第3章 chapter 3

ARM处理器技术

如前面章节提到的，嵌入式系统处理器的种类十分多样，在学习过程中，往往需要以某种或者某类处理器为例，具体对其结构、功能和使用说明进行学习。在嵌入式处理器中，ARM体系结构的处理器在嵌入式方面应用非常广泛。通过阅读本章，读者可以了解ARM不同系列的嵌入式芯片的设计中，厂商为处理器定义了哪些功能。本章重点介绍Cortex-A8的功能特点，以及生产厂商根据设计制作的对应Cortex-A8内核设计的实际处理器芯片S5PV210。

3.1 处理器体系结构

3.1.1 体系结构概述

为研究处理器技术，需要了解处理器技术包含的内容。在此之前，一起来看嵌入式系统的抽象层转换，它与一般计算机系统相同，如图3-1所示。

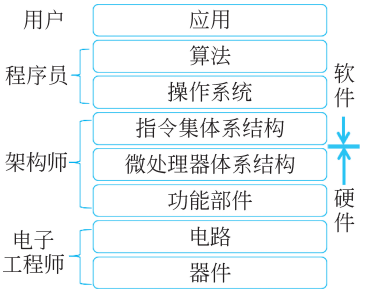


图3-1 嵌入式系统的抽象层转换

处理器体系结构

按照从硬件到软件的顺序，自下而上分别是器件、电路、功能部件、微处理器体系结构、指令集体系结构、操作系统、算法和应用。抽象层面对应的人员分别是电子工程师、架构师、程序员和用户。

体系结构是嵌入式系统硬件与软件的衔接，它确定嵌入式系统设计的部件、部件功能、部件间接口的设计，并集中于嵌入式系统的核心部分—处理器的运算与内存的存取。它包含一系列机器指令、基本数据类型、寄存器、寻址模式存储体系、中断和异常处理等内容。

处理器的体系结构，也称架构，包含指令集体系结构(Instruction Set Architecture，ISA)和微处理器体系结构或微架构(Microarchitecture)。指令集体系结构就是设计一系列指令(如数据处理和存储操作、算术和逻辑操作以及控制流操作等基本命令)，具体对应一系列机器二进制编码信号；微架构则是设计硬件电路如何具体执行指令，也就是处理器内部通过电压的高低电平对应上述的二进制编码指令。

软件要借助一定的词库和硬件沟通，这个词库就被称作“指令集”。大个头的计算机

里占据支配地位的思想是要有一个很大、很丰富的词库,可能有好几千个词,别的软硬件使用起来都比较方便。

体系结构定义了指令集和基于这一体系结构下处理器的编程模型,基于同种体系结构可以有多种处理器,每个处理器性能不同,所面向的应用不同,每个处理器的实现都要遵循这一体系结构。

体系结构主要分成以下两类。

复杂指令集(Complex Instruction Set Computer,CISC):通过设置一些功能复杂的指令,把一些原来由软件实现的、常用的功能改成用硬件的指令系统来实现,以此来提高计算机的执行速度。

精简指令集(Reduced Instruction Set Computer,RISC):通过简化计算机指令功能,使指令的平均执行周期减少,从而提高计算机的工作主频。

还有超长指令字(Very Long Instruction Word,VLIW)指令集等一些衍生指令集架构,通过将多条指令放入一个指令字实现指令集并行处理,从而提高计算运行效率。

3.1.2 流水线技术

在现代处理器中,流水线是一个最基本的概念。在了解 CPU 时,很多时候会提及拥有多少级流水线。虽然这个概念并不是在计算机技术中诞生的,但是这个技术却在处理器世界中大放异彩。

流水线(Pipeline)技术是指程序在执行时,多条指令重叠进行操作的一种准并行处理实现技术。通俗地讲,将一个时序过程,分解成若干子过程,每个过程都能有效地与其他子过程同时执行。这种思想最初是在 RISC 体系结构中出现的,旨在提高处理器处理效率,争取在一个时钟周期中完成一条指令。


流水线技术

1. 流水线指令执行分级

最经典的当属无内部互锁的流水线处理器(Microprocessor without Interlocked Piped Stages,MIPS)的 5 级流水线技术。MIPS 属于 RISC 体系结构,本身就是为了流水线而设计的,CPU 在高速缓存中运行,每条指令的执行过程都分成 5 级。每级成为一个流水线阶段,每个阶段占用固定的时间,通常是一个时钟周期。图 3-2 中为 MIPS 5 级流水线结构图。

1) 取指令

取指令(Instruction Fetch,IF)阶段是将一条指令从主存中取到指令寄存器的过程。

2) 指令译码

取出指令后,计算机立即进入指令译码(Instruction Decode,ID)阶段。在指令译码阶段,指令译码器按照预定的指令格式,对取回的指令进行拆分和解释,识别区分出不同的指令类别以及各种获取操作数的方法。

3) 指令执行

在取指令和指令译码阶段之后,接着进入执行指令(Execute,EX)阶段。此阶段的任务是完成指令所规定的各种操作,具体实现指令的功能。为此,CPU 的不同部分被连接起来,以执行所需的操作。

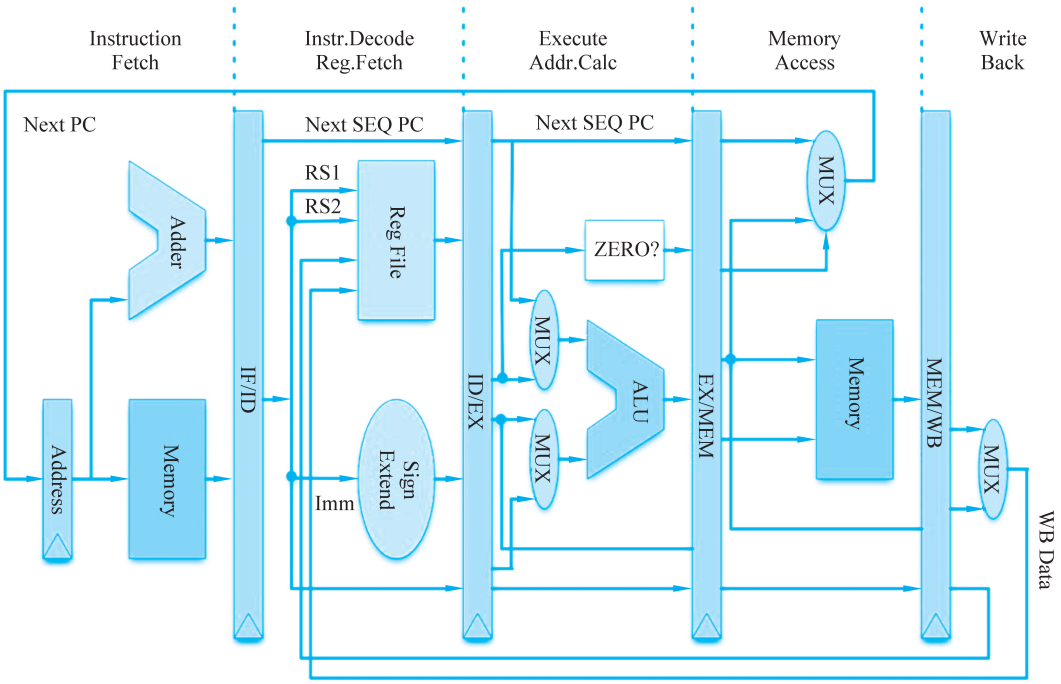


图 3-2 MIPS 5 级流水线结构图

4）访存取数阶段

根据指令需要，有可能要访问主存读取操作数，这样就进入了访存取数（Memory，MEM）阶段，此阶段的任务是根据指令地址码，得到操作数在主存中的地址，并从主存中读取该操作数用于运算。

5）结果写回阶段

作为最后一个阶段，结果写回（Write Back，WB）阶段把执行指令阶段的运行结果写回到某种存储形式。

如图 3-3 所示，假设每条指令的运行时间为一个时钟周期 1000ps，那么单级操作的时间为 200ps，执行 3 条指令不采用流水线和采用流水线方式的时间分别为 3000ps 和 1400ps。那么执行 n 条指令所需时间为 1000ps＋（n－1）×200ps。

2. 处理器冒险

很明显，如果只执行一条指令，流水线是不会提高效率的。但是如果要完成多条指令，利用流水线的并行原理，其实可以提高几倍的处理速度。那么流水线的级数是不是越多越好呢？流水线级数越多，在处理多指令的时候确实也会越高效，但也需要更多的寄存器，也就是用空间换时间；同时也会出现很多相关的副作用，被称为流水线冒险（HaZard），处理器中的冒险有以下 3 类。

1）结构冒险

由于处理器资源冲突，而无法实现某些指令或者阶段的组合实现，就称为处理器有结构冒险。比如，早期的处理器中，程序和数据是存储在一起的，当 IF 和 MEM 同时访问存储器导致有一个操作要等待，此时结构冒险就出现了。现在的处理器已经解决了该问题，指令存储在 L1P Cache 中，数据存储在 L1D Cache 中，单独访问，不会影响相互操作。

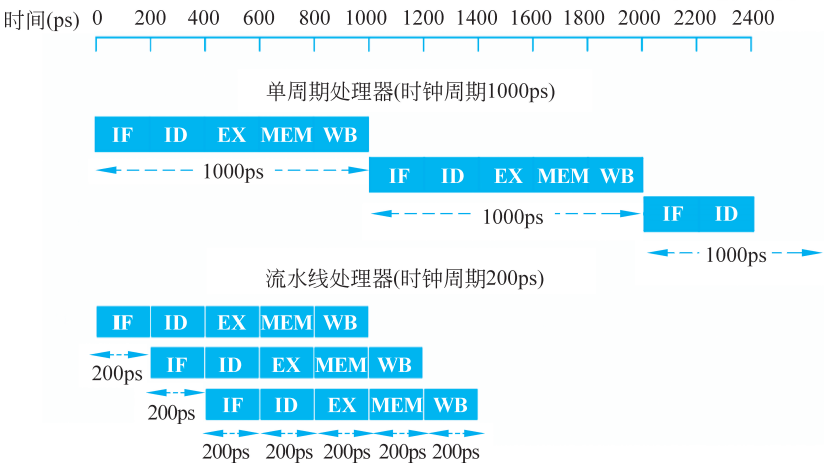


图 3-3　5 级流水线示意图

2）数据冒险

如果流水线中原来有先后顺序的指令在同一时刻处理时，可能会导致出现访问错误的数据的情况。比如，指令 1 将寄存器 R2、R3 的和赋予 R1，改变 R1 的值；而紧接着将 R1、R4 的和赋予 R5 则会使用 R1 的值，可是 R1 必须在指令 1 的第 5 级才能更新到寄存器中，指令 2 在第 4 级就要访问 R1，也就是说指令 2 此时在使用错误的 R1 的值，数据冒险就出现了。

最简单的处理方法是在两条指令中添加一条空指令—NOP，但是会影响处理器的指令的执行效率。在现代处理器技术中，已经用 forwarding 或 bypassing 的方式解决了如果处理器在检测到当前指令的源操作数正好在流水线的 EX 或者 MEM 阶段，直接将 EX 和 MEM 寄存器的值传递给 ALU 的输入，而不再从寄存器堆中获取数据，因为此时寄存器堆中的数据可能是没有被及时更新的。当然，不仅在 EX 阶段出现这种问题，在 MEM 阶段也容易出现。

3）控制冒险

流水线执行指令时，由于并行处理的关系，后面很多指令其实都在流水线中开始处理，包括预取值和译码。那么，如果此时程序中出现一条跳转语句怎么办？在流水线中，多条指令是并行执行的，在指令 1 执行的时候，后续的指令 2 和指令 3 可能已经完成 IF 和 ID 两个阶段等待被执行，此时如果指令 1 跳到其他地方，那么指令 2 和指令 3 的 IF 和 ID 就是无用功了，此时出现了控制冒险。这种情况对于程序和效率来说存在很大损失。

当然，简单的解决方法是在 Jump 指令后面(不会被真正使用，但是会进入流水线)添加空指令，在 MIPS 程序中，经常在 Jump 指令后面添加空指令，更好的方法是采用预测进行判断。

解决方法就是使用分支预测技术，通过预测的方式获取分支指令下一条指令的取值。分支预测技术分为静态分支预测和动态分支预测。静态分支预测如规定向后跳转的分支指令总是预测为跳，否则总是预测为不跳。动态分支预测即根据以往的跳转记录来判断当前是否应跳转。

3.1.3 寻址和存储器设计

CPU内部的核心组件有各类寄存器、控制单元(CU)、逻辑运算单元(ALU)、高速缓存。

CPU和外部交互的交通大动脉就是3种总线,即地址总线、数据总线和控制总线,I/O设备、RAM通过3种总线和CPU实现功能交互。

地址总线(Address Bus):当CPU要处理数据时,首先地址总线会根据要处理的数据地址在内存中找到要处理的数据,然后再对其进行相应处理。也就是说,地址总线的宽度决定CPU能够处理的数据地址的总量大小。由于地址总线总是由CPU发起的寻址要求,因此数据总线只能由CPU主动向外部寻找它所需要的地址,而不能被外部的I/O设备主动使用。因此,地址总线是单向(只能由CPU主动去读而不会对其进行写)三态(即高电平、低电平、高阻态)的。

数据总线(Data Bus):当CPU需要读取或写入数据的时候由地址总线先去寻址,然后再由数据总线对读到的地址进行读或写操作(CPU一次读写的数据总量由数据总线的宽度决定),最终实现CPU所要实现的相应功能。由于CPU会通过数据总线对所要处理的地址进行读和写两种操作,因此,数据总线是双向三态的。

控制总线(Control Bus):当CPU要对外部的外设(I/O设备或内存)进行操作的时候,控制总线会对相应的外设发出相应的控制指令,然后再由数据总线对其进行操作。由于控制总线要对所有的外设进行控制,因此,它是双向的且十分灵活,而其总线的宽度也是不确定的,主要由外设的数量决定其宽度。

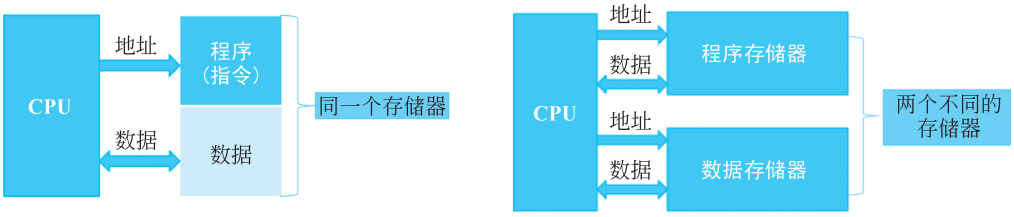
CPU在处理数据时其实是要先由控制总线发送控制指令,然后再对外设(这里的外设包括I/O设备和内存)进行寻址操作,只有找到要操作的硬件或内存的地址之后才能对其进行相应操作。而这个寻址方式是由CPU对地址的编址方式决定的。而编址方式就分为统一编址和独立编址。

统一编址是当CPU要对I/O设备进行操作时,就直接由地址总线进行寻址,因为地址总线的宽度是一定的,也就是所寻址的范围是一定的。而如果由它直接对I/O设备进行寻址,那势必会占用有限的地址,进而导致对其能够操作的内存大小就会有所限制。这也导致地址总线所能寻到的内存地址并不和自己的总线宽度对应。(假如一个32位宽度的地址总线本来可以寻址4GB的内存地址,但是由于一部分外设占用了部分地址,因此它所能使用的内存就会小于4GB。)

独立编址是指CPU在对I/O设备进行操作时会有专门的操作指令,而不会用内存操作指令,这样就会使地址总线仅服务于内存,而不会被其他I/O外设占用,也因此会使内存的大小等于地址总线的宽度。

在存储器设计思想上,有两种结构——冯·诺依曼结构和哈佛结构,其示意图如图3-4所示。

冯·诺依曼结构,也叫普林斯顿结构,指令和数据是不加区别地混合存储在同一个存储器中的,共享数据总线。指令和数据地址指向同一个存储器的不同物理位置,指令和数据的宽度相同。由于指令和数据放在同一个存储器中,因此冯·诺依曼结构中不能同时获取指令和数据。又由于存储器的速度远低于CPU的速度,从而使CPU与存储器



(a) 冯·诺伊曼结构　　(b) 哈佛结构

图 3-4　冯·诺依曼结构和哈佛结构

交换数据成了影响高速计算和系统性能的瓶颈。冯·诺依曼结构则是将逻辑代码段和变量统一存储在内存中,它们之间一般是按照代码的执行顺序依次存储。这样就会导致一个问题,如果当程序出现 BUG 时,由于程序没有对逻辑代码段的读/写限定,将拥有和普通变量一样的读/写操作权限,因此一旦逻辑执行出现问题就容易死机。但是,冯·诺依曼结构的好处是可以充分利用有限的内存空间,并且会使 CPU 对程序的执行方便。

哈佛结构,指令和数据是完全分开的,存储器分为程序存储器和数据存储器。哈佛结构至少拥有 2 组总线,即程序存储器的数据总线和地址总线,数据存储器的数据总线和地址总线,这种分开的程序总线和数据总线,可允许同时获取指令字(来自程序存储器)和操作数(来自数据存储器),是互不干扰。这意味着在一个机器周期内可以同时准备好数据和指令,本条指令执行时可以预取下一条指令,所以哈佛结构的 CPU 具有较高的执行效率。同时由于指令和数据分开存放,可以使指令和数据有不同的宽度。(例如,51 单片机的程序的逻辑代码段放在 ROM 中,而变量部分则放在 ROM 中;而 ARM 的逻辑代码和变量都存放在 RAM(内存)中,但是,它在内存中划分了两部分空间,其中一部分是逻辑代码,另一部分是变量,二者之间不会相互干扰)。哈佛结构的优点就是逻辑代码和变量单独存放,不会相互干扰,因而当程序出现 BUG 时,最多只会修改变量的值,而不会修改程序的执行顺序(即逻辑关系)。

许多现代微型计算机的高速缓冲存储器采用哈佛结构,将 Cache 分为指令 Cache 和数据 Cache,而主存采用冯·诺依曼结构,只有一个存储器,由数据和指令混用。因此,将哈佛结构和冯·诺依曼结构结合,不仅可以提高主存的利用率,而且可以提高程序执行的效率,缩短指令执行的时钟周期。

3.1.4　RISC 和 CISC 的区别

1. RISC 和 CISC 指令集的区别

RISC 处理器减少了指令种类,RISC 的指令种类只提供简单的操作,使一个周期就可以执行一条指令。编译器或者程序员通过几条简单指令的组合实现一个复杂的操作(如除法操作)。RISC 采用定长指令集,每条指令的长度都是固定的,允许流水线在当前指令译码阶段去取其下一条指令;而在 CISC 处理器中,指令长度通常不固定,执行也需要多个周期。因此,CISC 强调硬件的复杂性,RISC 注重编译器的复杂性,如图 3-5 所示。

2. RISC 和 CISC 流水线的区别

RISC 指令的处理过程被拆分成几个更小的、能够被流水线并行执行的单元。在理想情况下,流水线每周期前进一步,可获得最高的吞吐率;而 CISC 指令的执行需要调用

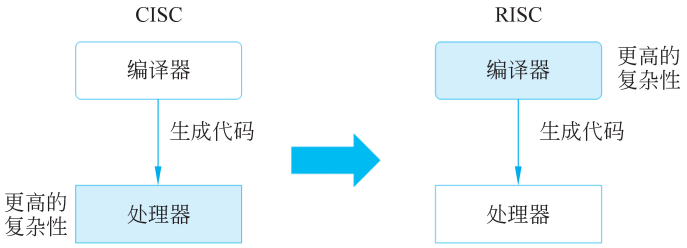


图 3-5 RISC 和 CISC 的区别

微代码的一个微程序。

3. RISC 和 CISC 寄存器的区别

RISC 处理器拥有更多的通用寄存器，每个寄存器都可存放数据或地址。寄存器可为所有的数据操作提供快速的局部存储访问，而 CISC 处理器都是用于特定目的的专用寄存器。

4. RISC 和 CISC 读取方式的区别


RISC 和 CISC

RISC 处理器只处理寄存器中的数据。独立的 load 和 store 指令用来完成数据在寄存器和外部存储器之间的传送。因为访问存储器很耗时，所以把存储器访问和数据处理分开。可反复地使用保存在寄存器中的数据，避免多次访问存储器，从而达到提高程序执行性能的目的。CISC 处理器能够直接处理存储器中的数据。

下节介绍的 ARM 处理器体系结构就属于 RISC 体系结构，RISC 的特点可以归纳如下。

- 指令数目少，采用相同的字节长度。
- 寻址方式简化，大部分使用寄存器寻址方式。
- 主要采用寄存器间的数据操作。
- 简化处理器结构，特别是控制器的设计。
- 使用处理器并行技术，适合流水线、超级流水线和超标量技术。

因此，基于此类体系结构的处理器更适合嵌入式系统。

3.2 ARM 处理器体系结构

3.2.1 ARM 简介

ARM 是 Advanced RISC Machines 的缩写，是一家微处理器行业的知名企业，该企业设计了大量高性能、廉价、耗能低的 RISC 处理器。ARM 公司的特点是只设计芯片，而不生产。它将技术授权给世界许多著名的半导体、软件和 OEM 厂商，并提供服务，图 3-6 展示了 ARM 的众多授权厂商。

ARM 处理器的应用非常广泛，包括以下几个领域。

1. 工业控制领域

作为 32 位的 RISC 架构，基于 ARM 微处理器核的微控制器芯片不但占据高端微控制器市场的大部分市场份额，同时也逐渐向低端微控制器应用领域扩展，ARM 微控制器

图 3-6 ARM 内核 IP 授权分布

的功耗低、性价比高,向传统的 8 位/16 位微控制器提出了挑战。

2. 无线通信领域

目前已有超过 85%的无线通信设备采用 ARM 技术,ARM 以其高性能和低成本的优势,在该领域的地位日益巩固。

3. 网络应用

随着宽带技术的推广,采用 ARM 技术的 ADSL 芯片正逐步获得竞争优势。此外,ARM 在语音及视频处理上进行了优化,并获得广泛支持,也对 DSP 的应用领域提出了挑战。

4. 消费类电子产品

ARM 技术在数字音频播放器、数字机顶盒和游戏机中广泛采用。

5. 成像和安全产品

现在流行的绝大部分数码相机和打印机中采用 ARM 技术,手机中的 32 位 SIM 智能卡也采用了 ARM 技术。

ARM 体系结构为嵌入式系统发展商提供很高的系统性能,同时保持优异的功耗和面积效率。


ARM 体系结构

3.2.2 ARM 体系结构版本

ARM 体系结构为满足 ARM 合作者及设计领域的一般需求正稳步发展。目前,ARM 体系结构共定义了 9 个版本,从版本 1 到版本 9,ARM 体系的指令集功能不断扩大,不同系列的 ARM 处理器,性能差别很大,应用范围和对象也不尽相同,但是,如果是相同的 ARM 体系结构,那么基于它们的应用软件是兼容的。

1. 版本 1(v1)

该版本的 ARM 体系结构,只有 26 位的寻址空间,没有商业化,其特点如下。

- 基本的数据处理指令(不包括乘法)。

- 字节、字和半字加载/存储指令。
- 具有分支指令，包括在子程序调用中使用的分支和链接指令。
- 在操作系统调用中使用的软件中断指令。
- 寻址空间为64MB。

2. 版本2(v2)

该版架构对v1版进行了扩展，如ARM2和ARM3(v2a)架构，包含对32位乘法指令和协处理器指令的支持。同样为26位寻址空间，现在已被废弃不再使用，它相对v1版本有以下改进。

- 具有乘法和乘加指令。
- 支持协处理器。
- 快速中断模式中的两个以上的分组寄存器。
- 具有原子性加载/存储指令SWP和SWPB。
- 寻址空间为64MB。

3. 版本3(v3)

ARM作为独立的公司，在1990年设计的第一个微处理器采用的是版本3的ARM6。它作为IP核、独立的处理器、具有片上高速缓存、MMU和写缓冲的集成CPU。变种版本有3G和3M。版本3G是不与版本2a向前兼容的版本3，版本3M引入有符号和无符号数乘法和乘加指令，这些指令产生全部64位结果。v3版架构(目前已废弃)对ARM体系结构作了较大的改动，具体包括以下内容。

- 寻址空间增至32位(4GB)。
- 分开的当前程序状态寄存器(CPSR)和备份的程序状态寄存器(SPSR)。
- 增加了两种异常模式，使操作系统代码可方便地使用数据访问中止异常、指令预取中止异常和未定义指令异常。
- 增加了MRS/MSR指令，以访问新增的CPSR/SPSR。
- 增加了从异常处理返回的指令功能。

4. 版本4(v4)

v4版架构在v3版上作了进一步扩充，v4版架构是目前应用较广的ARM体系结构，ARM7、ARM8、ARM9和StrongARM都采用该架构。v4不再强制要求与26位地址空间兼容，而且还明确哪些指令会引起未定义指令异常。指令集中增加了以下功能。

- 符号化和非符号化半字及符号化字节的存/取指令。
- 增加了T变种，处理器可工作在Thumb状态，增加了16位Thumb指令集。
- 完善了软件中断(SWI)指令的功能。
- 处理器系统模式引进特权方式时使用用户寄存器操作。
- 把一些未使用的指令空间捕获为未定义指令。

5. 版本5(v5)

v5版架构是在v4版基础上增加一些新的指令，ARM10和Xscale都采用该版架构。这些新增命令如下。

- 带有链接和交换的转移(BLX)指令。
- 计数前导零(CLZ)指令，中断(BRK)指令。

- 增加了数字信号处理指令(V5TE 版)。
- 为协处理器增加更多可选择的指令。
- 改进了 ARM/Thumb 状态之间的切换效率。
- E 变种—增强型 DSP 指令集,包括全部算法操作和 16 位乘法操作。

6. 版本 6(v6)

v6 版架构是 2001 年发布的,首先在 2002 年春季发布的 ARM11 处理器中使用。在降低耗电量的同时,还强化了图形处理性能。通过追加有效进行多媒体处理的单指令多数据(Single Instruction,Multiple Data,SIMD)功能,将语音及图像的处理功能提高到原型机的 4 倍。此架构在 v5 版基础上增加了以下功能。

- THUMBTM:35%代码压缩。
- DSP 扩充:高性能定点 DSP 功能。
- JazelleTM:Java 性能优化,可提高 8 倍。
- Media 扩充:音/视频性能优化,可提高 4 倍。

7. 版本 7(v7)

ARMv7 架构是在 ARMv6 架构的基础上诞生的,为 Cortex 系列内核使用,包括 A 系列、R 系列和 M 系列。

(1) Cortex-A(Application),主要高性能的处理器,相比于其他两种处理器,其特点是增加了内存管理单元(MMU),对于运行大型的应用操作系统,MMU 是必不可少的元件。

(2) Cortex-R(Real time),实时性代表的是处理时间上的确定性和低延迟,即一个操作可以在指定的短时间内完成,MMU 引入的地址转换通常不能满足其实时性的要求,所以 R 系列处理器并不挂载 MMU。

(3) Cortex-M(Microcontroller),微控制器,主打中低端市场,其特点在于低功耗、低成本,相对的高性能,在中低端市场,性价比通常是一个主要的衡量因素。

该架构采用了 Thumb-2 技术,它是在 ARM 的 Thumb 代码压缩技术的基础上发展起来的,并且保持了对现存 ARM 解决方案的完整的代码兼容性。Thumb-2 技术比纯 32 位代码少使用 31%的内存,减少了系统开销。同时,能够提供比已有的基于 Thumb 技术的解决方案高出 38%的性能。ARMv7 架构还采用了 NEON 技术,将 DSP 和媒体处理能力提高了近 4 倍,并支持改进的浮点运算,满足下一代 3D 图形、游戏物理应用以及传统嵌入式控制应用的需求。此外,ARMv7 还支持改良的运行环境,以迎合不断增加的 JIT(Just In Time)和 DAC(Dynamic Adaptive Compilation)技术的使用。

8. 版本 8(v8)

ARMv8 引入可用的 64 位和 32 位执行状态(Execution state),分别称为 AArch64 和 AArch32。AArch64 执行状态支持 A64 指令集,可以在 64 位寄存器中保存地址,并允许基本指令集中的指令使用 64 位寄存器进行处理。AArch32 执行状态是一个 32 位执行状态,它保留了与 Armv7-A 体系结构的向后兼容性,并增强了该体系结构,可以支持 AArch64 状态中包含的某些功能。它支持 T32 和 A32 指令集。

这是 ARM 公司的首款支持 64 位指令集的处理器架构。ARM 在 2012 年时推出基于 ARMv8 架构的处理器内核并开始授权,而面向消费者和企业的样机于 2013 年在苹果

的A7处理器上首次应用。ARMv7架构的主要特性在ARMv8架构中得以保留并进一步拓展，如TrustZone技术、虚拟化技术及NEON Advanced SIMD技术等。

9. 版本9（v9）

ARM在2021年正式发布了ARMv9，它在兼容ARMv8的基础上，继续使用AArch64作为基准指令集，保持向下兼容性，在此基础上分别在安全性、AI与ML以及可伸缩向量扩展和DSP上做出改进，扩展了应用范围。不再局限于移动/嵌入式市场，未来将发力PC、HPC高性能计算、深度学习等新市场，以满足全球对功能日益强大的安全、人工智能和无处不在的专用处理的需求。

ARMv9架构引入ARM机密计算体系结构（Confidential Compute Architecture，CCA）重新设计安全应用程序的工作方式。机密计算通过打造基于硬件的安全运行环境执行计算，保护部分代码和数据，免于被存取或修改，甚至不受特权软件的影响。

在机器学习方面，ARMv9架构支持BFloat16格式，从而更好地支撑Int8计算和BFloat16的机器学习；可伸缩向量扩展2（SVE2）的引入，则能更好地帮助开发者对高阶的应用场景进行开发，在处理5G、虚拟现实和增强现实以及图像和语音识别等任务负载时具有很大增益。ARM体系结构版本对应的处理器内核如表3-1所示。

表3-1 ARM体系结构版本对应的处理器内核

体系结构	内　核
v1	ARM1
v2	ARM2
v2a	ARM2aS，ARM3
v3	ARM6，ARM600，ARM610
v4	ARM7，ARM700，ARM710
v4T	ARM7TDMI，ARM710T，ARM720T，ARM740T
v5	Strong ARM，ARM8，ARM810
v5T	ARM9TDMI，ARM920T，ARM940T
v5TE	ARM9E-S，ARM10TDMI，ARM1020E
v6	ARM11，ARM1156T2-S，ARM1156T2F-S，ARM1176JZ-S，ARM11JZF-S
v7	ARM Cortex-A5，7，8，9，12，15；Cortex-R4，5，6；Cortex-M3，4
v8	ARM Cortex-A57，ARM Cortex-A72，ARM Cortex-A73
V9	ARM Cortex-X2，ARM Cortex-A710

ARM处理器为RISC芯片，其简单的结构使ARM内核非常小，这使得器件的功耗也非常低。它具有经典RISC的特点。为了使ARM指令集能够更好地满足嵌入式应用的需要，ARM指令集和单纯的RISC定义有以下几个方面的不同。

- 一些特定的指令周期数可变。
- 内嵌桶形移位器产生了更为复杂的指令。
- Thumb 16位指令集。
- 条件执行。

3.2.3 ARM 体系结构下处理器系列

1. ARM 处理器变种

ARM 每个系列都提供了一套特定的性能满足设计者对功耗、性能和体积的需求。在 ARM 体系中增加的某些特定功能称为 ARM 体系的某种变种(variant)。

1) Thumb 指令集(T 变种)

Thumb 指令集是将 ARM 指令集的一个子集重新编码而形成的一个指令集。ARM 指令长度为 32 位,Thumb 指令长度为 16 位。

与 ARM 指令集相比,Thumb 指令集具有一定的局限性,即完成相同的操作,Thumb 指令通常需要更多的指令。因此,在对系统运行时间要求苛刻的应用场合,ARM 指令集更适合。

Thumb 指令集不包含进行异常处理时需要的一些指令,所以在异常中断的低级处理时,还是需要使用 ARM 指令。这种限制决定了 Thumb 指令需要和 ARM 指令配合使用。

2) 长乘法指令(M 变种)

M 变种增加了两条用于进行长乘法操作的 ARM 指令:其中一条指令用于实现 32 位整数乘以 32 位整数,生成 64 位整数的长乘法操作;另一条指令用于实现 32 位整数乘以 32 位整数,然后再加上 32 位整数,生成 64 位整数的长乘加操作。

在需要长乘法的应用场合,使用 M 变种比较合适。然而,在有些应用场合,乘法操作的性能并不重要,在系统实现时就不适合增加 M 变种的功能。

3) 增强型 DSP 指令(E 变种)

E 变种包含一些附加的指令,这些指令用于增强处理器对一些典型 DSP 算法的处理性能,主要包括几条新的实现 16 位数据乘法和乘加操作的指令,实现饱和的带符号数的加减法操作的指令。

饱和的带符号数的加减法操作是在加减法操作溢出时,结果并不进行卷绕(wrapping around),而是使用最大的正数或最小的负数表示。进行双字数据操作的指令,包括双字读取指令 LDRD、双字写入指令 STRD 和协处理器的寄存器传输指令 MCRR/MRRC、Cache 预取指令 PLD。

4) Java 加速器 Jazelle(J 变种)

ARM 的 Jazelle 技术将 Java 的优势和先进的 32 位 RISC 芯片完美地结合在一起。Jazelle 技术提供了 Java 加速功能,可以得到比普通 Java 虚拟机高得多的性能。与普通的 Java 虚拟机相比,Jazelle 使 Java 代码运行速度提高 3 倍,功耗降低 80%。

Jazelle 技术使程序员可以在一个单独的处理器上同时运行 Java 应用程序、已经建立好的操作系统、中间件以及其他应用程序。与使用协处理器和双处理器相比,使用单独的处理器可以在提供高性能的同时,保证低功耗和低成本。

5) ARM 媒体功能扩展(SIMD 变种)

ARM 媒体功能扩展为嵌入式应用系统提供了高性能的音频/视频处理技术。这就要求处理器能够提供很强的数字信号处理能力,同时还必须保持低功耗,以延长电池的使用时间。ARM 的 SIMD 媒体功能扩展为这些应用需求提供了解决方案。

SIMD变种的主要特点是：可以同时进行两个16位操作数或者4个8位操作数的运算，提供了小数算术运算，用户可以定义饱和运算的模式，两套16位操作数的乘加/乘减运算，32位乘以32位的小数MAC，同时8位/16位选择操作。

ARM对处理器体系结构的命名规则为ARM[x][y][z][T][D][M][I][E][J][F][-S]，这些后缀的含义如下。

x：系统，如ARM7、ARM9。

y：存储管理/保护单元。

z：Cache。

T：Thumb 16位译码器(T变种)。

D：JTAG调试器。

M：长乘法指令(M变种)。

I：嵌入式跟踪宏单元。

E：增强型DSP指令(E变种)。

J：Java加速器Jazelle(J变种)。

F：向量浮点单元。

S：可综合版本。

ARM7TDMI之后的所有ARM内核，即使“ARM”标志后没有包含“TDMI”字符，也都默认包含TDMI的功能特性。

JTAG是由IEEE 1149.1标准测试访问端口和边界扫描结构描述的，是ARM用来发送和接收处理器内核与测试仪器之间调试信息的一系列协议。

嵌入式ICE宏单元是建立在处理器内部用来设置断点和观察点的调试硬件。

可综合版本意味着处理器内核是以源代码形式提供的。这种源代码形式可被编译成一种易于EDA工具使用的形式。

2. ARM处理器系列

ARM微处理器目前包括下面几个系列，以及其他厂商基于ARM体系结构的处理器，除具有ARM体系结构的共同特点外，每个系列的ARM微处理器都有各自的特点和应用领域。

- ARM7系列。
- ARM9系列。
- ARM9E系列。
- ARM10/10E系列。
- ARM11系列。
- SecurCore系列。
- Xscale处理器。
- StrongARM系列微处理器。
- Cortex系列处理器。

1）ARM7系列微处理器

ARM7内核采用冯·诺依曼体系结构，数据和指令使用同一条总线。内核有一条3级流水线，执行ARMv4指令集。该系列包括ARM7TDMI、ARM7TDMI-S、带有高速缓

存处理器宏单元的 ARM720T 和扩充了 Jazelle 的 ARM7EJ-S。该系列处理器提供 Thumb 16 位压缩指令集和 EmbeddedICE 软件调试方式,适用于更大规模的 SoC 设计中。

ARM7 系列微处理器主要用于对功耗和成本要求比较苛刻的消费类产品,其最高主频可达 130MIPS。

ARM7 系列微处理器的主要应用领域为工业控制、Internet 设备、网络和调制解调器设备、移动电话、PDA 等多种多媒体和嵌入式应用。

2) ARM9 系列微处理器

ARM9 系列采用 5 级指令流水线,能够运行在比 ARM7 更高的时钟频率上,改善处理器的整体性能。ARM9 的存储器系统根据哈佛体系结构重新设计,区分了数据总线和指令总线。

ARM9 系列的第一个处理器是 ARM920T,包含独立的数据指令 Cache 和 MMU。该处理器能够被用在要求有虚拟存储器支持的操作系统上。该系列包括 ARM9TDMI、ARM920T 和带有高速缓存处理器宏单元的 ARM940T。除了兼容 ARM7 系列,而且能够更加灵活地设计。

ARM9 系列微处理器主要应用于无线设备、引擎管理、仪器仪表、安全系统、机顶盒、汽车、通信和信息系统高端打印机、数字照相机和数字摄像机等。

3) ARM9E 系列微处理器

ARM9E 系列微处理器是 ARM9 内核带有 E 变种的一个可综合版本,使用单一的处理器内核提供了微控制器、DSP、Java 应用系统的解决方案,极大地减少芯片的面积和系统的复杂程度。ARM9E 系列微处理器提供了增强的 DSP 处理能力,很适合于那些需要同时使用 DSP 和微控制器的应用场合。

ARM9E 系列微处理器包含 ARM926EJ-S、ARM946E-S 和 ARM966E-S 3 种类型。

4) ARM10/10E 系列微处理器

ARM10E 系列微处理器具有高性能、低功耗的特点,由于采用新的体系结构和 6 级整数流水线,与同等的 ARM9 器件相比,在同样的时钟频率下,性能提高近 50%。同时,ARM10E 系列微处理器采用了两种先进的节能方式,使其功耗极低,且提供了 64 位的 Load/Store 体系,支持包括向量操作的、满足 IEEE 754 的浮点运算协处理器,系统集成更加方便。

ARM10 系列包括 ARM1020 和 ARM1020E 处理器核,其核心在于使用向量浮点(VFP)单元 VFP10 提供高性能的浮点解决方案,从而极大提高处理器的整型和浮点运算性能。ARM10 系列微处理器可以用于视频游戏机和高性能打印机等场合。

5) ARM11 系列微处理器

ARM1136J-S 发布于 2003 年,是针对高性能和高能效应而设计的。ARM1136J-S 是第一个执行 ARMv6 架构指令的处理器。它集成了一条具有独立的 Load/Store 和算术流水线的 8 级流水线。ARMv6 指令包含了针对媒体处理的单指令流多数据流扩展,采用特殊的设计改善视频处理能力。

ARM11 系列微处理器是 ARM 公司 2002 年推出的新一代 RISC 处理器,它是 ARMv6 指令架构的第一代设计实现。该系列主要有 ARM1136J、ARM1156T2 和 ARM1176JZ 3 个内核型号,分别针对不同应用领域。ARM11 系列微处理器内核适合新

一代消费类电子、无线设备、网络应用和汽车电子产品等需求。

6）SecurCore系列微处理器

SecurCore系列微处理器专为安全需要而设计，提供了完善的32位RISC技术的安全解决方案。SecurCore系列微处理器除具有ARM体系结构的低功耗、高性能的特点外，还具有其独特的优势，提供对安全解决方案的支持。

该系列涵盖SC100、SC110、SC200和SC210处理器核。该系列处理器主要针对新兴的安全市场，以一种全新的安全处理器设计为智能卡和其他安全IC开发提供独特的32位系统设计，并具有特定的反伪造方法，从而有助于防止对硬件和软件的盗版。

SecurCore系列微处理器除具有ARM体系结构的各种主要特点外，还在系统安全方面具有如下特点。

- 带有灵活的保护单元，以确保操作系统和应用数据的安全。
- 采用软内核技术，防止外部对其进行扫描探测。
- 可集成用户自己的安全特性和其他协处理器。

7）Xscale处理器

Xscale处理器是基于ARMv5TE体系结构的解决方案，是一款全性能、高性价比、低功耗的处理器。它支持16位的Thumb指令和DSP指令集，已使用在数字移动电话、个人数字助理和网络产品等场合。

Xscale处理器是Intel目前主要推广的一款ARM微处理器。

Intel Xscale体系架构是采用Intel Pentium技术实现的，是与ARMv5兼容的嵌入式微处理器架构。它提供全性能、高性价比、低功耗的解决方案，支持16位Thumb指令并集成数字信号处理(DSP)指令。

Xscale体系结构微控制器主频可高达1GHz，其设计目标是“面向特定应用的标准产品”，目前已经用于移动电话、PDA及网络设备中。

8）StrongARM系列微处理器

Intel StrongARM SA-1100处理器是采用ARM体系结构高度集成的32位RISC微处理器。它融合了Intel公司的设计和处理技术以及ARM体系结构的电源效率，采用在软件上兼容ARMv4体系结构、同时采用具有Intel技术优点的体系结构。

Intel StrongARM处理器是便携式通信产品和消费类电子产品的理想选择，已成功应用于多家公司的掌上电脑系列产品。

9）Cortex系列处理器

Cortex系列处理器发布于2004年，是基于ARMv7架构的，分为Cortex-M、Cortex-R和Cortex-A 3类。最早的型号是Cortex-M3，Cortex-A8内核于2005年10月4日发布，随后ARM在2006年5月15日发布Cortex-R4内核。

3.3 ARM Cortex-A8内核

ARM Cortex-A8处理器是第1款基于ARMv7架构的应用处理器，处理器的主频在600MHz～1GHz，既能满足低功耗移动设备的要求，又能满足需要高性能的消费类应用的要求。从高端特色手机到上网本、DTV、打印机和汽车信息娱乐，Cortex-A8处理器都

提供了可靠的高性能解决方案,在如今的终端设备中得到验证。

3.3.1 Cortex-A8 概述

图 3-7 为 ARM Cortex-A8 处理器内核,其主要特点和相关技术介绍如下。

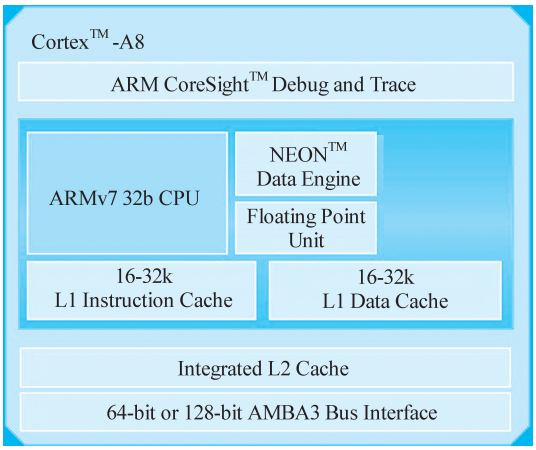


图 3-7 ARM Cortex-A8 处理器内核

1. ARMv7 架构规范

ARMv7 架构规范包括 32 位 ARM 指令集,并采用了实现更高的性能、能量效率和代码密度的 Thumb-2 技术,它是在 ARM 的 Thumb 代码压缩技术的基础上演进而来,并保持对当时 ARM 解决方案的代码兼容性。Thumb-2 技术比纯 32 位代码少使用 31% 的内存,减少了系统开销,同时能够提供比当时已有的基于 Thumb 技术的解决方案高出 38% 的性能。

2. NEON™ 数据引擎和 FP 单元

NEON 是一款 SIMD(单指令多数据)加速器处理器,其最大好处是如果想执行矢量操作,如视频编码/解码,则同时可以并行执行单精度浮点运算(浮点运算)。FP 是一个经典的浮点硬件加速器,并非并行架构,通常对一组输入执行一个操作并返回输出,其目的是加速浮点计算,它支持单精度浮点和双精度浮点。

采用 NEON 信号处理扩展,用于加速 H.264 和 MP3 等媒体编解码器,将 DSP 和媒体处理能力提高近 4 倍,并支持改良的浮点运算,能够满足 3D 图形、游戏物理应用以及传统嵌入式控制应用的需求。

NEON 单元包含一个 10 段 NEON 流水线,用于译码和执行高级 SIMD 多媒体指令集。NEON 单元包含以下几部分。

- NEON 指令队列。
- NEON 取数据队列。
- NEON 译码逻辑的两个流水线。
- 3 个用于高级 SIMD 整数指令的执行流水线。
- 2 个用于高级 SIMD 浮点数指令的执行流水线。
- 1 个用于高级 SIMD 和 FP 的存取指令的执行流水线。

- FP引擎，可完全执行FPv3数据处理指令集。

3. 专用的L2缓存和优化的L1缓存

L2缓存使用标准编译的RAM建立而成，64KB到2MB的可配置容量，其访问延时可编程控制；L1缓存经过性能和功耗的优化，结合最小访问延迟和散列确定方式，将性能最大化、功耗最小化。

4. Jazelle-Java加速

采用了高效支持预编译和即时编译Java及其他字节码语言的Jazelle®运行时编译目标(RCT)Java-加速技术，用于最优化即时(JIT)编译和动态自适应编译(DAC)，同时减少内存占用空间。Jazelle技术是ARM提供的组合型硬件和软件解决方案。ARM Jazelle技术软件是功能丰富的多任务Java虚拟机(JVM)，经过高度优化，可利用许多ARM处理器内核中提供的Jazelle技术体系结构扩展。

Cortex-A8处理器中与Jazelle扩展体系结构相关的寄存器有3个。

1) Jazelle标识寄存器(Jazelle Identity Register)

该寄存器为只读寄存器，在任何处理器模式和安全状态下均可访问，用于允许软件确认处理器是否实现了Jazelle扩展体系结构，正常情况下所读取的值为全零。

2) Jazelle主配置寄存器(Jazelle Main Configuration Register)

该寄存器用于控制Jazelle扩展体系结构的特征。

3) Jazelle OS控制寄存器(Jazelle OS Control Register)

该寄存器用于允许操作系统访问Jazelle扩展体系机构的硬件。

5. 13级超标量流水线

ARM Cortex-A8处理器复杂的流水线架构基于双对称的、顺序发射的、超标量13级主流水线，带有先进的动态分支预测，具有95%以上准确性。10级NEON媒体流水线，带有可编程的等待状态，以及基于全局历史的分支预测。结合功率优化的加载存储流水线，为功率敏感型应用提供2.0 DMIPS/MHz的速率。

6. AMBA总线

ARM研发的AMBA(Advanced Microcontroller Bus Architecture)提供一种特殊的机制，可将RISC处理器集成在其他IP芯核和外设中。

AMBA的4个版本如下。

AMBA 1：只有高级系统总线协议(Advanced System Bus，ASB)和高级外围总线协议(Advanced Peripheral Bus，APB)。

AMBA 2：引入高级高性能总线协议(Advanced High-performance Bus，AHB)，用于高速数据传输。

AMBA 3：为适应高吞吐量传输和调试引入高级可拓展接口(Advanced eXtensible Interface，AXI)和高级跟踪总线(Advanced Trace Bus，ATB)，而AHB协议缩减为AHB-lite，APB协议增加了PREADY和PSLVERR，ASB由于设计复杂而不再使用；AXI是系统总线的主要接口，64位或128位，用于执行L2 Cache的填充和L1 Cache指令和数据的访问。AXI总线时钟和CLK输入同步，可以通过ACLKEN信号使能。

AMBA 4：AXI得到了增强，引入QOS和long burst的支持，根据应用不同可选AXI4、AXI4-lite、AXI4-stream，同时为满足复杂SoC的操作一致性引入ACE(AXI

Coherency Extensions)和 ACE-lite 协议,APB 和 ATB 也同时得到增强。

7. CoreSight 片上调试和跟踪

随着 SoC 时代的来临和 Cache 的广泛使用,在处理器调试过程中片外仪器难以测量片内数据流和指令流,不得不以硅片面积为代价解决处理器运行状态的实时观测问题。越来越多的处理器厂商开始提供硬件片上 Trace 功能,片上 Trace 系统通过专用硬件非入侵地实时记录程序执行路径和数据读/写等信息,压缩成 Trace 数据流后,通过专用数据通道,输出端口传输至调试主机。该主机中的开发工具解压缩 Trace 数据流,恢复程序运行信息以供调试和性能分析。

1) CoreSight 介绍

图 3-8 是 CoreSight 系统的一个示例,包含两个 ARM 内核,一个 DSP,以及众多的 CoreSight 组件,实现对内核和 DSP 的 Debug 和 Trace 功能。

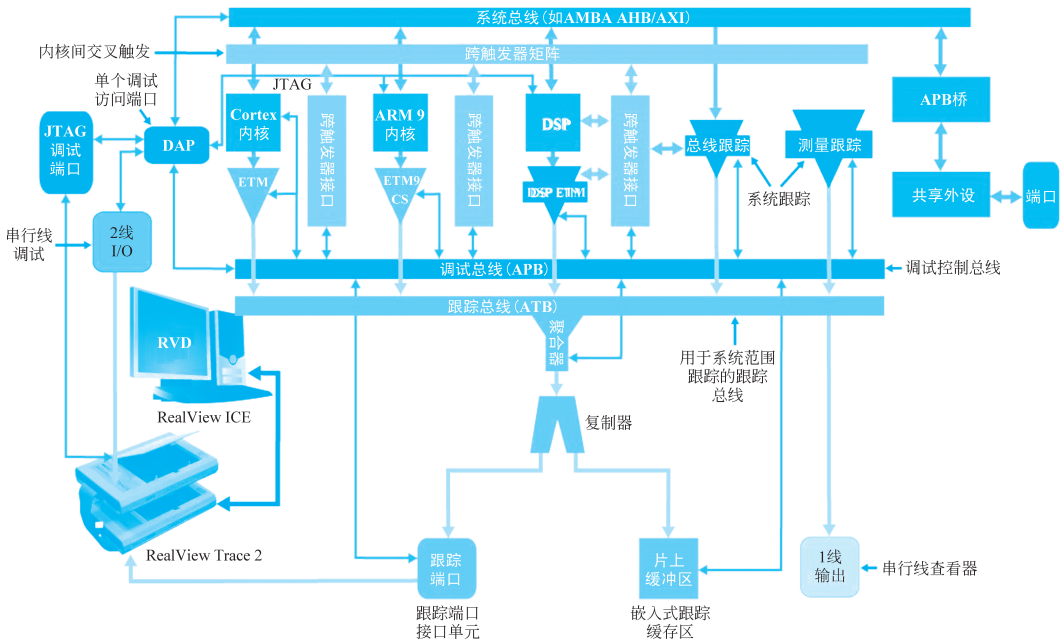


图 3-8 CoreSight 系统

ARM 的 CoreSight 是目前业界领先的多核片上 Trace 解决方案。CoreSight 体系结构非常灵活,其中各个部件可以根据不同处理器厂商的需要而进行组合,实现 Debug 和 Trace 的架构。该架构包含多个 CoreSight 组件。众多的 CoreSight 组件,构成了一个 CoreSight 系统。可以根据 CoreSight 架构,实现自己的 CoreSight 组件。每个 CoreSight 的组件,都要遵循 CoreSight 架构的要求。

2) CoreSight 组成

一个典型的 CoreSight 调试结构组成部分如下。

(1) 控制访问部件

控制访问部件配置和控制数据流的产生,但是不产生数据流。

DAP(Debug Access Port):可以实时访问 AMBA 总线上的系统内存,外设寄存器,以及所有调试配置寄存器,而不需挂起系统。

ECT(Embedded Cross Trigger)：包括CTI(Cross Trigger Interface)和CTM(Cross Trigger Matrix)，为ETM(Embedded Trace Macrocell)提供接口，用于将一个处理器的调试事件传递给另一个处理器。

(2) 源部件

源部件用于产生向ATB(AMBA Trace Bus)发送的跟踪数据，一般是APB总线。

HTM(AHB Trace Macrocell)：用于获取AHB总线跟踪信息，包括总线的层次、存储结构、数据流和控制流。

ETM(Embedded Trace Macrocell)：用于获取处理器核的跟踪信息。

(3) 汇聚点

汇聚点是芯片上跟踪数据的终点。

TPIU(Trace Port Interface Unit)：将片内各种跟踪数据源获取的信息按照TPIU帧的格式进行组装，然后通过Trace Port传送到片外。

ETB(Embedded Trace Buffer)：一个32位的RAM，作为片内跟踪信息缓冲区。

SWO(Serial Wire Output)：类似TPIU，但仅输出ITM单元的跟踪信息，只需要一个引脚。

使用Trace Port接口进行调试还需要专用的跟踪器，ARM公司的开发工具RVDS中RVT(RealView Tracer)就是这种跟踪器。仿真器RealView ICE(In-Circuit Emulator)是一种基于JTAG的调试解决方案。

Cortex-A8处理器的主频在600MHz～1GHz，能够满足那些需要工作在300mW以下功耗优化的移动设备的要求。

3.3.2 Cortex-A8处理器模式和状态

1. 处理器模式

Cortex-A8体系结构支持8种处理器模式，如表3-2所示。

表3-2 Cortex-A8内核的8种处理器模式

处理器模式			缩写	说明	备注
		用户	usr	正常程序执行模式	不能直接切换到其他模式
特权模式		系统	sys	运行特权操作系统任务	与用户模式类似，但拥有可以直接切换到其他模式等特权
	异常模式	管理	svc	操作系统保护模式	系统复位或软件中断时进入
		数据中止	abt	实现虚拟存储器或存储器保护	当存取异常时进入
		未定义	und	支持硬件协处理器的软件仿真	未定义指令异常响应时进入
		中断	irq	用于通用中断处理	IRQ异常响应时进入
		快中断	fiq	支持高速数据传送或通道处理	FIQ异常响应时进入
		安全监视	Mon	当处理器支持Security Extensions时才会使用该模式	可以在安全模式和非安全模式下转换

除用户模式外，其他模式均为特权模式。ARM内部寄存器和一些片内外设在硬件

设计上只允许(或者可选为只允许)特权模式下访问。此外,特权模式可以自由地切换处理器模式,而用户模式不能直接切换到别的模式。

管理、数据中止、未定义、中断和快中断 5 种模式称为异常模式。它们除可以通过程序切换进入外,也可以由特定的异常进入。当特定的异常出现时,处理器进入相应的模式。每种异常模式都有一些独立的寄存器,以避免异常退出时用户模式的状态不可靠。

用户模式和系统模式都不能由异常进入,而且它们使用完全相同的寄存器组。

系统模式是特权模式,不受用户模式的限制。操作系统在该模式下访问用户模式的寄存器就比较方便,而且操作系统的一些特权任务可以使用这个模式访问一些受控的资源。

2. 处理器状态

Cortex-A8 处理器有 3 种操作状态,这些状态由 CPSR 寄存器的 T 位和 J 位控制。

1) ARM 状态

该状态下,Cortex-A8 处理器执行 32 位字对齐的 ARM 指令,T 位和 J 位都为 0。

2) Thumb 状态

该状态下,Cortex-A8 处理器执行 16 位或 32 位半字对齐的 Thumb2 指令,T 位为 1,J 位都为 0。

3) ThumbEE 状态

该状态下,Cortex-A8 处理器执行为动态产生目标而设计的 16 位或 32 位半字对齐的 Thumb2 指令集的变体。T 位和 J 位都为 1。

处理器的操作状态可以在以下几种状态间转换。

ARM 状态和 Thumb 状态之间转换使用 BL 和 BLX 指令,并加载到 PC。

Thumb 状态和 ThumbEE 状态之间转换使用 ENTERX 和 LEAVEX 指令。

异常会导致处理器进入 ARM 状态或 Thumb 状态。一般情况下,当退出异常处理时,处理器会恢复原来的 T 位和 J 位的值。


Cortex-A8 寄存器组织

3.3.3 Cortex-A8 寄存器组织

CPU 寄存器是直接内置于 CPU 的小容量快速存储器,用于操作数据。ARM 的 CPU 都是 Load/Store 架构,这意味着 CPU 上进行的计算都直接作用在寄存器上。首先 CPU 在进行计算之前从主存储器中读取数据到寄存器中,然后可能再将计算结果写回主存储器。没有任何指令操作直接作用于主存储器上的值,这避免了每次操作之后都要往主存储器中写数据的过程,另外也简化了流水线,这对 RISC 处理器来说是十分关键的。Cortex-A8 处理器如图 3-9 所示。

Cortex-A8 处理器共有 40 个 32 位长的寄存器,其中包括 33 个通用寄存器和 7 个状态寄存器。7 个状态寄存器包括 1 个当前程序状态寄存器(Current Programs Status Register,CPSR)和 6 个备份程序状态寄存器(Saved Programs Status Register,SPSR)。仔细分析,大多数例程都可以用这 40 个寄存器创建,ARM 处理器比其他处理器具有更多的寄存器。

1. 命名规则

ARM-Thumb 过程调用标准(ARM-THUMB procedure call standard,ATPCS)规定