Dashboard hibrid de monitorizare (virtual & ambient)

Introducere

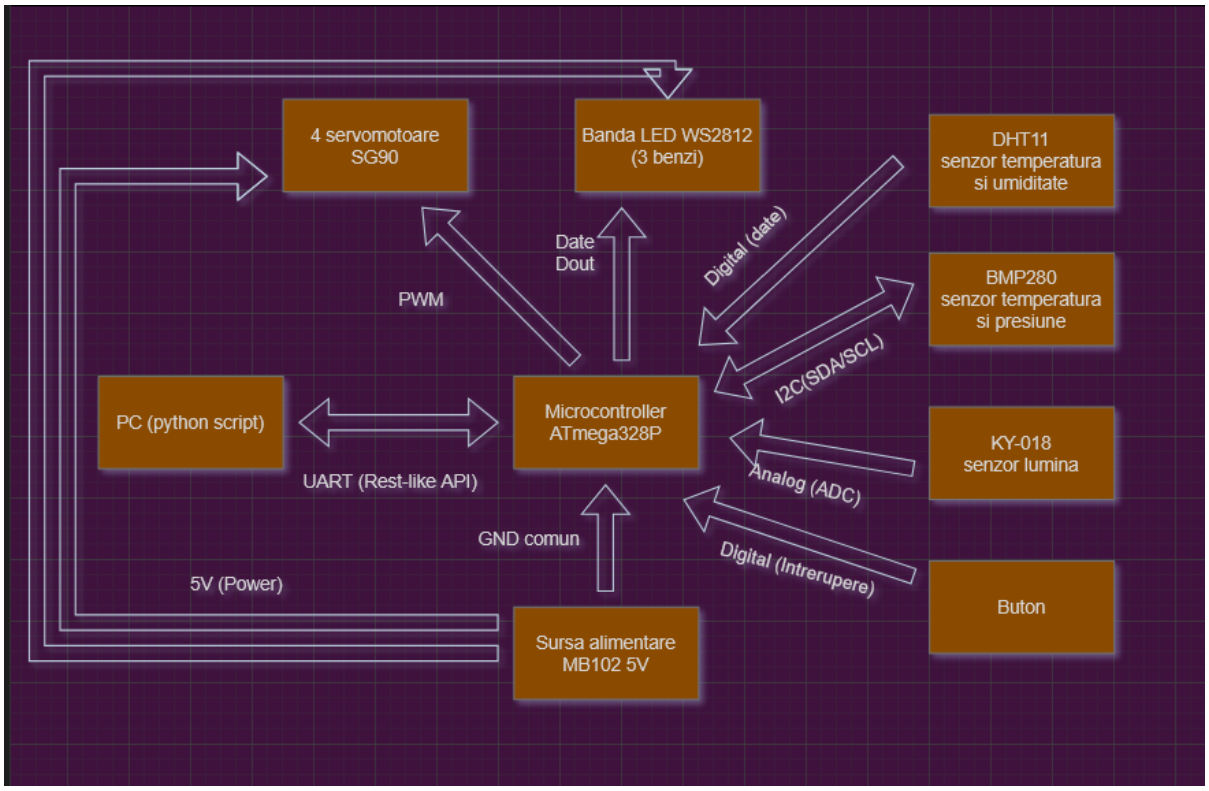
Ce face proiectul: Proiectul reprezinta un sistem hibrid de monitorizare care combina telemetria unui PC (grad de utilizare cpu,gpu,ram) cu achiziția de date din mediul fizic (temperatura, umiditate, luminozitate, presiune atmosferica). Microcontrollerul afiseaza aceste date pe 4 cadrane analogice (folosind servomotoare) si ofera feedback vizual printr-o banda led.

Scopul si ideea de pornire: Ideea a pornit de la nevoia de a avea un dashboard fizic pe birou care sa elimine necesitatea de a tine deschise aplicatii de tip Task Manager pe monitor. Scopul s-a extins apoi catre conceptul de wellness, sistemul monitorizand nu doar starea pc-ului, ci si pe cea a utilizatorului, coreland efortul termic al calculatorului cu temperatura si calitatea aerului din camera.

Utilitate: Proiectul este util intrucat ofera un feedback vizual non-intruziv si un avertisment atunci cand condițiile de mediu sau resursele sistemului ating un prag critic.

Descriere generală

Arhitectura sistemului se bazeaza pe un model **Master-Slave (REST-like API) pe interfata UART**. Microcontrolerul ATmega328P actioneaza ca Master, interogand periodic scriptul Python de pe PC (Slave) pentru date brute. Simultan, microcontrolerul achizitionează date de la 3 senzori locali. Datele sunt unite matematic intr-un **Wellness index**. Iesirile sistemului sunt controlate prin semnale PWM interpolate liniar pentru a oferi miscari naturale acelor indicat



Modulul de achizitie (intrari):

1. Mediul ambiental: Parametrii fizici sunt monitorizati prin senzori dedicati: DHT11 (comunicare digitala 1-Wire pentru temperatura/umiditate), BMP280 (comunicare I2C pe pinii SDA/SCL pentru presiune) si un fotorezistor KY-018 (citire analogica prin ADC pentru luminozitate).
2. Interactiunea utilizatorului: schimbarea modurilor de afisare se face printr-un buton conectat la un pin digital cu suport pentru intreruperi hardware (INT0), evitand astfel blocarea programului prin polling continuu.

Modulul de afisare (iesiri):

1. Afisajul analogic: este realizat cu ajutorul a 4 servomotoare SG90, controlate hardware prin semnale PWM generate de timerele interne ale microcontrolerului. Valorile brute sunt interpolate liniar pentru a asigura o miscare fluidă (0-180 grade).
2. Feedback vizual: O banda LED WS2812B primeste semnale digitale (protocol Dout/Din) pentru a semnaliza starea generala a sistemului prin coduri de culori.

Modulul de alimentare:

1. Pentru a asigura stabilitatea sistemului si a preveni resetarea microcontrolerului cauzata de fluctuatiile de curent, partea mai consumatoare (servomotoarele si banda LED) este izolata si alimentata exclusiv dintr-o sursa externa. Singura legatura electrica dintre cele doua circuite este un GND comun, esential pentru transmiterea corecta a semnalelor de date.

Hardware Design

Stadiul actual al implementarii hardware

In prezent, arhitectura hardware a fost complet proiectata si validata la nivel de conexiuni folosind mediul de simulare tinkercad. Componentele fizice au fost selectate si sunt in curs de comanda. S-a definitivat separarea alimentarii logice de cea de forta, stabilindu-se un ground comun intre sursa externa mb102 si placa de dezvoltare. Asamblarea fizica pe breadboard se va realiza urmand fidel schema de conexiuni deja testata in simulator.

Alocarea pinilor si detaliile tehnice

1. Buton comutare pe pinul d2: suporta intreruperi externe hardware (int0), esential pentru detectarea apasarii fara a bloca executia programului in polling.
2. Servomotoare sg90 pe pinii d3, d5, d6, d9: acesti pini dispun de suport hardware integrat pentru modulatia in latime a impulsurilor (pwm) generata de timerele 0, 1 si 2, controlul unghiului facandu-se in fundal, fara intarzieri software.
3. Banda led ws2812b pe pinul d7: pin digital standard necesar pentru trimiterea bitilor de date prin protocolul 1-wire, manipulat software prin bit-banging.
4. Senzor dht11 pe pinul d8: pin digital standard necesar pentru protocolul de comunicare proprietar lent pe un singur fir.
5. Senzor bmp280 pe pinii a4 (sda) si a5 (scl): pinii hardware dedicati exclusiv comunicarii pe magistrala i2c.
6. Senzor lumina ky-018 pe pinul a0: pin conectat direct la convertorul analog-digital (adc) pentru a transforma nivelul de tensiune in valori digitale de la 0 la 1023.

Schema electrica si explicatii

Schema electrica ilustreaza izolarea alimentarii actuatorilor de logica de control. Sursa externa mb102 livreaza 5v direct pe liniile vdd ale benzilor led montate in paralel si spre servomotoare. Senzorii primesc alimentare curata de la portul de 5v al microcontrollerului. O legatura critica pe schema este firul de masa comuna (gnd) care uneste cele doua circuite, oferind o referinta de zero volti unificata pentru semnalele de date. Rezistenta de protectie de 330 ohmi este prezenta pe linia de date a benzii led (din).

Diagrame de semnal

Sistemul utilizeaza preponderent semnale pwm pentru actionarea celor 4 servomotoare. Microcontrolerul genereaza un semnal dreptunghiular cu o perioada fixa de 20 milisecunde. Variatia factorului de umplere, prin emiterea unui puls pozitiv cu latimea cuprinsa intre 1 milisecunda (reprezentand unghiul de 0 grade) si 2 milisecunde (reprezentand unghiul de 180 de grade), dicteaza direct pozitia acului indicator.

Rezultatele simularii si dovada de functionare

Nota: schema este orientativa si ilustreaza conexiunile logice pentru montajul fizic. Spre exemplu, deoarece simulatorul tinkercad nu ofera un senzor bmp280 cu 4 pini, am folosit vizual un senzor ultrasonic pentru a putea trasa corect firele magistralei i2c. De asemenea, sursa de tensiune din simulare este una teoretica, urmand sa fie inlocuita in realitate de modulul mb102, care va fi alimentat la 9v pentru a furniza in siguranta cei 5v necesari actuatorilor.

Avand in vedere ca piesele hardware sunt in stadiul de comanda, validarea circuitului si dovada functionarii s-au realizat in mediul de simulare tinkercad. Am creat un cod minimal care activeaza pinul a0 si pinul pwm d3. Simularea demonstreaza ca divizorul de tensiune converteste corect

intensitatea luminii in valori scalate vizibile pe interfata seriala, iar semnalul modulat translateaza pozitia servomotorului la unghiul comandat.

Lista de componente si rolul lor

Componenta	Utilitate
ATmega328P (Xplained Mini)	actioneaza ca unitate centrala (master), proceseaza datele de la senzori si de la pc, generand semnalele de control pentru actuatori
Senzor DHT11	realizeaza achizitia temperaturii si umiditatii din mediul fizic
Senzor BMP280	realizeaza achizitia presiunii atmosferice (in simulator a fost inlocuit vizual cu un modul generic cu 4 pini pentru a respecta magistrala i2c)
Senzor lumina fotosensibil KY-018	format dintr-un fotorezistor inseriat cu o rezistenta de 10k pentru a crea un divizor de tensiune, monitorizeaza luminozitatea din camera
4x Servomotoare SG90	afiseaza in mod analogic, sub forma unor cadrane fizice, parametrii sistemului (utilizare cpu, memorie ram, temperatura ambientala, umiditate)
Banda LED WS2812B	ofera feedback vizual despre starea sistemului
Buton	permite utilizatorului comutarea manuala intre modurile de afisare (telemetrie pc sau parametri ambientali)
Modul sursă MB102 (9V)	asigura puterea necesara pentru actuatori, protejand stabilizatorul intern al placii de dezvoltare impotriva supraconsumului de curent.

Aici puneţi tot ce ţine de hardware design:

- listă de piese
- scheme electrice (se pot lua şi de pe Internet şi din datasheet-uri, e.g. http://www.captain.at/electronic-atmega16-mmc-schematic.png)
- diagrame de semnal
- rezultatele simulării

Software Design

Mediu de dezvoltare

Firmware-ul este scris în C++ și dezvoltat în **Arduino IDE 2.x**, ales pentru compatibilitatea directă cu ecosistemul de biblioteci AVR și pentru integrarea nativă a compilatorului avr-gcc. Scriptul Python rulează în **Python 3** și este independent de platformă (Windows/Linux/macOS).

Stadiul actual al implementării

Firmware-ul pentru ATmega328P este complet funcțional și implementat integral în fișierul `dashboard.ino`. Codul acoperă toate funcționalitățile descrise în arhitectură: comunicarea UART master-slave cu scriptul Python, achiziția de date de la cei trei senzori locali, controlul celor patru servomotoare cu interpolare liniară, gestionarea benzii LED WS2812B cu tranziții de culoare și luminozitate adaptivă, mașina de stări cu trei moduri și detectarea hardware a apăsării butonului prin întrerupere externă INT1.

Pe latura PC-ului, scriptul `pc_listener.py` este de asemenea finalizat. El rulează în Python 3 și folosește bibliotecile `pyserial`, `psutil` și `GPUtil` pentru a citi temperatura CPU, gradul de utilizare CPU și GPU și nivelul bateriei, transmițând valorile brute la cererea plăcuței.

Biblioteci și surse third-party

Bibliotecă	Rol	Motivație
Servo.h (Arduino built-in)	Controlul celor 4 servomotoare SG90	Utilizează Timer1 intern și generează semnale PWM de 20ms cu rezoluție în microsecunde pe orice pin digital, fără a ocupa un pin hardware PWM specific. Permite `detach()` pentru a opri semnalul și a elimina curentul de ținere — esențial pentru protejarea modulului MB102.
Wire.h (Arduino built-in)	Comunicare I2C cu BMP280	Abstractizează hardware-ul TWI al ATmega328P (registrele TWCR, TWDR, TWAR). Întrucât BMP280 este singurul dispozitiv I2C din sistem, nu există conflicte de magistrală.
Adafruit_BMP280	Driver senzor presiune/temperatură	Oferă acces direct la registrele de configurare ale BMP280 (oversampling, filtru IIR, mod standby), permițând configurarea precisă a calității și vitezei de achiziție, lucru imposibil printr-un driver generic.
DHT.h	Driver senzor DHT11	Implementează protocolul proprietar single-bus cu timing precis (semnale de 18ms / 40µs), greu de reprodus manual fără a bloca CPU-ul. Tratează și erorile de citire returnând `NaN`.
Adafruit_NeoPixel	Control bandă WS2812B	Implementează protocolul de comunicare 800kHz al WS2812B prin bit-banging optimizat în assembly inline, singura metodă practică pentru acest protocol pe AVR. Oferă control per-pixel al culorii RGB și al luminozității globale.
psutil (Python)	Date sistem PC	Bibliotecă cross-platform pentru temperatura CPU, utilizarea procesoarelor și starea bateriei, fără a necesita drepturi de administrator.
GPUtil (Python)	Date GPU	Interogare simplă a driver-ului NVIDIA pentru gradul de utilizare GPU.
pyserial (Python)	Comunicare UART cu plăcuța	Standard de facto pentru comunicarea serială în Python, cu suport pentru timeout și buffer de recepție.

Periferialele **UART, ADC și întreruperile externe** au fost inițializate direct prin **registre AVR** (`UCSR0B`, `ADMUX`, `ADCSRA`, `EICRA`, `EIMSK`) fără biblioteci intermediare, conform cerințelor din laboratoarele 1, 2 și 4.

Elementul de noutate

Noutatea principală constă în **fuziunea a două domenii de monitorizare distincte într-un singur instrument fizic cu feedback analogic**. Spre deosebire de aplicațiile software de tip Task Manager, acest sistem oferă informație periferică, non-intruzivă — datele sunt vizibile din colțul câmpului vizual, fără a întrerupe fluxul de lucru.

Al doilea element de noutate este **Wellness Index-ul calculat local pe microcontroler**, care corelează matematic confortul termic, umiditatea și presiunea atmosferică într-un scor unificat de 0–100, afișat în timp real pe un cadran fizic și reflectat cromatic pe banda LED. Indicele este calculat integral pe placă printr-un algoritm ponderat (temperatură 40%, umiditate 40%, presiune 20%), cu parametrii de confort optim conform standardelor de ergonomie termică (22°C, 50% RH, 1013.25 hPa):

```
float calcWellness(float t, float h, float p) {
    float ts = 100.0f - constrain(fabsf(t - 22.0f)   *  8.0f, 0.0f, 100.0f);
    float hs = 100.0f - constrain(fabsf(h - 50.0f)   *  2.0f, 0.0f, 100.0f);
    float ps = 100.0f - constrain(fabsf(p - 1013.25f)*  0.4f, 0.0f, 100.0f);
    return ts * 0.40f + hs * 0.40f + ps * 0.20f;
}
```

Al treilea element notabil este **arhitectura de protecție a consumului de curent**: servomotoarele se mișcă strict secvențial, semnalul PWM este oprit după fiecare mișcare prin `detach()`, banda LED rulează la 21% din luminozitatea maximă și se reduce automat la întuneric — toate prin mecanisme software, fără nicio componentă hardware suplimentară.

Justificarea utilizării funcționalităților din laborator

Laboratorul 0 — GPIO

Pinii PD2 (DHT11) și PD3 (buton) sunt configurați ca intrări digitale. Pull-up-ul intern al butonului este activat prin `pinMode(PIN_BTN, INPUT_PULLUP)`, evitând rezistența externă. Pinul PD6 (WS2812B) este ieșire digitală controlată de biblioteca NeoPixel. GPIO reprezintă baza pe care se construiesc toate celelalte funcționalități ale sistemului.

Laboratorul 1 — UART

Comunicarea cu PC-ul este implementată prin registrele hardware USART0: `UBRR0H/L` pentru baud rate (103 la 9600 baud, eroare <0.2%), `UCSR0B` pentru activarea TX, RX și a întreruperii de recepție, `UCSR0C` pentru formatul cadrului (8N1). Recepția este complet asincronă prin `ISR(USART_RX_vect)`, care acumulează caracterele într-un buffer până la delimitatorul `\n`. UART este interfața prin care plăcuța devine master într-un sistem distribuit cu PC-ul ca slave.

Laboratorul 2 — Întreruperi

Butonul de schimbare a modului folosește **INT1 pe PD3** (ales față de INT0/PD2 care era ocupat de DHT11). Întreruperea este configurată pe front descendent (`ISC11=1, ISC10=0` în `EICRA`) și activată prin `EIMSK |= _BV(INT1)`. `ISR(INT1_vect)` setează un flag `volatile bool btnEv` cu

debouncing software de 200ms. Flagul este procesat în `loop()`, nu în ISR, păstrând handler-ul scurt. Recepția UART folosește de asemenea o ISR dedicată (`USART_RX_vect`). Fără întreruperi, detectarea butonului ar bloca bucla principală prin polling.

Laboratorul 3 — Timere / PWM
Biblioteca `Servo.h` utilizează Timer1 al ATmega328P pentru semnale PWM cu perioadă de 20ms și lățimea impulsului între 544µs (0°) și 2400µs (180°). Algoritmul de **interpolare liniară** avansează unghiul curent cu 1° la fiecare 18ms, rezultând ~55°/secundă — mișcare naturală de indicator analogic. `servoTick()` procesează strict un singur servo per iterație, garantând că niciodată doi servo nu sunt activi simultan.

Laboratorul 4 — ADC
Senzorul KY-018 este citit pe canalul ADC0 (PC0) prin registrele `ADMUX` și `ADCSRA` direct. Referința este AVCC (5V), prescalerul /128 → Fadc = 125kHz (intervalul optim 10-bit conform datasheet ATmega328P). Valoarea brută 0–1023 scalează dinamic luminozitatea benzii LED între `LED_MIN_BR` (8) și `LED_MAX_BR` (55).

Laboratorul 6 — I2C / TWI
Senzorul BMP280 comunică pe magistrala I2C hardware (TWI), pe pinii PC4 (SDA) și PC5 (SCL). Adresa I2C este auto-detectată la startup (0x76 sau 0x77). BMP280 este configurat cu oversampling x16 pe presiune, filtru IIR de coeficient 16 și mod NORMAL cu standby de 500ms pentru stabilitate maximă a citirilor.

Scheletul proiectului și interacțiunea dintre funcționalități

Firmware-ul este organizat ca un **cooperative multitasking loop** fără RTOS, bazat pe timestamp-uri și mașini de stare. `loop()` nu conține niciun `delay()` — fiecare funcție verifică dacă intervalul său a expirat și returnează imediat dacă nu.

```
setup()
  ├── uartInit()      → USART0: 9600 baud, 8N1, RXCIE enabled (registre
directe)
  ├── adcInit()       → ADC0: AVCC ref, prescaler /128 (registre directe)
  ├── btnInit()       → INT1 pe PD3, front descendent (EICRA/EIMSK directe)
  ├── sei()           → activare întreruperi globale
  ├── Wire + BMP280   → I2C init, oversampling x16, filtru IIR x16
  ├── DHT.begin()     → GPIO PD2, delay 1200ms pentru stabilizare DHT11
  ├── servoInit()     → Timer1, toate acele la SRV_MIN=12°, detach
  ├── NeoPixel init   → PD6, animație startup albastru
  ├── readAmbient()   → prima citire senzori înainte de loop
  └── uartStr("READY\n") → semnal de pornire către Python

loop() [non-blocking, ~1-2ms per iterație]
  ├── [1] UART rx     → rxReady? → copie atomică → parsePcResp() → servoSet()
  ├── [2] Buton       → btnEv?   → schimbare sysMode → applyMode()
  ├── [3] Timeout PC  → >8s fără răspuns → MODE_STANDBY
  ├── [4] Scheduler   → round-robin CPU_TEMP/CPU_USAGE/GPU_USAGE + baterie
60s
  ├── [5] Ambient     → la 5s → readAmbient() → calcWellness() → servoSet()
```

```
  ├── [6] servoTick() → 1 pas/iterație, 1 servo activ, 18ms/pas, detach la
final
  └── [7] ledTick()   → la 80ms → lerp culoare, dimming ADC, leds.show()
```

Mașina de stări are trei moduri cu tranziții clare:

Tranziție	Condiție
MODE_PC → MODE_WELLNESS	apăsare buton
MODE_WELLNESS → MODE_PC	apăsare buton
oricare → MODE_STANDBY	timeout >8s fără răspuns PC
MODE_STANDBY → MODE_PC	apăsare buton (resetează timestamp conexiune)

Maparea cadranelor per mod:

Cadran	Mod PC	Mod Wellness
Stânga Sus (D7)	Temperatură CPU (30–100°C)	Temperatură ambientală (15–40°C)
Dreapta Sus (D8)	Utilizare CPU (0–100%)	Umiditate (20–90%)
Stânga Jos (D9)	Utilizare GPU (0–100%)	Presiune atmosferică (980–1040 hPa)
Dreapta Jos (D10)	Nivel baterie (0–100%)	Wellness Index (0–100)

Fluxul de date cu filtrare prin prag la fiecare etapă:

```
[Python psutil] ──UART──> ISR(USART_RX_vect) → buffer
                                ↓ rxReady=true
                           parsePcResp() → |Δval| ≥ THR?
                                ↓ DA
                           PCData struct → sysMode==PC?
                                ↓ DA
                           servoSet() → |Δangle| ≥ 2°?
                                ↓ DA
                           sv[i].tgt = angle
                                ↓
                           servoTick() [1°/18ms, secvențial]
                                ↓
                           SG90 mișcare fizică

[DHT11 + BMP280] ──5s──> readAmbient() → |Δval| ≥ THR?
                                ↓ DA
                           AmbData → calcWellness()
                                ↓
                           servoSet() + ledTick() [culoare wellness]
```

Calibrarea elementelor de senzoristica

KY-018 (senzor lumină — ADC)

Senzorul este folosit relativ (dimming LED), nu ca măsurătoare fizică absolută. Valorile brute ADC

(0–1023) sunt normalizate la [0.0, 1.0] prin formula `1.0 - ADC/1023.0` (inversie: rezistența fotorezistorului scade la lumină → tensiunea ADC crește → valoarea normalizată scade). Verificat acoperind senzorul (ADC ≈ 1020, `lightLevel()` ≈ 0.0) și expunând la lumina directă (ADC ≈ 60, `lightLevel()` ≈ 0.94).

DHT11 (temperatură și umiditate)
DHT11 are precizia specificată de ±2°C și ±5% RH. Pragurile de mișcare ale servo-ului (`THR_AMB_TEMP = 0.5°C`, `THR_HUMIDITY = 1.5%`) sunt sub precizia senzorului — compromis acceptat pentru acest tip de senzor. `dht.begin()` este urmat de `delay(1200)` obligatoriu (stabilizare internă DHT11). Citirile `NaN` sunt ignorate explicit prin `if (isnan(nt) || isnan(nh)) return`.

BMP280 (presiune atmosferică)
Configurat cu filtru IIR coeficient 16 și oversampling x16 → zgomot redus la ±0.12 hPa conform datasheet Bosch. Referința wellness index (1013.25 hPa) este presiunea standard la nivelul mării. Pragul `THR_PRESSURE = 0.8 hPa` este ales ușor deasupra zgomotului senzorului, astfel variațiile reale (fronturi atmosferice) sunt detectate dar zgomotul filtrat.

Servomotoare SG90
`SRV_MIN = 12°` și `SRV_MAX = 168°` sunt reduse față de limitele fizice (0°–180°) pentru a evita forțarea mecanică la capete. Valorile au fost determinate empiric prin `servo.write()` manual și observarea vibrației la capete. Funcția `angleFrom()` realizează interpolare liniară între domeniul parametrului și domeniul angular [12°, 168°].

Optimizări

1. Mișcare secvențială a servo-urilor
Problemă: Patru SG90 simultane consumă până la 4×250mA = 1A în stall, depășind MB102 și provocând reset-uri ale microcontrolerului.
Soluție: `servoTick()` procesează strict un singur servo per apel — primul cu `moving=true`. Instrucțiunea `return` la finalul blocului activ previne activarea celorlalți.

2. Detach după mișcare
Problemă: Un SG90 consumă 6–10mA stând nemișcat cu semnal PWM activ. Patru servo-uri = 24–40mA constant inutil.
Soluție: `sv[i].obj.detach()` imediat ce `sv[i].cur == sv[i].tgt`. La noul target, `attach()` este apelat automat la începutul mișcării.

3. Filtrare prin prag
Problemă: CPU usage fluctuează cu 1–3% între citiri, generând mișcare continuă inutilă a servo-ului.
Soluție: Servo actualizat doar dacă `|valoare_nouă - valoare_curentă| ≥ THR`. Praguri individuale per senzor: `THR_CPU_USAGE=2%`, `THR_CPU_TEMP=1.5°C`, `THR_AMB_TEMP=0.5°C`, `THR_PRESSURE=0.8hPa`.

4. Luminozitate adaptivă LED
Problemă: Banda LED la 100% brightness = 8×60mA = 480mA, depășind masiv MB102.
Soluție: `LED_MAX_BR=55` (21%) și `LED_MIN_BR=8` (~3%) noaptea. Luminozitatea efectivă: `br = LED_MIN_BR + lightLevel() * (LED_MAX_BR - LED_MIN_BR)`, calculat la fiecare refresh de 80ms din ADC-ul KY-018.

5. Tranziții graduale LED
Problemă: Salt brusc de culoare la schimbarea wellness index-ului este vizual neplăcut.
Soluție: Culoarea curentă `ledR, ledG, ledB` este interpolată liniar spre țintă cu pas 4 unități per refresh (80ms). Tranziție completă verde→roșu ≈ 4.4 secunde. Implementată prin funcția `lerp8()`.

6. Frecvență diferențiată UART
Problemă: Bateria se modifică rar; interogarea la 700ms generează trafic inutil.
Soluție: `pcScheduler()` implementează două niveluri: CPU Temp/Usage/GPU round-robin la 700ms, bateria o dată la 60s injectată cu prioritate în scheduler.

7. Copie atomică buffer UART
Problemă: `rxBuf[]` este scris din ISR și citit din `loop()` — risc de corupere date.
Soluție: Dezactivare temporară `RXCIE0` în timpul copierii buffer-ului în `tmp[]`, urmată de reactivare imediată. Durata dezactivării <1µs, neglijabilă față de perioada unui byte UART (~1ms la 9600 baud).

Demo video

(de completat după asamblarea fizică finală)

Demonstrația video va ilustra:

1. Pornire sistem — animație startup albastru, ace la zero
2. Mod PC activ — cadranele afișează CPU Temp, CPU Usage, GPU Usage, Baterie; benchmark CPU pentru mișcare reactivă
3. Comutare mod Wellness — apăsare buton, reconfigurare cadrane pe factori ambientali
4. Simulare condiții nefavorabile — aer cald pe DHT11, tranziție LED verde→roșu, rotire ac Wellness Index
5. Mod Standby — oprire script Python, ace la zero, LED pulsează mov după 8s

Rezultate Obţinute

Care au fost rezultatele obţinute în urma realizării proiectului vostru.

Concluzii

Download

Codul sursa complet este disponibil pe GitHub: https://github.com/stefannn24/Dashboard-monitorizare

Jurnal

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

09/05/2026 - Initializarea paginii si completarea sectiunilor de Introducere si Descriere generala.

15/05/2026 - efectuare comanda piese.

16/05/2026 - realizarea schemei electrice si a schemei de conectare. Intrucat piesele nu au ajuns inca, am simulat cu o bucata scurta de cod functionalitatea servomotoarelor si a benzii led.

18/05/2026 - Finalizarea layout-ului pinilor dupa analiza conflictelor hardware:butonul mutat de pe PD2 (ocupat de DHT11) pe PD3, servomotoarele realocate pe PD7,PB0,PB1,PB2 banda LED mutata pe PD6.

19/05/2026 - Scrierea firmware-ului si a scriptului Python (pc_listener.py). Implementare: protocol UART master-slave, interpolare liniara servo, wellness index, tranzitii LED, masina de stari cu 3 moduri, optimizari consum curent MB102.

21/05/2026 - Sosirea componentelor fizice. Asamblare pe breadboard conform schemei electrice.

22/05/2026 - Testarea individuala a componentelor: validare DHT11, BMP280 pe I2C, senzor lumina KY-018 pe ADC, servomotoare cu interpolare, banda LED cu tranzitii de culoare. Validare comunicare UART cu scriptul Python.

24/05/2026 - Incident hardware: corupere firmware mEDBG (ATmega32U4) cauzata de deconectare brusca USB.

Bibliografie/Resurse

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

Export to PDF