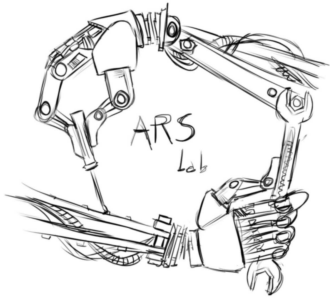
Organisation of STM32 CubeMX Projects

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course

Basic Organisation of STM32 CubeMX Projects

- The `main.c` file includes several auto-generated functions and variables that must coexists with user code
- User code must be placed within tags: `USER CODE BEGIN`, `USER CODE END`
- If the user code is placed elsewhere, it could be overwritten by the CubeMX tool

```
/* MCU Configuration--------------------------------- ... */

/* Reset of all peripherals, Initializes the Flash .... */
HAL_Init();

/* Configure the system clock */
SystemClock_Config();

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_TIM1_Init();

/* USER CODE BEGIN 2 */

/* USER CODE END 2 */
```

Parts of the `main.c` file

- Preamble:
  - **Includes**
  - Global variables
  - Function prototypes
- Functions:
  - main()
  - CubeMX auto-generated function
  - User defined functions

```
/* Includes ------------------------------------------------------------------
#include "main.h"
#include "stm32f3xx_hal.h"

/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */
```

Parts of the `main.c` file

- Preamble:
  - Includes
  - **Global variables**
  - Function prototypes
- Functions:
  - main()
  - CubeMX auto-generated function
  - User defined functions

```
/* Private variables ---------------------------------------------
TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------

/* USER CODE END PV */
```

Parts of the `main.c` file

- Preamble:
  - Includes
  - Global variables
  - **Function prototypes**
- Functions:
  - main()
  - CubeMX auto-generated function
  - User defined functions

```
/* Private function prototypes -----------------------------------------------
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM1_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------

/* USER CODE END PFP */
```

Parts of the `main.c` file

- Preamble:
  - Includes
  - Global variables
  - Function prototypes
- Functions:
  - **main()**
  - **CubeMX auto-generated function**
  - **User defined functions**

```
int main(void)
{ ... }

void SystemClock_Config(void)
{ ... }

static void MX_TIM1_Init(void)
{ ... }

static void MX_GPIO_Init(void)
{ ... }

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */
```

Parts of the `main()` function

- HAL initialization
- Clock Configuration
- Peripheral Configuration
- User code initialization
- Infinite loop

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM1_Init();

  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Peripherals Handling through HAL functions (1)

- Each peripheral is represented by a **handle**, an opaque variable used to access the peripheral through HAL

```
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart2;
```

- Initialization is performed by the CubeMX-generated function:

```
static void MX_TIM1_Init(void)
{
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 64000;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Peripherals Handling through HAL functions (2)

- HAL functions have the following form:

  `HAL_`*periph*`_`*func*`(`*params*`)`

- periph is the peripheral name, e.g. GPIO, TIM1, TIM2, USART1, ...
- func is the function of the peripheral to be called, e.g. ReadPin, WritePin, Base_Start, ...
- params are the function parameters: the **pointer to the handle** is (in the majority of cases) the first parameter; the other are dependent of the function type

Some Examples:

- `HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3)`, reads the PIN 3 of GPIO port A
- `HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET)`, sets the PIN 12 of GPIO port C
- `HAL_UART_Transmit(&huart2, buffer, 12, 1000)`, writes 12 bytes from buffer to the UART2

Peripherals Handling through Special Function Registers

- Each peripheral is represented by **pointer to a structure** that allows the access to SFRs
- All the SFRs of the peripherals are the *fields* of the structure
- The names of the pointer variable and fields are the same as the SFR reported in the MCU manual

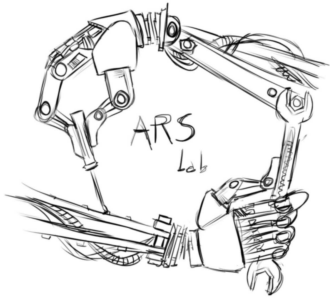
Some Examples:

- `GPIOA`, the pointer to the structure representing SFRs of GPIO port A
- `IDR`, the "input data register" used to read pins set as inputs
- `GPIO->IDR` accesses the input data register of the GPIO port A
- `TIM1`, the pointer to the structure representing SFRs of TIMER1
- `CNT`, the "counter register" of the timer
- `TIM1->CNT` accesses the counter register of the timer 1

Organisation of STM32 CubeMX Projects

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course