Cos de gunoi inteligent

Introducere

Ti-a fost vreodata lene sa deschizi cosul de gunoi si ai vrut sa se deschida singur? Nu? Minti, cu totii vrem asta. De aceea am venit cu ideea sa fac un cos de gunoi inteligent ce se deschide automat atunci cand te apropii cu mana de el si contorizeaza cate lucruri ai aruncat in el, perfect pentru a stii cat de plin este cosul.

Ideea a pornit de la faptul ca in baie am un cos de gunoi ce normal ar trebui sa se deschida prin apasarea cu piciorul pe o placa de presiune, dar mecanismul nu mai functioneaza.

Cum functioneaza exact? Folosind un senzor ultrasonic si un servo motor, capacul cosului s-ar deschide automat, iar folosind un display LCD am contoriza cate obiecte au fost aruncate in cos. Scopul este de a crea un dispozitiv practic si igienic, ce ar fi util pentru oricine.

Descriere generală

Sistemul este compus din urmatoarele:

Module hardware:

- **ATmega328P-XMINI** - microcontrollerul principal, creierul intregii operatiuni, cel care primeste datele de la senzor si decide ce se intampla mai departe
- **HC-SR04P** - senzorul ultrasonic care detecteaza daca te-ai apropiat cu mana de cos
- **SG90** - servo motorul care deschide si inchide capacul cosului
- **LCD 1602 + modul I2C** - display-ul care iti arata cate obiecte ai aruncat in cos
- **Buzzer activ 5V** - emite un sunet atunci cand capacul se termina de inchis
- **LED rosu + verde** - indica starea cosului, deschis sau inchis
- **Rezistente 1kΩ** - protejeaza LED-urile de la a se arde

Module software: (prone to changes)

- **Driver TWI/I2C** - se ocupa de comunicatia dintre microcontroller si LCD
- **Modul HC-SR04** - gestioneaza intreruperile externe de pe pinul ECHO si calculeaza distanta pana la mana
- **Control servo PWM** - foloseste Timer1 in modul Fast PWM pentru a controla unghiul capacului
- **Logica principala** - decide cand sa deschida capacul, cand sa il inchida, incrementeaza contorul si actualizeaza display-ul

Hardware Design

Lista Componente

Componenta	Model / Specificatii	Cantitate
Microcontroller	ATmega328P-XMINI	1
Breadboard	400 puncte	1
Fire tata-tata	20cm, dupont	~8
Fire mama-tata	20cm, dupont	~8
Rezistenta	1KΩ	4
Senzor ultrasonic	HC-SR04P	1
LED rosu	5mm, 3.5V	2
LED verde	5mm, 3.5V	2
Servo motor	SG90, 5V	1
Buzzer activ	5V, 2300Hz	1
Display LCD + modul I2C	LCD 1602, interfata I2C	1

Componente folosite si Rolul lor

Componenta	Rol in proiect
Microcontroller ATmega328P-XMINI	Creierul sistemului. Ruleaza codul in C pur, citeste senzorul prin pooling si controleaza actuatorii (timere, registre).
Senzor ultrasonic HC-SR04P	Modulul de input. Masoara distanta pana la mana utilizatorului prin emiterea si receptia de unde sonore.
Servo motor SG90	Actuatorul principal. Actioneaza mecanic tija (push-rod) pentru a ridica si cobori capacul cosului.
Display LCD 1602 + modul I2C	Interfata vizuala. Afiseaza mesaje de stare (“Apropie mana”, “Capac Deschis!”) si contorul de utilizari.
Buzzer activ	Feedback sonor. Emite un semnal acustic scurt (un “beep”) in momentul in care capacul se inchide.

Pinii folositi

Pin ATmega328P	Eticheta	Componenta	Rol
PD2	BUZZER	Buzzer Activ	Iesire digitala - Controlul on/off al semnalului sonor
PD3	LED_ROSU	LED-uri Rosii	Iesire digitala - Activarea indicatorului de capac deschis
PD4	LED_VERDE	LED-uri Verzi	Iesire digitala - Activarea indicatorului de stare de veghe
PD5	US_ECHO	Senzor HC-SR04	Intrare digitala - Masurarea duratei pulsului de ecou reflectat
PD6	US_TRIG	Senzor HC-SR04	Iesire digitala - Generarea pulsului de declansare de 10us

PB1	SERVO_PWM	Servo SG90	Iesire PWM - Generat de Timer 1 (OC1A) la 50Hz pentru unghi
PC4	SDA	Modul LCD I2C	Linie de date bidirectionala (interfata hardware TWI)
PC5	SCL	Modul LCD I2C	Linie de ceas/sincronizare (interfata hardware TWI)
VCC / +5V	VCC	Toate	Magistrala pozitiva de alimentare (5V din USB-ul placii)
GND	GND	Toate	Masa comuna a circuitului (intoarcerea curentului la sursa)

Schema electrica

Nu am gasit in KiCad placa ATmega328P Xplained Mini si am folosit echivalentul ATmega328P-P

Schema prezinta:

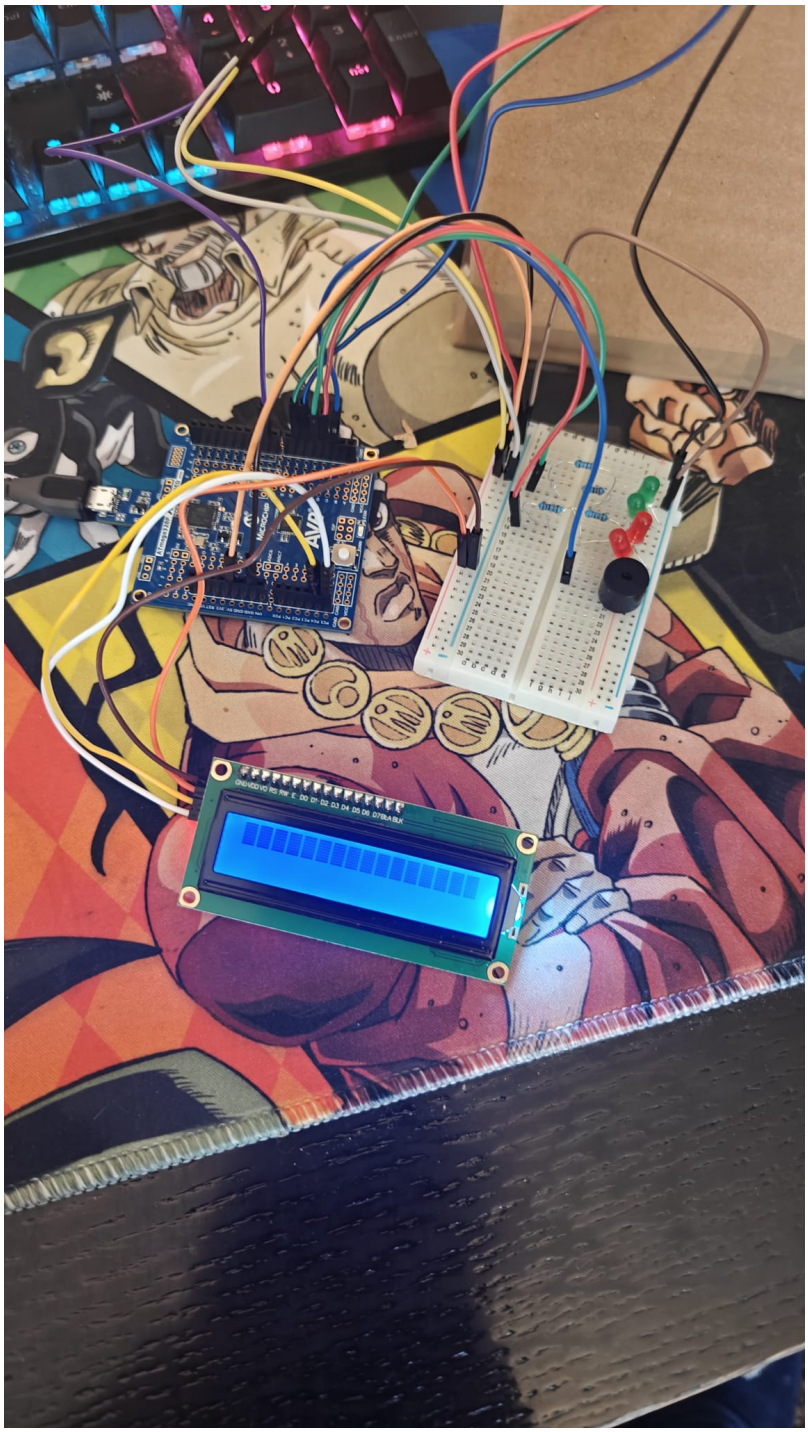
- ATmega328P Xplained Mini in centru, ca unitate de control
- HC-SR04 conectat pe pinii TRIG (PD6) si ECHO (PD5) - reprezinta senzorul ultrasonic pentru masurarea distantei
- Servo motor SG90 conectat pe pinul PWM (PB1) - reprezinta actuatorul folosit pentru ridicarea capacului
- LCD I2C conectat pe magistrala I2C (SDA=PC4, SCL=PC5) cu alimentare de 5V - afiseaza starea si numarul de utilizari
- Buzzer conectat pe PD2 pentru activare si semnalizare sonora scurta la inchidere.
- LED-uri verzi conectate in paralel pe pinul PD4 prin rezistente de 220 Ohm - indicator pentru starea de repaus
- LED-uri rosii conectate in paralel pe pinul PD3 prin rezistente de 220 Ohm - indicator pentru starea activa (capac deschis)

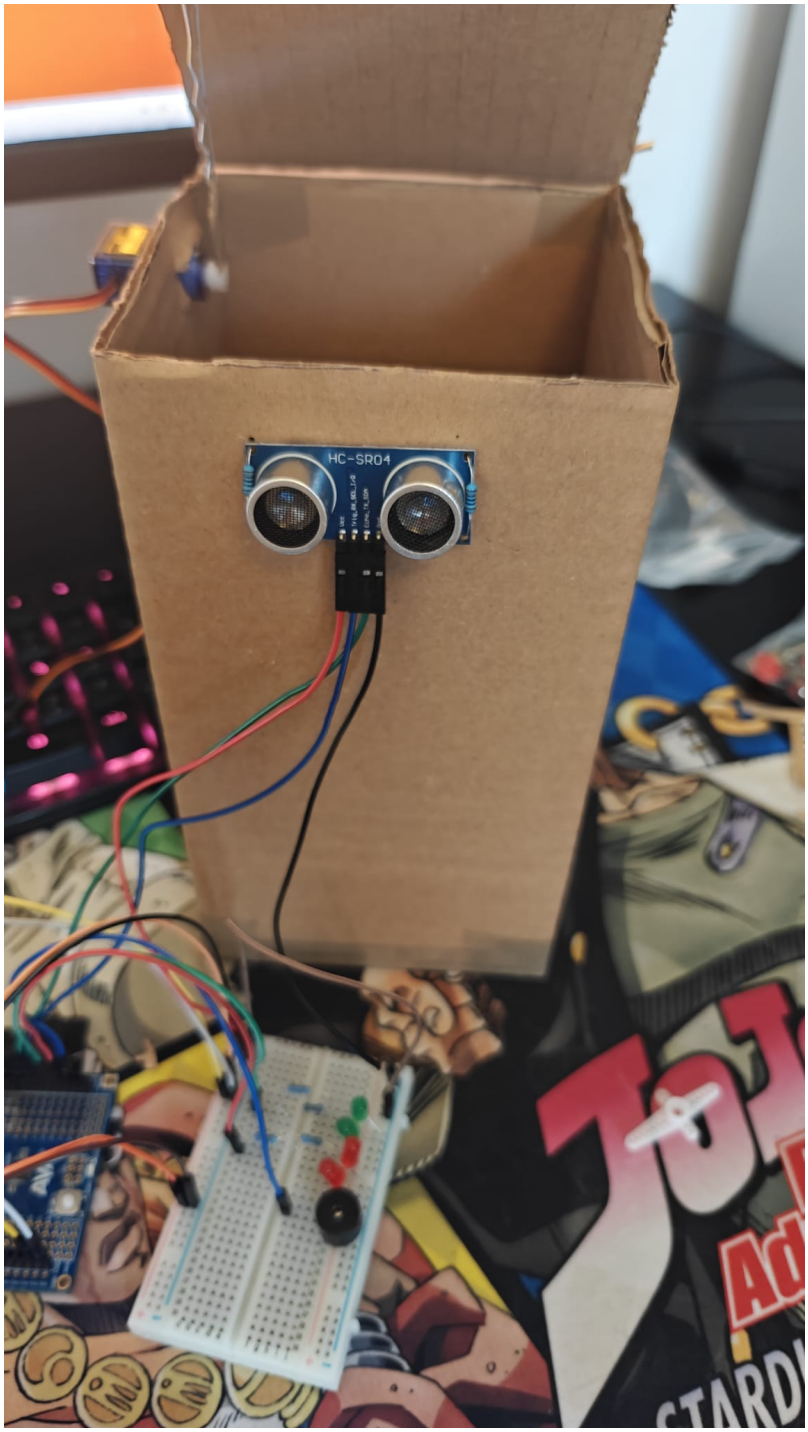
Principiu de functionare

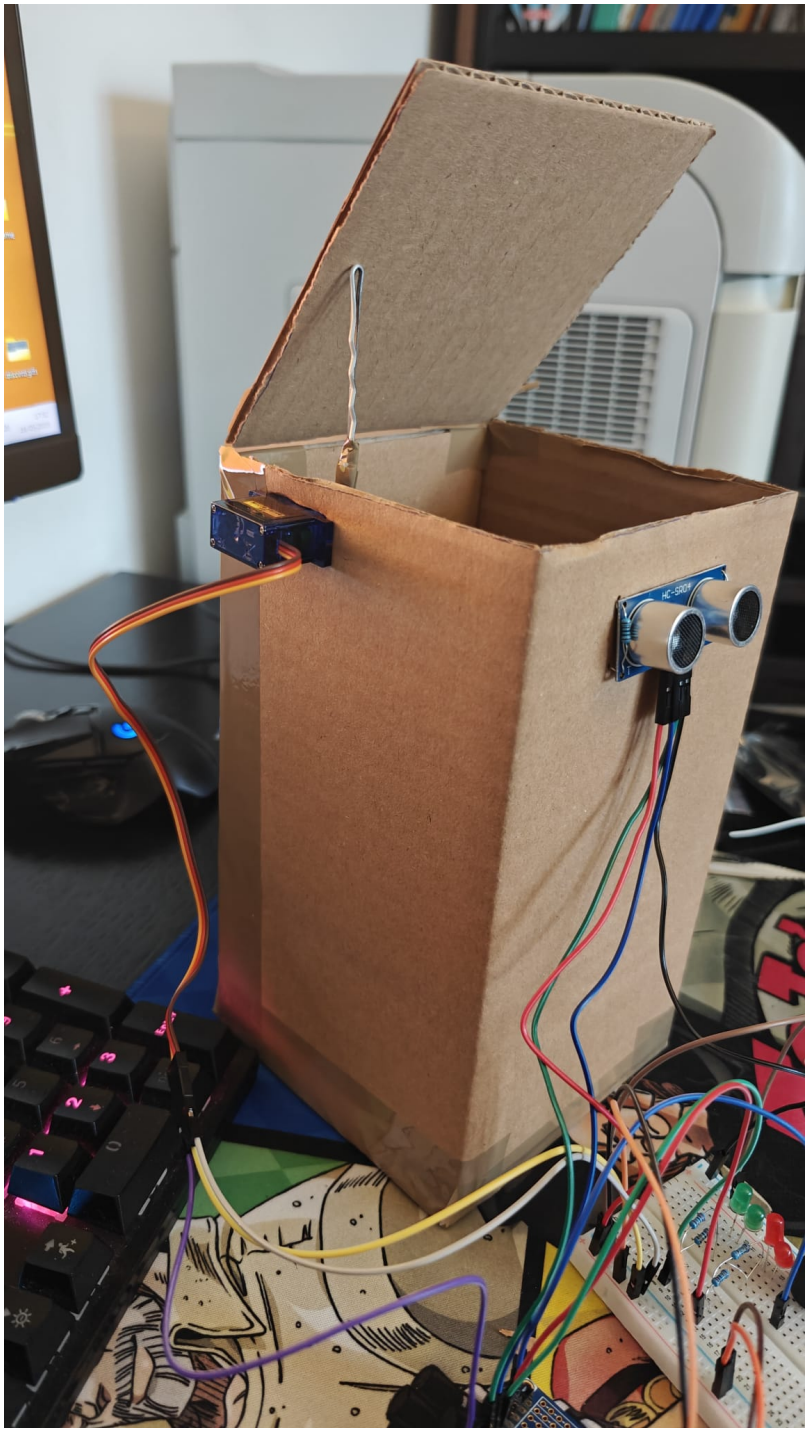
Sistemul functioneaza pe baza unei bucle infinite de citire si reactie, impartita in trei stadii:

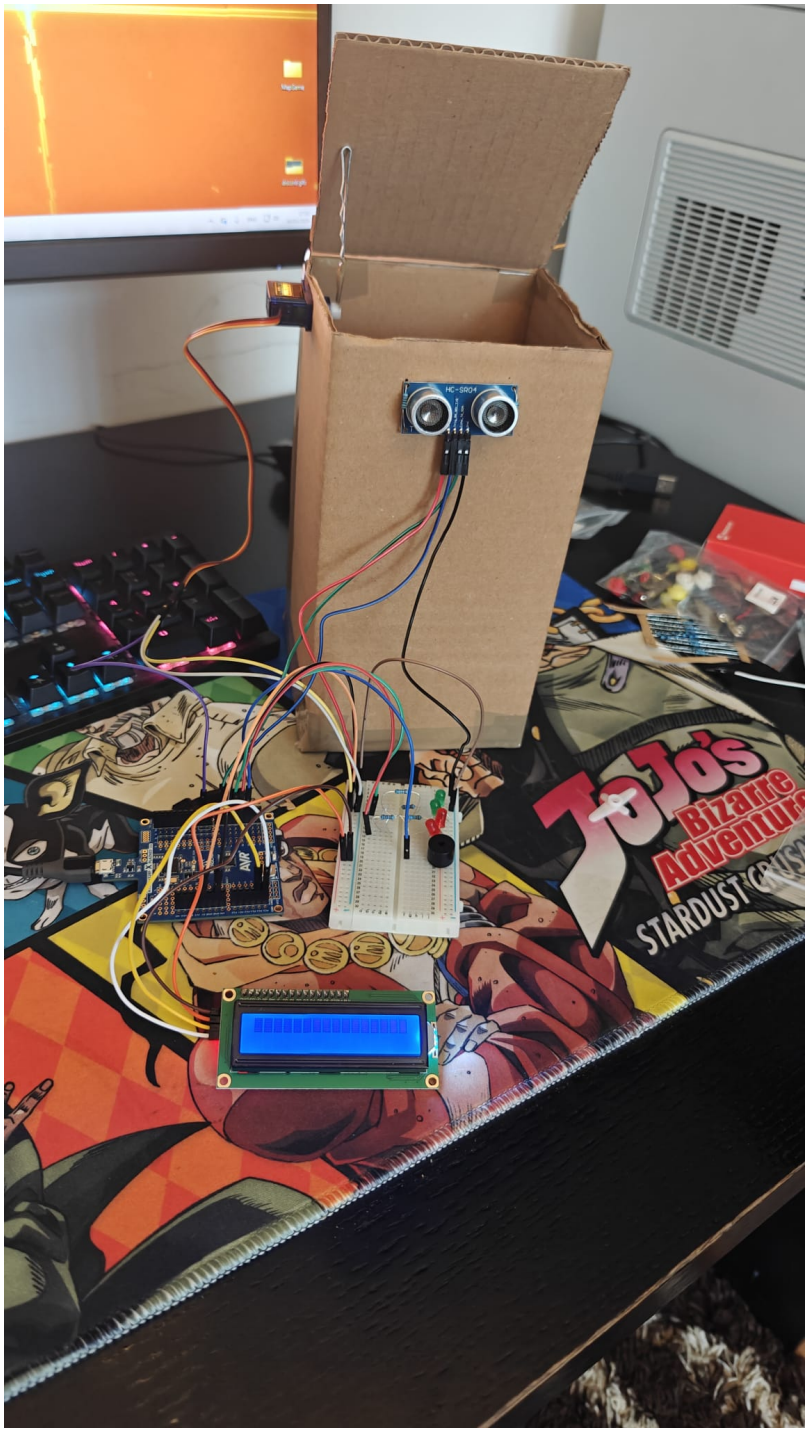
- Starea de Veghe (Repaus): Senzorul ultrasonic emite unde in permanenta. Cat timp distanta masurata este mai mare de 8 cm, capacul ramane inchis, LED-urile verzi sunt aprinse (indicand starea de veghe), iar ecranul LCD afiseaza textul stabil “Apropie mana” alaturi de numarul curent de utilizari.
- Starea Activa (Detectie): Cand o mana se apropie la sub 8 cm, microcontrollerul comuta starea: LED-urile verzi se sting, cele rosii se aprind, iar ecranul isi schimba textul in “Capac Deschis!”. In acelasi timp, contorul de utilizari creste cu +1, iar Timerul 1 modifica factorul de umplere (Duty Cycle) pe pinul PB1, determinand servomotorul sa roteasca axul rapid la 90 de grade, ridicand capacul prin intermediul tijei mecanice.
- Temporizare si Revenire: Sistemul mentine capacul deschis timp de 4 secunde pentru a permite aruncarea deseurilor. Dupa expirarea timpului, motorul revine lin la 0 grade, LED-urile rosii se sting, cele verzi se reaprind, iar ecranul revine la mesajul de veghe, actualizat cu noul numar de utilizari. Buzzer-ul bipaie rapid, semnalizand inchiderea cosului si terminarea acelui ciclu de utilizare.

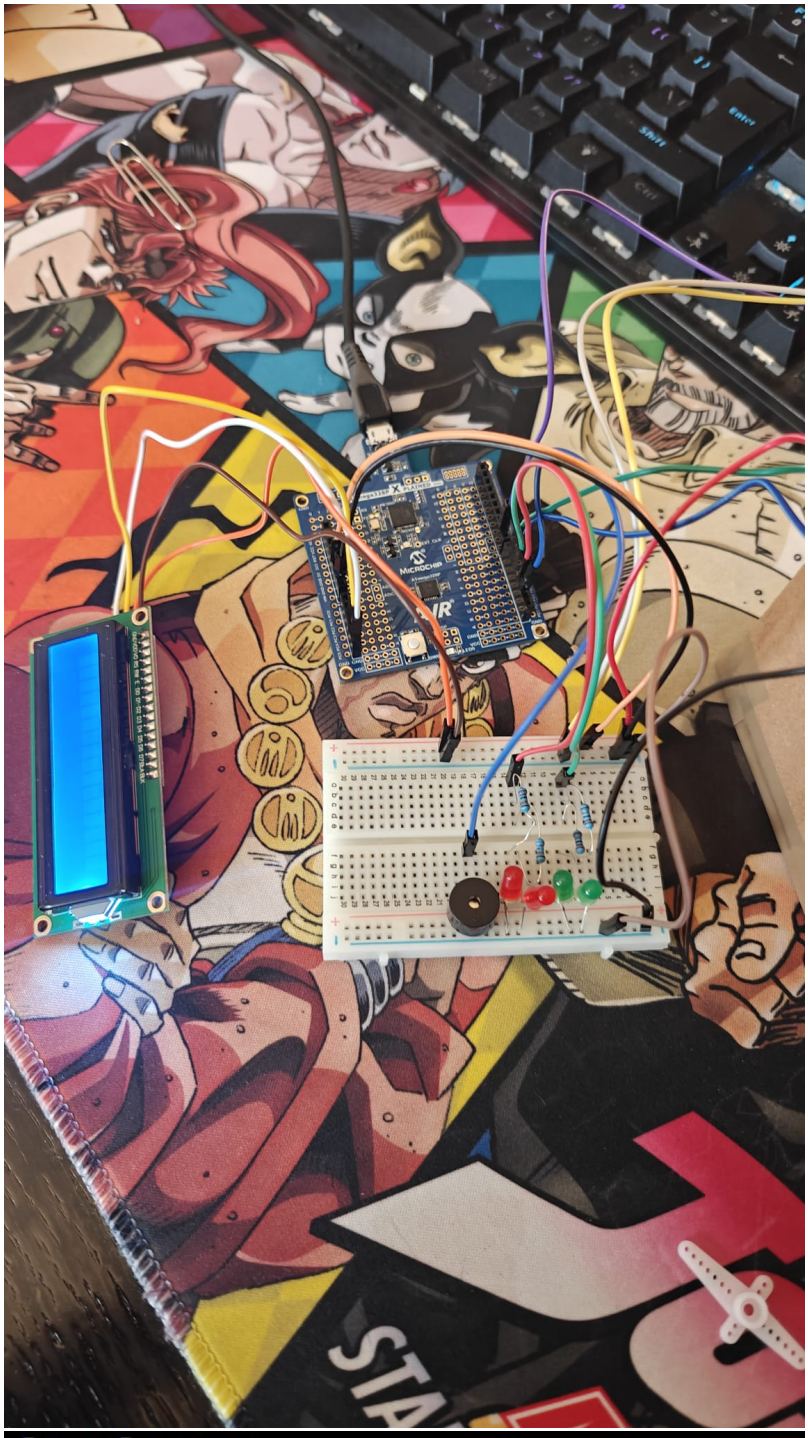
Imagini Hardware




HC-SR04


HC-SR04


HC-SR04
JoJo's
Bizarre
Adventure
STARDUST



```
[DIST] 67 cm
[DIST] 65 cm
[DIST] 66 cm
[DIST] 89 cm
[DIST] 16 cm
[DIST] 9 cm
[DIST] 5 cm
[STATE] Object detected -> opening lid
[STATE] Holding open for 4 s
```

In imaginea de mai sus se poate observa cum senzorul a detectat ca m-am apropiat sub threshold-ul

de 8cm si trimite semnal la SG90 sa deschida cosul, ca sa aruncam gunoiul in el.

Software Design

Stadiul actual al implementarii

Toate functionalitatile planificate sunt implementate si testate fizic pe hardware:

- Detectia proximitatii prin HC-SR04 si declansarea automata a capacului - **functional**
- Control servo SG90 cu deschidere instantanee si inchidere lina - **functional**
- Afisare LCD 1602 cu mesaje de stare si contor de utilizari - **functional**
- Indicatori LED (verde standby / rosu activ) - **functional**
- Semnal sonor buzzer la finalul ciclului de inchidere - **functional**
- Debug serial USART la 9600 baud pe COM3 - **functional**

Mediu de dezvoltare

Codul a fost scris in **C pur** (standard C99), fara nicio librarie Arduino, folosind exclusiv manipulare directa de registre AVR. Toolchain-ul folosit este **AVR-GCC** integrat in **Microchip Studio**, iar programarea placii se face prin interfata **mEDBG** (debugger-ul integrat al placii ATmega328P Xplained Mini), care expune si un **port serial CDC (COM3)** pentru debug la 9600 baud.

Motivatia alegerii bibliotecilor

S-a ales in mod deliberat sa nu se foloseasca librarii de nivel inalt (Arduino, Procyon AVRlib etc.), din urmatoarele motive:

- **<avr/io.h>** - singura modalitate portabila de a accesa registrele ATmega328P (TCCR1A, TWCR, PORTD etc.) fara hardcodarea adreselor de memorie
- **<avr/interrupt.h>** - necesara pentru declararea ISR-urilor cu sintaxa corecta si pentru `sei()` / `cli()`; alternativa manuala ar implica setarea directa a vectorilor de intrerupere, mai error-prone
- **<util/delay.h>** - ofera `_delay_ms` si `_delay_us` calibrate exact la F_CPU; o implementare manuala cu bucle ar fi dependenta de optimizarile compilatorului si impredictibila
- **<stdio.h>** - folosita exclusiv pentru `sprintf` la formatarea liniei “Utilizari: N” pe LCD; toate celelalte afisari seriale sunt implementate manual prin `usart_print_u16` tocmai pentru a evita overhead-ul lui `printf`

Aceasta abordare ofera **control complet asupra timingurilor si registrelor**, esential pentru un sistem real-time cu PWM, I2C si intreruperi simultane.

Elementul de noutate

Fata de un proiect tipic bazat pe Arduino cu librarii gata-facute, acest proiect implementeaza **de la zero, la nivel de registre**:

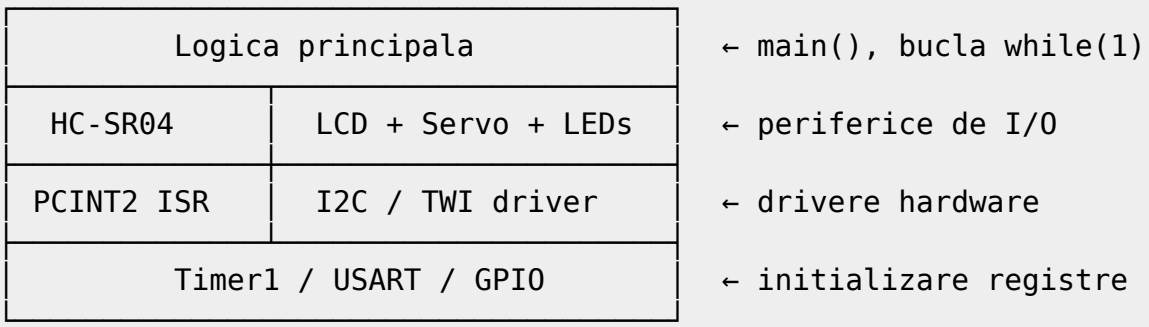
1. Un **driver complet I2C/TWI** cu mecanism de bus recovery activ (9 pulsuri SCL manuale la pornire), care rezolva scenariul real in care un reset in mijlocul unui transfer I2C lasa busul blocat
2. Un **driver LCD HD44780 in mod 4-bit** prin expandor PCF8574, cu respectarea exacta a timingurilor din datasheet (pulse width E, setup time)
3. **Masurarea distantei prin intreruperi** (PCINT2) in loc de polling, cu tratarea corecta a cazului de overflow al Timer1 - tehnica mai robusta decat solutia clasica cu `pulseIn()`
4. **Inchiderea lina a servo-ului** prin incrementare graduala a OCR1A (50 pasi × 40 ticks × 20ms), eliminand socurile mecanice ale unei comutari bruste

Justificarea utilizarii functionalitatilor din laborator

Functionalitate	Laborator	Justificare in proiect
Intreruperi externe (PCINT2)	Lab 2	Masurarea duratei pulsului ECHO de la HC-SR04 necesita capturarea exacta a fronturilor, imposibil de realizat corect prin polling in bucla principala
Timer1 Fast PWM	Lab 3	Servomotorul SG90 necesita un semnal PWM de 50 Hz cu duty cycle precis (0.5–1.5 ms); Timer1 cu TOP=ICR1 ofera rezolutia necesara (0.5 µs/tick)
TWI / I2C	Lab 6	Singurul protocol disponibil pentru comunicatia cu backpack-ul PCF8574 al LCD-ului; implementarea hardware TWI a ATmega elibereaza CPU-ul fata de o implementare bit-bang
USART	Lab 1	Debug in timp real al distantei masurate si al tranzitiilor de stare, esential in faza de calibrare si validare

Scheletul proiectului si interactiunea dintre module

Sistemul are o **arhitectura stratificata**:



Interactiunea dintre module:

- **ISR(PCINT2_vect)** ruleaza asincron si populeaza variabilele `echo_start`, `echo_duration`, `echo_ready` - marcate `volatile` pentru a preveni optimizarile gresite ale compilatorului

- **`measure_distance_cm()`** declanseaza un puls TRIG si asteapta (cu timeout) ca ISR-ul sa seteze `echo_ready`, apoi calculeaza distanta
- **Bucla principala** citeste distanta si comuta starea sistemului, apeland modulele de servo, LCD si GPIO in ordine determinista
- **Timer1** ruleaza independent in background, generand PWM continuu pe PB1 - OCR1A este modificat direct din cod fara a opri timerul

Validarea functionarii:

- Distanta masurata a fost validata prin output-ul serial (`[DIST] X cm`) comparat cu o rigla fizica - threshold-ul de 8 cm a fost ales dupa testare manuala pentru a evita false triggers
- Tranzitiile de stare sunt logate explicit: `[STATE] Object detected`, `[STATE] Holding open`, `[STATE] Closing lid`, `[STATE] Back to idle`
- LCD-ul ofera confirmare vizuala in timp real a fiecarei stari

Demo video

In video se poate observa:

1. Starea de veghe cu LCD afisand “Apropie mana” si LED verde aprins
2. Detectia mainii la sub 8 cm si deschiderea imediata a capacului
3. LED-ul rosu aprins si LCD-ul afisand “Capac Deschis!” cu contorul incrementat
4. Inchiderea lina a capacului dupa 4 secunde
5. Semnalul sonor al buzzerului la finalul inchiderii
6. Revenirea la starea de veghe cu contorul actualizat

Calibrarea senzorilor

HC-SR04 - Senzor ultrasonic

Constanta de conversie **116 ticks/cm** a fost derivata astfel:

- Timer1 tick = 0.5 µs (prescaler 8, 16 MHz)
- Viteza sunetului = 343 m/s = 0.0343 cm/µs
- Dus-intors: distanta = durata_µs × 0.0343 / 2
- Cu ticks: distanta_cm = ticks × 0.5 / (1 / 0.0343 × 2) = ticks / **116.3** ≈ ticks / 116

Threshold-ul de **8 cm** a fost ales experimental: la valori mai mari apareau false triggers cauzate de reflexii ale mediului; la valori mai mici sistemul nu detecta mana la timp.

SG90 - Servo motor

Valorile OCR1A au fost calibrate manual dupa montarea fizica a servo-ului pe cos:

- **`SERVO_CLOSED = 3000`** (1.5 ms) - pozitia in care capacul sta inchis fara sa forteze mecanic
- **`SERVO_OPEN = 1000`** (0.5 ms) - pozitia de deschidere maxima (CCW)

Procedura: dupa `init_pwm()`, servo-ul porneste la OCR1A=3000; cornul servo-ului a fost repoziționat fizic pana cand aceasta valoare corespunde capacului inchis, iar 1000 corespunde deschiderii dorite.

Optimizari realizate

Optimizare	Unde	De ce
`usart_print_u16` custom in loc de `printf`	USART debug	`printf` cu %d trage ~1.5 KB de cod suplimentar din libc; functia custom foloseste ~20 bytes
Calcul UBRR_VAL la compile-time cu `#define`	USART init	Evita o impartire la runtime; valoarea e constanta si cunoscuta la compilare
Variabile ISR marcate `volatile`	PCINT2 / main	Previne ca GCC sa cache-uiasca valorile in registre CPU si sa ignore scrierile din ISR
Timeout pe toate operatiile I2C (10 000 iteratii)	TWI driver	Previne blocarea infinita a sistemului in caz de defect al LCD-ului sau al busului
`i2c_bus_recovery()` inainte de TWEN	TWI init	Elibereaza busul blocat dintr-un reset anterior, elimina necesitatea repornirii manuale
Dead-band 500 ms dupa revenire	Main loop	Previne re-triggering-ul imediat daca mana e inca in raza senzorului
Inchidere lina servo (50 pasi)	`servo_close_smooth`	Reduce stresul mecanic pe capac si elimina sunetul de buzz al servo-ului la comutare brusca

Structura software

Firmware-ul este organizat in **7 module functionale**, fiecare corespunzand unui periferic sau unei functionalitati distincte:

Modul	Functii principale	Rol
USART	`usart_init`, `usart_print`, `usart_println`, `usart_print_u16`	Debug serial la 9600 baud prin COM3
GPIO	`init_gpio`	Configureaza directia si starea initiala a pinilor
Timer1 / PWM	`init_pwm`, `servo_set`, `servo_close_smooth`	Genereaza semnal PWM 50 Hz pentru servomotorul SG90
PCINT2 / ISR	`ISR(PCINT2_vect)`	Masoara durata pulsului ECHO de la HC-SR04
TWI / I2C	`init_i2c`, `i2c_bus_recovery`, `i2c_start`, `i2c_stop`, `i2c_write_byte`, `pcf8574_write`	Comunicatie I2C cu backpack-ul PCF8574 al LCD-ului
LCD HD44780	`lcd_init`, `lcd_command`, `lcd_data_char`, `lcd_set_cursor`, `lcd_print`, `lcd_clear`	Driver complet pentru display in mod 4-bit prin I2C
Logica principala	`main`, `measure_distance_cm`, `display_idle`, `display_active`	Bucla infinita de citire senzor si control actuatori

Descrierea modulelor

USART - Debug serial

USART0 este configurat in modul **TX-only, 8N1 la 9600 baud**. Valoarea registrului UBRR0 este calculata la compile-time:

```
#define BAUD      9600UL
#define UBRR_VAL  ((F_CPU / 16UL / BAUD) - 1UL)   /* = 103 la 16 MHz ->
eroare 0.16% */
```

Functia `usart_print_u16` converteste un `uint16_t` la sir de caractere fara a folosi `printf`, evitand includerea librariei grele <`stdio.h`> pentru uz general (aceasta este inclusa oricum pentru `sprintf` folosit la LCD).

GPIO - init_gpio

Seteaza directia pinilor din DDRD si starea initiala din PORTD la pornire:

- **PD2** (buzzer) - iesire, initial LOW (oprit)
- **PD3** (LED rosu) - iesire, initial LOW (stins)
- **PD4** (LED verde) - iesire, initial HIGH (aprins - stare de veghe)
- **PD5** (ECHO) - intrare, fara pull-up intern
- **PD6** (TRIG) - iesire, initial LOW

Timer1 / PWM - Control servo SG90

Servomotorul SG90 este controlat prin **Timer1 in modul Fast PWM (Mode 14, TOP = ICR1)**, cu prescaler 8, ceea ce da o rezolutie de **0.5 µs/tick** la 16 MHz:

```
ICR1  = 39999;       /* perioda = 20 ms -> 50 Hz                    */
OCR1A = SERVO_CLOSED /* 3000 ticks = 1.5 ms -> pozitie inchis       */
/* SERVO_OPEN = 1000 ticks = 0.5 ms -> pozitie deschis (CCW, maxim) */
```

Inchiderea capacului se face **lin** prin functia `servo_close_smooth`: OCR1A creste de la 1000 la 3000 in **50 de pasi de cate 40 de ticks**, cu `_delay_ms(20)` intre pasi - rezultand o miscare de 1 secunda, pentru a evita socuri mecanice.

ISR(PCINT2_vect) - Masurarea distantei

Intreruperea **Pin Change pe grupul PCINT2 (Port D)** este activata exclusiv pentru pinul PD5 (ECHO). ISR-ul functioneaza astfel:

1. **Front crescator** (`PIND & (1«PD5) = 1`): salveaza valoarea curenta a `TCNT1` in `echo_start`
2. **Front descrescator**: calculeaza durata pulsului, tratand si cazul rar de **overflow al Timer1** (wrap la ICR1=39999):

```
echo_duration = (now >= echo_start)
                ? (now - echo_start)
                : ((uint16_t)(40000u - echo_start) + now);
echo_ready = 1;
```

Distanta in centimetri se obtine impartind durata la **116 ticks/cm** (derivat din: viteza sunet = 343 m/s, dus-intors, cu 0.5 µs/tick).

TWI / I2C - Comunicatie cu LCD

Busul I2C este initializat la **100 kHz** (SCL = 16MHz / (16 + 2×72×1)):

```
TWSR = 0x00;   /* prescaler = 1  */
TWBR = 72;     /* -> SCL = 100 kHz */
```

O functionalitate importanta este **`i2c_bus_recovery`**: inainte de a activa blocul TWI, se genereaza manual **9 pulsuri de clock pe SCL** cu SDA liber, urmata de o conditie STOP manuala. Aceasta elibereaza orice slave (PCF8574) care retinuse SDA jos dintr-o tranzactie intrerupta anterior (de exemplu, la un reset in mijlocul unui transfer), prevenind blocarea busului la pornire.

Toate functiile I2C includ **timeout** (10 000 de iteratii) pentru a evita blocarea infinita in caz de defect hardware.

LCD HD44780 - Driver 4-bit prin PCF8574

LCD-ul 1602 este conectat prin backpack-ul **PCF8574** (adresa I2C 0x27), care mapeaza cei 8 biti ai expander-ului la interfata 4-bit a HD44780:

```
/* Bit mask-uri PCF8574 */
#define LCD_BL  0x08   /* P3 -> backlight        */
#define LCD_EN  0x04   /* P2 -> Enable           */
/* P1 -> RW (tinut la 0) */
#define LCD_RS  0x01   /* P0 -> Register Select  */
/* P7-P4 -> D7-D4 (nibble de date) */
```

Secventa de initializare respecta protocolul HD44780 pentru intrarea in modul 4-bit: **3 pulsuri cu nibble 0x03**, urmat de nibble 0x02, dupa care se trimit comenzile standard (2 linii, font 5×8, display ON, cursor OFF).

Functia de baza `lcd_pulse_nibble` seteaza datele pe bus, genereaza un puls pe linia **E** (Enable), si elibereaza busul:

```
pcf8574_write(base);              /* date valide, E=0 */
_delay_us(1);
pcf8574_write(base | LCD_EN);     /* E=1 -> LCD citeste */
_delay_us(2);
pcf8574_write(base);              /* E=0                */
_delay_us(50);
```

Logica principala - main loop

Dupa initializarea tuturor perifericelor si activarea intreruperilor globale (`sei()`), programul intra intr-o **bucla infinita** cu trei stari:

1. **Starea de veghe (IDLE)**: senzorul masoara distanta la fiecare ~60 ms. LCD afiseaza “Apropie mana” si numarul de utilizari. LED verde aprins, LED rosu stins.
2. **Detectie obiect (dist > 0 && dist ≤ 8 cm)**:
   1. LED verde se stinge, LED rosu se aprinde
   2. Servo sare imediat la SERVO_OPEN (1000)
   3. Contorul de utilizari creste cu 1, LCD afiseaza “Capac Deschis!”
   4. Capacul ramane deschis **4 secunde** (`_delay_ms(4000)`)
3. **Revenire**: servo inchide lin capacul prin `servo_close_smooth`, LED-urile revin la starea de veghe, LCD se actualizeaza, urmat de un **dead-band de 500 ms** pentru a preveni retriggering-ul imediat. Buzzer-ul emite un scurt semnal sonor pentru a semnaliza ca s-a terminat ciclul de deschidere-inchidere.

Diagrama de stare a buclei principale:

```
     ┌──────────────────────────────────────────────────┐
     │                                                  │
     ▼                                                  │
[IDLE / VEGHE]                                          │
LED verde ON                                            │
LCD: "Apropie mana | Utilizari: N"                      │
Masoara distanta la ~60 ms                              │
     │                                                  │
     │  dist ≤ 8 cm                                     │
     ▼                                                  │
[ACTIV / DESCHIS]                                       │
LED rosu ON, verde OFF                                  │
Servo -> OPEN (1000)                                    │
usage_count++                                           │
LCD: "Capac Deschis! | Utilizari: N"                    │
Asteapta 4 secunde                                      │
     │                                                  │
     ▼                                                  │
[REVENIRE]                                              │
servo_close_smooth() - 1 secunda                        │
```

```
   Buzzer ON (semnal sonor de confirmare)                    │
   LED verde ON, rosu OFF                                    │
   LCD: "Apropie mana | Utilizari: N"                        │
   Dead-band 500 ms ─────────────────────────────────────────┘
```

Surse si functii implementate

Functie	Fisier	Descriere
`usart_init()`	main.cpp	Initializeaza USART0 la 9600 8N1 TX-only
`usart_print / println / print_u16`	main.cpp	Trimitere debug serial fara printf
`init_gpio()`	main.cpp	Configureaza DDR si PORT pentru toti pinii folositi
`init_pwm()`	main.cpp	Configureaza Timer1 Fast PWM Mode 14, 50 Hz pe PB1
`init_interrupts()`	main.cpp	Activeaza PCINT2 pe PD5 pentru masurarea ECHO
`ISR(PCINT2_vect)`	main.cpp	Masoara durata pulsului ECHO cu Timer1
`i2c_bus_recovery()`	main.cpp	Elibereaza busul I2C inainte de initializare
`init_i2c()`	main.cpp	Initializeaza TWI la 100 kHz
`pcf8574_write()`	main.cpp	Trimite un byte la expanderul I2C al LCD-ului
`lcd_init()`	main.cpp	Secventa completa de initializare HD44780 4-bit
`lcd_command / lcd_data_char`	main.cpp	Trimitere comanda / caracter la LCD
`lcd_set_cursor / lcd_print / lcd_clear`	main.cpp	Functii de nivel inalt pentru afisare
`servo_set()`	main.cpp	Seteaza pozitia servo prin OCR1A
`servo_close_smooth()`	main.cpp	Inchide lin capacul in 1 secunda (50 pasi)
`hcsr04_trigger()`	main.cpp	Genereaza puls TRIG de 10 µs
`measure_distance_cm()`	main.cpp	Declanseaza masurarea si returneaza distanta in cm
`display_idle / display_active`	main.cpp	Actualizeaza LCD pentru fiecare stare
`main()`	main.cpp	Initializare sistem + bucla infinita de control

Download

https://github.com/RomanianExe/ProiectPM