Ministerul Educaţiei, 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,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Sarcina:

Să se realizeze o aplicație ce va implementa comunicatiile intre echipamente dupa cum urmeaza:

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

- 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.
- MCU2 - executa cererea prin interfata I2C catre sesorul digital ultrasonic (MCU+HCS-04) si afiseaza datele pe interata seriala

Recomandare:

- asa cum se va utiliza interfata seriala, se implementarea pentru interfata seriala va fi un textuala
- Reutilizati la maxim solutiile prezentate in laboratoarele precedente
- revizuiti resursele predate la curs

Punctaj:

- nota 5 - simpla aplicatie de comunicare I²C
- +1.0 - pentru implementare modulara a proiectului
- +1.0 - MCU1 trimite date impachetare prin I²C,
- +1.0 - MCU2 decodifica pachete venite si retransmite pe 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 I²C;
- Implimentarea relației Master-Slave între microcontorlere;
- Prezentarea circuitului fizic.

INTRODUCERE

Comunicarea între microcontrolere prin intermediul protocolului I²C reprezintă un pas esențial în dezvoltarea sistemelor integrate moderne. În această lucrare, se implementează interfațarea între două microcontrolere, unul configurat ca Master și celălalt ca Slave, pentru transferul de date despre distanță măsurată de un senzor ultrasonic. 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.

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:

1. Microcontroler Slave (MCU1):

- Conectarea senzorului HCSR-04:
    - PIN_TRIG la pinul digital 8 al Arduino;
    - PIN_ECHO la pinul digital 7 al Arduino.
- Conectarea liniilor I²C (SDA și SCL) la plăcile Master și Slave.
    - SDA: Pin A4 la Slave și Master;
    - SCL: Pin A5 la Slave și Master.
- Alimentarea senzorului HCSR-04 (Vcc și GND).

2. Microcontroler Master (MCU2):

- Conectarea liniilor SDA și SCL comune cu Slave.
- Alimentarea microcontrolerelor dintr-o sursă de tensiune comună.

Mersul lucrării

Configurarea Slave-ului:

Codul pentru microcontrolerul Slave implementează funcționalitatea de a citi datele de la senzorul ultrasonic și de a le transmite către Master prin protocolul I²C. Acesta se configurează cu o adresă specifică (**0x08**) și răspunde la cereri prin transmiterea distanței măsurate.

```
void setup()
{
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  Wire.begin(slaveAddress);  // 0x08 Inițializează microcontrolerul ca Slave I²C
  Wire.onRequest(sendData); // Specifică funcția care va fi apelată atunci când Master-ul trimite o cerere.
}

void sendData() {
  int distance = readDistance(); // Citește distanța de la senzor.
  byte highByte = (distance >> 8) & 0xFF; // Extrage byte-ul superior.
  byte lowByte = distance & 0xFF;         // Extrage byte-ul inferior.
  Wire.write(highByte); // Trimite byte-ul superior către Master.
  Wire.write(lowByte);  // Trimite byte-ul inferior către Master.
}

int readDistance() {
  digitalWrite(PIN_TRIG, HIGH); // Trimite impuls de 10 µs pentru activare.
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  return pulseIn(PIN_ECHO, HIGH) / 58; // Calculează distanța în cm.
}
```

Activează senzorul HCSR-04 și măsoară timpul (în microsecunde) în care semnalul reflectat este recepționat. Împărțirea la 58 convertește timpul măsurat în distanță exprimată în centimetri.

Configurarea Master-ului:

Codul pentru microcontrolerul Master inițiază cererea de date către Slave și procesează informația primită. Distanța măsurată este afișată pe interfața serială.

```
void setup() {
  Wire.begin();        // Inițializează comunicația ca Master.
  Serial.begin(9600);  // Inițializează interfața serială pentru afișare.
}

void loop() {
  Wire.requestFrom(slaveAddress, 2); // Solicită 2 octeți de la Slave.
  if (Wire.available() == 2) {
    byte highByte = Wire.read(); // Citește byte-ul superior.
```

```
    byte lowByte = Wire.read();  // Citește byte-ul inferior.
    int distance = (highByte << 8) | lowByte; // Reconstruiește distanța.
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
  }
  delay(500); // Așteaptă 500 ms înainte de următoarea cerere.
}
```

Rezumat al fluxului:

1. Master-ul trimite o cerere Slave-ului pentru date.
2. Slave-ul măsoară distanța cu senzorul HCSR-04 și transmite valoarea în doi octeți.
3. Master-ul citește datele, reconstruiește distanța și o afișează pe interfața serială.

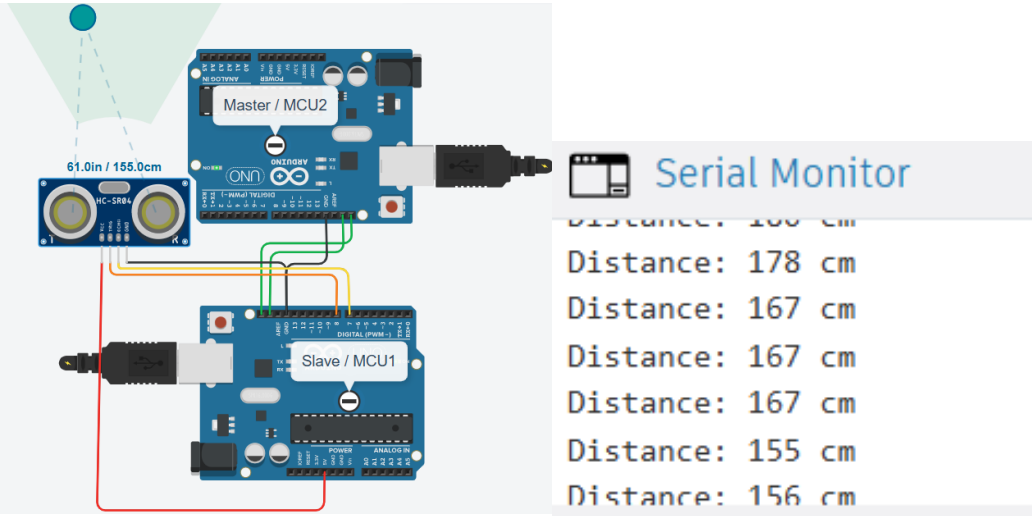

Figura 1 – Rezultatele circuitului asamblat virtual

Figura 2 – Asamblare circuitului fizic

CONCLUZIE

Implementarea comunicării între microcontrolere utilizând protocolul I²C a demonstrat eficiența și versatilitatea acestei interfețe în proiectele IoT. Datorită utilizării a doar două linii de comunicație (SDA și SCL), protocolul I²C permite interconectarea mai multor dispozitive pe același bus, păstrând simplitatea circuitului și reducând numărul de pini necesari.

Lucrarea a evidențiat următoarele aspecte importante:

Simplicitatea implementării

Configurarea rolurilor Master-Slave și utilizarea funcțiilor dedicate precum Wire.begin() și Wire.onRequest() fac ca procesul de inițializare și utilizare să fie intuitiv.

Flexibilitatea

Protocolul permite transferul eficient de date, indiferent de tipul sau volumul acestora, fiind potrivit pentru aplicații diverse, cum ar fi senzori, memorii sau afișaje.

Fiabilitatea

Transferul sincronizat de date și mecanismele integrate de adresare contribuie la o comunicare robustă, cu o probabilitate redusă de erori.

În cadrul experimentului, s-a demonstrat cum datele furnizate de un senzor ultrasonic pot fi preluate de un microcontroler Slave, împachetate și transmise către Master, care apoi le afișează pe interfața serială. Acest flux de lucru subliniază importanța interfeței I²C în dezvoltarea de sisteme integrate eficiente și scalabile.

În concluzie, protocolul I²C reprezintă o soluție ideală pentru proiectele ce implică multiple componente interconectate, oferind un echilibru între performanță, complexitate și costuri.

BIBLIOGRAFII

1. Resursa electronică: https://www.tinkercad.com/things/lwftwvbRGCc-laborator-71/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

Slave.ino

```
#include <Wire.h>
#define PIN_TRIG 8
#define PIN_ECHO 7
#define slaveAddress 0x08

int readDistance();
void sendData();

void setup()
{
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  Wire.begin(slaveAddress);
  Wire.onRequest(sendData);
}

void loop(){}

void sendData(){
  int distance = readDistance();
  byte highByte = (distance >> 8) & 0xFF;
  byte lowByte = distance & 0xFF;
  Wire.write(highByte);
  Wire.write(lowByte);
}

int readDistance(){
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
```

```
  return pulseIn(PIN_ECHO, HIGH)/58;
}
```

Master.ino

```
#include <Wire.h>
#define slaveAddress 0x08

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

void loop() {
  Wire.requestFrom(slaveAddress, 2);
  if (Wire.available() == 2) {
    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("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
  }
  delay(500);
}
```