```c
/* USER CODE BEGIN Header */
/**

******************************************************************************
***
 * @file           : main.c
 * @brief          : Main program body

******************************************************************************
***
 * @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 <stdio.h>
#include "string.h"
#include "sht3x.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

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

/* USER CODE END PD */

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

/* USER CODE END PM */

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

I2C_HandleTypeDef hi2c2;
```

```c
UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_rx;
DMA_HandleTypeDef hdma_usart1_tx;

/* USER CODE BEGIN PV */
int isDataSent = 1;
uint32_t adcValue = 0;
char data[100];
int windowStatus;
float humidity, temperature;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
---*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_I2C2_Init(void);
/* USER CODE BEGIN PFP */
void transmistDataOverUART1(char *data);
void sendLightSenorData(void);
void setWindowSensorPinAsInput(void);
void sendTempAndHumidityData(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------
---*/
/* USER CODE BEGIN 0 */

// Data Transmit CALL BACK
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
	isDataSent = 1;
}

// SHT30 code
sht3x_handle_t handle = { .i2c_handle = &hi2c2, .device_address =
SHT3X_I2C_DEVICE_ADDRESS_ADDR_PIN_LOW };

int checkSHT30(void) {
	if (!sht3x_init(&handle)) {
		return 0;

	} else {
		return 1;
	}
}

/* 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_DMA_Init();
	MX_ADC1_Init();
	MX_USART1_UART_Init();
	MX_I2C2_Init();
	/* USER CODE BEGIN 2 */
	void setWindowSensorPinAsInput(void);
	if (HAL_GPIO_ReadPin(WINDOW_SENSOR_GPIO_Port, WINDOW_SENSOR_Pin)
			== GPIO_PIN_SET) {
		windowStatus = 1;
		char buff[] =

	"publish:esp01_waleed/sensors/window_sensor|{\"state\":\"close\"}\n\0";
		HAL_UART_Transmit(&huart1, (uint8_t*) buff, strlen(buff),
		HAL_MAX_DELAY);

	} else {
		windowStatus = 0;
		char buff[] =

	"publish:esp01_waleed/sensors/window_sensor|{\"state\":\"open\"}\n\0";
		HAL_UART_Transmit(&huart1, (uint8_t*) buff, strlen(buff),
		HAL_MAX_DELAY);
	}
	MX_GPIO_Init();
	checkSHT30();

	/* USER CODE END 2 */

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

		/* USER CODE BEGIN 3 */
		sendLightSenorData();
		sht3x_read_temperature_and_humidity(&handle, &temperature, &humidity);
		sendTempAndHumidityData();
```

```
            HAL_Delay(5000);
      }
      /* USER CODE END 3 */
}

/**
 * @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_HSE;
      RCC_OscInitStruct.HSEState = RCC_HSE_ON;
      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_HSE;
      RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
      RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
      RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

      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_12B;
	hadc1.Init.ScanConvMode = DISABLE;
	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;
	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_1;
	sConfig.Rank = 1;
	sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
		Error_Handler();
	}
	/* USER CODE BEGIN ADC1_Init 2 */

	/* USER CODE END ADC1_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 = 1000;
	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 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 = 115200;
	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 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void) {

	/* DMA controller clock enable */
	__HAL_RCC_DMA2_CLK_ENABLE();

	/* DMA interrupt init */
	/* DMA2_Stream2_IRQn interrupt configuration */
	HAL_NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 0);
	HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQn);
	/* DMA2_Stream7_IRQn interrupt configuration */
	HAL_NVIC_SetPriority(DMA2_Stream7_IRQn, 0, 0);
	HAL_NVIC_EnableIRQ(DMA2_Stream7_IRQn);

}

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

	/*Configure GPIO pin : ESP_STATUS_Pin */
	GPIO_InitStruct.Pin = ESP_STATUS_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull = GPIO_NOPULL;
	HAL_GPIO_Init(ESP_STATUS_GPIO_Port, &GPIO_InitStruct);

	/*Configure GPIO pin : WINDOW_SENSOR_Pin */
	GPIO_InitStruct.Pin = WINDOW_SENSOR_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
	GPIO_InitStruct.Pull = GPIO_NOPULL;
	HAL_GPIO_Init(WINDOW_SENSOR_GPIO_Port, &GPIO_InitStruct);

	/* EXTI interrupt init*/
	HAL_NVIC_SetPriority(EXTI2_IRQn, 1, 0);
	HAL_NVIC_EnableIRQ(EXTI2_IRQn);

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

/* USER CODE BEGIN 4 */

// Interrupt handler
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
	if (GPIO_Pin == WINDOW_SENSOR_Pin) {

		if (HAL_GPIO_ReadPin(WINDOW_SENSOR_GPIO_Port,
WINDOW_SENSOR_Pin)
				== GPIO_PIN_SET) {
			strcpy(data,

	"publish:esp01_waleed/sensors/window_sensor|{\"state\":\"close\"}\n\0
");
//			HAL_UART_Transmit(&huart1, (uint8_t*) data, strlen(data),
//			HAL_MAX_DELAY);

			transmistDataOverUART1(data);
			windowStatus = 1;
			return;

		} else {

			strcpy(data,

	"publish:esp01_waleed/sensors/window_sensor|{\"state\":\"open\"}\n\0"
);

			transmistDataOverUART1(data);
			windowStatus = 0;
			return;
```

```
			}

		}
}

void transmistDataOverUART1(char *dataToTransmit) {
		while (isDataSent != 1) {
			// wait untill previous transmission is ongoing
		}
		HAL_UART_Transmit_DMA(&huart1, (uint8_t*) dataToTransmit,
				strlen(dataToTransmit));
		isDataSent = 0;

}

void sendLightSenorData(void) {
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
		unsigned int adc_value = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
		HAL_Delay(10);
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
		adc_value += HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
		HAL_Delay(10);
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
		adc_value += HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);

		adc_value = (adc_value / 3);
		adcValue = adc_value;

		char buffer[70];
		sprintf(buffer,

		"publish:esp01_waleed/sensors/lightlevel|{\"lux\":\"%d\"}\n",
				adc_value);
//		HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer),
//				HAL_MAX_DELAY);
		transmistDataOverUART1(buffer);
		HAL_Delay(10);
}

void setWindowSensorPinAsInput(void) {
		GPIO_InitTypeDef GPIO_InitStruct = { 0 };
		/*Configure GPIO pin : WINDOW_SENSOR_Pin */
		GPIO_InitStruct.Pin = WINDOW_SENSOR_Pin;
		GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
		GPIO_InitStruct.Pull = GPIO_NOPULL;
		HAL_GPIO_Init(WINDOW_SENSOR_GPIO_Port, &GPIO_InitStruct);
}

void sendTempAndHumidityData(void) {
		char buffer[70];
		sprintf(buffer,

		"publish:esp01_waleed/sensors/TH_sensor|{\"temprature\":\"%d\"}\n",
				(int) temperature);
		transmistDataOverUART1(buffer);
```

```
        HAL_Delay(10);
        sprintf(buffer,

        "publish:esp01_waleed/sensors/TH_sensor|{\"humidity\":\"%d\"}\n",
                    (int) humidity);
        transmistDataOverUART1(buffer);
        HAL_Delay(10);

}
/* 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 */
```