```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 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 "fatfs.h"
#include "ADT74x0.h" // https://github.com/axoulc/ADT74x0-STM32

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>

// Differential sensor ------------------------------------------------------
#define DIFF_WRITE 0b01010010
#define DIFF_READ 0b01010011
#define SINGLE_MES 0b10101010

// Temperature sensor-------------------------------------------------------
#define TEMP1_ADDR 0x4a
#define TEMP2_ADDR 0x4b

// -------------------------------------------------------------------------

CAN_HandleTypeDef hcan;

I2C_HandleTypeDef hi2c1; // meracia sustava 1
I2C_HandleTypeDef hi2c2; // meracia sustava 2

SPI_HandleTypeDef hspi2; // pre sd kartu

TIM_HandleTypeDef htim2; // timer

UART_HandleTypeDef huart1; // gps
UART_HandleTypeDef huart2; // putty

ADT74X0 temp1; // temp sensor 1
ADT74X0 temp2;
```

```
// GPS
// #define USE_GPS
#define GPS_DEBUG 0 // debug GPS
#define GPS_USART &huart1
#define GPSBUFSIZE 128 // GPS buffer size

uint8_t rx_data = 0;
uint8_t rx_buffer[GPSBUFSIZE];
uint8_t rx_index = 0;

typedef struct //struktura parsovania gps dat
{
  // calculated values
  float dec_longitude;
  float dec_latitude;
  float altitude_ft;

  // GGA - Global Positioning System Fixed Data
  float nmea_longitude;
  float nmea_latitude;
  float utc_time;
  char ns, ew;
  int lock;
  int satelites;
  float hdop;
  float msl_altitude;
  char msl_units;

  // RMC - Recommended Minimmum Specific GNS Data
  char rmc_status;
  float speed_k;
  float course_d;
  int date;

  // GLL
  char gll_status;

  // VTG - Course over ground, ground speed
  float course_t; // ground speed true
  char course_t_unit;
  float course_m; // magnetic
  char course_m_unit;
  char speed_k_unit;
  float speed_km; // speek km/hr
  char speed_km_unit;
} GPS_t;

GPS_t GPS;

// -------------------------------------------------------------------------------
// Globals

uint8_t inter = 0;
uint8_t diff_write = DIFF_WRITE;
```

```
uint8_t diff_read = DIFF_READ;
uint8_t single_mes = SINGLE_MES;

float H1 = 0.0;
float H2 = 0.0;
float v_vertical = 0; // vertikalna rychlost

float Hrel = 0; // relativna vyska

float T1 = 0; // pociatocna teplota v aktual ref. krabicke

float pressure = 0; // tlak v aktualnom mieste kde sa nachadza lietadlo
float Pzero = 1011; // tlak na urovni letiska

int sensor = 0;      // id diff. senzoru, 0 - prvy senzor, 1 - druhy senzor
float Pdiff = 0;     // tlak z diff. senzoru
float Pcorrected = 0; // tlak z diff. senzoru

const float DIFF_SENSOR_RANGE = 75.000;                 // max rozsah diff. senzoru v hPa
const float MAX_OPERATING_RANGE = 0.8 * DIFF_SENSOR_RANGE; // operacny rozsah, v 80% max rozsahu sa prepnu
senzory
float pressure_level = 0;                           // pocet rozsahov diff. senzoru od 0 vysky

FATFS FatFs;                // Fatfs handle
FIL logfile;                // File handle
FRESULT file_open = FR_NO_FILE;

// can
CAN_TxHeaderTypeDef TxHeader;
CAN_RxHeaderTypeDef RxHeader;

uint8_t TxData[8];
uint8_t RxData[8];

uint32_t TxMailbox;
int datacheck = 0;

// -----------------------------------------------------------------------------

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_CAN_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM2_Init(void);

void open_valve(int sensors, bool open);

// GPS prototypes ------------------------------------------------------------
void GPS_Init();
void GPS_print(char *data);
```

```
void GPS_UART_CallBack();
int GPS_validate(char *nmeastr);
float GPS_nmea_to_dec(float deg_coord, char nsew);
void GPS_parse(char *GPSstrParse);
void GPS_print_data();
void GPS_fix(); // wait until fix is estabilished
// GPS ------------------------------------------------------------------------

void temp_init();
float get_temperature(int sensor);

// vlastny printf na putti
void uprintf(const char *fmt, ...)
{
  static char buffer[256];
  va_list args;
  va_start(args, fmt);
  vsnprintf(buffer, sizeof(buffer), fmt, args);
  va_end(args);

  int len = strlen(buffer);
  HAL_UART_Transmit(&huart2, (uint8_t *)buffer, len, -1);
}

// void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
// {
//   HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &RxHeader, RxData);
//   if (RxHeader.DLC == 2)
//   {
//     datacheck = 1;
//   }
// }

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
  FRESULT fres; // Result after operations

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

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



  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init(); // putty uart
  MX_CAN_Init();
  MX_USART1_UART_Init(); // gps uart
  MX_I2C1_Init();
```

```
  MX_I2C2_Init();
  MX_SPI2_Init();
  MX_FATFS_Init();
  MX_TIM2_Init();
// init teplomeru, inicializovane, este pred spustenim timeru
  temp_init();
  // start can zbernice
  // HAL_CAN_Start(&hcan);
  // HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO1_MSG_PENDING);

  //-------------------------------------------
  // Open the file system
  //
  // fres = f_mount(&FatFs, "", 1); // 1=mount now
  // if (fres == FR_OK)
  // {
  //   fres = f_open(&logfile, "log.txt", FA_OPEN_ALWAYS | FA_WRITE);

  //   if (fres == FR_OK)
  //   {
  //     uprintf("Logovanie zahajene \r\n");
  //     file_open = FR_OK;

  //     const char buffer[] = "Starting log";
  //     unsigned int bytes_wrote = 0;

  //     fres = f_write(&logfile, buffer, strlen(buffer), &bytes_wrote);
  //     if (fres == FR_OK)
  //       uprintf("Zapsanie do suboru sa podarilo!");
  //   }
  //   else
  //     uprintf("f_open open error (%i)\r\n", fres);
  // }
  // else
  // {
  //   uprintf("f_mount error (%i)\r\n", fres);
  // }

  //-------------------------------------------
  // init GPS

#ifdef USE_GPS
  GPS_Init();
  // wait to connection
  GPS_fix();
#endif

  //-------------------------------------------
  // init ventilu a spustenie timeru

  // open_valve(0, false);
  // open_valve(1, false);

  inter = 0;
```

```
 // spustime timer2 a zacneme meranie
 HAL_TIM_Base_Start_IT(&htim2);

 //-------------------------------------------
 while (1)
 {
  //
 }

 // zavrieme subor a unmoutneme sd kartu
 // f_close(&logfile);
 // f_mount(NULL, "", 0);
}

void temp_init()
{
 temp1.adti2c = &hi2c1; // Your I2C Handler

 ADT74x0_Init(&temp1, TEMP1_ADDR);           // I2C address depends of A0 and A1 pins
 ADT74x0_Reset(&temp1);                      // Optional
 ADT74x0_SetResolution(&temp1, ADT74X0_16BITS); // Put the device in 16 bits resolution

 temp2.adti2c = &hi2c1; // Your I2C Handler

 ADT74x0_Init(&temp2, TEMP2_ADDR);           // I2C address depends of A0 and A1 pins
 ADT74x0_Reset(&temp2);                      // Optional
 ADT74x0_SetResolution(&temp2, ADT74X0_16BITS); // Put the device in 16 bits resolution
}

void open_valve(int sensors, bool open)
{

 if (sensors == 0)
 {
  if (open)
  {
   // Otvorenie ventilu 1
   HAL_GPIO_WritePin(VENTIL_1_GPIO_Port, VENTIL_1_Pin, GPIO_PIN_SET);
  }
  else
  {
   // zavrenie ventilu 1
   HAL_GPIO_WritePin(GPIOA, VENTIL_1_Pin, GPIO_PIN_RESET);
  }
 }
 else
 {
  if (open)
  {
   // Otvorenie ventilu 2
   HAL_GPIO_WritePin(VENTIL_2_GPIO_Port, VENTIL_2_Pin, GPIO_PIN_SET);
  }
  else
  {
   // zavrenie ventilu 2
```

```
    HAL_GPIO_WritePin(GPIOB, VENTIL_2_Pin, GPIO_PIN_RESET);
   }
  }
}

float get_pressure(int sensor) //vracia tlak z aktualneho senzoru
{

 I2C_HandleTypeDef *hi2c = NULL;
 HAL_StatusTypeDef ret;

 uint8_t buffer[7] = {0}; // buffer pre prijem tlaku
 uint32_t Praw = 0;      //
 int32_t iPraw = 0;
 double diff_pressure = 0;

 // podla aktivneho senzoru
 if (sensor == 0)
  hi2c = &hi2c2;
 else
  hi2c = &hi2c1;

 // kontinuálne vypisovanie tlaku
 // ret = HAL_I2C_Master_Transmit(hi2c, diff_write, (uint8_t *)&single_mes, 1, 10);
 ret = HAL_I2C_Master_Transmit(hi2c, diff_write, (uint8_t *)&single_mes, 1, 10);
 if (ret != HAL_OK)
 {
  // strcpy((char *)msg_buffer, "Error Tx\r\n");
  // HAL_UART_Transmit(&huart2, msg_buffer, strlen((char *)msg_buffer), HAL_MAX_DELAY);
  uprintf("Differential sensor %d: Error Tx\r\n", sensor);
  Error_Handler();
 }
 else
 {
  //  HAL_Delay(6);

  ret = HAL_I2C_Master_Receive(hi2c, diff_read, buffer, 7, 10);
  if (ret != HAL_OK)
  {
   uprintf("Differential sensor %d: Error Rx\r\n", sensor);
   Error_Handler();
  }
  else
  {
   Praw = (buffer[1] << 16 | buffer[2] << 8 | buffer[3]);
   iPraw = Praw - 8388608;
   diff_pressure = (((float)iPraw / (float)16777216) * 1.25 * 60) * 2.49;

   // Process the pressure data as needed
   // Print the pressure value
   // gcvt(diff_tlak, 8, str_buffer);
   // int pos = 0;
   // pos += sprintf(&msg_buffer[pos], "Pressure: %s hPa\r\n", str_buffer);
   // HAL_UART_Transmit(&huart2, msg_buffer, strlen((char *)msg_buffer), HAL_MAX_DELAY);
  }
```

```
  }

  return diff_pressure;
}

float get_vertical_velocity()
{
  float vertical_speed = 0;

  H2 = Hrel;
  vertical_speed = H2 - H1;

  H1 = H2; // nastavime H2 jako novou H1

  return vertical_speed;
}

float get_temperature(int sensor)
{
  float temperature = 0;

  if (sensor == 0)
  {
    if (ADT74x0_ReadTemp(&temp1) != HAL_OK)
    {
      Error_Handler();
    }
    temperature = temp1.deg_data;
  }
  else
  {
    if (ADT74x0_ReadTemp(&temp2) != HAL_OK)
    {
      Error_Handler();
    }
    temperature = temp2.deg_data;
  }

  return temperature;
}

float pressure_correction(float temp, float diff_pressure)
{
  // temp - aktualna teplota
  // T1 - teplota kedy sa uzavree ventil na ref. nadrzi
  return diff_pressure - diff_pressure * ((temp - T1) / T1);
}

void log_data(float Hrel, float v_vertical)
{
  FRESULT fres; // Result after operations

  if (file_open == FR_OK)
  {
    char buffer[50];
```

```
   unsigned int bytes_wrote;

   sprintf((char *)buffer, "Hrel: %.1f, Vertical speed: %.1f \r\n", Hrel, v_vertical);
   fres = f_write(&logfile, buffer, strlen((char *)buffer), &bytes_wrote);
   if (fres != FR_OK)
   {
     uprintf("Cannot write to logfile \r\n");
   }
  }
}

// void transmit_can()
// {
//   //

//   TxHeader.DLC = 2; // data length
//   TxHeader.IDE = CAN_ID_STD;
//   TxHeader.RTR = CAN_RTR_DATA;
//   TxHeader.StdId = 0x103; // ID

//   TxData[0] = 200; // ms delay
//   TxData[1] = 20;  // loop rep

//   if (datacheck)
//   {
//     // blink the LED
//     for (int i = 0; i < RxData[1]; i++)
//     {
//       HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
//       HAL_Delay(RxData[0]);
//     }

//     datacheck = 0;

//     HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
//   }
// }

float meters_to_feet(float Hrel)
{
  return Hrel / 0.3048;
}

// -------------------------------------------------------------------------------
// Callbacks
// -------------------------------------------------------------------------------

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  float temp = 0; // aktualna teplota vo vybranej ref. nadrzi

  if (inter == 0)
  {
    open_valve(sensor, true);
    open_valve(1 - sensor, false);
```

```
  inter += 1;
 }
 else if (inter == 1)
 {
  open_valve(1 - sensor, true);
  open_valve(sensor, false);
  inter += 1;

  // zistime pociatecnu teplotu v aktualnej ref. nadrzi, pre vypocet korekcie
  T1 = get_temperature(sensor);
 }

 // merenie tlaku z diff. senzoru
 Pdiff = get_pressure(sensor);

 // aktualna teplota vo vybranej ref. nadrzi
 temp = get_temperature(sensor);
 // uprintf("Temp: %f \r\n", temp);

 //  korekcia tlaku na teplote
 Pcorrected = pressure_correction(temp, Pdiff);

 // vypocitame tlak v aktualnom mieste kde se nachadza lietadlo
 pressure = Pzero + (Pcorrected + pressure_level);
 // uprintf("Pressure: %f \r\n", pressure);

 // vypocet Hrel
 Hrel = 44330.769 * (1 - pow((pressure / Pzero), 0.19026));

 // vypocet a nastavenie vertical speed
 v_vertical = get_vertical_velocity();

 // zapis do suboru + odoslanie dat cez CAN
 //  log_data(Hrel, v_vertical);

 // can_transsmit();

#ifdef USE_GPS
  uprintf("Hrel: %.1f m (%.1f feet), Vertical speed: %.1f, pressure: %.1f, Pdiff: %.3f \r\n", Hrel, meters_to_feet(Hrel),
v_vertical, pressure, Pcorrected);
#else
  uprintf("Hrel: %.1f m, Vertical speed: %.1f, pressure: %.1f, Pdiff: %.3f \r\n", Hrel, v_vertical, pressure, Pcorrected);
#endif

 // ked dojde k prekroceniu rozsahu
 if (abs(Pdiff) >= MAX_OPERATING_RANGE)
 {

  pressure_level = pressure_level + Pdiff;

  // nastavenie ze v dalsom interupeu casovaca dojde k prehodeniu ventilu a diff
  inter = 0;

  Pdiff = 0;
```

```
   // prepneme diff. senzor na druhy
   sensor = 1 - sensor;
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart == &huart1)
  {
    GPS_UART_CallBack();
    // GPS_print_data();
  }
}

// GPS -----------------------------------
void GPS_print(char *data)
{
  char buf[GPSBUFSIZE] = {
     0,
  };
  // sprintf(buf, "%s\n", data);
  // CDC_Transmit_FS((unsigned char *)buf, (uint16_t)strlen(buf));
  // HAL_UART_Transmit(&huart2, (uint8_t *)buf, strlen(buf), 1000);

  uprintf("%s \r\n", data);
}

void GPS_Init()
{
  HAL_UART_Receive_IT(GPS_USART, &rx_data, 1);
}

void GPS_UART_CallBack()
{
  if (rx_data != '\n' && rx_index < sizeof(rx_buffer))
  {
    rx_buffer[rx_index++] = rx_data;
  }
  else
  {

#if (GPS_DEBUG == 1)
   GPS_print((char *)rx_buffer);
#endif

   if (GPS_validate((char *)rx_buffer))
     GPS_parse((char *)rx_buffer);
   rx_index = 0;
   memset(rx_buffer, 0, sizeof(rx_buffer));
  }

  HAL_UART_Receive_IT(GPS_USART, &rx_data, 1);
}

int GPS_validate(char *nmeastr)
```

```
{
  char check[3];
  char checkcalcstr[3];
  int i;
  int calculated_check;

  i = 0;
  calculated_check = 0;

  // check to ensure that the string starts with a $
  if (nmeastr[i] == '$')
    i++;
  else
    return 0;

  // No NULL reached, 75 char largest possible NMEA message, no '*' reached
  while ((nmeastr[i] != 0) && (nmeastr[i] != '*') && (i < 75))
  {
    calculated_check ^= nmeastr[i]; // calculate the checksum
    i++;
  }

  if (i >= 75)
  {
    return 0; // the string was too long so return an error
  }

  if (nmeastr[i] == '*')
  {
    check[0] = nmeastr[i + 1]; // put hex chars in check string
    check[1] = nmeastr[i + 2];
    check[2] = 0;
  }
  else
    return 0; // no checksum separator found there for invalid

  sprintf(checkcalcstr, "%02X", calculated_check);
  return ((checkcalcstr[0] == check[0]) && (checkcalcstr[1] == check[1])) ? 1 : 0;
}

float GPS_nmea_to_dec(float deg_coord, char nsew)
{
  int degree = (int)(deg_coord / 100);
  float minutes = deg_coord - degree * 100;
  float dec_deg = minutes / 60;
  float decimal = (float)degree + dec_deg;
  if (nsew == 'S' || nsew == 'W')
  { // return negative
    decimal *= -1;
  }
  return decimal;
}

void GPS_parse(char *GPSstrParse)
{
```

```
 // GGA https://receiverhelp.trimble.com/alloy-gnss/en-us/NMEA-0183messages_GGA.html
 // RMC https://receiverhelp.trimble.com/alloy-gnss/en-us/NMEA-0183messages_RMC.html
 // GLL https://receiverhelp.trimble.com/alloy-gnss/en-us/NMEA-0183messages_GLL.html
 // VTG https://receiverhelp.trimble.com/alloy-gnss/en-us/NMEA-0183messages_VTG.html
 //
 if (!strncmp(GPSstrParse, "$GPGGA", 6))
 {
   if (sscanf(GPSstrParse, "$GPGGA,%f,%f,%c,%f,%c,%d,%d,%f,%f,%c", &GPS.utc_time, &GPS.nmea_latitude, &GPS.ns,
&GPS.nmea_longitude, &GPS.ew, &GPS.lock, &GPS.satelites, &GPS.hdop, &GPS.msl_altitude, &GPS.msl_units) >= 1)
   {
     GPS.dec_latitude = GPS_nmea_to_dec((float)GPS.nmea_latitude, GPS.ns);
     GPS.dec_longitude = GPS_nmea_to_dec((float)GPS.nmea_longitude, GPS.ew);
     return;
   }
 }
 else if (!strncmp(GPSstrParse, "$NGGA", 5))
 {
   if (sscanf(GPSstrParse, "$NGGA,%f,%f,%c,%f,%c,%d,%d,%f,%f,%c", &GPS.utc_time, &GPS.nmea_latitude, &GPS.ns,
&GPS.nmea_longitude, &GPS.ew, &GPS.lock, &GPS.satelites, &GPS.hdop, &GPS.msl_altitude, &GPS.msl_units) >= 1)
   {
     GPS.dec_latitude = GPS_nmea_to_dec((float)GPS.nmea_latitude, GPS.ns);
     GPS.dec_longitude = GPS_nmea_to_dec((float)GPS.nmea_longitude, GPS.ew);
     return;
   }
 }
 else if (!strncmp(GPSstrParse, "$GPRMC", 6))
 {
   if (sscanf(GPSstrParse, "$GPRMC,%f,%f,%c,%f,%c,%f,%f,%d", &GPS.utc_time, &GPS.nmea_latitude, &GPS.ns,
&GPS.nmea_longitude, &GPS.ew, &GPS.speed_k, &GPS.course_d, &GPS.date) >= 1)
     return;
 }
 else if (!strncmp(GPSstrParse, "$GNRMC", 6))
 {
   if (sscanf(GPSstrParse, "$GNRMC,%f,%f,%c,%f,%c,%f,%f,%d", &GPS.utc_time, &GPS.nmea_latitude, &GPS.ns,
&GPS.nmea_longitude, &GPS.ew, &GPS.speed_k, &GPS.course_d, &GPS.date) >= 1)
     return;
 }
 else if (!strncmp(GPSstrParse, "$GPGLL", 6))
 {
   if (sscanf(GPSstrParse, "$GPGLL,%f,%c,%f,%c,%f,%c", &GPS.nmea_latitude, &GPS.ns, &GPS.nmea_longitude, &GPS.ew,
&GPS.utc_time, &GPS.gll_status) >= 1)
     return;
 }
 else if (!strncmp(GPSstrParse, "$GPVTG", 6))
 {
   if (sscanf(GPSstrParse, "$GPVTG,%f,%c,%f,%c,%f,%c,%f,%c", &GPS.course_t, &GPS.course_t_unit, &GPS.course_m,
&GPS.course_m_unit, &GPS.speed_k, &GPS.speed_k_unit, &GPS.speed_km, &GPS.speed_km_unit) >= 1)
     return;
 }
}

void GPS_print_data()
{
 //----------------------
 // float dec_longitude;
```

```
 // float dec_latitude;
 // float altitude_ft;
 //
 // // GGA - Global Positioning System Fixed Data
 // float nmea_longitude;
 // float nmea_latitude;
 // float utc_time;
 // char ns, ew;
 // int lock;
 // int satelites;
 // float hdop;
 // float msl_altitude;
 // char msl_units;
 //
 // // RMC - Recommended Minimmum Specific GNS Data
 // char rmc_status;
 // float speed_k;
 // float course_d;
 // int date;
 //
 // // GLL
 // char gll_status;
 //
 // // VTG - Course over ground, ground speed
 // float course_t; // ground speed true
 // char course_t_unit;
 // float course_m; // magnetic
 // char course_m_unit;
 // char speed_k_unit;
 // float speed_km; // speek km/hr
 // char speed_km_unit;

 uprintf("%f, lon: %f, lat: %f, sat: %d \r\n", GPS.utc_time, GPS.dec_longitude, GPS.dec_latitude, GPS.satelites);
}

void GPS_fix()
{
 // GPS Quality indicator :
 // 0 : Fix not valid
 // 1 : GPS fix
 // 2 : Differential GPS fix(DGNSS), SBAS, OmniSTAR VBS, Beacon, RTX in GVBS mode
 // 3 : Not applicable
 // 4 : RTK Fixed, xFill
 // 5 : RTK Float, OmniSTAR XP / HP, Location RTK, RTX
 // 6 : INS Dead reckoning

 while (GPS.lock == 0)
 {
  // wait until fix is established
 }
}

// ----------------------------------------

/**
```

```
 * @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_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  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_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
 * @brief CAN Initialization Function
 * @param None
 * @retval None
 */
static void MX_CAN_Init(void)
{

  /* USER CODE BEGIN CAN_Init 0 */

  /* USER CODE END CAN_Init 0 */

  /* USER CODE BEGIN CAN_Init 1 */

  /* USER CODE END CAN_Init 1 */
  hcan.Instance = CAN1;
  hcan.Init.Prescaler = 16;
  hcan.Init.Mode = CAN_MODE_NORMAL;
```

```
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_1TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
  hcan.Init.AutoBusOff = DISABLE;
  hcan.Init.AutoWakeUp = DISABLE;
  hcan.Init.AutoRetransmission = DISABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN_Init 2 */

  /* USER CODE END CAN_Init 2 */
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
```

```
 */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.ClockSpeed = 100000;
  hi2c2.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */
}

/**
 * @brief SPI2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI2_Init(void)
{

  /* USER CODE BEGIN SPI2_Init 0 */

  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
```

```
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI2_Init 2 */

  /* USER CODE END SPI2_Init 2 */
}

/**
  * @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_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 = 7200;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 2000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  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_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */
```

```
  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);
}

/**
 * @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 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */
}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
```

```
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */
}

/**
 * @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();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin | VENTIL_1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, SD_CS_Pin | VENTIL_2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin VENTIL_1_PIN */
  GPIO_InitStruct.Pin = LD2_Pin | VENTIL_1_Pin;
  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);

  /*Configure GPIO pins : SD_CS_NSS_Pin VENTIL_2_PIN */
  GPIO_InitStruct.Pin = SD_CS_Pin | VENTIL_2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

  /* USER CODE BEGIN MX_GPIO_Init_2 */
  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @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 */
```