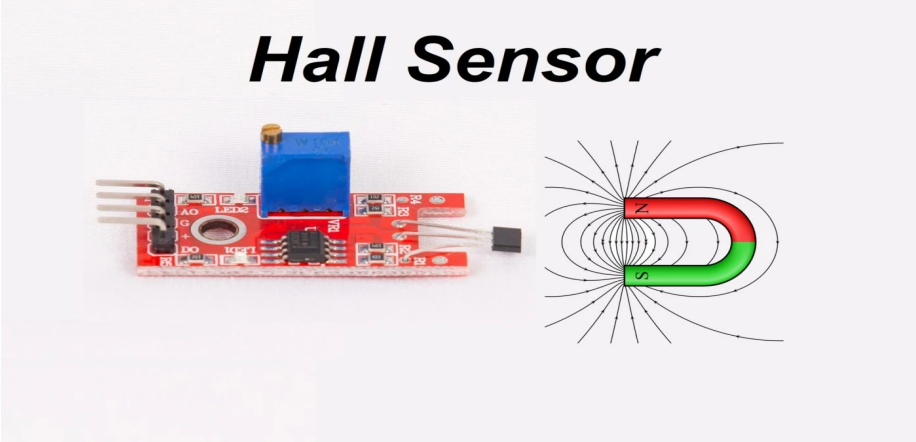
KY_024 Linear Hall Magnetic Sensor



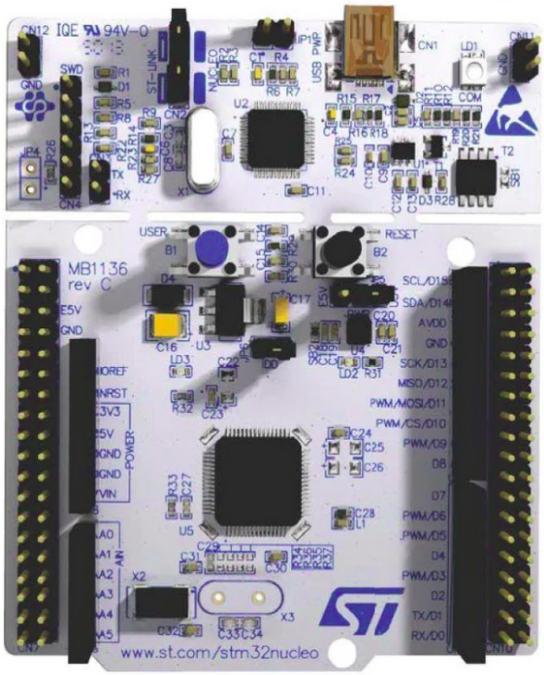


TABLE OF CONTENTS

INDEX:

1. Project Summary

The KY-024 Linear magnetic Hall sensor reacts in the presence of a magnetic field. It has a potentiometer to adjust the sensitivity of the sensor and it provides both analog and digital outputs.

The digital output acts as a switch that will turn on/off when a magnet is near, On the other hand, the analog output can measure the polarity and relative strength of the magnetic field.

This sensor is ideally suited for threshold measurement. This means that the sensor emits a digital high signal as soon as a threshold value set by the user is exceeded. However, this also means that the analog measured values are not suitable for conversions, as the analog signal is also influenced by the rotary potentiometer.

Digital output: If a magnetic field is detected, a signal is output here.

Analog output: Direct measured value of the sensor unit

LED1: Indicates that the sensor is supplied with voltage

LED2: Indicates that a magnetic field has been detected

This project proposes the development using STM32, hall Sensor, and W10 WiFi modules. The system will use an STM32 microcontroller to acquire sensor data from sensor. The sensor data will then be published to MQTT using the W10 WiFi module,

The result of this project where an STM32 microcontroller interacts with a linear Hall magnetic sensor to detect magnetic fields. The LED provides a visual indicator of the sensor's status, and messages are transmitted over UART for further analysis or monitoring. The practical application of this sensor would involve connecting the Hall sensor to the appropriate pins and adapting the configuration to the specific STM32 microcontroller and sensor being used.

2. key features of the project

Common function for UART receive data parsing.

Timer based delay.

STM32 Application is Master and configured with W10 (Wi-Fi) module accordingly with AT commands.

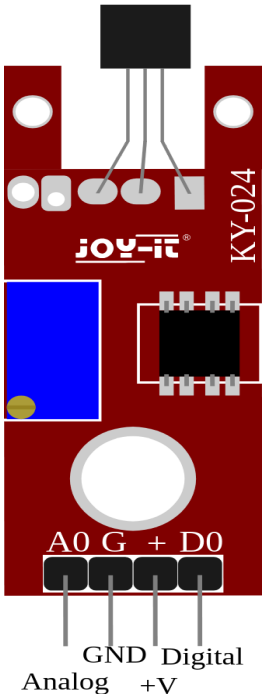
Data is pushed to the MQTT server

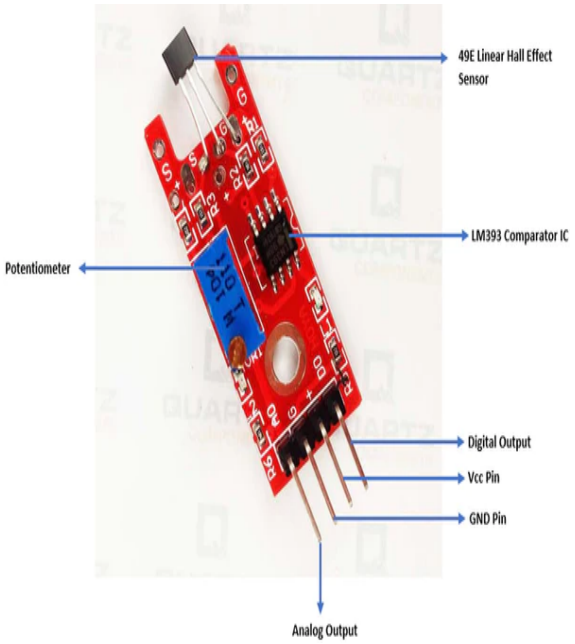
LED Indicator

Analog Output

Digital Output

2.1 Module pinout:





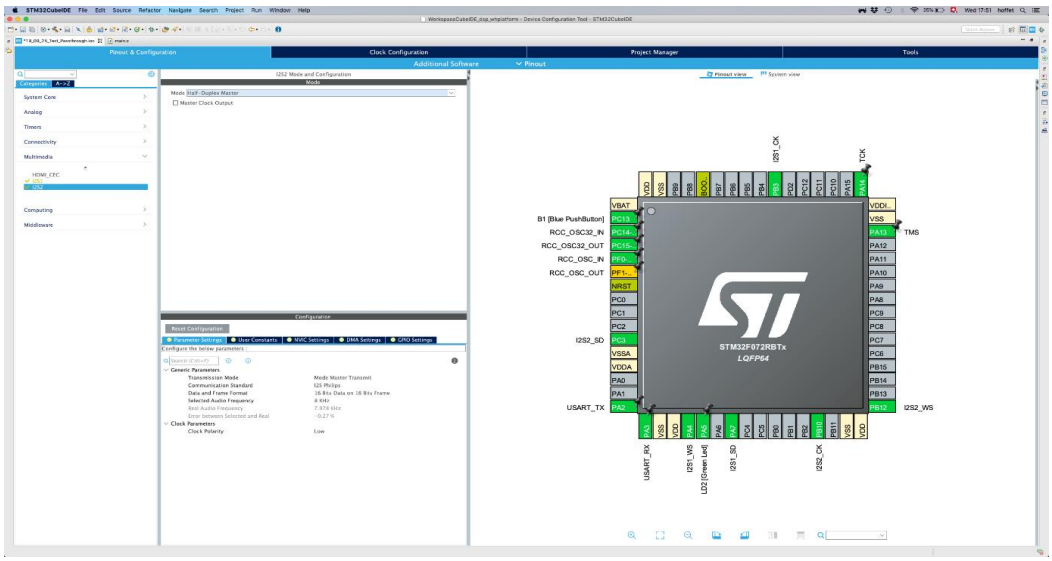
3. Hardware and Software Used and there Specification

3.1. STM32F446RE microcontroller

The STM32F446RE 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

- 180 MHz max CPU frequency
- VDD from 1.7 V to 3.6 V
- 512 KB Flash
- 128 KB SRAM System
- 4 KB SRAM Backup
- USB OTG Full Speed and High Speed
- RTC

Example 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:

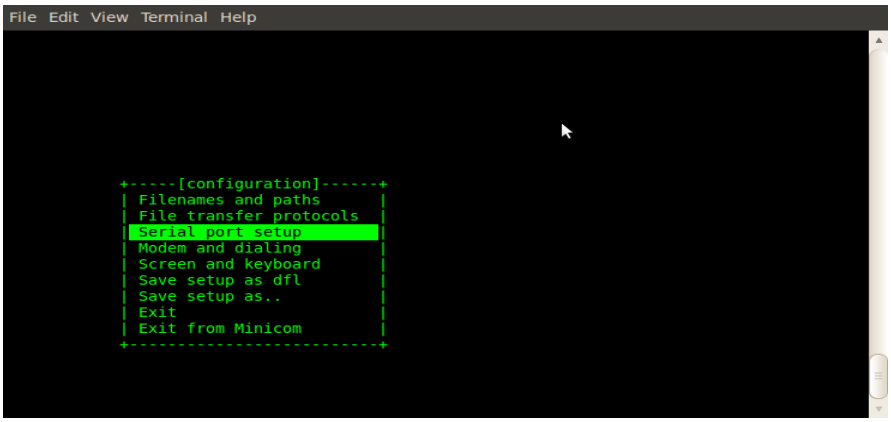
• After configuring the serial port, you can interact with the device. Minicom allows you to send commands and receive responses. This is particularly useful for debugging purposes and for configuring devices that have a serial console.

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.



3.3 MQTT:

The code you provided is a initialize a WE10 module and connect it to an MQTT broker. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.

The next few lines of code send the CMD+MQTTNETCFG command to the WE10 module. This command configures the module to connect to the MQTT broker at dev.rightech.io on port 1883. The CMD+MQTTCONCFG command configures the module to connect to the MQTT broker as client with the username mqtt-cherukusivani-kfggbi and no password. The CMD+MQTTSTART

command starts the MQTT client and connects to the broker. The CMD+MQTTSUB command subscribes the client to the topic base/relay/led1.

The MQTT_Init() function is a simple example of how to initialize a WE10 module and connect it to an MQTT broker. The function takes no arguments and it returns void.

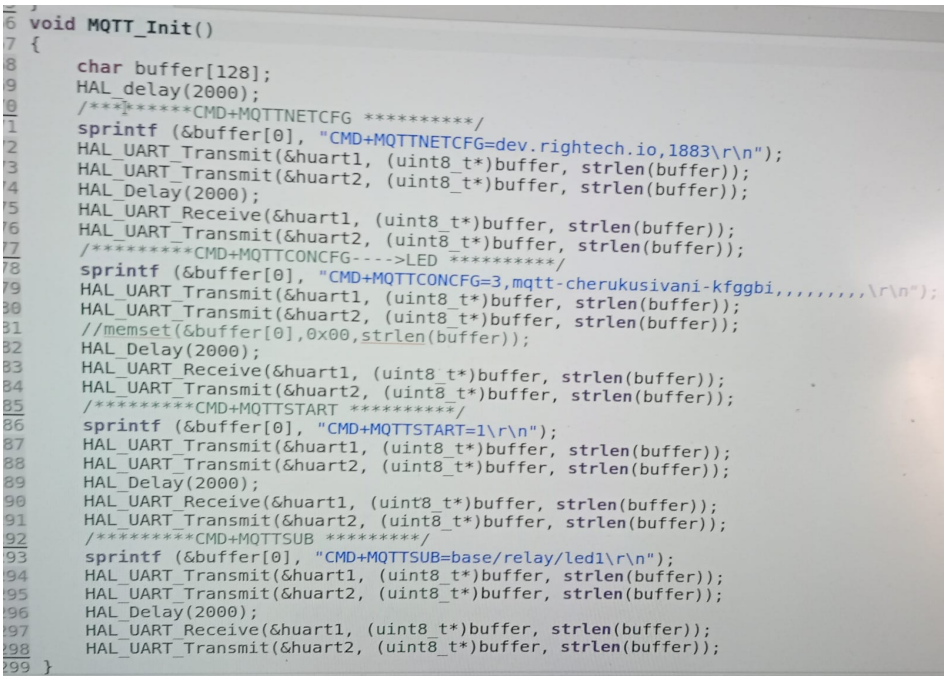
Here is a more detailed explanation of the code:

The CMD+MQTTNETCFG command is used to configure the MQTT parameters of theWE10module. The first parameter is the hostname or IP address of the MQTT broker. The second parameter is the port number of the MQTT broker.

The CMD+MQTTCONCFG command is used to configure the MQTT client of the WE10 module. The first parameter is the username of the MQTT client. The second parameter is the password of the MQTT client.

The CMD+MQTTSTART command is used to start the MQTT client of the WE10 module. This command connects the client to the MQTT broker.

The CMD+MQTTSUB command is used to subscribe the MQTT client to a topic. The first parameter is the topic that the client wants to subscribe to.



```
void MQTT_Init()
{
    char buffer[128];
    HAL_delay(2000);
    /*********CMD+MQTTNETCFG **********/
    sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTCONCFG---->LED **********/
    sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-cherukusivani-kfggbi,,,,,,,,,\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    //memset(&buffer[0],0x00,strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTSTART **********/
    sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    /*********CMD+MQTTSUB *********/
    sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
    HAL_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer));
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer));
}
```

3.4 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.

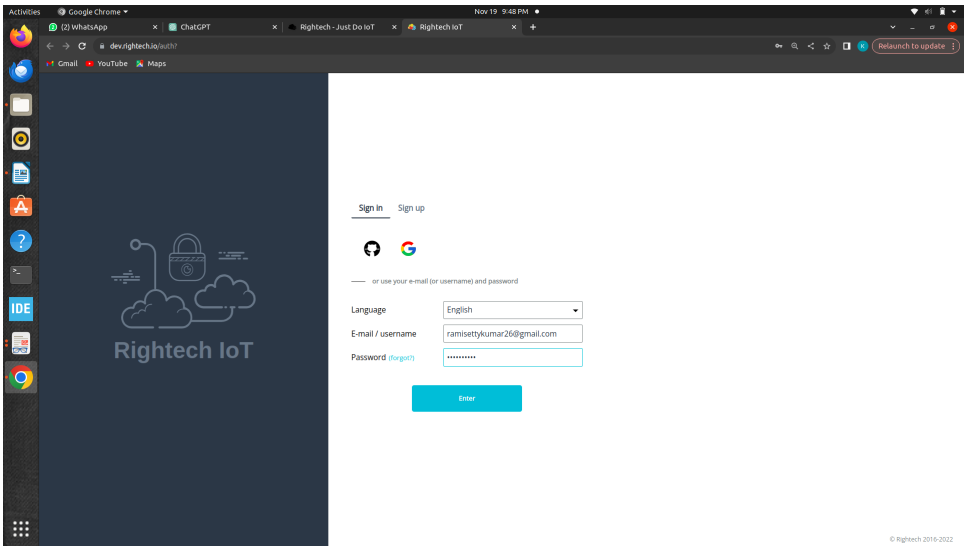
• Visit the "RightTech IoT" Website: Go to the official website or portal of "RightTech IoT."

• 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.

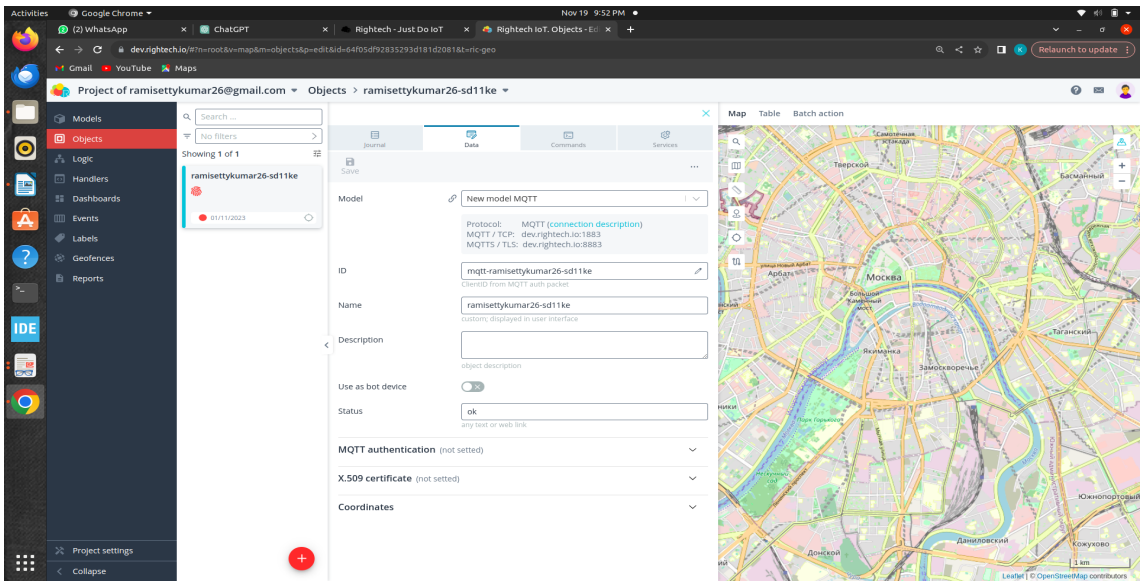
• Account Verification: Some platforms may require you to verify your email address by clicking on a verification link sent to your email. Complete the verification process if required.

• 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.



3.5 Linear-Hall Magnetic Sensor

Specifications:

- Operating Voltage Range: 2.7V to 6.5V
- Sensitivity 1.0mV per
- Board Dimension 1.5cm×3.6cm
- Responds to Either Positive or Negative
- Linear Output for circuit design flexibility

KY-024 Sensor

3.6 Components Used:

STM32 Microcontroller: The code is written for an STM32 microcontroller, which is a family of 32-bit ARM-based microcontrollers manufactured by STMicroelectronics.

Linear Hall Magnetic Sensor:

The linear Hall magnetic sensor is used to detect the presence of a magnetic field. The analog output from the sensor is connected to an ADC pin on the STM32 microcontroller.

LED:

An LED is connected to a GPIO pin on the microcontroller. The LED is controlled based on the digital value read from the digital output pin of the Hall sensor.

UART (Universal Asynchronous Receiver-Transmitter):

UART is configured for serial communication. This allows the microcontroller to transmit messages over a serial interface, which can be useful for debugging or communication with other devices.

3.6.1 Code Overview:

System Initialization:

The system clock is configured using the SystemClock_Config function. GPIO pins for the Hall sensor and LED are initialized using the MX_GPIO_Init function. UART interfaces (USART1 and USART2) are initialized using the MX_USART1_UART_Init and MX_USART2_UART_Init functions.

ADC Configuration: The configureADC function initializes the ADC peripheral to read The analog output from the Hall sensor. The ADC is configured for 12-bit resolution, right alignment, and a single conversion mode.

LED Configuration:

The configureLED function initializes the GPIO pin connected to the LED as an output.

Main Loop:

The main loop continuously performs the following steps:

Reads the digital value from the Hall sensor's digital pin.

Reads the analog value from the Hall sensor using the ADC.

Controls the LED based on the digital value and transmits messages over UART.

UART Communication:

Messages are transmitted over UART to provide information about whether a magnetic field is detected or not. The messages include the digital value and the ADC reading.

3.7 Operation:

Sensor Reading:

The analog output from the Hall sensor is read using the ADC.The digital output from the sensor is read to determine the presence or absence of a magnetic field.

LED Control:

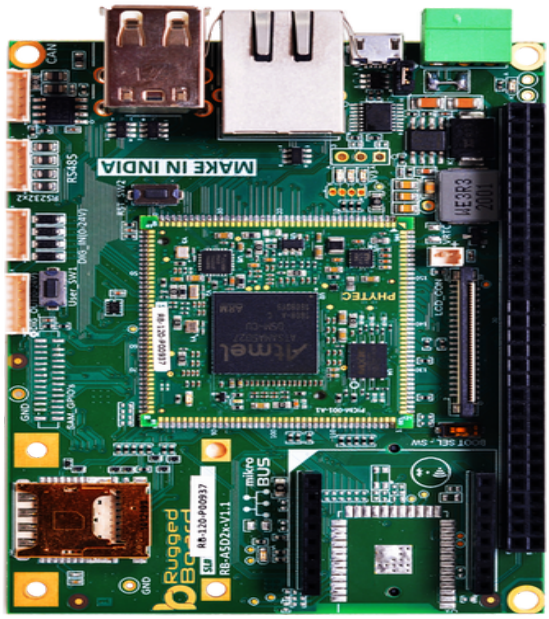
If a magnetic field is detected (digital value is high), the LED is turned on.

If no magnetic field is detected (digital value is low), the LED is turned off.

3.8 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.



Rugged Board a5d2x

3.9 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

```c
void WE10_Init (char *SSID, char *PASSWD)
{
    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_Delay(2000);
    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_Delay(2000);
    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=Anusha,chinnu321\r\n", SSID, PASSWD);
    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_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
   /********* CMD?WIFI**********/
    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_Delay(2000);
    HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
}
```

1. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
2. The next few lines of code send the CMD+RESET command to the WE10 module. This command resets the module to its default state.
3. The next line of code sends the CMD+WIFIMODE=1 command to the WE10 module. This command sets the module to operate in WiFi mode.
4. The next line of code sends the CMD+CONTOAP=SSID, PASSWD command to the WE10 module. This command configures the module to connect to the WiFi network with the specified SSID and password.
5. The next line of code sends the CMD.WIFI command to the WE10 module. This command queries the module for its WiFi status.
6. The last line of code waits for 2000 milliseconds and then receives response from the WE10 module. The response is stored in the buffer.
7. The WE10_Init() function is a simple example of how to initialize a WE10 module and connect it to a WiFi network. The function takes no arguments and it returns void

4 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 Stage 1 : Connection between stm32 and sensor module

Required parameters:

1.microcontroller

2.Linear hall Sensor(KY-024).

3. Jumpers wires

Connections:

1. Digital Pin -> PB13

2. VCC - > +5v

3. GND -> GND

4.Analog Pin -> PA0

Connection Diagram

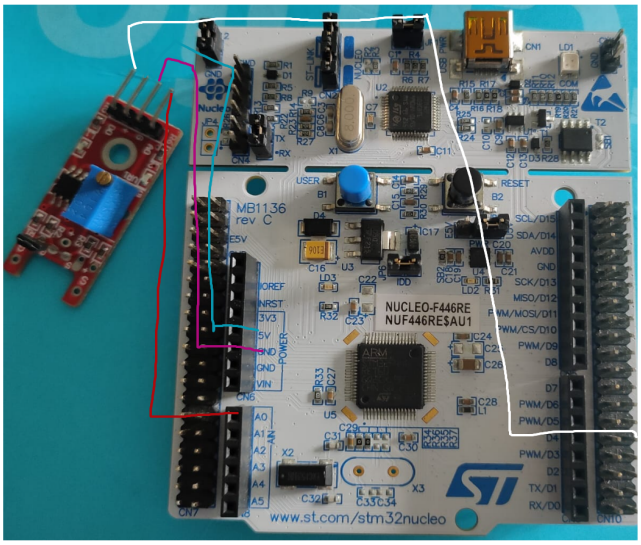
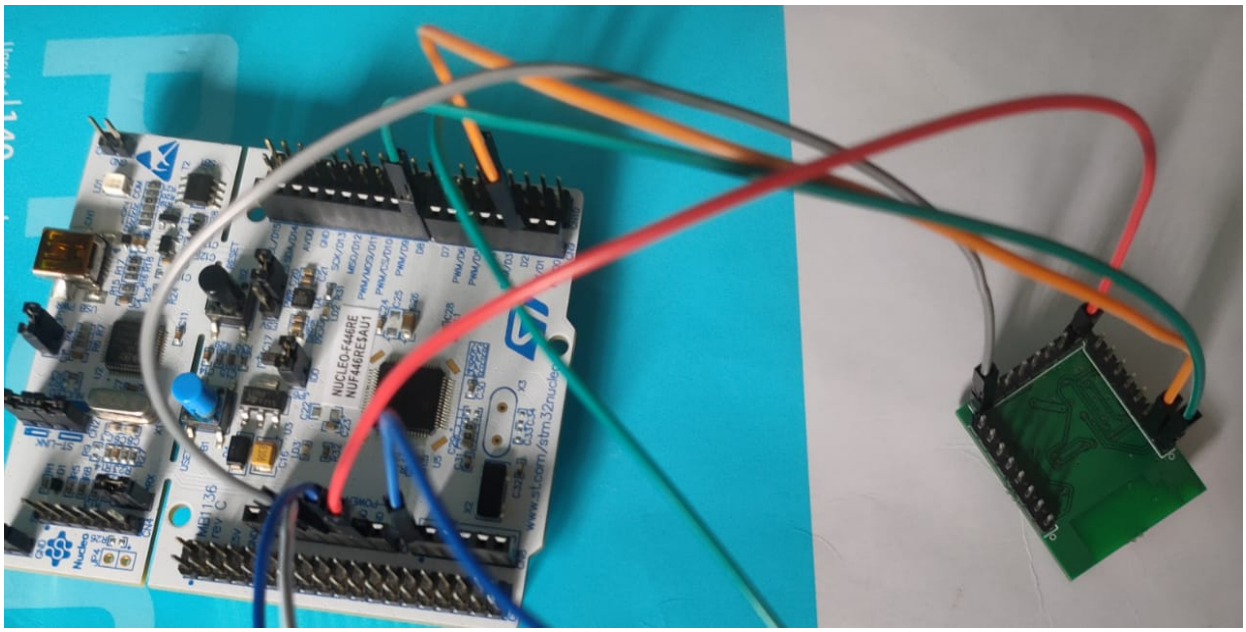


Fig: stage1 pin connections

STM32F446RE W10



4.1 Pin Configuration:

PIN	PIN NUMBER	COLOUR	COMPONENTS
Vcc 5v	vcc	Grey,Blue	W10 and KY-024
ground	GND	Red,Pink	W10 and KY-024
Analog	PA0	Red	KY-024
Digital	PB13	White	KY-024
Tx	PA9	Green	W10
Rx	PA10	Orange	W10

TABLE –1

4.2 Stage 2: Connection between stm32 and sensor module by using wifi module

Required parameters

1.microcontroller 2.Linear hall Sensor(KY-024)

3. Jumper wires 4.wifi module

Connection between stm32 to sensor module:

1. Digital Pin -> PB13 2. VCC - > +5v

3. GND -> GND 4.Analog Pin -> PA0

Connection between stm32 to wifi module:

1.RX -> D8(PA9) 3.VCC->3V

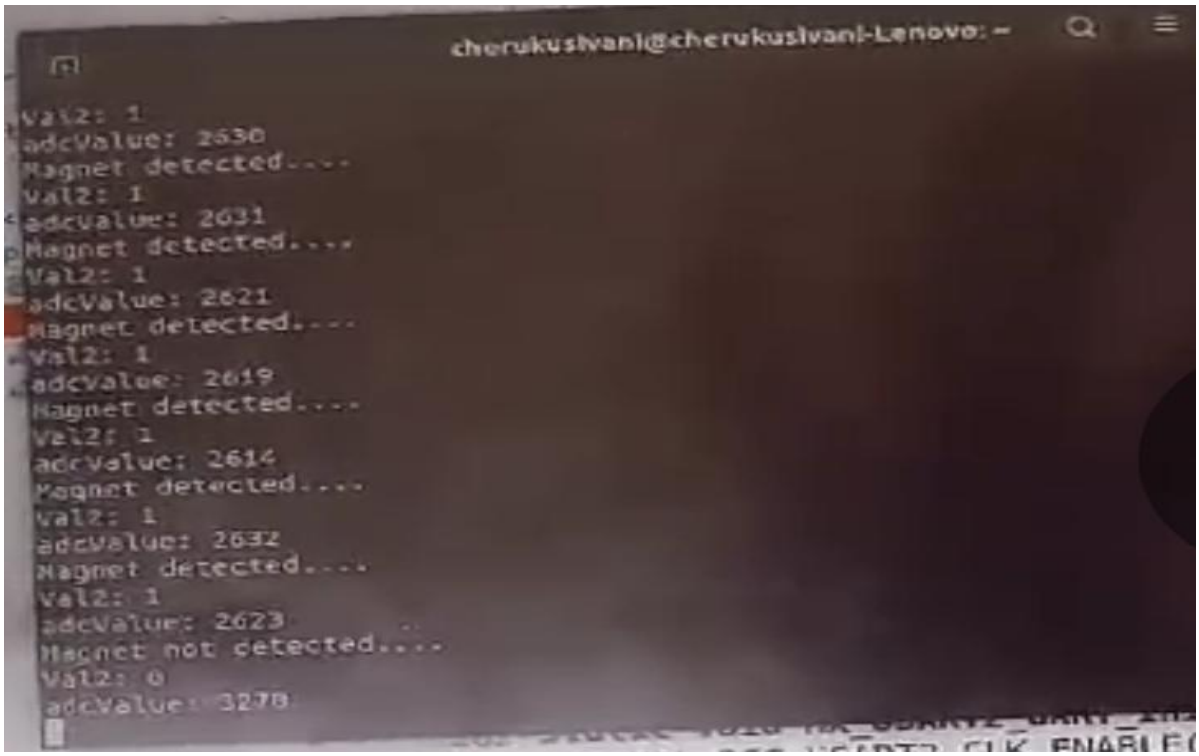
2.TX -> D2(PA10 4.GND->GND

Code:

```
while (1) {

// Read digital value from the Hall Sensor's digital pin

readSensor();

uint8_t sensorStatus = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13);

Val2 = HAL_GPIO_ReadPin(Hall_Sensor_D_GPIO_Port, Hall_Sensor_D_Pin);

printf("Digital value: %d\r\n", (Val2 == GPIO_PIN_SET) ? 1 : 0);

if (Val2 == GPIO_PIN_SET) {

// If Val2 is equal to 1 (digital high), turn on the LED

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

sprintf(message, "magnet detected val2:%d\r\n adcValue: %d\r\n", Val2,adcValue);

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

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

} else {

// If Val2 is not equal to 1 (digital low), turn off the LED

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

sprintf(message, "magnet not detected Val2:%d\r\n adcValue: %d\r\n", Val2,adcValue);

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);

    }

}
```

Output



CODE

```
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_Delay(50);

HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);

HAL_Delay(50);

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_Delay(50);
```

```
HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);

HAL_Delay(50);

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=Anusha,chinnu321\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(50);

HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);

HAL_Delay(50);

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_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 **/
sprintf(&buffer[0],"CMD+MQTTCONCFG=3,mqtt-cherukusivani-kfggbi,,,,,,,,\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);

}

CC_OscInitStruct.PLL.PLLR = 2;

if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)

{

Error_Handler();

}
```

4.3 Stage 3: Steps to connect stm32 to rugged board to control KY-034 flash led

Required parameters:

1.microcontroller 2.Linear hall Sensor(KY-024)

3. Jumper wires 4.Rugged board.

Connection between stm32 to sensor

1.Digital Pin -> PB13 2.VCC - > +5v.

3.GND -> GND. 4.Analog Pin -> PA0

Connection between stm32 to rugged board :

1.By using stm32 uart1 the PA9 pin is connected to the rugged board microbus (uart3) RX pin.

2.In rugged board GND pin is connected to stm32 GND.

4.4 Stage 4: Steps to connect stm32 to rugged board to control KY-034 flash led BY Using righttech:

Required parameters:

1.microcontroller 2.Linear hall Sensor(KY-024)

3. Jumper wires 4.Rugged board. 5.WIFI module

Connection between stm32 to sensor :

1.Digital Pin -> PB13 2.VCC - > +5v.

3.GND -> GND. 4.Analog Pin -> PA0

Connection between microcontroller and rugged board: In rugged board we have to take one UART for transmit the status of Linear hall sensor (KY-024) from STM32 to rugged board.we have to take micro bus for uart.Connect From stm32 PA0 pin to rugged board receiver pin in micro bus.

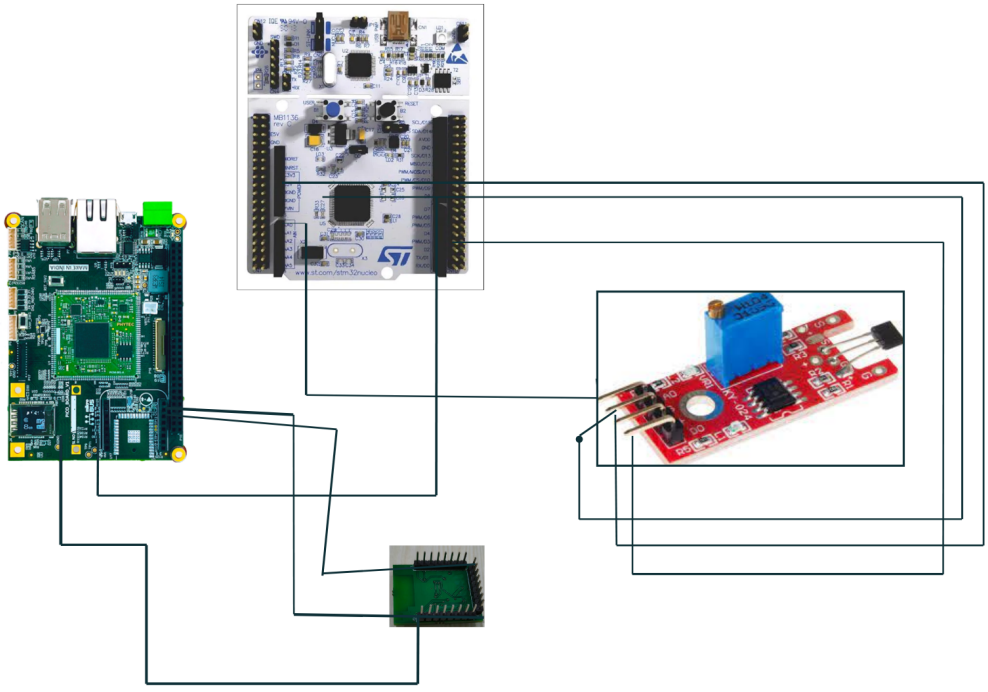


Fig: Pin Connections

Code for STM32

```
#include "main.h"

#include "stdio.h"

#include "string.h" // Added for strlen()

uint8_t sensorStatus;

uint16_t adcValue;

// Include the MQTT library for your microcontroller and Rightech platform

#define LED_PIN GPIO_PIN_5 // Replace with the GPIO pin for the LED

#define LED_PORT GPIOA // Replace with the GPIO port for the LED

ADC_HandleTypeDef hadc; // Declare an ADC handle structure

GPIO_InitTypeDef GPIO_InitStruct = {0};

#define Hall_Sensor_Pin GPIO_PIN_5 // Replace with your assigned pin for the analog output

#define Hall_Sensor_GPIO_Port GPIOA // Replace with the GPIO port corresponding to the analog pin

#define Hall_Sensor_D_Pin GPIO_PIN_13 // Replace with your assigned pin for the digital output

#define Hall_Sensor_D_GPIO_Port GPIOB // Replace with the GPIO port corresponding to the digital pin

UART_HandleTypeDef huart2;

UART_HandleTypeDef huart1;
```

```
// Include necessary Wi-Fi and MQTT configuration here

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_USART2_UART_Init(void);

static void MX_USART1_UART_Init(void);

int Val2 = 0;

int ledState = 0; // 0 for off, 1 for on

void configureADC() {

HAL_Init();

// Initialize ADC peripheral

__HAL_RCC_ADC1_CLK_ENABLE();

hadc.Instance = ADC1;

hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;

hadc.Init.Resolution = ADC_RESOLUTION_12B;

hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;

hadc.Init.ScanConvMode = DISABLE;

hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;

HAL_ADC_Init(&hadc);

ADC_ChannelConfTypeDef sConfig;
```

```
sConfig.Channel = ADC_CHANNEL_0; // Assuming you've connected the sensor to PA0

sConfig.Rank = 1;

sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;

HAL_ADC_ConfigChannel(&hadc, &sConfig);

}

void readSensor() {

HAL_ADC_Start(&hadc);

HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);

adcValue = HAL_ADC_GetValue(&hadc);

HAL_ADC_Stop(&hadc);

}

void configureLED() {

__HAL_RCC_GPIOA_CLK_ENABLE(); // Enable GPIOA clock

GPIO_InitStruct.Pin = GPIO_PIN_5; // Assuming you're using PA5 for the LED

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);

}
```

```
int main(void) {
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 MX_USART1_UART_Init();
 configureADC();
 configureLED();
 //WE10_Init();
 //MQTT_Init();
// Configure LED pin
 char message[50];
 // Connect to Wi-Fi here
 // Initialize MQTT client here
 while (1) {
 // Read digital value from the Hall Sensor's digital pin
 readSensor();
 uint8_t sensorStatus = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13);
Val2=HAL_GPIO_ReadPin(Hall_Sensor_D_GPIO_Port,Hall_Sensor_D_Pin);
```

```
        printf("Digital value: %d\r\n", (Val2 == GPIO_PIN_SET) ? 1 : 0);

        if (Val2 == GPIO_PIN_SET) {

        // If Val2 is equal to 1 (digital high), turn on the LED

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

        sprintf(message, "magnet detected val2:%d\r\n adcValue: %d\r\n",
        Val2,adcValue);

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

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

        } else {

        // If Val2 is not equal to 1 (digital low), turn off the LED

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

        sprintf(message, "magnet not detected Val2:%d\r\n adcValue: %d\r\n",
        Val2,adcValue);

        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);

}
```

```
}

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_VOL
TAGE_SCALE3);

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();
```

```
}

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();

}

}

// System Clock Configuration

// Configure the system clock as needed.

 static void MX_GPIO_Init(void) {

__HAL_RCC_GPIOA_CLK_ENABLE(); // Enable the clock for the GPIO
port used by the analog pin

__HAL_RCC_GPIOB_CLK_ENABLE(); // Enable the clock for the GPIO
port used by the digital pin

GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
// Configure the Hall effect sensor digital pin

GPIO_InitStruct.Pin = Hall_Sensor_D_Pin;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL;

HAL_GPIO_Init(Hall_Sensor_D_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : PB10 */

GPIO_InitStruct.Pin = GPIO_PIN_10;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL;

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

// You may need to configure the analog pin as an ADC input here

// Configure any other pins as needed

// Configure the UART pins for serial communication

// This depends on the specific UART and pins you're using

}

static void MX_USART2_UART_Init(void) {

__HAL_RCC_USART2_CLK_ENABLE();

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_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();
}
}
void Error_Handler(void) {
while (1) {
// An error occurred, stay in this loop.
}
}
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line) {
while (1) {
// If you want to debug, place a breakpoint here.
}
}
#end
```

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);
tty.c_cflag &= ~CSIZE;
tty.c_cflag |= CS8;
```

```
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 (testator(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=\"realme X7 Max\",\"0987654321\"\r\n";

char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";

char arr4[] = "CMD+MQTTCONCFG=3,mqtt-cherukusivani-kfggbi,,,,,,,,,\r\n";

char arr5[] = "CMD+MQTTSTART=1\r\n";

char arr6[] = "CMD+MQTTSUB=base/relay/led1\r\n";

char arr7[] = "CMD+MQTTPUB=sensor/voltage\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);

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);
```

intret=snprintf(buffer,sizeof(buffer),"CMD+MQTTPUB=reading/sensorval,%s\r\n", buf);

```
        if (ret < 0) {

        } else {

        ssize_t wlen = write(fd, buffer, ret);

        sleep(3);

        if (when == -1) {

        }}}

        } close(fd);

        return 0;
```

Output

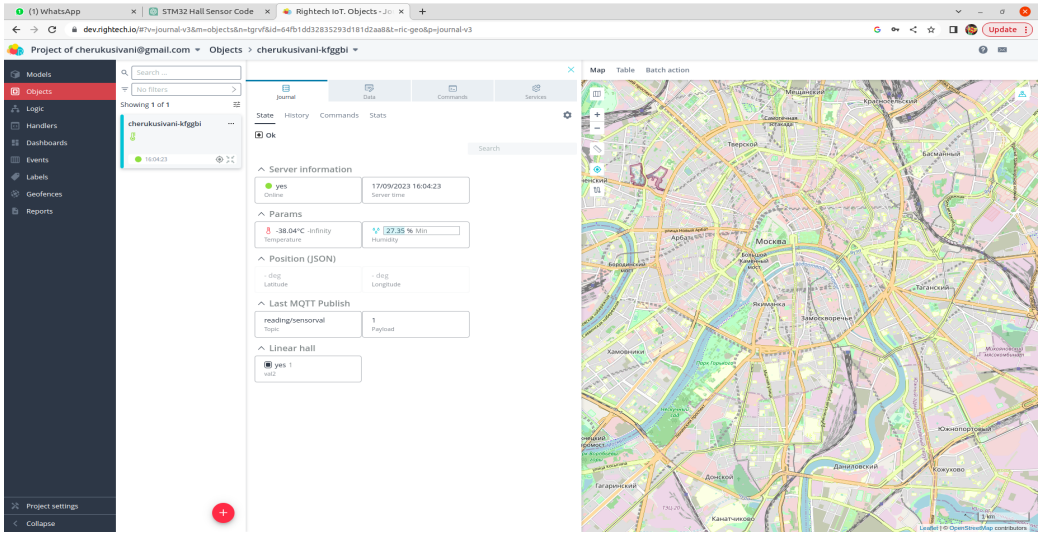


Fig : Magnet Detected

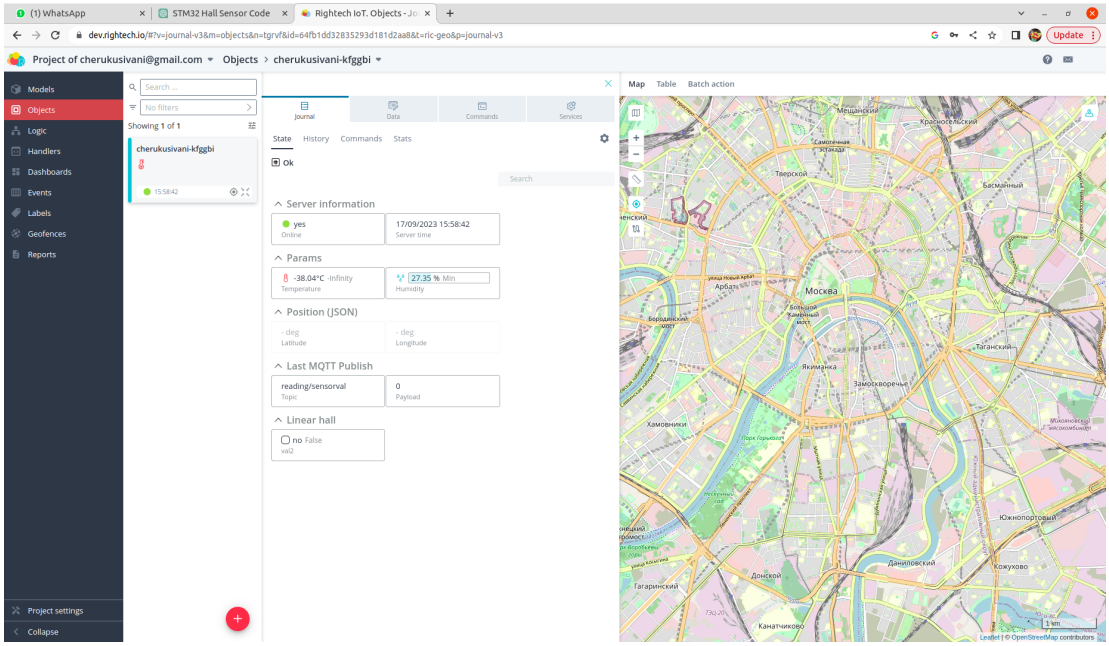


Fig 2: Magnet Not Detected

5. References

1.https://arduinomodules.info/ky-035-analog-hall-magnetic-sensor-module/

2. https://datasheetspdf.com/pdf/1402045/Joy-IT/KY-035/

3. https://sensorkit.joy-it.net/en/sensors/ky-035/

4.https://www.st.com/en/microcontrollers-microprocessors/stm32f446re.html/