AN4667
Application note

STM32F7 Series system architecture and performance

Introduction

The STM32F7 Series devices are the first ARM® Cortex®-M7 based 32-bit microcontrollers. Taking advantage of ST’s ART accelerator™ as well as an L1-cache, the STM32F7 Series devices deliver the maximum theoretical performance of the Cortex®-M7.

The benchmark scores steadily reach 1082 CoreMark and 462 DMIPS, whether the code is executed from the embedded Flash memory, from the internal RAMs or from the external memories (SRAM, SDRAM or Quad-SPI Flash memory).

The STM32F7 Series devices bring a high level of performance thanks to:

- A powerful superscalar pipeline and DSP capabilities providing a fast real time response with a low interrupt latency
- An efficient access to the large external memories
- A high performance floating point capability for complex calculations

This application note presents the STM32F7 global architecture as well as the memory interfaces and features which provide a high flexibility to achieve the best performance and additional code and data sizes. It also presents the multi-master architecture that contributes to the system performance and offloads the CPU.

The application note also provides a software demonstration of the architecture performance of the STM32F7 Series devices in various memory partitioning configurations (different code and data locations) as well as the performance of architecture where DMAs are enabled.

This application note is provided with the X-CUBE-32F7PERF embedded software package that includes two projects:

- Stm32f7_performances project aims to demonstrate the performance of the STM32F7 architecture in different configurations, that is, the execution of the code and the data storage in different memory locations using ART accelerator™ and caches.
- Stm32f7_performances_DMAs aims to demonstrate the performance of the architecture in multi-master configuration.

Each project is done for all available STM32F7xxx-EVAL boards.

Contents

List of tables

List of figures

1 STM32F7 Series system architecture overview

1.1 Cortex®-M7 core

The STM32F7 Series devices are built on a high-performance ARM® Cortex®-M7 32-bit RISC core operating up to 216 MHz frequency. The Cortex®-M7 core features a high performance floating point unit (FPU). The core can feature a single precision floating point unit or a double precision floating point unit (depending on the STM32F7 Series device), which support all ARM® single-precision and double precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) that enhances the application security. A forward compatibility from the Cortex®-M4 to the Cortex®-M7 allows binaries, compiled for the Cortex®-M4 to run directly on the Cortex®-M7.

The Cortex®-M7 features a 6/7-stage superscalar pipeline with a branch prediction and dual issue instructions. The branch prediction feature allows the resolution of branches to anticipate the next branch and therefore decrease the cycle number consumed by loops from 4 and 3 cycles down to 1 cycle per loop. The dual instruction feature allows the core to execute two instructions simultaneously and usually no matter of their order, to increase instruction through-put.

1.2 Cortex®-M7 system caches

The devices embed the Cortex®-M7 that features a level1 cache (L1-cache), which is splitted into two separated caches: the data cache (D-cache) and the instruction cache (I-cache) allowing to have a Harvard architecture bringing the best performance. These caches allow to reach a performance of 0-wait state even at high frequencies.

By default, the instruction and data caches are disabled.

The ARM CMSIS library provides two functions that enable data and instruction caches:

- SCB_EnableICache() to enable the instruction cache
- SCB_EnableDCache() to enable and invalidate the data cache

For more information how to enable and invalidate the cache, refer to the “ARMv7-M architecture reference manual”.

Refer also to *Level 1 cache on STM32F7 Series* application note (AN4839) for more details on L1-cache usage on STM32F7 Series.

Table 1. STM32F7 Series device cache sizes

Device	Instruction cache size	Data cache size
STM32F74xxx and STM32F75xxx	4 Kbytes	4 Kbytes
STM32F76xxx and STM32F77xxx	16 Kbytes	16 Kbytes

1.3 Cortex®-M7 bus interfaces

The Cortex®-M7 has five interfaces: AXIM, ITCM, DTCM, AHBS and AHBP. This section describes each of them.

1.3.1 AXI bus interface

AXI as Advanced eXtensible Interface. The Cortex®-M7 implements the AXIM AMBA4, which is a 64-bit wide interface for more instruction fetch and data load bandwidth.

Any access that is not for the TCM or the AHBP interface, is handled by the appropriate cache controller if the cache is enabled. The user should take into account that not all the memory regions are cacheable, it depends on their types. The memory regions having the types: Shared Memory, Device or Strongly Ordered are not cacheable. Only the Normal Non-Shared memory type is cacheable.

For more information on the general rules about the memory attributes and behaviors, refer to the "ARMv7-M architecture reference manual".

In order to modify the type and the attribute of a memory region, the MPU can be used to allow it cacheable. This is done by configuring the TEX field and S, C and B bits in the MPU_RASR register.

Table 2 summarizes the memory region attributes after cortex®-M7 reset.

Table 2. Cortex®-M7 default memory attributes after reset

Address range	Region name	Type	Attributes	Execute Never?
0x00000000-0x1FFFFFFF	Code	Normal	Cacheable, Write-Through, Allocate on read miss	No
0x20000000-0x3FFFFFFF	SRAM	Normal	Cacheable, Write-Back, Allocate on read and write miss	No
0x40000000-0x5FFFFFFF	Peripheral	Device	Non-shareable	Yes
0x60000000-0x7FFFFFFF	RAM	Normal	Cacheable, Write-Back, Allocate on read and write miss	No
0x80000000-0x9FFFFFFF	RAM	Normal	Cacheable, Write-Through, Allocate on read miss	No
0xA0000000-0xBFFFFFFF	External Device	Device	Shareable	Yes
0xC0000000-0xDFFFFFFF	External Device	Device	Non-shareable	Yes
0xE0000000-0xE000FFFF	Private peripheral bus	Strongly ordered	-	Yes
0xE0010000-0xFFFFFFFF	Vendor system	Device	Non-shareable	Yes

In the STM32F7 Series devices, the 64-bit AXI Master bus connects the core to the bus matrix through a high performance AXI to multi-AHB bridge device, which has four master interfaces:

- 1x 64-bit AHB to the internal Flash memory
- 3x 32-bit AHB to the bus matrix

1.3.2 TCM bus interface

TCM as Tightly-Coupled Memory is provided to connect the core to the internal RAM memory. The TCM interface has an Harvard architecture, so there are an ITCM (Instruction TCM) and a DTCM (Data TCM) interfaces. The ITCM has one 64-bit memory interface while The DTCM is splitted into two ports of 32-bit: D0TCM and D1TCM.

1.3.3 AHBS bus interface

The Cortex®-M7 AHBS (AHB slave) is a 32-bit wide interface that provides system access to the ITCM, D1TCM, and D0TCM. However, in the STM32F7 architecture, AHBS allows only data transfer from/to DTCM-RAM (see *Figure 1*). The ITCM bus is not accessible on AHBS, so the DMA data transfer to/from ITCM RAM is not supported. For DMA transfer to/from the Flash memory on ITCM interface, all the transfers are forced through AHB bus. The AHBS interface can be used when the core is in sleep state, therefore, DMA transfers can be performed in low-power modes.

1.3.4 AHBP bus interface

The AHBP interface (AHB peripheral) is a single 32-bit wide interface that is dedicated to connect the CPU to the peripherals. It is used only for the data access. The instruction fetches are never performed on this interface. In the STM32F7 architecture, this bus connects the AHBP peripheral bus of the Cortex-M7 core to the AHB Bus Matrix. The targets of this bus are the AHB1, AHB2, APB1 and APB2 peripherals.

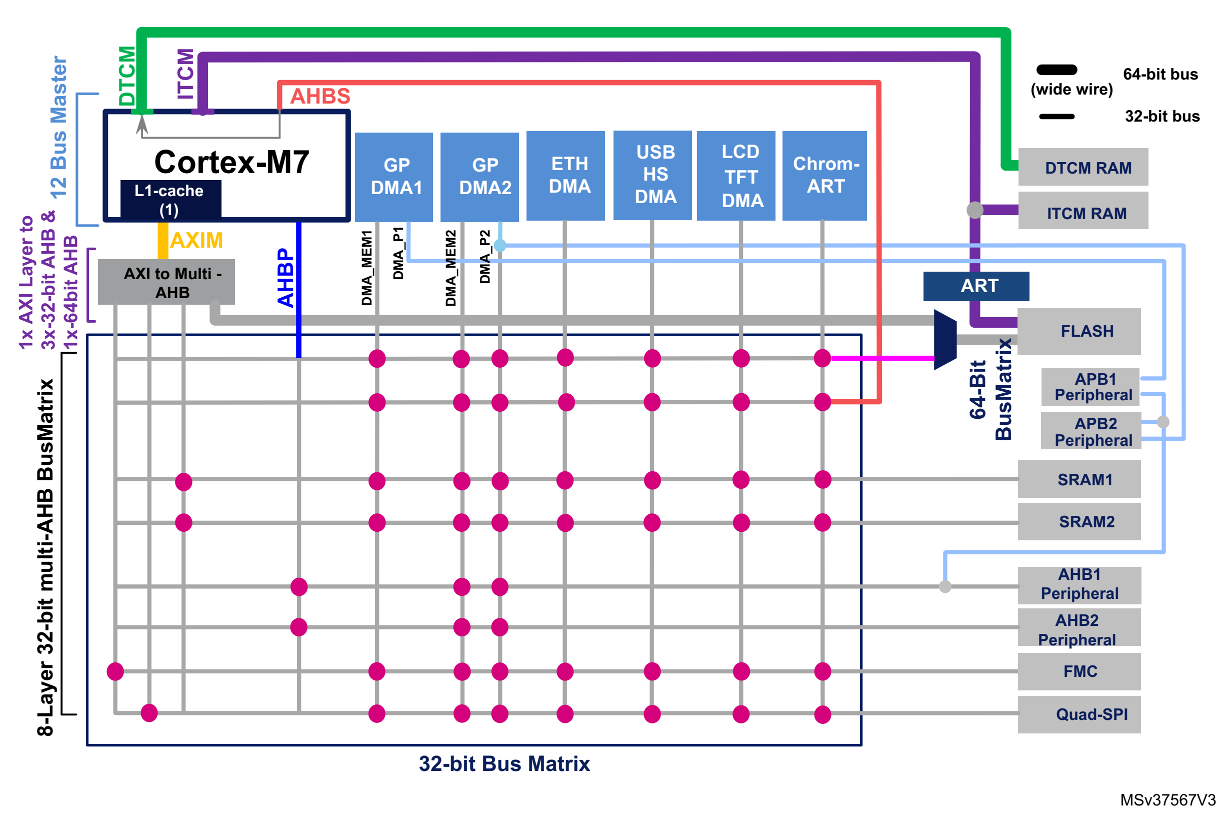
1.4 STM32F7 bus matrix

The STM32F7 Series devices feature a 216 MHz bus matrix that interconnects the core, masters and the slaves. It allows a number of parallel access paths between the core buses, masters buses and the slaves buses enabling a concurrent access and efficient operation even when several high-speed peripherals work simultaneously. The CPU and its bus matrix can run at the same frequency, that is, 216 MHz.

An internal arbiter resolves the conflicts and the bus concurrency of masters on the bus matrix. It uses a round-robin algorithm.

Figure 1 shows the overall system architecture of STM32F7 Series devices as well as the bus matrix connections.

Figure 1. STM32F7 Series system architecture



1. I/D cache size:
   - For STM32F74xxx and STM32F75xxx devices: 4 Kbytes.
   - For STM32F76xxx and STM32F77xxx devices: 16 Kbytes.

STM32F7 bus matrix interconnects:

- Twelve bus masters or initiators:
  - Three 32-bit AHB buses that outcome from the AXI to AHB bridge
  - One 64-bit AHB bus connected to the embedded Flash memory that outcomes from the AXI to AHB bridge
  - Cortex®-M7 AHB peripherals bus
  - DMA1 memory bus
  - DMA2 memory bus
  - DMA2 peripheral bus
  - Ethernet DMA bus
  - USB OTG HS DMA bus
  - LCD-TFT controller DMA-bus
  - Chrom-Art Accelerator™ (DMA2D) memory bus
- And eight bus slaves:
  - The embedded Flash memory on AHB bus (for Flash read/write access, for the code execution and data access)
  - Cortex®-M7 AHBS slave interface for DMAs data transfer on DTCM-RAM only
  - Main internal SRAM1
  - Auxiliary internal SRAM2
  - AHB1 peripherals including AHB to APB bridges, APB1 and APB2 peripherals
  - AHB2 peripherals
  - FMC memory interface
  - Quad-SPI memory interface

1.5 STM32F7 memories

The STM32F7 devices embed a Flash memory with different sizes depending on the STM32F74xxx/STM32F75xxx or the STM32F76xxx/STM32F77xxx devices (refer to *Table 3* and *Table 4*), a SRAM with different sizes, a scattered architecture and external memory interfaces such as the FMC and the Quad-SPI. This configuration gives the flexibility to the user to partition his application memory resources following his needs and to get the right compromise performance versus the application code size.

1.5.1 Embedded Flash memory

Each STM32F7 Series device has its own Flash memory size (refer to *Table 3* and *Table 4*). In the STM32F74xxx and STM32F75xxx devices, the Flash memory is accessible with 256-bit wide data read. In the STM32F76xxx and STM32F77xxx devices, the Flash memory is accessible with 128-bit or 256-bit wide data read following the bank mode enabled (256-bit access in single bank mode and 128-bit access in dual bank mode).

In all the devices the Flash memory is accessible via three main interfaces for read or/and write accesses.

- A 64-bit ITCM interface:

  It connects the embedded Flash memory to the Cortex-M7 via the ITCM bus (path1 in *Figure 2*) and is used for the program execution and data read access for constants.

  The write access to the Flash memory is not permitted via this bus. The Flash memory is accessible by the CPU through ITCM starting from the address 0x00200000.

  As the embedded Flash memory is slow compared to the core, the Adaptive Real-Time accelerator (ART) is provided to unleash the Cortex-M7 core performance and allow 0-wait execution from the Flash memory at a CPU frequency up to 216 MHz. The STM32F7 ART is available only for a Flash memory access on the ITCM interface. It implements an unified cache of instruction and branch cache of 256 bits x 64 lines in the STM32F74xxx and STM32F75xxx and 128/256 bits x 64 lines in the STM32F76xxx and STM32F77xxx devices following the bank mode selected. The ART is available for both the instruction and data access, which increases the execution speed of sequential code and loops. The ART implements also a Prefetcher (ART-Prefetch).
- A 64-bit AHB interface:

  It connects the embedded Flash memory to the Cortex-M7 via the AXI/AHB bridge (path 2 in *Figure 2*). It is used for the code execution, read and write accesses. The Flash memory is accessible by the CPU through AXI starting from the address 0x08000000.
- A 32-bit AHB interface:

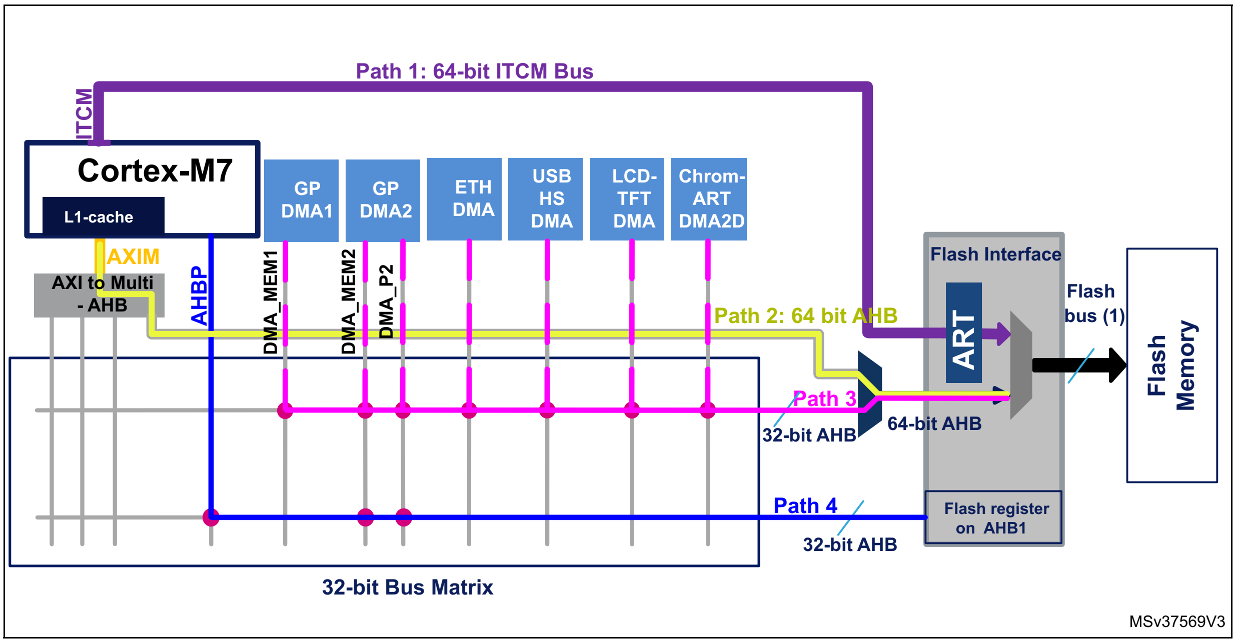
  It is used for DMAs transfers from the Flash memory (path 3 in *Figure 2*). The DMAs Flash memory access is performed starting from the address 0x0800 0000.

For the control, configuration and status register accesses, the Flash memory interface is accessible through the AHBP/ AHB1 peripheral path, which is a 32-bit AHB bus (path 4 in *Figure 2*).

If the access to the Flash memory is done starting from the address 0x0800 0000, it is performed automatically via AXI/AHB. The instruction or/and data caches should be enabled in this configuration to get 0-wait-state-like access to the Flash memory.

If the access to the Flash memory is done starting from the address 0x0200 0000, it is performed automatically via the ITCM bus. The ART accelerator™ should be enabled to get the equivalent of 0-wait state access to the Flash memory via the ITCM bus. The ART is enabled by setting the bit 9 in the FLASH_ACR register while the ART-Prefetch is enabled by setting the bit 8 in the same register.

Figure 2. Flash memory interfaces (pathes: 1, 2, 3, 4)



1. The Flash memory wide size:
   - For STM32F74xxx and STM32F75xxx devices: 256 bits.
   - For STM32F76xxx and STM32F77xxx devices: 256 bits in single bank and 128 bits in dual bank mode.

1.5.2 Embedded SRAM

The STM32F7 Series devices feature a large SRAM with a scattered architecture. It is divided into up to four blocks:

- The instruction RAM (ITCM-RAM) mapped at the address 0x0000 0000 and accessible only by the core, that is, through path 1 in *Figure 3*. It is accessible by bytes, half-words (16 bits), words (32 bits) or double words (64 bits). The ITCM-RAM can be accessed at a maximum CPU clock speed without latency. The ITCM-RAM is protected from a bus contention since only the CPU can access to this RAM region.
- The DTCM-RAM mapped on the TCM interface at the address 0x2000 0000 and accessible by all AHB masters from the AHB bus Matrix: by the CPU through the DTCM bus (path 5 in *Figure 3*) and by DMAs through the specific AHBS bus of the core that is path 6 in *Figure 3*. It is accessible by bytes, half-words (16 bits), words (32 bits) or double words (64 bits). The DTCM-RAM is accessible at a maximum CPU clock speed without latency. The concurrency access to the DTCM-RAM by the masters (DMAs) and their priorities can be handled by the slave control register of the Cortex-M7 (CM7_AHBSCR register). A higher priority can be given to the CPU to access the DTCM-RAM versus the other masters (DMAs). For more details of this register, please refer to “ARM® Cortex®-M7 processor - technical reference manual”.
- The SRAM1 is accessible by all the AHB masters from the AHB bus Matrix, that is, all general purpose DMAs as well as dedicated DMAs. The SRAM1 is accessible by bytes, half-words (16 bits) or words (32 bits). Refer to *Figure 3* (path 7) for possible SRAM1 accesses. It can be used for the data load/store as well as the code execution.

- The SRAM2 is accessible by all the AHB masters from the AHB bus matrix. All the general purpose DMAs as well as the dedicated DMAs can access to this memory region. The SRAM2 is accessible by bytes, half-words (16 bits) or words (32 bits). Refer to *Figure 3* (path 8) for possible SRAM2 accesses. It can be used for the data load/store as well as the code execution.

Figure 3. Flash memory interfaces (pathes: 5, 6, 7, 8)

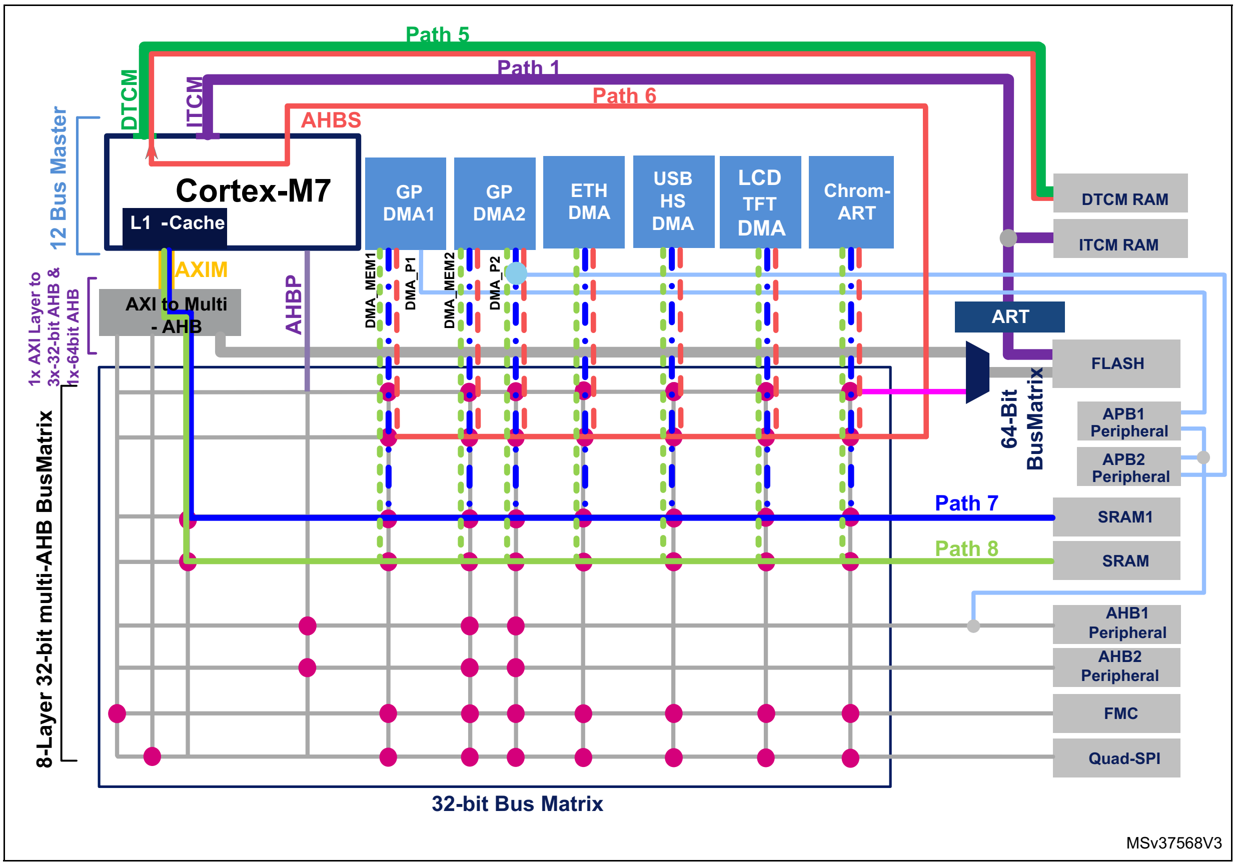


Table 3 and Table 4 summarize the internal memory mapping and the memory sizes of the STM32F7 Series devices:

Table 3. internal memory summary of the STM32F74xxx/STM32F75xxx devices

Memory type	Memory region	Address start	Address end	Size	Access interfaces
FLASH	FLASH-ITCM	0x0020 0000	0x002F FFFF	1 Mbyte	ITCM (64-bit)
	FLASH-AXIM	0x0800 0000	0x080F FFFF		AHB (64-bit) AHB (32-bit)
RAM	DTCM-RAM	0x2000 0000	0x2000 FFFF	64 Kbytes	DTCM (64-bit)
	ITCM-RAM	0x0000 0000	0x0000 3FFF	16 Kbytes	ITCM (64-bit)
	SRAM1	0x2001 0000	0x2004 BFFF	240 Kbytes	AHB (32-bit)
	SRAM2	0x2004 C000	0x2004 FFFF	16 KBytes	AHB (32-bit)

Table 4. internal memory summary of the STM32F76xxx/STM32F77xxx devices

Memory type	Memory region	Address start	Address end	Size	Access interfaces
FLASH	FLASH-ITCM	0x0020 0000	0x003F FFFF	2 Mbytes[1]	ITCM (64-bit)
	FLASH-AXIM	0x0800 0000	0x081F FFFF		AHB (64-bit) AHB (32-bit)
RAM	DTCM-RAM	0x2000 0000	0x2001 FFFF	128 Kbytes	DTCM (64-bit)
	ITCM-RAM	0x0000 0000	0x0000 3FFF	16 Kbytes	ITCM (64-bit)
	SRAM1	0x2002 0000	0x2007 BFFF	368 Kbytes	AHB (32-bit)
	SRAM2	0x2007 C000	0x2007 FFFF	16 KBytes	AHB (32-bit)

[1] 2 Mbytes in single bank and 2 x 1 Mbyte in dual bank.

1.5.3 External memories

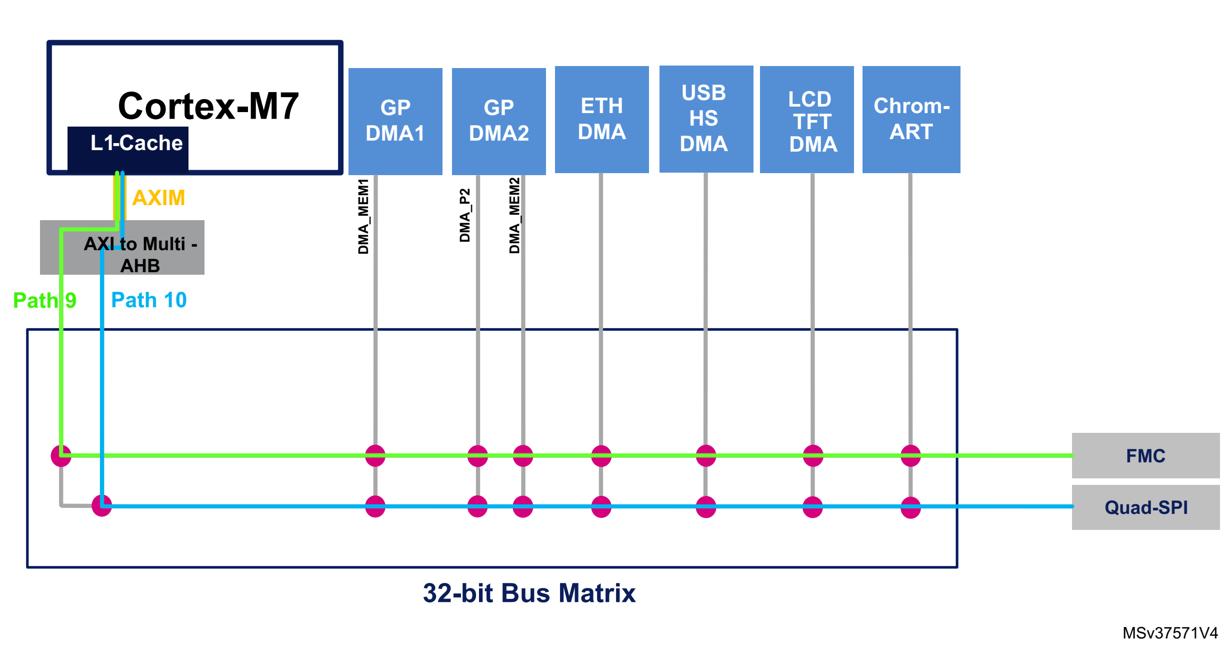
In addition to the internal memories and storage controllers such as the USB and the SDMMC, the user can extend the STM32F7 memories with the flexible memory controller (FMC) and the Quad-SPI controller.

Figure 4 shows the possible paths that interconnect the CPU with these external memories via the AXI/AHB buses. As shown in Figure 4, The external memories can benefit of the Cortex®-M7 cache, therefore, can get the maximum of the performance whether the data load/store or the code execution. This allows to mix the performance and the large memory size.

Path 9 in Figure 4 shows the connection between the external memories, by means of the FMC controller, and the CPU.

Path 10 in Figure 4 shows the connection between the Quad-SPI Flash memories and the CPU by means of the Quad-SPI controller.

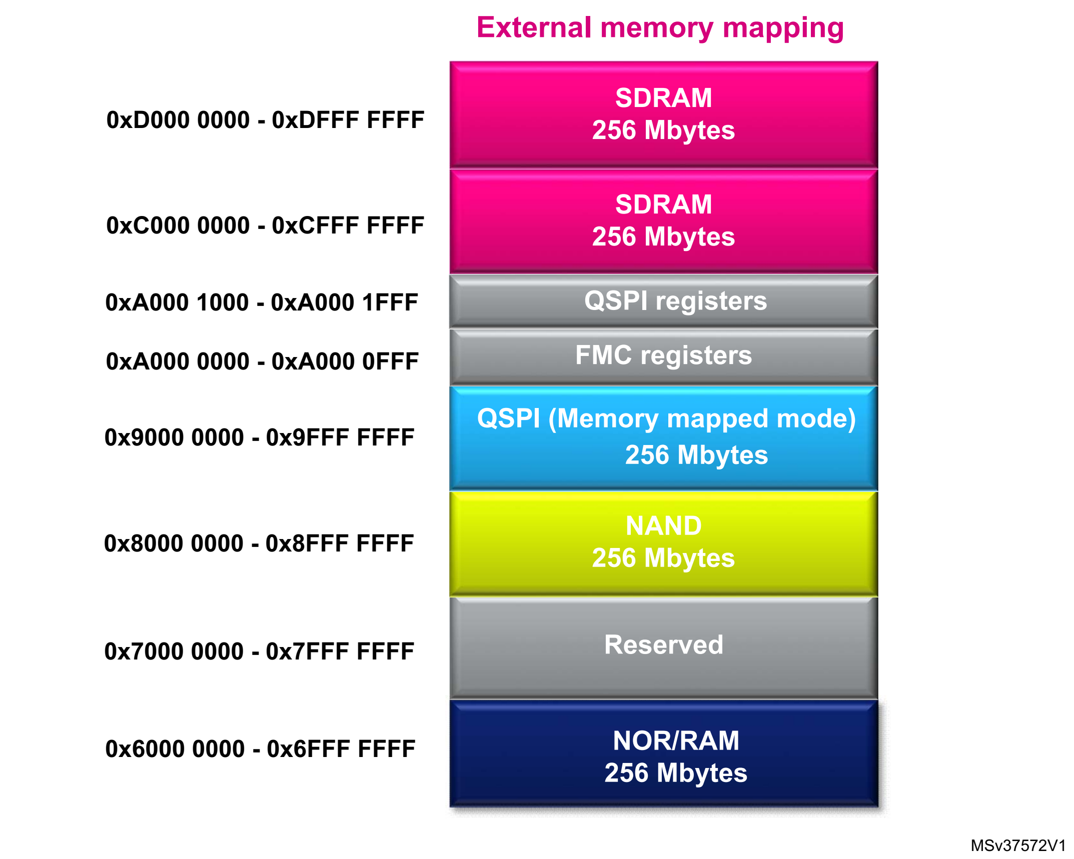
Figure 4. External memory interfaces (pathes: 9, 10)



All the external memories are accessible by all the masters. So the memory to memory DMAx transfers or peripheral/memory DMAx transfers are allowed.

Figure 5 summarizes the memory mapping of the external memories and their address ranges after reset (SWP_FMC[1:0] bits are set to 0 in SYSCFG_MEMRMP register).

Figure 5. External memory mapping (mapping after reset)


External memory mapping
0xD000 0000 - 0xDFFF FFFF
SDRAM 256 Mbytes
0xC000 0000 - 0xCFFF FFFF
SDRAM 256 Mbytes
0xA000 1000 - 0xA000 1FFF
QSPI registers
0xA000 0000 - 0xA000 0FFF
FMC registers
0x9000 0000 - 0x9FFF FFFF
QSPI (Memory mapped mode) 256 Mbytes
0x8000 0000 - 0x8FFF FFFF
NAND 256 Mbytes
0x7000 0000 - 0x7FFF FFFF
Reserved
0x6000 0000 - 0x6FFF FFFF
NOR/RAM 256 Mbytes
MSv37572V1

Flexible memory controller (FMC) interface

The STM32F7 FMC controller interfaces the memory mapped devices including SRAMs, ROMs, NOR/NAND Flash memories and SDRAM devices. It is used either for the program execution (except for NAND Flash) or the data load/store.

STM32F7 FMC features:

- 4 banks to support different memories at the same time
- Independent chip select control for each memory bank
- Independent configuration for each memory bank
- Programmable timings to support a wide range of devices
- 8/16/32-bit data bus
- External asynchronous wait control
- Interfaces with synchronous DRAM (SDRAM) that provides two SDRAM banks

All the FMC external memories share the addresses, data and control signals with the controller.

Each external device is accessed with an unique chip select. The FMC performs only one access at a time to an external device.

The two default regions of SDRAM banks are not cacheable. So even if the cache is enabled, the data or instructions will not go through the cache. To benefit from the cache acceleration, the SDRAM banks can be remapped from 0xC000 0000 and 0xD000 0000 to 0x6000 0000 and 0x7000 0000 respectively, which are, by default, cacheable regions. This is done by setting the field SWP_FMC [1:0] = 01 in SYSCFG_MEMRMP register. If the remapping is not suitable for the application, the Cortex®-M7 MPU can be used to modify the propriety of the default SDRAM memory region to be cacheable.

All the external memories that would be connected to the FMC will benefit from the data and the L1-cache (path 9 in *Figure 4*) allowing to have more data or code sizes with the maximum of performance.

Quad-SPI interface

The STM32F7 Series devices embed a Quad-SPI memory interface, which is a specialized communication interface targeting single, dual or Quad-SPI Flash memories. This multiple width interface supports a traditional SPI single bit serial input and output as well as two bits and four bit serial commands. In addition, the interface supports double data rate (DDR) read commands, meaning that the address transfer and data read are performed on both edge of the communication clock. It allows to multiply by two the data/instruction throughput and therefore to increase the access efficiency to an external Quad-SPI Flash memory.

It can work in one of the three following modes:

- Direct mode: all the operations are performed through the Quad-SPI registers
- Status polling mode: the external Flash memory status register is periodically read and an interrupt can be generated in case of a flag setting
- Memory mapped mode: the external Flash is memory mapped and is seen by the system as if it was an internal memory

The STM32F7 Quad-SPI interface is able to manage up to 256 Mbyte Flash memory starting from 0x9000 0000 to 0X9FFF FFFF in the memory mapped mode. It is mapped on an executable area, so the remap is not needed.

Compared to the FMC, the Quad-SPI allows to connect an external Flash memory with a reduced cost with small packages (reducing PCB area) and a reduced GPIOs usage. 6 GPIOs are used in single-quad mode (4-bit) for any Flash memory size or 10 GPIOs in dual-quad mode (8-bit).

As shown in *Figure 4* (path 10), the Quad-SPI is mapped on a dedicated layer on the AHB and can benefit from the L1-cache. This allows to execute the code and load the data from the Quad-SPI with good performances.

The Quad-SPI is also accessible by all the masters on the AHB bus matrix, especially the Chrom-ART accelerator™ and the LCD-TFT, enabling an efficient data transfer, particularly images, for graphical applications where a high frame rate of display is needed.

1.6 DMAs

The STM32F7 Series devices embed two general purpose Direct Memory Access (GP DMAx) that can provide a high speed data transfer between memories and between a memory and a peripheral. The DMAs are provided to offload the CPU and to allow some transfers when the core is in a low-power mode. Each DMA has 8 streams and 8 channel per stream.

The STM32F7 Series devices embed also other dedicated DMAs for Ethernet, USB OTG, LCD-TFT and Chrom-ART accelerator™. All the DMAs have access to the following internal memories: embedded Flash memory, SRAM1, SRAM2 and DTCM through the AHBS bus of the Cortex®-M7. All the DMAs also have an access to external memories through FMC and Quad-SPI controllers via the bus matrix.

The ITCM bus is not accessible on the AHBS. So the DMA data transfer to/from the ITCM RAM is not supported. For the DMA transfer to/ from the Flash memory on the ITCM interface, all the transfers are forced through the AHB bus.

The possible GP DMAs transfers are listed below:

- GP DMA1 transfers:
  - DMA1 cannot address AHB1/AHB2 peripherals
  - DMA1 can only make APB1 from/to memory transfers
- GP DMA2 transfers:
  - DMA2 can do all possible transfers

Thanks to the AXI-to multi-AHB Bridge and the bus matrix architecture, the concurrent access by the CPU and the DMAs to different slaves are managed efficiently in order to minimize the latency. The contention is also managed in the Cortex-M7 at the TCU level between the LSU (Load/Store Unit) and AHB Slave bus accesses when the CPU and other system master attempt to access simultaneously to the DTCM-RAM. Only one Master can gain the access to the AHBS, which is managed by the bus matrix arbitration.

The CPU and the DMA access to the same memory may impact the system performance.

If the CPU and the DMA(s) do not access the same memory at the same time, no contention is present. Therefore the performance remains the same as if there no other master active with the CPU (see *Figure 6*).

If there is an access concurrency to a memory between CPU and one or more master (see *Figure 7*), the performance decreases depending on the access speed of the memory accessed by the DMAs (the access to internal memory is relatively fast versus external memory) and if the concurrency is done on read or write.

The latency generated by concurrent master accesses, depends on several factors that some of them are treated in *Section 3.2: CPU memory access performance with DMA usage*.

Figure 6. No memory concurrency between DMA and CPU

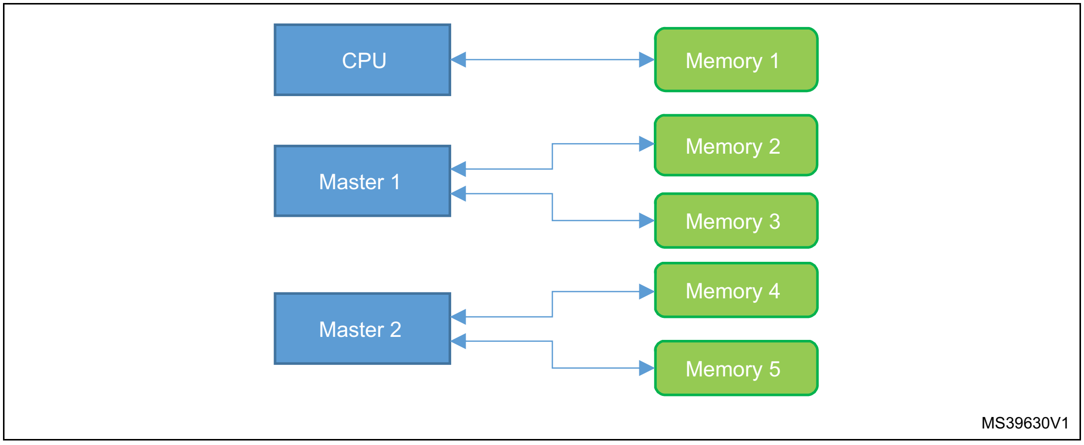
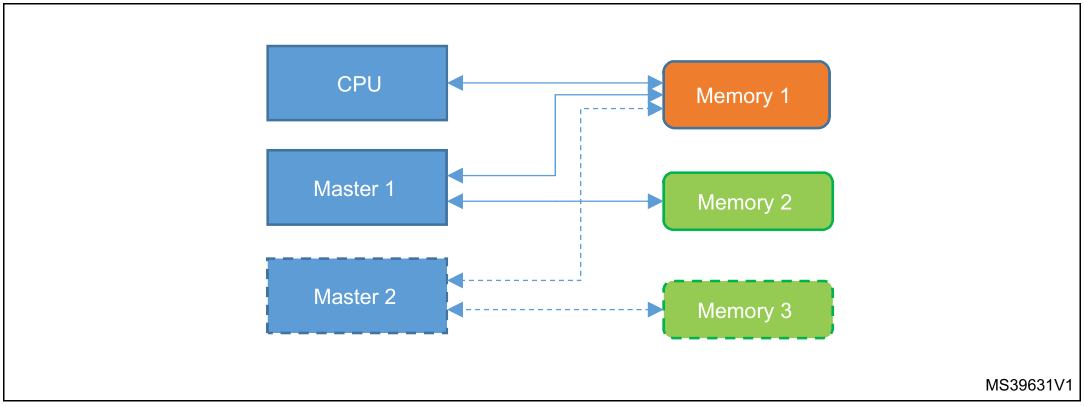


Figure 7. Memory concurrency between one or more masters and CPU



Section 2.3 presents the demonstration of the system performance when enabling multi masters (DMAs) in different scenarios.

Section 3.2 provides the different results and analysis obtained in typical scenarios based on the figures 6 and 7.

1.7 Main differences between the STM32F7 Series devices from architecture point of view

Table 5 summarizes only the differences between the STM32F7 Series devices linked to the architecture (peripheral differences not included).

Table 5. Differences between the STM32F7 Series devices

	STM32F74xxx/STM32F75xxx	STM32F76xxx/STM32F77xxx
Cortex-M7 revision	r0P1	r1P0
Instruction and Data cache sizes	4 Kbytes	16 Kbytes
FPU	Single precision floating point	Single and double precision floating point
Flash size	1 Mbyte	2 Mbytes
Bank mode	Single bank with 256bit access	Dual bank: 128bit access in dual bank and 256bit access in single bank
DTCM-RAM	Size: 64 Kbytes @ 0x20000000	Size: 128 Kbytes @ 0x20000000
SRAM1	Size: 240 Kbytes @ 0x20010000	Size: 368 Kbytes @ 0x20020000
SRAM2	Size: 16 Kbytes @ 0x2004C000	Size: 16 Kbytes @ 0x2007C000

2 Typical application

This application note provides two software examples that show first the STM32F7 performance of the CPU memory access either for the data storage or the code execution and either for internal or external memories. Secondly, the examples show the impact of the DMA usage when the CPU is loading/storing data in a memory in typical scenarios.

The example used is the FFT example provided in the CMSIS library. The stm32f7_performances project can be used as a skeleton where the user can integrate his application. The same example is used for the stm32f7_performances_DMAs project except the magnitude calculation that has been removed to allocate more RAM for DMAs transfers.

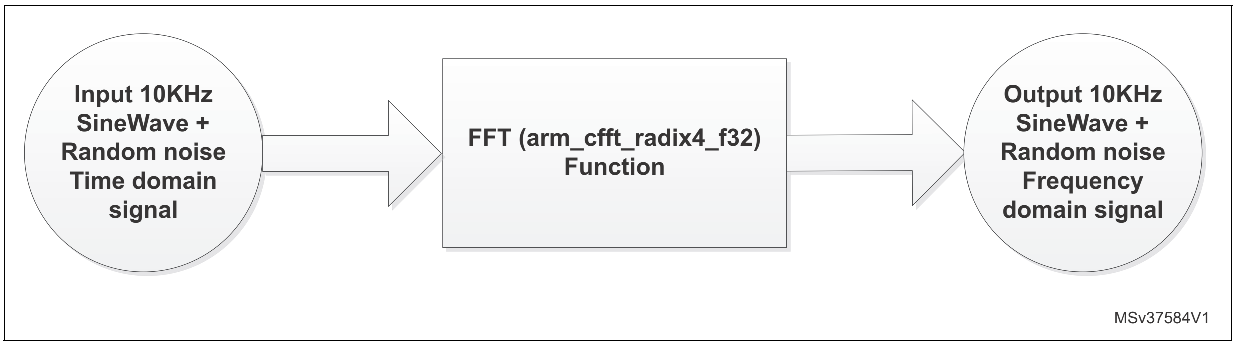
2.1 FFT demonstration

The FFT example is used as it benefits from the floating point unit of the Cortex-M7, contains several loops and data load/store, and can be done in different paths/memories. The code can be executed from internal or external memories. The example consists in the calculation of the maximum energy bin in the frequency domain of the input signal with the use of complex FFT, complex magnitude, and maximum functions. It uses the FFT 1024 points and the calculation is based on a single precision floating point. The input signal is a 10 KHz sine wave merged with white noise. *Figure 8* shows the block diagram of the transformation.

The number of cycles consumed by the FFT process is also calculated based on the system-tick timer. The example has been run on the STM32756G-EVAL and STM32769I-EVAL boards and the results are shown on the LCD-TFT or on the Hyperterminal through the UART or on the IDE printf viewer.

The FFT demonstration shows the current project configuration, the system frequency, the different configuration of caches, the ART, the ART-Prefetch (ON/OFF) and the memory configuration in case of an external memory (SDRAM or Quad-SPI).

Figure 8. FFT example block diagram


Input 10KHz SineWave + Random noise Time domain signal
FFT (arm_cfft_radix4_f32) Function
Output 10KHz SineWave + Random noise Frequency domain signal
MSv37584V1

2.2 Project configuration of the CPU memory access demonstration

The CPU memory access demonstration is provided with the Keil MDK-ARM tool chain. The seven configurations of this project allow the data selection and code locations.

The configurations are named using the following rules:

N-ExecutionRegionPath_rwDataRegion where:

N: the number of the configuration.

ExecutionRegionPath: the memory location and the path of the code execution. The user must differentiate between the execution region and the load region. The execution region is the memory location where the application is executed. The load region is the memory location where the application has been loaded initially by the Flash loader and copied later on, in the execution region if the two address locations are different.

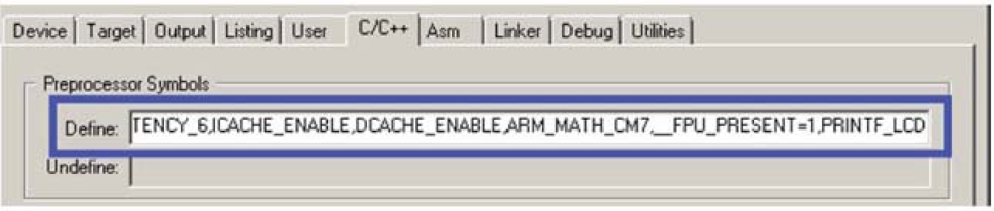
rwDataRegion: the memory location of RW/Zero Initialized data, stack and heap.

The following configurations are proposed:

- ***1-FlashITCM_rwRAM-DTCM***: the program is executed from the Flash-ITCM at 7 wait states with the ART and the ART-prefetch enabled and the data loaded/stored in the DTCM-RAM.
- ***2-FlashITCM_rwSRAM1***: the program is executed from the Flash-ITCM at 7 wait states with the ART and the ART-prefetch enabled and the data loaded/stored in the SRAM1 with the D-cache enabled.
- ***3-FlashAXI_rwRAM-DTCM***: the program is executed from the Flash-AXI at 7 wait states with the I-cache enabled and the data loaded/stored in the DTCM-RAM. The data cache is also enabled because some constants are loaded from the Flash-AXI to compute the FFT.
- ***4-FlashAXI_rwSRAM1***: the program is executed from the Flash-AXI at 7 wait states with I-cache enabled and the data loaded/stored in the SRAM1 with D-cache enabled.
- ***5-RamITCM_rwRAM-DTCM***: the program is executed from the ITCM-RAM and the data loaded/stored in the DTCM-RAM. In this configuration nothing is enabled.
- ***6-Quad SPI_rwRAM-DTCM***: This configuration covers two cases for which the Quad-SPI Flash memory runs at 54 MHz with the DDR mode enabled:
  - Case 1 (***6_1-Quad SPI_rwRAM-DTCM***): the code is executed from the Quad-SPI Flash memory with I-cache enabled and the data loaded/stored in the DTCM-RAM. Since the FFT process uses huge constants, the read-only data are located, in this case, in the Quad-SPI Flash memory. So, D-cache is also enabled.
  - Case 2 (***6_2-Quad SPI_rwRAM-DTCM***): only the code is located and executed in the Quad-SPI Flash memory. The read-only data are located in the Flash-ITCM. The read/write data are located in the DTCM-RAM. The I-cache, the ART and the ART-prefetch are enabled in this case.
- ***7-ExtSDRAM-Swapped_rwDTCM***: the program is executed from the FMC-SDRAM and the data are loaded/stored in the DTCM-RAM. The I-cache and D-cache are enabled. Note that in this configuration the constant data are placed in the SDRAM, that is why the D-cache is enabled. In this configuration the CPU clock is 200 MHz while the SDRAM runs at 100 MHz.

Each configuration has its own flag set. These flags are settable on the configuration project. *Figure 9* shows where these flags are defined for the MDK-ARM tool chain.

Figure 9. MDK-ARM flag configuration



Define: TENCY_6,ICACHE_ENABLE,DCACHE_ENABLE,ARM_MATH_CM7,__FPU_PRESENT=1,PRINTF_LCD

Undefine:

The code is optimized for time level 3 for all the configurations.

Project flags description:

- **DCACHE_ENABLE**: if defined in the configuration project, the data cache is enabled.
- **ICACHE_ENABLE**: if defined in the configuration project, the instruction cache is enabled.
- **ART_ENABLE**: if defined in the configuration project, the ART Accelerator is enabled.
- **PF_ART_ENABLE**: if defined in the configuration project, the prefetch of the ART Accelerator is enabled.
- **FLASH_WS**: configures the number of the internal Flash wait states:

  FLASH_WS=FLASH_LATENCY_X, where X= 0 (0 wait state) to 15 (15 wait states).
- **DATA_IN_ExtSRAM**: if defined in the configuration project, the external SRAM is configured to be used either for data load/store or for code execution.
- **DATA_IN_ExtSDRAM**: if defined in the configuration project, the SDRAM is configured to be used either for data load/store or for code execution.
- **SDRAM_MEM_BUS_WIDTH**: configures the bus width of the external SDRAM and the configuration is as following:

  SDRAM_MEM_BUS_WIDTH= FMC_SDRAM_MEM_BUS_WIDTH_X, where X= 8, 16 or 32.
- **SDRAM_ADDRESS_SWAPPED**: if defined in the configuration project, the SDRAM address is remapped from 0xC000 0000 to 0x6000 0000. Remapping SDRAM, allows to relocate it in a cacheable region.
- **DATA_IN_QSPI**: if defined in the configuration project, the Quad-SPI Flash memory is configured to be used for the code execution.
- **QSPI_CLK_PRESCALER:** defines the Quad-SPI clock prescaler and the configuration is as following:

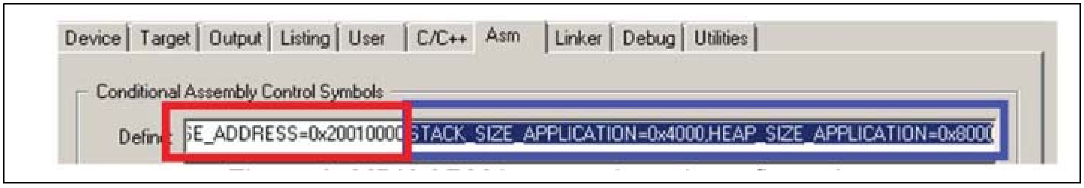
  ***QSPI_CLK_PRESCALER*=X,** where X = 0 to 255**.**
- **QSPI_DDRMODE**: if defined in the configuration project, the Quad-SPI is configured in DDR mode.
- **QSPI_INSRUCTION_1_LINE, QSPI_INSRUCTION_4_LINES:** the first flag is to configure the Quad-SPI Flash instruction to one line, while the second configures the instruction to four lines. If no flag is present, the Quad-SPI will be configured to one line instruction.
- **QSPI_XIP_MODE**: if defined in the configuration project, the Quad-SPI is configured in XIP mode, that is, only the instruction is sent first. Note that XIP mode has no effect on the performance when the cache is enabled.
- **PRINTF_LCD, PRINTF_UART, PRINTF_VIEWER**: these flags are used to display the results of the demonstration, respectively, on LCD, through hyperterminal (9600, 7 bits, parity odd, one stop bit, no HW flow control) or on IDE printf viewer.

The user can create new code execution/data load store location configurations based on these templates: by merging the adequate settings, by modifying the scatter files for MDK-ARM and by setting the adequate Flash loader.

Note also, that for the MDK-ARM tool chain, in order to modify the RAM regions in the scatter files, that is, the stack and heap regions, the user must modify accordingly the stack and heap sizes in the ASM menu. The size of the region in the scatter file is not considered as the real stack size of the main application. The user must modify the STACK_SIZE_APPLICATION and HEAP_SIZE_APPLICATION flag values in order to force their values in line with the heap/stack size regions configured in the scatter files.

Figure 10 shows where to modify these flags (framed in blue). There is also an initial stack pointer that is used when external memories are used for data load/store. Its size is 1Ko and can be modified by Stack_Size_Init variable in the startup_stm32f756xx.s and startup_stm32f769xx.s files. The initial stack pointer base address is configurable in the ASM menu as shown in *Figure 10* framed in red.

Figure 10. MDK ARM heap and stack configurations



Defin: ;E_ADDRESS=0x20010000 STACK_SIZE_APPLICATION=0x4000,HEAP_SIZE_APPLICATION=0x800(

The scatter files of different configurations are located under *MDK-ARM\scatter_files* path in the project of the demonstration.

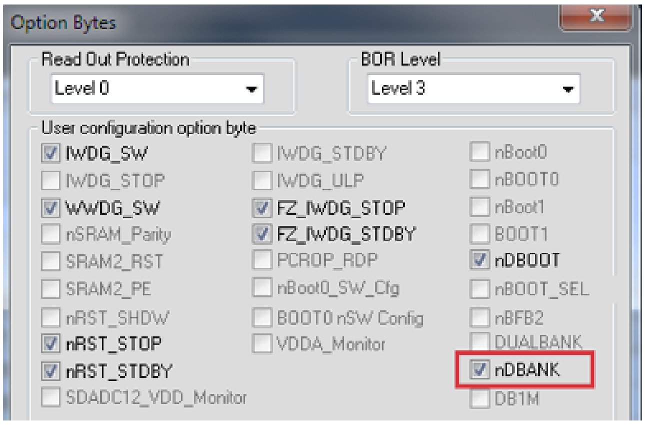
Note also that in the project template, the interrupts are executable from the internal Flash memory: only the system tick timer interrupt is used.

The project does not contain the environment for dual bank configuration. However, the user can use the same project in single bank but he must follow the two following steps:

1. Configure the Flash memory in dual bank using ST-Link utility (v3.8.0 or later). See *Figure 11*.

   Un-check the nDBANK box to set the Flash memory in dual bank mode.

Figure 11. ST-Link utility settings for dual bank configuration



Option Bytes

Read Out Protection: Level 0

BOR Level: Level 3

User configuration option byte

☑ IWDG_SW
☐ IWDG_STOP
☑ WWDG_SW
☐ nSRAM_Parity
☐ SRAM2_RST
☐ SRAM2_PE
☐ nRST_SHDW
☑ nRST_STOP
☑ nRST_STDBY
☐ SDADC12_VDD_Monitor

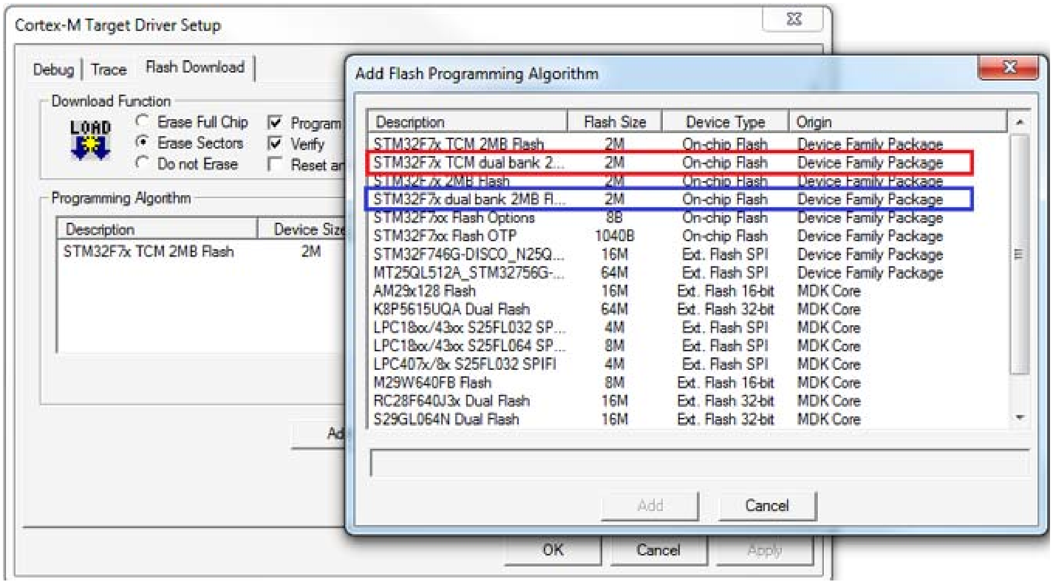
☐ IWDG_STDBY
☐ IWDG_ULP
☑ FZ_IWDG_STOP
☑ FZ_IWDG_STDBY
☐ PCROP_RDP
☐ nBoot0_SW_Cfg
☐ BOOT0 nSW Config
☐ VDDA_Monitor

☐ nBoot0
☐ nBOOT0
☐ nBoot1
☐ BOOT1
☑ nDBOOT
☐ nBOOT_SEL
☐ nBFB2
☐ DUALBANK
☑ nDBANK
☐ DB1M

2. Set the Flash loader dual bank in the toolchain. See *Figure 12*.

   If the user uses the Flash-TCM, the Flash loader (framed in red) must be selected. In case of a Flash-AXI usage the Flash loader (framed in blue) must be selected as shown in *Figure 12*.

Figure 12. Flash loader settings in dual bank mode



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Download Function

○ Erase Full Chip ☑ Program
● Erase Sectors ☑ Verify
○ Do not Erase ☐ Reset ar

Programming Algorithm

Description	Device Size
STM32F7x TCM 2MB Flash	2M

Add Flash Programming Algorithm

Description	Flash Size	Device Type	Origin
STM32F7x TCM 2MB Flash	2M	On-chip Flash	Device Family Package
STM32F7x TCM dual bank 2...	2M	On-chip Flash	Device Family Package
STM32F7x 2MB Flash	2M	On-chip Flash	Device Family Package
STM32F7x dual bank 2MB Fl...	2M	On-chip Flash	Device Family Package
STM32F7xx Flash Options	8B	On-chip Flash	Device Family Package
STM32F7xx Flash OTP	1040B	On-chip Flash	Device Family Package
STM32F746G-DISCO_N25Q...	16M	Ext. Flash SPI	Device Family Package
MT25QL512A_STM32756G-...	64M	Ext. Flash SPI	Device Family Package
AM29x128 Flash	16M	Ext. Flash 16-bit	MDK Core
K8P5615UQA Dual Flash	64M	Ext. Flash 32-bit	MDK Core
LPC18xx/43xx S25FL032 SP...	4M	Ext. Flash SPI	MDK Core
LPC18xx/43xx S25FL064 SP...	8M	Ext. Flash SPI	MDK Core
LPC407x/8x S25FL032 SPIFI	4M	Ext. Flash SPI	MDK Core
M29W640FB Flash	8M	Ext. Flash 16-bit	MDK Core
RC28F640J3x Dual Flash	16M	Ext. Flash 32-bit	MDK Core
S29GL064N Dual Flash	16M	Ext. Flash 32-bit	MDK Core

Note: The user can get back to the single bank configuration by restoring ST-Link/Flash loader older configurations.

2.3 Project configuration of the CPU memory access with DMA activation demonstration

The DMA activation demonstration is provided with the Keil MDK-ARM tool chain. The project configurations are based on the same template described previously in *Section 2.2*. The project has three configurations:

1. FlashITCM_rwRAM-DTCM
2. FlashITCM_rwSRAM1
3. FlashAXI_rwRAM-DTCM

The DMA activation demonstration uses the same configurations than the CPU memory access demonstration with the following differences:

- The SDRAM access is used for all the configurations for DMA transfers.
- The CPU frequency used is 200 MHz instead of 216 MHz for SDRAM access.
- The scatter files are modified to separate the execution/data memories used by the FFT process and the other modules of the demonstration in order to avoid indeterminism inducted by the linker that impacts the results.
- The data cache is enabled for all projects.

The same FFT demonstration already used for the CPU memory access performance (stm32f7_performances) is used for the DMA activation demonstration performance (stm32f7_performances_DMAs).

Only the FFT computation part is kept, the magnitude calculation part is removed in order to allocate more RAM for DMA transfers.

Two masters are provided in the demonstration to perform the transfers: the two general purpose DMAs: DMA1 and DMA2. The DMA2 is configured to perform memory to memory transfers, while the DMA1 is used to perform memory to SPI3 and SPI3 to memory transfers. The memory source and destination are configurable for each DMA separately. The configuration is done in main.h file, which contains all the needed definition to perform DMA1/2 transfers.

To activate the DMA1/SPI3 transfer, the SPI3_MOSI (PC12) must be connected to the SPI3_MISO (PC11). The microSD card must be removed from its slot.

There are three steps to get a result for a given scenario:

- Step 1: configure the different DMA parameters in main.h
- Step 2: check mode: check the configuration and the DMA(s) transfer(s)
- Step 3: result mode: get the results in term of cycles number

2.3.1 Step 1: configure the different DMA parameters

There are basically four parameters for each master to be configured:

- Enable/disable of the DMA transfer
- The DMA source address configuration
- The DMA destination address configuration
- The DMA transfer size configuration

Enable/disable of the DMA transfer:

To enable or disable a DMA the define “#define USE_DMA**X”** has to be settled to 1 or 0 to respectively enable or disable DMA**X**. Where **X** = 1 or 2.

The user can enable each DMA separately or enable them together by setting the defines USE_DMA1 and USE_DMA2 to 1.

The DMA source and destination address configurations:

The memory source and destination addresses should be configured by the following defines:

For the DMA1:

#define DMA**X**_SRC_ADDRESS: the start address of the memory source

#define DMA**X**_DST_ADDRESS: the start address of the memory destination

The DMA1 transfers the data from memory source to memory destination through SPI3 (memory source -> DMA1_Stream5 -> SPI3_TX -> SPI3_RX -> DMA1_Stream0 -> memory destination).

The DMA2 transfers data directly from memory A to memory B.

The DMA**X**_SRC_ADDRESS and DMA**X**_DST_ADDRESS possible values:

- FLASHAXI_DMA_START_ADDRESS
- FLASHTCM_DMA_START_ADDRESS
- DTCMRAM_DMA_START_ADDRESS
- SRAM1_DMA_START_ADDRESS
- SRAM2_DMA_START_ADDRESS
- SDRAM_DMA_START_ADDRESS

Do not modify these defines provided in dma_utilities.h file, which are in line with the mapping configured in the scatter files.

The DMA transfer size configuration:

The DMA**X** transfer size must be configured by the following define:

#define DMA**X**_TRANSFER_SIZE: the transfer size in byte

The transfer size must be the smallest value between the available size of the DMA memory source and the available size of the DMA memory destination. The macro MIN(a,b) is provided for this purpose. This to prevent any DMA to access to an invalid memory zone.

The DMA**X**_TRANSFER_SIZE possible values:

- DTCMRAM_DMA_AVAILABLE_SIZE
- SRAM1_DMA_AVAILABLE_SIZE
- SRAM2_DMA_AVAILABLE_SIZE
- SDRAM_DMA_AVAILABLE_SIZE
- FLASHAXI_DMA_AVAILABLE_SIZE
- FLASHTCM_DMA_AVAILABLE_SIZE

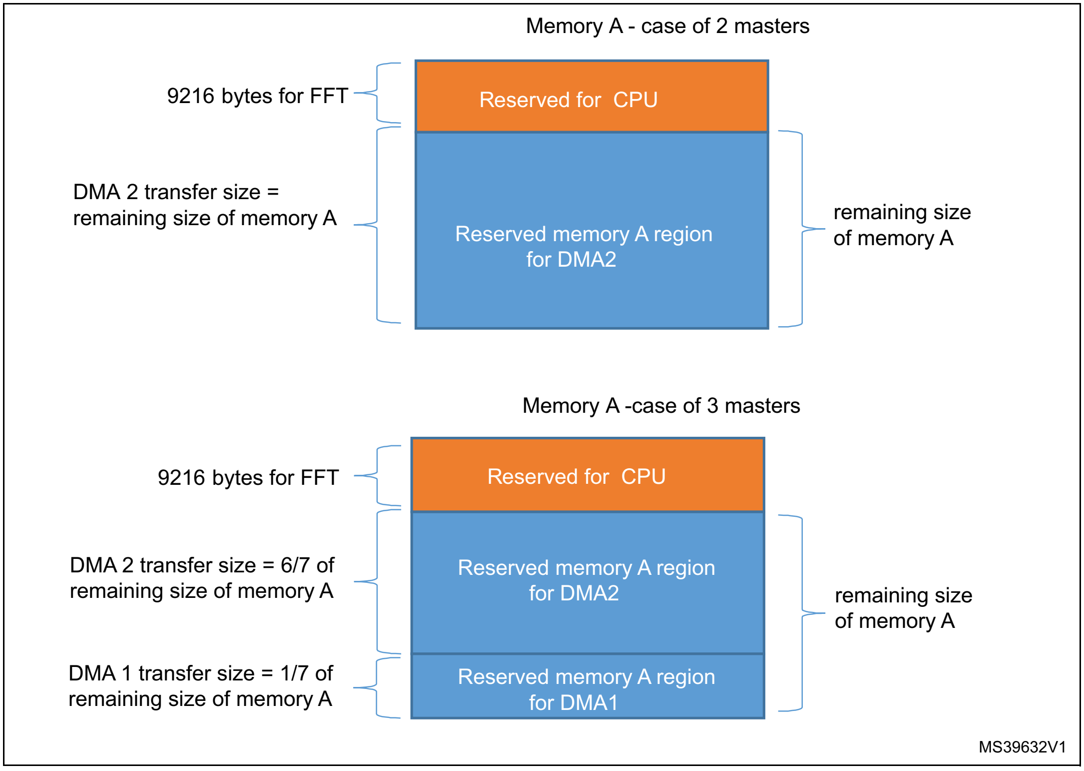
Example of the DMA2 transfer size configuration where the SRAM1 is the source and the DTCMRAM is the destination:

#define DMA2_TRANSFER_SIZE MIN(SRAM1_DMA_AVAILABLE_SIZE, DTCMRAM_DMA_AVAILABLE_SIZE)

If the minimum size is greater than 64 Kbytes the transfer size is forced to 64 Kbytes.

When the user runs DMA**X** with concurrency with the CPU on memory **A**, he must split the memory as shown in *Figure 13*. The figure shows how to partition a memory **A** (for example the SRAM1) in case of concurrency between two and three masters in the software. The partitioning at the top of the figure shows the memory **A** shared between the CPU and the DMA2. The partitioning at the bottom of the figure shows the memory **A** shared between three masters: CPU, DMA1 and DMA2. This partitioning allows each DMA to complete its transfer before the FFT process to ensure that concurrency is guaranteed during all the FFT process. The RAM size reserved for the FFT is 9216 bytes in the scatter file (for CPU access).

Figure 13. How to split a memory A in case of concurrency between all the masters


Memory A - case of 2 masters
9216 bytes for FFT
Reserved for CPU
DMA 2 transfer size = remaining size of memory A
Reserved memory A region for DMA2
remaining size of memory A
Memory A -case of 3 masters
9216 bytes for FFT
Reserved for CPU
DMA 2 transfer size = 6/7 of remaining size of memory A
Reserved memory A region for DMA2
remaining size of memory A
DMA 1 transfer size = 1/7 of remaining size of memory A
Reserved memory A region for DMA1
MS39632V1

Two examples below to show how to configure The DMA1 and the DMA2 in two scenarios (see *Section 3.2: CPU memory access performance with DMA usage on page 36*):

Configuration 2 /scenario 3:

```
/* DMA2 */
#define DMA2_SRC_ADDRESS        DTCMRAM_DMA_START_ADDRESS
#define DMA2_DST_ADDRESS        SRAM1_DMA_START_ADDRESS
#defineDMA2_TRANSFER_SIZE       MIN (DTCMRAM _REMAINING_SIZE,
SRAM1_REMAINING_SIZE)
```

Configuration 1 /scenario 4:

```
/* DMA2 */
#define DMA2_SRC_ADDRESS          SRAM1_DMA_START_ADDRESS
#define DMA2_DST_ADDRESS          DTCMRAM_DMA_START_ADDRESS
#defineDMA2_TRANSFER_SIZE         MIN (SRAM1_REMAINING_SIZE,
(DTCMRAM_REMAINING_SIZE*6/7))

/* DMA1 */
#define DMA1SPI_SRC_ADDRESS       SRAM2_DMA_START_ADDRESS
#define DMA1SPI_DST_ADDRESS       (DTCMRAM_DMA_START_ADDRESS +
(DTCMRAM_REMAINING_SIZE*6/7))
#define DMA1SPI_TRANSFER_SIZE     MIN (SRAM2_REMAINING_SIZE,
(DTCMRAM_REMAINING_SIZE/7))
```

2.3.2 Step 2: check the configuration and the DMA transfer(s)

After the configuration of the different parameters of the DMAs, the user must check this configuration and the DMA transfer(s) before getting the results by running the example in “check mode”.

The check is done for two purposes:

- The DMA transfer integrity of data is correct.
- The DMA transfer is completed after the FFT process to ensure that during all the FFT process the DMA still stresses the bus matrix.
- No hard fault has occurred (the LCD displays all the configurations and there is at least a LED lit).

If any of the above conditions is not matched, the result is not consistent and the user has to modify the DMA configuration(s).

To run any scenario in “check mode”, the following define should be set to 1:

```
#define CHECK_TRANSFER              1
```

Then compile the example and run it.

If the three check conditions are met, the LCD displays the current configuration and the LED4 (blue LED) is lit. In this case, the chosen scenario is ready for Step 3 to get the results.

If the LED2 is lit, there is at least one DMA transfer that has not been performed properly. The error message is displayed on the LCD showing which DMA has the error and the offset of the wrong data. At this stage the user has to verify the memory address ranges used by the DMA(s) and recompile the example in order to have only the LED4 on.

If the LED2 is blinking, the data transfers integrity is correct but there is at least one DMA transfer that is completed before the FFT transfer completion. An error message is displayed that provides which DMA has the issue. At this stage the user has to increase the transfer size of this DMA.

If a hard fault occurred, or the application crushed, this indicates that a DMA overwrote data used by the application including the stack. The user must be careful on the address choices accessed by the DMA mainly when the CPU, the DMA1 and the DMA2 access the same memory to reproduce the concurrency presented in *Figure 7*.

Note that when CHECK_TRANSFER = 0, the DMA1 transfers data in circular mode while it transfers data in normal mode when CHECK_TRANSFER = 1. Regardless of the value of this define, the DMA2 transfers the data in normal mode.

2.3.3 Step 3: get the results

If only the LED4 is lit and the configuration is displayed on the LCD, the result will be consistent and ready to be got for this scenario.

At this stage, the user can recompile the example in “result mode” (CHECK_TRANSFER = 0), load it and run it. The LCD displays the current configuration as well as the cycles number at the bottom of the display.

3 Results and analysis

3.1 CPU memory access performance

This section explains each feature activation following the configuration used and presents the obtained results in term of number of cycles consumed by the FFT process.

For the STM32F74xxx and STM32F75xxx devices, the results are obtained with the KEIL MDK-ARM v5.14.0 tool chain, the STM32F7xx pack version 1.1.0 and the Cube firmware version 1.0.0.

For the STM32F76xxx and STM32F77xxx devices, the results are obtained with the KEIL MDK-ARM v5.17.0 tool chain, the STM32F7xx pack version 2.6.0 and the Cube firmware version 1.4.0.

The MDK-ARM code optimization configuration is level 3 (optimize for time).

For all the configurations that have the code execution from the Flash-ITCM, the ART accelerator and the ART-prefetch are enabled since the instructions and read-only data (constant variables) flow through the ITCM (path 1 in *Figure 2*). This is the case of the following configurations:

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

For all the configurations that have the code execution from the Flash-AXI (AXI/AHB), the instruction cache is enabled since the instructions flow through the path 2 in *Figure 2*. In this configuration, the read-only data (constant variables) flow also through the path 2 (same figure) that is why the data cache is also enabled. This is the case of the following configurations:

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

In the case of “5-RamITCM_rwRAM-DTCM” configuration, nothing is enabled since the code and data are located in the ITCM-RAM and the DTCM-RAM respectively.

In the case of “6-Quad SPI_rwRAM-DTCM” configuration, two cases are present to show the difference of the performance between:

- Locating the read-only data in the Quad-SPI Flash memory, that is, the same location as the instruction (case 1).
- Locating the read-only data in a memory other than the Quad-SPI Flash memory, it will be in the Flash-ITCM (case 2).

Since the instruction and data are located in the Quad-SPI Flash memory for case 1, only the I-cache and D-cache are enabled.

For case 2, since the read-only data are located in the Flash-ITCM, the ART and the ART-prefetch are enabled and the D-cache is OFF. For the two cases, the read/write data are located in the DTCM-RAM.

For all the configurations that have the access to the DTCM-RAM as the data load/store memory (path 5 in *Figure 3*), they do not have any specific feature activation since the core is coupled directly to this RAM with 0-wait state access.

For all the configurations that have the access to the SRAM1 as the data load/store memory (path 7 in *Figure 3*), the data cache is enabled. This is the case for the following configurations:

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

If the application has the access to the external memory through the FMC for the data load/store data memory and/or the code execution (path 9 in *Figure 4*), the data and/or the instruction caches are enabled. This is the case for the following configuration:

- “7-ExtSDRAM-Swapped_rwDTCM”

Note: For this configuration, the SDRAM has been swapped (remapped from 0xC000 0000 to 0x6000 0000 in order to allow the cache usage) since the default MPU attribute region starting from 0xA000 0000 to 0xDFFF FFFF is not cacheable as it is a Device memory type region.

Results

The results are obtained with the STM32756G-EVAL and STM32769I-EVAL boards. The CPU running at 216 MHz, V_{DD}=3.3 V and with 7-wait state access to the internal Flash memory.

Table 6 and *Table 7* show the obtained results for the FFT demonstration for MDK-ARM in each configuration for the STM32F74xxx/STM32F75xxx and the STM32F76xxx/STM32F77xxx devices respectively:

Table 6. MDK-ARM results for the STM32F74xxx and STM32F75xxx devices

Feature configuration	Memory location configuration	CPU cycle number[1]
ART + ART-PF ON	1-FlashITCM_rwRAM-DTCM	118577
ART + ART-PF + D-cache ON	2-FlashITCM_rwSRAM1	135296
I-cache + D-cache ON	3-FlashAXI_rwRAM-DTCM	118653
I-cache + D-cache ON	4-FlashAXI_rwSRAM1	145917
-	5-RAMITCM_rwRAM-DTCM	112428
I-cache + D-cache ON (Const data in Quad-SPI)	6_1-QuadSPI_rwRAM-DTCM	176441
I-cache + ART + ART-PF ON (Const data in Flash TCM)	6_2-QuadSPI_rwRAM-DTCM	126900
I-cache + D-cache ON (Const data in SDRAM)	7-ExtSDRAM-Swapped_rwDTCM[2]	128398

[1] The cycles number values may change from a version to another of the tool chain.

[2] HCLK is running at 200 MHz.

Table 7. MDK-ARM results for the STM32F76xxx and STM32F77xxx devices (single bank/single precision FPU)

Feature configuration	Memory location configuration	CPU cycle number[1]
ART + ART-PF ON	1-FlashITCM_rwRAM-DTCM	117059
ART + ART-PF + D-cache ON	2-FlashITCM_rwSRAM1	115004
I-cache + D-cache ON	3-FlashAXI_rwRAM-DTCM	108491
I-cache + D-cache ON	4-FlashAXI_rwSRAM1	112372
-	5-RAMITCM_rwRAM-DTCM	106275
I-cache + D-cache ON (Const data in Quad-SPI)	6_1-QuadSPI_rwRAM-DTCM	142177
I-cache + ART + ART-PF ON (Const data in Flash TCM)	6_2-QuadSPI_rwRAM-DTCM	120315
I-cache + D-cache ON (Const data in SDRAM)	7-ExtSDRAM-Swapped_rwDTCM[2]	117744

1. The cycles number values may change from a version to another of the tool chain.
2. HCLK is running at 200 MHz.

Table 8. MDK-ARM results for the STM32F76xxx and STM32F77xxx devices (dual bank/single precision FPU)

Feature configuration	Memory location configuration	CPU cycle number[1]
ART + ART-PF ON	1-FlashITCM_rwRAM-DTCM	123237
ART + ART-PF + D-cache ON	2-FlashITCM_rwSRAM1	119645
I-cache + D-cache ON	3-FlashAXI_rwRAM-DTCM	109209
I-cache + D-cache ON	4-FlashAXI_rwSRAM1	112844

1. The cycles number values may change from a version to another of the tool chain.

In *Table 8* the configurations 5, 6_1, 6_2 and 7 are removed since they do not depend on the internal Flash memory. They have the same performance as the single bank configuration.

The charts in *Figure 14* show the relative ratio of each configuration versus the configuration 5 that has the best results.

Relative ratio = cycles_number_config_X / cycles_number_config_5

The relative ratio calculation allows to compare the performance versus the best performance (5-RamITCM_rwRAM-DTCM) and to compare a configuration with another one.

Figure 14. STM32F74xxx and STM32F75xxx FFT relative ratio cycle number with MDK-ARM

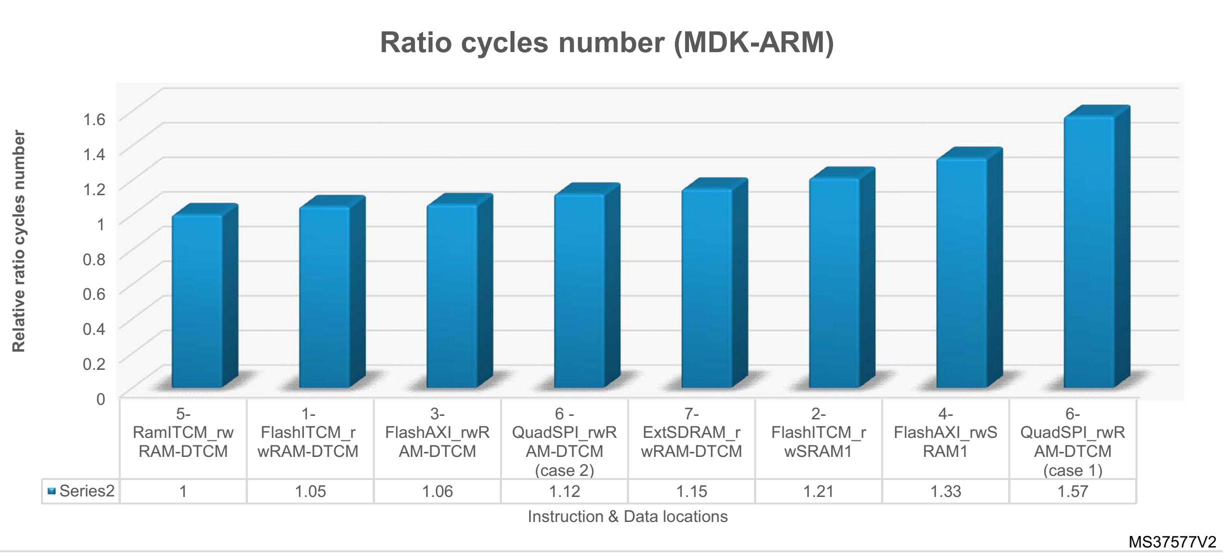
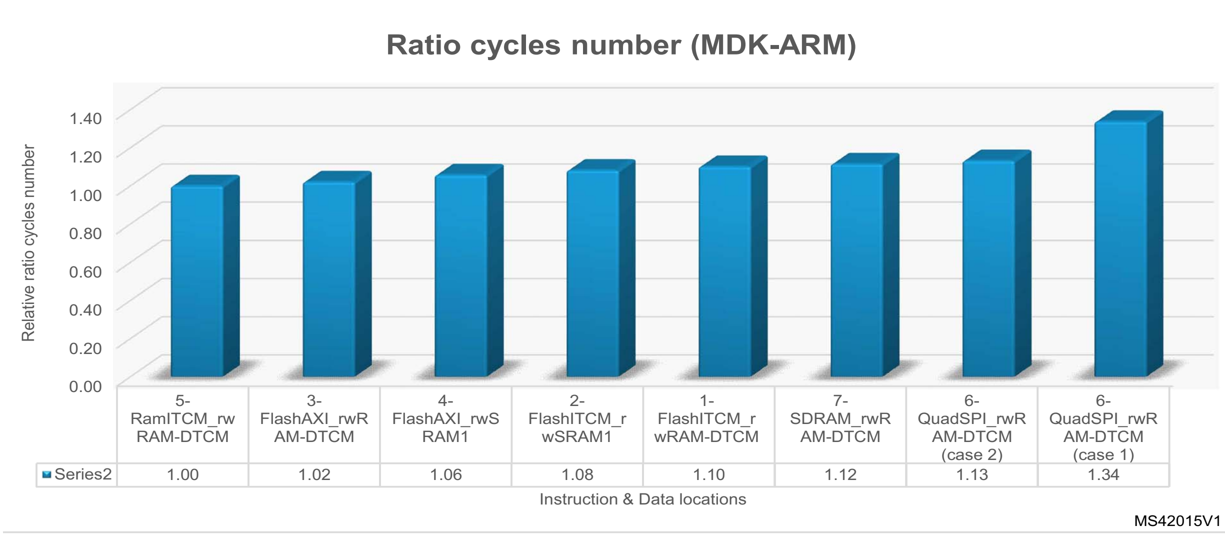


Figure 15. STM32F76xxx and STM32F77xxx FFT relative ratio cycle number with MDK-ARM



Analysis

STM32F74xxx and STM32F75xxx devices:

If the user fixes the data memory location to the DTCM-RAM and varies the code execution memory location, the relative ratios demonstrate that whether the code is executed from the Flash-ITCM or the Flash-AXI, the performance is almost the same if the ART or the instruction cache are respectively enabled.

This is the case for the configurations:

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

If the user fixes the execution memory location to the Flash-ITCM or the Flash-AXI and varies the data memory location, the respective ratios show that no matter which internal RAM is used for data load/store, the results are almost the same since the data cache is enabled for RAMs connected to the AXI/AHB.

This is the case for the couple of configurations:

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

If the ratio of the “7-ExtSDRAM_Swapped_rwDTCM” configuration is compared to the “1-FlashITCM_rwRAM-DTCM” or “3-FlashAXI_rwRAM-DTCM” configurations, the results are close since the cache is enabled for this configuration. Thus, the code execution or the data storage from the external memory does not impact the performance.

Note that the case of the “5-RamITCM_rwRAM-DTCM” configuration has the best performance (the lowest cycle number). It is due to the fact that there is no contention on the ITCM and the DTCM buses since no cache maintenance is performed (if compared to the cache usage). Also, there is no concurrent access of the CPU to the DTCM-RAM with other masters in the provided FFT example.

If the two ratios of the case 1 and case 2 of the “6-Quad SPI_rwRAM-DTCM” configuration (case X) are compared, note that there is a significant difference in term of a performance since the demonstration uses a huge constant data.

For the case 1 (6_1-Quad SPI_rwRAM-DTCM), since the read-only data and the instructions are located in the Quad-SPI Flash memory, a latency occurs due to the concurrency access of the instruction fetch and the read-only data loaded on the bus matrix.

For the case 2 (6_2-Quad SPI_rwRAM-DTCM), the read-only data and code are separated. The read-only data are located in the Flash-TCM, therefore, the concurrency of the read-only data and the instruction fetch is avoided and the CPU can fetch the instruction from AXI and the data loaded from TCM at the same time. That is why the performance of the second case is clearly better than the first one.

STM32F76xxx and STM32F77xxx devices:

For these devices the analysis is almost the same as for the STM32F74xxx and STM32F75xxx devices except in case when the cache is used.

If the configuration 5 of *Table 6* and the configuration 5 of *Table 7* are compared, one can notice that the performance is increased of about 5% for the STM32F76xxx and STM32F77xxx devices versus the STM32F74xxx/ STM32F75xxx devices. This is possible thanks to some enhancements that have been done in the FPU of the Cortex-M7 version, which is embedded in the STM32F76xxx and STM32F77xxx devices.

Thanks to the cache size of the STM32F76xxx and STM32F77xxx devices, the performance is increased in all the cases using the cache and AXI path.

For example the configurations 3 and 4 lead the first place ranking versus the STM32F74xxx and STM32F75xxx devices thanks to the increase of the cache size from 4 Kbytes to 16 Kbytes.

In the case of the Quad-SPI and more precisely in the configuration 6_2, the performance is increased of about 19.5% versus the STM32F74xxx and STM32F75xxx devices including the performance increase thanks to the FPU enhancement.

In the case of external memories such as the SDRAM, the performance is also increased of about 10% versus the STM32F74xxx and STM32F75xxx devices thanks to the cache size increase and including the performance increase thanks to the FPU enhancement.

In the case of a dual bank usage and even if the Flash memory interface width is decreased from 256-bit down to 128-bit, the performance is similar to or better than the STM32F74xxx and STM32F75xxx device performance for some cases when the Flash memory is accessed through the AXI/AHB and the cache is enabled.

Table 9. Single bank versus dual bank performance comparison

Project Configuration	Single bank mode (CPU cycle number)	Dual bank mode (CPU cycle number)	Decrease dual bank vs Single bank (%)
1-FlashITCM_rwRAM-DTCM	117059	123237	5.28
2-FlashITCM_rwSRAM1	115004	119645	4.04
3-FlashAXI_rwRAM-DTCM	108491	109209	0.66
4-FlashAXI_rwSRAM1	112372	112844	0.42

Table 9 shows that in case of a Flash memory access through the AXI/AHB, the performance is the same either in single or dual bank mode. If the Flash memory access is done through the TCM, the performance is decreased of about 5% with the current algorithm.

3.2 CPU memory access performance with DMA usage

This section treats the architecture performance where one or more masters are activated.

For the STM32F74xxx and STM32F75xxx devices, the results are obtained with the KEIL MDK-ARM v5.16.0 tool chain, the STM32F7xx pack version 2.2.0 and the cube firmware version 1.0.0.

For The STM32F76xxx and STM32F77xxx devices, the results are obtained with the KEIL MDK-ARM v5.17.0 tool chain, the STM32F7xx pack version 2.6.0 and the cube firmware version 1.4.0.

The MDK-ARM code optimization configuration is level 3: optimize for time.

Results

Table 10, *Table 11*, and *Table 12* summarize the number of cycles consumed by the FFT process in different scenarios and the performance decrease in percent in case where one or more masters access to the same memory as the CPU.

The results are obtained with the STM32756G-EVAL and STM32769I-EVAL boards, the CPU at 200 MHz (to allow the SDRAM access @100 MHz), VDD=3.3 V and with 6-wait state access to the internal Flash memory. The estimated error of measurement is 0.4% due to some alignments of data and instructions in the Flash memory, and in the cache.

The conditions of the results:

- The DMA1 and the DMA2 access priority level has been configured to the high priority and the data size is a byte.
- The SPI3 baudrate is 25 Mbit/s.
- The CM7_AHBSCR register is kept at its reset value.

STM32F74xxx and STM32F75xxx devices:

Table 10. Configuration 1: execution from the Flash-ITCM / FFT CPU data storage in the DTCM-RAM

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
1	DMA2	-	-	69247	-
	DMA1_SPI3	-	-		
2	DMA2	SDRAM	SRAM1	69247	0
	DMA1_SPI3	-	-		
3	DMA2	SRAM1	**DTCM-RAM**(1)	70929	2.43
	DMA1_SPI3	-	-		
4	DMA2	SRAM1	**DTCM-RAM**(1)	71275	2.93
	DMA1_SPI3	SRAM2			
5	DMA2	SDRAM	**DTCM-RAM**(1)	70830	2.29
	DMA1_SPI3	SRAM2			
6	DMA2	**DTCM-RAM**(2)	SRAM1	70140	1.29
	DMA1_SPI3	-	-		
7	DMA2	**Flash-AXI**(2)	SRAM1	69596	0.50
	DMA1_SPI3	-	-		

1. Memories with a concurrency on write access between CPU and DMA(s).
2. Memories with a concurrency on read access between CPU and DMA(s).

Table 11. Configuration 2: execution from the Flash-ITCM / FFT CPU data storage in the SRAM1

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
1	DMA2	-	-	79925	-
	DMA1_SPI3	-	-		
2	DMA2	SDRAM	SRAM2	79925	0
	DMA1_SPI3	-	-		
3	DMA2	DTCM-RAM	**SRAM1**(1)	81720	2.25
	DMA1_SPI3	-	-		
4	DMA2	DTCM-RAM	**SRAM1**(1)	81828	2.38
	DMA1_SPI3	SRAM2			

Table 11. Configuration 2: execution from the Flash-ITCM / FFT CPU data storage in the SRAM1 (continued)

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
5	DMA2	SDRAM	**SRAM1**[1]	81674	2.19
	DMA1_SPI3	SRAM2			
6	DMA2	**SRAM1**[2]	DTCM-RAM	81410	1.86
	DMA1_SPI3	-	-		

1. Memories with a concurrency on write access between CPU and DMA(s).
2. Memories with a concurrency on read access between CPU and DMA(s).

Table 12. Configuration 3: execution from the Flash-AXI / FFT CPU data storage in the DTCM-RAM

Scenario	Master	Source	Destination	FFT cycles number	Degradation (%)
1	DMA2	-	-	68206	-
	DMA1_SPI3	-	-		
2	DMA2	**Flash-AXI**[1]	SRAM1	68865	0.97
	DMA1_SPI3	-	-		

1. Memories with a concurrency on read access between CPU and DMA(s).

STM32F76xxx and STM32F77xxx devices:

Only the results of the Flash memory in single bank mode are presented.

Table 13. Configuration 1: execution from the Flash-ITCM / FFT CPU data storage in the DTCM-RAM (single bank/single precision FPU)

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
1	DMA2	-	-	69250	-
	DMA1_SPI3	-	-		
2	DMA2	SDRAM	SRAM1	69250	0
	DMA1_SPI3	-	-		
3	DMA2	SRAM1	**DTCM-RAM**[1]	70873	2.34
	DMA1_SPI3	-	-		
4	DMA2	SRAM1	**DTCM-RAM**[1]	71191	2.80
	DMA1_SPI3	SRAM2			
5	DMA2	SDRAM	**DTCM-RAM**[1]	70825	2.27
	DMA1_SPI3	SRAM2			
6	DMA2	**DTCM-RAM**[2]	SRAM1	70150	1.30
	DMA1_SPI3	-	-		

Table 13. Configuration 1: execution from the Flash-ITCM / FFT CPU data storage in the DTCM-RAM (single bank/single precision FPU) (continued)

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
7	DMA2	**Flash-AXI**[2]	SRAM1	69587	0.49
	DMA1_SPI3	-	-		

[1] Memories with a concurrency on write access between CPU and DMA(s).

[2] Memories with a concurrency on read access between CPU and DMA(s).

Table 14. Configuration 2: execution from the Flash-ITCM / FFT CPU data storage in the SRAM1 (single bank/single precision FPU)

Scenario	Master	Source	Destination	FFT cycles number	Decrease (%)
1	DMA2	-	-	72648	-
	DMA1_SPI3	-	-		
2	DMA2	SDRAM	SRAM2	72648	0
	DMA1_SPI3	-	-		
3	DMA2	DTCM-RAM	**SRAM1**[1]	73071	0.58
	DMA1_SPI3	-	-		
4	DMA2	DTCM-RAM	**SRAM1**[1]	73086	0.60
	DMA1_SPI3	SRAM2			
5	DMA2	SDRAM	**SRAM1**[1]	72964	0.43
	DMA1_SPI3	SRAM2			
6	DMA2	**SRAM1**[2]	DTCM-RAM	73082	0.60
	DMA1_SPI3	-	-		

[1] Memories with a concurrency on write access between CPU and DMA(s).

[2] Memories with a concurrency on read access between CPU and DMA(s).

Table 15. Configuration 3: execution from the Flash-AXI / FFT CPU data storage in the DTCM-RAM (single bank/single precision FPU)

Scenario	Master	Source	Destination	FFT cycles number	Degradation (%)
1	DMA2	-	-	65897	-
	DMA1_SPI3	-	-		
2	DMA2	**Flash-AXI**[1]	SRAM1	66155	0.39
	DMA1_SPI3	-	-		

[1] Memories with a concurrency on read access between CPU and DMA(s).

Table 16. Configuration 3: execution from the Flash-AXI / FFT CPU data storage in the DTCM-RAM (dual bank/single precision FPU)

Scenario	Master	Source	Destination	FFT cycles number	Degradation (%)
1	DMA2	-	-	66370	-
	DMA1_SPI3	-	-		
2	DMA2	**Flash-AXI**[1]	SRAM1	66960	0.89
	DMA1_SPI3	-	-		

[1] Memories with a concurrency on read access between CPU and DMA(s).

Analysis

STM32F74xxx and STM32F75xxx devices:

1. If the CPU and one or more masters do not access the same memory, no contention is present. The same result is obtained from scenario 1 and 2 for the configuration 1 and 2. Refer to the results given by:
   - Configuration 1/scenario 2 versus configuration 1/scenario 1.
   - Configuration 2/scenario 2 versus configuration 2/scenario 1.
2. When the CPU and one DMA access the same memory, a contention occurs since there is an arbitration either on the bus matrix for memories connected to the bus matrix including external memories (SRAM1, SRAM2, SDRAM, etc...) or to the CPU side (Tightly-Coupled Unit interface: TCU unit) where the concurrency is performed at the DTCM-RAM level. Refer to the results given by:
   - Configuration 1/scenario 1 versus configuration 1/scenario 3.
   - Configuration 2/scenario 1 versus configuration 2/scenario 3.

As shown by the results, the concurrency of the CPU and the DMA2 on write on the DTCM-RAM is about 2.5% of the performance decrease.

SRAM1/SRAM2: the CPU and DMA2 concurrency on write access is about 2.5% of the performance decrease.

3. There is an increase of about 0.5 % of latency when additional master has a write access to the same memory as the CPU. Refer to the results given by:
   - Configuration 1/scenario 3 versus configuration 1/scenario 4.
   - Configuration 2/scenario 3 versus configuration 1/scenario 4.
4. The contention effect on read on a given memory between the CPU and a DMA is less than a contention effect on write access. Refer to the results given by:
   - Configuration 1/scenario 3 versus configuration 1/scenario 6.
   - Configuration 2/scenario 3 versus configuration 2/scenario 6.

As shown by the results, the concurrency of the CPU and the DMA2 on read access at the DTCM-RAM level is between 1% and 2% of the performance decrease.

SRAM1/SRAM2: the CPU and DMA2 concurrency on read access is between 1.5% and 2.5% of the performance decrease.

5. If the memory source is relatively slow, such as the SDRAM, the concurrency effect decreases since there is some latency on this memory, which provides a bandwidth for the CPU access to this memory on the bus matrix. Refer to the results given by:

   - Configuration 1/scenario 4 versus configuration 1/scenario 5.
   - Configuration 2/scenario 4 versus configuration 2/scenario 5.

In the configuration 1, the DTCM-RAM in scenario 4 is replaced by the SDRAM in scenario 5. The same case applies for the configuration 2.

6. There is no significant contention when executing data from the Flash-AXI or the Flash-ITCM and at the same time performing a transfer from the Flash memory to another memory. This due to the fact that the access of the DMA to the Flash memory is slow compared to the CPU access that uses the cache. The DMA access to the Flash memory is cadenced by the number of wait states configured in the Flash memory interface (in our case six wait states) while the instructions/data are loaded in the cache at the CPU side or in the ART in the case of the Flash-ITCM access. This configuration allows to cancel the latency effects. Refer to the results given by:

   - Configuration 1/scenario 7 versus configuration 3/scenario 2.

Since the DMA1 can perform transfers between memory and peripheral, the impact of the concurrency on the memory between the CPU and the DMA1 depends on the speed of the peripheral. The faster the peripheral is, the more visible the impact of the concurrency is. For example if the SPI3 that has the maximum speed of 25 MHz is replaced by the SPI4 that has the maximum speed of 50 MHz, the latency will be more important since the data transfer bandwidth is high. Nevertheless, the latency is still reasonable.

In summary, there is a very low performance decrease of about 3% due to the concurrency between the CPU and the DMA in all the scenarios for all different memories access on read or write (DTCM-RAM, SRAMx, external memory). The overall performance is not impacted by the multi-master concurrency thanks to the smart system architecture and bus matrix.

STM32F76xxx and STM32F77xxx devices:

For the STM32F76xxx and STM32F77xxx devices the analysis is almost the same as for the STM32F74xxx and STM32F75xxx devices except when the AXI/AHB is used for the CPU access.

In *Table 13* the performance is the same as that of the STM32F74xxx and STM32F75xxx devices (see *Table 10*) because the devices are in the same architecture configuration.

But in *Table 14* the performance decrease does not exceed 0.6%, since the access concurrency to a memory between the CPU and one or more masters does not impact the performance. This is possible thanks to the effect of the cache size increase of the STM32F76xxx and STM32F77xxx devices.

In the case of the dual bank Flash configuration (see *Table 16*), and if the access concurrency is done to the Flash memory, the performance decrease is the same as the STM32F74xxx and STM32F75xxx devices. However in the case of the single bank Flash configuration (see *Table 15*), the decrease of the performance is insignificant as if there was no access concurrency.

4 Software memory partitioning and tips

This section provides some tips on how to partition the code and the data in the STM32F7 memory to get the best compromise performance depending on the code and the data sizes.

4.1 Software memory partitioning

As the CPU has a direct access to TCM memories with 64-bit wide and 0-wait state access, the DTCM-RAM and the ITCM-RAM locations are the best locations for the read/write data and the instruction fetch respectively.

Therefore, the ITCM-RAM (16 Kbytes) is reserved for a critical code with a deterministic execution, such as interrupt handlers that cannot wait for cache misses and some critical control loops targeting motor control applications.

The DTCM-RAM is reserved for the data load/store regular access and for a critical real time and data, such as stack and heap. In real time applications that use RTOS, generally, heap is massively used. For example if the DTCM-RAM is 64 Kbytes and if that size is enough for the application, the DTCM-RAM scattering (32 Kbytes for heap, 8 Kbytes for stack and 24 Kbytes for global variables) would be relatively reasonable. In foreground/background applications, heap is not almost used, the DTCM-RAM will be scattered between stack and global variables.

When the code size of the user application fits into the internal Flash memory, the latter would be the best execution region either:

- Through TCM (Flash-ITCM) by enabling the ART-accelerator or
- Through AXI/AHB by enabling the cache in order to reach 0-wait state at 216 MHz

Note that the execution from the Flash-ITCM/data in the DTCM-RAM and the Flash-AXI/data in the DTCM-RAM have the same CoreMark score, which is 5 CoreMark/MHz.

The SRAM1 can be reserved for a graphic frame buffer in graphic applications using QVGA TFTs in 16-bit mode that need a relatively huge graphic data and performance of display, achieved by the LCD-TFT and the DMA2D DMAs.This memory can be used also for the data load/store when no more space is available in the DTCM-RAM. In that case a region from the SRAM1, with the data cache enabled, can be reserved for global variables to leave more space for critical data.

The SRAM2 can be reserved for peripherals such as Ethernet and USB to store data such as buffers and descriptors. This memory can also be used for global variables. When the application needs more memory and its code or data, or both, do not fit in the internal memories, the external memories can be used to expand the memory size without a loss of performance.

For example an external NOR Flash memory up to 250 Mbytes of size connected through the FMC can contain the application instructions with the instruction cache enabled, while the constant data are located in the internal Flash memory (by enabling the ART or the cache). This is the case where huge constants are used in the application and to avoid a concurrency access of instructions and data on the same path (path 9 in *Figure 4*) on the bus matrix.

In case of a lack of internal RAMs, the data storage can be achieved on the external SRAM or the SDRAM through the FMC interface and by enabling the data cache. These memories can contain either frame buffers for graphic applications or for non-critical data. At the same time, more priority is given for very critical data to be fitted in the DTCM-RAM.

The Quad-SPI Flash memory could be used either to store read-only data (relatively huge image files or waves) or to keep almost the same level of performance as an internal Flash memory access by enabling the data cache.

The Quad-SPI Flash memory can be used also to contain the application in the memory mapped mode up to 256 Mbytes and at the same time to save several GPIOs in smaller STM32F7 packages, compared to parallel Flash memories that have to be connected to the FMC interface. In that case, when the CPU accesses regularly to read-only data, the latter should be mapped in the internal Flash memory. If the application needs more size and more execution performance the user can load his application in the Quad-SPI (load region) and use an external SDRAM, where the application is copied and executed (execution region).

4.2 Tips

In the case where the DMA accesses to the DTCM-RAM at the same time as the CPU, the application speed can decrease if the CPU does not have the highest priority access. This priority can be managed by software using the CM7_AHBSCR register in the critical code section that loads/stores data in the DTCM-RAM.

To get the best performance with the STM32F76xxx and STM32F77xxx devices, and in the case of dual bank usage, it is recommended to access the Flash memory through the AXI/AHB interface instead of the TCM to take advantage of the performance induced by the cache size.

When an external memory is used for an execution region, care should be taken if this memory is mapped in a region having the default attribute Execute-Never (XN). In that case the user has to modify the memory into an executable region using the MPU, otherwise a hard fault occurs. This is the case of the SDRAM bank regions after reset. Refer to *Table 2* for the default executable regions of ARM.

When an external memory is used for the data load or store and it is not mapped in a cacheable region, either use the remapping when the relocation to a cacheable region is possible, which is the case for SDRAM banks (SWP_FMC [1:0] = 01 in the SYSCFG_MEMRMP register) or use the MPU to modify the memory type to Normal-type memory.

When huge constants are loaded by the CPU in the application (which is the case of the FFT), the user should verify if they are located in the correct memory location and if they are intended to be located other than in the internal Flash memory. In some cases, if the variables are unintentionally located in the internal Flash memory (TCM or AXI) and the ART accelerator or D-cache, or both, are not enabled, the application bangs all the Flash wait states and the application slows down significantly.

When it is possible, try to separate the data and code locations especially when their memory is connected to the AHB/AXI, in order to avoid a concurrent access on the bus matrix.

It is not recommended to enable the cache before calling the main function, that is, before the scatter load phase, because a hard fault may occur.

5 Conclusion

Today, the applications are becoming more and more complex and require more microcontroller efficiency and performance. Thanks to the STM32F7 smart architecture, the STM32F7 Series device is a suitable platform for the Internet of Things (IoT) application. It enables the developers that need a fast microcontroller to ease the code optimization.

A better application responsiveness is obtained thanks to STM32F7 two independent mechanisms to reach 0-wait execution performance: ST's ART Accelerator™ for an internal Flash memory and the L1-cache (instruction and data caches) for an internal Flash memory and other memories. Thanks to the cache, the user spends less time optimizing the code and the data size by adding external memory resources with no performance penalty. Even the DMA transfers are done at the same time as the CPU, the performance is not impacted thanks to the STM32F7 system architecture.

For a higher performance the user can choose the STM32F76xxx and STM32F77xxx devices to take advantage of the cache size increase.

6 Revision history

Table 17. Document revision history

Date	Revision	Changes
19-Jun-2015	1	Initial release.
18-Nov-2015	2	Updated cover page. Updated *Figure 2: Flash memory interfaces (pathes: 1, 2, 3, 4)* and *Figure 4: External memory interfaces (pathes: 9, 10)*. Updated *Section 1.6: DMAs* description adding *Figure 6: No memory concurrency between DMA and CPU* and *Figure 8: FFT example block diagram*. Updated *Section 2: Typical application* adding *Section 2.3: Project configuration of the CPU memory access with DMA activation demonstration*. Updated *Section 3: Results and analysis* adding *Section 3.2: CPU memory access performance with DMA usage*.
27-May-2016	3	Updated *Section 1.2: Cortex®-M7 system caches* adding STM32F76xxx and STM32F77xxx cache sizes. Updated *Section 1.4: STM32F7 bus matrix* description. Updated *Figure 1: STM32F7 Series system architecture* L1-cache, memory size and adding note 1. Updated *Section 1.5: STM32F7 memories*: – Updated *Figure 1.5.1: Embedded Flash memory* adding STM32F76xxx and STM32F77xxx Flash memory description. – Updated *Figure 2: Flash memory interfaces (pathes: 1, 2, 3, 4)* L1-cache size and adding note 1. Updated *Section 1.5.2: Embedded SRAM*: – Added *Table 4: internal memory summary of the STM32F76xxx/STM32F77xxx devices*. – Updated *Figure 3: Flash memory interfaces (pathes: 5, 6, 7, 8)* and *Figure 4: External memory interfaces (pathes: 9, 10)* removing L1-Cache size. Added *Section 1.7: Main differences between the STM32F7 Series devices from architecture point of view*. Updated *Section 2.2: Project configuration of the CPU memory access demonstration*: – Added *Figure 11: ST-Link utility settings for dual bank configuration*. – Added *Figure 12: Flash loader settings in dual bank mode*.

Table 17. Document revision history

Date	Revision	Changes
27-May-2016	3 (continued)	Updated *Section 3.1: CPU memory access performance*. – Added *Table 7: MDK-ARM results for the STM32F76xxx and STM32F77xxx devices (single bank/single precision FPU)*. – Added *Table 8: MDK-ARM results for the STM32F76xxx and STM32F77xxx devices (dual bank/single precision FPU)*. – Updated *Figure 14: STM32F74xxx and STM32F75xxx FFT relative ratio cycle number with MDK-ARM*. – Added *Figure 15: STM32F76xxx and STM32F77xxx FFT relative ratio cycle number with MDK-ARM*. – Added STM32F76xxx and STM32F77xxx analysis. – Added *Table 9: Single bank versus dual bank performance comparison*. Updated *Section 3.2: CPU memory access performance with DMA usage*. – Added *Table 13: Configuration 1: execution from the Flash-ITCM / FFT CPU data storage in the DTCM-RAM (single bank/single precision FPU)*. – Added *Table 14: Configuration 2: execution from the Flash-ITCM / FFT CPU data storage in the SRAM1 (single bank/single precision FPU)*. – Added *Table 15: Configuration 3: execution from the Flash-AXI / FFT CPU data storage in the DTCM-RAM (single bank/single precision FPU)*. – Added *Table 16: Configuration 3: execution from the Flash-AXI / FFT CPU data storage in the DTCM-RAM (dual bank/single precision FPU)*. – Added STM32F76xxx and STM32F77xxx analysis. – Updated *Section 4: Software memory partitioning and tips*. – Updated *Section 5: Conclusion*.