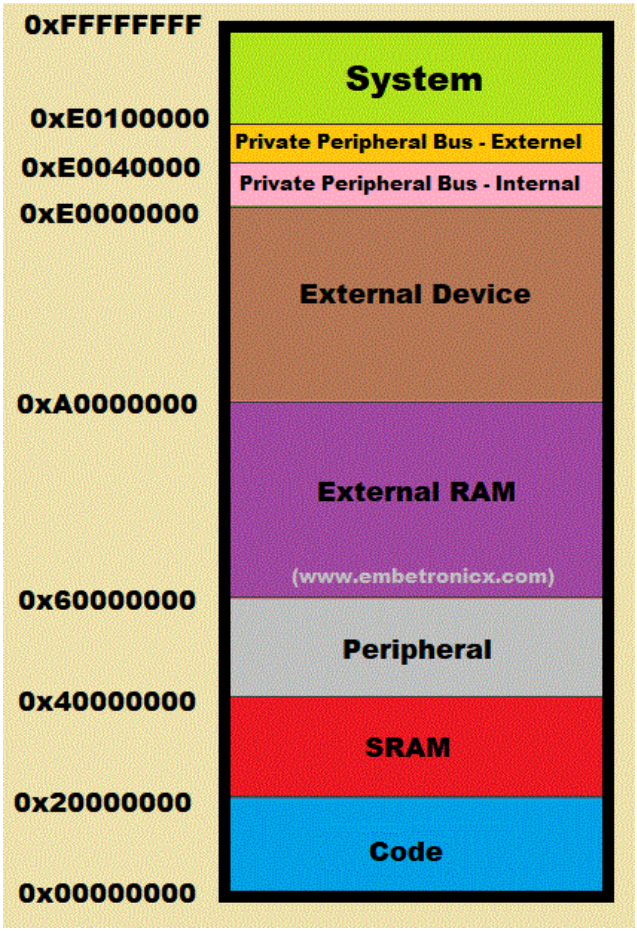
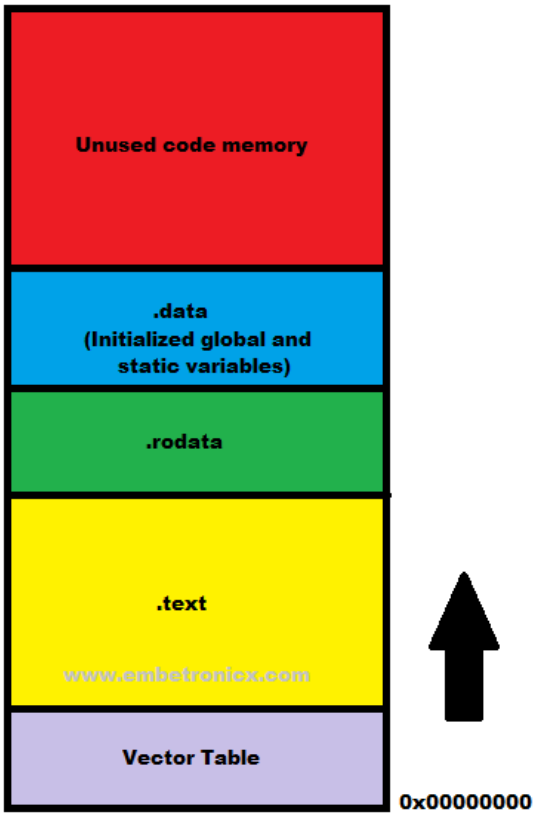
Reset Sequence:

- https://embetronicx.com/tutorials/microcontrollers/stm32/reset-sequence-in-arm-cortex-m4/ ← details for a cortex m4 but m7 is similarly structured



- Code region (0x00000000 - 0x1FFFFFFF): our application code / binary
- SRAM region (0x2000000 - 0x3FFFFFFF): stack, heap, global RW variables, static variables for the program

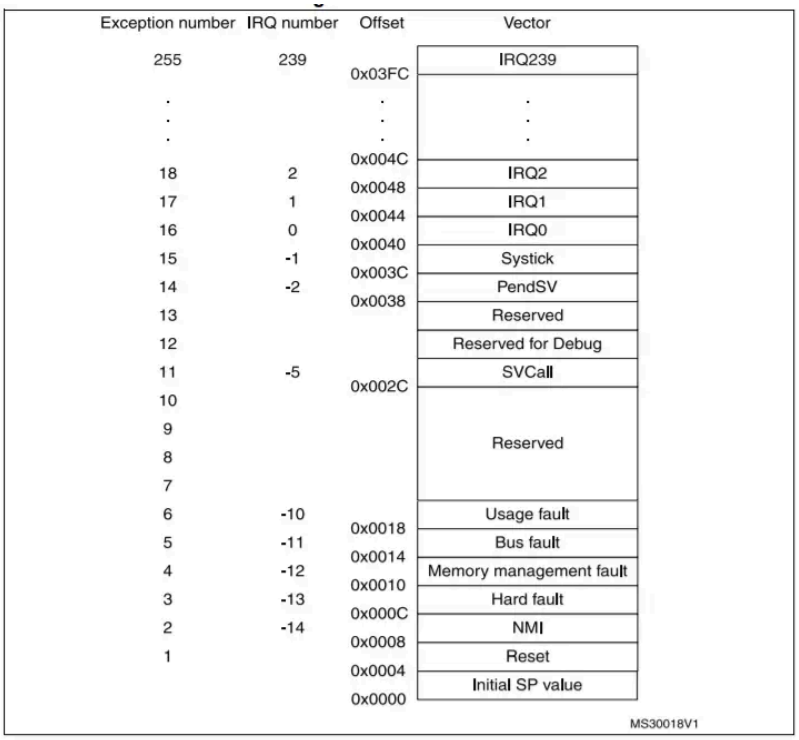
Code Region:



- See for more details:
    - https://embetronicx.com/tutorials/p_language/c/compilation-steps-and-memory-layout-of-the-c-program/#Memory_Layout_of_the_C_Program
- .text / .code segment : contains the machine code of the compiled program into **read-only** segment
- .data : stores program data, stores initialized global / static variables and uninitialized data in a region called BSS (within itself)

Vector Table:

- contains locations of all exception and interrupt service routines



- Address 0x0000 0000 contains the initial stack pointer address
- Address 0x0000 0004 contains the reset handler address

Sequence (Non-Bootloader, for your understanding) :

1. The PC (program counter) or instruction pointer will be loaded with the address 0x0000 0000 → read from that address
2. The MSP (main stack pointer) register will then use the initial SP address from 0x0000 0000
3. PC advances, to load the reset handlers address next

Note: These above 3 steps are done by the hardware (This is architecture specific).

4. Reset handler performs the following:
    a. Initialize system
    b. Copy initialized global variables, static variables from .data region to SRAM
    c. Copy the un-initialized data (.BSS region) to SRAM and initialize to 0
    d. Calls main()

Note: This process is applicable when you don't have a bootloader. If you have a bootloader, then it will execute in a different way. See below:

Bootloader Sequence:

- Since a bootloader is also firmware that is loaded, there will be **two binaries**
    - The bootloader image and the application binary
- Bootloader is loaded onto 0x0000 0000 with a **vector table**
- The application will be loaded onto another area of flash memory with its **own vector table**

1. Press reset button: start from bootloader since the PC or instruction pointer always goes to 0x0000 0000
2. Copy the stack pointer address to MSP from the bootloader vector table
3. PC moves to reset handler:
    a. Bootloader does operations like check firmware version, upgrade firmware or in our case… load program to flash memory via CAN, I guess?
4. Once done, jumps to the applications vector table via PC
5. Initialize MSP again with the initial stack pointer value of the applications vector table
6. Tell controller to use applications vector table rather than the bootloaders vector table by setting the Vector Table Offset Register (VTOR)
7. Advance PC to the applications reset handler (Same as before):
    a. Initialize system
    b. copy .data region and the .bss subregion to SRAM
    c. Move PC to the main function of the application

NOTE: ALL THE INFO ABOVE WAS FOR CORTEX M4 PROCESSORS, CORE PRINCIPLES SHOULD WORK THE SAME FOR M7

Bootloader and Memory Basics

https://embetronicx.com/tutorials/microcontrollers/stm32/bootloader/bootloader-in-stm32-bootloader-design/
https://www.st.com/resource/en/reference_manual/rm0410-stm32f76xxx-and-stm32f77xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

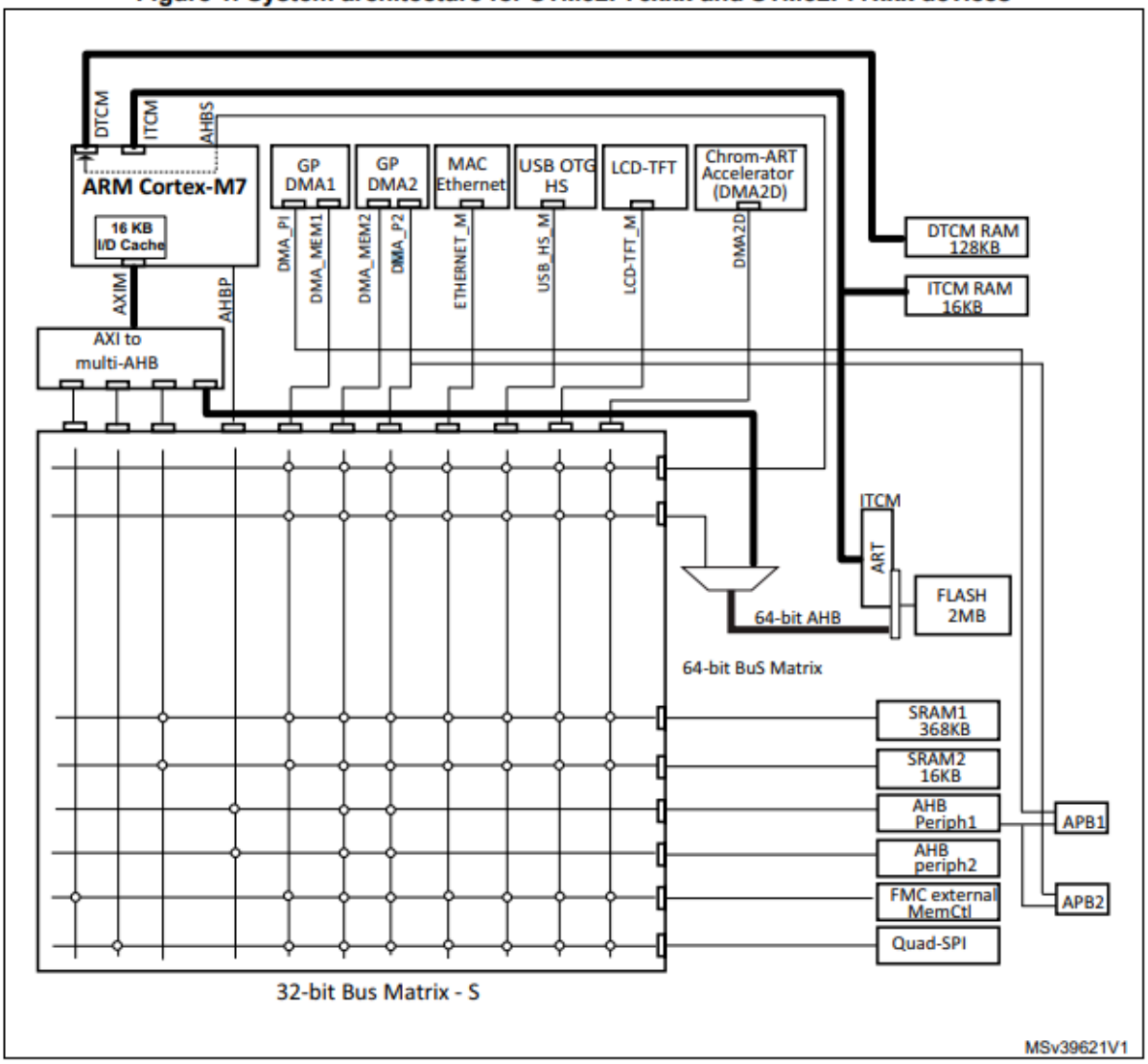
- Firstly, what is AHB and APB?
    - https://www.tutorialspoint.com/difference-between-ahb-and-apb
    - AHB and APB are both high-performance bus protocols for connecting memory and peripherals
- What is DMA? **RESEARCH MORE**
    - Direct Memory Access interface → allows for writing
    - GP-DMA → general purpose direct memory access

Memory Basics:

- Up to 2MB of flash memory, organized into two banks for RW access
    - nDBANK register: 1 = read access of 256 bits, 0 = read access of 128 bits
    - When nDBANK = 0 ; we can read and write at the same time (reason for 2 banks)
- SRAM 512KB
    - 128 KB of data TCM (tightly-coupled memory) RAM (in **DTCM RAM** accessed via **DTCM BUS**, also accessed through the **AHBS Bus via AXIM interface** for peripherals and general purpose DMA's)
        - Dedicated memory region that is very close to the CPU → fast, no cache → memory access is predictable
        - Used for critical data: interrupt handlers, and other important things
        - Can also be used for instruction fetching
    - 16 KB of instruction TCM RAM (in **ITCM RAM** accessed via **ITCM Bus**, not available to peripherals or general purpose DMA's)
        - For critical routines and instruction fetching
        - Reserved only for the CPU, peripherals cannot access it
    - 368 KB Regular SRAM (located in **SRAM1 bank**, via **AXI to AHB BUS** with **DMA**)
- Auxiliary Internal RAM 16 KB via SRAM2

System Architecture and Busses:

Figure 1. System architecture for STM32F76xxx and STM32F77xxx devices



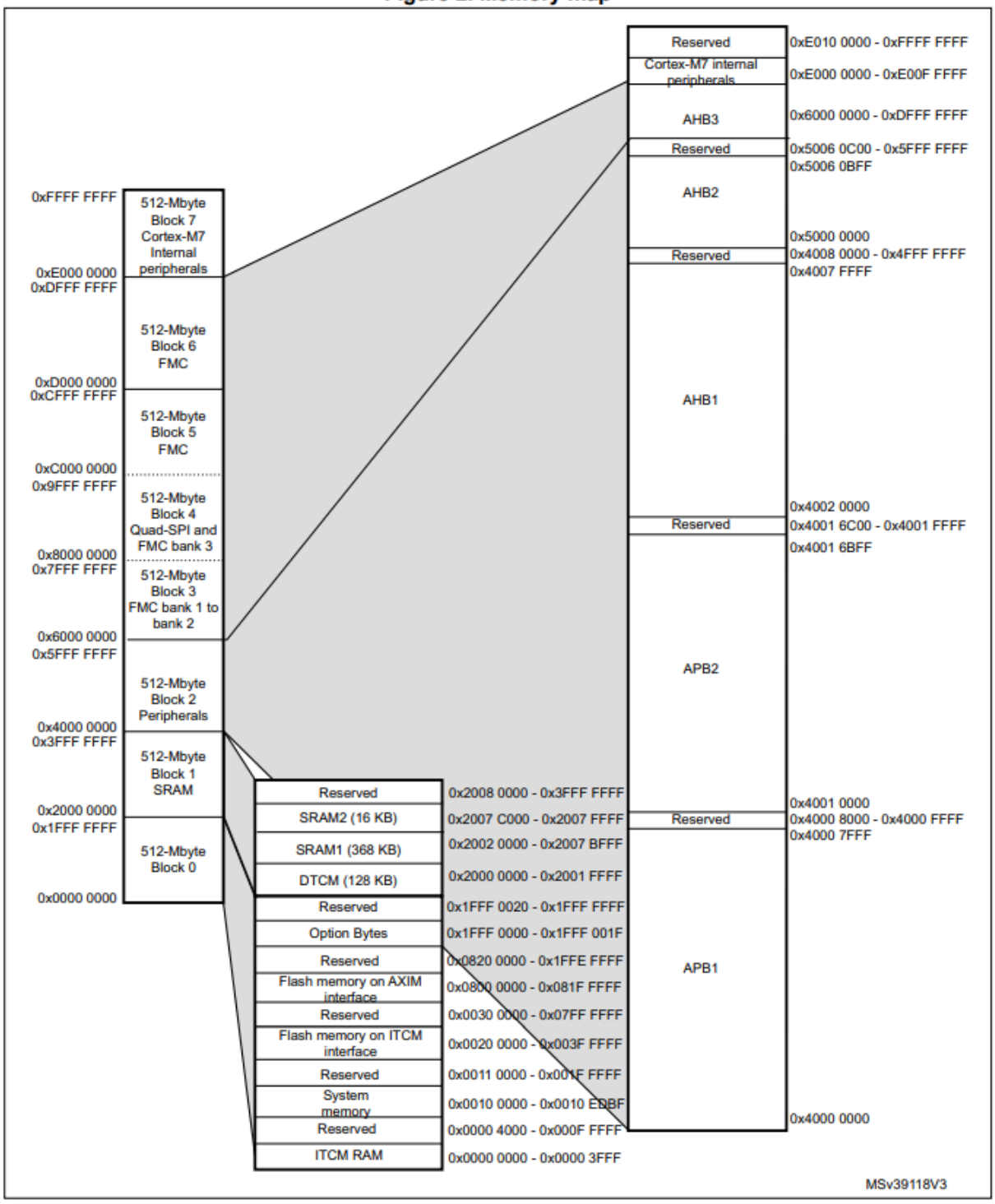
- The **AXIM bus**, it is actually is an AXI to multi AHB bridge with 4 AXI bus targets:
    - 1x AXI bus to 64-bit AHB connected to the **embedded flash** (this is what we use to **write** to our flash memory) ; we have **READ AND WRITE** access here
    - 3x AXI to 32-bit AHB connected to the AHB bus matrix; connected to SRAM, Quad-SPI access and external memory FMC (flexible memory controller, for ethernet, LCD etc.), respectively.
- The **ITCM bus** is very fast (TCM as mentioned above)
    - ITCM has access to program memory (2MB flash) but is **READ-ONLY**
        - This bus has an ART interface in-between, just a technical detail but it allows faster access via caching
    - ITCM has its own ITCM RAM, but is reserved only for the CPU

- The **DTCM bus** is also TCM based, and is fast it has its own **DTCM RAM** as mentioned above in memory
- **CPU AHBS Bus:**
    - **Allows** for **DMAs** data transfer to the **DTCM RAM** only (peripheral access)
    - DMA data transfer to/from **ITCM RAM** is **not supported**

Memory Organization:

- **ALL** program memory, data memory, registers, memory mapped I/O ports are organized within a 4GB **address space** (does **NOT** correspond **1:1** to *physical* memory)
    - Bytes are coded in little endian format
    - Divided into 8 main blocks, each being 512MB
    - 4GB is much more than the onboard memory for the STM32, so most of this is all **virtual memory**
- The diagram below should now make some more sense:
    - Block 0:
        - Contains the addresses for accessing the flash memory (both correspond to the *same* **physical memory**):
            - Flash memory on AXIM interface is read and write as mentioned before
            - Flash memory on ITCM interface is read only but fast, as mentioned before
        - Also holds our **option bytes**,
        - Holds **system memory or ROM** (read-only memory), which also holds the **system bootloader**
        - Holds our **ITCM RAM**, which starts at 0x0000 0000
    - Block 1:
        - Holds addresses for SRAM1, SRAM2 and DTCM RAM, all accessible through the AXIM interface via the AHB and AHBS bus
    - Block 2-6:
        - Pages 77 - 80 describes the register memory boundaries for all peripherals of the STM32 via AHB and APB busses

Figure 2. Memory map



Flash Memory Organization:

- Overview from Manual:

2.4 Flash memory overview

The Flash memory interface manages CPU AXI and TCM accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms. It accelerates code execution with ART on TCM interface or L1-Cache on AXIM interface.

The Flash memory is organized as follows:

- A main memory block divided into sectors.
- A second block:
  - System memory from which the device boots in System memory boot mode
  - 1024 OTP (one-time programmable) bytes for user data.
  - Option bytes to configure read and write protection, BOR level, software/hardware watchdog, boot memory base address and reset when the device is in Standby or Stop mode.

Refer to *Section 3: Embedded Flash memory (FLASH)* for more details.

The Flash memory has the following main features:

- Capacity up to 2 Mbytes, in single bank mode (read width 256 bits) or in dual bank mode (read width 128 bits)
- Supports dual boot mode thanks to nDBOOT option bit (only in dual bank mode nDBANK=0)
- 256 bits wide data read in single bank mode or 128 bits wide data read in dual bank configuration
- Byte, half-word, word and double word write
- Sector, mass erase and bank mass erase (only in dual bank mode)

- For our case, assuming that we are **NOT** going to be using dual bank mode, and that we are using an STM32f767zi (2MB of flash), this is the following table listing the flash memory sectors (if needed see page 88 and onwards on the technical manual):

Table 3. 2 Mbytes Flash memory single bank organization (256 bits read width)

Block	Name	Bloc base address on AXIM interface	Block base address on ICTM interface	Sector size
Main memory block	Sector 0	0x0800 0000 - 0x0800 7FFF	0x0020 0000 - 0x0020 7FFF	32 KB
	Sector 1	0x0800 8000 - 0x0800 FFFF	0x0020 8000 - 0x0020 FFFF	32 KB
	Sector 2	0x0801 0000 - 0x0801 7FFF	0x0021 0000 - 0x0021 7FFF	32 KB
	Sector 3	0x0801 8000 - 0x0801 FFFF	0x0021 8000 - 0x0021 FFFF	32 KB
	Sector 4	0x0802 0000 - 0x0803 FFFF	0x0022 0000 - 0x0023 FFFF	128 KB
	Sector 5	0x0804 0000 - 0x0807 FFFF	0x0024 0000 - 0x0027 FFFF	256 KB
	Sector 6	0x0808 0000 - 0x080B FFFF	0x0028 0000 - 0x002B FFFF	256 KB
	Sector 7	0x080C 0000 - 0x080F FFFF	0x002C 0000 - 0x002F FFFF	256 KB
	Sector 8	0x0810 0000 - 0x0813 FFFF	0x0030 0000 - 0x0033 FFFF	256 KB
	Sector 9	0x0814 0000 - 0x0817 FFFF	0x00340000 - 0x0037 FFFF	256 KB
	Sector 10	0x0818 0000 - 0x081B FFFF	0x0038 0000 - 0x003B FFFF	256 KB
	Sector 11	0x081C 0000 - 0x081F FFFF	0x003C 0000 - 0x003F FFFF	256 KB
Information block	System memory	0x1FF0 0000 - 0x1FF0 EDBF	0x0010 0000 - 0x0010 EDBF	60 Kbytes
	OTP	0x1FF0 F000 - 0x1FF0 F41F	0x0010 F000 - 0x0010 F41F	1024 bytes
	Option bytes	0x1FFF 0000 - 0x1FFF 001F	-	32 bytes

Flash Write / Erase Operations:

- All operations are controlled via the FLASH_CR register, this allows us to set the size of the operation (byte, sector or whole bank), the operation (read or write), start the operation and see the current status of the flash operation.
- After every reset writing is not allowed in the FLASH_CR register for protection, in order to unlock it you must follow a sequence of setting bits on page 93
- For more information on the specific sequences used for flash programming see pages 93 -97 in the technical manual

Flash Option Bytes:

- As discussed option bytes are stored in flash memory:

Table 10. Option byte organization

AXI address	[63:16]	[15:0]
0x1FFF 0000	Reserved	ROP & user option bytes (**RDP** & **USER**)
0x1FFF 0008	Reserved	IWDG_STOP, IWDG_STBY and nDBANK, nDBOOT and Write protection nWRP/NWRPDB (sector 0 to 11) and user option bytes
0x1FFF 0010	Reserved	BOOT_ADD0
0x1FFF 0018	Reserved	BOOT_ADD1

- Therefore they can be read / written from the above memory locations
- To write option bytes, you will use the FLASH_CR register from before (since option bytes are in flash memory), but you will also use the FLASH_OPTCR register for option byte write / erase operations.
- You must also unlock the option bytes to write to them just like regular flash memory via the OPTLOCK bit in the FLASH_OPTCR register by doing a certain sequence defined on page 102

All Flash Registers (for convenience):

- See pages 108-116 for a summary and detailed overview of ALL registers related to FLASH, OTP and option byte programming

Table 13. Flash register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	FLASH_ACR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ARTRST	Res.	ARTEN	PRFTEN	Res.	Res.	Res.	Res.	LATENCY[3:0]			
	Reset value																					0		0	0					0	0	0	0
0x04	FLASH_KEYR	KEY[31:16]																KEY[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	FLASH_ OPTKEYR	OPTKEYR[31:16]																OPTKEYR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	FLASH_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BSY	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ERSERR	PGPERR	PGAERR	WRPERR	Res.	Res.	OPERR	EOP
	Reset value																0									0	0	0	0			0	0
0x10	FLASH_CR	LOCK	Res.	Res.	Res.	Res.	Res.	ERRIE	EOPIE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	STRT	MER2	Res.	Res.	Res.	Res.	Res.	PSIZE[1:0]		SNB[4:0]					MER/MER1	SER	PG
	Reset value	1						0	0								0	0						0	0	0	0	0	0	0	0	0	0
0x14	FLASH_OPTCR	IWDG_STOP	IWDG_STDBY	nDBANK	nDBOOT	nWRP[11:0]												RDP[7:0]								nRST_STDBY	nRST_STOP	IWDG_SW	WWDG_SW	BOR_LEV[1:0]		OPTSTRT	OPTLOCK
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	1	1	1	0	1
0x18	FLASH_ OPTCR1	BOOT_ADD1[15:0]																BOOT_ADD0[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Flash OTP:

- One time programmable bytes
- Can be read from application to change behavior based on these bytes, completely programmable and customizable
- Only programmed during flash operations aka firmware update / load, therefore can be used when uploading the same program to multiple boards but each board needs some special customization

Flash Read / Write Protection:

- Manual explains read protection well:

3.5.1 Read protection (RDP)

The user area in the Flash memory can be protected against read operations by an entrusted code. Three read protection levels are defined:

- Level 0: no read protection

  When the read protection level is set to Level 0 by writing 0xAA into the read protection option byte (RDP), all read/write operations (if no write protection is set) from/to the Flash memory or the backup SRAM are possible in all boot configurations (Flash user boot, debug or boot from RAM).

- Level 1: read protection enabled

  It is the default read protection level after option byte erase. The read protection Level 1 is activated by writing any value (except for 0xAA and 0xCC used to set Level 0 and Level 2, respectively) into the RDP option byte. When the read protection Level 1 is set:

  – No access (read, erase, program) to Flash memory or backup SRAM can be performed while the debug feature is connected or while booting from RAM or system memory bootloader. A bus error is generated in case of read request.

  – When booting from Flash memory, accesses (read, erase, program) to Flash memory and backup SRAM from user code are allowed.

  When Level 1 is active, programming the protection option byte (RDP) to Level 0 causes the Flash memory and the backup SRAM to be mass-erased. As a result the user code area is cleared before the read protection is removed. The mass erase only erases the user code area. The other option bytes including write protections remain unchanged from before the mass-erase operation. The OTP area is not affected by mass erase and remains unchanged. Mass erase is performed only when Level 1 is active and Level 0 requested. When the protection level is increased (0->1, 1->2, 0->2) there is no mass erase.

- Level 2: debug/chip read disabled

  The read protection Level 2 is activated by writing 0xCC to the RDP option byte. When the read protection Level 2 is set:

  – All protections provided by Level 1 are active.

  – Booting from RAM or system memory bootloader is no more allowed.

  – JTAG, SWV (serial-wire viewer), ETM, and boundary scan are disabled.

  – User option bytes can no longer be changed.

  – When booting from Flash memory, accesses (read, erase and program) to Flash memory and backup SRAM from user code are allowed.

  Memory read protection Level 2 is an irreversible operation. When Level 2 is activated, the level of protection cannot be decreased to Level 0 or Level 1.

Table 11. Access versus read protection level

Memory area	Protection Level	Debug features, Boot from RAM or from System memory bootloader			Booting from Flash memory		
		Read	Write	Erase	Read	Write	Erase
Main Flash Memory and Backup SRAM	Level 1	NO		NO(1)	YES		
	Level 2	NO			YES		
Option Bytes	Level 1	YES			YES		
	Level 2	NO			NO		
OTP	Level 1	NO		NA	YES		NA
	Level 2	NO		NA	YES		NA

1. The main Flash memory and backup SRAM are only erased when the RDP changes from level 1 to 0. The OTP area remains unchanged.

- As for write protection there are nWRP bits in the FLASH_OPTCR register that need to be set to allow / disallow writing to certain sectors in FLASH memory. See pages 105 - 107 for a detailed overview.

Boot Configuration and Activation:

- Boot Address:

Table 2. Boot modes

Boot mode selection		Boot area
BOOT	**Boot address option bytes**	
0	BOOT_ADD0[15:0]	Boot address defined by user option byte BOOT_ADD0[15:0] ST programmed value: Flash on ITCM at 0x0020 0000
1	BOOT_ADD1[15:0]	Boot address defined by user option byte BOOT_ADD1[15:0] ST programmed value: System bootloader at 0x0010 0000

Bits 31:16 **BOOT_ADD1[15:0]**: Boot base address when Boot pin =1
BOOT_ADD1[15:0] correspond to address [29:14],
The boot memory address can be programmed to any address in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB.
Example:
BOOT_ADD1 = 0x0000: Boot from ITCM RAM (0x0000 0000)
BOOT_ADD1 = 0x0040: Boot from System memory bootloader (0x0010 0000)
BOOT_ADD1 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)
BOOT_ADD1 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)
BOOT_ADD1 = 0x8000: Boot from DTCM RAM (0x2000 0000)
BOOT_ADD1 = 0x8004: Boot from SRAM1 (0x2002 0000)
BOOT_ADD1 = 0x8013: Boot from SRAM2 (0x2004 C000)

Bits 15:0 **BOOT_ADD0[15:0]**: Boot base address when Boot pin =0
BOOT_ADD0[15:0] correspond to address [29:14],
The boot base address can be programmed to any address in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB.
Example:
BOOT_ADD0 = 0x0000: Boot from ITCM RAM (0x0000 0000)
BOOT_ADD0 = 0x0040: Boot from System memory bootloader (0x0010 0000)
BOOT_ADD0 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)
BOOT_ADD0 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)
BOOT_ADD0 = 0x8000: Boot from DTCM RAM (0x2000 0000)
BOOT_ADD0 = 0x8004: Boot from SRAM1 (0x2002 0000)
BOOT_ADD0 = 0x8013: Boot from SRAM2 (0x2004 C000)

- When the BOOT pin is LOW, it reads from BOOT_ADD0 to see which address to boot from and vice versa for BOOT pin HIGH
    - BOOT_ADD can be any address from 0x0000 0000 to 0x3FFF FFFF ; this includes all flash address space and sram address space as well as the system bootloader
    - If out of defined memory range or in reserved memory regions, the default values are:
        - BOOT_ADD0 → ITCM-FLASH at 0x0020 0000
        - BOOT_ADD1 → ITCM-RAM at 0x0000 0000
    - When **FLASH level 2 protection** is **enabled** only booting from FLASH (ITCM or AXIM) and the system bootloader is available → if out of range, default value is ICTM-FLASH at 0x0020 0000

- When nDBANK bit is 0, dual memory bank is set, therefore we can boot from either bank 0 or bank 1:
    - The choice between bank 0 and bank 1 is set using the nDBOOT bit
- With the ability to select our boot location we can do the following:
    - Directly run the application by setting the boot address to flash or sram memory
    - Run a bootloader first which runs the application alter by setting the boot address to where the bootloader is programed (which may also be in flash or sram memory)

System Bootloader:

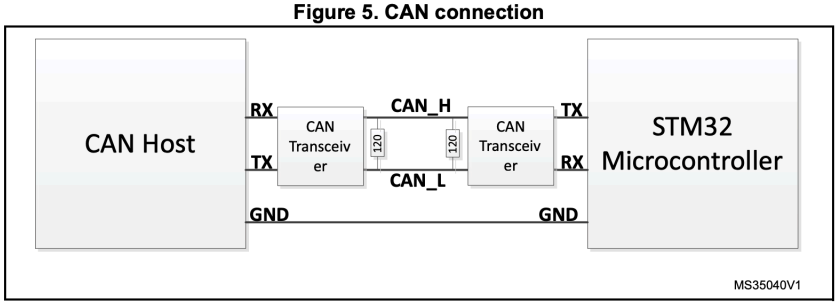
- **<u>The following information is only for the system bootloader located in system memory, NOT for any custom bootloaders that we write</u>**
- See this application note (AN2606): https://www.st.com/resource/en/application_note/an2606-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf
- Programmed by ST during production is stored at 0x0010 0000, stored in ROM (**read-only memory** or **system memory**)
    - "Bootloader activation pattern" → simply sets the correct registers from what we now know as BOOT_ADD to boot from 0x0010 0000
- Code is executed from the TCM interface (ITCM-FLASH)
- When readout protection is level 2, the MCU does not boot from system memory or SRAM, only FLASH. Therefore only certain operations in the system bootloader are available, specifically the GET, GETID, and GETVERSION commands.
    - See page 32 on AN2606 note for device ID meanings

STM32F76xxx/77xxx	USART1/USART3/ CAN2 DFU (USB device FS)/ I2C1/I2C2/I2C3/ SPI1/SPI2/SPI4	0x93	0x1FF0EDBE	USART (V3.1) CAN (V2.0) DFU (V2.2) I2C (V1.2) SPI (V1.2)

- HSE (external clock) may be required DFU / CAN interface operations **DOUBLE CHECK!!!**
- For hardware connection requirements (CAN, UART, SPI, etc..) see page 40-41 on AN2606
    - Recommends that we tie all other interface pins to a known level, to prevent activating them during bootloader sequence

To use the CAN interface, the host must be connected to the RX and TX pins of the desired CANx interface via CAN transceiver and a serial cable. A 120 Ω resistor must be added as terminating resistor.

Figure 5. CAN connection



- All write operations using bootloader commands MUST be word-aligned (address is a multiple of 4). The number of data to be written must also be a multiple of 4 (non-aligned half page write addresses are accepted).

    - STM32F7 Special Case: The memory format that can be written is typically in one byte format, however supplying a “voltage range” allows for different formats. See page 42 and 43 on AN2606
- Here is the list of all valid commands depending on memory region:

The table below lists the valid memory areas, depending upon the bootloader commands.

Table 5. Supported memory area by Write, Read, Erase, and Go commands

Memory area	Write command	Read command	Erase command	Go command
Flash	Supported	Supported	Supported	Supported
RAM	Supported	Supported	Not supported	Supported
System memory	Not supported	Supported	Not supported	Not supported
Data memory	Supported	Supported	Not supported	Not supported
OTP memory	Supported	Supported	Not supported	Not supported

- Additionally when writing to FLASH memory there are additional constraints on alignment on top of the base write operation constraints:
    - For the STM32F7 all writes must be aligned to 8 bytes on FLASH
        - Specifically for the STM32F7, you can change the alignment constraint by writing in the “device feature space” **RESEARCH MORE**
    - For a list off constraints based on MCU / device see page 45, table 7 on AN2606

STM32F7 Specific Notes:

Table 87. STM32F76xxx/77xxx configuration in system memory boot mode

Bootloader	Feature/Peripheral	State	Comment
Common to all bootloaders	RCC	HSI enabled	The HSI is used at startup as clock source for system clock configured to 60 MHz and for USART and I2C bootloader operation.
		HSE enabled	The HSE is used only when the CAN or the DFU (USB FS device) interfaces are selected. In this case the system clock configured to 60 MHz with HSE as clock source. The HSE frequency must be multiple of 1 MHz and ranging from 4 MHz to 26 MHz.
		-	The CSS interrupt is enabled for the CAN and DFU bootloaders. Any failure (or removal) of the external clock generates system reset.
	RAM	-	16 Kbytes, starting from address 0x20000000 are used by the bootloader firmware
	System memory	-	59 Kbytes, starting from address 0x1FF00000, contain the bootloader firmware
	IWDG	-	The IWDG prescaler is configured to its maximum value. It is periodically refreshed to prevent watchdog reset (if the hardware IWDG option was previously enabled by the user).
	Power	-	The voltage range is [1.8V, 3.6V] In this range: - Flash wait states: 3. - System clock frequency 60 MHz. - ART Accelerator enabled. - Flash write operation by byte (refer to bootloader memory management section for more information).
CAN2 bootloader	CAN2	Enabled	Once initialized the CAN2 configuration is: Baudrate 125 kbps, 11-bit identifier. **Note:** CAN1 is clocked during CAN2 bootloader execution because CAN1 manages the communication between CAN2 and SRAM.
	CAN2_RX pin	Input	PB5 pin: CAN2 in reception mode. Used in alternate push-pull, pull-up mode.
	CAN2_TX pin	Output	PB13 pin: CAN2 in transmission mode. Used in alternate push-pull, pull-up mode.
CAN2 and DFU bootloaders	TIM11	Enabled	This timer is used to determine the value of the HSE. Once HSE frequency is determined, the system clock is configured to 60 MHz using PLL and HSE.

The system clock is derived from the embedded internal high-speed RC for USARTx and I2Cx bootloaders. This internal clock is also used for CAN and DFU (USB FS device), but only for the selection phase. An external clock multiple of 1 MHz (between 4 and 26 MHz) is required for CAN and DFU bootloader execution after the selection phase.

Note: Due to HSI deviation and since HSI is used to detect HSE value, the user must use low rather than high frequency HSE crystal values (low frequency values are better detected due to larger error margin). For example, it is better to use 8 MHz instead of 25 MHz.