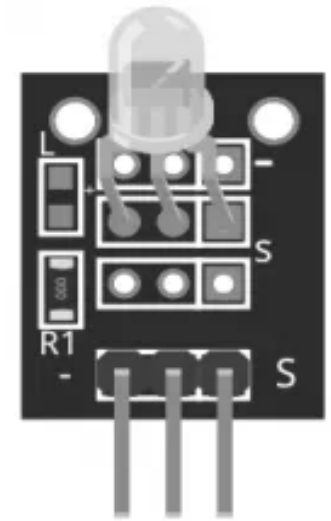
MINI TWO COLOUR LED
KY-029 SENSOR



Project done by
B.vijaya lakshmi

TABLE OF CONTENTS :

- Project summary
- Features of Project
- Project Introduction
- Working
- Advantages of sensor
- Hardware used and their specifications
- Why UART based communication
- Connection Diagram
- Project code
- Rightech iot cloud
- References

1.PROJECT SUMMARY

The Internet of Things (IoT) is a rapidly growing field that has the potential to revolutionise the way we interact with the world around us. One of the most promising applications of IoT is in the area of environmental monitoring. By deploying IoT devices in various environments, we can collect real-time data on environmental conditions such as temperature, and humidity. This data can then be used to track changes in environmental conditions over time, identify potential problems, and take corrective action.

This project involves 4 stages:

- First stage is to blink two colours by interfacing the sensor with a microcontroller.
- The second stage is to obtain the output state and send it to the right tech cloud through the wifi module.
- Third stage is to send the output state of the sensor from the microcontroller to the Rugged Board.
- Fourth stage is to send the data received by the Rugged Board to the right tech cloud using the wifi module.

The system will be deployed in a real-world environment to test its performance. The deployment results will be used to assess the system's feasibility and identify any potential improvements.

The proposed system has the potential to be a valuable tool for tracking and monitoring environmental conditions. The system is relatively low-cost and

easy to deploy, making it a feasible option for a wide range of applications. The system is also scalable, making it possible to deploy it in a variety of environments.

The results of this project will contribute to the body of knowledge on IoTbased environmental monitoring systems. The project will also provide a valuable case study for other researchers and developers who are interested in developing IoT-based ecological monitoring systems.

2.Features of project :

- This sensor is used to blink two colours.
- In the First stage we are interfacing the mini two colour sensor with stm32f411re microcontroller to obtain the output.
- The second stage is to obtain the output state using uart and send it to the right tech cloud through the wifi module.
- Third stage is to send the output state of the sensor from the microcontroller to the Rugged Board using uart.
- Fourth stage is to send the data received by the Rugged Board to the right tech cloud using the wifi module.

3.Introduction to two-colour sensor :

Two-colour sensors are devices designed to detect and measure the intensity of light in two specific wavelengths or colours. These sensors are commonly used in various applications, ranging from industrial processes to consumer electronics. The basic principle behind two-colour sensors involves comparing the amount of light absorbed or transmitted at two distinct wavelengths, allowing for accurate measurements and analysis. Here's an overview of the key components and applications of two-colour sensors:

Key Components :

1. **Light Source :**
    - Two-colour sensors typically incorporate a light source that emits light in two specific wavelengths. This source could be LEDs (Light-Emitting Diodes) or other types of light emitters, depending on the application.
2. **Photodetectors :**
    - The sensors include photodetectors or photoresistors capable of capturing light at the specified wavelengths. These detectors convert light energy into electrical signals.
3. **Filtering Mechanism :**
    - Filters are employed to isolate the desired wavelengths. These filters allow only specific colours to reach the photodetectors, enhancing the sensor's accuracy and reliability.
4. **Signal Processing Unit :**
    - A signal processing unit analyzes the electrical signals generated by the photodetectors. It compares the intensities of light at the two wavelengths, providing valuable data for further processing or control.

4.Working Principle :

The working principle of two-colour sensors involves measuring the differential response of the system to the two distinct wavelengths. By comparing the light intensity at these two points, the sensor can gather information about the composition or properties of the material being analyzed.

Applications:

1. **Colour Sensing:**
    - Two-colour sensors are widely used in applications where colour discrimination is critical, such as in colour sorting systems for manufacturing processes.
2. **Chemical Analysis:**
    - In chemical analysis, two-colour sensors can be employed to identify and quantify specific substances based on their absorption or reflection properties at different wavelengths.
3. **Industrial Automation:**
    - These sensors find applications in industrial automation for tasks like quality control, where they can be used to detect defects or variations in products.
4. **Environmental Monitoring:**
    - Two-colour sensors can be utilised in environmental monitoring to measure parameters like water quality, where different wavelengths can provide information about specific pollutants or substances.
5. **Medical Devices:**
    - In medical devices, two-colour sensors may be used for diagnostic purposes, such as in blood glucose monitoring or oxygen saturation measurement.

5.Advantages:

1. **Precision:**

    - Two-colour sensors offer higher precision compared to single-wavelength sensors, as they provide a more comprehensive analysis of the light interaction with the material.
2. **Versatility:**
    - These sensors can be adapted to various applications due to their ability to measure different properties based on the selected wavelengths.
3. **Reliability:**
    - The use of specific wavelengths and filtering mechanisms enhances the reliability and accuracy of measurements.

In conclusion, two-colour sensors play a crucial role in a wide range of applications by providing a more nuanced and detailed analysis of light interactions with materials. Their precision and versatility make them valuable tools in fields where accurate colour or composition measurements are essential.

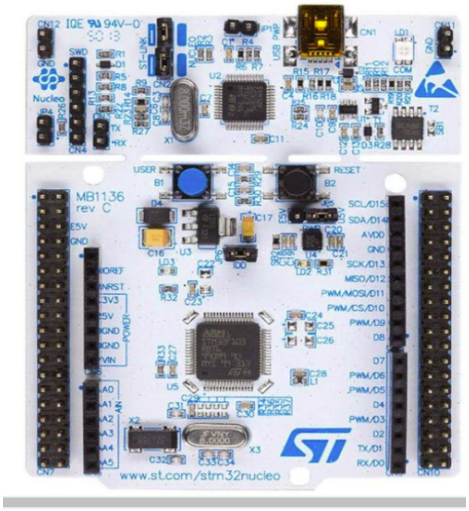
6.HARDWARE USED :

- MINI TWO COLOUR SENSOR
- STM32F411RE MICROCONTROLLER
- RUGGED BOARD
- WIFI MODULE

1.STM32F411RE microcontroller

The STM32F411RE is a high-performance microcontroller based on the ARM Cortex-M4 processor. It has a number of features that make it well-suited for a variety of applications, including:

- 100 MHz CPU clock speed
- 128 KB of RAM
- 512 KB of Flash memory
- Floating point unit (FPU)
- 11 general-purpose timers
- 13 communication interfaces
- USB
- RTC



2.RUGGED BOARD - A5D2X:

- Single board computer

RuggedBoard - A5D2x is a Single Board Computer providing an easy migration path from Microcontroller to Microprocessor. RuggedBoard is enabled with industry Standard Yocto Build Embedded Linux platform and open source libraries for industrial application development.RuggedBoard is an Open source Industrial single board computer powered by ARM Cortex-A5 SoC @500 MHz, implemented with the finest platform for rapid prototyping. The usage of System On Module over a System On Chip is the most rapid way to achieve time to market, curtail development risks for product quantities ranging from a few hundred to thousands.

RuggedBoard- A5D2x consists of Multiple Interfaces such as Ethernet, RS232, CAN, RS485, Digital Input and Digital Output with optically isolated, Standard MikroBus header for Add-On Sensors, Actuators and Multiple Wireless Modules such as ZigBee, LoRa, Bluetooth etc. mPCIe connector with USB interface used for Cloud Connectivity modules 3G, 4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI, PWR etc.

RuggedBoard - A5D2x Specification:

System On Module

SOC Microchip ATSAMA5d2x Cortex-A5

Frequency 500MHz

RAM 64 MB DDR3

Flash 32 MB NOR flash

SD Card SD Card Upto 32 GB

Industrial Interface

RS232 2x RS232

USB 2 x USB*(1x Muxed with mPCIe)

Digital Input4x DIN (Isolated ~ 24V)

Digital Output 4x DOUT (Isolated ~ 24V)

RS485 1xRs485

CAN 1xCAN

Internet Access

Ethernet 1 x Ethernet 10/100

Wi-Fi/BT Optional on Board Wi-Fi/BT

SIM Card 1 x SIM Slot (for mPCIe Based GSM Module)

Add-On Module Interfaces

Mikro-BUS Standard Mikro-BUS

mPCIe 1 x mPCIe* (Internally USB Signals is used)

Expansion Header SPI, I2C, UART, PWM, GPIO,ADC

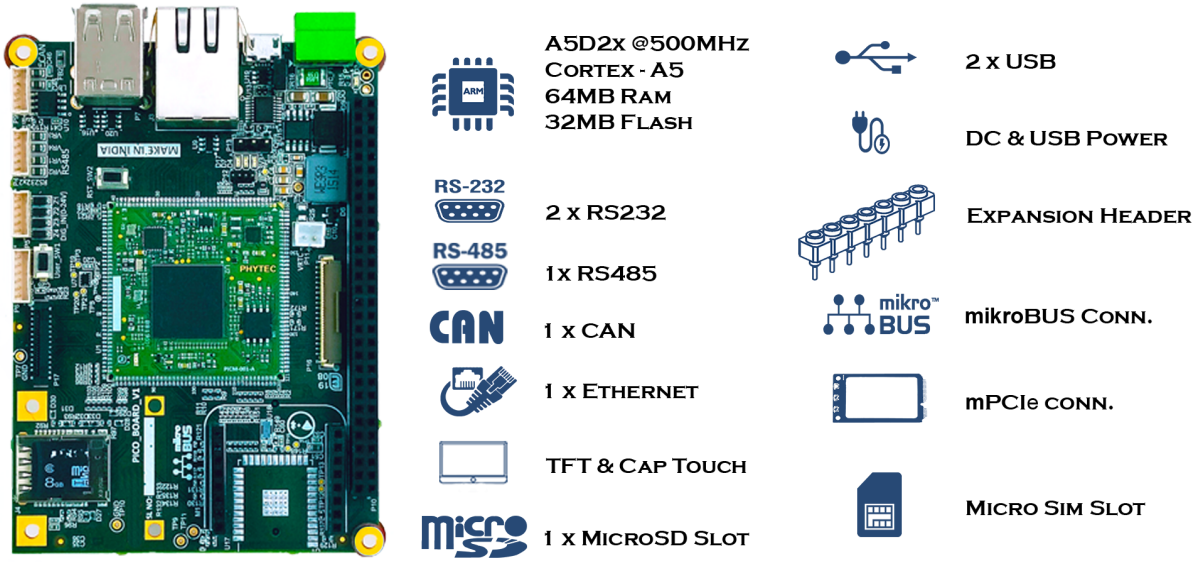
Power

Input Power DC +5V or Micro USB Supply

Temperature Range - 40° to + 85°C

Optional Accessories

Accessories Set Micro USB Cable, Ethernet Cable, Power Adapter 5V/3A



3. Mini two colour led:

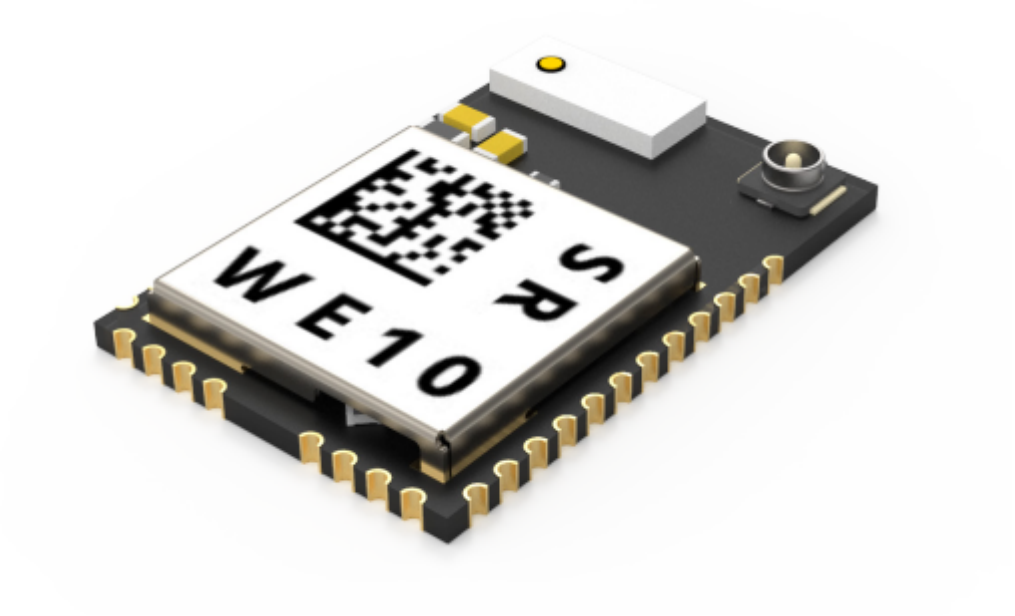
KY-029 Specifications

This module consists of a common cathode 5mm red/green LED, a 0Ω resistor and 3 male pin headers. Use this module with limiting resistors to prevent burnout of the LED when working for long periods of time.

4. W10 WiFi:

The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

- 100mW transmit power
- 11Mbps data rate
- 802.11 b/g/n compatibility
- Integrated antenna



7.Why UART based communication:

The advantages of interrupt-based UART communication over other methods:

Improved efficiency: The CPU is not constantly polling the UART status, so it can be used for other tasks. This can improve the efficiency of the system and reduce power consumption.

Reduced latency: Interrupt-based communication can be faster than polling, as the CPU is only interrupted when data is ready to be sent or received. This can improve the responsiveness of the system.

Better multitasking: Interrupt-based communication allows the CPU to handle multiple UART events simultaneously. This can be useful for applications that require frequent and asynchronous serial communication.

Here are some of the disadvantages of interrupt-based UART communication:

More complex code: Interrupt-based communication is more complex to implement than polling. This is because the programmer needs to write code to handle the interrupts.

More overhead: Interrupt-based communication can have more overhead than polling. This is because the CPU needs to save and restore its state when it is interrupted.

Overall, interrupt-based UART communication is a more efficient and responsive way to communicate with serial devices. However, it is more complex to implement and can have more overhead.

Here are some examples of applications where interrupt-based UART communication would be a good choice:

Sensor networks: Sensor networks often need to communicate with each other or with a central server. Interrupt-based communication can be used to improve the efficiency and responsiveness of these networks.

Wireless modules: Wireless modules often use UART to communicate with the host microcontroller. Interrupt-based communication can be used to improve the performance of these modules.

Real-time systems: Real-time systems often need to communicate with other devices in a timely manner. Interrupt-based communication can be used to ensure that these communications are not missed.

There are 4 stages in this project:

STAGE 1:
First stage is to blink two colours by interfacing the sensor with a microcontroller.

STAGE 2:

The second stage is to obtain the output state and send it to the right tech cloud through the wifi module.

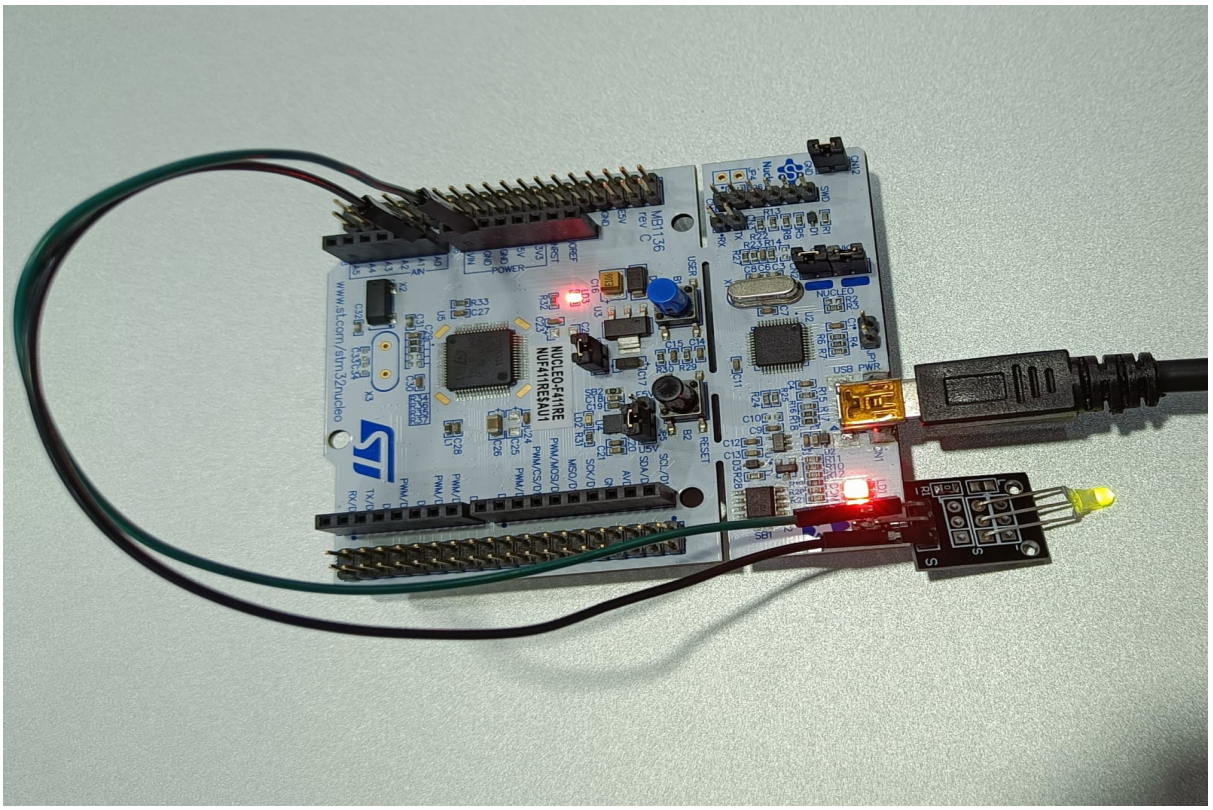
STAGE 3:

Third stage is to send the output state of the sensor from the microcontroller to the Rugged Board.

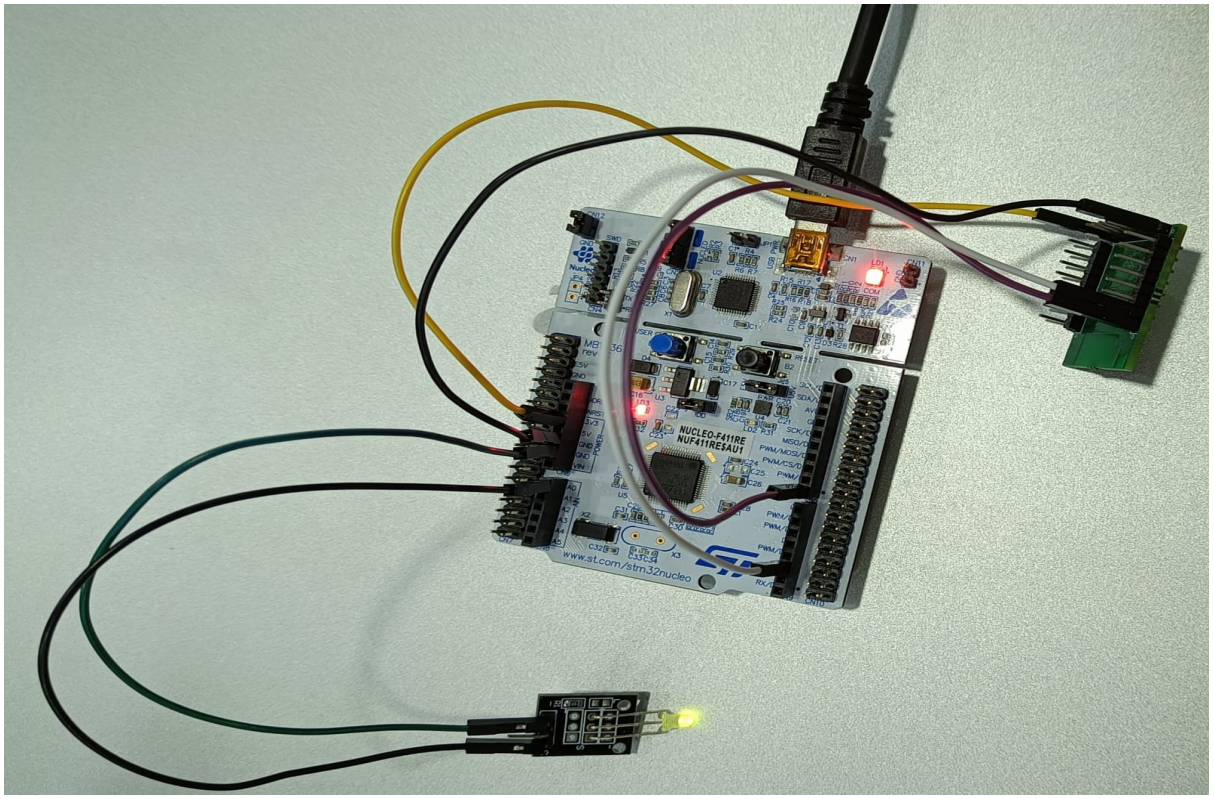
STAGE 4: Fourth stage is to send the data received by the Rugged Board to the right tech cloud using the wifi module.

8.CONNECTION DIAGRAM :

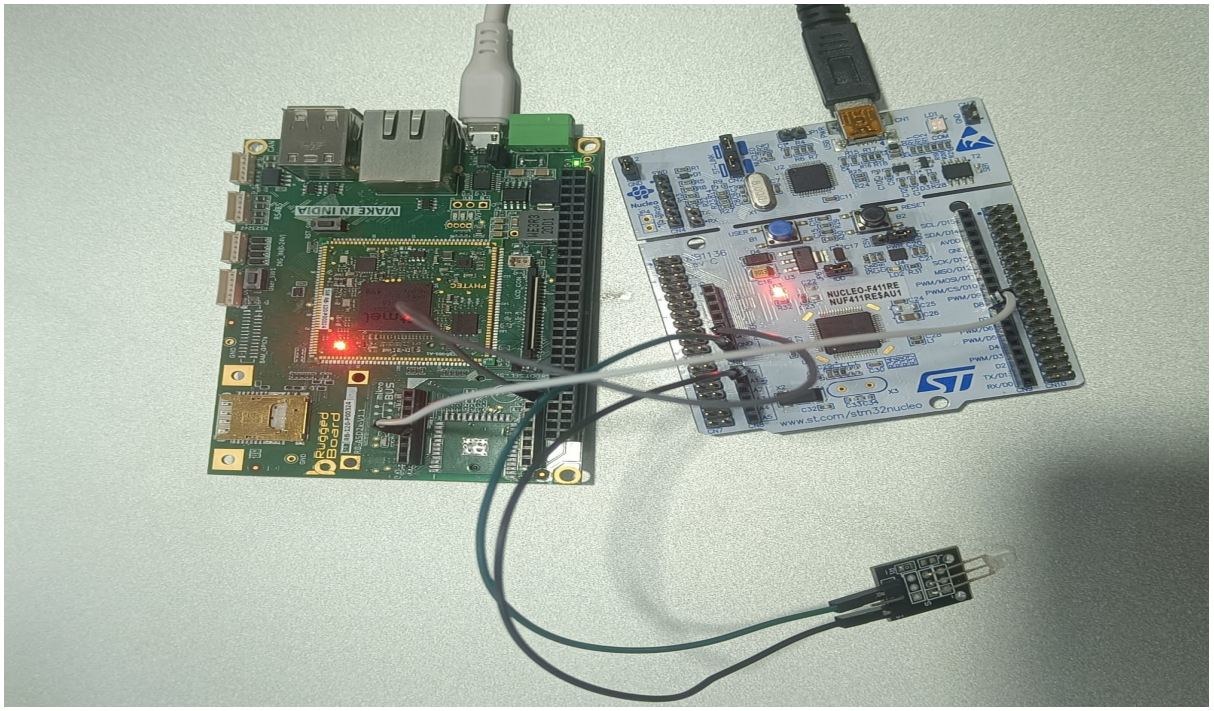
Stage 1:



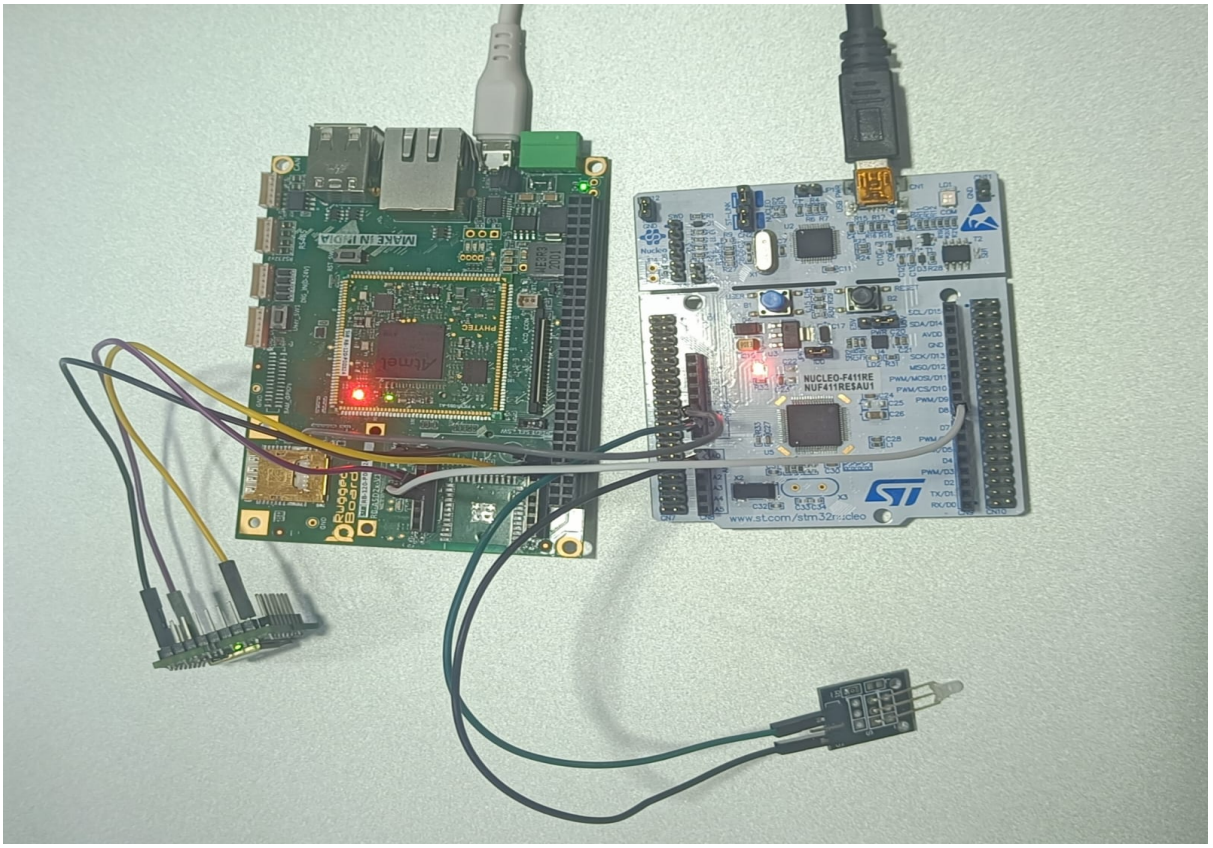
Stage 2:



Stage 3:



Stage 4:



9.PROJECT CODE :

STAGE 2:

STM32F411RE:

```
#include "stm32f4xx_hal.h"
#include <stdio.h> // For sprintf function
#include <string.h>
void MQTT_Init();
void mqtt_data_send(uint8_t n);
void WE10_Init ();
void Error_Handler(void)
```

```
{
while (1)
        {
        // Add error handling or debugging code here
        }
}
// Define GPIO pins for the common cathode dual-color LED module
#define YELLOW_LED_PIN GPIO_PIN_0
// User switch connected to PC13
#define USER_SWITCH_PIN GPIO_PIN_13
// UART handle
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
void GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART1_UART_Init(void);
int main(void)
{
        HAL_Init();
        // Initialize HAL Library
        HAL_InitTick(0);  // Initialize the HAL Tick (if required)
       // Initialize peripherals
        __HAL_RCC_GPIOA_CLK_ENABLE();
        __HAL_RCC_GPIOC_CLK_ENABLE();
```

```
GPIO_Init();

MX_USART2_UART_Init();

MX_USART1_UART_Init();// Initialize UART communication

WE10_Init ();

MQTT_Init();

char message[100]; // Buffer for the message

 while (1)

{

// Read the state of the user switch

uint8_t switchValue = HAL_GPIO_ReadPin(GPIOC,
USER_SWITCH_PIN);

if (switchValue == GPIO_PIN_SET)

{

// User switch is not pressed

HAL_GPIO_WritePin(GPIOA, YELLOW_LED_PIN,
GPIO_PIN_RESET);

// Send a message to Minicom

sprintf(message, "LED is off\r\n");

HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message),
HAL_MAX_DELAY);

mqtt_data_send(0);

}

else

{

// User switch is pressed

HAL_GPIO_WritePin(GPIOA, YELLOW_LED_PIN, GPIO_PIN_SET);
```

```
        // Send a message to Minicom
        sprintf(message, "LED is on\r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message),
HAL_MAX_DELAY);
        mqtt_data_send(1);
        }
        }
}
void GPIO_Init(void)
{
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        GPIO_InitStruct.Pin = YELLOW_LED_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);

        // Configure the user switch pin as input with pull-up
        GPIO_InitStruct.Pin = USER_SWITCH_PIN;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}
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 = 38400;
  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 */
}
static void MX_USART2_UART_Init(void)
{
        huart2.Instance = USART2;
        huart2.Init.BaudRate = 38400; // Adjust the baud rate as needed
        huart2.Init.WordLength = UART_WORDLENGTH_8B;
```

```
        huart2.Init.StopBits = UART_STOPBITS_1;
        huart2.Init.Parity = UART_PARITY_NONE;
        huart2.Init.Mode = UART_MODE_TX;
        huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart2.Init.OverSampling = UART_OVERSAMPLING_16;
        if (HAL_UART_Init(&huart2) != HAL_OK)
        {
        Error_Handler();
        }
}
void WE10_Init ()
{
   char buffer[128];
   /* CMD+RESET **/
   //memset(&buffer[0],0x00,strlen(buffer));
   sprintf (&buffer[0], "CMD+RESET\r\n");
   HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

   HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
   /*  CMD+WIFIMODE=1  **/
   //memset(&buffer[0],0x00,strlen(buffer));
   sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
   HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
```

```
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    /* CMD+CONTOAP=SSID,PASSWD **/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0],"CMD+CONTOAP=vivo 1917,11111111\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    //memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
    /* CMD?WIFI**/
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD?WIFI\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
//    memset(&buffer[0],0x00,strlen(buffer));
//    HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
}
void MQTT_Init()
```

```
{
   char buffer[128];
   /*CMD+MQTTNETCFG */
   //memset(&buffer[0],0x00,strlen(buffer));
   sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
   HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
   //memset(&buffer[0],0x00,strlen(buffer));
   //HAL_Delay(500);
   HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
   HAL_Delay(500);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
   /*CMD+MQTTCONCFG---->LED */
   //memset(&buffer[0],0x00,strlen(buffer));
   sprintf (&buffer[0],
"CMD+MQTTCONCFG=3,mqtt-vijayalakshmivijji1408-bafbxc,,,,,,,,,\r\n");
   HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
   //memset(&buffer[0],0x00,strlen(buffer));
   //HAL_Delay(500);
   HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
   HAL_Delay(500);
   HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
   /*CMD+MQTTSTART */
  //memset(&buffer[0],0x00,strlen(buffer));
```

```
    sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
//    memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(5000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    /CMD+MQTTSUB */
    //memset(&buffer[0],0x00,strlen(buffer));
    sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_Delay(500);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
}
void mqtt_data_send(uint8_t n)
{
char buffer[50];
sprintf(&buffer[0], "CMD+MQTTPUB=reading/mini-colour,%d\r\n", n);
HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), 1000);
HAL_Delay(100);
}
```

RUGGED BOARD :

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>

 int set_interface_attribs(int fd, int speed) {
 struct termios tty;
if (tcgetattr(fd, &tty) < 0) {
printf("Error from tcgetattr: %s\n", strerror(errno));
 return -1;
}
  cfsetispeed(&tty, (speed_t)speed);
          tty.c_cflag |= (CLOCAL | CREAD); /* Ignore modem controls */
         tty.c_cflag &= ~CSIZE;
         tty.c_cflag |= CS8;             /* 8-bit characters */
         tty.c_cflag &= ~PARENB;         /* No parity bit */
          tty.c_cflag &= ~CSTOPB;        /* Only need 1 stop bit */
          tty.c_cflag &= ~CRTSCTS;       /* No hardware flow control */
          tty.c_iflag = IGNPAR;
          tty.c_lflag = 0;
          tty.c_cc[VMIN] = 1;
          tty.c_cc[VTIME] = 1;
```

```
        if (tcsetattr(fd, TCSANOW, &tty) != 0) {
        printf("Error from tcsetattr: %s\n", strerror(errno));
        return -1;
        }
        return 0;
}
int main() {
        char *portname = "/dev/ttyS3";
        int fd;
        int rdlen;
        unsigned char buf[256]; // Adjust buffer size as needed
        fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
        if (fd < 0) {
        printf("Error opening %s: %s\n", portname, strerror(errno));
        return -1;
        }
        if (set_interface_attribs(fd, B9600) != 0) {
        close(fd);
        return -1;
        }
        while (1) {
        rdlen = read(fd, buf, sizeof(buf) - 1); // Read data into the buffer
        if (rdlen > 0) {
        buf[rdlen] = '\0'; // Null-terminate the received data
        printf("Received data: %s\n", buf);
```

```
        } else {
        printf("No data received.\n");
        }
        }
        close(fd);
        return 0;
}
```

STAGE 4 :

STM32F411RE CODE:

```
  #include "main.h"
#include "stm32f4xx_hal.h"
#include <stdio.h> // For sprintf function
#include <string.h>
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */
#define YELLOW_LED_PIN GPIO_PIN_0
#define USER_SWITCH_PIN GPIO_PIN_13
int main(void)
{
```

```
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_USART1_UART_Init();
  char message[100]; // Buffer for the message
  while (1)
  {
/* USER CODE END WHILE */
        uint8_t switchValue = HAL_GPIO_ReadPin(GPIOC,
USER_SWITCH_PIN);
     if (switchValue == GPIO_PIN_SET)
        {
              // User switch is not pressed
HAL_GPIO_WritePin(GPIOA, YELLOW_LED_PIN, GPIO_PIN_RESET);
sprintf(message, "LED is off\r\n");
 HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message),
HAL_MAX_DELAY);
 HAL_UART_Transmit(&huart1, (uint8_t*)message, strlen(message),
HAL_MAX_DELAY);
HAL_Delay(500);
        }
        else
        {
          // User switch is pressed
        HAL_GPIO_WritePin(GPIOA, YELLOW_LED_PIN, GPIO_PIN_SET);
```

```
                // Send a message to Minicom

                sprintf(message, "YELLOW led is on\r\n");

        HAL_UART_Transmit(&huart2, (uint8_t*)message, strlen(message),
HAL_MAX_DELAY);

                HAL_UART_Transmit(&huart1, (uint8_t*)message,
strlen(message), HAL_MAX_DELAY);

HAL_Delay(1000);

/* USER CODE BEGIN 3 */

   }

  /* USER CODE END 3 */

  }

}

void SystemClock_Config(void)

{

  RCC_OscInitTypeDef RCC_OscInitStruct = {0};

  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTA
GE_SCALE1);

  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 = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
        Error_Handler();
  }
  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();
  }
}
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 38400;
  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();
  }
}
static void MX_USART2_UART_Init(void)
{
  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();
  }
 }
static void MX_GPIO_Init(void)
```

```
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  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);
}
void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```
#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif /* USE_FULL_ASSERT */
```

RUGGED BOARD CODE:

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
int set_interface_attribs(int fd, int speed)
{
struct termios tty;
if (tcgetattr(fd, &tty) < 0)
{
printf("Error from tcgetattr: %s\n", strerror(errno));
return -1;
}
cfsetispeed(&tty, (speed_t)speed);
tty.c_cflag |= (CLOCAL | CREAD); /* ignore modem controls */
tty.c_cflag &= ~CSIZE;
tty.c_cflag |= CS8; /* 8-bit characters */
```

```
tty.c_cflag &= ~PARENB; /* no parity bit */
tty.c_cflag &= ~CSTOPB; /* only need 1 stop bit */
tty.c_cflag &= ~CRTSCTS; /* no hardware flowcontrol */
tty.c_iflag = IGNPAR;
tty.c_lflag = 0;
tty.c_cc[VMIN] = 1;
tty.c_cc[VTIME] = 1;
if (tcsetattr(fd, TCSANOW, &tty) != 0)
{
printf("Error from tcsetattr: %s\n", strerror(errno));
return -1;
}
return 0;
}
int main()
{
char *portname = "/dev/ttyS3";
int fd;
int wlen;
int rdlen;
int ret;
char res[5];
char arr1[] = "CMD+RESET\r\n";
char arr2[] = "CMD+WIFIMODE=1\r\n";
char arr[] = "CMD+CONTOAP=\"hari\",\"9865453376\"\r\n";
```

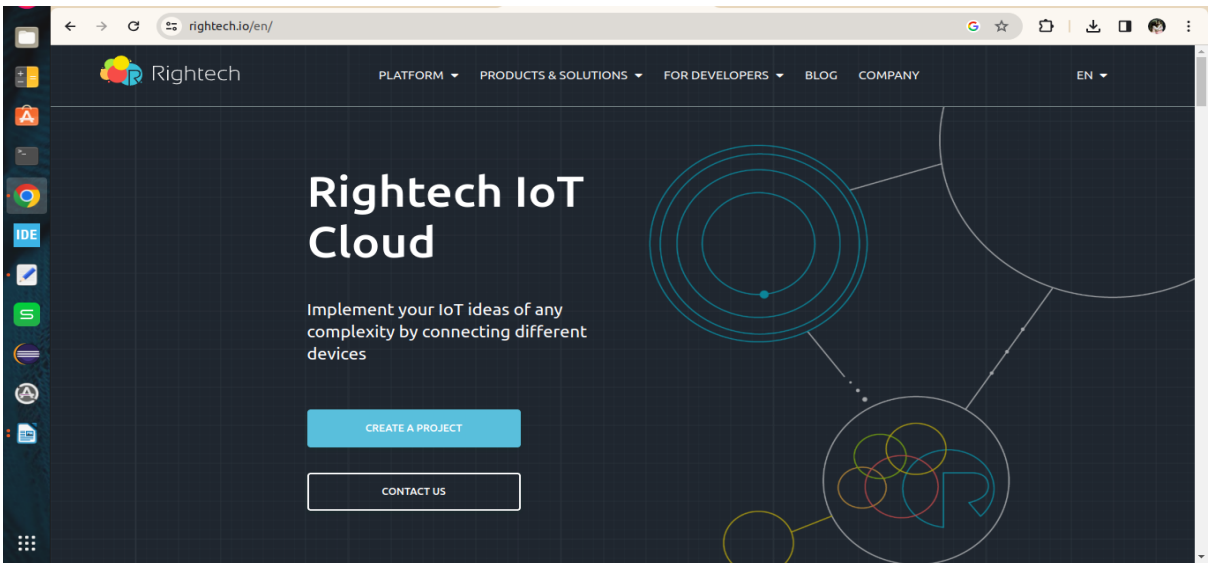
```
char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";
char arr4[] =
"CMD+MQTTCONCFG=3,mqtt-harishkumarslm-qcangb,,,,,,,,,\r\n";
char arr5[] = "CMD+MQTTSTART=1\r\n";
char arr6[] = "CMD+MQTTSUB=base/state/centimeter\r\n";
unsigned char buf[100];
fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
printf("Error opening %s: %s\n", portname, strerror(errno));
return -1;
}
set_interface_attribs(fd, B38400);
printf("%s", arr1);
wlen = write(fd, arr1, sizeof(arr1) - 1);
sleep(3);
printf("%s", arr2);
wlen = write(fd, arr2, sizeof(arr2) - 1);
sleep(3);
printf("%s", arr);
wlen = write(fd, arr, sizeof(arr) - 1);
sleep(3);
printf("%s", arr3);
wlen = write(fd, arr3, sizeof(arr3) - 1);
sleep(3);
```

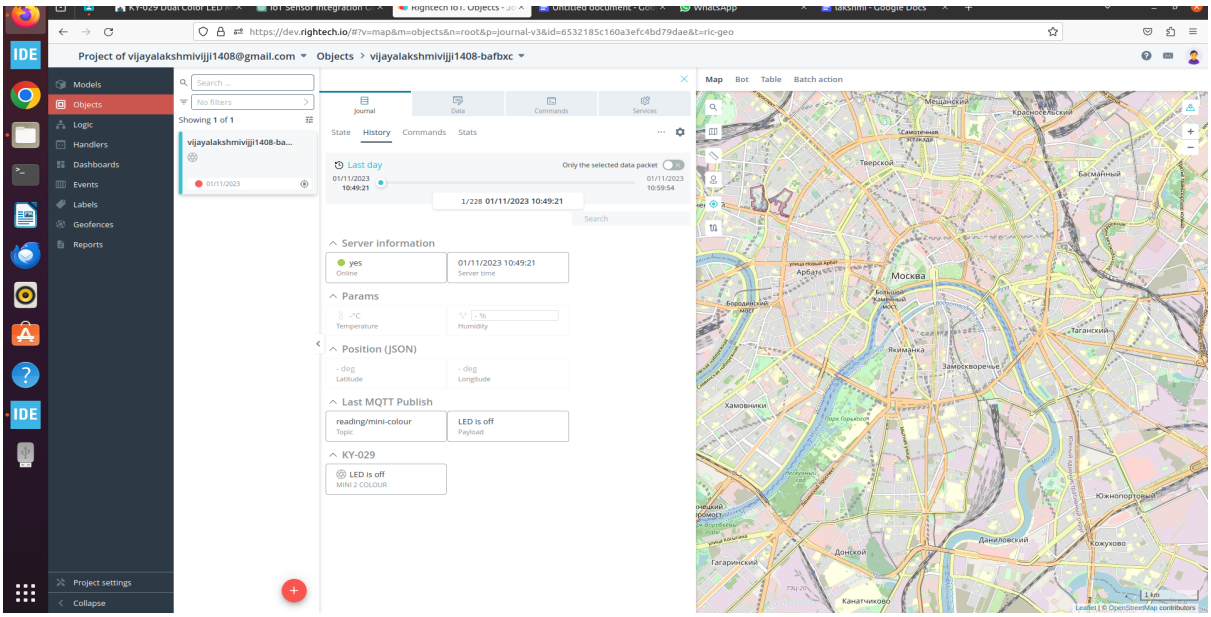
```
printf("%s", arr4);
wlen = write(fd, arr4, sizeof(arr4) - 1);
sleep(3);
printf("%s", arr5);
wlen = write(fd, arr5, sizeof(arr5) - 1);
sleep(3);
printf("%s", arr6);
wlen = write(fd, arr6, sizeof(arr6) - 1);
sleep(3);
while(1){
rdlen = read(fd, buf, sizeof(buf) - 1);
if (rdlen > 0) {
buf[rdlen] = '\0'; // Null-terminate the received data
printf("%s\n", buf);
int ret = snprintf(buffer, sizeof(buffer),
"CMD+MQTTPUB=base/state/centimeter,%s\r\n", buf);
if (ret < 0) {
// Handle the error if sprintf fails
} else {
// Open the file descriptor 'fd' if not already opened
// Write the formatted message to the file descriptor
ssize_t wlen = write(fd, buffer, ret);
sleep(3);
if (wlen == -1) {
// Handle the write error if needed
```

```
}

}

}

}

close(fd);

return 0;

}
```

10.RIGHTECH IOT CLOUD :

Rightech IoT Cloud is a tool for developers. RIC is independent of specific equipment and protocols, which makes it easier for developers to combine different devices under one solution.Platform tools allow developers to create IoT solutions without extra code and reuse 90% of that solution to launch similar cases.





REFERENCE:

1. https://arduinomodules.info/ky-029-dual-color-led-module/

2.https://sensorkit.joy-it.net/en/sensors/ky-029

3.https://daroghawala.org/product/ky-029-ky-029-two-color-red-and-green-led-common-cathode-sensor-module-in-pakistan/