```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  * Author          : Bowen Zheng
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 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"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum {
    LED_OFF,
    LED_RED,
    LED_GREEN,
    LED_BLUE,
```

```
    LED_PURPLE
} LED_Color;

typedef enum {
    INLET,
    ZONE_1,
    ZONE_2,
    ZONE_3
} Zone;
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

  /* -- UART -- */
uint8_t byte;
uint8_t user_input_flag = 0;

  /* -- MOTOR PMW Option -- */
volatile uint8_t INLET_PWM = 0;
volatile uint8_t Zone_1_PWM = 0;
volatile uint8_t Zone_2_PWM = 0;
volatile uint8_t Zone_3_PWM = 0;

  /* -- CLOCK Option -- */
volatile uint8_t WALL_CLK_START = 00;
volatile uint8_t INLET_CLK_START = 00;
volatile uint8_t INLET_CLK_STOP = 00;
volatile uint8_t Zone_1_CLK_START = 00;
volatile uint8_t Zone_1_CLK_STOP = 00;
volatile uint8_t Zone_2_CLK_START = 00;
volatile uint8_t Zone_2_CLK_STOP = 00;
volatile uint8_t Zone_3_CLK_START = 00;
volatile uint8_t Zone_3_CLK_STOP = 00;

 /* -------- RUN MODE VAR -------- */
```

```
volatile uint8_t RUN_MODE_START_FLAG = 0;
volatile Zone Current_Zone;
const char* Zone_Names[] = {"INLET", "ZONE 1", "ZONE 2", "ZONE 3"};
volatile uint8_t System_Interlock_Flag = 0;

 /* -- DC Motor -- */
volatile uint64_t rpm_tick_count = 0;
volatile uint64_t last_rpm_tick_count = 0;
volatile uint32_t DC_Motor_RPM = 0;
volatile uint8_t Current_DC_Motor_Percent_PWM = 0;

 /* -- Wall Clock -- */
volatile uint64_t simulate_seconds = 0;
volatile uint8_t Current_Wall_CLK_Hour = 00;

 /* -- US100 Distance Sensor -- */
const uint8_t cmd_dist = 0x55; // US100-Trigger, command byte  = 0x55;
volatile uint8_t us100_Rx_flag = 0;
volatile uint16_t distance_mm = 0;
uint8_t us100_buffer[2] = {0};
uint8_t msg_buffer[64] = {0};
volatile uint8_t Current_Water_Percent_depth = 0;
const volatile uint16_t Tank_Level_Lo = 1000;
const volatile uint16_t Tank_Level_Hi = 100;
volatile uint8_t Tank_Level_Lo_Alarm = 0;
volatile uint8_t Tank_Level_Hi_Alarm = 0;



/* USER CODE END PD */

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

/* USER CODE END PM */
```

```c
/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim5;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart6;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM5_Init(void);
/* USER CODE BEGIN PFP */
void DIGITS_Display(uint8_t DIGIT_A, uint8_t DIGIT_B);
void ADC_Select_CH(int CH );
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void UART_Parse_MSG(char* option_msg, char *msg, volatile uint8_t *Zone, unsigned int input_char_num);
void UART_Send_MSG(char* msg);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
void Control_DC_Motor_PWM(uint8_t percent, uint8_t direction); // 0 - reverse; 1 - forward
void Control_Servo_Motor_PWM(uint8_t direction); // 0 - INLET; 1 - OUTLET_1; 2 - OUTLET_2; 3 - OUTLET_3
void Update_Wall_CLK_Display(volatile uint64_t simulate_seconds);
```

```c
uint8_t ADC_Manual_Control_Percent_PWM();
void Get_Water_Percent_depth();
void Set_LED_Color(LED_Color color);
void System_Interlock();
/* 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_USART2_UART_Init();
MX_TIM3_Init();
MX_TIM2_Init();
MX_USART1_UART_Init();
MX_USART6_UART_Init();
MX_ADC1_Init();
MX_TIM5_Init();
/* USER CODE BEGIN 2 */

  /* ------------------ SET UP MODE BEGIN ------------------ */
// Turn off Nucleo green LED
HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
UART_Send_MSG("\r SETUP MODE");
//parse PWM options
UART_Parse_MSG("\r\n\n PMW option: 0) Manual Control; 1) 60% PMW; 2) 80% PMW; 3) 99% PMW;",
                    "\r\n INLET MOTOR SPEED PWM (option 0-3): ", &INLET_PWM, 1);
UART_Parse_MSG("",
                      "\r\n ZONE 1 MOTOR SPEED PWM (option 0-3): ", &Zone_1_PWM, 1);
UART_Parse_MSG("",
                    "\r\n ZONE 2 MOTOR SPEED PWM (option 0-3): ", &Zone_2_PWM, 1);
UART_Parse_MSG("",
                    "\r\n ZONE 3 MOTOR SPEED PWM (option 0-3): ", &Zone_3_PWM, 1);
//parse CLOCK options
UART_Parse_MSG("\r\n\n CLOCK option: 00 - Midnight, 01 - 1:00am, ... , 12 - noon, 13 - 1:00pm",
                    "\r\n CURRENT WALL CLOCK START TIME(0-23): ", &WALL_CLK_START, 2);
UART_Parse_MSG("",
                    "\r\n\n INLET WALL CLOCK START TIME(0-23): ", &INLET_CLK_START, 2);
UART_Parse_MSG("",
                    "\r\n INLET WALL CLOCK STOP TIME(0-23): ", &INLET_CLK_STOP, 2);
UART_Parse_MSG("",
                    "\r\n\n ZONE 1 WALL CLOCK START TIME(0-23): ", &Zone_1_CLK_START, 2);
UART_Parse_MSG("",
                    "\r\n ZONE 1 WALL CLOCK STOP TIME(0-23): ", &Zone_1_CLK_STOP, 2);
```

```
UART_Parse_MSG("",
                    "\r\n\n ZONE 2 WALL CLOCK START TIME(0-23): ", &Zone_2_CLK_START, 2);
UART_Parse_MSG("",
                    "\r\n ZONE 2 WALL CLOCK STOP TIME(0-23): ", &Zone_2_CLK_STOP, 2);
UART_Parse_MSG("",
                    "\r\n\n ZONE 3 WALL CLOCK START TIME(0-23): ", &Zone_3_CLK_START, 2);
UART_Parse_MSG("",
                    "\r\n ZONE 3 WALL CLOCK STOP TIME(0-23): ", &Zone_3_CLK_STOP, 2);
UART_Send_MSG("\r\n\n SETUP MODE END");
// wait for run mode to start (blue PB)
while (RUN_MODE_START_FLAG == 0) {
  // Flash controller green LED
  HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  HAL_Delay(100);
};
  /* ------------------ SET UP MODE END ------------------ */

  /* ------------------ RUN MODE ------------------ */
  // Turn ON controller green LED
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
  UART_Send_MSG("\r\n\n RUN MODE");
  UART_Send_MSG("\r\n");

// start up TIMER 5 for one second interrupts
  // start up TIMER 3 for DC motor control
  // start up TIMER 2 for Servo motor control
HAL_TIM_Base_Start_IT(&htim5);
HAL_TIM_Base_Init(&htim3);
  HAL_TIM_Base_Start(&htim2);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_Base_Init(&htim3);
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
/* USER CODE END 2 */

/* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* -- INLET -- */
  Tank_Level_Lo_Alarm = 0;
  Tank_Level_Hi_Alarm = 0;
  while ( (Current_Wall_CLK_Hour >= INLET_CLK_START && Current_Wall_CLK_Hour <= INLET_CLK_STOP) || Tank_Level_Hi_Alarm == 0 ) {
      Current_Zone = INLET;
      Set_LED_Color(LED_PURPLE);
      Control_Servo_Motor_PWM(0);
      Get_Water_Percent_depth();

      // Tank Full
      if (Tank_Level_Hi_Alarm == 1) {
          //Turn off Motor
          Control_DC_Motor_PWM(0, 0);
          //Wait until the current sequence finish
          while (Current_Wall_CLK_Hour >= INLET_CLK_START && Current_Wall_CLK_Hour <= INLET_CLK_STOP ) {
              Current_DC_Motor_Percent_PWM = 0;
          }
          break;
      }

      if (INLET_PWM == 0) {
          Current_DC_Motor_Percent_PWM = ADC_Manual_Control_Percent_PWM();
          Control_DC_Motor_PWM( ADC_Manual_Control_Percent_PWM() , 0);
      } else if (INLET_PWM == 1) {
          Current_DC_Motor_Percent_PWM = 60;
          Control_DC_Motor_PWM( 60 , 0);
      } else if (INLET_PWM == 2) {
          Current_DC_Motor_Percent_PWM = 80;
          Control_DC_Motor_PWM( 80 , 0);
      } else if (INLET_PWM == 3) {
          Current_DC_Motor_Percent_PWM = 99;
          Control_DC_Motor_PWM( 99 , 0);
      }
```

```
}

// Turn off motor between Switching sequence
Control_DC_Motor_PWM(0, 0);
HAL_Delay(1000);

/* -- ZONE 1 -- */
Tank_Level_Lo_Alarm = 0;
Tank_Level_Hi_Alarm = 0;
while (Current_Wall_CLK_Hour >= Zone_1_CLK_START && Current_Wall_CLK_Hour <= Zone_1_CLK_STOP) {
    Current_Zone = ZONE_1;
    Set_LED_Color(LED_RED);
    Control_Servo_Motor_PWM(1);
    Get_Water_Percent_depth();

    //RESERVOIR IS EMPTY
    if (Tank_Level_Lo_Alarm == 1) {
        System_Interlock();
    }

    if (Zone_1_PWM == 0) {
        Current_DC_Motor_Percent_PWM = ADC_Manual_Control_Percent_PWM();
        Control_DC_Motor_PWM( ADC_Manual_Control_Percent_PWM() , 1);
    } else if (Zone_1_PWM == 1) {
        Current_DC_Motor_Percent_PWM = 60;
        Control_DC_Motor_PWM(60 , 1);
    } else if (Zone_1_PWM == 2) {
        Current_DC_Motor_Percent_PWM = 80;
        Control_DC_Motor_PWM(80 , 1);
    } else if (Zone_1_PWM == 3) {
        Current_DC_Motor_Percent_PWM = 99;
        Control_DC_Motor_PWM(99 , 1);
    }
}

// Turn off motor between Switching sequence
```

```
Control_DC_Motor_PWM(0, 0);
HAL_Delay(1000);

/* -- ZONE 2 -- */
Tank_Level_Lo_Alarm = 0;
Tank_Level_Hi_Alarm = 0;
while (Current_Wall_CLK_Hour >= Zone_2_CLK_START && Current_Wall_CLK_Hour <= Zone_2_CLK_STOP) {
    Current_Zone = ZONE_2;
    Set_LED_Color(LED_GREEN);
    Control_Servo_Motor_PWM(2);
    Get_Water_Percent_depth();

    //RESERVOIR IS EMPTY
    if (Tank_Level_Lo_Alarm == 1) {
        System_Interlock();
    }

    if (Zone_2_PWM == 0) {
        Current_DC_Motor_Percent_PWM = ADC_Manual_Control_Percent_PWM();
        Control_DC_Motor_PWM( ADC_Manual_Control_Percent_PWM() , 1);
    } else if (Zone_2_PWM == 1) {
        Current_DC_Motor_Percent_PWM = 60;
        Control_DC_Motor_PWM(60 , 1);
    } else if (Zone_2_PWM == 2) {
        Current_DC_Motor_Percent_PWM = 80;
        Control_DC_Motor_PWM(80 , 1);
    } else if (Zone_2_PWM == 3) {
        Current_DC_Motor_Percent_PWM = 99;
        Control_DC_Motor_PWM(99 , 1);
    }
}

// Turn off motor between Switching sequence
Control_DC_Motor_PWM(0, 0);
HAL_Delay(1000);
```

```
  /* -- ZONE 3 -- */
  Tank_Level_Lo_Alarm = 0;
  Tank_Level_Hi_Alarm = 0;
  while (Current_Wall_CLK_Hour >= Zone_3_CLK_START && Current_Wall_CLK_Hour <= Zone_3_CLK_STOP) {
      Current_Zone = ZONE_3;
      Set_LED_Color(LED_BLUE);
      Control_Servo_Motor_PWM(3);
      Get_Water_Percent_depth();

      //RESERVOIR IS EMPTY
      if (Tank_Level_Lo_Alarm == 1) {
          System_Interlock();
      }

      if (Zone_3_PWM == 0) {
          Current_DC_Motor_Percent_PWM = ADC_Manual_Control_Percent_PWM();
          Control_DC_Motor_PWM( ADC_Manual_Control_Percent_PWM() , 1);
      } else if (Zone_3_PWM == 1) {
          Current_DC_Motor_Percent_PWM = 60;
          Control_DC_Motor_PWM(60 , 1);
      } else if (Zone_3_PWM == 2) {
          Current_DC_Motor_Percent_PWM = 80;
          Control_DC_Motor_PWM(80 , 1);
      } else if (Zone_3_PWM == 3) {
          Current_DC_Motor_Percent_PWM = 99;
          Control_DC_Motor_PWM(99 , 1);
      }
  }

    /* ------------------ RUN MODE REPEAT ------------------ */

  /* USER CODE END WHILE */

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

```c
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

  /** 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_NONE;
  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_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```c
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_8B;
  hadc1.Init.ScanConvMode = ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```c
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_9;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_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_ClockConfigTypeDef sClockSourceConfig = {0};
```

```c
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 = 16-1;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 20000-1;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
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_1) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

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

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 16-1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 2000-1;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &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(&htim3, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}
```

```c
/**
  * @brief TIM5 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM5_Init(void)
{

  /* USER CODE BEGIN TIM5_Init 0 */

  /* USER CODE END TIM5_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM5_Init 1 */

  /* USER CODE END TIM5_Init 1 */
  htim5.Instance = TIM5;
  htim5.Init.Prescaler = 16000-1;
  htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim5.Init.Period = 1000-1;
  htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```c
  if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM5_Init 2 */
  /* USER CODE END TIM5_Init 2 */

}

/**
  * @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();
  }
```

```c
  /* 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 */
```

```c
}

/**
  * @brief USART6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART6_UART_Init(void)
{

  /* USER CODE BEGIN USART6_Init 0 */

  /* USER CODE END USART6_Init 0 */

  /* USER CODE BEGIN USART6_Init 1 */

  /* USER CODE END USART6_Init 1 */
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 9600;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART6_Init 2 */

  /* USER CODE END USART6_Init 2 */

}
```

```c
/**
  * @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_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

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

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, DIGIT_B0_Pin|DIGIT_B1_Pin|DIGIT_B2_Pin|DIGIT_B3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, DIGIT_A0_Pin|DIGIT_A1_Pin|DIGIT_A2_Pin|DIGIT_A3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : BLU_PB_Pin */
  GPIO_InitStruct.Pin = BLU_PB_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BLU_PB_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin BLU_Pin GRN_Pin RED_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|BLU_Pin|GRN_Pin|RED_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 pin : RPM_TICK_Pin */
  GPIO_InitStruct.Pin = RPM_TICK_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(RPM_TICK_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : DIGIT_B0_Pin DIGIT_B1_Pin DIGIT_B2_Pin DIGIT_B3_Pin */
  GPIO_InitStruct.Pin = DIGIT_B0_Pin|DIGIT_B1_Pin|DIGIT_B2_Pin|DIGIT_B3_Pin;
  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 pins : DIGIT_A0_Pin DIGIT_A1_Pin DIGIT_A2_Pin DIGIT_A3_Pin */
  GPIO_InitStruct.Pin = DIGIT_A0_Pin|DIGIT_A1_Pin|DIGIT_A2_Pin|DIGIT_A3_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(EXTI2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI2_IRQn);

  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 */
void DIGITS_Display(uint8_t DIGIT_A, uint8_t DIGIT_B)
```

```
{
    uint8_t DIGITA_VAL = 0x0F & DIGIT_A; //mask off higher4 bits
    int Abit0 = (DIGITA_VAL ) & 1;      // extract Abit0 of the 4-bit value
    int Abit1 = (DIGITA_VAL >> 1) & 1;  // extract Abit1 of the 4-bit value
    int Abit2 = (DIGITA_VAL >> 2) & 1;  // extract Abit2 of the 4-bit value
    int Abit3 = (DIGITA_VAL >> 3) & 1;  // extract Abit3 of the 4-bit value

    uint8_t DIGITB_VAL = 0x0F & DIGIT_B; //mask off higher4 bits
    int Bbit0 = (DIGITB_VAL ) & 1;      // extract Bbit0 of the 4-bit value
    int Bbit1 = (DIGITB_VAL >> 1) & 1;  // extract Bbit1 of the 4-bit value
    int Bbit2 = (DIGITB_VAL >> 2) & 1;  // extract Bbit2 of the 4-bit value
    int Bbit3 = (DIGITB_VAL >> 3) & 1;  // extract Bbit3 of the 4-bit value

    if (Abit0 == (0))
    {
     HAL_GPIO_WritePin(GPIOB, DIGIT_A0_Pin, GPIO_PIN_RESET);
    }
    else
    {
     HAL_GPIO_WritePin(GPIOB, DIGIT_A0_Pin, GPIO_PIN_SET);

    }
    if (Abit1 == (0))
    {
     HAL_GPIO_WritePin(GPIOB, DIGIT_A1_Pin, GPIO_PIN_RESET);
    }
    else
    {
     HAL_GPIO_WritePin(GPIOB, DIGIT_A1_Pin, GPIO_PIN_SET);

    }
    if (Abit2 == (0))
    {
     HAL_GPIO_WritePin(GPIOB, DIGIT_A2_Pin, GPIO_PIN_RESET);
    }
    else
```

```
{
 HAL_GPIO_WritePin(GPIOB, DIGIT_A2_Pin, GPIO_PIN_SET);

}
if (Abit3 == (0))
{
 HAL_GPIO_WritePin(GPIOB, DIGIT_A3_Pin, GPIO_PIN_RESET);
}
else
{
 HAL_GPIO_WritePin(GPIOB, DIGIT_A3_Pin, GPIO_PIN_SET);

}


if (Bbit0 == (0))
{
 HAL_GPIO_WritePin(GPIOC, DIGIT_B0_Pin, GPIO_PIN_RESET);
}
else
{
 HAL_GPIO_WritePin(GPIOC, DIGIT_B0_Pin, GPIO_PIN_SET);

}
if (Bbit1 == (0))
{
 HAL_GPIO_WritePin(GPIOC, DIGIT_B1_Pin, GPIO_PIN_RESET);
}
else
{
 HAL_GPIO_WritePin(GPIOC, DIGIT_B1_Pin, GPIO_PIN_SET);

}
if (Bbit2 == (0))
{
 HAL_GPIO_WritePin(GPIOC, DIGIT_B2_Pin, GPIO_PIN_RESET);
```

```
    }
    else
    {
     HAL_GPIO_WritePin(GPIOC, DIGIT_B2_Pin, GPIO_PIN_SET);

    }
    if (Bbit3 == (0))
    {
     HAL_GPIO_WritePin(GPIOC, DIGIT_B3_Pin, GPIO_PIN_RESET);
    }
    else
    {
     HAL_GPIO_WritePin(GPIOC, DIGIT_B3_Pin, GPIO_PIN_SET);
    }
}

void ADC_Select_CH(int CH)
{
  ADC_ChannelConfTypeDef sConfig = {0};
    switch(CH)
    {
    case 0:
    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = 1;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
    Error_Handler();
    }
    break;
    case 1:
    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = 1;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
    Error_Handler();
    }
```

```
break;
case 2:
sConfig.Channel = ADC_CHANNEL_2;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 3:
sConfig.Channel = ADC_CHANNEL_3;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 4:
sConfig.Channel = ADC_CHANNEL_4;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 5:
sConfig.Channel = ADC_CHANNEL_5;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 6:
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
```

```
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 7:
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 8:
sConfig.Channel = ADC_CHANNEL_8;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 9:
sConfig.Channel = ADC_CHANNEL_9;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 10:
sConfig.Channel = ADC_CHANNEL_10;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
```

```
break;
case 11:
sConfig.Channel = ADC_CHANNEL_11;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 12:
sConfig.Channel = ADC_CHANNEL_12;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 13:
sConfig.Channel = ADC_CHANNEL_13;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 14:
sConfig.Channel = ADC_CHANNEL_14;
sConfig.Rank = 1;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}
break;
case 15:
sConfig.Channel = ADC_CHANNEL_15;
sConfig.Rank = 1;
```

```
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
    Error_Handler();
    }
    break;
    }
}

void UART_Parse_MSG(char *option_msg, char *msg, volatile uint8_t *Zone, unsigned int input_char_num)
{
    user_input_flag = 0;
  uint8_t txd_msg_buffer[64] = {0};
  uint8_t option_buffer[128] = {0};
  uint8_t temp_zone[4] = {0};

  memset(temp_zone, 0, sizeof(temp_zone));

  sprintf((char*)txd_msg_buffer, "%s", msg);
  sprintf((char*)option_buffer, "%s", option_msg);
  // Send the prompt to the user
  HAL_UART_Transmit(&huart6, option_buffer, strlen((char*)option_buffer), 1000);
  HAL_UART_Transmit(&huart6, txd_msg_buffer, strlen((char*)txd_msg_buffer), 1000);
  // Initialize user input interrupt to write input to PWM_Zone
  HAL_UART_Receive_IT(&huart6, temp_zone, input_char_num);
  // Wait for the reception to complete
  while(user_input_flag == 0) {};
  temp_zone[input_char_num] = '\0';
  // Convert received data
  *Zone = atoi((const char *)temp_zone);
  // Echo the received byte back to UART to confirm reception
  sprintf((char*)txd_msg_buffer, "%i", *Zone);
  HAL_UART_Transmit(&huart6, txd_msg_buffer, strlen((char*)txd_msg_buffer), 1000);
}

void UART_Send_MSG(char* msg)
{
```

```
    uint8_t msg_buffer[128] = {0};
    sprintf((char*)msg_buffer, "%s", msg);
    HAL_UART_Transmit(&huart6, msg_buffer, strlen((char*)msg_buffer), 1000);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART6) {
        user_input_flag = 1;
    }

    if (huart->Instance == USART1)
    {
        us100_Rx_flag = 01; //this flag is set to show that an receiver interrupt has occurred
    }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == BLU_PB_Pin) {
        RUN_MODE_START_FLAG = 1;
    }

    if(GPIO_Pin == RPM_TICK_Pin) {
        rpm_tick_count++;  // Ensure this is incrementing

    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM5) {
        // MOTRO RPM calculation
        uint32_t ticks = rpm_tick_count - last_rpm_tick_count;
        if (ticks > 0) {
                DC_Motor_RPM = (ticks * 60) / 20;  // Calculate RPM based on ticks
```

```
        } else {
                DC_Motor_RPM = 0;  // Set RPM to zero if no ticks are detected
        }
        last_rpm_tick_count = rpm_tick_count;  // Update last count

        // Update wall clock second every real second
        simulate_seconds += 600;  // Simulating 24 hour minutes per 2.4 min
        Update_Wall_CLK_Display(simulate_seconds);
    }

}

void Control_DC_Motor_PWM(uint8_t percent, uint8_t direction)
{
  if (percent == 0) {
        TIM3->CCR1 = 0;
        TIM3->CCR3 = 0;
        return;
  }

    uint32_t pwm_value = (2000 * percent) / 100;  // Calculate the PWM value based on the percentage
    // Set motor direction 0 - reverse, 1 - forward
    if (direction == 1) {
        // Forward direction
        TIM3->CCR1 = pwm_value; // Set PWM for forward
        TIM3->CCR3 = 0;         // Ensure the reverse is 0
    } else {
        // Reverse direction
        TIM3->CCR1 = 0;         // Ensure the forward is 0
        TIM3->CCR3 = pwm_value; // Set PWM for reverse
    }
}

void Control_Servo_Motor_PWM(uint8_t direction)
{
    // Set motor direction 0 - INLET
```

```
    if (direction == 0) {
        TIM2->CCR1 = 2500;
    } else if (direction == 1){
        TIM2->CCR1 = 500;
    } else if (direction == 2) {
        TIM2->CCR1 = 1200;
    } else if (direction == 3 ) {
        TIM2->CCR1 = 1900;
    }
}

void Update_Wall_CLK_Display(volatile uint64_t simulated_seconds)
{
        static uint8_t last_hour_displayed = 255;
        if (simulate_seconds >= 3600) {
            simulate_seconds -= 3600;
            Current_Wall_CLK_Hour++;
            if (Current_Wall_CLK_Hour >= 24) {
                    Current_Wall_CLK_Hour = 0;
            }
        }
        uint8_t scaled_hour = (WALL_CLK_START + Current_Wall_CLK_Hour) % 24;

    if ( (last_hour_displayed != scaled_hour) && System_Interlock_Flag == 0 ) {
            last_hour_displayed = scaled_hour;
            char buf[256];
            // Format and send the message every hour
            sprintf(buf, "Wall-Clock Time: %02d | Zone/Inlet: %s | Motor Speed %%PWM: %d%% | Motor RPM: %lu | Water Reservoir
Depth: %d%%\r\n",
                            Current_Wall_CLK_Hour, Zone_Names[Current_Zone], Current_DC_Motor_Percent_PWM, DC_Motor_RPM,
Current_Water_Percent_depth);

            HAL_UART_Transmit(&huart6, (uint8_t*)buf, strlen(buf), 1000);
    }

    uint8_t digit_a = scaled_hour / 10; // Tens digit of the hour
```

```
    uint8_t digit_b = scaled_hour % 10; // Units digit of the hour

    DIGITS_Display(digit_a, digit_b); // Display the scaled hour
}

uint8_t ADC_Manual_Control_Percent_PWM()
{
  uint8_t adc_value = 0; // 0 - 255
    ADC_Select_CH(9);
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 1000);
    adc_value = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);

    // Calculate percentage (0 to 100%)
    uint8_t percentage = (uint8_t)((adc_value * 100) / 255);

  return percentage;
}

void Get_Water_Percent_depth()
{
    HAL_UART_Receive_IT(&huart1, us100_buffer, 2);
    HAL_UART_Transmit(&huart1, &cmd_dist, 1, 500);
    HAL_Delay(0.005);
    while( us100_Rx_flag == (00) ) {};
    // Combine the two bytes into a single 16-bit integer
    distance_mm = ((uint16_t)us100_buffer[0] << 8) | us100_buffer[1];
  uint8_t calculated_depth = 0;

    if( distance_mm >= Tank_Level_Lo ) {
        calculated_depth = 0;
        Tank_Level_Lo_Alarm = 1;
    } else if( distance_mm <= Tank_Level_Hi ) {
        calculated_depth = 99;
        Tank_Level_Hi_Alarm = 1;
```

```
    } else {
        calculated_depth = (uint8_t) (100 - ((distance_mm - Tank_Level_Hi) * 100 / (Tank_Level_Lo - Tank_Level_Hi)));
    }

  Current_Water_Percent_depth = calculated_depth;
  HAL_Delay(50);
}

void Set_LED_Color(LED_Color color)
{
    switch (color) {
        case LED_RED:
            HAL_GPIO_WritePin(GPIOA, RED_Pin, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOA, GRN_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, BLU_Pin, GPIO_PIN_RESET);
            break;

        case LED_GREEN:
            HAL_GPIO_WritePin(GPIOA, RED_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GRN_Pin, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOA, BLU_Pin, GPIO_PIN_RESET);
            break;

        case LED_BLUE:
            HAL_GPIO_WritePin(GPIOA, RED_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GRN_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, BLU_Pin, GPIO_PIN_SET);
            break;

        case LED_PURPLE:
            HAL_GPIO_WritePin(GPIOA, RED_Pin, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOA, GRN_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, BLU_Pin, GPIO_PIN_SET);
            break;

        case LED_OFF:
```

```
        default:
            HAL_GPIO_WritePin(GPIOA, RED_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GRN_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, BLU_Pin, GPIO_PIN_RESET);
            break;
    }
}

void System_Interlock()
{
    System_Interlock_Flag = 1;
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    UART_Send_MSG("\r\n\n RESERVOIR IS EMPTY");

    while (1) {
        // Turn off DC Motor
        Control_DC_Motor_PWM(0, 0);
        Set_LED_Color(LED_RED);
        HAL_Delay(500);
        Set_LED_Color(LED_GREEN);
        HAL_Delay(500);
        Set_LED_Color(LED_BLUE);
        HAL_Delay(500);
    }
}
/* 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();
```

```c
  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 */
```