**UNIVERSITATEA POLITEHICA TIMIȘOARA**

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**ANUL III – CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI – LB. ROMÂNĂ**

**SISTEME ÎNCORPORATE**

**SISTEM DE MONITORIZARE A NUMĂRULUI DE LOCURI LIBERE ÎNTR-O PARCARE**

**ECHIPA: CALAPIȘ ANA-MARIA, CRÎȘCIU EMANUEL-ILIE**

**ANUL UNIVERSITAR 2023-2024**

**TEMA PROIECTULUI**

**SISTEM DE MONITORIZARE A NUMĂRULUI DE LOCURI LIBERE ÎNTR-O PARCARE, IMPLEMENTAT CU AJUTORUL UNEI MACHETE DEMONSTRATIVE ȘI INCLUZÂND O BARIERĂ COMANDATĂ DE UN SERVOMOTOR**

**CARACTERISTICI:**

• Se vor utiliza fotodiode sau senzori de detecție a obiectelor reflectorizante („reflective object sensors”) sau senzori cu ultrasunete pentru detectarea mașinilor care intră si ies. Se pot folosi alte tipuri de senzori conform cu decizia proiectantului.

• Modalitatea de legare a senzorilor va cuprinde cât mai puține fire (se vor alege interfețe seriale, precum I2C, CAN, LIN etc).

• Se va asigura afișarea numărului de locuri libere din parcare (afișaje cu segmente sau matrice de LED-uri sau afișaj LCD).

• Numărul inițial al locurilor libere de parcare va fi prestabilit.

• Aplicația va dispune de 2 LED-uri: unul din ele va fi pornit atâta vreme cât mai există locuri libere, al doilea va fi pornit când nu vor mai fi locuri disponibile în cadrul parcării. Nu este posibilă pornirea concomitentă a celor două LED-uri.

• Demonstrarea practică a proiectului va include o machetă cu un anumit număr de locuri de parcare delimitate, iar la intrarea în parcare va fi montată o barieră comandată de un servomotor. Accesul unui automobil în parcare este permis prin ridicarea barierei doar dacă mai este cel puțin un loc liber disponibil. La fiecare ridicare a barierei pentru intrare în parcare, numărul de locuri libere disponibile va fi decrementat, iar la fiecare ridicare a barierei pentru ieșire din parcare, numărul de locuri libere va fi incrementat.

• Codul sursă va trebui să țină cont în redactare de constrângerile specifice care pot apărea în cadrul unui sistem încorporat.

**DESCRIEREA SISTEMULUI:**

Principalul obiectiv al proiectului este automatizarea accesului într-o parcare, astfel că a trebuit să găsim o soluție pentru a comanda bariera doar atunci când situația reală o cere. Prima problemă ce s-a pus a fost cum putem face ca bariera să se deschidă și să se închidă. Astfel că am utilizat un servomotor pentru a o acționa. Dar acest servomotor trebuie comandat numai la momentul potrivit. Această problemă este rezolvată de cei doi senzori ultrasonici plasați de o parte și de alta a barierei, care detectează mașina ce se apropie de barieră.

Dat fiind faptul că utilizăm o singură barieră, a trebuit să ne asigurăm că ea este folosită în mod corect. Mai exact, atunci când o mașină dorește să intre în parcare, primul senzor care detectează o apropiere este cel îndreptat spre drumul de acces către parcare. Iar atunci când o mașină dorește să iasă din parcare, primul senzor care detectează o apropiere este cel îndreptat spre drumul ce duce spre interiorul parcării. Deci, din ordinea de activare a senzorilor ne putem da seama dacă se dorește ridicarea barierei pentru intrare, respectiv pentru ieșire. Lucrul acesta este important pentru a ști dacă incrementăm sau decrementăm contorul ce indică locurile de parcare libere.

Pentru a comunica cu mediul exterior în ceea ce privește situația numărului de locuri libere, am ales să folosim un ecran LCD pe care să afișăm acest număr. Afișajul este însoțit și de confirmarea unui LED verde ce stă aprins cât timp în parcare există cel puțin un loc liber. Atunci când numărul de locuri libere ajunge la 0, LED-ul verde se va stinge și LED-ul roșu se va aprinde, indicând astfel faptul că parcarea este ocupată în întregime.

Am tratat și situațiile conflictuale în care două mașini solicită simultan ridicarea barierei, aflându-se pe părți opuse ale acesteia. În acest caz am folosit ecranul LCD pentru a-i cere mașinii ce dorește să intre în parcare să dea cu spatele până când mașina ce dorește să iasă trece de barieră. Cu alte cuvinte, am prioritizat ieșirea din parcare pentru a crea un loc liber în plus în aceste situații.

**COMPONENTELE UTILIZATE:**

* Placa de dezvoltare Arduino UNO R3 cu microcontrolerul ATMega328P
* Senzori ultrasonici HC-SR04
* Servomotor MG996R
* Ecran LCD 1602 cu modul I2C
* LED-uri roșii și verzi de 5mm
* Breadboard 400 puncte
* Rezistențe 220 Ohmi
* Fire jumper tată-tată și mamă-tată de diverse lungimi

**DESCRIEREA PLĂCII DE DEZVOLTARE ARDUINO UNO R3**

A computer diagram of a circuit board

Description automatically generated

**CARACTERISTICI GENERALE:**

Placa de dezvoltare Arduino UNO R3 este folosită pentru familiarizarea cu lumea electronicii și a codului. Este o placă versatilă ce se utilizează atât în scopuri educaționale, cât și în industrie. Placa este echipată cu microcontrolerul ATMega328P, 14 pini digitali de intrare / ieșire, 6 intrări analogice, conexiuni USB, header ICSP și buton de reset.

Se recomandă ca placa să opereze între -40°C și 85°C. Placa poate fi alimentată prin intermediul unui cablu USB-A la USB-B direct din computer (5V) sau direct cu ajutorul unui cablu de alimentare introdus în mufa jack de alimentare (6-20V).

Topologia plăcii, cu vedere de sus:

A circuit board with many small components

Description automatically generated with medium confidence

A table with text and images

Description automatically generated with medium confidence

**TIPUL DE MICROCONTROLER ȘI PARTICULARITĂȚILE ACESTUIA:**

Placa de dezvoltare Arduino UNO R3 utilizează un microcontroler ATMega328P, împreună cu microcontrolerul ATMega16U2, ce are rolul de a gestiona comunicarea USB dintre ATMega328P și computer, cu scopul permiterii programării Arduino-ului.

Microcontrolerul ATMega328P prezintă următoarele caracteristici:

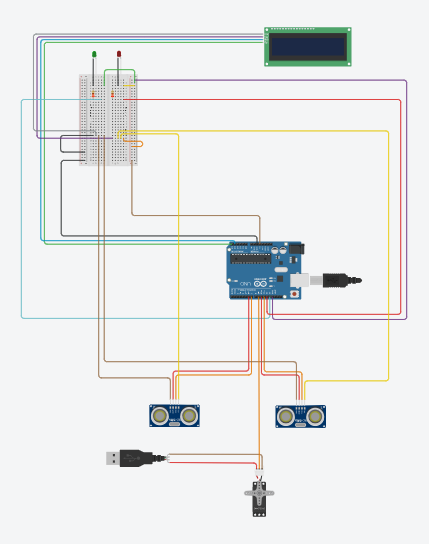
* Memoria:
  + CPU AVR cu frecvența de până la 16MHz
  + Flash de 32KB
  + SRAM de 2KB
  + EEPROM de 1KB
* Securitate
  + Power On Reset (POR)
  + Brown Out Detection (BOD)
* Periferice
  + 2x Timer/Numărător pe 8 biți cu un registru de perioadă și canale de comparare dedicate
  + 1x Timer/Numărător pe 16 biți cu un registru de perioadă și canale de intrare pentru capturare și comparare dedicate
  + 1x USART cu generator de baud rate fracționar și detecție start-of-frame
  + 1x controler SPI (Serial Peripheral Interface)
  + 1x controler mod dual I2C
  + 1x comparator analogic (AC) cu intrare de referință scalabilă
  + Timer Watchdog cu oscilator separat
  + 6 canale PWM
  + Întrerupere și “wake-up” la schimbarea stării unui pin
* Microcontrolerul ATMega16U2
  + Microcontroler RISC AVR pe 8 biți
  + Memoria:
    - Flash ISP de 16KB
    - EEPROM de 512B
    - SRAM de 512B
    - Interfață debugWIRE pentru debugging și programare
* Alimentare:
  + 2.7-5.5 V

**SCHEMA ELECTRICĂ A PLĂCII DE DEZVOLTARE:**

**A computer diagram of a computer

Description automatically generated with medium confidence**

**ARHITECTURA SISTEMULUI**



**SCHEMA BLOC A SISTEMULUI**

A screenshot of a computer

Description automatically generated

**DESCRIEREA MODULELOR MICROCONTROLERULUI CARE AU FOST IMPLICATE ÎN REALIZAREA PROIECTULUI**

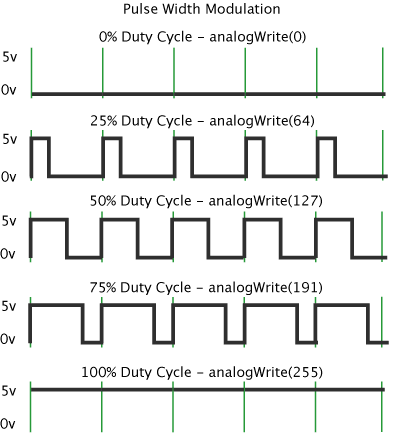
**MODULUL PWM:**

Funcția PWM este intens utilizată pentru controlul motoarelor de curent DC și de aceea toate microcontrolerele vin echipate cu o astfel de funcționalitate. Practic prin mărirea sau micșorarea factorului de umplere al unui semnal putem sa determinăm o viteză de dorit a motorului.

Modulul PWM (Pulse Width Modulation) de pe placa Arduino UNO R3 este un instrument esențial pentru controlul dispozitivelor care necesită o variație a puterii sau a intensității luminii, cum ar fi motoarele DC, LED-urile, sau alte dispozitive.

Pe Arduino UNO R3, pinii digitali 3, 5, 6, 9, 10 și 11 sunt capabili să producă semnale PWM.

La noi în proiect, modulul PWM a fost folosit pentru controlarea servomotorului ce acționează bariera.



**INTERFAȚA UART:**

Interfața serială UART (Universal Asynchronous Receiver-Transmitter) este utilizată în comunicarea dintre Arduino și un calculator. Pe placa de dezvoltare Arduino UNO R3, pinii seriali sunt 0 (RX) și 1 (TX).

Câteva caracteristici ale interfeței UART sunt: comunicarea este asincronă, datele sunt transmise bit cu bit pe un singur fir de comunicație și permite utilizatorilor să configureze diferite viteze de transfer (baud rate).

Interfața UART are mai mulți parametrii ce pot fi configurați:

- număr de biți de date: depinde de producător (între 5 și 9 biți)

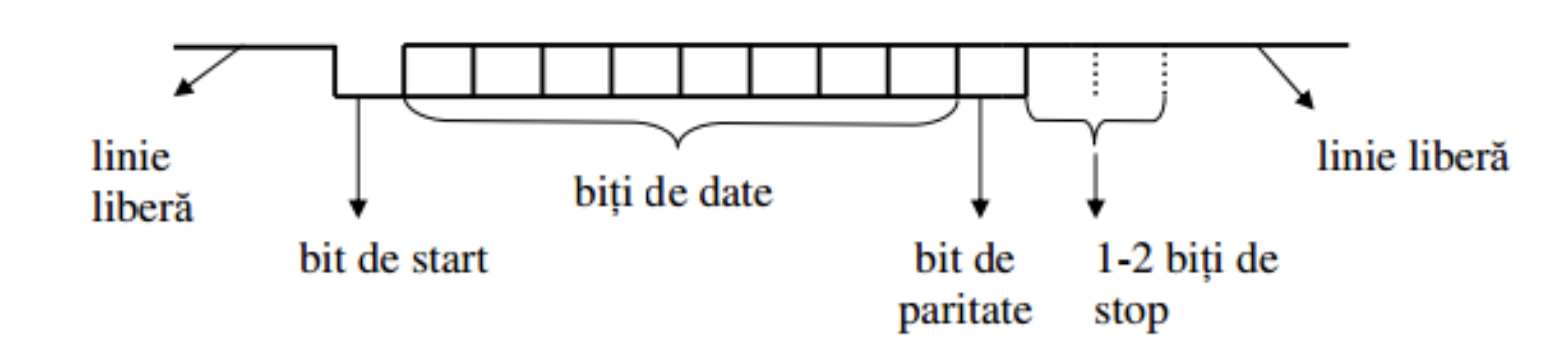
- bit de paritate (pentru cazul parității pare):

1 - dacă numărul de biți de 1 logic din datele utile este impar, pentru a avea un număr par de biți de 1 cu tot cu bitul de paritate

0 – dacă numărul de biți de 1 logic din datele utile este deja par

- rată de transfer

- bit / biți de stop



**INTERFAȚA I2C:**

Interfața Inter-Integrated Cicruit (I2C) este o interfață utilizată foarte mult în sistemele încorporate și de aceea este prezentă în majoritatea microcontrolerelor. Interfața I2C este utilă pentru a conecta periferice cu viteză mică la un sistem, și care sunt utilizate pentru a transmite datele pe distanțe scurte. I2C prezintă de asemenea o comunicare bazată pe acknowledge și utilizează pentru transfer doar două linii numite serial data SDA și serial clock SCL.

Pe placa Arduino UNO R3, pinii dedicați pentru I2C sunt:

- Pinul A4: SDA

- Pinul A5: SCL

În cadrul proiectului nostru, interfața I2C a fost folosită la conectarea ecranului LCD.

A diagram of a device

Description automatically generated

Se observă cele 2 linii care sunt legate în manieră open-drain. Din acest motiv este esențial să se ducă liniile la 1 logic printr-o rezistență de pull-up de 4.7KOhmi.

Tot din imagine ne dăm seama că se pot conecta mai multe dispozitive la magistrala I2C. Cele două rezistențe creează un ȘI și astfel orice dispozitiv care pune 0 pe linii va trage întreaga linie la 0. Linia este 1 când nici un dispozitiv nu comunică pe magistrală.

**DESCRIEREA COMPONENTELOR CE ALCĂTUIESC SISTEMUL**

**SENZORII**

În proiectul nostru folosim doi senzori ultrasonici HC-SR04, cu ajutorul cărora detectăm prezența unei mașini în zona barierei.

**SCHEMA BLOC:**

A blue circuit board with wires and wires

Description automatically generated

A white rectangular object with black text

Description automatically generated with medium confidence

**PRINCIPIUL DE FUNCȚIONARE:**

Senzorul este alcătuit din două traductoare cu ultrasunete: un transmițător și un receptor. Transmițătorul emite pulsuri ultrasonice, iar receptorul așteaptă ca unda reflectată să se întoarcă spre senzor. Această tehnică este utilizată pentru a putea calcula distanța față de obiectul care reflectă unda ultrasonică.

A diagram of a wave

Description automatically generated

Pentru a genera ultrasunetul trebuie să setăm pinul Trig pe 1 logic timp de 10µs. Acest lucru va duce la emiterea unei “explozii” ultrasonice de 8 cicluri ce va călători cu viteza sunetului (343 m/s). Pinul Echo se setează pe 1 logic imediat după ce explozia este trimisă și începe să asculte sau să aștepte ca unda să fie reflectată de un obiect. Dacă nu există niciun obiect sau puls reflectat, pinul Echo se va opri după 38 ms și va reveni pe 0 logic.

A diagram of a circuit

Description automatically generated

Dacă primim un impuls reflectat, pinul Echo va cădea pe 0 logic mai devreme decât cele 38ms. În funcție de timpul în care pinul Echo a fost pe 1 logic, putem determina distanța parcursă de unda sonoră, deci distanța de la senzor la obiect, folosind formula:

**Distanță = Viteză x Timp.**

Cunoaștem de fapt atât valoarea vitezei, cât și cea a timpului. Timpul este perioada în care pinul Echo a fost pe 1 logic, iar viteza este viteza sunetului, care este de 343 m/s. Există un pas suplimentar pe care trebuie să-l facem: împărțirea rezultatului final la 2. Asta pentru că măsurăm durata de care unda sonoră are nevoie pentru a călători până la obiect și a reveni.

A diagram of a device with a cylinder and a cylinder

Description automatically generated with medium confidence

**CONECTAREA LA ARDUINO:**

Pinul GND trebuie conectat la un pin GND de pe Arduino. Pinul VCC trebuie conectat la pinul 5V de pe Arduino. Pinii Trig și Echo pot fi conectați la oricare pini digitali de pe Arduino, cu condiția ca Trig să fie declarat ca pin de ieșire, iar Echo ca pin de intrare.

A circuit board with wires and wires

Description automatically generated

**SERVOMOTORUL**

Folosim un servomotor MG996R cu cuplu mare ce se poate roti aproximativ 180°. Servomotorul MG996R este un servomotor popular și larg utilizat în domeniul roboticii și în alte aplicații care necesită controlul precis al poziției.

A close-up of a device

Description automatically generated

**SCHEMA BLOC:**

**A diagram of a circuit board

Description automatically generated**

**PRINCIPIUL DE FUNCȚIONARE:**

Pinul PWM este cel care controlează mișcarea servomotorului. Fiind conectat la un pin cu funcție PWM de pe Arduino, acesta poate fi programat pentru a roti servomotorul la un anumit unghi, folosind funcția *write(int angle)* din biblioteca *Servo.* Această funcție modulează factorul de umplere al semnalului PWM pentru a controla poziția servomotorului.

**CONECTAREA LA ARDUINO:**

Pinul PWM trebuie conectat la orice pin digital cu funcție PWM de pe Arduino. Pinul 5V/VCC și pinul GND trebuie conectați la pinul +, respectiv pinul S al unui adaptor cu 5 pini – USB. Acest lucru este necesar deoarece pinul 5V de pe Arduino nu poate furniza suficientă putere la pornirea servomotorului.

A white rectangular object with wires connected to it

Description automatically generated

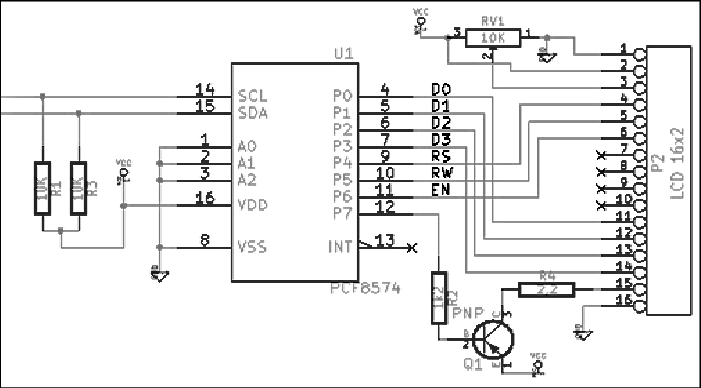
**ECRANUL LCD**

Folosim ecranul LCD 1602 cu modul pentru interfața I2C. Ecranul este organizat sub forma unei matrice de pixeli, cu 16 caractere pe linie și 2 linii. Tensiunea de alimentare este de 5V, contrastul se poate regla cu ajutorul potențiometrului integrat.

A close-up of a circuit board

Description automatically generated

**SCHEMA BLOC:**

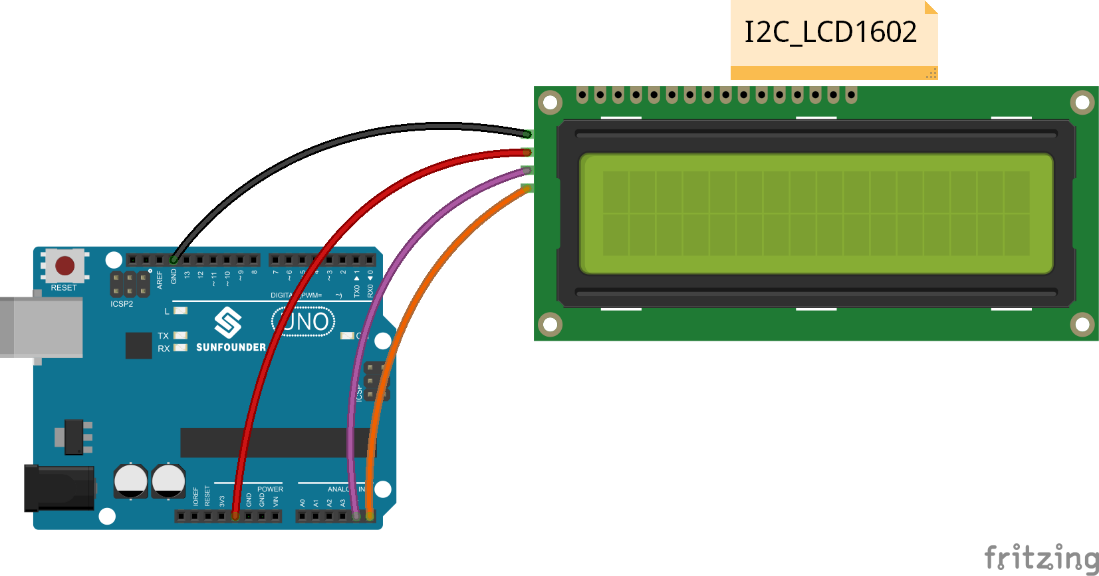


**PRINCIPIUL DE FUNCȚIONARE:**

Ecranul LCD 1602 conține un strat subțire de cristale lichide între două straturi de sticlă. Aceste cristale pot fi polarizate pentru a permite sau bloca trecerea luminii prin ele, în funcție de tensiunea aplicată. Ecranul este echipat și cu o sursă de lumină din spate (backlight) pentru a îmbunătăți vizibilitatea textului.

**CONECTAREA LA ARDUINO:**

Pinul SCL se conectează la pinul A5 de pe Arduino, iar pinul SDA se conectează la pinul A4, deoarece pinii A5 și A4 sunt pinii dedicați interfeței I2C. Pinul VCC se conectează la pinul 5V de pe Arduino, iar pinul GND al modulului I2C la pinul GND.



**LED-URILE**

Un **LED** (light-emitting diode = diodă emițătoare de lumină) este o [diodă semiconductoare](https://ro.wikipedia.org/wiki/Diod%C4%83_semiconductoare) ce emite lumină la polarizarea directă a [joncțiunii p-n](https://ro.wikipedia.org/wiki/Jonc%C8%9Biune_p-n).



**SCHEMA BLOC:**

A diagram of a blue and black electronic device

Description automatically generated with medium confidence

**PRINCIPIUL DE FUNCȚIONARE:**

LED-urile nu au un filament care să ardă, și nu se încălzesc prea tare. Acestea sunt iluminate numai de mișcarea electronilor într-un material semiconductor. [Culoarea](https://ro.wikipedia.org/wiki/Culoare) luminii emise depinde de compoziția și de starea materialului semiconductor folosit și poate fi în spectrul [infraroșu](https://ro.wikipedia.org/wiki/Infraro%C8%99u), [vizibil](https://ro.wikipedia.org/wiki/Lumin%C4%83) sau [ultraviolet](https://ro.wikipedia.org/wiki/Ultraviolet).

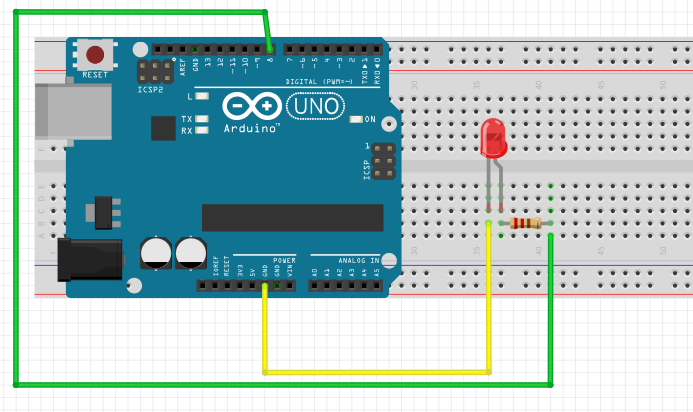
Câteva avantaje ale LED-urilor: consum mic de putere, dimensiune mică, durată de viață mare.

Într-un LED, terminalul pozitiv este numit anod și terminalul negativ este numit catod. Pentru ca LED-ul să funcționeze corect, anodul LED-ului ar trebui să aibă un potențial mai mare decât catodul, deoarece curentul prin LED "curge" de la anod la catod.

Pentru a controla curentul care trece printr-un LED, utilizăm rezistențe.

**CONECTAREA LA ARDUINO:**

Rezistența se află conectată la pinul digital de pe placa Arduino. Se conectează anodul (piciorul mai lung) în serie cu rezistența (220 ohmi). Catodul LED-ului (partea mai scurtă) este conectat la GND.

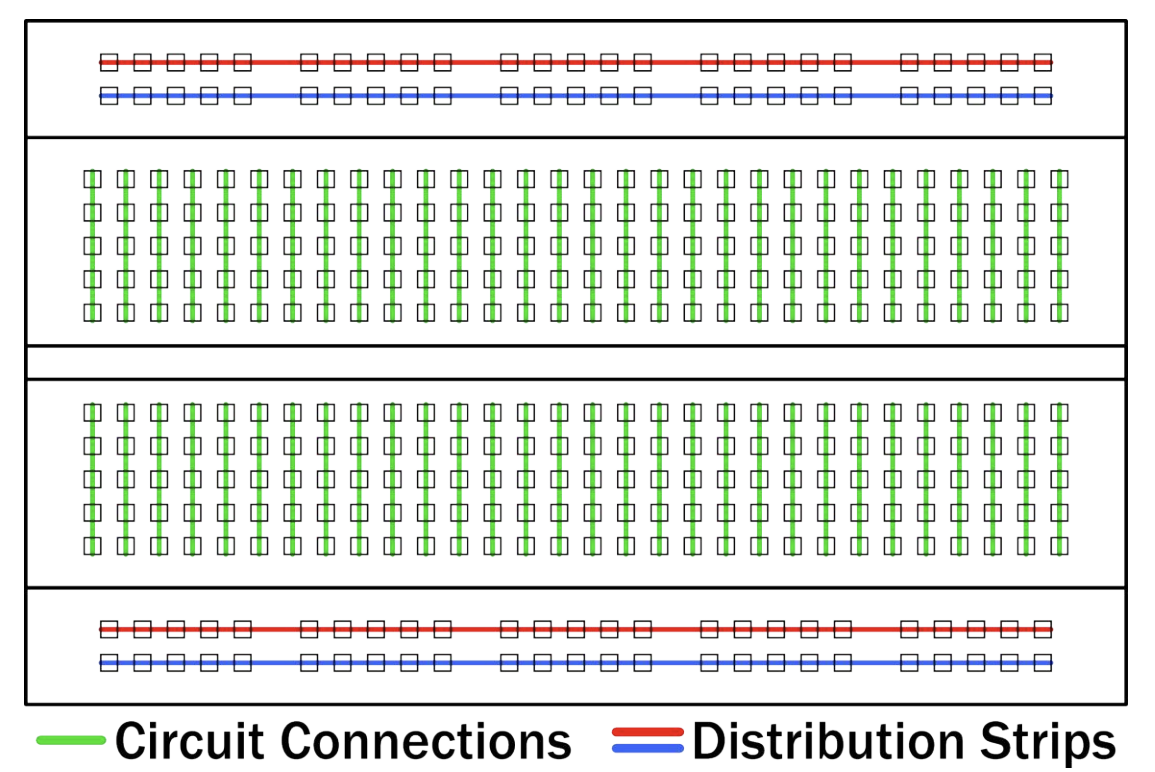


**BREADBOARD-UL**

Un [breadboard](https://www.robofun.ro/breadboard) este o placă de test cu anumite terminale și contacte, unde se pot introduce diferite componente electronice, pentru a realiza circuite temporare.

Breadboard-ul folosit de noi are 400 de pini. Are în componență 60 de coloane verticale destinate circuitelor (cele cu verde) și 4 rânduri orizontale pentru distribuția semnalului de GND și 5V (2 pentru GND și 2 pentru 5V).

**SCHEMA BLOC:**



**PRINCIPIUL DE FUNCȚIONARE:**

Componentele electronice se introduc în găurile aferente, firele introduse pe aceeași linie vor fi conectate între ele. Contactele sunt legate pe verticală, iar în partea de sus a plăcii se găsesc două linii de contacte legate pe orizontală. Acestea din urmă de folosesc pentru alimentarea cu curent electric.

**CONECTAREA LA ARDUINO:**

Pentru a conecta dispozitivele se folosesc fire jumper tată-tată (cu pini la ambele capete), care se introduc în găurile din breadboard.

**REZISTENȚELE**

Rezistențele sunt componente electronice utilizate pentru a limita fluxul de curent într-un circuit electric. Acest lucru este util pentru a proteja alte componente ale circuitului care nu pot tolera curenți mari. Unitatea de măsură a rezistențelor este Ohm-ul.

A resistor with numbers and a number

Description automatically generated

În proiectul nostru, utilizăm un LED roșu și unul verde. Căderile de tensiune de pe cele două LED-uri sunt similare, aproximativ 2V. Rezultă că pe rezistență, căderea de tensiune va fi de 3V, deoarece tensiunea de alimentare de la Arduino este de 5V. Curentul prin LED trebuie să fie limitat la 10-30mA. Presupunând că ne dorim un curent de 10mA și utilizând legea lui Ohm (R = U/I) va rezulta o rezistență R = 3V/0.01A = 300 Ohmi. Așadar, având cea mai apropiată rezistență cu valoarea de 220Ohmi vom testa dacă este potrivită.

I = U/R = 3V/220Ohmi = 0.0136A = 13.6mA. Deci rezultă o valoare corectă a curentului ce străbate LED-ul.

**DESCRIEREA SOFTWARE**

**CODUL ARDUINO CU EXPLICAȚII**

#include <Wire.h> *// bibliotecă (încorporată deja în Arduino IDE) ce ne permite să utilizăm modulul I2C*

#include <LiquidCrystal\_I2C.h> *// biblioteca conține funcțiile clear, print și setCursor utile pentru a reseta ecranul, afișa un text dorit pe ecran și a muta cursorul ce indică începutul unei noi porțiuni de text*

*// link preluare bibliotecă: https://robojax.com/learn/arduino/?vid=robojax-LCD1602-I2C*

#include <Servo.h> *// bibliotecă (încorporată deja în Arduino IDE) ce ne ajută să lucrăm cu un servomotor și conține funcția write ce ne permite să controlam rotația servomotorului, precum și funcția attach ce ne permite să precizăm pinul de pe placă corespunzător pinului PWM al servomotorului*

*// declarăm pinii digitali*

const int greenLedPin = 13;

const int redLedPin = 12;

const int trigPinIn = 11;

const int echoPinIn = 10;

const int servoPin = 9;

const int trigPinOut = 7;

const int echoPinOut = 6;

LiquidCrystal\_I2C lcd(0x27, 16, 2); *// declarăm un obiect de tip ecran LCD cu 2 rânduri a câte 16 celule și adresa 0x27*

Servo servo; *// declarăm un obiect de tip Servo*

int parkingSpots = 5; *// numărul de locuri de parcare prestabilit*

float durationIn, distanceIn; *// variabile utilizate pentru a colecta măsuratorile de la senzorul ultrasonic de la intrarea in parcare*

float durationOut, distanceOut; *// variabile utilizate pentru a colecta măsuratorile de la senzorul ultrasonic de la ieșirea din parcare*

int simultan = 0; *// flag folosit pentru a indica prezența unei situații când două mașini doresc utilizarea barierei simultan*

int entryDistance = 10*; // când o mașină se află la o distanță mai mică decât acest prag față de senzor, atunci este considerată suficient de aproape de barieră pentru a o deschide*

int farThreshold = 20; *// când o mașină se află la o distanță mai mare decât acest prag față de senzor, atunci este considerată îndepărtată suficient*

*// funcția setup, în care se configurează pinii digitali*

void setup() {

pinMode(greenLedPin, OUTPUT);

pinMode(redLedPin, OUTPUT);

lcd.begin();

servo.attach(servoPin);

servo.write(0);

pinMode(trigPinIn, OUTPUT);

pinMode(echoPinIn, INPUT);

pinMode(trigPinOut, OUTPUT);

pinMode(echoPinOut, INPUT);

Serial.begin(9600);

}

void loop() {

while(1)

{

*// controlăm LED-urile*

if(parkingSpots > 0) *// atunci când parcarea mai are locuri libere, LED-ul verde va fi aprins, iar cel roșu stins*

{

digitalWrite(greenLedPin, HIGH);

digitalWrite(redLedPin, LOW);

}

else *// atunci când parcarea nu mai are locuri libere, LED-ul roșu va fi aprins, iar cel verde stins*

{

digitalWrite(greenLedPin, LOW);

digitalWrite(redLedPin, HIGH);

}

*// afișăm pe ecranul LCD numărul de locuri libere*

lcd.clear();

lcd.print("Locuri neocupate");

lcd.setCursor(7,1);

lcd.print(parkingSpots);

*// senzorul de la intrarea în parcare ia măsurători*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2); *// resetăm pinul Trig pe 0 logic pentru 2 microsecunde ca sa ne asiguram ca porneste de pe 0 logic*

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10); *// setăm pinul Trig pe 1 logic pentru 10 microsecunde pentru ca senzorul să genereze o explozie ultrasonică de 8 cicluri*

digitalWrite(trigPinIn, LOW); *// resetăm pinul Trig pe 0 logic, lucru ce duce la setarea imediată a pinului Echo pe 1 logic*

durationIn = pulseIn(echoPinIn, HIGH); *// stocăm rezultatul funcției pulseIn ce ne măsoară timpul în care pinul Echo rămâne pe 1 logic*

distanceIn = (durationIn\*.0343)/2; *// calculăm distanța în cm față de obiectul detectat*

Serial.print("Distance IN: ");

Serial.println(distanceIn);

*// senzorul de la ieșirea din parcare ia măsurători în mod similar cu cel de la intrare*

digitalWrite(trigPinOut, LOW);

delayMicroseconds(2);

digitalWrite(trigPinOut, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinOut, LOW);

durationOut = pulseIn(echoPinOut, HIGH);

distanceOut = (durationOut\*.0343)/2;

Serial.print("Distance OUT: ");

Serial.println(distanceOut);

*// verificăm dacă se dorește ridicarea barierei*

if(distanceIn < entryDistance || distanceOut < entryDistance)

{

if(distanceOut < entryDistance) *// tratăm cazul în care o mașină dorește să iasă din parcare*

{

int time = 0;

while (time < 700) *// timp de 700ms verificăm dacă pe partea cealaltă nu se dorește simultan intrarea în parcare*

{

*// verificăm distanța calculată de senzorul de la intrarea în parcare*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

if (distanceIn < entryDistance) *// dacă există în același timp o mașină pe partea opusă a barierei,*

{

*// atunci vom începe secvența de verificare a îndepărtării mașinii de la intrare*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

while (distanceIn < farThreshold) *// cât timp mașina este în curs de îndepărtare,*

{

*// vom afișa un mesaj specific pe ecranul LCD*

lcd.clear();

lcd.print("Va rugam dati cu");

lcd.setCursor(4, 1);

lcd.print("spatele!");

*// și vom continua cu colectarea distanței curente*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

delay(200); *// verificăm deplasamentul la fiecare 200ms*

}

*// când îndepărtarea s-a finalizat cu succes, pe ecranul LCD revine mesajul inițial*

lcd.clear();

lcd.print("Locuri neocupate");

lcd.setCursor(7, 1);

lcd.print(parkingSpots);

break; *// situația de solicitare simultană a barierei a fost rezolvată, așa că se continuă cu ridicarea barierei*

}

time += 70; *// time se incrementează cu 70 la fiecare pas, iar cum condiția din while îl limitează la valoarea 700, rezultă că vom face 10 măsurători*

delay(70); *// iar cele 10 măsurători vor introduce fiecare câte 70ms de întârziere, ceea ce duce la un delay cumulat de 700ms*

}

delay(300); *// pentru a avea o mișcare cât mai realistă a barierei, vom mai aștepta un timp scurt până ridicăm bariera, nu o acționăm instantaneu*

*// se ridică bariera în mod fluent*

for (int angle = 0; angle <= 90; angle += 1)

{

servo.write(angle);

delay(10);

}

parkingSpots++; *// ieșirea din parcare este în curs deci se incrementează numărul de locuri disponibile*

*// se verifică dacă mașina a trecut și de senzorul de la intrare*

distanceIn = 50;

while(distanceIn > entryDistance)

{

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

}

*// începe secvența de verificare a îndepărtării mașinii față de senzorul de la intrare, deoarece el e ultimul pe lângă care trece*

*// algoritmul este identic cu cel aplicat în situația solicitării simultane a barierei*

*// această verificare e necesară deoarece nu ne dorim ca bariera să se ridice prea rapid, creându-se eroarea cum că mașina care tocmai iasă ar dori de fapt să reintre imediat*

*// astfel se generează un delay într-un mod mai dinamic*

while (distanceIn < farThreshold) *// cât timp mașina este în curs de îndepărtare,*

{

*// și vom continua cu colectarea distanței curente*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

delay(200); *// verificăm deplasamentul la fiecare 200ms*

}

delay(300); *// se mai așteaptă puțin, tot din motive de realism*

*// apoi se coboară bariera*

for (int angle = 90; angle >= 0; angle -= 1)

{

servo.write(angle);

delay(10);

}

*// actualizare stare LED-uri*

if(parkingSpots > 0)

{

digitalWrite(greenLedPin, HIGH);

digitalWrite(redLedPin, LOW);

}

else

{

digitalWrite(greenLedPin, LOW);

digitalWrite(redLedPin, HIGH);

}

*// afișăm pe ecranul LCD numărul actualizat de locuri libere*

lcd.clear();

lcd.print("Locuri neocupate");

lcd.setCursor(7,1);

lcd.print(parkingSpots);

}

else if(distanceIn < entryDistance) *//tratăm cazul în care se dorește intrarea în parcare*

{

if(parkingSpots > 0)

{

int time = 10;

while (time < 700) *// timp de 700ms verificăm dacă pe partea cealaltă nu se dorește simultan ieșirea din parcare*

{

*// situația se tratează similar cu cea întâlnită la cazul ieșirii din parcare*

digitalWrite(trigPinOut, LOW);

delayMicroseconds(2);

digitalWrite(trigPinOut, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinOut, LOW);

durationOut = pulseIn(echoPinOut, HIGH);

distanceOut = (durationOut\*.0343)/2;

Serial.print("Distance OUT: ");

Serial.println(distanceOut);

if (distanceOut < entryDistance) *// dacă există în același timp o mașină pe partea opusă a barierei,*

{

simultan = 1; *// atunci setăm variabila "simultan", deoarece ne aflăm într-un caz mai special*

*// gestionând intrarea în parcare, ne dorim să prioritizăm mașina ce dorește să iasă pentru a se elibera astfel un loc*

*// vom forța astfel ca mașina ce așteaptă intrarea în parcare să se îndepărteze pentru a-i face loc celei care dorește să iasă*

*// începe secvența care verifică îndepărtarea mașinii de la intrare*

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

while (distanceIn < farThreshold)

{

lcd.clear();

lcd.print("Va rugam dati cu");

lcd.setCursor(4, 1);

lcd.print("spatele!");

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

delay(200);

}

lcd.clear();

lcd.print("Locuri neocupate");

lcd.setCursor(7, 1);

lcd.print(parkingSpots);

break; *// dacă mașina de la intrare s-a îndepărtat cu succes de barieră înseamnă că putem continua spre ridicarea barierei*

}

simultan = 0;

time += 70;

delay(70);

}

delay(300); *// pentru a avea o mișcare cât mai realistă a barierei, vom mai aștepta un timp scurt până ridicăm bariera, nu o acționăm instantaneu*

*// se ridică bariera*

for (int angle = 0; angle <= 90; angle += 1)

{

servo.write(angle);

delay(10);

}

*// dacă bariera a fost ridicată în urma creării unei situații de "conflict" înseamnă că s-a produs o ieșire, deci incrementăm numărul de locuri libere*

*// în caz contrar, înseamnă că s-a produs o intrare, deci decrementăm numărul de locuri libere*

if(simultan) parkingSpots++;

else parkingSpots--;

if(simultan) *// dacă am avut de a face cu situația de "simultan", atunci așteptăm ca mașina ce a ieșit să treacă și de senzorul de la intrare*

{

distanceIn = 50;

while(distanceIn > entryDistance)

{

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

}

}

else *// altfel, așteptăm ca mașina ce a intrat să treacă și de senzorul de la ieșire*

{

distanceOut = 50;

while(distanceOut > entryDistance)

{

digitalWrite(trigPinOut, LOW);

delayMicroseconds(2);

digitalWrite(trigPinOut, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinOut, LOW);

durationOut = pulseIn(echoPinOut, HIGH);

distanceOut = (durationOut\*.0343)/2;

Serial.print("Distance OUT: ");

Serial.println(distanceOut);

}

}

*// similar cu cazul anterior, pentru a nu se ridica bariera prea rapid la venirea unei eventuale viitoare cereri, vom aștepta îndepărtarea mașinii ce a intrat/ieșit*

if(simultan) *// dacă am avut de a face cu situația de "simultan", atunci vom aștepta îndepărtarea mașinii ce a ieșit din parcare*

{

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

while(distanceIn < farThreshold)

{

digitalWrite(trigPinIn, LOW);

delayMicroseconds(2);

digitalWrite(trigPinIn, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinIn, LOW);

durationIn = pulseIn(echoPinIn, HIGH);

distanceIn = (durationIn\*.0343)/2;

Serial.print("Distance IN: ");

Serial.println(distanceIn);

delay(200);

}

}

else *// altfel, vom aștepta îndepărtarea mașinii ce a intrat în parcare*

{

digitalWrite(trigPinOut, LOW);

delayMicroseconds(2);

digitalWrite(trigPinOut, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinOut, LOW);

durationOut = pulseIn(echoPinOut, HIGH);

distanceOut = (durationOut\*.0343)/2;

Serial.print("Distance OUT: ");

Serial.println(distanceOut);

while(distanceOut < farThreshold)

{

digitalWrite(trigPinOut, LOW);

delayMicroseconds(2);

digitalWrite(trigPinOut, HIGH);

delayMicroseconds(10);

digitalWrite(trigPinOut, LOW);

durationOut = pulseIn(echoPinOut, HIGH);

distanceOut = (durationOut\*.0343)/2;

Serial.print("Distance OUT: ");

Serial.println(distanceOut);

delay(200);

}

}

delay(300); *// se mai așteaptă puțin, tot din motive de realism*

*// se coboară bariera*

for (int angle = 90; angle >= 0; angle -= 1)

{

servo.write(angle);

delay(10);

}

*// actualizare stare LED-uri*

if(parkingSpots > 0)

{

digitalWrite(greenLedPin, HIGH);

digitalWrite(redLedPin, LOW);

}

else

{

digitalWrite(greenLedPin, LOW);

digitalWrite(redLedPin, HIGH);

}

*// afișăm pe ecranul LCD numărul actualizat de locuri libere*

lcd.clear();

lcd.print("Locuri neocupate");

lcd.setCursor(7,1);

lcd.print(parkingSpots);

}

}

}

delay(100); *// senzorii calculează distanțe la fiecare 100ms pentru a capta într-un timp cât mai apropiat de cel real o solicitare de ridicare a barierei*

}

}

**BIBLIOGRAFIE**

**FOI DE CATALOG:**

<https://docs.arduino.cc/hardware/uno-rev3/> - Foile de catalog ale plăcuței de dezvoltare

<https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf> - Foaia de catalog a senzorului ultrasonic

<https://www.handsontec.com/dataspecs/motor_fan/MG996R.pdf> - Foaia de catalog a servomotorului

<https://www.handsontec.com/dataspecs/module/I2C_1602_LCD.pdf> - Foaia de catalog a ecranului LCD cu modul I2C

<https://www.mouser.com/datasheet/2/58/BPS-DAT-(BB400)-Datasheet-932623.pdf> – Foaia de catalog a breadboard-ului

**SURSE DE INFORMARE:**

<https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/> - Utilizat pentru funcționarea senzorilor

<https://mschoeffler.com/2021/07/17/arduino-tutorial-mg-996r-servo-motor/> - Utilizat pentru funcționarea servomotorului

<https://docs.arduino.cc/learn/electronics/lcd-displays/> <https://www.youtube.com/watch?v=q9YC_GVHy5A> – Utilizate pentru funcționarea ecranului LCD

<https://docs.arduino.cc/built-in-examples/basics/Blink/> <https://roboticsbackend.com/arduino-led-complete-tutorial/> - Utilizate pentru funcționarea LED-urilor

<https://blog.robofun.ro/2021/09/24/ce-este-un-breadboard-si-care-este-scopul-in-proiecte-de-electronica/> <https://sites.google.com/site/arduinoelectronicasiprogramare/prima-lectie/a-treia-lectie> - Utilizate pentru funcționarea breadboard-ului