Temperature and Pressure Monitoring System using STM32F103

Chirag Satapathy (19BEI0107), Hrishikesh Gokhale (19BEI0129)

Vellore Institute of Technology, Vellore

(Electronics and Instrumentation)

Under

Prof. Selvakumar K (VIT)

Embedded System Design

December 2021

Abstract

This project implements a weather monitoring system by measuring temperature and pressure of the given environment on a daily basis to keep a track of the upcoming weather conditions and help forecasting it via news to the public. This system will use an LM35 temperature sensor and also a pressure sensor to measure temperature and pressure of the surrounding environment continuously. The main control unit of this device is an ARM based microcontroller called the STM32.A Liquid Crystalline Display (LCD) is used for displaying the values of the weather forecast so that the user can have a clear view on the temperature and pressure. Also, Universal Synchronous Asynchronous Receiver Transmitter (USART) communication protocol is implemented to the device and the values can be seen on a separate terminal by the controller of the device to have a much clear view of the forecast. The proposed system will help in determining the weather forecast in the upcoming days, weeks, months or years and also determine the condition of the weather and let the public know about this so that they can plan their day. For e.g.: if the weather forecast shows it will probably rain during the evening 4pm, then those people can carry an umbrella to keep themselves dry during the rain.

Keywords**:** STM32F103, USART, Weather Monitoring

Hardware Components (Simulation):

1. ARM based microprocessor – STM32F103
2. LM35 Temperature Sensor
3. MPX4115 Pressure Sensor
4. LCD

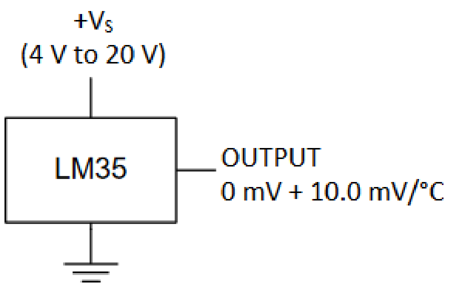
Software Used:

1. STM32CUBE IDE
2. Proteus 8.14

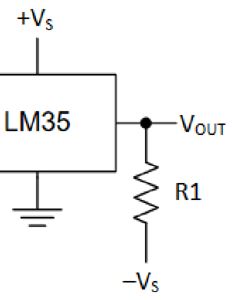
LM35 Temperature Sensor:

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly- proportional to the Centigrade temperature. The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range. Lower cost is assured by trimming and calibration at the wafer level. The low-output impedance, linear output, and precise inherent calibration of the LM35 device makes interfacing to readout or control circuitry especially easy. The device is used with single power supplies, or with plus and minus supplies. As the LM35 device draws only 60 µA from the supply, it has very low self-heating of less than 0.1°C in still air. The LM35 device is rated to operate over a −55°C to 150°C temperature range, while the LM35C device is rated for a −40°C to 110°C range (−10° with improved accuracy).

Basic Centigrade Temperature Sensor (2°C to 150°C)

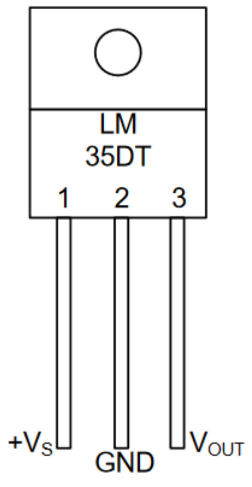


Full-Range Centigrade Temperature Sensor



Choose $R_1 = -V_S$ / 50 µA
V_{OUT} = 1500 mV at 150°C
V_{OUT} = 250 mV at 25°C
V_{OUT} = –550 mV at –55°C

NEB Package
3-Pin TO-220
(Top View)

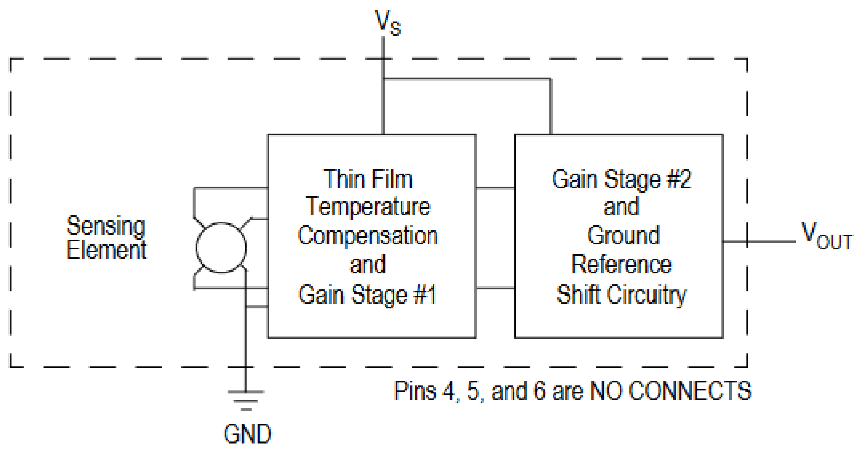


Features of LM35:

1. Calibrated directly in Celsius (Centigrade)
2. Linear + 10-mV/°C Scale Factor
3. 0.5°C Ensured Accuracy (at 25°C)
4. Rated for Full −55°C to 150°C Range
5. Suitable for Remote Applications
6. Low-Cost Due to Wafer-Level Trimming
7. Operates From 4 V to 30 V
8. Less Than 60-µA Current Drain
9. Low Self-Heating, 0.08°C in Still Air
10. Non-Linearity Only ±¼°C Typical
11. Low-Impedance Output, 0.1 Ω for 1-mA Load

<u>MPX4115 Pressure Sensor:</u>

The MPX4115 is designed to sense absolute air pressure in an altimeter or barometer (BAP) application. The sensor integrates on-chip, bipolar op amp circuitry and thin film resistor networks to provide a high-level analog output signal and temperature compensation. The small form factor and high reliability of on-chip integration makes the sensor a logical and economical choice.


V_S
Sensing Element
Thin Film Temperature Compensation and Gain Stage #1
Gain Stage #2 and Ground Reference Shift Circuitry
V_OUT
GND
Pins 4, 5, and 6 are NO CONNECTS

Features of MPX4115:

1. 1.5% Maximum Error over 0° to 85°
2. Ideally suited for Microprocessor or Microcontroller-Based Systems
3. Available in Absolute, Differential and Gauge Configurations
4. Durable Epoxy Unibody Element
5. Easy-to-Use Chip Carrier Option

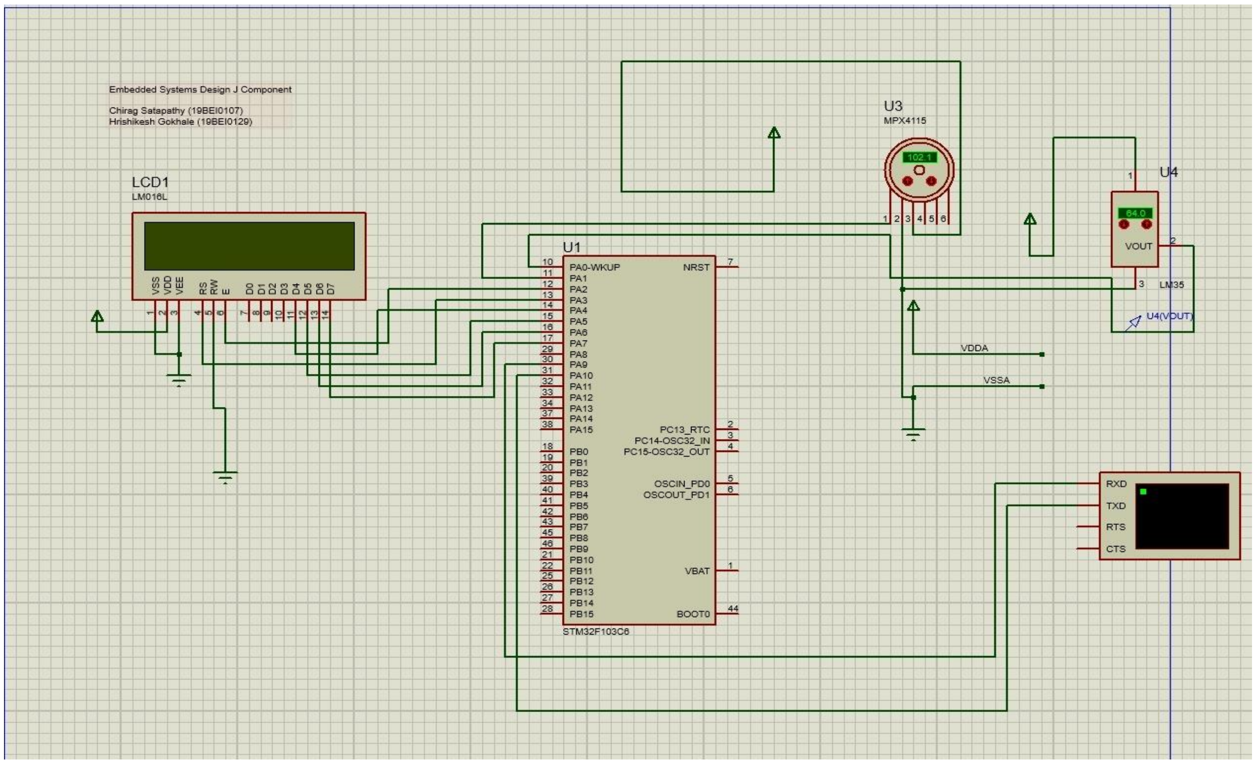
Construction:

The main microcontroller used for the project is the STM32F103 which uses the Cortex-M3 core, with a maximum CPU speed of 75MHz. Firstly, we have connected the LM35 based temperature sensor to the microcontroller. We have chosen this sensor as it does not require any external calibration over other thermistors and is low-cost. The coating over it also protects it from self-heating. It is also used to measure precise centigrade temperature like of the surroundings. Pin 1 of the LM35 is connected to the power supply, followed by pin 2 connected

A pressure sensor is connected to the control unit to measure the pressure change in the surroundings and thus complete the weather forecast monitoring system. The pressure sensor we have used is the MPX4115. This is an integrated silicon sensor is designed to sense absolute air pressure, temperature compensated and calibrated. It is small in size, consumes low power and also has the highest precision for air pressure measurement. Pin 3 of the sensor is connected to the power supply.

We have also used a 16x2 liquid crystalline display (LCD) to display the values of the obtained temperature and pressure in the weather monitoring system. Pins D4 – D7 of the LCD is connected to pins PA3, PA5, PA6 and PA7 of the microcontroller. Pins VSS, VEE and RW are connected to ground (GND) and VDD connected to 5V.

The overall circuit is designed, built and simulated on Proteus which is a simulation software. This can be seen in the figure below. The microcontroller STM32F103 is connected to the LCD, temperature sensor and the pressure sensor. In the figure, pins PA9 and PA10 of the microcontroller are connected to the RxD and TxD pins of the output terminal. These pins are called receiver data and transmit data. Using the communication protocol of USART, the values from both the sensors will be displayed on an external terminal so that the it would be easy to read the values obtained.


Embedded Systems Design J Component
Chirag Satapathy (19BEI0107)
Hrishikesh Gokhale (19BEI0129)
LCD1
LM016L
U1
STM32F103C6
U3
MPX4115
U4
LM35
VOUT
U4(VOUT)
VDDA
VSSA
RXD
TXD
RTS
CTS

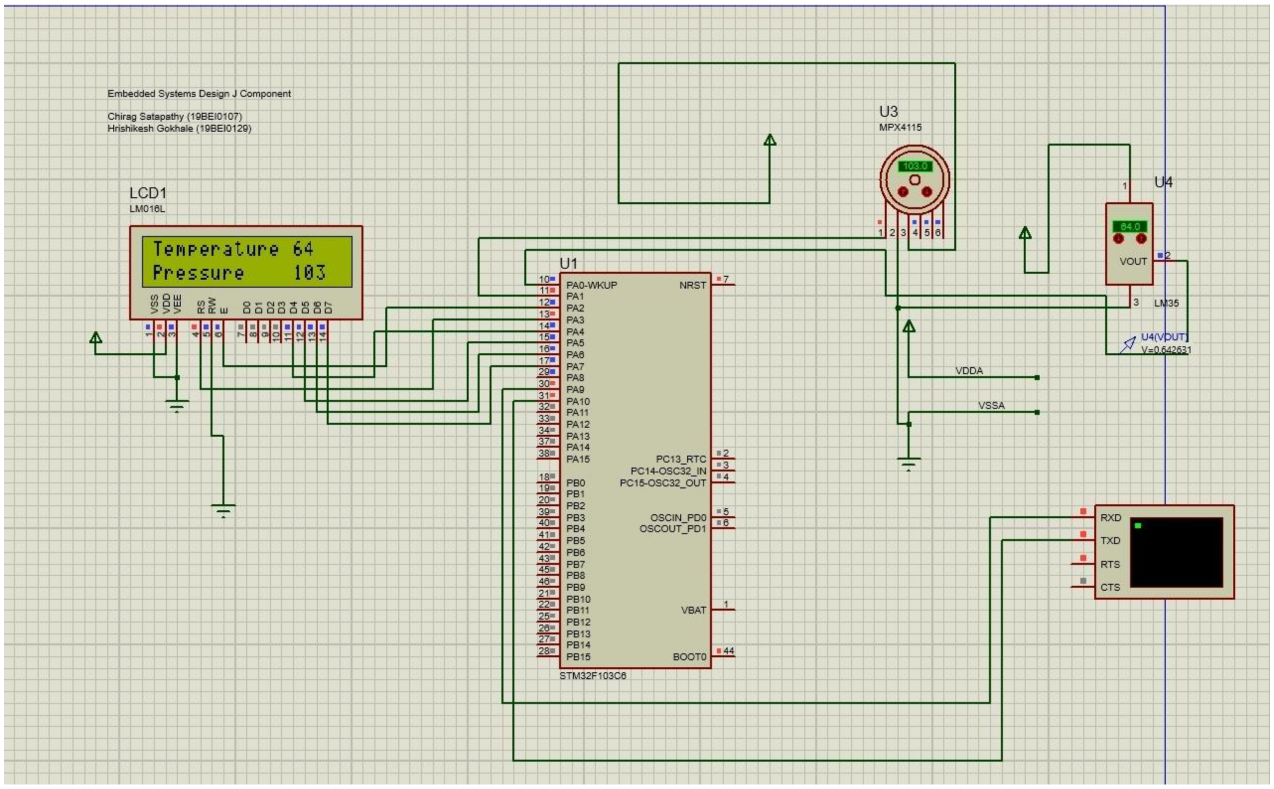
Result:

The LCD displays the values of the temperature and pressure collected by the respective sensors (LM35 and MPX4115) in real time.

The values of the MPX4115 Pressure Sensor are also rounded off after 0.5, i.e., if the sensor records 103.4, it will be displayed as 103 but if it records 103.7, it will be displayed as 104.

The working of the project is demonstrated in video format, which is provided here:

https://drive.google.com/file/d/1b55ZNzah4hGY7Jo5GbS-ipyOKYn0jhPn/view?usp=sharing



Conclusion:

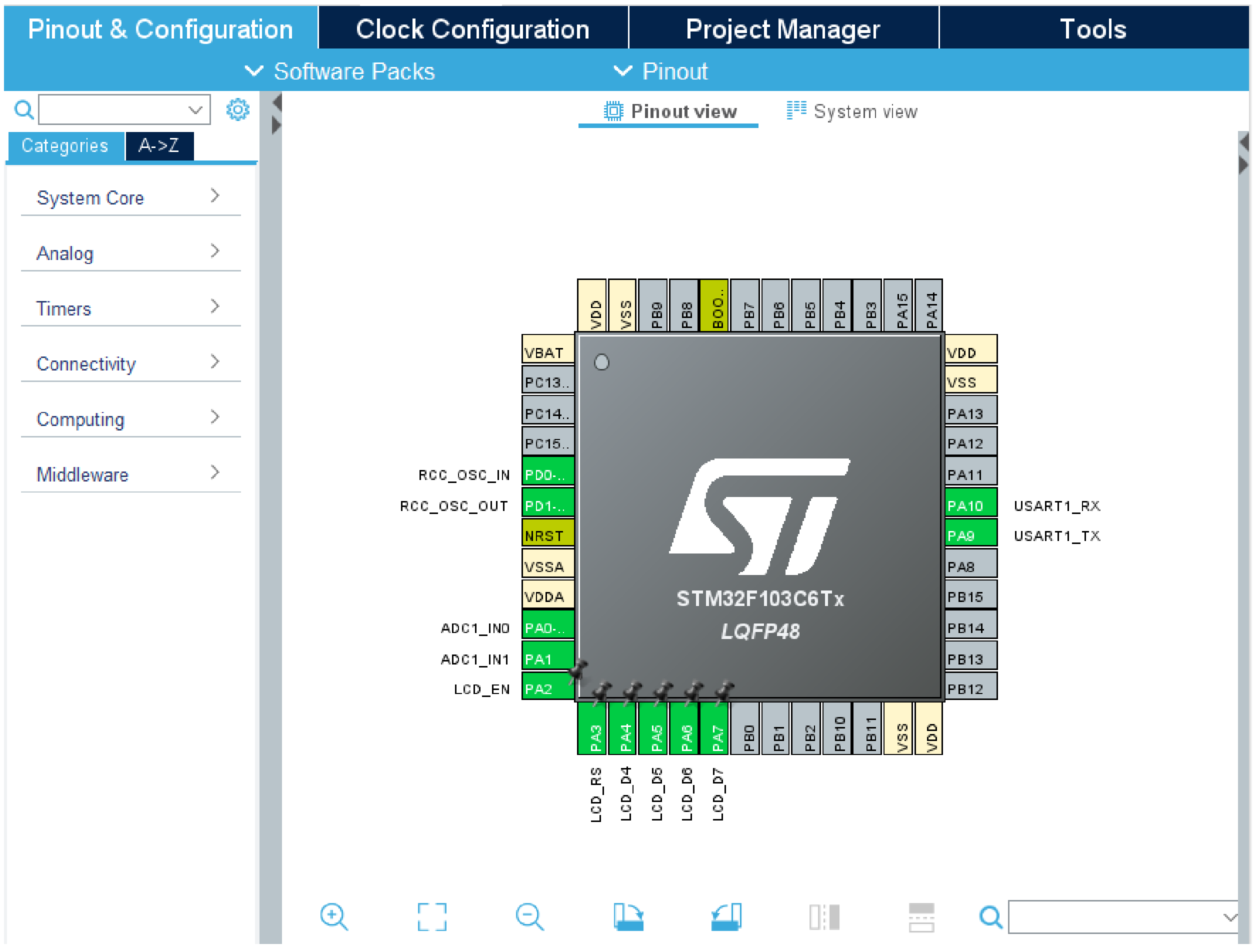
Due to unavailability of hardware components and a dysfunctional microcontroller, this entire project was completed using the electronics simulation-based platform – Proteus.

The microcontroller used in this project is STM32F103 and the programming language used to program the microcontroller is Embedded C. Embedded C was used using the STM32Cube IDE cross compiler.

References:

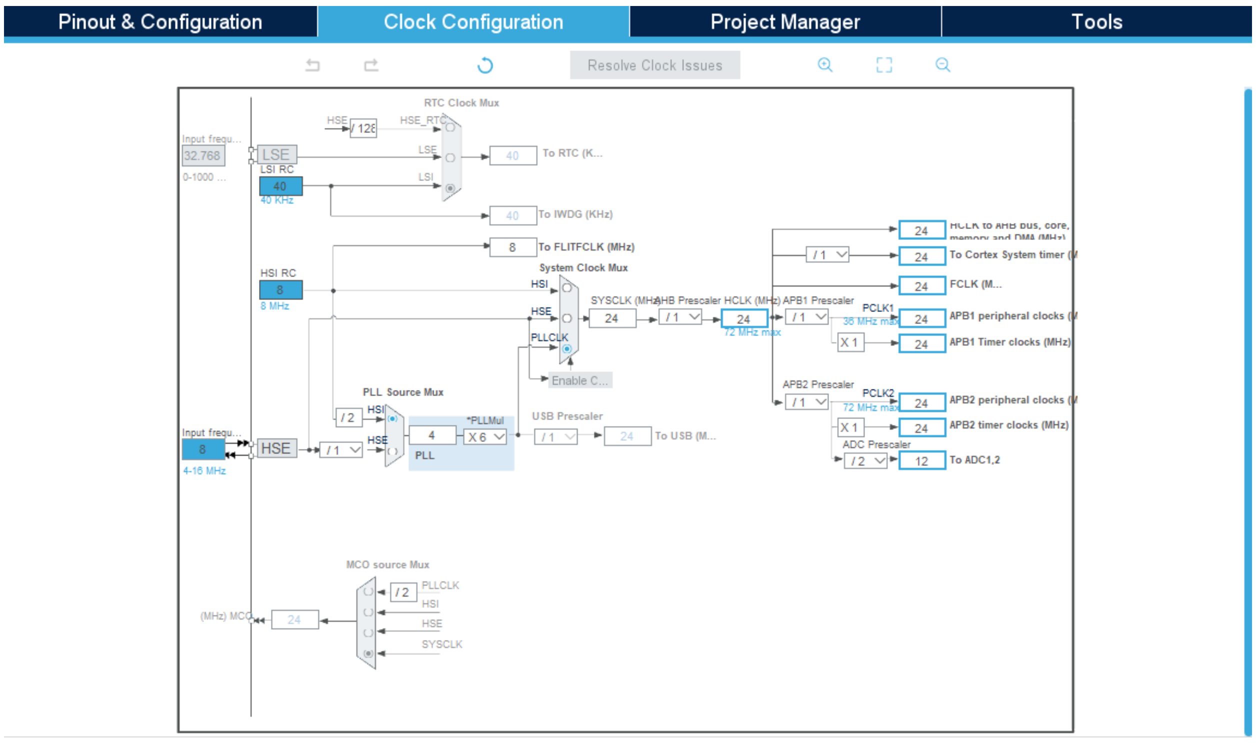
1. https://how2electronics.com/humidity-temperature-monitor-dht11-stm32-microcontroller/
2. https://www.ti.com/lit/ds/symlink/lm35.pdf (Datasheet - LM35)
3. https://www.nxp.com/files-static/sensors/doc/data_sheet/MPX4115.pdf (Datasheet - MPX4115)
4. https://www.scirp.org/journal/paperinformation.aspx?paperid=111770

Appendix – 1 (Pinout and configuration):



RCC_OSC_IN PD0-..
RCC_OSC_OUT PD1-..
ADC1_IN0 PA0-..
ADC1_IN1 PA1
LCD_EN PA2
LCD_RS PA3
LCD_D4 PA4
LCD_D5 PA5
LCD_D6 PA6
LCD_D7 PA7
USART1_RX PA10
USART1_TX PA9

Appendix – 2 (Clock configuration):



Appendix – 3 (Embedded C Program) – Compiled using STM32Cube IDE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
```

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "LCD.h"
#include<stdio.h>
#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 hadc1;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint32_t adcVal0, adcVal1;
ADC_ChannelConfTypeDef sConfig;

char msg[32];
char tem1[32];
/* 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_ADC1_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  lcd_init(_LCD_4BIT, _LCD_FONT_5x8, _LCD_2LINE);

  lcd_print(1,1,"Displaying Values");
  HAL_Delay(100);
  lcd_clear();
  /* Initializes SHT2x temperature/humidity sensor and sets the resolution. */
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
         sConfig.Channel=ADC_CHANNEL_0;
         HAL_ADC_ConfigChannel(&hadc1, &sConfig);
         HAL_ADC_Start(&hadc1);
         if(HAL_ADC_PollForConversion(&hadc1, 5)==HAL_OK)
         {
              adcVal0=HAL_ADC_GetValue(&hadc1);
              adcVal0=(adcVal0*500)/4096;
              //adcVal0=(adcVal0/4095+0.095)/0.009;
              HAL_Delay(50);
              lcd_clear();
              char tem[32]="";
              sprintf(tem,"%lu",adcVal0);
```

```
                lcd_print(1,1,"Temperature");
                lcd_print(1,13,tem);
                HAL_Delay(50);
            }
            //HAL_Delay(1000);
            sConfig.Channel=ADC_CHANNEL_1;
            HAL_ADC_ConfigChannel(&hadc1, &sConfig);
            HAL_ADC_Start(&hadc1);
            if(HAL_ADC_PollForConversion(&hadc1, 5)==HAL_OK)
            {
                adcVal1=HAL_ADC_GetValue(&hadc1);
                //adcVal1=(adcVal1*510)/4096;
                adcVal1=(((float)adcVal1/(float)4096)+0.095)/0.009-1;
                //adcVal0=(adcVal0/4095+0.095)/0.009;
                HAL_Delay(50);
                //lcd_clear();
                char tem1[32]="";
                sprintf(tem1,"%lu",adcVal1);
                lcd_print(2,13,tem1);
                lcd_print(2,1,"Pressure");
                HAL_Delay(100);
            }
            sprintf(msg,"Pressure=%lu Percent\r\n",adcVal1);
            HAL_UART_Transmit(&huart1,(uint8_t*)msg,strlen(msg),HAL_MAX_DELAY);
            HAL_Delay(10);
            char msg[32]="";
            sprintf(msg,"Temperature=%lu Celcius\r\n",adcVal0);
            HAL_UART_Transmit(&huart1,(uint8_t*)msg,strlen(msg),HAL_MAX_DELAY);
            HAL_Delay(10);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {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_MUL6;
  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_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != 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 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_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 */

}

/**
  * @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_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LCD_EN_Pin|LCD_RS_Pin|LCD_D4_Pin|LCD_D5_Pin
                          |LCD_D6_Pin|LCD_D7_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : LCD_EN_Pin LCD_RS_Pin LCD_D4_Pin LCD_D5_Pin
                           LCD_D6_Pin LCD_D7_Pin */
  GPIO_InitStruct.Pin = LCD_EN_Pin|LCD_RS_Pin|LCD_D4_Pin|LCD_D5_Pin
                          |LCD_D6_Pin|LCD_D7_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);

}

/* 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 */

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