Introduction to the system architecture and performance in the STM32H5 MCUs

Introduction

The STM32H5 series devices are microcontrollers based on the high-performance Arm® Cortex®-M33 32-bit RISC core. The high-performance Arm® Cortex®-M33 32-bit RISC core is an implementation of Armv8-M with TrustZone® security technology that provides security to the system.

Reaching 1.5 DMIPS/MHz and 4.092 CoreMark/MHz, the Cortex®-M33 achieves the perfect combination between real-time determinism, energy efficiency, software productivity, and system security. It enables the STM32H5 series devices to be the right fit for many high performance applications.

The STM32H5 series devices bring the best performance and a high level security with:

- A processor that increases in performance in comparison to other Cortex®-M CPUs
- The digital signal processing (DSP) extension
- A floating-point unit (FPU)
- A coprocessor interface to offload compute intensive operations
- The TrustZone® feature to guarantee the maximum of security

This application note presents the STM32H5 global architecture and the memory interfaces and features. The memory interfaces and features provide a high flexibility to achieve the best performance and additional code and data sizes. It also presents the multimaster architecture that contributes to the system performance and offloads the CPU.

This application note provides a software demonstration of the architecture performance of the STM32H5 series devices in various memory partitioning configurations (different code and data locations) as well as the performance of the architecture where DMAs are enabled.

This application note is provided with the X-CUBE-PERF-H5 expansion package that includes two projects:

- STM32H5_performances project aims at demonstrating the performance of the STM32H5 architecture in different configurations. These configurations are the execution of the code and the data storage in different memory locations using both ICACHE and DCACHE.
- STM32H5_performances_DMAs aims at demonstrating the performance of the architecture in multimaster configuration.

1 General information

This document applies to STM32 MCUs based on Arm® Cortex®-M processor.

Note: Arm is a registered trademark of Arm limited (or its subsidiaries) in the US and/or elsewhere.

2 STM32H5 series system architecture

2.1 Cortex®-M33 core

The STM32H5 architecture relies on an Arm® Cortex®-M33 core, which is a highly energy-efficient processor.

The Cortex®-M33 operates at up to 250 MHz and features an optional floating-point unit (FPU), which supports single precision arithmetic.

It also implements an important set of DSP instructions, a memory protection unit (MPU), and security attribution that enhances the application security.

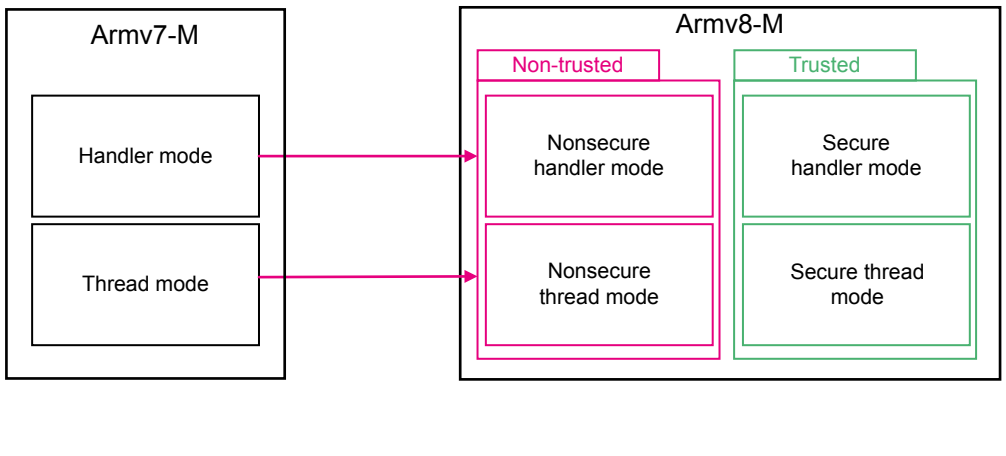
The Cortex®-M33 processor includes a coprocessor interface for frequently used compute-intensive operations. This interface provides a mechanism to increase the general-purpose compute capability.

The Cortex®-M33 processor features a 3-stage pipeline. It includes the TrustZone® technology that creates a secure isolated world to provide confidentiality and integrity to the system, protecting it from software attacks.

2.2 TrustZone® security architecture

The Cortex®-M33 processor features the Arm® TrustZone® technology that is based on the Armv8-M security extension. When enabled, it defines the access permissions based on secure and nonsecure states Figure 1.

Figure 1. Armv8-M security extension



The TZEN option byte in the FLASH _OPTSR2 register activates the TrustZone® security. When the TrustZone® is enabled, the SAU (security-attribution unit) and IDAU (implementation-defined-attribution unit) define the access permissions based on secure and nonsecure states.

When the TrustZone® is enabled, the default security states are the following:

- CPU: The Cortex®-M33 is in secure state after reset. The boot address must be at a secure address.
- Memory map: the SAU is fully secure after reset. As a result, the whole memory map is fully secure. Up to eight SAU configurable regions are available for security attributions.
- Flash memory:
  - Watermark user options define the flash memory security area. All flash memories are fully secure.
  - The flash memory block-based features are nonsecure after reset. Even if the flash memory is configured as nonsecure through IDAU/SAU and through the flash memory secure watermark option bytes, it is possible to configure volatile secure areas using the flash memory block-based feature as secure mode, and this by using the flash memory interface block-based configuration registers.
  - SRAM: the whole SRAM is secure after reset. The block-based memory protection controller (MPCBB) is secure.

The secure memory space is divided into two memory types:

- a secure world, where usually security sensitive applications are run and critical resources are located
- a nonsecure or public world (such as standard nonsecure operating system and user space)

2.3 Instruction and data caches

The devices embed the instruction cache (ICACHE), introduced on the C-AHB bus together with the data cache (DCACHE), introduced on the S-AHB system bus of the Arm® Cortex®-M33 processor.

In most use cases, the ICACHE enables the device to achieve close to zero wait state performance. On another side, the DCACHE enables the device to achieve optimum performance on external memory data accesses.

2.3.1 Instruction cache

The instruction cache (ICACHE) is accessible via two interfaces:

- AHB fast master1 interface with 128-bit data
- AHB slow master2 interface with 32-bit data

The purpose of the instruction cache is to cache instruction fetches or constants data loads coming from the processor. Therefore, ICACHE only manages read transactions and does not manage write transactions.

In order to activate the ICACHE functioning, the EN bit must be set in the ICACHE_CR register.

It caches the code memory region, ranging from address 0x0000 0000 to 0x1FFF FFFF of the memory map.

2.3.2 Data cache

The data cache (DCACHE) is accessible with the AHB master interface with 32-bit data.

The purpose of the data cache is to cache external memories data loads and stores coming from the processor. These accesses include the instruction fetches that may occur to an external memory address. Therefore, DCACHE manages both read and write transactions.

In order to activate the DCACHE functioning, the EN bit must be set in the DCACHE_CR register.

It caches only the external RAM memory region (OCTOSPI and FMC), located from 0x6000 0000 to 0xDFFF FFFF, of the memory map.

Table 1 shows the different cache sizes available on STM32H5.

Table 1. STM32H5 series cache sizes

Device	Instruction cache size	Data cache size
STM32H523xx STM32H533xx STM32H562xx STM32H563xx STM32H573xx	8 Kbytes	4 Kbytes
STM32H503xx	8 Kbytes	N/A

2.4 Cortex®-M33 bus interfaces

The Cortex®–M33 has the following interfaces:

- Code AHB (C-AHB) interface
- System AHB (S-AHB) interface

This section describes each of them.

2.4.1 Code AHB (C-AHB) interface

The code AHB (C-AHB) interface is used for any instruction fetch and data access to the code region of the Armv8-M memory map.

The C-bus of the Cortex-M33 core is connected to the internal flash memory and to the bus matrix via the instruction cache and by means of the fast C-bus. The fast C-bus is used for instruction fetch and data access to the internal memories mapped in the code region. It targets the internal flash memory and the internal SRAMs (SRAM1, SRAM2, and SRAM3). SRAM1, SRAM2, and SRAM3 are accessible on this bus with a continuous mapping.

The C-bus of the Cortex-M33 core is also connected to the bus matrix via the instruction cache and by means of the slow C-bus. The slow C-bus is used for instruction fetch and data access to the external memories mapped in the code region. It also targets the external memories (FMC and OCTOSPI).

2.4.2 System AHB (S-AHB) interface

The system AHB (S-AHB) interface is used for any instruction fetch and data access to the memory-mapped SRAM, peripheral, external RAM and external device.

The S-AHB bus is connected to the bus matrix via the S-bus. The S-bus is used by the core to access data located in a peripheral or an SRAM area. It targets the internal SRAMs (SRAM1, SRAM2, SRAM3, and BKPSRAM), the AHB1 peripherals including the APB1, APB2, AHB2, AHB3, and AHB4 peripherals. SRAM1, SRAM2, and SRAM3 are accessible on this bus with a continuous mapping.

Note: The bus matrix has a zero latency when accessing SRAM1, SRAM2, and SRAM3.

It is also connected to the bus matrix via the data cache by means of the DCACHE S-bus. This bus is used for instruction fetch and data access to the external memories mapped in the data region and targets the external memories (FMC and OCTOSPI).

Note: Fetching instructions through this bus is less efficient than fetching instructions through the slow C-bus.

The processor contains a bus matrix that arbitrates instruction fetches and memory accesses from the processor core between the external memory system and the internal system control space (SCS) and debug components.

Priority is usually given to the processor to ensure that any debug accesses are as nonintrusive as possible.

The system memory map is Armv8-M Main extension compliant, and is common both to the debugger and processor accesses.

The default memory map provides user and privileged access to all regions except for the private peripheral bus (PPB). The PPB space is privileged access only.

Table 2 shows the default memory map. This is the memory map that is used for implementations without the optional MPUs, or when the included MPUs are disabled. The attributes and permissions of all regions, except that targeting the NVIC and debug components, can be modified using an implemented MPU.

Table 2. Memory access behavior

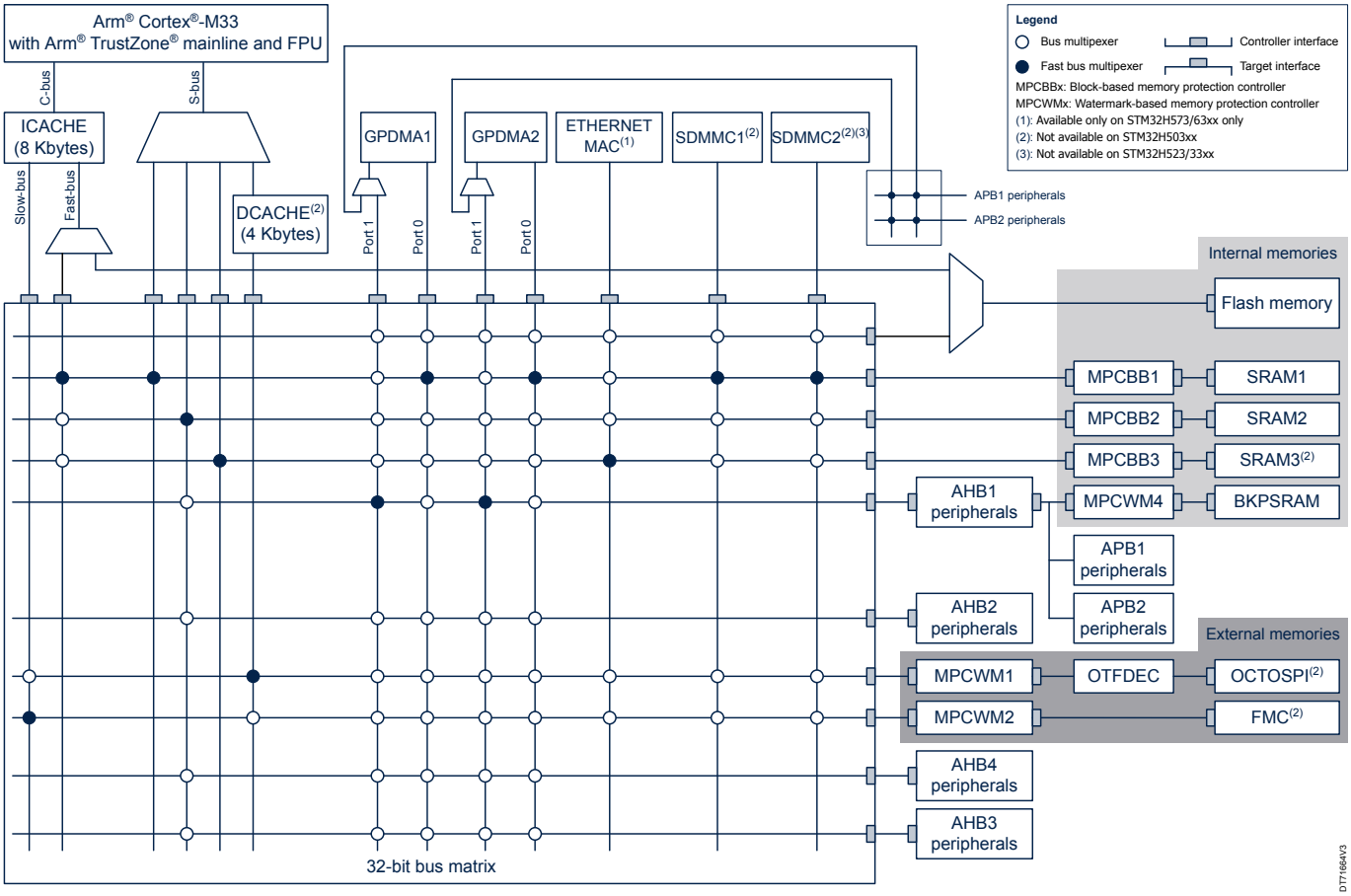
Address range	Memory region	Memory type	Shareability	XN	Description
0x00000000-0x1FFFFFFF	Code	Normal	Nonshareable	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal	Nonshareable	-	Executable region for data. You can also put code here.
0x40000000-0x5FFFFFFF	Peripheral	Device, nGnRE	Shareable	XN	On-chip device memory.
0x60000000-0x9FFFFFFF	RAM	Normal	Nonshareable	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device, nGnRE	Shareable	XN	External device memory.
0xE0000000-0xE003FFFF	Private peripheral bus	Device, nGnRnE	Shareable	XN	This region includes the SCS, NVIC, MPU, SAU, BPU, ITM, and DWT registers.
0xE0040000-0xE0043FFF	Device	Device, nGnRnE	Shareable	XN	This region is for debug components and can include the ETM, CTI, and TPIU configuration registers or none.
0xE0044000-0xE00FFFFF	Private peripheral bus	Device, nGnRnE	Shareable	XN	This region includes the ROM tables.
0xE0100000-0xFFFFFFFF	Vendor_SYS	Device, nGnRE	Shareable	XN	Vendor specific.

2.5 STM32H5 bus matrix

The STM32H5 series devices feature a 32-bit multilayer AHB bus matrix that interconnects the core, masters, and slaves. It enables a number of parallel access paths between the core buses, the masters buses, and the slaves buses. Thus, it provides a concurrent access and efficient operation even when several high-speed peripherals work simultaneously. The STM32H5 bus matrix manages the access arbitration between masters. An internal arbiter resolves the conflicts and the bus concurrency of masters on the bus. It uses a round-robin algorithm. This bus matrix features a fast bus multiplexer used to connect each master to a given slave without latency.

Figure 2 shows the overall system architecture of the STM32H5 series devices as well as the bus matrix connections.

Figure 2. STM32H5 series system architecture



The STM32H5 bus matrix interconnects:

- 13 bus masters or initiators
  - Fast C-bus, connecting Cortex-M33 core C-bus to the internal SRAMs through the instruction cache
  - Slow C-bus, connecting Cortex-M33 core C-bus to the external memories through the instruction cache
  - Cortex-M33 core S-bus (three masters connected to three internal SRAMs without latency along with AHB and APB peripherals)
  - Cortex-M33 core S-bus connected to the external memories through the data cache
  - GPDMA1 (general purpose DMA featuring two master reports)
  - GPDMA2 (general purpose DMA featuring two master ports)
  - SDMMC1
  - SDMMC2
  - Ethernet MAC
- 10 bus slaves
  - The embedded flash memory
  - Internal SRAM1 as a default slave
  - Internal SRAM2
  - Internal SRAM3
  - AHB1 peripherals and backup RAM (4-Kbyte BKPSRAM) including AHB to APB bridges and APB peripherals (connected to APB1 and APB2)
  - AHB2 peripherals
  - FMC (flexible memory controller) memory interface
  - OCTOSPI memory interface
  - AHB3 peripherals, including AHB to APB bridge and APB peripherals (connected to APB3)
  - AHB4 peripherals

2.6 STM32H5 memories

The STM32H5 devices embed a flash memory with different sizes depending on the STM32H5 part number. It also embeds SRAMs with different sizes as well as an external memory interface such as the FMC and the OCTOSPI.

This configuration gives the flexibility to users to partition their application memory resources following their needs. It enables them also to get the right compromise performance versus the application code size.

2.6.1 Embedded flash memory

Each device of the STM32H5 series has its own flash memory size (refer to Section 2.8). The flash memory interface manages the accesses of any master to the embedded nonvolatile memory: It implements the read, program and erase operations, error corrections, as well as various integrity and confidentiality protection mechanisms.

The flash memory interface is accessible via two AHB connections:

- The flash memory AHB register interface
  - It is used to read and write registers (except for some) by 8, 16, and 32 bits. When an unlock sequence for control and option registers is wrong, a bus error is raised, otherwise no read or write errors are generated on the bus.
- The main AHB interface
  - This interface is used to handle three different targets:
    - Code placed in user and system memory, protected by 9 bits of ECC[3]
    - Secure keys placed in OBKEY sectors, protected by 9 bits of ECC and OTP, read only
    - Flash memory high-cycle data area protected by 6 bits of ECC.

By default, the whole AHB memory range is cacheable. For regions where caching is not practical (OTP[4], RO[4], data area), MPU has to be used to disable caching.

One of the main features of the flash memory is the prefetch. The prefetch reads the next sequential instruction from the flash memory. The prefetch is enabled by setting the PRFTEN bit in the flash memory access control register (FLASH_ACR). PRFTEN must be set only if at least one wait state is needed to access the flash memory.

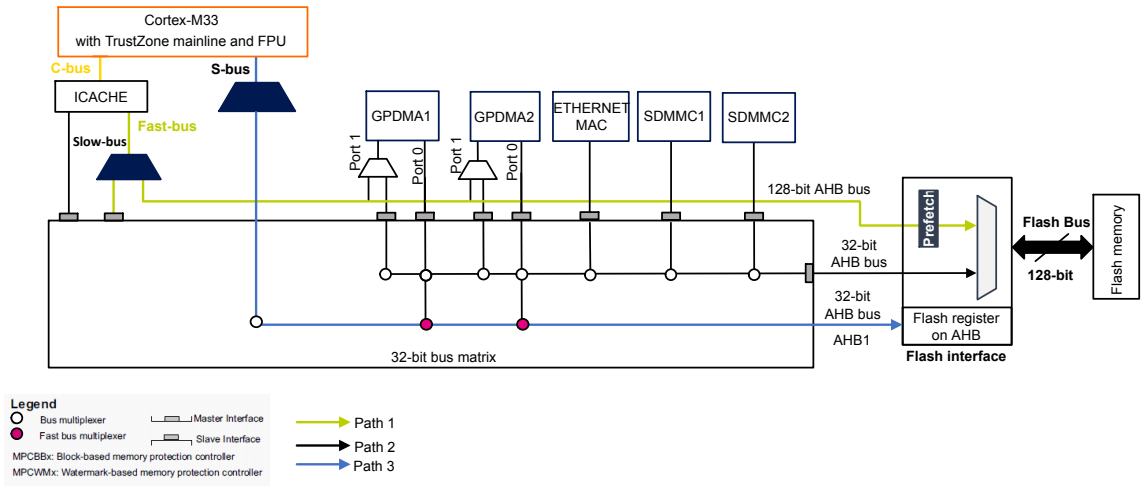
Note: *[3]Error correction codes (ECC): the embedded flash memory supports an error correction code (ECC) mechanism. It is based on the SECDED algorithm in order to correct single errors and detect double errors.*

[4] OTP (one-time programmable) and RO (read only) memories are accessed through the main AHB interface. The OTP is accessible at the address 0x08FF_F000 to 0x08FF F7FF and the read only section is accessible from 0x08FF F800 to 0x08FF_FFFF.

Note: *Features availability depends on the part number. Refer to the related datasheet for more information.*

Figure 3 shows the possible accesses and connection to the flash memory.

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



2.6.2 Embedded SRAM

The STM32H5 series devices feature a large SRAM with a scattered architecture. It is divided into up to four blocks: SRAM1, SRAM2, SRAM3, and BKPSRAM.

These SRAMs are accessible by bytes, half-words (16 bits), or full words (32 bits) and can be addressed both by CPU and DMAs. The CPU can access the SRAM1, SRAM2, and SRAM3 through the S-bus or through the C-bus depending on the selected address. The CPU can access the BKPSRAM through the S-bus only. When the TrustZone® security is enabled, all SRAMs are secure after reset.

Note: *All the SRAMs are consecutive in terms of memory location.*

SRAM blocks availability depends on the part number. Refer to the related datasheet for more information.

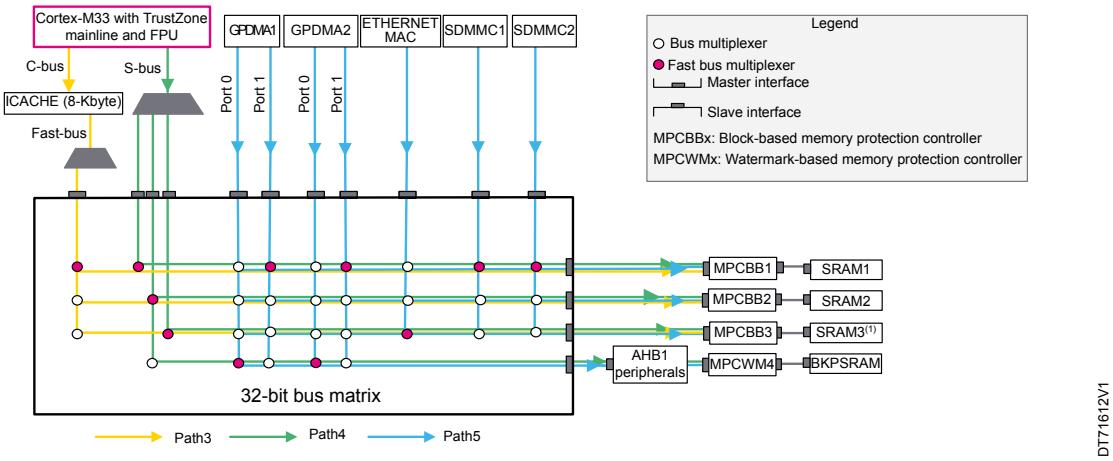
The possible accesses and connection to the different SRAMs are shown in Figure 4.

Path 3 shows the connection between the C-bus of the Cortex-M33 core and the internal SRAMs through the bus matrix.

Path 4 shows the connection of the S-bus of the Cortex-M33 core to the internal SRAMs.

Path 5 shows the connection of the GPDMAs, the Ethernec MAC, and SDMMCs buses to the internal SRAMs via the bus matrix.

Figure 4. SRAMs possible connections (paths: 3, 4 and 5)



1. *SRAM3 is not available in STM32H503xx devices.*

2.6.3 External memories

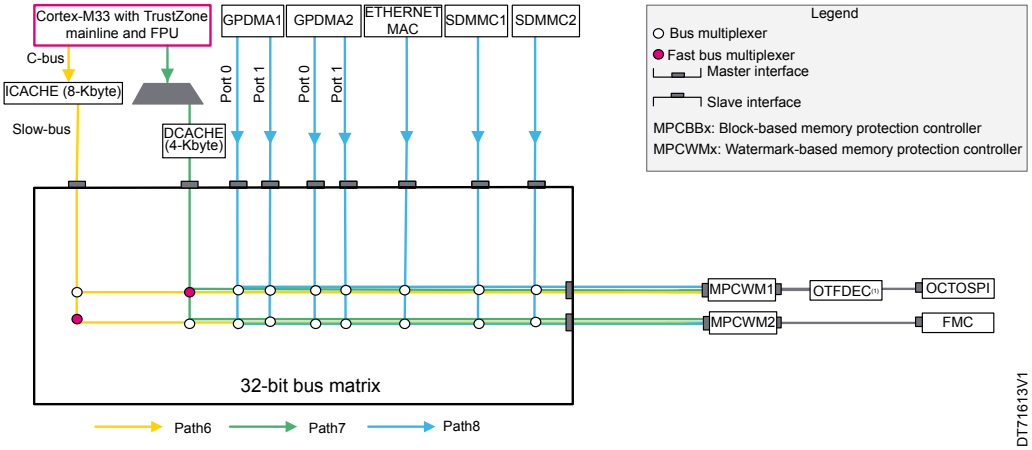
The STM32H5 has several internal memories and storage controllers, such as the USB and the SDMMC. The user can extend these with:

- the flexible memory controller (FMC)
- the OCTOSPI controller

Note that external memories are not available in STM32H503xx devices.

- Figure 5 shows the possible paths that interconnect the CPU and the different DMAs.
- Path 6 shows the connection of the Cortex®-M33 core C-bus to the external memories. The connection is made through the instruction cache and through the bus matrix.
- Path 7 shows the connection of the Cortex®-M33 core S-bus to the external memories. The connection is made through the data cache through the bus matrix.
- Path 8 shows the connection of the GPDMAs, the Ethernet MAC, and SDMMCs buses to the external memories. This connection is made through the bus matrix.

Figure 5. External memory connections

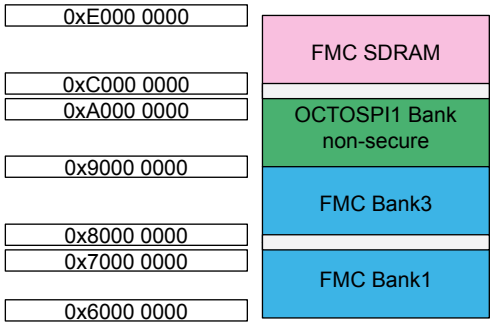


1. *The OTFDEC TrustZone-aware peripheral proposes on-the-fly decryption of encrypted images. These are stored on external flash memory that is connected through the OCTOSPI.*

All the external memories are accessible by all the masters (CPU and DMAs). Thus, the memory to memory DMAx transfer or peripheral/memory DMAx transfer is allowed.

Figure 6 summarizes the memory mapping of the external memories and their address ranges.

Figure 6. External memory based on IDAU mapping



DT71614V1

2.6.3.1 *Flexible memory controller (FMC)*

The STM32H5 FMC controller interfaces the memory mapped devices including SRAMs, ROMs, NOR/NAND flash memories and SDRAM devices. Except for NAND flash memory, it is used either for the program execution, or the data load/store operations.

STM32H5 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 for each memory bank
- 8-bit, 16-bit data bus width
- 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 by means of a unique chip selected.

The FMC performs only one access at a time to an external device.

All the external memories that are connected to the FMC benefit from the data cache (DCACHE) (path 7 in Figure 5). It allows then to have more data or/and code sizes with the maximum of performance.

2.6.3.2 *Octo-SPI interface*

The STM32H5 series devices embed an octo-SPI memory interface, which is a specialized communication interface targeting single, dual, quad, or OCTOSPI flash memories. It supports most external serial memories such as serial PSRAMs, serial NAND, and serial NOR flash memories. It supports also hyper RAM and hyper flash memories starting from 0x9000 0000 to 0x9FFF FFFF in the memory mapped mode. Besides, it supports a dual-quad configuration, where eight bits can be sent/received simultaneously by accessing two quad memories in parallel.

In addition, the interface supports single data rate (SDR), which means that the transfer of data is performed only once per clock pulse, and double-transfer rate (DTR), meaning that the data is transferred on both the rising and falling edges of the clock signal. It can work in one of the three following modes:

- Indirect mode: all the operations are performed using the OCTOSPI registers
- Automatic status-polling mode: the external memory status register is periodically read and an interrupt can be generated in case of flag setting.
- Memory-mapped mode: the external memory is memory mapped and it is seen by the system as if it was an internal memory, supporting both read and write operations

Compared to the FMC, the OCTOSPI enables the connection to an external flash memory with a reduced cost, small packages (reducing the PCB area), and a reduced GPIOs usage.

As shown in Figure 5 (path 7), the OCTOSPI is mapped on a dedicated layer on the AHB. It can benefit from the DCACHE. This allows executing the code and loading the data from the OCTOSPI with good performances. This interface is also accessible by all the masters.

The OTFDEC (on-the-fly decryption) adds a layer of security when interfacing external flash memory with OCTOSPI. It is a security feature that uses AES-128 in counter mode to achieve the lowest possible latency, therefore enhancing security of the system without heavily impacting its performance.

When OTFDEC is used in conjunction with OCTOSPI, it is mandatory to access the flash memory using the memory-mapped mode of the flash memory controller.

The OTFDEC allows for up to four independent encrypted regions, with a granularity of the region definition of 4096 bytes. Each region has its own 128-bit key, two bytes firmware version, and eight bytes application-defined nonce, which must be changed each time an encryption is performed by the application.

OTFDEC also supports OCTOSPI pre-fetching mechanism and an enhanced encryption mode to add a proprietary layer of protection on top of AES stream cipher (execute only).

2.7 DMAs

The STM32H5 series devices embed two identical general-purpose direct memory access (GPDMA) controllers that have two ports each. They are used to perform programmable data transfers between memory-mapped peripherals and/or memories. These DMAs are provided to offload the CPU and have eight channels implemented differently. They enable users to allocate and select the best for their application, and this makes the best use of the GPDMA performances.

There are two instances of the GPDMA in the devices, named as GPDMA1 and GPDMA2. Each GPDMA instance has access to the following internal memories: embedded flash memory, SRAM1, SRAM2, SRAM3, and the BKPSRAM.

They also have access to external memories through FMC and OCTOSPI controllers via the bus matrix.

Thanks to 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.

2.8 Main differences between the STM32H5 series devices

Table 3. Main differences from architecture point of view

Feature	STM32H503xx	STM32H523/533xx	STM32H562/STM32H563/ STM32H573xx
Instruction cache size	8 Kbytes	8 Kbytes	8 Kbytes
Data cache size	Not available	4 Kbytes	4 Kbytes
FPU	VFPv5 single precision floating-point	VFPv5 single precision floating-point	VFPv5 single precision floating-point
Flash memory size	128 Kbytes	512 Kbytes	2 Mbytes
Flash memory width	up to 128-bit access	up to 128-bit access	up to 128-bit access
SRAM1	16 Kbytes	128 Kbytes	256 Kbytes
SRAM2	16 Kbytes	80 Kbytes	64 Kbytes
SRAM3	Not available	64 Kbytes	320 Kbytes
BKPSRAM	2 Kbytes	2 Kbytes	4 Kbytes
Ethernet(1)	Not available	Not available	Available
SDMMCx(2)	Not available	Available	Available
FMC(3)	Not available	Available	Available
OCTOSPI	Not available	Available	Available

1. *Ethernet MAC is available only in STM32H573xx*
2. *SDMMC2 is not available on STM32H523/33xx devices.*
3. *SDRAM is not available on STM32H523/33xx devices.*

3 Typical application

This application note provides two software examples that show, first, the STM32H5 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 STM32H5_performances project can be used as a skeleton where the user can integrate his application. The same example is used for the STM32H5_performances_DMAs project except the magnitude calculation that has been removed to allocate more RAM for DMAs transfers.

3.1 FFT demonstration

The FFT example is used as it benefits from the floating-point unit of the Cortex-M33. It also contains several loops and data load/store operations, 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.

It is calculated in the frequency domain of the input signal with the use of:

- complex FFT
- complex magnitude
- 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.

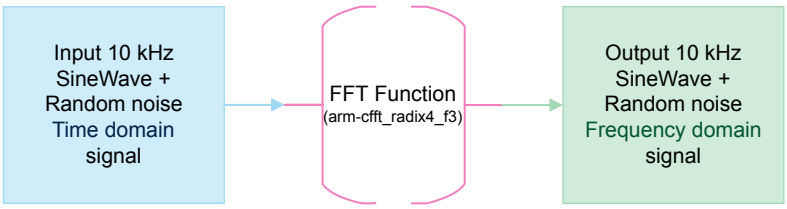
Figure 7 shows the block diagram of the transformation.

The number of cycles consumed by the FFT process is also calculated based on the SysTick. The results are shown on the HyperTerminal.

The FFT demonstration shows:

- the current project configuration
- the system frequency
- the different configuration of caches
- the flash memory prefetch (ON/OFF)
- the memory configuration in the case of an external memory (OCTOSPI)

Figure 7. FFT example block diagram



3.2 Project configuration of the CPU memory access demonstration

The CPU memory access demonstration is provided with the IAR Embedded Workbench®. The six configurations of this project enable 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 flash memory loader loads initially the application. That application is also copied later on (by subroutine generated automatically by some toolchains), 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-Flash_rwSRAM1: the program is executed from the flash memory at five wait states with the **ICACHE enabled** and the data are loaded/stored in the SRAM1.
- 2-Flash_rwSRAM2: the program is executed from the flash memory at five wait states with the **ICACHE enabled** and the data are loaded/stored in the SRAM2.
- 3-Flash_rwSRAM3: the program is executed from the flash memory at five wait states with the **ICACHE disabled** and the data are loaded/stored in the SRAM3.
- 4-Flash_rwSRAM3: the program is executed from the flash memory at five wait states with the **ICACHE enabled** and the data are loaded/stored in the SRAM3.
- 5-SRAM1_rwSRAM3: the program is executed from the SRAM1 with the **ICACHE enabled** and the data are loaded/stored in the SRAM3.
- 6- OCTOSPI_rwSRAM1: the code is executed from the OCTOSPI flash memory with **ICACHE and DCACHE enabled**.

By merging the adequate settings, the user can create new code execution/data storage location configurations based on these templates, modifying the linkers and setting the adequate flash memory loader.

The user can modify the linkers through the IAR (EWARM) toolchain. The linkers are located under EWARM\icf_files. The modifications enable the change of the RAM regions being the stack and heap regions.

3.3 Project configuration of the CPU memory with DMA activation demonstration

The DMA activation demonstration is provided with the IAR Embedded Workbench. The project configurations are based on the same template described previously in Section 3.2. The project has one configuration, Flash_rw-SRAM3.

The I-Cache is enabled throughout the DMA activation demonstration. The execution/data memories used by the FFT process and the other modules are modified and separated in the icf files.

The same FFT demonstration already used for the CPU memory access performance (STM32H5_performances) is used for the DMA activation demonstration performance (STM32H5_performances_DMAs).

In this section, only the FFT computation is kept and the magnitude calculation part has been removed in order to allocate more RAM for DMA transfers.

The DMA2 is used in the demonstration to perform memory to memory transfers.

The memory source and destination are configurable separately for each configuration. That is done in a main.h file, which contains all the needed definition to perform the transfer.

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

- Step 1: configure the different DMAs' parameters in main.h
- Step 2: check mode: check the configuration and the DMAs' transfers
- Step 3: result mode: get the results in term of cycles number

3.3.1 Step 1: configuring 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

Enabling/disabling the DMA transfer

To enable or disable a DMA, the define “#define USE_DMA2” has to be set or cleared to respectively enable or disable DMA2.

DMA source address configuration

The following defines configure the memory source and destination addresses:

- #define DMA2_SRC_ADDRESS: The start address of the memory source
- #define DMA2_DST_ADDRESS: The start address of the memory destination

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

The DMA2_SRC_ADDRESS and DMA_DST_ADDRESS possible values are the following:

- SRAM1_DMA_START_ADDRESS
- SRAM2_DMA_START_ADDRESS
- SRAM3_DMA_START_ADDRESS

Do not modify these data provided in dma_utilities.h file.

DMA destination address configuration

The DMA2 transfer size must be configured by the following define:

- #define DMA2_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. The objective is to prevent any DMA access to an invalid memory zone.

DMA2 transfer size

The DMA2_TRANSFER_SIZE possible values are:

- SRAM1_DMA_AVAILABLE_SIZE
- SRAM2_DMA_AVAILABLE_SIZE
- SRAM3_DMA_AVAILABLE_SIZE

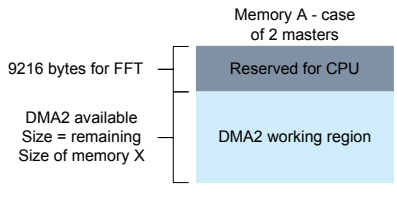
Example of DMA2 transfer size configuration where the SRAM1 is the source and the SRAM2 is the destination:

- #define DMA2_TRANSFER_SIZE min (SRAM1_DMA_AVAILABLE_SIZE, SRAM2_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(for example DMA2) with concurrency with the CPU on a given memory (memory A), it must split the memory as shown in Figure 8. The figure shows how to partition memory A (for example the SRAM1) in case of concurrency between two masters in the software. The partitioning shows that memory A is shared between the CPU and the DMA2. This partitioning allows the DMA to complete its transfer after 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 linker file (for CPU access).

Figure 8. How to split memory A in case of concurrency between all the masters



3.3.2 Step 2: checking the configuration and the DMA transfer(s)

3.3.3 Step 3: getting the results

At this stage, getting the DMA transfers are done successfully and the user can compile the example, load it and run it.

The HyperTerminal displays the current configuration as well as the number of cycles at the bottom of the display.

4 Results and analysis

4.1 CPU memory access performance

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

For this purpose, the IAR Embedded Workbench® IDE-ARM 9.30.1 is used.

For all the configurations for which the code is executed from the flash memory or one of the SRAMs, the ICACHE is enabled. This is the case of the following configurations:

1. Flash_rw-SRAM1
2. Flash_rwSRAM2
3. Flash_rwSRAM3
4. SRAM1_rwSRAM3

For the configuration that has the code execution from the external memory OCTOSPI, the data cache is enabled.

Results

Table 4 shows the obtained results for the FFT demonstration, IAR Embedded Workbench®, and Arm®, in each configuration of the STM32H5 series devices.

Table 4. IAR results for the STM32H5xx series devices

Feature configuration	Memory location configuration	Number of CPU cycles
ICACHE ON + prefetch ON	1-Flash_rwSRAM1	155146
ICACHE ON + prefetch OFF	2-Flash_rwSRAM2	157296
ICACHE OFF + prefetch OFF	3-Flash_rwSRAM3	301007
ICACHE ON + prefetch OFF	4-Flash_rwSRAM3	157296
ICACHE ON	5-SRAM1_rwSRAM3	171755
DCACHE ON	6-OCTOSPI_rwSRAM1	203478

Note: The cycle number values may change from a toolchain version to another.

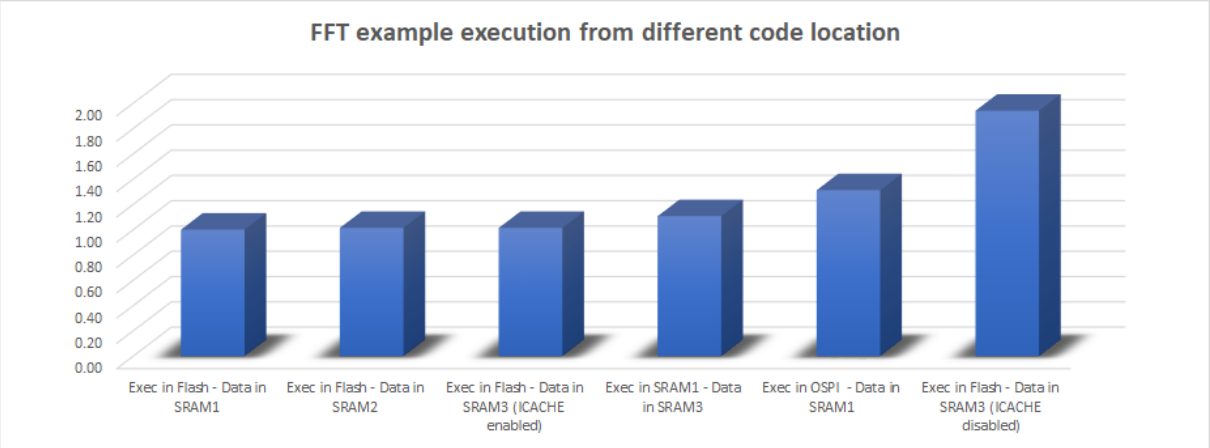
Note: Different features are enabled for the first three configurations otherwise the obtained results may be almost the same.

The charts in Section 4.1 show the relative ratio of each configuration versus the configuration 1 that has the best results for the STM32H5 series devices.

$$Relative\ ratio = Number\ of\ cycles\ of\ configuration\ X \div Number\ \ of\ cycles\ for\ \ the\ best\ configuration$$

The relative ratio calculation enables the comparison of two configurations, one of them being the configuration with the least number of cycles (1-Flash-rwSRAM1).

Figure 9. **STM32H5xxx relative ratio cycle number with IAR**



Analysis

If the user fixes the execution memory location into the flash memory and changes the read/write data memory location, the relative ratios show that, no matter with the internal RAM used for the data storage, the results are the same.

It includes the following configurations:

- 2-Flash_rwSRAM2
- 4-Flash_rwSRAM3

If the user fixes the code execution memory location in the flash memory and fixes the data memory location in SRAM3, but changes the ICACHE state, the relative ratios demonstrate that the user must enable the instruction cache. The goal is to increase the product performance and it occurs when using the flash memory as code execution location. It includes the following configurations:

- 3-Flash_rwSRAM3
- 4-Flash_rwSRAM3

If the user fixes the read/write data memory location and changes the code execution memory location, the relative ratios show that the use of an internal memory as the code execution memory location enhances the device performance.

It includes the following configurations:

- 1-Flash_rwSRAM1
- 6-OCTOSPI_rwSRAM1

If the user fixes the read/write data memory location and varies the code execution memory location, the relative ratios show that using the flash memory as the code execution memory location increases the performance of the device. It includes the following configurations:

- 4-Flash_rwSRAM3
- 5-SRAM1_rwSRAM3

Note:

1. *In order to obtain the maximum of device performance, it is recommended to enable both ICACHE and flash memory prefetch during the use of the flash memory as code execution memory location.*
2. *When the OCTOSPI is remapped to 0x0000000 and ICACHE is enabled, the performance increases.*
3. *These tests were executed with OTFDEC OFF. Enabling this feature may impact the performance.*
4. *Enabling TrustZone® does not impact the performance of the device.*

4.2 CPU memory access performance with DMA usage

This section treats the performance where one or more masters are activated. The results are obtained with the IAR Embedded Workbench for Arm 9.30.1.

Results

Table 5 summarizes the number of cycles consumed by the execution of the FFT process in different scenarios. The results are obtained with 5-wait states access to the internal flash memory.

Table 5. Configuration 1: execution from the Flash/FFT CPU data storage in the SRAM3

Scenario	Master	Source	Destination	FFT cycle number
1	DMA2	-	-	133624
2	DMA2	SRAM1	SRAM3	135240
3	DMA2	SRAM3	SRAM1	135237

Analysis

If the CPU and one or more masters do not access the same memory, no contention is present.

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 (SRAM1, SRAM2, etc.), or on the CPU side where the concurrency is performed at the SRAM3 level. As shown by the results, the concurrency of the CPU and the DMA2 when writing to either the SRAM3, or the SRAM1 is about 1.2% of the performance decrease.

5 Conclusion

The applications become more complex and require efficient microcontrollers that provide performance. Through its smart and secure architecture, the STM32H5 device is a suitable platform for the internet of things (IoT) applications. It enables the developers in need of a fast microcontroller to ease the code optimization. A better application responsiveness is obtained with the remarkable mechanisms of the STM32H5. They enable reaching almost 0-wait states in code execution and a higher performance ranking. Besides, with the DCACHE, the user is able to spend less time optimizing the code and the data size by adding external memory resources with no performance penalty. Even if the DMA transfers are done at the same time as the CPU, the STM32H5 system architecture offers a steady performance.

Revision history

Table 6. **Document revision history**

Date	Version	Changes
29-Jun-2023	1	Initial release.
04-Apr-2024	2	Updated: • Section 2.6.3.2: Octo-SPI interface • Figure 2. STM32H5 series system architecture • Table 1. STM32H5 series cache sizes • Section 2.8: Main differences between the STM32H5 series devices

Contents

List of tables

List of figures