INSTITUTO FEDERAL
São Paulo

RELATÓRIO PARCIAL DO PROJETO DA DISCIPLINA DE MICROCONTROLADORES

Allan B. da Silva; Edson R. da C. Filho; Mateus C. Lucas; Wilson B. R. Luo

1. INTRODUÇÃO

A Internet das Coisas (IoT) é uma rede que conecta diversos dispositivos através da Internet, permitindo a comunicação entre eles e também com a nuvem. Isso é possível devido à evolução de chips de computador mais acessíveis e à alta largura de banda das telecomunicações. Essa conexão possibilita que objetos comuns do dia a dia, como escovas de dentes, aspiradores, carros e máquinas, utilizem sensores para coletar dados e responder de forma inteligente aos usuários. Essa tendência de integrar objetos cotidianos à Internet começou nos anos 1990, mas avançou lentamente devido ao tamanho dos chips de computador. Com o tempo, esses chips se tornaram menores, mais rápidos e mais inteligentes, reduzindo o custo de adicionar capacidade computacional a objetos pequenos.

Atualmente, é possível equipar objetos como interruptores de luz com conectividade, incluindo recursos de serviços de voz como assistentes virtuais. Isso deu origem a um setor dedicado a equipar residências, empresas e escritórios com dispositivos IoT. Esses objetos inteligentes podem transmitir dados automaticamente para a Internet, e todo esse conjunto de "dispositivos de computação invisíveis" e sua tecnologia são conhecidos como Internet das Coisas. Um sistema IoT opera através da coleta e troca de informações em tempo real. Este sistema consiste em três elementos fundamentais:

Dispositivos Inteligentes, como televisões, câmeras de segurança e equipamentos de exercício, são equipados com recursos de processamento. Eles coletam dados do ambiente, interações dos usuários e padrões de uso, e compartilham esses dados através da Internet, estabelecendo comunicação bidirecional com a aplicação de IoT.

Aplicação de IoT, sendo esta um conjunto de serviços e software que integra dados provenientes de diversos dispositivos IoT. Ela utiliza tecnologia de aprendizado de máquina ou inteligência artificial (IA) para analisar esses dados e tomar decisões baseadas em informações. As decisões são então comunicadas de volta aos dispositivos IoT, que respondem de maneira inteligente às entradas recebidas.

A Comunicação pela Internet desempenha um papel crucial na conexão entre dispositivos inteligentes e a aplicação da Internet das Coisas (IoT), facilitando a transmissão contínua de dados em tempo real e permitindo respostas rápidas às informações processadas. A IoT impacta significativamente a vida cotidiana e o ambiente de trabalho, capacitando máquinas a executar tarefas pesadas e eliminando atividades monótonas. Além disso, proporciona uma compreensão mais profunda e controle de objetos e ambientes, resultando em maior eficiência e tomada de decisões em tempo real com base em dados. A implementação da IoT promete reduzir desperdícios, diminuir custos e evitar inconvenientes, enquanto o aumento de dispositivos conectados abre novas oportunidades para startups e empresas. A principal vantagem é a criação de um ambiente mais sustentável e produtivo, contribuindo para uma vida mais saudável e confortável para as pessoas.

2. ESTUDO SOBRE AS CARACTERÍSTICAS DOS SENSORES

2.1. Sensor de umidade relativa do ar (DHT11)

É um sensor de saída digital, composto por um sensor resistivo de umidade e um sensor NTC medidor de temperatura. Esse sensor possui 4 pinos, sendo o pino 1 destinado ao VCC, pino 2 ao canal de dados, pino 3 é nulo e o quarto pino correspondente ao GND.

A tensão de alimentação desse sensor é 3-5.5V DC. Segundo o Datasheet do sensor, no pino de transferência de dados é recomendado o uso de um resistor de pull-up de 5k entre o pino e o microcontrolador a ser usado.

A interface de comunicação para conectar o sensor ao microcontrolador é a Interface Serial (Single-Wire Two-way). Basicamente o processo ocorre da seguinte forma: O microcontrolador envia um sinal de início, fazendo com que o sensor DTH11 saia do modo de baixo consumo e passe para o modo de transferência de dados. Assim que o sensor setar o modo de transferência de dados, é enviado um sinal de resposta para o microcontrolador, que contém 40 bits de dados, sendo esses 40 bits estruturados conforme o seguinte modelo:

8 bits parte inteira da UR + 8 bits da parte decimal da UR + 8 bits da parte inteira da T + 8 bits da parte decimal da T + 8 bits de checksum

Onde UR = Umidade Relativa e T = Temperatura.

O sinal de cada bit de dados começa com um sinal de 50 µs de baixa tensão, e o comprimento do próximo sinal de alto nível determinará se o bit corresponde a 0 ou 1. Sinais de alto nível de 26-28µs correspondem ao bit 0, e 70 µs correspondem ao bit 1.

Quando o último bit é transmitido, o sensor DHT envia um sinal de baixo nível de 50µs para indicar o fim da transmissão, e em seguida a tensão volta para alto nível para indicar que o canal de transmissão está livre para uma nova troca de dados.

O diagrama abaixo exemplifica a linha do tempo de uma transmissão entre o microcontrolador e o DHT11.

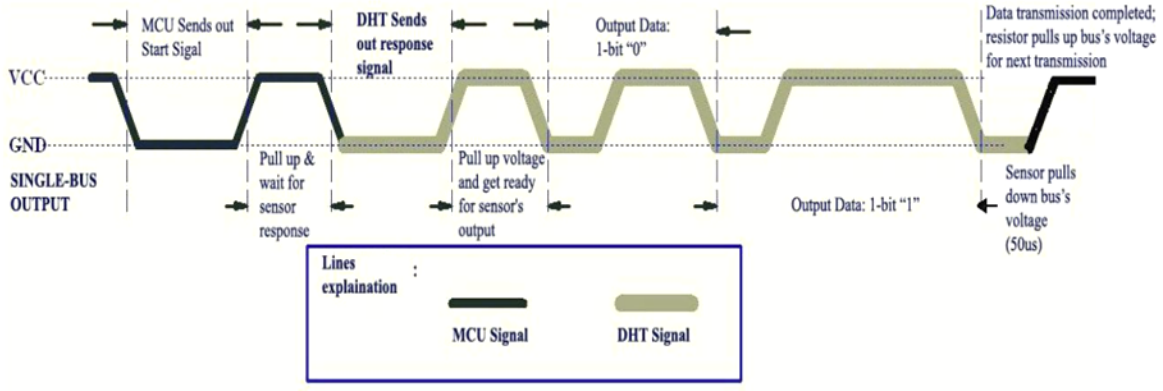

Figura 1 - Diagrama de transmissão do DHT11 ao longo do tempo

2.2. Sensor de pressão (BMP280)

O sensor BMP 280 é um sensor digital de pressão atmosférica absoluta, desenvolvido especialmente para aplicações de pequeno porte. Consiste basicamente em

um sensor de pressão piezoresistivo e um sinal misto ASIC, no qual são realizadas as conversões A/D e em seguida, os dados transmitidos através de uma interface digital.

Esse sensor piezoresistivo se deforma ligeiramente conforme a variação de pressão, fazendo com que um sinal elétrico proporcional a pressão seja gerado, para que possa ser convertido em um sinal digital e enviado ao microcontrolador interpretar tal valor. Seu range de pressão varia de 300 hPa até 1100 hPa, o que é equivalente a 9000 metros acima do nível do mar até -500 metros abaixo do nível do mar.

Esse sensor opera em tensões que podem variar de no mínimo 1.71 V até no máximo 3.6V. Como interface de comunicação digital, pode atuar tanto usando a I²C como a SPI, operando como escravo em ambos os protocolos.

Na interface I²C são utilizados os seguintes pinos:

- SCK : Serial Clock(SCL)
- SDI: Data(SDA)
- VCC (3.3V)
- GND

Para realizar uma leitura do sensor através dessa interface, é preciso primeiro enviar uma condição de start, e o endereço do registrador escravo como modo de escrita(RW=0). Em seguida, uma condição de stop ou start deve ser gerada, permitindo que agora o escravo seja endereçado como modo de leitura. (RW = 1).

Após isso, o escravo envia para o mestre os registros de dados de 8 bits, existentes em registradores auto incrementais até que uma condição de NOACK(Not acknowledge by master) e de stop ocorra. A figura abaixo exemplifica essa operação:

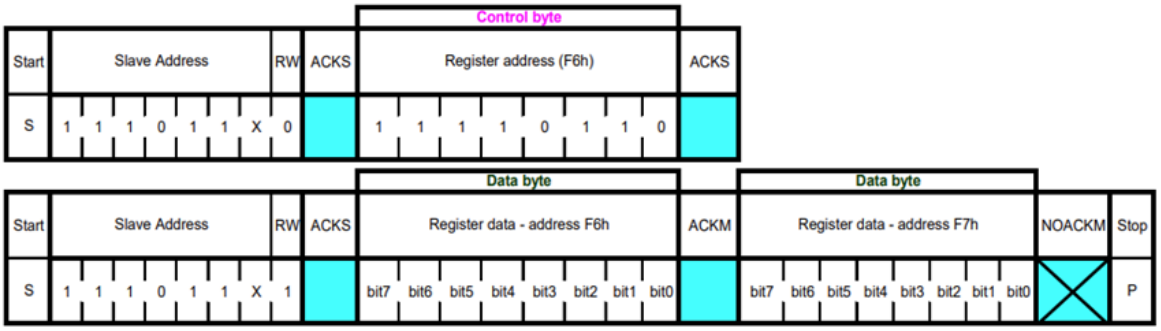

Figura 2 - Leitura de múltiplos bytes na interface I²C

Já na interface SPI, além do VCC e GND, são utilizados os pinos:

- CSB: chip select, baixo ativo
- SCK: serial clock
- SDI: serial data input
- SDO: serial data output

Para que a leitura seja feita nessa interface, é preciso enviar um sinal de nível lógico baixo no CSB, e em seguida enviar um byte de controle, que contém o endereço do registrador a ser lido(7 bits) e o comando indicando que é uma leitura(1 bit, RW=1). Após isso, os dados começam a ser transferidos através do pino SDO. O esquema abaixo representa essa operação de leitura no protocolo SPI:

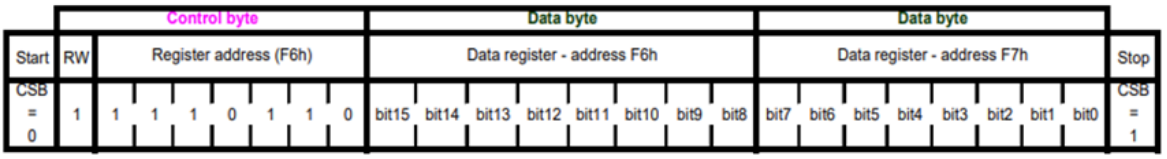

Figura 3 - Leitura de múltiplos bytes na interface SPI

2.3. Sensor de Temperatura (LM35)

O LM35 é um sensor de temperatura analógico amplamente utilizado para medições de temperatura em projetos eletrônicos. Este dispositivo fornece uma precisão exata e tem uma tensão de saída linearmente proporcional à temperatura em graus Celsius.

O sensor LM35 produz um sinal de tensão que varia 10mV para cada ºC, sendo que ele é capaz de operar em uma escala de temperatura que pode variar entre -55 ºC até 150 ºC. Mas vale ressaltar que o resultado medido e o valor do sinal exato são bem próximos, isso faz com que possa ocorrer uma variação de temperatura aproximada de 0,4 ºC até 1,5 ºC do valor real.

Outra característica muito interessante do sensor LM35 é que ele apenas consome corrente de uma fonte de 60 µA, resultando em um autoaquecimento muito baixo, inferior a 0,1 °C sob certas condições. Em aplicações que necessitam de uma precisão com mais exatidão, é essencial levar em consideração esta pequena variação da temperatura.

O sensor é composto por 3 pinos:

- Vcc: pino de alimentação positivo, capaz de usar de 4 a 20V.
- Vout: pino com tensão de saída, que é proporcional à temperatura de saída.
- GND: pino terra

Para medir a tensão de saída do sensor, empregamos a função “analogRead()” disponibilizada pelo microcontrolador. Essa função realiza a leitura da voltagem analógica de um pino específico e a converte em um valor digital, variando de 0 a 1023. Esse processo é facilitado pelo Conversor Analógico-Digital (ADC) integrado ao microcontrolador.

Posteriormente, é essencial estabelecer uma relação entre a medida de saída da tensão e a faixa de operação do sensor LM35, que abrange de -55 °C a 150 °C. Ao concluir esse processo, será possível obter o resultado final da medição de temperatura.

3. MATERIAL E MÉTODOS

Utilizou-se os seguintes materiais para montagem do hardware do projeto:

1. 1 ESP – WROOM – 32
2. 1 Display LCD com interface I²C
3. 1 DHT11
4. 1 BMP280
5. 1 LM35
6. 2x Resistores de 10k Ohms
7. 1 diodo
8. 1 botão
9. 3 Protoboards
10. Jumpers

Para desenvolver o firmware do projeto, houve inclusão das seguintes bibliotecas:

1. "LiquidCrystal_I2C.h" de Frank de Brabander
2. "DHT11.h" de Dhruba Saha
3. "Adafruit_BMP280.h" de Adafruit
4. "NTPClient.h" de Fabrice Weinberg
5. "PubSubClient.h" de Nick O'Leary

3.1. Processo de funcionamento geral do firmware

O código em C++ realiza leituras frequentes dos sensores de temperatura, umidade e pressão, bem como do servidor NTP. Os dados obtidos são armazenados em um conjunto de variáveis no escopo global do código. Posteriormente, funções para atualização do LCD, assim como funções MQTT, utilizam os dados dessas variáveis para seus próprios processos.

Desse modo, é possível manter os dados do LCD atualizados e publicar regularmente os dados da aplicação via MQTT. Isso proporciona um monitoramento contínuo das condições climáticas e a capacidade de compartilhar essas informações remotamente.

3.2. Display LCD

Para desenvolver a parte do projeto com LCD 16 x 2 com interface I2C, utilizou-se a seguinte montagem de hardware representada na Figura 5 e detalhada na Tabela 1.

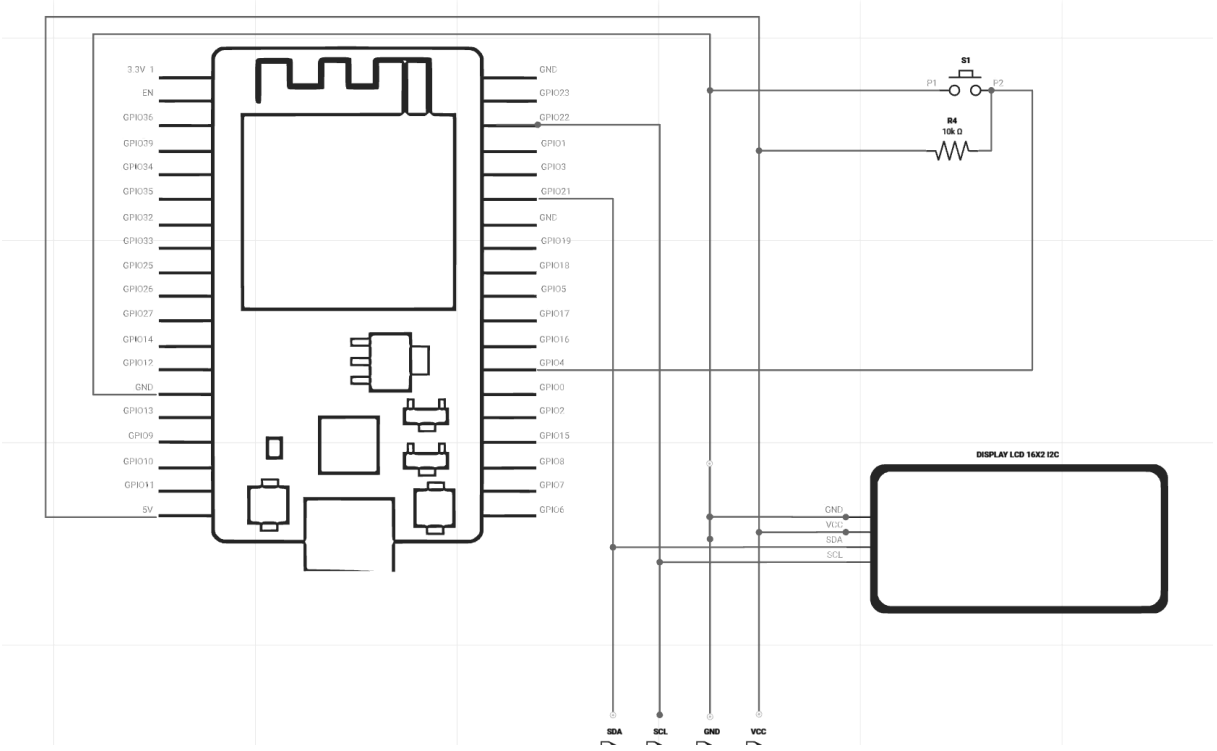

Figura 4 - Circuito para o Display LCD

Tabela 1 - Descrição dos pinos da placa ESP32-WROOM-DA utilizados no projeto.		
Número do Pino	Número do GPIO	Função
D4	4	Botão para troca de telas no display LCD

D21	21	SDA para comunicação I2C com o LCD
D22	22	SCL para comunicação I2C com o LCD

No início, definiu-se algumas constantes: o pino usado para o botão (pin 4) e três constantes para representar os sensores (TEMPERATURA, UMIDADE e PRESSAO). Após, definiu-se variáveis como "nSensores" para o número de sensores, "toggle" para determinar o sensor a ser exibido no LCD e outras variáveis necessárias para implementar a rotina do botão, o qual funciona através de interrupção. Na figura 6 é possível verificar a função utilizada para tratar o evento de pressionar o botão.

```
void IRAM_ATTR tratamentoBotao()
{
  if((millis() - lastDebounceTime) > debounceDelay)
  {
   lastDebounceTime = millis();
   botPress = true;
  }
}
```

Figura 5 - Rotina de interrupção do botão

Existem três funções, “setFormatLcdTemperatura”, “setFormatLcdUmidade” e “setFormatLcdPressao”, que configuram o LCD para exibir as leituras de temperatura, umidade e pressão respectivamente. Cada função limpa o LCD e define o texto apropriado e unidades. Abaixo, há um exemplo de uma delas.

```
void setFormatLcdTemperatura()
{
  lcd.clear();

  lcd.setCursor(1,0);
  lcd.printstr("Temperatura:");

  lcd.setCursor(1,1);
  lcd.printstr("T = ");

  lcd.setCursor(11,1);
  lcd.write((byte)0);
  lcd.printstr("C");
}
```

Figura 6 - Função para atribuir a formatação de temperatura no display LCD

3.3. Sensor de umidade relativa do ar (DHT11)

Em relação ao sensor DHT11, para utilizar o sensor fez-se a seguinte montagem de hardware representada na figura 8 e detalhada na tabela 2.

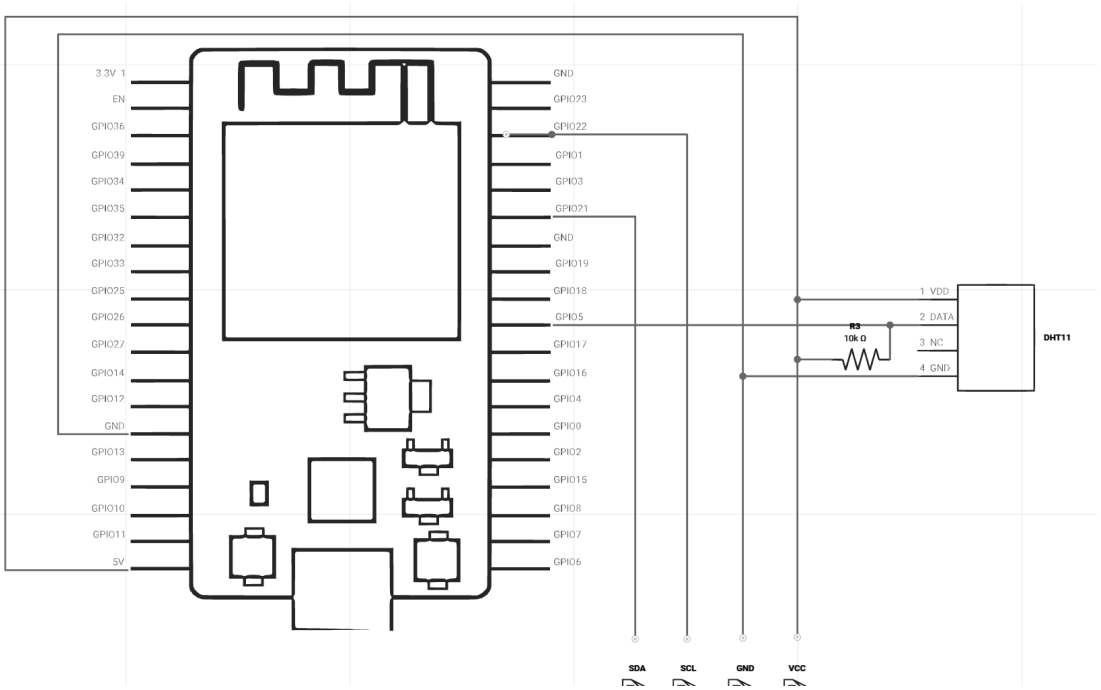

Figura 7 - Circuito para o DHT11

Tabela 2 - Descrição dos pinos da placa ESP32-WROOM-DA utilizados no projeto.		
Número do Pino	Número do GPIO	Função
D5	5	Leitura dos dados captados pelo sensor DHT11

Para medir a umidade ambiente e armazená-la, utilizou-se a função “updateUmidade”. Essa função acessa o DHT11, mede o valor de umidade e armazena a medição na variável global “umidade”. Na figura 9, é mostrada a função em questão.

```
void updateUmidade()
{
  umidade = dht11.readHumidity();
}
```

Figura 8 - Função para atualizar valor da variável umidade

Para atualizar o Display LCD com o valor da variável umidade, adicionou-se a função “updateValueLcdUmidade” representada na figura 10, que irá ser chamada em loop caso o toggle indique que a umidade deve ser exibida no display LCD.

```
void updateValueLcdUmidade()
{
  lcd.setCursor(7,1);
  if(umidade != DHT11::ERROR_CHECKSUM && umidade != DHT11::ERROR_TIMEOUT)
  {
    lcd.print(umidade);
  }
}
```

Figura 9 - Função para atualizar o valor de umidade no LCD

3.4. Sensor de pressão (BMP280)

Em relação ao sensor BMP280, para utilizar o sensor fez-se a seguinte montagem de hardware mostrada na figura 11 e especificada na tabela 3.

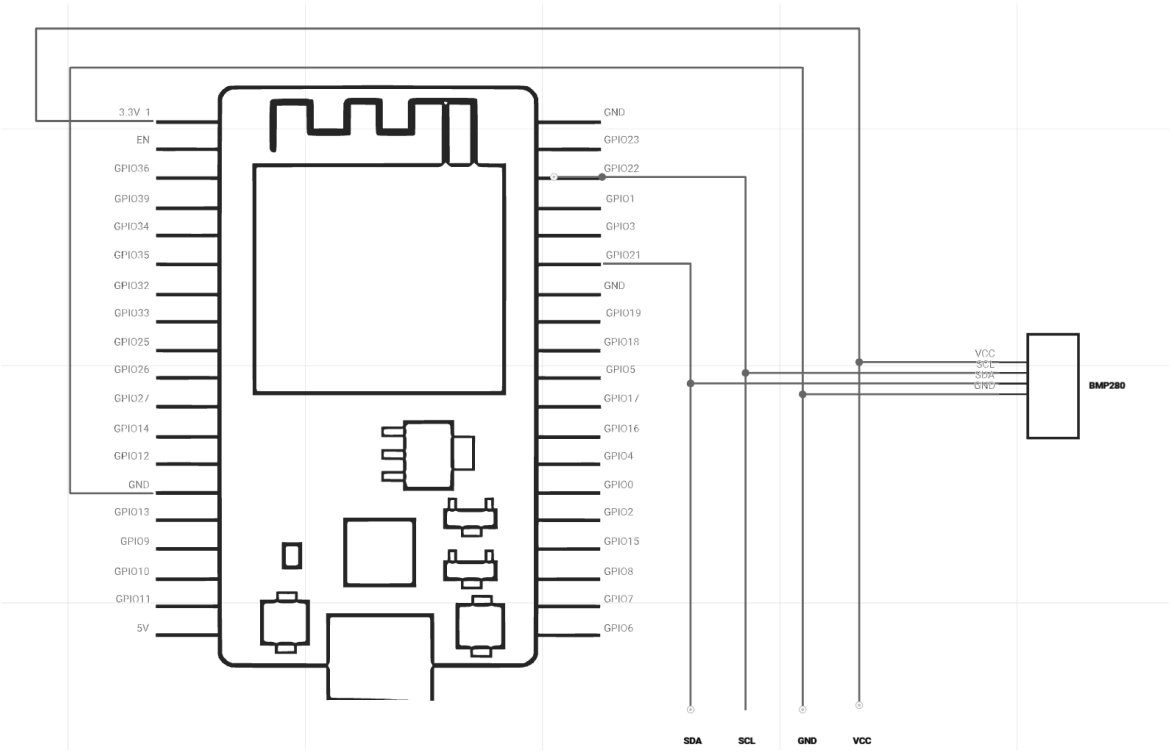

Figura 10 - Circuito para o BMP280

Tabela 3 - Descrição dos pinos da placa ESP32-WROOM-DA utilizados no projeto.		
Número do Pino	Número do GPIO	Função
D21	21	SDA para comunicação I2C com o BMP280
D22	22	SCL para comunicação I2C com o BMP280

Em relação ao software, adotou-se uma abordagem semelhante à utilizada com o sensor DHT11. Isto é, criou-se uma função “updatePressao” (figura 12) para atualizar a variável de pressão frequentemente e outra função “updateValueLcdPressao” (figura 13) para alterar o valor de pressão no display caso o toggle indique isso.

```
void updatePressao()
{
  bmp.takeForcedMeasurement();
  pressao = bmp.readPressure()/1000.0;
}
```

Figura 11 - Função para atualizar o valor da variável pressão

```
void updateValueLcdPressao()
{
  lcd.setCursor(5,1);
  lcd.print(pressao);
}
```

Figura 12 - Função para atualizar o valor de pressão no LCD

3.5. Sensor de temperatura (LM35)

Para utilizar o sensor LM35 utilizou-se montagem de hardware exibida na figura 14 e especificada na tabela 4.

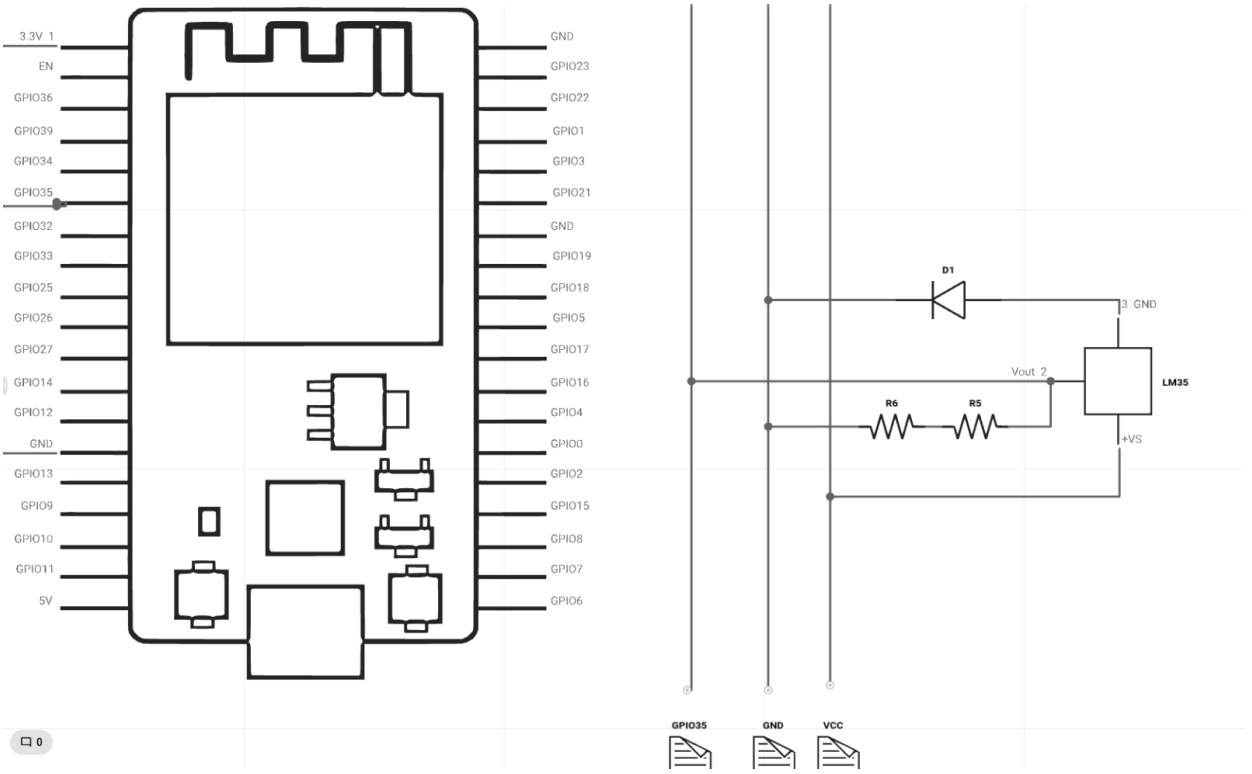

Figura 13 - Circuito para o LM35

Tabela 4 - Descrição dos pinos da placa ESP32-WROOM-DA utilizados no projeto.		
Número do Pino	Número do GPIO	Função
D35	35	Pino para leitura analógica do LM35

Assim como foi feito para o DHT11 e BMP280, criou-se as funções “updateTemperatura” (figura 15) e “updateValueLcdTemperatura” (figura 16).

```
void updateTemperatura()
{
  float n = 0;
  float nMedio = 0;
  float tensao = 0;
  float tempAux = 0;

  for(int i = 0; i < nMedidas; i++)
  {
    n += analogRead(tempPin);
    delay(20);
  }

  nMedio = n / nMedidas;
  tensao = 0.00079267912 * nMedio + 0.115770059;
  tempAux = (tensao - 0.45) / 0.01;

  if(temperatura - tempAux > 0.2 || temperatura - tempAux < -0.2) temperatura = tempAux;
}
```

Figura 14 - Função para atualizar o valor da variável temperatura

```
void updateValueLcdTemperatura()
{
  lcd.setCursor(5,1);
  lcd.print(temperatura);
}
```

Figura 15 - Função para atualizar o valor de temperatura no LCD

3.5. Servidor NTP

Com o objetivo de manter o horário do sistema sempre preciso, o firmware utiliza um servidor NTP (pool.ntp.org). Este servidor sincroniza o relógio do dispositivo, garantindo que as leituras dos sensores e as publicações MQTT sejam registradas com timestamps confiáveis. Dessa forma, asseguramos a integridade temporal das informações coletadas.

Para isso, criou-se uma função “updateTimestamp” (figura 17) que mantém o horário do sistema frequentemente atualizado.

```
void updateTimestamp()
{
  timeClient.update();
  timeStamp = timeClient.getEpochTime();
}
```

Figura 16 - Função para atualizar o valor da variável timestamp

3.6. Implementação da conexão WiFi no módulo ESP32

O ESP32 possui embutido em seu hardware um módulo Wi-Fi que opera em uma frequência de 2.4Ghz e implementa o protocolo TCP/IP, no padrão 802.11 b/g/n Wi-Fi MAC Protocol. Para fazer uso desse recurso, primeiro foi preciso importar a biblioteca “WiFi.h”, que permite fazer uso do recurso Wi-Fi.

Em seguida, definiu-se as variáveis globais, referentes ao SSID e senha da rede em que foi feita a conexão. O código abaixo exemplifica a criação dessas variáveis:

```
// Variáveis para conectar ao Wi-Fi
const char* ssid = "<ssid>";
const char* password = "<senha>";
```

Figura 17 - Variáveis referente a rede Wi-Fi utilizada no ESP32

Após inserir um ssid e senha válidos, foi definida uma função para tratar diretamente a questão da conexão e também como saber se foi concluída com sucesso. Na imagem abaixo, vemos as linhas iniciais da função que, basicamente, inicia uma variável i, que servirá como um contador,e depois algumas configurações referentes ao display LCD, para mostrar a mensagem “WiFi Conectando”.

```
void conectarWifi()
{
  int i = 0;

  lcd.clear();
  lcd.setCursor(5,0);
  lcd.printstr("WiFi");
  lcd.setCursor(1,1);
  lcd.printstr("Conectando");
```

Figura 18 - Configurações iniciais da função “conectarWifi”

A segunda parte, tenta conectar-se ao Wi-Fi propriamente dito, e em seguida entra em um loop de verificação do status da conexão. Caso a conexão não tenha sido efetuada, verifica-se se o contador já atingiu o valor 4. Caso tenha atingido, seta um espaço em branco após o fim da palavra “Conectando”, inserida na configuração inicial, e faz com que o contador “i” receba o valor 0 novamente. Caso não seja, insere um “.” a cada iteração do loop While, no fim de “Conectando”, para produzir um efeito visual no display.

A partir do momento que a conexão é efetuada com sucesso, limpa-se o display LCD, e mostra o status “Conectado”, seguido na linha de baixo pelo IP atribuído ao ESP32. Depois de 5 segundos, é chamado a função “setFormatLcdTemperatura()” que será responsável por ser a tela inicial do display, se tratando das leituras dos sensores. A imagem abaixo mostra a implementação realizada:

```
  // Conecta-se à rede Wi-Fi
  WiFi.begin(ssid, password);

  // Tentativa de estabelecer conexão
  while(WiFi.status() != WL_CONNECTED)
  {
    if(i == 4)
    {
      lcd.setCursor(11,1);
      lcd.printstr("    ");
      i = 0;
      lcd.setCursor(11,1);
    }
    lcd.printstr(".");
    i++;
    delay(1000);
  }

  // Após a conexão ser estabelecida
  lcd.clear();
  lcd.setCursor(1,0);
  lcd.printstr("WiFi conectado");
  lcd.setCursor(1,1);
  lcd.print(WiFi.localIP());
  delay(5000);

  // Começar com formatação da temperatura
  setFormatLcdTemperatura();
}
```

Figura 19 - Efetivando a conexão Wi-Fi

3.7. Configuração do ESP32 como cliente MQTT

Para configurar o ESP32 como cliente de um servidor MQTT, primeiro foi definido algumas variáveis globais, com informações a respeito da conexão como: hostname do broker, porta do broker e credenciais válidas para tal broker.

Além disso, usou-se as bibliotecas“PubSubClient.h” e “WiFiClientSecure.h”, que permitem criar um objeto PubSubClient, que facilita a comunicação com brokers MQTT, e a criação de um cliente WiFi seguro para se comunicar com o broker, respectivamente. Ele usa o cliente WiFi seguro para estabelecer uma conexão segura usando TLS/SSL com o broker MQTT.

A figura abaixo exibe a definição dessas variáveis:

```
// Variáveis para se conectar ao broker MQTT
const char* brokerMqtt = "80fe29ce8268427c9a4a9aeb6cabf603.s2.eu.hivemq.cloud";
int brokerPort = 8883;
const char* mqtt_username = "redes1";
const char* mqtt_password = "projetoredes1";

WiFiClientSecure espClient;
PubSubClient MQTT(espClient);
```

Figura 20 - Variáveis referente ao Broker MQTT

Para efetivar a conexão, foi criada a função “iniciaMQTT” e “reconectaMQTT”:

```
// Inicia a conexão do esp32 com o broker MQTT
void iniciaMqtt()
{
  espClient.setCACert(root_ca);
  MQTT.setServer(brokerMqtt, brokerPort);
  reconectaMQTT();
}
```

Figura 21 - Função para iniciar conexão com o broker

```
// Conecta o esp32 com o broker MQTT caso esteja desconectado
void reconectaMQTT()
{
  while(!MQTT.connected())
  {
    Serial.print("* Tentando se conectar ao Broker MQTT: ");
    Serial.println(brokerMqtt);
    if (MQTT.connect(idMQTT, mqtt_username, mqtt_password))
    {
      Serial.println("Conectado com sucesso ao broker MQTT!");
    }
    else
    {
      Serial.println("Falha ao reconectar no broker.");
      Serial.println("Havera nova tentatica de conexao em 2s");
      delay(2000);
    }
  }
}
```

Figura 22 - Função para reconectar o ESP ao servidor MQTT

3.8. Envio dos dados dos sensores para o broker MQTT

Para enviar os dados dos sensores ao broker, foi criado uma função chamada “publishValores()”. Essa função é chamada na “loop()”, toda vez que as leituras do sensores são realizadas e a verificação presente na imagem abaixo é feita. Basicamente, a ideia é publicar os valores a cada 30 segundos, e para isso foi medido o tempo que demora para cada loop acontecer. Com a variável “delayPub”, é configurado nesse intervalo de tempo, podendo ser maior ou menor, dependendo da necessidade da aplicação.

```
// Estrutura que detecta o momento de publicar os valores
if(numLoop >= ((60000 * delayPub) / (delayMs + 622)))
{
  if(!MQTT.connected()) reconectaMQTT();
  publishValores();
  numLoop = 0;
}

numLoop++;

delay(delayMs);
```

Figura 23 - Chama da função “publishValores” a cada 30 segundos

A função, propriamente dita, transformará todas as leituras feitas, juntamente do timestamp atual, em uma string concatenada no formato csv, separado pelo caractere “;”. A imagem abaixo exibe tal concatenação:

```
void publishValores()
{
  // Formata a string com os valores sensoriados
  String csvData = String(temperatura) + ";" + String(umidade) + ";" + String(pressao) + ";" + String(timeStamp);
```

Figura 24 - Concatenação dos valores para o padrão CSV

Após formatar a string que mantém todos os dados de leitura dos sensores, é necessário convertê-la em um vetor de caracteres, para que o Publish possa ser realizado no tópico “dados”, conforme exibido na imagem a seguir:

```
// Converte a String para um array de caracteres (char[])
char payloadCsv[csvData.length() + 1];
csvData.toCharArray(payloadCsv, csvData.length() + 1);

// Publica o valor no tópico desejado
MQTT.publish("dados", payloadCsv);
```

Figura 25 - Conversão para um array de caracteres e publicação dos valores

3.9. Recepção de dados do broker MQTT para acionamento dos leds de irrigação e adubação

Para fins de melhor visualização, a parte de recepção dos dados referente ao acionamento dos leds de irrigação e semeadura foi feita usando um outro ESP32, dedicado somente a isso. Toda a parte de conexão e configurações são idênticas às que já foram apresentadas anteriormente, diferenciando-se apenas que agora o cliente MQTT se inscreve em dois tópicos: irrigação e semeadura.

Uma função chamada “callback” foi implementada, com o intuito de monitorar cada payload que é recebido. Como são dois tópicos diferentes, é preciso verificar o qual tal payload corresponde. Para isso, é comparado o nome do tópico referente ao payload com as variáveis globais definidas, conforme imagem abaixo:

```
const char* irrigacao_topic= "irrigacao";
const char* semeadura_topic="semeadura";
```

Figura 26 - Definição do nome dos tópicos

Depois de comparar e associar o tópico corretamente, mais uma comparação é feita para determinar o valor presente em tal tópico, se é “1” ou “0”, onde no caso de ser “1” o led é aceso e, “0” apagado. Isso vale tanto para a semeadura como para a irrigação. A imagem abaixo exibe a implementação dessa função:

```
void callback(char* topic, byte* payload, unsigned int length) {
  String incommingMessage = "";
  for (int i = 0; i < length; i++) incommingMessage+=(char)payload[i];
  Serial.println("Message arrived ["+String(topic)+"] "+incommingMessage);

  if( strcmp(topic,irrigacao_topic) == 0){
    if (incommingMessage.equals("1")) {
      digitalWrite(LED_1, HIGH);
    } else {
      digitalWrite(LED_1, LOW);
    }
  }


  if( strcmp(topic,semeadura_topic) == 0){
    if (incommingMessage.equals("1")) {
      digitalWrite(LED_2, HIGH);
    } else {
      digitalWrite(LED_2, LOW);
    }
  }
}
```

Figura 27 - Recebimento e tratamento do payload que aciona os LED's

4. RESULTADOS E DISCUSSÃO

Os resultados obtidos demonstram a viabilidade e funcionalidade do projeto de microcontrolador proposto que visa desenvolver um sistema de monitoramento ambiental utilizando um microcontrolador (ESP32) capaz de medir temperatura, umidade e pressão, transmitindo esses dados por meio de Wi-Fi para um broker MQTT privado, hospedado na nuvem HiveMQ.

No entanto, algumas considerações podem ser levantadas, é importante considerar alguns aspectos críticos para o aprimoramento contínuo do projeto. A estabilidade da conexão Wi-Fi merece atenção para garantir a consistência na transmissão de dados em diferentes condições de rede. Além disso, é fundamental assegurar a precisão das leituras dos sensores, especialmente em ambientes com variações de temperatura, umidade e pressão.

As imagens documentam fases distintas do processo, desde a conexão inicial do circuito à rede Wi-Fi até a exibição das leituras nos dispositivos de saída e o subsequente tratamento dos dados recebidos para acionar componentes externos. Essas representações visuais ilustram as etapas-chave do funcionamento do sistema, evidenciando sua capacidade de coletar, processar e interagir com os dados ambientais capturados.

A partir das Figuras 28 e 29, observamos o processo de conexão do circuito à rede Wi-Fi. O circuito demonstrou a capacidade de estabelecer uma conexão bem-sucedida com a rede, conforme indicado pela transição do estado de tentativa de conexão (Figura 28) para a confirmação de conexão (Figura 29). Essa etapa é fundamental para a transmissão eficiente dos dados coletados para o broker MQTT.

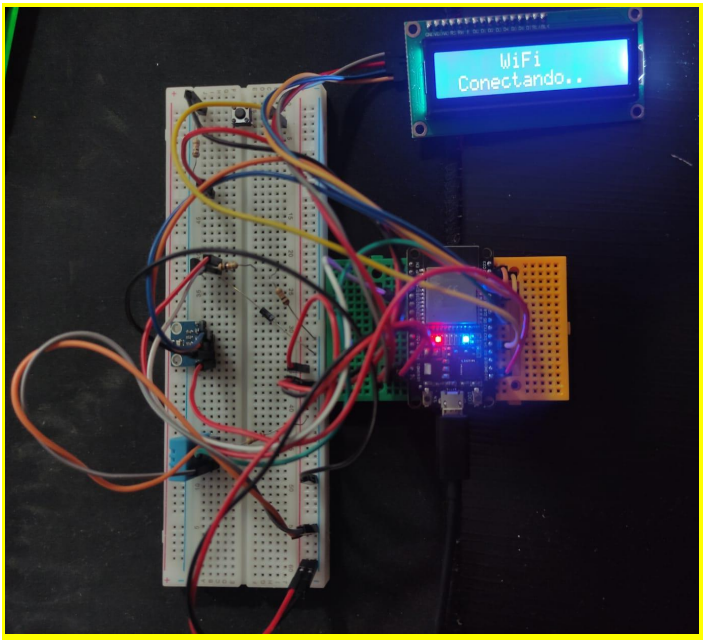

Figura 28 - Circuito tentando se conectar ao WiFi.

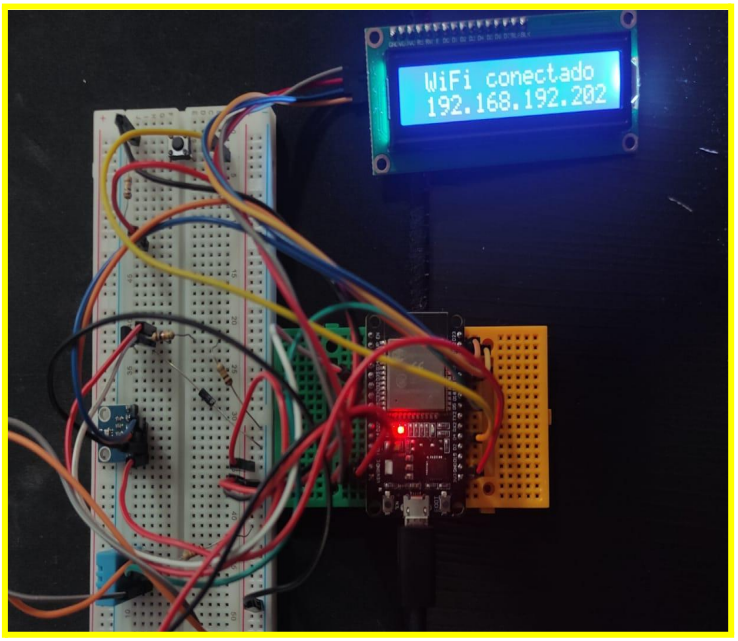

Figura 29 - Circuito conectado ao WiFi.

As Figuras 30, 31 e 32 exibem a funcionalidade do circuito na leitura e exibição dos valores de temperatura, umidade e pressão, respectivamente, no LCD. As leituras aparentam estar alinhadas com as expectativas do projeto, evidenciando a capacidade do microcontrolador em coletar com precisão as informações dos sensores correspondentes.

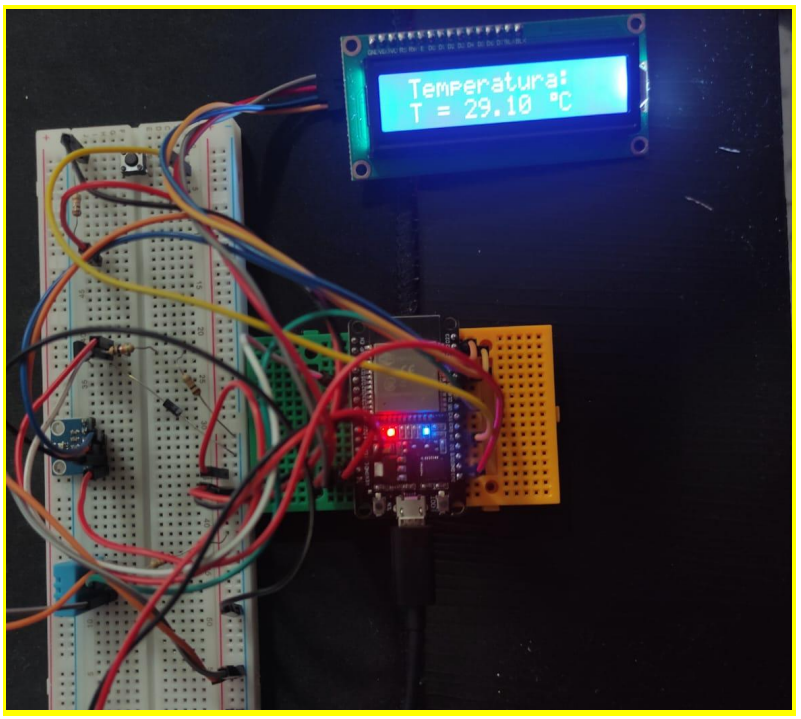

Figura 30 - Circuito mostrando a temperatura medida no LCD.

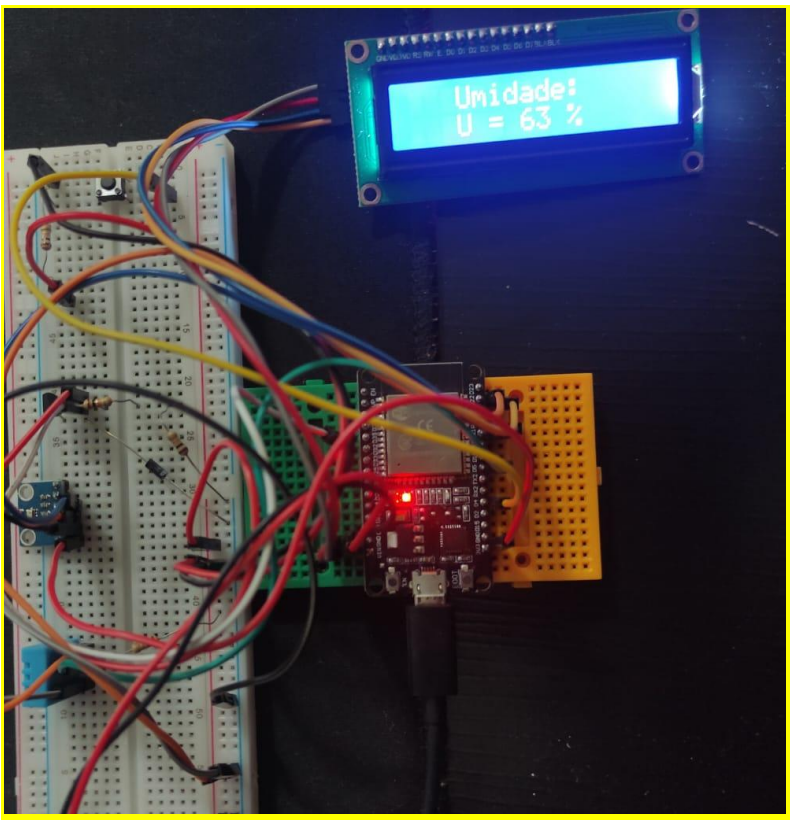

Figura 31 - Circuito mostrando a umidade medida no LCD.

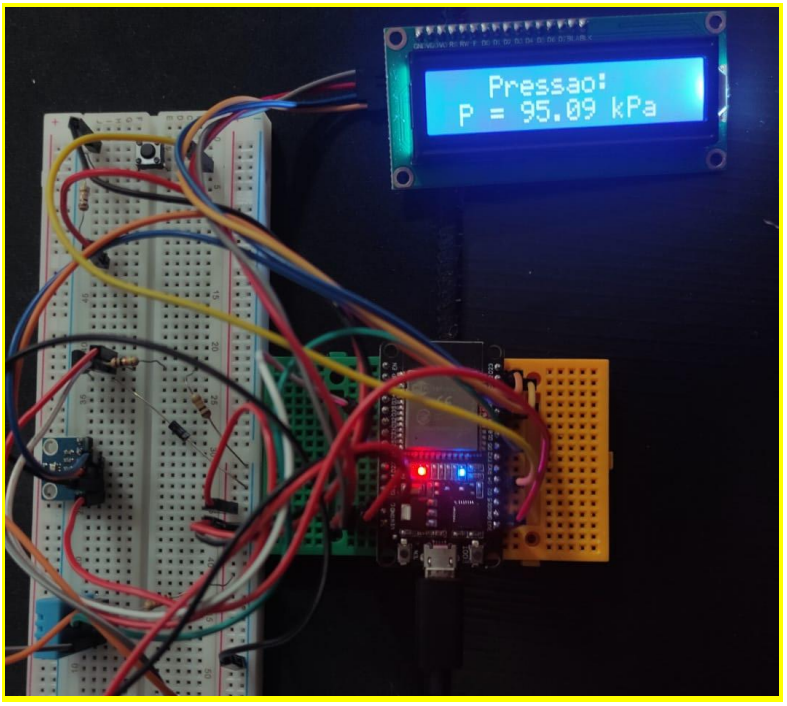

Figura 32 - Circuito mostrando a pressão medida no LCD.

Na Figura 33, é possível observar o circuito de recebimento e tratamento do payload, resultando no acionamento dos LED's. Esta etapa representa a capacidade do sistema em receber dados do broker MQTT e executar ações correspondentes, neste caso, a ativação ou não dos LED's com base nas informações recebidas.

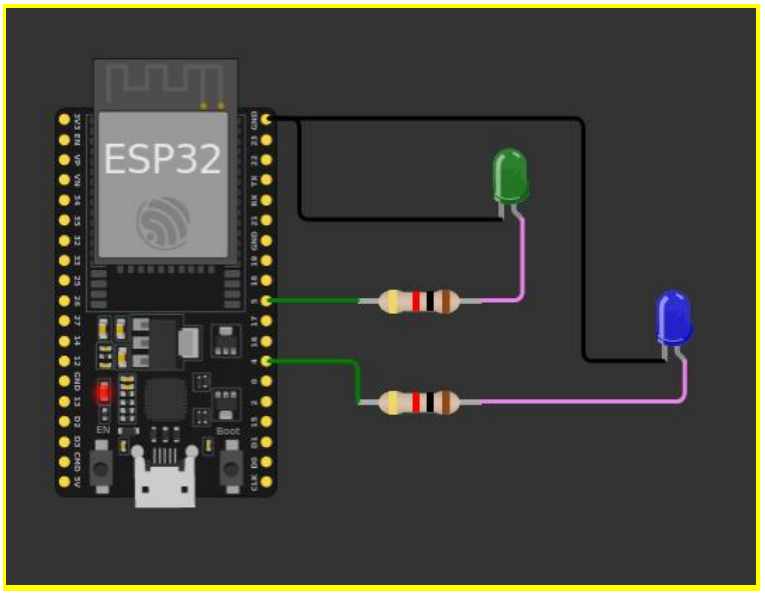

Figura 33 - Circuito do recebimento e tratamento do payload que aciona os LED’s

5. CONCLUSÃO

Em síntese, a implementação do projeto de agricultura de precisão revelou-se bem-sucedida ao empregar uma combinação de sensores (LM35, DHT11, BMP280), um microcontrolador (ESP32) e tecnologias como Wi-Fi, servidor NTP e broker MQTT. Essa abordagem proporcionou a transmissão rápida de dados e comunicação assíncrona entre dispositivos conectados.

A interação eficaz desses componentes conferiu ao sistema a capacidade de coletar informações em tempo real, um atributo crucial para a tomada de decisões ágeis no contexto agrícola.

Em última análise, a convergência desses sensores, o microcontrolador ESP32 e as tecnologias de conectividade podem significar um avanço significativo na modernização e otimização dos processos agrícolas. Há potencial para incrementar a produtividade, reduzir desperdícios e fomentar práticas agrícolas sustentáveis.

6. REFERÊNCIAS

Amazon Web Services. O que é IoT(Internet of Things)? Disponível em: https://aws.amazon.com/pt/what-is/iot/. Acesso em: 20 de setembro de 2023

Via UFSC. Afinal, o que é IoT? Disponível em: https://via.ufsc.br/afinal-o-que-e-iot/. Acesso em: 20 de setembro de 2023.

Adafruit. Datasheet BMP280. Disponível em: https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf. Acesso em: 22 de setembro de 2023.

ETC. Datasheet DHT11. Disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/1440068/ETC/DHT11.html. Acesso em: 10 de setembro de 2023.

Adafruit et al. Adafruit BMP280 Library. Disponível em: https://github.com/adafruit/Adafruit_BMP280_Library. Acesso em: 02 de outubro de 2023.

dhrubasaha08 et al. DHT11 Library for Arduino. Disponível em: https://github.com/dhrubasaha08/DHT11. Acesso em: 02 de outubro de 2023.

mateusz-szafraniec et al. LiquidCrystal_I2C.h. Disponível em: https://github.com/johnrickman/LiquidCrystal_I2C/blob/master/LiquidCrystal_I2C.h. Acesso em: 03 de outubro de 2023.

Manual da Eletrônica. Sensor de Temperatura LM35: Características e Aplicações. Disponível em: https://www.manualdaeletronica.com.br/sensor-temperatura-lm35-caracteristicas-aplicacoes/ . Acesso em: 03 de dezembro de 2023.