Projekt i realizacja autonomicznego rejestratora parametrów ruchu GPS z interfejsem bezprzewodowym opartym o układ ESP32

Paweł

January 7, 2026

Contents

1 Wstęp

Rozwój technologii Internetu Rzeczy (IoT - Internet of Things) umożliwił miniaturyzację systemów telemetrycznych, które jeszcze dekadę temu wymagałyby rozbudowanej infrastruktury sprzętowej. Celem niniejszej pracy jest zaprojektowanie i wykonanie urządzenia typu "GPS Tracker", służącego do rejestracji trasy, monitorowania parametrów ruchu oraz wizualizacji danych w czasie rzeczywistym.

Głównym założeniem projektu było stworzenie systemu autonomicznego, zasilanego bateryjnie, który łączy w sobie funkcjonalność klasycznego rejestratora danych (tzw. "czarnej skrzynki") z nowoczesnym interfejsem użytkownika dostępnym przez przeglądarkę internetową (Web Interface).

1.1 Koncepcja architektury systemu telemetrycznego

Głównym wyzwaniem inżynierskim przy projektowaniu przenośnych rejestratorów danych jest znalezienie kompromisu między mocą obliczeniową a zużyciem energii. W przeciwieństwie do klasycznych, pasywnych logerów zapisujących jedynie strumień danych, nowoczesne rozwiązania IoT dążą do paradygmatu *Edge Computing* – przetwarzania danych bezpośrednio na urządzeniu.

W niniejszym projekcie przyjęto założenie, że urządzenie nie tylko gromadzi surowe dane pomiarowe, ale pełni również rolę niezależnego serwera aplikacji, hostującego kompletny interfejs graficzny. Takie podejście wymusiło rezygnację z prostych mikrokontrolerów 8-bitowych na rzecz zaawansowanych układów SoC (*System on Chip*).

Do realizacji zadania wybrano platformę **ESP32**, kierując się jej unikalną architekturą dwurdzeniową. Pozwala ona na separację procesów krytycznych (obsługa przerwań GPS i IMU) od obsługi stosu sieciowego WiFi, co jest kluczowe dla zachowania ciągłości zapisu danych. Dodatkowo, wysoki stopień integracji peryferiów w jednym układzie krzemowym pozwolił zminimalizować wymiary urządzenia oraz uprościć proces montażu prototypu w warunkach laboratoryjnych.

Projekt został zrealizowany jako unikatowe rozwiązanie inżynierskie (Proof of Concept), mające na celu weryfikację możliwości budowy kompaktowego systemu telemetrycznego w oparciu o tanie i powszechnie dostępne komponenty COTS (Commercial Off-The-Shelf).

2 Przegląd technologii i narzędzi

Realizacja złożonego systemu wbudowanego wymaga integracji wielu standardów komunikacyjnych oraz zrozumienia protokołów wymiany danych. Poniżej przedstawiono kluczowe technologie wykorzystane w projekcie.

2.1 System globalnego pozycjonowania (GNSS)

Podstawą działania trackera jest odbiornik GPS, komunikujący się z mikrokontrolerem za pomocą protokołu tekstowego NMEA 0183. Moduł odbiorczy (w tym projekcie u-blox NEO-6M) wysła sekwencje danych, z których najważniejsze dla nawigacji są ramki:

- **$GPGGA** (Global Positioning System Fix Data) – zawierająca informacje o szerokości i długości geograficznej, wysokości n.p.m. oraz liczbie widocznych satelitów.
- **$GPRMC** (Recommended Minimum Specific GPS/Transit Data) – dostarczająca dane o prędkości oraz kursie przemieszczania się.

Dla mikrokontrolera obsługa strumienia danych NMEA jest zadaniem czasochłonnym, wymagającym ciągłego parsowania łańcuchów znaków (String Parsing), co w przypadku ESP32 realizowane jest sprzętowo i programowo z wykorzystaniem dedykowanych bibliotek.

2.2 Magistrale komunikacyjne w systemach wbudowanych

W projekcie wykorzystano trzy główne standardy komunikacji szeregowej, co pozwoliło na optymalne rozłożenie obciążenia i podłączenie różnorodnych peryferiów.

- **I2C (Inter-Integrated Circuit):** Magistrala dwuprzewodowa (SDA - linia danych, SCL - linia zegarowa), wykorzystywana do komunikacji z urządzeniami znajdującymi się w bliskiej odległości. W projekcie na jednej szynie I2C pracują równolegle wyświetlacz OLED oraz czujnik inercyjny IMU. Rozróżnienie urządzeń następuje poprzez unikalne adresy sprzętowe (0x3C dla OLED, 0x68 dla MPU).
- **SPI (Serial Peripheral Interface):** Magistrala synchroniczna o wysokiej przepustowości (Full Duplex). Została wybrana do obsługi karty pamięci SD, ponieważ zapis logów wymaga szybkiego transferu bloków danych, czego wolniejszy interfejs I2C mógłby nie zapewnić.
- **UART (Universal Asynchronous Receiver-Transmitter):** Asynchroniczny port szeregowy. Wykorzystywany do komunikacji z modułem GPS. ESP32 posiada sprzętowe porty UART (Hardware Serial), co odciąża procesor od konieczności programowej emulacji transmisji (SoftwareSerial), zapewniając stabilność odbioru danych nawet przy wysokich częstotliwościach pracy.

3 Projekt sprzętowy

Warstwa sprzętowa urządzenia została zaprojektowana w sposób modułowy, co ułatwia diagnostykę oraz ewentualną wymianę komponentów. Sercem układu jest płytka rozwojowa ESP32 DevKit V1.

[Rysunek 1: Schemat blokowy połączeń systemu (ESP32 w centrum, peryferia wokół)]

3.1 Szczegółowa konfiguracja interfejsów

Wszystkie moduły peryferyjne zostały połączone z mikrokontrolerem zgodnie z załączonym schematem elektrycznym.

[Rysunek 1a: Schemat elektryczny połączeń]

3.2 System zasilania i stabilizacja

Jednym z największych wyzwań w projektach mobilnych jest zapewnienie stabilnego zasilania dla zróżnicowanych komponentów. Źródłem energii w systemie jest ogniwo litowo-jonowe typu 18650 o napięciu nominalnym 3.7V (zakres roboczy 3.0V - 4.2V).

Bezpośrednie zasilanie modułów z akumulatora byłoby ryzykowne ze względu na spadek napięcia w miarę rozładowywania. Moduł GPS oraz niektóre czytniki kart SD wymagają stabilnego napięcia, często w standardzie 5V. Dlatego w projekcie zastosowano przetwornicę podwyższającą napięcie (Step-Up Converter) do poziomu 5V, które następnie podawane jest na pin VIN płytki ESP32. Wbudowany w płytkę ESP32 stabilizator liniowy obniża to napięcie do bezpiecznych 3.3V dla rdzenia procesora.

Wspólna Masa (Common Ground): Fundamentalnym wymogiem poprawności działania układu, a szczególnie magistral cyfrowych (I2C, UART, SPI), jest połączenie wszystkich mas (GND) modułów w jeden wspólny potencjał. Brak wspólnej masy prowadziłby do powstawania tzw. pętli mas oraz pływania poziomów logicznych, co uniemożliwiłoby interpretację stanów logicznych (0/1) przez mikrokontroler.

[Rysunek 2: Schemat ideowy sekcji zasilania]

4 Implementacja oprogramowania

Oprogramowanie układowe (Firmware) zostało napisane w języku C++. Zdecydowano się na wykorzystanie środowiska PlatformIO, które znacznie ułatwia zarządzanie zewnętrznymi bibliotekami w porównaniu do standardowego Arduino IDE.

Kod programu jest strukturą wielowątkową, co oznacza, że procesor wykonuje kilka zadań "jednocześnie" (w praktyce szybko się między nimi przełącza lub wykonuje je na dwóch rdzeniach). Wyróżniamy dwie główne sfery działania:

1. **Rdzeń Logiczny:** Zajmuje się "brudną robotą" - ciągle pyta GPS o pozycję, sprawdza czujniki ruchu i zapisuje wszystko na kartę pamięci. To dzieje się w głównej pętli programu (funkcja `loop`).

2. **Rdzeń Komunikacyjny:** Odpowiada za obsługę WiFi i strony internetowej. Kiedy użytkownik klika przycisk w telefonie, to właśnie ten wątek reaguje.

4.1 Bezpieczeństwo danych (Mutex)

Największym wyzwaniem przy dwóch działających równolegle wątkach jest dostęp do wspólnych zasobów, takich jak karta SD. Wyobraźmy sobie sytuację, w której wątek logiczny próbuje zapisać współrzędne trasy, a w tym samym ułamku sekundy wątek internetowy chce odczytać listę plików, by wyświetlić ją na ekranie telefonu. Bez odpowiedniego nadzoru mogłoby to doprowadzić do pomieszania danych i uszkodzenia plików.

Rozwiązaniem jest zastosowanie tzw. **Mutexu** (od ang. *Mutual Exclusion* - Wzajemne Wykluczanie). Działa on jak klucz do toalety w pociągu - w danym momencie tylko jedna osoba (wątek) może z niej korzystać. Jeśli wątek zapisujący dane "weźmie klucz" (wywoła funkcję `xSemaphoreTake`), drugi wątek musi grzecznie poczekać w kolejce, aż klucz zostanie zwrócony (`xSemaphoreGive`).

W kodzie wygląda to następująco:

```
// Czekaj na dostęp do karty SD (maksymalnie 10ms)
if(xSemaphoreTake(sdMutex, ...) == true) {
    // Teraz mamy wyłączność! Możemy bezpiecznie czytać/pisać.
    zapiszDaneNaKarte();

    // Koniec pracy, oddajemy klucz innym.
    xSemaphoreGive(sdMutex);
}
```

Dzięki temu mechanizmowi mamy 100% pewności, że dane na karcie się nie uszkodzą, nawet przy intensywnej pracy urządzenia.

4.2 Struktura danych (Współdzielony Status)

Aby wątek internetowy wiedział, co aktualnie dzieje się z urządzeniem (np. jaka jest prędkość), musi "podglądać" zmienne z wątku głównego. Aby uniknąć sytuacji, w której odczytamy "starą" szerokość geograficzną i "nową" długość (co dałoby błędną pozycję), wszystkie kluczowe dane zostały zgrupowane w jeden "kontener" (strukturę):

```
struct TrackerStatus {
    double latitude;  // Szerokość geog.
    double longitude; // Długość geog.
    float speed;      // Prędkość
    float battery;    // Napięcie baterii
    // ... inne parametry
} sharedStatus;
```

Aktualizacja tego kontenera również jest chroniona wspomnianym wcześniej kluczem (Mutexem), więc serwer WWW zawsze pobiera kompletny i spójny zestaw danych.

4.3 Logika sterowania (Maszyna Stanów)

Zamiast pisać skomplikowane warunki "jeśli to i tamto", program został podzielony na jasne tryby pracy, podobnie jak w magnetowidzie. Nazywamy to **Maszyną Stanów**. Urządzenie zawsze znajduje się w jednym z trzech stanów:

- **IDLE (Czuwanie):** Urządzenie jest włączone, łapie sygnał GPS, można połączyć się z nim przez WiFi, ale nie zapisuje trasy na kartę SD. Dioda statusowa (jeśli jest) mruga powoli.
- **RECORDING (Nagrywanie):** Użytkownik wcisnął "START". Każdy nowy punkt GPS jest dopisywany do pliku na karcie.
- **PAUSED (Pauza):** Nagrywanie wstrzymane (ręcznie lub automatycznie na postoju). Plik jest otwarty, ale system czeka na ruch. Co ważne, w momencie przejścia w pauzę, wszystkie dane z pamięci podręcznej są natychmiast zrzucane na kartę ("Flush"), aby nie zginęły w razie wyłączenia prądu.

4.4 Interfejs WWW (Strona wbudowana w układ)

Unikalną cechą tego projektu jest to, że nie potrzebuje on żadnej zewnętrznej aplikacji na telefon. Cała strona internetowa (HTML, wygląd CSS i skrypty) jest zapisana w pamięci samego procesora ESP32.

Zastosowano tu podejście tzw. **Single Page Application (SPA)**. Oznacza to, że telefon pobiera stronę tylko raz - na samym początku. Potem, zamiast przeładowywać całą stronę co sekundę (co by "zamuliło" wolne łącze), strona wysyła w tle tylko małe zapytania o dane (technologia AJAX). Procesor odpowiada krótkim tekstem w formacie JSON, np.: `{ "speed": 15.2, "batt": 4.1 }`

Telefon (przeglądarka) odbiera te liczby i sama rysuje wykresy oraz przesuwa kropkę na mapie. To genialne posunięcie, bo przesuwa ciężką pracę (rysowanie grafiki) z małego procesora ESP32 na potężny procesor smartfona.

4.5 Tryb głębokiego uśpienia (Deep Sleep)

Mimo zastosowania wydajnego akumulatora, kluczowym aspektem każdego urządzenia przenośnego jest maksymalizacja czasu pracy. W tym celu zaimplementowano funkcję **Deep Sleep** (Głębokiego Uśpienia).

Jest to specjalny tryb pracy mikrokontrolera ESP32, w którym wyłączana jest większość jego modułów wewnętrznych:

- Wyłączane jest całe radio WiFi oraz Bluetooth (największy pożeracz energii).
- Zatrzymywane są oba rdzenie procesora (CPU).
- Wyłączany jest kontroler pamięci Flash.
- Zgaszony zostaje wyświetlacz OLED.

Jedynym elementem, który pozostaje aktywny, jest szczątkowy procesor (ULP - Ultra Low Power Coprocessor) oraz kontroler RTC (Zegar Czasu Rzeczywistego), który pobiera śladowe ilości prądu rzędu kilku-kilkunastu mikroamperów (μA). W tym projekcie wybudzenie następuje poprzez sygnał sprzętowy (przycisk RESET), co przywraca pełną funkcjonalność w ciągu ułamka sekundy. Pozwala to na drastyczne oszczędzanie energii, gdy tracker nie jest używany, bez konieczności odłączania baterii.

4.6 Filtracja błędów GPS (Auto-Pause)

Tanie moduły GPS mają to do siebie, że nawet gdy leżą nieruchomo na biurku, potrafią pokazywać, że "pełzają" z prędkością 0.5-1 km/h (tzw. dryft). Aby trasa na mapie nie wyglądała jak kłębek wełny w miejscu postoju, zaimplementowano logiczny filtr: Jeśli prędkość spadnie poniżej 2 km/h na dłużej niż 5 sekund, system uznaje, że stoimy i przestaje zapisywać punkty. Wznowi zapis dopiero, gdy ruszymy szybciej.

5 Testy i analiza wyników

W celu weryfikacji założeń projektowych przeprowadzono serię testów sprawdzających niezawodność urządzenia w warunkach rzeczywistych. Skupiono się na trzech krytycznych aspektach: czasie akwizycji sygnału GPS, odporności na błędy zapisu oraz wydajności energetycznej.

5.1 Testy modułu GPS (Cold Start vs Hot Start)

Jednym z kluczowych parametrów użytkowych lokalizatora jest czas oczekiwania na ustalenie pierwszej pozycji (Time To First Fix - TTFF). Procedura testowa obejmowała uruchomienie urządzenia w otwartej przestrzeni po długim okresie nieaktywności (powyżej 4 godzin).

- **Zimny start (Cold Start):** Moduł nie posiada aktualnych danych almanachu ani efemeryd. Zaobserwowane czasy oczekiwania wynosiły od 45 sekund do nawet 15 minut w trudnych warunkach (np. gęsta zabudowa). Jest to zachowanie typowe dla modułów klasy NEO-6M bez podtrzymania bateryjnego pamięci backup.
- **Gorący start (Hot Start):** Po krótkim zaniku zasilania (poniżej 30 minut) moduł odzyskuje fix niemal natychmiastowo (zazwyczaj poniżej 5 sekund), co potwierdza poprawność działania układu zasilania.

[Tutaj należy wstawić tabelę z pomiarami czasu startu w 5-10 próbach]

5.2 Stabilność zapisu danych w sytuacjach awaryjnych

Symulowano awarię nośnika danych poprzez wyjęcie karty SD w trakcie trwania aktywnej sesji nagrywania (stan RECORDING). Oczekiwane zachowanie systemu:

1. Wykrycie błędu zapisu przez funkcję biblioteczną `SD.open()`.
2. Zalogowanie błędu na porcie szeregowym.
3. Kontynuacja pracy pozostałych modułów (wyświetlanie danych na OLED i stronie WWW) pomimo braku możliwości archiwizacji.

Testy potwierdziły, że dzięki zastosowaniu mechanizmów obsługi wyjątków (try-catch lub sprawdzeń warunkowych `if(!file)`), awaria karty SD nie powoduje zawieszenia całego mikrokontrolera (Kernel Panic), a jedynie wstrzymanie funkcji rejestrującej.

5.3 Bilans energetyczny i czas pracy

Urządzenie zasilane jest ogniwem Li-Ion 18650 o pojemności nominalnej (wstawić pojemność, np. 2600 mAh). Szacunkowy pobór prądu przez poszczególne moduły:

- ESP32 (z WiFi w trybie AP+STA): ok. 150-200 mA
- Moduł GPS: ok. 40 mA
- OLED i czujniki: ok. 10-15 mA

Łączny pobór prądu oscyluje w granicach 250 mA. Teoretyczny czas pracy wynosi zatem $T = \frac{2600mAh}{250mA} \approx 10.4h$. W praktyce, uwzględniając sprawność przetwornicy Step-Up (ok. 85-90%), realny czas pracy wynosi około 8-9 godzin, co jest wynikiem satysfakcjonującym dla jednodniowej wycieczki rowerowej lub pieszego monitoringu.

[W tym miejscu sugerowane wstawienie wykresu napięcia baterii w czasie podczas testu rozładowania]

6 Podsumowanie i wnioski

Zrealizowany projekt inżynierski spełnił postawione przed nim założenia. Udało się zbudować autonomiczny, miniaturowy system rejestracji trasy, który z powodzeniem zastępuje komercyjne rozwiązania w zastosowaniach amatorskich i półprofesjonalnych.

6.1 Osiągnięcia

Względem pierwotnych prototypów, obecna wersja urządzenia charakteryzuje się:

- **Pełną wielozadaniowością:** Rozdzielenie logiki zapisu danych od obsługi interfejsu użytkownika wyeliminowało problemy z "przycinaniem się" strony WWW.
- **Jakością danych:** Zastosowanie filtru *Auto-Pause* znacząco poprawiło czytelność generowanych śladów GPX/TXT, eliminując szum pomiarowy na postojach.
- **Ergonomią:** Interfejs WWW dostępny na smartfonie eliminuje konieczność montowania fizycznych przycisków i dużych wyświetlaczy LCD na obudowie.

6.2 Ograniczenia i plany rozwojowe

Obecna rewizja urządzenia posiada pewne ograniczenia, które wyznaczają kierunki dalszych prac rozwojowych:

1. **Brak łączności globalnej:** Urządzenie wymaga połączenia WiFi z telefonem użytkownika. Dodanie modułu GSM (np. SIM800L) pozwoliłoby na wysyłanie pozycji do chmury niezależnie od obecności operatora.

2. **Zarządzanie energią:** Obecny kod nie wykorzystuje trybów uśpienia (Light/Deep Sleep). W przyszłości planowane jest wprowadzenie algorytmu usypiania ESP32 podczas długich postojów, co mogłoby wydłużyć czas pracy na baterii nawet dwukrotnie.

3. **Obudowa:** Projekt wymaga zaprojektowania dedykowanej obudowy (np. w technologii druku 3D) zapewniającej ochronę przed wilgocią (klasa szczelności IP54), co jest kluczowe w zastosowaniach outdoorowych.

Podsumowując, zaprojektowany system stanowi solidną bazę do dalszego rozwoju systemów IoT przeznaczonych do monitoringu floty i aktywności fizycznej.