Lecture 4
STM32CubeIDE, HAL

Cris Ababei
Dept. of Electrical and Computer Engineering

MARQUETTE UNIVERSITY

BE THE DIFFERENCE.

1

1

Outline

- STM32CubeIDE
- ST HAL
- CMSIS
- STM32CubeMX

2

2

1

New Project in STM32CubeIDE

- When a new project is created in STM32CubeIDE, it downloads the corresponding Cube Firmware Package for the selected Nucleo board

- This includes several components, including:
  - Complete HAL for given STM32 family: Hardware Abstraction Layer (HAL)
    - Set of libraries that allow to drive the microcontroller peripherals and core features without dealing with the details of the given MCU
  - Additional Middleware packages
  - Examples projects for development boards

3

3

Hardware Abstraction Layer (HAL)

User Program

HAL (Defined by ST) CMSIS (Defined by ARM)

```
 97    while (1)
 98    {
 99      /* USER CODE END WHILE */
100      /* USER CODE BEGIN 3 */
101      HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
102      HAL_Delay(500);
103    }


 74      /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 75      HAL_Init();

 81      /* Configure the system clock */
 82      SystemClock_Config();
```


4

4

2

Cortex Microcontroller Software Interface Standard (CMSIS)

- ARM - actively working on a way to standardize the software infrastructure among MCUs vendors
- This is an evolving effort
- Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for Cortex-M processors
- CMSIS consists of many components, including:
  ° CMSIS-CORE: API for the Cortex-M processor core and peripherals
  ° CMSIS-Driver: Defines generic peripheral driver interfaces for middleware
  ° CMSIS-RTOS API: Common API for Real-Time Operating Systems
  ° ... many more

5

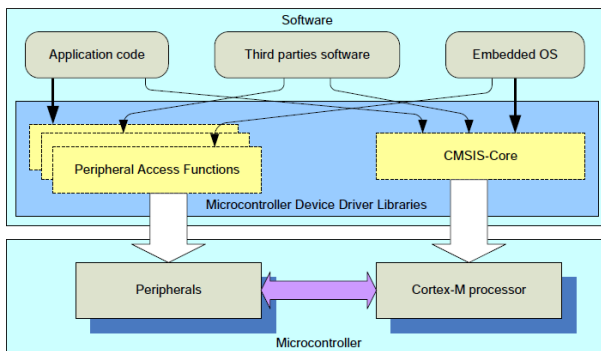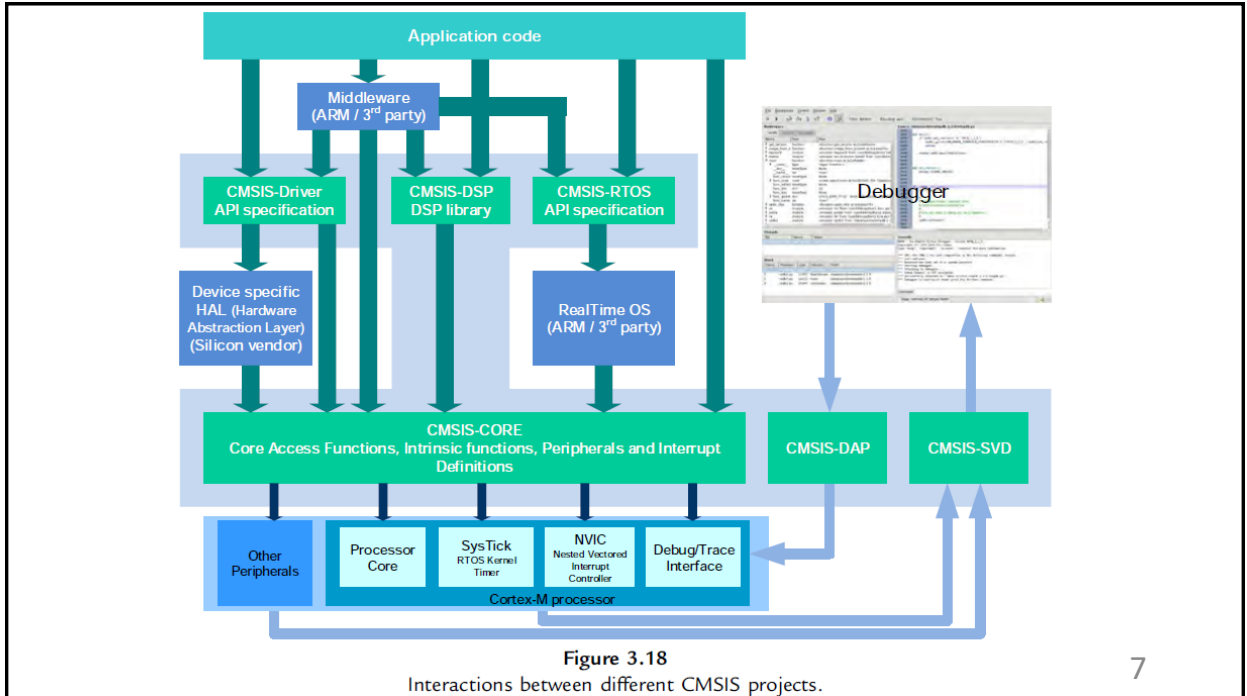# Cortex Microcontroller Software Interface Standard (CMSIS)

Figure 3.17
CMSIS-CORE provides standardized access functions for processor features.

- CMSIS contains definitions for the various registers
- Allows easy access to MCU's registers
  - One can then implement his/her own HAL
- CMSIS is defined by ARM and not ST
  - Greatly aids in portability
- HAL may have more bugs
  - CMSIS cleaner and more stable
- One can code by using HAL and CMSIS

6

Figure 3.18
Interactions between different CMSIS projects.

7

STM32CubeMX Tool

- A fundamental tool integrated inside STM32CubeIDE.

- Upon creation of new project, it can be used for Peripheral Configurations.
  - It makes it easy for programmers; otherwise, one needs to dig into datasheets and user manuals
  - Programmer can ignore specific implementation details underlying a peripheral configuration

- Used both to:
  1. Choose the right hardware connections
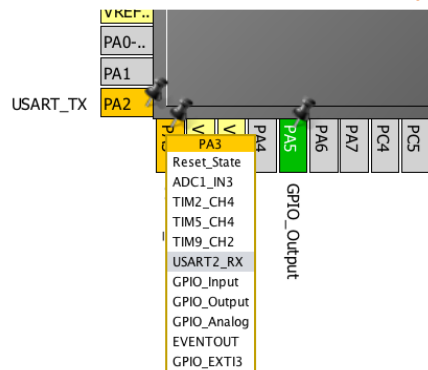 2. Generate the code necessary to configure the ST HAL

8

STM32CubeMX Tool

- Once a new project has been created, STM32CubeIDE automatically opens a file in the project folder named <project-name>.ioc

- This file is the CubeMX main project file, containing all the configurations performed in CubeMX

- Starting from them, CubeMX will generate the corresponding project structure, with all source files and libraries needed to use the selected peripherals and Middleware components

9

STM32CubeMX Tool

- Example of usage:
  - ° Most of MCU pins can have alternate functionalities
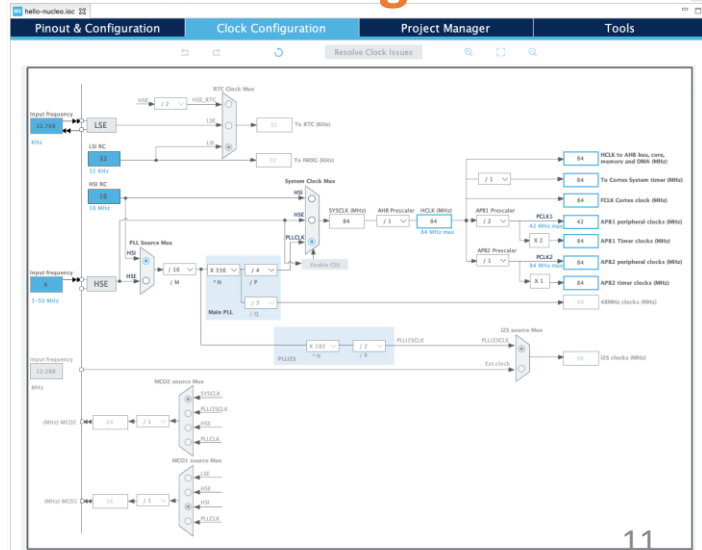 - ° CubeMX can be used to select desired functionality for a pin
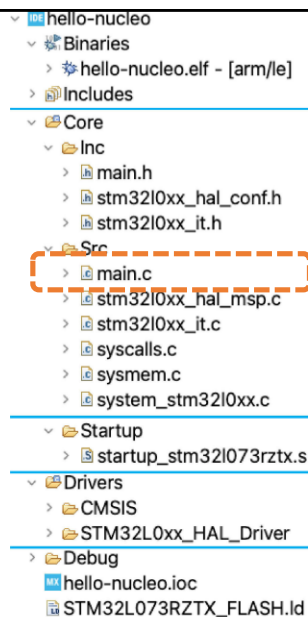

10

STM32CubeMX Tool - Clock Configuration View

- Clock Configuration view is the pane where all configurations related to clocks management take place

Project Structure

- After all configurations with CubeMX are done, the <project-name>.ioc file is saved and the skeleton files for the project are created; including main.c

Binaries and **Includes** are auto-generated folders containing the object file created by the compiler (in the ELF format) and the list of all *include paths* where the compiler (GCC) looks for header files.

The **Core** folder contains all application specific source files. Those files are strictly connected with the project and MCU settings in CubeMX(including Middleware components).
While it is strongly suggested to add application specific files here, Eclipse allows you to rearrange the project structure as you want, unless all include paths are properly configured. However, by changing the project structure, you will no longer able to perform changes to CubeMX configurations without compromising the whole project. Trust me: leave them as-is.

The **Startup** folder contains a source file coded in assembler named *startup file*, which contains the very first code executed after a Reset. We will analyze it later.

The **Drivers** folder contains both the CMSIS and CubeHAL library related files. The content of these folders is generated by CubeMX, and you should never change it unless you exactly do what are you doing.

The **Debug** folder contains all files generated by the compiler (relocatable, intermediate files, etc) and by Eclipse to generate the final binary file. The name of this folder is related to the active Build Profile. It is safe to delete it, if needed.
The **.ioc** file is the CubeMX project file, while the **.ld** file is a linker script used to define the MCU's memories layout (FLASH, RAM, CCM, etc.). We will deal with these files later in the book.

main.c **Automatically Created**

- The project generated by CubeIDE automatically contains all the necessary code to build a self-consistent application

- main.c includes already a template, which includes code that takes care of default configurations
  - One can then add code to this file to implement desired functionality
  - For example, the blink LED example needs just a few additional lines of code

```
97    while (1)
98    {
99      /* USER CODE END WHILE */
100     /* USER CODE BEGIN 3 */
101     HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
102     HAL_Delay(500);
103   }
```

13

main.c

- **HAL_Init()**
  - Initializes the CubeHAL framework.
  - Responsible of the very first MCU initialization.

- **SystemClock_Config()**
  - Configures the MCU to work with one of the possible clock sources.

- **MX_GPIO_Init()**
  - Initializes I/O pins, according to the graphical configuration done in STM32CubeMX.

- **MX_USART2_UART_Init()**
  - Initializes the UART2 peripheral, which is wired to the ST-LINK interface in all Nucleo boards.

14

Blink LD2 LED Example
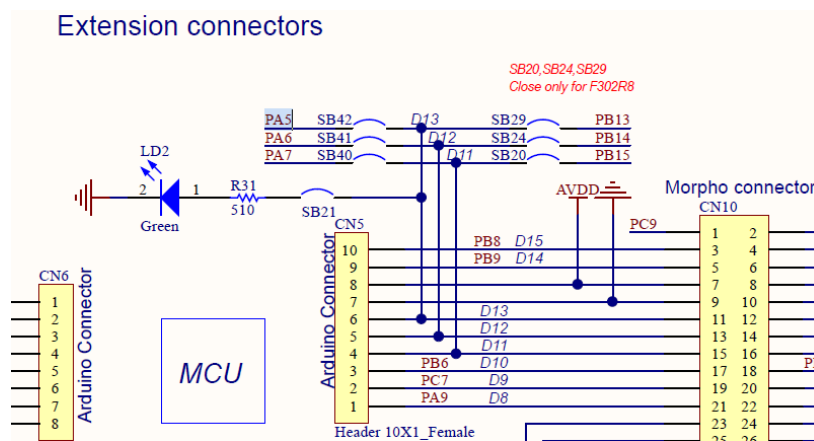
- HAL_GPIO_TogglePin() function inverts the logical state of the PIN connected to the LD2 LED (corresponds to PIN 5 of the GPIO port A in all Nucleo-64 boards)

- HAL_Delay() introduces a delay of 500ms (LD2 will blink at 1Hz rate)

- PA5 is shorthand for PIN5 of GPIO port A, which is the standard way to indicate a GPIO in STM32 world.

- STM32CubeMX automatically defines the macro LD2_GPIO_Port and LD2_Pin so that their expansion corresponds to GPIOA port and PIN5.

15

Schematic Diagram of Board

- See Schematic Diagram of Nucleo board to check connection of PA5:

16

Core/Inc/stm32XXxx_hal_conf.h

- File where HAL configurations are translated into C code, using several macros.
- These macros are used to "instruct" the HAL about enabled MCU functionalities.
- They are used to selectively include HAL modules at compile time.
- When you need a module, you can simply uncomment the corresponding macro.

```
Filename: Core/Inc/stm32XXxx_hal_conf.h

55   #define HAL_UART_MODULE_ENABLED
56   /**define HAL_USART_MODULE_ENABLED    */
57   /**define HAL_IRDA_MODULE_ENABLED     */
58   /**define HAL_SMARTCARD_MODULE_ENABLED   */
59   /**define HAL_SMBUS_MODULE_ENABLED    */
60   /**define HAL_WWDG_MODULE_ENABLED     */
61   /**define HAL_PCD_MODULE_ENABLED      */
62   #define HAL_GPIO_MODULE_ENABLED
63   #define HAL_EXTI_MODULE_ENABLED
64   #define HAL_DMA_MODULE_ENABLED
65   #define HAL_I2C_MODULE_ENABLED
66   #define HAL_RCC_MODULE_ENABLED
67   #define HAL_FLASH_MODULE_ENABLED
68   #define HAL_PWR_MODULE_ENABLED
69   #define HAL_CORTEX_MODULE_ENABLED
```

17

17

Core/Inc/stm32XXxx_it.h and Core/Src/stm32XXxx_it.c

- Where all the Interrupt Service Routines (ISR) generated by CubeMX are stored

- For example, in the case of Blink LED LD2 project:
  - void SysTick_Handler(void)
  - This function is the ISR of the SysTick timer - the routine invoked when the SysTick timer reaches 0. But where is this ISR invoked?
    - A: Nested Vectored Interrupt Controller (NVIC)
  - Cortex-M defines the SysTick_Handler to be the fifteenth exception in the NVIC vector array. But where is this array defined?
    - A: Inside the Core/Startup folder, a special assembly file: **Core/Startup/startup_stmXXxx.s**

18

18

```
S startup_stm32l053r8tx.s  ×
118 *****************************************************************************/
119    .section   .isr_vector,"a",%progbits
120    .type  g_pfnVectors, %object
121    .size  g_pfnVectors, .-g_pfnVectors
122
123
124 g_pfnVectors:
125    .word  _estack
126    .word  Reset_Handler
127    .word  NMI_Handler
128    .word  HardFault_Handler
129    .word  0
130    .word  0
131    .word  0
132    .word  0
133    .word  0
134    .word  0
135    .word  0
136    .word  SVC_Handler
137    .word  0
138    .word  0
139    .word  PendSV_Handler
140    .word  SysTick_Handler
141    .word     WWDG_IRQHandler              /* Window WatchDog            */
142    .word     PVD_IRQHandler               /* PVD through EXTI Line detection */
```

9

19

Core/Src/stm32XXxx_hal_msp.c

- "MSP" stands for MCU Support Package

- Defines all initialization functions used to configure the on-chip peripherals according to the user configuration
  - PIN allocation
  - Enabling of clock
  - Use of DMA and Interrupts

20

20

- Example: A peripheral is essentially composed of two things:
  1. Peripheral itself (for example, the SPI2 interface)
  2. Hardware pins associated with this peripheral
  ○ ST HAL is designed so that the SPI module of the HAL is generic and abstracted from the specific I/O settings - which may differ due to the MCU package and the user-defined hardware configuration
  ○ ST developers left to the user the responsibility to "fill" this piece of the HAL with the code necessary to configure the peripheral - using a sort of **callback** routines.
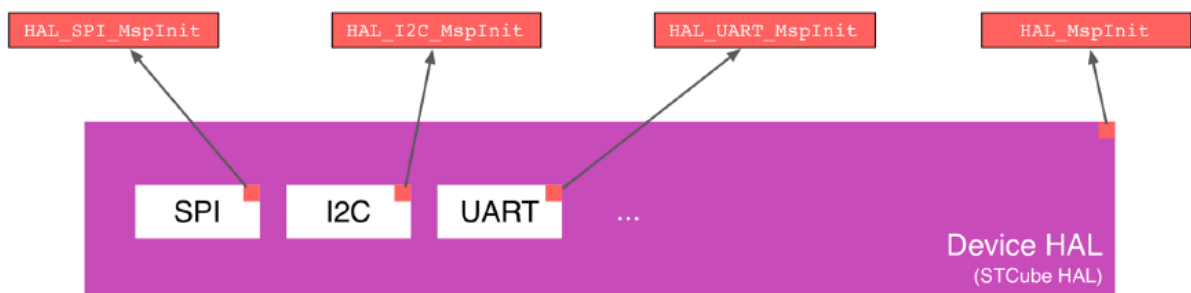   - This code resides inside Core/Src/stm32XXxx_hal_msp.c

21

Figure 4.15: The relation between MSP files and the HAL

22

```
  main.c      stm32l0xx_hal_msp.c  ×
85⊖ void HAL_UART_MspInit(UART_HandleTypeDef* huart)
86 {
87    GPIO_InitTypeDef GPIO_InitStruct = {0};
88    if(huart->Instance==USART2)
89    {
90    /* USER CODE BEGIN USART2_MspInit 0 */
91
92    /* USER CODE END USART2_MspInit 0 */
93      /* Peripheral clock enable */
94      __HAL_RCC_USART2_CLK_ENABLE();
95
96      __HAL_RCC_GPIOA_CLK_ENABLE();
97⊖    /**USART2 GPIO Configuration
98    PA2      ------> USART2_TX
99    PA3      ------> USART2_RX
100    */
101    GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
102    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
103    GPIO_InitStruct.Pull = GPIO_NOPULL;
104    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
105    GPIO_InitStruct.Alternate = GPIO_AF4_USART2;
106    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
107
108    /* USER CODE BEGIN USART2_MspInit 1 */
109
110    /* USER CODE END USART2_MspInit 1 */
111    }
112
113 }
```

23

Call Hierarchy

• main.c → MX_USART2_UART_Init() →
HAL_UART_Init() → HAL_UART_MspInit()

Figure 4.16: The Call Hierarchy of the function HAL_UART_MspInit()

24

24