**Introduction**

Name – Mohammad Sahil

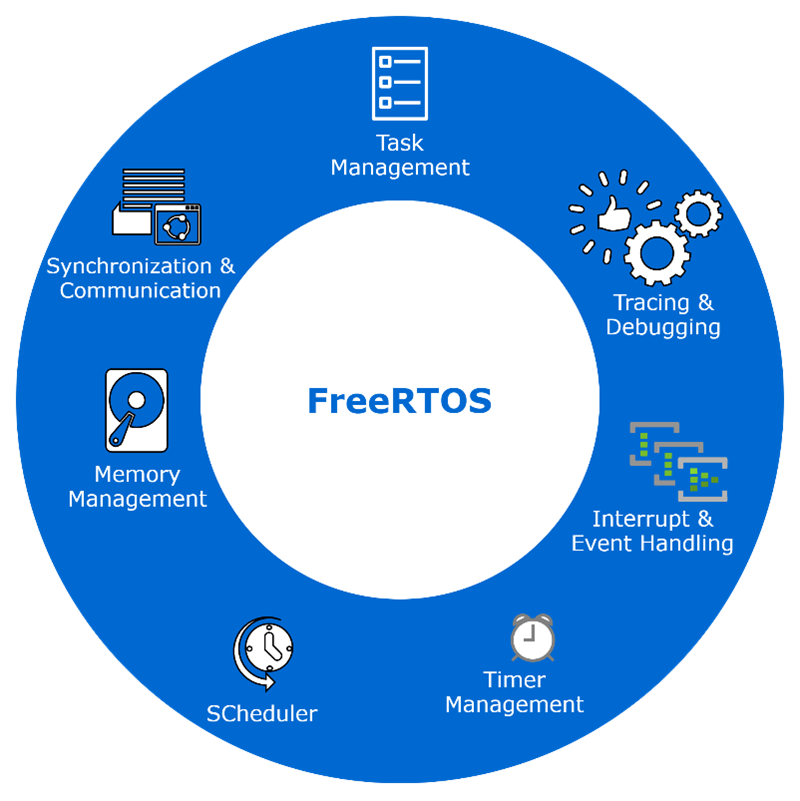
ECE – 6th Sem

Project name – Implementing FreeRTOS on STM32

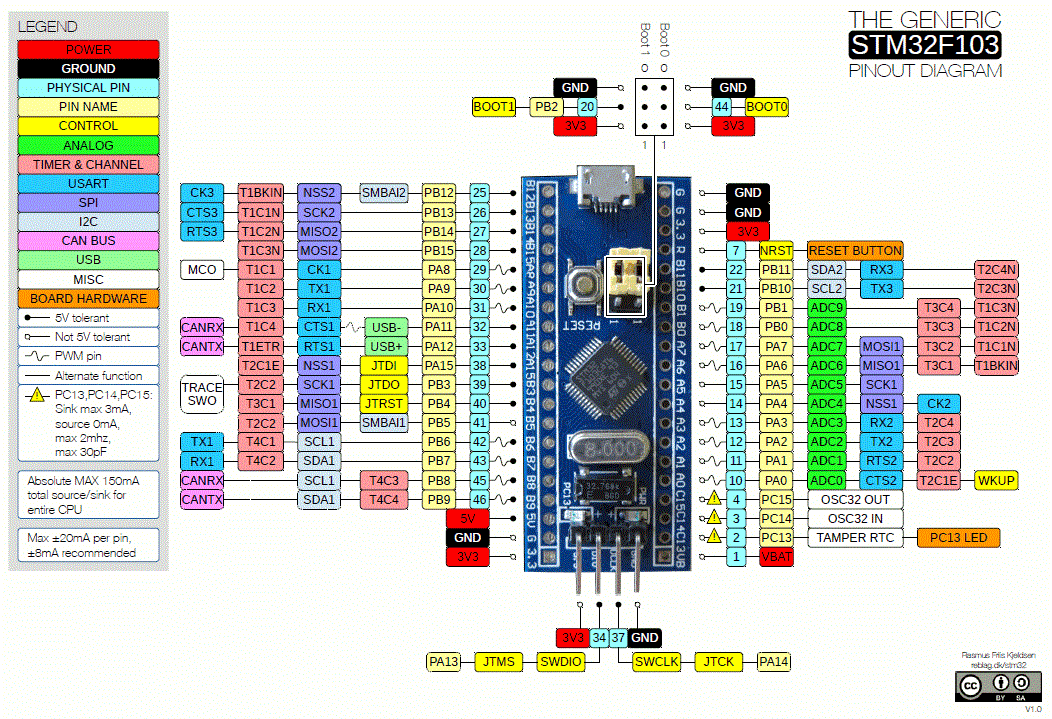
**Here is a comparison between General-Purpose Operating Systems (GPOS) and Real-Time Operating Systems (RTOS):**

| **Feature** |  |  | **General-Purpose Operating System (GPOS)** | **Real-Time Operating System (RTOS)** |
| --- | --- | --- | --- | --- |
| **Purpose** |  |  | Designed for general computing tasks like desktop applications, gaming, and browsing. | Designed for time-critical applications like robotics, avionics, and industrial automation. |
| **Responsiveness** |  |  | Prioritizes high throughput and user experience. May have unpredictable delays. | Ensures predictable and deterministic response to events. |
| **Task Scheduling** |  |  | Non-deterministic scheduling (e.g., time-sharing, round-robin, or priority-based). | Deterministic scheduling, often using fixed-priority or pre-emptive algorithms. |
| **Latency** |  |  | Can tolerate higher latency for tasks. | Low and predictable latency is critical for real-time operations. |
| **Pre-emption** |  |  | Tasks may not be pre-empted immediately, leading to delays. | Supports immediate pre-emption to meet real-time deadlines. |
| **Resource Management** |  |  | Optimized for general efficiency and multitasking. | Optimized for timely resource allocation to critical tasks. |
| **Interrupt Handling** |  |  | Can have delayed interrupt handling due to lower priority mechanisms. | Interrupts are handled promptly with minimal latency. |
| **Example Use Cases** |  |  | Desktops, smartphones, servers (e.g., Windows, macOS, Linux). | Embedded systems, medical devices, automotive systems (e.g., FreeRTOS, VxWorks). |
| **Error Tolerance** |  |  | Designed to recover from crashes without stringent real-time guarantees. | Focused on high reliability with fault-tolerance mechanisms. |
| **Hardware Requirements** |  |  | May require higher memory and processing power. | Designed to run efficiently on resource-constrained devices. |
| **Cost** |  |  | May have higher costs depending on licensing and hardware needs. | Often lightweight and tailored for specific hardware, reducing cost. |

* **Objective**: To implement FreeRTOS on STM32 Blue Pill for efficient multitasking and real-time performance.
* **What is FreeRTOS?**
  + FreeRTOS is an open-source real-time operating system designed for embedded systems.
  + Provides multitasking capabilities, resource management, and synchronization mechanisms.



* **What is STM32 Blue Pill?**
  + A low-cost development board based on the STM32F103C8T6 microcontroller.
  + Features include a 72 MHz ARM Cortex-M3 core, 64 KB Flash memory, and 20 KB SRAM.
  + 3 Uart, 3 Spi,2 CAN,2 I2c.



**: Why Use FreeRTOS on STM32?**

* Efficient multitasking: Enables running multiple tasks simultaneously.
* Real-time performance: Provides predictable behaviour for time-sensitive applications.
* Hardware abstraction: Simplifies hardware management.

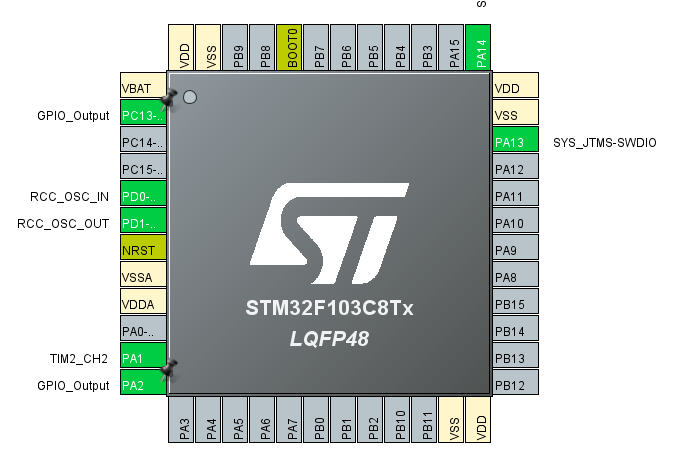
**Hardware Requirements**

The hardware listed are suited for the following project and listed source code

Though you can use any component you want by tweaking the source code

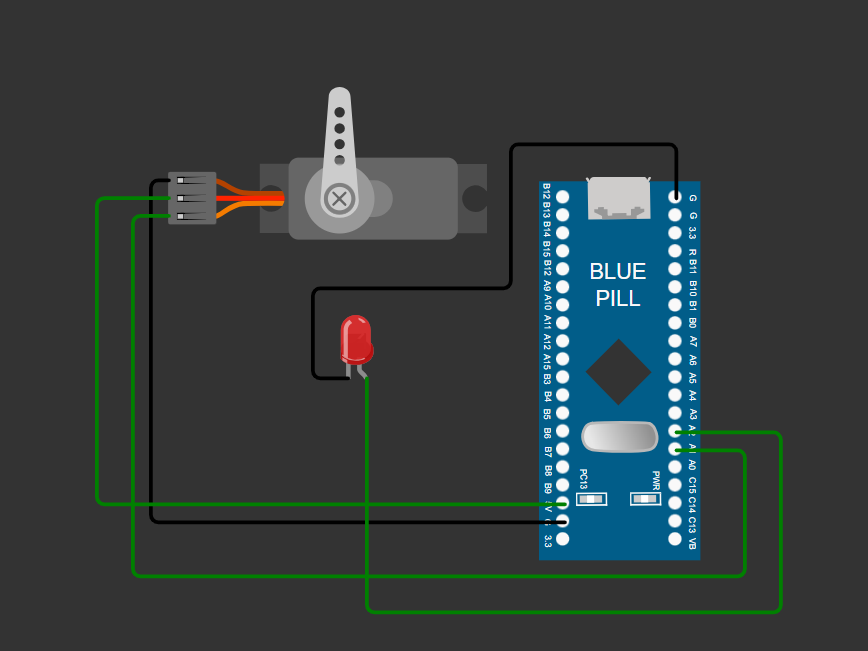
* ST-Link V2 programmer
* Bluepill board
* A servo
* Some LEDs (optional)

**IOC FILE**

****

In STM32 Cube IDE, an "IOC file" refers to a project configuration file generated by the STM32CubeMX tool, which stores all the settings related to your microcontroller's peripheral initialization, pin assignments, and clock configuration, essentially acting as a central hub for managing your device's low-level settings within the development environment; allowing for easy code generation and project management

**Schematic**

****

**Power given externally to the servo**

As Blue pill is operating upon 3.3V which is not sufficient for the servo as they are designed to operate between 4.8 to 6 V (SG90).

**CODE**

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2025 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

#include "cmsis\_os.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

TIM\_HandleTypeDef htim2;

osThreadId Blink1Handle;

osThreadId BLINK\_2Handle;

osThreadId Servo\_ControlHandle;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM2\_Init(void);

void GREEN\_LED(void const \* argument);

void WHITE\_LED(void const \* argument);

void Servo(void const \* argument);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_TIM2\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_TIM\_PWM\_Start(&htim2,TIM\_CHANNEL\_2);

/\* USER CODE END 2 \*/

/\* USER CODE BEGIN RTOS\_MUTEX \*/

/\* add mutexes, ... \*/

/\* USER CODE END RTOS\_MUTEX \*/

/\* USER CODE BEGIN RTOS\_SEMAPHORES \*/

/\* add semaphores, ... \*/

/\* USER CODE END RTOS\_SEMAPHORES \*/

/\* USER CODE BEGIN RTOS\_TIMERS \*/

/\* start timers, add new ones, ... \*/

/\* USER CODE END RTOS\_TIMERS \*/

/\* USER CODE BEGIN RTOS\_QUEUES \*/

/\* add queues, ... \*/

/\* USER CODE END RTOS\_QUEUES \*/

/\* Create the thread(s) \*/

/\* definition and creation of Blink1 \*/

osThreadDef(Blink1, GREEN\_LED, osPriorityNormal, 0, 128);

Blink1Handle = osThreadCreate(osThread(Blink1), NULL);

/\* definition and creation of BLINK\_2 \*/

osThreadDef(BLINK\_2, WHITE\_LED, osPriorityNormal, 0, 128);

BLINK\_2Handle = osThreadCreate(osThread(BLINK\_2), NULL);

/\* definition and creation of Servo\_Control \*/

osThreadDef(Servo\_Control, Servo, osPriorityNormal, 0, 128);

Servo\_ControlHandle = osThreadCreate(osThread(Servo\_Control), NULL);

/\* USER CODE BEGIN RTOS\_THREADS \*/

/\* add threads, ... \*/

/\* USER CODE END RTOS\_THREADS \*/

/\* Start scheduler \*/

osKernelStart();

/\* We should never get here as control is now taken by the scheduler \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL2;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = 31;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 9999;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim2);

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_13, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_2, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PC13 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_13;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\*Configure GPIO pin : PA2 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_2;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\* USER CODE BEGIN Header\_GREEN\_LED \*/

/\*\*

\* @brief Function implementing the Blink1 thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_GREEN\_LED \*/

void GREEN\_LED(void const \* argument)

{

/\* USER CODE BEGIN 5 \*/

/\* Infinite loop \*/

for(;;)

{

HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_13);

osDelay(500);

}

/\* USER CODE END 5 \*/

}

/\* USER CODE BEGIN Header\_WHITE\_LED \*/

/\*\*

\* @brief Function implementing the BLINK\_2 thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_WHITE\_LED \*/

void WHITE\_LED(void const \* argument)

{

/\* USER CODE BEGIN WHITE\_LED \*/

/\* Infinite loop \*/

for(;;)

{

HAL\_GPIO\_TogglePin(GPIOA,GPIO\_PIN\_2);

osDelay(200);

}

/\* USER CODE END WHITE\_LED \*/

}

/\* USER CODE BEGIN Header\_Servo \*/

/\*\*

\* @brief Function implementing the Servo\_Control thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_Servo \*/

void Servo(void const \* argument)

{

/\* USER CODE BEGIN Servo \*/

/\* Infinite loop \*/

for(;;)

{

for(int i=1000;i<2500;i=i+10)

{

\_\_HAL\_TIM\_SET\_COMPARE(&htim2,TIM\_CHANNEL\_2,i);

osDelay(10);

}

osDelay(1);

for(int i=2500;i>1000;i=i-10)

{

\_\_HAL\_TIM\_SET\_COMPARE(&htim2,TIM\_CHANNEL\_2,i);

osDelay(10);

}

}

/\* USER CODE END Servo \*/

}

/\*\*

\* @brief Period elapsed callback in non blocking mode

\* @note This function is called when TIM3 interrupt took place, inside

\* HAL\_TIM\_IRQHandler(). It makes a direct call to HAL\_IncTick() to increment

\* a global variable "uwTick" used as application time base.

\* @param htim : TIM handle

\* @retval None

\*/

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

/\* USER CODE BEGIN Callback 0 \*/

/\* USER CODE END Callback 0 \*/

if (htim->Instance == TIM3) {

HAL\_IncTick();

}

/\* USER CODE BEGIN Callback 1 \*/

/\* USER CODE END Callback 1 \*/

}

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/