ARM

白皮书

ARM Cortex-M 处理器入门

ARM Cortex-M 处理器家族介绍和比较

Joseph Yiu, 高级嵌入式技术经理, ARM

三月 2017

概要
ARM Cortex-M 处理器家族现在有 8 款处理器成员。在本文中，我们会比较 Cortex-M 系列处理器之间的产品特性，重点讲述如何根据产品应用选择正确的 Cortex-M 处理器。本文中会详细的对照 Cortex-M 系列处理器的指令集和高级中断处理能力，以及 SoC 系统级特性，调试和追踪功能和性能的比较。

1 简介

今天， ARM Cortex-M 处理器家族有 8 款处理器成员。除此之外，ARM 的产品系列还有很多其他的处理器成员。对很多初学者，甚至某些芯片设计经验丰富但是不熟悉 ARM 系列处理器的设计者来说，也是很容易混淆这些产品的。不同的 ARM 处理器有不同的指令集，系统功能和性能。本文会深入展现 Cortex-M 系列处理器之间的关键区别，以及它们和 ARM 其他系列处理器的不同。

1.1 ARM 处理器家族

多年来， ARM 已经研发了相当多的不同的处理器产品。 如下图中（图 1）: ARM 处理器产品分为经典 ARM 处理器系列和最新的 Cortex 处理器系列。并且根据应用范围的不同，ARM 处理器可以分类成 3 个系列。

Application Processors（应用处理器）–面向移动计算，智能手机，服务器等市场的的高端处理器。这类处理器运行在很高的时钟频率（超过 1GHz），支持像 Linux，Android，MS Windows 和移动操作系统等完整操作系统需要的内存管理单元（MMU）。 如果规划开发的产品需要运行上述其中的一个操作系统，你需要选择 ARM 应用处理器.

Real-time Processors （实时处理器）–面向实时应用的高性能处理器系列，例如硬盘控制器，汽车传动系统和无线通讯的基带控制。多数实时处理器不支持 MMU，不过通常具有 MPU、Cache 和其他针对工业应用设计的存储器功能。实时处理器运行在比较高的时钟频率（例如 200MHz 到 >1GHz ），响应延迟非常低。虽然实时处理器不能运行完整版本的 Linux 和 Windows 操作系统， 但是支持大量的实时操作系统（RTOS）。

Microcontroller Processors（微控制器处理器）–微控制器处理器通常设计成面积很小和能效比很高。通常这些处理器的流水线很短，最高时钟频率很低（虽然市场上有此类的处理器可以运行在 200Mhz 之上）。 并且，新的 Cortex-M 处理器家族设计的非常容易使用。因此，ARM 微控制器处理器在单片机和深度嵌入式系统市场非常成功和受欢迎。

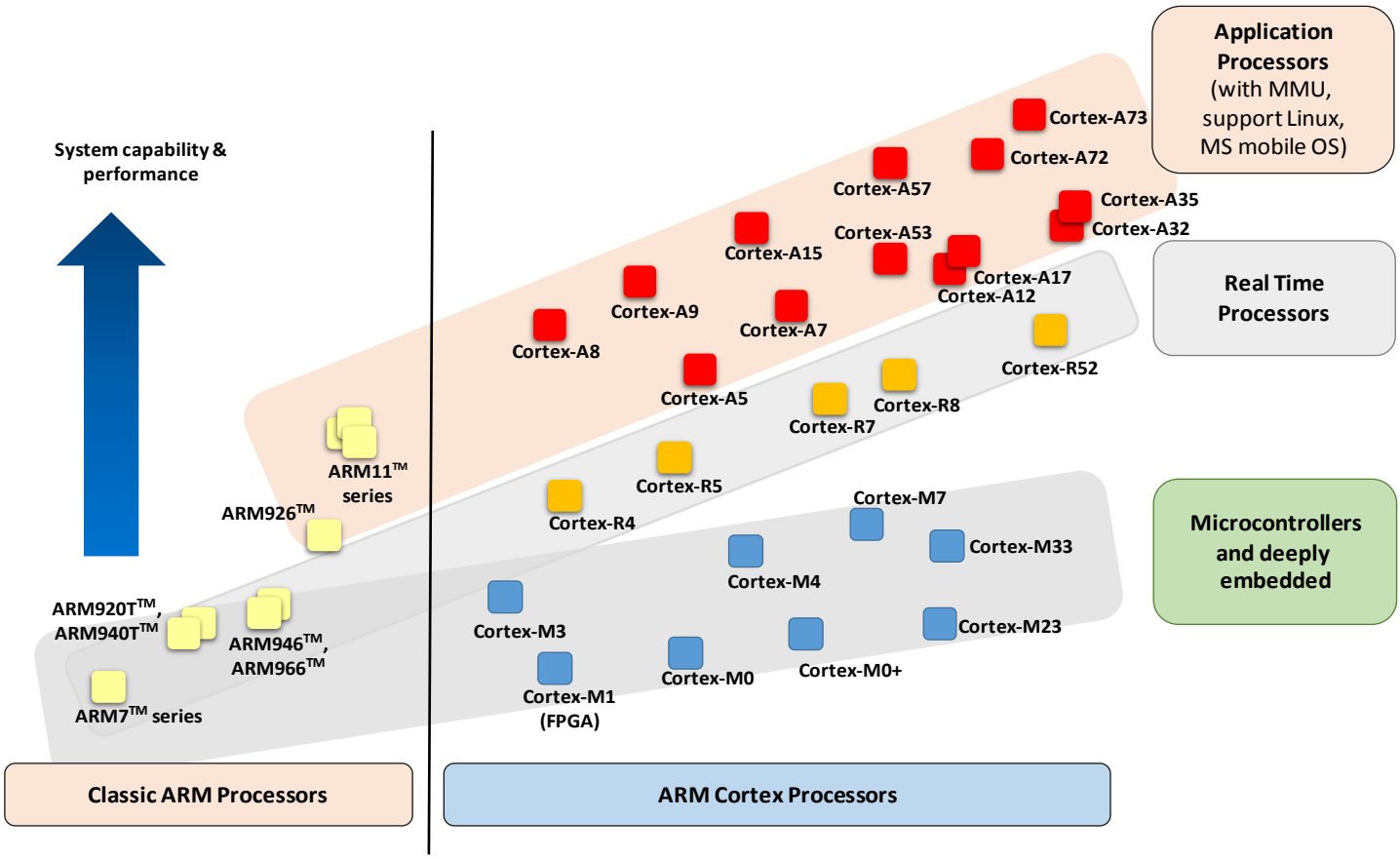

图 1: 处理器家族

表 1 总结了三个处理器系列的主要特征。

	Application processors	Real-time processors	Microcontroller processors
设计特点	高时钟频率，长流水线，高性能，对媒体处理支持（NEON 指令集扩展），	高时钟频率，较长的流水线，高确定性（中断延迟低）	通常较短的流水线，超低功耗
系统特性	内存管理单元 (MMU), cache memory, ARM TrustZone® 安全扩展	内存保护单元 (MPU), cache memory, 紧耦合内存 (TCM)	内存保护单元(MPU), 嵌套向量中断控制器 (NVIC), 唤醒中断控制器 (WIC), 最新 ARM TrustZone® 安全扩展.
目标市场	移动计算，智能手机，高能效服务器，高端微处理器	工业微控制器，汽车电子，硬盘控制器，基带	微控制器，深度嵌入系统（例如，传感器，MEMS，混合信号 IC, IoT）

表 1: 处理器特性总结

1.2 Cortex-M 处理器家族

Cortex-M 处理器家族更多的集中在低性能端，但是这些处理器相比于许多微控制器使用的传统处理器性能仍然很强大。例如，Cortex-M4 和 Cortex-M7 处理器应用在许多高性能的微控制器产品中，最大的时钟频率可以达到 400Mhz。

当然，性能不是选择处理器的唯一指标。在许多应用中，低功耗和成本是关键的选择指标。因此，Cortex-M 处理器家族包含各种产品来满足不同的需求：

处理器	描述
Cortex-M0	面向低成本，超低功耗的微控制器和深度嵌入应用的非常小的处理器（最小 12K 门电路）
Cortex-M0+	针对小型嵌入式系统的最高能效的处理器，与 Cortex-M0 处理器接近的尺寸大小和编程模式，但是具有扩展功能，如单周期 I/O 接口和向量表重定位功能
Cortex-M1	针对 FPGA 设计优化的小处理器，利用 FPGA 上的存储器块实现了紧耦合内存（TCM）。和 Cortex-M0 有相同的指令集
Cortex-M3	针对低功耗微控制器设计的处理器，面积小但是性能强劲，支持可以处理器快速处理复杂任务的丰富指令集。具有硬件除法器和乘加指令（MAC）.并且，M3 支持全面的调试和跟踪功能，使软件开发者可以快速的开发他们的应用
Cortex-M4	不但具备 Cortex-M3 的所有功能，并且扩展了面向数字信号处理（DSP）的指令集，比如单指令多数据指令（SMID）和更快的单周期 MAC 操作。此外，它还有一个可选的支持 IEEE754 浮点标准的单精度浮点运算单元
Cortex-M7	针对高端微控制器和数据处理密集的应用开发的高性能处理器。具备 Cortex-M4 支持的所有指令功能，扩展支持双精度浮点运算，并且具备扩展的存储器功能，例如 Cache 和紧耦合存储器（TCM）
Cortex-M23	面向超低功耗，低成本应用设计的小尺寸处理器，和 Cortex-M0 相似，但是支持各种增强的指令集和系统层面的功能特性。M23 还支持 TrustZone 安全扩展
Cortex-M33	主流的处理器设计，与之前的 Cortex-M3 和 Cortex-M4 处理器类似，但系统设计更灵活，能耗比更高效，性能更高。M33 还支持 TrustZone 安全扩展

表 2: Cortex-M 处理器家族

不同于老的经典 ARM 处理器（例如，ARM7TDMI, ARM9）， Cortex-M 处理器有一个非常不同的架构。例如：

- 仅支持 ARM Thumb®指令，已扩展到同时支持 16 位和 32 位指令 Thumb-2 版本
- 内置的嵌套向量中断控制负责中断处理，自动处理中断优先级，中断屏蔽，中断嵌套和系统异常处理。
- 中断处理函数可以使用标准的 C 语言编程，嵌套中断处理机制避免了使用软件判断哪一个中断需要响应处理。同时，中断响应速度是确定性的，低延迟的
- 向量表从跳转指令变为中断和系统异常处理函数的起始地址。
- 寄存器组和某些编程模式也做了改变。

这些变化意味着许多为经典 ARM 处理器编写的汇编代码需要修改，老的项目需要修改和重新编译才能迁移到 Cortex-M 的产品上。软件移植具体的细节记录在 ARM 文档：

ARM Cortex-M3 Processor Software Development for ARM7TDMI Processor Programmers
http://www.arm.com/files/pdf/Cortex-M3_programming_for_ARM7_developers.pdf

1.3 Cortex-M 系列处理器的共同特性

Cortex-M0, M0+, M3, M4 and M7 之间有很多的相似之处，例如：

- 基本编程模型 (章节 **Error! Reference source not found.**)
- 嵌套向量中断控制器（NVIC）的中断响应管理
- 架构设计的休眠模式：睡眠模式和深度睡眠模式 (章节 **Error! Reference source not found.**)
- 操作系统支持特性 (章节 **Error! Reference source not found.**)
- 调试功能 (章节 **Error! Reference source not found.**)
- 易用性

例如，嵌套向量中断控制器是内置的中断控制器

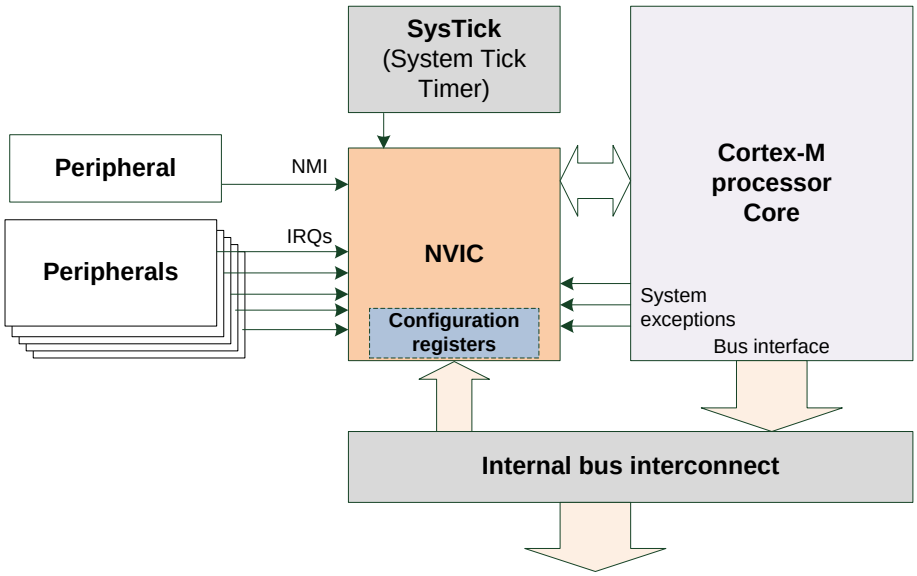

图 2: Cortex-M 处理器的嵌套向量中断控制器

支持许多外围设备的中断输入，一个不可屏蔽的中断请求，一个来自内置时钟（SysTick）的中断请求（见章节 3.3）和一定数量的系统异常请求。NVIC 处理这些中断和异常的优先级和屏蔽管理。

NVIC 以及异常处理模型的更多的内容在章节 3.2 描述。其他 Cortex-M 处理器间的异同点会在本文的其余部分讲解。

2 Cortex-M 处理器指令集

2.1 指令集简介

大多数情况下，应用程序代码可以用 C 或其他高级语言编写。但是，对 Cortex-M 处理器支持指令集的基本了解有助于开发者针对具体应用选择合适的 Cortex-M 处理器。指令集（ISA）是处理器架构的一部分，Cortex-M 处理器可以分为几个架构规范

架构	描述
ARMv6-M	Cortex-M0, Cortex-M0+ 和 Cortex-M1 支持的架构
ARMv7-M	Cortex-M3, Cortex-M4 和 Cortex-M7 支持的架构.ARMv7-M 扩展的 DSP 类型指令（SMID）也被称为 ARMv7E-M.
ARMv8-M	这个架构进一步分为： Baseline 子规范– Cortex-M23 对应的架构 Mainline 子规范– Cortex-M33 对应的架构 For more information about ARMv8-M architecture, please see in ARMv8-M 关于 ARMv8-M 架构的更多信息,请参见 ARMv8-M 的 ARMv8-M Architecture Technical Overview https://community.arm.com/docs/DOC-10896

表 3: Cortex-M 处理器 ARM 架构规范的规范

所有的 Cortex-M 处理器都支持 Thumb 指令集。整套 Thumb 指令集扩展到 Thumb-2 版本时变得相当大。但是，不同的 Cortex-M 处理器支持不同的 Thumb 指令集的子集，如图 3 所示

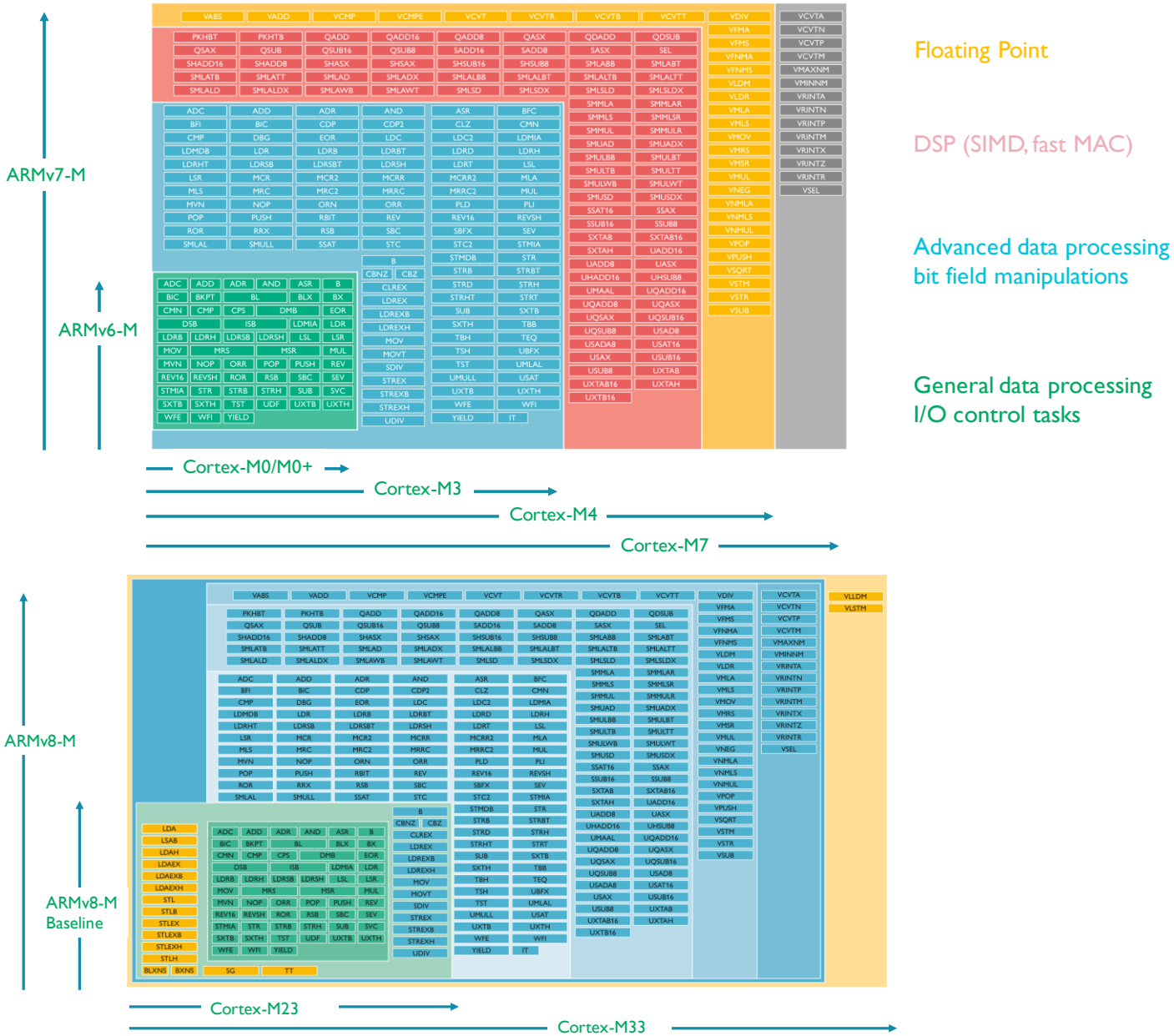

图 3: Cortex-M 处理器的指令集

2.2 Cortex-M0/M0+/M1 指令集

Cortex-M0/M0+/M1 处理器基于 ARMv6-M 架构。这是一个只支持 56 条指令的小指令集，大部分指令是 16 位指令，如图 3 所示只占很小的一部分。但是，此类处理器中的寄存器和处理的数据长度是 32 位的。对于大多数简单的 I/O 控制任务和普通的数据处理，这些指令已经足够了。这么小的指令集可以用很少的电路门数来实现处理器设计，Cortex-M0 和 Cortex-M0+最小配置仅仅 12K 门。然而，其中的很多指令无法使用高位寄存器（R8 到 R12），并且生成立即数的能力有限。这是平衡了超低功耗和性能需求的结果。

2.3 Cortex-M3 指令集

Cortex-M3 处理器是基于 ARMv7-M 架构的处理器，支持更丰富的指令集，包括许多 32 位指令，这些指令可以高效的使用高位寄存器。另外，M3 还支持：

- 查表跳转指令和条件执行（使用 IT 指令）
- 硬件除法指令
- 乘加指令（MAC）
- 各种位操作指令

更丰富的指令集通过几种途径来增强性能；例如，32 位 Thumb 指令支持了更大范围的立即数，跳转偏移和内存数据范围的地址偏移。支持基本的 DSP 操作（例如，支持若干条需要多个时钟周期执行的 MAC 指令，还有饱和运算指令）。最后，这些 32 位指令允许用单个指令对多个数据一起做桶型移位操作。

支持更丰富的指令导致了更大的面积成本和更高的功耗。典型的微控制器，Cortex-M3 的电路门数是 Cortex-M0 和 Cortex-M0+两倍还多。但是，处理器的面积只是大多数现代微控制器的很小的一部分，多出来的面积和功耗经常不那么重要。

2.4 Cortex-M4 指令集

Cortex-M4 在很多地方和 Cortex-M3 相同：流水线，编程模型。Cortex-M4 支持 Cortex-M3 的所有功能，并额外支持各种面向 DSP 应用的指令，像 SIMD, 饱和运算指令，一系列单周期 MAC 指令（Cortex-M3 只支持有限条 MAC 指令，并且是多周期执行的），和可选的单精度浮点运算指令。

Cortex-M4 的 SIMD 操作可以并行处理两个 16 位数据和 4 个 8 位数据。例如，图 4 展示的 QADD8 和 QADD16 操作：

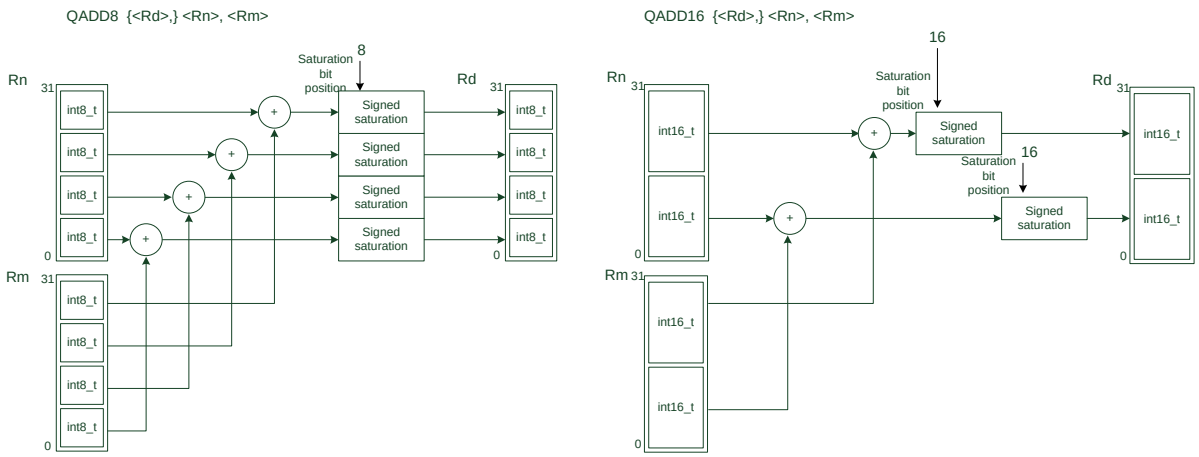

图 4: SIMD 指令例子: QADD8 and QADD16

The uses of SIMD enable much faster computation of 16-bit and 8-bit data in certain DSP operations as the calculation can be parallelized. However, in general programming, C compilers are unlikely to utilize the SIMD capability. That is why the typical benchmark results of the Cortex-M3 and Cortex-M4. However, the internal data path of the Cortex-M4 is different from Cortex-M3, which enable faster operations in a few cases (e.g. single cycle MAC, and allow write back of two registers in a single cycle).在某些 DSP 运算中，使用 SIMD 可以加速更快计算 16 位和 8 位数据，因为这些运算可以并行处理。但是，一般的编程中，C 编译器并不能充分利用 SIMD 运算能力。这是为什么 Cortex-M3 和 Cortex-M4 典型 benchmark 的分数差不多。然而，Cortex-M4 的内部数据通路和 Cortex-M3 的不同，某些情况下 Cortex-M4 可以处理的更快（例如，单周期 MAC，可以在一个周期中写回到两个寄存器）。

2.5 Cortex-M7 指令集

Cortex-M7 支持的指令集和 Cortex-M4 相似，添加了：

- 浮点数据架构是基于 FPv5 的，而不是 Cortex-M4 的 FPv4，所以 Cortex-M7 支持额外浮点指令
- 可选的双精度浮点数据处理指令
- 支持缓存数据预取指令（PLD）

Cortex-M7 的流水线和 Cortex-M4 的非常不同。Cortex-M7 是 6 级双发射流水线，可以获得更高的性能。多数为 Cortex-M4 设计的软件可以直接运行在 Cortex-M7 上。但是，为了充分利用流水线差异来达到最好的优化，软件需要重新编译，并且在许多情况下，软件需要一些小的升级，以充分利用像 Cache 这样的新功能。

2.6 Cortex-M23 指令集

Cortex-M23 的指令集是基于 ARMv8-M 的 Baseline 子规范，它是 ARMv6-M 的超集。扩展的指令包括：

- 硬件除法指令
- 比较和跳转指令，32 位跳转指令
- 支持 TrustZone 安全扩展的指令
- 互斥数据访问指令（通常用于信号量操作）

- 16 位立即数生成指令
- 载入获取及存储释放指令（支持 C11）

在某些情况下，这些增强的指令集可以提高处理器性能，并且对包含多个处理器的 SoC 设计有用（例如，互斥访问对多处理器的信号量处理有帮助）

2.7 | Cortex-M33 指令集

因为 Cortex-M33 设计是非常可配置的，某些指令也是可选的。例如：

- DSP 指令（Cortex-M4 和 Cortex-M7 支持的）是可选的
- 单精度浮点运算指令是可选的，这些指令是基于 FPv5 的，并且比 Cortex-M4 多几条。

: Cortex-M33 也支持那些 ARMv8-M Mainline 引入的新指令：

- 支持 TrustZone 安全扩展的指令
- 载入获取及存储释放指令（支持 C11）

2.8 指令集特性比较总结

ARMv6-M, ARMv7-M 和 ARMv8-M 架构有许多指令集功能特点， 很难介绍到所有的细节。但是，下面的表格（表 4）总结了那些关键的差异。

	Cortex-M0/M0+	Cortex-M1	Cortex-M3	Cortex-M4	Cortex-M7	Cortex-M23	Cortex-M33
Architecture	ARMv6-M	ARMv6-M	ARMv7-M	ARMv7E-M	ARMv7E-M	ARMv8-M Baseline	ARMv8-M Mainline
v4T,v5T, v6-M Thumb ISA	Y	Y	Y	Y	Y	Y	Y
v7-M Thumb ISA	-	-	Y	Y	Y	-	Y
Low power / Sleep mode : WFE, WFI, SEV	Y	Execute as NOP	Y	Y	Y	Y	Y
Single cycle Multiply (32-bit result)	Y	Y	Y	Y	Y	Y	Y
Bit field processing	-	-	Y	Y	Y	-	Y
Hardware divide (integer)	-	-	Y	Y	Y	Y	Y
Unaligned data access	-	-	Y	Y	Y	-	Y
Table branch	-	-	Y	Y	Y	-	Y
Conditional execution (IT)	-	-	Y	Y	Y	-	Y
Compare & branch (CBZ, CBNZ)	-	-	Y	Y	Y	Y	Y
Floating point	-	-	-	Single	Single	-	Single

				precision (optional)	precision / Single + double precision (optional)		precision (optional)
MAC	-	-	Y (multi-cycle, limited)	Y (single cycle)	Y (single cycle)	-	Y (single cycle)
SIMD	-	-	-	Y	Y	-	Y
Saturation	-	-	USAT, SSAT only	Y	Y	-	Y
Exclusive access	-	-	Y	Y	Y	Y	Y
Load acquire, store release	-	-	-	-	-	Y	Y
Memory barrier	Y	Y	Y	Y	Y	Y	Y
SVC	Y	Optional	Y	Y	Y	Y	Y
TrustZone support	-	-	-	-	-	Y	Y

表 4: 指令集特性总结

Cortex-M 处理器指令集的最重要的特点是向上兼容。Cortex-M3 的指令是 Cortex-M0/M0+/M1 的超集。所以，理论上讲，如果存储空间分配是一致的，运行在 Cortex-M0/M0+/M1 上的二进制文件可以直接运行在 Cortex-M3 上。同样的原理也适用于 Cortex-M4/M7 和其他的 Cortex-M 处理器；Cortex-M0/M0+/M1/M3 支持的指令也可以运行在 Cortex-M4/M7 上。

虽然 Cortex-M0/M0+/M1/M3/M23 处理器没有浮点运算单元配置选项，但是处理器仍然可以利用软件来做浮点数据运算。这也适用于基于 Cortex-M4/M7/M33 但是没有配置浮点运算单元的产品。在这种情况下，当程序中使用了浮点数，编译工具包会在链接阶段插入需要的运行软件库。软件模式的浮点运算需要更长的运行时间，并且会略微的增加代码大小。但是，如果浮点运算不是频繁使用的，这种方案是适合这种应用的。

3 架构特性

3.1 编程模型

Cortex-M 处理器家族的编程模型是高度一致的。例如所有的 Crotex-M 处理器都支持 R0 到 R15，PSR, CONTROL 和 PRIMASK。两个特殊的寄存器— FAULTMASK 和 BASEPRI—只有 Cortex-M3, Cortex-M4, Cortex-M7 和 Cortex-M33 支持；浮点寄存器组和 FPSCR（浮点状态和控制寄存器）寄存器，是 Cortex-M4/M7/M33 可选的浮点运算单元使用的。

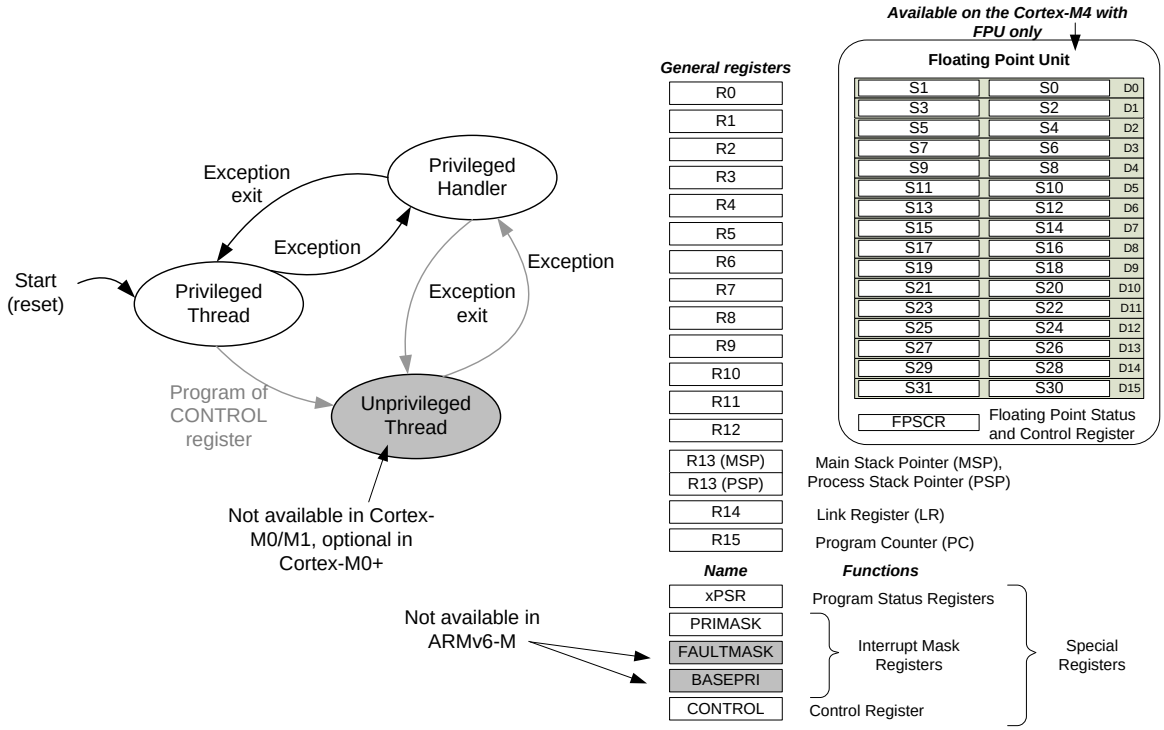

图 5: 编程模型

BASEPRI 寄存器允许程序阻止指定优先级或者低的优先级中断和异常。对 ARMv7-M 来说这是很重要的，因为 Cortex-M3, Cortex-M4, Cortex-M7 和 Cortex-M33 有大量的优先级等级，而 ARMv6-M 和 ARMv8-M Baseline 只有有限的 4 个优先等级。FAULTMASK 通常用在复杂的错误处理上（查看章节 3.4）

非特权级别的实现对 ARMv6-M 处理器是可选的，对 ARMv7-M 和 ARMv8-M 处理器一直支持的。对 Cortex-M0+ 处理器，它是可选的, Cortex-M0 and Cortex-M1 不支持这个功能。这意味着在各种 Cortex-M 处理器的 CONTROL 寄存器是稍微不同的。FPU 的配置也会影响到 CONTROL 寄存器，如图 6 所示。

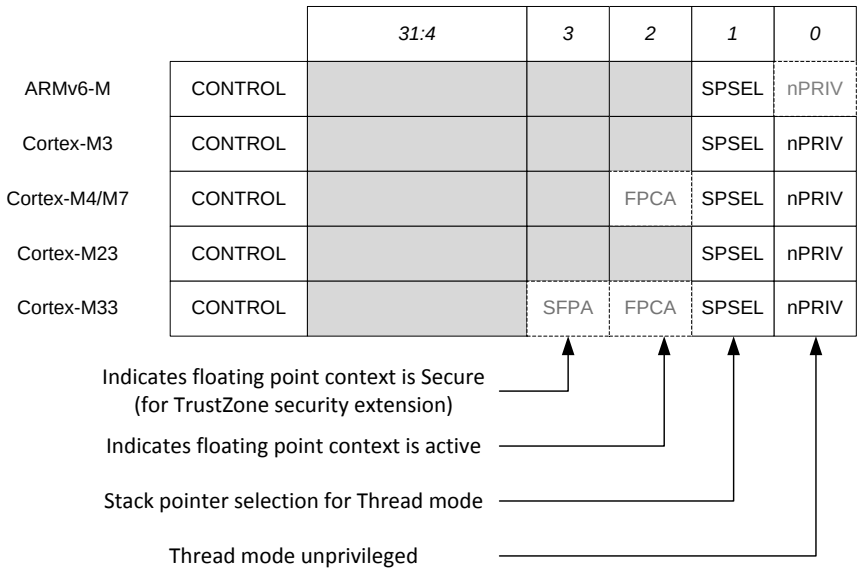

图 6: CONTROL 寄存器

另外一个编程模型之间的不同是 PSR 寄存器(程序状态寄存器)的细节。所有的 Cortex-M 处理器，PSR 寄存器都被再分成应用程序状态寄存器(APSR)，执行程序状态寄存器(EPSR)和中断程序状态寄存器(IPSR)。 ARMv6-M 和 ARMv8-M Baseline 系列的处理器不支持 APSR 的 Q 位和 EPSR 的 ICI/IT 位。ARMv7E-M 系列 (Cortex-M4, Cortex-M7) 和 ARMv8-M Mainline (配置了 DSP 扩展的 Cortex-M33)支持 GE 位。另外，ARMv6-M 系列处理器 IPSR 的中断号数字范围很小，如图 7 所示。

Exception Number

	31	*30*	*29*	*28*	*27*	*26:25*	*24*	*23:20*	*19:16*	*15:10*	*9*	*8*	*7*	*6*	*5*	*4:0*
ARMv6-M (Cortex-M0/M0+)	N	Z	C	V			T									
ARMv7-M (Cortex-M3)	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				
ARMv7E-M (Cortex-M4/M7)	N	Z	C	V	Q	ICI/IT	T		GE[3:0]	ICI/IT		Exception Number				
ARMv8-M Baseline (Cortex-M23)	N	Z	C	V			T					Exception Number				
ARMv8-M Mainline (Cortex-M33)	N	Z	C	V	Q	ICI/IT	T		GE[3:0]	ICI/IT		Exception Number				

图 7: PSR 差异

请注意 Cortex-M 的编程模型和 ARM7TDMI 等这些经典的 ARM 处理器是不一样的。除了寄存器组不同外，经典 ARM 处理器中“模式”和“状态”的定义与 Cortex-M 中的也是不同的。Cortex-M 只有两个模式：线程模式（Thread）和管理者模式（Handler），并且 Cortex-M 处理器一直运行在 Thumb 状态（不支持 ARM 指令）

3.2 异常处理模型和嵌套向量中断控制器 NVIC

所有的 Cortex-M 处理器都包含了 NVIC 模块，采用同样的异常处理模型。如果一个异常中断发生，它的优先等级高于当前运行等级，并且没有被任何的中断屏蔽寄存器屏蔽，处理器会响应这个中断/异常，把某些寄存器入栈到当前的堆栈上。这种堆栈机制下，中断处理程序可以编写为一个普通的 C 函数，许多小的中断处理函数可以立即直接响应工作而不需要额外的堆栈处理花销。

一些 ARMv7-M/ARMv8-M Mainline 系列的处理器使用的中断和系统异常并不被 ARMv6-M/ARMv8-M Baseline 的产品支持，如图 8. 例如，Cortex-M0, M0+ 和 M1 的中断数被限制在 32 个以下，没有调试监测异常，错误异常也只限于 HardFault（错误处理细节请参看章节 3.4）。相比之下，Cortex-M23, Cortex-M3, Cortex-M4 和 Cortex-M7 处理器可以支持到多达 240 个外围设备中断。Cortex-M33 支持最多 480 个中断。

另外一个区别是可以使用的优先等级数量：

ARMv6-M 架构 - ARMv6-M 支持 2 级固定的（NMI 和 HardFault）和 4 级可编程的（由每个优先等级寄存器的两个位表示）中断/异常优先级。这对大多数的微控制器应用来说足够了。

ARMv7-M 架构 - ARMv7-M 系列处理器的可编程优先级等级数范围，根据面积的限制，可以配置成 8 级（3 位）到 256 级（8 位）。ARMv7-M 处理器还有一个叫做中断优先级分组的功能，可以把中断优先级寄存器再进一步分为组优先级和子优先级，这样可以详细地制定抢占式优先级的行为。

ARMv8-M Baseline – 类似 ARMv6-M，M23 也有 2 位的优先级等级寄存器。借助可选的 TrustZone 安全扩展组件，安全软件可以把非安全环境中的中断的优先等级转换到优先等级区间的下半区，这就保证了安全环境中的某些中断/异常总是比非安全环境中的优先级要高。

ARMv8-M Mainline – 类似于 ARMv7-M。 可以支持 8 到 256 个中断优先等级和中断优先级分组。还支持 ARMv8-M Baseline 具有的优先等级调整功能。

Exception Type	ARMv6-M (Cortex-M0/M0+/M1)	ARMv7-M (Cortex-M3/M4/M7)	ARMv8-M Baseline (Cortex-M23)	ARMv8-M Mainline (Cortex-M33)	Vector Table	Vector address offset (initial)
495		Not supported in Cortex-M3/M4/M7	Not supported in Cortex-M23	Device Specific Interrupts	Interrupt#479 vector [1]	0x000007BC
256						
255		Device Specific Interrupts	Device Specific Interrupts		Interrupt#239 vector [1]	0x000003FC
31	Device Specific Interrupts				Interrupt#31 vector [1]	0x000000BC
17					Interrupt#1 vector [1]	0x00000044
16					Interrupt#0 vector [1]	0x00000040
15	SysTick	SysTick	SysTick	SysTick	SysTick vector [1]	0x0000003C
14	PendSV	PendSV	PendSV	PendSV	PendSV vector [1]	0x00000038
13	Not used	Not used	Not used	Not used	Not used	0x00000034
12		Debug Monitor		Debug Monitor	Debug Monitor vector [1]	0x00000030
11	SVC	SVC	SVC	SVC	SVC vector [1]	0x0000002C
10	Not used	Not used	Not used	Not used	Not used	0x00000028
9					Not used	0x00000024
8					Not used	0x00000020
7				SecureFault	SecureFault (ARMv8-M Mainline) [1]	0x0000001C
6		Usage Fault		Usage Fault	Usage Fault vector [1]	0x00000018
5		Bus Fault		Bus Fault	Bus Fault vector [1]	0x00000014
4		MemManage (fault)		MemManage (fault)	MemManage vector [1]	0x00000010
3	HardFault	HardFault	HardFault	HardFault	HardFault vector [1]	0x0000000C
2	NMI	NMI	NMI	NMI	NMI vector [1]	0x00000008
1					Reset vector [1]	0x00000004
0					MSP initial value	0x00000000

图 8: Cortex-M 处理器异常中断类型

所有的 Cortex-M 处理器在异常处理是都要依靠向量表。向量表保存着异常处理函数的起始地址（如图 8 所示）。向量表的起始地址由名为向量表偏移寄存器（VTOR）决定。

- Cortex-M0+, Cortex-M3 and Cortex-M4 processors: by default the vector table is located in the starting of the memory map (address 0x0). Cortex-M0+, Cortex-M3 and Cortex-M4： 向量表默认放在存储空间的起始地址（地址 0x0）。
- In Cortex-M7, Cortex-M23 and Cortex-M33 processors: the default value for VTOR is defined by chip designers. Cortex-M23 and Cortex-M33 processors can have two separated vector tables for Secure and Non-secure exceptions/interrupts. Cortex-M7, Cortex-M23 and Cortex-M33：VTOR 的初始值由芯片设计者定义。Cortex-M23 and Cortex-M33 处理器面向安全和非安全的异常/中断有两个独立的向量表。

- Cortex-M0 and Cortex-M1 does not implement programmable VTOR and vector table starting address is always 0x00000000. Cortex-M0 and Cortex-M1 没有实现可编程的 VTOR，向量表起始地址一直为 0x00000000。

Cortex-M0+ 和 Cortex-M23 处理器的 VTOR 是可选项。如果 VTOR 被实现了，向量表的起始地址可以通过设置 VTOR 来改变，这个功能对下列情况有用：

- 重定位向量表到 SRAM 来实现动态改变异常处理函数入口点
- 重定位向量表到 SRAM 来实现更快的向量读取（如果 flash 存储器很慢）
- 重定位向量表到 ROM 不同位置（或者 Flash），不同的程序运行阶段可以有不同的异常处理程序

不同的 Cortex-M 处理器之间的 NVIC 编程模型也有额外的不同。差异点总结在表 5 中：

	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M3/M4/M7	Cortex-M23	Cortex-M33
Number of Interrupts	Up to 32	Up to 32	Up to 32	Up to 240	Up to 240	Up to 480
NMI	Y	Y	Y	Y	Y	Y
SysTick	Y (optional)	Y (optional)	Y (optional)	Y	Y (optional)	Y
Fault handlers	1 (HardFault)	1 (HardFault)	1 (HardFault)	4	1 (HardFault)	4
VTOR	-	Y (optional)	-	Y	Y (optional)	Y
Dbg Monitor	-	-	-	Y	-	Y
Programmable priority levels	4	4	4	8 to 256	4	8 to 256
Software trigger interrupt register	-	-	-	Y	-	Y
Interrupt Active status	-	-	-	Y	Y	Y
Register accesses	32-bit	32-bit	32-bit	8/16/32-bit	32-bit	8/16/32-bit
Dynamic priority change	-	-	-	Y	Y	Y

表 5: NVIC 编程模型和特性差异

大部分情况下，对 NVIC 的中断控制特性的操作都是通过 CMSIS-CORE 提供的 APIs 处理的，他们在微控制器厂商提供的设备驱动程序库里。对 Cortex-M3/M4/M7/M23/M33 处理器，即使中断被使能了，它的优先级也可以被改变。ARMv6-M 处理器不支持动态优先等级调整，当你需要改变中断优先等级是，需要暂时的关掉这个中断。

3.3 操作系统支持特性

Cortex-M 处理器架构在设计时就考虑到了操作系统的支持。针对操作系统的特性有：

- 影子堆栈指针
- 系统服务调用（SVC）和可挂起系统调用（PenSV）异常
- SysTick – 24 位递减计时器，为操作系统的计时和任务管理产生周期性的异常中断
- Cortex-M0+/M3/M4/M7/M23/M33 支持的非特权执行和存储保护单元（MPU）

系统服务调用（SVC）异常由 SVC 指令触发，他可以让运行在非特权状态的应用任务启动特权级的操作系统服务。可挂起系统调用异常在操作系统中像上下文切换这样的非关键操作的调度非常有帮助。

为了能把 Cortex-M1 放到很小的 FPGA 器件中，所有用来支持操作系统的特性对 Cortex-M1 都是可选的。对 Cortex-M0, Cortex-M0+ 和 Cortex-M23 处理器，系统时钟 SysTick 是可选的。

通常，所有的 Cortex-M 处理器都支持操作系统。执行在 Cortex-M0+, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-M23 和 Cortex-M33 的应用可以运行在非特权运行状态，并且可以同时利用可选的存储器管理单元（MPU）以避免内存非法访问。这可以增强系统的鲁棒性。

3.4 TrustZone 安全扩展

近几年来， 物联网（IoT）成为了嵌入式系统开发者们的热门话题。IoT 系统产品变得更加复杂，上市时间的压力也与日俱增。嵌入式系统产品需要更好的方案来保证系统的安全，但是同时又要方便软件开发者开发。传统的方案是通过把软件分成特权和非特权两部分解决的，特权级软件利用 MPU 防止非特权的应用访问包含安全敏感信息在内的的关键的系统资源。这种方案对一些 IoT 系统非常适合，但是在一些情况下，只有两层划分是不够的。特别是那些包含很多复杂特权级别的软件组件的系统，特权级的代码的一个缺陷就可以导致黑客彻底的控制这个系统

ARMv8-M 架构包含了一个叫做 TrustZone 的安全扩展，TrustZone 导入了安全和非安全状态的正交划分。

- 普通应用是非安全态
- 软件组件和安全相关的资源（例如，安全存储，密码加速器，真随机数发生器（TRNG））处在安全状态。

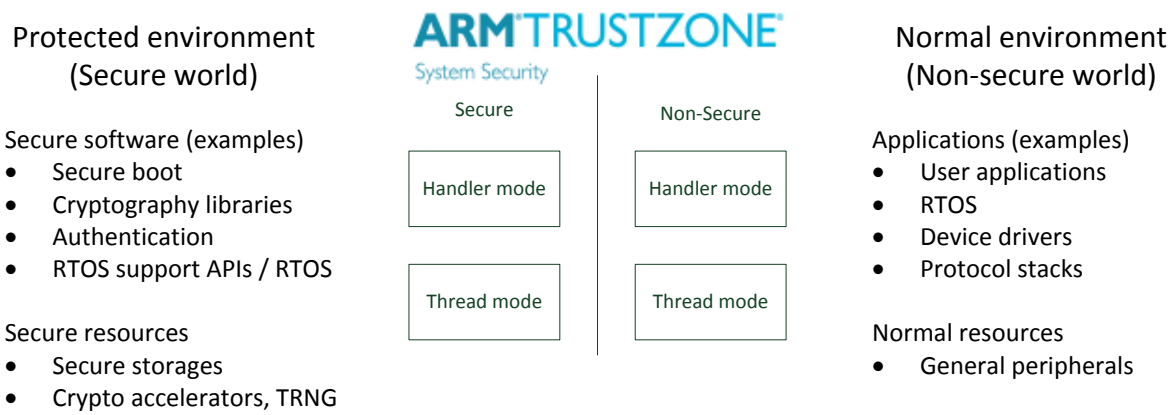

图 9: 安全状态和非安全状态的隔离

非安全状态的软件只能访问非安全状态的存储空间和外围设备，安全软件可以访问两种状态下的所有资源。

用这种方案，软件开发者可以用以往的方式开发非安全环境下的应用程序。同时，他们可以借助芯片厂商提供的安全通讯软件库执行安全物联网连接。并且即使运行在非安全环境的特权级的程序有漏洞，TrustZone 安全机制可以阻止黑客控制整个设备，限制了攻击的影响，还可以实现系统远程恢复。此外，ARMv8-M 架构也引入了堆栈边界检查和增强的 MPU 设计，促使额外安全措施的采用。

安全架构定义也扩展到了系统级别，每个中断都可以被设置为安全或者非安全属性。中断异常处理程序也会自动保存和恢复安全环境中的寄存器数据以防止安全信息泄露。所以，TrustZone 安全扩展让系统能够支持实时系统的需求，为 IoT 应用提供了坚实的安全基础，并且容易让软件开发在此技术上开发应用程序。

TrustZone 模块对 Cortex-M23 and Cortex-M33 处理器是可选的。关于 ARMv8-M TrustZone 更多的信息请查找 The Next Steps in the Evolution of Embedded Processors for the Smart Connected Era。更多的 TrustZone 的资源请查看 community.arm.com 网站上的“TrustZone for ARMv8-M Community”，

3.5 错误处理

ARM 处理器和其他架构的微控制器的一个区别是错误处理能力。当错误被检测到时，一个错误异常处理程序被触发去执行恰当的处理。触发错误的情况可能是：

- 未定义的指令（例如，Flash 存储器损坏）
- 访问非法地址空间（例如，堆栈指针崩溃）或者 MPU 非法访问
- 非法操作（例如，当处理器已经在优先级高于 SVC 的中断中试图触发 SVC 异常）

错误处理机制使嵌入式系统能够更快的响应各种问题。否则，如果系统死机了，看门狗定时需要非常长的时间重启系统。

ARMv6-M 架构中，所有的错误事件都会触发 HardFault 处理程序，它的优先级是-1（优先级比所有的可编程异常都高，但是仅低于非屏蔽中断 NMI）。 所有的错误事件都被认为是不可恢复的，通常我们在 HardFault 处理程序中仅运行错误报告然后进一步触发自动复位。

ARMv8-M Baseline 架构和 ARMv6-M 类似，只有一个错误异常（HardFault）。但是 ARMv8-M Baseline 的 HardFault 优先级可以是-1 或者当实现了 TrustZone 安全扩展时优先级是-3.

ARMv7-M 和 ARMv8-M Mainline 产品除了 HardFault 还有几个可配置的错误异常：

- Memmanage（内存管理错误）
- 总线错误（总线返回错误的响应）
- 用法错误（未定义指令或者其他的非法操作）
- SecureFault（只用 ARMv8-M Mainline 产品支持，处理 TrustZone 安全扩展中的安全非法操作）

这些异常的优先级可以编程改变，可以单独的打开和关掉。如果需要，它们也可以利用 FAULTMASK 寄存器把它们的优先级提高到和 HardFault 相同的级别。ARMv7-M 和 ARMv8-M Mainline 产品还有几个错误状态寄存器可以提供关于触发错误异常事件的线索和错误地址的寄存器，用来确定触发这个错误异常的访问地址，使调试更加容易。

ARMv7-M 和 ARMv8-M Mainline 产品子规范中额外的错误处理程序提供了灵活的错误处理能力，错误状态寄存器让错误事件的定位和调试更加容易。很多商业开发套件中的调试器已经内嵌了使用错误状态寄存器来诊断错误事件的功能。此外，错误处理程序可以在运行时做一些修复工作。

	ARMv6-M (Cortex-M0, Cortex-M0+, Cortex-M1) and ARMv8-M Baseline (Cortex-M23)	ARMv7-M (Cortex-M3, Cortex-M4, Cortex-M7) and ARMv8-M Mainline (Cortex-M33)
HardFault	Y	Y
MemManage	-	Y
Usage Fault	-	Y
Bus Fault	-	Y
SecureFault	-	ARMv8-M Mainline only
Fault Status Registers	- (one Debug FSR for debug only)	Y
Fault Address Register	-	Y

表 6: 错误处理特性比较总结

4 系统特性

4.1 低功耗

低功耗是 Cortex-M 处理器的一个关键优点。低功耗是其架构的组成部分：

- WFI 和 WFE 指令
- 架构级的休眠模式定义

此外，Cortex-M 支持许多其他的低功耗特性：

- 休眠和深度休眠模式：架构级支持的特性，通过设备特定的功耗管理寄存器可以进一步扩展。
- Sleep-on-exit 模式：中断驱动的应用的低功耗技术。开启设置后，当异常处理程序结束并且没有其他等待处理的异常中断时，处理器自动进入到休眠模式。这样避免了额外的线程模式中指令的执行从而省电，并且减少了不必要的堆栈读写操作。
- 唤醒中断控制器（WIC）:一个可选的特性，在特定的低功耗状态，由一个独立于处理器的小模块侦测中断情况。例如，在状态保留功耗管理（SRPG）设计中，当处理器被关电的设计。
- 时钟关闭和架构级时钟关闭：通过关闭处理器的寄存器或者子模块的时钟输入来省电

所有这些特性都被 Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-M23 和 Cortex-M33 支持。此外，各种低功耗设计技术被用来降低处理器功耗。

因为更少的电路，Cortex-M0 and Cortex-M0+处理器比 Cortex-M3, Cortex-M4 和 Cortex-M7 功耗低。此外，Cortex-M0+额外优化减少了程序存取（例如跳转备份）来保持系统层级的低功耗。

Cortex-M23 没有 Cortex-M0 和 Cortex-M0+那么小，但是在相同的配置下，仍然和 Cortex-M0+能效一样。由于更好性能和低功耗优化，在相同配置下，Cortex-M33 比 Cortex-M4 能效比更好。

4.2 Bit-band

Cortex-M3 和 Cortex-M4 处理器支持一个叫做 bit-band 的可选功能，支持两段 1MB 的地址空间中的每一位(bit)映射到另一段别名地址空间的每个字(word) ,允许处理器以访问别名地址空间的字的形式来访问 1MB 地址空间的位数据（一段在从地址 0x20000000 起始的 SRAM 空间。另一段是从地址 0x40000000 起始的外围设备空间）。Cortex-M0, M0+ 和 Cortex-M1 不支持位段（bit-band）功能，但是可以利用 ARM Cortex-M 系统设计套件（CMSDK）中的总线级组件在系统层面实现位段（bit-band）功能。Cortex-M7 不支持位段（bit-band），因为 M7 的 Cache 功能不能与位段一块使用（Cache 控制器不知道内存空间的别名地址）。

ARMv8-M 的 TrustZone 不支持 bit-band， 这是由于位段别名需要的两个不同的地址可能会在不同的安全域中。对于这些系统，外围设备数据的位操作反而可以在外围设备层面处理（例如，通过添加位设置和清除寄存器）。

4.3 存储器保护单元（MPU）

除了 Cortex-M0, 其他的 Cortex-M 处理器都有可选的 MPU 来实现存储空间访问权限和存储空间属性或者存储区间的定义。运行实时操作系统的嵌入式系统， 操作系统会每个任务定义存储空间访问权限和内存空间配置来保证每个任务都不会破坏其他的任务或者操作系统内核的地址空间。Cortex-M0+, Cortex-M3 和 Cortex-M4 都有 8 个可编程区域空间和非常相似的编程模型。主要的区别是 Cortex-M3/M4 的 MPU 允许两级的存储空间属性（例如，系统级 cache 类型），Cortex-M0+仅支持一级。Cortex-M7 的 MPU 可以配置成支持 8 个或者 16 个区域，两级的存储空间属性。Cortex-M0 和 Cortex-M1 不支持 MPU.

Cortex-M23 和 Cortex-M33 也支持 MPU 选项，如果实现了 TrustZone 安全扩展（一个用于安全软件程序，另一个用于非安全软件程序）可以有最多两个 MPU。

4.4 单周期 I/O 接口

单周期 I/O 接口是 Cortex-M0+处理器独特的功能，这使 Cortex-M0+可以很快的运行 I/O 控制任务。Cortex-M 大多数的处理器的总线接口是基于 AHB Lite 或者 AHB 5 协议的，这些接口都是流水实现总线协议，运行在高时钟频率。但是，这意味着每个传输需要两个时钟周期。单时钟周期 I/O 接口添加了额外的简单的非流水线总线接口，连接到像 GPIO（通用输入输出）这样的一部分设备特定的外设上。结合单周期 I/O 和 Cortex-M0+天然比较低的跳转代价（只有两级流水线），许多 I/O 控制操作都会比大多数其他微控制器架构的产品运行的更快。

5 性能考虑

5.1 通用数据处理能力

在通用微控制器市场，benchmark 数据经常用来衡量微控制器的性能，表 7 是 Cortex-M 处理器常用 benchmark 测试的性能数据：

	Dhrystone DMIPS/MHz (v2.1) – official	Dhrystone DMIPS/MHz (v2.1) – full optimization	Coremark/MHz (v1.0)
Cortex-M0	0.84	1.21	2.33
Cortex-M0+	0.94	1.31	2.42
Cortex-M3	1.25	1.89	3.32
Cortex-M4	1.25	1.95	3.40
Cortex-M7	2.14	2.55	5.01
Cortex-M23	0.98	-	2.5
Cortex-M33	1.5	-	3.86

表 7: Cortex-M 处理器常用 benchmakr 的性能分数

（来源：CoreMark.org 网站 and ARM 网站）

关于 Dhrystone 需要注意的是用来测试的 Dhrystone 是由官方源程序在没有启用 inline and 和 multi-file compilation 编译选项的情况编译出来的（官方分数）。但是，很多微控制器厂商引用的是完全优化编译的 Dhrystone 测试出来的数据。

但是，benchmark 工具的性能测试数据可能无法准确反应你的应用能达到的性能。例如，单周期 I/O 接口和 DSP 应用中使用 SIMD，或者 Cortex-M4/M7 中使用 FPU 的加速效果并没有在这些测试数据中体现出来。

通常，Cortex-M3 和 Cortex-M4 由于以下原因提供了更高的数据处理性能：

- 更丰富的指令集
- 哈佛总线架构
- 写缓存（单周期写操作）
- 跳转目标的预测取指

Cortex-M33 也是基于哈佛总线的架构，有丰富的指令集。但是不像 Cortex-M3 和 Cortex-M4，Cortex-M33 处理器流水线是重新设计的高效流水线，支持有限的指令双发射（可以在一个时钟周期中执行最多两条指令）。

Cortex-M7 支持更高的性能，这是因为 M7 拥有双发射六级流水线并支持分支预测。而且，通过支持指令和数据 Cache，和即便使用慢速内存（例如，嵌入式 Flash）也能避免性能损失的紧耦合内存，来实现更高的系统级性能。

但是，某些 I/O 操作密集的任务在 Cortex-M0+上运行更快，这是因为：

- 更短的流水线（跳转只需要两个周期）
- 单周期 I/O 端口

当然也有设备相关的因素。例如，系统级设计，内存的速度也会影响到系统的性能。

你自己的应用程序经常是你需要的最好的 benchmark。CoreMark 分数是另外一个处理器两倍的处理器并不意味着执行你的应用也快一倍。对 I/O 密集操作的应用来说，设备相关的系统级架构对性能有巨大的影响。

5.2 中断延迟

性能相关的另外一个指标是中断延迟。这通常用从中断请求到中断服务程序第一条指令执行的时钟周期数来衡量。表 8 列出了 Cortex-M 处理器在零等待内存系统条件下的中断延迟比较。

	Interrupt latency (number of clock cycles)
Cortex-M0	16
Cortex-M0+	15
Cortex-M23	15
Cortex-M3	12
Cortex-M4	12
Cortex-M7	Typically 12, worst case 14
Cortex-M33	12

表 8: 零等待内存系统条件下的中断延迟比较

事实上，真正的中断延迟受到内存系统等待状态的影响。例如，许多运行频率超过 100Mhz 的微控制器搭配的是非常慢的 Flash 存储器（例如 30 到 50MHz）。虽然使用了 Flash 访问加速硬件来提高性能，中断延迟仍然受到 Flash 存储系统等待状态的影响。所以完全有可能运行在零等待内存系统 Cortex-M0/M0+系统比 Cortex-M3/M4/M7 有更短的中断延迟。

当评估性能的时候，不要忘记把中断处理程序的执行时间考虑在内。某些 8 位或者 16 位处理器架构可能中断延迟很短，但是会花费数倍的时钟周期完成中断处理。非常短的中断响应时间和很短的中断处理时间才是实际有效的。

6 调试和跟踪特性

6.1 调试和跟踪特性简介

不同 Cortex-M 处理器之间有若干区别。总结在表 9 中。

	Cortex-M0/M1	Cortex-M0+	Cortex-M3/M4	Cortex-M7	Cortex-M23	Cortex-M33
Protocol of debug connection	JTAG / Serial Wire	JTAG / Serial Wire	JTAG / Serial Wire / both	JTAG / Serial Wire	JTAG / Serial Wire	JTAG / Serial Wire
Protocol of trace connection	-	-	Trace port (4 pin data + clock) / Serial Wire Viewer (SWV, one pin)	Trace port (4 pin data + clock) / Serial Wire Viewer (SWV, 1 pin)	Trace port (4 pin data + clock) / Serial Wire Viewer (SWV, 1 pin)	Trace port (4 pin data + clock) / Serial Wire Viewer (SWV, 1 pin)
Hardware Breakpoint comparators	Up to 4	Up to 4	Up to 8 (6 for instruction addresses, 2 literal addresses)	Up to 8 (all for instruction addresses)	Up to 4	Up to 8 (all for instruction addresses)
Software breakpoint (bkpt instruction)	Y	Y	Y	Y	Y	Y
Data Watchpoint comparators	Up to 2	Up to 2	Up to 4	Up to 4	Up to 4	Up to 4
Instruction Trace	-	Limited history using MTB	ETM trace with unlimited history	ETM trace with unlimited history	ETM / MTB	ETM / MTB
Data Trace	-	-	Selective data trace using DWT	Selective data trace using DWT, full data trace via ETM (optional)	-	Selective data trace using DWT
Event & profiling Trace	-	-	Using DWT	Using DWT	-	Using DWT
Instrumentation (Software) trace	-	-	Instrumentation Trace Macrocell (ITM)	Instrumentation Trace Macrocell (ITM)	-	Instrumentation Trace Macrocell (ITM)
Trace timestamp	-	-	Y	Y	Y (ETM)	Y
On the fly memory accesses	Y	Y	Y	Y	Y	Y
Debug using software agent (Debug monitor)	-	-	Y	Y	-	Y
Multi-core debug synchronization	Y	Y	Y	Y	Y	Y
PC sampling	Y (via debug connection)	Y (via debug connection)	Y (via debug connection or by trace)	Y (via debug connection or by trace)	Y (via debug connection)	Y (via debug connection or by trace)

表 9: 调试和跟踪特性比较

Cortex-M 处理器的调试架构是基于 ARM CoreSight 调试架构设计的，它是个非常容易扩展的架构，支持多处理器系统。

表 9 列出的是典型设计需要考虑的。在 CoreSight 架构下，调试接口和跟踪接口模块是和处理器分离的。因此你采用的设备的调试和跟踪连接和表 9 的可能不一样。也可能通过添加一些额外的 CoreSight 调试组件来增加一些调试特性。

6.2 Debug connections 调试接口

调试接口可以让调试者实现

- 访问控制调试和跟踪特性的寄存器。
- 访问内存空间。对 Cortex-M 系列处理器，及时当处理器运行时也可以执行内存空间访问。这被称作实时内存访问。
- 访问处理器核心寄存器。这只能当处理器停止的时候才可以操作。
- 访问 Cortex-M0 处理器中微跟踪缓存（MTB）生成的跟踪历史记录。

另外，调试接口也会用作：

- Flash 编程

Cortex-M 系列处理器可以选择传统的 4 到 5 个引脚（TDI, TDO, TCK, TMS 和可选的 nTRST）的 JTAG 接口，或者选择新的只需要两个引脚的串行调试协议接口，串行调试接口对有限数目引脚的设备是非常适合的。

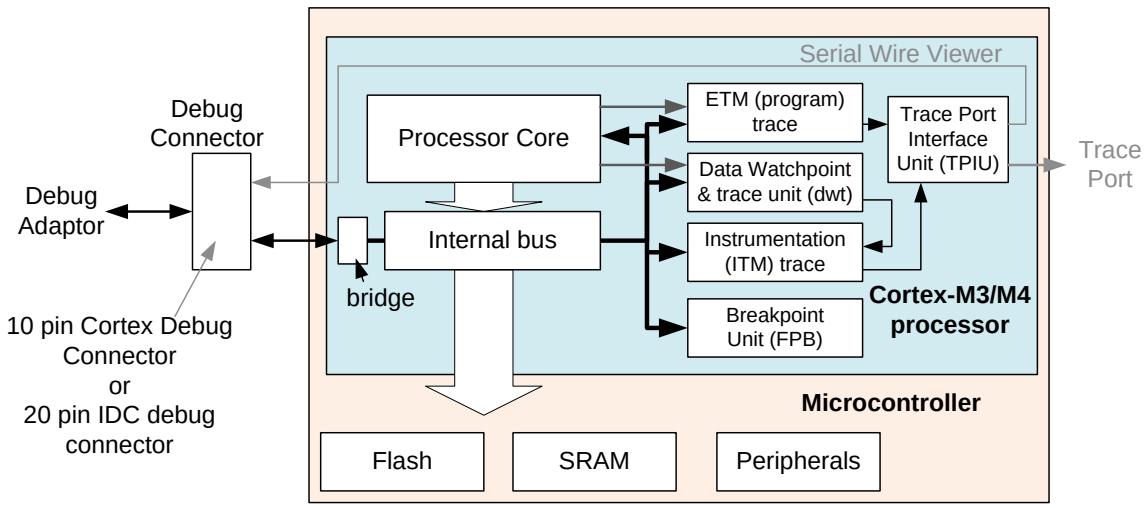

图 10: 串口线或者 JTAG 调试接口 allows access to processor's debug features and memory space including peripherals

串行线调试协议接口可以处理 JTAG 支持的所有特性，支持奇偶校验。串行调试协议被 ARM 工具厂商广泛的采用，许多调试适配器两种协议都支持，串行线型号共享调试接口上 TCK 和 TMS 针脚。

6.3 跟踪接口

跟踪接口让调试者可以在程序执行时实时的（很小的延时）收集程序运行的信息。收集的信息可以是 Cortex-M3/M4/M7/M33 支持的嵌入式跟踪单元（ETM）生成的程序指令流信息（指令跟踪），可以是数据跟踪单元（DWT）生成的数据/事件/性能分析信息，或者是软件控制数据跟踪单元（ITM）生成的信息。

有两种类型的跟踪接口可用：

- 跟踪端口（Trace port）– 多个数据线加上时钟信号线。比 SWV 有更高的跟踪带宽，可以支持 SWV 的所有跟踪类型加上指令跟踪。Cortex-M3/M4/M7 或者 Cortex-M33 的设备上，跟踪端口通常有 4 个数据线和一个时钟线。（图 11）
- 串行监视器（SWV）– 单引脚线跟踪接口，可以选择性的支持数据跟踪，事件跟踪，性能分析和测量跟踪。（图 12）

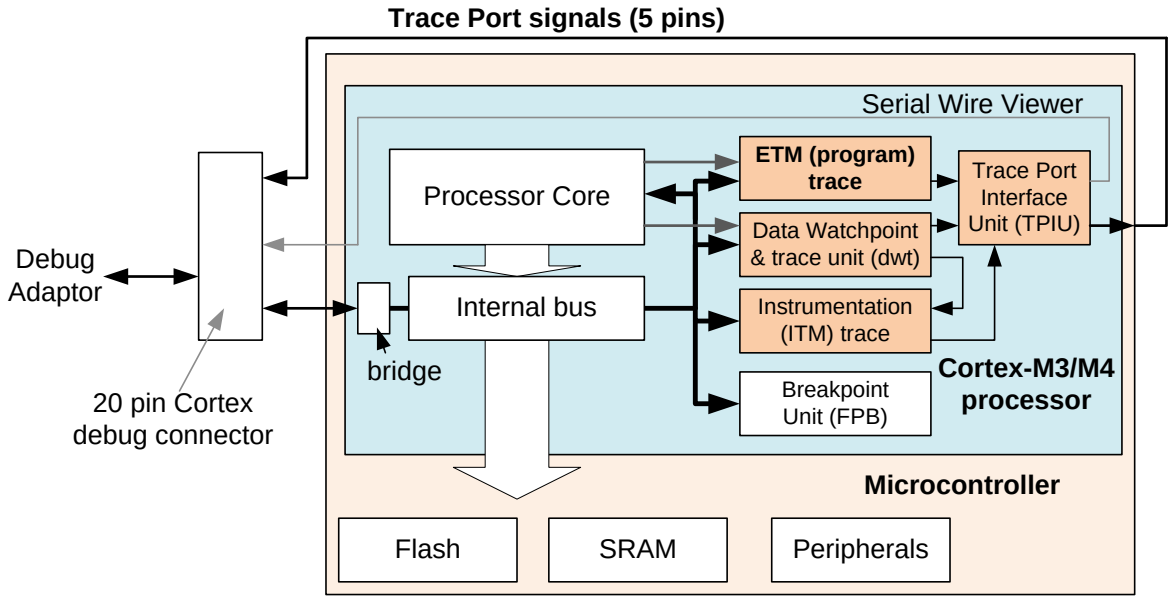

图 11: Trace port 支持指令跟踪和其他跟踪功能必要的带宽

跟踪接口提供了在处理器运行的时候获取大量有用信息的能力。例如嵌入式跟踪单元（ETM）可以获取指令运行历史记录，数据跟踪单元（ITM）让软件产生消息（例如，通过 printf）并利用 Trace 接口获取。另外，Cortex-M3/M4/M7/M33 支持数据跟踪单元（DWT）模块。

- 可选的数据跟踪：内存地址的信息（例如，地址，数据和时间戳的组合）可以在处理器访问这个地址的时候采集
- 性能分析跟踪：CPU 在不同操作任务使用的时钟周期数（例如，内存访问，休眠）
- 事件跟踪：提供服务器响应的中断/异常的运行时间和历史

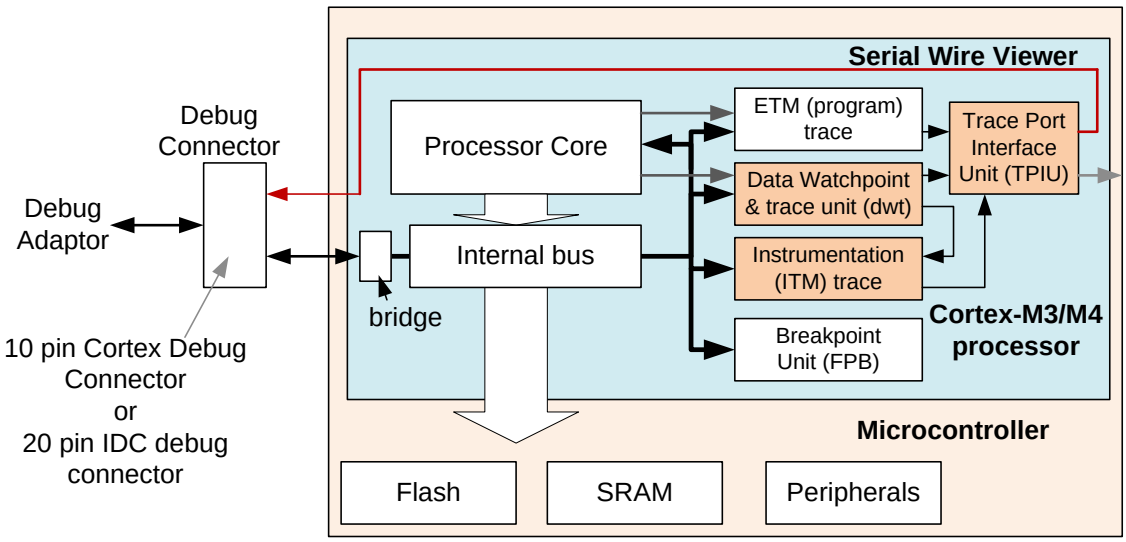

图 12: Serial wire viewer 提供了低成本，少引脚的跟踪方案

这些跟踪特性被各种工具厂商广泛采用，采集的信息也被以各种方式直观的展现出来。例如 DWT 获取的数据可以在 Keil µVision 调试器中以波形的方式展现出来（Keil 微控制器开发工具的一部分）如图 13 所示。

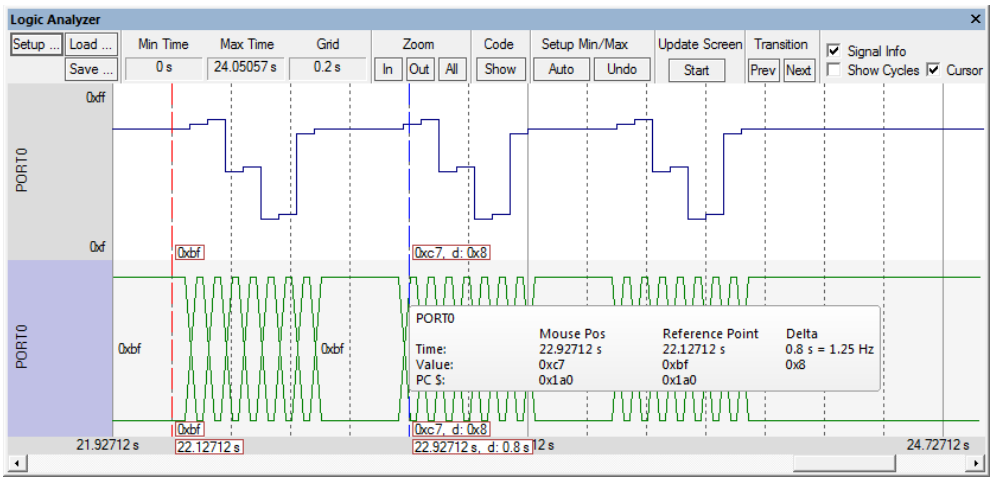

Figure 13: Keil µVision 调试器的逻辑分析器

虽然 Cortex-M0 和 Cortex-M0+不支持跟踪接口，Cortex-M0+支持叫做微跟踪缓存的特性（MTB，图 14）。MTB 让用户分配一小块系统 SRAM 作为存储指令的缓存，通常设置为循环缓存，这样可以抓取最新的指令执行历史并在调试器上显示出来。

这个 MTB 跟踪特性也被 Cortex-M23 and Cortex-M33 支持。

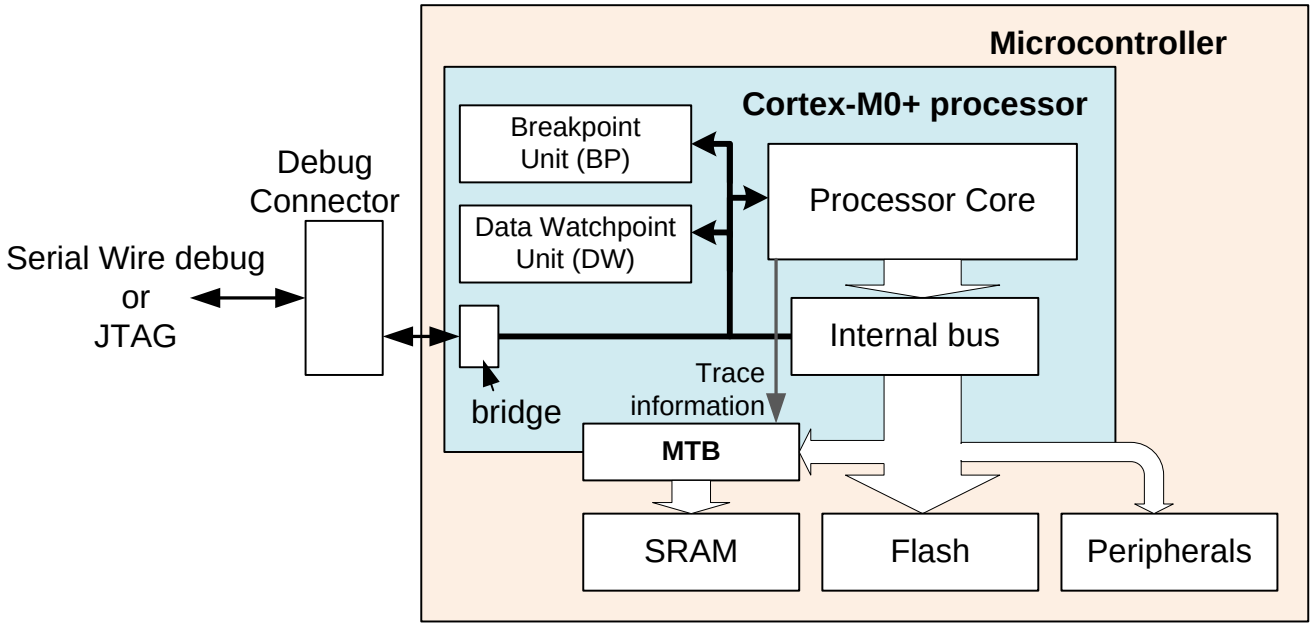

图 14: Cortex-M0+/M23/M33 的 MTB 提供了低成本指令跟踪方案

7 基于 Cortex-M 处理器的产品开发

7.1 为什么 Cortex-M 系列处理器容易使用

虽然 Cortex-M 系列处理器有非常多的特性，但是很容易使用的。例如，差不多所有的开发都可以用像 C 语言这样的高级编程语言。虽然，基于 Cortex-M 系列处理器产品都大不相同（例如，有不同大小的内存，不同的外设，性能和封装等等），架构的一致性让开发者一旦对他们其中的一块有开发经验，就很容易开始使用新的 Cortex-M 处理器。

为了实现更容易的软件开发，更好的软件重用性和可移植性，ARM 开发了 CMSIS-CORE，这儿 CMSIS 表示 Cortex-Microcontroller Software Interface Standard，CMSIS-CORE 通过一组 APIs 为处理器的各种特性像终端管理控制提供了一个标准的硬件抽象层（HAL）, CMSIS-CORE 集成在各种微处理器厂商提供的设备驱动程序库里，被各种开发工具套件支持。

除了 CMSIS-CORE， CMSIS 还包含一个 DSP 软件库（CMSIS-DSP）。这个库提供了为 Cortex-M4 和 Cortex-M7 优化过的各种 DSP 函数，当然也支持其他的 Cortex-M 系列处理器。CMSIS-CORE 和 CMSIS-DSP 库都是免费的，可以从 GitHub (CMSIS 4, CMSIS 5)下载到，并被许多工具厂商支持。

7.2 处理器选择

对大多数微控制器用户来说，微控制器设备的选择标准主要取决于成本和外设的支持情况。但是，你们中间的很多人可能是为下个芯片产品选择处理器核心芯片设计者，这种情况下，处理器 本身会是考虑的焦点。

明显的，在这样的情况下，性能，芯片面积，功耗和成本会是至关重要的因素。同时，还有各种其他的因素需要考虑。例如，如果你在开发一款互联网连接产品，你也许需要选择有 TrustZone 安全扩展和 MPU 的处理器，这样你可以用 TrustZone 保护关键的安全特性数据，运行某些任务在非特权级别并用 MPU 来保护内存空间。另一方面，如果你需要在某些方面认证你的产品，Cortex-M23, Cortex-M33, Cortex-M3, Cortex-M4 和 Cortex-M7 支持的 ETM 生成的指令跟踪会对代码覆盖率认证非常有帮助。

在其他的芯片设计领域，如果你正在设计可以运行在能量采集设备供电的小传感器，那么 Cortex-M23 和 Cortex-M0+会是最好的选择，因为他们非常小并且做了最先进的功耗优化。

7.3 生态系统

使用 ARM Cortex-M 系列处理器的关键优势之一是广泛的成熟设备，开发工具链和软件库的支持。目前有

- 超过 15 家微控制器厂商正在销售基于 ARM Cortex-M 系列内核的微控制器产品
- 超过 10 种开发套件支持 ARM Cortex-M 系列处理器
- 40 多家操作系统厂商的操作系统支持 Cortex-M 系列处理器

这给了你大量选择，让你可以获得适合目标应用的最好的设备，开发工具和中间件组合。

8 总结

性能，特性和芯片面积，功耗之间总是需要平衡。为此，ARM 开发了各种 Cortex-M 处理器，拥有不同级别的指令集特性，性能，系统和调试特性。本文介绍了 Cortex-M 处理器家族各种异同。

虽然存在这差别，但架构的一致性和 CMSIS-CORE 标准化的 APIs 都让 Cortex-M 系列处理器软件有更好的移植性和可重用性。同时，Cortex-M 系列处理器非常方便使用。因此，Cortex-M 系列处理器很快成为微控制器市场的最受欢迎的 32 位处理器架构。

额外的资源

Cortex-M 系列处理器产品信息可以查找 https://developer.arm.com/products/processors/cortex-m

一系列有用的 Cortex-M 资源存在下面的网址 https://community.arm.com/processors/b/blog/posts/cortex-m-resources

关于 ARMv8-M TrustZone 的其他的有用的资源可以查找 ARM 社区（community.arm.com）的“TrustZone for ARMv8-M Community”。ARM 社区是为开发者和开发工具厂商，产品方案商之间提供的一个免费的，开放的，非正式的交流区