

SECRETARIA DE EDUCAÇÃO E CIÊNCIA
INSTITUTO POLITÉCNICO DE BRAGANÇA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

LICENCIATURA EM ENGENHARIA DE INFORMÁTICA
6º PERÍODO

FERNANDO SOUZA FURTADO CARRILHO
JOSÉ RAFAEL SOARES BORGES

RELATÓRIO PRÁTICO 2:
NODE-RED E O MICROCONTROLADOR ESP

BRAGANÇA
2023

FERNANDO SOUZA FURTADO CARRILHO
JOSÉ RAFAEL SOARES BORGES

RELATÓRIO PRÁTICO 2:
NODE-RED E O MICROCONTROLADOR ESP

Este relatório objetiva a obtenção de nota na disciplina de Internet das Coisas dos graduandos no curso de Engenharia de Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança. Seu conteúdo é composto pela observação, descrição e aplicação referente ao Protocolo MQTT por intermédio do Node-RED e microcontrolador ESP.

BRAGANÇA
2023

Sumário

1	Questão 03: Códigos de Blink e Blink Without Delay	5
2	Questão 04: Funções no Blink e Blink Wihtout Delay	8
3	Questão 07: Aplicação do MQTT na Placa ESP	9
4	Questão 08: Comunicação ESP com Node-RED	12
5	Questão 09: Comunicação ESP com Potenciômetro	20

Lista de Figuras

1	Código do exemplo Blink no Arduino	5
2	Código do exemplo Blink Without Delay no Arduino	6
3	Bloco de código Callback	9
4	Bloco de código Reconnect	10
5	Estrutura condicional do código	10
6	Código no Node-RED com Switch	12
7	Configuração no Nó do Switch no Node-RED	13
8	Respaldo do Switch no Dashboard do Node-RED	14
9	Saída do Payload em JSON no Node-RED do Switch	14
10	LED azul ligado na ESP por ação no Node-RED	15
11	LED azul desligado na ESP por ação no Node-RED	16
12	Atuação da função callback na ESP	17
13	Código <i>publish-subscribe</i> no Arduino ESP	18
14	Função <i>setup()</i> do código ESP	19
15	Fluxo do Node-RED em conexão ao ESP	20
16	Configuração de função no Node-RED	21
17	Simulação do ESP em conexão à um Potenciômetro	22
18	Saída no Node-RED dos dados publicado pelo ESP	22
19	Gauche no Dashboard do Node-RED em conexão ao ESP	23
20	Código de definição de mensagens no ESP	23

1 Questão 03: Códigos de Blink e Blink Without Delay

No Arduino há exemplos de códigos, dentre eles estão os de Blink e Blink Without Delay. Diante disso, para analisá-los, a seguir, está apresentado suas principais diferenças, as quais são que, o primeiro liga um LED por um segundo e após apaga por um segundo, ininterruptamente.

Já o segundo, liga o LED conectado a um pino digital, sem utilizar a função *delay*; o que permite que outro código possa ser executado simultaneamente sem ser interrompido pelo código destinado ao LED.

Para melhor compreensão, a seguir consta a Figura 1, a qual apresenta o código do Blink. E a posteriori, é possível ver a Figura 2, a qual referencia o código do Blink Without Delay: para efeito de comparação.

```
Blink.ino
1 void setup() {
2   pinMode(LED_BUILTIN, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(LED_BUILTIN, HIGH);
7   delay(1000);
8   digitalWrite(LED_BUILTIN, LOW);
9   delay(1000);
10 }
11
```

Figura 1: Código do exemplo Blink no Arduino

Diante disso, a efeito de compreensão, na Figura 1, acima apresentada, é possível ver a função *void setup()*, ela é responsável por configurar e executar uma vez quando se pressiona reset ou liga a placa.

Em seguida, a chamada de função *digitalWrite*, inicializa o pino digital *LED_BUILTIN* como uma saída. Após isso, a função *void loop()*, é executada e os comandos dentro do escopo da função repetirá indefinidamente, o que define o estado de looping.

Prontamente, vê-se ainda na Figura 1, o seu escopo, o que é visível as chamadas de funções *digitalWrite*, a qual faz uma escrita digital, e as chamadas de funções *delay*, que

se destina a fazer aguardar.

Consoante, ao atentar a ordem das funções dentro do escopo da figura em questão, a primeira chamada da função *digitalWrite* liga o LED, o delay faz esperar 1 segundo, a segunda função *digitalWrite* desliga o LED e espera mais um segundo, feito isso a função *loop* se repetirá indefinidamente.

Dessa forma, torna-se transparente que o exemplo do Blink, Figura 1, toda a sua funcionalidade destina-se a fazer o LED piscar, sem a atuação simultânea de outra funcionalidade. Para isso, está a Figura 2, abaixo apresentada, a qual além de manter o LED piscando, já a outra funcionalidade implementada que será expressa a posteriori.

```
BlinkWithoutDelay.ino  BlinkWithoutDelay.ino
1  const int ledPin = LED_BUILTIN;
2  int ledState = LOW;
3  unsigned long previousMillis = 0;
4  const long interval = 1000;
5
6  void setup() { pinMode(ledPin, OUTPUT); }
7
8  void loop() {
9
10     unsigned long currentMillis = millis();
11
12     if (currentMillis - previousMillis >= interval) {
13         previousMillis = currentMillis;
14
15         if (ledState == LOW) ledState = HIGH;
16         else ledState = LOW;
17
18         digitalWrite(ledPin, ledState);
19     }
20 }
21
```

Figura 2: Código do exemplo Blink Without Delay no Arduino

Conforme vê-se na Figura 2, as quatro primeiras linhas se referem às variáveis para atuação no sistema. À vista disso, é válido apontar que a variável *ledPin* se destina a número do pino do LED, enquanto a variável *ledState* é para definir o estado do LED, sendo *LOW*, baixo e *HIGHT*, alto.

Consoante, as duas demais variáveis, respectivamente, atua em que a variável *previousMillis* armazene a última atualização que o LED recebeu e a variável *interval* constitua

o intervalo para o LED piscar, em milissegundos.

Em continuidade em referência à análise do código, conforme apresentado na Figura 1, na Figura 2, o *void setup()* faz a mesma funcionalidades, ela é responsável por configurar e executar uma vez quando se pressiona reset ou liga a placa.

Em seu escopo, da função *setup*, a chamada de função *pinMode* define o pino digital como saída. Dessa forma, partindo para a função *loop*, que será repetida indefinidamente, ela se inicia com a criação de uma variável nomeada em *currentMillis*, a qual define o intervalo de tempo, em milissegundos, para o LED piscar.

Em seguida, é visto no código da Figura 2, a existência de uma estrutura condicional, *if*, a qual verifica se a diferença dos valores da variáveis *currentMillis* e *previousMillis* é maior ou igual ao valor da variável *interval*. Diante disso, sendo esta verificação for falsa, seu escopo é ignorado, caso contrário, se a verificação for verdadeira, seu escopo é executado.

Diante disso, em seu escopo, a variável *previousMillis* armazena a última vez em que o LED piscou. Posto isso, é verificado, condicionalmente, se o estado do LED é apagado ou aceso, se for apagado, a variável *ledState* recebe alto, agora se for aceso, recebe apagado. Feito isso, por fim, a chamada de função *digitalWrite* recebe e inscreve os valores das variáveis *ledPin* e *ledState*.

2 Questão 04: Funções no Blink e Blink Without Delay

Dentro dos códigos existentes nos exemplos do Blink e Blink Without Delay, há funções e seus atributos, entretanto, será citado, sobre suas diferenças, as de ***void setup()*** e ***void loop***. Posto isso, para melhor compreender os motivos de utilizá-las, é preciso entender o que fazem estas funções, isto é, quais suas funcionalidades.

Em primeiro plano, com foco na abstração da função ***void setup***, é possível dizer que, conforme o próprio site do arduino afirma, (Arduino.cc, b), esta função é usada para inicializar variáveis, configurar o modo dos pinos (INPUT ou OUTPUT), inicializar bibliotecas, entre outras funcionalidades fundamentais para preparação do código que se deseja implementar.

Consoante, é válido dizer que a função é executada apenas uma vez, após a placa ser alimentada ou caso um reset seja aplicado. É por este motivo que as variáveis fundamentais devem ser iniciadas/instanciadas nesta função.

Por conseguinte, em segundo plano, a função ***void loop*** pode ser explicada, pelo próprio site do Arduino, (Arduino.cc, a), como a função que faz precisamente o que o seu nome sugere, loop, laço ininterrupto de repetição.

Dessa forma, isso implica que o escopo desta função se repete consecutivamente enquanto a placa estiver ligada, permitindo o seu programa mudar e responder a essas mudanças. Use-a para controlar ativamente uma placa Arduino.

Em outras palavras, ***void loop*** implica que é nela o local onde se aplica o escopo do código que precisa estar em execução de maneira integral e ininterrupta, isto é, o bloco de código que é preciso que esteja em atuação repetidamente; ela é usada para controlar ativamente a placa desejada, a exemplo Arduino Nano.

Ademais, é possível ver que em ambos os exemplos, Blink e Blink Without Delay, precisaram utilizar as funções ***void setup*** e ***void loop***, uma vez que ambos precisam de uma função que armazene os valores principais para serem utilizados no código e um ambiente que se repita indefinidamente para que seus objetivos sejam concluídos.

3 Questão 07: Aplicação do MQTT na Placa ESP

Na aplicação do MQTT na Placa ESP, da atividade de Ficha 2 em referência Internet das Coisas, houve a explanação de códigos C, o qual pode ser acessado ao [clique aqui](#). Desses blocos de códigos explanados, a seguir constam os três principais, de acordo com as Figuras 3, Figura 4 e Figura 5.

Prontamente, em referência à Figura 3, na próxima página, é possível dizer que o bloco de código nomeado em *callback* destina-se propriamente à apresentação na tela da chegada de mensagem do tópico, conforme desejado, escrita e que, de acordo com a detecção de presença, ou não, a partir do valor recebido do tópico, liga ou desliga o LED.

Em seguida, também na próxima página, na Figura 4, a função de nome *reconnect* tem como objetivo: enquanto não estiver conectado, haverá tentativas de conexão ao broker; caso não seja possível conectar-se ao broker, será apresentado na tela, e se for conectado, também será apresentado na tela.

Dessa forma, é válido apontar que a cada tentativa de conexão, é adicionado valor em *string* na variável que se destina ao *id* do cliente. Consoante, e é possível a partir do valor do *id* do cliente que se conecta, ou não, ao broker.

```
54 void callback(String topic, byte* payload, unsigned int length) {
55
56     Serial.print("Message arrived [");
57     Serial.print(topic);
58     Serial.print("] ");
59     String strPayload="";
60     for (int i=0;i<length;i++) {
61         strPayload = strPayload + (char)payload[i];
62     }
63
64     Serial.println(strPayload);
65     StaticJsonBuffer<200> jsonBuffer;
66     JsonObject& data = jsonBuffer.parseObject(strPayload);
67
68     if(topic == "IPB/IoT/Lab/Presence" ){
69         if(data["Detection"] == "1"){
70             digitalWrite(LED_BUILTIN, LOW); //Turn off the LED
71         }else{
72             digitalWrite(LED_BUILTIN, HIGH); //Turn on the LED
73         }
74     }
75     Serial.println();
76 }
```

Figura 3: Bloco de código Callback

Não distante, é importante apontar algumas das funcionalidades, de funções, nativas da própria linguagem de programação C, que atua nas Figuras 3, Figura 4 e Figura 5.

Entre as funções que serão comentadas, a primeira será a ***Serial***, logo após a ***print*** e por fim o ***digitalWrite***.

```
78 void reconnect() {
79     // Loop until we're reconnected
80     while (!client.connected()) {
81         Serial.print("Attempting MQTT connection...");
82         // Create a random client ID
83         String clientId = "AulaIoT-";
84         clientId += String(random(0xffff), HEX);
85         // Attempt to connect
86         if(client.connect(clientId.c_str())) {
87             Serial.println("connected to the Broker");
88             client.subscribe("IPB/IoT/Lab/Presence"); //Tópico que o ESP subscrive.
89         } else {
90             Serial.print("failed, rc=");
91             Serial.print(client.state());
92             Serial.println("try again in 5 seconds");
93             delay(5000);
94         }
95     }
96 }
```

Figura 4: Bloco de código Reconnect

Sendo assim, com referência ao comando ***serial***, Fábio Souza afirma que a comunicação se restringa à comunicação serial, recurso poderoso que possibilita a comunicação entre a placa e um computador (SOUZA, F., 2023).

Em seguida, o comando ***print***, destina-se ao objetivo de escrever na tela, console, o que está em seu argumento, isto é, o que está dentro dele. Dado isso, o comando ***digitalWrite*** é o responsável por interligar o mundo virtual ao real, o que implica que envia um comando digital para a placa que está a ser utilizada.

Por conseguinte, com foco à interpretação do último bloco de código, em estrutura condicional, consta-se a Figura 5, abaixo expressa.

```
118 if (now - lastTime > 2000) {
119     lastTime = now;
120     ++value;
121     snprintf (msg, Msg_Size, "hello world %ld", value);
122     Serial.print("Publish message: ");
123     Serial.println(msg);
124     client.publish("IPB/IoT/Aula02/AlunoX", msg);
125 }
```

Figura 5: Estrutura condicional do código

Sem demora, ao atentar para a Figura 5, acima apresentada, é possível ver que seu

bloco de código se destina em uma condição, estrutura condicional. Isto posto, nela é validado se o tempo de agora subtraído o último tempo coletado, anteriormente no código fonte, que pode ser acessado [aqui](#), se é maior que 2 segundos, isto é, 2000 milissegundos.

Desse modo, se a verificação anterior for falsa, o bloco de código dentro da estrutura condicional é ignorada, agora, se for verdadeira, seu conteúdo é executado. Ao ser verdade, a validação, a variável *lasTime* recebe o tempo de agora, que neste caso, refere-se ao momento em que o código foi executado.

Então, a variável *value* recebe, incrementalmente, o valor 1. Em seguida, há a chamada de função *snprintf*, a qual imprime em uma única linha saídas formatadas, interpolando variáveis a uma string de formatação, e tendo confortos integrados que incluem a definição de quantidade de casas decimais, e outras formatações que podem ser conhecidas no site do [arduino](#).

Ademais, vê-se na Figura 5, que mensagens são impressas no console, e por fim, o cliente publica informações no tópico, na imagem especificado, no broker, já conectado.

4 Questão 08: Comunicação ESP com Node-RED

Em continuidade, por conseguinte, a seguir será construído um sistema de controle remoto para um LED usando um microcontrolador ESP e o Node-RED. Diante disso, o Node-RED será usado para criar um dashboard com um botão do tipo "switch" que permitirá aos utilizadores ligar ou desligar o LED remotamente.

Para isso, será utilizado o protocolo MQTT para estabelecer a comunicação entre o Node-RED e o ESP. Dessa forma, o Node-RED irá publicar informações sobre o estado do switch em um tópico MQTT específico (por exemplo, "IPB/IoT/Aula02/a39041"). À vista disso, o ESP irá subscrever-se a esse tópico e receber as informações do switch para ligar ou desligar o LED integrado.

Posto isso, a demonstração deste exercício foi feita com recurso ao simulador [Wokwii](#) (interface gráfica para criar, simular e testar circuitos eletrônicos, como o ESP, além de permitir a integração com outras plataformas, como o Node-RED). Logo, isso permitiu simular a comunicação entre o microcontrolador ESP e o Node-RED com êxito.

Com isso em mente, no Node-RED definimos os conjuntos de nós, dos quais, no switch do dashboard, há um componente gráfico que permite ao usuário interagir com o dashboard, o que fortalece na forma de entrada para controlar o fluxo de dados.

Então, este nó cria um botão de "switch" na interface do usuário, que pode ser ligado ou desligado clicando nele. Feito isso, quando o botão é pressionado, ele envia um sinal de saída para o fluxo do Node-RED, indicando se está ligado ou desligado.

Dessa forma, para melhor compreender esse cenário, nas próximas páginas constam as figuras: Figura 6, a Figura 7 e Figura 8 as quais detalham este caso.

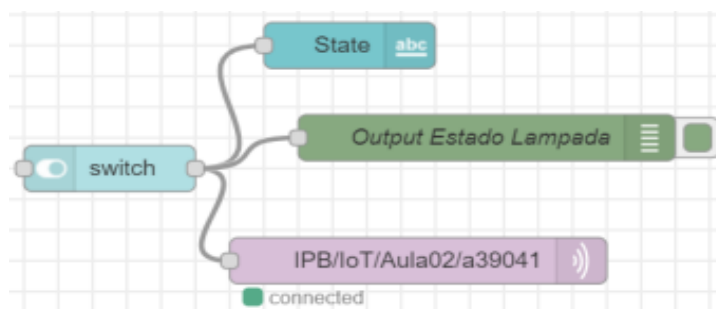


Figura 6: Código no Node-RED com Switch

Sendo assim, com base no resultado da ação realizada na Figura 8, configurada conforme vê-se na Figura 6, um vez que o utilizador interage com o switch-dashboard, a ação de ligar ou desligar o LED é atuada.

Por conseguinte, para melhor visualização da configuração do Switch, a Figura 7, elucida os atributos da mensagem a ser enviada os quais são definidos no “On Payload”.

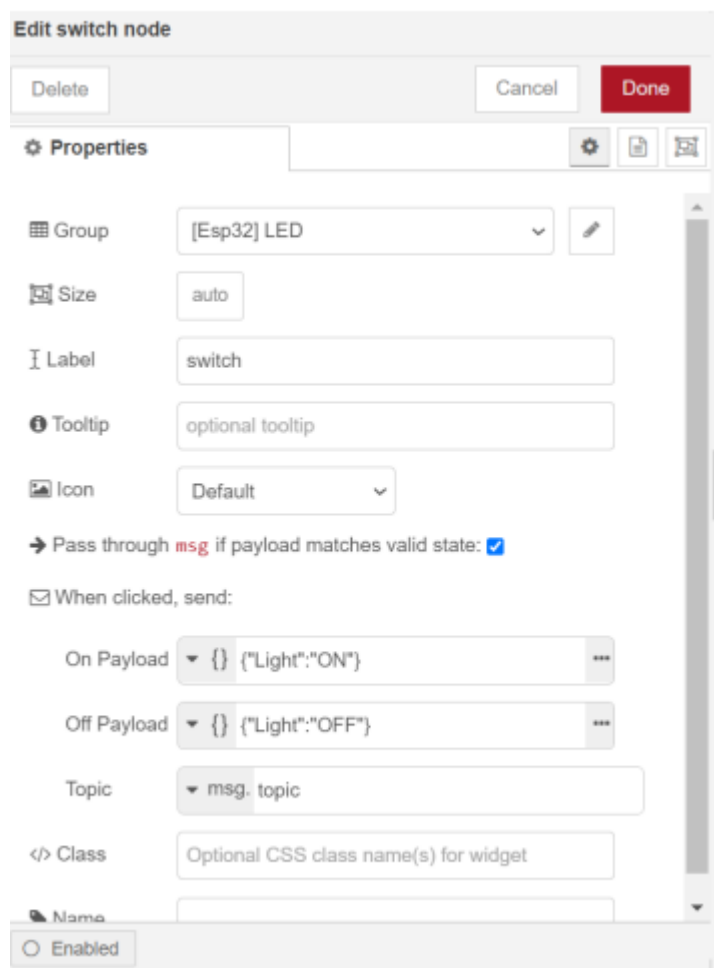


Figura 7: Configuração no Nó do Switch no Node-RED

Posto isso, na Figura 7, é possível ver o campo de “On Payload”, é neste campo em que os valores em objeto JSON, como “*LIGHT*:”*ON*” e o “Off Payload” igual a “*LIGHT*:”*OFF*” são definidos.

E isto significa que, quando o usuário aciona o botão do tipo “switch” no dashboard para “ligar” o dispositivo, o nó enviará uma mensagem contendo o payload “*LIGHT*:”*ON*” para o restante do fluxo do Node-RED.

Da mesma forma, quando o usuário acionar o botão do tipo “switch” para “desligar” o

dispositivo, o nó enviará uma mensagem contendo o payload *"LIGHT": "OFF"*.

Veja que, no nó state do dashboard, Node-RED, pode ser usado em conjunto com o nó switch-dashboard para exibir o status atual do dispositivo IoT controlado pelo switch-dashboard, o qual fornece um feedback visual para o utilizador.

Tendo isso em vista, prepara que o valor colocado pode variar entre ON e OFF, definido como STATE ON/OFF. E para melhor vislumbre desse contexto, a Figura 8, estabelece contato visual direto, em como é possível ativar ou desativar o LED de maneira prática e inteligente.

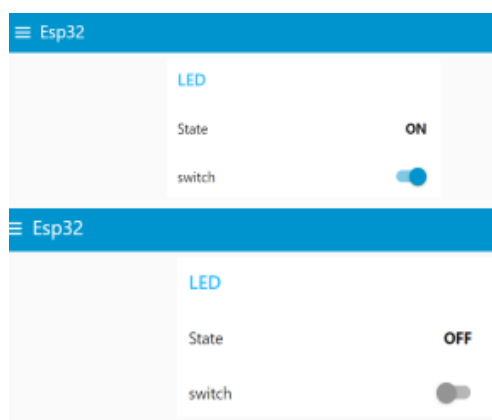


Figura 8: Respaldo do Switch no Dashboard do Node-RED

E é por essa razão que, quando o usuário aciona o botão do tipo "switch" no dashboard, conforme é possível ver na Figura 8, acima apresentada, o nó switch-dashboard envia uma mensagem contendo um objeto JSON para o restante do fluxo do Node-RED, segundo expressa a Figura 9, a seguir elucidada.

Em função disso, o nó de output pode ser usado para exibir esse objeto JSON em um formato legível para o usuário. E a partir disso, fica a cargo do desejo do usuário a maneira de abstrair e utilizar essas informações a critério próprio.

```
24/03/2023, 00:01:50 node: Output Estado Lampada
msg.payload: Object
  ▶ { Light: "ON" }

24/03/2023, 00:01:56 node: Output Estado Lampada
msg.payload: Object
  ▶ { Light: "OFF" }
```

Figura 9: Saída do Payload em JSON no Node-RED do Switch

Consoante, é bom apontar que o nó mqtt-out é um nó de saída do Node-RED que permite enviar mensagens para um broker MQTT.

Dito isso, no contexto deste fluxo, no Node-RED, o nó mqtt-out é usado para enviar um objeto JSON contendo o valor "ON" ou "OFF" para um tópico MQTT, que será posteriormente subscrito pelo microcontrolador ESP no wokwi.

Assim, ao depender do valor no "switch" no dashboard, é enviada uma mensagem para o restante do fluxo do Node-RED com um payload contendo uma mensagem específica, como *"LIGHT": "ON"*.

E para expressar essa ideia, via simulação da realidade, a Figura 10 e Figura 11, abaixo expressam a maneira em que seria feito, na realidade, a lógica apresentada, e a seguir detalhada; sendo que na Figura 10 o LED azul está ligado, em detrimento do "ON" no Dashboard no Node-RED e na Figura 11 o LED azul está apagado por estar "OFF" no Dashboard do Node-RED.



Figura 10: LED azul ligado na ESP por ação no Node-RED

Em seguida, conforme é visível na Figura 10, acima apresentada, o ESP recebe o valor

ON do tópico [IPB/IoT/Aula02/a39041] e acende o LED interno, com a luz azul, em acordo com as ações/informações que serão expressas no Dashboard do Node-RED.

Dessa forma, à medida em que o nó mqtt-out é usado para enviar este objeto JSON para um tópico MQTT específico, o microcontrolador ESP no wokwi foi configurado para subscrever a este tópico MQTT e receber mensagens sempre que o valor deste tópico for atualizado.

Quando o ESP recebe uma mensagem deste tópico contendo o valor "ON", conforme expressa-se na Figura 10, da página anterior, o LED azul está ligado, e quando recebe uma mensagem com o valor "OFF", como expressa-se na Figura 11, abaixo apresentada, em que o LED azul está desligado.



Figura 11: LED azul desligado na ESP por ação no Node-RED

Em seguida, conforme é visível na Figura 11, acima apresentada, o ESP subscreve o tópico [IPB/IoT/Aula02/a39041], para receber as ações/informações que serão expressas no Dashboard do Node-RED.

Posto isto, na próxima página conta a Figura 12, em que nela há um trecho de código, com uma função de nome *callback*, a qual é chamada sempre que uma nova mensagem

MQTT é recebida pelo microcontrolador ESP.

Sendo assim, esta função de *callback* é responsável por interpretar as mensagens MQTT recebidas pelo microcontrolador ESP e controlar o LED com base no conteúdo dessas mensagens.

Consoante, esta função usa a biblioteca ArduinoJson para fazer o parsing do objeto JSON contido no payload da mensagem e usa a lógica condicional para decidir se o LED deve ser ligado ou desligado com base no valor da chave *"Light"*; veja este cenário, abaixo, na Figura 12.

```
void callback(String topic, byte* payload, unsigned int length) {  
  
    Serial.print("Message arrived [");  
    Serial.print(topic);  
    Serial.print("] ");  
    String strPayload="";  
    for (int i=0;i<length;i++) {  
        //byte to char  
        strPayload = strPayload + (char)payload[i];  
    }  
  
    StaticJsonBuffer<200> jsonBuffer;  
    JsonObject& data = jsonBuffer.parseObject(strPayload);  
  
    if(topic == "IPB/IoT/Aula02/a39041" ){  
        if(data["Light"] == "OFF"){  
            digitalWrite(LED_BUILTIN, LOW); //Turn off the LED  
            Serial.println("OFF");  
        }else{  
            digitalWrite(LED_BUILTIN, HIGH); //Turn on the LED  
            Serial.println("ON");  
        }  
    }  
    Serial.println();  
}
```

Figura 12: Atuação da função callback na ESP

De acordo com que contempla-se na Figura 12, acima, a primeira parte da função, simplesmente imprime o tópico e o payload recebidos no monitor serial, para fins de depuração.

Em seguida, o payload é convertido de um array de bytes para uma string usando um loop for, e a biblioteca ArduinoJson é usada para fazer o parsing do payload em um objeto JSON.

A seguir, a função verifica se o tópico da mensagem recebida corresponde a um tópico específico, "IPB/IoT/Aula02/a39041". Se sim, ela verifica o valor da chave "Light" no

objeto JSON. Se o valor for "OFF", a função desliga o LED conectado ao pino digital "LED_BUILTIN" do ESP, e se o valor for "ON", a função liga o LED.

À vista disso, em próximo passo, na Figura 13, abaixo, está a configuração de um cliente MQTT que se conecta a uma rede WiFi e se comunica com um servidor MQTT específico.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

//wifi credentials
const char* ssid = "Wokwi-GUEST";
const char* password = "";

// MQTT client
WiFiClient wifiClient;
PubSubClient client(wifiClient);

//broker variables
const char* mqtt_server = "broker.mqtt-dashboard.com";
int mqttPort = 1883;

void connectWiFi(){
  Serial.print("Connecting to ");

  WiFi.begin(ssid, password,6);
  Serial.println(ssid);

  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.println("Wifi connecting...");
  }
  Serial.print("Wifi connected, IP address: ");
  Serial.println(WiFi.localIP());
}

void brokerReconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("a39041")) {
      Serial.println("connected");
      client.subscribe("IPB/IoT/Aula02/a39041");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Figura 13: Código *publish-subscribe* no Arduino ESP

Em consonância, ele inclui três bibliotecas: a biblioteca WiFi, a biblioteca PubSubClient (implementação do protocolo MQTT em C++) e a biblioteca ArduinoJson (para manipular dados em formato JSON).

Deste modo, o código apresentado implementa a conexão com a rede Wi-Fi e com o

servidor MQTT para permitir que um dispositivo ESP possa publicar e/ou subscrever tópicos MQTT.

Destarte, a função `connectWiFi()` é responsável por conectar o ESP à rede Wi-Fi, utilizando as credenciais definidas, e exibe o endereço IP atribuído ao dispositivo. Por conseguinte, a função `brokerReconnect()` é responsável por tentar conectar o ESP ao servidor MQTT especificado e, em caso de sucesso, subscreve o tópico indicado.

Consequentemente, caso a conexão não seja bem sucedida, a função exibe o código de erro retornado pelo cliente MQTT e espera 5 segundos antes de tentar novamente. E essas funções são importantes para garantir que o ESP esteja conectado à rede Wi-Fi e ao servidor MQTT antes de iniciar a troca de mensagens, e para lidar com desconexões ou problemas de conexão durante a execução do código.

Deste jeito, a função `setup()`, abaixo apresentada na Figura 14, expressa a realização de a inicialização do programa e configura as variáveis e os dispositivos a serem utilizados. Consoante, com a função `loop()` é possível executar a continuamente e também as ações necessárias para o programa funcionar corretamente.

```
void setup() {  
  Serial.begin(9600);  
  pinMode(LED_BUILTIN, OUTPUT);  
  connectWiFi();  
  client.setServer(mqtt_server, mqttPort);  
  // set the callback function  
  client.setCallback(callback);  
  
}  
  
void loop() {  
  if (!client.connected()) {  
    brokerReconnect();  
  }  
  client.loop();  
}
```

Figura 14: Função *setup()* do código ESP

5 Questão 09: Comunicação ESP com Potenciômetro

Isto posto, atual atividade acima expressa, é uma extensão do exercício anterior, em que agora um potenciômetro é conectado ao ESP. E isso, aponta que o valor do potenciômetro é lido ao usufruir da função *analogRead(A0)* e publicado no tópico MQTT (*IPB/IoT/Aula02/a39041/Potenciometro*).

Diante disso, do exercício anterior, o ESP8266 foi utilizado apenas como um subscritor MQTT, que recebia um valor do Node-RED e controlava o LED com base nesse valor. Agora, no atual exercício, a comunicação torna-se bidirecional, ou seja, tanto o ESP quanto o Node-RED atuam como publicadores e subscritores MQTT. Sendo assim, para melhor compreensão do fluxo do Node-RED com o ESP, abaixo está a Figura 15.

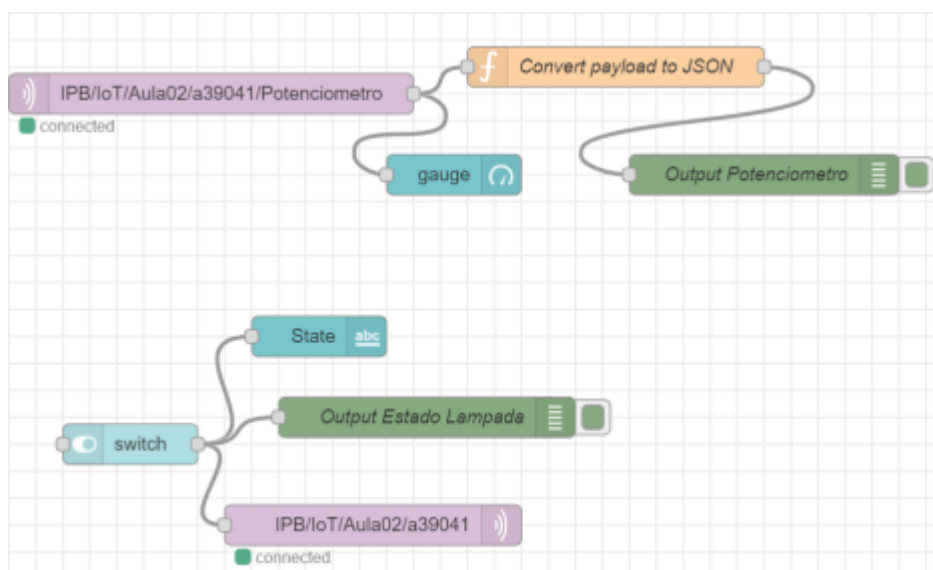


Figura 15: Fluxo do Node-RED em conexão ao ESP

Prontamente, o ESP publica o valor coletado pelo potenciômetro em um determinado tópico MQTT (*IoT/Ficha2/a39041/Potenciometro*) e o Node-RED subscrive esse tópico por meio de um nó MQTT-IN.

Por esse motivo, o valor numérico recebido é então passado por um nó de função para formatá-lo em um objeto JSON, o que torna a leitura mais compreensível para o utilizador, como por exemplo: *"Potenciometro":123*.

Em continuidade, é válido dizer que a ordenação nos módulos, da Figura 16, cada mini-

bloco de cor variada, é dito pelos comandos internos os que pre-dizem suas funcionalidades e submissões. Também, no que se refere às conexões entre estes blocos, são elas outro fator que estabelece conexão de ordenação lógica do passo a passo do que se deve ser feito.

Por conseguinte, veja que, se dois blocos, ou mais unidades de blocos, não estão conectados, isto não implica que não estão em conexão, isso é devido o broker fazer ponto de liga entre publicador e subscritor.

Por fim, esta configuração apresentada, é possível vê-la na Figura 16, na próxima página, em que no "On Message" é inserido comando em, JavaScript, JSON para submissão nos trâmites posteriores.

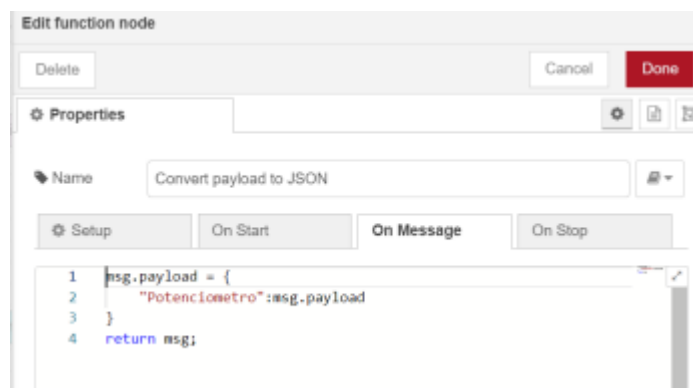


Figura 16: Configuração de função no Node-RED

Diante do exposto, anteriormente, ao retomar o foco ao sudo direto no ESP, na simulação, o microcontrolador está conectado ao Wokwi, que fornece uma interface virtual para um potenciômetro, conforme abaixo está exposto na Figura 17.

Em consideração a isso, o ESP continua a subscrever o estado do LED, permitindo que o usuário possa ligar e desligar o LED através do dashboard do Node-RED. Além disso, foi adicionado um potenciômetro virtual no Wokwi, que é lido pelo ESP através da função *analogRead(A0)*.

Sendo assim, em cada 2.5 segundos, o valor do potenciômetro é publicado no tópico *[IoT/Ficha2/a39041/Potenciometro]* utilizando o protocolo MQTT. Agora, em referência ao Node-RED, ele é utilizado um nó MQTT-In para subscrever a esse tópico e receber os valores publicados pelo ESP.

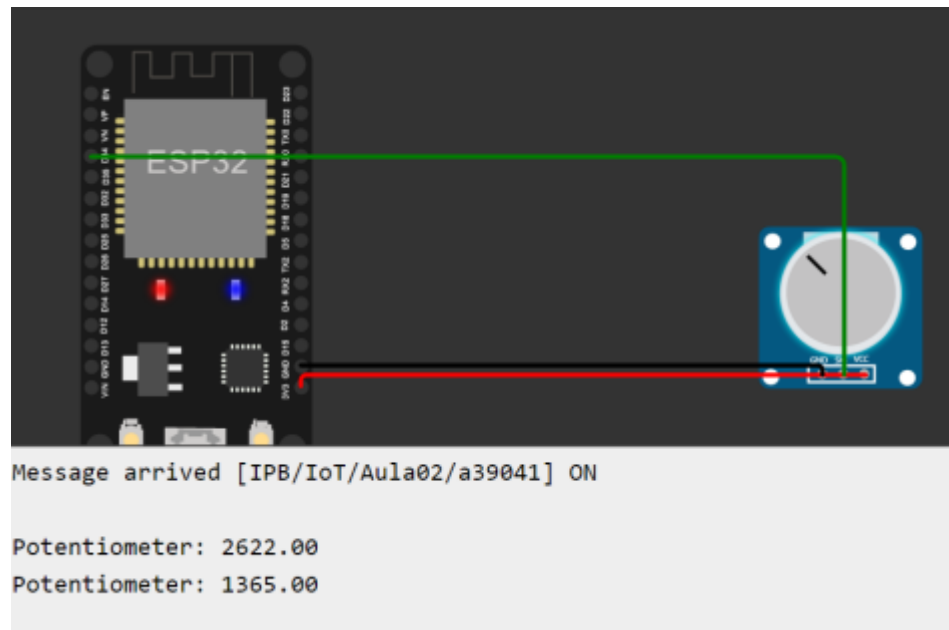


Figura 17: Simulação do ESP em conexão à um Potenciômetro

Em seguida, é usado um nó de função para transformar o valor recebido em um objeto JSON com a chave "Potenciometro" e seu valor correspondente, o qual pode ser visualizado abaixo na Figura 18.

```
23/03/2023, 23:26:28 node: Output Estado Lampada
msg.payload : Object
  { Light: "ON" }

23/03/2023, 23:26:31 node: Output Potenciometro
IPB/IoT/Aula02/a39041/Potenciometro : msg.payload :
  { Potenciometro: 2622 }

23/03/2023, 23:26:37 node: Output Potenciometro
IPB/IoT/Aula02/a39041/Potenciometro : msg.payload :
  { Potenciometro: 1365 }
```

Figura 18: Saída no Node-RED dos dados publicado pelo ESP

Consequentemente, a partir das configurações no Node, conforme a Figura 15, com exemplo de uma dessas configurações, na Figura 16 e com saída possível com relação a Figura 18, um gráfico do tipo gauge é adicionado ao dashboard do Node-RED para exibir o valor do potenciômetro em tempo real, expresso abaixo na Figura 19.

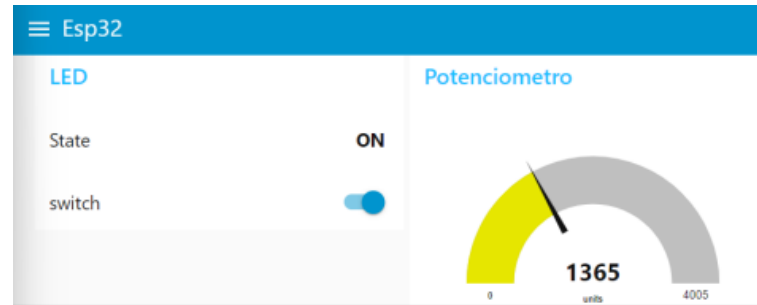


Figura 19: Gauge no Dashboard do Node-RED em conexão ao ESP

Diante disso, para entender melhor a Figura 19, a Figura 20, expressa o código no ESP para que os valores resultantes do potenciômetro saia no Dashboard do Node-RED.

```
//messages definitions
long lastMsg = 0;
char msg[50];
int value = 0;

const int potPin = 34;
float potValue = 0;

void loop() {
  if (!client.connected()) {
    brokerReconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2500) {
    lastMsg = now;

    potValue = analogRead(potPin);

    char potString[8];
    dtostrf(potValue, 1, 2, potString);
    Serial.print("Potentiometer: ");
    Serial.println(potString);
    client.publish("IPB/IoT/Aula02/a39041/Potenciometro", potString);
    delay(500);
  }
}
```

Figura 20: Código de definição de mensagens no ESP

Dessa forma, este trecho de código, acima definido, na Figura 20, é apenas a parte diferente do exercício anterior, utiliza a função *millis()* para medir o tempo desde a última mensagem publicada. Consoante, se o tempo decorrido for superior a 2,5 segundos, é recolhido o valor do potenciômetro através da função *analogRead(potPin)*.

Em seguida, é convertido o valor recolhido para uma string utilizando a função *dtostrf()*, e a mensagem é publicada no tópico *"IPB/IoT/Aula02/a39041/Potenciometro"*

através da função `client.publish()`.

Por fim, ademais, o valor do potenciômetro é também impresso no monitor serial para fins de depuração; o `delay()` é usada para aguardar 500 milissegundos antes da próxima iteração do `loop()`.

Referências

Arduino.cc. Função loop. Disponível em: <https://cdn.arduino.cc/reference/pt/language/functions/main/loop/>. Acesso em: 25 Mar. 2023, a.

Arduino.cc. Função setup. Disponível em: <https://cdn.arduino.cc/reference/pt/language/functions/main/setup>. Acesso em: 25 Mar. 2023, b.

SOUZA, F. Arduino - comunicação serial. Disponível em: <https://embarcados.com.br/arduino-comunicacao-serial/>. Acesso em: 25 Mar. 2023.