

LABORATORUL 02

Stările de consum și autonomia unui nod ESP32-C6

Durată: 3 ore | Suport: Capitolele 4, 5 și 10 | Platformă: ESP32-C6 DevKitC-1 | Carte capitolul de referință

CUPRINS	
1.	Obiectivele lucrării
2.	Ce construim
3.	Materiale necesare
4.	Stările de consum
5.	Calculatorul de buget energetic
6.	Pregătirea mediului de lucru
7.	Somnul adânc și memoria care supraviețuiește
8.	Sursele de trezire
9.	Scurtarea fazei active
10.	Nodul complet
11.	Protocolul de măsurare
12.	Sarcini de lucru
13.	Provocare
14.	Întrebări de verificare
15.	Resurse

Un senzor care trebuie să reziste un an pe o baterie lucrează, de fapt, sub o secundă pe oră. Restul timpului doarme. În această lucrare măsurăm cât consumă fiecare stare a unui ESP32-C6, construim bugetul energetic al unui nod real și descoperim că cea mai mare economie nu vine din a dormi mai adânc, ci din a te trezi mai puțin timp.

1 Obiectivele lucrării

- ☐ Identificarea stărilor de consum ale unui microcontroler modern și a ceea ce rămâne pornit în fiecare
- ☐ Măsurarea curentului în fiecare stare, de la sute de miliamperi la câțiva microamperi
- ☐ Programarea somnului adânc și a surselor de trezire: temporizator și pin de intrare
- ☐ Păstrarea informației peste repornire folosind memoria din domeniul mereu alimentat
- ☐ Calculul curentului mediu, al bugetului energetic și al autonomiei pe baterie
- ☐ Reducerea duratei fazei active - de departe cea mai eficientă optimizare
- ☐ Înțelegerea diferenței dintre consumul cipului și consumul plăcii de dezvoltare

2 Ce construim

Construim un **nod senzor autonom**: se trezește la un interval fix, citește temperatura și umiditatea, trimite valorile prin Wi-Fi și adoarme la loc. Apoi îl optimizăm pas cu pas și urmărim cum autonomia calculată crește de la câteva zile la peste un an - fără să schimbăm bateria și fără să schimbăm cipul.

MICROCONTROLER

ESP32-C6

RISC-V pe 32 de biți, până la 160 MHz

PARTICULARITATE

nucleu de consum redus

un al doilea nucleu la 20 MHz, care poate lucra cât timp cel principal doarme

COMUNICAȚII

Wi-Fi 6 · BLE 5 · 802.15.4

Thread și Zigbee, pe lângă Wi-Fi

DOMENIU MEREU ALIMENTAT

memorie și temporizator

supraviețuiesc somnului adânc

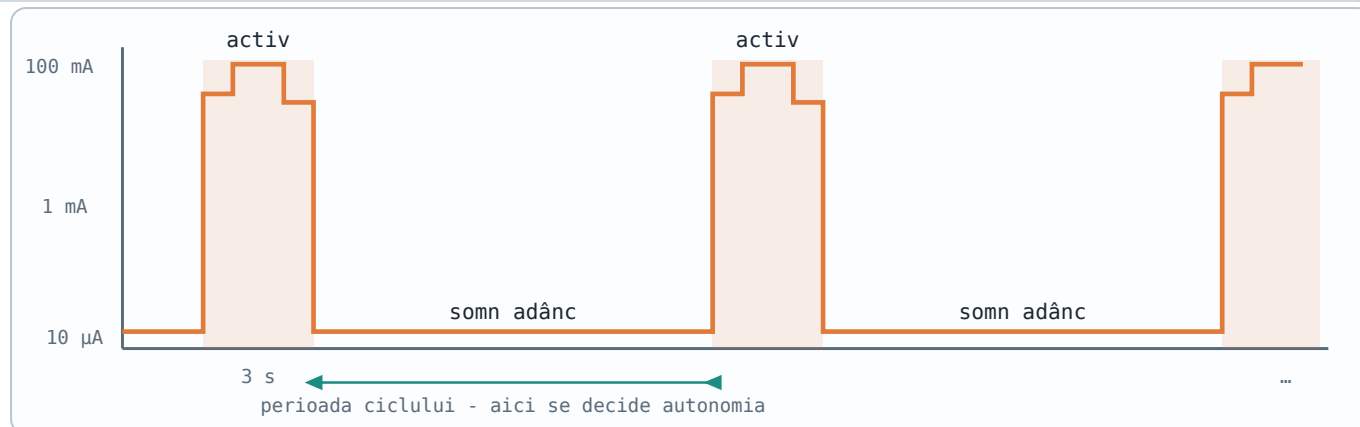


Fig. 1 - Profilul de curent al unui nod pe baterie, cu axa verticală logaritmică. Vârfurile scurte sunt emisiile Wi-Fi. Suprafața de sub curbă este sarcina consumată - și, în ciuda aparențelor, cea mai mare parte din ea se află în vârfurile înguste, nu în platoul lung de jos.

3 Materiale necesare

1

ESP32-C6-
DevKitC-1

1

Cablu USB-C
pentru date

1

Modul INA219 sau
multimetru cu
domeniu de μA

1

Senzor BME280
(I²C) sau DHT22

1

Buton tactil +
rezistor 10 k Ω

1

Breadboard

8

Fire jumper

1

Suport 2×AA sau
acumulator LiPo
(opțional)

DESPRE MĂSURAREA CURENȚILOR FOARTE MICI

Un multimetru obișnuit are, pe domeniul de μA , o rezistență internă suficient de mare încât să provoace o cădere de tensiune care resetează microcontrolerul la fiecare vârf de curent. Dacă plăcuța repornește ciudat în timpul măsurătorilor, aceasta este cauza. INA219 nu are această problemă, pentru că șuntul lui este de doar 0,1 Ω .

4 Stările de consum

Un microcontroler modern nu are două stări (pornit / oprit), ci o scară întreagă. Fiecare treaptă oprește ceva în plus și, în schimb, are nevoie de mai mult timp ca să revină la lucru.

Stare	Ce rămâne pornit	Ordin de mărime	Revenire
Activ, radio pornit	procesor, memorie, Wi-Fi în emisie	zeci-sute de mA	-
Modem-sleep	procesor și memorie; radioul se stinge între cadre	câteva zeci de mA	imediată

Light-sleep	memoria își păstrează conținutul, ceasurile se opresc	sute de μA	microsecunde, programul continuă de unde a rămas
Deep-sleep	doar domeniul mereu alimentat: temporizatorul și o memorie mică	câțiva μA	milisecunde, dar programul o ia de la capăt
Hibernare	practic nimic în afară de circuitul de trezire	sub 1 μA	ca la o pornire completă

DIFERENȚA CARE CONTEAZĂ CEL MAI MULT

La

LIGHT-SLEEP

execuția continuă din punctul în care a fost întreruptă: variabilele sunt intacte, conexiunea Wi-Fi poate fi păstrată. La

DEEP-SLEEP

, cipul repornește: `setup()` se execută din nou, iar toate variabilele obișnuite sunt pierdute. În schimb consumul este de aproximativ o sută de ori mai mic. Alegerea între ele este primul compromis de proiectare al oricărui nod pe baterie.

Capcana plăcii de dezvoltare

CITIȚI ACEST PARAGRAF ÎNAINTE DE A VĂ MIRA DE REZULTATE

Fișa tehnică promite câțiva microamperi în somn adânc. O placă de dezvoltare va consuma, cel mai probabil, de o mie de ori mai mult. Vinovații sunt

PE PLACĂ, NU ÎN CIP

:

- **CONVERTORUL USB-SERIAL**
- rămâne alimentat și consumă permanent
- **REGULATORUL DE TENSIUNE**
- are un curent propriu de repaus, uneori de ordinul miliamperilor
- **LED-UL DE ALIMENTARE**
- 2–5 mA, adică singur cât de o mie de ori cipul adormit
- **REZISTOARELE DE FORȚARE**
de pe liniile de programare

Un produs real folosește modulul gol, nu placa de dezvoltare. Măsurați totuși placa așa cum este: diferența dintre cifra din fișa tehnică și cifra măsurată este

CEA MAI VALOROASĂ LECȚIE A ACESTEI LUCRĂRI

.

5 Calculatorul de buget energetic

Introduceți curenții măsurați și perioada dorită. Calculatorul arată curentul mediu, repartiția sarcinii între cele două faze și autonomia estimată.

► CURENT MEDIU, REPARTIȚIA SARCINII ȘI AUTONOMIA

CURENT ACTIV (mA)

80

DURATA ACTIVĂ (S)

4

CURENT ÎN SOMN (mA)

15

PERIOADA CICLULUI (S)

900

CAPACITATE BATERIE (MAH)

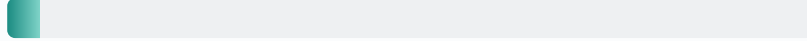
2600

faza activă



96.0 %

somn adânc



4.0 %

factor de umplere: **0.444 %**sarcină per ciclu: **92.622 μAh** (activ 88.889 + somn 3.733)curent mediu: **370.5 μA** = 0.3705 mAautonomie estimată: **292.4 zile (9.6 luni)**

Faza activă domină: scurtați-o (conectare Wi-Fi mai rapidă, mai puține măsurători) -
 reducerea curentului de somn nu ajută aproape deloc.

Autonomia calculată presupune descărcare completă și ignoră autodescărcarea bateriei și randamentul regulatorului. În practică înmulțiți rezultatul cu 0,7–0,8.

Versiune interactivă pe site.

EXPERIMENTUL DE GÂNDIRE CARE SCHIMBĂ TOTUL

Porniți de la valorile implicite (80 mA timp de 4 s, la fiecare 15 minute) și încercați două optimizări:

- **SOMN DE ZECE ORI MAI BUN:**

coborâți curentul de somn de la 15 μA la 1,5 μA. Priviți autonomia.

- **TREZIRE DE PATRU ORI MAI SCURTĂ:**

readuceți somnul la 15 μA și coborâți durata activă de la 4 s la 1 s.

A doua optimizare câștigă detașat, deși pare mai modestă. Motivul: la aceste valori faza activă domină bugetul.

De aceea efortul de inginerie trebuie pus în

A TE TREZI MAI PUȚIN TIMP

- și exact asta facem la secțiunea 9.

6 Pregătirea mediului de lucru

1

Instalați suportul pentru ESP32 în Arduino IDE

În File → Preferences → Additional Boards Manager URLs adăugați:

adresă

https://espressif.github.io/arduino-esp32/package_esp32_index.json

Apoi Tools → Board → Boards Manager, căutați „esp32” și instalați pachetul **esp32 by Espressif Systems**, versiunea 3.0 sau mai nouă. Versiunile 2.x nu cunosc ESP32-C6.

2

Selectați placa și portul

Tools → Board → ESP32 Arduino → ESP32C6 Dev Module. La Port alegeți portul care apare după conectarea plăcii.

DACĂ NU APARE NICIUN PORT

Cablul USB-C este, în multe cazuri, doar pentru încărcare. Încercați alt cablu înainte de a bănui driverul. Plăcuța C6 are două conectori USB - folosiți-l pe cel marcat USB (nu UART) pentru portul serial nativ.

3

Verificați lanțul de compilare cu un program minimal

```
test.ino

void setup() {
  Serial.begin(115200);
  delay(1000);                      // lasa portul serial nativ sa se initializeze
  Serial.println("ESP32-C6 functional");
}

void loop() {
  Serial.printf("timp: %lu ms\n", millis());
  delay(1000);
}
```

Încărcați și deschideți monitorul serial la 115200 baud. Dacă vedeți mesajele, mediul este gata.

4

Instalați biblioteca pentru senzor

Tools → Manage Libraries, căutați **Adafruit BME280** și instalați-o împreună cu dependențele propuse (Adafruit Unified Sensor).

7 Somnul adânc și memoria care supraviețuiește

Primul program demonstrează comportamentul care îi surprinde pe toți: după somnul adânc, microcontrolerul **nu continuă**, ci repornește. Singurul lucru care supraviețuiește este ce am pus în memoria din domeniul mereu alimentat.

```
somn_01_baza.ino

#include <esp_sleep.h>

#define uS_PER_S 1000000ULL
#define SOMN_S 10 // cate secunde dormim

// Atributul RTC_DATA_ATTR aseaza variabila in memoria din domeniul mereu
// alimentat. Ea supravietuieste somnului adanc; o variabila obisnuita nu.
RTC_DATA_ATTR int numar_treziri = 0;
```

```

RTC_DATA_ATTR uint32_t timp_total_activ_ms = 0;

int contor_obisnuit = 0;           // acesta va fi mereu 0 dupa trezire

void afiseaza_motivul() {
    switch (esp_sleep_get_wakeup_cause()) {
        case ESP_SLEEP_WAKEUP_TIMER: Serial.println("trezit de temporizator"); break;
        case ESP_SLEEP_WAKEUP_GPIO:  Serial.println("trezit de un pin");      break;
        case ESP_SLEEP_WAKEUP_EXT1:   Serial.println("trezit de EXT1");       break;
        default:                      Serial.println("pornire normala (nu din somn)");
    }
}

void setup() {
    uint32_t t0 = millis();
    Serial.begin(115200);
    delay(200);                      // fara aceasta pauza pierdem primele mesaje

    numar_treziri++;
    contor_obisnuit++;

    Serial.println("-----");
    afiseaza_motivul();
    Serial.printf("treziri (memorie RTC) : %d\n", numar_treziri);
    Serial.printf("contor obisnuit      : %d  <- mereu 1\n", contor_obisnuit);
    Serial.printf("timp activ cumulat   : %lu ms\n", timp_total_activ_ms);

    // Aici ar veni treaba propriu-zisa: citire senzor, transmisie.
    delay(500);

    timp_total_activ_ms += millis() - t0;

    Serial.printf("adorm %d secunde...\n", SOMN_S);
    Serial.flush();                      // golim tamponul serial INAINTE de a adormi

    esp_sleep_enable_timer_wakeup(SOMN_S * uS_PER_S);
    esp_deep_sleep_start();              // de aici incolo nu se mai executa nimic
}

void loop() {
    // Nu se ajunge niciodata aici: esp_deep_sleep_start() nu se intoarce.
}

```

TREI LUCRURI DE OBSERVAT

1. numar_treziri crește la fiecare ciclu, dar contor_obisnuit rămâne mereu 1 - memoria obișnuită se pierde.
2. Instrucțiunile de după esp_deep_sleep_start() nu se execută niciodată; funcția nu se întoarce.
3. Fără Serial.flush() ultimele caractere se pierd: cipul adoarme înainte ca ele să apuce să plece pe fir. Este o greșală care produce mesaje trunchiate și multă confuzie.

MEMORIA RTC NU ESTE NELIMITATĂ

Sunt câțiva kiloocteți. Puneți acolo doar strictul necesar: contoare, ultima valoare citită, canalul Wi-Fi. Nu încercați să stocați istoricul măsurătorilor.

8 Sursele de trezire

Temporizatorul nu este singura cale de a ieși din somn. Un nod real trebuie să reacționeze și la evenimente externe: un buton, un senzor de mișcare, un contact de ușă.

somn_02_trezire.ino

```
#include <esp_sleep.h>

#define uS_PER_S    1000000ULL
#define SOMN_S      30
#define PIN_BUTON   GPIO_NUM_4    // pe C6, trezirea prin pin cere GPIO0..GPIO7

RTC_DATA_ATTR int treziri_temporizator = 0;
RTC_DATA_ATTR int treziri_buton = 0;

void setup() {
    Serial.begin(115200);
    delay(200);

    esp_sleep_wakeup_cause_t motiv = esp_sleep_get_wakeup_cause();
    if (motiv == ESP_SLEEP_WAKEUP_TIMER) {
        treziri_temporizator++;
        Serial.println("temporizator - masuratoare programata");
    } else if (motiv == ESP_SLEEP_WAKEUP_GPIO) {
        treziri_buton++;
        Serial.println("BUTON - eveniment neasteptat!");
    } else {
        Serial.println("pornire de la zero");
    }
    Serial.printf("temporizator: %d   buton: %d\n",
                  treziri_temporizator, treziri_buton);

    // Butonul leaga pinul la masa cand este apasat, deci il tinem sus printr-o
    // rezistenta interna si ne trezim pe front descendent.
    pinMode(PIN_BUTON, INPUT_PULLUP);

    // Doua surse de trezire active simultan.
    esp_sleep_enable_timer_wakeup(SOMN_S * uS_PER_S);
    esp_deep_sleep_enable_gpio_wakeup(BIT(PIN_BUTON), ESP_GPIO_WAKEUP_GPIO_LOW);

    Serial.println("adorm - apasati butonul sau asteptati 30 s");
    Serial.flush();
    esp_deep_sleep_start();
}

void loop() {}
```

API-UL DE TREZIRE DIFERĂ ÎNTRE VARIANTELE ESP32

Aceasta este una dintre cele mai frecvente surse de erori la trecerea de la un cip la altul:

Cip	Funcția de trezire prin pin	Pini permiși
ESP32 clasic	<code>esp_sleep_enable_ext0_wakeup()</code> / <code>ext1</code>	pinii RTC
ESP32-C3 / C6 / H2	<code>esp_deep_sleep_enable_gpio_wakeup()</code>	GPIO0–GPIO7
ESP32-S3	ambele variante	pinii RTC

Verificați întotdeauna documentația pentru cipul concret. Un program scris pentru ESP32 clasic

NU SE COMPILEAZĂ

pe C6, iar mesajul de eroare nu spune limpede de ce.

Trezirea din light-sleep

Când aveți nevoie ca programul să continue de unde a rămas - de exemplu pentru a păstra o conexiune deschisă - folosiți somnul ușor. Diferența în cod este minimă, dar comportamentul este cu totul altul:

light sleep - programul continuă

```
esp_sleep_enable_timer_wakeup(5 * uS_PER_S);
esp_light_sleep_start();           // aceasta functie SE INTOARCE

Serial.println("m-am trezit si continui de aici");
Serial.printf("variabilele sunt intacte: %d\n", contor_obisnuit);
```

9 Scurtarea fazei active

Calculatorul de la secțiunea 5 ne-a arătat unde se duce energia. Acum atacăm faza activă. Un nod neoptimizat petrece cea mai mare parte a timpului de veghe **așteptând să se conecteze la Wi-Fi**, nu citind senzorul.

Ce durează, de fapt, o conectare

Etapă	Durată tipică	Se poate elimina?
Scanarea tuturor canalelor	1–3 s	da - dacă memorăm canalul și adresa punctului de acces
Asocierea și autentificarea	100–300 ms	parțial
Obținerea adresei prin DHCP	0,5–2 s	da - cu adresă IP fixă
Transmisia propriu-zisă	10–50 ms	nu, dar este oricum neglijabilă

OBSERVAȚIA CENTRALĂ

Datele utile ocupă sub 5 % din timpul de veghe. Restul este pregătire - și aproape toată poate fi memorată de la ciclul anterior în memoria RTC.

wifi_rapid.ino - conectare cu date memorate

```

#include <WiFi.h>
#include <esp_wifi.h>

const char* SSID = "reteaua_laboratorului";
const char* PAROLA = "parola";

// Tot ce merita memorat de la ciclul anterior, ca sa nu-l refacem.
RTC_DATA_ATTR uint8_t canal_salvat = 0;
RTC_DATA_ATTR uint8_t bssid_salvat[6];
RTC_DATA_ATTR bool avem_date_salvate = false;
RTC_DATA_ATTR uint32_t esecuri_la_rand = 0;

// Adresa fixa elimina complet negocierea DHCP.
IPAddress ip_local(192, 168, 1, 77);
IPAddress poarta (192, 168, 1, 1);
IPAddress masca (255, 255, 255, 0);
IPAddress dns (192, 168, 1, 1);

bool conecteaza(uint32_t limita_ms = 8000) {
    uint32_t t0 = millis();

    WiFi.mode(WIFI_STA);
    WiFi.config(ip_local, poarta, masca, dns); // fara DHCP
    WiFi.setSleep(true); // radioul se stinge intre cadre

    if (avem_date_salvate) {
        // Stim exact pe ce canal si la ce punct de acces: fara scanare.
        WiFi.begin(SSID, PAROLA, canal_salvat, bssid_salvat);
    } else {
        WiFi.begin(SSID, PAROLA); // prima data, cautam
    }

    while (WiFi.status() != WL_CONNECTED && millis() - t0 < limita_ms) {
        delay(10);
    }

    if (WiFi.status() != WL_CONNECTED) {
        esecuri_la_rand++;
        // Dupa trei esecuri banuim ca punctul de acces s-a mutat pe alt canal
        // si stergem datele memorate, ca sa reluam cu scanare completa.
        if (esecuri_la_rand >= 3) avem_date_salvate = false;
        return false;
    }

    canal_salvat = WiFi.channel();
    memcpy(bssid_salvat, WiFi.BSSID(), 6);
    avem_date_salvate = true;
    esecuri_la_rand = 0;

    Serial.printf("conectat in %lu ms (canal %d, IP %s)\n",
        millis() - t0, canal_salvat, WiFi.localIP().toString().c_str());
    return true;
}

```

DE CE CONTORUL DE EȘECURI ESTE OBLIGATORIU

Punctele de acces își schimbă canalul singure, ca să evite interferențele. Dacă nodul insistă pe canalul memorat, rămâne deconectat pentru totdeauna. Trei încercări eșuate declanșează o scanare completă. Fără această plasă de siguranță, optimizarea transformă nodul într-un dispozitiv care moare silențios după câteva săptămâni - un tip de defect foarte greu de diagnosticat pe teren.

Măsurăți câștigul

Rulați conectarea de zece ori cu `avem_date_salvate = false` forțat, apoi de zece ori normal. Notați timpul mediu în fiecare caz și recalculați autonomia cu widgetul de la secțiunea 5. Diferența este, de obicei, un factor de trei până la cinci.

10 Nodul complet

Punem totul cap la cap: trezire, citire senzor, transmisie rapidă, adormire. Programul își măsoară singur durata fazei active și o raportează, ca să putem urmări efectul optimizărilor.

nod_autonom.ino

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <esp_sleep.h>

#define uS_PER_S      1000000ULL
#define PERIOADA_S    900           // 15 minute
#define SDA_PIN       6
#define SCL_PIN       7

const char* SSID      = "reteaua_laboratorului";
const char* PAROLA     = "parola";
const char* SERVER    = "http://192.168.1.10:8000/masuratori";

RTC_DATA_ATTR uint32_t ciclu = 0;
RTC_DATA_ATTR uint32_t timp_activ_total_ms = 0;
RTC_DATA_ATTR uint8_t  canal_salvat = 0;
RTC_DATA_ATTR uint8_t  bssid_salvat[6];
RTC_DATA_ATTR bool     avem_date_salvate = false;

Adafruit_BME280 bme;

void adormi(const char* motiv) {
    timp_activ_total_ms += millis();
    Serial.printf("[%s] activ %lu ms (cumulat %lu ms in %lu cicluri)\n",
                  motiv, millis(), timp_activ_total_ms, ciclu);
    Serial.flush();

    WiFi.disconnect(true);           // oprim radioul explicit
    WiFi.mode(WIFI_OFF);

    esp_sleep_enable_timer_wakeup((uint64_t) PERIOADA_S * uS_PER_S);
    esp_deep_sleep_start();
}
```

```

void setup() {
  Serial.begin(115200);
  delay(150);
  ciclu++;

  // --- 1. senzorul, cat mai devreme: se poate citi fara Wi-Fi pornit ---
  Wire.begin(SDA_PIN, SCL_PIN);
  if (!bme.begin(0x76)) { // unele module raspund la 0x77
    Serial.println("senzorul nu raspunde");
    adormi("fara senzor");
  }

  // Modul forțat: senzorul face o singura masuratoare la cerere si se opreste.
  // Este exact ce ne trebuie - un senzor care masoara continuu consuma degeaba.
  bme.setSampling(Adafruit_BME280::MODE_FORCED,
                  Adafruit_BME280::SAMPLING_X1, // temperatura
                  Adafruit_BME280::SAMPLING_X1, // presiune
                  Adafruit_BME280::SAMPLING_X1, // umiditate
                  Adafruit_BME280::FILTER_OFF);
  bme.takeForcedMeasurement();

  float temperatura = bme.readTemperature();
  float umiditate    = bme.readHumidity();
  float presiune     = bme.readPressure() / 100.0F;

  Serial.printf("ciclul %lu: %.2f gradeC  %.1f%%  %.1f hPa\n",
               ciclu, temperatura, umiditate, presiune);

  // --- 2. rețeaua, cat mai tarziu si cat mai scurt ---
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(true);
  if (avem_date_salvate) WiFi.begin(SSID, PAROLA, canal_salvat, bssid_salvat);
  else WiFi.begin(SSID, PAROLA);

  uint32_t t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 8000) delay(10);
  if (WiFi.status() != WL_CONNECTED) {
    avem_date_salvate = false; // reluam cu scanare data viitoare
    adormi("fara rețea");
  }
  canal_salvat = WiFi.channel();
  memcpy(bssid_salvat, WiFi.BSSID(), 6);
  avem_date_salvate = true;

  // --- 3. transmisia ---
  char corp[160];
  snprintf(corp, sizeof(corp),
           "{\"ciclu\":%lu,\"temp\":%.2f,\"umid\":%.1f,\"pres\":%.1f,\"activ_ms\":%lu}",
           ciclu, temperatura, umiditate, presiune, millis());

  HTTPClient http;
  http.begin(SERVER);
  http.addHeader("Content-Type", "application/json");
  int cod = http.POST(corp);
  Serial.printf("server: %d\n", cod);
  http.end();

  adormi("ok");
}

```

```
void loop() {}
```

ORDINEA OPERAȚIILOR NU ESTE ÎNTÂMPLĂTOARE

Senzorul se citește

ÎNAINTE

de a porni Wi-Fi-ul. Dacă am face invers, radioul ar sta pornit degeaba în timpul celor 10–20 ms de măsurătoare - o risipă mică luată separat, dar care se adună de zeci de mii de ori pe durata de viață a bateriei. Aceeași logică explică modul „forțat” al senzorului: măsoară o singură dată, la comandă, apoi se oprește singur.

11 Protocolul de măsurare

Completați acest tabel cu propriile valori. Coloana din dreapta este cea care contează pentru referat.

Stare măsurată	Cum o provocați	Curent măsurat
Activ, Wi-Fi conectat și transmite	în timpul apelului <code>http.POST()</code>	 mA
Activ, Wi-Fi pornit dar inactiv	după conectare, în buclă de <code>delay()</code>	 mA
Activ, Wi-Fi oprit	<code>WiFi.mode(WIFI_OFF)</code> apoi buclă goală	 mA
Light-sleep	<code>esp_light_sleep_start()</code> cu trezire la 30 s	 μA
Deep-sleep	<code>esp_deep_sleep_start()</code>	 μA
Deep-sleep, LED de alimentare dezlipit	opțional, doar cu acordul cadrului didactic	 μA

Cum obțineți o măsurătoare corectă

- ☐ Alimentați plăcuța de la baterie sau de la o sursă separată, nu prin USB de la calculator
- ☐ Deconectați cablul de programare în timpul măsurătorii - convertorul USB adaugă un consum propriu
- ☐ Așteptați cel puțin 5 secunde după intrarea în somn înainte de a citi: primul interval conține tranzitorii
- ☐ Pentru faza activă folosiți valoarea **medie** pe toată durata, nu vârful
- ☐ Repetați fiecare măsurătoare de trei ori și notați dispersia

MĂSURAREA VÂRFURILOR DE CURENT

Emisia Wi-Fi durează câteva milisecunde și poate depăși 300 mA. Un multimetru obișnuit, care face o citire la fiecare câteva sute de milisecunde, nu va vedea niciodată acest vârf. Dacă aveți acces la un osciloscop, măsurați căderea de tensiune pe un rezistor de $1\ \Omega$ montat în serie - imaginea vârfurilor este spectaculoasă și explică de ce unele baterii cu rezistență internă mare nu pot alimenta un nod Wi-Fi, deși au capacitate suficientă.

12 Sarcini de lucru

1. Încărcați `somn_01_baza.ino` și lăsați-l să ruleze zece cicluri. Explicați în referat de ce `numar_treziri` crește iar `contor_obisnuit` nu.
2. Completați integral tabelul de măsurători de la secțiunea 11.
3. Calculați, cu widgetul de la secțiunea 5, autonomia nodului pe două acumulatori AA (aproximativ 2600 mAh) pentru perioade de 1, 15 și 60 de minute. Realizați un grafic autonomie–perioadă.
4. Măsurați durata fazei active **înainte** și **după** optimizările de la secțiunea 9 (canal memorat și adresă IP fixă). Recalculați autonomia în ambele cazuri și exprimați câștigul în procente.
5. Adăugați trezirea prin buton la nodul complet, astfel încât o apăsare să provoace o măsurătoare imediată, în afara programului obișnuit. Verificați, prin numărătorul din memoria RTC, că ambele surse funcționează.
6. Comparați light-sleep cu deep-sleep pentru o perioadă de 5 secunde. La ce perioadă devine deep-sleep mai avantajos, ținând cont că repornirea completă durează și ea, și consumă?

13 Provocare**NODUL CARE ÎȘI ALEGE SINGUR RITMUL**

Un senzor de temperatură care raportează la fiecare 15 minute risipește energie când temperatura nu se schimbă și pierde informație când aceasta se schimbă brusc. Implementați un

INTERVAL ADAPTIV

:

- păstrați în memoria RTC ultima valoare transmisă;
- dacă valoarea nouă diferă cu mai puțin de $0,3\ ^\circ\text{C}$,
NU TRANSMITEȚI DELOC
și dublați intervalul de somn (până la maximum o oră);
- dacă diferă cu peste $1\ ^\circ\text{C}$, transmiteți și reduceți intervalul la minimum (un minut);
- transmiteți totuși obligatoriu cel puțin o dată pe oră, ca serverul să știe că nodul este viu.

Măsurați consumul mediu pe o oră, o dată într-o încăpere cu temperatură stabilă și o dată cu senzorul ținut în mână. Discutați în referat compromisul dintre

AUTONOMIE

și

REZOLUȚIA TEMPORALĂ

a datelor, și de ce regula „cel puțin o dată pe oră” este obligatorie într-un sistem real.

14 Întrebări de verificare

Ce se întâmplă cu variabilele obișnuite după un ciclu de somn adânc?

Își păstrează valoarea

Se pierde, pentru că microcontrolerul repornește

Sunt salvate automat pe memoria flash

Devin zero doar dacă sunt globale

Un nod consumă 80 mA timp de 4 s, apoi 15 μ A restul unei perioade de 900 s. Care fază domină bugetul?

Somnul, pentru că durează mult mai mult

Faza activă

Contribuie în mod egal

Depinde de tensiunea bateriei

De ce consumă o placă de dezvoltare mult mai mult decât cifra din fișa tehnică a cipului?

Fișa tehnică este greșită

Din cauza componentelor de pe placă: convertorul USB, regulatorul și LED-ul de alimentare

Pentru că microcontrolerul nu intră cu adevărat în somn

Din cauza lungimii traseelor

Care este diferența esențială dintre light-sleep și deep-sleep?

Light-sleep consumă mai puțin

La light-sleep execuția continuă de unde a rămas, la deep-sleep programul repornește

Deep-sleep nu permite trezirea prin pin

Nu există diferență la ESP32-C6

De ce memorăm canalul Wi-Fi și adresa punctului de acces în memoria RTC?

Ca să economisim memorie flash

Ca să evităm scanarea tuturor canalelor, care durează secunde întregi

Pentru că altfel parola se pierde

Ca să mărim raza de acțiune

De ce este obligatoriu Serial.flush() înainte de a adormi?

Ca să golim memoria RAM

Pentru că altfel cipul adoarme înainte ca ultimele caractere să apuce să fie transmise

Ca să resetăm portul serial

Nu este obligatoriu, doar recomandat

De ce citim senzorul înainte de a porni Wi-Fi-ul?

Pentru că senzorul nu funcționează cu radioul pornit

Ca radioul să nu stea alimentat degeaba în timpul măsurătorii

Pentru că I²C și Wi-Fi folosesc aceiași pini

Ca să evităm interferențele electromagnetice

Răspunsuri: 1-b, 2-b, 3-b, 4-b, 5-b, 6-b, 7-b

1. Somnul adânc oprește alimentarea memoriei principale. La trezire se execută din nou setup(). Doar variabilele marcate RTC_DATA_ATTR supraviețuiesc.
2. Faza activă: $80 \text{ mA} \cdot 4 \text{ s} = 320 \text{ mA}\cdot\text{s}$. Somnul: $0,015 \text{ mA} \cdot 896 \text{ s} \approx 13,4 \text{ mA}\cdot\text{s}$. Faza activă consumă de aproape 24 de ori mai mult, deși durează de 224 de ori mai puțin.
3. LED-ul singur poate consuma 2–5 mA, adică de sute de ori cât cipul adormit. Un produs real folosește modulul gol, nu placa de dezvoltare.
4. Light-sleep păstrează memoria și contextul de execuție, deci esp_light_sleep_start() se întoarce. Deep-sleep oprește alimentarea memoriei și repornește cipul, cu un consum de aproximativ o sută de ori mai mic.
5. Scanarea completă durează 1–3 secunde din faza activă. Cunoscând canalul și adresa, conectarea coboară sub o secundă - cea mai mare economie posibilă la un nod pe baterie.
6. Transmiterea serială este tamponată și se face în fundal. esp_deep_sleep_start() oprește totul instantaneu, deci mesajele rămase în tampon se pierd.
7. Radioul consumă zeci de miliamperi. Fiecare milisecundă în care este pornit fără să transmită este energie pierdută - și se adună peste zeci de mii de cicluri.

15 Resurse

Capitolul 4 - Consumul de energie în sistemele embedded

Capitolul 5 - Tehnici de optimizare a consumului energetic

Capitolul 10 - Energy harvesting și sisteme autonome energetice

Documentația Espressif - modurile de somn pe ESP32-C6

Documentația nucleului Arduino pentru ESP32