

PORTOFOLIU DE PRACTICĂ

MOBILITATE CIPRU

10.05.2026 – 23.05.2026

VACARU GEORGE

VELEZ VERONICA

VÎLCU DAVID

Cuprins

PROIECT: Semafor 3

PROIECT: Sistem Tactil cu Confirmare Sonoră prin Impulsuri 6

PROIECT: Translator & Decodor Bidirecțional de Cod Morse..... 9

PROIECT: MOTOR STEPPER 11

PROIECT : Sistem Conveior cu Motor Pas-cu-Pas, Tastatură I2C și EEPROM |
Stepper Belt Conveyor with I2C Keypad & EEPROM 14

PROIECT FINAL: Conveyor Belt with Ejection Platform 30

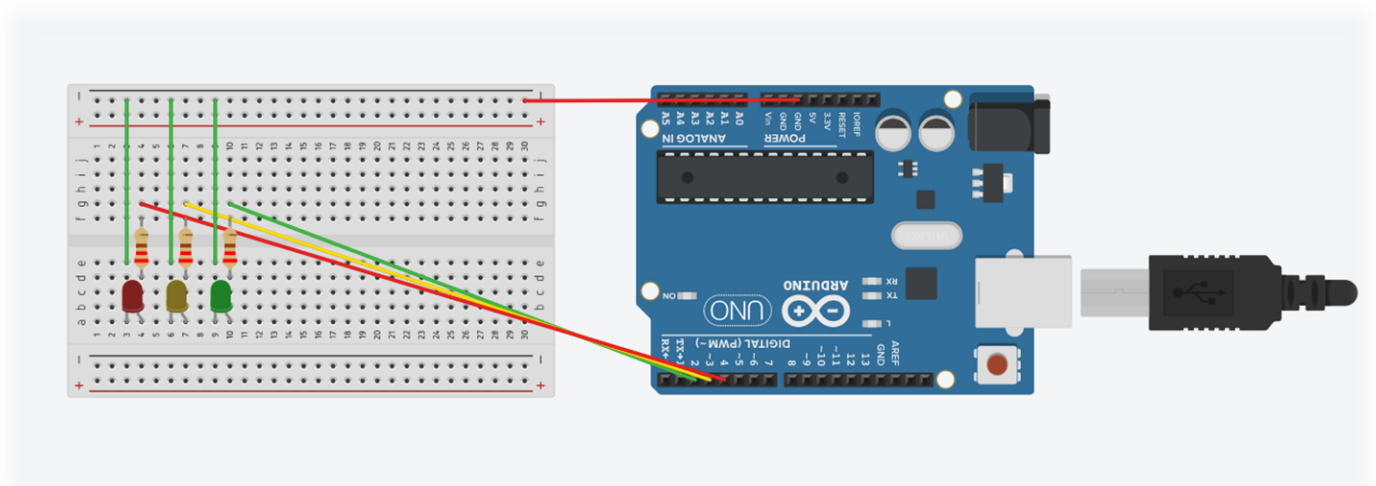
PROIECT: Semafor

1. Obiectivul proiectului:

Crearea unui semafor funcțional cu trei LED-uri (verde, galben, roșu). Sistemul simulează ciclul unui semafor real: verde (3s) – galben (1.5s) – roșu (3s), repetând la infinit.

2. Lista de piese necesare:

- a. 1 x Placă **Arduino Uno**
- b. 1 x **LED Verde** (pinul 4)
- c. 1 x **LED Galben** (pinul 3)
- d. 1 x **LED Roșu** (pinul 2)
- e. 3 x **Rezistențe** (220 Ohmi, câte una pentru fiecare LED)
- f. 1 x **Breadboard** și fire de conexiune (jumper wires)



3. Schema de conectare:

a. LED Verde (ledG):

- i. Anodul (+) prin rezistență de 220Ω la pinul digital 4; catodul (-) la GND.

b. LED Galben (ledY):

- i. Anodul (+) prin rezistență de 220Ω la pinul digital 3; catodul (-) la GND.

c. LED Roșu (ledR):

- i. Anodul (+) prin rezistență de 220Ω la pinul digital 2; catodul (-) la GND.

- d. **Notă:** Catodul (piciorușul scurt) se conectează la GND; anodul (piciorușul lung) la rezistență.

4. Codul proiectului:

```
#define ledR 2
#define ledG 4
#define ledY 3

void setup(){
    pinMode(ledR, OUTPUT);
    pinMode(ledG, OUTPUT);
    pinMode(ledY, OUTPUT);
}

void loop(){
    digitalWrite(ledG, HIGH);
    delay(3000);
    digitalWrite(ledG, LOW);
    digitalWrite(ledY, HIGH);
    delay(1500);
    digitalWrite(ledY, LOW);
    digitalWrite(ledR, HIGH);
    delay(3000);
    digitalWrite(ledR, LOW);
}
```

5. Exerciții și Provocări

1. **Ajustarea timpilor:** Modifică valorile din delay() pentru a simula un semafor mai rapid sau mai lent. Încearcă delay(500) pentru galben – observi diferența?
2. **Semafor pentru pietoni:** Adaugă două LED-uri în plus (roșu și verde) care să se comporte invers față de semaforul auto: când mașinile au verde, pietonii au roșu și invers.
3. **Fază roșu-galben:** Semafoarele reale au și faza roșu+galben înainte de verde. Adaugă această fază în cod: aprinde atât ledR cât și ledY timp de 1 secundă înainte de verde.

!!! Sfat: Funcția `delay()` blochează complet Arduino-ul pe durata ei – nimic altceva nu rulează! Dacă vrei să faci ceva mai complex (ex: să răspunzi la un buton în timp ce LED-ul e aprins), folosește `millis()` în loc de `delay()`.

PROIECT: Sistem Tactil cu Confirmare Sonoră prin Impulsuri

Scopul Proiectului

Proiectul își propune să înregistreze o secvență de atingeri repetate ale unui senzor tactil și să redea un număr echivalent de semnale acustice odată ce utilizatorul a terminat de atins senzorul (după expirarea unui timp de așteptare).

Funcționare și Logică de Control

1. Detectarea Atingerilor (Numărare):

- Senzorul de atingere este citit continuu pe pinul digital 2.
- Codul folosește **detectarea pe front crescător** (`touchState == HIGH && lastTouchState == LOW`), ceea ce înseamnă că sistemul înregistrează o atingere *doar în momentul apăsării*, nu și dacă degetul este ținut apăsat continuu.
- La fiecare atingere, variabila `count` crește cu 1, iar temporizatorul `lastTouchTime` este actualizat la ora curentă furnizată de funcția `millis()`.
- Include un scurt timp de întârziere (debounce) de 200 ms pentru a evita înregistrările false sau duble provocat de zgomotul electric.

2. Temporizarea și Redarea Semnalului (Buzzer):

- **Timpul de așteptare (1.5 secunde):** Cât timp utilizatorul continuă să atingă senzorul la un interval mai mic de 1500 ms, sistemul doar adună atingerile.
- **Secvența acustică:** Dacă au trecut 1.5 secunde de la ultima atingere și `count` este mai mare decât 0, sistemul declanșează buzzerul de un număr de ori egal cu numărul de atingeri înregistrate (fiecărui bip îi corespund 200ms sunet + 200ms pauză).
- **Resetare:** După finalizarea secvenței sonore, contorul `count` se resetează la 0, pregătind sistemul pentru o nouă serie.

Conexiuni Hardware (Pinout)

- **Senzor Tactil (Touch Sensor):** Pinul de semnal este conectat la pinul digital 2.
- **Buzzer (Piezoelectric / Activ):** Conectat la pinul digital 8.

Aplicații Practice

Codul reprezintă o bază excelentă pentru:

- **Interfețe HMI minimale:** Meniuri tactile simple fără afișaj ecran (de exemplu: 1 atingere = Mod 1, 2 atingeri = Mod 2, cu confirmare sonoră).
- **Sisteme de acces / Cod de securitate simplu:** Numărare de impulsuri tactile.
- **Jucării interactive sau dispozitive de asistență:** Oferă un feedback audio bazat pe numărul de interacțiuni tactile ale utilizatorului.

COD:

```
const int touchPin = 2;
const int buzzerPin = 8;

int touchState = 0;
int lastTouchState = 0;
int count = 0;

unsigned long lastTouchTime = 0;
const int waitTime = 1500;

void setup() {
  pinMode(touchPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  touchState = digitalRead(touchPin);

  // Detectare front: trecere de la LOW la HIGH = o atingere noua
  if (touchState == HIGH && lastTouchState == LOW) {
    count++;
    lastTouchTime = millis();
    delay(200);    // debouncing software
  }

  lastTouchState = touchState;

  // Dupa 1.5s fara atingere, bipaie de "count" ori
  if (count > 0 && (millis() - lastTouchTime > waitTime)) {
    for (int i = 0; i < count; i++) {
```

```
    digitalWrite(buzzerPin, HIGH);  
    delay(200);  
    digitalWrite(buzzerPin, LOW);  
    delay(200);  
}  
count = 0;  
}  
}
```

PROIECT: Translator & Decodor Bidirecțional de Cod Morse

Scopul Proiectului

Proiectul își propune să creeze o interfață interactivă de comunicare bazată pe alfabetul Morse. Acesta funcționează în ambele sensuri:

1. **Intrare Tactilă → Text:** Convertește durata atingerilor pe un senzor în puncte (.) și linii (-), pe care le traduce ulterior în litere sau cifre afișate în *Serial Monitor*.
2. **Intrare Text → Cod Morse:** Primește caractere trimise din *Serial Monitor* și le „cântă”/luminează automat sub formă de secvențe Morse.

Funcționare și Logică de Control

1. **Modul Decodificare (De la Atingere la Text):**
 - **Interpretarea duratei:**
 - Atingere **scurtă** (< 300 ms): Înregistrează un punct (.)
 - Atingere **lungă** (≥ 300 ms): Înregistrează o linie (-).
 - **Măsurarea pauzelor (Delimitare):**
 - Pauză > 1 secundă (1000 ms): Înseamnă că ai terminat de introdus o literă → sistemul o caută în tabelul intern, o afișează pe ecran și resetează secvența.
 - Pauză > 3 secunde (3000 ms): Marchează finalul unui cuvânt și afișează [spatiu].
 - **Feedback instant:** Cât timp ții degetul pe senzor, LED-ul stă aprins, iar buzzerul emite un sunet cu frecvența de 800 Hz.
2. **Modul Encodare și Redare (De la Text la Sunet/Lumină):**
 - Când introduci o literă în consola Serială, Arduino o caută în dicționarul morseTable (litere de la A-Z și cifre 0-9).
 - Dacă o găsește, o reda automat folosind funcția playMorse():
 - **Punct (.)**: Bip scurt (150 ms).
 - **Linie (-)**: Bip lung (400 ms).
 - LED-ul conectat pâlpâie sincronizat cu buzzerul.

Conexiuni Hardware (Pinout)

- **Senzor Tactil (Touch Pin):** Pinul digital **2**
- **Buzzer Piezoelectric:** Pinul digital **8**
- **LED (Status/Semnal):** Pinul digital **13** (sau LED-ul integrat pe placă)

Aplicații Practice

- **Proiecte educaționale:** Învățarea și exersarea alfabetului Morse într-un mod interactiv.

- **Dispozitive de asistență:** Muncă de cercetare pe interfețe de comunicare minimale pentru persoane cu dizabilități motorii (ex: transmiterea de mesaje prin atingeri simple).
- **Comunicații criptate/alternative:** Transmiterea de date acustice sau optice pe distanțe scurte.

PROIECT: MOTOR STEPPER

[Link material video](#)

1. Obiectivul proiectului:

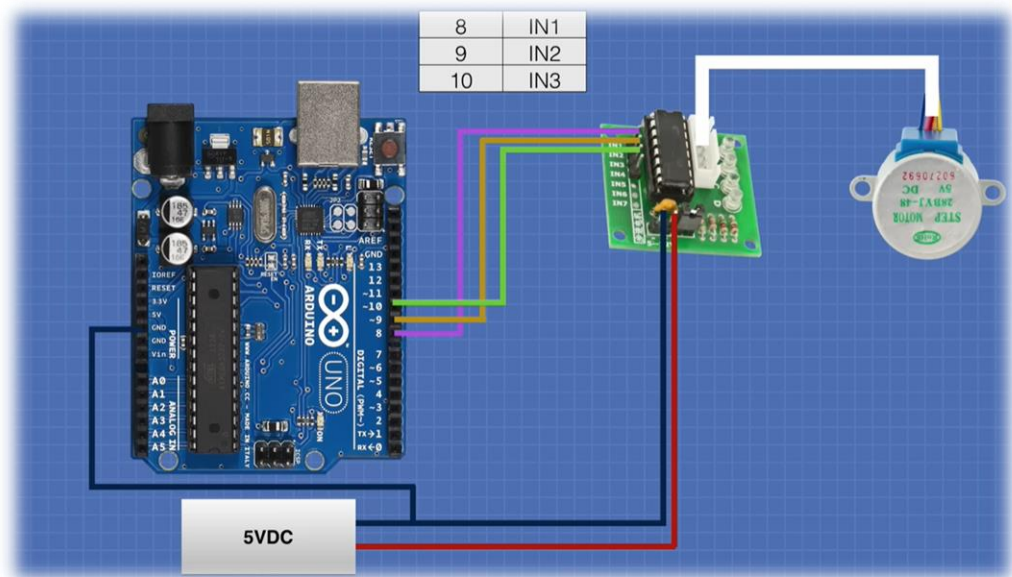
Controlul unui motor pas cu pas (stepper) 28BYJ-48 prin driverul ULN2003. Motorul execută o secvență repetată: câțiva pași lent CW (pentru a observa LED-urile driverului), jumătate de turație lent CW, jumătate de turație rapid CCW – demonstrând controlul vitezei și al direcției de rotație.

2. Lista de piese necesare:

- a. 1 x Placă **Arduino Uno**
- b. 1 x **Motor pas cu pas** 28BYJ-48 (5V, 64:1 gear reduction)
- c. 1 x **Driver motor** ULN2003 (cu modul de breakout)
- d. 1 x **Sursa de alimentare 5V** (recomandat separat pentru motor, nu direct de pe Arduino)
- e. **Fire de conexiune** (jumper wires)

3. Schema de conectare:

- a. **ULN2003 → Arduino:**
 - i. In1 → pin 8, In2 → pin 10, In3 → pin 9, In4 → pin 11
 - ii. Ordinea pinilor în cod este 8-10-9-11 (secvența 1-3-2-4) – necesară pentru pașii corecți ai motorului.
- b. **ULN2003 alimentare:**
 - i. VCC → 5V (preferabil sursa externă), GND → GND comun cu Arduino.
- c. **Motor 28BYJ-48:**
 - i. Se conectează direct la portul de 5 pini al modulului ULN2003 (conector dedicat).
- d. **Notă:** Biblioteca Stepper setează automat pinii ca OUTPUT – setup() poate rămâne gol.



4. Codul proiectului:

```
#include <Stepper.h>

const float STEPS_PER_REV = 32;
const float GEAR_RED = 64;
const float STEPS_PER_OUT_REV = STEPS_PER_REV * GEAR_RED;

int StepsRequired;

Stepper steppermotor(STEPS_PER_REV, 8, 10, 9, 11);

void setup()
{
  // Nothing (Stepper Library sets pins as outputs)
}

void loop()
{
  // Slow - 4-step CW sequence to observe lights on driver board
  steppermotor.setSpeed(1);
  StepsRequired = 4;
  steppermotor.step(StepsRequired);
  delay(2000);
}
```

```

// Rotate CW 1/2 turn slowly
StepsRequired = STEPS_PER_OUT_REV / 2;
steppermotor.setSpeed(100);
steppermotor.step(StepsRequired);
delay(1000);

// Rotate CCW 1/2 turn quickly
StepsRequired = - STEPS_PER_OUT_REV / 2;
steppermotor.setSpeed(700);
steppermotor.step(StepsRequired);
delay(2000);
}

```

5. Exerciții și Provocări

1. **Turație completă:** Modifică codul să facă o turație completă CW (360°) în loc de jumătate. Câți pași sunt necesari? (Hint: folosește STEPS_PER_OUT_REV fără să împarți la 2.)
2. **Control prin Serial:** Adaugă Serial.begin(9600) în setup() și trimite numărul de pași dorit prin Serial Monitor. Motorul să se rotească cu acel număr de pași la primirea comenzii.
3. **Viteză progresivă:** Creează o buclă for care crește viteza de la 50 la 700 RPM în trepte de 50, făcând câțiva pași la fiecare viteză. Observi cum se comportă motorul la viteze mari?

!!! Sfat: Valoarea din setSpeed() nu este viteza reală în RPM-uri absolute – este mai degrabă un parametru de timing. La viteze foarte mari (ex: 700+) motorul poate pierde pași sau să nu se miște deloc. Dacă motorul vibrează fără să se rotească, scade viteza!

PROIECT : Sistem Conveior cu Motor Pas-cu-Pas, Tastatură I2C și EEPROM | *Stepper Belt Conveyor with I2C Keypad & EEPROM*

[Link material video](#)

Obiectivul proiectului / *Project objective*

RO: Controlul unui motor pas-cu-pas unipolar (28BYJ-48) printr-un driver dedicat (ULN2003APG), conectat la un Arduino Uno R3. Motorul antrenează o curea transportoare de 800 mm, pe care este prinsă cu șuruburi o mică platformă dreptunghiulară. Sistemul deplasează platforma pe o distanță programabilă, menținând poziția în EEPROM. Tastatura I2C (PCF8574) permite pornirea/oprirea, schimbarea direcției, setarea distanței țintă și recalibrarea poziției. Un buzzer semnalizează evenimentele importante. **EN:** *Control a unipolar stepper motor (28BYJ-48) through a dedicated driver (ULN2003APG), connected to an Arduino Uno R3. The motor drives an 800 mm conveyor belt, onto which a small rectangular platform is screwed. The system moves the platform a programmable distance, storing the position in EEPROM. An I2C keypad (PCF8574) allows start/stop, direction change, target distance setting, and position recalibration. A buzzer signals key events.*

Lista de piese necesare / *Required components*

- **1 x** Placă Arduino Uno R3 / *Arduino Uno R3 Board*
- **1 x** Motor pas-cu-pas unipolar 28BYJ-48, montat mecanic pe cadrul conveiorului / *28BYJ-48 Unipolar Stepper Motor, mechanically mounted on the conveyor frame*
- **1 x** Curea transportoare (lungime 800 mm) antrenată de motor prin angrenaj / *Conveyor belt (800 mm length) driven by the motor through a gear*
- **1 x** Platformă dreptunghiulară mică, prinsă cu șuruburi de curea – transportată de-a lungul conveiorului / *Small rectangular platform, screwed onto the belt – carried along the conveyor*
- **1 x** Modul driver ULN2003APG (cu placă de test) / *ULN2003APG Driver Module (with breakout board)*
- **1 x** Tastatură matricială 4x4 cu modul I2C PCF8574 (adresă 0x20) / *4x4 Matrix Keypad with PCF8574 I2C module (address 0x20)*
- **1 x** Buzzer pasiv (5V, pin digital 4) / *Passive buzzer (5V, digital pin 4)*
- **1 x** Baterie externă 5V (power bank sau sursă 5V/1A) / *External 5V battery (power bank or 5V/1A supply)*
- **1 x** Breadboard și fire de conexiune (jumper wires) / *Breadboard and jumper wires*
- **1 x** Cablu USB pentru Arduino / *USB cable for Arduino*
- Biblioteci Arduino: EEPROM.h (built-in), Wire.h (built-in), I2CKeyPad.h (Rob Tillaart) / *Arduino Libraries: EEPROM.h (built-in), Wire.h (built-in), I2CKeyPad.h (by Rob Tillaart)*

Schema de conectare detaliată / Detailed wiring / connection diagram

Motor 28BYJ-48 → Driver ULN2003 / Motor Pins → Driver

Pin Motor / Motor Pin	Pin Driver / Driver Pin
Conector 5 pini (albastru/roz/galben/portocaliu/roșu)	Conector motor pe modulul ULN2003
Fir roșu (VCC +5V)	VCC pe modulul driver / VCC on driver module

Driver ULN2003 → Arduino Uno / Driver → Arduino

Pin Driver / Driver Pin	Pin Arduino / Arduino Pin
IN1	Pin digital 8
IN2	Pin digital 9
IN3	Pin digital 10
IN4	Pin digital 11
GND	GND pe Arduino / GND on Arduino

Tastatură I2C PCF8574 → Arduino Uno / I2C Keypad PCF8574 → Arduino Uno

Pin Tastatură / Keypad Pin	Pin Arduino / Arduino Pin
VCC	5V
GND	GND
SDA	Pin A4 (SDA)
SCL	Pin A5 (SCL)

Buzzer → Arduino Uno / Buzzer → Arduino Uno

Pin Buzzer / Buzzer Pin	Pin Arduino / Arduino Pin
(+) Semnal	Pin digital 4
(-) GND	GND

Alimentare / Power Supply

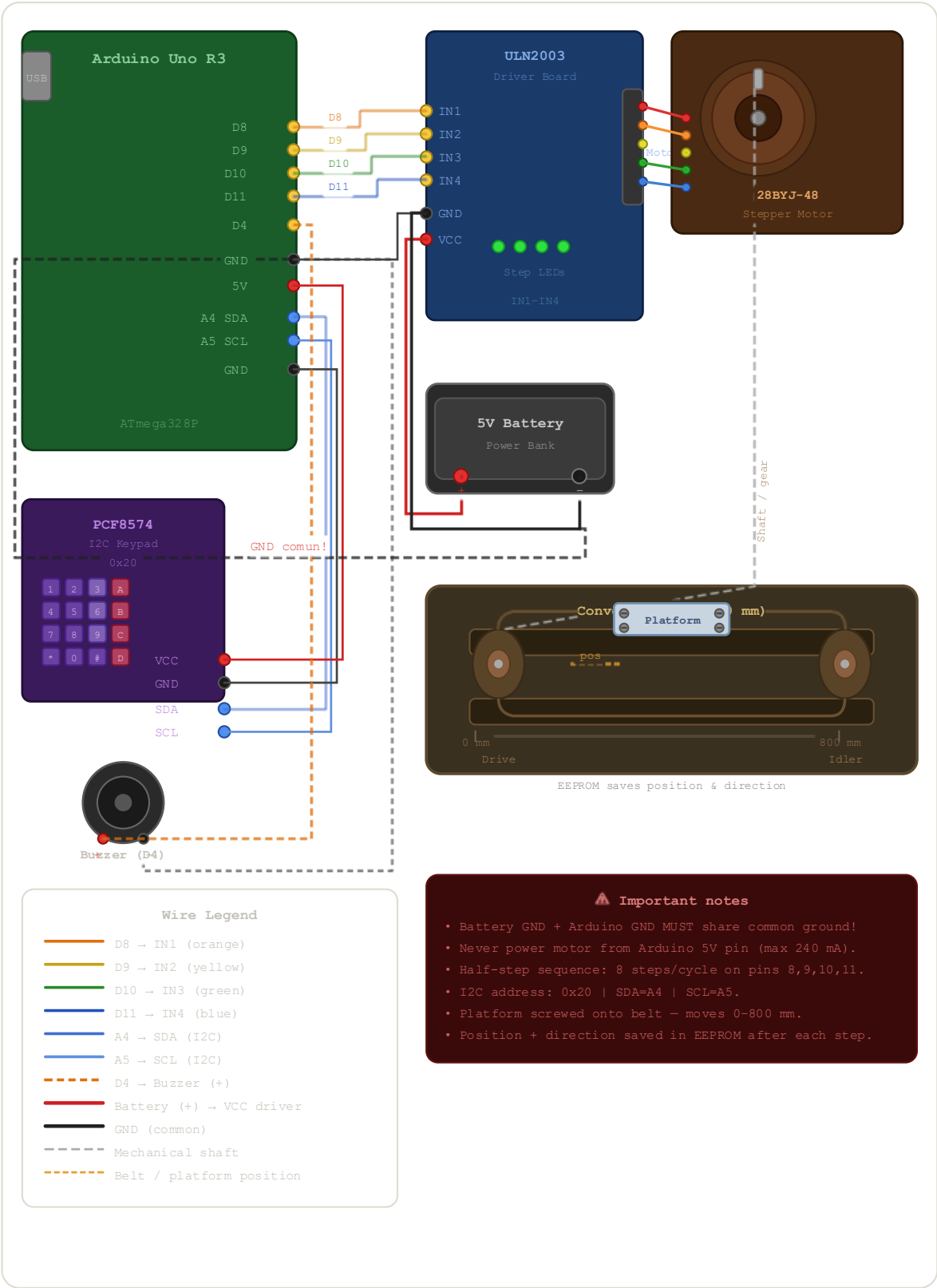
Sursă / Source	Destinație / Destination
Baterie 5V – fir roșu (+)	VCC pe modulul driver ULN2003
Baterie 5V – fir negru (-)	GND pe modulul driver ULN2003
GND baterie (comun)	GND Arduino (masă comună obligatorie!)

 **Atenție:** GND-ul bateriei și GND-ul Arduino TREBUIE conectate împreună (masă comună)! Fără aceasta, motorul nu va funcționa corect.  **Warning:** Battery GND and Arduino GND MUST be connected together (common ground)! Without this, the motor will not function correctly.

Comenzile tastaturii / Keypad commands

Tastă / Key	Funcție (RO)	Function (EN)
A	Schimbă direcția de mișcare	Toggle movement direction
B	Pornește / Oprește motorul	Start / Stop the motor
C	Introduce distanța Țintă (mm)	Set target distance (mm)
D	Setează poziția curentă (mm)	Set current position (mm)
0-9	Introduce cifre pentru distanță / poziție	Enter digits for distance / position
C (modul intr.)	Confirmă valoarea introdusă	Confirm entered value

Tastă / Key			Funcție (RO)		Function (EN)	
* (modul intr.)			Șterge ultima cifră (backspace)		Delete last digit (backspace)	
# (modul intr.)			Anulează introducerea datelor		Cancel data entry	



Codul proiectului / *Project code*

RO: Codul complet al proiectului, include toate funcțiile: beep(), saveState(), opreste(), faUnPas(), proceseazaTasta(), setup() și loop(). **EN:** Complete project code, including all functions: beep(), saveState(), opreste(), faUnPas(), proceseazaTasta(), setup() and loop().

```
#include <EEPROM.h>

#include <Wire.h>

#include <I2CKeyPad.h>


const int INTERNAL_STEPS_PER_REVOL = 32;

const int GEAR_REDUCTION = 64;


const float STEPS_PER_MM = 51.2;

const int BELT_LENGTH = 800;

const int STEP_SIZE = 50;


const int BUZZER_PIN = 4;


const int motorPins[4] = {8, 9, 10, 11};


const int stepSequence[8][4] = {

    {1, 0, 0, 0},

    {1, 1, 0, 0},

    {0, 1, 0, 0},

    {0, 1, 1, 0},

    {0, 0, 1, 0},

    {0, 0, 1, 1},

    {0, 0, 0, 1},

    {1, 0, 0, 1}

};


int currentStep = 0;

long stepsRamasi = 0;
```

```
int directiePas = 1;

float stepsPerMlmtr = STEPS_PER_MM;

int pozitie = 0;

bool running = false;

long distantaTarget = 0;

long distantaParcursa = 0;

bool setandDistanta = false;

bool setandPozitie = false;

String inputDistanta = "";

const int ADDR_POZITIE = 0;

const int ADDR_DIRECTIE = 2;

const int STEP_DELAY_US = 1000;

unsigned long lastStepTime = 0;

const uint8_t KEYPAD_ADDRESS = 0x20;

I2CKeyPad keypad(KEYPAD_ADDRESS);

char keys[] = "123A456B789C*0#D";

char lastKey = 0;

void beep(int ori, int durata) {
    for (int i = 0; i < ori; i++) {
        tone(BUZZER_PIN, 400);

        delay(durata);

        noTone(BUZZER_PIN);

        delay(durata);
    }
}
```

```

    }

}

void saveState() {

    EEPROM.put(ADDR_POZITIE, pozitie);

    int directie = (stepsPerMlmtr > 0) ? 1 : -1;

    EEPROM.put(ADDR_DIRECTIE, directie);

}

void opreste() {

    running = false;

    stepsRamasi = 0;

    beep(3, 300);

    Serial.print("Distanta parcursa: ");

    Serial.print(distantaParcursa);

    Serial.println(" mm. Oprit.");

}

void faUnPas() {

    if (stepsRamasi == 0) return;

    unsigned long now = micros();

    if (now - lastStepTime < STEP_DELAY_US) return;

    lastStepTime = now;

    currentStep = (currentStep + directiePas + 8) % 8;

    for (int i = 0; i < 4; i++) {

        digitalWrite(motorPins[i], stepSequence[currentStep][i]);

    }

    stepsRamasi--;

```

```
if (stepsRamasi == 0) {

    int nextPozitie = pozitie + (stepsPerMlmtr > 0 ? STEP_SIZE : -STEP_SIZE);

    distantaParcursa += STEP_SIZE;

    bool laCapat = false;

    if (nextPozitie >= BELT_LENGTH) {

        nextPozitie = BELT_LENGTH;

        stepsPerMlmtr = -STEPS_PER_MM;

        laCapat = true;

    } else if (nextPozitie < 0) {

        nextPozitie = 0;

        stepsPerMlmtr = STEPS_PER_MM;

        laCapat = true;

    }

    pozitie = nextPozitie;

    saveState();

    if (laCapat) {

        beep(5, 80);

    }

    Serial.print("Pozitie: ");

    Serial.print(pozitie);

    Serial.print(" mm | Directie: ");

    Serial.print(stepsPerMlmtr > 0 ? "inainte" : "inapoi");

    Serial.print(" | Parcurs: ");

    Serial.print(distantaParcursa);

    Serial.print("/");
```

```

Serial.print(distantaTarget);

Serial.println(" mm");

if (distantaTarget > 0 && distantaParcursa >= distantaTarget) {
    opreste();
    return;
}

if (running) {
    stepsRamasi = (long) (STEP_SIZE * STEPS_PER_MM);
    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
}

void proceseazaTasta(char key) {
    if (setandDistanta) {
        if (key >= '0' && key <= '9') {
            inputDistanta += key;

            Serial.print(setandPozitie ? "Pozitie introdusa: " : "Distanta introdusa: ");
            Serial.println(inputDistanta);
        }
        else if (key == 'C') {
            long valoare = inputDistanta.toInt();

            if (setandPozitie) {
                if (valoare >= 0 && valoare <= BELT_LENGTH) {
                    pozitie = (int)valoare;

                    saveState();

                    Serial.print("Pozitie setata la: ");
                    Serial.print(pozitie);
                    Serial.println(" mm");
                } else {

```

```

        Serial.print("Pozitie invalida! Trebuie intre 0 si ");

        Serial.println(BELT_LENGTH);

    }

    setandPozitie = false;
} else {

    if (valoare > 0) {

        distantaTarget = valoare;

        distantaParcursa = 0;

        Serial.print("Distanța target setata: ");

        Serial.print(distantaTarget);

        Serial.println(" mm");

    } else {

        Serial.println("Distanța invalida!");

    }

}

inputDistanța = "";

setandDistanța = false;

}

else if (key == '*') {

    if (inputDistanța.length() > 0) {

        inputDistanța.remove(inputDistanța.length() - 1);

        Serial.print(setandPozitie ? "Pozitie introdusa: " : "Distanța introdusa: ");

        Serial.println(inputDistanța);

    }

}

else if (key == '#') {

    inputDistanța = "";

    setandDistanța = false;

    setandPozitie = false;

    Serial.println("Anulat.");

}

return;

```

```

}

if (key == 'A') {
    stepsPerMlmtr = -stepsPerMlmtr;
    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
    saveState();
    beep(2, 100);
    Serial.println(stepsPerMlmtr > 0 ? "Directie: inainte" : "Directie: inapoi");
}
else if (key == 'B') {
    running = !running;
    beep(1, 750);
    Serial.println(running ? "Pornit" : "Oprit");
    if (running && stepsRamasi == 0) {
        if (distantaParcursa >= distantaTarget) {
            distantaParcursa = 0;
        }
        stepsRamasi = (long) (STEP_SIZE * STEPS_PER_MM);
        directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
    }
    if (!running) {
        stepsRamasi = 0;
    }
}
else if (key == 'C') {
    setandDistanta = true;
    setandPozitie = false;
    inputDistanta = "";
    Serial.println("Introdu distanta (mm) si apasa C pentru confirmare, # pentru anulare:");
}
else if (key == 'D') {
    setandDistanta = true;

```

```

        setandPozitie = true;

        inputDistanța = "";

        Serial.println("Introdu poziția curentă (mm) și apasă C pentru confirmare, # pentru
anulare:");

    }

}

void setup() {

    Serial.begin(9600);

    Wire.begin();

    pinMode(BUZZER_PIN, OUTPUT);

    digitalWrite(BUZZER_PIN, LOW);

    for (int i = 0; i < 4; i++) {

        pinMode(motorPins[i], OUTPUT);

        digitalWrite(motorPins[i], LOW);

    }

    if (!keypad.begin()) {

        Serial.println("Keypad negăsit! Verifică conexiunile.");

        while (1);

    }

    int savedPozitie;

    int savedDirectie;

    EEPROM.get(ADDR_POZITIE, savedPozitie);

    EEPROM.get(ADDR_DIRECTIE, savedDirectie);

    bool pozitieValida = (savedPozitie >= 0 && savedPozitie <= BELT_LENGTH);

    bool directieValida = (savedDirectie == 1 || savedDirectie == -1);

```

```

if (pozitieValida && directieValida) {
    pozitie = savedPozitie;
    stepsPerMlmtr = STEPS_PER_MM * savedDirectie;
    Serial.print("Pozitie restaurata: ");
    Serial.print(pozitie);
    Serial.println(" mm");
    Serial.print("Directie: ");
    Serial.println(savedDirectie == 1 ? "inainte" : "inapoi");
} else {
    pozitie = 0;
    stepsPerMlmtr = STEPS_PER_MM;
    Serial.println("Prima pornire, start de la 0.");
    saveState();
}

Serial.println("A=directie, B=stop/start, C=seteaza distanta, D=seteaza pozitie");
}

void loop() {
    char key = 0;

    if (keypad.isPressed()) {
        uint8_t idx = keypad.getKey();
        if (idx != I2C_KEYPAD_NOKEY) {
            key = keys[idx];
        }
    }

    if (key != 0 && lastKey == 0) {
        Serial.print("Tasta apasata: ");
        Serial.println(key);
        proceseazaTasta(key);
    }
}

```

```

}

lastKey = key;

faUnPas();

}

```

Exerciții și provocări / Exercises & challenges

1. Calculul pașilor pe mm / 1. Steps per mm calculation

RO: Motorul 28BYJ-48 are 32 de pași interni pe rotație și un raport de reducere 64:1. Verifică: $32 \times 64 = 2048$ pași/rotație. În modul half-step (8 pași/ciclu) și cu diametrul de angrenaj dat, STEPS_PER_MM = 51.2. Calculează diametrul rodanței folosite. **EN:** The 28BYJ-48 has 32 internal steps per revolution and a 64:1 gear ratio. Verify: $32 \times 64 = 2048$ steps/revolution. In half-step mode (8 steps/cycle) with the given gear diameter, STEPS_PER_MM = 51.2. Calculate the pitch diameter of the gear used.

2. Schimbarea lungimii curelei / 2. Change belt length

RO: Modifică constanta BELT_LENGTH de la 800 la 500 mm și STEP_SIZE de la 50 la 25 mm. Ce se întâmplă cu frecvența bip-urilor la capăt de cursă? Ajustează logica de inversare a direcției. **EN:** Change the BELT_LENGTH constant from 800 to 500 mm and STEP_SIZE from 50 to 25 mm. What happens to the end-of-travel beep frequency? Adjust the direction-reversal logic accordingly.

3. Comandă Serial / 3. Serial control

RO: Extinde codul astfel încât comenzile A, B, C, D să poată fi trimise și prin Serial Monitor (9600 baud). La primirea comenzii 'S' prin serial, oprește imediat motorul și afișează poziția în mm. **EN:** Extend the code so that A, B, C, D commands can also be sent via Serial Monitor (9600 baud). Upon receiving the 'S' command via serial, immediately stop the motor and print the current position in mm.

4. Viteză variabilă / 4. Variable speed

RO: STEP_DELAY_US controlează viteza motorului (implicit 1000 μs între pași). Adaugă tastele 1, 2, 3 de pe tastatură pentru a selecta viteze de 500 μs, 1000 μs și 2000 μs. Observă efectele asupra torsiunii. **EN:** STEP_DELAY_US controls motor speed (default 1000 μs between steps). Add keypad keys 1, 2, 3 to select speeds of 500 μs, 1000 μs, and 2000 μs. Observe the effects on torque.

5. EEPROM și restaurarea stării / 5. EEPROM & state restoration

RO: Desconectează Arduino-ul după ce motorul a ajuns la poziția 400 mm. Reconectează-l și verifică prin Serial Monitor că poziția și direcția sunt restaurate corect din EEPROM. Adaugă un mesaj suplimentar de confirmare la pornire. **EN:** Disconnect the Arduino after the motor reaches

position 400 mm. Reconnect it and verify through Serial Monitor that position and direction are correctly restored from EEPROM. Add an extra confirmation message on startup.

6. Monitorizare Serial / 6. Serial Monitor

RO: Deschide Serial Monitor (Ctrl+Shift+M), setează viteza la 9600 baud. Apasă B pentru pornire și observă mesajele de poziție afisate la fiecare 50 mm. Interpretează mesajul "Distanța parcursă: X mm. Oprit." **EN:** *Open Serial Monitor (Ctrl+Shift+M), set baud rate to 9600. Press B to start and observe position messages printed every 50 mm. Interpret the message "Distanța parcursă: X mm. Oprit."*

Sfaturi importante / Important tips

!!! Sfat / Tip: Motorul 28BYJ-48 folosește modul half-step cu 8 pași pe ciclu (secvența stepSequence[8][4]). Dacă ordinea pinilor este greșită, motorul va vibra fără să se rotească. / *The 28BYJ-48 uses half-step mode with 8 steps per cycle (stepSequence[8][4] array). If the pin order is wrong, the motor will vibrate without rotating.*

!!! Sfat / Tip: Nu alimenta motorul direct de la pinii de 5V ai Arduino-ului! Folosește întotdeauna bateria externă. Motorul poate consuma până la 240 mA, ceea ce poate deteriora placa Arduino. / *Never power the motor directly from the Arduino 5V pin! Always use the external battery. The motor can draw up to 240 mA, which may damage the Arduino board.*

!!! Sfat / Tip: Biblioteca I2CKeyPad necesită instalare manuală din Library Manager (autor: Rob Tillaart). Fără aceasta, codul nu se va compila. / *The I2CKeyPad library must be installed manually from Library Manager (author: Rob Tillaart). Without it, the code will not compile.*

!!! Sfat / Tip: Valoarea poziției și direcția sunt salvate în EEPROM după fiecare pas de 50 mm. La repornire, sistemul reia automat de la ultima poziție cunoscută. / *Position and direction are saved to EEPROM after each 50 mm step. On restart, the system automatically resumes from the last known position.*

!!! Sfat / Tip: Asigură-te că masa comună (GND) este conectată între Arduino, modulul driver ULN2003 și bateria externă. Fără aceasta, semnalele de control nu vor funcționa corect. / *Make sure the common ground (GND) is connected between the Arduino, the ULN2003 driver module, and the external battery. Without this, control signals will not work correctly.*

PROIECT FINAL: Conveyor Belt with Ejection Platform

Project Objective: Design and build an automated conveyor belt system controlled by an Arduino. The belt moves a platform across a 1,650 mm track using a stepper motor. When the platform reaches the end of the belt, a servo motor physically ejects an object. The system features a 4x4 keypad for user commands, a 16x2 LCD for live status feedback, a buzzer for audio alerts, and EEPROM memory to save the current position and direction across power cycles.

Required Parts List: 1 x **Arduino Uno** (or compatible ATmega328P board)

1 x **28BYJ-48 Stepper Motor** with ULN2003 driver module

1 x **SG90 / MG90S Servo Motor** (for the ejection arm)

1 x **16x2 LCD Display** (standard 4-bit parallel interface)

1 x **4x4 Matrix Keypad** (I2C via PCF8574, address 0x20)

1 x **Active Buzzer**

1 x **Breadboard** and jumper wires

1 x **9V power supply** (USB or external)

- Wiring Diagram: **LCD (LiquidCrystal, 4-bit mode):**
- RS → Pin 6
- E → Pin 7
- D4 → Pin 12 | D5 → Pin 13 | D6 → Pin 3 | D7 → Pin 5
- VSS, RW, K → GND | VDD, A → 5V (use 220 Ω resistor on A for backlight)
- Stepper Motor (via ULN2003): IN1 → Pin 8 | IN2 → Pin 9 | IN3 → Pin 10 | IN4 → Pin 11
- Motor VCC → 5V | GND → GND
- Servo Motor: Signal → Pin A0 | VCC → 5V | GND → GND
- I2C Keypad (PCF8574):
- SDA → A4 | SCL → A5 | VCC → 5V | GND → GND
- Confirm I2C address is 0x20 (all address jumpers open)
- Buzzer: Positive → Pin 4 | Negative → GND

Project Code:

```
#include <EEPROM.h>
#include <Wire.h>
#include <I2CKeyPad.h>
#include <LiquidCrystal.h>
#include <Servo.h>
```

```

// ===== LCD =====
LiquidCrystal lcd(6, 7, 12, 13, 3, 5);
// ===== SERVO =====
Servo ejectServo;
const int SERVO_PIN = A0;
const int SERVO_IDLE = 0;
const int SERVO_THROW = 90;
// ===== MOTOR =====
const float STEPS_PER_MM = 51.2;
const int BELT_LENGTH = 1650;
const int STEP_SIZE = 50;
const int BUZZER_PIN = 4;
const int motorPins[4] = {8, 9, 10, 11};
const int stepSequence[8][4] = {
{1,0,0,0},{1,1,0,0},{0,1,0,0},{0,1,1,0},
{0,0,1,0},{0,0,1,1},{0,0,0,1},{1,0,0,1}
};
int currentStep = 0;
long stepsRamasi = 0;
int directiePas = 1;
float stepsPerMlmtr = STEPS_PER_MM;
int pozitie = 0;
bool running = false;
long distantaTarget = 0;
long distantaParcursa = 0;
bool setandDistanta = false;
bool setandPozitie = false;
String inputDistanta = "";
const int ADDR_POZITIE = 0;
const int ADDR_DIRECTIE = 2;
const int STEP_DELAY_US = 1000;
unsigned long lastStepTime = 0;
const uint8_t KEYPAD_ADDRESS = 0x20;
I2CKeyPad keypad(KEYPAD_ADDRESS);
char keys[] = "123A456B789C*0#D";
char lastKey = 0;
unsigned long lastLCDUpdate = 0;
// --- LCD update helpers ---

```

```

void updateLCD(String line1, String line2) {
  lcd.clear();
  lcd.setCursor(0, 0); lcd.print(line1);
  lcd.setCursor(0, 1); lcd.print(line2);
}

void afisareStatus() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("P:"); lcd.print(pozitie); lcd.print("mm ");
  lcd.print(running ? "RUN" : "STOP");
  lcd.setCursor(0, 1);
  lcd.print(stepsPerMlmtr > 0 ? "DIR:>" : "DIR:<");
  lcd.print(" T:"); lcd.print(distantaTarget - distantaParcursa);
}

void beep(int times, int duration) {
  for (int i = 0; i < times; i++) {
    tone(BUZZER_PIN, 400); delay(duration);
    noTone(BUZZER_PIN); delay(duration);
  }
}

void saveState() {
  EEPROM.put(ADDR_POZITIE, pozitie);
  int dir = (stepsPerMlmtr > 0) ? 1 : -1;
  EEPROM.put(ADDR_DIRECTIE, dir);
}

void aruncaObiect() { // Eject object via servo
  bool wasRunning = running;
  running = false;
  for (int i = 0; i < 4; i++) digitalWrite(motorPins[i], LOW);
  updateLCD("OBJECT ARRIVED", "EJECTING...");
  beep(1, 200);
  delay(1000);
  ejectServo.write(SERVO_THROW);
  delay(600);
  ejectServo.write(SERVO_IDLE);
  delay(500);
}

```



Cofinanțat de
Uniunea Europeană



Start în Robotică cu Erasmus+
2025-1-RO01-KA122-VET-000340573

```
beep(2, 80);  
updateLCD("EJECTION", "COMPLETE");  
  
delay(700);  
if (wasRunning) {  
    running = true;  
    if (stepsRamasi == 0)  
        stepsRamasi = (long)(STEP_SIZE * STEPS_PER_MM);  
    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;  
}  
  
void opreste() { // Stop conveyor  
    running = false; stepsRamasi = 0;  
    beep(3, 300);  
    updateLCD("CONVEYOR", "STOPPED");  
}  
  
void faUnPas() { // Execute one motor step  
    if (stepsRamasi == 0) return;  
    unsigned long now = micros();  
    if (now - lastStepTime < STEP_DELAY_US) return;  
    lastStepTime = now;  
    currentStep = (currentStep + directiePas + 8) % 8;  
    for (int i = 0; i < 4; i++)  
        digitalWrite(motorPins[i], stepSequence[currentStep][i]);  
    stepsRamasi--;  
    if (stepsRamasi == 0) {  
        int next = pozitie + (stepsPerMlmtr > 0 ? STEP_SIZE : -STEP_SIZE);  
        distantaParcursa += STEP_SIZE;  
        bool atEnd = false;  
        if (next >= BELT_LENGTH) {  
            next = BELT_LENGTH;  
            aruncaObiect();  
            stepsPerMlmtr = -STEPS_PER_MM;  
        }  
    }  
}
```

DISCLAIMER: „Finanțat de Uniunea Europeană. Punctele de vedere și opiniile exprimate aparțin, însă, exclusiv autorului (autorilor) și nu reflectă neapărat punctele de vedere și opiniile Uniunii Europene sau ale Agenției Naționale pentru Programe Comunitare în Domeniul Educației și Formării Profesionale (ANPCDEFP). Nici Uniunea Europeană și nici ANPCDEFP nu pot fi ținute răspunzătoare pentru acestea.”



Start în Robotică cu Erasmus+
2025-1-RO01-KA122-VET-000340573

```
atEnd = true;
} else if (next < 0) {
next = 0;
stepsPerMlmtr = STEPS_PER_MM;
atEnd = true;
}
pozitie = next;
saveState();
if (atEnd) beep(5, 80);
if (distanțaTarget > 0 && distanțaParcursa >= distanțaTarget) {
oprește(); return;
}
if (running) {
stepsRamasi = (long)(STEP_SIZE * STEPS_PER_MM);
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
}
}

void proceseazaTasta(char key) { // Handle keypad input
if (key == '*') { updateLCD("MANUAL EJECT","ACTIVATED"); aruncaObiect();
return; }

if (setandDistanța) {
if (key >= '0' && key <= '9') { inputDistanța += key; /* update LCD */ }
else if (key == 'C') {
long val = inputDistanța.toInt();
if (setandPozitie) {
if (val >= 0 && val <= BELT_LENGTH) { pozitie = (int)val; saveState(); }
setandPozitie = false;
} else {
if (val > 0) { distanțaTarget = val; distanțaParcursa = 0; }
}
inputDistanța = ""; setandDistanța = false; delay(1200);
}
}
```



Start în Robotică cu Erasmus+
2025-1-RO01-KA122-VET-000340573

```
else if (key == '#') { inputDistanța = ""; setandDistanța = false;
setandPozitie = false; delay(1000); }

return;
}

if (key == 'A') { // Reverse direction
stepsPerMlmtr = -stepsPerMlmtr;
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
saveState(); beep(2, 100);
} else if (key == 'B') { // Start / Stop toggle
running = !running; beep(1, 750);
if (running && stepsRamasi == 0) {
if (distanțaParcursa >= distanțaTarget) distanțaParcursa = 0;
stepsRamasi = (long)(STEP_SIZE * STEPS_PER_MM);
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
if (!running) stepsRamasi = 0;
delay(1000);
} else if (key == 'C') { // Set target distance
setandDistanța = true; setandPozitie = false; inputDistanța = "";
updateLCD("SET DISTANCE", "Enter mm");
} else if (key == 'D') { // Set current position
setandDistanța = true; setandPozitie = true; inputDistanța = "";
updateLCD("SET POSITION", "Enter mm");
}
}

void setup() {
Serial.begin(9600); Wire.begin();
lcd.begin(16, 2);
updateLCD("CONVEYOR", "BOOTING...");
pinMode(BUZZER_PIN, OUTPUT); digitalWrite(BUZZER_PIN, LOW);
for (int i = 0; i < 4; i++) { pinMode(motorPins[i], OUTPUT);
digitalWrite(motorPins[i], LOW); }
ejectServo.attach(SERVO_PIN); ejectServo.write(SERVO_IDLE);
if (!keypad.begin()) { lcd.clear(); lcd.print("KEYPAD ERROR"); while(1); }
```



Start în Robotică cu Erasmus+
2025-1-RO01-KA122-VET-000340573

```
int savedPos, savedDir;
EEPROM.get(ADDR_POZITIE, savedPos); EEPROM.get(ADDR_DIRECTIE, savedDir);
if (savedPos >= 0 && savedPos <= BELT_LENGTH && (savedDir == 1 || savedDir == -1)) {
pozitie = savedPos; stepsPerMlmtr = STEPS_PER_MM * savedDir;
} else { pozitie = 0; stepsPerMlmtr = STEPS_PER_MM; saveState(); }
updateLCD("SYSTEM READY", "READY"); delay(1500);
}

void loop() {
char key = 0;
if (keypad.isPressed()) {
uint8_t idx = keypad.getKey();
if (idx != I2C_KEYPAD_NOKEY) key = keys[idx];
}
if (key != 0 && lastKey == 0) proceseazaTasta(key);
lastKey = key;
faUnPas();
if (millis() - lastLCDUpdate > 400 && !setandDistanta) {
afisareStatus(); lastLCDUpdate = millis();
}
}
```