Ministerul Educaţiei, 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 între dispozitive - Protocol SW Sierial

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

**Sarcina:**

Să 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 |
|---|---|---|---|---|---|---|---|---|

- indicator de start pachet : STX;
- indicator de sfarsit : ETX;
- contorizare pachete : P nr;
- ID emitator : SRC;
- ID receptor : DST;
- tipul pachetului : P_ID;
- <alte campuri optional> : CMD;
- date pachet : Payload;
- suma de control - suma tuturor valorilor numerice din pachet : SC;

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.

**Recomandare:**

- asa cum se va utiliza interfata seriala, se recomanda invocarea a unei a doua interfete seriale (Serial1 sau SoftwareSerial);
- Reutilizati la maxim solutiile prezentate in laboratoarele precedente
- revizuiti resursele predate la curs

**Punctaj**:

- nota 5 - simpla aplicatie de comunicare,
- +1.0 - pentru implementare modulara a proiectului,
- +1.0 - MCU1 trimite date impachetate conform protocolului catre MCU2,
- +1.0 - MCU2 detecteaza si interpreteaza corect datele de la MCU1 prin interfata seriala,
- +1.0 - pentru demonstrarea probelor de implementare fizica

**NOTA**: punctaj maxim posibil doar la prezentare functionare fizica !!

**Penalitati:**

- -1 - penalizare pentru fiecare saptamana intarziere de la deadline
- -1 - penalizare pentru nerespectare format raport

**Obiective:**

- Studiul procesului de comunicare în baza de interfață SW Serial;
- Implimentarea relației Master-Slave între microcontorlere;
- Prezentarea circuitului fizic.

## INTRODUCERE

Comunicarea între dispozitive reprezintă un aspect esențial al tehnologiilor moderne, permițând schimbul de informații între diverse componente hardware și software. În acest context, utilizarea protocoalelor eficiente și adaptabile este crucială pentru asigurarea unui flux de date fiabil și rapid. Protocolul SW Serial este o soluție utilizată frecvent în scenarii în care comunicarea serială între dispozitive trebuie să fie realizată cu un minim de resurse hardware și software.

Protocolul SW Serial se bazează pe principiul comunicării seriale figura 1, în care informația este transmisă bit cu bit, utilizând o linie de transmisie (TX) și o linie de recepție (RX). Acest protocol este implementat în principal prin software, fără a necesita module hardware dedicate, fiind astfel o soluție economică și flexibilă. În mod particular, SW Serial este utilizat pentru:

a) comunicarea între microcontrolere;

b) interfațarea dispozitivelor periferice, cum ar fi senzori, afișaje și module wireless;

c) dezvoltarea și testarea de prototipuri, unde resursele hardware sunt limitate.


**Figura 1 – Comunicarea serial**

## Materiale necesare și asamblarea circuitului

**Materiale necesare:**

- Microcontrolere: 2 plăci Arduino (ex. Arduino Uno);
- Senzor ultrasonic: HCSR-04;
- Fire de conectare: pentru alimentare și semnal;
- Breadboard: pentru conectarea componentelor. (opțional)

**Asamblarea circuitului: (figura 2)**

1. Microcontroler Slave (MCU1):

- Conectarea senzorului HCSR-04:
    - PIN_TRIG la pinul digital 8 al Arduino;
    - PIN_ECHO la pinul digital 7 al Arduino.
- Configurarea liniei Software Serial.
    - RX (recepție): conectat la un pin digital configurabil - 2;
    - TX (transmisie): conectat la un alt pin digital configurabil - 3.
- Alimentarea senzorului HCSR-04 (Vcc și GND).

2. Microcontroler Master (MCU2):

- Configurarea liniei Software Serial:
    - RX conectat la pinul TX al Slave (pinul 2);
    - TX conectat la pinul RX al Slave (pinul 3).
- Alimentarea microcontrolerelor dintr-o sursă de tensiune comună.

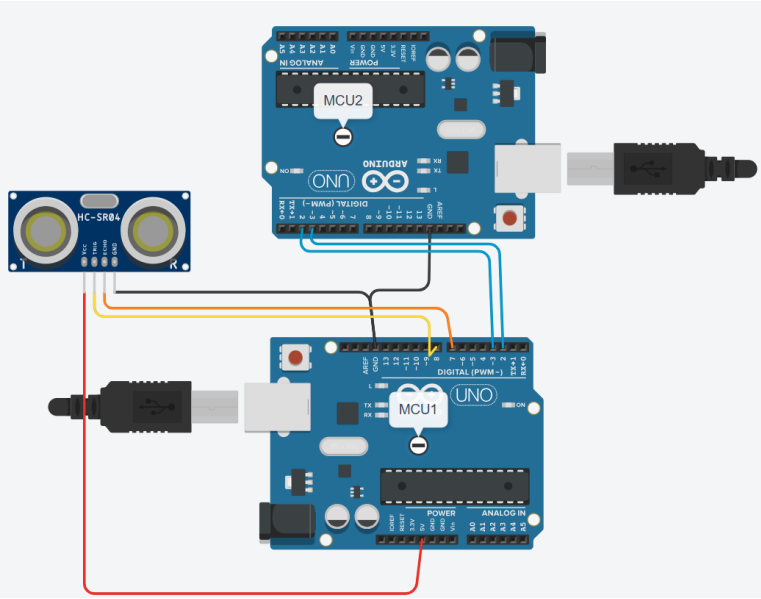



**Figura 2 – Asamblare circuitului virtual**

## Mersul lucrării

### 1 Definirea constantelor pentru pachet

```
#define SOF 0x02      // Start of packet
#define EOF 0x03      // End of packet
#define SENDER_ID 0x01
#define RECEIVER_ID 0x02
#define PACKET_TYPE 0x01 // Packet type (e.g., DATA)
#define TIMEOUT_MS 1000  // Timeout pentru citirea pachetului
```

Aceste constante definesc formatul pachetului:

- SOF (Start of Frame) și EOF (End of Frame) marchează începutul și sfârșitul unui pachet.
- SENDER_ID și RECEIVER_ID identifică emițătorul și receptorul, prevenind confuzia în cazul unei rețele cu mai multe dispozitive.
- PACKET_TYPE permite clasificarea pachetelor, indicând tipul datelor transmise (de exemplu, distanță).

### 2 Inițializarea SoftwareSerial

```
SoftwareSerial softSerial(2, 3); // RX = 2, TX = 3
softSerial.begin(300);
```

Pe fiecare MCU, biblioteca SoftwareSerial este utilizată pentru a configura comunicația serială pe pinii 2 (RX) și 3 (TX). Baud rate-ul este setat la 300, un standard redus, potrivit pentru aplicații simple, unde integritatea datelor este prioritară.

### 3 Construcția unui pachet de date pe MCU1

```
byte* packDistancePacket(int distance) {
  static byte packet[8];
  packet[0] = SOF; // Start of packet (SOF)
  packet[1] = SENDER_ID; // Sender ID
  packet[2] = RECEIVER_ID; // Receiver ID
  packet[3] = PACKET_TYPE; // Packet type (e.g., DATA)
  packet[4] = (byte)((distance >> 8) & 0xFF); // High byte of distance
  packet[5] = (byte)(distance & 0xFF);        // Low byte of distance
  packet[6] = calculateChecksum(packet);      // Checksum
  packet[7] = EOF;                           // End of packet (EOF)
  return packet;
}
```

Structura pachetului este construită secvențial, fiecare componentă fiind plasată într-o poziție specifică. Distanța este împărțită în do byte (high și low) pentru a asigura compatibilitatea cu valori mai mari de 255. Checksum-ul este calculat pentru a verifica integritatea datelor pe partea receptorului.

### 4 Transmiterea unui pachet de date pe MCU1

```
void sendPacket(byte* packet) {
  for (int i = 0; i < 8; i++) {
    softSerial.write(packet[i]);
  }
}
```

Transmiterea datelor se realizează trimițând fiecare byte din pachet în ordine, asigurând respectarea structurii definite.

### 5 Citirea unui pachet pe MCU2

```
int readPacket(byte* packet) {
  int bytesRead = 0;
  unsigned long startTime = millis();
  while (bytesRead < 8) {
    if (softSerial.available() > 0) {
      packet[bytesRead++] = softSerial.read();
    }
    if (millis() - startTime > TIMEOUT_MS) {
      return 0; // Timeout
    }
  }
  return bytesRead;
}
```

Receptorul citește pachetul byte cu byte, verificând dacă datele sunt disponibile. Se implementează un timeout pentru a evita blocarea în cazul în care pachetul nu este complet recepționat.

### 6 Validarea unui pachet pe MCU2

```
bool isValidPacket(byte* packet) {
  return (isCorrectSOF(packet) && isCorrectEOF(packet) && isChecksumValid(packet));
}
```

Funcția integrează verificări critice pentru validitatea pachetului:

- SOF și EOF confirmă delimitarea corectă.
- checksum-ul confirmă că datele nu au fost corupte.

## 7 Calculul checksum-ului

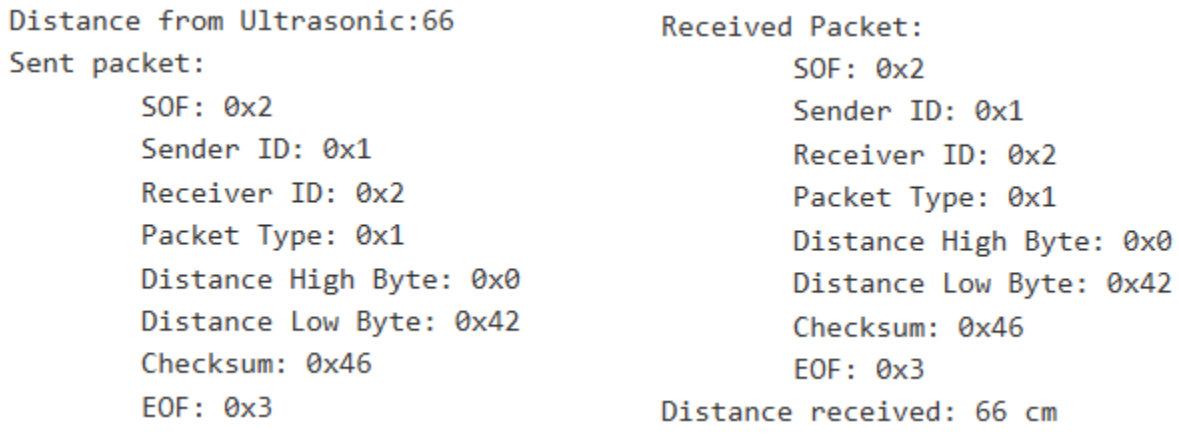
```
int calculateChecksum(byte* packet) {
  int checksum = 0;
  for (int i = 1; i <= 5; i++) {
    checksum += packet[i];
  }
  return checksum & 0xFF; // Păstrăm doar ultimul byte
}
```

Checksum-ul este calculat ca suma componentelor esențiale ale pachetului (fără SOF, EOF). Se ia doar ultimul byte din sumă pentru a păstra compatibilitatea cu dimensiunea unui byte.

## 8 Extracția distanței pe MCU2

```
int extractDistance(byte* packet) {
  return (packet[4] << 8) | packet[5]; // Reconstituim distanța
}
```

Byte-urile care conțin informația despre distanță sunt combinate folosind un shift pentru a reconstrui valoarea inițială




**a) MCU1** **b) MCU2**

**Figura 3 – Rezultatele pe circuitul virtual**

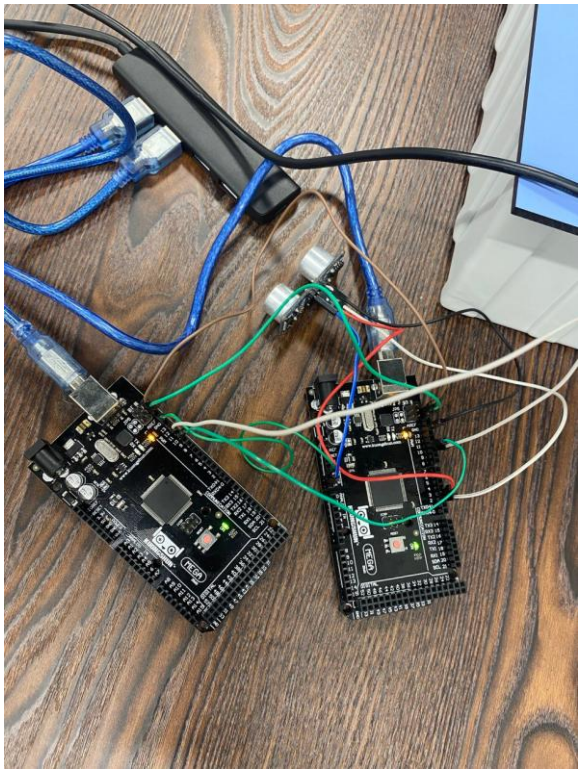


**Figura 4 – Asamblarea circuitului fizic**

# CONCLUZIE

Protocolul SW Serial demonstrează o utilitate semnificativă în implementările unde resursele hardware sunt limitate sau când este necesară o soluție flexibilă pentru comunicarea între dispozitive. Printre beneficiile sale principale se numără:

**Flexibilitate:**

SW Serial permite implementarea unei interfețe de comunicație serială fără necesitatea unor module hardware suplimentare, fiind o soluție economică și ușor de adaptat.

**Simplitate în utilizare:**

Configurarea și utilizarea bibliotecii SoftwareSerial sunt intuitive, ceea ce o face potrivită atât pentru începători, cât și pentru aplicații de prototipare rapidă.

**Compatibilitate extinsă:**

Protocolul poate fi utilizat pentru a integra o gamă largă de dispozitive, de la senzori și afișaje până la alte microcontrolere, sprijinind astfel dezvoltarea de sisteme complexe.

**Robusteză în verificarea datelor:**

Prin utilizarea unui format structurat al pachetelor și a unui mecanism de validare precum checksum-ul, SW Serial asigură integritatea datelor transmise.

În concluzie, protocolul SW Serial reprezintă o soluție esențială pentru comunicațiile seriale în cadrul proiectelor IoT și al altor aplicații embedded, oferind un echilibru între eficiență, costuri și performanță. Implementarea sa corectă contribuie la crearea unor sisteme interconectate fiabile și versatile.

## BIBLIOGRAFII


1. Resursa electronică: https://www.tinkercad.com/things/lRQftS6NLKO-laborator-72/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;
4. Proiectul pe GitHub, Resursa electronică: https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator7 - Regim de acces;

# Anexa 1

## Fișierul 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();
  }
}
```

## Anexa 2

### MCU1.ino

```
#include <SoftwareSerial.h>

// Constante pentru pachet
#define SOF 0x02      // Start of packet
#define EOF 0x03      // End of packet
#define SENDER_ID 0x01
#define RECEIVER_ID 0x02
#define PACKET_TYPE 0x01 // Packet type (e.g., DATA)
#define TIMEOUT_MS 1000  // Timeout pentru citirea pachetului

#define TRIG_PIN 8
#define ECHO_PIN 7

SoftwareSerial softSerial(2, 3); // RX = 2, TX = 3

void setup() {
  Serial.begin(9600);
  softSerial.begin(300);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  Serial.println("Slave initiated!");
}

void loop() {
  int distance = getDistance();
```

```
  byte* packet = packDistancePacket(distance);
  sendPacket(packet);

  delay(1000);
}

int getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

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

byte* packDistancePacket(int distance) {
  static byte packet[8];
  packet[0] = SOF; // Start of packet (SOF)
  packet[1] = SENDER_ID; // Sender ID
  packet[2] = RECEIVER_ID; // Receiver ID
  packet[3] = PACKET_TYPE; // Packet type (e.g., DATA)
  packet[4] = (byte)((distance >> 8) & 0xFF); // High byte of distance
  packet[5] = (byte)(distance & 0xFF);        // Low byte of distance

  packet[6] = calculateChecksum(packet);      // Checksum
  packet[7] = EOF;                           // End of packet (EOF)

  return packet;
}

int calculateChecksum(byte* packet) {
  int checksum = 0;
  for (int i = 1; i <= 5; i++) {
    checksum += packet[i];
  }
  return checksum & 0xFF; // Păstrăm doar ultimul byte
}

void sendPacket(byte* packet) {
  Serial.println("Sent packet:");
  printPacket(packet);
  for (int i = 0; i < 8; i++) {
    softSerial.write(packet[i]);
  }
}

void printPacket(byte* packet) {
  Serial.print("  SOF: 0x");
  Serial.println(packet[0], HEX);
```

```
  Serial.print("  Sender ID: 0x");
  Serial.println(packet[1], HEX);

  Serial.print("  Receiver ID: 0x");
  Serial.println(packet[2], HEX);

  Serial.print("  Packet Type: 0x");
  Serial.println(packet[3], HEX);

  Serial.print("  Distance High Byte: 0x");
  Serial.println(packet[4], HEX);

  Serial.print("  Distance Low Byte: 0x");
  Serial.println(packet[5], HEX);

  Serial.print("  Checksum: 0x");
  Serial.println(packet[6], HEX);

  Serial.print("  EOF: 0x");
  Serial.println(packet[7], HEX);
}
```

**MCU2.ino**

```
#include <SoftwareSerial.h>

// Constante pentru pachet
#define SOF 0x02      // Start of packet
#define EOF 0x03      // End of packet
#define SENDER_ID 0x01
#define RECEIVER_ID 0x02
#define PACKET_TYPE 0x01 // Packet type (e.g., DATA)
#define TIMEOUT_MS 1000  // Timeout pentru citirea pachetului

SoftwareSerial softSerial(2, 3); // RX = 2, TX = 3

void setup() {
  Serial.begin(9600);
  softSerial.begin(300);
  Serial.println("Master initiated!");
}

void loop() {
  if (softSerial.available() > 0) {
    byte packet[8];
    int bytesRead = readPacket(packet);

    if (bytesRead > 0) {
      Serial.println("Received Packet: ");
      printPacket(packet);

      if (isValidPacket(packet)) {
```

```
        int distance = extractDistance(packet);
        Serial.print("Distance received: ");
        Serial.print(distance);
        Serial.println(" cm");
      } else {
        Serial.println("Invalid packet received!");
      }
    } else {
      Serial.println("No valid packet read.");
    }
    Serial.println();
  }
}

int readPacket(byte* packet) {
  int bytesRead = 0;
  unsigned long startTime = millis();

  // Citim pachetul complet
  while (bytesRead < 8) {
    if (softSerial.available() > 0) {
      packet[bytesRead++] = softSerial.read();
    }
    if (millis() - startTime > TIMEOUT_MS) { // Timeout de 1 secundă
      Serial.println("Timeout while reading packet!");
      return 0;
    }
  }
  return bytesRead;
}

bool isValidPacket(byte* packet) {
  return (isCorrectSOF(packet) && isCorrectEOF(packet) && isChecksumValid(packet));
}

bool isCorrectSOF(byte* packet) {
  if (packet[0] != SOF) {
    Serial.println("SOF mismatch!");
    return false;
  }
  return true;
}

bool isCorrectEOF(byte* packet) {
  if (packet[7] != EOF) {
    Serial.println("EOF mismatch!");
    return false;
  }
  return true;
}

bool isChecksumValid(byte* packet) {
```

```
  int checksum = calculateChecksum(packet);
  if (checksum != packet[6]) {
    Serial.print("Checksum mismatch! Calculated: ");
    Serial.print(checksum, HEX);
    Serial.print(" Received: ");
    Serial.println(packet[6], HEX);
    return false;
  }
  return true;
}

int calculateChecksum(byte* packet) {
  int checksum = 0;
  for (int i = 1; i <= 5; i++) {
    checksum += packet[i];
  }
  return checksum & 0xFF; // Păstrăm doar ultimul byte
}

int extractDistance(byte* packet) {
  return (packet[4] << 8) | packet[5]; // Reconstituim distanța
}

void printPacket(byte* packet) {
  Serial.print("  SOF: 0x");
  Serial.println(packet[0], HEX);

  Serial.print("  Sender ID: 0x");
  Serial.println(packet[1], HEX);

  Serial.print("  Receiver ID: 0x");
  Serial.println(packet[2], HEX);

  Serial.print("  Packet Type: 0x");
  Serial.println(packet[3], HEX);

  Serial.print("  Distance High Byte: 0x");
  Serial.println(packet[4], HEX);

  Serial.print("  Distance Low Byte: 0x");
  Serial.println(packet[5], HEX);

  Serial.print("  Checksum: 0x");
  Serial.println(packet[6], HEX);

  Serial.print("  EOF: 0x");
  Serial.println(packet[7], HEX);
}
```