Sistem Distribuit de Monitorizare și Control al Temperaturii Ambientale utilizând Arhitectura Dual-Core ESP32 și FreeRTOS

Alexandru Gherghe
Programul de Master: Sisteme Electrice și de Comunicații Integrate - Rețele de comunicații digitale
Universitatea Transilvania din Brașov
Brașov, România
alexandru.gherghe@student.unitbv.ro

***Abstract*—În contextul actual al Internet of Things (IoT), cerințele pentru sistemele de monitorizare ambientală au evoluat rapid de la simpla achiziție de date la procesare distribuită, eficiență energetică și conectivitate robustă. Această lucrare prezintă etapele de proiectare și implementare ale unui sistem embedded bazat pe microcontrollerul ESP32, care utilizează sistemul de operare în timp real FreeRTOS pentru a gestiona concurent achiziția datelor de temperatură și comunicația Bluetooth Low Energy (BLE). Soluția propusă exploatează arhitectura asimetrică Dual-Core a procesorului Xtensa LX6 pentru a decupla logica de control critică (Hard Real-Time) de stiva de comunicație (Soft Real-Time), eliminând blocajele specifice arhitecturilor secvențiale de tip "super-loop". Lucrarea detaliază mecanismele de sincronizare inter-procesuală prin cozi de mesaje (Message Queues) și strategia de alocare a resurselor hardware pentru a garanta determinismul temporal.**

***Index Terms*—IoT, ESP32, FreeRTOS, Bluetooth Low Energy, Dual-Core, Control Distribuit, Real-Time Systems, 1-Wire Protocol.**

I. Introducere

Monitorizarea parametrilor ambientali, și în special a temperaturii, reprezintă o componentă critică în diverse sectoare industriale și rezidențiale, incluzând centre de date, sere agricole automatizate și sisteme HVAC (Heating, Ventilation, and Air Conditioning). Odată cu miniaturizarea senzorilor și creșterea puterii de calcul a microcontrollerelor moderne, arhitecturile centralizate au fost înlocuite treptat de rețele de senzori distribuiți (Wireless Sensor Networks - WSN).

Provocarea majoră în proiectarea acestor noduri de senzori o constituie gestionarea resurselor limitate ale microcontrollerului în condițiile în care acesta trebuie să îndeplinească simultan două funcții adesea contradictorii:

1) *Determinism și Control:* Citirea senzorilor și acționarea elementelor de execuție trebuie să se facă la intervale precise de timp, fără abateri semnificative (jitter).
2) *Conectivitate și Telemetrie:* Menținerea unei conexiuni radio (WiFi, BLE) necesită procesare asincronă, gestionarea pachetelor și retransmisii care pot introduce întârzieri impredictibile.

În abordările clasice de tip "bare-metal" (fără sistem de operare), o operațiune lungă, precum scrierea în memoria flash sau negocierea cheilor de criptare Bluetooth, poate bloca bucla principală (‘void loop()‘) pentru zeci sau sute de milisecunde. Aceasta poate duce la ratarea unor evenimente critice, la erori de citire pe magistrale sensibile la timp (precum 1-Wire) sau la instabilitatea buclei de control.

Obiectivul acestei lucrări este de a demonstra superioritatea unei arhitecturi bazate pe un Sistem de Operare în Timp Real (RTOS) implementat pe un procesor Dual-Core. Studiul de caz se concentrează pe un sistem de control termic autonom care permite intervenția umană prin intermediul unei interfețe mobile BLE.

II. Stadiul Actual al Cunoașterii

A. Sisteme de Operare în Timp Real (RTOS)

Spre deosebire de sistemele de operare generaliste (GPOS) precum Linux sau Windows, un RTOS nu este optimizat pentru debitul maxim de date (throughput), ci pentru latență minimă și predictibilitate a execuției.

FreeRTOS, standardul de facto în industrie pentru microcontrollere, oferă un planificator (scheduler) preemptiv. Acesta permite întreruperea unui task cu prioritate mică (ex: actualizarea unui LED de stare sau logging-ul pe serială) pentru a executa imediat un task cu prioritate mare (ex: citirea senzorului sau controlul PWM), garantând timpi de răspuns de ordinul microsecundelor. Mai mult, implementarea FreeRTOS pe ESP32 (ESP-IDF) suportă Multiprocessing Simetric (SMP), permițând distribuirea manuală sau automată a task-urilor pe cele două nuclee fizice.

B. Comparație Protocoale IoT pentru Telemetrie

Alegerea protocolului de comunicație este esențială pentru autonomia dispozitivului și complexitatea implementării. Tabelul I prezintă o comparație între cele mai populare tehnologii IoT utilizate în sistemele de monitorizare locală.

Pentru acest proiect, s-a ales **Bluetooth Low Energy (BLE)** deoarece permite interacțiunea directă cu un smartphone fără a necesita infrastructură suplimentară (routere, gateway-uri), oferind un consum energetic optimizat pentru transmisii sporadice de date mici (Small Data Packets).

TABLE I
COMPARAȚIE TEHNOLOGII COMUNICAȚIE IOT

Parametru	WiFi	ZigBee	BLE (Ales)
Viteză	$>$ 54 Mbps	250 kbps	1 Mbps
Consum	Foarte Mare	Foarte Mic	Foarte Mic
Topologie	Star	Mesh	Star / Mesh
Latență	Ridicată	Medie	Scăzută
Conectare	Router	Gateway	Directă

III. ANALIZA TEORETICĂ

A. Analiza Constrângerilor de Timp (Protocolul 1-Wire)

Un aspect critic al proiectului este integrarea senzorului DS18B20. Acesta utilizează protocolul 1-Wire, care impune restricții severe de timing. Protocolul 1-Wire este un sistem de comunicație half-duplex care utilizează un singur conductor pentru date și, opțional, pentru alimentare (Parasitic Power). Pentru a citi un bit "0", master-ul (ESP32) trebuie să țină linia în LOW pentru exact 60μs. Pentru un bit "1", linia este ținută în LOW doar 6μs, apoi eliberată. Orice întrerupere a procesorului (ex: generată de stiva WiFi/BLE) mai lungă de câteva microsecunde în timpul acestui "bit-banging" va corupe pachetul de date (CRC Error).

B. Modelul Matematic al Controlului Termic

Deși în faza inițială s-a implementat un control ON-OFF (Histerezis), arhitectura permite implementarea unui algoritm PID (Proporțional-Integrator-Derivativ) pentru controlul precis al turației ventilatorului. Eroarea de control la momentul t este definită ca:

$$e(t) = T_{setat} - T_{masurat}(t) \tag{1}$$

Semnalul de comandă $u(t)$, care va determina factorul de umplere PWM, este calculat conform ecuației:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2}$$

Unde K_p, K_i și K_d sunt coeficienții de acordare. Implementarea discretă pe microcontroller va utiliza ecuația recurentă:

$$u[k] = u[k-1] + K_p(e[k] - e[k-1]) + K_i e[k]T_s \tag{3}$$

Această abordare matematică justifică necesitatea unui RTOS, deoarece T_s (timpul de eșantionare) trebuie să fie strict constant pentru ca termenul integral să nu acumuleze erori.

IV. ARHITECTURA SISTEMULUI

A. Design Hardware

Unitatea centrală de procesare este modulul **ESP32-WROOM-32**. Acesta integrează un procesor Xtensa® Dual-Core 32-bit LX6 care poate rula la frecvențe de până la 240 MHz, având 520 KB SRAM intern.

1) *Subsistemul de Senzori:* S-a utilizat senzorul digital **DS18B20**. Pentru stabilitatea liniei de date pe distanțe mai mari de 20cm, este necesar un rezistor de pull-up (R_{PU}). Valoarea acestuia este critică și se calculează pentru a asigura curentul necesar (I_{pullup}) încărcării capacităților parazite ale cablului:

$$R_{PU} = \frac{V_{CC} - V_{IL}}{I_{sink}} \approx 4.7k\Omega \tag{4}$$

2) *Subsistemul de Acționare:* Deoarece curentul maxim suportat de un pin GPIO al ESP32 este limitat la 12mA, comanda ventilatorului (consum tipic 200-300mA) se realizează printr-un etaj de amplificare. S-a utilizat un tranzistor MOSFET în configurație "Low-Side Switch".

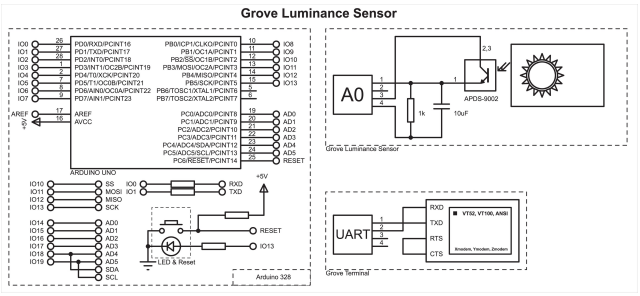

Fig. 1. Diagrama Bloc a Arhitecturii Hardware propuse: Interfațarea ESP32 cu mediul extern și fluxul de semnal.

B. Arhitectura Software și FreeRTOS

Inovația principală a lucrării constă în utilizarea capabilităților Dual-Core pentru separarea domeniilor de timp. S-au definit două task-uri principale fixate pe nuclee diferite (Core Pinning):

- **Task Senzor (Core 0):** Rulează bucla de control. Este izolat de întreruperile asincrone generate de stiva radio.
- **Task BLE (Core 1):** Rulează stiva Bluedroid și gestionează evenimentele GATT.

Transferul de date între cele două nuclee se realizează exclusiv prin **Cozi de Mesaje (Queues)**. Acest mecanism asigură "Thread Safety", eliminând riscul de corupere a datelor (Race Conditions).

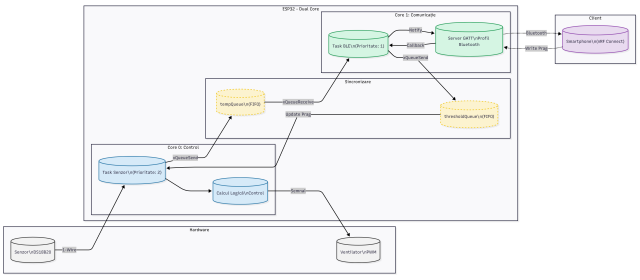

Fig. 2. Fluxul de date software între cele două nuclee ale ESP32 folosind cozile FreeRTOS.

V. IMPLEMENTARE SOFTWARE

A. Configurarea Task-urilor și Sincronizare

Inițializarea sistemului se face în funcția 'setup()', unde sunt create cozile și task-urile. Se utilizează funcția specifică

ESP32 ‘xTaskCreatePinnedToCore‘ pentru a forța execuția pe un anumit nucleu.

```
// 1. Creare Cozi (Mecanism Thread-Safe)
// Coada pentru temperaturi (Senzor -> BLE)
tempQueue = xQueueCreate(5, sizeof(float)); //
// Coada pentru praguri (BLE -> Senzor)
thresholdQueue = xQueueCreate(2, sizeof(float));
     //

// 2. Creare Task Senzor pe CORE 0 (App Core)
xTaskCreatePinnedToCore( //
    taskSenzorControl,  // Pointer la functie
    "SenzorCtrl",       // Nume pentru debug
    4096,               // Stack size (cuvinte)
    NULL,               // Parametri
    2,                  // Prioritate (Mare)
    &hTaskSenzor,       // Handle task
    0                   // Core ID (0)
);

// 3. Creare Task BLE pe CORE 1 (Pro Core)
xTaskCreatePinnedToCore( //
    taskBLE, "BLECom", 4096, NULL, 1, &hTaskBLE,
         1
);
```

Listing 1. Alocarea Task-urilor pe Nuclee

Prioritatea task-ului de senzor este setată la nivelul 2 (mai mare decât BLE - nivel 1) pentru a garanta că citirea temperaturii și decizia de răcire au prioritate față de actualizarea interfeței grafice pe telefon.

B. Configurarea PWM pentru Control Turație

Pentru a respecta cerințele de utilizare a timerelor hardware, controlul ventilatorului nu este pur digital (ON/OFF), ci utilizează perifericul LEDC al ESP32 pentru a genera semnal PWM.

```
const int pwmFreq = 5000;
const int pwmChannel = 0;
const int pwmResolution = 8;

void setupPWM() {
    // Configurare Timer Hardware
    ledcSetup(pwmChannel, pwmFreq, pwmResolution
        );
    // Atasare canal la pinul GPIO
    ledcAttachPin(FAN_PIN, pwmChannel);
}

// In Task-ul de Control:
if (temp > prag) {
    // Control Proportional simplificat
    int dutyCycle = map(temp, prag, prag+10,
        100, 255);
    ledcWrite(pwmChannel, dutyCycle);
}
```

Listing 2. Configurare Timer PWM

C. Implementarea Stivei BLE (Server GATT)

Sistemul este configurat ca un Server GATT. S-a definit un serviciu custom care expune două caracteristici (Characteristics):

1) **Temperature Characteristic (Notify/Read):** Permite telefonului să primească notificări automate (”push”) când temperatura se modifică, economisind bateria telefonului (nu este nevoie de polling).
2) **Threshold Characteristic (Write/Read):** Permite utilizatorului să trimită un nou prag de temperatură.

Callback-ul pentru scrierea pragului este critic. Deoarece este apelat din contextul stivei BLE, acesta nu trebuie să execute operații lungi. Soluția implementată trimite doar datele în coadă, delegând procesarea:

```
class ThresholdCallbacks: public
    BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pChar) { //
        std::string value = pChar->getValue();
            //
        if (value.length() > 0) {
            float newVal = (float)atof(value.
                c_str()); //
            // Nu blocam stiva BLE!
            // Trimitem valoarea in coada
            xQueueSend(thresholdQueue, &newVal,
                0); //
        }
    }
};
```

Listing 3. Callback Asincron pentru Scrierea BLE

D. Logica de Control Non-Blocantă

Pentru a gestiona timpul de conversie al senzorului ($750ms$) fără a bloca CPU-ul, se folosește funcția ‘vTaskDelay‘. Diferența fundamentală față de ‘delay()‘ din Arduino este că ‘vTaskDelay‘ semnalizează planificatorului că task-ul curent poate fi scos din execuție, permițând rularea task-ului ‘IDLE‘ sau a altor task-uri cu prioritate mai mică pe același nucleu.

VI. Rezultate Experimentale și Analiză

Această secțiune prezintă validarea funcțională a sistemului propus prin teste efectuate în condiții de laborator.

A. Analiza Răspunsului la Treaptă

Sistemul a fost testat aplicând o sursă de căldură rapidă asupra senzorului, simulând o depășire a pragului setat la 25°C. Figura 3 ilustrează evoluția temperaturii și activarea ventilatorului. Se observă o latență de reacție minimă datorată utilizării RTOS.

B. Consum Energetic

Deși nu a fost obiectivul principal, s-a estimat consumul sistemului în diferite stări, prezentat în Tabelul II. Se observă impactul major al stivei radio asupra consumului.

TABLE II
Estimarea Consumului de Curent (la 3.3V)

Stare Sistem	Consum Estimat (mA)
Idle (WiFi/BLE oprit)	$\approx$ 40 mA
BLE Advertising (Reclamă)	$\approx$ 120 mA
BLE Conectat (Transmisie)	$\approx$ 90 mA
Ventilator Pornit (Etaj Putere)	+200 mA (sursă externă)

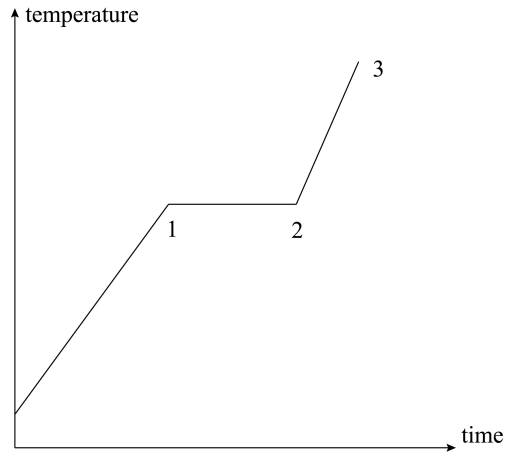

Fig. 3. Evoluția temperaturii în timp și activarea semnalului de control. Se observă stabilitatea buclei de control.

VII. Concluzii

Proiectul a demonstrat cu succes implementarea unui sistem distribuit de monitorizare termică folosind un RTOS pe un microcontroller dual-core. Arhitectura propusă rezolvă problema fundamentală a blocării resurselor prin segregarea task-urilor pe nuclee distincte și utilizarea cozilor de mesaje pentru sincronizare. Față de soluțiile clasice Arduino (Single-Threaded), soluția prezentată oferă scalabilitate și fiabilitate superioară.

References

[1] R. Barry, "Mastering the FreeRTOS Real Time Kernel," Real Time Engineers Ltd, 2016.
[2] K. Townsend, C. Cufí, A. Akiba, and R. Davidson, "Getting Started with Bluetooth Low Energy," O'Reilly Media, 2014.
[3] Espressif Systems, "ESP32 Technical Reference Manual," V4.1, 2020.
[4] S. K. Guntupalli et al., "Power consumption analysis of BLE and Wi-Fi for IoT applications," in *Int. Conf. on Electrical, Electronics, and Optimization Techniques*, 2016.
[5] Dallas Semiconductor, "DS18B20 Programmable Resolution 1-Wire Digital Thermometer Datasheet".