Section I: Project Title and Student Detail

Field	Details
Project Title	Smart Home HVAC Controller
Student Name(s)	Anjali Bhamare(A-11), Sanket Jagtap(A-57), Rachana Mahangare(B-103)
Department	Electronics And Telecommunication
College Name	Aissms Ioit
Project Guide	Mr. Parag
Submission Date	7-11-25

Section II: Project Summary

Field	Details
Objective	To design and implement a smart temperature monitoring system using LM35 sensor and buzzer alert functionality.
Scope	The system reads temperature data, alerts users via buzzer and UART terminal.
Tools & Technologies	LM35 sensor, UART, buzzer, C programming, serial terminal, Free RTOS, stm32f446re
Expected Outcomes	Accurate temperature readings, real-time alerts, fault detection, and robust UART communication.
Real-World Relevance	Useful for industrial temperature monitoring, server rooms, and home automation systems.

This project focuses on building a smart temperature monitoring system using an LM35 sensor and STM32F446RE microcontroller. It continuously reads temperature data and displays it via UART, while also triggering a buzzer alert if the temperature exceeds a defined threshold. Designed for real-time performance and reliability, it finds applications in industrial safety, server rooms, and smart homes. The project successfully integrates ADC, UART, and FreeRTOS to achieve accurate temperature readings, responsive alerts, and efficient multitasking, while the Tickless Idle feature ensures low-power operation during idle periods.

Section III: Agile Process Documentation

★ Overview Paragraph

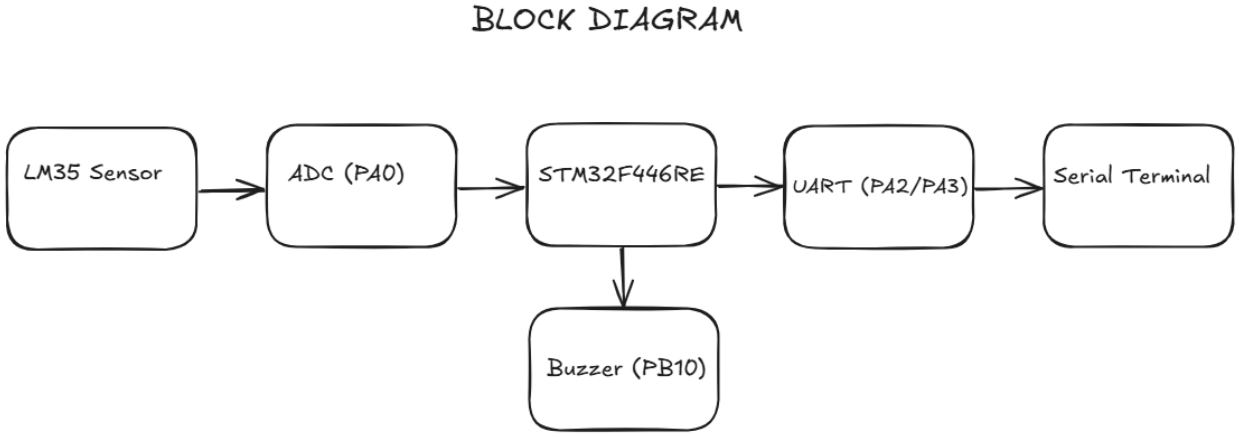
The project followed an Agile development approach, structured around iterative sprints and evolving user requirements. Each sprint focused on delivering specific features through well-defined user stories. Tasks were broken down into actionable items with measurable acceptance criteria to ensure quality and traceability. This section documents the core user stories, their implementation goals, and validation benchmarks.

User Story ID	User Story	Action Items	Acceptance Criteria
US1	As a developer, I want to read sensor data so that I can monitor temperature.	• Interface **LM35 sensor** to STM32F446RE using **ADC pin PA0**. • Configure ADC to read analog voltage (0 - 3.3 V). • Implement **SensorTask** in FreeRTOS to sample temperature every 1 second. • Convert ADC value to °C and send it to a message queue.	Sensor reads every 3s, values are accurate ±1°C
US2	As a user, I want temperature readings to be sent over UART so that I can monitor system status.	• Implement **CommTask** using UART2 • Use mutex for safe UART access. • Format readings as “Temp: XX °C	Time: XXXX ms

US3	As a tester, I want a buzzer alert for high temperature so that I can take timely action.	• Connect buzzer to GPIO pin **PB10** and configure as output. • Implement MonitorTask to receive temperature data from queue. • Activate buzzer if temperature > 26 °C and deactivate if < 26 °C.	Buzzer turns ON when temp > 26 °C.

IV. System Design

1. Block Diagram:



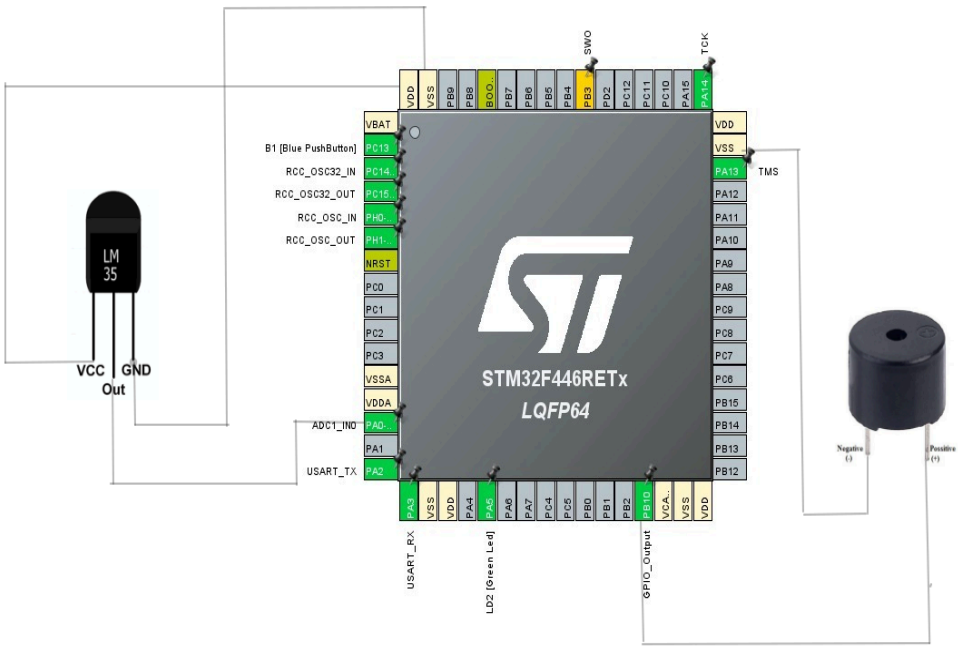
2. Hardware Components Table:

Component	Specification	Purpose
STM32F446RE	ARM Cortex-M4, 180 MHz, 12-bit ADC	Core processing, ADC, UART, GPIO control
LM35 Sensor	Analog temperature sensor	Measures ambient temperature
Buzzer	5V active buzzer.	Alerts on high temperature

3. Pin Mapping Summary:

Component	Signal	STM32 Pin
UART TX	Serial Out	PA2 (USART2_TX)
UART RX	Serial In	PA3 (USART2_RX)
LM35 Sensor	Analog Out	PA0 (ADC Channel 0)
Buzzer	Digital Out	PB0 (GPIO Output) or fault

- **CIRCUIT DIAGRAM:**

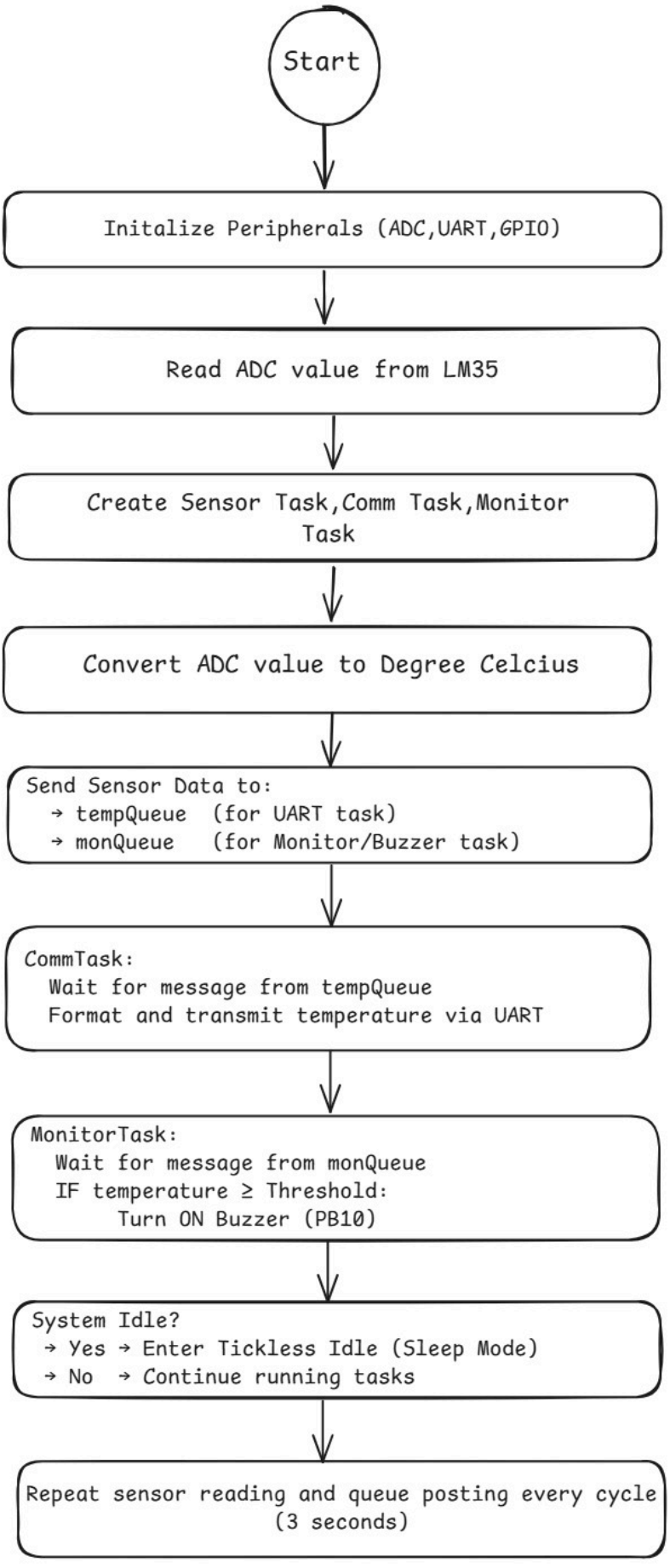


V. Implementation

Calculations:

temperature = ((adc_value * 330) / 4095);

FlowChart:


Start
Initalize Peripherals (ADC,UART,GPIO)
Read ADC value from LM35
Create Sensor Task,Comm Task,Monitor Task
Convert ADC value to Degree Celcius
Send Sensor Data to:
→ tempQueue (for UART task)
→ monQueue (for Monitor/Buzzer task)
CommTask:
Wait for message from tempQueue
Format and transmit temperature via UART
MonitorTask:
Wait for message from monQueue
IF temperature ≥ Threshold:
Turn ON Buzzer (PB10)
System Idle?
→ Yes → Enter Tickless Idle (Sleep Mode)
→ No → Continue running tasks
Repeat sensor reading and queue posting every cycle (3 seconds)

VI. Testing & Results:

Test No	Test Name	What To Do	Expected Result	Passed Test Cases
1	Sensor Reading	Power board, open UART terminal	Temperature readings shown every 1 sec	Passed
2	FIFO Check	Warm LM35 slowly	UART shows increasing temperature in correct increments	Passed
3	High Temp Alarm	Heat LM35 (above 26°C)	Buzzer ON immediately	Passed
4	Normal Temp	Cool LM35 (below 26°C)	Buzzer OFF	Passed
5	UART Message Queue	Add 5 min continuously	No message missing or jumbled	Passed
6	Low-Power Mode	Leave system idle	Power or current decreases, still reads values every 1 sec	Passed
7	Restart Test	Press reset	System starts again, UART resumes readings	Passed
8	Fault Test	Disconnect LM35	Shows “SENSOR FAULT” or buzzer ON	Test not passed

Logs & ScreenShots:

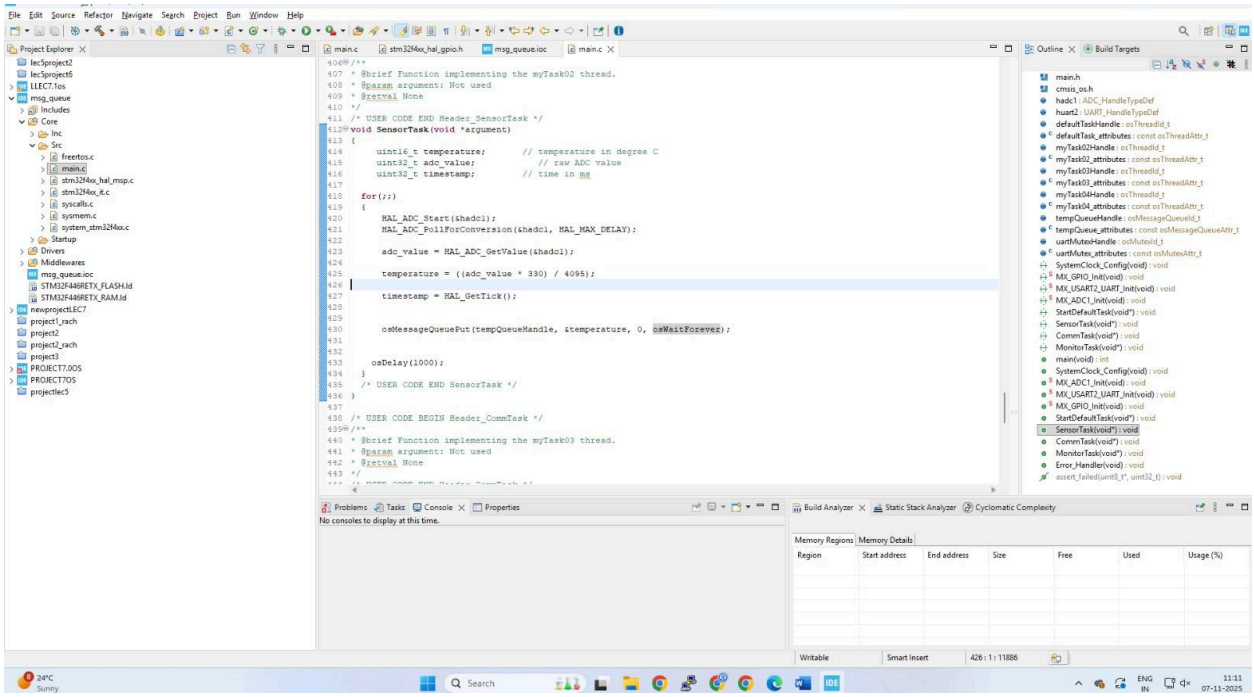


Figure 1: Sensor Task Implementation

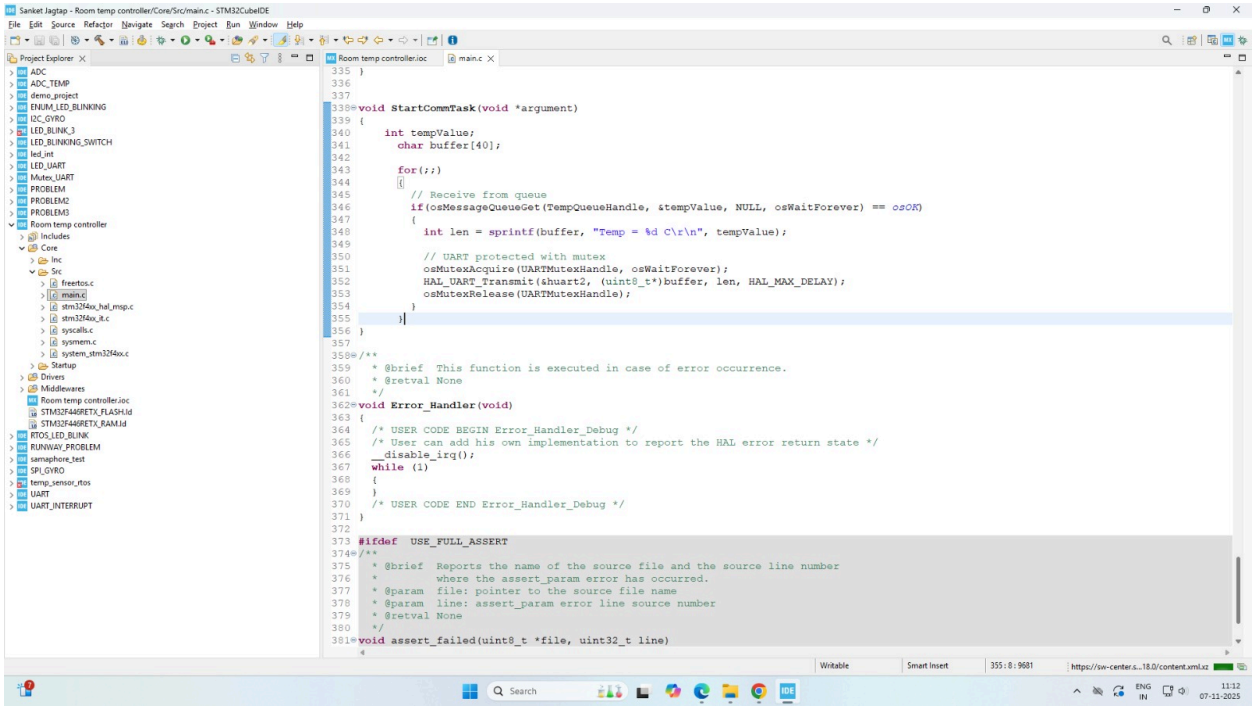


Figure 2: Communication Task Implementation

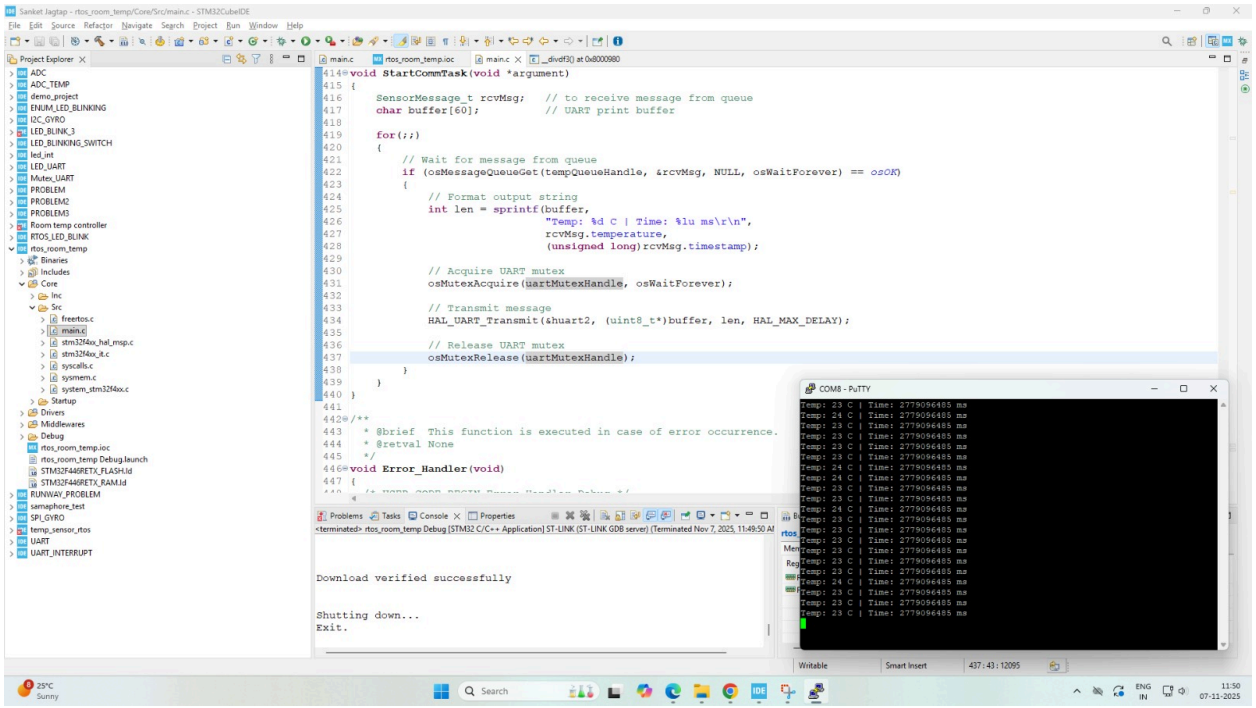


Figure 3: Monitor Task and testing

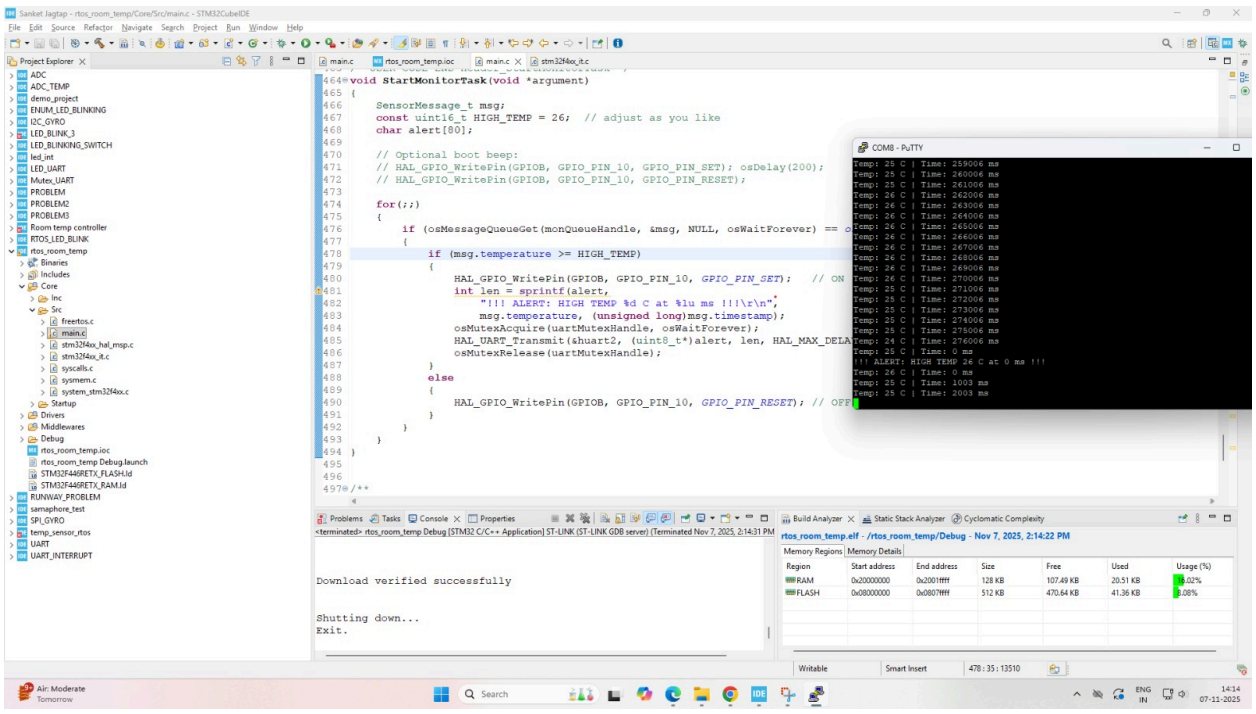


Figure 4: buzzer testing

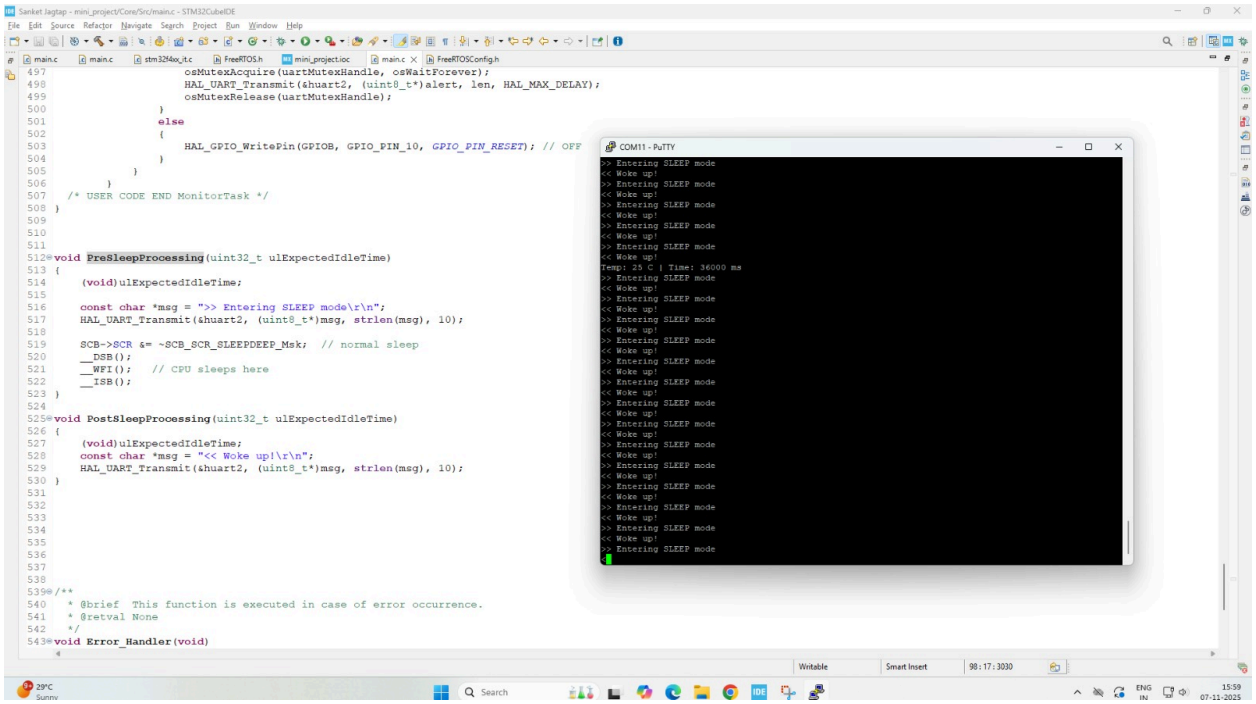


Figure 5 :Final Output

VII. Sprint Review and Retrospective:

Completed Stories:

The following user stories were successfully implemented and validated during the sprint cycles:

- **US1**: Sensor data acquisition and sending to queue
- **US2**: Sending temperature through UART to terminal
- **US3**: Buzzer turns on when temperature exceeds threshold

---

Unfinished Stories:

The following story was partially implemented or deferred due to time/resource constraints:

- Bluetooth module integration for wireless monitoring
  Reason: Hardware delivery delays and UART port conflicts with existing serial terminal
- Fault detection by removing LM35 pin

What Went Well:

- Modular code structure with clear separation of tasks (SensorTask, CommTask, MonitorTask)
- Successful integration of FreeRTOS with mutex and queue handling
- Accurate temperature readings and responsive buzzer alerts

What to Improve:

- Add Bluetooth support with UART multiplexing or DMA
- Optimize power consumption further using sleep modes beyond tickless idle
- Improve ADC sampling rate for faster response in critical environments
- Enhance documentation with inline comments and header file summaries
- Include automated test scripts for regression testing

VIII. Discussion – Challenges Faced and Solutions

Challenges & Solutions:

- **Challenge 1: UART Conflicts with Debugging Tools**
  Solution: Used USART2 (PA2/PA3) for UART communication and disabled ST-Link UART during runtime to avoid conflicts.
- **Challenge 2: LM35 Sensor Disconnection Handling**
  Solution: Implemented fault detection logic that checks for ADC value = 0 and sends "INVALID" to Putty terminal.
- **Challenge 3: Task Synchronization in FreeRTOS**
  Solution: Introduced UART_Mutex to prevent simultaneous access to UART from multiple tasks.
- **Challenge 4: Power Optimization**
  Solution: Integrated tickless idle mode and idle hook to reduce power consumption during system inactivity.

IX. Conclusion – Summary of Achievements and Success Criteria

The project successfully achieved its core objectives of real-time temperature monitoring, fault detection, and alert generation using STM32F446RE and LM35. The system was modular, responsive, and power-efficient.

Achievements:

- Accurate temperature readings via ADC
- Real-time UART output to Putty terminal
- Buzzer alerts for threshold breaches
- Fault detection for sensor disconnection
- RTOS-based task management with mutex and queue
- Low-power behavior using tickless idle

XI.Appendix - Full source code, extra test data:

```
/* 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<string.h>

/* 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;

UART_HandleTypeDef huart2;

/* Definitions for defaultTask */
osThreadId_t defaultTaskHandle;
const osThreadAttr_t defaultTask_attributes = {
  .name = "defaultTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for myTask02 */
osThreadId_t myTask02Handle;
const osThreadAttr_t myTask02_attributes = {
  .name = "myTask02",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityBelowNormal,
};
/* Definitions for myTask03 */
osThreadId_t myTask03Handle;
const osThreadAttr_t myTask03_attributes = {
  .name = "myTask03",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityAboveNormal,
};
/* Definitions for tempQueue */
osMessageQueueId_t tempQueueHandle;
const osMessageQueueAttr_t tempQueue_attributes = {
  .name = "tempQueue"
};
/* Definitions for monQueue */
osMessageQueueId_t monQueueHandle;
const osMessageQueueAttr_t monQueue_attributes = {
  .name = "monQueue"
};
/* Definitions for uartMutex */
osMutexId_t uartMutexHandle;
const osMutexAttr_t uartMutex_attributes = {
  .name = "uartMutex"
};
```

```
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
void SensorTask(void *argument);
void CommTask(void *argument);
void MonitorTask(void *argument);
void PreSleepProcessing(uint32_t ulExpectedIdleTime);
void PostSleepProcessing(uint32_t ulExpectedIdleTime);
void vApplicationSleep(TickType_t xExpectedIdleTime);




/* 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
  */

typedef struct
{
  uint16_t temperature;
  uint32_t timestamp;
} SensorMessage_t;
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_ADC1_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();
/* Create the mutex(es) */
/* creation of uartMutex */
uartMutexHandle = osMutexNew(&uartMutex_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 */

/* Create the queue(s) */
/* creation of tempQueue */
```

```
  tempQueueHandle = osMessageQueueNew (16, sizeof(SensorMessage_t), &tempQueue_attributes);

  /* creation of monQueue */
  monQueueHandle = osMessageQueueNew (16, sizeof(SensorMessage_t), &monQueue_attributes);

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

  /* Create the thread(s) */
  /* creation of defaultTask */
  defaultTaskHandle = osThreadNew(SensorTask, NULL, &defaultTask_attributes);

  /* creation of myTask02 */
  myTask02Handle = osThreadNew(CommTask, NULL, &myTask02_attributes);

  /* creation of myTask03 */
  myTask03Handle = osThreadNew(MonitorTask, NULL, &myTask03_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 */
  while (1)
  {
    /* 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};

 /** Configure the main internal regulator output voltage
 */
 __HAL_RCC_PWR_CLK_ENABLE();
 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

 /** 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;
 RCC_OscInitStruct.PLL.PLLM = 16;
 RCC_OscInitStruct.PLL.PLLN = 336;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
 RCC_OscInitStruct.PLL.PLLQ = 2;
 RCC_OscInitStruct.PLL.PLLR = 2;
 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_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != 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_0;
  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 USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
```

```
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);

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

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PB10 */
  GPIO_InitStruct.Pin = GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

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

```
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/* USER CODE BEGIN Header_SensorTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_SensorTask */
void SensorTask(void *argument)
{
  /* USER CODE BEGIN 5 */
        SensorMessage_t msg;
                uint32_t adc_value;

                for(;;)
                {
                   HAL_ADC_Start(&hadc1);
                   HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

                   adc_value = HAL_ADC_GetValue(&hadc1);

                   msg.temperature = ((adc_value * 330) / 4095);
                   msg.timestamp = osKernelGetTickCount();

                   // ✅ Send to CommTask
                   osMessageQueuePut(tempQueueHandle, &msg, 0, 0);

                   // ✅ Send to MonitorTask
                   osMessageQueuePut(monQueueHandle, &msg, 0, 0);

                   osDelay(3000);
                }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_CommTask */
/**
* @brief Function implementing the myTask02 thread.
* @param argument: Not used
```

```
* @retval None
*/
/* USER CODE END Header_CommTask */
void CommTask(void *argument)
{
  /* USER CODE BEGIN CommTask */
        SensorMessage_t rcvMsg;   // to receive message from queue
          char buffer[60];          // UART print buffer

          for(;;)
          {
            // Wait for message from queue
            if (osMessageQueueGet(tempQueueHandle, &rcvMsg, NULL, osWaitForever) == osOK)
            {
              // Format output string
              int len = sprintf(buffer,
                        "Temp: %d C | Time: %lu ms\r\n",
                        rcvMsg.temperature,
                        (unsigned long)rcvMsg.timestamp);

              // Acquire UART mutex
              osMutexAcquire(uartMutexHandle, osWaitForever);

              // Transmit message
              HAL_UART_Transmit(&huart2, (uint8_t*)buffer, len, HAL_MAX_DELAY);

              // Release UART mutex
              osMutexRelease(uartMutexHandle);
            }
          }
  /* USER CODE END CommTask */
}

/* USER CODE BEGIN Header_MonitorTask */
/**
* @brief Function implementing the myTask03 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_MonitorTask */
void MonitorTask(void *argument)
{
  /* USER CODE BEGIN MonitorTask */
        SensorMessage_t msg;
```

```
            const uint16_t HIGH_TEMP = 26;  // adjust as you like
            char alert[80];

            // Optional boot beep:
            // HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET); osDelay(200);
            // HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);

            for(;;)
            {
               if (osMessageQueueGet(monQueueHandle, &msg, NULL, osWaitForever) == osOK)
               {
                  if (msg.temperature >= HIGH_TEMP)
                  {
                     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);   // ON
                     int len = sprintf(alert,
                        "!!! ALERT: HIGH TEMP %d C at %lu ms !!!\r\n",
                        msg.temperature, (unsigned long)msg.timestamp);
                     osMutexAcquire(uartMutexHandle, osWaitForever);
                     HAL_UART_Transmit(&huart2, (uint8_t*)alert, len, HAL_MAX_DELAY);
                     osMutexRelease(uartMutexHandle);
                  }
                  else
                  {
                     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // OFF
                  }
               }
            }
  /* USER CODE END MonitorTask */
}



void PreSleepProcessing(uint32_t ulExpectedIdleTime)
{
   (void)ulExpectedIdleTime;

   const char *msg = ">> Entering SLEEP mode\r\n";
   HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 10);

   SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;  // normal sleep
   __DSB();
   __WFI();   // CPU sleeps here
   __ISB();
}
```

```
void PostSleepProcessing(uint32_t ulExpectedIdleTime)
{
   (void)ulExpectedIdleTime;
   const char *msg = "<< Woke up!\r\n";
   HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 10);
}


/**
  * @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 */
```