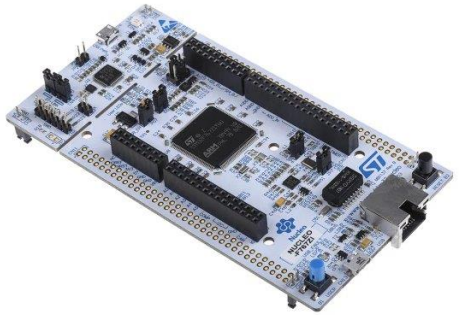
life.augmented

STM32H7R/S workshop bootflash MCU + OSPI

Hands-On Session Objectives

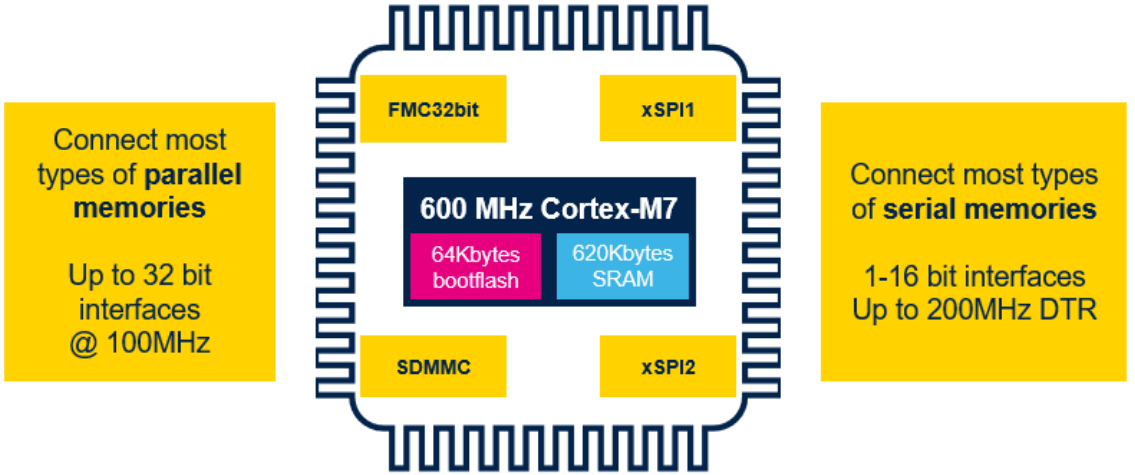
- The example will guide you through creating a project based on an STM32H7R/S bootflash MCU interfacing an external NOR serial flash memory over OSPI
- OSPI (Octal Serial Peripheral Interface) utilizes eight data lines to connect an external NOR serial flash memory to the MCU
- We will learn how to leverage the STM32Cube ecosystem for developing applications that utilize external memory
- We'll be utilizing NUCLEO-H7S3L8 development board as our hardware platform



life.augmented

What is a bootflash MCU?

- The bootflash MCU is an MCU that comes with a small embedded Flash (64KB) used primarily for the initial boot sequence with the user application residing in external memory, offering scalability and providing flexibility in external memory size and type
- High-speed external memory interfaces provide the flexibility to select from a wide range of memory options and architectures (up to 200 MHz DTR)
- Cache boosts system performance by enabling rapid access to frequently used code/data, vastly reducing the number of CPU accesses to the external memory



life.augmented

Which projects are needed in application utilizing external memory?

- **Bootloader:** to configure necessary hardware incl. the xSPI (serial memory interface) and jump to the application located in external memory
- **External Memory Loader:** to access and manage the external memory (read/write/erase), to get the application firmware to the external memory
- **Application:** the application firmware, which is located in the external flash memory

Bootloader

- Its task is to initialize the system's hardware, system clock and particularly configure the serial memory interface (xSPI) and hand over the code execution to the main application firmware in external memory
- The bootloader code will be located in the MCU's internal flash

Boot

External memory loader

- It initializes the serial memory interface (xSPI) and then enables to manage the external memory, allowing for programming, reading, and erasing of its contents
- It will be used to get the application firmware to the external memory
- During its use, the code is loaded into MCU's internal SRAM and executed from there, ensuring that the internal flash content remains unchanged
- It can be used with multiple programming tools, such as STM32CubeIDE, STM32CubeProgrammer, IAR EWARM, Keil-MDK

External memory loader

Application

- The application firmware, which is located in external memory
- For simplicity, we will use the 'Hello World' of embedded systems (toggling an LED)

Enhanced and Extended STM32Cube Ecosystem

- STM32CubeMX can generate all three projects for us!

Application

Boot

External memory loader

Application Project

Stored in external memory and launched by Boot/FSBL

BOOT/FSBL (First Boot Loader) Project

Manages the startup of the application subproject

EXTMEMLOADER Project

Tool to download the application binary to external memory

User Application

External memory configuration

Code fully generated! No updates necessary but manual adaptation is of course possible if needed…

life.augmented

Initialization Code Generation for Application

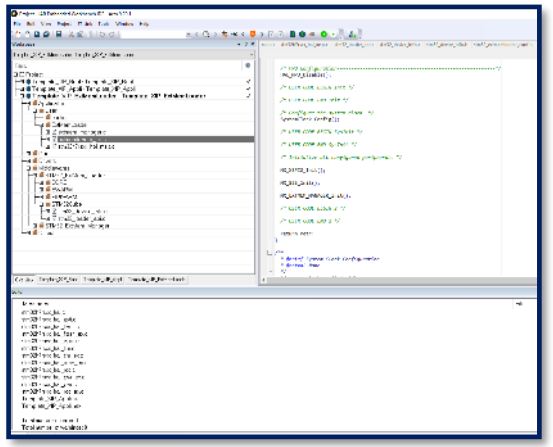
Application

STM32CubeMX takes care of project setup with pinout, clock tree configuration, peripheral initialization, and middleware setup, along with initialization C code generation.

STM32CubeMX

Configure and generate code

STM32CubeIDE IAR SYSTEMS KEIL Tools by ARM Visual Studio Code

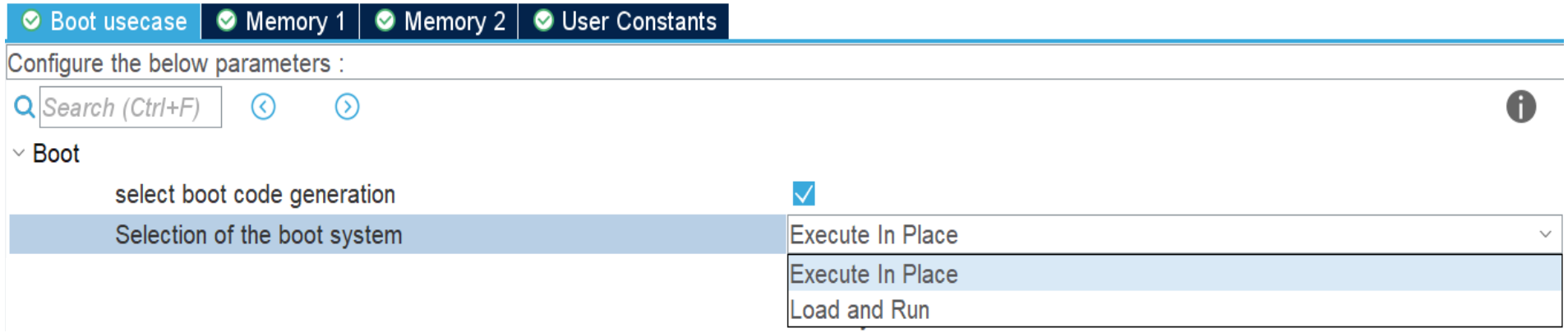


Edit, Build and debug

Bootloader Generation

Boot

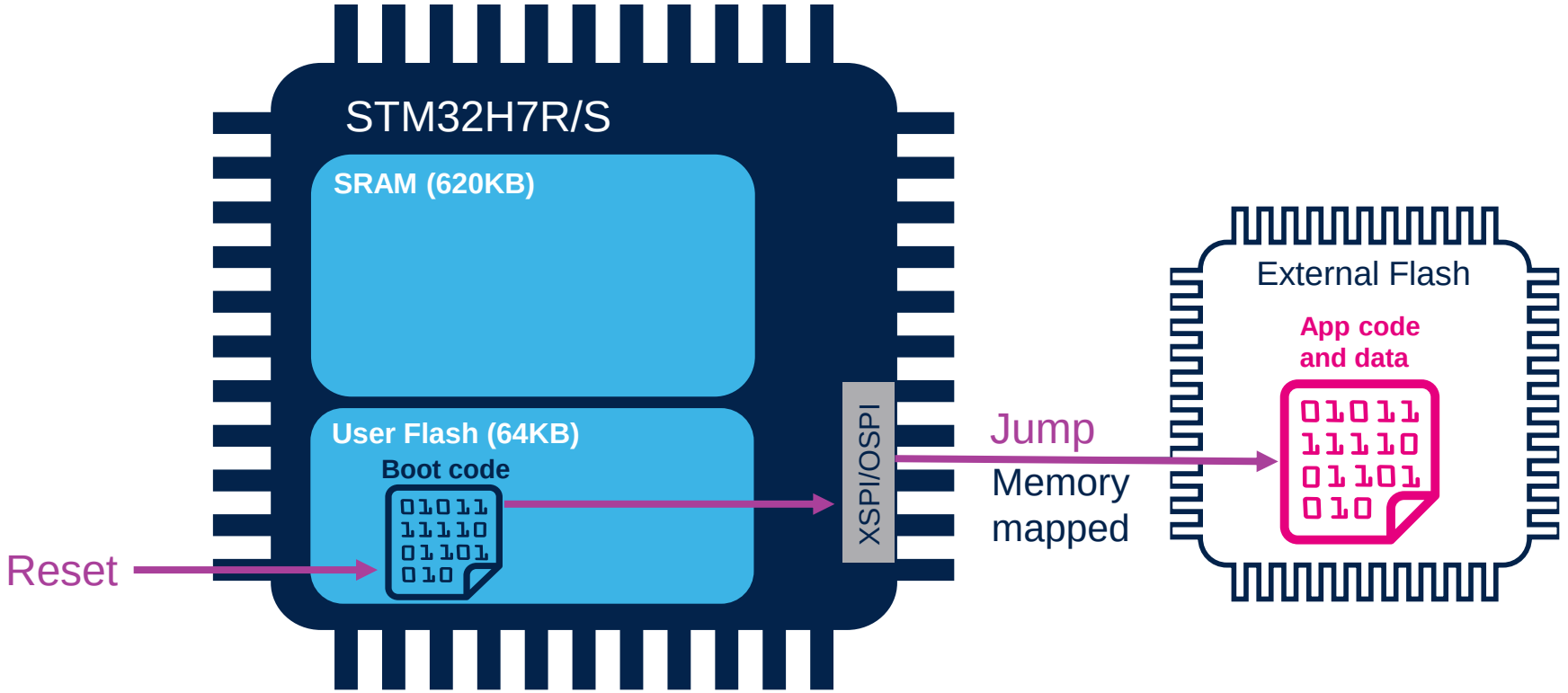
STM32CubeMX facilitates the creation of your boot project including managing access to your selected external memory.



Boot Option Execute-in-Place

Boot

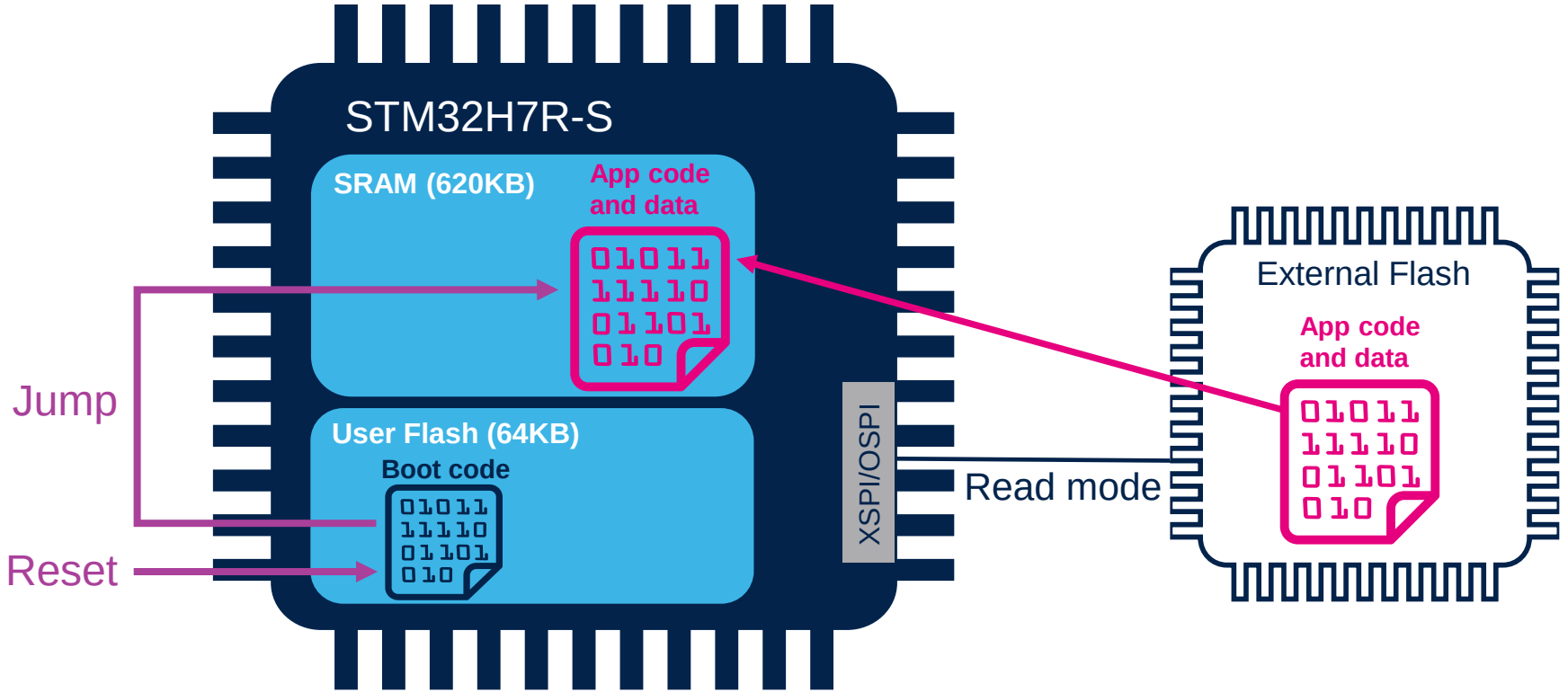
STM32CubeMX facilitates the creation of your boot project including managing access to your selected external memory for the Execute-in-Place boot option.



Boot Option Load-and-Run

Boot

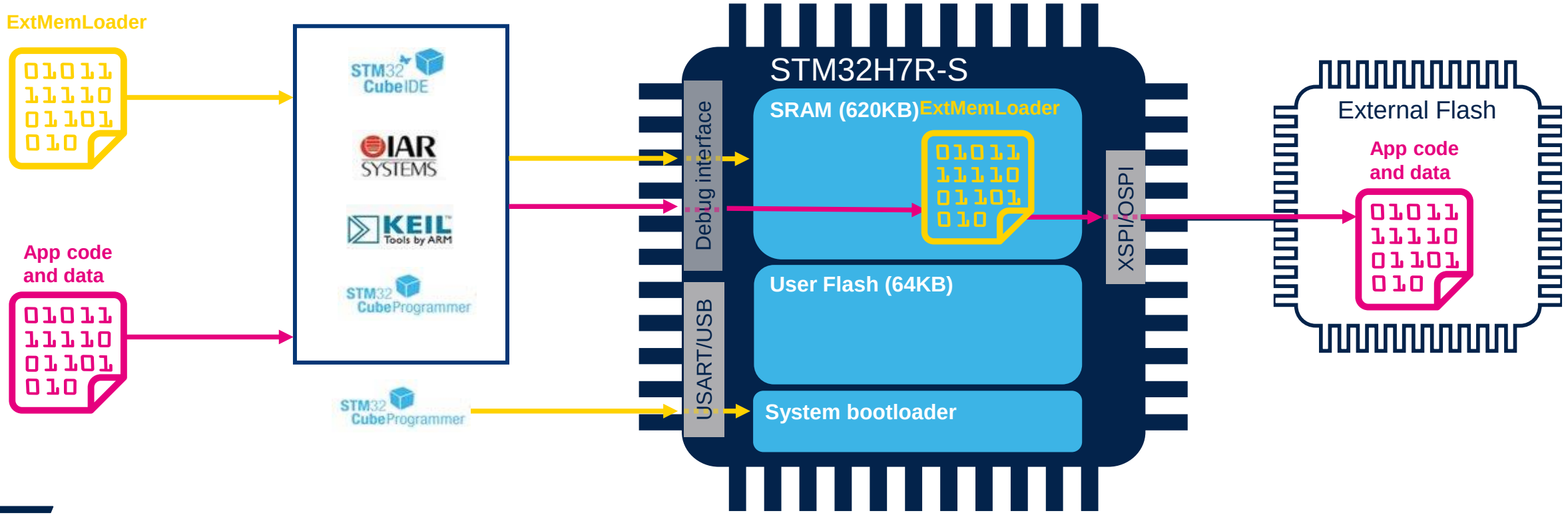
STM32CubeMX facilitates the creation of your boot project including managing access to your selected external memory for the Load-and-Run boot option.



External Memory Loader Generation

External memory loader

STM32CubeMX assists in creating an external memory loader that is customized for your selected external memory.



New Middleware for External Memory Management

New !

EXTMEM_Manager middleware

- A uniform API for accessing different types of external memory (NOR SFDP, SD Card, PSRAM)
- A BOOT system to launch a user application stored in an external memory

EXTMEM_Loader middleware

- Middleware that assists users in creating customized external memory loaders in the specific formats required by differents IDEs
- It relies on the EXTMEM_Manager and its API

IAR SYSTEMS KEIL Tools by ARM STM32 CubeProgrammer STM32 CubeIDE

life.augmented

Key Concepts and Principles to Understand Before the Hands-On Session

HSLV (High-Speed Low Voltage)

- HSLV is a characteristic of certain I/O ports that allows them to operate at higher speeds while using lower voltage levels.
- This feature is particularly useful for applications that require fast data transfer rates without the higher power consumption typically associated with standard voltage levels.
- In our case, the XSPIM2 domain is powered by 1.8 V.
  Since we aim to operate the serial memory interface at its maximum speed of 200 MHz, we need to enable the HSLV.
- HSLV is only to be activated when VDD ≤ 2.7 V!

life.augmented

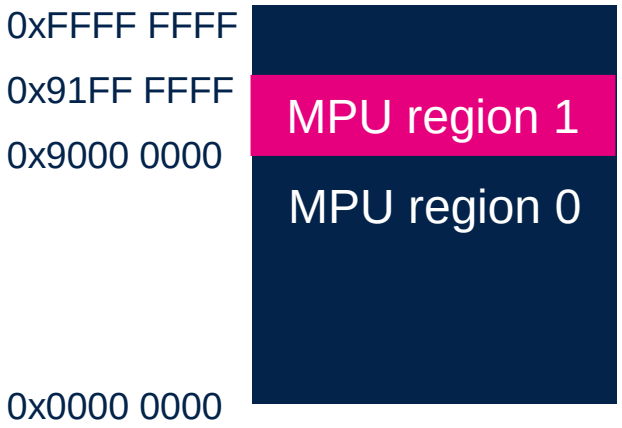
Warning: Ensure that the HSLV (High-Speed Low-Voltage) feature is enabled only when the domain is actually operating at low voltage!
Activating HSLV while the memory is powered by 3.3V could potentially damage the STM32 MCU!
There is no built-in hardware protection, therefore the user bears complete responsibility for ensuring that the HSLV feature is activated under the correct conditions.

MPU (Memory Protection Unit)

- A part of the ARM Cortex-M7, providing hardware-based memory access control
- The MPU allows to divide the memory space into a number of regions and define access permissions and memory attributes (e.g. cacheable) for each of them
- When an access rule is violated, a fault exception is triggered
- Some MPU use cases:
  - To isolate memory regions among different user tasks in an RTOS environment
  - Preventing application tasks from corrupting stack and data memory used by other tasks and the OS kernel
  - Defining some parts of memory as non-executable (to prevent code injection attacks in SRAM)
  - Prohibiting the ARM Cortex-M7 core from performing optimizations or speculative prefetch in selected memory areas
  - Designate parts of memory as non-cacheable to maintain data coherency (e.g., Ethernet + DMA)

MPU (Memory Protection Unit) Use in the Hands-On

- In our hands-on session, we will let STM32CubeMX to generate a background region (MPU region 0) with recommended safe configuration for the entire addressable memory space of 4 GB.
- This background region configures the entire external memory area in the memory map as non-accessible, prohibiting the use of external memory.
- We need to create an additional MPU region 1 with a size equal to that of the external memory, and define its access permissions and memory attributes (e.g., allow code execution from this part of memory and making it cacheable to benefit from a performance boost due to caching)
- Both MPU regions will be overlapping, in that case the priority is as follows: MPU region 1 > MPU region 0



life.augmented

What is SFDP protocol?

- SFDP (Serial Flash Discoverable Parameters) is a JEDEC standard supported (mainly) by serial NOR flash memories
- It allows to discover basic characteristics and capabilities of serial NOR flash devices without needing prior knowledge of the specific flash device
- The common way to retrieve memory device information simplifies handling flash memory devices from different manufacturers
- The device information is stored in the form of internal parameter tables
- SFDP is used by External Memory Manager middleware

Definition / Function
Sector size, BP bits type
4KB Erase opcode
Read mode, Address mode, DTR mode
Flash density
Read mode interface, Mode bits, dummy cycle
Sector size, Sector erase opcode

Theory's Over - Let's Begin the Practical Session

life.augmented

Our technology starts with You

Find out more at www.st.com

life.augmented