HOME AUTOMATION USING MQTT LIKE THINGS BOARD

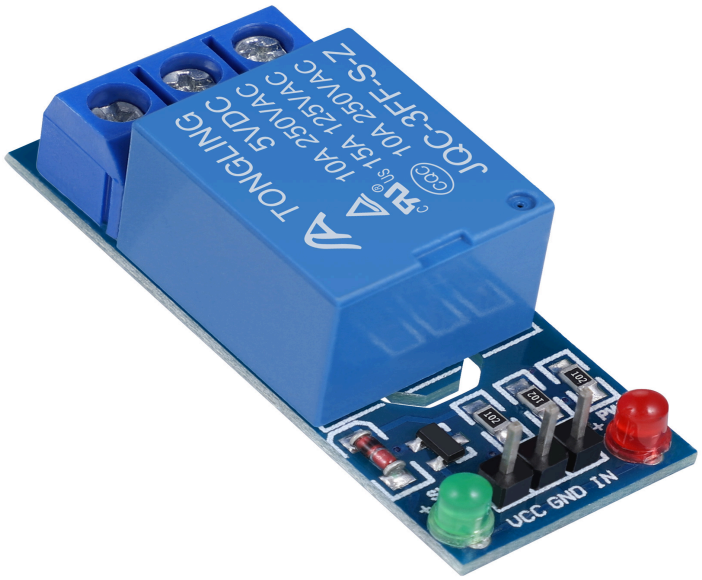
PRESENTED BY

Aravind.P
Prem Kumar.K
Saisitha.P

Table of content

1.Introduction

A relay is an electrical switch that is operated by an electromagnet. It is a device designed to open or close one or more contacts to control the flow of electrical current in a circuit. Relays are commonly used in electrical and electronic systems to enable low-power signals to control higher-power devices, allowing for the automation and remote control of various processes. The key components of a relay include:



Signal Amplification and Isolation: Relays can amplify weak electrical signals and provide isolation between the input and output circuits. This is valuable in scenarios where the control signal comes from a low-power source or needs to be separated from the high-power load.

Remote Control: Relays enable remote control of electrical devices. By using a low-power control signal, often from a microcontroller or a switch, a relay can open or close a circuit that controls a more significant load located at a distance.

Switching High-Power Devices: Relays are commonly employed to control high-power electrical devices, such as motors, heaters, or lights, using a lower-power control signal. This allows for the safe and efficient operation of various electrical systems.

Automotive Applications: Relays are widely used in automotive systems to control functions like headlights, starter motors, and cooling fans. They help manage the high currents associated with these components.

Protection and Safety: Relays can be integrated into circuits to provide protection and safety features. For example, they can be part of overcurrent protection systems, disconnecting power in case of a fault.

Industrial Automation: In industrial settings, relays play a crucial role in the automation of processes, allowing control systems to manage complex sequences of events and control multiple devices simultaneously.

There are different types of relays, including electromagnetic relays, solid-state relays, and reed relays, each with specific advantages and use cases. The choice of relay depends on factors such as the application's requirements, switching speed, and environmental conditions.

2.Key Features of the Project:

MQTT Configuration:

The paho library is used for MQTT communication.
The ACCESS_TOKEN variable holds the ThingsBoard device access token.
The MQTT broker's host (broker) and port (port) are specified.

Usage:
Replace the placeholder access token ('5VNLaN6qQUKMKhTnWzVf') with the actual access token of your ThingsBoard device.
Adjust the MQTT broker's host (broker), port (port), and the delays in the loop as needed for your application.
This script essentially creates a simple environmental monitoring system, where the Relay sensor is used to the data is sent to ThingsBoard for visualization and analysis.

ThingsBoard Platform:

Device Management: ThingsBoard provides a device management interface to add and manage devices in the platform.

Device Configuration:

Device Type and Attributes: Define the device type and attributes associated with the bulb. Attributes could include properties like status, brightness, and color.

MQTT Integration:

MQTT Broker Configuration: Configure ThingsBoard to connect to your MQTT broker. You'll need to provide the MQTT broker's connection details, such as server address, port, and credentials.

Device Connectivity:

MQTT Device Connectivity: Configure the bulb as an MQTT device in ThingsBoard. Specify the MQTT topics for device telemetry (sensor data) and attributes (device configuration).

Bulb Control Panel:

Dashboard Creation: Create a dashboard in ThingsBoard to visualize and control the bulb. Add widgets for on/off switches, sliders for brightness control, and color pickers if the bulb supports color changes.

MQTT Topics:

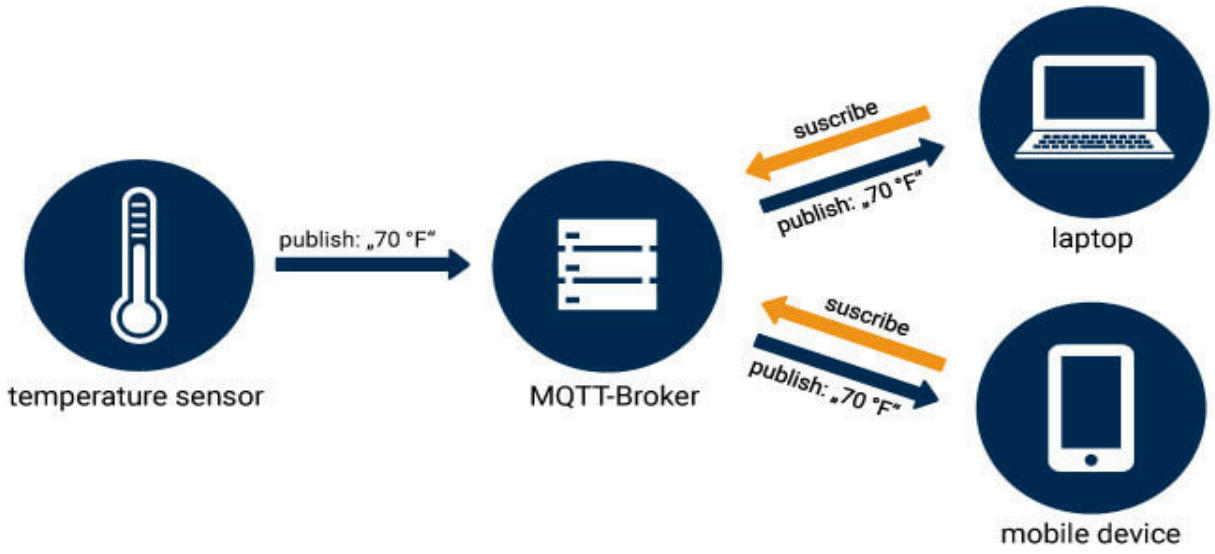
Define Topics: Determine the MQTT topics for sending commands to control the bulb. Common topics include topics for turning the bulb on/off, adjusting brightness, and changing color.

MQTT Communication:

Publish and Subscribe:
Configure the bulb to publish its status (telemetry) to ThingsBoard on specific MQTT topics. Also, configure ThingsBoard to subscribe to the relevant MQTT topics for receiving control commands.

Rule Engine (Optional):

Custom Logic: ThingsBoard has a rule engine that allows you to define custom logic based on incoming telemetry or events. You can use this to implement automation or custom actions when certain conditions are met.



Security:
Secure Connection: Ensure that your MQTT connection is secure, especially if the IoT devices are deployed in production environments. Use secure protocols (such as MQTT over TLS/SSL) and implement proper authentication mechanisms.

Monitoring and Logging:

Logging: Set up logging and monitoring within ThingsBoard to track device communication, diagnose issues, and ensure the system's reliability.
Testing:

Device Simulation: For testing purposes, you can simulate the behavior of the bulb by using simulated devices in ThingsBoard before deploying it with physical bulbs.

Documentation:
Documentation and Support: Refer to the official ThingsBoard documentation and community forums for detailed guides, troubleshooting, and support. Always refer to the latest documentation provided by ThingsBoard, as configurations and features may evolve over time. Additionally, consider the specific capabilities and protocols supported by the MQTT-enabled bulb you are using.

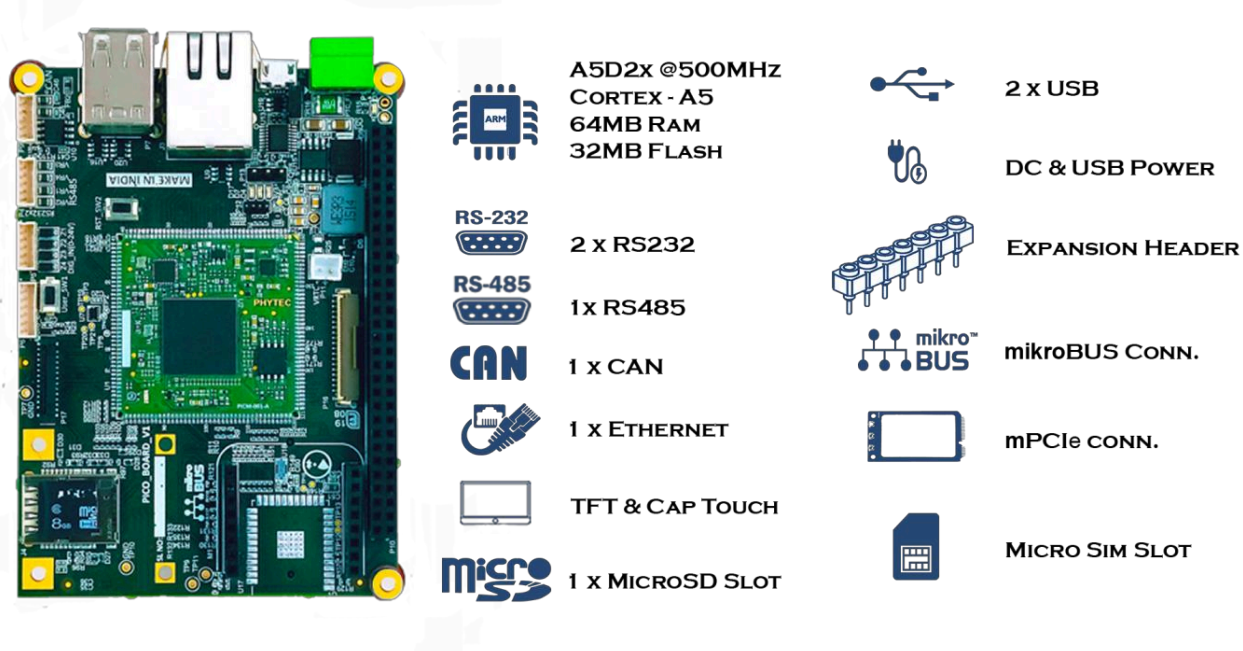
3.Hardware used and there Specification

Rugged Board - A5D2x is an Single Board Computer providing as easy migration path from Micro controller to Microprocessor. Rugged Board 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 ARMCortex-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 Input 4x 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



Relay:-

A relay is an electrical switch that is operated by an electromagnet. It is a device designed to open or close one or more contacts to control the flow of electrical current in a circuit. Relays are commonly used in electrical and electronic systems to enable low-power signals to control higher-power devices, allowing for the automation and remote control of various processes. The key components of a relay include:

Electromagnet (Coil): The coil of wire is wound around a core, often made of iron. When an electrical current flows through the coil, it creates a magnetic field.

Armature: Connected to the coil is an armature, a movable iron component. The armature is attracted to the magnetic field created by the coil.

Contacts: The armature is mechanically linked to a set of electrical contacts. These contacts can be either normally open (NO), normally closed (NC), or both, depending on the type of relay.

- Normally Open (NO): In the resting state (when the relay is not energized), the contacts are open, and no electrical current flows through them.
- Normally Closed (NC): In the resting state, the contacts are closed, allowing electrical current to flow through the circuit.

Spring: Relays often have a spring or other mechanism to return the armature and contacts to their original position when the coil is de-energized.

Operation:

- When an electrical current is applied to the coil (energizing the relay), the coil generates a magnetic field, which attracts the armature.
- The movement of the armature causes the contacts to change their state. In a Normally Open (NO) relay, the contacts close, completing the circuit. In a Normally Closed (NC) relay, the contacts open, interrupting the circuit.
- When the coil is de-energized, the spring returns the armature to its original position, and the contacts return to their resting state.

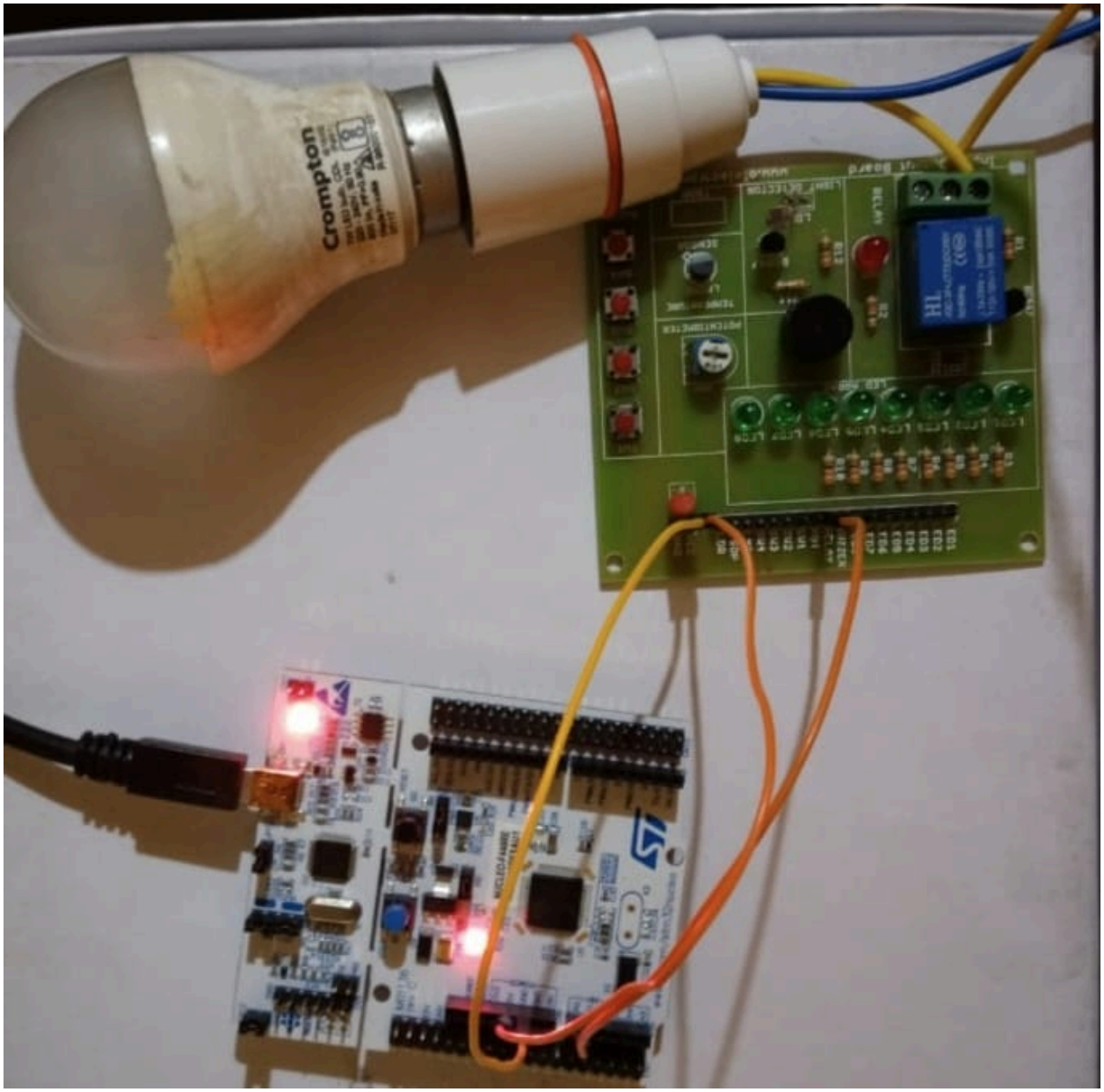
Applications:

- Relays are widely used in various applications, such as industrial automation, automotive systems, control circuits, and electronic devices.

- They allow a low-power control signal, like one from a microcontroller or a switch, to manage the operation of high-power devices or circuits.

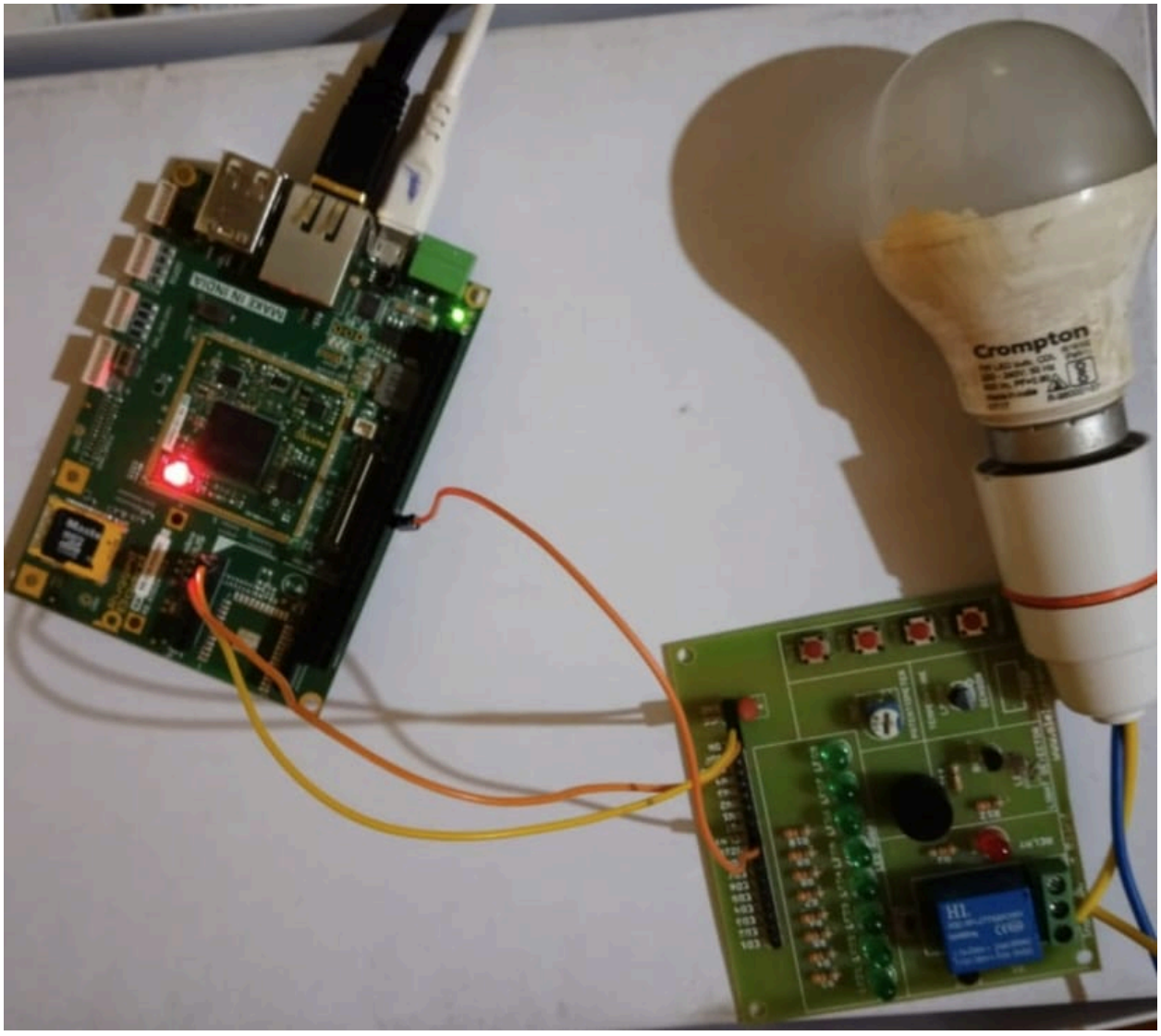
Relays are essential for providing electrical isolation, protecting sensitive control circuits, and facilitating the control of diverse electrical systems. They come in various types, including electromagnetic relays, solid-state relays, and others, each suited to specific applications and requirements.

4.Connection Diagram



Stm32f446	RELAY

VCC 5V	VCC
GROUND	GND
PA0	Signal



PIN CONFIGURATION TABLE

RUGGED BOARD	RELAY
VCC 5V	VCC
GROUND	GND
PA31	Signal

5.Project Code

Interfacing the relay with Stm32f446 board

Stage 1

```
/* USER CODE BEGIN Header */
/**
  ****************************
  * @file           : main.c
  * @brief          : Main program body
  ****************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
```

```
  *
  **************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

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

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
```

```
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

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

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

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

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

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
          HAL_Delay(2000);

          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
          HAL_Delay(2000);
```

```
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

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

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

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

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);

  /*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);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
```

```
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Stage 2:-

Interfacing with Ruggedboard

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define RELAY_PIN "/sys/class/gpio/PA31/value"
```

```
void error(const char *msg) {
    perror(msg);
    exit(1);
}

void setRelayState(int state) {
    FILE *fp = fopen(RELAY_PIN, "A");
    if (fp == NULL) {
        error("Error opening relay pin");
    }

    fprintf(fp, "%d", state);  // 0 for OFF, 1 for ON
    fclose(fp);
}

int main() {
    // Assuming GPIO pin 31 is not exported, you might need to export it first
    FILE *exportFile = fopen("/sys/class/gpio/export", "A");
    if (exportFile == NULL) {
        error("Error exporting GPIO pin");
    }

    fprintf(exportFile, "31");
    fclose(exportFile);

    // Set the GPIO pin direction to out (for writing)
    FILE *directionFile = fopen("/sys/class/gpio/PC23/direction", "A");
    if (directionFile == NULL) {
        error("Error setting direction for GPIO pin");
    }

    fprintf(directionFile, "out");
    fclose(directionFile);

    while (1) {
        // Turn ON the relay (activate the connected device)
        setRelayState(1);
        printf("Relay is ON\n");
        sleep(2);  // Wait for 2 seconds

        // Turn OFF the relay (deactivate the connected device)
        setRelayState(0);
        printf("Relay is OFF\n");
        sleep(2);  // Wait for 2 seconds
```

```
    }

    // Unexport GPIO pin 31 before exiting
    FILE *unexportFile = fopen("/sys/class/gpio/unexport", "A");
    if (unexportFile == NULL) {
        error("Error unexporting GPIO pin");
    }

    fprintf(unexportFile, "31");
    fclose(unexportFile);

    return 0;
}
```

Stage 3:-

Interfacing Relay with Ruggedboard and checking the output in Things Board

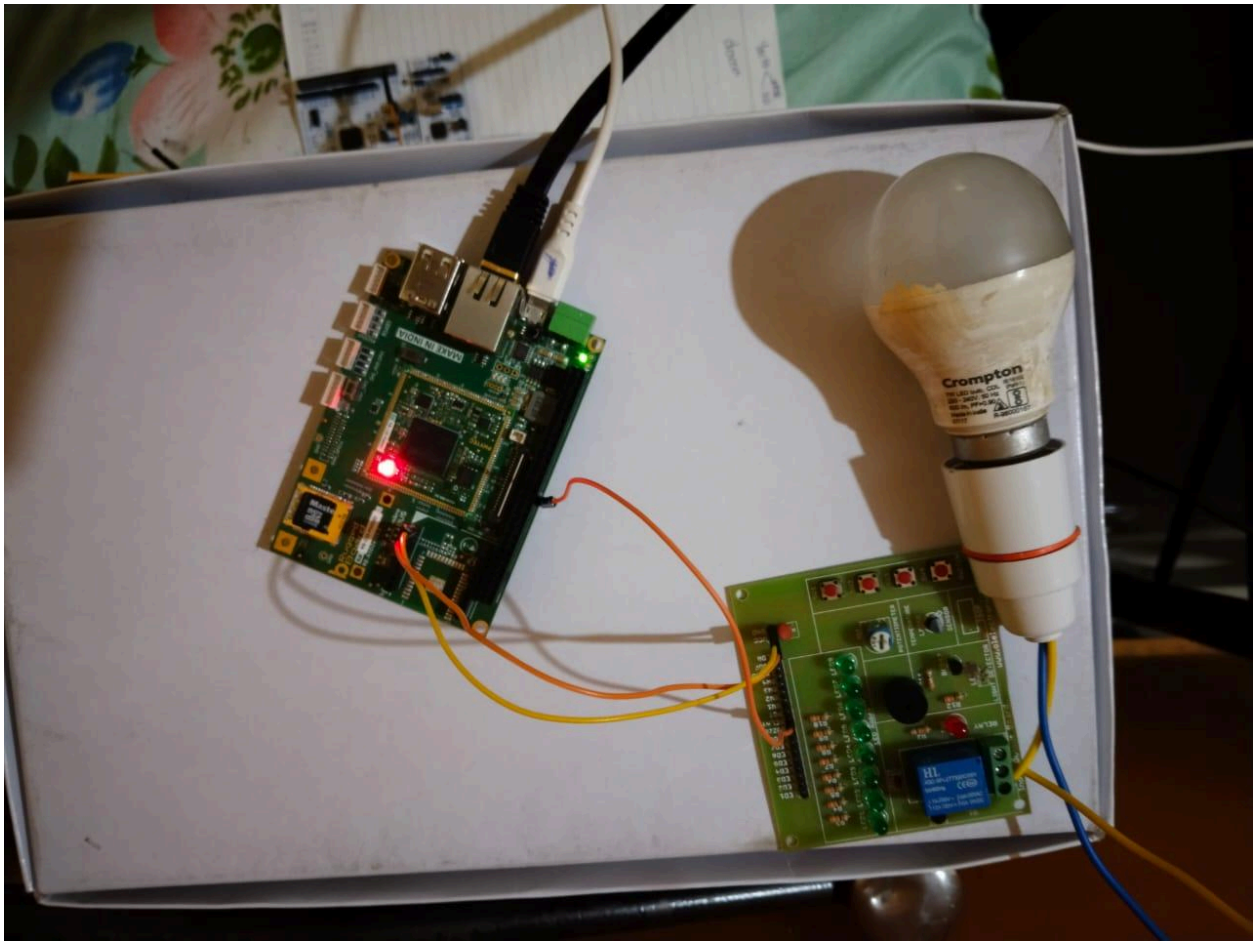
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <math.h>
#include "MQTTClient.h"
#define RELAY_PIN_PATH "/sys/class/gpio/PC23/value"

// MQTT Settings
#define ACCESS_TOKEN “ 5VNLaN6qQUKMKhTnWzVf”
#define MQTT_ADDRESS "tcp://phyclouds.com:8080"
#define MQTT_CLIENTID "iiscSmartSwitch"
#define MQTT_TOPIC "v1/devices/me/telemetry"
#define MQTT_QOS 1

// Global MQTT client variable
MQTTClient client;
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
```

```
// Function to publish data to MQTT
void publishToMQTT(char *topic, char *payload)
{
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    pubmsg.payload = payload;
    pubmsg.payloadlen = strlen(payload);
    pubmsg.qos = 1;
    pubmsg.retained = 0;
    MQTTClient_deliveryToken token;

    MQTTClient_publishMessage(client, topic, &pubmsg, &token);
    MQTTClient_waitForCompletion(client, token, 1000);

    int rc = MQTTClient_waitForCompletion(client, token, 10000);

    if (rc == MQTTCLIENT_SUCCESS)
    {
        // Print information about the published data
        printf("Published MQTT data - Topic: %s, Payload: %s\n", topic, payload);
    }
    else
    {
        fprintf(stderr, "Failed to publish file data. Return code: %d\n", rc);
    }
}

// Function to set relay state
void setRelayState(int state)
{
    FILE *fp = fopen(RELAY_PIN_PATH, "A");
    if (fp == NULL)
    {
        perror("Error opening relay pin");
        exit(EXIT_FAILURE);
    }

    fprintf(fp, "%d", state);  // Writing 0 turns the relay off, 1 turns it on
    fclose(fp);
}

int main()
{
    // MQTT initialization
```

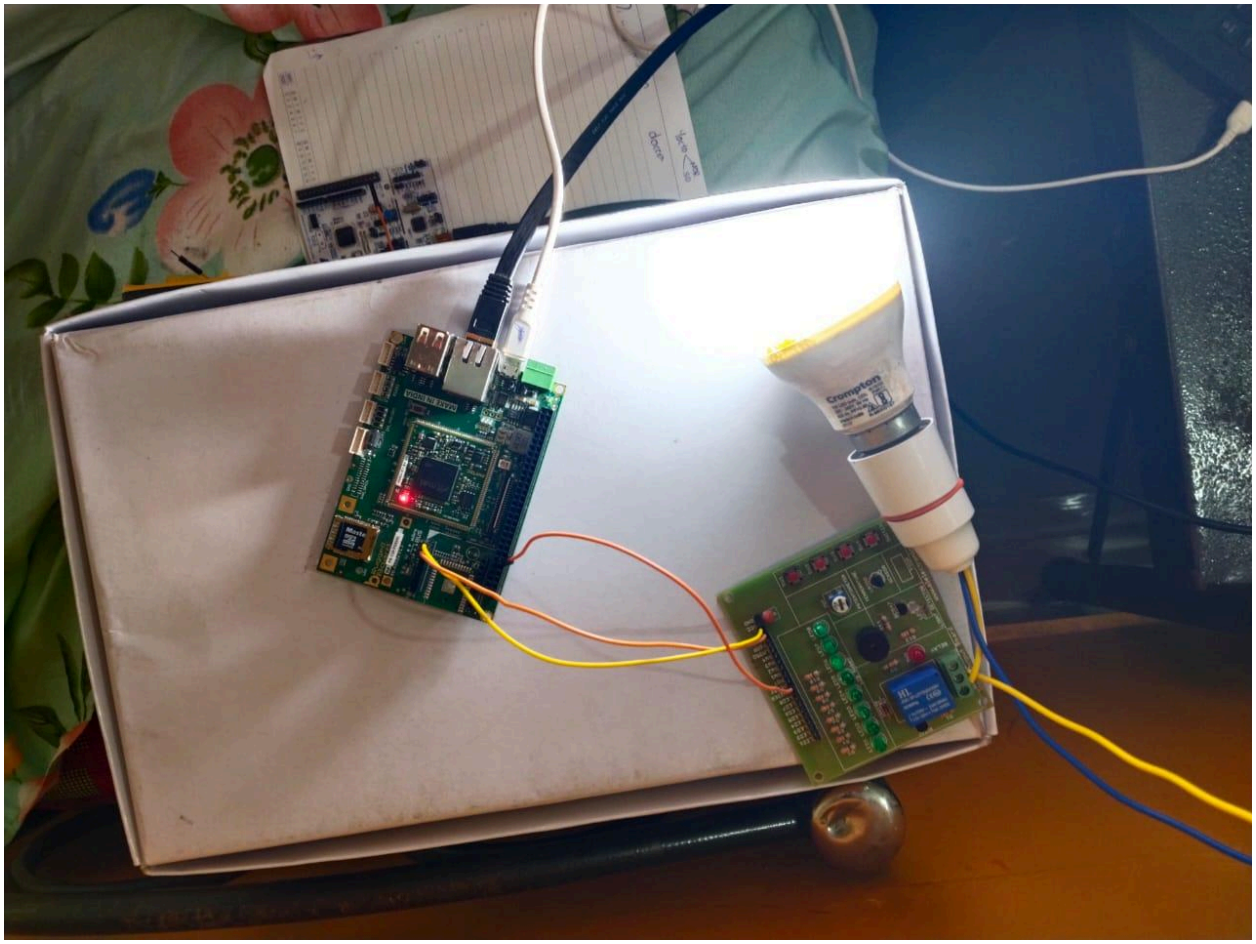
```
    MQTTClient_create(&client, MQTT_ADDRESS, MQTT_CLIENTID,
MQTTCLIENT_PERSISTENCE_NONE, NULL);
    conn_opts.keepAliveInterval = 20;
    conn_opts.cleansession = 1;
    conn_opts.username = ACCESS_TOKEN;

    if (MQTTClient_connect(client, &conn_opts) != MQTTCLIENT_SUCCESS)
    {
        fprintf(stderr, "Failed to connect to MQTT server\n");
        return -1;
    }

    while (1)
    {

        setRelayState(1);
            printf("Relay is ON\n");
        // Publish the relay status to ThingsBoard
        char mqtt_payload[500];
        snprintf(mqtt_payload, sizeof(mqtt_payload), "{\"RelayStatus\": ON}");
        publishToMQTT(MQTT_TOPIC, mqtt_payload);
            sleep(2);
            setRelayState(0);
        printf("Relay is OFF\n");
            snprintf(mqtt_payload, sizeof(mqtt_payload), "{\"RelayStatus\": OFF}");
        publishToMQTT(MQTT_TOPIC, mqtt_payload);
            sleep(2);

    }

    // Cleanup MQTT resources
    MQTTClient_destroy(&client);

    return 0;
}
```

6.Output:-

Off Condition



On Condition


Crompton