

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра САП



**Лабораторна робота №4**

з дисципліни “Програмування інтелектуальних вбудованих систем”

на тему: “Розробка програми для роботи з сенсорами та датчиками на основі мікроконтролера STM32.

”

**Виконав**

ст. гр. ПП-31

Гаврилюк Назар

**Прийняв:**

Колесник К.К.

**Мета роботи.** ознайомитися з розробкою програмного забезпечення для взаємодії з сенсорами та актуаторами на основі мікроконтролера STM32, що забезпечує збір, обробку даних та керування зовнішніми пристроями у режимі реального часу.

**Завдання.**

Розробіть кімнатну метеостанцію. На платі розширення використайте датчик температури LM335Z (U9 на схемі, пін PB1) і датчик вологості DHT11 (U8 на схемі, пін PD11) та модуль барометр, компас GY-625 (HMC5983, BMP180). В UART виводьте температуру, вологість і тиск (BMP180 з плати GY-625). Натиснення на синю кнопку міняє режими роботи – вивід тільки температури, тільки вологості, тільки тиску, все одразу. Дані мають мінятися не довше ніж раз в секунду.

**Код програми:**

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

```
/**
```

```
*****  
*****
```

```
 * @file      : main.c
```

```
 * @brief     : Main program body
```

```
*****  
*****
```

```
 * @attention
```

```
 *
```

```
 * Copyright (c) 2025 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 "cmsis_os.h"

#include <stdio.h>

#include <stdbool.h>

#include <math.h> // Для возможных вычислений тиску

#include <string.h>


#define BMP280_ADDR    (0x76 << 1) // 7-битовая адреса, зсунута для HAL

#define BMP280_REG_TEMP  0xFA
#define BMP280_REG_PRESS 0xF7
#define BMP280_REG_CTRL  0xF4
#define BMP280_REG_CONFIG 0xF5

/* Private includes -----*/

/* USER CODE BEGIN Includes */


/* 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 hi2c1;

UART_HandleTypeDef huart3;

/* Definitions for defaultTask */
osThreadId_t defaultTaskHandle;
const osThreadAttr_t defaultTask_attributes = {
    .name = "defaultTask",
    .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for LedMutex */
osMutexId_t LedMutexHandle;
const osMutexAttr_t LedMutex_attributes = {
    .name = "LedMutex"
};
/* USER CODE BEGIN PV */
bool flag=false;
/* USER CODE END PV */

```

```

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
void StartDefaultTask(void *argument);

/* USER CODE BEGIN PFP */

/* 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_USART3_UART_Init();
MX_ADC1_Init();
MX_I2C1_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();

/* Create the mutex(es) */
/* creation of LedMutex */
//LedMutexHandle = osMutexNew(&LedMutex_attributes);

/* USER CODE BEGIN RTOS_MUTEX */

```

```

/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
/* USER CODE END RTOS_QUEUES */
uint8_t ctrl = 0x27; // вимірювання температури і тиску, нормальний режим
uint8_t config = 0xA0; // 16-бітний тиск, standby 1000ms
HAL_I2C_Mem_Write(&hi2c1, BMP280_ADDR, BMP280_REG_CTRL, 1, &ctrl,
1, HAL_MAX_DELAY);
HAL_I2C_Mem_Write(&hi2c1, BMP280_ADDR, BMP280_REG_CONFIG, 1,
&config, 1, HAL_MAX_DELAY);
/* Create the thread(s) */
/* creation of defaultTask */
defaultTaskHandle = osThreadNew(StartDefaultTask, NULL,
&defaultTask_attributes);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* USER CODE BEGIN RTOS_EVENTS */
/* add events, ... */

```

```

/* USER CODE END RTOS_EVENTS */

/* Start scheduler */
osKernelStart();

/* We should never get here as control is now taken by the scheduler */

/* Infinite loop */
/* USER CODE BEGIN WHILE */

/* 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_SCALE1);

    /** 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_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
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_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) !=
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_DIV4;
    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_11;
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 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 USART3 Initialization Function

\* @param None

\* @retval None

\*/

```
static void MX_USART3_UART_Init(void)
```

```
{
```

```
/* USER CODE BEGIN USART3_Init 0 */
```

```
/* USER CODE END USART3_Init 0 */
```

```
/* USER CODE BEGIN USART3_Init 1 */
```

```
/* USER CODE END USART3_Init 1 */
```

```
huart3.Instance = USART3;
```

```
huart3.Init.BaudRate = 115200;
```

```
huart3.Init.WordLength = UART_WORDLENGTH_8B;
```

```
huart3.Init.StopBits = UART_STOPBITS_1;
```

```
huart3.Init.Parity = UART_PARITY_NONE;
```

```
huart3.Init.Mode = UART_MODE_TX_RX;
```

```
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

```
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
```

```
if (HAL_UART_Init(&huart3) != HAL_OK)
```

```
{
```

```
    Error_Handler();
```

```
}
```

```
/* USER CODE BEGIN USART3_Init 2 */
```

```
/* USER CODE END USART3_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_GPIOE_CLK_ENABLE();
```

```
    __HAL_RCC_GPIOC_CLK_ENABLE();
```

```
    __HAL_RCC_GPIOH_CLK_ENABLE();
```

```
    __HAL_RCC_GPIOA_CLK_ENABLE();
```

```
    __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
    __HAL_RCC_GPIOD_CLK_ENABLE();
```

```
    /*Configure GPIO pin Output Level */
```

```
    HAL_GPIO_WritePin(CS_I2C_SPI_GPIO_Port, CS_I2C_SPI_Pin,  
GPIO_PIN_RESET);
```

```
    /*Configure GPIO pin Output Level */
```

```
    HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port,  
OTG_FS_PowerSwitchOn_Pin, GPIO_PIN_SET);
```

```
    /*Configure GPIO pin Output Level */
```

```
HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin  
|Audio_RST_Pin, GPIO_PIN_RESET);
```

```
/*Configure GPIO pin : CS_I2C_SPI_Pin */
```

```
GPIO_InitStruct.Pin = CS_I2C_SPI_Pin;  
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;  
HAL_GPIO_Init(CS_I2C_SPI_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
```

```
GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;  
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;  
HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : PDM_OUT_Pin */
```

```
GPIO_InitStruct.Pin = PDM_OUT_Pin;  
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;  
GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;  
HAL_GPIO_Init(PDM_OUT_GPIO_Port, &GPIO_InitStruct);
```

```
/*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 pin : I2S3_WS_Pin */
GPIO_InitStruct.Pin = I2S3_WS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
HAL_GPIO_Init(I2S3_WS_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : SPI1_SCK_Pin SPI1_MISO_Pin SPI1_MOSI_Pin */
GPIO_InitStruct.Pin = SPI1_SCK_Pin|SPI1_MISO_Pin|SPI1_MOSI_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : BOOT1_Pin */
GPIO_InitStruct.Pin = BOOT1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : CLK_IN_Pin */
GPIO_InitStruct.Pin = CLK_IN_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
HAL_GPIO_Init(CLK_IN_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : PD11 */
```

```
GPIO_InitStruct.Pin = GPIO_PIN_11;  
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_1;  
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
```

```
Audio_RST_Pin */
```

```
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin  
|Audio_RST_Pin;  
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;  
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : I2S3_MCK_Pin I2S3_SCK_Pin I2S3_SD_Pin */
```

```
GPIO_InitStruct.Pin = I2S3_MCK_Pin|I2S3_SCK_Pin|I2S3_SD_Pin;  
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;  
GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;  
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : VBUS_FS_Pin */
```

```
GPIO_InitStruct.Pin = VBUS_FS_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(VBUS_FS_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : OTG_FS_ID_Pin OTG_FS_DM_Pin OTG_FS_DP_Pin */
```

```
GPIO_InitStruct.Pin = OTG_FS_ID_Pin|OTG_FS_DM_Pin|OTG_FS_DP_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
GPIO_InitStruct.Alternate = GPIO_AF10_OTG_FS;
```

```
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
```

```
GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : MEMS_INT2_Pin */
```

```
GPIO_InitStruct.Pin = MEMS_INT2_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);
```

```
/* USER CODE BEGIN MX_GPIO_Init_2 */
```

```
/* USER CODE END MX_GPIO_Init_2 */
```

```
}
```

```

/* USER CODE BEGIN 4 */

uint32_t BMP280_ReadRawPressure(void)
{
    uint8_t buf[3];

    HAL_I2C_Mem_Read(&hi2c1, BMP280_ADDR, BMP280_REG_PRESS, 1, buf,
3, HAL_MAX_DELAY);

    return ((uint32_t)buf[0] << 12) | ((uint32_t)buf[1] << 4) | (buf[2] >> 4);
}

uint32_t BMP280_ReadRawTemperature(void)
{
    uint8_t buf[3];

    HAL_I2C_Mem_Read(&hi2c1, BMP280_ADDR, BMP280_REG_TEMP, 1, buf,
3, HAL_MAX_DELAY);

    return ((uint32_t)buf[0] << 12) | ((uint32_t)buf[1] << 4) | (buf[2] >> 4);
}

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
 * @brief Function implementing the defaultTask thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
    char buf[640];

```

```

ADC_ChannelConfTypeDef sConfig;
bool sw1Flag=false;
uint32_t lastTime=0, curTime =0;
int count=0;
for(;;)
{
    curTime = HAL_GetTick();

    uint32_t rawPress = BMP280_ReadRawPressure();
    float pressure = rawPress / 256.0f; // приблизно в Па
    // ---- Температура (PB1 = ADC_CHANNEL_9) ----
    sConfig.Channel = ADC_CHANNEL_9;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    uint32_t adcTemp = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);

    float voltageT = (adcTemp / 4095.0f) * 3.3f;
    float temperatureC = (voltageT * 100.0f)-140;

    // ---- Вологість (ADC_CHANNEL_11) ----
    sConfig.Channel = ADC_CHANNEL_11;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);
}

```

```

HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 100);
uint32_t adcHum = HAL_ADC_GetValue(&hadc1);
HAL_ADC_Stop(&hadc1);

float voltageH = (adcHum / 4095.0f) * 3.3f;
float humidity = voltageH * 30.0f; // замінити формулою для свого сенсора

// ---- Вивід обох значень ----
// Дебаунс 50 мс
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET &&
sw1Flag == false && (curTime - lastTime) >= 50)
{
    count++;
    lastTime = curTime;
    sw1Flag = true;
}

if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET &&
sw1Flag == true && (curTime - lastTime) >= 50)
{
    lastTime = curTime;
    sw1Flag = false;
}

if(count==0)
{
    int len = snprintf(buf, sizeof(buf),

```

```
"\033[2J\033[HTe: %.2f C",  
temperatureC);
```

```
        HAL_UART_Transmit(&huart3, (uint8_t*)buf, len,  
HAL_MAX_DELAY);
```

```
    }
```

```
    if(count==1)
```

```
    {
```

```
        int len = snprintf(buf, sizeof(buf),
```

```
        "\033[2J\033[HHum: %.2f%%",
```

```
        humidity);
```

```
    HAL_UART_Transmit(&huart3, (uint8_t*)buf, len, 50);
```

```
    }
```

```
    if(count==2)
```

```
    {
```

```
        int len = snprintf(buf, sizeof(buf),
```

```
        "\033[2J\033[HP: %.2f Pa",
```

```
        pressure);
```

```
    HAL_UART_Transmit(&huart3, (uint8_t*)buf, len, 50);
```

```
    }
```

```
    if(count==3)
```

```
    {
```

```
        int len = snprintf(buf, sizeof(buf),
```

```
        "\033[2J\033[HTe: %.2f C Hum: %.2f%% P: %.2f Pa",
```

```
        temperatureC, humidity, pressure);
```

```
    HAL_UART_Transmit(&huart3, (uint8_t*)buf, len, 50);
```

```
    }
```

```
    if(count>=4)
```

```

    {
        count=0;
    }
    memset(buf, 0, sizeof(buf));
    HAL_Delay(100); // затримка 100 мс
}
}

/**
 * @brief Period elapsed callback in non blocking mode
 * @note This function is called when TIM1 interrupt took place, inside
 * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
 * a global variable "uwTick" used as application time base.
 * @param htim : TIM handle
 * @retval None
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* USER CODE BEGIN Callback 0 */

    /* USER CODE END Callback 0 */
    if (htim->Instance == TIM1)
    {
        HAL_IncTick();
    }
    /* USER CODE BEGIN Callback 1 */

    /* USER CODE END Callback 1 */
}

```

```

/**
 * @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 */

```

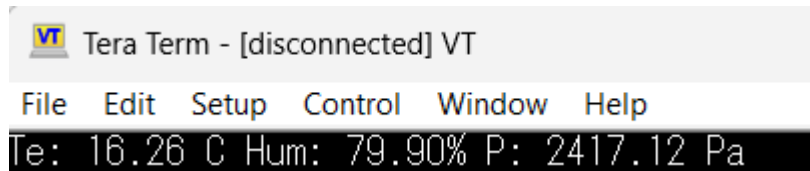


Рис. 1. Результат роботи

Додаткове завдання.

Потрібно реалізувати вивід через ведення команд.

Код:

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

```
/**
```

```
*****  
*****
```

```
 * @file      : main.c
```

```
 * @brief     : Main program body
```

```
*****  
*****
```

```
 * @attention
```

```
 *
```

```
 * Copyright (c) 2025 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 "cmsis_os.h"

#include <stdio.h>

#include <stdbool.h>

#include <math.h> // Для можливих обчислень тиску

#include <string.h>


#define BMP280_ADDR    (0x76 << 1) // 7-бітова адреса, зсунута для HAL
#define BMP280_REG_TEMP  0xFA
#define BMP280_REG_PRESS 0xF7
#define BMP280_REG_CTRL  0xF4
#define BMP280_REG_CONFIG 0xF5

/* Private includes -----*/
/* USER CODE BEGIN Includes */

/* 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 hi2c1;

UART_HandleTypeDef huart3;

/* Definitions for defaultTask */
osThreadId_t defaultTaskHandle;
const osThreadAttr_t defaultTask_attributes = {
    .name = "defaultTask",
    .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for LedMutex */
osMutexId_t LedMutexHandle;
const osMutexAttr_t LedMutex_attributes = {
    .name = "LedMutex"
};
/* USER CODE BEGIN PV */
bool flag=false;
/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_ADC1_Init(void);

```

```

static void MX_I2C1_Init(void);
void StartDefaultTask(void *argument);

/* USER CODE BEGIN PFP */

/* 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_USART3_UART_Init();
MX_ADC1_Init();
MX_I2C1_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();
/* Create the mutex(es) */
/* creation of LedMutex */
//LedMutexHandle = osMutexNew(&LedMutex_attributes);

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */

```

```

/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
/* USER CODE END RTOS_QUEUES */

uint8_t ctrl = 0x27; // вимірювання температури і тиску, нормальний режим
uint8_t config = 0xA0; // 16-бітний тиск, standby 1000ms

HAL_I2C_Mem_Write(&hi2c1, BMP280_ADDR, BMP280_REG_CTRL, 1, &ctrl,
1, HAL_MAX_DELAY);

HAL_I2C_Mem_Write(&hi2c1, BMP280_ADDR, BMP280_REG_CONFIG, 1,
&config, 1, HAL_MAX_DELAY);

/* Create the thread(s) */

/* creation of defaultTask */

defaultTaskHandle = osThreadNew(StartDefaultTask, NULL,
&defaultTask_attributes);

/* USER CODE BEGIN RTOS_THREADS */

/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* USER CODE BEGIN RTOS_EVENTS */

/* add events, ... */
/* USER CODE END RTOS_EVENTS */

/* Start scheduler */
osKernelStart();

```

```

/* We should never get here as control is now taken by the scheduler */

/* Infinite loop */
/* USER CODE BEGIN WHILE */

/* 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_SCALE1);

    /** 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_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

```

```

RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
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_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) !=
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_DIV4;
    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_11;
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 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 USART3 Initialization Function
```

```
 * @param None
```

```
 * @retval None
```

```
 */
```

```
static void MX_USART3_UART_Init(void)
```

```

{

/* USER CODE BEGIN USART3_Init 0 */

/* USER CODE END USART3_Init 0 */

/* USER CODE BEGIN USART3_Init 1 */

/* USER CODE END USART3_Init 1 */
huart3.Instance = USART3;
huart3.Init.BaudRate = 115200;
huart3.Init.WordLength = UART_WORDLENGTH_8B;
huart3.Init.StopBits = UART_STOPBITS_1;
huart3.Init.Parity = UART_PARITY_NONE;
huart3.Init.Mode = UART_MODE_TX_RX;
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart3) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART3_Init 2 */

/* USER CODE END USART3_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_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(CS_I2C_SPI_GPIO_Port, CS_I2C_SPI_Pin,
GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port,
OTG_FS_PowerSwitchOn_Pin, GPIO_PIN_SET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
|Audio_RST_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin : CS_I2C_SPI_Pin */
    GPIO_InitStruct.Pin = CS_I2C_SPI_Pin;

```

```
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(CS_I2C_SPI_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
```

```
GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : PDM_OUT_Pin */
```

```
GPIO_InitStruct.Pin = PDM_OUT_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
HAL_GPIO_Init(PDM_OUT_GPIO_Port, &GPIO_InitStruct);
```

```
/*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 pin : I2S3_WS_Pin */
```

```
GPIO_InitStruct.Pin = I2S3_WS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
HAL_GPIO_Init(I2S3_WS_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : SPI1_SCK_Pin SPI1_MISO_Pin SPI1_MOSI_Pin */
GPIO_InitStruct.Pin = SPI1_SCK_Pin|SPI1_MISO_Pin|SPI1_MOSI_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : BOOT1_Pin */
GPIO_InitStruct.Pin = BOOT1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : CLK_IN_Pin */
GPIO_InitStruct.Pin = CLK_IN_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
HAL_GPIO_Init(CLK_IN_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : PD11 */
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

```
GPIO_InitStruct.Pin = GPIO_PIN_1;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
```

```
Audio_RST_Pin */
```

```
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
```

```
|Audio_RST_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : I2S3_MCK_Pin I2S3_SCK_Pin I2S3_SD_Pin */
```

```
GPIO_InitStruct.Pin = I2S3_MCK_Pin|I2S3_SCK_Pin|I2S3_SD_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
```

```
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
/*Configure GPIO pin : VBUS_FS_Pin */
```

```
GPIO_InitStruct.Pin = VBUS_FS_Pin;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
HAL_GPIO_Init(VBUS_FS_GPIO_Port, &GPIO_InitStruct);
```

```
/*Configure GPIO pins : OTG_FS_ID_Pin OTG_FS_DM_Pin OTG_FS_DP_Pin */
GPIO_InitStruct.Pin = OTG_FS_ID_Pin|OTG_FS_DM_Pin|OTG_FS_DP_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF10_OTG_FS;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

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

```
/*Configure GPIO pin : MEMS_INT2_Pin */
GPIO_InitStruct.Pin = MEMS_INT2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);
```

```
/* USER CODE BEGIN MX_GPIO_Init_2 */
```

```
/* USER CODE END MX_GPIO_Init_2 */
```

```
}
```

```
/* USER CODE BEGIN 4 */
```

```
uint32_t BMP280_ReadRawPressure(void)
```

```
{
```

```
    uint8_t buf[3];
```

```

    HAL_I2C_Mem_Read(&hi2c1, BMP280_ADDR, BMP280_REG_PRESS, 1, buf,
3, HAL_MAX_DELAY);

    return ((uint32_t)buf[0] << 12) | ((uint32_t)buf[1] << 4) | (buf[2] >> 4);
}

```

```

uint32_t BMP280_ReadRawTemperature(void)
{
    uint8_t buf[3];

    HAL_I2C_Mem_Read(&hi2c1, BMP280_ADDR, BMP280_REG_TEMP, 1, buf,
3, HAL_MAX_DELAY);

    return ((uint32_t)buf[0] << 12) | ((uint32_t)buf[1] << 4) | (buf[2] >> 4);
}

```

```

/* USER CODE END 4 */

```

```

/* USER CODE BEGIN Header_StartDefaultTask */

```

```

/**

```

```

 * @brief Function implementing the defaultTask thread.

```

```

 * @param argument: Not used

```

```

 * @retval None

```

```

 */

```

```

/* USER CODE END Header_StartDefaultTask */

```

```

void StartDefaultTask(void *argument)

```

```

{

```

```

    uint8_t chr;

```

```

    char buf[20];

```

```

    int idx = 0;

```

```

    char buf2[64];

```

```

    ADC_ChannelConfTypeDef sConfig;

```

```

    bool sw1Flag=false;

```

```

uint32_t lastTime=0, curTime =0;
int count=0;
for(;;)
{
    curTime = HAL_GetTick();

    uint32_t rawPress = BMP280_ReadRawPressure();
    float pressure = rawPress / 256.0f; // приблизно в Па
    // ---- Температура (PB1 = ADC_CHANNEL_9) ----
    sConfig.Channel = ADC_CHANNEL_9;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 100);
    uint32_t adcTemp = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);

    float voltageT = (adcTemp / 4095.0f) * 3.3f;
    float temperatureC = (voltageT * 100.0f)-140;

    // ---- Вологість (ADC_CHANNEL_11) ----
    sConfig.Channel = ADC_CHANNEL_11;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    HAL_ADC_Start(&hadc1);

```

```

HAL_ADC_PollForConversion(&hadc1, 100);
uint32_t adcHum = HAL_ADC_GetValue(&hadc1);
HAL_ADC_Stop(&hadc1);

float voltageH = (adcHum / 4095.0f) * 3.3f;
float humidity = voltageH * 30.0f; // замінити формулою для свого сенсора

// ---- Вивід обох значень ----
// Дебаунс 50 мс
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
if(HAL_UART_Receive(&huart3, &chr, 1, 100) == HAL_OK)
{

    HAL_UART_Transmit(&huart3, &chr, 1, 100);

    if(chr == '\r' || chr == '\n')
    {
        buf[idx] = 0;
        idx = 0;

        buf[idx] = 0; // завершити рядок
        if(strncmp(buf, "COMMAND ", 8) == 0)
        {
            int led = atoi(buf + 8); // перетворюємо тільки цифри після
пробілу

            // обробка led

            int led = atoi(buf + 8);

```

```

        if(count==0)
        {
            int len = snprintf(buf2, sizeof(buf2),
                                "\033[2J\033[HTe: %.2f C",
                                temperatureC);

            HAL_UART_Transmit(&huart3, (uint8_t*)buf2,
len, HAL_MAX_DELAY);
        }
        if(count==1)
        {
            int len = snprintf(buf2, sizeof(buf2),
                                "\033[2J\033[HHum: %.2f%%",
                                humidity);
            HAL_UART_Transmit(&huart3, (uint8_t*)buf2, len, 50);
        }
        if(count==2)
        {
            int len = snprintf(buf2, sizeof(buf2),
                                "\033[2J\033[HP: %.2f Pa",
                                pressure);
            HAL_UART_Transmit(&huart3, (uint8_t*)buf2, len, 50);
        }
        if(count==3)
        {
            int len = snprintf(buf2, sizeof(buf2),
                                "\033[2J\033[HTe: %.2f C Hum: %.2f%% P:
%.2f Pa",
                                temperatureC, humidity, pressure);
            HAL_UART_Transmit(&huart3, (uint8_t*)buf2, len, 50);

```

```

    }

}

    idx = 0;      // скидаємо індекс
    memset(buf, 0, sizeof(buf)); // очищаємо буфер

    const char newline[] = "\r\n";
    HAL_UART_Transmit(&huart3, (uint8_t*)newline, 2, 100);
}
else
{
    buf[idx++] = chr;
    if(idx >= sizeof(buf)-1) idx = sizeof(buf)-1;
}
}

}

}

/**
 * @brief Period elapsed callback in non blocking mode
 * @note This function is called when TIM1 interrupt took place, inside
 * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
 * a global variable "uwTick" used as application time base.
 * @param htim : TIM handle
 * @retval None
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{

```

```

/* USER CODE BEGIN Callback 0 */

/* USER CODE END Callback 0 */
if (htim->Instance == TIM1)
{
    HAL_IncTick();
}
/* USER CODE BEGIN Callback 1 */

/* USER CODE END Callback 1 */
}

/**
 * @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 */

```

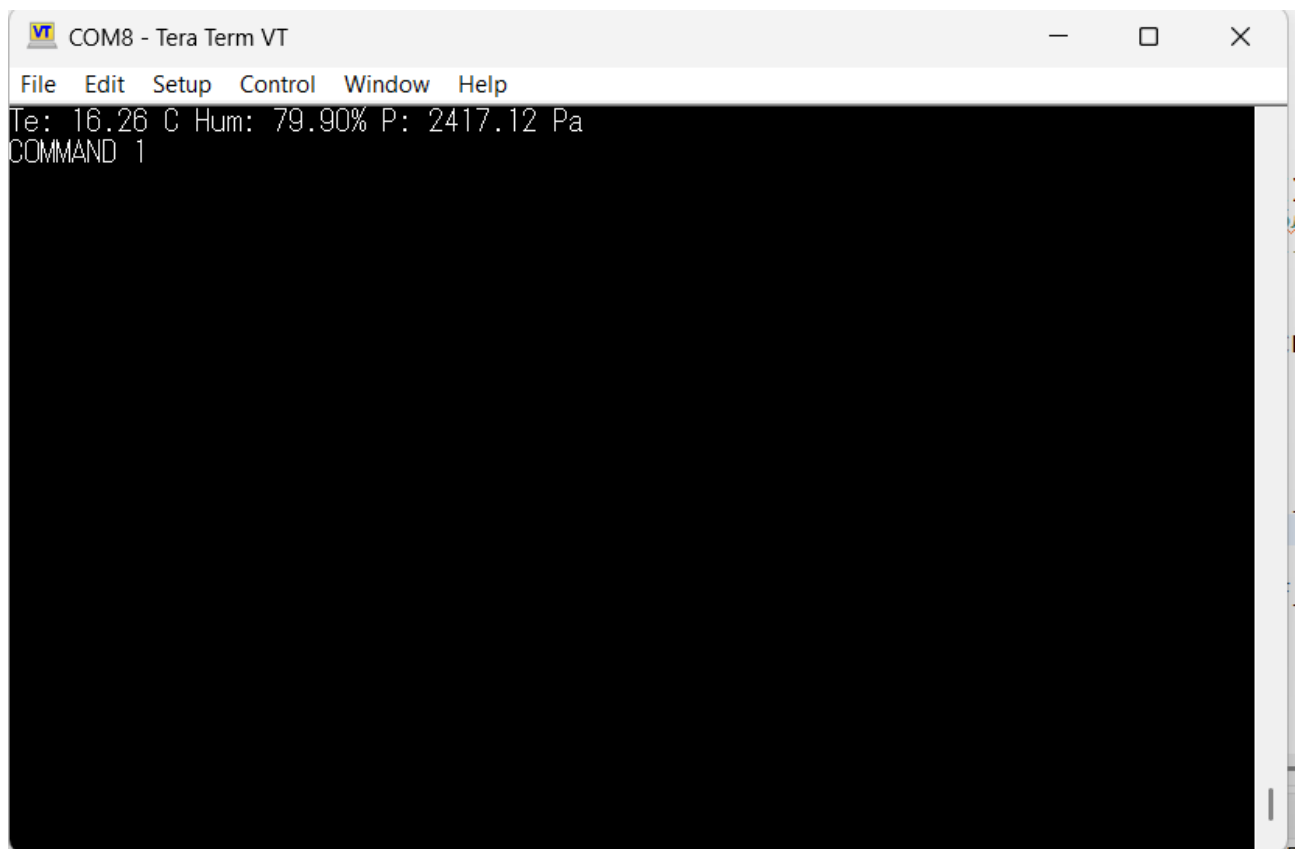


Рис. 2. Результат роботи

### Контрольні запитання:

1. Пояснити як працює ваш код.

MCU читає температуру, тиск і вологість через ADC і BMP280 по I2C, обробляє дані і відправляє їх по UART за командою від користувача.

2. Пояснити чому ви використали чи ні RTOS?

Використав RTOS (FreeRTOS), щоб організувати задачу defaultTask і потенційно додавати інші задачі чи синхронізацію, але в цьому прикладі лише одна задача, тож можна було б обійтися без RTOS.

3. Що таке I2C і SPI.

Це послідовні протоколи обміну даними між мікроконтролером і периферією. I2C – двохпровідний (SCL, SDA), SPI – чотирипровідний (MISO, MOSI, SCK, CS).

4. Яка різниця між I2C і SPI.

I2C повільніший, використовує дві лінії, має адресацію пристроїв. SPI швидший, синхронний, використовує окрему лінію CS для кожного пристрою, без вбудованої адресації.

5. Що таке ADC?

Analog-to-Digital Converter – перетворює аналоговий сигнал (наприклад, від сенсора) у цифровий для обробки мікроконтролером.

6. Що таке DMA?

Direct Memory Access – дозволяє периферії (ADC, UART, SPI) передавати дані у пам'ять без участі CPU, економлячи ресурси.

7. Пояснити як працюють ваші сенсори/актуатори.

BMP280 – вимірює температуру і тиск по I2C. Потенціометр і датчик вологості – через ADC читаються аналогові напруги, які конвертуються в фізичні величини. Актуатори (світлодіоди) керуються GPIO.

## **Висновок:**

На лабораторній роботі я ознайомився з розробкою програмного забезпечення для взаємодії з сенсорами та актуаторами на основі мікроконтролера STM32, що забезпечує збір, обробку даних та керування зовнішніми пристроями у режимі реального часу.