```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 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"
/* 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 hadc;

I2C_HandleTypeDef hi2c2;

TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint16_t readValue;
#define RT0 10000  // Ω
#define B 3470      // K

#define VCC 3.3   //Supply voltage
```

```c
#define R 10000
/* 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_I2C2_Init(void);
static void MX_TIM1_Init(void);
static void MX_ADC_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
//General purpose Function to send a char array over the UART and to automatically
send a new line character after it
void debugPrintln(UART_HandleTypeDef *uart_handle,char _out[])
{
	HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);
	HAL_UART_Transmit(uart_handle, (uint8_t *) _out,strlen(_out), 60);
	char newline[2] = "\r\n";
	HAL_UART_Transmit(uart_handle, (uint8_t *)newline, 2, 10);

}

char str[60] = { 0 };
#define DHT11_PORT GPIOA
#define DHT11_PIN GPIO_PIN_4
uint8_t RHI, RHD, TCI, TCD, SUM;
uint32_t pMillis, cMillis;
float tCelsius = 0;
float tFahrenheit = 0;
float RH = 0;

void microDelay (uint16_t delay)
{
__HAL_TIM_SET_COUNTER(&htim1, 0);
while (__HAL_TIM_GET_COUNTER(&htim1) < delay);
}

uint8_t DHT11_Start (void)
{
uint8_t Response = 0;
GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
GPIO_InitStructPrivate.Pin = DHT11_PIN;
GPIO_InitStructPrivate.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as output
HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);   // pull the pin low
HAL_Delay(20);   // wait for 20ms
HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 1);   // pull the pin high
microDelay (30);   // wait for 30us
```

```
GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
GPIO_InitStructPrivate.Pull = GPIO_PULLUP;
HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as input
microDelay (40);
if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
{
  microDelay (80);
  if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
}
pMillis = HAL_GetTick();
cMillis = HAL_GetTick();
while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
{
  cMillis = HAL_GetTick();
}
return Response;
}

uint8_t DHT11_Read (void)
{
uint8_t a,b;
for (a=0;a<8;a++)
{
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go high
    cMillis = HAL_GetTick();
  }
  microDelay (40);   // wait for 40 us
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
    b&= ~(1<<(7-a));
  else
    b|= (1<<(7-a));
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go low
    cMillis = HAL_GetTick();
  }
}
return b;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```c
  /* 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_I2C2_Init();
  MX_TIM1_Init();
  MX_ADC_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim1);
  float RT, VR, ln12, TX, T0, VRT;
  T0 = 298.15;



  char str[60] = { 0 }; //Useful buffer for printing to UART
  uint8_t I2CReturn = 0; //Status var to indicate if HAL_I2C operation has succeeded
(1) or failed (0);
  uint8_t i, j, Loop = 0; //Loop counters
  //Setup variables for reading and writing
  uint16_t EEPROM_DEVICE_ADDR = 0x50 << 1; //Address of EEPROM device on I2C bus
  uint16_t madd = 0x00; //Memory address variable containing a starting memory
address for a location of memory in the EEPROM
  uint8_t Data = 0x10;//Data variable containing sStarting value to write to memory,
could be any 8bit value
  uint8_t Data2 = 0x10;
  uint8_t *sData2 = &Data2;
  uint8_t *sData = &Data; //Pointer to sending Data variable
  uint8_t Result = 0x00; //Variable to stored value read back from memory in
  uint8_t *rData = &Result; //Pointer to result data variable
  //Say hello over UART
  debugPrintln(&huart2, "Hello, this is STMF0 Discovery board: ");
  /* USER CODE END 2 */

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

```
                    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_SET)
                    {
                                break;
                    }
                    else
                    {


                      if(DHT11_Start())
                                      {
                                        RHI = DHT11_Read(); // Relative humidity
integral
                                        RHD = DHT11_Read(); // Relative humidity
decimal


                                        TCI = DHT11_Read(); // Celsius integral
                                        TCD = DHT11_Read(); // Celsius decimal
                                    Data2 = RHI;
                                    Data = TCI;
                                       sprintf(str, "Relative humidity decimal :
%d.%d ", RHI, RHD);
                                                                debugPrintln(&huart2,
str);
                                         sprintf(str, "Temperature celsius decimal :
%d.%d ", TCI, TCD);
                                                                debugPrintln(&huart2,
str);
                                      }
                      HAL_Delay(2000);

                      HAL_ADC_Start(&hadc);
                      HAL_ADC_PollForConversion(&hadc,1000);
                      readValue = HAL_ADC_GetValue(&hadc);
                      //VRT = readValue;              //Acquisition analog value of
VRT
                       VRT = (3.3 / 4050) * readValue;      //Conversion to voltage
                       VR = VCC - VRT;
                       RT = VRT / (VR / R);               //Resistance of RT

                       ln12 = log(RT / RT0);
                       TX = (1 / ((ln12 / B) + (1 / T0))); //Temperature from
thermistor

                      TX = TX - 287.13;                 //Conversion to Celsius


                         sprintf(str, "Temperature celsius decimal from analog : %f
\n", TX);

debugPrintln(&huart2, str);
                         HAL_ADC_Stop(&hadc);
                         HAL_Delay(1000);
```

```
                        //WRITING
                                memset(str, 0, sizeof(str));
                                sprintf(str, "Writing the temperature %d to EEPROM
address 0x%X", Data, madd);
                                debugPrintln(&huart2, str);
                                I2CReturn = HAL_I2C_Mem_Write(&hi2c2,
EEPROM_DEVICE_ADDR, madd, 2, sData, 1, HAL_MAX_DELAY);
                                if (I2CReturn != HAL_OK) {
                                debugPrintln(&huart2, "Write to address FAILED");
                                }
                                //READING
                                memset(str, 0, sizeof(str));
                                sprintf(str, "Reading from EEPROM address 0x%X ",
madd);
                                debugPrintln(&huart2, str);
                                I2CReturn = HAL_I2C_Mem_Read(&hi2c2,
EEPROM_DEVICE_ADDR, madd, 2, rData, 1, HAL_MAX_DELAY);
                                if (I2CReturn != HAL_OK) {
                                debugPrintln(&huart2, "Read from address FAILED");
                                }
                                //PRINT READ VALUE
                                memset(str, 0, sizeof(str));
                                sprintf(str, "Received temperature data: %d \n",
Result);
                                debugPrintln(&huart2, str);
                                //Increment address and data values and clear
Result holder
                                madd = madd + 1;
                                Result = 0x00;


              //WRITING
                                        memset(str, 0, sizeof(str));
                                        sprintf(str, "Writing the humidity %d to
EEPROM address 0x%X", Data2, madd);
                                        debugPrintln(&huart2, str);
                                        I2CReturn = HAL_I2C_Mem_Write(&hi2c2,
EEPROM_DEVICE_ADDR, madd, 2, sData2, 1, HAL_MAX_DELAY);
                                        if (I2CReturn != HAL_OK) {
                                        debugPrintln(&huart2, "Write to address
FAILED");
                                        }
                                        //READING
                                        memset(str, 0, sizeof(str));
                                        sprintf(str, "Reading from EEPROM address
0x%X ", madd);
                                        debugPrintln(&huart2, str);
                                        I2CReturn = HAL_I2C_Mem_Read(&hi2c2,
EEPROM_DEVICE_ADDR, madd, 2, rData, 1, HAL_MAX_DELAY);
                                        if (I2CReturn != HAL_OK) {
                                        debugPrintln(&huart2, "Read from address
FAILED");
                                        }
                                        //PRINT READ VALUE
                                        memset(str, 0, sizeof(str));
```

```
                                        sprintf(str, "Received humidity data: %d
\n", Result);
                                        debugPrintln(&huart2, str);
                                        //Increment address and data values and
clear Result holder
                                        madd = madd + 1;
                                        Result = 0x00;
                                    HAL_Delay(1000);
                        }

    /* USER CODE END WHILE */

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

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

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSI14;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.HSI14CalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  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_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/**
  * @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 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = ENABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_55CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */

  /* USER CODE END ADC_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.Timing = 0x2010091A;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

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

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 47;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_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 */
```

```c
  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  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;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_SWAP_INIT;
  huart2.AdvancedInit.Swap = UART_ADVFEATURE_SWAP_ENABLE;
  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};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_6|GPIO_PIN_7,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, LD4_Pin|LD3_Pin, GPIO_PIN_RESET);

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

  /*Configure GPIO pins : PA1 PA4 PA6 PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_6|GPIO_PIN_7;
  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 : LD4_Pin LD3_Pin */
  GPIO_InitStruct.Pin = LD4_Pin|LD3_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 pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}

/* 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 */
```