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

RAPORT

Lucrare de laborator Nr.7.1
Disciplina: IoT
Tema: Comunicare cu periferii - I²C

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

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

Sa se realizeze o aplicație ce va implementa comunicațiile intre echipamente după cum urmează:

1. Protocol fizic de comunicare - Comunicarea intre doua Microcontrolere prin interfața I²C

- MCU1 - implementează senzorul digital cu interfața I2C pentru senzorul ultrasonic HCS-04, unde se executa colectarea datelor de la interfața senzorului si se retransmite către interfața I2C la detectarea unei cereri de citire a datelor.
- MCU2 - executa cererea prin interfața I2C către serosul digital ultrasonic (MCU+HCS-04) si afișează datele pe integrata seriala

Obiective:

- Implementare modulara a proiectului
- Mcu1 trimite date împachetare prin i²c,
- Mcu2 decodifica pachete venite si retransmite pe interfața seriala
- Demonstrarea probelor de implementare fizica

Materiale necesare:

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

INTRODUCERE

Comunicarea cu periferii este un aspect esențial al sistemelor embedded, asigurând interacțiunea între microcontrolere (MCU) și diverse dispozitive periferice. Protocolul I²C (Inter-Integrated Circuit) este o metodă standardizată și eficientă utilizată pentru a conecta mai multe componente într-un sistem. Acest protocol serial sincron este caracterizat prin utilizarea a doar două linii de comunicație (SDA și SCL), ceea ce îl face ideal pentru aplicații în care economia de pini și simplitatea implementării sunt prioritare.

În contextul microcontrolerelor, I²C permite interacțiunea cu o gamă largă de periferice precum senzori, memorii EEPROM, convertoare ADC/DAC sau afișaje LCD. Utilizarea acestui protocol oferă o metodologie clară și organizată pentru transferul datelor, facilitând scrierea unui cod robust și ușor de întreținut. Implementarea comunicației I²C poate fi îmbunătățită prin utilizarea automatelor finite, care structurează procesul de control și sincronizare a transferului, asigurând o dezvoltare simplificată și logică a aplicațiilor. Astfel, I²C devine un instrument indispensabil în proiectarea sistemelor moderne de comunicare cu periferii.

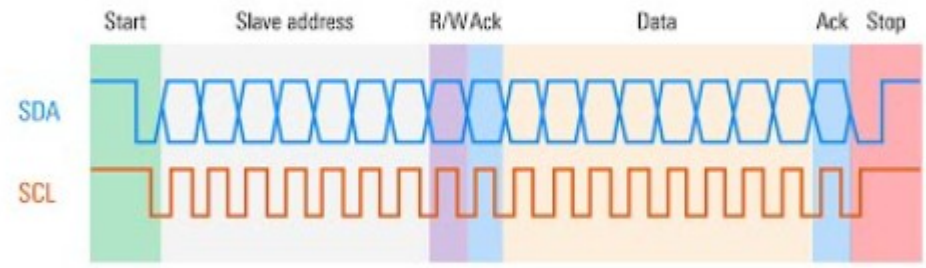

Figura 1 - Prezentare generală a cadrelor I²C

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.

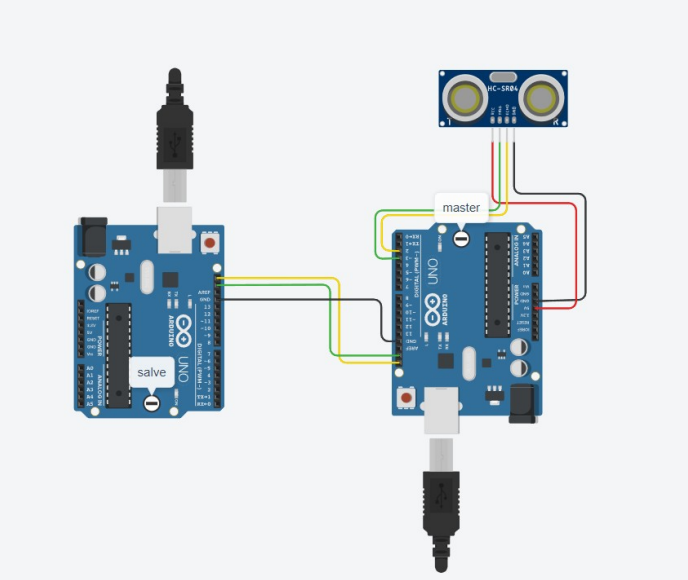

Figura 2 – Circuitul virtual

MCU1 este configurat ca un dispozitiv slave I²C și are rolul de a măsura distanța utilizând senzorul HC-SR04 și de a transmite această valoare către master (MCU2) atunci când primește o cerere. Codul următor configurează pinii senzorului și interfața I²C:

Codul sursa:

```
#include <Wire.h>

// Pini pentru senzorul ultrasonic HC-SR04
const int trigPin = 3;
const int echoPin = 2;

void setup() {
  pinMode(trigPin, OUTPUT); // Configurare pin pentru trimiterea impulsului
  pinMode(echoPin, INPUT);  // Configurare pin pentru recepționarea ecoului
  Wire.begin(8);            // Setare adresa I²C a slave-ului
  Wire.onRequest(sendData); // Setare handler pentru cererea de date de la master
}
```

Funcția de măsurare a distanței trimite un impuls ultrasonic și calculează distanța pe baza timpului de răspuns al ecoului:

Codul sursa:

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

  return pulseIn(echoPin, HIGH) / 58; // Timpul ecoului este convertit în cm
}
```

Când masterul I²C solicită date, slave-ul trimite valoarea distanței împărțită în doi octeți (superior și inferior):

Codul sursa:

```
void sendData() {
  long distance = readUltrasonicDistance();
  byte highByte = (distance >> 8) & 0xFF; // Byte superior
  byte lowByte = distance & 0xFF;         // Byte inferior
  Wire.write(highByte);
  Wire.write(lowByte);
}
```

MCU2 funcționează ca master I²C. Acesta solicită periodic date de la slave și afișează valoarea distanței măsurate pe monitorul serial. Configurarea interfeței I²C și a monitorului serial este prezentată mai jos:

Codul sursa:

```
#include <Wire.h>

void setup() {
  Serial.begin(9600); // Inițializare monitor serial
  Wire.begin();       // Configurare MCU2 ca master I²C
}
```

Masterul trimite cereri de date către slave la fiecare 500 ms, citește cei doi octeți și reconstruiește valoarea distanței:

Codul sursa

```
void loop() {
  Wire.requestFrom(8, 2); // Solicită 2 octeți de la slave-ul cu adresa 8
  if (Wire.available() == 2) { // Verifică dacă există 2 octeți disponibili
    byte highByte = Wire.read();  // Citește byte-ul superior
    byte lowByte = Wire.read();   // Citește byte-ul inferior
    int distance = (highByte << 8) | lowByte; // Reconstruiește valoarea
    Serial.print("Distanta masurata: ");
    Serial.print(distance);
    Serial.println(" cm");
  }
  delay(500); // Așteaptă înainte de următoarea cerere
}
```

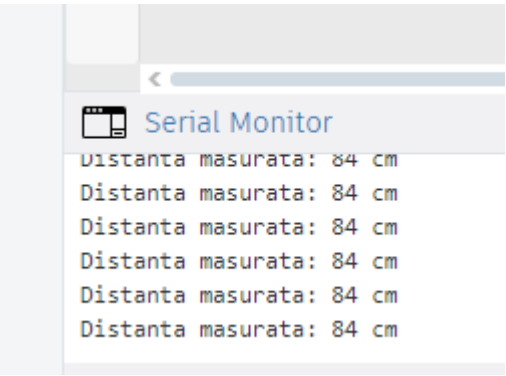

Figura 3 – Rezultatul circuitului virtual

Fluxul de funcționare al sistemului

1. Configurarea și inițializarea dispozitivelor:

- MCU1 setează pinii senzorului și se pregătește să răspundă cererilor I²C.
- MCU2 inițializează comunicarea I²C ca master și configurează monitorul serial.

2. Măsurarea distanței de către MCU1:
   - MCU1 calculează distanța utilizând senzorul HC-SR04 atunci când masterul solicită date.
3. Solicitarea și transmiterea datelor:
   - MCU2 trimite o cerere către adresa I²C a MCU1.
   - MCU1 răspunde cu valoarea distanței împărțită în doi octeți.
4. Reconstruirea și afișarea valorii:
   - MCU2 primește cei doi octeți, reconstruiește valoarea distanței și o afișează pe monitorul serial.

Această interacțiune asigură o separare clară a funcționalităților între cele două microcontrolere, simplificând designul și reducând complexitatea codului.

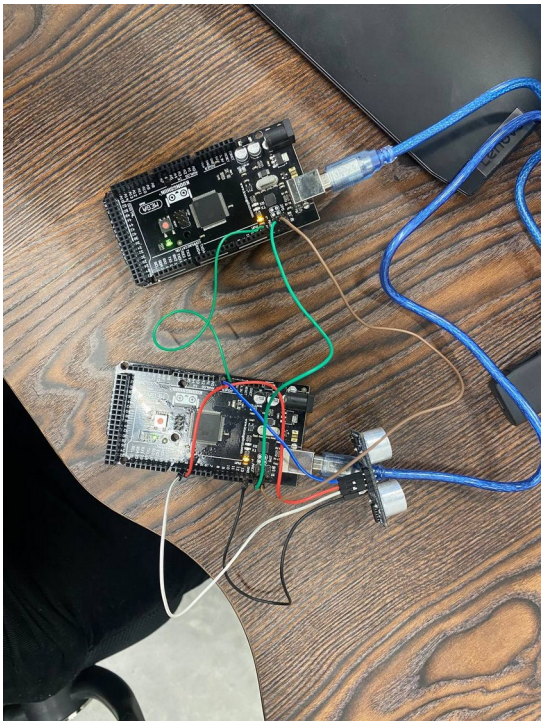

Figura 4 – Circuitul fizic asamblat

```
Distanta masurata: 6 cm
Distanta masurata: 4108 cm
Distanta masurata: 4059 cm
Distanta masurata: 1632 cm
Distanta masurata: 3677 cm
Distanta masurata: 4108 cm
Distanta masurata: 747 cm
Distanta masurata: 9 cm
Distanta masurata: 9 cm
```

Figura 5 – Rezultatul circuitul fizic asamblat

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 <Wire.h>
#include "stdinout.h" // Include the custom stdinout library

void setup() {
  Serial.begin(9600);
  Wire.begin(); // Configure MCU1 as master I²C
}

void loop() {
  Wire.requestFrom(8, 2); // Request 2 bytes from MCU2 (address 8)
  if (Wire.available() == 2) {
    byte highByte = Wire.read();
    byte lowByte = Wire.read();
    int distance = (highByte << 8) | lowByte; // Reconstruct the value
    printf("Distanta masurata: %d cm\n", distance); // Use printf for output
  }
  delay(500); // Wait a bit between reads
}
```

Fisierul MCU2(Master).ino

```
#include <Wire.h>
#include "stdinout.h" // Include the custom stdinout library

// Pins for HC-SR04
const int trigPin = 3;
const int echoPin = 2;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Wire.begin(8); // Set MCU2 I²C address
  Wire.onRequest(sendData); // Set handler for data request
}

void loop() {
  // No operation here; MCU2 only sends data on request
}

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

  long distance = pulseIn(echoPin, HIGH) / 58; // Convert to cm
  printf("Distanta calculata: %ld cm\n", distance); // Use printf for debugging
  return distance;
}

void sendData() {
  long distance = readUltrasonicDistance();
  byte highByte = (distance >> 8) & 0xFF;
  byte lowByte = distance & 0xFF;
  Wire.write(highByte); // Send high byte
  Wire.write(lowByte);  // Send low byte
}
```

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();
  }
}
```