AN4667
应用笔记

STM32F7 系列系统架构和性能

前言

STM32F7 系列器件是首款基于 ARM® Cortex®-M7 的 32 位微控制器。利用 ST 的 ART 加速器 ™ 和 L1 缓存的优势， STM32F7 系列器件实现了 Cortex®-M7 的最大理论性能。

基准测试分数稳步达到了 1082 CoreMark 和 462 DMIPS，无论代码是通过嵌入式 Flash 存储器执行，还是通过内部 RAM 或者外部存储器 （SRAM、 SDRAM 或者 Quad SPI Flash 存储器) 执行。

STM32F7 系列器件的高性能源自：

- 强力的超标量流水线和 DSP 性能提供了一个具有低中断时延的快速实时响应
- 对大容量外部存储的高效访问
- 适合复杂计算的高性能浮点运算能力

本应用笔记呈现了 STM32F7 的全面架构以及存储接口和特性，它们提供了更高的灵活度以实现最佳的性能以及额外的代码和数据大小。

本应用笔记同样提供了一个 STM32F7 系列器件在多种存储分区配置下 （不同代码和数据位置) 的性能演示。

本应用笔记随 X-CUBE-32F7PERF 嵌入式软件包一同提供，包括一个 stm32f7_performance 工程，其目的是演示 STM32F7 在不同配置下的性能，即代码的执行和数据的存储在不同存储空间中，并使用 ART 加速器和缓存。

目录

表格索引

图片索引

1 STM32F7 系列的系统架构概览

1.1 Cortex®-M7 内核

STM32F7 系列器件基于高性能的 ARM® Cortex®-M7 32 位 RISC 内核，工作频率高达 216 MHz。 Cortex®-M7 内核带有高性能单 / 双精度浮点运算单元 (ARM)，支持单 / 双精度数据处理指令和数据类型。它还具有一整套 DSP 指令和提高应用安全性的一个存储器保护单元 (MPU)。 Cortex®-M4 到 Cortex®-M7 的向上兼容性允许为 Cortex®-M4 编译的二进制数直接在 ®-M7 上运行。

Cortex®-M7 的特性是具有分支预测和双指令执行的 6/7- 级超标量流水线。分支预测允许分支解析以预测下一个分支，因此将循环消耗的周期数从每个循环 4~3 个周期减少为 1 个周期。双指令的特征是允许内核同时执行两条指令，并且与指令的顺序无关，由此来增加指令吞吐率。

1.2 Cortex®-M7 系统缓存

STM32F7 集成了 Cortex®-M7，其特点是具有 1 级缓存 （L1- 缓存），该缓存分为两个缓存：数据缓存 （D- 缓存）和指令缓存 （I- 缓存），这样可以实现具有最佳性能的哈佛架构。这些缓存使得即使在高频率下也可以达到零等待状态。

默认情况下，指令和数据缓存是禁用的。

ARM CMSIS 库提供了两个使能数据和指令缓存的函数：

- SCB_EnableICache() 用于使能指令缓存
- SCB_EnableDCache() 用于使能数据缓存

更多关于怎样使能和停用缓存的信息，请参考 “ARMv7-M 架构参考手册 ”。

STM32F74xxx 和 STM32F75xxx 器件有各 4KB 大小的指令缓存和数据缓存。

1.3 Cortex®-M7 总线接口

Cortex®-M7 具有五个接口：AXIM、ITCM、DTCM、AHBS 和 AHBP。这部分将对它们进行逐一阐述。

1.3.1 AXI 总线接口

AXI 作为高级可扩展接口。 Cortex®-M7 实现了 AXIM AMBA4，它是一个 64 位宽的接口，用以获得更大取指和数据加载带宽。

如果缓存使能，任何不是对 TCM 或者 AHBP 接口的访问由适当的缓存控制器处理。用户需要考虑到并非所有的存储区域都可以缓存，这取决于它们的类型。具有共享存储、器件或者强秩序类型的存储区域无法缓存。只有典型的非共享存储器才可以缓存。

更多关于存储器属性和行为的信息，请参考 “ARMv7-M 架构参考手册 ”。

可以使用 MPU 来修改存储区域的类型和属性使其可以缓存。这可以通过配置 MPU_RASR 寄存器中的 TEX 字段 S、 C 和 B 位来完成。

表 1 总结了在 cortex®-M7 复位之后存储区域的属性。

表 1. Cortex®-M7 复位之后默认的存储属性

地址范围	区域名称	类型	属性	Execute Never?
0x00000000-0x1FFFFFFF	代码	Normal	可缓存、透写、读缺失分配	无
0x20000000-0x3FFFFFFF	SRAM	Normal	可缓存、回写、读缺失和写缺失分配	无
0x40000000-0x5FFFFFFF	外设	设备	非共享	有
0x60000000-0x7FFFFFFF	RAM	Normal	可缓存、回写、读缺失和写缺失分配	无
0x80000000-0x9FFFFFFF	RAM	Normal	可缓存、透写、读缺失分配	无
0xA0000000-0xBFFFFFFF	外部器件	设备	可共享	有
0xC0000000-0xDFFFFFFF	外部器件	设备	非共享	有
0xE0000000-0xE000FFFF	私有外设总线	强序	-	有
0xE0010000-0xFFFFFFFF	供应商系统	设备	非共享	有

在 STM32F7 系列器件中， 64 位 AXI 主控总线通过一个高性能的 AXI 到 multi-AHB 桥接器件来连接内核到总线矩阵，该桥接器件具有 4 个接口：

- 到内部 Flash 存储的 1x 64 位 AHB
- 到总线矩阵的 3x 32 位 AHB

1.3.2 TCM 总线接口

作为紧密耦合的 TCM 存储器用来提供内核到内部 RAM 存储器的连接。 TCM 接口具有哈佛架构，因此这里有一个 ITCM （指令 TCM）和 DTCM （数据 TCM）接口。 ITCM 是一个 64 位的存储接口，而 DTCM 分为两个 32 位端口：D0TCM 和 D1TCM。

1.3.3 AHBS 总线接口

Cortex®-M7 AHBS （AHB 从设备）是一个 32 位宽接口，可以提供系统到 ITCM、 D1TCM 和 D0TCM 的访问。然而在 STM32F7 架构中， AHBS 只允许与 DTCM-RAM 的数据相互传输 （见*图 1*）。 AHBS 上无法访问 ITCM 总线，因此不支持 DMA 与 ITCM RAM 之间的数据传输。对于 DMA 与 ITCM 接口上的 -Flash 存储之间的数据传输，所有的传输被强制通过 AHB 总线。 AHBS 接口可以在内核处于睡眠状态时使用，因此 DMA 传输可以在低功耗模式下进行。

1.3.4 AHBP 总线接口

AHBP 接口 （AHB 外设）是一个单独的 32 位宽的接口，专门用于 CPU 和外设的连接。它只用于数据访问。取指令从不在该接口上进行。在 STM32F7 架构中，这条总线连接 Cortex-M7 内核的 AHBP 外设总线到 AHB 总线矩阵。该总线连接到 AHB1、 AHB2、 APB1 和 APB2 外设。

1.4 STM32F7 总线矩阵

STM32F7 系列器件的特征是具有一个 216 MHz 的总线矩阵，该总线矩阵实现内核、主设备和从设备的互连。该总线矩阵允许内核、主设备和从设备之间的多个并行访问路径存在，即使当几个高速外设同时工作时，也可以实现并发访问和高效运行。CPU 和它的总线矩阵可以工作在同样的频率，即 216 MHz。

一个内部的仲裁器解决了总线矩阵上的冲突和主设备的总线并发访问。仲裁器采用轮询调度算法。

图 1. STM32F74xxx 与 STM32F75xxx 系统架构

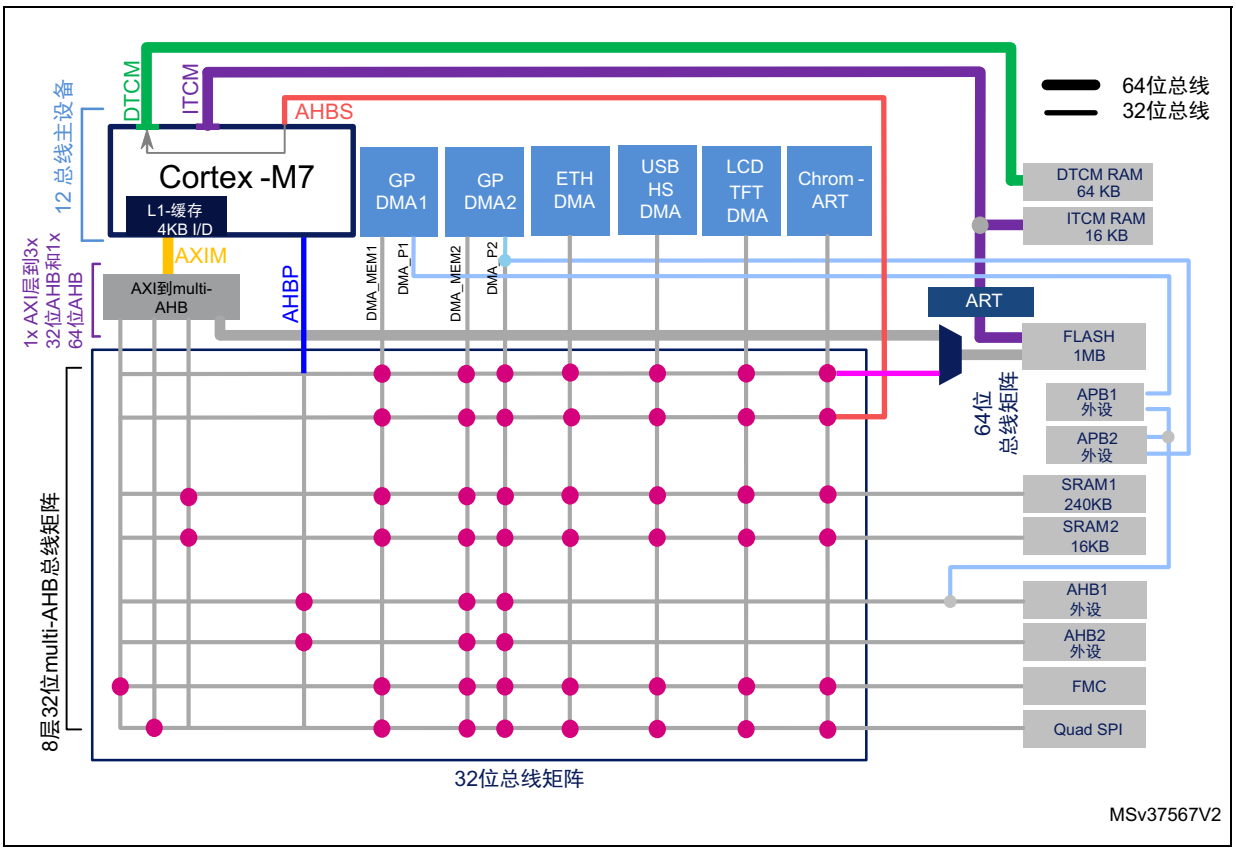


图 1 显示了 STM32F74xxx 和 STM32F75xxx 器件的整体系统架构和总线矩阵连接。

STM32F7 总线矩阵互连：

- 十二个总线主设备或发起者：
  - 三条从 AXI 转 AHB 桥出来的 32 位 AHB 总线
  - - 一条从 AXI 转 AHB 桥出来的 64 位 AHB 总线，连接到嵌入式 FLASH 存储器
  - Cortex®-M7 AHB 外设总线
  - DMA1 存储器总线
  - DMA2 存储器总线
  - DMA2 外设总线
  - 以太网 DMA 总线
  - USB OTG HS DMA 总线
  - LCD-TFT 控制器 DMA 总线
  - Chrom-Art 加速器 ™ (DMA2D) 存储器总线
- 八个总线从设备：
  - AHB 总线上的内嵌 Flash 存储器 （用于 Flash 读 / 写访问、代码执行和数据访问）
  - Cortex®-M7 AHBS 从接口 （仅用于 DTCM-RAM 的 DMA 数据传输）。
  - 内部主 SRAM1(240 KB)
  - 内部辅助 SRAM2(16 KB)
  - AHB1 外设 （包括 AHB 至 APB 总线桥、 APB1 和 APB2 外设）
  - AHB2 外设
  - FMC 存储器接口
  - Quad SPI 存储器接口

1.5 STM32F7 存储器

器件集成了 1 MB 大小的 Flash 存储器、不同大小的 SRAM、分散架构和外部存储接口，比如 FMC 和 Quad SPI。这样的配置为用户分配应用存储资源提供了灵活性，用户可以根据需要配置，并且获得性能与应用代码长度之间的最好平衡。

1.5.1 嵌入式 Flash

嵌入式 Flash 存储器为 1 MB 大小，具有 256 位宽的数据读出。可以通过三个主要的接口来进行读或 / 和写访问。

- 64 位 ITCM 接口：

  它通过 ITCM 总线 （*图 2* 中的路径 1）连接嵌入式 Flash 存储器与 Cortex-M7，同时用于程序执行和常量的数据读取。

  到 Flash 存储器写访问不允许通过这条总线完成。 Flash 存储器可以由 CPU 通过 ITCM 从开始地址 0x00200000 访问。

  由于嵌入式 Flash 存储器相对内核速度较慢，自适应实时加速器 (ART™) 用来释放 Cortex-M7 内核的性能，同时在高达 216 MHz 的 CPU 频率时，实现从 FLASH 执行零等待。STM32F7 ART™ 仅在通过 ITCM 接口访问 FLASH 时可用，它还拥有一个预取指

buffer 和一个 256 位 X64 行指令和数据分支缓存，这样增加了顺序执行代码和环路的执行速度。 ART 同样实现了预取 （ART 预取）。

- 64 位 AHB 接口：

  它通过 AXI/AHB 桥 （*图 2* 中的路径 2）连接嵌入式 Flash 存储器和 Cortex-M7。它用于代码执行、读访问和写访问。 Flash 存储器可以由 CPU 通过 AXI 从开始地址 0x08000000 访问。

- 32 位 AHB 接口：

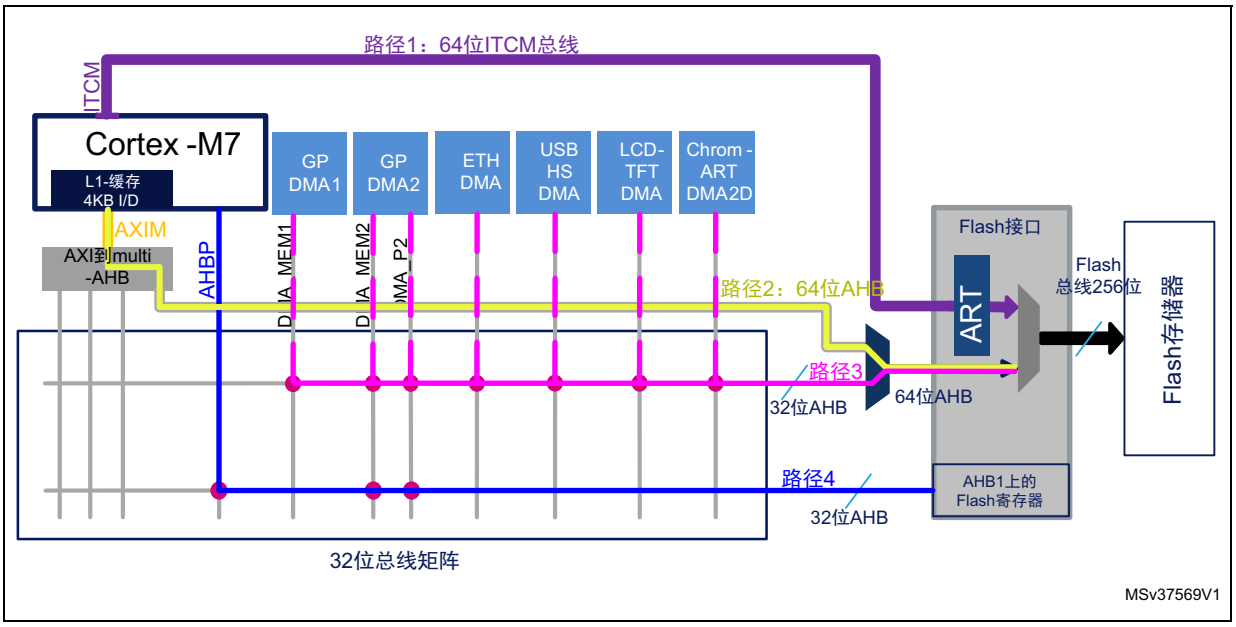
  它用于 DMA 和 Flash 存储器的传输 （*图 2* 中的路径 3）。 DMA Flash 存储器访问在 0x0800 0000 到 0x080F FFFF 范围内执行。

对于控制、配置、寄存器访问， Flash 存储器接口可以通过 AHBP/AHB1 外设路径访问，这个路径是一条 32 位 AHB 总线 （*图 2* 中的路径 4）。

如果到 Flash 存储器的访问在 0x08000000 到 0x080FFFFF 范围内完成，这就会自动通过 AXI/AHB 执行。指令或 / 和数据缓存应该在这种配置下使能以获得对 Flash 存储器的零等待访问。

如果到 Flash 存储器的访问在 0x002000000 到 0x002FFFFF 范围内完成，这就会自动通过 ITCM 总线执行。应使能 ART 加速器以获得通过 ITCM 总线到 Flash 存储器的零等待访问。通过设置 FLASH_ACR 寄存器中的位 9 来使能 ART，通过设置相同寄存器中的位 8 来使能 ART 预取。

图 2. Flash 存储器接口 （路径：1、 2、 3、 4）



1.5.2 内部 SRAM

STM32F7 系列器件具有 320KB 大小和分散架构的大 SRAM。它一共分为四个块：

- 映射到地址 0x0000 0000 的指令 RAM (ITCM-RAM)，仅可供内核访问，即通过*图 3* 中的路径 1。它可按字节、半字 （16 位）、全字 （32 位）或双字 （64 位）访问。 ITCM-RAM 可以在最大的 CPU 时钟速度下访问，没有任何延迟。 ITCM-RAM 由总线竞争保护，因为只有 CPU 可以访问这片 RAM 区域。

- DTCM-RAM 在 TCM 接口上映射到地址 0x2000 0000，可以被来自 AHB 总线矩阵上的所有 AHB 主设备访问：由 CPU 通过 DTCM 总线 （*图 3* 中的路径 5）和由 DMA 通过内核的专用 AHBS 总线，即*图 3* 中的路径 6。它可按字节、半字 （16 位）、全字 （32 位）或双字 （64 位）访问。 DTCM-RAM 可以在最大的 CPU 时钟速度下访问，没有任何延迟。通过主设备 (DMA) 到 DTCM-RAM 的并发访问和它们的优先级可以被 Cortex-M7 中的从控制寄存器 （CM7_AHBSCR 寄存器）处理，相比其它主设备 (DMA)，可以给与 CPU 更高的优先级访问 DTCM-RAM。更多关于这个寄存器的信息，请参考 “ARM® Cortex®-M7 处理器技术参考手册 ”。
- 映射到地址 0x2001 0000 的 SRAM1 可以被所有来自 AHB 总线矩阵的 AHB 主设备访问，也就是所有通用 DMA 和专用 DMA。 SRAM1 可按字节、半字 （16 位）、或全字 （32 位）访问。对于可能的 SRAM1 访问，请参考*图 3* （路径 7）。它可以用于数据读取和代码执行。
- 映射到地址 0x2004 C000 的 SRAM2 可以被所有来自 AHB 总线矩阵的 AHB 主设备访问，也就是所有通用 DMA 和专用 DMA 都可以访问这个存储区域。 SRAM2 可按字节、半字 （16 位）、或全字 （32 位）访问。对于可能的 SRAM2 访问增加 “ 路径 ”，请参考*图 3* （路径 8）。它可以用于数据读取和代码执行。

图 3. Flash 存储器接口 （路径：5、 6、 7、 8）

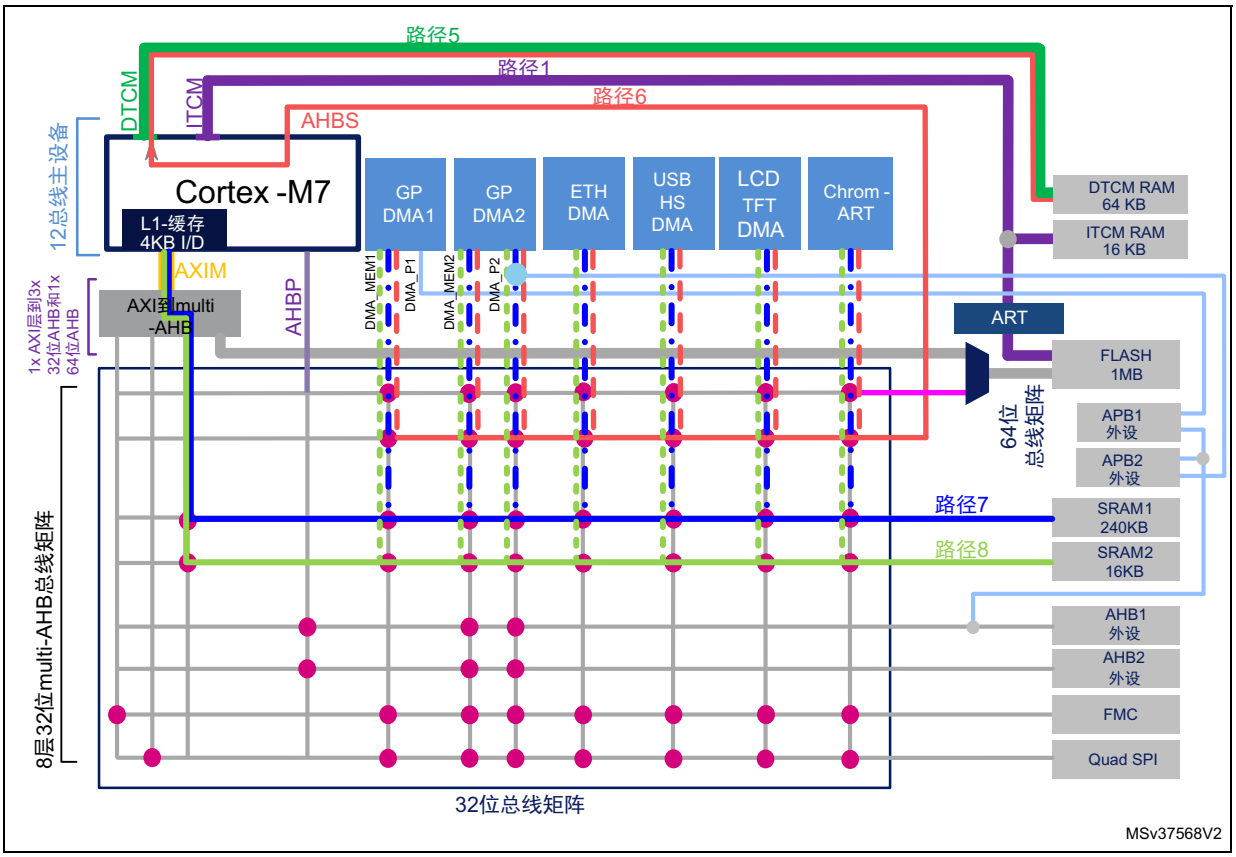


表 2 汇总了 STM32F74xxx 和 STM32F75xxx 内部存储映射和它们各自的大小：

表 2. 内部存储器总览

存储器类型	存储区域	起始地址	结束地址	大小	访问接口
闪存	FLASH-ITCM	0x0020 0000	0x002F FFFF	1 MB	ITCM （64 位）
	FLASH-AXIM	0x0800 0000	0x080F FFFF		AHB （64 位） AHB （32 位）
RAM	DTCM-RAM	0x2000 0000	0x2000 FFFF	64 KB	DTCM （64 位）
	ITCM-RAM	0x0000 0000	0x0000 3FFF	16 KB	ITCM （64 位）
	SRAM1	0x2001 0000	0x2004 BFFF	240 KB	AHB （32 位）
	SRAM2	0x2004 C000	0x2004 FFFF	16 KB	AHB （32 位）

1.5.3 外部存储器

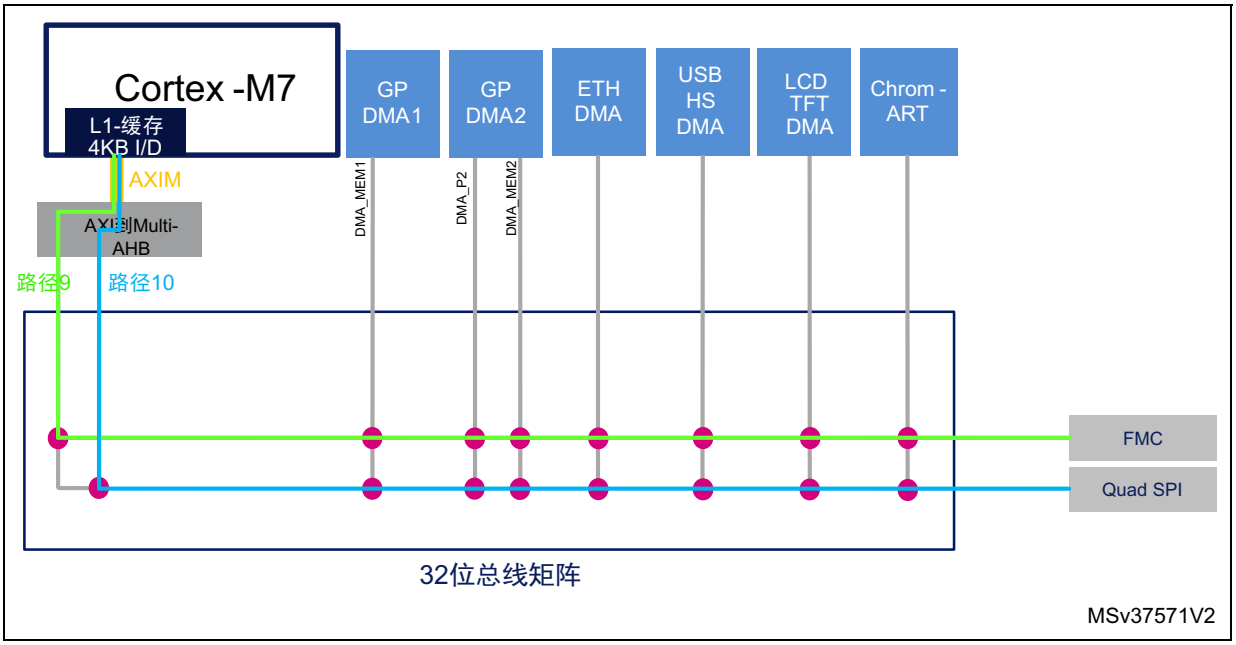
除了内部存储器和存储控制器，比如 USB 和 SDMMC 之外，用户可以用可变存储控制器 (FMC) 和 Quad SPI 控制器扩展 STM32F7 的存储空间。

图 4 显示了通过 AXI/AHB 总线互连 CPU 和外部存储的可能路径。如*图 4* 所示，它们可以利用 Cortex®-M7 缓存的优点，因此无论是数据载入 / 存储或者代码执行，都可以获得最高性能。这样性能与和大存储容量就可以相容。

图 4 中的路径 9 显示了外部存储器和 CPU 的连接，通过 FMC 控制器控制。

图 4 中的路径 10 显示了 Quad SPI Flash 存储器和 CPU 的连接，通过 Quad SPI 控制器控制。

图 4. 外部存储器接口 （路径：9、 10）



所有的外部存储器可以被所有的主设备访问。因此允许存储器 / 存储器 DMA 传输或者外设 / 存储 DMA 传输。

图 5 汇总了复位之后的外部存储的存储映射和它们的地址范围 （SYSCFG_MEMRMP 寄存器中的 SWP_FMC[1:0] 字节设置为 0）。

图 5. 外部存储映射 （复位后映射）

可变存储控制器 (FMC) 接口

STM32F7 FMC 控制器连接存储映射器件，包括 SRAM、 ROM、 NOR/NAND Flash 存储和 SDRAM 器件。它用于程序执行 （除了 NAND Flash 之外）或者数据载入 / 存储。

STM32F7 FMC 特征：

- 4 个同时支持不同存储器的存储区域。
- 每个存储区域有独立的片选控制
- 每个存储区域可独立配置
- 可对时序进行编程，以支持各种器件
- 8/16/32 位数据总线
- 外部异步等待控制
- 有两个 bank 提供与同步 DRAM(SRAM) 存储器的连接

所有 FMC 外部存储器与控制器共享地址、数据和控制信号。

每个外部器件有自己独立的片选信号。 FMC 一次只能访问一个外部器件。

两个默认的 SDRAM 存储区域不可缓存。因此即使缓存使能，数据和指令也不会通过缓存。为了利用缓存加速的优点， SDRAM 存储区域可以由 0xC000 0000 和 0xD000 0000 分别重新映射到 0x6000 0000 和 0x7000 0000，这是默认的可缓存区域。通过设置 SYSCFG_MEMRMP 寄存器中的字段 SWP_FMC [1:0] = 01 来完成。如果重映射在应用中不适用， Cortex®-M7 MPU 可以用来修改默认的 SDRAM 存储器的性质，使其可以缓存。

所有可能链接到 FMC 的外部存储器将会受益于此和 L1 缓存 （*图 4* 中的路径 9），使得可以拥有更多的数据和代码空间，并达到最高的性能。

Quad SPI 接口

STM32F7 系列器件集成了一个 Quad SPI 存储接口，它是一个专用的通信接口，连接单、双或者 Quad SPI Flash 存储器。这种多宽度接口支持传统的单线 SPI 串行输入输出，也支持双线和四线的串行命令。除此之外，接口支持双倍数据速率 (DDR) 读出命令，这意味着地址传输和地址读出在通信时钟的两个边沿执行。它允许两倍的数据 / 指令吞吐率，因此提高了对外部 Quad SPI Flash 存储器的访问效率。

它可以在以下三种模式下工作：

- 直接模式：使用 Quad SPI 寄存器执行全部操作
- 状态轮询模式：周期性读取外部 FLASH 状态寄存器，而且标志位置 1 时会产生中断（如擦除或烧写完成，会产生中断）
- 存储映射模式：外部 Flash 进行存储映射，系统将其视作内部存储器

在存储映射模式下， STM32F7 Quad SPI 接口可以管理高达 256MB 的 Flash 存储器，存储地址从 0x90 000 000 到 0X9FFF FFFF。它映射到一个可执行区域，因此不需要重映射。

相比 FMC， Quad SPI 允许通过较少的 GPIO 口连接到外部 Flash，降低成本，实现更小封装 (PCB 的面积减小了)。对于任何大小的 Flash 存储器，在单片四线模式下 （4 位）只需要 6 个 GPIO 口，在双片四线模式下 （8 位）只需要 10 个 GPIO 口。

如*图 4* （路径 10）所示， Quad SPI 映射到一个 AHB 上的专用层，并且可以受益于 L1- 缓存。这允许以良好的性能执行代码并且从 Quad SPI 载入数据。

Quad SPI 同样可以通过所有的在 AHB 总线矩阵上的主设备访问，特别是 Chrom-ART 加速器和 LCD-TFT，能够实现高效的数据传输，特别是图像，因为图像应用中需要高帧率的显示。

1.6 DMA

STM32F7 系列器件集成了两个通用的直接存储器访问 (GP DMAx)，这可以提供存储器之间和存储与外设之间的高速数据传输。它们可以减轻 CPU 的负荷，并且允许一些传输在内核处于低功耗模式时进行。每个 DMA 具有 8 个数据流，每个数据流具有 8 个通道。

STM32F7 系列器件也集成了其它用于以太网、USB OTG、LCD-TFT 和 Chrom-ART™ 加速器的专用 DMA。所有的 DMA 可以访问下列内部存储器：嵌入式 Flash 存储器、 SRAM1、SRAM2 和通过 Cortex®-M7 的 AHBS 总线访问 DTCM。所有 DMA 同样可以通过 FMC 和 Quad SPI 控制器通过总线矩阵访问外部存储器。

AHBS 上无法访问 ITCM 总线，因此不支持 ITCM RAM 上的 DMA 数据传输。对于在 ITCM 接口上的 Flash 存储与 DMA 之间的数据传输，所有的传输被强制通过 AHB 总线。

可能的 GP DMA 传输列举如下：

- GP DMA1 传输：
  - DMA1 不能寻址 AHB1/AHB2 外设
  - DMA1 只能完成 APB1 与存储器之间的相互传输
- GP DMA2 传输：
  - DMA2 可以完成所有可能的传输

2 典型应用

本应用笔记提供了一个软件示例，可以用来演示 STM32F7 的性能。使用的是 FFT 示例，由 CMSIS 库提供。stm32f7_performance 项目可以用来作为一个框架，用户可以在项目中集成自己的应用。

2.1 FFT 演示

使用 FFT 示例是因为它受益于 Cortex-M7 的浮点单元，并且包括多个循环、数据载入 / 存储，可以在不同的路径 / 存储空间完成。代码可以从内部或者外部存储执行。示例包括使用复杂 FFT 算法，复数幅度计算和最大值函数计算输入信号在频域的最大频率点。它使用 FFT 1024 点，计算基于单精度浮点。输入信号是一个 10 KHz 的混有白噪声的正弦波。*图 6* 显示了转换的框图。

FFT 处理消耗的周期数也基于系统节拍定时器计算。示例在 TSTM32756G-EVAL 上运行，结果在 LCD-TFT 上显示或者通过 UART 在 Hyperterminal 上显示，或者在 IDE printf 显示窗上显示。

配置信息也同样显示出来，包括当前项目配置、系统频率、缓存的不同配置、 ART、 ART 预取 （开 / 关）和有外部存储 （SDRAM 或 Quad SPI）情况下的存储配置。

图 6. FFT 示例框图

2.2 项目配置

本应用笔记的演示程序使用的是 KEIL MDK ARM 工具链。项目有七个配置，这些配置用于选择数据和代码位置。

配置采用以下的规则命名：

N-ExecutionRegionPath_rwDataRegion 其中：

N：配置的序号。

ExecutionRegionPath：存储地址和代码执行路径。用户需要区分执行区域和载入区域。如果两个地址位置不同，执行区域是指应用执行的存储位置，而载入区域是指应用最初保存的地址，稍后将被 Flash Loader 复制到执行区域中。

rwDataRegion：RW/Zero 初始化数据、堆栈的存储位置。

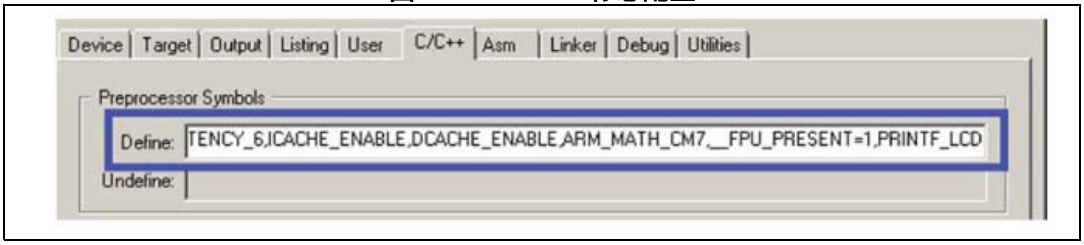
我们有如下配置：

- ***1-FlashITCM_rwRAM-DTCM***：程序从 FLASH-ITCM 执行有 7 个等待周期，ART 和 ART 预取使能并且数据载入 / 存储在 DTCM-RAM。
- ***2-FlashITCM_rwRAM-DTCM***：程序从 FLASH-ITCM 执行有 7 个等待周期，ART 和 ART 预取使能并且数据载入 / 存储在 SRAM1，使能数据缓存。
- ***3-FlashAXI_rwRAM-DTCM***：程序从 Flash-AXI 执行有 7 个等待周期，使能指令缓存并且数据载入 / 存储在 DTCM-RAM，使能数据缓存。因为一些常量从 Flash-AXI 载入来计算 FFT，所以数据缓存也被使能。
- ***4-FlashITCM_rwRAM-DTCM***：程序从 Flash-AXI 执行有 7 个等待周期，使能指令缓存并且数据载入 / 存储在 SRAM1，使能数据缓存。
- ***5-RamITCM_rwRAM-DTCM***：程序从 ITCM-RAM 开始执行，数据载入 / 存储在 DTCM-RAM。在这个配置中 ART 加速器和缓存都不需要使能。
- ***6-Quad SPI_rwRAM-DTCM***：这种配置存在两种情况：
  - 情况 1 (***6_1-Quad SPI_rwRAM-DTCM***)：代码从 Quad SPI Flash 存储器开始执行，指令缓存使能并且数据载入 / 存储在 DTCM-RAM。因为 FFT 处理使用大量常数，在这种情况下，只读数据放置在 Quad SPI Flash 存储器中。因此数据缓存也被使能。
  - 情况 2 (***6_2-Quad SPI_rwRAM-DTCM***)：只有代码在 Quad SPI Flash 中存储和执行。只读数据存储在 Flash-ITCM 中。读 / 写数据存储在 DTCM-RAM 中。指令缓存、 ART 和 ART 预取指在这种情况下使能。

  对于两种情况， Quad SPI Flash 存储器运行在 54 MHz 的频率， DDR 模式使能。
- ***7-ExtSDRAM-Swapped_rwDTCM***：程序从 FMC-SDRAM 开始执行，数据载入 / 存储在 DTCM-RAM。使能指令缓存和数据缓存。注意在这种配置下，常量数据存储在 SDRAM 中，这就是为什么使能数据缓存的原因。在这种配置下， CPU 的时钟是 200 MHz，而 SDRAM 运行频率是 100 MHz。

每个配置都有自己的标志设置。这些标志在配置项目中是可以设置的。*图 7* 显示了这些为 MDK-ARM 工具链定义的标志。

图 7. MDK-ARM 标志配置



对于所有的配置，代码优化配置为 3 级优化，时间优化优先。

项目标志描述：

- **DCACHE_ENABLE**：如果在配置项目中定义，则数据缓存使能。
- **DCACHE_ENABLE**：如果在配置项目中定义，则指令缓存使能。
- **DCACHE_ENABLE**：如果在配置项目中定义，则 ART 加速器使能。
- **DCACHE_ENABLE**：如果在配置项目中定义，则 ART 加速器中的预取使能。
- **FLASH_WS**：配置内部 Flash 等待周期的数量：

  FLASH_WS=FLASH_LATENCY_X，其中 X= 0 （0 等待周期）到 15 （15 等待周期）。
- **DATA_IN_ExtSRAM**：如果在配置项目中定义，外部 SRAM 被配置成用于数据载入 / 存储或者代码执行。
- **DATA_IN_ExtSRAM**：如果在配置项目中定义，SDRAM 被配置成用于数据载入 / 存储或者代码执行。
- **SDRAM_MEM_BUS_WIDTH**：配置外部 SDRAM 的总线带宽，配置如下所示：

  SDRAM_MEM_BUS_WIDTH= FMC_SDRAM_MEM_BUS_WIDTH_X，其中 X= 8、16 或者 32。
- **SDRAM_ADDRESS_SWAPPED**：如果在配置项目中定义， SDRAM 地址从 0xC000 0000 到 0x6000 0000 重新映射。重映射 SDRAM 允许将其放在在可缓存的区域。
- **DATA_IN_QSPI**：如果在配置项目中定义，Quad SPI Flash 存储器被配置成用于代码执行。
- **QSPI_CLK_PRESCALER**：定义了 Quad SPI 时钟预分频器，配置如下：

  ***QSPI_CLK_PRESCALER*=X**，其中 X = 0 到 255。
- **QSPI_DDRMODE**：如果在配置项目中定义， Quad SPI 配置在 DDR 模式。
- **QSPI_INSRUCTION_1_LINE**，**QSPI_INSRUCTION_4_LINES**：第一个标志用于配置 Quad SPI Flash 指令发送为一线模式，而第二个标志配置指令发送为四线模式。如果不存在标志， Quad SPI 将会被配置为指令发送为一线模式。
- **QSPI_XIP_MODE**：如果在配置项目中定义，Quad SPI 配置在 XIP 模式，也就是直接在芯片内执行。注意当缓存使能时， XIP 模式对性能没有影响。
- **PRINTF_LCD**，**PRINTF_UART**，**PRINTF_VIEWER**：这些标志用于显示示例的结果，分别在 LCD、 hyperterminal （9600、 7 位、奇校验、一个停止位、没有 HW 流控制）或者在 IDE printf 显示窗上显示。

基于这些模板，用户可以修改自己的代码执行 / 数据载入存储位置配置，通过合并合适的设置，修改 MDK-ARM 分散文件和设置合适的 Flash 加载器。

同样注意，对于 MDK-ARM 工具链，为了修改分散文件中的 RAM 区域，即堆栈区域，用户应该相应地修改 ASM 菜单中堆栈的大小，因为分散文件中的区域大小并没有作为主要应用中的真正的栈大小。用户应该对 STACK_SIZE_APPLICATION 和 HEAP_SIZE_APPLICATION 标志值进行修改，以使它们的值与分散文件中配置的堆栈大小区域一致。

图 8 显示了修改这些标志的地方 （蓝色框标出）。此外还有一个在外部存储用来数据载入 / 存储时使用的初始堆栈指针。它的大小是 1Ko，可以被 startup_stm32f756xx.s 文件中的 Stack_Size_Init 变量修改。初始堆栈指针基本地址可在 ASM 菜单中配置，如*图 8* 红色框所示。

图 8. MDK ARM 堆栈配置

不同配置的分散文件放置在演示项目的 *MDK-ARM\scatter_files* 路径下。

同样注意，在模板项目中，中断可从内部 Flash 存储器执行：只使用系统节拍定时器中断。

3 结果和分析

本节将解释配置使能后各个特性的激活情况，同时还会给出得到的 FFT 执行花费的周期数，最后进行结果分析。

结果使用 KEIL MDK-ARM v5.14.0 工具链获得。

MDK-ARM 代码优化配置是 3 级：为时间优化。

所有从 Flash-ITCM 开始执行代码的配置、 ART 加速器、 ART 预取均使能，因为指令和只读数据 （常量）流经 ITCM （*图 2* 中的路径 1）。下列配置属于这种情况：

- “1-FlashITCM_rwRAM-DTCM”
- “2-FlashITCM_rwSRAM1”

所有从 Flash-AXI (AXI/AHB) 开始执行代码的配置、指令缓存均使能，因为指令流经了*图 2* 中的路径 2。在这种配置下，只读数据 （常量）同样流经了路径 2 （同样的图片），这就是为什么数据缓存也使能的原因。下列配置属于这种情况：

- “3-FlashAXI_rwRAM-DTCM”
- “4-FlashAXI_rwSRAM1”

在 “5-RamITCM_rwRAM-DTCM” 配置情况下，什么都不需要使能，因为代码和数据分别存储在 ITCM-RAM 和 DTCM-RAM。

在 “6-Quad SPI_rwRAM-DTCM” 配置情况下，两种情况显示了性能之间的不同：

- 存储只读数据在 Quad SPI Flash 存储器中，即与指令相同的位置 （实例 1）。
- 存储只读数据在除了 Quad SPI Flash 存储器的其它存储器中，将会存储在 Flash-ITCM （实例 2）。

因为实例 1 中指令和数据存储在 Quad SPI Flash 存储器中，只有
指令缓存和数据缓存使能。

对于实例 2，因为只读数据存储在 Flash-ITCM 中， ART 和 ART 预取使能并且数据缓存关闭。对于这两种情况，读 / 写数据存储在 DTCM-RAM 中。

对于所有可以访问 DTCM-RAM 的作为数据载入 / 存储存储器的配置 （*图 3* 中的路径 5），它们不需要激活任何特性，因为内核直接以零等待状态访问 DTCM-RAM。

对于所有访问 SRAM1 作为数据载入 / 存储存储器的配置 （*图 3* 中的路径 7），数据缓存使能，即下列配置的情况：

- “2-FlashITCM_rwSRAM1”
- “4-FlashAXI_rwSRAM1”

如果应用可以通过 FMC 访问外部存储器，数据载入 / 存储和代码执行都在外部存储器中（*图 4* 中的路径 9），那么数据和 / 或指令缓存使能，即下列配置的情形：

– “7-ExtSDRAM-Swapped_rwDTCM”

注： *对于这种配置，SDRAM 发生交换（从 0xC000 0000 到 0x6000 0000 重映射以允许使用缓存），因为默认的从 0xA000 0000 到 0xDFFF FFFF 的 MPU 属性区域是一个器件存储类型区域，无法缓存。*

3.1 结果

结果在 STM32756G-EVAL 上获得，CPU 频率为 216 MHz，V_{DD}=3.3 V，访问内部 Flash 有 7 个等待周期。

表 3 显示了在不同配置下得到的对于 MDK-ARM 的 FFT 演示结果：

表 3. MDK-ARM 结果

特征配置	存储位置配置	CPU 周期数(1)
ART + ART-PF 开启	1-FlashITCM_rwRAM-DTCM	118577
ART + ART-PF + D- 缓存开启	2-FlashITCM_rwSRAM1	135296
I- 缓存 + D- 缓存开启	3-FlashAXI_rwRAM-DTCM	118653
I- 缓存 + D- 缓存开启	4-FlashAXI_rwSRAM1	145917
-	5-RAMITCM_rwRAM-DTCM	112428
I- 缓存 + D- 缓存开启（常量在 Quad SPI 中）	6_1-QuadSPI_rwRAM-DTCM	171056
I- 缓存 + ART + ART-PF 开启（常量在 Flash TCM 中）	6_2-QuadSPI_rwRAM-DTCM	126900
I- 缓存 + D- 缓存开启（常量在 SDRAM 中）	7-ExtSDRAM-Swapped_rwDTCM(2)	128398

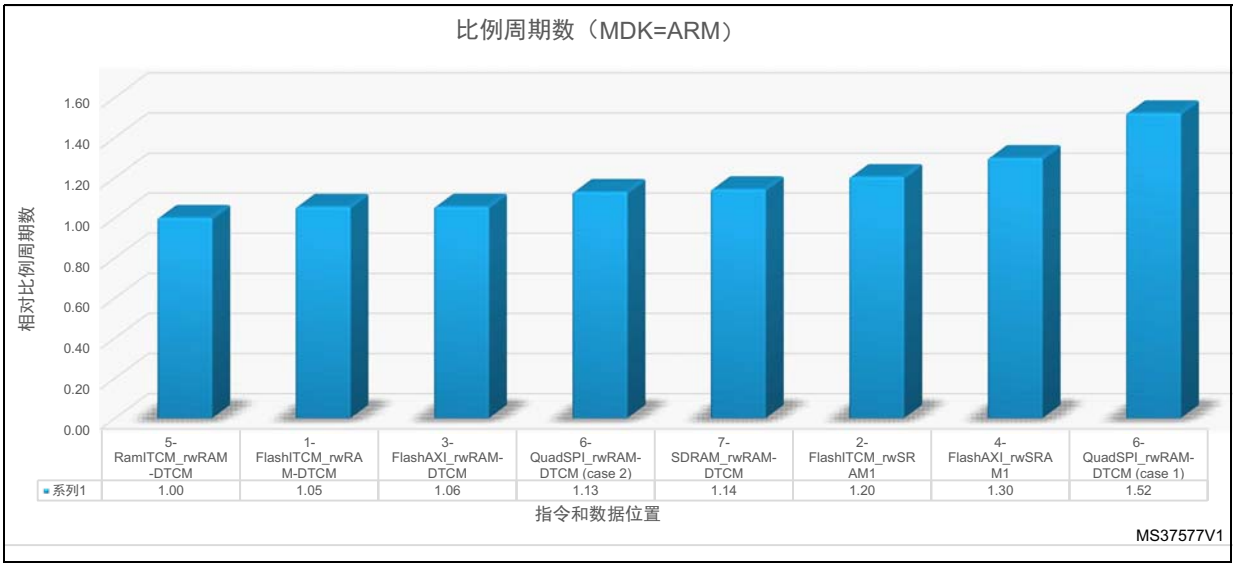
1. 周期数的值可能会随着工具链的版本变化而变化。
2. HCLK 以 200 MHz 运行。

图 9 中的表格显示了每个配置与具有最好结果的配置 5 之间的相对比例。

相对比例 = cycles_number_config_X / cycles_number_config_5

相对比例计算可以允许各个性能与最好性能做比较 (5-RamITCM_rwRAM-DTCM)，还可以在配置之间相互比较。

图 9. 使用 MDK-ARM 作 FFT 的相对比例周期数



3.2 分析

如果用户锁定数据存储位置在 DTCM-RAM，并且改变代码执行存储位置，即下列配置的情况：

- “1-FlashITCM_rwRAM-DTCM”
- “3-FlashAXI_rwRAM-DTCM”

相对比例证明了无论代码是从 Flash-ITCM 或者 Flash-AXI 开始执行，如果 ART 或者指令缓存分别使能，性能几乎是一样的。

如果用户锁定执行存储位置在 Flash-ITCM 或者 Flash-AXI，并且改变数据存储位置，即下列一对配置的情况：

- “1-FlashITCM_rwRAM-DTCM” 和 “2-FlashITCM_rwSRAM1”
- “3-FlashAXI_rwRAM-DTCM” 和 “4-FlashAXI_rwSRAM1”

对应的比例显示无论内部哪个 RAM 是用于数据载入 / 存储，结果几乎是一样的，因为对于连接到 AXI/AHB 的 RAM，数据缓存已经使能。

如果 “7-ExtSDRAM_Swapped_rwDTCM” 配置相对比例与 “1-FlashITCM_rwRAM-DTCM” 或者 “3-FlashAXI_rwRAM-DTCM” 配置的相比较，结果是接近的，因为在这些配置中都使能了缓存。因此从外部存储器执行代码或者存取数据不影响性能。

在 “5-RamITCM_rwRAM-DTCM” 配置情况下，我们注意到它具有最佳的性能 （最低的周期数）。这是因为 ITCM 和 DTCM 总线上没有竞争，因为没有执行缓存维护 （如果与缓存使用相比）。同样在给出的 FFT 示例中，没有 CPU 和其它主设备到 DTCM-RAM 的并发访问。

如果将 “6-Quad SPI_rwRAM-DTCM” 配置 （实例 X）的实例 1 和实例 2 的两个比例进行比较，我们注意到在性能上存在明显的不同，这是因为演示程序使用了大量常量数据。

对于实例 1(6_1-Quad SPI_rwRAM-DTCM) 来说，只读数据和指令存储在 Quad SPI Flash 存储器，因为指令获取和读取数据在总线矩阵上产生了并发访问，由此产生了一个延迟。

对于实例２ (6_2-Quad SPI_rwRAM-DTCM) 来说，只读数据和代码是分开的。只读数据存储在 Flash-TCM，因此避免了只读数据和指令获取产生的并发访问， CPU 从 AXI 获取指令和数据从 TCM 载入可以同时发生。这就是为什么第二例的性能明显比第一例好的原因。

4 软件存储分区和建议

本节给出了如何分配代码和数据在 STM32F7 存储器中的位置，以获得代码大小和性能的最佳平衡。

4.1 软件存储分区

由于 CPU 可以 64 位宽和零等待周期的方式直接访问 TCM 存储器， DTCM-RAM 和 ITCM-RAM 的位置对于读 / 数据和执行代码分别是最佳位置。

因此 ITCM-RAM (16 KB) 用于对执行的确定性要求高的关键代码，比如不能等待缓存缺失的中断处理以及一些针对电机控制应用的关键控制环。

DTCM-RAM (64 KB) 用于数据载入 / 存储等常规访问，以及堆栈等关键实时数据。在使用 RTOS 的实时应用中，通常堆会被大量使用，因此举例来说，如果 64 KB 的 RAM 对应用而言已足够，则 DTCM-RAM 分散方式是 32 KB 为堆、 8 KB 为栈和 24 KB 为全局变量，这种方式相对合理。在前后台应用中，几乎不使用堆， DTCM-RAM 将分给栈和全局变量。

当用户应用的代码长度不超过内部 Flash 存储器时，内部 Flash 将是最佳执行区域，可以采用下面这两种访问方式中的任何一种：

- 通过 TCM 接口访问内部 Flash，并且使能 ART 加速器，或者
- 通过 AXI 接口，并使能缓存，以在 216MHz 时实现零等待周期

通过 TCM 接口访问内部的 Flash/ 数据在 DTCM-RAM 和通过 AXI 接口访问内部 Flash/ 数据在 DTCM-RAM 中具有相同的 CoreMark 值，即 5 CoreMark/MHz。

SRAM1 (240 KB) 可以在图像应用中作为一个图像帧缓冲器，这种应用使用 16 位模式的 QVGA TFT，这需要相当大的图像数据，由 LCD-TFT 和 DMA2D DMA 实现图形显示。当 DTCM-RAM 没有更多空间可用时，这个存储器也可以用于数据载入 / 存储，所以在这样的情况下，SRAM1 的区域 （使能数据缓存）可以用于全局变量，从而为关键数据留有更多的空间。例如在 DTCM-RAM 中，堆是 48 KB，栈是 16 KB。

SRAM2 (16 KB) 可以用于外设使用，比如用作以太网和 USB 存储数据缓冲、描述符等等 ..... 这个存储器也可以用于全局变量。当应用需要更多的存储空间，它的代码或 / 和数据超过内部存储空间时，可以使用外部存储器扩展存储容量，并且不会造成性能损失。

例如一个通过 FMC 可以扩展高达 250 MB 大小的外部 NOR Flash 存储器，应用程序的代码放在其中，同时指令缓存使能，而常量数据存储在内部 Flash 存储器 （使能 ART 或者缓存）。这就是应用中使用大量常量时，为了避免指令和数据的在总线矩阵上同一路径 （*图 4* 中的路径 9）发生并发访问的情况。

在缺少内部 RAM 的情况中，数据存储可以通过使能数据缓存，经由 FMC 接口在外部 SRAM 或 SDRAM 上实现。这些存储器可能同时包含用于图像应用的帧缓冲或者非关键数据，更多的优先权给了将要放在 DTCM-RAM 中的非常关键的数据。

Quad SPI Flash 存储器可以用来存储只读数据，也就是相对大的图像文件、音频数据等 ..... 或者通过使能数据缓存，同时保持与内部 Flash 访问相同水平的性能。

Quad SPI Flash 存储器也可以工作在存储器映射模式，在这种模式下最高能支持 256MB 的地址空间，而且相比于必须连接到 FMC 接口的并行 Flash 存储器，用最小的 STM32F7 封装可以节省若干 GPIO。在 CPU 需要定期访问 QuadSPI 中的只读数据的情况下，应该将其映射到内部 Flash 的地址范围内。如果应用需要更大的空间和更好的执行性能，用户可以将代码保存在 QuadSPI （QuadSPI 是加载区），同时使用一个外部 SDRAM 作为执行区，代码将被复制到 SDRAM，并在 SDRAM 中执行。

4.2 建议

在 DMA 和 CPU 同时访问 DTCM-RAM 情况下，如果 CPU 不具有最高的访问优先级，那么应用速度可能会减慢。优先级可以可以通过软件修改 CM7_AHBSCR 寄存器来进行管理。

当一个外部存储器用作执行区域时，如果存储器映射的区域具有默认属性 Execute-Never (XN)，则需要注意。在这种情况下，用户必须使用 MPU 将其修改成为一个可执行区域，不然会出现硬故障。这就是复位后 SDRAM 存储区域就属于不可执行区域。参考*表 1* 查看关于 ARM 的默认可执行区域。

需要注意当一个外部存储器用于数据载入或者存储时，而它又没有映射到一个可缓存区域，因此可能的话，或者使用重映射来重新将其放置在一个可缓存区域，这就是 SDRAM 存储（SWP_FMC [1:0] = 01 在 SYSCFG_MEMRMP 寄存器）中的情况，或者使用 MPU 将存储类型修改为通常类型存储。

此外还需要注意，当大量的常量在应用中通过 CPU 载入时 （这就是 FFT 应用的情况），用户应该验证它们是否存储在正确的存储位置如果它们预期是存储在非内部 Flash 存储器的其它位置。在某些情况下，如果变量意外存储在内部 Flash 存储器 （TCM 或 AXI）中，且 ART 加速器或 / 和 D- 缓存没有使能，这时应用就会明显变慢。

尽可能分开数据和代码的位置，特别是当它们的存储连接到 AHB/AXI 时，这是为了避免总线矩阵上的并发访问。

不建议在调用 main 函数之前使能缓存，即在分散载入阶段之前，因为可能会出现硬故障。

5 结论

如今应用变得更加复杂，需要更多的微控制器提高效率和性能。得益于 STM32F7 的智能架构，该器件对于物联网 (IoT) 应用是一个合适的平台，同时当新代码约束不断提高创造力的时候，对于需要快速微控制器简化优化过程的开发者也是一个合适的平台。

由于 STM32F7 具有两个独立的实现零等待执行性能的机制：用于内部 Flash 存储器的 ST ART 加速器 ™ 和用于内部 Flash 存储器和其它存储器的 L1- 缓存（指令和数据缓存），可以得到更好的应用响应。用户可以借助缓存减少花在优先代码和数据大小上的时间，只需要添加外部存储资源即可，而且不会造成性能损失。

许多应用同时需要高性能和低功耗。多种技术可以应对这些挑战。这些将在其他应用笔记中阐述。

6 修订历史

表 4. 文档修订历史

日期	版本	变更
2015 年 6 月 19 日	1	初始版本。

表 5. 中文文档修订历史

日期	版本	变更
2015 年 9 月 22 日	1	中文初始版本。

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。