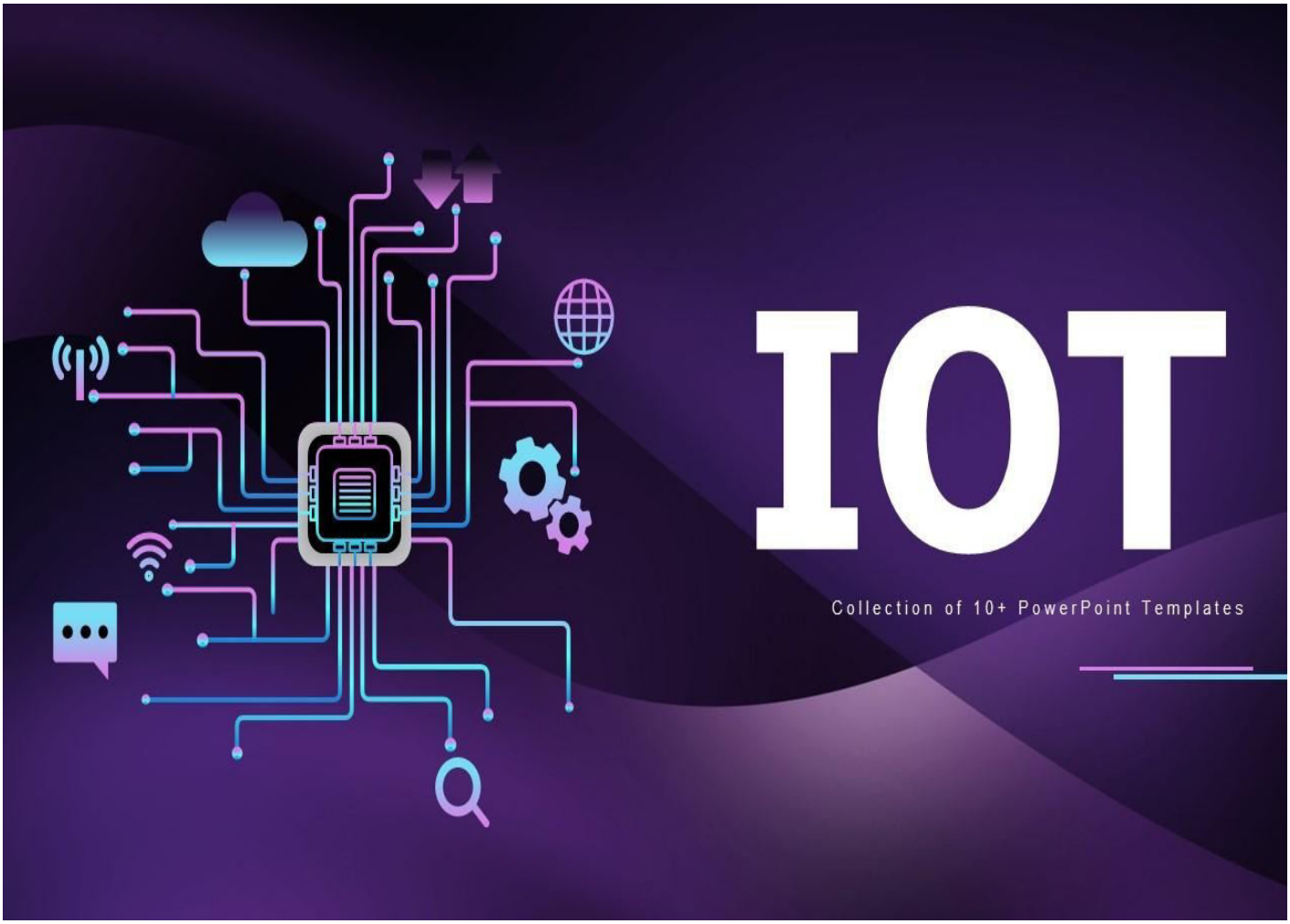
KY-011

TWO COLOR SENSOR



Presented by
K. Prem kumar
SBCS indian pvt ltd

Table of Contents

1.0 Introduction

The KY-011 two-colour sensor is a compact electronic module designed for colour detection and recognition in a variety of electronic projects. Comprising two key components—a red-sensitive photodiode and a green-sensitive photodiode—the sensor operates on the fundamental principle of colour differentiation based on light absorption and reflection.

At its core, the KY-011 utilises the unique properties of materials to selectively absorb and reflect light at specific wavelengths. The red photodiode primarily responds to light in the red spectrum, typically ranging from 600 to 750 nanometers, while the green photodiode is sensitive to light in the green spectrum, generally between 500 and 600 nanometers.

The sensor's working mechanism involves exposing the object of interest to light, and the photodiodes generate analog voltage signals corresponding to the intensity of red and green light detected. These signals are then processed by a connected microcontroller, often an Arduino, which interprets the colour information.

One noteworthy feature of the KY-011 sensor is its potential for sensitivity adjustment. Some versions of the module include a potentiometer, allowing users to fine-tune the sensor's responsiveness to different lighting conditions.

The analog output from the KY-011 sensor lends itself to straightforward integration with microcontrollers, enabling the implementation of colour recognition algorithms. These algorithms can define specific thresholds for red and green values, facilitating the identification of distinct colours based on the sensor's readings.While the KY-011 sensor provides a cost-effective solution for basic colour detection, it is important to consider its limitations. The sensor is optimised for red and green detection and may not cover the entire visible spectrum. Additionally, ambient light conditions can impact its accuracy, necessitating thoughtful consideration during application design.

2.0 Hardware Components

2.1 STM32 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 OTG
- RTC

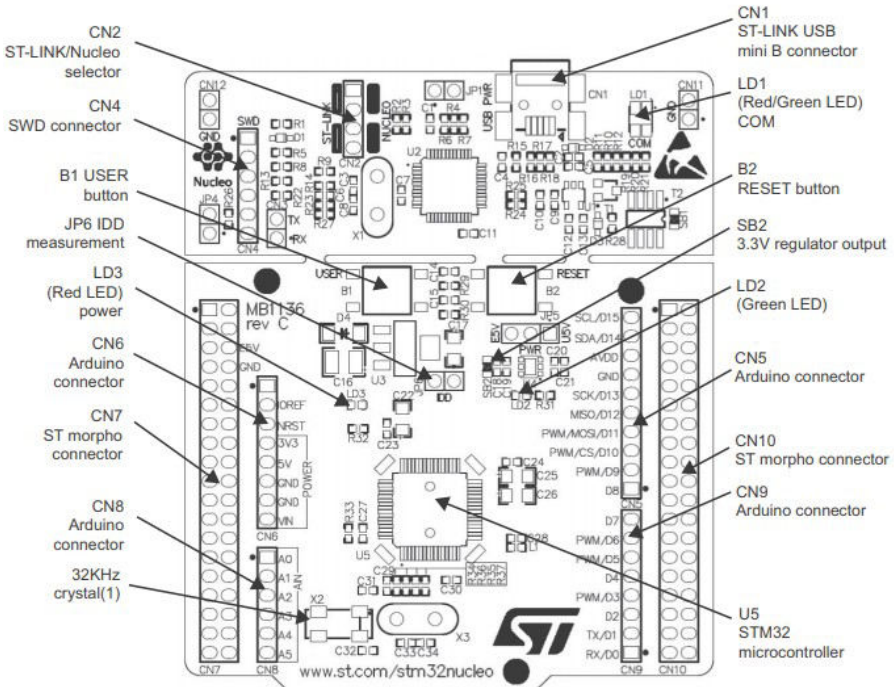


Fig: STM32 pin configuration

2.2 KY-011 two color Sensor

This module consist of a common cathode 3mm red/green LED, a 0Ω resistor, and 3 male header pins. Since the operating voltage is between 2.0v and 2.5v, you'll have to use limiting resistors to prevent burnout when connecting to the Arduino/STM32.

Operating Voltage	2.0v ~ 2.5v
Working Current	10mA
Package Type	Diffusion
Color	Red + Blue
Beam Angle	150

2.3 Connection Diagram

Connect the Red pin (R) on the board to Pin 10 on the Arduino, connect the Blue pin (B) to pin 11. Lastly, connect the ground pin (Y) to GND.

We'll use a couple of resistors between the board and the STM32 to prevent burning the LED.

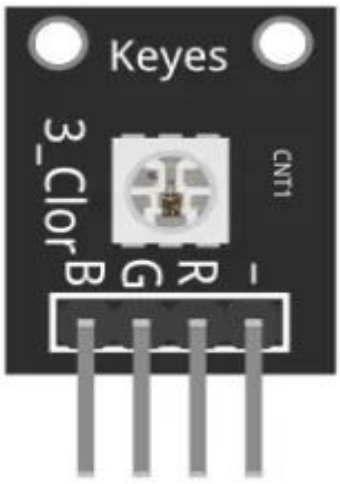


Fig: Two color sensor

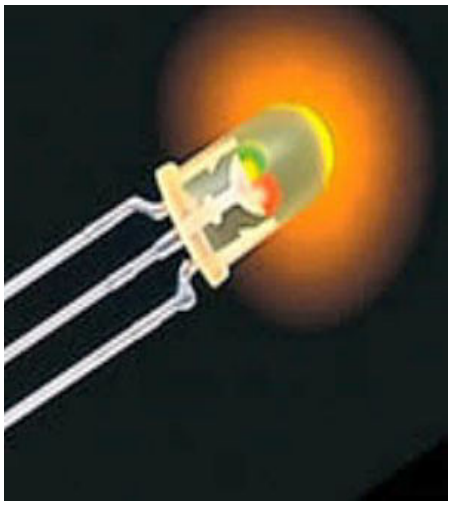


Fig: Internal led in the sensor

2.4 Rugged Board

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. mPCIeconnector with USB interface used for Cloud Connectivity modules 3G,4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI,PWR etc.

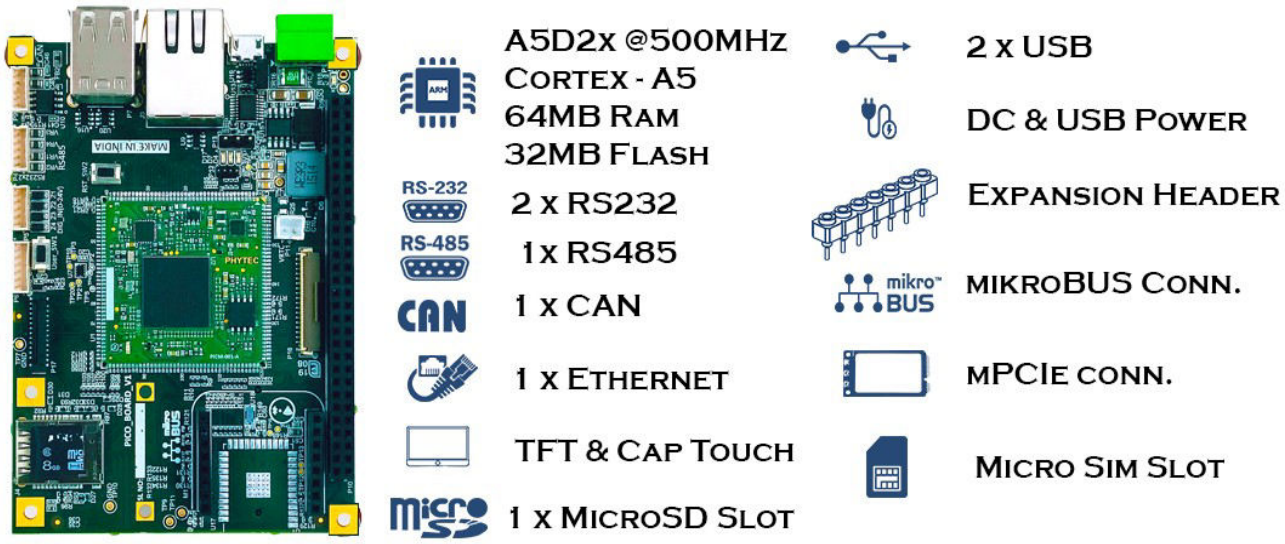


Fig: RuggedBoard-A5D2x

2.5 WE10 Module

The WE10 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 WE10 module is a compact wireless communication solution tailored for IoT applications. With Wi-Fi connectivity, it seamlessly integrates with microcontrollers, supporting diverse communication protocols. Known for its compact form factor, it suits space-constrained applications, coupled with low power consumption for extended device life. Delivering reliable performance, it ensures stable connectivity in various IoT environments. Security features are embedded for confidential data transmission. Its design allows for easy integration into existing hardware setups, simplifying development. The WE10 module finds versatile applications in smart homes, industrial automation, and remote monitoring systems. Its advanced features make it a reliable choice for developers seeking efficient and secure wireless connectivity in their IoT projects.

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

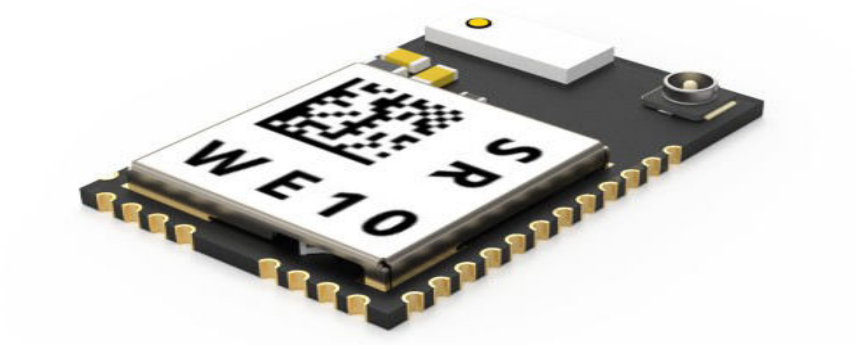


Fig: Wifi Module

3.0 SOFTWARE COMPONENTS

3.1 STM32 IDE Tool

STM32CubeIDE is an integrated development environment (IDE) developed by STMicroelectronics for STM32 microcontroller applications. It combines a user-friendly code editor, compiler, debugger, and peripheral configuration tools into a single platform.

This IDE streamlines development by integrating with STM32CubeMX, a graphical configuration tool for STM32 microcontrollers. Key features include a Hardware Abstraction Layer (HAL) for simplified peripheral interactions, advanced debugging capabilities compatible with popular probes like ST-Link and J-Link, and cross-platform support for Windows, Linux, and macOS.

STM32CubeIDE supports efficient source code management with built-in Git support and facilitates collaborative development. Its real-time operating system (RTOS) integration, particularly with FreeRTOS, enhances multitasking capabilities. The IDE's plugin system allows developers to extend functionality, adapting the environment to specific project needs. Overall, STM32CubeIDE accelerates STM32 microcontroller application development by providing an integrated and feature-rich platform for configuration, code generation, debugging, and collaborative coding.

3.1.1 Overview

1. Integrated Development Environment (IDE):

• The IDE is the primary interface for software development. It include code editing, debugging, and project management tools.

2. STM32CubeMX:

• STM32CubeMX is a graphical configuration tool that allows you to for your STM32 microcontroller.

3. HAL (Hardware Abstraction Layer) Library:

• STM32CubeIDE integrates the HAL library, which provides a set of high-level functions to interact with the microcontroller peripherals.

4. CMSIS (Cortex Microcontroller Software Interface Standard):

• CMSIS provides a standardized interface to the core functions of a Cortex-M microcontroller, including the NVIC (Nested Vector Interrupt Controller) and SysTick.

5. Debugger:

• The debugger allows you to set breakpoints, inspect variables, and step through code during the debugging process.

6. Project Configuration:

• You can configure various project settings, including compiler options, linker scripts, and build configurations.

7. Console Output:

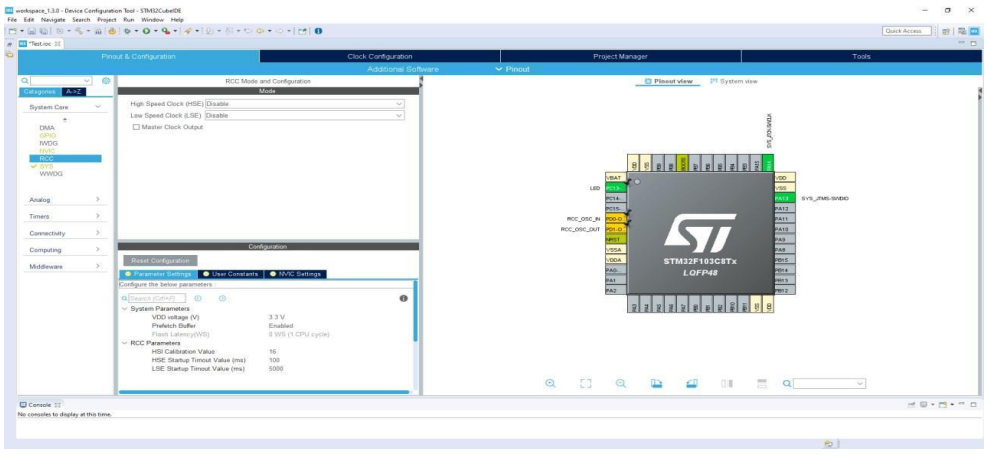
• The console provides feedback from the compiler and debugger, displaying messages, warnings, and errors.

8. Peripheral Configuration:

• STM32CubeMX generates initialization code based on your peripheral configurations, helping to set up the microcontroller's features.

9. Project Explorer:

• The Project Explorer organizes your project files and allows you to navigate through the source code and configuration files.



3.1.2 Workflow

1. Project Initialization:

• Create a new project in STM32CubeIDE.

• Configure the microcontroller and peripherals using STM32CubeMX.

2. Code Development:

• Write and edit your C code in the IDE.

3. Build and Compilation:

• Compile your code to generate the binary file.

4. Debugging:

• Use the debugger to find and fix issues in your code.

5. Flash and Run:

• Flash the compiled code onto the STM32 microcontroller and run the application.

3.2 MINICOM

Minicom is a text-based serial communication program that is commonly used to connect to and communicate with devices over a serial port. It is often used 15 for debugging and configuring devices, especially in embedded systems and projects involving microcontrollers or other hardware components. Below is an explanation of how minicom can be used in a project.

1. Installation:

• Before using minicom, you need to install it on your system. You can typically install it using your system's package manager.

• bash

• sudo apt-get install minicom

2. Connecting to a Serial Port:

• Minicom is primarily used for serial communication, so you need to be connect it to the serial port of the device you want to communicate with.Use the following command to open minicom:

• bash

• minicom -D /dev/ttyUSB0

• Here, /dev/ttyUSB0 is the path to the serial port. The actual port may vary depending on your system and the connected device.

3. Configuration:

• Once minicom is open, you may need to configure the serial port settings such as baud rate, data bits, stop bits, and parity. This is often necessary to match the settings of the device you are communicating with. You can access the configuration menu by pressing Ctrl-A followed by Z.

4. Interacting with the Device:

• After configuring the serial port, you can interact with the device. Minicom allows you to send commands and receive responses. This is particularly useful for debugging purposes and for configuring devices that have a serial console.

5. Exiting Minicom:

• To exit minicom, you can use the Ctrl-A followed by X shortcut.

6. File Transfer:

• Minicom also supports file transfer using protocols like Xmodem or Ymodem. This can be useful for updating firmware or transferring files between your computer and the connected device.

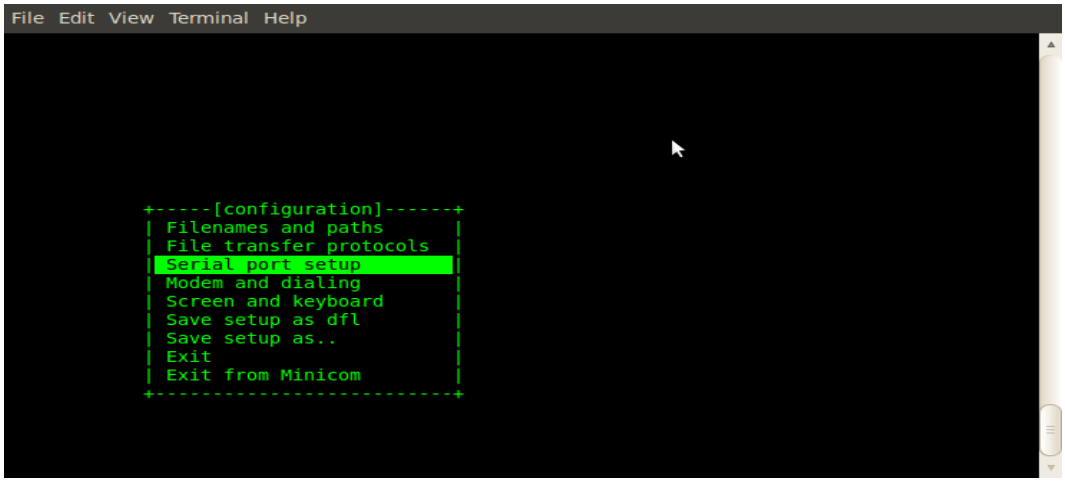


Fig: Minicom Window

3.3 RIGHTTECH 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.

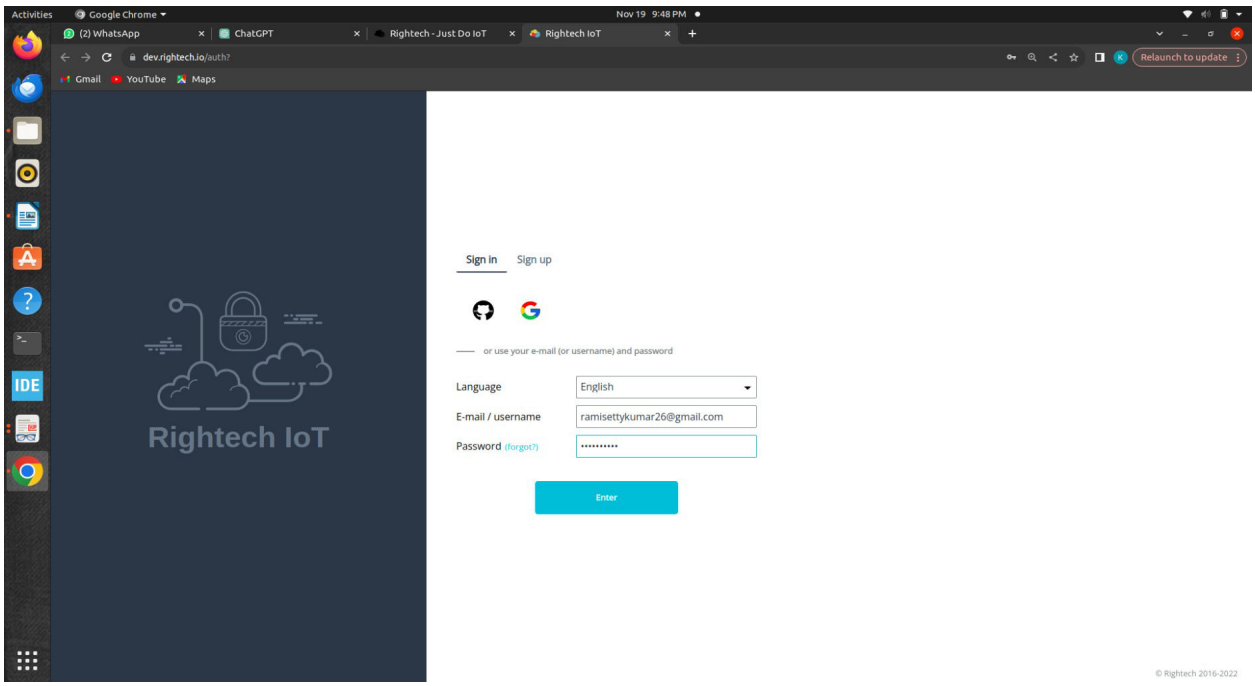
• Visit the "RightTech IoT" Website: Go to the official website or portal of "RightTech IoT."

• Registration: Look for a "Register" or "Sign Up" option on their website. Click on it to start the registration process.

• Fill Out Registration Form: Provide the required information, such as your name, email address, password, and any other details that the platform requests.

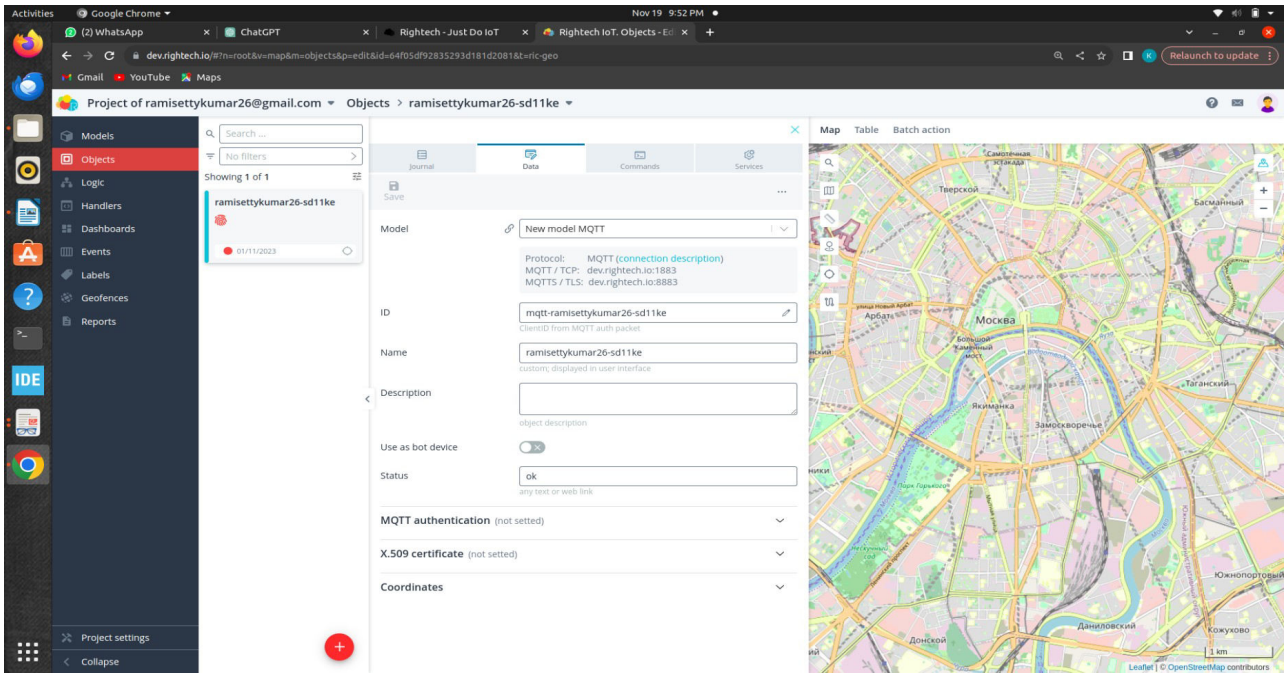
• Account Verification: Some platforms may require you to verify your email address by clicking on a verification link sent to your email. Complete the verification process if required.

• Log In: Once your registration is complete and your account is verified, log in to your "RightTech IoT" account using your credentials.



• Explore the Platform: Navigate through the platform to understand its features, dashboard, and settings. You should look for an option related to creating or managing parameters, which are typically settings or values used to configure and control IoT devices or data.

• Create Parameters: Depending on the platform's interface and options, you may find a section where you can create parameters or configure settings for your IoT devices or applications.



4.0 STAGES

We have different stages to do this module:

- STM32 to Minicom
- STM32 to Minicom & Rightech IOT Cloud
- STM32 to Rugged board a5d2x
- Rugged board a5d2x to Rightech IOT Cloud

4.1 STM32 to Minicom

4.1.1 Pin Configurations

• Connect the GND of sensor to the GND of STM32 MCU.

• Connect Red pin of sensor to analog pin(A0) of STM32 MCU.

• Connect Blue pin of sensor to analog pin(A) of STM32 MCU.

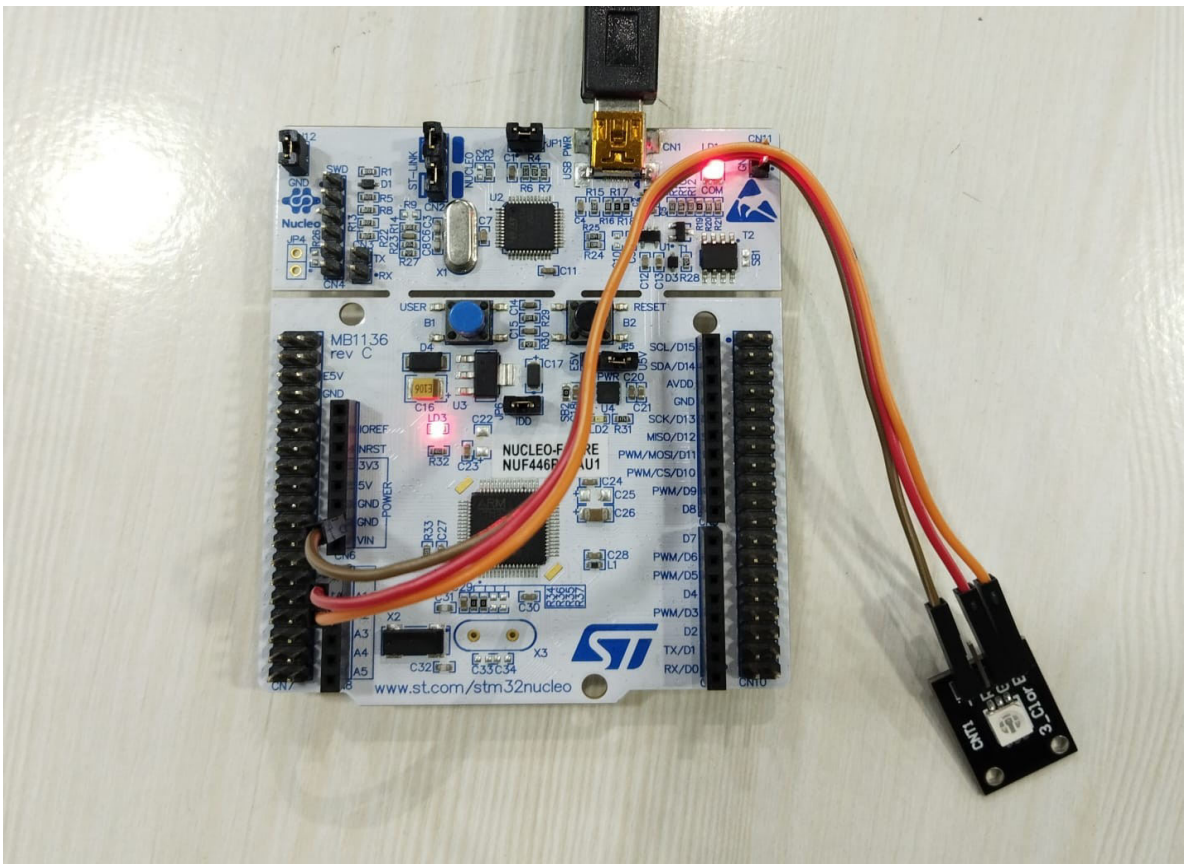


Fig: stage1 pin connections

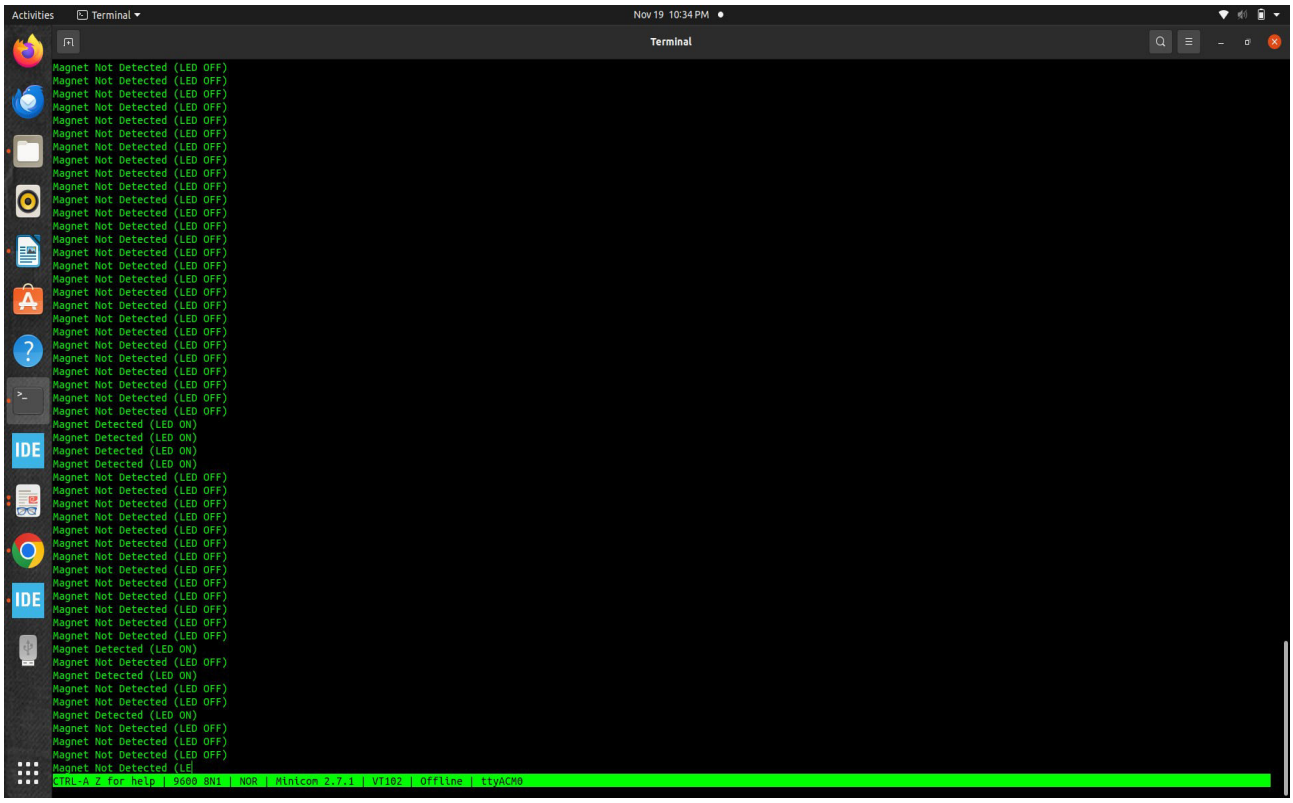
4.1.2 Code

```
while (1)
    {
        uint16_t switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
        if (switchState == GPIO_PIN_RESET)
        {
           HAL_Delay(100); // Debounce the switch
           switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
           if (switchState == GPIO_PIN_RESET)
           {
              ledState = 1 - ledState; // Toggle between 0 and 1
              if (ledState == 0)
               {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_SET);
                 HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN,
GPIO_PIN_RESET);
                  switchvalue = 1;
                  sprintf(message, "red led is on : %d\r\n", switchvalue);
              }
              else
              {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN,
GPIO_PIN_RESET);
                  HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_SET);
                  switchvalue = 2;
                  sprintf(message, "blue led is on : %d\r\n", switchvalue);
              }
              HAL_Delay(1000); // Delay for 1 second to debounce the switch
           }
        }
         else
          {
            switchvalue = 0;
             HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_RESET);
            sprintf(message, "led is off : %d\r\n", switchvalue);
          }
        HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message),
HAL_MAX_DELAY);
      }
```

4.1.3 Explaination

- Reads the state of a switch connected to `SWITCH_PIN` on GPIOC into `switchState`.
- Implements switch debouncing using a 100-millisecond delay and re-reading the switch state.
- Toggles the `ledState` variable between 0 and 1.
- Controls LED states (red and blue) based on the value of `ledState`.
- If `ledState` is 0:
  - Turns on the red LED (`RED_LED_PIN`) and turns off the blue LED (`BLUE_LED_PIN`).
  - Sets `switchvalue` to 1.
  - Constructs a message indicating that the red LED is on.
- If `ledState` is 1:
  - Turns off the red LED and turns on the blue LED.
  - Sets `switchvalue` to 2.
  - Constructs a message indicating that the blue LED is on.
- Implements a 1000-millisecond delay after processing the switch press for further debouncing.
- If the switch state is not pressed:
  - Sets `switchvalue` to 0 and Turns off both the red and blue LEDs and Constructs a message indicating that the LEDs are off.
- Uses HAL UART (`HAL_UART_Transmit`) to send the constructed message (`message`) via UART communication to `huart2`.
- The message includes information about the current state of the LEDs (`switchvalue`).

4.1.4 Output



4.2 STM32 to Minicom & Rightech IOT Cloud

4.2.1 Pin Configurations

- Connect the VCC of sensor to the 5V of STM32 MCU.
- Connect the GND of sensor to the GND of STM32 MCU.
- Connect signal pin of sensor to analog pin(A0) of STM32 MCU.
- Connect WE10 module R_X to STM UART1 T_X(D8) and WE10 T_X to STM UART1 R_X(D2).
- Connect WE10 GND to STM32 GND.
- Connect WE10 VCC to STM VCC(3.3V).

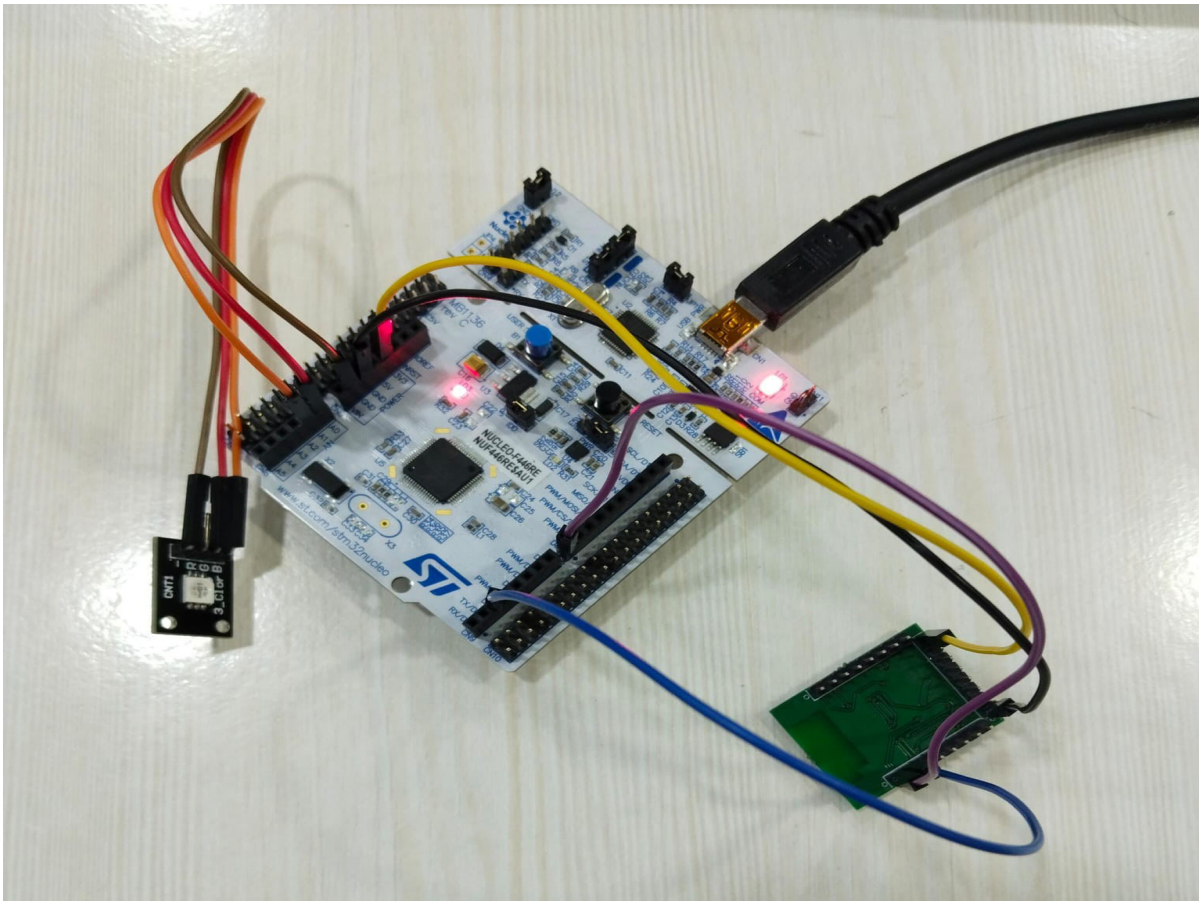


Fig: Stage2 Pin Connections

4.2.2 Code

```
void WE10_Init ()
{
  char buffer[128];
        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);
        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);

        sprintf (&buffer[0],"CMD+CONTOAP=Prem,123456789\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
        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);
        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);
        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];
        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);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer,  strlen(buffer), 10000);
        HAL_Delay(500);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
        sprintf(&buffer[0],"CMD+MQTTCONCFG=3,mqtt-premkonisetty26-sd11ke,,,,,,,,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_Delay(500);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer,  strlen(buffer),  1000);
        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);
        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);
        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);
}
 while (1)
    {
         uint16_t switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);

         if (switchState == GPIO_PIN_RESET)
         {
            HAL_Delay(100); // Debounce the switch
            switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);

            if (switchState == GPIO_PIN_RESET) {
               ledState = 1 - ledState; // Toggle between 0 and 1
```

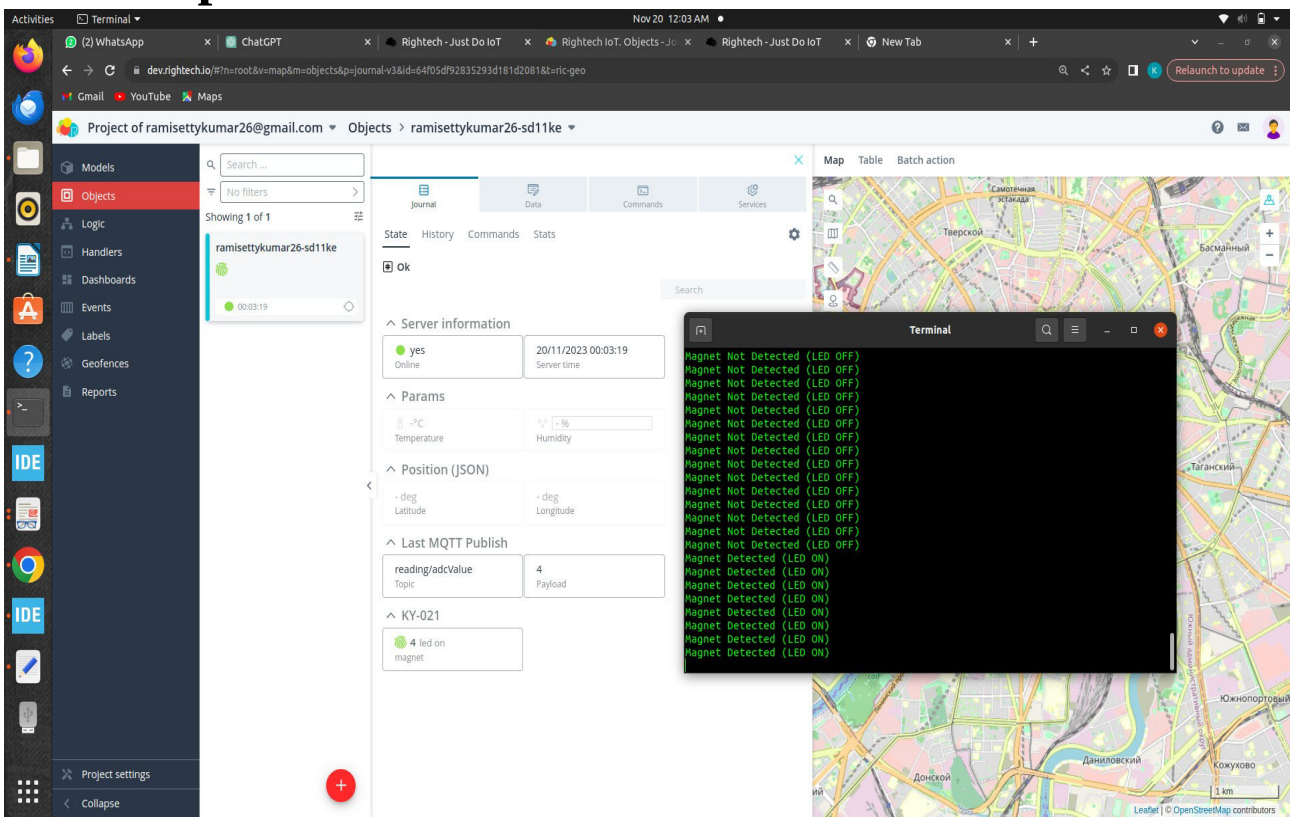
```
            if (ledState == 0)
             {
               HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_SET);
               HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_RESET);
               switchvalue = 1;
               sprintf(message, "red led is on : %d\r\n", switchvalue);
               mqtt_data_send(1);
            } else
            {
               HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_RESET);
               HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_SET);
               switchvalue = 2;
               sprintf(message, "blue led is on : %d\r\n", switchvalue);
               mqtt_data_send(2);
            }

            HAL_Delay(1000); // Delay for 1 second to debounce the switch
          }
        }
         else
          {
           switchvalue = 0;
            HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_RESET);
           sprintf(message, "led is off : %d\r\n", switchvalue);
          }
              HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message),
HAL_MAX_DELAY);
}
```

4.2.3 Explaination

- Continuous loop monitors sensor for magnet presence.
- If SwitchValue is 1, turns on Red LED, sends "Red light is ON" via UART, and transmits sensor data via MQTT.
- If SwitchValue is 2, turns on Blue LED, sends "Blue light is ON" via UART, and transmits sensor data via MQTT.
- If SwitchValue is 0, turns off both Blue and red LEDs and sends "LED is OFF" via UART, and transmits sensor data via MQTT.
- Utilizes STM32 HAL GPIO functions for LED control.
- Real-time UART communication for status updates.
- LED and messages dynamically reflect LED status changes.
- Overall, the code efficiently manages LED blinking and communication through UART and MQTT.

4.2.4 Output



4.3 STM32 to Minicom & Rugged Board a5d2x

4.3.1 Pin Configurations

• Connect the VCC of sensor to the 5V of STM32 MCU.

• Connect the GND of sensor to the GND of STM32 MCU.

• Connect signal pin of sensor to analog pin(A0) of STM32 MCU.

• Connect the STM32 UART1 T_X(D8) to R_X (UART3) of rugged board-a5d2x.

• Connect STM GND to Rugged board –a5d2x GND.

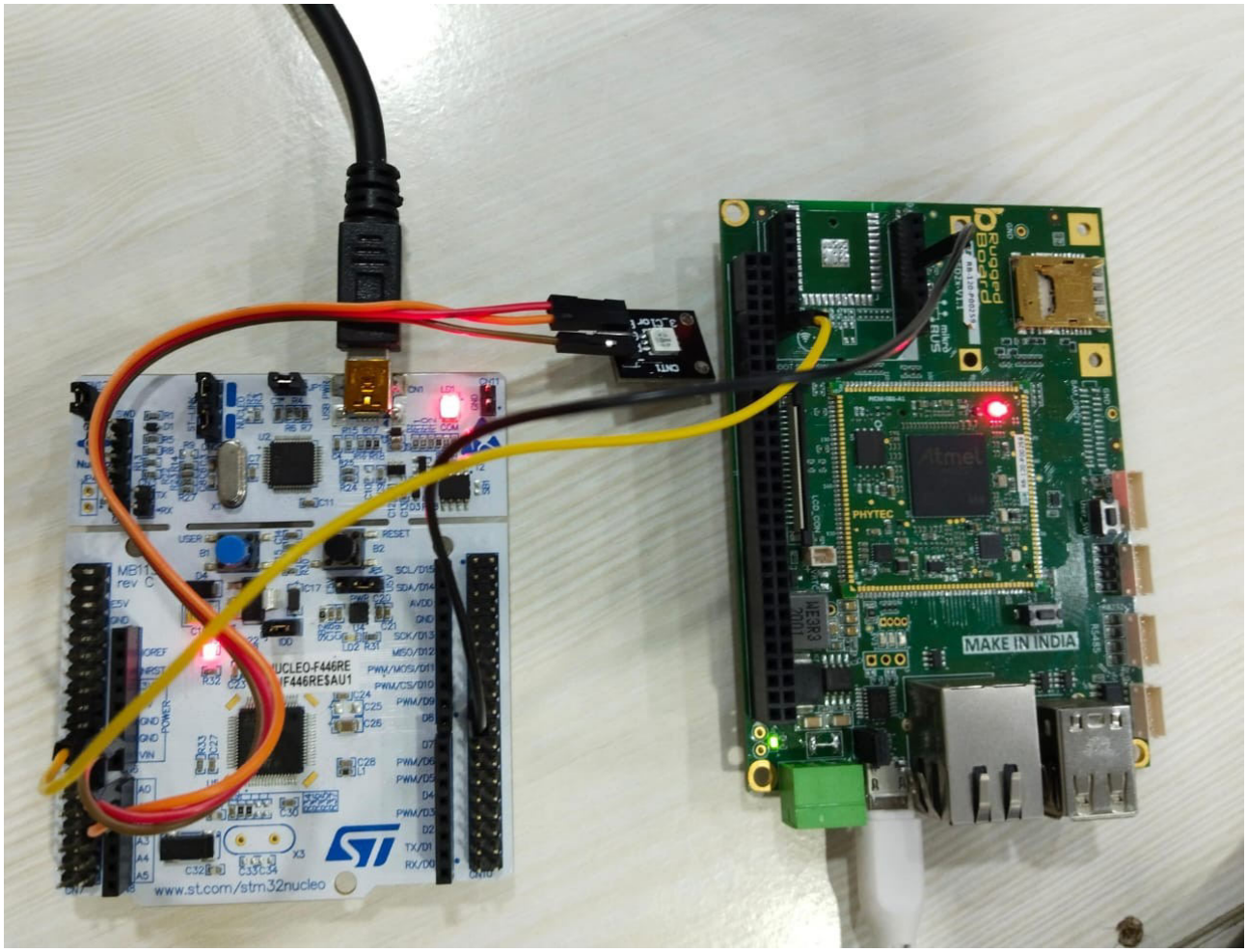


Fig: Stage3 Pin Connections

4.3.2 Code For STM32

```
while (1)
    {
        uint16_t switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
        if (switchState == GPIO_PIN_RESET)
        {
           HAL_Delay(100); // Debounce the switch
           switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
           if (switchState == GPIO_PIN_RESET)
           {
              ledState = 1 - ledState; // Toggle between 0 and 1
              if (ledState == 0)
               {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_SET);
                 HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN,
GPIO_PIN_RESET);
                  switchvalue = 1;
                  sprintf(message, "red led is on : %d\r\n", switchvalue);
              }
              else
              {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN,
GPIO_PIN_RESET);
                  HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_SET);
                  switchvalue = 2;
                  sprintf(message, "blue led is on : %d\r\n", switchvalue);
              }
              HAL_Delay(1000); // Delay for 1 second to debounce the switch
           }
        }
         else
          {
            switchvalue = 0;
             HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_RESET);
            sprintf(message, "led is off : %d\r\n", switchvalue);
          }
        HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message),
HAL_MAX_DELAY);
      }
```

4.3.2 Code For 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";
```

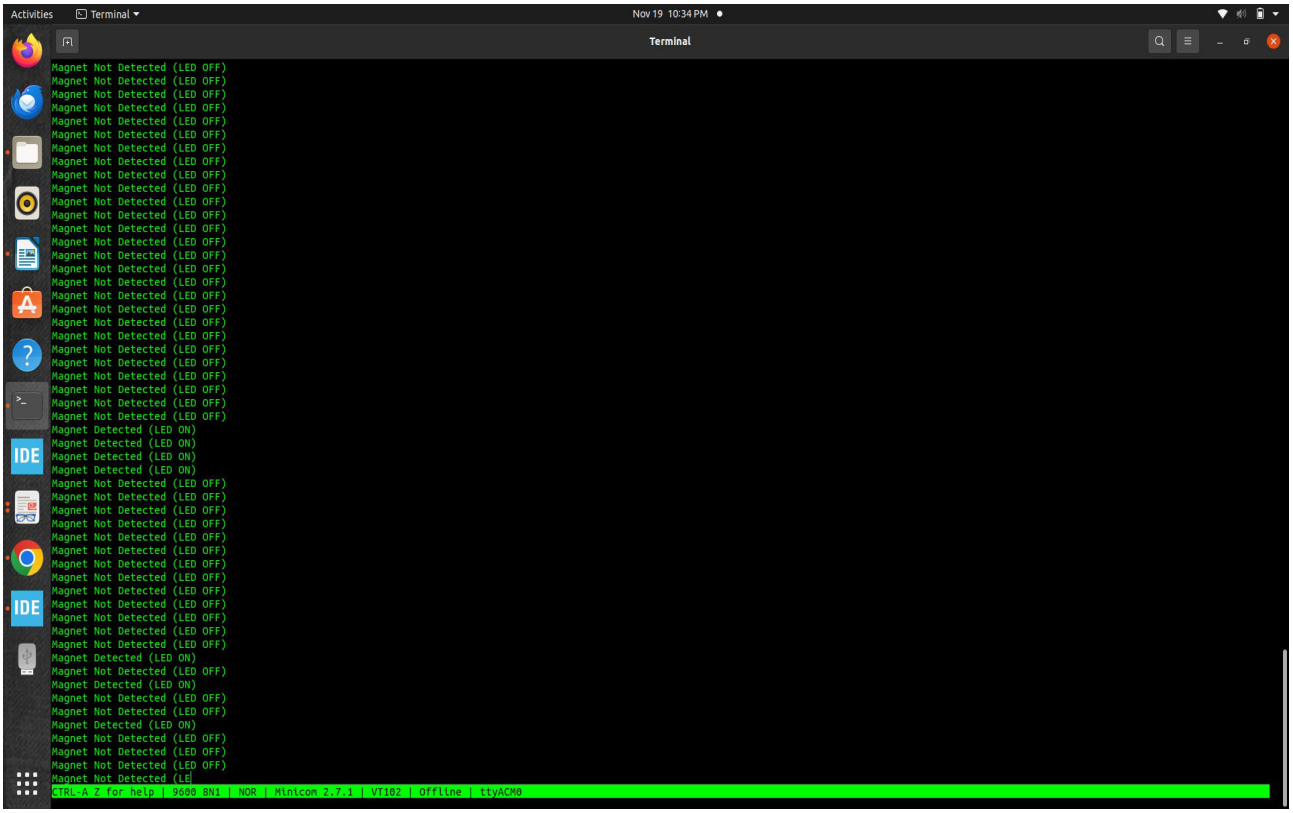
```
int fd;
int rdlen;
unsigned char buf[256]; // Adjust buffer size as needed
fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
  {
    printf("Error opening %s: %s\n", portname, strerror(errno));
    return -1;
  }
if (set_interface_attribs(fd, B9600) != 0)
 {
    close(fd);
    return -1;
 }
while (1)
  {
    rdlen = read(fd, buf, sizeof(buf) - 1);
    if (rdlen > 0)
     {
        buf[rdlen] = '\0';
        printf("Received data: %s\n", buf);
     }
   else
    {
        printf("No data received.\n");
    }
}
 close(fd);
 return 0;
}
```

4.3.3 Explaination

- The STM32 code continuously reads a sensor, controls an LED, and transmits sensor data via UART2 to minicom and UART1 to the rugged board.
- If sensor < 15, LED is turned on, and "Magnet Detected" message is sent with sensor value to both UART2 and UART1.

- The rugged board code initializes UART3, opens the serial port "/dev/ttyS3," and configures it for a baud rate of 9600.
- In an infinite loop, the rugged board code reads data from UART3, prints received data if available, and notifies if no data is received.
- Both devices use common baud rates and message formats for effective communication.
- UART configurations such as baud rates and buffer sizes are adjustable in both codes for flexibility.
- Both codes include error handling mechanisms for UART initialization and data reception.

4.3.4 Output



4.4 STM32 to Minicom & Rugged Board And Rugged Board to Rightech Cloud

4.4.1 Pin Configurations

• Connect the VCC of sensor to the 5V of STM32 MCU.

• Connect the GND of sensor to the GND of STM32 MCU.

• Connect signal pin of sensor to analog pin(A0) of STM32 MCU.

• Connect the STM32 UART1 T_X(D8) to R_X (UART3) of Rugged board.

• Connect WE10 R_X to Rugged board-a5d2x T_X(UART3).

• Connect WE10 GND to Rugged board-a5d2x GND.

• Connect WE10 VCC to Rugged board-a5d2x (3.3v).

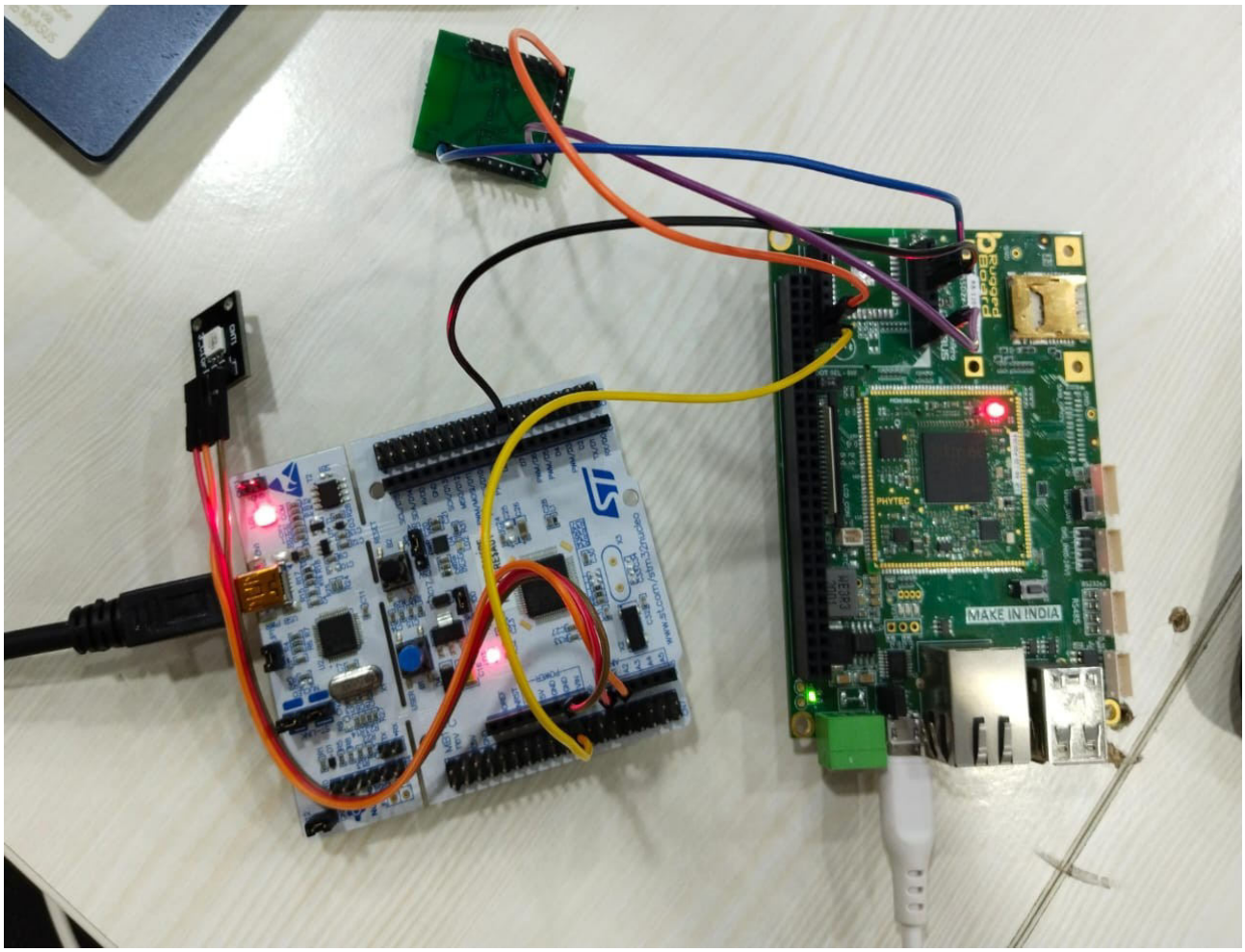


Fig: Stage4 Pin Configuration

4.4.2 Code For STM32

```
while (1)
    {
        uint16_t switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
        if (switchState == GPIO_PIN_RESET)
        {
           HAL_Delay(100); // Debounce the switch
           switchState = HAL_GPIO_ReadPin(GPIOC, SWITCH_PIN);
           if (switchState == GPIO_PIN_RESET)
           {
              ledState = 1 - ledState; // Toggle between 0 and 1
              if (ledState == 0)
               {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_SET);
                 HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN,
GPIO_PIN_RESET);
                  switchvalue = 1;
                  sprintf(message, "red led is on : %d\r\n", switchvalue);
              }
              else
              {
                 HAL_GPIO_WritePin(GPIOA, RED_LED_PIN,
GPIO_PIN_RESET);
                  HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_SET);
                  switchvalue = 2;
                  sprintf(message, "blue led is on : %d\r\n", switchvalue);
              }
              HAL_Delay(1000); // Delay for 1 second to debounce the switch
           }
        }
         else
          {
            switchvalue = 0;
             HAL_GPIO_WritePin(GPIOA, RED_LED_PIN, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(GPIOA, BLUE_LED_PIN, GPIO_PIN_RESET);
            sprintf(message, "led is off : %d\r\n", switchvalue);
          }
        HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message),
HAL_MAX_DELAY);
       }
```

4.4.3 Code For 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 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=\"Phani\",\"123456789\"\r\n";
    char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";
    char arr4[] = "CMD+MQTTCONCFG=3,mqtt-panidharece2023-
vt8h5q,,,,,,,,,\r\n";
    char arr5[] = "CMD+MQTTSTART=1\r\n";
    char arr6[] = "CMD+MQTTSUB=base/relay/led1\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);
```

```
    // Send CMD+WIFIMODE=1
    printf("%s", arr2);
    wlen = write(fd, arr2, sizeof(arr2) - 1);
    sleep(3);

    // Send CMD+CONTOAP
    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);

char buffer[100]; // Create a buffer to hold the formatted message

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=reading/adcValue,%s\r\n", buf);

    if (ret < 0) {

    } else {
        ssize_t wlen = write(fd, buffer, ret);
        sleep(3);
        if (wlen == -1) {

        }
    }
}
}
    close(fd);
    return 0;
}
```

4.4.3 Explanation

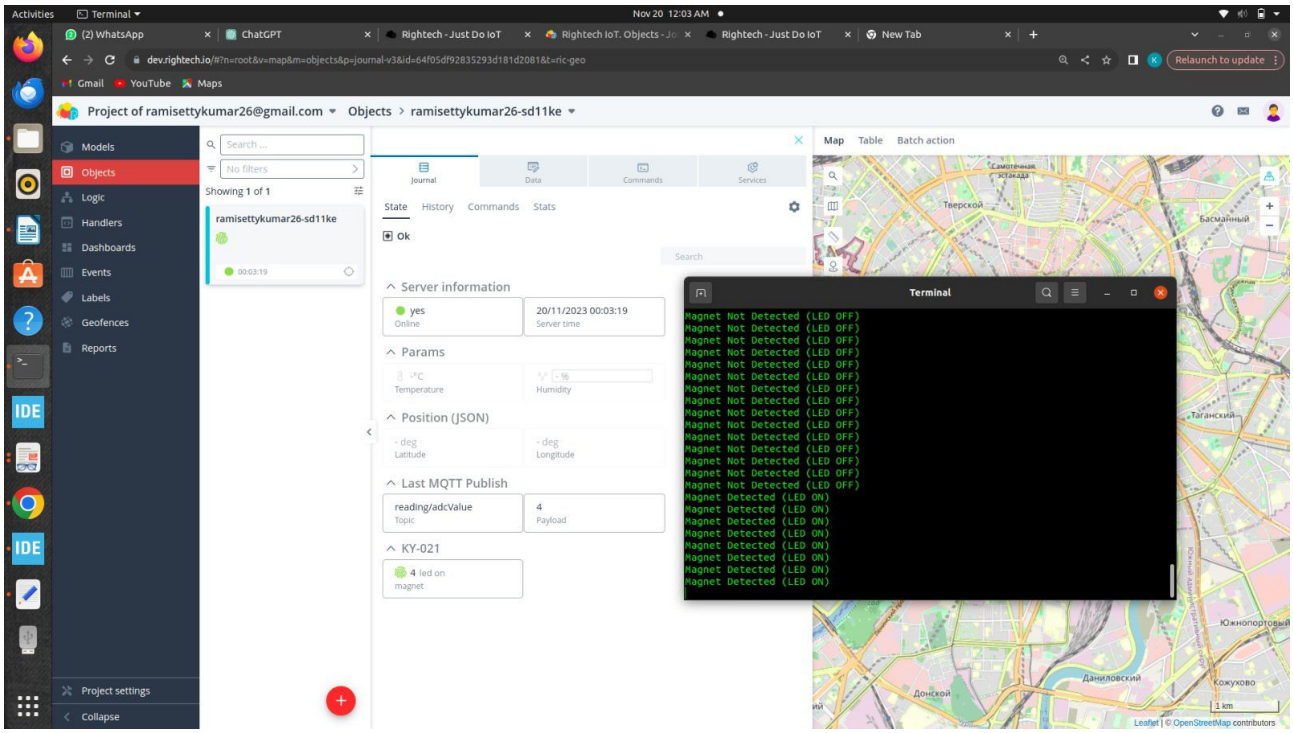
• The program configures a WE10 module through UART3 communication. It sends commands for resetting, setting Wi-Fi mode, connecting to an access point, configuring MQTT parameters, and starting MQTT.

• After configuring, it enters an infinite loop, continuously reading data from the serial port.

• The received data is printed, and formatted MQTT messages are created and sent back to the device.

• The loop runs indefinitely with a 3-second delay after each transmission. The serial port is closed after exiting the loop.

4.4.4 Output



5.0 Real time applications

The KY-011 two-color sensor, also known as the RGB module, is a simple module that consists of a red and green LED. While it might not be as sophisticated as more advanced color sensors, Here are some potential real-time applications for the KY-011 two-color sensor:

1. Color Sorting Systems:

Implementing a small-scale color sorting system for objects based on their color. The sensor can detect the color of the objects and trigger sorting mechanisms accordingly.

2. Traffic Light Simulation:

Simulating a basic traffic light system for educational purposes. The red and green LEDs on the module can represent the different states of a traffic light.

3. Ambient Light Adjustment:

Adjusting the intensity or color temperature of ambient lighting in a room based on the detected color. For example, creating mood lighting that changes based on the environment.

4. Plant Growth Monitoring:

Monitoring the light conditions for plants in a garden or indoor setting. The sensor can be used to adjust artificial lighting for optimal plant growth.

5. Interactive Art Installations:

Incorporating the sensor into interactive art installations where the color or brightness of the artwork changes based on environmental conditions or user interaction.

6. Color-based Alarm System:

Creating a simple alarm system that triggers different responses based on the color detected. For instance, red might indicate a critical condition, while green could signify normal operation.

7. Color Identification for Educational Purposes:

Building educational projects for teaching color identification to children. The sensor can be part of a hands-on learning experience.

8. Feedback for Accessibility Devices:

Integrating the sensor into accessibility devices to provide feedback to users with visual impairments. For example, different colors could represent different states or conditions.

9. Game Development:

Incorporating the sensor into interactive games where the detected color influences the gameplay or visual effects.

10. Quality Control in Manufacturing:

Implementing a basic quality control system on a production line where the color of items is checked, and appropriate actions are taken based on the results.

6.0 REFFERENCES

1. https://arduinomodules.info/ky-009-rgb-full-color-led-smd-module/
2. https://forum.arduino.cc/t/multiple-color-recognition-sensors-on-one-arduino/279572
3. https://arduinomodules.info/ky-011-two-color-led-module-3mm/

7.0 CONCLUSION

In conclusion, the KY-011 two-color sensor, with its dual LED configuration and photodiode for color detection, offers a simple and cost-effective solution for basic color sensing applications. While its features are limited compared to more advanced color sensors, it finds practical use in educational settings, prototyping, and DIY projects. The sensor's ability to detect and differentiate between red and green light, coupled with its ease of integration, makes it a popular choice for beginners and hobbyists exploring fundamental concepts of color mixing and light intensity measurement.