

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

RAPORT

Lucrare de laborator Nr.3
Disciplina: IoT
Tema: Senzori.
Achiziție și condiționare semnal

A efectuat:

st.gr.TI-212,
Muntean Mihai

A verificat :

asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

Să se realizeze o aplicație în bază de MCU care va condiționa semnalul preluat de la sensor (Sarcina 1), și va afișa parametrul fizic la un terminal (LCD și/sau Serial).

Fiecare student va selecta un sensor fie analogic fie digital (nu binar) din PDF atașat sau:

<http://www.37sensors.com/>

Obiective:

1. Să se achiziționeze semnalul de la sensor;
2. Să se condiționeze semnalul implicând filtre digitale și alte metode;
3. Să se afișeze datele pe afișor LCD și / sau Serial.

Introducere

Senzorii sunt dispozitive esențiale în diverse domenii ale ingineriei și științelor, fiind capabili să măsoare și să detecteze parametri fizici din mediul înconjurător, cum ar fi temperatura, presiunea, lumina, umiditatea sau mișcarea. Aceștia transformă mărimile fizice într-un semnal electric măsurabil, care poate fi utilizat pentru monitorizare, control sau analiză în sisteme automate.

Achiziția de semnal de la un senzor analogic implică mai multe etape, începând cu captarea semnalului de către un **convertor analog-digital** (ADC) și terminând cu conversia acestuia într-un parametru fizic măsurabil. Senzorii analogici, precum termistorii, fotorezistențele sau alți senzori de mediu, generează semnale continue care variază în funcție de mărimea fizică măsurată (de exemplu, temperatura sau lumină). Pentru a putea utiliza aceste semnale într-un sistem digital, acestea trebuie convertite în valori numerice folosind un ADC.

1 Captarea informației prin ADC

Microcontrolerele, precum cele din gama Arduino, sunt echipate cu convertoare ADC care transformă semnalul analogic generat de un senzor într-o valoare numerică discretă. Acest proces constă în eșantionarea semnalului de tensiune la intervale fixe și conversia acestei tensiuni într-o valoare digitală proporțională.

- **Tensiunea de referință (V_{REF})** reprezintă valoarea maximă pe care ADC-ul o poate măsura (de obicei 5V sau 3.3V pentru multe plăci de dezvoltare).
- **Valoarea maximă a ADC-ului (ADC_{MAX})** este determinată de numărul de biți de rezoluție. De exemplu, un ADC de 10 biți poate genera valori între 0 și 1023.

Formula de conversie a valorii ADC în tensiune este:

$$Tensiune = \frac{adcValue * V_{REF}}{ADC_{MAX}}$$

unde $adcValue$ este valoarea citită de la ADC, V_{REF} este tensiunea maximă măsurabilă, iar ADC_{MAX} este valoarea maximă pe care ADC-ul o poate returna (1023 pentru un ADC de 10 biți).

2 Conversia Tensiunii în Parametrul Fizic

După ce semnalul analogic este transformat într-o valoare numerică de către ADC și ulterior convertit în tensiune, următorul pas este să transformăm această tensiune în parametrul fizic măsurat de senzor, cum ar fi temperatura, presiunea sau lumina. Această conversie necesită utilizarea unei **funcții de transfer** care descrie relația dintre tensiunea măsurată și parametrul fizic.

De exemplu, în cazul unui **termistor NTC** (Negative Temperature Coefficient), tensiunea măsurată trebuie utilizată pentru a calcula **rezistența termistorului** (NTC). Formula care leagă tensiunea și rezistența NTC este:

$$Rezistența_{NTC} = R_{REF} * \left(\frac{V_{REF}}{V_{NTC}} - 1 \right)$$

unde:

- R_{REF} este rezistența de referință din circuit (de obicei 10kΩ),
- V_{REF} este tensiunea de referință (de exemplu, 5V),
- V_{NTC} este tensiunea măsurată la bornele termistorului.

După ce rezistența NTC este calculată, se poate aplica **ecuația Steinhart-Hart** sau o formulă simplificată pentru a converti această rezistență în temperatura corespunzătoare, cum ar fi formula BETA:

$$Temperatura = \frac{1}{\frac{1}{T_0} + \frac{1}{\beta} * \ln\left(\frac{R_{NTC}}{R_{REF}}\right)} - 273,15$$

unde:

- T_0 este temperatura de referință (298.15K),
- β (BETA) este o constantă specifică termistorului,
- R_{NTC} este rezistența calculată a termistorului.

3 Condiționare semnal

Condiționarea semnalului include procese cum ar fi filtrarea, amplificarea, conversia semnalului și alte tehnici care să asigure acuratețea datelor măsurate.

Filtrul „Sare și Piper” (Salt and Pepper Noise Filter), este un filtru pentru zgomotul impulsiv, care apare ca niște puncte albe și negre dispersate în datele senzorului, de obicei cauzate de interferențe electrice sau erori în comunicație.

Filtrul de zgomot „sare și piper” este utilizat pentru a elimina aceste erori brute din semnalul achiziționat. Un filtru mediu (median) este deseori folosit pentru a detecta și elimina aceste valori anormale: Filtrul median ia un grup de valori înregistrate într-o fereastră de timp și selectează valoarea mediană, eliminând astfel influența valorilor extreme.

Acest filtru funcționează bine împotriva zgomotului de tip sare și piper, deoarece ignoră complet valorile care sunt mult mai mari sau mai mici decât media.

Filtrul de Mediere Ponderată (Weighted Moving Average Filter) este utilizat pentru a reduce zgomotul din semnalul achiziționat și pentru a netezi variațiile bruște sau fluctuațiile semnalului. Acesta calculează media unui grup de mostre (date de la senzor), dar atribuie fiecărei mostre o greutate (pondere) diferită, în funcție de cât de recentă este mostra (cu cât mai recent cu atât mai mare este ponderea).

Materiale și metode:

Pentru a putea efectua această lucrare de laborator și ansambla un circuit fizic, au fost necesare următoarele materiale:

- Microcontroler Arduino MEGA 2560;
- Senzor de distanță *Ultrasonic*;
- Senzor NTC: sensor analog de temperatură (Negative Temperature Coefficient) thermistor;
- Rezistor de 10k Ω ;
- Fire de conexiune.

MCU este componenta centrală a sistemului, responsabilă pentru colectarea și procesarea datelor de la senzori. Aceasta este folosită pentru a converti valorile analogice și pentru a controla senzorii. **Senzorul NTC** este un termistor, adică un rezistor a cărui rezistență variază în funcție de temperatură. Cu cât temperatura crește, cu atât rezistența scade. Senzorul NTC este parte a unui divizor de tensiune împreună cu un rezistor fix de 10k Ω . **Rezistorul** este conectat în serie cu senzorul NTC pentru a forma un **divizor de tensiune**. Divizorul de tensiune este esențial pentru măsurarea variabilei de rezistență a senzorului NTC prin măsurarea tensiunii de ieșire a acestui circuit. **Senzorul ultrasonic HC-SR04** măsoară distanța trimițând unde ultrasonice și detectând timpul de întoarcere a undelor reflectate de un obiect. Este conectat la pinii digitali 7 și 8 ai plăcii Arduino pentru a trimite semnalul de declanșare (TRIG) și pentru a recepționa semnalul de ecou (ECHO). (vezi Figura 1)

Senzorul NTC:

- Pinul OUT al senzorului NTC este conectat la pinul analogic A0 al plăcii Arduino Mega. Acest pin citește tensiunea rezultată din divizorul de tensiune.
- Pinul GND al senzorului NTC este conectat la GND-ul plăcii Arduino, oferind referință de tensiune.
- Pinul VCC al senzorului NTC este conectat la un capăt al rezistorului de 1k Ω , care, la rândul său, este conectat la pinul de 5V al Arduino, alimentând circuitul.

Senzorul ultrasonic HC-SR04:

- Pinul GND al senzorului HC-SR04 este conectat la GND-ul plăcii Arduino Mega, oferind referință comună de tensiune.
- Pinul VCC al senzorului este conectat la pinul de 5V al Arduino Mega pentru alimentare.
- Pinul TRIG este conectat la pinul digital 8, care trimite semnalul de declanșare pentru unda ultrasonică.
- Pinul ECHO este conectat la pinul digital 7, pentru a primi semnalul de întoarcere și a calcula distanța.


```

    double resistance = calculate_ntc_resistance(voltage);
    dtostrf(resistance, 5, 2, buffer);
    printf("Resistance: %sOhms\n", buffer);

    double temperature = convert_to_Celsius(resistance);
    dtostrf(temperature, 5, 2, buffer);
    printf("Temperature: %s°C\n", buffer);
}

void Ultrasonic_task()
{
    int distance_CM = get_distance_CM();
    dtostrf(distance_CM, 5, 2, buffer);
    printf("Distance in CM: %s\n", buffer);

    int distance_inches = get_distance_inches();
    dtostrf(distance_inches, 5, 2, buffer);
    printf("Distance in inches: %s\n", buffer);
}

```

Sarcina 2.

Sarcina doi combină deja condiționarea semnalului prin adăugarea de filtru: sare și piper, filtrul de mediere ponderată.

Filtrul Sare și Piper:

```

double saltAndPapperFilter(double newValue) {
    adcValues[index] = newValue;

    int sortedValues[SAMPLES_NUM];
    memcpy(sortedValues, adcValues, sizeof(adcValues));

    // Folosim o sortare mai eficientă (Insertion Sort pentru liste mici)
    for (int i = 1; i < SAMPLES_NUM; i++) {
        int key = sortedValues[i];
        int j = i - 1;
        while (j >= 0 && sortedValues[j] > key) {
            sortedValues[j + 1] = sortedValues[j];
            j = j - 1;
        }
        sortedValues[j + 1] = key;
    }

    showValues();
    move();
}

```

```

    return sortedValues[SAMPLES_NUM / 2];
}

```

, unde:

adcValues – este un array ce descrie un grup de valori recente;

SAMPLES_NUM – numărul de valori ce se ia în calcul;

sortedValues – array-ul de valori sortat;

newValue – valoarea care este primită cel mai recent;

showValues() – funcție care afișează valorile recente;

move() – funcția care resetează valorile curente.

Filtrul Sare și Piper:

```

double weightedMovingAverageFilter(double newValue) {
    double weightedAverage = 0;
    double weightSum = 0;
    adcValues[index] = newValue;

    for (int i = 0; i < SAMPLES_NUM; i++) {
        weightedAverage += adcValues[i] * alpha[i];
        weightSum += alpha[i];
    }

    showValues();
    move();

    return weightedAverage / weightSum;
}

```

, unde:

weightedAverage - variabila care acumulează suma ponderată a valorilor din array-ul adcValues;

weightedSum - suma totală a ponderilor din array-ul alpha;

alpha - array de valori care reprezintă ponderile fiecărei mostre din array-ul adcValues.

Rezultate obținute:

```
Voltage: 2.56V
Resistance: 9560.230hms
Temperature: 23.99°C

Distance in CM: 105.00
Distance in inches: 41.00
```

Figura 2 – Afișare Serial Monitor

```
Voltage: 2.56V
Resistance: 9560.230hms
[ 23 23 23 23 23 ]Temperature: 23.00°C

Voltage: 2.35V
Resistance: 11268.190hms
[ 23 23 23 23 27 ]Temperature: 24.56°C

Voltage: 2.35V
Resistance: 11268.190hms
[ 23 23 23 27 27 ]Temperature: 25.67°C

Voltage: 1.44V
Resistance: 24677.970hms
[ 23 23 27 27 46 ]Temperature: 33.72°C

Voltage: 1.44V
Resistance: 24677.970hms
[ 23 27 27 46 46 ]Temperature: 39.44°C

Voltage: 3.66V
Resistance: 3676.470hms
[ 27 27 46 46 4 ]Temperature: 26.50°C
```

a) Mediere ponderată

```
Voltage: 0.56V
Resistance: 78956.520hms
[ 0 23 23 30 80 ]Temperature: 23.00°C

Voltage: 3.22V
Resistance: 5523.520hms
[ 23 23 30 80 12 ]Temperature: 23.00°C

Voltage: 1.36V
Resistance: 26798.560hms
[ 23 30 80 12 48 ]Temperature: 30.00°C

Voltage: 1.72V
Resistance: 19062.500hms
[ 30 80 12 48 40 ]Temperature: 40.00°C

Voltage: 1.36V
Resistance: 26798.560hms
[ 80 12 48 40 48 ]Temperature: 48.00°C

Voltage: 1.36V
Resistance: 26798.560hms
[ 12 48 40 48 48 ]Temperature: 48.00°C
```

b) Sare și piper

Figura 2 – Afișare pe Serial Monitor după filtre

CONCLUZIE

În această lucrare de laborator, am explorat metode fundamentale de achiziție, prelucrare și filtrare a semnalelor senzorilor analogici, cu accent pe senzorii NTC și pe măsurarea distanței utilizând senzorul ultrasonic. Prin utilizarea unui microcontroller Arduino Mega, am implementat și testat filtre pentru prelucrarea semnalelor brute, precum filtrul sare și piper și filtrul de mediere ponderată.

Filtrul sare și piper a demonstrat eficiența în eliminarea valorilor extreme din semnalele achiziționate, iar filtrul de mediere ponderată a permis o ajustare fină a contribuției fiecărei mostre la rezultatul final, îmbunătățind stabilitatea și precizia măsurărilor. Aceste tehnici de condiționare a semnalului sunt esențiale în aplicații reale, unde zgomotul și variabilitatea pot afecta acuratețea senzorilor.

În concluzie, am reușit să înțelegem și să aplicăm principii esențiale de achiziție și condiționare a semnalului, obținând măsurători mai precise și mai stabile în medii variabile. Aceste tehnici vor putea fi utilizate cu succes în proiecte viitoare care implică senzori și microcontrollere.

BIBLIOGRAFII

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – 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/Laborator3](https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator3) - 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 available
    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

main.cpp

```

#include <Arduino.h>
#include "NTC.h"
#include "ultrasonic.h"
#include "filters.h"

void NTC_task();
void Ultrasonic_task();

char buffer[20];

void setup()
{
    Serial.begin(9600);
    setup_ultrasonic();

    printf("All done. Start meassuring:\n");
}

void loop()
{
    printf("\n");
    NTC_task();
    printf("\n");
    // Ultrasonic_task();

    delay(2000);
}

void NTC_task()
{

```

```

    int16_t adcValue = analogRead(ANALOG_PIN);

    double voltage = adc_to_Voltage(adcValue);
    dtostrf(voltage, 5, 2, buffer);
    printf("Voltage: %sV\n", buffer);

    double resistance = calculate_ntc_resistance(voltage);
    dtostrf(resistance, 5, 2, buffer);
    printf("Resistance: %sOhms\n", buffer);

    double temperature = convert_to_Celsius(resistance);

    // double filteredMedianValue = saltAndPapperFilter(temperature);
    double filteredMedianValue = weightedMovingAverageFilter(temperature);

    dtostrf(filteredMedianValue, 5, 2, buffer);
    printf("Temperature: %s°C\n", buffer);
}

void Ultrasonic_task()
{
    int distance_CM = get_distance_CM();
    dtostrf(distance_CM, 5, 2, buffer);
    printf("Distance in CM: %s\n", buffer);

    int distance_inches = get_distance_inches();
    dtostrf(distance_inches, 5, 2, buffer);
    printf("Distance in inches: %s\n", buffer);
}

```

NTC.h

```

#ifndef NTC_H
#define NTC_H

#include <stdint.h>
#include <math.h>

#define BETA 3950 // valoarea BETA a termistorului NTC
#define ANALOG_PIN A0 //pinul analogic de conexiune
#define V_REF 5.0 // tensiunea de referință utilizată de ADC
#define R_REF 10000.0 // rezistența de referință utilizată în divizorul de tensiune
#define ADC_MAX 1023.0 // valoarea maximă pe care o poate avea un ADC de 10 biți

double get_temperature_directly(int16_t adcValue);
double adc_to_Voltage(int16_t adcValue);
double calculate_ntc_resistance(double voltage);
double convert_to_Kelvin(double ntcResistance);
double convert_to_Celsius(double ntcResistance);

#endif //NTC_h

```

NTC.cpp

```
#include "NTC.h"

double get_temperature_directly(int16_t adcValue){
    return 1 / (log(1 / (ADC_MAX / adcValue - 1))
        / BETA + 1.0 / 298.15) - 273.15;
}

double adc_to_Voltage(int16_t adcValue){
    return (adcValue / ADC_MAX) * V_REF;
}

double calculate_ntc_resistance(double voltage) {
    return R_REF * (V_REF / voltage - 1);
}

double convert_to_Kelvin(double ntcResistance) {
    return 1.0 / (log( R_REF / ntcResistance ) / BETA + 1.0 / 298.15);
}

double convert_to_Celsius(double ntcResistance) {
    return convert_to_Kelvin(ntcResistance) - 273.15;
}
```

ultrasonic.h

```
#ifndef ULTRASONIC_H
#define ULTRASONIC_H

#include <Arduino.h>

#define PIN_TRIG 8
#define PIN_ECHO 7

void setup_ultrasonic();
int16_t get_distance_CM();
int16_t get_distance_inches();

#endif //ULTRASONIC_H
```

ultrasonic.cpp

```
#include "ultrasonic.h"

int16_t duration = 0;

void set_trig_time(int16_t micro_seconds);

void setup_ultrasonic(){
    pinMode(PIN_TRIG, OUTPUT);
    pinMode(PIN_ECHO, INPUT);
}
```

```

void get_duration(){
    set_trig_time(10);
    duration = pulseIn(PIN_ECHO, HIGH);
}

int16_t get_distance_CM(){
    get_duration();
    return duration / 58;
}

int16_t get_distance_inches(){
    get_duration();
    return duration / 148;
}

void set_trig_time(int16_t micro_seconds){
    digitalWrite(PIN_TRIG, HIGH);
    delayMicroseconds(micro_seconds);
    digitalWrite(PIN_TRIG, LOW);
}

```

filters.h

```

#ifndef FILTERS_H
#define FILTERS_H

#include <string.h>
#include <Arduino.h>

#define SAMPLES_NUM 5

typedef struct {
    int values[SAMPLES_NUM];    // Valorile mostrelor
    int index;                  // Indexul curent pentru valori
    double alpha[SAMPLES_NUM]; // Ponderi pentru weighted moving average (dacă e necesar)
} FilterData;

double saltAndPapperFilter(double newValue);
double weightedMovingAverageFilter(double newValue);

#endif //FILTERS_H

```

filters.cpp

```

#include "filters.h"

void showValues(int* values);
void move(int* values);

FilterData saltAndPepperData = {{0}, SAMPLES_NUM - 1, {1}};
FilterData weightedAvgData = {{0}, SAMPLES_NUM - 1, {0.1, 0.2, 0.3, 0.5, 0.7}};

```



```

double saltAndPepperFilter(double newValue) {
    saltAndPepperData.values[saltAndPepperData.index] = newValue;

    int sortedValues[SAMPLES_NUM];
    memcpy(sortedValues, saltAndPepperData.values, sizeof(saltAndPepperData.values));

    // Folosim o sortare mai eficientă (Insertion Sort pentru liste mici)
    for (int i = 1; i < SAMPLES_NUM; i++) {
        int key = sortedValues[i];
        int j = i - 1;
        while (j >= 0 && sortedValues[j] > key) {
            sortedValues[j + 1] = sortedValues[j];
            j = j - 1;
        }
        sortedValues[j + 1] = key;
    }

    showValues(saltAndPepperData.values);
    move(saltAndPepperData.values);

    return sortedValues[SAMPLES_NUM / 2];
}

double weightedMovingAverageFilter(double newValue) {
    double weightedAverage = 0;
    double weightSum = 0;
    weightedAvgData.values[weightedAvgData.index] = newValue;

    for (int i = 0; i < SAMPLES_NUM; i++) {
        weightedAverage += weightedAvgData.values[i] * weightedAvgData.alpha[i];
        weightSum += weightedAvgData.alpha[i];
    }

    showValues(weightedAvgData.values);
    move(weightedAvgData.values);

    return weightedAverage / weightSum;
}

void showValues(int* values) {
    printf("[ ");
    for (int i = 0; i < SAMPLES_NUM; i++) {
        printf("%d ", values[i]);
    }
    printf("]");
}

void move(int* values) {
    for (int i = 0; i < SAMPLES_NUM - 1; i++) {
        values[i] = values[i + 1];
    }
}

```