```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2019 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

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

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "unicon.h"
#include "ds18b20.h"
#include "motor.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define CH_POT_RV1       LL_ADC_CHANNEL_0
#define CH_POT_RV2       LL_ADC_CHANNEL_1
#define CH_HUMIDITY      LL_ADC_CHANNEL_5
#define CH_TEMPERATURE   LL_ADC_CHANNEL_4
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
const char version[] = "Ver. 1.0-20190423";
SysData_TypeDef SysData;

uint8_t ds_status;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM16_Init(void);
/* USER CODE BEGIN PFP */

static void LedHandler(void);
static void NewMessageHandler(void);
static void SendOk(void);
static void TestModeHandler(SysData_TypeDef *self);
/* 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 */
    static volatile uint32_t delay = 0;
    static uint8_t ds_delay = 0;

    /* USER CODE END 1 */

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

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


    LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_SYSCFG);
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

    /* System interrupt init*/

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

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

    /* USER CODE BEGIN SysInit */
    SysTick_Config(SystemCoreClock/1000);
    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_ADC_Init();
    MX_USART1_UART_Init();
    MX_TIM16_Init();
    /* USER CODE BEGIN 2 */

    LL_mDelay(10);

    SysData.TestMode = 0;
    SysData.HighTemperature = 25;
    SysData.LowTemperature = 22;
    SysData.HighHumidity = HUM_LEVEL_DEF;
    SysData.MotorSpeed = MOTOR_SPEED_DEF;
    SysData.MotorRunTime = MOTOR_DELAY_TIME_DEF;
    SysData.MotorDelayTime = MOTOR_PAUSE_TIME_DEF;

    UNI_Start();

    DS18B20_PortInit();
    ds_status = DS18B20_Init(SKIP_ROM);

    DS18B20_MeasureTemperCmd(SKIP_ROM, 0);
    Delay_ms(100);


    L298_Init(&SysData);


    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */

    while (1) {

        if(delay <= timestamp) {

            delay = timestamp + 100;

            ADC_Read_VREFINT();

            SysData.ADC_Data.ch0 = ADC_ReadAnalog(CH_POT_RV1);
            Delay_ms(10);

            SysData.ADC_Data.ch1 = ADC_ReadAnalog(CH_POT_RV2);
            Delay_ms(10);

            SysData.ADC_Data.ch2 = ADC_ReadAnalog(CH_HUMIDITY);
            Delay_ms(10);


            if(ds_delay == 0) {

                ds_delay = 10;

                DS18B20_ReadStratchpad(SKIP_ROM, 0);
                SysData.temper = DS18B20_Convert(0);

                Delay_ms(10);
                DS18B20_MeasureTemperCmd(SKIP_ROM, 0);
            } else {

                ds_delay--;
            }


            if(SysData.TestMode == 0) {

                UNI_Process();

                if(SysData.ADC_Data.ch2 >= SysData.HighHumidity) {
                    L298_CloseWindow(&SysData);
                    LED6_ON();
                } else {

                    LED6_OFF();

                    if(SysData.temper >= SysData.HighTemperature) {
                        L298_OpenWindow(&SysData);
                        LED7_ON();
                    } else {

                        if(SysData.temper < SysData.LowTemperature) {
                            L298_CloseWindow(&SysData);
                        }

                        L298_Process(&SysData);

                        LED7_OFF();
                    }
                }
            } else {
                /* testas */

                L298_Process(&SysData);

                TestModeHandler(&SysData);
            }
        }


        LedHandler();

        if(NewMessageFlag) {

            NewMessageHandler();

            NewMessageFlag = false;
        }

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */


    }
    /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void) {
    LL_FLASH_SetLatency(LL_FLASH_LATENCY_1);

    if(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_1) {
        Error_Handler();
    }
    LL_RCC_HSI_Enable();

    /* Wait till HSI is ready */
    while(LL_RCC_HSI_IsReady() != 1) {

    }
    LL_RCC_HSI_SetCalibTrimming(16);
    LL_RCC_HSI14_Enable();

    /* Wait till HSI14 is ready */
    while(LL_RCC_HSI14_IsReady() != 1) {

    }
    LL_RCC_HSI14_SetCalibTrimming(16);
    LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSI_DIV_2, LL_RCC_PLL_MUL_12);
    LL_RCC_PLL_Enable();

    /* Wait till PLL is ready */
    while(LL_RCC_PLL_IsReady() != 1) {

    }
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);

    /* Wait till System clock is ready */
    while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL) {

    }
    LL_Init1msTick(48000000);
    LL_SYSTICK_SetClkSource(LL_SYSTICK_CLKSOURCE_HCLK);
    LL_SetSystemCoreClock(48000000);
    LL_RCC_HSI14_EnableADCControl();
    LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_HSI);
}

/**
  * @brief ADC Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC_Init(void) {

    /* USER CODE BEGIN ADC_Init 0 */

    /* USER CODE END ADC_Init 0 */

    LL_ADC_InitTypeDef ADC_InitStruct = {0};
    LL_ADC_REG_InitTypeDef ADC_REG_InitStruct = {0};

    LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Peripheral clock enable */
    LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_ADC1);

    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    /**ADC GPIO Configuration
    PA0   ------> ADC_IN0
    PA1   ------> ADC_IN1
    PA2   ------> ADC_IN2
    PA5   ------> ADC_IN5
    */
    GPIO_InitStruct.Pin = AD1_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(AD1_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = AD2_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(AD2_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = ADC_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(ADC_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = HUMIDITY_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(HUMIDITY_GPIO_Port, &GPIO_InitStruct);

    /* USER CODE BEGIN ADC_Init 1 */

    /* USER CODE END ADC_Init 1 */
    /** Configure Regular Channel
    */
    LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_0);
    /** Configure Regular Channel
    */
    LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_1);
    /** Configure Regular Channel
    */
    LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_2);
    /** Configure Regular Channel
    */
    LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_5);
    /** Configure Regular Channel
    */
    LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_PATH_INTERNAL_VREFINT);
    /** Configure Internal Channel
    */
    LL_ADC_SetCommonPathInternalCh(__LL_ADC_COMMON_INSTANCE(ADC1),
LL_ADC_CHANNEL_VREFINT);
    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment
and number of conversion)
    */
    ADC_InitStruct.Clock = LL_ADC_CLOCK_ASYNC;
    ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B;
    ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
    ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
    LL_ADC_Init(ADC1, &ADC_InitStruct);
    ADC_REG_InitStruct.TriggerSource = LL_ADC_REG_TRIG_SOFTWARE;
    ADC_REG_InitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_DISABLE;
    ADC_REG_InitStruct.ContinuousMode = LL_ADC_REG_CONV_SINGLE;
    ADC_REG_InitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_LIMITED;
    ADC_REG_InitStruct.Overrun = LL_ADC_REG_OVR_DATA_PRESERVED;
    LL_ADC_REG_Init(ADC1, &ADC_REG_InitStruct);
    LL_ADC_REG_SetSequencerScanDirection(ADC1, LL_ADC_REG_SEQ_SCAN_DIR_FORWARD);
    LL_ADC_SetSamplingTimeCommonChannels(ADC1, LL_ADC_SAMPLINGTIME_1CYCLE_5);
    LL_ADC_DisableIT_EOC(ADC1);
    LL_ADC_DisableIT_EOS(ADC1);
    /* USER CODE BEGIN ADC_Init 2 */

    /* USER CODE END ADC_Init 2 */

}

/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void) {

    /* USER CODE BEGIN TIM16_Init 0 */

    /* USER CODE END TIM16_Init 0 */

    LL_TIM_InitTypeDef TIM_InitStruct = {0};
    LL_TIM_OC_InitTypeDef TIM_OC_InitStruct = {0};
    LL_TIM_BDTR_InitTypeDef TIM_BDTRInitStruct = {0};

    LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Peripheral clock enable */
    LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_TIM16);

    /* USER CODE BEGIN TIM16_Init 1 */

    /* USER CODE END TIM16_Init 1 */
    TIM_InitStruct.Prescaler = 0;
    TIM_InitStruct.CounterMode = LL_TIM_COUNTERMODE_UP;
    TIM_InitStruct.Autoreload = 1000;
    TIM_InitStruct.ClockDivision = LL_TIM_CLOCKDIVISION_DIV1;
    TIM_InitStruct.RepetitionCounter = 0;
    LL_TIM_Init(TIM16, &TIM_InitStruct);
    LL_TIM_DisableARRPreload(TIM16);
    LL_TIM_OC_EnablePreload(TIM16, LL_TIM_CHANNEL_CH1);
    TIM_OC_InitStruct.OCMode = LL_TIM_OCMODE_PWM1;
    TIM_OC_InitStruct.OCState = LL_TIM_OCSTATE_DISABLE;
    TIM_OC_InitStruct.OCNState = LL_TIM_OCSTATE_DISABLE;
    TIM_OC_InitStruct.CompareValue = 500;
    TIM_OC_InitStruct.OCPolarity = LL_TIM_OCPOLARITY_HIGH;
    TIM_OC_InitStruct.OCNPolarity = LL_TIM_OCPOLARITY_HIGH;
    TIM_OC_InitStruct.OCIdleState = LL_TIM_OCIDLESTATE_HIGH;
    TIM_OC_InitStruct.OCNIdleState = LL_TIM_OCIDLESTATE_LOW;
    LL_TIM_OC_Init(TIM16, LL_TIM_CHANNEL_CH1, &TIM_OC_InitStruct);
    LL_TIM_OC_DisableFast(TIM16, LL_TIM_CHANNEL_CH1);
    TIM_BDTRInitStruct.OSSRState = LL_TIM_OSSR_DISABLE;
    TIM_BDTRInitStruct.OSSIState = LL_TIM_OSSI_DISABLE;
    TIM_BDTRInitStruct.LockLevel = LL_TIM_LOCKLEVEL_OFF;
    TIM_BDTRInitStruct.DeadTime = 0;
    TIM_BDTRInitStruct.BreakState = LL_TIM_BREAK_DISABLE;
    TIM_BDTRInitStruct.BreakPolarity = LL_TIM_BREAK_POLARITY_HIGH;
    TIM_BDTRInitStruct.AutomaticOutput = LL_TIM_AUTOMATICOUTPUT_DISABLE;
    LL_TIM_BDTR_Init(TIM16, &TIM_BDTRInitStruct);
    /* USER CODE BEGIN TIM16_Init 2 */

    /* USER CODE END TIM16_Init 2 */
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
    /**TIM16 GPIO Configuration
    PB8   ------> TIM16_CH1
    */
    GPIO_InitStruct.Pin = L298_IN2_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
    LL_GPIO_Init(L298_IN2_GPIO_Port, &GPIO_InitStruct);

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void) {

    /* USER CODE BEGIN USART1_Init 0 */

    /* USER CODE END USART1_Init 0 */

    LL_USART_InitTypeDef USART_InitStruct = {0};

    LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Peripheral clock enable */
    LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_USART1);

    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    /**USART1 GPIO Configuration
    PA9   ------> USART1_TX
    PA10   ------> USART1_RX
    */
    GPIO_InitStruct.Pin = LL_GPIO_PIN_9;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_1;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_1;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* USART1 interrupt Init */
    NVIC_SetPriority(USART1_IRQn, 0);
    NVIC_EnableIRQ(USART1_IRQn);

    /* USER CODE BEGIN USART1_Init 1 */

    /* USER CODE END USART1_Init 1 */
    USART_InitStruct.BaudRate = 38400;
    USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
    USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
    USART_InitStruct.Parity = LL_USART_PARITY_NONE;
    USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
    USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
    USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
    LL_USART_Init(USART1, &USART_InitStruct);
    LL_USART_DisableIT_CTS(USART1);
    LL_USART_DisableOverrunDetect(USART1);
    LL_USART_ConfigAsyncMode(USART1);
    LL_USART_Enable(USART1);
    /* USER CODE BEGIN USART1_Init 2 */

    /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_GPIO_Init(void) {
    LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
    LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);

    /**/
    LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);

    /**/
    LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);

    /**/
    LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);

    /**/
    LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);

    /**/
    LL_GPIO_ResetOutputPin(GPIOB, LL_GPIO_PIN_12);

    /**/
    LL_GPIO_ResetOutputPin(L298_IN1_GPIO_Port, L298_IN1_Pin);

    /**/
    LL_GPIO_ResetOutputPin(L298_ENA_GPIO_Port, L298_ENA_Pin);

    /**/
    GPIO_InitStruct.Pin = LED5_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = LED2_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = LED7_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = LED6_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = LL_GPIO_PIN_11;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = LL_GPIO_PIN_12;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_MEDIUM;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = L298_IN1_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(L298_IN1_GPIO_Port, &GPIO_InitStruct);

    /**/
    GPIO_InitStruct.Pin = L298_ENA_Pin;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    LL_GPIO_Init(L298_ENA_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

static void LedHandler(void) {

    static uint32_t delay = 0;
    static uint8_t led = 0;

    if(delay < timestamp) {

        delay = timestamp + 100;

        if(SysData.TestMode) {

            LED2_OFF();
            LED5_OFF();
            LED6_OFF();
            LED7_OFF();

            switch(led) {
            case 0:
                LED2_ON();
                led = 1;
                break;
            case 1:
                LED5_ON();
                led = 2;
                break;
            case 2:
                LED6_ON();
                led = 3;
                break;
            case 3:
                LED7_ON();
                led = 0;
                break;
            }

        } else {

            if(SysData.WindowState) LED2_ON();
```

```c
            else LED2_OFF();

            if(LL_GPIO_IsInputPinSet(GPIOB, LL_GPIO_PIN_9) ) LED5_ON();
            else LED5_OFF();
        }

    }
}


/*  */
static void NewMessageHandler(void) {

    if( !strncmp(ptrPrimaryRxBuffer, "AT+HTEMP=", 9 )) {

        uint8_t tmp = atoi(ptrPrimaryRxBuffer+9);

        if(tmp < 22 || tmp > 32) {

            sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: HTEMP=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.HighTemperature = tmp;
        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+HTEMP?", 9 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "HTEMP=", SysData.HighTemperature, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);
        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+LTEMP=", 9 )) {

        uint8_t tmp = atoi(ptrPrimaryRxBuffer+9);

        if(tmp < 15 || tmp > 28) {

            sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: LTEMP=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.LowTemperature = tmp;
        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+LTEMP?", 9 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "LTEMP=", SysData.LowTemperature, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);
        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+HUMLEVEL=", 12 )) {

        uint16_t tmp = atoi(ptrPrimaryRxBuffer+12);

        if(tmp < 200 || tmp > 800) {

            sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: HUMLEVEL=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.HighHumidity = tmp;
        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+HUMLEVEL?", 12 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "HUMLEVEL=", SysData.HighHumidity, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);
        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+SPEED=", 9 )) {

        uint16_t tmp = atoi(ptrPrimaryRxBuffer+9);

        if(tmp < 300 || tmp > 800) {

            sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: SPEED=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.MotorSpeed = tmp;

        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();

```

```c
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+SPEED?", 9 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%u%s", "SPEED=", SysData.MotorSpeed, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);

        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+RUNTIME=", 11 )) {

        uint8_t tmp = atoi(ptrPrimaryRxBuffer+11);

        if(tmp < 2 || tmp > 10) {

            sprintf(ptrPrimaryTxBuffer, "%s%u%s", "BAD PARAM: RUNTIME=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.MotorRunTime = tmp;
        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+RUNTIME?", 11 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%u%s", "RUNTIME=", SysData.MotorRunTime, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);
        return;
    }



    if( !strncmp(ptrPrimaryRxBuffer, "AT+DELAYTIME=", 13 )) {

        uint8_t tmp = atoi(ptrPrimaryRxBuffer+13);

        if(tmp < 1 || tmp > 3) {

            sprintf(ptrPrimaryTxBuffer, "%s%u%s", "BAD PARAM: DELAYTIME=", tmp, "\r\n");
            USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
            while(RespondWaitingFlag);

            USART_ClearRxBuffer(PRIMARY_PORT);
            return;
        }

        SysData.MotorDelayTime = tmp;
        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }

    if( !strncmp(ptrPrimaryRxBuffer, "AT+DELAYTIME?", 13 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%u%s", "DELAYTIME=", SysData.MotorDelayTime, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);
        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+INF", 6 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%02.02f%s%u%s", "TEMP=", SysData.temper, " HUM=", SysData.ADC_Data.ch2, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);

        return;
    }


    if( !strncmp(ptrPrimaryRxBuffer, "AT+VER", 6 )) {

        sprintf(ptrPrimaryTxBuffer, "%s%s", version, "\r\n");
        USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
        while(RespondWaitingFlag);

        USART_ClearRxBuffer(PRIMARY_PORT);

        return;
    }



    if( !strncmp(ptrPrimaryRxBuffer, "AT+TEST=", 8 )) {

        SysData.TestMode = atoi(ptrPrimaryRxBuffer+8);

        if(SysData.TestMode) {
            /* jungiam Testini tezima */
            L298_CloseWindow(&SysData);
            SysData.WindowState = Closed;
        } else {



        }

        USART_ClearRxBuffer(PRIMARY_PORT);
        SendOk();
        return;
    }


    /* papildomos testinio rezimo komandos */
    if(SysData.TestMode) {

        if( !strncmp(ptrPrimaryRxBuffer, "AT+OPEN", 7 )) {

            if(SysData.WindowState != Opened){

                L298_OpenWindow(&SysData);
                SysData.WindowState = Opened;

                SendOk();
            }else{

                USART_SendString(PRIMARY_PORT, "Window is opened!\r\n");
                while(RespondWaitingFlag);
            }

            return;
        }

        if( !strncmp(ptrPrimaryRxBuffer, "AT+CLOSE", 8 )) {

            if(SysData.WindowState != Closed){

                L298_CloseWindow(&SysData);
                SysData.WindowState = Closed;

                SendOk();
            }else{

                USART_SendString(PRIMARY_PORT, "Window is closed!\r\n");
                while(RespondWaitingFlag);
            }

            return;
        }


    }


    USART_SendString(PRIMARY_PORT, "BAD_CMD\r\n");
    while(RespondWaitingFlag);

    USART_ClearRxBuffer(PRIMARY_PORT);
}


/*   */
static void SendOk(void) {

    USART_SendString(PRIMARY_PORT, "OK\r\n");
    while(RespondWaitingFlag);
}


/*   */
static void TestModeHandler(SysData_TypeDef *self) {





}


/* USER CODE END 4 */

/**
```

```c
  * @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 */

    /* 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(char *file, uint32_t line) {
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```