# Distance Measurement

## Introducere

Dispozitiv care masoara distanta folosind senzor de ultrasunete.

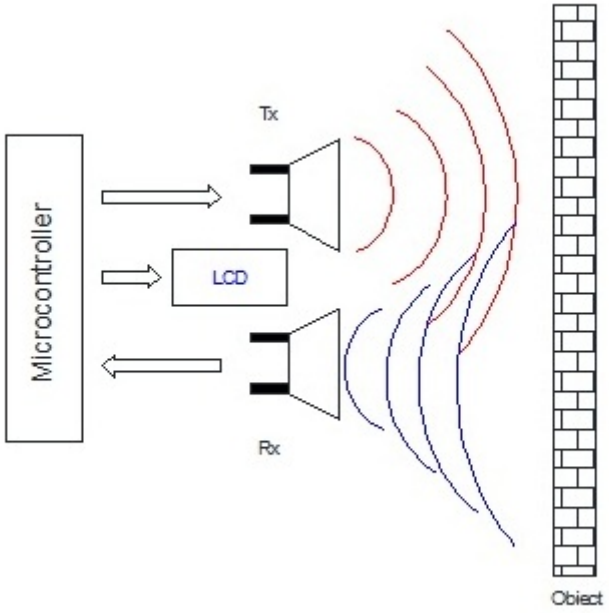
Prezentarea pe scurt a proiectului :

- ce face - masoara distanta precis
- care este scopul lui - poate fi folosit la masurarea adancimii apei; mai poate fi folosit la masinute de jucarie pentru a localiza obstacolele
- care a fost ideea de la care aţi pornit
- de ce credeţi că este util pentru alţii şi pentru voi

## Descriere generală

Senzorul emite ultrasunete, iar in momentul in care undele transmise intalnesc un obstacol, ele sunt transmise inapoi la senzor, folosind o tehnica numita ECHO. Trebuie calculat timpul de propagare al sunetelor, de la sursa la obstacol si inapoi.

Distanta se calculeaza cu ajutorul vitezei sunetului.(Distanta = Timp / Viteza sunetului)




Senzorul ultrasonic HC-SR04 poate masura distante intre 2cm si 400cm cu precizie care poate ajunge la 3mm. Fiecare modul include un transmitator ultrasonic, un receptor si un circuit de comanda. Pentru functionare senzorul are nevoie de 4 pini VCC(Power), Trig(Trigger), Echo(Primire) si GND.

Caracteristici senzor:

- -tensiune de operare: DC 5V
- -curentul de functionare: 15 mA
- -unghi de functionare: 15 grade

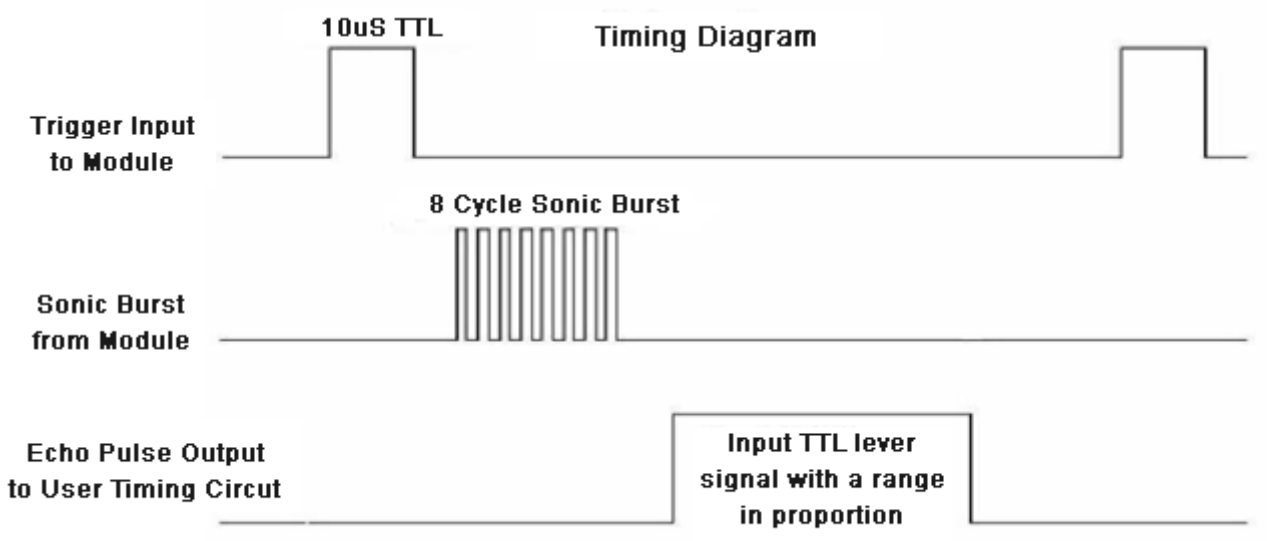
Principiul de functionare:

- 1.Un semnal High este trimis pentru 10us, folosind Trigger.
- 2.Modulul trimite 8 semnale automat de 40kHz, si dupa verifica daca semnalul este receptionat sau nu.
- 3.Daca semnalul este receptionat, atunci este prin nivelul High. Timpul cat dureaza High este reprezentat de decalajul de timp dintre trimiterea si receptionarea semnalului.

Diagrama de timp - senzor HC-SR04

- Senzorul furnizeaza un semnal de output proportional cu distanta masurata pe baza ecoului.Senzorul genereaza un sunet (Trigger) si dupa asteapta receptia undelor. Viteza sunetului este 220 m/s, iar timpul necesar ecoului pentru a ajunge la sursa furnizeaza un semnal de output proportional cu distanta.
- La inceput este necesara initierea senzorului pentru a masura distanta, un semnal logic High, la pinul Trigger al senzorului pentru mai mult de 10uS, dupa aceea un sunet este transmis de senzor, dupa un ecou, senzorul furnizeaza un semnal la pinul de iesire ale carei latimi este proportionala cu distanta dintre sursa si obstacol.
- Distanta se calculeaza ca, distanta(cm)= latimea semnalului de iesire(in uS)/58.

- Latimea semnalului poate fi luata ca multiplu de uS (micro secunde sau 10^-6)




O schemă bloc cu toate modulele proiectului vostru, atât software cât şi hardware însoţită de o descriere a acestora precum şi a modului în care interacţionează.

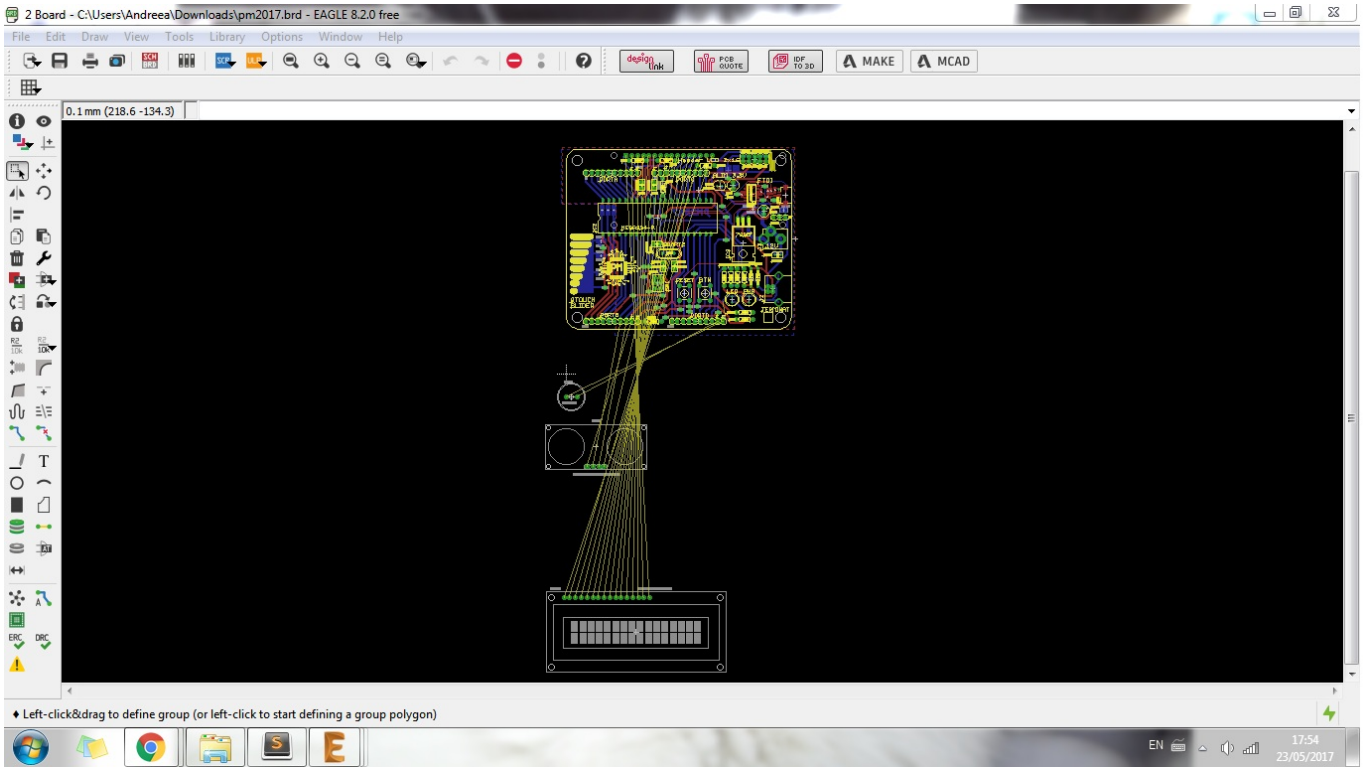
# Hardware Design

- listă de piese :
  - placa de baza PM
  - ATMEGA324
  - sezor ultrasonic HC-SR04
  - LCD 16×2
  - rezistente
  - condensatori
  - cablu USB 2.0 imprimanta
  - butoane
  - buzzer
  - LED-uri

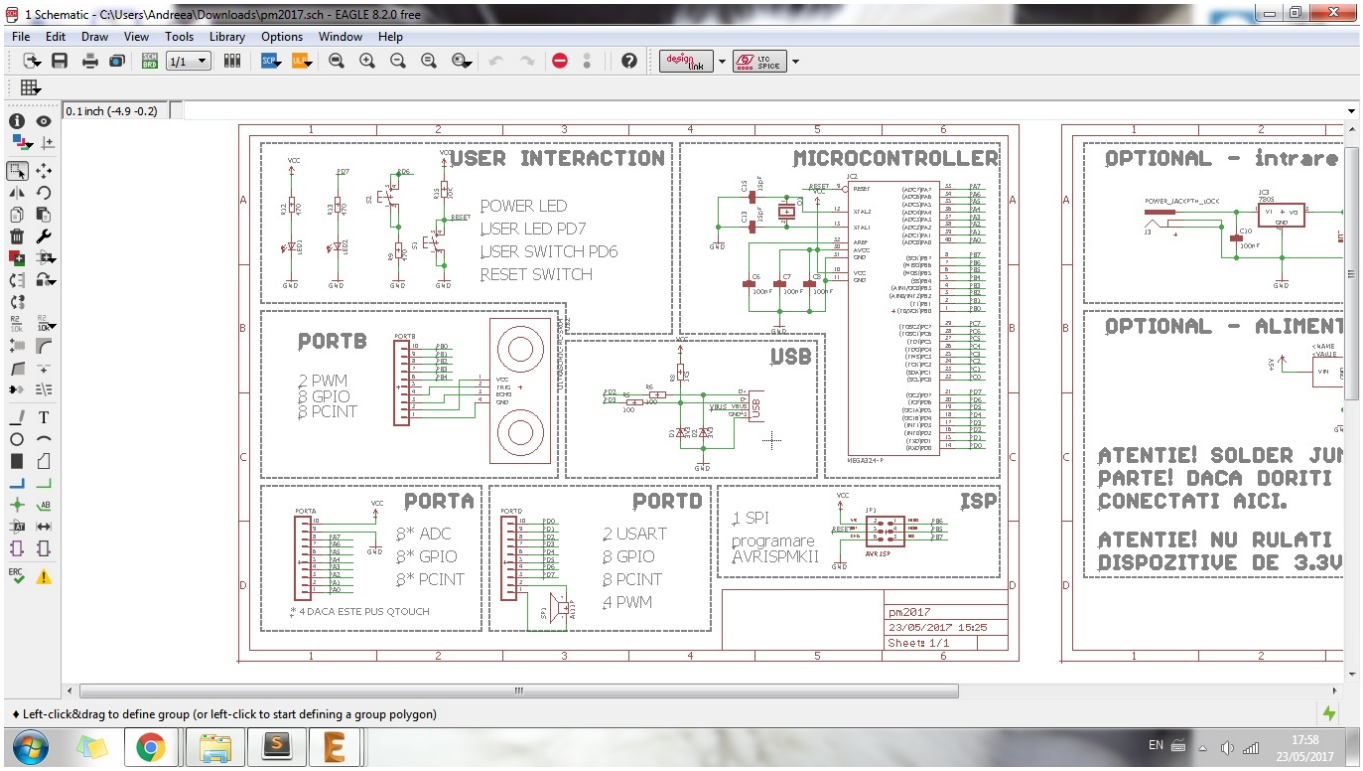

Pini senzor:

```
Vcc: +5VDC
Trig:Trigger(INPUT)
Echo:Echo(OUTPUT)
GND: GND
```

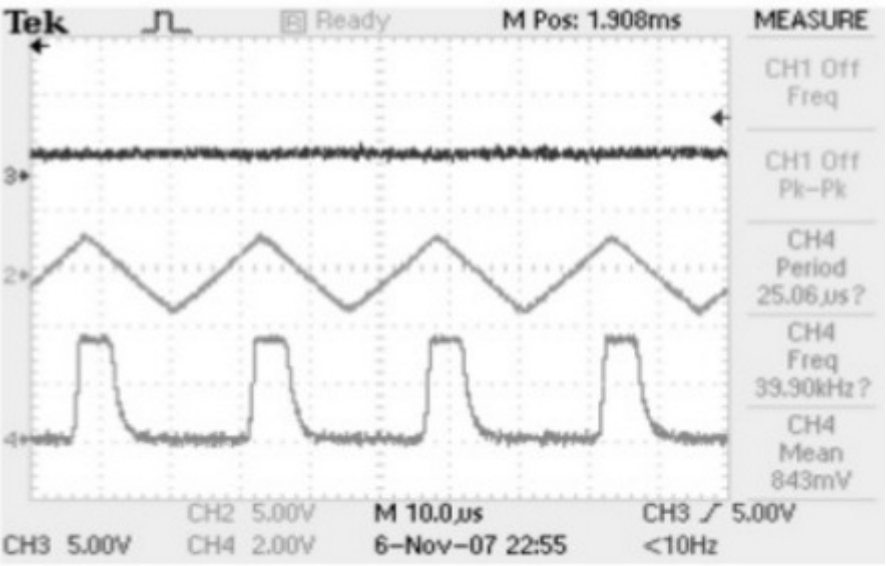
- scheme electrice




*




- Semnalul in sectiunea receiver-ului




Channel 3 : Output receiver amplifier
Channel 4 : Input pulses to the microcontroller.

## Rezultatele simulării

- Microcontroller-ul asteapta primirea impulsurilor pentru o durata maxima de 12 milisecunde. Acesta este timpul necesar undelor ultrasonice pentru a parcuge distanta maxima de 4 metri. Daca nu primeste impulsul in acest interval de timp se considera ca nu exista niciun obstacol sau ca obiectul este inafara razei.
- Odata ce impulsurile au fost primite, microcontroller-ul numara 10 impulsuri in intervale de timp de 25 microsecunde, doar atunci masuratoarea este considerata valida.

## Software Design

```
--lcd.c--

#include "lcd.h"

// Initializare LCD considerand o interfatare cu 4 pini de date.
// Trebuie apelata inainte de a face orice operatie cu LCD-ul.

void LCD_init(void)
{
    // setam pinii de date ca pini de iesire
    LcdDATA_DDR |= LCD_DATA_MASK;
    // setam pinii de comenzi ca pini de iesire
    LcdCTRL_DDR |= LCD_CTRL_MASK;

    // intram in modul comenzi: RS - low, RW - low
    LCD_RS_LOW();
    LCD_RW_LOW();

    // asteptam timpul de set-up pentru RS si RW
    LCD_DELAY();

    // initializarea pentru 4 fire de date necesita transferarea lui 0100 ->
D4-D7

    // E pe high incepe transferul
    LCD_ENABLE();

    // scriem cei mai semnificativi 4 biti pe firele de date, fara a
modifica alti biti
    LcdDATA_PORT = (LcdDATA_PORT & ~LCD_DATA_MASK) |
                   (0 << LcdD7) |
                   (0 << LcdD6) |
                   (1 << LcdD5) |
                   (0 << LcdD4);

    // asteptam timpul minim de set-up
    LCD_DELAY();
    // E pe low termina transferul
    LCD_DISABLE();
    // asteptam timpul minim pana la urmatorul E (contine timpul de hold)
    LCD_DELAY();

    // transmitem configuratia initiala
    // 4 biti de date, 2 linii, font 8x5
    LCD_writeInstr(LCD_INSTR_4wire);
```

```
    // display pornit, cursor afisat, blink activat
    LCD_writeInstr(LCD_INSTR_displayConfig);
    // auto-increment, fara shiftare
    LCD_writeInstr(LCD_INSTR_incNoShift);
    // sterge display
    LCD_writeInstr(LCD_INSTR_clearDisplay);
}

// Executa secventa de citire a unui octet de date de la LCD.
// Pentru interfatarea cu 4 pini de date sunt necesare 2 transferuri.

uint8_t LCD_read(void)
{
    // setam pinii de date ca intrari pentru a putea primit date de la LCD
    LcdDATA_DDR &= ~LCD_DATA_MASK;
    // dezactivam rezistentele de pull-up pentru pinii de date
    LcdDATA_PORT &= ~LCD_DATA_MASK;

    // stocheaza valoarea care va fi citita
    uint8_t data = 0;

    // transferul primilor 4 biti de la LCD

    LCD_ENABLE();
    // asteptam timpul minim de set-up
    LCD_DELAY();
    // citim de pe pinii de date cei mai semnificativi 4 biti
    data |= (((LcdDATA_PIN >> LcdD7) & 0x01) << 7) |
            (((LcdDATA_PIN >> LcdD6) & 0x01) << 6) |
            (((LcdDATA_PIN >> LcdD5) & 0x01) << 5) |
            (((LcdDATA_PIN >> LcdD4) & 0x01) << 4);
    LCD_DISABLE();
    // asteptam timpul minim pana la urmatorul E
    LCD_DELAY();

    // transferul ultimilor 4 biti de la LCD

    LCD_ENABLE();
    // asteptam timpul minim de set-up
    LCD_DELAY();
    // citim de pe pinii de date cei mai putin semnificativi 4 biti
    data |= (((LcdDATA_PIN >> LcdD7) & 0x01) << 3) |
            (((LcdDATA_PIN >> LcdD6) & 0x01) << 2) |
            (((LcdDATA_PIN >> LcdD5) & 0x01) << 1) |
            (((LcdDATA_PIN >> LcdD4) & 0x01) << 0);
    LCD_DISABLE();
    // asteptam timpul minim pana la urmatorul E
    LCD_DELAY();

    // setam pinii de date inapoi ca iesiri
    LcdDATA_DDR |= LCD_DATA_MASK;
```

```
    // returnam octetul citit
    return data;
}

// Citeste starea LCD-ului (contine busy flag).
uint8_t LCD_readStatus(void)
{
    // intram in modul citire stare (busy flag + addresa curenta)
    LCD_RS_LOW();
    LCD_RW_HIGH();

    // asteptam timpul de set-up pentru RS si RW
    LCD_DELAY();

    // citim si returnam datele
    return LCD_read();
}

// Citeste un octet din ultima memorie folosita (DDRAM sau CGRAM).
uint8_t LCD_readData(void)
{
    // intram in modul citire memorie
    LCD_RS_HIGH();
    LCD_RW_HIGH();

    // asteptam timpul de set-up pentru RS si RW
    LCD_DELAY();

    // citim si returnam datele
    return LCD_read();
}

// Returneaza starea LCD-ului: 1 - busy, 0 - available
uint8_t LCD_isBusy(void)
{
    // busy flag este bitul 7: 1 -> busy, 0 -> available
    return (LCD_readStatus() & (1 << 7)) != 0;
}

// Asteapta pana cand LCD-ul devine disponibil pentru o noua comanda.
void LCD_waitNotBusy(void)
{
    while(LCD_isBusy());
}

// Executa secventa de trimitere a unui octet de date catre LCD.
// Pentru interfatarea cu 4 pini de date sunt necesare 2 transferuri.

void LCD_write(uint8_t data)
{
```

```
    // transferul primilor 4 biti catre LCD

    LCD_ENABLE();
    // scriem pe pinii de date cei mai semnificativi 4 biti, fara a modifica
alti biti in registru
    LcdDATA_PORT = (LcdDATA_PORT & ~LCD_DATA_MASK) |
                   (((data >> 7) & 0x01) << LcdD7) |
                   (((data >> 6) & 0x01) << LcdD6) |
                   (((data >> 5) & 0x01) << LcdD5) |
                   (((data >> 4) & 0x01) << LcdD4);
    // asteptam timpul minim de set-up
    LCD_DELAY();
    LCD_DISABLE();
    // asteptam timpul minim pana la urmatorul E (contine timpul de hold)
    LCD_DELAY();

    // transferul ultimilor 4 biti catre LCD

    LCD_ENABLE();
    // scriem pe pinii de date cei mai putin semnificativi 4 biti, fara a
modifica alti biti in registru
    LcdDATA_PORT = (LcdDATA_PORT & ~LCD_DATA_MASK) |
                   (((data >> 3) & 0x01) << LcdD7) |
                   (((data >> 2) & 0x01) << LcdD6) |
                   (((data >> 1) & 0x01) << LcdD5) |
                   (((data >> 0) & 0x01) << LcdD4);
    // asteptam timpul minim de set-up
    LCD_DELAY();
    LCD_DISABLE();
    // asteptam timpul minim pana la urmatorul E (contine timpul de hold)
    LCD_DELAY();
}

// Trimite o instructiune de control catre LCD.
void LCD_writeInstr(uint8_t instr)
{
    // asteptam ca LCD-ul sa poata accepta instructiuni
    LCD_waitNotBusy();

    // intram in modul comenzi: RS - low, RW - low
    LCD_RS_LOW();
    LCD_RW_LOW();

    // asteptam timpul de set-up pentru RS si RW
    LCD_DELAY();

    // trimitem datele
    LCD_write(instr);
}

// Trimite o instructiune de scriere date catre LCD.
```

```
void LCD_writeData(uint8_t data)
{
    // asteptam ca LCD-ul sa poata accepta instructiuni
    LCD_waitNotBusy();

    // intram in modul scriere date
    LCD_RS_HIGH();
    LCD_RW_LOW();

    // asteptam timpul de set-up pentru RS si RW
    LCD_DELAY();

    // trimitem datele
    LCD_write(data);
}

// Afiseaza caracterul pe LCD la adresa curenta.

void LCD_putChar(char c)
{
    LCD_writeData(c);
}

// Afiseaza caracterul pe LCD la adresa primita.

void LCD_putCharAt(uint8_t addr, char c)
{
    /* TODO task 1 LCD */

    LCD_writeInstr(LCD_INSTR_DDRAM + addr);
    LCD_putChar(c);
}


// Afiseaza string-ul pe LCD incepand de la adresa curenta.

void LCD_print(const char* msg)
{
    while(*msg)
        LCD_putChar(*msg++);
}

// Afiseaza string-ul pe LCD incepand de la adresa primita.

void LCD_printAt(uint8_t addr, const char* msg)
{
    /* TODO task 1 LCD */

    LCD_writeInstr(LCD_INSTR_DDRAM | addr);
    LCD_print(msg);
```

```
}


--main.c--


#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000L
#include <util/delay.h>
#include <stdlib.h>

static volatile int pulse = 0;
static volatile int i = 0;

ISR(INT2_vect)
{
        if (i==1)
        {
                TCCR1B=0;
                pulse=TCNT1;
                TCNT1=0;
                i=0;
        }
        if (i==0)
        {
                TCCR1B|=(1<<CS10);
                i=1;
        }
}

int main(void)
{

        LCD_init();
        DDRD |= _BV(PD0);
        PORTD &= ~_BV(PD0);
        TCCR1A = 0;

        int16_t COUNTA = 0;
        char SHOWA [16];

        EICRA |= _BV(ISC20);
        EIMSK |= _BV(INT2);
        char temp[20];
    /* Replace with your application code */
    while (1)
    {
                PORTD |= _BV(PD0);
                _delay_us(15);
                PORTD &= ~_BV(PD0);
```

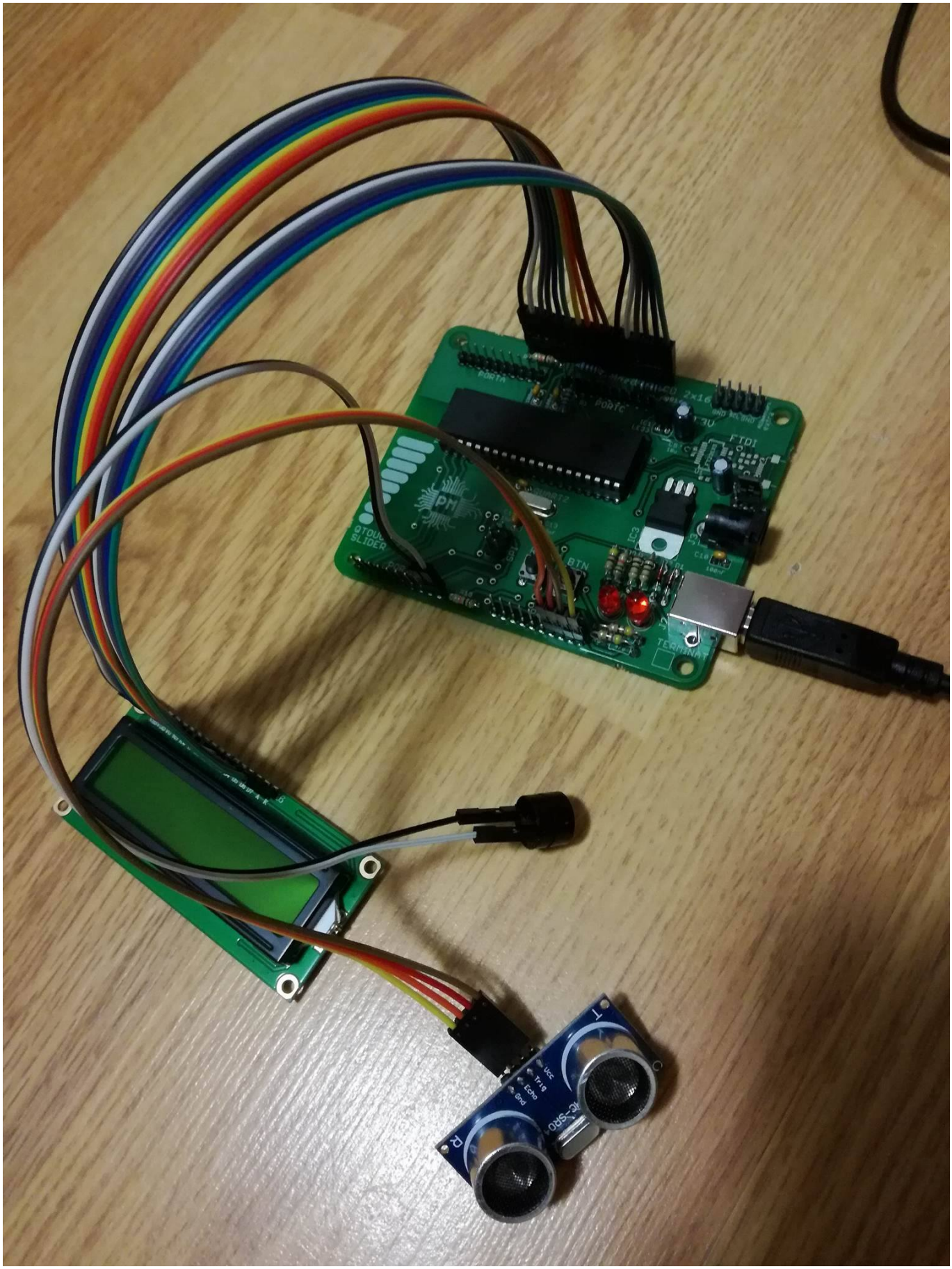
```
                COUNTA = pulse / 58;

                LCD_printAt(0, "Distanta:");
                sprintf(SHOWA, "%d", COUNTA);
                LCD_printAt(0x20, SHOWA);
    }
}
```

# Rezultate Obţinute

Dispozitivul poate masura corect pana la 4m

# Concluzii


PORTA
PORTC
FTDI
BTN
TERMINAL
Vcc
Trig
Echo
Gnd

# Download

pm2017.sch

Codul pentru proiect se afla aici : distancemeasurement.zip

# Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.

# Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

- Documentația în format PDF