Universitatea Politehnica Timișoara
Facultatea de Automatică și Calculatoare

Proiect la disciplina Comunicații de Date

Anul universitar 2021/2022

Student: Sfrangeu Bogdan-Otniel

Grupa: 5.2

Contents

Scopul Proiectului

Scopul proiectului: împreuna cu experimentele efectuate la laborator, proiectul urmărește fundamentarea cunoștințelor teoretice si dezvoltarea unor abilități de utilizare practica a acestora. Realizarea experimentelor, explicarea, interpretarea si prezentarea rezultatelor practice obținute - prin prisma cunoștințelor teoretice – dovedesc indubitabil calitățile inginerești…

Sunt abordate si prezentate următoarele aspecte:

1- Efectele mediului asupra semnalelor

2- Magistrala I2C

3- Magistrala SPI

4- Transmisia serial si dispozitive UART/USART

5- Unelte pentru comunicații CAN

6- Concluzii

La Concluzii fiecare să scrie aspecte pozitive și lipsuri care ar trebui remediate ca și conținut, sau ca si desfășurare, în viziunea sa.

1. Efectul mediului asupra semnalelor

1.1. Cerințe

a) Cu ajutorul mediului de simulare Proteus realizați o schemă a unui bloc care să permită simularea unui filtru trece-sus.
b) Reluați pașii de la punctul a) pentru un filtru oprește-bandă.
c) Prezentați comparativ cele două filtre.

1.2. Prezentarea soluției subpunct a)

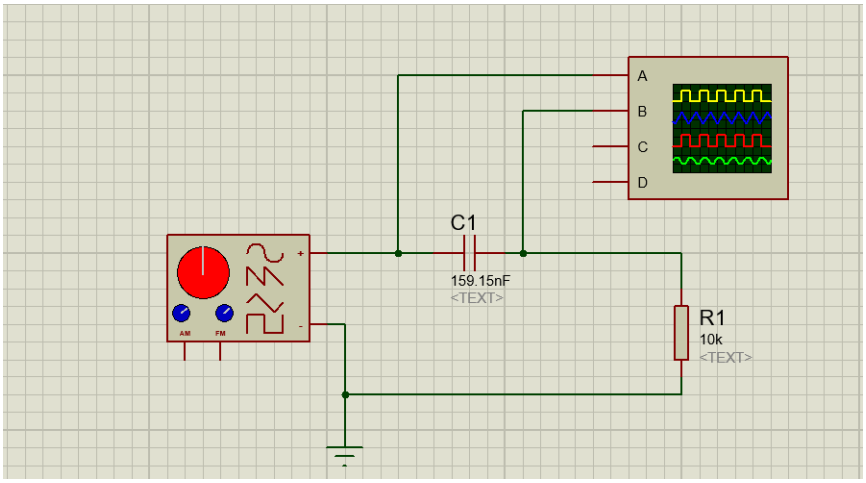


Fig. 1 Captură ecran schema realizată

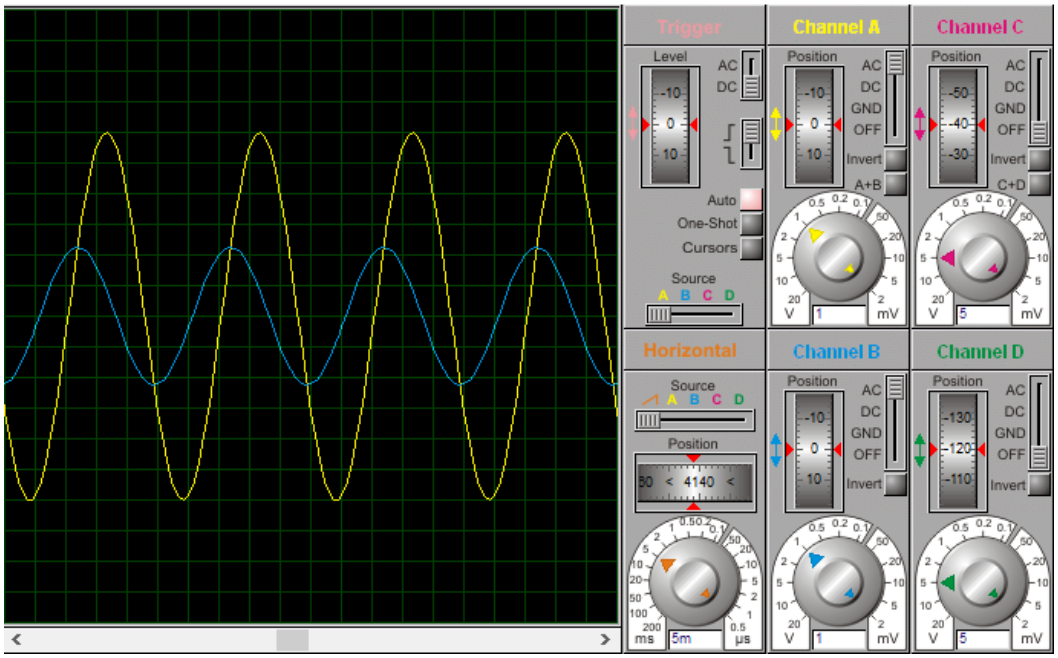


Fig. 2 Captură ecran osciloscop virtual semnal sub frecventa de tăiere [40Hz]

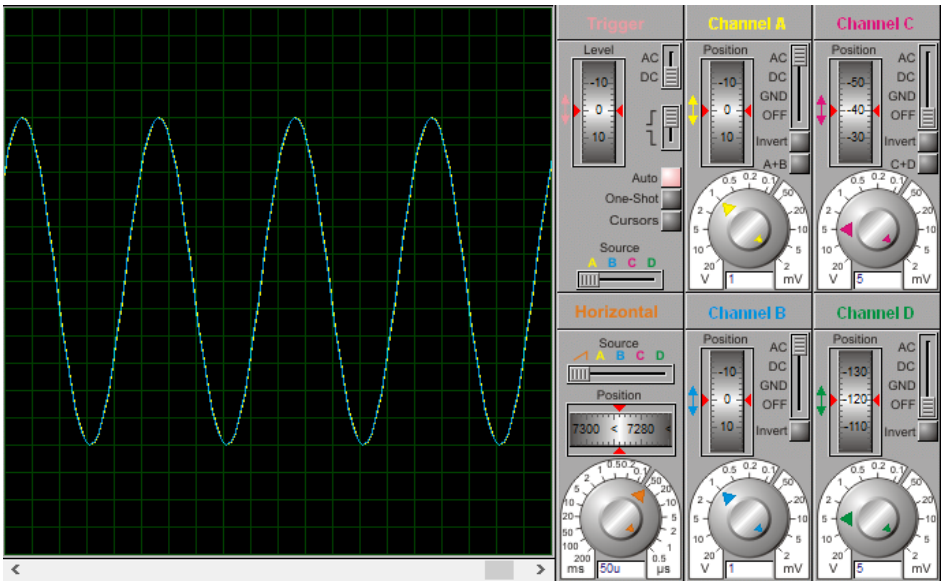


Fig. 3 Captură ecran osciloscop virtual semnal peste frecventa de tăiere [4KHz]

1.3. Descriere rezolvare (detalii de implementare, fenomenele observate) subpunct a)

Filtrul trece-sus a fost implementat in varianta RC, cu rezistenta R si condensator C in serie. Pentru implementarea filtrului am ales o frecventa de taiere de ft = 100 Hz, iar pentru aceasta am initializat rezistorul cu rezistenta R = 10 KOhm si condensatorul cu capacitatea C = 159.15 nF. Folosind formula $f_t = \frac{1}{2\pi RC}$ am obtinut frecventa de taiere mentionata mai sus ft = 100 Hz.

Din ce am observat, pentru filtrul trece-sus, orice semnal de intrare cu o frecventa mai mica decat frecventa de taiere va fi atenuat, iar semnalele de intrare cu frecventa mai mare sau egala cu frecventa de taiere va fi lasat sa treaca in sensul ca atenuarea semnalului este nesimnificativa.

1.4. Prezentarea soluției subpunct b)

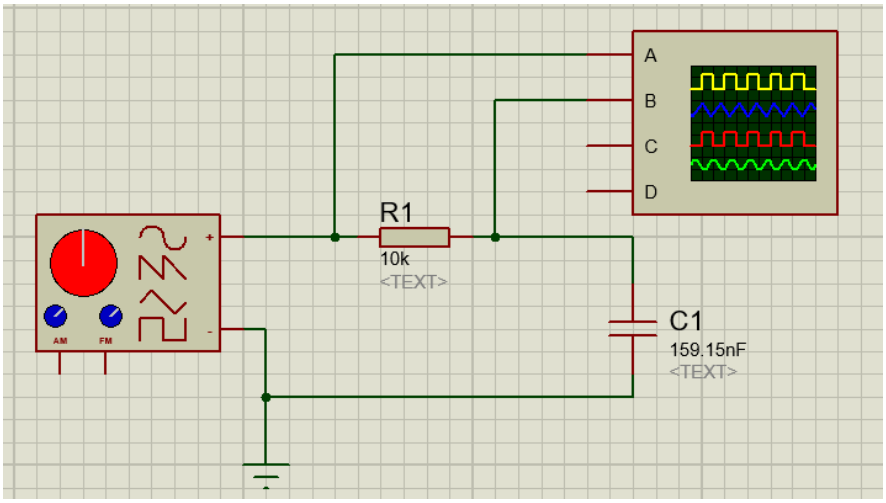


Fig. 4 Captură ecran schema realizată

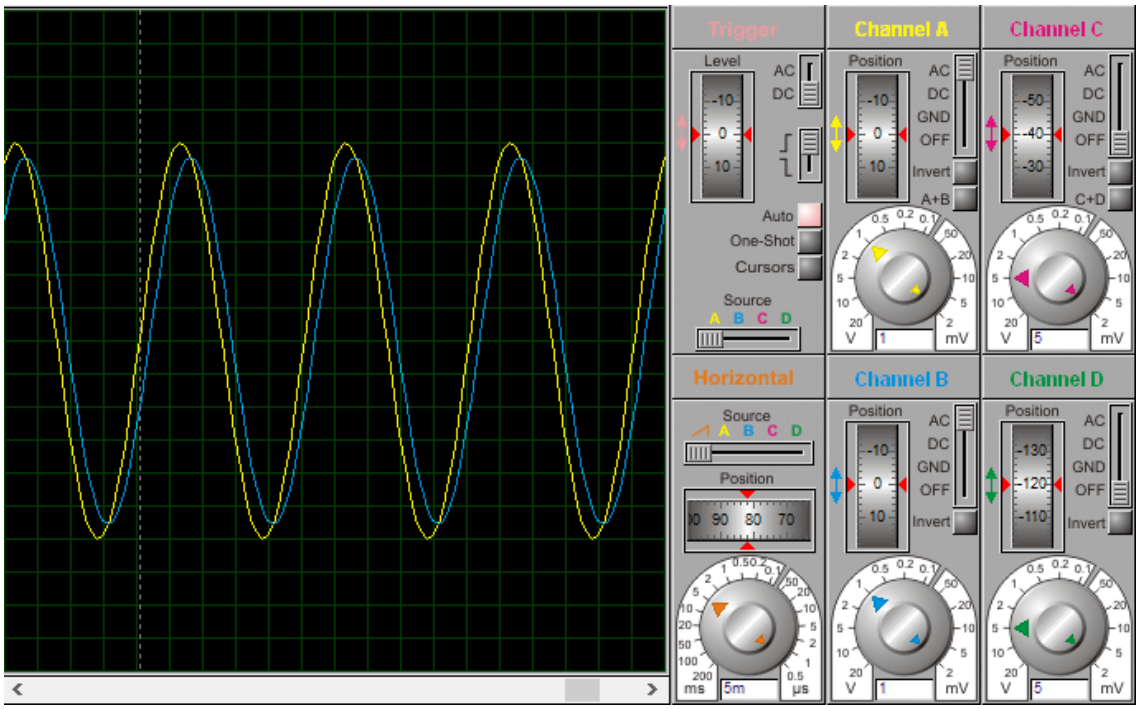


Fig. 5 Captură ecran osciloscop virtual semnal sub frecventa de tăiere

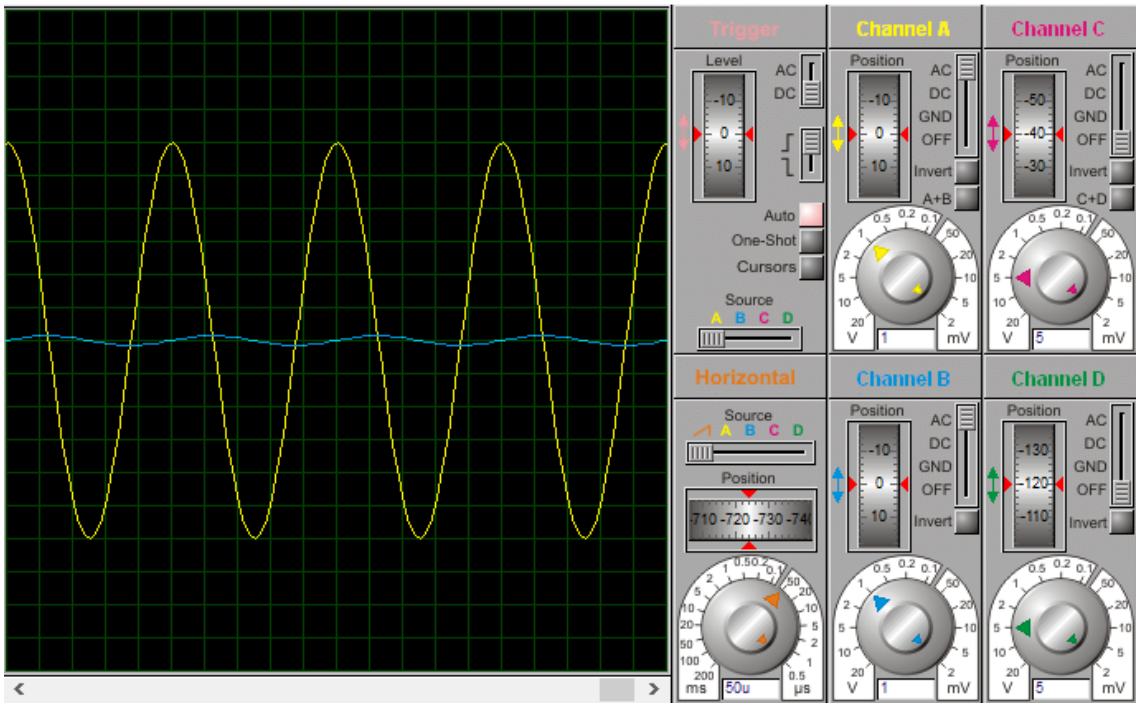


Fig. 6 Captură ecran osciloscop virtual semnal peste frecventa de tăiere

1.5. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica) subpunct b)

Filtrul trece-jos a fost implementat in varianta RC, cu rezistenta R si condensator C in serie. Pentru implementarea filtrului am ales o frecventa de taiere de ft = 100 Hz, iar pentru aceasta am initializat rezistorul cu rezistenta R = 10 KOhm si condensatorul cu capacitatea C = 159.15 nF. Folosind formula $f_t = \frac{1}{2\pi RC}$ am obtinut frecventa de taiere mentionata mai sus ft = 100 Hz.

Din ce am observat, pentru filtrul trece-jos, orice semnal de intrare cu o frecventa mai mare decat frecventa de taiere va fi atenuat, iar semnalele de intrare cu frecventa mai mica sau egala cu frecventa de taiere va fi lasat sa treaca in sensul ca atenuarea semnalului este nesimnificativa.

1.6. Prezentarea soluției subpunct c)

Cele doua filtre au un comportament opus, filtrul trece-sus lasand sa treaca semnalele sinusoidale de frecvente mai mari decat frecventa de taiere si atenuand puternic semnalele sinusoidale de frecvente mai mici decat frecventa de taiere, iar filtrul trece-jos lasand sa treaca semnalele sinusoidale de frecvente mai mici decat frecventa de taiere si atenuand puternic semnalele sinusoidale de frecvente mai mari decat frecventa de taiere.

Ambele filtre sunt de tip pasiv, folosind doar componente de tip rezistor sau condensator in implementarea acestora.

2. Transmisia serială. Dispozitive UART/USART

2.1. Cerințe

Să se realizeze în mediul Proteus comunicația între 2 microcontrolere ATmega2560 utilizând UART. Primul dispozitiv citește propoziții NMEA de la un modul GPS iar mai apoi le trimite prin UART către al doilea dispozitiv. După recepționarea propozițiilor, al doilea dispozitiv, extrage latitudinea urmând ca aceasta să fie afișată pe un terminal virtual.

2.2. Prezentarea soluției

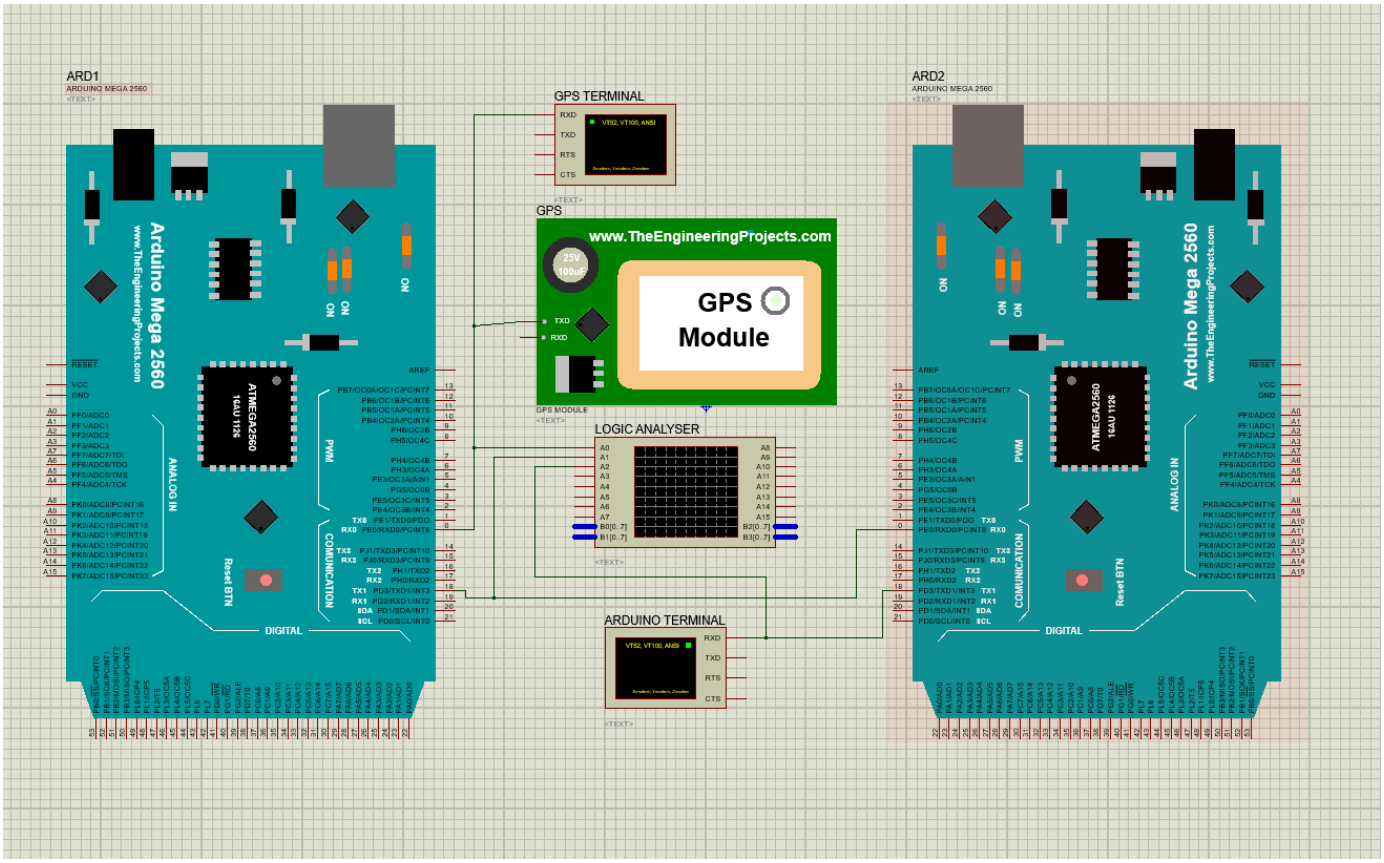


Fig. 7 Captură ecran schema realizată

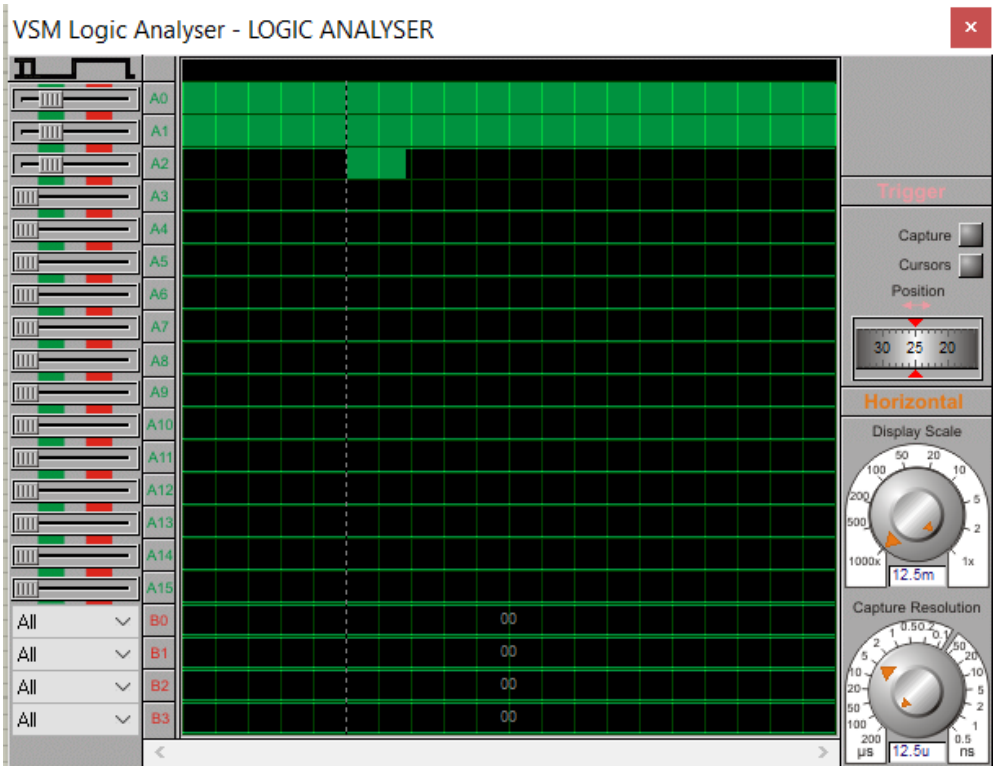


Fig. 8 Captură ecran analizor logic virtual

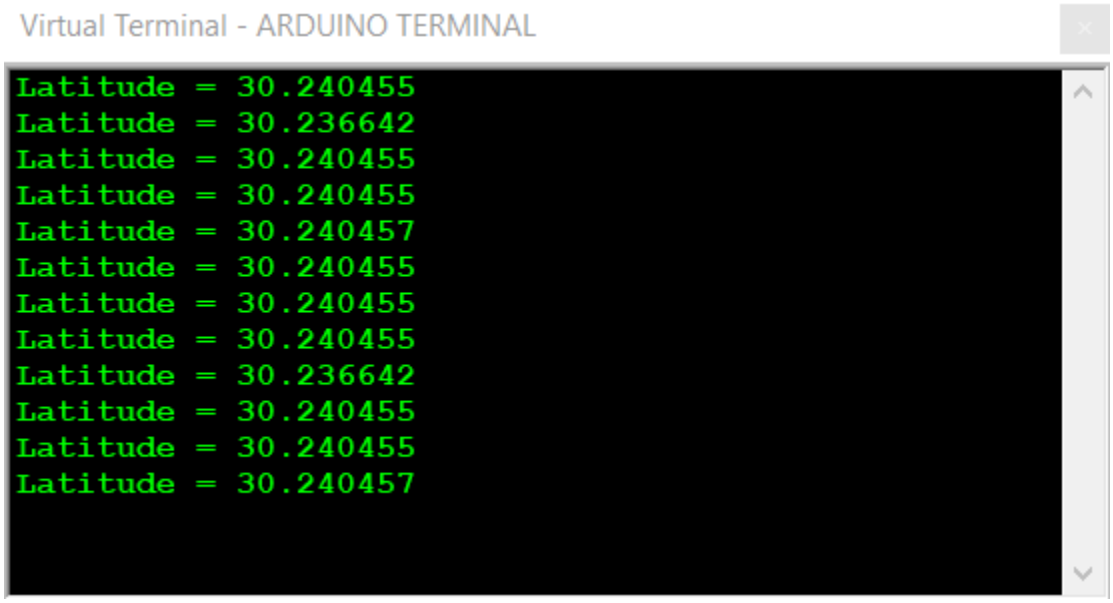


Fig. 9 Captură ecran funcționare 1

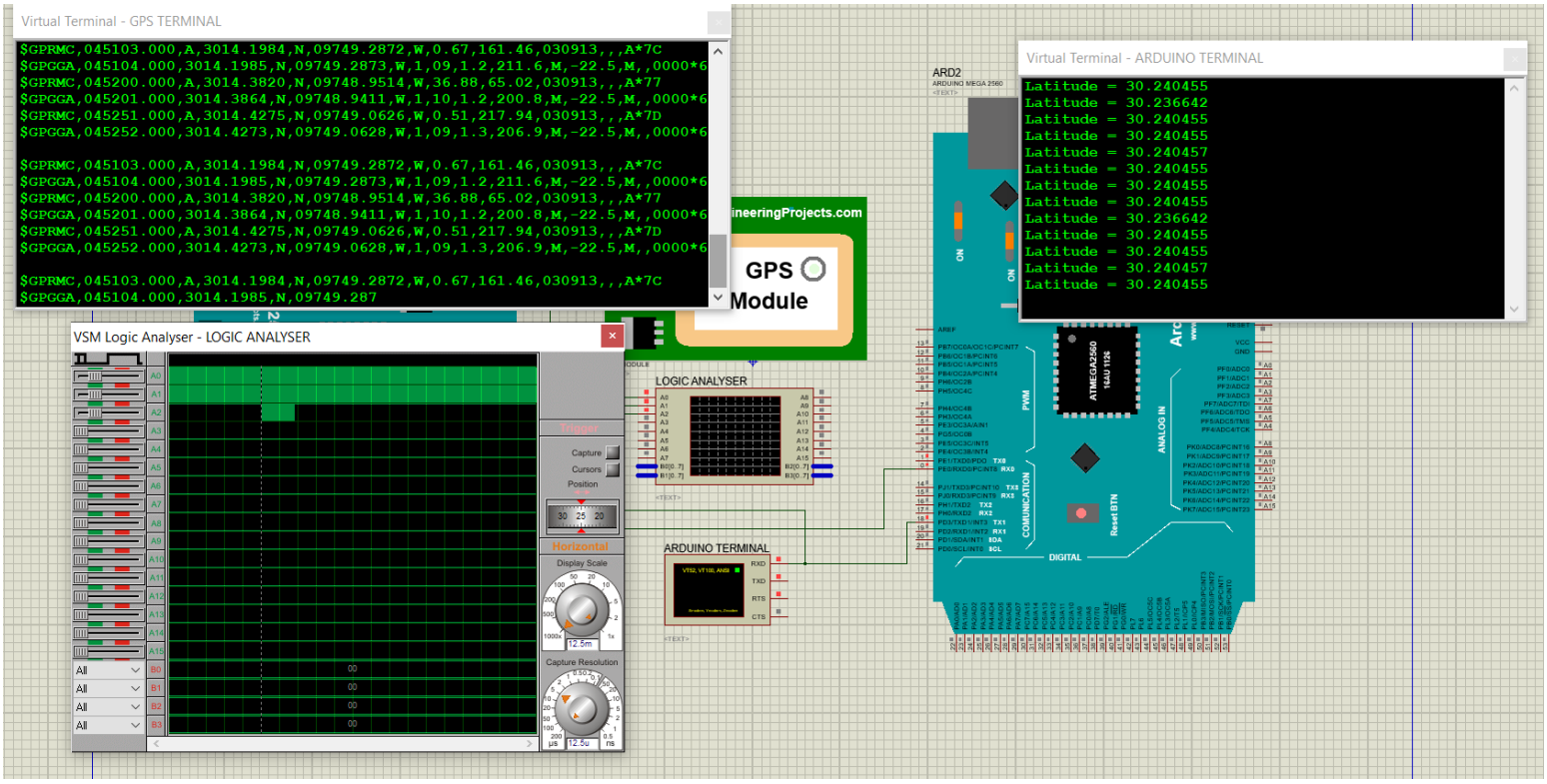


Fig. 10 Captură ecran funcționare 2

2.3. Codul

```
/*Codul Arduino Board conectat la modulul GPS – ARD1*/

/*

* Author:      Sfrangeu Bogdan-Otniel

* Purpose:     Receptie mesaje de la modul GPS si transmiterea lor la al doilea arduino.

* Language: C

*/

void setup()

{

        Serial.begin(9600);

        Serial1.begin(9600);

}
```

```
void loop()
{
	while (Serial.available())
	{
		char c = Serial.read();
		Serial1.write(c);
	}
}


/*Codul Arduino Board conectat la primul Arduino Board – ARD2*/


/*
* Author:	Sfrangeu Bogdan-Otniel
* Purpose:	Receptie mesaje de la primul arduino, extragerea latitudinii si afisarea acesteia.
* Language: C
*/
#include <TinyGPS.h>
TinyGPS gps; //Creates a new instance of the TinyGPS object
void setup()
{
  Serial.begin(9600);
  Serial1.begin(9600);
}
void loop()
{
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed; // For one second we parse GPS data and report some key values
```

```
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (Serial.available())
    {
      char c = Serial.read();
      if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }
  if (newData) //If newData is true
  {
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
    Serial1.print("Latitude = ");
    Serial1.println(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
  }
}
```

2.4. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica)

Pentru rezolvarea exercitiului s-a creat o schema in mediul de dezvoltare Proteus ce include 2 placi Arduino Mega 2560, un modul GPS, un Analizator Logic și doua terminale virtuale.

Prima placa de dezvoltare Atmega2560, ARD1 in schema, va receptiona date de la modulul GPS sub forma unor propozitii NMEA, dupa care le va transmite catre a doua placa de dezvoltare Atmega2560, ARD2 in schema, folosind comunicatia seriala UART.

Cea de a doua placa de dezvoltare ATmega2560, ARD2, dupa ce va receptiona mesajul de la prima placa de dezvoltare ARD1, va extrage latitudinea si va trasmite aceste date catre un terminal virtual unde se pot vizualiza .

Datele receptionate de placa arduino ARD1, receptionate de la modulul GPS vor fi si afisate pe un terminal virtual.

Pentru studierea comportamentului intregului program, vom folosi un Logic Analyser prin care se poate afirma comunicatia dintre modulul GPS, placa arduino ARD1 si placa arduino ARD2.

Nu au aparut probleme in rezolvarea exercitiului, acesta asemanandu-se foarte mult cu problema studiata in cadrul laboratorului „Transimisia seriala. UART & USART”.

3. Conceptul de magistrală I2C

3.1. Cerințe

În mediul de simulare Proteus să se realizeze un montaj între un Arduino UNO și trei senzori de temperatura DS1621*, comunicația având loc pe magistrala I2C. Modulul Ardunino va citi valorile de temperatură (virtuale) măsurate de cei trei senzori in urma recepționari unei comenzi pe interfața UART. Afișați valorile citite pe un terminal virtual. Daca o temperatura citită depășește 30 grade C se va aprinde un LED de culoare roșie.
Exemplu comanda:
Read_Sensor_1 – citire senzor 1
Read_Sensor_2 – citire senzor 2
Read_Sensor_3 – citire senzor 3

* http://datasheets.maximintegrated.com/en/ds/DS1621.pdf

3.2. Prezentarea soluției

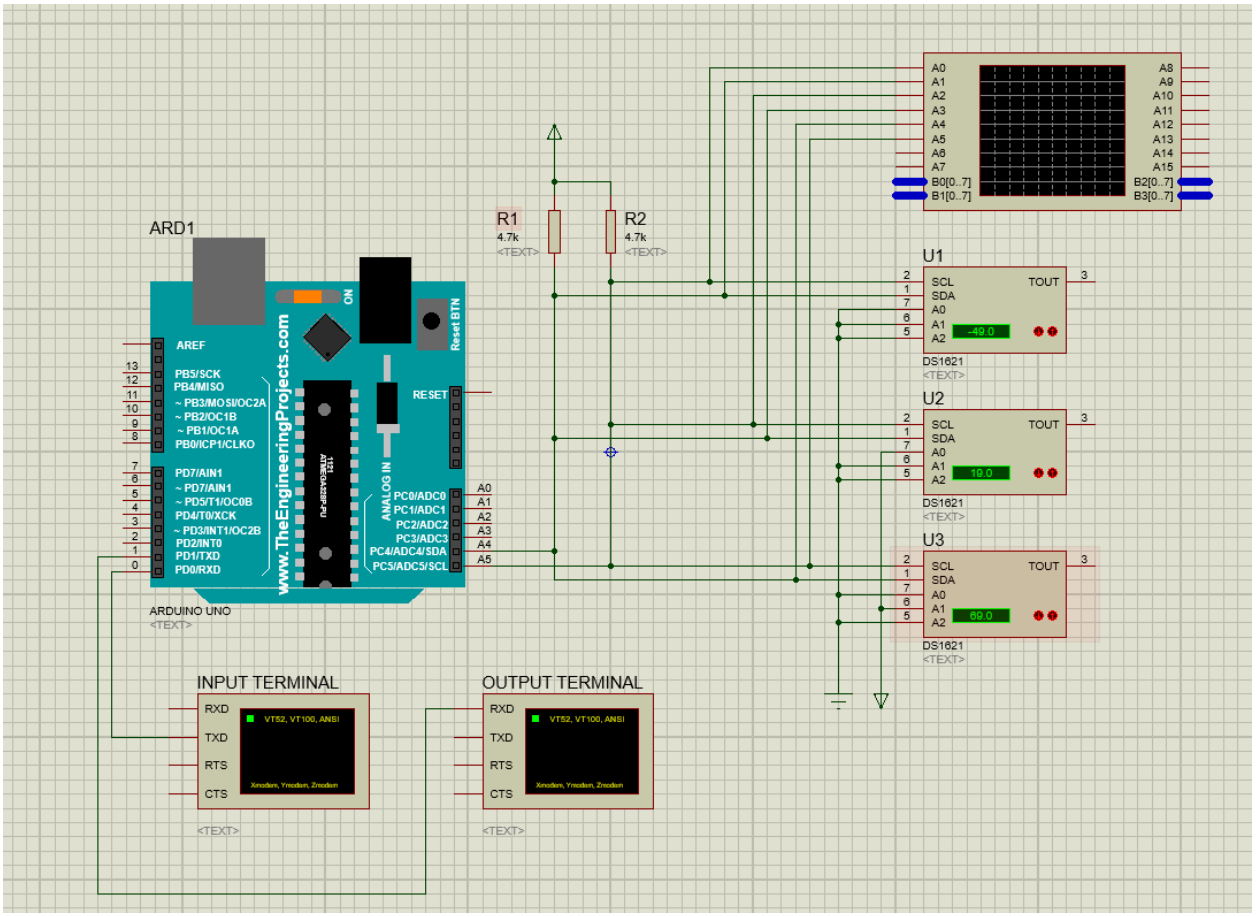


Fig. 11 Captură ecran schema realizată

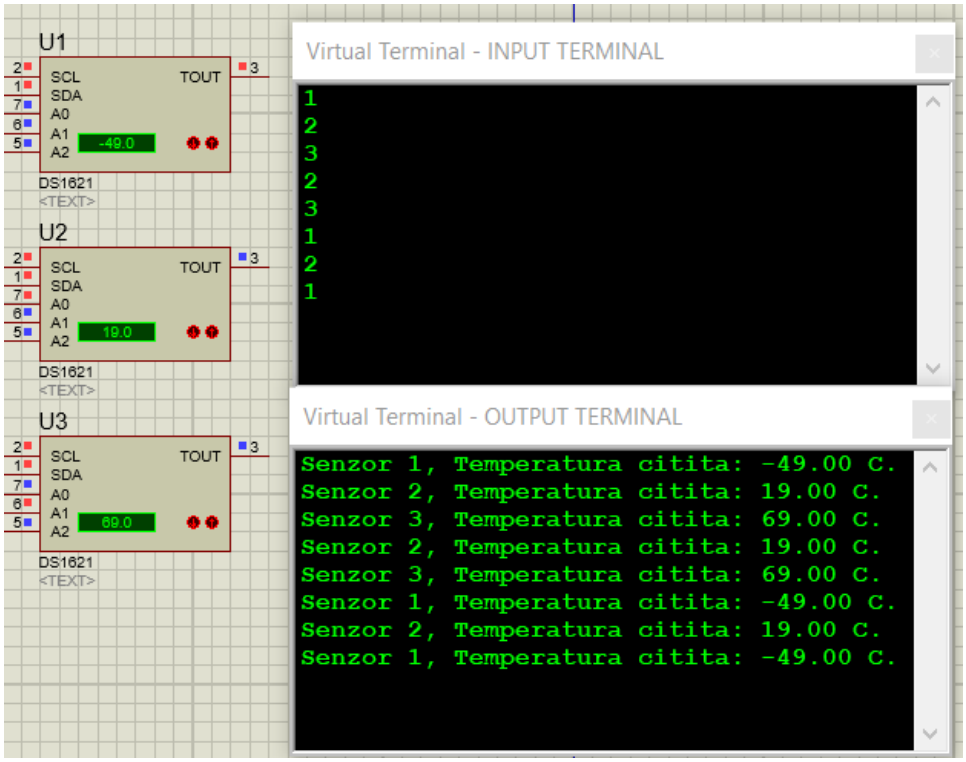


Fig. 12 Captură ecran citire temperatura

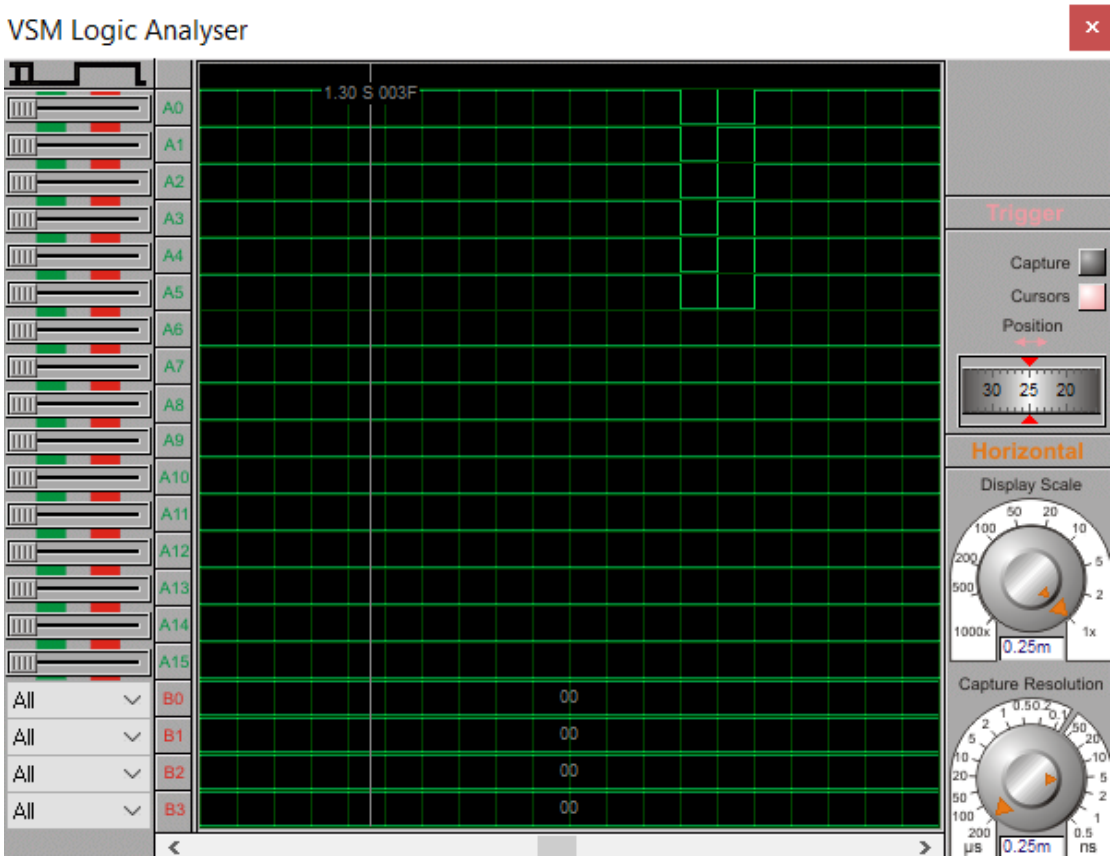


Fig. 13 Captură ecran analizor logic virtual

3.3. Codul sursă

```
/*
* Author:   Sfrangeu Bogdan-Otniel
* Purpose:  Citire sir comanda prin comunicatie seriala UART, procesare sir comanda si citire temperatura de la senzor specific prin i2c, afisarea temperaturii in terminal virtual.
* Language: C
*/


#include <Wire.h>
// define DS1621 I2C slave address (1001+A2+A1+A0)
// Sensor 1                     --> 1001 0  0  0 = 0x48
// Sensor 2                     --> 1001 0  0  1 = 0x49
// Sensor 3                     --> 1001 0  1  1 = 0x4A
#define DS1621_S1_ADDRESS  0x48
#define DS1621_S2_ADDRESS  0x49
#define DS1621_S3_ADDRESS  0x4A


char commandCharacter;
float temperature;


void setup()
{
  Serial.begin(9600);
  Wire.begin();                                 // join i2c bus
  // initialize DS1621 sensor 1
  Wire.beginTransmission(DS1621_S1_ADDRESS);  // connect to DS1621 (send DS1621 address)
  Wire.write(0xAC);                             // send configuration register address (Access Config)
```

```
  Wire.write(0);                                  // perform continuous
conversion

  Wire.beginTransmission(DS1621_S1_ADDRESS);  // send repeated start
condition

  Wire.write(0xEE);                               // send start temperature
conversion command

  Wire.endTransmission();                         // stop transmission and
release the I2C bus


  // initialize DS1621 sensor 2

  Wire.beginTransmission(DS1621_S2_ADDRESS);  // connect to DS1621 (send
DS1621 address)

  Wire.write(0xAC);                               // send configuration
register address (Access Config)

  Wire.write(0);                                  // perform continuous
conversion

  Wire.beginTransmission(DS1621_S2_ADDRESS);  // send repeated start
condition

  Wire.write(0xEE);                               // send start temperature
conversion command

  Wire.endTransmission();                         // stop transmission and
release the I2C bus


  // initialize DS1621 sensor 3

  Wire.beginTransmission(DS1621_S3_ADDRESS);  // connect to DS1621 (send
DS1621 address)

  Wire.write(0xAC);                               // send configuration
register address (Access Config)

  Wire.write(0);                                  // perform continuous
conversion

  Wire.beginTransmission(DS1621_S3_ADDRESS);  // send repeated start
condition

  Wire.write(0xEE);                               // send start temperature
conversion command

  Wire.endTransmission();                         // stop transmission and
release the I2C bus
```

```
}

void loop()
{
  if(Serial.available() > 0)
  {
    commandCharacter = Serial.read();
  }
  if (commandCharacter == '1')
  {
    temperature = get_temperature(DS1621_S1_ADDRESS);
    if(temperature > 125) temperature -= 256;
    Serial.print("Senzor 1, Temperatura citita: ");
    Serial.print(temperature);
    Serial.println(" C.");
    commandCharacter = '0';
  }

  if (commandCharacter == '2')
  {
    temperature = get_temperature(DS1621_S2_ADDRESS);
    if(temperature > 125) temperature -= 256;
    Serial.print("Senzor 2, Temperatura citita: ");
    Serial.print(temperature);
    Serial.println(" C.");
    commandCharacter = '0';
  }

  if (commandCharacter == '3')
  {
```

```
    temperature = get_temperature(DS1621_S3_ADDRESS);
    if(temperature > 125) temperature -= 256;
    Serial.print("Senzor 3, Temperatura citita: ");
    Serial.print(temperature);
    Serial.println(" C.");
    commandCharacter = '0';
  }
}


float get_temperature(uint8_t DS1621_S_ADDRESS)
{
  Wire.beginTransmission(DS1621_S_ADDRESS); // connect to DS1621 (send
DS1621 address)
  Wire.write(0xAA);                         // read temperature command
  Wire.endTransmission(false);              // send repeated start
condition
  Wire.requestFrom(DS1621_S_ADDRESS, 2);    // request 2 bytes from DS1621
and release I2C bus at end of reading
  uint8_t t_msb = Wire.read();              // read temperature MSB
register
  uint8_t t_lsb = Wire.read();              // read temperature LSB
register


  float raw_t = t_msb;
  if(t_lsb) raw_t += 0.5;
  return raw_t;
}
```

3.4. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica)

In rezolvarea problemei s-a creat o schema de simulare in mediul de dezvoltare Proteus, utilizand o placa Arduino UNO, trei senzori de temperatură DS1621, doua terminale virtuale si un analizor logic.

Senzorii de temperatura comunica cu placa Arduino UNO folosind protocolul $I2C$. Fiecarui senzor ii corespunde un identificator, sau o adresa specifica, prin care se poate realiza comunicarea Arduino UNO – Senzor DS1621, prin aceasta adresa se determina cu care Senzor DS1621 va comunica placa Arduino UNO. Adresele fiecarui senzor sunt prestabilite, sau „codate hardware" prin conectarea pinilor A2, A1, A0 fie la GND fie la VCC.

Pentru definirea adreselor ne-am folosit de urmatoarea sintaxa: Adresa Senzor = [1001 + A2 + A1 + A0], unde + semnifica concatenarea starii logice a pinilor A2, A1, A0. Ca un exemplu de adresare a senzorilor, primul senzor va avea adresa: [1001 + 0 + 0 + 0] = 0x48, iar ultimul senzor va avea adresa: [1001 + 0 + 1 + 0] = 0x4A.

Prin comunicatia seriala UART, se va trimite o comanda placii Arduino UNO, comanda prin care va fi selectat senzorul de temperatura DS1621 si de la care se va astepta 2 bytes de date, prin care va fi calculata temperatura si afisata intr-un terminal virtual.

In rezolvarea exercitiului, au existat initial probleme in configurarea si utilizarea corecta a senzorilor de temperatura DS1621, atat in configuratia hardware, cat si in cea software, datorita adresarii senzorilor prin pinii A2, A1, A0.

4. Conceptul de magistrală SPI

4.1. Cerințe

Să se implementeze un sistem Single-Master – Multi-Slave pentru citirea și scrierea datelor din două memorii EEPROM (25LC040A) pe baza protocolului definit mai jos:

Operație	Separator	Identificator Slave	Separator	Adresa	Separator	Valoare

Operație: 0 – scriere, 1 – citire

Identificator Slave: 0 – slave 0, 1 – slave 1

Adresa: adresa de scriere in EEPROM

Valoare: valoarea transmisa pentru scriere in EEPROM (în cazul citirii se va trimite valoarea 0xFF)

Separator: caracterul „*"

Comenzile vor fi primite prin intermediul unui terminal virtual:

Exemplu: 0*1*25*12 – se va face o scriere în memoria slave 1, la adresa 25, valoarea 12

4.2. Prezentarea soluției

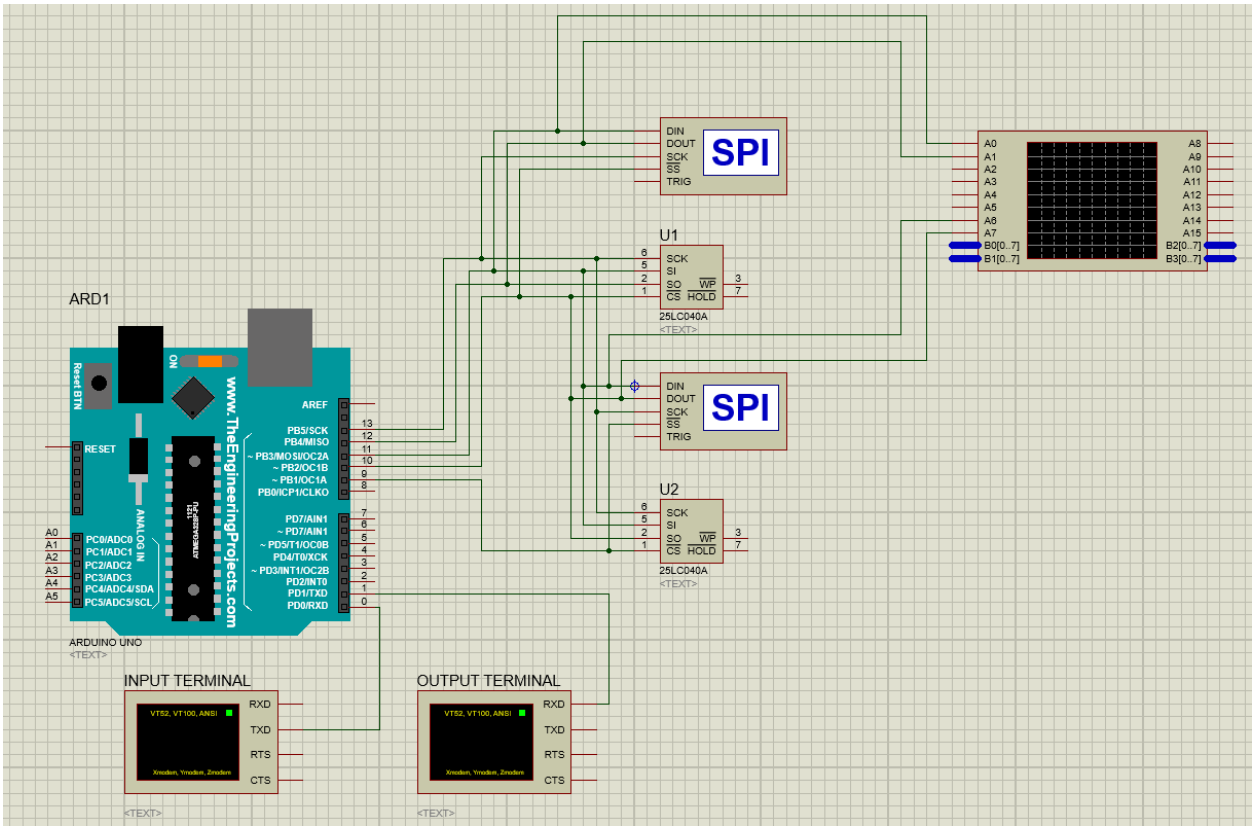


Fig. 14 Captură ecran schema realizată

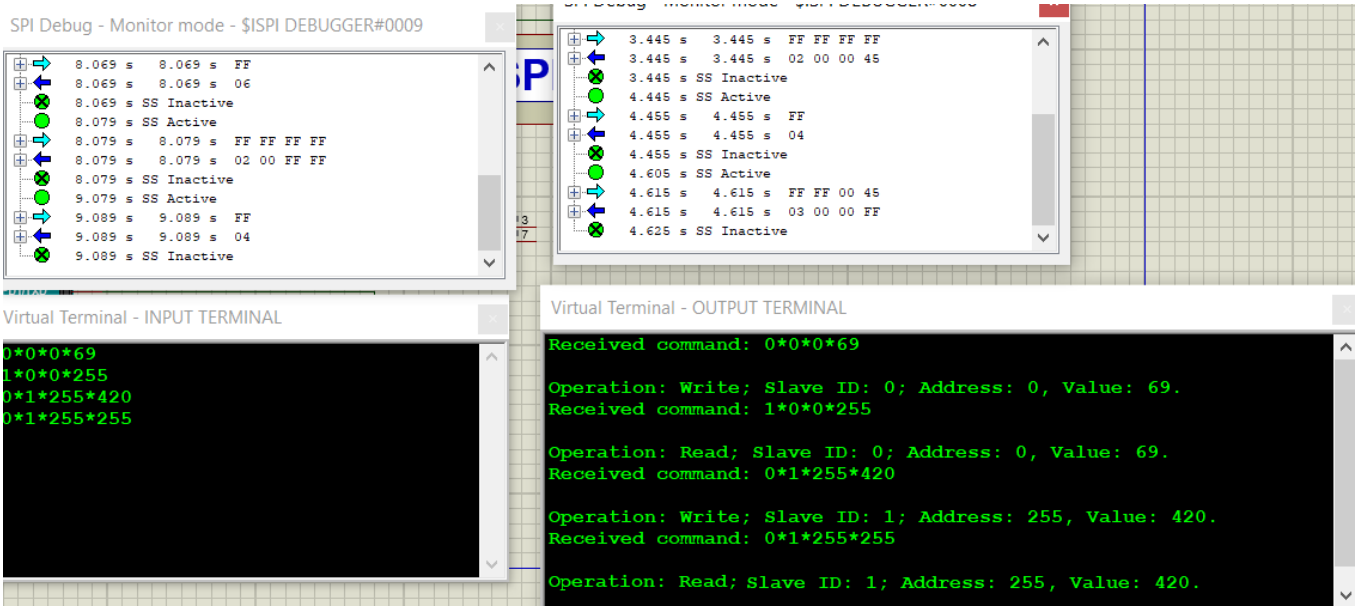


Fig. 15 Captură ecran din timpul funcționării

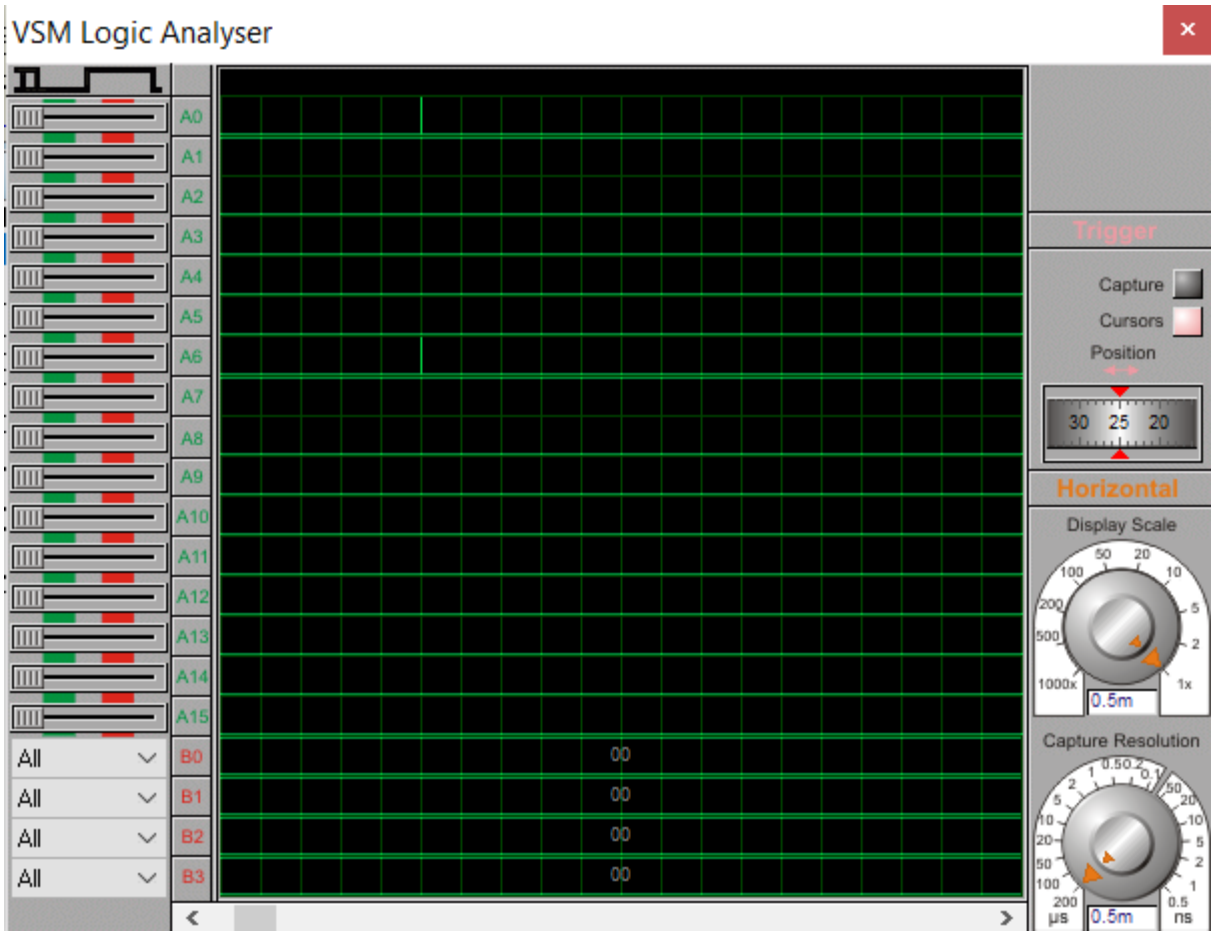


Fig. 16 Captură ecran analizor logic virtual

4.3. Codul sursă

```
/*
 * Author:    Sfrangeu Bogdan-Otniel
 * Purpose:   Receptie sir comanda prin UART, dupa care parseaza si
comanda operatii de scriere/citire in memoriile eeprom prin spi.
 * Language:  C
 */


#define MOSI 11               //MASTER OUT SLAVE IN
#define MISO 12               //MASTER IN SLAVE OUT
#define SPICLOCK 13           //MASTER CLOCK OUTPUT
#define SLAVESELECTEEPROM1 10 // SLAVE SELECT CHIP EEPROM 1
#define SLAVESELECTEEPROM2  9 // SLAVE SELECT CHIP EEPROM 2
```

```
#define WREN 6    //Activare scriere pentru EEPROM
#define WRDI 4    //Dezactivare scriere pentru EEPROM
#define READ 3    //Operatie citire EEPROM
#define WRITE 2   //Operatie scriere EEPROM

#define SSTRANSITIONDELAY 10
#define WRITEDELAY 1000



//Variabile pentru datele de intrare/iesire EEPROM
byte eeprom_output_data = 0;
byte eeprom_input_data = 0;

String inputString = "";        // Variabila pentru stocarea comenzii
boolean stringComplete = false; // Comanda finalizata

uint8_t operation, id;
uint16_t address, value;

//Functie pentru transferul datelor
char spi_transfer(volatile char data)
{
  SPDR = data;                    // Incarcare date
  while (!(SPSR & (1 << SPIF))) {} // Pooling pentru a asigura transferul complet al datelor
  return SPDR;                    // recuperarea datelor citite
}

//Functie pentru citirea EEPROM
```

```
byte read_eeprom(int EEPROM_address, int EEPROM_slave)
{
  int data;                                   //Variabila temporara pentru
stocarea datelor citite
  digitalWrite(EEPROM_slave, LOW);            //Activare circuit eeprom
  delay(SSTRANSITIONDELAY);
  spi_transfer(READ);                         //Transmisie comanda citire
  spi_transfer((char)(EEPROM_address >> 8));  //Transmisie adresa MSB
first
  spi_transfer((char)(EEPROM_address));       //Transmisie adresa
  data = spi_transfer(0xFF);                  //Receptie caracter
  delay(SSTRANSITIONDELAY);
  digitalWrite(EEPROM_slave, HIGH);           //Dezactivare eeprom
  return data;
}


//Functie pentru scriere EEPROM
void write_eeprom(uint8_t eepromSelect, uint16_t eepromAddress, uint16_t
eepromValue)
{
  digitalWrite(eepromSelect, LOW);          //enable device
  delay(SSTRANSITIONDELAY);
  spi_transfer(WREN);                       //write enable
  digitalWrite(eepromSelect, HIGH);         //disable device
  delay(SSTRANSITIONDELAY);
  digitalWrite(eepromSelect, LOW);          //enable device
  spi_transfer(WRITE);                      //write instruction
  spi_transfer((char)(eepromAddress >> 8)); //send MSByte address first
  spi_transfer((char)(eepromAddress));      //send LSByte address
  spi_transfer((char)eepromValue);          //write data byte
  digitalWrite(eepromSelect, HIGH);         //disable device
```

```
  delay(WRITEDELAY);                          //wait for eeprom to finish
writing
  digitalWrite(eepromSelect, LOW);          //enable device
  delay(SSTRANSITIONDELAY);
  spi_transfer(WRDI);                       //write disable
  digitalWrite(eepromSelect, HIGH);         //disable device
}


void handleMemoryOperation()
{


  switch (operation)
  {
    case '0'://write operation
      switch(id)
      {
        case '0':
          Serial.print("Operation: Write; Slave ID: 0; Address: ");
          Serial.print(address);
          Serial.print(", Value: ");
          Serial.print(value);
          Serial.println(".");
          write_eeprom(SLAVESELECTEEPROM1, address, value);
        break;
        case '1':
          Serial.print("Operation: Write; Slave ID: 1; Address: ");
          Serial.print(address);
          Serial.print(", Value: ");
          Serial.print(value);
          Serial.println(".");
```

```
          write_eeprom(SLAVESELECTEEPROM2, address, value);
        break;
        default:
          Serial.println("Error: Invalid Slave ID.");
        break;
      }
      break;
    case '1'://read operation
      switch(id)
      {
        case '0':
          eeprom_output_data = read_eeprom(address, SLAVESELECTEEPROM1);
          Serial.print("Operation: Read; Slave ID: 0; Address: ");
          Serial.print(address);
          Serial.print(", Value: ");
          Serial.print(eeprom_output_data, DEC);
          Serial.println(".");
        break;
        case '1':
          eeprom_output_data = read_eeprom(address, SLAVESELECTEEPROM2);
          Serial.print("Operation: Read; Slave ID: 1; Address: ");
          Serial.print(address);
          Serial.print(", Value: ");
          Serial.print(eeprom_output_data, DEC);
          Serial.println(".");
        break;
        default:
          Serial.println("Error: Invalid Slave ID.");
        break;
      }
```

```
      break;
    default:
      Serial.println("Error: Invalid Operation.");
      break;
  }
}

//Functie pentru receptionarea datelor pe interfata seriala
void serialEvent()
{
  while (Serial.available() > 0)
  {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\r') stringComplete = true;
  }
}
void setup()
{
  Serial.begin(9600);                    //Initializarea interfata
seriala
  inputString.reserve(10);               //Alocare date comanda
  pinMode(MOSI, OUTPUT);                 //Configurare pin iesire
  pinMode(MISO, INPUT);                  //Configurare pin intrare
  pinMode(SPICLOCK, OUTPUT);             //Configurare pin iesire
  pinMode(SLAVESELECTEEPROM1, OUTPUT);   //Configurare pin iesire
  pinMode(SLAVESELECTEEPROM2, OUTPUT);   //Configurare pin iesire
  digitalWrite(SLAVESELECTEEPROM1, HIGH); //Dezactivare EEPROM 1
  digitalWrite(SLAVESELECTEEPROM2, HIGH); //Dezactivare EEPROM 2
```

```
  // SPCR = 01010000
  //interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  //sample on leading edge of clk,system clock/4 rate (fastest)
  SPCR = (1 << SPE) | (1 << MSTR);
  byte clr;   //Variabila temporara pentru curatarea registrelor SPE si
MSTR
  clr = SPSR; // "Curatarea" registrului SPSR
  clr = SPDR; // "Curatarea" registrului SPDR
}

void loop()
{
  /*
  if(Serial.available() > 0)
  {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\r') stringComplete = true;
  }
  */
  serialEvent();
  if (stringComplete)
  {
    Serial.print("Received command: ");
    Serial.println(inputString);

    char *cmd[4];
    char *ptr = NULL;
    byte i = 0;
    ptr = strtok(inputString.c_str(), "*");
```

```
    while(ptr != NULL)
    {
      cmd[i] = ptr;
      i++;
      ptr = strtok(NULL, "*");
    }
    operation = *cmd[0];
    id =*cmd[1];
    address = atoi(cmd[2]);
    value = atoi(cmd[3]);

    //Alegere operatie
    handleMemoryOperation();
    // Resetare variabile globale
    inputString = "";
    stringComplete = false;
  }
}
```

4.4. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica)

Pentru rezolvarea exercitiului s-a realizat o schema in mediul de dezvoltare Proteus, schema cuprinzând o placa Arduino UNO, doua memorii EEPROM 25LC040A, doua terminale virtuale, doua analizatoare SPI si un analizor logic.

Se transmite un sir de caractere placii Arduino UNO prin intermediul unui Terminal Virtual utilizand interfata UART, acest sir de caractere este un sir comanda, care dupa ce va fi parsat si prelucrat, in functiile de valorile obtinute, se va scrie sau citi in/din memoriile eeprom la adresele specificate, valorile specificate. Scrierea in memoriile eeprom se va face prin interfata SPI.

S-au intampinat cateva probleme in parsarea sirului comanda si in citirea informatiilor din memoriile eeprom, toate problemele au fost rezolvate in final.

5. Unelte pentru comunicații CAN

5.1. Cerințe

a) Să se realizeze următoarea rețea utilizând mediul CANoe:

-Nodul 1 - senzor de umiditate - acesta returnează aleatoriu valori între 0 si 30 cu o ciclicitate de 20 de secunde și le trimite pe magistrala. Legătura dintre intensitatea ploii si valorile senzorului sunt prezentate in tabelul următor:

Interval valori	Intensitate ploaie
0-2	Nu plouă
3-10	Redusa
11-20	Medie
21-30	Mare

-Nodul 2 - control ștergătoare - acest nod poate primi comenzi care sa indice viteza cu care trebuie sa se miște ștergătoarele. 0 - oprit, 1, 2, 3 - viteza maximă, >3 nu se ia in considerare comanda. Nodul va recepționa comenzi (actualiza viteza daca este cazul) si va trimite pe magistrala viteza curenta. La apăsarea tastei “S”, viteza curenta a ștergătoarelor va fi trimisa pe magistrala.

-Nodul 3 - ECU central - acesta monitorizează mesajele de pe magistrala și emite comenzi. Procesează datele de la senzorul de umiditate iar mai apoi trimite comenzi.

Fiecare mesaj de pe magistrala va fi afișat in consola. Ex. Viteza curentă ștergătoare: 2

Acolo unde nu este precizat, identificatorii mesajelor de CAN precum și datele trimise vor fi definite de student.

b) Utilizând mediul CANoe să se realizeze citirea unui text dintr-un fișier și transmiterea acestuia pe magistrală. Cadrele transmise vor fi afișate.

5.2. Prezentarea soluției subpunct a)

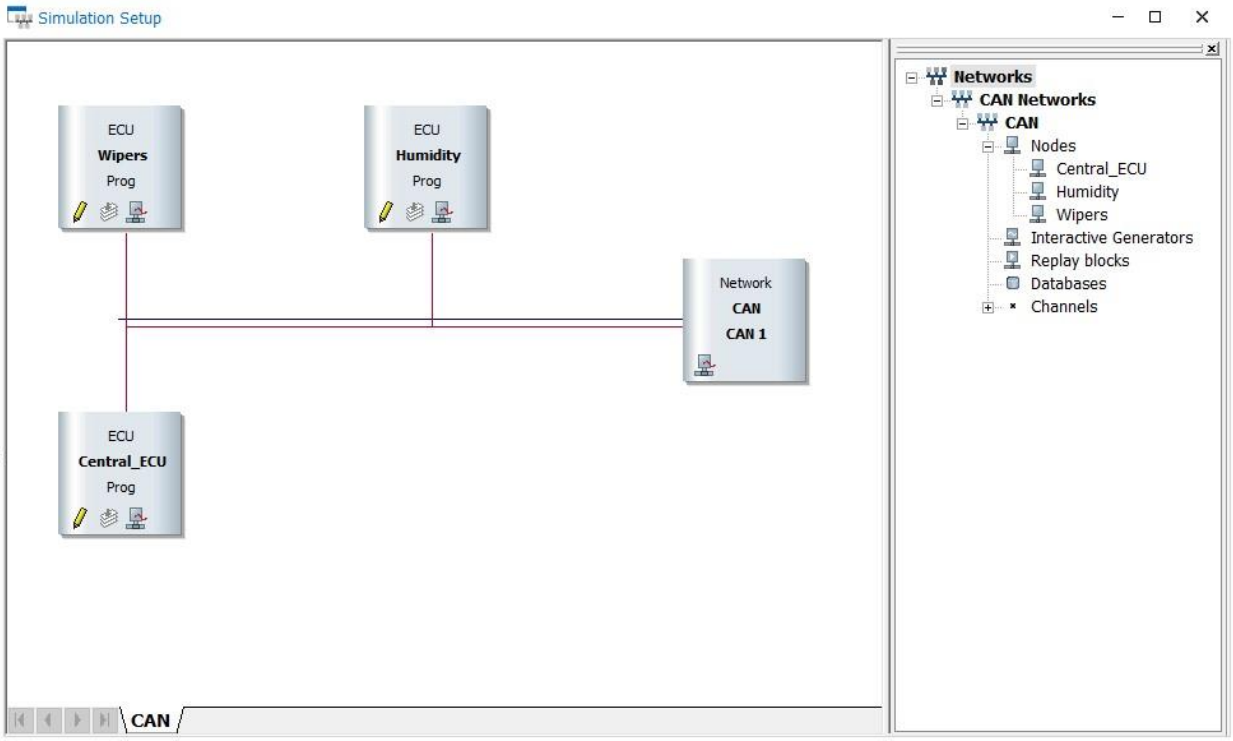


Fig. 17 Captură ecran configuratie CAN

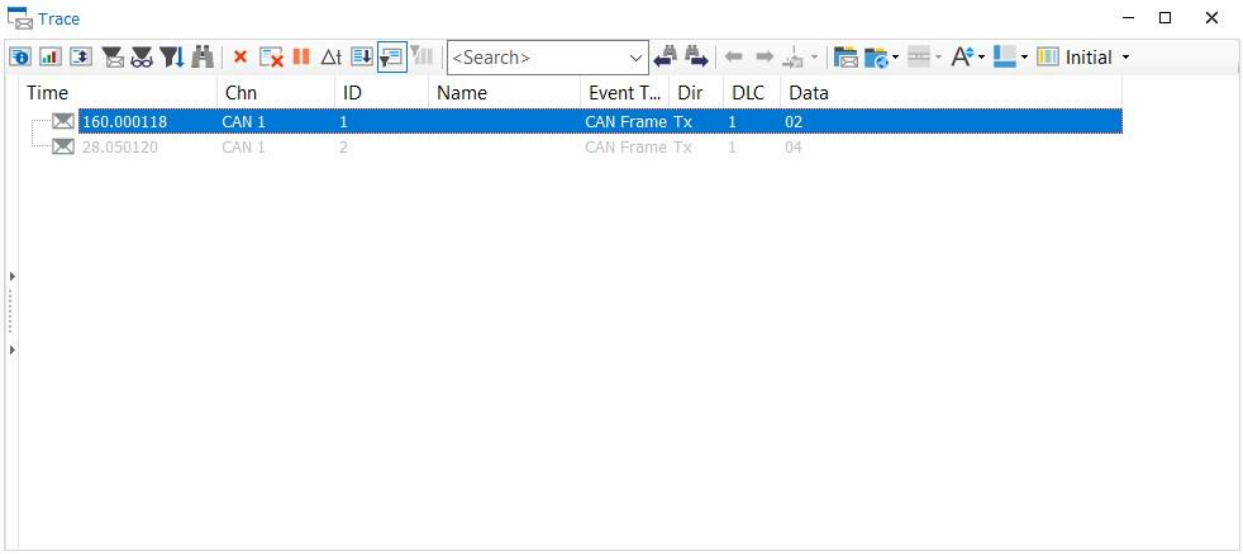


Fig. 18 Captură ecran Trace Window

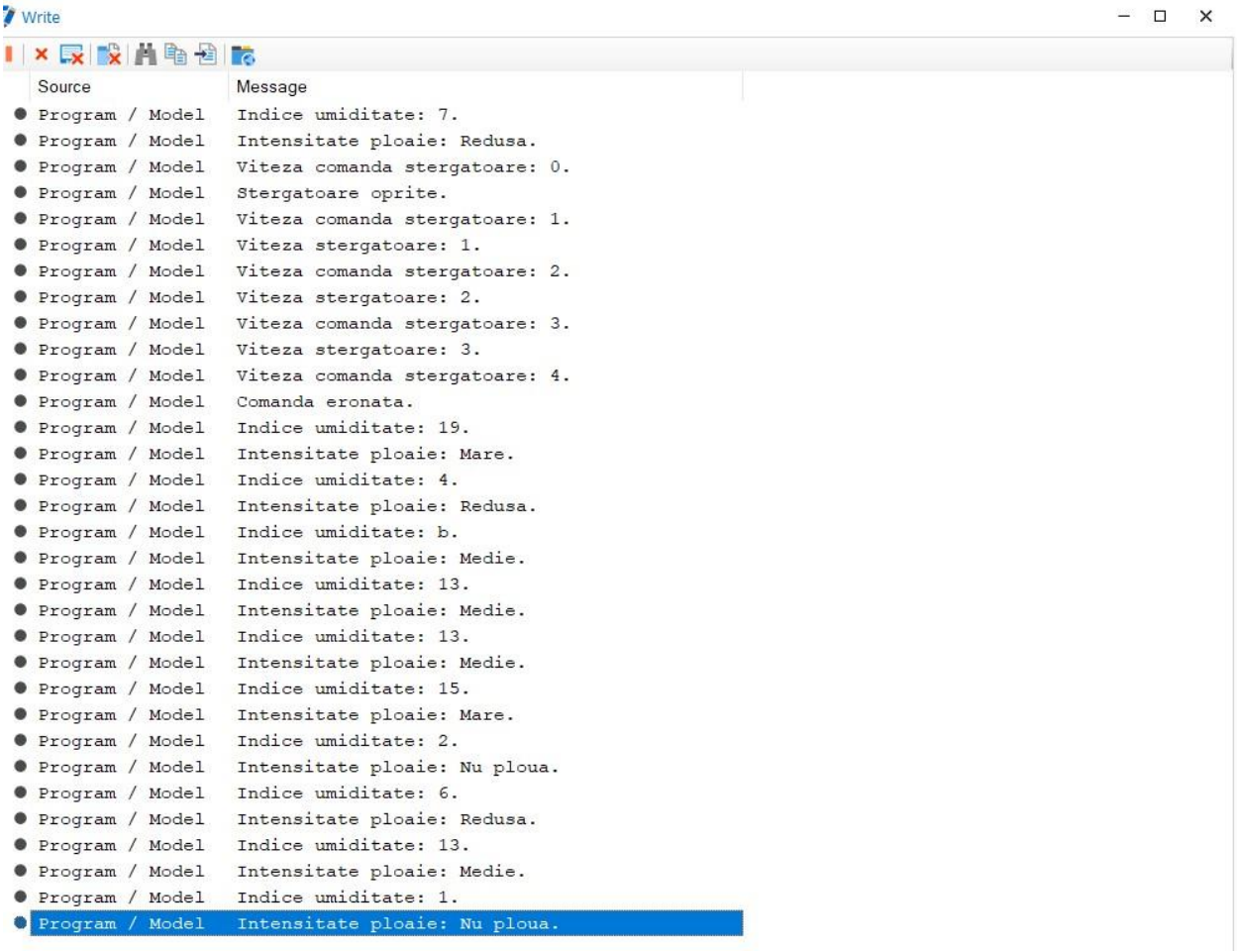


Fig. 19 Captură ecran Write Window

5.3. Cod subpunct a)

```
/*
* Author:    Sfrangeu Bogdan-Otniel
* Purpose:   Central ECU
* Language:  CAPL
*/
/*@!Encoding:1252*/
includes
{
```

```
}

variables
{
  byte myNumber = 0;
}

on message 0x1
{
  if(this.byte(0) < 3) write("Intensitate ploaie: Nu ploua.");
  else if(this.byte(0) < 11) write("Intensitate ploaie: Redusa.");
  else if(this.byte(0) < 21) write("Intensitate ploaie: Medie.");
  else if(this.byte(0) < 31) write("Intensitate ploaie: Mare.");
}

on key 's'
{
  Message* temporaryMessage;
  temporaryMessage.id = 0x2;
  temporaryMessage.dlc = 1;
  temporaryMessage.byte(0) = myNumber;
  write("Viteza comanda stergatoare: %x.", myNumber);
  if(myNumber > 3)  myNumber = 0;
  else myNumber++;
  output(temporaryMessage);
}
```

```
/*
* Author:    Sfrangeu Bogdan-Otniel
* Purpose:   Humidity Sensor
* Language:  CAPL
*/
/*@!Encoding:1252*/
includes
{

}

variables
{
  msTimer myTimer;
  byte myNumber = 0;
}

on timer myTimer
{
  Message* temporaryMessage;
  temporaryMessage.id = 0x1;
  temporaryMessage.dlc = 1;
  myNumber = random(31);
  temporaryMessage.byte(0) = myNumber;
  write("Indice umiditate: %x.", myNumber);
  output(temporaryMessage);
  setTimer(myTimer, 20000);
}
on start
{
```

```
  setTimer(myTimer, 20000);
}

/*
* Author:    Sfrangeu Bogdan-Otniel
* Purpose:   Wiper Controller
* Language:  CAPL
*/
/*@!Encoding:1252*/
includes
{

}

variables
{

}

on message 0x2
{
  if(this.byte(0) == 0) write("Stergatoare oprite.");
  else if(this.byte(0)== 1) write("Viteza stergatoare: 1.");
  else if(this.byte(0)== 2) write("Viteza stergatoare: 2.");
  else if(this.byte(0)== 3) write("Viteza stergatoare: 3.");
  else write("Comanda eronata.");
}
```

5.4. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica) subpunct a)

Pentru rezolvare exercitiului, a fost creata o retea CAN pur virtuala in mediul de dezvoltare CANoe. Reteaua contine 3 ECU Nodes si 1 Network Node.

Node-ul Humidity are rolul de a genera un numar aleator in intevalul [0, 30] la fiecare 20 de secunde, afisarea numarului aleator generat si transmiterea unui mesaj magistralei CAN, identificat printr-un id specific.

Node-ul Wipers are rolul de preluare a unei comenzi, iar in functie de aceasta comanda, ajusteaza viteza stergatoarelor, si afisarea operatiei efectuate.

Node-ul Central ECU are rolul de a trimite si a primi mesaje de comanda, primeste de la senzorul de umiditate o valoare in functie de care decide Intensitatea ploii, de asemenea afiseaza aceasta intensitate, mai are rolul ca la fiecare actionare a tastei ‚s' sa transmita un mesaj comanda catre Node-ul Wipers pentru a controla viteza stergatoarelor.

5.5. Prezentarea soluției subpunct b)

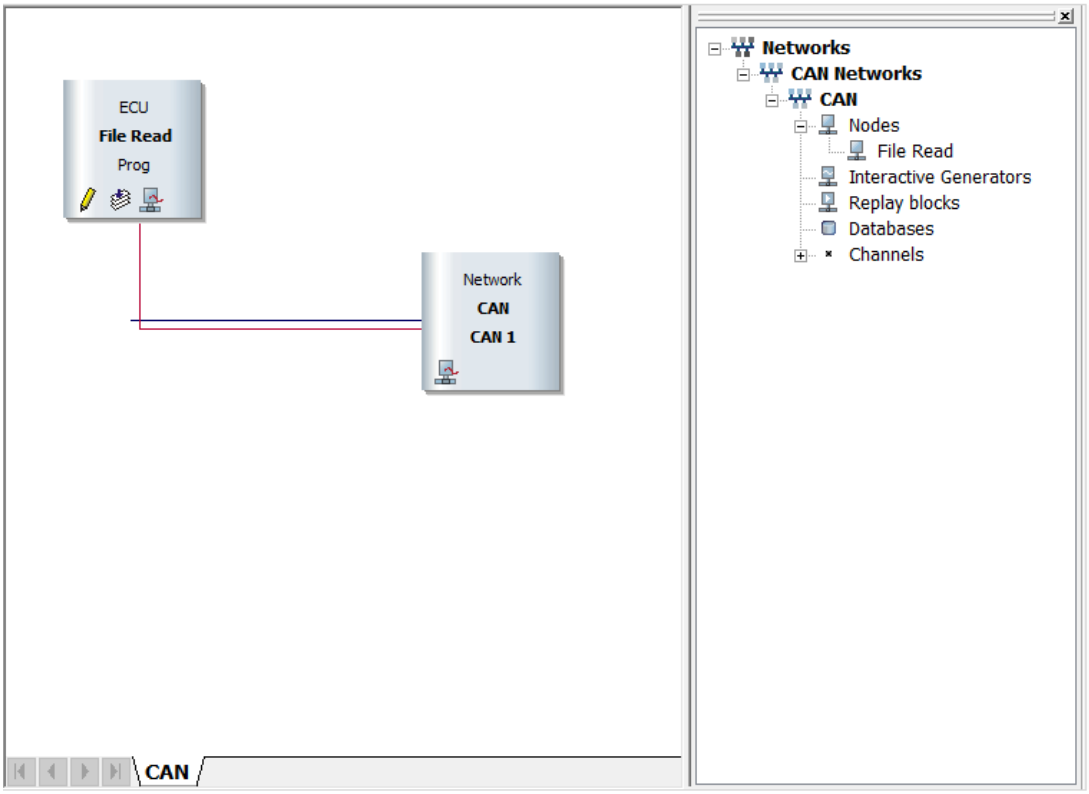
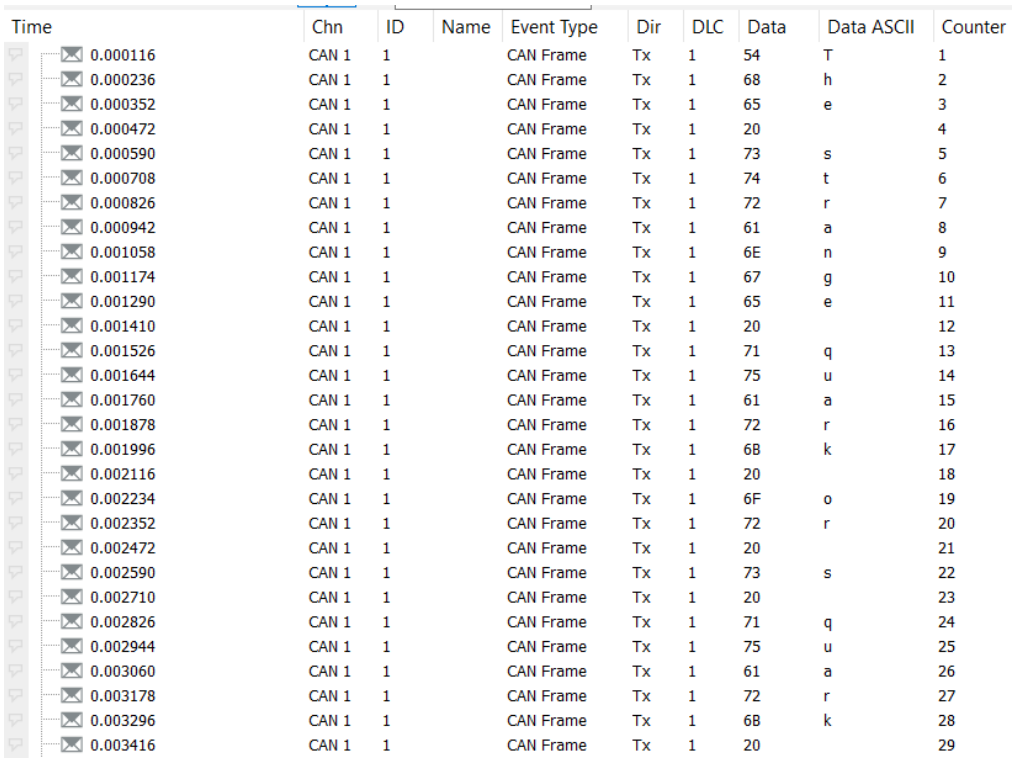
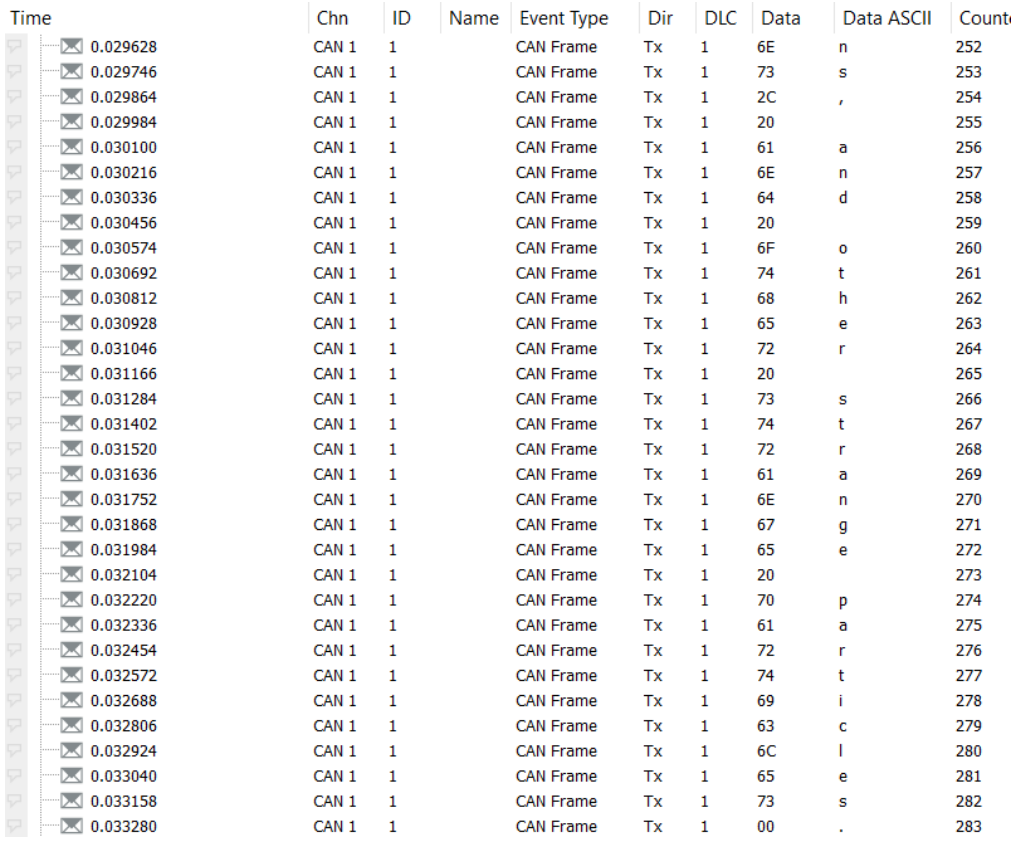


Fig. 20 Captură ecran configuratie CAN



Time	Chn	ID	Name	Event Type	Dir	DLC	Data	Data ASCII	Counter
0.000116	CAN 1	1		CAN Frame	Tx	1	54	T	1
0.000236	CAN 1	1		CAN Frame	Tx	1	68	h	2
0.000352	CAN 1	1		CAN Frame	Tx	1	65	e	3
0.000472	CAN 1	1		CAN Frame	Tx	1	20		4
0.000590	CAN 1	1		CAN Frame	Tx	1	73	s	5
0.000708	CAN 1	1		CAN Frame	Tx	1	74	t	6
0.000826	CAN 1	1		CAN Frame	Tx	1	72	r	7
0.000942	CAN 1	1		CAN Frame	Tx	1	61	a	8
0.001058	CAN 1	1		CAN Frame	Tx	1	6E	n	9
0.001174	CAN 1	1		CAN Frame	Tx	1	67	g	10
0.001290	CAN 1	1		CAN Frame	Tx	1	65	e	11
0.001410	CAN 1	1		CAN Frame	Tx	1	20		12
0.001526	CAN 1	1		CAN Frame	Tx	1	71	q	13
0.001644	CAN 1	1		CAN Frame	Tx	1	75	u	14
0.001760	CAN 1	1		CAN Frame	Tx	1	61	a	15
0.001878	CAN 1	1		CAN Frame	Tx	1	72	r	16
0.001996	CAN 1	1		CAN Frame	Tx	1	6B	k	17
0.002116	CAN 1	1		CAN Frame	Tx	1	20		18
0.002234	CAN 1	1		CAN Frame	Tx	1	6F	o	19
0.002352	CAN 1	1		CAN Frame	Tx	1	72	r	20
0.002472	CAN 1	1		CAN Frame	Tx	1	20		21
0.002590	CAN 1	1		CAN Frame	Tx	1	73	s	22
0.002710	CAN 1	1		CAN Frame	Tx	1	20		23
0.002826	CAN 1	1		CAN Frame	Tx	1	71	q	24
0.002944	CAN 1	1		CAN Frame	Tx	1	75	u	25
0.003060	CAN 1	1		CAN Frame	Tx	1	61	a	26
0.003178	CAN 1	1		CAN Frame	Tx	1	72	r	27
0.003296	CAN 1	1		CAN Frame	Tx	1	6B	k	28
0.003416	CAN 1	1		CAN Frame	Tx	1	20		29

Fig. 21 Captură ecran Trace Window 1



Time	Chn	ID	Name	Event Type	Dir	DLC	Data	Data ASCII	Count
0.029628	CAN 1	1		CAN Frame	Tx	1	6E	n	252
0.029746	CAN 1	1		CAN Frame	Tx	1	73	s	253
0.029864	CAN 1	1		CAN Frame	Tx	1	2C	,	254
0.029984	CAN 1	1		CAN Frame	Tx	1	20		255
0.030100	CAN 1	1		CAN Frame	Tx	1	61	a	256
0.030216	CAN 1	1		CAN Frame	Tx	1	6E	n	257
0.030336	CAN 1	1		CAN Frame	Tx	1	64	d	258
0.030456	CAN 1	1		CAN Frame	Tx	1	20		259
0.030574	CAN 1	1		CAN Frame	Tx	1	6F	o	260
0.030692	CAN 1	1		CAN Frame	Tx	1	74	t	261
0.030812	CAN 1	1		CAN Frame	Tx	1	68	h	262
0.030928	CAN 1	1		CAN Frame	Tx	1	65	e	263
0.031046	CAN 1	1		CAN Frame	Tx	1	72	r	264
0.031166	CAN 1	1		CAN Frame	Tx	1	20		265
0.031284	CAN 1	1		CAN Frame	Tx	1	73	s	266
0.031402	CAN 1	1		CAN Frame	Tx	1	74	t	267
0.031520	CAN 1	1		CAN Frame	Tx	1	72	r	268
0.031636	CAN 1	1		CAN Frame	Tx	1	61	a	269
0.031752	CAN 1	1		CAN Frame	Tx	1	6E	n	270
0.031868	CAN 1	1		CAN Frame	Tx	1	67	g	271
0.031984	CAN 1	1		CAN Frame	Tx	1	65	e	272
0.032104	CAN 1	1		CAN Frame	Tx	1	20		273
0.032220	CAN 1	1		CAN Frame	Tx	1	70	p	274
0.032336	CAN 1	1		CAN Frame	Tx	1	61	a	275
0.032454	CAN 1	1		CAN Frame	Tx	1	72	r	276
0.032572	CAN 1	1		CAN Frame	Tx	1	74	t	277
0.032688	CAN 1	1		CAN Frame	Tx	1	69	i	278
0.032806	CAN 1	1		CAN Frame	Tx	1	63	c	279
0.032924	CAN 1	1		CAN Frame	Tx	1	6C	l	280
0.033040	CAN 1	1		CAN Frame	Tx	1	65	e	281
0.033158	CAN 1	1		CAN Frame	Tx	1	73	s	282
0.033280	CAN 1	1		CAN Frame	Tx	1	00	.	283

Fig. 22 Captură ecran Trace Window 2

5.6. Cod subpunct b)

```
/*
* Author:    Sfrangeu Bogdan-Otniel
* Purpose:   Read text from text file.
* Language:  CAPL
*/
/*@!Encoding:1252*/
includes
{

}

variables
{
  dword file;
  char buffer[283];
  int index;
}

on start
{
  Message* temporaryMessage;
  temporaryMessage.id = 0x1;
  temporaryMessage.dlc = 1;
  file = openFileRead("source.txt", 0);
  while(fileGetString(buffer,elcount(buffer),file)!=0)
  {
    for(index = 0;index < elcount(buffer); index++)
    {
      temporaryMessage.byte(0) = buffer[index];
```

```
      output(temporaryMessage);
    }
  }
  fileClose(file);
}
```

5.7. Descriere rezolvare si comentarii (detalii de implementare, probleme întâmpinate, fenomenele observate, interpretare tehnica) subpunct b)

Pentru rezolvarea problemei am folosit un ECU Node si un Network Node. Rolul de citire a datelor din fisierul text sursa „source.txt” ii revine Node-ului ECU denumit FileRead.

Datele vor fi citite din fisier, caracter cu caracter, iar la fiecare preluare a unui caracter, se va transmite un mesaj pe magistrala CAN, mesaj cu ID 0x1 si DLC 1.

Se poate observa din Trace Window, in coloana Data ASCII si valoarea in ASCII a caracterelor trimise pe magistrala CAN.

6. Concluzii

Nu prea am ce sa zic, materia a fost destul de interesanta, pot sa zic ca am ajuns un nivel destul de basic in intelegerea transmisiilor de date prin diferite protocoale, UART, SPI, I2C, CAN, si in cazul filtrelor.

In urma rezolvarii proiectului, consider ca am ajuns sa ating un nivel destul de basic in Comunicatii de Date