Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Lucrare de laborator Nr. 7

Disciplina: Internetul lucrurilor (IOT)
Tema: Aplicații cu comunicații interne, externe și internet

A efectuat: Popa Cătălin, gr. TI-211

A verificat: Cristian Lupan, asist. univ.

Chișinău 2024

7.1 Problema propusă:

Realizarea applicatiilor cu comunicatii interne - intre coponentele electronice ale dispozitivului cu interfete HW de comunicare.

Sa se realizeze o aplicatie ce va implementa comunicatiile intre echipamente dupa cum urmeaza:

Protocol fizic de comunicare - Comunicarea intre DOUA Microcontrollere prin interfata I²C

a. MCU1 - implementeaza sensorul digital cu interfata I2C pentru sensorul ultrasonic HCS-04, unde se executa colectarea datelor de la interfata sensorului si se retransmite catre interfata I2C la detectarea unei cereri de citire a datelor.
b. MCU2 - executa cererea prin interfata I2C catre sesorul digital ultrasonic (MCU+HCS-04) si afiseaza datele pe interata seriala.

7.2 Problema propusă:

Realizarea applicatiilor cu comunicatii externe - Intre dispozitive cu protocoale de comunicare SW.

Sa se realizeze o aplicatie ce va implementa comunicatiile intre echipamente dupa cum urmeaza:

Protocol logic de comunicare - cererea de date prin interfata serial, in format text respectand un protocol de comunicare care va avea campurile:

Protocol Frame

STX	P nr	SRC	DST	P_ID	CMD	Payload (4 bytes)	SC	ETX

1. indicator de start pachet
2. indicator de sfarsit
3. contorizare pachete
4. ID emitator
5. ID receptor
6. tipul pachetului
7. <alte campuri optional>
8. date pachet – Payload
9. suma de control - suma tuturor valorilor numerice din pachet

Cererile venite din interfata seriala vor fi verificate dupa patern, si in caz de pachet valid se va intereta comanda. se va raspunde cu un pachet conform aceluias protocol.implementare la o comanda obligatorie pentru implementare este cererea de date de la sensorul digital implementat in lab precedent.sa si implementezi inca o camanda la alegere, pentru diversitate.

7.3 Problema propusă:

Realizarea applicatiilor cu comunicatii Internet - prin protocolul MQTT penru interactiunea cu o resursa Cloud.

Sa se realizeze o aplicatie ce va implementa comunicatiile intre echipamente dupa cum urmeaza:

1. Realizarea unei aplicatii decomunicare Internet prin protocolul MQTT penru interactiunea cu o resursa Cloud
2. Colectarea datelor de la sensori si trimitere catre un broker MQTT
3. urmarirea mesajelor de la un proker MQTT si setarea starii unui actuator la alegere
4. Datele sunt vizualizate si controlate de la un dashboard Internet (ThingsBoard sau HiveMQ).

Obiective

1. Crearea unui sistem robust de transmitere a datelor între două microcontrolere prin protocolul I²C..
2. Asigurarea unei comunicări coerente și verificabile între dispozitive prin interfața serială.
3. Dezvoltarea unei aplicații IoT care să faciliteze colectarea, transmiterea și controlul datelor prin Internet utilizând protocolul MQTT.

Introducere

Microcontrolerele permit controlul și monitorizarea dispozitivelor fizice, aducând tehnologia mai aproape de utilizator prin intermediul unei interfețe intuitive și simplificate. Un exemplu clasic de interacțiune între utilizator și dispozitive fizice este controlul unui LED prin apăsarea unui buton sau trimiterea de comenzi printr-o interfață serială. Această tehnologie este utilizată într-o gamă largă de aplicații, de la sisteme de iluminat inteligente până la dispozitive electronice complexe utilizate în automatizări industriale. Unul dintre avantajele microcontrolerelor este costul redus și flexibilitatea lor, permițând implementarea rapidă de prototipuri și teste pentru diverse aplicații IoT. Microcontrolerele precum cele din familia Arduino sau ESP sunt folosite frecvent pentru prototiparea sistemelor IoT datorită ecosistemului extins de biblioteci și comunități de dezvoltatori care susțin aceste platforme .

Materiale și metode

Arduino UNO - platformă de microcontroler open-source bazată pe microcontrolerul ATmega328P. Este ușor de utilizat și permite conectarea ușoară a componentelor externe prin pini digitali și analogici. Acesta este ideal pentru prototiparea de sisteme IoT și proiecte electronice interactive. Afișarea este în figura 1.1.

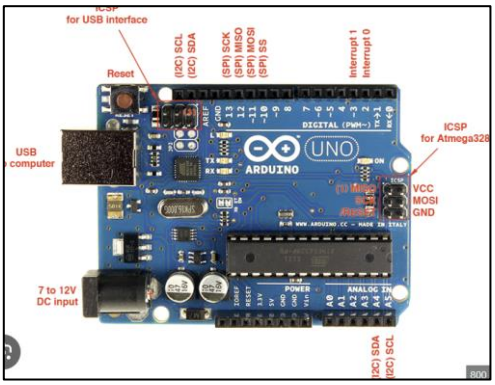


Fig. 1.1. Arduino UNO

- 14 pini digitali I/O (din care 6 sunt PWM);
- 6 intrări analogice;
- memorie flash de 32 KB;
- tensiune de operare: 5V;
- interfață de programare: Arduino IDE.

Rezistor - Rezistorul este utilizat pentru a limita curentul ce trece prin LED, prevenind arderea acestuia. Pentru a proteja LED-urile, se vor folosi rezistoare de aproximativ 220Ω. Afișarea este în figura 1.2.

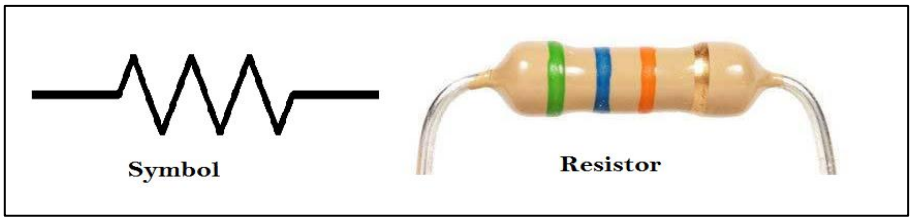


Fig. 1.2. Rezistor

Fire Jumper - Firele jumper sunt utilizate pentru a face legături electrice între diverse componente și plăcuța Arduino. Acestea permit conectarea componentelor fără a fi necesară lipirea acestora. Afișarea este în figura 1.3.

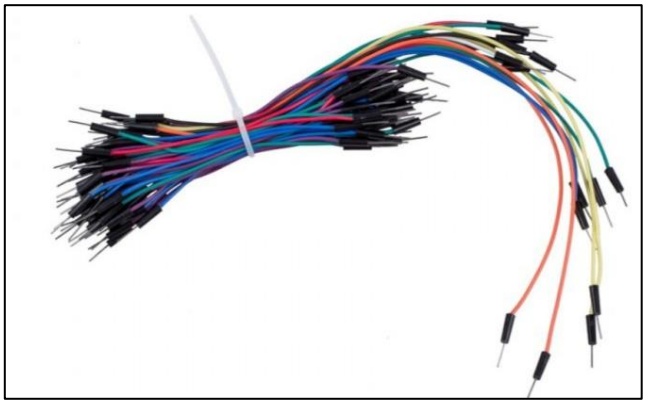


Fig. 1.3. Fire Jumper

LED-ul - (diodă emițătoare de lumină) este utilizată pentru a semnaliza starea sistemului în funcție de comenzile primite sau de validitatea codului introdus pe tastatură. Afișarea este în figura 1.4.

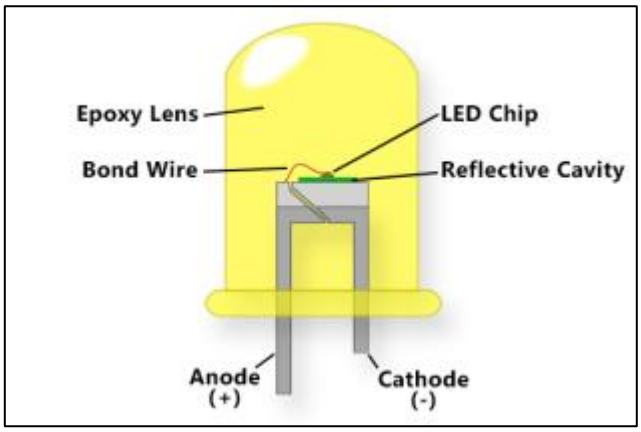


Fig. 1.4. Fire Jumper

ESP32 - un microcontroller dezvoltat de Espressif Systems, renumit pentru performanțele sale și pentru capacitățile extinse de conectivitate. Este o soluție foarte populară în dezvoltarea de proiecte IoT (Internet of Things), datorită versatilității, puterii de procesare și prețului accesibil. Afișarea este în figura 1.5.

Fig. 1.5. ESP32

Metode

Configurarea Arduino UNO: Arduino va fi configurat folosind Arduino IDE. Se va implementa codul pentru a citi inputurile de la potențiometru și encoder, pentru ca apoi să aplice acțiuni asupra dispozitivelor externe și afișarea pe LCD.

Interacțiunea cu utilizatorul: Sistemul va primi comenzi potențiometru și sencoder și va răspunde prin intermediul LCD-ului şi serial monitor.

Validare în simulator: Codul și conexiunile vor fi testate inițial într-un simulator de circuite, cum ar fi Tinkercad, Wokwi, pentru a verifica funcționalitatea corectă. După simulare, se va testa și pe un circuit real.

7.1

Pentru prima sarcină am avut nevoie de un senzor ultrasonic și 2 plăci de arduino. Simularea realizată este în figura 1.6. Arduino 1 ia datele de la potențiometru și le trimite la arduino 2, care la rândul său le afișează în serial monitor.

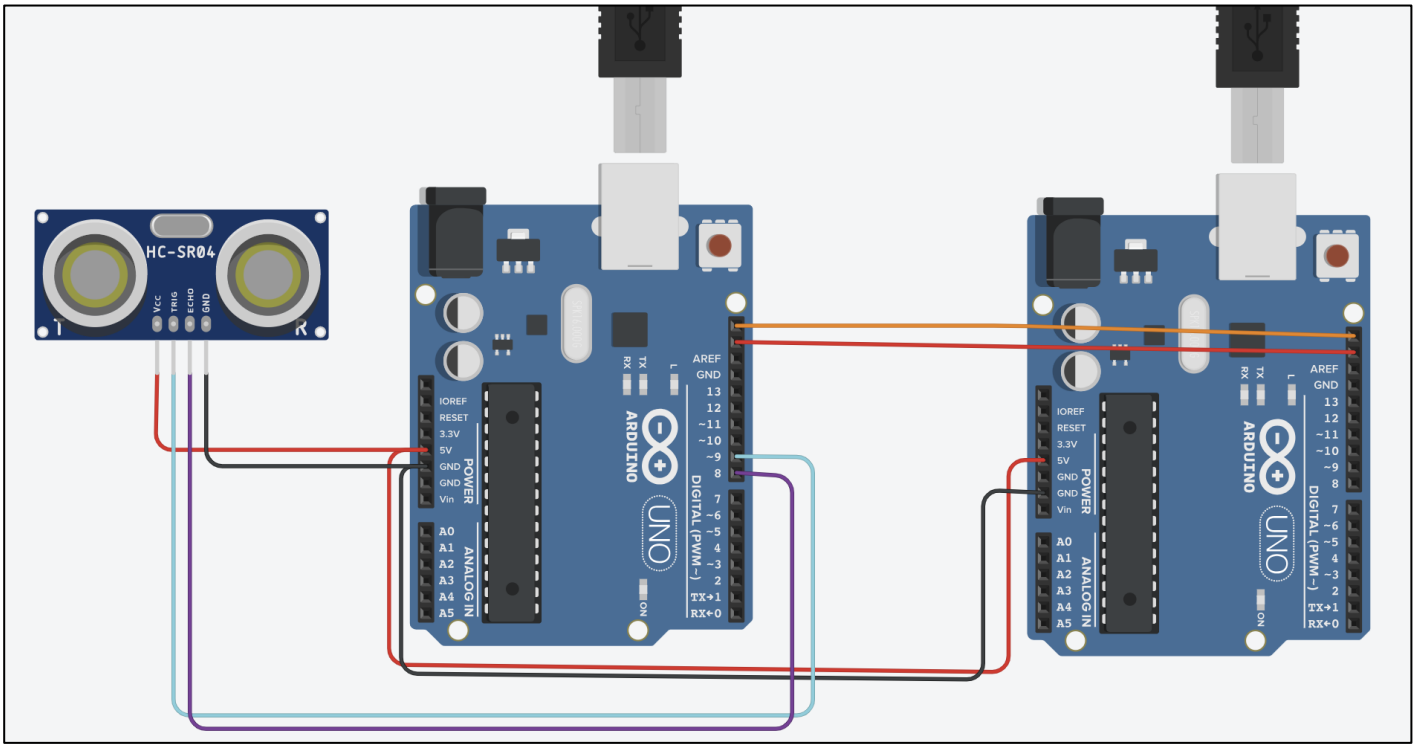


Figura 1.6 – Simulare comunicații cu periferii – I^2C

Mai întâi am definit pinii la un senzor ultrasonic. Apoi am creat funcția readDistance în MCU1. Aici are loc activarea senzorului, măsurarea duratei și calcularea distanței.

```
long readDistance() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);
    long distance = duration * 0.034 / 2;
    return distance;
}
```

Se trimite un semnal LOW timp de 2 microsecunde pentru a asigura că senzorul este resetat. Apoi se trimite un semnal HIGH timp de 10 microsecunde pentru a activa senzorul. Se folosește pulseIn pentru a măsura timpul în care pinul echoPin este HIGH, ceea ce corespunde timpului necesar pentru ca unda sonoră să se întoarcă la senzor. Distanța se calculează folosind formula duration * 0.034 / 2, unde 0.034 este viteza sunetului în cm/µs și împărțim la 2 deoarece măsurăm drumul dus-întors.

Apoi configurăm setup.

```
void setup() {
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    Wire.begin(8);
    Wire.onRequest(requestEvent);
}
```

trigPin este setat ca OUTPUT (ieșire), iar echoPin ca INPUT (intrare). Se inițiază comunicația I2C cu adresa 8. Se înregistrează funcția requestEvent, care va fi apelată când un dispozitiv I2C solicită date. Această funcție rămâne goală în acest cod, deoarece nu sunt necesare acțiuni repetate.

Mai departe a fost creată funcția requestEvent. Se apelează funcția readDistance pentru a obține distanța măsurată. Distanța este trimisă prin I2C către dispozitivul care a făcut cererea.

```
void requestEvent() {
    long distance = readDistance();
    Wire.write((byte*)&distance, sizeof(distance));
}
```

Apoi am creat funcționalul pentru MCU2.

```
void setup() {
    Wire.begin();
    Serial.begin(9600);
}
```

Se inițiază comunicația I2C fără a specifica o adresă (dispozitivul va acționa ca master). Se configurează viteza de comunicație serială la 9600 bps pentru a putea trimite date către monitorul serial.

```
void loop() {
    Wire.requestFrom(8, sizeof(long));
    long distance = 0;
    if (Wire.available() == sizeof(long)) {
        Wire.readBytes((char*)&distance, sizeof(distance));
    }
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    delay(500);
}
```

Se face o cerere către dispozitivul cu adresa 8 (care este senzorul de distanță din primul cod) pentru a primi un număr de tip long. Se verifică dacă datele sunt disponibile și dacă numărul de bytes primit este corect. Dacă datele sunt disponibile, se citesc și se stochează în variabila distance. Distanța măsurată este afișată pe monitorul serial. O întârziere de 500 ms este adăugată pentru a evita solicitările excesive.

Aceste două fragmente de cod colaborează pentru a citi distanța utilizând un senzor ultrasonic conectat prin I2C. Primul cod acționează ca un dispozitiv slave care măsoară distanța și răspunde la cererile masterului, iar al doilea cod acționează ca master care solicită aceste date și le afișează pe monitorul serial. Această structură permite o comunicare eficientă între două dispozitive folosind protocolul I2C.

7.2

Pentru realizarea comunicării între două plăci Arduino prin protocolul SW Serial, am avut nevoie de două plăci Arduino Uno, un ultrasonic și fire de conectare. Simularea este afișată în figura 1.7.

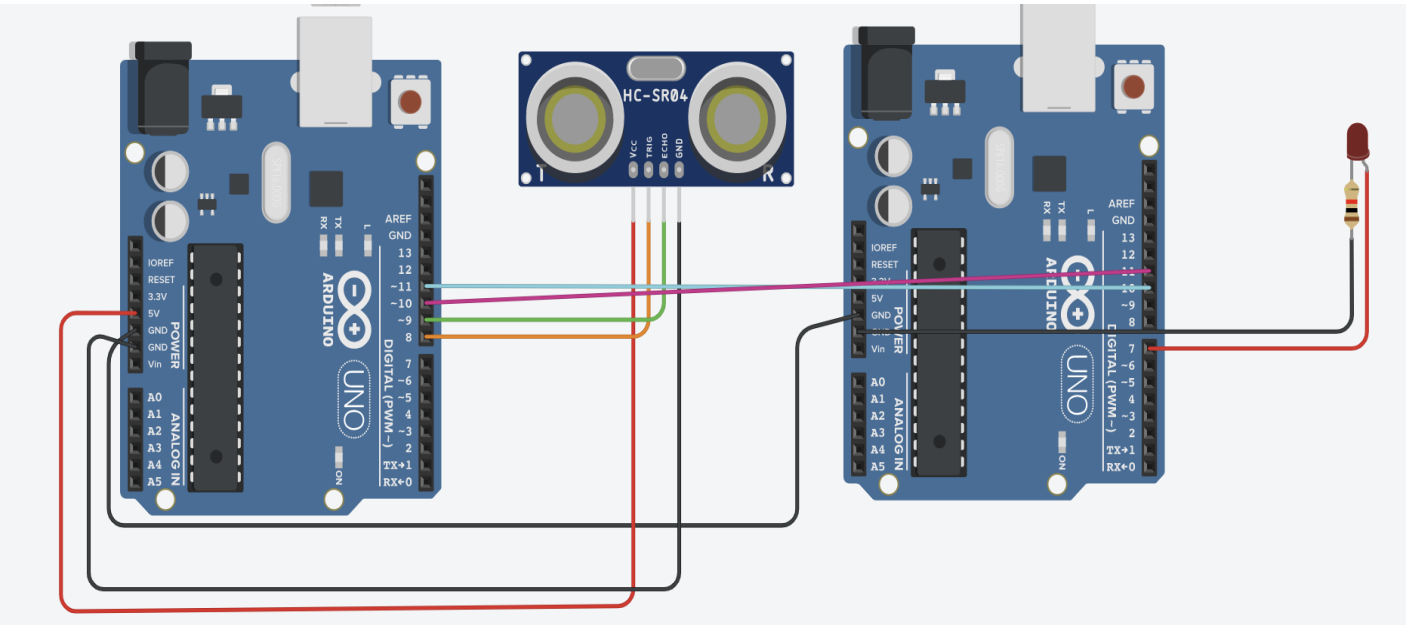


Figura 1.7. Simularea prin protocolul SW Serial

Pentru implementarea comunicării între dispozitive, am utilizat o metodă de transmitere a datelor bazată pe un protocol serial customizat. Această metodă respectă structura și cerințele unui protocol definit pentru trimiterea și recepționarea pachetelor de date, asigurând integritatea, identificarea și validarea informațiilor.

Structura pachetului

- STX: indică începutul pachetului, astfel încât receptorul să se sincronizeze corect cu fluxul de date;
- SRC: identifică dispozitivul sursă;
- DST: identifică dispozitivul destinație;
- P_ID: numărul pachetului;
- CMD: specifică tipul comenzii transmise;
- Payload: informația utilă transmisă, distanța măsurată de senzorul HC-SR04;
- SC: o sumă de control calculată ca suma modul 256 a tuturor celorlalte câmpuri. Aceasta verifică integritatea datelor transmise;
- ETX: indică sfârșitul pachetului, permițând receptorului să știe că pachetul este complet.

MCU1

Pentru a realiza această comunicare, este nevoie de librăria SoftwareSerial. Inițial definim SoftwareSerial pe pinii 10 (TX) și 11 (RX) SoftwareSerial mySerial(10, 11).

În setup am definit pinii și am deschis o comunicare atât cu Monitorul Seriall, cât și cu MCU2 prin SoftwareSerial.

```
void setup() {
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.begin(4800);
  mySerial.begin(4800);
  Serial.println("MCU1 - Pregătit pentru trimiterea datelor");
}
```

În continuare am creat funcția de măsurare a distanței cu senzorul HC-SR04. Mai întâi asigurăm un semnal LOW înainte de a trimite pulsul, facem o pauză de 2 microsecunde, trimitem pulsul HIGH pentru 10 microsecunde și oprim pulsul. Apoi măsurăm durata semnalului ECHO și calculăm distanța pe baza duratei.

```
long readDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
```

```
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = duration * 0.034 / 2;
  return distance;
}
```

Următoarea funcție creată este sendPacket(), care se folosește pentru a trimite și crea un pachet conform protocolului.

```
    void sendPacket(byte cmd, int payload) {
  packetID++;
  if (packetID > 255) packetID = 0;
  byte stx = 0x02;
  byte src = 0x01;
  byte dst = 0x02;
  byte p_id = packetID;
  byte payloadHigh = payload >> 8;
  byte payloadLow = payload & 0xFF;
  byte checksum = (src + dst + p_id + cmd + payloadHigh + payloadLow) % 256;
  byte etx = 0x03;
  mySerial.write(stx);
  mySerial.write(src);
  mySerial.write(dst);
  mySerial.write(p_id);
  mySerial.write(cmd);
  mySerial.write(payloadHigh);
  mySerial.write(payloadLow);
  mySerial.write(checksum);
  mySerial.write(etx);
}
```

În final în loop() facem apelare la funcția sendPacket de 3 ori. Aici se trimite pachet cu distanță, apoi cu comanda LED_ON și comanda LED_OFF. Între comenzi are loc o pauză de 5 secunde, pentru a trimite pachetul cu date întreg.

```
void loop() {
  long distance = readDistance();
  sendPacket(0x01, distance);
  delay(5000);
  sendPacket(LED_CMD, 1);
  delay(5000);
  sendPacket(LED_CMD, 0);
  delay(5000);
}
```

MCU2

Pentru MCU2, la fel au fost definiți aceeași pini de RX și TX, și a fost creat un SoftwareSerial. La fel a fost defini un pin pentru led-ul care se va aprinde la comandă. În setup am configurat ledul, am deschis o comunicare cu Monitor Serial și cu MCU1 la o frecvență de 4800.

```
void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(4800);
  mySerial.begin(4800);
  Serial.println("MCU2 - Pregătit de primit datele");
}
```

În loop mai întâi se verifică dacă există date pe linia serială, apoi are loc citirea lor. Mai întâi are loc verificarea integrității pachetului, apoi se calculează checksum-ul pentru validate. Dacă suma primită a pachetului este egală cu cea trimisă și pachetul se termină cu 0x03 pachetul se consideră valid. Apoi urmează 3 if-uri unde se verifică dacă se așteaptă afișarea distanței, comandă de a apinrde LED-ul sau de a stinge LED-ul.

```
void loop() {
  if (mySerial.available() > 0) {
    byte stx = mySerial.read();
    if (stx == 0x02) { // Verificam daca pachetul incepe cu STX
      byte src = mySerial.read();
      byte dst = mySerial.read();
      byte p_id = mySerial.read();
      byte cmd = mySerial.read();
      byte payloadHigh = mySerial.read();
      byte payloadLow = mySerial.read();
      byte checksum = mySerial.read();
      byte etx = mySerial.read();

      byte calculatedChecksum = (src + dst + p_id + cmd + payloadHigh + payloadLow) %
256;

      if (checksum == calculatedChecksum && etx == 0x03) {
        int payload = (payloadHigh << 8) | payloadLow;

        if (cmd == 0x01) {
          Serial.print("Distanta primita: ");
          Serial.print(payload);
          Serial.println(" cm");
        } else if (cmd == 0x02) {
          if (payload == 1) {
            digitalWrite(LED_PIN, HIGH);
            Serial.println("LED aprins");
          } else if (payload == 0) {
            digitalWrite(LED_PIN, LOW);
            Serial.println("LED stins");
          }
```

```
        }
      } else {
        Serial.println("Eroare: Checksum invalid sau ETX gresit.");
      }
    } else {
      Serial.println("Eroare: Pachet invalid (fara STX).");
    }
  }
  delay(5000);
}
```

Afișare test.

MCU1

```
Trimit pachet: CMD=0x1 PAYLOAD=112
Trimit pachet: CMD=0x2 PAYLOAD=1
Trimit pachet: CMD=0x2 PAYLOAD=0
```

MCU2

```
Distanta primita: 112 cm
LED aprins
LED stins
```

7.3

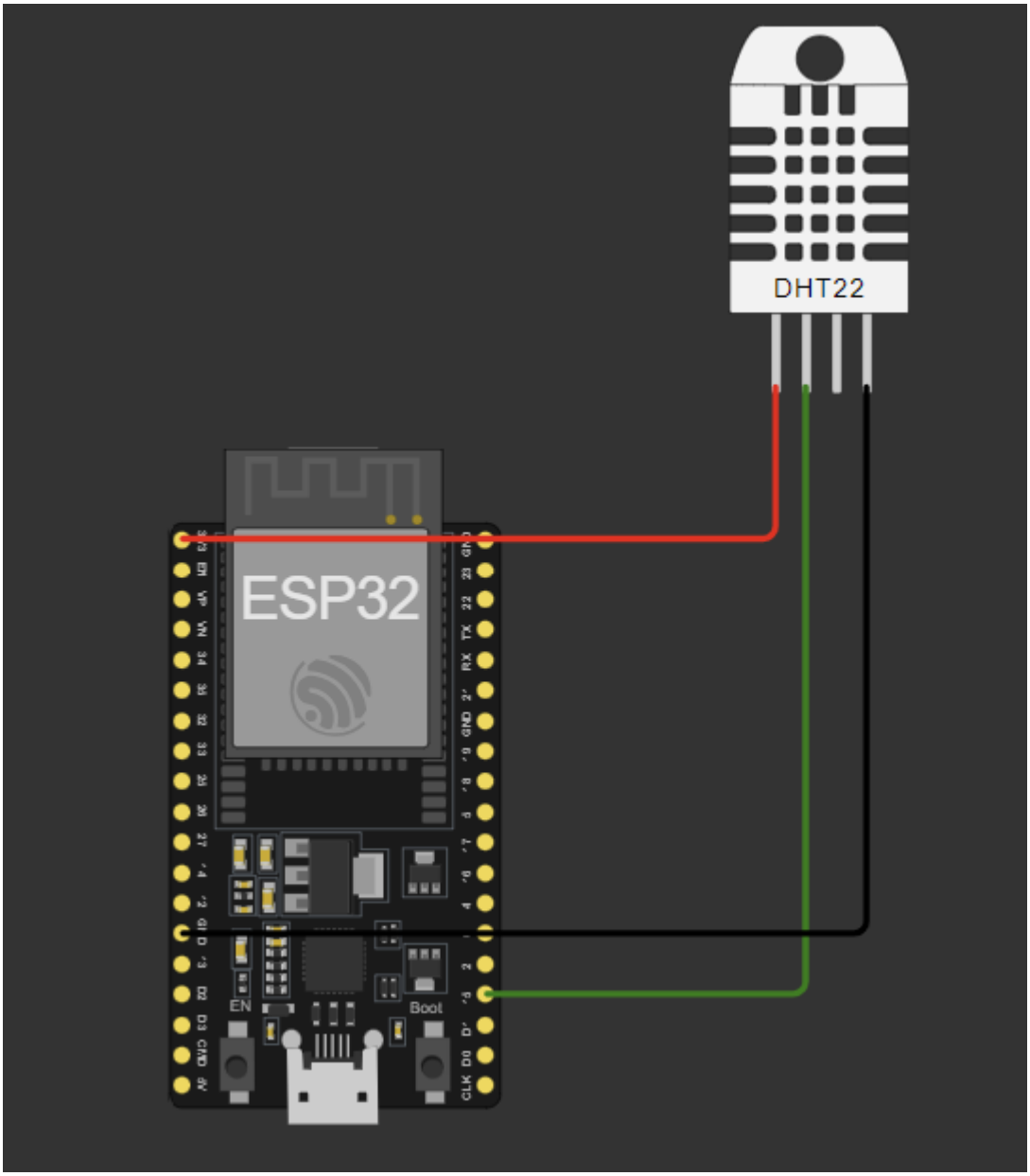


Figura 1.8 Simularea ESP32

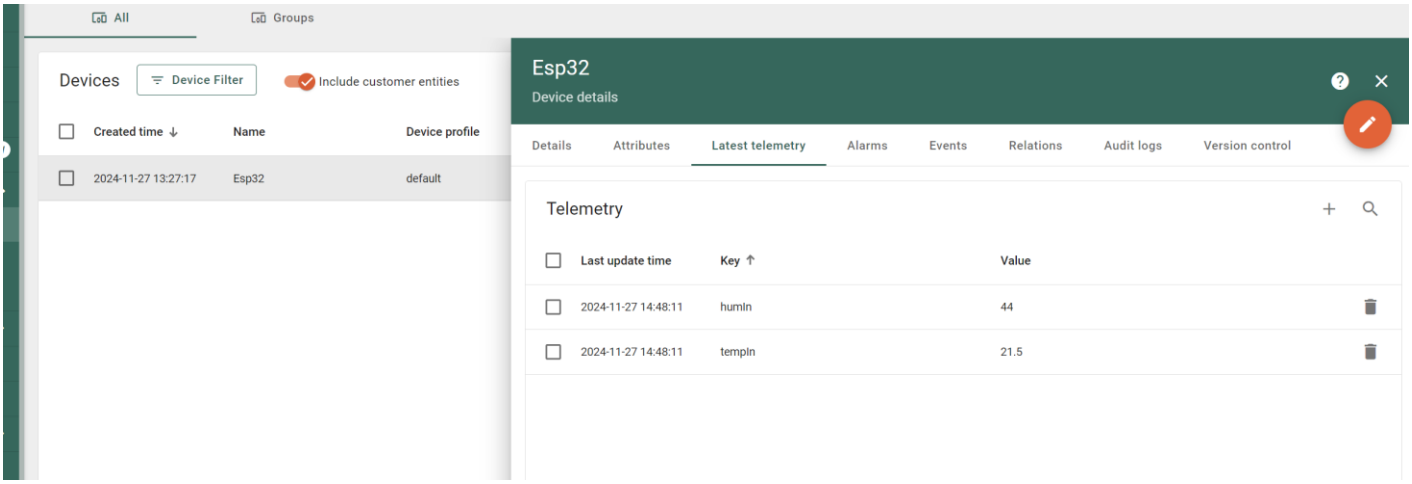
Pentru a 3 sarcină am avut nevoie de trimis datele către un broker MQTT. Mai întâi ne-am conectat la internet prin WiFi, apoi am făacut conexiunea la ThingsBoard prin token de acces. La server au fost trimise date precum umeditatea si temperatura. Pentru a ceasta am avut nevoie de un ESP32 și sensorul de umeditate. Mai jos sun afișate procesul de colectare a datelor și afișarea pe ThingsBoard a datelor transmise.

Din fiugra 1.9, se observă cum mai întâi dispozitivul se conectează la WiFI, apoi după conexiunea cu succes, se conectează la ThingsBooard serever pentru a trimite umeditatea și temperatura curentpă.

```
Mock Temperature: 28.60
Mock Humidity: 63
Connecting to ThingsBoard server
Connected to ThingsBoard
Data sent successfully
Mock Temperature: 21.50
Mock Humidity: 44
Connecting to ThingsBoard server
Connected to ThingsBoard
Data sent successfully
Mock Temperature: 26.10
Mock Humidity: 35
Data sent successfully
Mock Temperature: 21.00
Mock Humidity: 60
Connecting to ThingsBoard server
```

Figura 1.9 Procesul de traansfer a datelor către broker

În figura 1.10., se observă datele primite de la dispozitiv.



Concluzii

Raportul evidențiază realizarea unui sistem integrat de comunicații între microcontrolere și dispozitive IoT, utilizând protocoale precum I²C, serial personalizat și MQTT. Implementarea practică a demonstrat fiabilitatea transferului de date, securitatea comunicațiilor și integrarea cu platforme cloud. Soluția oferă o bază solidă pentru dezvoltarea de aplicații scalabile și conectate, subliniind relevanța acestei tehnologii în automatizări și IoT.

Anexă

7.1.1

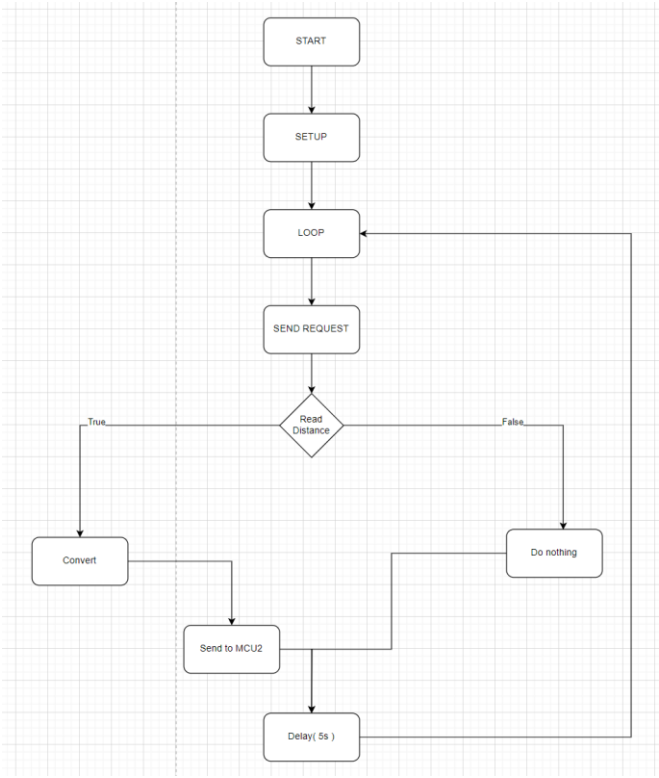
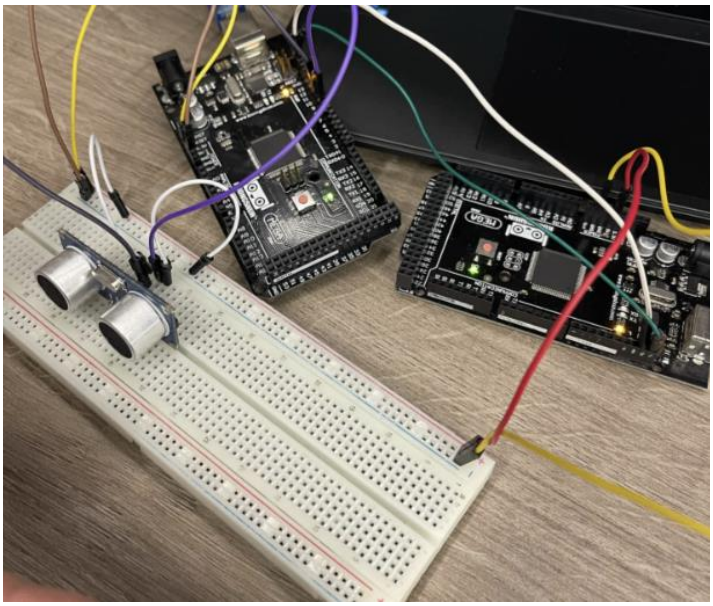
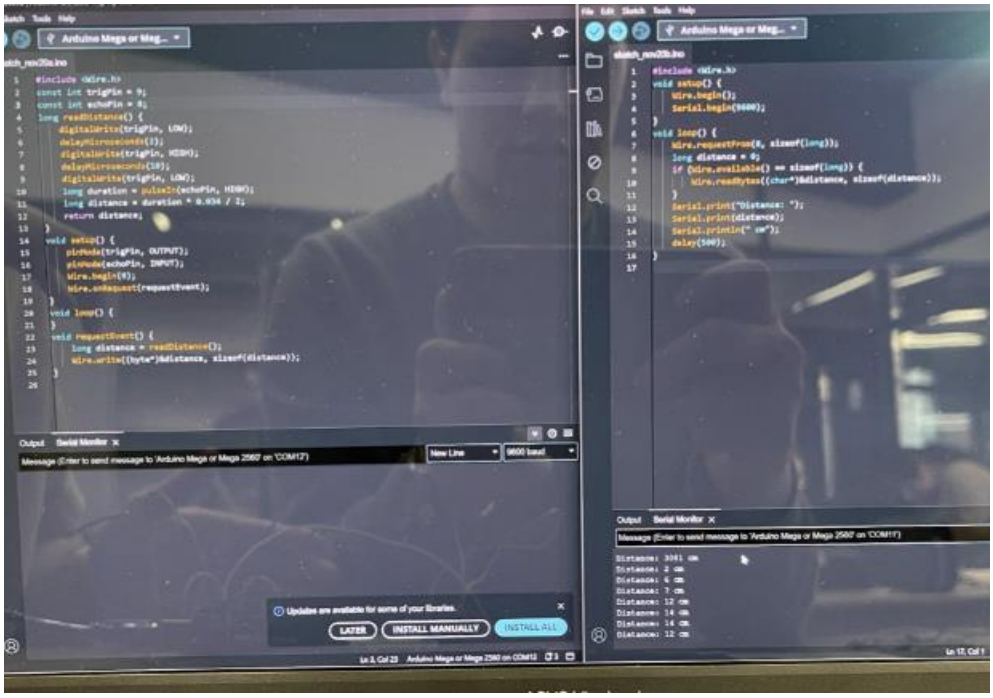


Figura A.1.1 Schema bloc





```
#include <Wire.h>
const int trigPin = 9;
const int echoPin = 8;
long readDistance() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duration = pulseIn(echoPin, HIGH);
    long distance = duration * 0.034 / 2;
    return distance;
}
void setup() {
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    Wire.begin(8);
    Wire.onRequest(requestEvent);
}
```

```
void loop() {
}
void requestEvent() {
    long distance = readDistance();
    Wire.write((byte*)&distance, sizeof(distance));
}
```

7.1.2

```
#include <Wire.h>
void setup() {
    Wire.begin();
    Serial.begin(9600);
}
void loop() {
    Wire.requestFrom(8, sizeof(long));
    long distance = 0;
    if (Wire.available() == sizeof(long)) {
        Wire.readBytes((char*)&distance, sizeof(distance));
    }
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    delay(500);
}
```

7.2.1

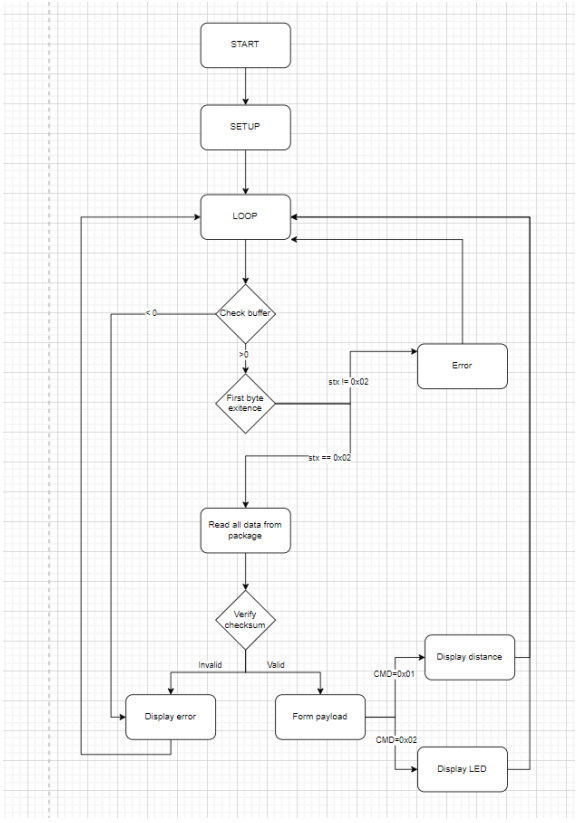
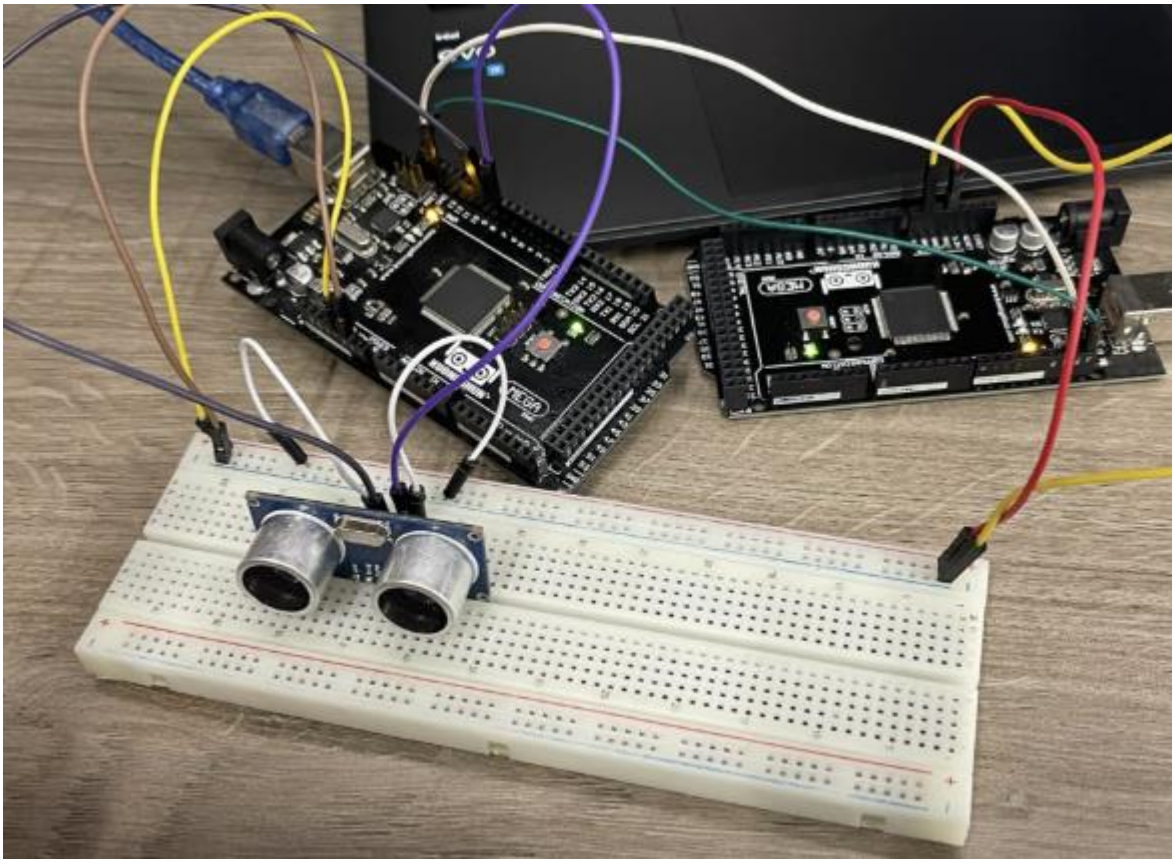


Figura A.1.2 Schema block



```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11);
#define TRIG_PIN 8
#define ECHO_PIN 9
#define LED_CMD 0x02
int packetID = 0;
void setup() {
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.begin(4800);
  mySerial.begin(4800);

  Serial.println("MCU1 - Pregatit pentru a trimite date");
}
long readDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = duration * 0.034 / 2;
  return distance;
}

void sendPacket(byte cmd, int payload) {
  packetID++;
  if (packetID > 255) packetID = 0;

  byte stx = 0x02;
  byte src = 0x01;
```

```
  byte dst = 0x02;
  byte p_id = packetID;
  byte payloadHigh = payload >> 8;
  byte payloadLow = payload & 0xFF;
  byte checksum = (src + dst + p_id + cmd + payloadHigh + payloadLow) % 256;
  byte etx = 0x03;

  Serial.print("Trimit pachet: CMD=0x");
  Serial.print(cmd, HEX);
  Serial.print(" PAYLOAD=");
  Serial.println(payload);

  mySerial.write(stx);
  mySerial.write(src);
  mySerial.write(dst);
  mySerial.write(p_id);
  mySerial.write(cmd);
  mySerial.write(payloadHigh);
  mySerial.write(payloadLow);
  mySerial.write(checksum);
  mySerial.write(etx);
}

void loop() {
  long distance = readDistance();
  sendPacket(0x01, distance);
  delay(5000);

  sendPacket(LED_CMD, 1);
  delay(5000);

  sendPacket(LED_CMD, 0);
```

```
  delay(5000);
}
```

7.2.2

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11);
#define LED_PIN 7
void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(4800);
  mySerial.begin(4800);
  Serial.println("MCU2 - Pregatit de primit datele"); }

void loop() {
  if (mySerial.available() > 0) {
    byte stx = mySerial.read();
    if (stx == 0x02) {
      byte src = mySerial.read();
      byte dst = mySerial.read();
      byte p_id = mySerial.read();
      byte cmd = mySerial.read();
      byte payloadHigh = mySerial.read();
      byte payloadLow = mySerial.read();
      byte checksum = mySerial.read();
      byte etx = mySerial.read();
      byte calculatedChecksum = (src + dst + p_id + cmd + payloadHigh + payloadLow) %
256;
      if (checksum == calculatedChecksum && etx == 0x03) {
        int payload = (payloadHigh << 8) | payloadLow;
        if (cmd == 0x01) {
          Serial.print("Distanta primita: ");
          Serial.print(payload);
          Serial.println(" cm");
```

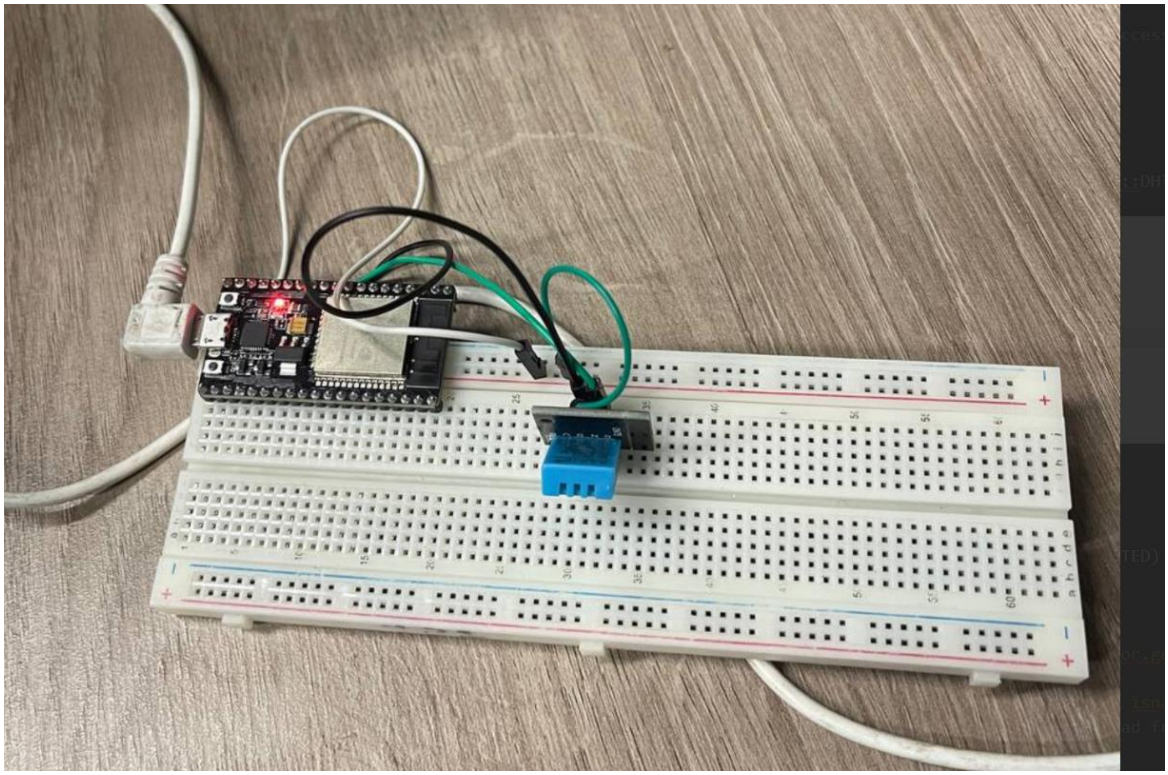
```
        } else if (cmd == 0x02) {
          if (payload == 1) {
            digitalWrite(LED_PIN, HIGH);
            Serial.println("LED aprins");
          } else if (payload == 0) {
            digitalWrite(LED_PIN, LOW);
            Serial.println("LED stins");
          }
        }
      } else {
        Serial.println("Eroare: Checksum invalid sau ETX gresit.");
      }
    } else {
      Serial.println("Eroare: Pachet invalid (fara STX).");
    }
  }
  delay(5000);
}
```

7.3

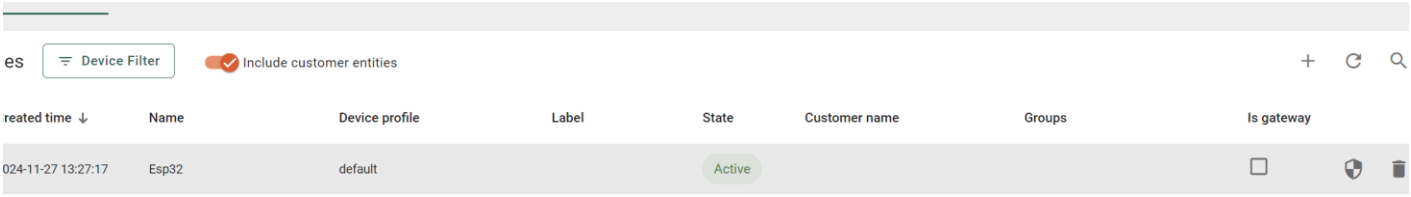
```
#include <DHTesp.h>
#include <WiFi.h>
#include <ThingsBoard.h>
#include <Arduino_MQTT_Client.h>

#define pinDht 27
DHTesp dhtSensor;

#define WIFI_AP "Pixel_3071"
#define WIFI_PASS "helloworld"

#define TB_SERVER "thingsboard.cloud"
#define TOKEN "vnuATv76ZzuRlPSLniVs"

constexpr uint16_t MAX_MESSAGE_SIZE = 256U;

WiFiClient espClient;
```

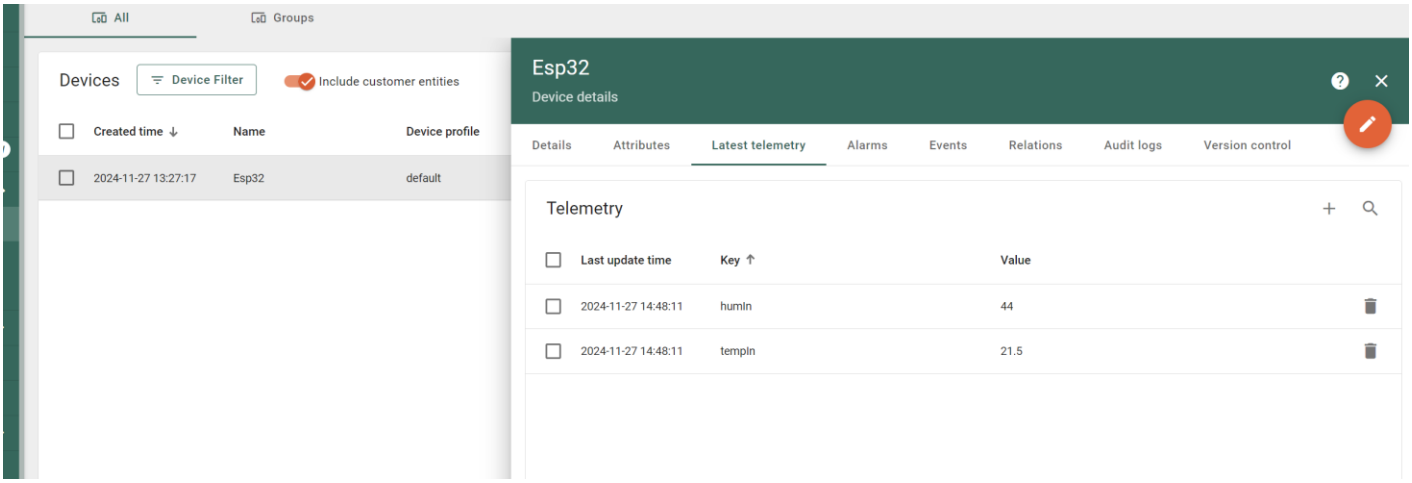
```
Arduino_MQTT_Client mqttClient(espClient);
ThingsBoard tb(mqttClient, MAX_MESSAGE_SIZE);

void connectToWiFi() {
  Serial.println("Connecting to WiFi...");
  WiFi.begin(WIFI_AP, WIFI_PASS);
  int attempts = 0;

  while (WiFi.status() != WL_CONNECTED && attempts < 20) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nFailed to connect to WiFi.");
  } else {
    Serial.println("\nConnected to WiFi");
  }
}

void connectToThingsBoard() {
  if (!tb.connected()) {
    Serial.println("Connecting to ThingsBoard server");

    if (!tb.connect(TB_SERVER, TOKEN)) {
      Serial.println("Failed to connect to ThingsBoard");
    } else {
      Serial.println("Connected to ThingsBoard");
    }
  }
}

void sendDataToThingsBoard(float temp, int hum) {
  StaticJsonDocument<256> jsonDoc;
  jsonDoc["tempIn"] = temp;
  jsonDoc["humIn"] = hum;
  if (!tb.sendTelemetryJson(jsonDoc, measureJson(jsonDoc))) {
    Serial.println("Failed to send telemetry");
  } else {
    Serial.println("Data sent successfully");
  }
}

void setup() {
  Serial.begin(9600);
  dhtSensor.setup(pinDht, DHTesp::DHT22);
```

```
  connectToWiFi();
  connectToThingsBoard();
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectToWiFi();
  }

  TempAndHumidity data = dhtSensor.getTempAndHumidity();

  if (isnan(data.temperature) || isnan(data.humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  float temp = data.temperature;
  float hum = data.humidity;

  Serial.println(temp);
  Serial.println(hum);

  if (!tb.connected()) {
    connectToThingsBoard();
  }

  sendDataToThingsBoard(temp, hum);

  tb.loop();
  delay(3000);
}
```



```
Mock Temperature: 28.60
Mock Humidity: 63
Connecting to ThingsBoard server
Connected to ThingsBoard
Data sent successfully
Mock Temperature: 21.50
Mock Humidity: 44
Connecting to ThingsBoard server
Connected to ThingsBoard
Data sent successfully
Mock Temperature: 26.10
Mock Humidity: 35
Data sent successfully
Mock Temperature: 21.00
Mock Humidity: 60
Connecting to ThingsBoard server
```


es
Device Filter
Include customer entities
reated time
Name
Device profile
Label
State
Customer name
Groups
Is gateway
024-11-27 13:27:17
Esp32
default
Active


All
Groups
Devices
Device Filter
Include customer entities
Created time
Name
Device profile
2024-11-27 13:27:17
Esp32
default
Esp32
Device details
Details
Attributes
Latest telemetry
Alarms
Events
Relations
Audit logs
Version control
Telemetry
Last update time
Key
Value
2024-11-27 14:48:11
humIn
44
2024-11-27 14:48:11
tempIn
21.5