装载多个寄存器
- 保存多个寄存器
- 数据交换
- 软件中断和异常入口
- 协处理器数据处理操作
- 装载协处理器寄存器（从存储器到协处理器）
- 保存协处理器寄存器（从协处理器到存储器）
- 协处理器寄存器转送（从协处理器传送到 ARM 寄存器）
- 协处理器寄存器转送（从 ARM 寄存器传送到协处理器）
- 未定义的指令和协处理器空缺
- 未执行的指令

7.1 关于指令周期时序

TANS[1:0]信号预告下个周期的类型。这些信号在它们被应用的周期的前一个周期中依次传递，并且以下面的表格中列出。

在该章的表格中，下面的信号在它们所应用的周期中被记录：

- Address 为 **ADDR[31:0]**
- Lock 为 **LOCK**
- Size 为 **SIZE[1:0]**
- Write 为 **WRITE**
- Prot1 和 Prot0 为 **PROT[1:0]**
- Tbit 为 **CPTBIT**

在大多数情况下，地址因为预取指指令而增加。地址的增量以指令的长度为单位：

- ARM 状态下为 4 字节
- Thumb 状态下为 2 字节

注：

字母 i 表示指令的长度。

Size 指示传输的宽度：

- w（字）代表 32 位数据访问和 ARM 操作码取指
- h（半字）代表 16 位数据访问或 Thumb 操作码取指

- b（字节）代表 8 位数据访问。

CPA 和 CPB 在被 ARM7TDMI-S 处理器采样时依次输入并显示。

传输的类型如表 7-1 所示。

表 7-1 传输类型

TRANS[1:0]	传输类型	描述
00	I 周期	下一个为内部（只寻址）周期
01	C 周期	下个周期进行协处理器传输
10	N 周期	存储器访问的下一个地址不连续
11	S 周期	存储器访问的下一个地址连续

注：

该章中所有周期计数都假设为零等待状态存储器访问。在一个使用 CLKEN 增加等待状态的系统中，您必须相应地调整周期计数值。

7.2 指令周期计数值汇总

在 ARM7TDMI-S 内核的流水线结构中，当一条指令被取指时，前一条指令被译码，再前一条指令正在执行。表 7-2 所示为一个指令在到达执行阶段时所需要的周期数。

您可以根据表 7-2 中的数字计算一个程序的周期数。这些数字是指令执行所需要的周期数，未执行的指令占用一个周期。在表 7-2 中：

n 传输的字的数目

m 如果乘法器操作数的 bits[32:8]为全零或 1，m 等于 1
如果乘法器操作数的 bits[32:16]为全零或 1，m 等于 2
如果乘法器操作数的 bits[32:24]为全零或 1，m 等于 3
其它情况 m 等于 4

b 协处理器忙-等待循环（可以是零个或多个循环）所花费的周期数

当条件不满足时，所有指令都占用一个 S 周期。

表 7-2 指令周期计数值

指令	限制条件	周期计数值
任何未执行的指令	条件代码失效	+ S
数据处理	单周期	+ S
数据处理	寄存器相关移位	+ I + S
数据处理	R15 为目标	+ N + 2S
数据处理	R15, 寄存器相关移位	+ I + N + 2S
MUL	-	+ (m)I + 2S
MLA	-	+ I + (m)I + S
MULL	-	+ (m)I + I + S
MLAL	-	+ I + (m)I + I + S
B, BL	-	+ N + 2S
LDR	非 R15 目标	+ N + I + S
LDR	R15 为目标	+ N + I + N + 2S
STR	-	+ N + N
SWP	-	+ N + N + I + S
LDM	非 R15 目标	+ N + (n-1)S + I + S

LDM	R15 为目标	+ N + (n-1)S + I + N + 2S
STM	-	+ N + (n-1)S + I + N
MSR, MRS	-	+ S
SWI, trap	-	+ N + 2S
CDP	-	+ (b)I + S
MCR	-	+ (b)I + C + N
MRC	-	+ (b)I + C + I + S
LDC, STC	-	+ (b)I + N + S + N

周期类型 N, S, I 和 C 都在表 7-1 中定义。

7.3 分支和带连接的 ARM 分支

任何 ARM 或 Thumb 分支和一个带链接的 ARM 分支指令都占用 3 个周期：

1. 在第 1 个周期内，分支指令在执行当前 PC 的预取指时计算分支的目标。预取指在任何情况下都会完成，因为在到达决定执行分支的时间时，已经无法阻止预取指的执行。
2. 在第 2 个周期内，ARM7TDMI-S 内核执行分支目标的预取指。如果链接位置位，则返回地址保存在 r14 中。
3. 在第 3 个周期内，ARM7TDMI-S 内核执行目标+ i 的取指，重新填充流水线。当执行带链接的分支时，r14 被修改（减去 4）并执行 MOV PC,R14。此修改确保了类型 STM..{R14} LDM..{PC}的子程序正确工作。

表 7-3 所示为周期的时序，此处：

pc 分支指令的地址

pc’ ARM7TDMI-S 内核计算得到的地址

(pc’) 地址的内容

表 7-3 分支指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+2i	w/h	0	(pc+2i)	N 周期	0
2	pc’	w’/h’	0	(pc’)	S 周期	0
3	pc’+i	w’/h’	0	(pc’+i)	S 周期	0
	pc’+2i	w’/h’	-	-	-	-

注：

该数据仅用于 ARM 和 Thumb 状态下的分支以及 ARM 状态下的带链接分支。

7.4 带链接的 Thumb 分支

带链接（BL）的 Thumb 分支操作包括两条连续的 Thumb 指令并占用 4 个周期：

1. 第 1 条指令执行简单的数据操作。它占用单个周期，使 PC 增加到偏移地址的上半部分并将结果保存到 r14（LR）。
2. 第 2 条指令与 ARM BL 指令相似，它占用 3 个周期：
   - 在第 1 个周期中，ARM7TDMI-S 内核在执行当前 PC 的预取指时计算最终的分支目标地址。
   - 在第 2 个周期中，ARM7TDMI-S 对分支目标地址执行取指。返回的地址保存在 r14。
   - 在第 3 个周期中，ARM7TDMI-S 对目标地址+2 执行取指来重新填充指令流水线并修改 r14（减去 2）并简单返回 MOV PC,R14。此修改确保了类型 PUSH {..,LR}; POP {..,PC}的子程序正确工作。

表 7-4 所示为完整操作的周期时序。

表 7-4 Thumb 带链接的长分支

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+4	h	0	(pc+4)	S 周期	0
2	pc+6	h	0	(pc+6)	N 周期	0
3	pc’	h	0	(pc’)	S 周期	0
4	pc’+2	h	0	(pc’+2)	S 周期	0
	pc’+4	-	-	-	-	-

注：

PC 是操作的第一条指令的地址。

Thumb BL 指令在 *ARM 体系结构参考手册*中详细讲述。

7.5 分支和交换

一个分支和交换（BX）操作占用 3 个周期，它与分支相似：

1. 在第 1 个周期中，ARM7TDMI-S 内核在当前 PC 执行预取指时提取分支目标地址并从寄存器源提取新的内核状态，预取指在任何情况下都会完成，因为在到达决定执行分支的时间时，已经无法阻止预取指的执行。
2. 在第 2 个周期中，ARM7TDMI-S 内核根据所选择的状态，使用新的指令宽度对分支目标执行取指。
3. 在第 3 个周期中，ARM7TDMI-S 内核根据新的指定状态，对目标地址+2 或+4 进行取指，以重新填充流水线。

表 7-5 分支和交换指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+2i	w/h	0	(pc+2i)	N 周期	t
2	pc’	w’/h’	0	(pc’)	S 周期	t’
3	pc’+i	w’/h’	0	(pc’+i)	S 周期	t’
	pc’+2i	-	-	-	-	-

注：

i 和 i'分别代表 BX 之前和之后的指令宽度。

在 ARM 状态下，规格为 2 而在 Thumb 状态下规格为 1。当从 Thumb 状态变为 ARM 状态时，i 等于 1，i'等于 2。

t 和 t’分别代表 BX 之前和之后的 T 位状态。在 ARM 状态下，T 位为 0，而在 Thumb 状态下 T 位为 1。当从 ARM 状态变为 Thumb 状态时，t 等于 0，t’等于 1。

7.6 数据操作

数据操作在单个数据路径周期内执行，但当移位由寄存器的内容决定时例外。ARM7TDMI-S 内核将第 1 个寄存器读取到 A 总线，并将第 2 个寄存器或立即地址读取到 B 总线。

ALU 根据指令所指定的操作将 A 总线和移位的 B 总线资源结合。ARM7TDMI-S 内核将结果写入目标寄存器（比较和测试不产生结果，只影响 ALU 状态标志）。

指令预取指与数据操作同时发生，并且 PC 增加。

当寄存器指定移位的宽度时，在数据操作将寄存器的低 8 位复制到桶形寄存器锁存之前产生一个额外的数据周期。指令预取指在这个周期中发生。操作周期为内部周期（不需要存储器）。由于地址在这两个周期保持稳定，存储器管理器可以将这个内部周期与下一个连续访问合并。

PC 可以是一个或多个寄存器操作数。当 PC 为目标地址时，外部总线的活动会受到影响。当 ARM7TDMI-S 内核将结果写入 PC 时，指令流水线的内容无效。ARM7TDMI-S 内核将 ALU 中的内容，而不是地址增加器作为下一条指令预取指的地址。ARM7TDMI-S 处理器在任何指令执行之前重新填充流水线。在这段时间内，异常被关闭。

PSR 转移操作与数据操作具有相同的时序特性，区别在于 PC 从不作为源或者目标寄存器。

表 7-6 数据操作指令的周期

周期		地址	规格	写	数据	TRANS[1:0]	Prot0
正常	1	pc+2i	w/h	0	(pc+2i)	S 周期	0
		pc+3i	-	-	-	-	-
dest=pc	1	pc+2i	w/h	0	(pc'+i)	N 周期	0
	2	pc'	w/h	0	(pc')	S 周期	0
	3	pc'+i	w/h	0	(pc'+i)	S 周期	0
		pc'+2i	-	-	-	-	-
移位(Rs)	1	pc+2i	w/h	0	(pc+2i)	I 周期	0
	2	pc+3i	w/h	0		S 周期	1
		pc+3i	-	-	-	-	-
移位(Rs)	1	pc+8	w	0	(pc+8)	N 周期	0
dest=pc	2	pc+12	w	0	-	S 周期	1
	3	pc'	w	0	(pc')	S 周期	0
	4	pc'+4	w	0	(pc'+4)	S 周期	0
		pc'+8	-	-	-	-	-

7.7 乘法和乘法累加

乘法指令使用特殊的硬件实现整数乘法。所有周期除第一个外都为内部周期。

周期时序见表 7-7 到 7-10，其中 m 为乘法算法所要求的周期数（见指令周期计数值汇总）。

表 7-7 乘法指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+2i	w/h	0	(pc+2i)	I 周期	0
2	pc+3i	w/h	0	-	I 周期	1
•	pc+3i	w/h	0	-	I 周期	1
m	pc+3i	w/h	0	-	I 周期	1
m+1	pc+3i	w/h	0	-	S 周期	1
	pc+3i	-	-	-	-	-

表 7-8 乘法-累加指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+2i	w/h	0	(pc+2i)	I 周期	0
2	pc+3i	w/h	0	-	I 周期	1
•	pc+3i	w/h	0	-	I 周期	1
m	pc+3i	w/h	0	-	I 周期	1
m+1	pc+3i	w/h	0	-	I 周期	1
m+2	pc+3i	w/h	0	-	S 周期	1
	pc+3i	-	-	-	-	-

表 7-9 乘法长指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+8	w	0	(pc+8)	I 周期	0
2	pc+12	w	0	-	I 周期	1
•	pc+12	w	0	-	I 周期	1
m	pc+12	w	0	-	I 周期	1
m+1	pc+12	w	0	-	I 周期	1
m+2	pc+12	w	0	-	S 周期	1
	pc+12	-	-	-	-	-

表 7-10 乘法-累加长指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+8	w	0	(pc+8)	I 周期	0
2	pc+12	w	0	-	I 周期	1
•	pc+12	w	0	-	I 周期	1
m	pc+12	w	0	-	I 周期	1
m+1	pc+12	w	0	-	I 周期	1
m+2	pc+12	w	0	-	I 周期	1
m+3	pc+12	w	0	-	S 周期	1
	pc+12	-	-	-	-	-

注：

乘法-累加长指令只能用于 ARM 状态。

7.8 装载寄存器

装载寄存器指令占用的周期数可变：

1. 在第 1 个周期中，ARM7TDMI-S 处理器计算要装载的地址。
2. 在第 2 个周期中，ARM7TDMI-S 处理器从存储器中取数据并执行基址寄存器修改（如果需要）。
3. 在第 3 个周期中，ARM7TDMI-S 处理器将数据转移到目的寄存器（不使用外部存储器）。通常，ARM7TDMI-S 内核将该周期与预取指合并以构成一个存储器 N 周期。

装载寄存器周期时序见表 7-11，其中：

b,h,w 表 5-6 中所定义的字节、半字和字

s 代表当前值取决于超级用户模式。

u 当指令中指定了强制译码位时为 0，其它任何时候都为 s。

表 7-11 装载寄存器指令周期操作

周期		地址	规格	写	数据	TRANS[1:0]	Prot0	Prot1
正常	1	pc+2i	w/h	0	(pc+2i)	N 周期	0	s
	2	pc’	w/h/b	0	(pc’)	I 周期	1	u/s
	3	pc+3i	w/h	0	-	S 周期	1	s
		pc+3i	-	-	-	-	-	-
dest=pc	1	pc+8	w	0	(pc+8)	N 周期	0	s
	2	da	w/h/b	0	pc’	I 周期	1	u/s
	3	pc+12	w	0	-	N 周期	1	s
	4	pc’	w	0	(pc’)	S 周期	0	s
	5	pc’+4	w	0	(pc’+4)	S 周期	0	s
		pc’+8	-	-	-	-	-	-

基址或目标地址（或两者同时）可以是 PC。当 PC 被指令所影响时，预取指顺序发生改变。如果数据取指中止，ARM7TDMI-S 处理器防止对目标寄存器寄存器进行修改。

7.9 保存寄存器

保存寄存器需要两个周期：

1. 在第 1 个周期中，ARM7TDMI-S 内核计算所要保存的地址。
2. 在第 2 个周期中，ARM7TDMI-S 内核执行基址修改并将数据写入存储器（如果需要）。

保存寄存器周期时序见表 7-12，其中：

s 代表当前值与模式有关

t 当指令（STRT）中指定了 T 位时为 0，其它任何时候都为 c。

表 7-12 保存寄存器指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0	Prot1
1	pc+2i	w/h	0	(pc+2i)	N 周期	0	s
2	da	w/h/b	1	Rd	N 周期	1	t
	pc+3i	-	-	-	-	-	-

7.10 装载多个寄存器

装载多个寄存器（LDM）占用 4 个周期：

1. 在第 1 个周期中，ARM7TDMI-S 内核在执行预取指时计算要转移的第 1 个字的地址。
2. 在第 2 个周期中，ARM7TDMI-S 内核取第 1 个字并执行基址修改。
3. 在第 3 个周期中，ARM7TDMI-S 内核将第 1 个字移到相应的的目标寄存器并从存储器中取第 2 个字。ARM7TDMI-S 内部锁存修改的基址，以备在出现中止时之需。第 3 个周期连续的读取直到访问完最后一个数据字。
4. 在第 4 个和最后（内部）一个周期中，ARM7TDMI-S 内核将最后一个字移入目标寄存器。最后一个周期可以与下一个指令预取指合并构成一个存储器 N 周期。

当发生中止时，指令继续执行直到完成。ARM7TDMI-S 内核在中止后禁止所有寄存器的写操作。ARM7TDMI-S 内核改变最后一个周期以恢复修改的基址寄存器（在中止发生之前的装载可能被覆盖）。

当 PC 位于要装载的寄存器列表中时，ARM7TDMI-S 内核任务当前指令流水线无效。PC 总是最后一个装载的寄存器，因此在任意点发生的中止不会使 PC 被改写。

注：

在 Thumb 状态下，LDM 的目标为 PC 时不能被执行。但 POP {Rlist,PC}指令等效于目标为 PC 的 LDM 指令。

表 7-13 LDM 指令周期时序操作

周期		地址	规格	写	数据	TRANS[1:0]	Prot0
1 个寄存器	1	pc+2i	w/h	0	(pc+2i)	N 周期	0
	2	da	w	0	da	I 周期	1
	3	pc+3i	w/h	0	-	S 周期	1
		pc+3i	-	-	-	-	-
1 个寄存器	1	pc+2i	w/h	0	(pc'+2i)	N 周期	0
dest=pc	2	da	w	0	pc'	I 周期	1
	3	pc+3i	w/h	0	-	N 周期	1
	4	pc'	w/h	0	(pc')	S 周期	0
	5	pc'+i	w/h	0	(pc'+i)	S 周期	0
		pc'+2i	-	-	-	-	-
n 个寄存器	1	pc+2i	w/h	0	(pc+2i)	N 周期	0
（n>1）	2	da	w	0	da	S 周期	1
	•	da++	w	0	(da++)	S 周期	1
	n	da++	w	0	(da++)	S 周期	1
	n+1	da++	w	0	(da++)	I 周期	1
	n+2	pc+3i	w/h	0	-	S 周期	1
		pc+3i	-	-	-	-	-
n 个寄存器	1	pc+2i	w/h	0	(pc+2i)	N 周期	0
（n>1）	2	da	w	0	da	S 周期	1
incl pc	•	da++	w	0	(da++)	S 周期	1
	n	da++	w	0	(da++)	S 周期	1
	n+1	da++	w	0	pc'	I 周期	1
	n+2	pc+3i	w/h	0	-	N 周期	1
	n+3	pc'	w/h	0	(pc')	S 周期	0
	n+4	pc'+i	w/h	0	(pc'+i)	S 周期	0
		pc'+2i	-	-	-	-	-

7.11 保存多个寄存器

装载多个寄存器（LDM）占用 4 个周期：

1. 在第 1 个周期中，ARM7TDMI-S 内核计算要保存的第 1 个字的地址。
2. 在第 2 个周期中，ARM7TDMI-S 内核执行基址修改并将数据写入存储器。

由于没有寄存器的改写，因此可以直接重新启动。

STM 周期时序见表 7-14。

表 7-14 保存多个寄存器指令周期操作

周期		地址	规格	写	数据	TRANS[1:0]	Prot0
1 个寄存器	1	pc+2i	w/h	0	(pc+2i)	N 周期	0
	2	da	w	1	R	N 周期	1
		pc+3i	-	-	-	-	-
n 个寄存器	1	pc+8	w/h	0	(pc+2i)	N 周期	0
（n>1）	2	da	w	1	R	S 周期	1
	•	da++	w	1	R’	S 周期	1
	n	da++	w	1	R”	S 周期	1
	n+1	da++	w	1	R”’	N 周期	1
		pc+12	-	--	-	-	-

7.12 数据交换

数据交换与装载与保存寄存器指令相似，但交换发生在第 2 个和 3 个周期。数据在第 2 个周期中从外部存储器中取出，并在第 3 个周期将源寄存器的那如写入外部存储器。在第 4 个周期内，第 2 个周期读出的数据写入目标寄存器。

数据交换可以字节为单位，也可以字为单位。

ARM7TDMI-S 内核在读或写周期中有可能中止交换操作。交换操作（读或写）不影响目标寄存器。

数据交换周期时序见表 7-15，其中 b 和 w 分别代表字节和字。

表 7-15 数据交换指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0	Lock
1	pc+8	w	0	(pc+8)	N 周期	0	0
2	Rn	w/b	0	(Rn)	N 周期	1	1
3	Rn	w/b	1	Rm	I 周期	1	1
4	pc+12	w	0	-	S 周期	1	0
	pc+12	-	-	-	-	-	-

注：

在 Thumb 状态下，不能执行数据交换。

ARM7TDMI-S 处理器的 **LOCK** 输出在装载和保存数据周期都被驱动为高电平，用来告知存储器控制器这是一个很小的操作。

7.13 软件中断和异常入口

异常和软件中断（SWI）强制 PC 为一个指定的值，并从该地址开始重新填充指令流水线：

1. 在第 1 个周期中，ARM7TDMI-S 内核建立一个强制地址，并有可能发生模式改变。ARM7TDMI-S 内核将返回地址存入 r14 并将 CPSR 复制到 SPSR_svc。
2. 在第 2 个周期中，ARM7TDMI-S 内核修改返回地址以实现返回。
3. 第 3 个周期只有在完成流水线的重新填充时才需要。

SWI 周期时序见表 7-16，其中：

s 代表当前值与超级用户模式有关

t 代表当前 Thumb 状态值

pc 对于软件中断，为 SWI 指令的地址；对于异常来说，则是进入异常之前所执行的最后一条指令

的地址；对于预取指中止，为中止指令的地址；对于数据中止，PC 值为试图执行被中止数据传输的指令的下一条指令的地址。

Xn 为适当的捕获地址

表 7-16 软件中断指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0	Prot1	模式	Tbit
1	pc+2i	w/h	0	(pc+8)	N 周期	0	s	旧模式	t
2	Xn	w’	0	(Xn)	S 周期	0	1	异常模式	0
3	Xn+4	w’	0	(Xn+4)	S 周期	0	1	异常模式	0
	Xn+8								

7.14 协处理器数据处理操作

协处理器数据处理（CDP）操作是 ARM7TDMI-S 内核发出的要求协处理器执行某个动作的请求。不需要立即完成动作，但协处理器必须在将 CPB 驱动为低之前完成。

如果协处理器不能执行所请求的任务，它将 **CPA** 和 **CPB** 保持为高。但协处理器能够执行任务，但不能立即执行时，协处理器将 **CPA** 拉低，但保持 **CPB** 为高。ARM7TDMI-S 处于忙-等待状态，直到 **CPB** 变为低电平为止。但是一个中断可能导致 ARM7TDMI-S 内核放弃忙-等待协处理器指令。

协处理器数据操作周期时序见表 7-17。

表 7-17 协处理器数据操作资料周期操作

周期		地址	写	规格	数据	TRANS[1:0]	Prot0	CPnI	CPA	CPB
就绪	1	pc+8	0	w	(pc+8)	N 周期	0	0	0	0
		pc+12								
未就绪	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
	2	pc+8	0	w	-	I 周期	1	0	0	1
	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	N 周期	1	0	0	0
		pc+12								

7.15 装载协处理器寄存器（从存储器到协处理器）

装载协处理器（LDC）操作将一个或多个数据从存储器传送到协处理器寄存器。

协处理器只有在它准备接收数据时才提交传输。**WRITE** 线在传输周期中驱动为低电平。当 **CPB** 变低时，ARM7TDMI-S 内核产生地址并期待协处理器在随后的周期中接收数据。协处理器负责确定传输的字数。中断可导致 ARM7TDMI-S 内核放弃忙-等待的协处理器指令。

第 1 个周期（和任何的忙-等待周期）产生传输地址。第 2 个周期执行基地址的写回。协处理器通过将 **CPA** 和 **CPB** 都驱动为高来指示最后一个传输周期。

装载协处理器寄存器周期时序见表 7-18。

表 7-18 装载协处理器寄存器指令周期操作

周期		地址	写	规格	数据	TRANS[1:0]	Prot0	CPnI	CPA	CPB
1 个寄存器	1	pc+8	0	w	(pc+8)	N 周期	0	0	0	0
就绪	2	da		w	(da)	N 周期	1	1	1	1
		pc+12								
1 个寄存器	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
未就绪	2	pc+8	0	w	-	I 周期	1	0	0	1
	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	N 周期	1	0	0	0
	n+1	da	0	w	(da)	N 周期	1	1	1	1
		pc+12								
m 个寄存器	1	pc+8	0	w	(pc+8)	N 周期	0	0	0	0
(m>1)	2	da	0	w	(da)	S 周期	1	1	0	0
就绪	•	da++	0	w	(da++)	S 周期	1	1	0	0
	m	da++	0	w	(da++)	S 周期	1	1	0	0
	m+1	da++	0	w	(da++)	N 周期	1	1	1	1
		pc+12								
m 个寄存器	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
(m>1)	2	pc+8	0	w	-	I 周期	1	0	0	1
未就绪	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	N 周期	1	0	0	0
	n+1	da	0	w	(da)	S 周期	1	1	0	0
	•	da++	0		(da++)	S 周期	1	1	0	0
	n+m	da++	0	w	(da++)	S 周期	1	1	0	0
	n+m+1	da++	0	w	(da++)	N 周期	1	1	1	1
		pc+12								

注：

在 ARM 状态下无法执行协处理器操作。

7.16 保存协处理器寄存器（从协处理器到存储器）

保存协处理器（STC）操作将一个或多个字数据从协处理器传送到存储器。

协处理器只有在它准备写数据时才提交传输。**WRITE** 线在传输周期中驱动为低电平。当 **CPB** 变低时，ARM7TDMI-S 内核产生地址并期待协处理器在随后的周期中写数据。协处理器负责确定传输的字数。中断可导致 ARM7TDMI-S 内核放弃忙-等待的协处理器指令。

第 1 个周期（和任何的忙-等待周期）产生传输地址。第 2 个周期执行基地址的写回。协处理器通过将 **CPA** 和 **CPB** 都驱动为高来指示最后一个传输周期。

保存协处理器寄存器周期时序见表 7-19。

表 7-19 保存协处理器寄存器指令周期操作

周期		地址	写	规格	数据	TRANS[1:0]	Prot0	CPnI	CPA	CPB
1 个寄存器	1	pc+8	0	w	(pc+8)	N 周期	0	0	0	0
就绪	2	da	1	w	CPdata	N 周期	1	1	1	1
		pc+12								
1 个寄存器	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
未就绪	2	pc+8	0	w	-	I 周期	1	0	0	1
	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	N 周期	1	0	0	0
	n+1	da	1	w	CPdata	N 周期	1	1	1	1
		pc+12								
m 个寄存器	1	pc+8	0	w	(pc+8)	N 周期	0	0	0	0
(m>1)	2	da	1	w	CPdata	S 周期	1	1	0	0
就绪	•	da++	1	w	CPdata’	S 周期	1	1	0	0
	m	da++	1	w	CPdata”	S 周期	1	1	0	0
	m+1	da++	1	w	CPdata”’	N 周期	1	1	1	1
		pc+12								
m 个寄存器	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
(m>1)	2	pc+8	0	w	-	I 周期	1	0	0	1
未就绪	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	N 周期	1	0	0	0
	n+1	da	1	w	CPdata	S 周期	1	1	0	0
	•	da++	1		CPdata	S 周期	1	1	0	0
	n+m	da++	1	w	CPdata	S 周期	1	1	0	0
	n+m+1	da++	1	w	CPdata	N 周期	1	1	1	1
		pc+12								

注：

在 ARM 状态下无法执行协处理器操作。

7.17 协处理器寄存器传输（从协处理器传送到 ARM 寄存器）

MRC 操作将单个协处理器寄存器值读入指定的 ARM 寄存器。

数据在第 2 个周期传送并在第 3 个周期写入 ARM 寄存器。

如果协处理器信号因为 **CPB** 置位而处于忙-等待，一个中断可导致 ARM7TDMI-S 内核放弃协处理器指令。

在所有 ARM7TDMI-S 寄存器装载指令执行时，ARM7TDMI-S 内核会将第 3 个周期与下一个预取指周期合并为一个 I-S 周期。

MRC 指令时序见表 7-20 所示。

表 7-20 协处理器寄存器传送（MRC）

周期		地址	写	规格	数据	TRANS[1:0]	Prot0	CPnI	CPA	CPB
就绪	1	pc+8	0	w	(pc+8)	C 周期	0	0	0	0
	2	pc+12	0	w	CPdata	I 周期	1	1	1	1
	3	pc+12	0	w	-	S 周期	1	1	-	-
		pc+12								
未就绪	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
	2	pc+8	0	w	-	I 周期	1	0	0	1
	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	C 周期	1	0	0	0
	n+1	pc+12	0	w	CPdata	I 周期	1	1	1	1
	n+2	pc+12	0	w	-	S 周期	1	1	-	-
		pc+12								

注：

在 Thumb 状态下无法执行该操作。

7.18 协处理器寄存器传送（从 ARM 寄存器传送到协处理器）

MCR 操作将单个 ARM 寄存器值传送到指定的协处理器寄存器。

数据在第 2 个周期传送。如果协处理器信号因为 **CPB** 置位而处于忙-等待，一个中断可导致 ARM7TDMI-S 内核放弃协处理器指令。

MCR 指令时序见表 7-21 所示。

表 7-21 协处理器寄存器传送（MCR）

周期		地址	写	规格	数据	TRANS[1:0]	Prot0	CPnI	CPA	CPB
就绪	1	pc+8	0	w	(pc+8)	C 周期	0	0	0	0
	2	pc+12	1	w	Rd	N 周期	1	1	1	1
		pc+12								
未就绪	1	pc+8	0	w	(pc+8)	I 周期	0	0	0	1
	2	pc+8	0	w	-	I 周期	1	0	0	1
	•	pc+8	0	w	-	I 周期	1	0	0	1
	n	pc+8	0	w	-	C 周期	1	0	0	0
	n+1	pc+12	1	w	Rd	N 周期	1	1	1	1
		pc+12								

注：

该协处理器操作只能在 ARM 状态下执行。

7.19 未定义指令和协处理器缺席

如果执行一个未定义的指令，则产生未定义指令陷阱。关于未定义指令的定义，请参阅 *ARM 体系结构参考手册*。

如果没有协处理器能够接受协处理器指令，那么将该指令和一个未定义指令一样对待。这样在没有硬件协处理器时可使用软件仿真协处理器指令。

注：

默认情况下，CPA 和 CPB 必须驱动为高，除非协处理器指令已经由协处理器执行。

未定义指令的周期时序见表 7-22。

表 7-22 未定义指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0	CPnI	CPA 和 CPB	Prot1	模式	Tbit
1	pc+2i	w/h	0	(pc+2i)	I 周期	0	0	1	s	旧模式	t
2	pc+2i	w/h	0	-	N 周期	0	1	1	s	旧模式	t
3	Xn	w’	0	(Xn)	S 周期	0	1	1	1	00100	0
4	Xn+4	w’	0	(Xn+4)	S 周期	0	1	1	1	00100	0
	Xn+8										

此处：

s 表示当前的值与模式有关

t 表示当前的值与状态有关

注：

该协处理器操作只能在 ARM 状态下执行。

7.20 未执行的指令

当任何指令的条件代码没有得到满足时，该指令不会被执行。任何未执行的指令都占用一个周期。

未执行指令的时序见表 7-23。

表 7-23 未执行的指令周期操作

周期	地址	规格	写	数据	TRANS[1:0]	Prot0
1	pc+2i	w/h	0	(pc+2i)	S 周期	0
	pc+3i					

第 8 章 AC 参数

第 8 章给出了 ARM7TDMI-S 处理器的 AC 时序参数。包含以下小节：

- 时序图
- AC 时序参数定义

8.1 时序图

这一节包含以下时序图：

- 数据访问的时序参数
- 协处理器时序
- 异常和配置输入时序
- 调试时序
- 扫描时序

8.1.1 数据访问的时序参数

数据访问的时序参数见图 8-1。

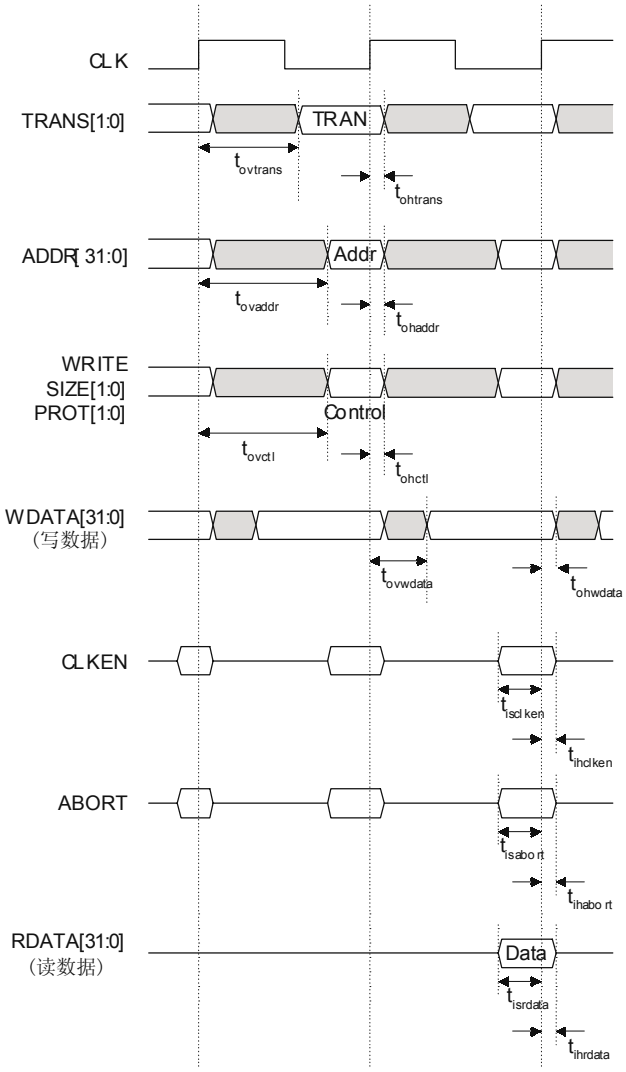

图 8-1 数据访问时序参数

注：

图 8-1 包含了数据读和写访问的时序。**WRITE** 信号用来指示是否使用 **RDATA** 或 **WDATA** 端口。

当读或写处理不能在一个周期内完成时，CLKEN 为低使数据访问延长。当处理信号 TRANS[1:0]指示一个有效的存储器周期或一个协处理器寄存器传送周期时，数据总线只用于传送。

8.1.2 协处理器时序

协处理器时序如图 8-2 所示。

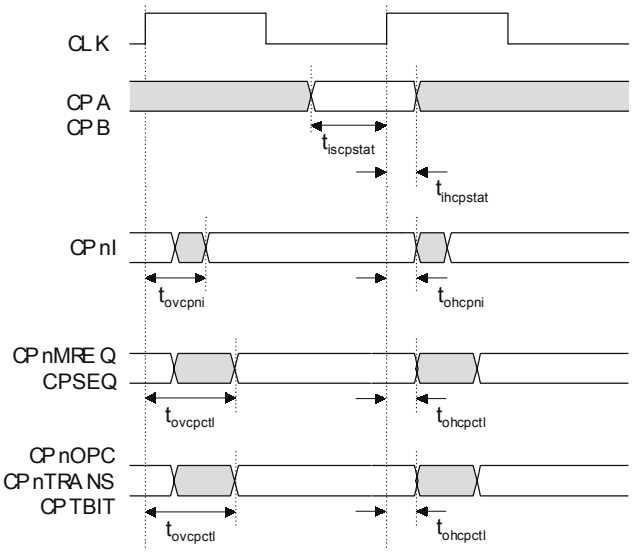

图 8-2 协处理器时序

8.1.3 异常和配置输入时序

异常和配置输入时序参数如图 8-3 所示。

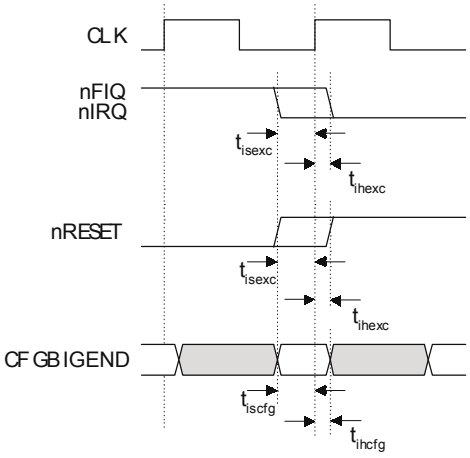

图 8-3 异常和配置输入时序

8.1.4 调试时序

调试时序参数如图 8-4 所示。

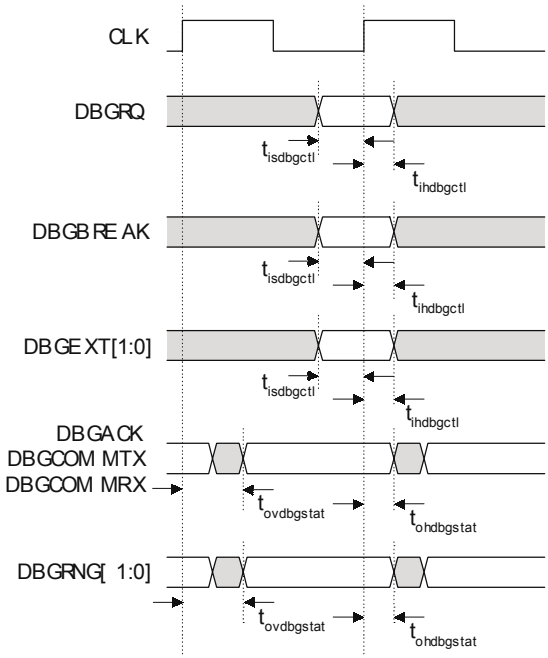

图 8-4 调试时序

注：

DBGBREAK 在时钟上升沿采样，因此外部取决于数据的断点和观察点必须在此边沿匹配和通知。

8.1.5 扫描时序

扫描时序参数如图 8-5 所示。

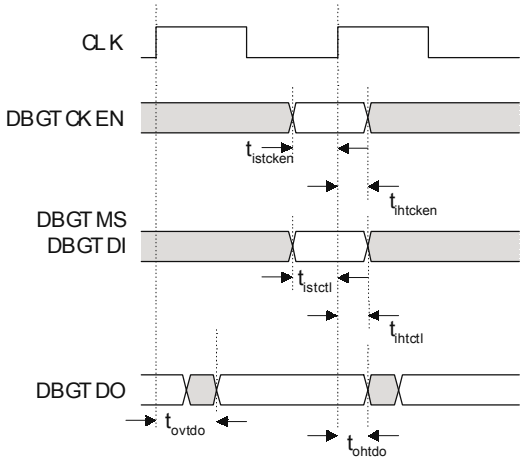

图 8-5 扫描时序

8.2 AC 时序参数定义

表 8-1 所示为目标 AC 参数。在最大操作频率时，所有数字都以 CLK 周期的百分比形式表示。

注：

当显示 0%时，表示保存时间到时钟边沿加上最大的时钟失真用于内部时钟缓冲。

表 8-1 暂时性的 AC 参数

符号	参数	最小	最大
t_{cyc}	**CLK** 周期时间	100%	-
$t_{isclken}$	**CLKEN** 输入建立到 **CLK** 上升沿	40%	-
$t_{ihclken}$	**CLK** 上升沿到 **CLKEN** 输入保持	-	0%
$t_{isabort}$	**ABORT** 输入建立到上升沿 **CLK**	15%	-
$t_{ihabort}$	**CLK** 上升沿到 **ABORT** 输入保持	-	0%
$t_{isrdata}$	**RDATA** 输入建立到 **CLK** 上升沿	10%	-
$t_{ihrdata}$	**CLK** 上升沿到 **RDATA** 输入保持	-	0%
t_{ovaddr}	**CLK** 上升沿到 **ADDR** 有效	-	90%
t_{ohaddr}	**CLK** 上升沿到 **ADDR** 保持	>0%	-
t_{ovctl}	**CLK** 上升沿到控制有效	-	90%
t_{ohctl}	**CLK** 上升沿到控制保持时间	>0%	-
$t_{ovtrans}$	**CLK** 上升沿到处理类型有效	-	50%
$t_{ohtrans}$	**CLK** 上升沿到处理类型保持时间	>0%	-
$t_{ovwdata}$	**CLK** 上升沿到 **WDATA** 有效	-	40%
$t_{ohwdata}$	**CLK** 上升沿到 **WDATA** 保持时间	>0%	-
$t_{iscpstat}$	**CPA**, **CPB** 输入建立到 **CLK** 上升沿	20%	-
$t_{ihcpstat}$	**CLK** 上升沿到 **CPA**, **CPB** 输入保持	-	0%
$t_{ovcpctl}$	**CLK** 上升沿到 协处理器控制有效	-	80%
$t_{ohcpctl}$	**CLK** 上升沿到协处理器控制保持时间	>0%	-
t_{ovcpni}	**CLK** 上升沿到协处理器 **CPnI** 有效	-	40%
t_{ohcpni}	**CLK** 上升沿到协处理器 **CPnI** 保持时间	>0%	-
t_{isexc}	**nFIQ**, **nIRQ**, **nRESET** 建立到 **CLK** 上升沿	10%	-
t_{ihexc}	**CLK** 上升沿到 **nFIQ**, **nIRQ**, **nRESET** 保持	-	0%
t_{iscfg}	**CFGBIGEND** 建立到 **CLK** 上升沿	10%	-
t_{ihcfg}	**CLK** 上升沿到 **CFGBIGEND** 保持	-	0%
$t_{isdbgstat}$	调试状态输入建立到 **CLK** 上升沿	10%	-
$t_{ihdbgstat}$	**CLK** 上升沿到调试状态输入保持	-	0%
$t_{ovdbgctl}$	**CLK** 上升沿到调试控制有效	-	40%
$t_{ohdbctl}$	**CLK** 上升沿到调试控制保持时间	>0%	-
$t_{istcken}$	**DBGTCKEN** 输入建立到 **CLK** 上升沿	40%	-
$t_{ihtcken}$	**CLK** 上升沿到 **DBGTCKEN** 输入保持	-	0%
t_{istctl}	**DBGTDI**, **DBGTMS** 输入建立到 **CLK** 上升沿	35%	-
t_{ihtctl}	**CLK** 上升沿到 **DBGTDI**, **DBGTMS** 输入保持	-	0%
t_{ovtdo}	**CLK** 上升沿到 **DBGTDO** 有效	-	20%
t_{ohtdo}	**CLK** 上升沿到 **DBGTDO** 保持时间	>0%	-
$t_{ovdbgstat}$	**CLK** 上升沿到调试状态有效	40%	-
$t_{ohdbgstat}$	调试状态保持时间	>0%	-

附录 A 信号描述

表 A-1 信号描述

名称	类型	描述
ABORT	输入	存储器访问中止或总线错误 这是一个输入信号，被存储器系统用于向处理器通知所请求的访问不被接受。
ADDR[31:0]	输出	处理器地址总线
CFGBIGEND	输入	Big-endian 配置 当该信号为高时，处理器将存储器中的字节看作是 big-endian 格式。当信号为低时，存储器被看作是 little-endian 格式。 **CFGBIGEND** 通常为一个静态配置信号。 该信号与硬件宏单元的 **BIGEND** 类似。
CLK	输入	时钟输入 该时钟为所有 ARM7TDMI-S 存储器访问和内部操作定时。所有输出都在 **CLK** 上升沿变化，而所有输入信号都在 **CLK** 上升沿采样。 **CLKEN** 输入可以和一个自由运行的 **CLK** 一起使用以增加同步的等待状态。另外，该时钟可以在任何阶段无限延长，这样可用于访问低速的外设或存储器或使系统进入低功耗状态。**CLK** 还与 EmbeddedICE-RT 工具链一起用于串行扫描链调试操作。 该信号类似于硬件宏单元的反相 **MCLK** 信号。
CLKEN	输入	等待状态控制 当访问低速外设时，ARM7TDMI-S 可通过将 **CLKEN** 拉低来等待整数个 **CLK** 周期。当不使用 **CLKEN** 控制时，必须将其接高电平。 该信号与硬件宏单元的 **nWAIT** 类似。
CPA	输入	协处理器缺席握手信号 一个能够执行 ARM7TDMI-S 所请求的操作 (通过声明 **CPnI**) 的协处理器在先于协处理器访问的周期边沿将 **CPA** 拉低。当 **CPA** 为高电平并且正在执行协处理器周期（由 **CPnI** 低电平指示）时，ARM7TDMI-S 中止协处理器握手并执行未定义指令陷阱。当 **CPA** 为低并且保持为低，ARM7TDMI-S 处于忙等待直到 **CPB** 为低为止，然后完成协处理器指令。
CPB	输入	协处理器忙握手信号 一个能够执行 ARM7TDMI-S 所请求的操作(通过声明 **CPnI**)的协处理器。但不能立即启动操作，这通过将 **CPB** 拉高来指示。当协处理器准备启动时将 CPB 拉低，信号在协处理器指令执行周期开始之前建立。
CPnI	输出	/协处理器指令信号 当 ARM7TDMI-S 执行一个协处理器指令时，它将该输出信号拉低并等待协处理器的响应。所执行的动作取决于协处理器通过 **CPA** 和 **CPB** 输入所作出的响应。
CPnMREQ	输出	/存储器请求 当为低时，该信号指示处理器在下一次处理时要求对存储器进行访问。 该信号与硬件宏单元的 **nMREQ** 类似
CPnOPC	输出	/操作码取指 当为低时，该信号指示处理器正在从存储器对指令取指。当为高电平时，数据（如果存在）被传送。 该信号与硬件宏单元的 **nOPC** 和 AMBA ASB 的 **BPROT[0]**类似。
CPSEQ	输出	连续地址 当下一个存储器周期的地址与前一个存储器访问的地址相关时，该输出信号变为高电平。新的地址在 ARM 状态下，等于前一个地址加 4，而 Thumb 状态下等于前一个地址加 2。 该信号与硬件宏单元的 **SEQ** 类似。

CPTBIT	输出	当为高电平时，该信号通知协处理器，处理器正在执行 Thumb 指令集。为低电平时，表示处理器正在执行 ARM 指令集。
CPnTRANS	输出	/存储器转换 当为低电平时，该信号指示处理器处于用户模式。它可在地址旁路转换时通知存储器管理硬件或作为特权模式动作的指示信号。 该信号类似于硬件宏单元的 **nTRANS**。
DBGACK	输出	调试应答 为高电平时，该信号指示 ARM7TDMI-S 处于调试状态。它只有当 **DBGEN** 为高电平时才可使能.
DBGBREAK	输入	EmbeddedICE-RT 断点/观察点指示信号 该信号使能外部硬件来暂停处理器调试的执行。当为高电平时，该信号导致当前存储器访问被中断。当存储器访问为指令取指时，如果指令到达流水线的执行阶段，则 ARM7TDMI-S 进入调试状态。当访问存储器数据时，ARM7TDMI-S 在当前指令执行完毕后进入调试状态。这实现了由 EmbeddedICE-RT 模块提供的内部断点的扩展。**DBGBREAK** 只有在 **DBGEN** 为高电平时才可使能。 该信号与硬件宏单元的 **BREAKPT** 类似。
DBGCOMMRX	输出	EmbeddedICE-RT 通信通道接收 为高电平时，该信号指示通信通道接收缓冲区已满。**DBGCOMMRX** 只有在 **DBGEN** 为高时才可使能。 该信号类似于硬件宏单元的 **COMMRX**。
DBGCOMMTX	输出	EmbeddedICE-RT 通信通道发送 为高电平时，该信号通信通道发送缓冲区为空。**DBGCOMMTX** 只有在 **DBGEN** 为高时才可使能。 该信号类似于硬件宏单元的 **COMMTX**。
DBGEN	输入	调试使能 该输入信号使能 ARM7TDMI-S 的调试特性。如果您想要使用 ARM7TDMI-S 的调试特性，请将 该信号接高电平。只有在不需要调试时才将其接低电平。
DBGnEXEC	输出	未执行 为高电平时，该信号表示在执行单元内的指令没有被执行（例如因为条件代码检测失败）。
DBGEXT[1:0]	输入	EmbeddedICE-RT 外部输入 0 和 1 EmbeddedICE-RT 宏单元逻辑的输入信号，可通过它们实现由外部条件决定的断点和/或观察点。只有在 **DBGEN** 为高电平时才可使能。 该信号类似于硬件宏单元的 **EXTERN[1:0]**。
DBGINSTRVALID	输出	指令执行信号 在每个指令到达流水线的执行阶段时输出一个周期的高电平。ETM7 用它来跟踪 ARM7TDMI-S 处理器流水线。 该信号类似于硬件宏单元的 **INSTRVALID**。
DBGRNG[1:0]	输出	EmbeddedICE-RT rangeout 该信号指示 EmbeddedICE-RT 观察点寄存器与当前地址、数据和控制总线的条件相匹配。该信号与观察点使能控制位的状态无关。该信号只有在 **DBGEN** 为高电平时才可使能。 该信号类似于硬件宏单元的 **RANGE[1:0]**。
DBGRQ	输出	调试请求 该内部同步输入信号向处理器请求进入调试状态。该信号只有在 **DBGEN** 为高电平时才可使能。
DBGTCKEN	输入	测试时钟使能 该信号只有在 **DBGEN** 为高电平时才可使能。
DBGTDI	输入	EmbeddedICE-RT 数据输入/JTAG 测试数据输入 该信号只有在 **DBGEN** 为高电平时才可使能。
DBGTDO	输出	EmbeddedICE-RT 数据输出/JTAG 扫描输出 该信号只有在 **DBGEN** 为高电平时才可使能。
DBGnTDOEN	输出	**/DBGTDO** 使能 为低电平时，该信号表示串行数据已经从 **DBGTDO** 端口

		输出。**DBGnTDOEN** 通常作为 **DBGTDO** 管脚的输出使能。
DBGTMS	输入	EmbeddedICE-RT 模式/JTAG 测试模式选择 **DBGTMS** 只有在 **DBGEN** 为高电平时才可使能。
DBGnTRST	输入	/测试复位 这是一个提供给 EmbeddedICE-RT 宏单元内部状态的低有效复位信号。
DMORE	输出	声明 LDM 和 STM 指令 该信号使存储器访问效率更高。
nFIQ	输入	低有效快速中断请求 这是一个向处理器发出的高优先级的同步中断请求。如果处理器在该信号为低电平时使能相应中断，则进入中断状态。该信号为电平触发，必须保持低电平直到接收到处理器对应的中断应答 当 **ISYNC** 为高电平时，该信号类似于硬件宏单元的 **nFIQ**。
nIRQ	输入	低有效中断请求 这是一个向处理器发出的低优先级的同步中断请求。如果处理器在该信号为低电平时使能相应中断，则进入中断状态。该信号为电平触发，必须保持低电平直到接收到处理器对应的中断应答 当 **ISYNC** 为高电平时，该信号类似于硬件宏单元的 **nIRQ**。
LOCK	输出	锁定的处理操作 当 **LOCK** 为高电平时，处理器执行一个锁定的存储器访问，仲裁器必须一直等待 **LOCK** 变为低电平之后才能允许其它器件访问总线。
PROT[1:0]	输出	这些输出到存储器系统的信号用于指示输出为代码还是数据，以及访问属于用户模式还是特权模式： x0 操作码取指 x1 数据访问 0x 用户模式访问 1x 超级用户或特权模式访问
RDATA[31:0]	输入	读数据输入总线 这是一个用于传输处理器和存储器之间的指令和数据的读数据总线。该总线上的数据由处理器在读访问时钟周期结束时进行采样（也就是说 **WRITE** 为低电平）。 该信号类似于硬件宏单元的 **DIN[31:0]**。
nRESET	输入	/复位 该输入信号强制处理器终止当前的指令并进入超级用户模式下的复位向量。它必须保持至少 2 个周期。 低电平强制正在执行的指令在下一个非等待周期异常终止并导致处理器在总线接口执行空闲周期。当 **nRESET** 恢复到高电平至少 1 个周期之后，处理器从地址 0 开始重新启动。
SCANENABLE	输入	扫描测试路径使能（用于自动测试模式产生） 该信号在正常系统配置时为低电平，在扫描测试时为高电平。
SCANIN	输入	扫描测试路径串行输入（用于自动测试模式产生） 当 **SCANENABLE** 有效(高电平)时，串行移位寄存器输入有效。
SCANOUT	输出	扫描测试路径串行输出（用于自动测试模式产生） 当 **SCANENABLE** 有效(高电平)时，串行移位寄存器输出有效。

SIZE[1:0]	输出	存储器访问宽度 该输出信号指示外部存储器系统要求字或者半字或者字节长度的传输： 00 8 位字节访问 (由 **ADDR[1:0]** 在字当中寻址) 01 16 位半字访问 (由 **ADDR[1]** 在字当中寻址) 10 32 字访问 (总是按字对齐) 11 (保留) 该信号类似于硬件宏单元的 **MAS[1:0]**。
TRANS[1:0]	输出	下一次处理的类型 **TRANS** 用于指示下一次处理的类型： 00 仅为地址 (内部操作周期) 01 协处理器 10 非连续地址的存储器访问 11 连续突发地址的存储器访问 **TRANS[1]** 信号类似于硬件宏单元的 **nMREQ**，而 **TRANS[0]** 信号类似于硬件宏单元的 **SEQ**。**TRANS** 该信号类似于 AMBA 系统总线的 **BTRAN**。
V_{DD}		电源
V_{SS}		所有信号的参考地
WDATA[31:0]	输出	写数据输出总线 它用于将数据从处理器传输到存储器或协处理器系统。写数据在存储访问周期结束时（也就是说当 **WRITE** 为高电平时）建立，并在整个等待状态保持有效。 该信号类似于硬件宏单元的 **DOUT[31:0]**。
WRITE	输出	写/读访问 为高电平时，**WRITE** 表示处理器的写周期，为低电平时表示处理器读周期。 该信号类似于硬件宏单元的 **nRW** 。