Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

# RAPORT

Lucrare de laborator Nr.7.2
Disciplina: IoT
Tema: Comunicare intre dispozitive - Protocol SW Sierial

A efectuat: st.gr.TI-212,
**Vlasitchi Stefan**

A verificat : asist. univ.
**Lupan Cristian**

Chișinău 2024

**Definirea problemei:**

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

1. 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 |
|---|---|---|---|---|---|---|---|---|

- indicator de start pachet
- indicator de sfarsit
- contorizare pachete
- ID emitator
- ID receptor
- tipul pachetului
- <alte campuri optional>
- date pachet - Payload
- 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 la precedent. Sa si implementezi inca o camanda la alegere, pentru diversitate.

**Materiale necesare:**

- 2x Placă pe bază de Arduino;
- Senzor ultrasonic HCS-04,;
- Cabluri pentru conectare.

# INTRODUCERE

Comunicarea între dispozitive reprezintă un aspect fundamental al sistemelor embedded și al dispozitivelor electronice moderne. Protocolul Software Serial (SW Serial) este o soluție flexibilă și eficientă utilizată pentru a permite microcontrolerelor să comunice între ele sau cu alte periferice printr-un port serial software. Acest protocol este esențial în proiectele în care microcontrolerul are un număr limitat de porturi hardware seriale sau în care acestea sunt deja ocupate de alte funcții critice.

Software Serial funcționează prin emularea unui port UART (Universal Asynchronous Receiver-Transmitter) în software, utilizând pinii digitali ai microcontrolerului pentru transmiterea și recepția datelor. Această abordare permite extinderea funcționalităților de comunicare ale unui microcontroler, fiind ideală pentru aplicații precum transmiterea datelor de la senzori, interacțiunea cu module wireless sau schimbul de mesaje între mai multe microcontrolere. Simplitatea implementării și compatibilitatea cu majoritatea platformelor embedded fac din Software Serial o alegere versatilă pentru o gamă largă de proiecte.

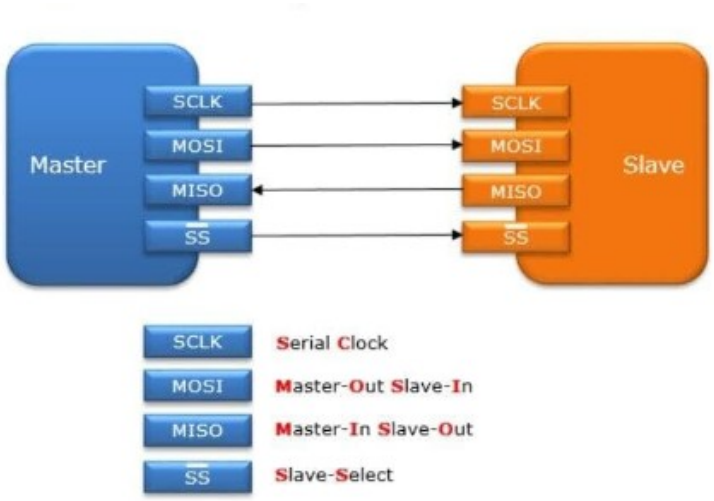


**Figura 1 - Protocolului de comunicație al interfeței periferice seriale.**

Codul prezentat implementează un sistem de măsurare a distanței folosind un senzor ultrasonic HC-SR04, cu datele transmise între două microcontrolere prin protocolul I²C. Sistemul prezent in figura 2 este alcătuit din două microcontrolere: unul configurat ca slave (MCU1), care efectuează măsurarea distanței, și celălalt configurat ca master (MCU2), care solicită și afișează valorile citite.

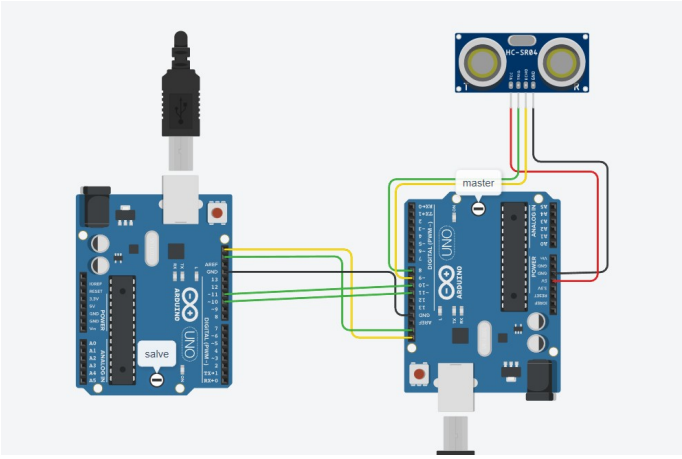


**Figura 2 – Circuitul virtual**

Acest proiect implementează un sistem de comunicare între două microcontrolere utilizând protocolul Software Serial. Microcontrolerul MCU1 acționează ca un dispozitiv de măsurare a distanței, iar MCU2 interpretează și procesează pachetele de date primite de la MCU1. Comunicația se bazează pe un format de pachet bine definit, incluzând câmpuri pentru ID-uri, tipul de pachet, date utile și un checksum pentru validare.

**MCU1: Transmițătorul (Master)**

Acest microcontroler citește datele de la un senzor ultrasonic (HC-SR04) și trimite periodic pachete formatate către MCU2. Creează o instanță Software Serial utilizând pinii digitali 10 și 11 pentru comunicare. Funcția trimite un impuls de 10 µs pe pinul *trigger* și măsoară durata răspunsului ecoului pentru a calcula distanța.

**Codu sura:**

```
SoftwareSerial softSerial1(10, 11); // RX = 10, TX = 11
long getDistance() {
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    long duration = pulseIn(echoPin, HIGH);
    long distance = duration * 0.0344 / 2;  // Calculul distanței în cm
    return distance;
}
String packet = SOF + senderID + "|" + receiverID + "|" + packetType + "|" + payload + "|" +
String(checksum) + BOF;
```

Pachetul este format cu:

- SOF ([): începutul pachetului.
- Sender ID și Receiver ID (01, 02): identificatori unici ai emițătorului și receptorului.
- Packet Type (DATA): indică tipul pachetului.
- Payload: date utile, cum ar fi valoarea măsurată a distanței.
- Checksum: validarea pachetului printr-o sumă modulară.

Pachetul este transmis pe interfața Software Serial și afișat pentru debug pe Monitorul Serial.Acest microcontroler primește pachete de la MCU1, le validează și le procesează. Pachetul este citit caracter cu caracter până la indicatorul de sfârșit (]), apoi se validează completitudinea. Validarea pachetului include verificarea indicatorilor de început și sfârșit ([, ]) și compararea checksum-ului calculat cu cel primit. Decodifică fiecare câmp al pachetului, cum ar fi ID-uri, tipul de pachet și payload. Dacă pachetul conține date de tip DATA, extrage distanța și o afișează.

**Codu sura:**

```
softSerial1.println(packet);
Serial.println("Sent: " + packet);

String receivedPacket = Serial1.readStringUntil(']'); // Citește până la ']'
receivedPacket += "]";
bool isValidPacket(String packet) {
    if (packet[0] != '[' || packet[packet.length() - 1] != ']') return false;

    int checksumIndex = packet.lastIndexOf('|');
    int receivedChecksum = packet.substring(checksumIndex + 1, packet.length() - 1).toInt();

    String packetWithoutChecksum = packet.substring(1, checksumIndex);
    int calculatedChecksum = calculateChecksum("[" + packetWithoutChecksum);

    return receivedChecksum == calculatedChecksum;
}
void parsePacket(String packet) {
    int firstPipe = packet.indexOf('|');
    String senderID = packet.substring(1, firstPipe);
    String receiverID = packet.substring(firstPipe + 1, packet.indexOf('|', firstPipe + 1));
    Serial.println("Decoded Packet: " + packet);
}
if (packetType == "DATA" && payload.indexOf("Distance") >= 0) {
    int distance = payload.substring(payload.indexOf('=') + 1).toInt();
    Serial.println("Distance from sensor: " + String(distance) + " cm");
}
```

```
MCU1 ready!
Sent: [01|02|DATA|Distance=180|12]
Sent: [01|02|DATA|Distance=183|15]
```

**Figura 3 – Rezultatul circuitului virtual(master)**

```
MCU2 ready!
Raw Packet: [01|02|DATA|Distance=180|12]
Valid Packet: [01|02|DATA|Distance=180|12]
Decoded Packet:
  Sender ID: 01
  Receiver ID: 02
  Packet Type: DATA
  Payload: Distance=180
Distance from sensor: 180 cm
Raw Packet:
[01|02|DATA|Distance=183|15]
```

**Figura 4 – Rezultatul circuitului virtual(slave)**

**Fluxul de funcționare al sistemului**

1. **MCU1**:
   - Măsoară distanța utilizând senzorul HC-SR04.
   - Creează un pachet formatat și calculează checksum-ul pentru validare.
   - Trimite pachetul către MCU2.
2. **MCU2**:
   - Primește pachetul și îl verifică folosind indicatorii [, ] și checksum-ul.
   - Decodifică datele pachetului și afișează informațiile primite pe Monitorul Serial.

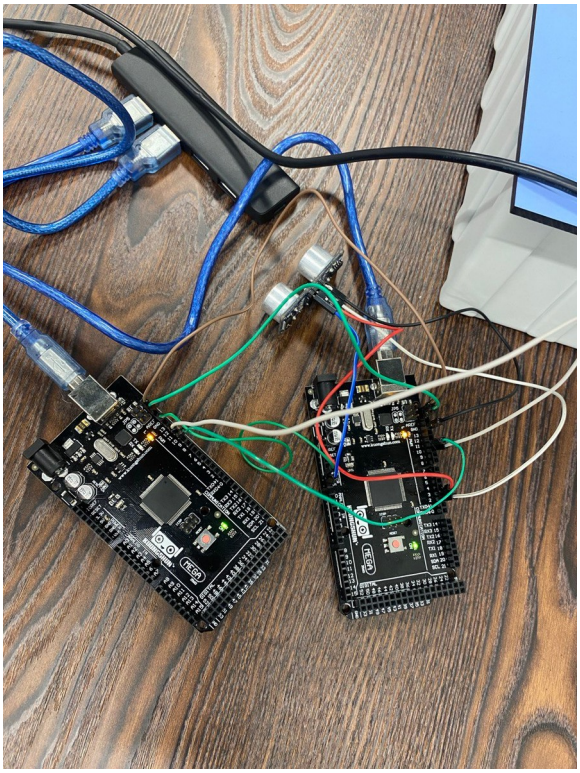

**Figura 4 – Circuitul fizic asamblat**

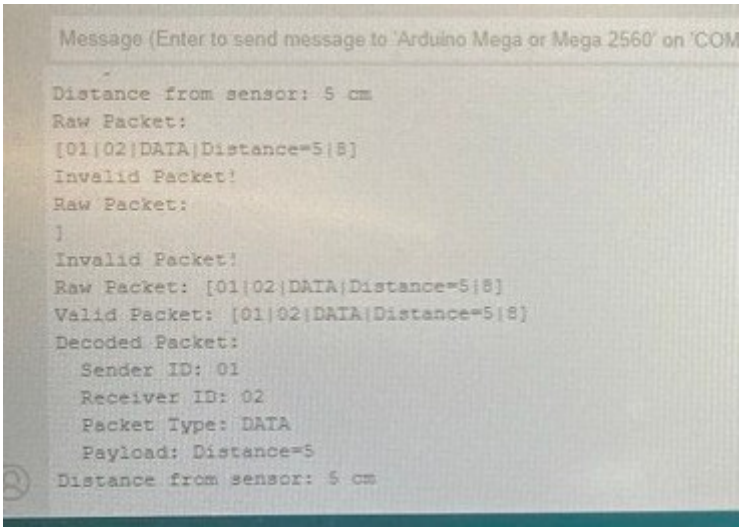


**Figura 5 - Rezultatul circuitului fizic (slave)**

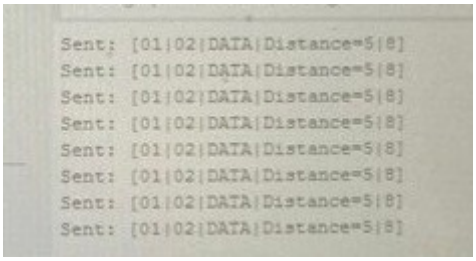


**Figura 6 - Rezultatul circuitului fizic (master)**

## CONCLUZIE

Sistemul implementat demonstrează eficiența și simplitatea utilizării protocolului I²C pentru comunicarea între microcontrolere. Prin separarea funcționalităților între master și slave, proiectul evidențiază modul în care dispozitivele embedded pot colabora pentru a realiza sarcini complexe. MCU1 se concentrează pe măsurarea distanței utilizând senzorul HC-SR04, în timp ce MCU2 gestionează cererile de date și afișarea rezultatelor. Această abordare modulară contribuie la un design clar, ușor de întreținut și scalabil pentru aplicații mai complexe.

Utilizarea protocolului I²C aduce avantaje majore, precum reducerea numărului de conexiuni fizice și extinderea facilă a sistemului prin adăugarea altor dispozitive. Acest exemplu subliniază potențialul comunicării seriale sincronizate în proiectele embedded, fiind o soluție robustă și versatilă pentru aplicații care implică senzori, dispozitive periferice sau alte unități de procesare. Proiectul poate fi extins pentru a integra mai mulți senzori sau funcționalități suplimentare, păstrând structura modulară și claritatea codului.

# Anexa 1

## Fisierul MCU1(Slave).ino

```
#include <SoftwareSerial.h>
#include "stdinout.h"

SoftwareSerial Serial1(10, 11); // RX = 10, TX = 11

void setup() {
  Serial1.begin(300);  // Baud rate pentru comunicarea serială
  printf("MCU2 ready!\n");
}

void loop() {
  if (Serial1.available() > 0) {
    String receivedPacket = Serial1.readStringUntil(']'); // Citeşte până la ']'
    receivedPacket += "]";  // Adaugă EOF pentru validare completă
    printf("Raw Packet: %s\n", receivedPacket.c_str());

    // Verifică dacă pachetul este valid
    if (isValidPacket(receivedPacket)) {
      printf("Valid Packet: %s\n", receivedPacket.c_str());
      parsePacket(receivedPacket); // Decodifică pachetul
    } else {
      printf("Invalid Packet!\n");
    }
  }
}

bool isValidPacket(String packet) {
  if (packet[0] != '[' || packet[packet.length() - 1] != ']') {
    return false;
  }

  int checksumIndex = packet.lastIndexOf('|');
  int receivedChecksum = packet.substring(checksumIndex + 1, packet.length() - 1).toInt();
  String packetWithoutChecksum = packet.substring(1, checksumIndex);
  int calculatedChecksum = calculateChecksum("[" + packetWithoutChecksum);

  return receivedChecksum == calculatedChecksum;
}

void parsePacket(String packet) {
  int firstPipe = packet.indexOf('|');
  int secondPipe = packet.indexOf('|', firstPipe + 1);
  int thirdPipe = packet.indexOf('|', secondPipe + 1);
  int fourthPipe = packet.indexOf('|', thirdPipe + 1);

  String senderID = packet.substring(1, firstPipe);
  String receiverID = packet.substring(firstPipe + 1, secondPipe);
  String packetType = packet.substring(secondPipe + 1, thirdPipe);
  String payload = packet.substring(thirdPipe + 1, fourthPipe);

  printf("Decoded Packet:\n");
  printf("  Sender ID: %s\n", senderID.c_str());
  printf("  Receiver ID: %s\n", receiverID.c_str());
  printf("  Packet Type: %s\n", packetType.c_str());
  printf("  Payload: %s\n", payload.c_str());

  if (packetType == "DATA" && payload.indexOf("Distance") >= 0) {
    int distance = payload.substring(payload.indexOf('=') + 1).toInt();
    printf("Distance from sensor: %d cm\n", distance);
  }
}

int calculateChecksum(String data) {
  int sum = 0;
```

```
  for (int i = 0; i < data.length(); i++) {
    if (isDigit(data[i])) {
      sum += data[i] - '0';
    }
  }
  return sum % 256;
}
```

### Fisierul MCU2(Master).ino

```
#include <SoftwareSerial.h>
#include "stdinout.h"

SoftwareSerial softSerial1(10, 11); // RX = 10, TX = 11

// Define sensor pins (for example, HCSR04)
const int triggerPin = 8;
const int echoPin = 9;

void setup() {
  softSerial1.begin(300);   // Baud rate for serial communication
  printf("MCU1 ready!\n");
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  long distance = getDistance();

  String SOF = "[";
  String BOF = "]";
  int packetID = 1;
  String senderID = "01";
  String receiverID = "02";
  String packetType = "DATA";
  String payload = "Distance=" + String(distance);

  int checksum = calculateChecksum(senderID + "|" + receiverID + "|" + packetType + "|" +
payload);

  String packet = SOF + senderID + "|" + receiverID + "|" + packetType + "|" + payload + "|" +
String(checksum) + BOF;

  softSerial1.println(packet);
  printf("Sent: %s\n", packet.c_str());

  delay(1000);  // Send a packet every second
}

long getDistance() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.0344 / 2;  // Calculate distance in cm
}

int calculateChecksum(String data) {
  int sum = 0;
  for (int i = 0; i < data.length(); i++) {
    if (isDigit(data[i])) {
      sum += data[i] - '0';
    }
  }
  return sum % 256;
}
```

### Fişierul stdinout.h

```
#ifndef _STDINOUT_H
```

```
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
        static size_t initnum;
public:
        // Constructor
        initializeSTDINOUT();
};

// Call the constructor in each compiled file this header is included in
// static means the names won't collide
static initializeSTDINOUT initializeSTDINOUT_obj;

#endif
```

### Fișierul stdinout.cpp

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <stdio.h>
#include "stdinout.h"

// Function that printf and related will use to print
static int serial_putchar(char c, FILE *f)
{
  if (c == '\n') {
    serial_putchar('\r', f);
  }

  return Serial.write(c) == 1 ? 0 : 1;
}
// Function that scanf and related will use to read
static int serial_getchar(FILE *)
{
  // Wait until character is avilable
  while (Serial.available() <= 0) { ; }

  return Serial.read();
}

static FILE serial_stdinout;

static void setup_stdin_stdout()
{
  // Set up stdout and stdin
  fdev_setup_stream(&serial_stdinout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
  stdout = &serial_stdinout;
  stdin  = &serial_stdinout;
  stderr = &serial_stdinout;
}

// Initialize the static variable to 0
size_t initializeSTDINOUT::initnum = 0;

// Constructor that calls the function to set up stdin and stdout
initializeSTDINOUT::initializeSTDINOUT()
{
  if (initnum++ == 0) {
    setup_stdin_stdout();
  }
}
```