6.3. Pojemnik transportowy

6.3.1. Narzędzia

1) Język programowania
    - C/C++
2) Środowisko
    - ESP-IDF
        - używane po przejściu na ESP32
    - ESP8266 RTOS SDK
        - używane przed przejściem na ESP32
3) Testowanie kodu
    - Catch2 v3
4) Zarządzanie procesem kompilacji
    - CMake
5) Visual Studio Code z dodatkiem Espressif IDF
    - IDE używane do programowania mikrokontrolera.

6.3.2. Biblioteki

W projekcie korzystano tylko i wyłącznie z bibliotek o licencji MIT, która pozwala na edycję kodu oraz na dowolne korzystanie z bibliotek, również w celach zarobkowych, co mogłoby być ważnym aspektem z punktu widzenia firmy.

1) I2Cdev, w tym biblioteka MPU6050
    - obsługa MPU6050
2) minmea
    - parser komunikatów NMEA0183
3) cJSON
    - parser formatu JSON, używany do tworzenia wiadomości wysyłanych do serwera oraz odbieranych od serwera

Biblioteki do innych modułów niż MPU6050 (akcelerometr) zostały napisane samodzielnie z powodu braku dostępnych bibliotek na ESP-IDF SDK, licencje inne niż MIT, niewystarczające funkcje biblioteki itp. Biblioteki pisano opierając się na dokumentacjach technicznych modułów.

6.3.3. Hardware

6.3.3.1. Komponenty

1) Pojemnik transportowy
    - Metalowa kasetka o rozmiarach zewnętrznych 24,0x30,0x9,2 cm.
    - Wybrana z powodu „lepszego” wyglądu przez klienta
    - Początkowo planowane było użycie plastikowego pojemnika o wymiarach odpowiadających ¼ standardowej palety (długość, szerokość), na wzór innych przewoźników prowadzących usługi transportowe. Pojemniki miały występować w różnych wysokościach.
    - W załącznikach do sprawozdania znajduje się analiza wymiarów pojemników używanych przez innych przewoźników. Dane pochodzą ze stron internetowych przewoźników.

2) Elektrozamek
    - Zasilany napięciem 12 V.
    - Zużywa 1 A w czasie otwarcia.

Rysunek 4. Elektrozamek, źródło Botland.com.pl

3) Akumulator
   - Akumulator żelowy o napięciu 12 V, pojemności 2,3 Ah, wadze 700 g.
   - Wybrany z powodu niskiej ceny w porównaniu do akumulatorów Li-Ion (do zasilenia zamka potrzebny jeden akumulator o wartości 46,90 zł lub 3/4 akumulatory Li-Ion o wartości ~75,00/100,00 zł).

Rysunek 5. Akumulator, źródło Botland.com.pl

4) Bezpiecznik
   - Ochrona akumulatora przed zbyt wysokim prądem ładowania, bezpiecznik 1 A.
5) Gniazdo ładowania
6) Przycisk zewnętrzny płaski

Rysunek 6. Przycisk, źródło Botland.com.pl

7) Stycznik
   - Sprawdzenie zamknięcia klapy.
8) ESP32 DEVKIT V1

- Jednostka główna.
- Początkowo używane ESP8266, urządzenie zmienione na prośbę firmy.
- Napięcie zasilania 5 V.

9) SIM800l
   - Tani moduł do komunikacji poprzez sieć komórkową GSM/GPRS.
   - Napięcie zasilania 4,2 V.

Rysunek 7. SIM800l

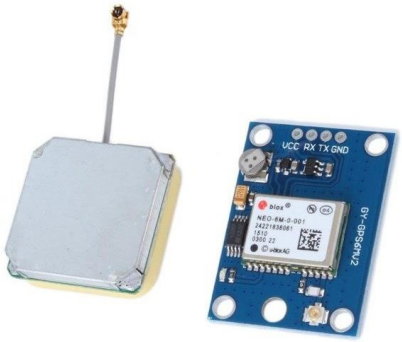
10) MPU6050
   - Akcelerometr.
   - Napięcie zasilania 3,3 V.

Rysunek 8. MPU6050, źródło sklep.msalamon.pl

11) SHT3X
   - Badanie temperatury i wilgotności wewnątrz pojemnika transportowego.
   - Napięcie zasilania 3,3 V.

Rysunek 9. SHT30, źródło sklep.msalamon.pl

12) NEO-6m-000-01
   - Moduł odczytujący koordynaty GPS.
   - Napięcie zasilania 3,3 V.

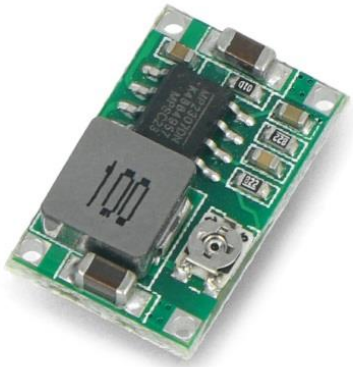


Rysunek 10. NEO-6m-000-01

13) Przetwornica step-up/step-down
- Służy dostarczeniu zasilania 12 V do elektrozamka.
- Przetwornik napięcia przetwornicą step-down LM2596 oraz przetwornicą step-up LM2577.
- Napięcie wejściowe: 4-35 V.
- Regulowane napięcie wyjściowe: 1,25-25 V.
- Maksymalny prąd wyjściowy: 3 A.
- Sprawność: 80%.

Rysunek 11. Przetwornik step-up/step-down, źródło kamami.pl

14) Dwie przetwornice step-down (MPI581)
- Służą dostarczeniu stałego zasilania 5 V od ESP32 oraz 4,2 V do SIM800l.
- Napięcie wejściowe od 4,75-23 V.
- Regulowane napięcie wyjściowe 1,0-17 V
- Maksymalny prąd wyjściowy: 3 A



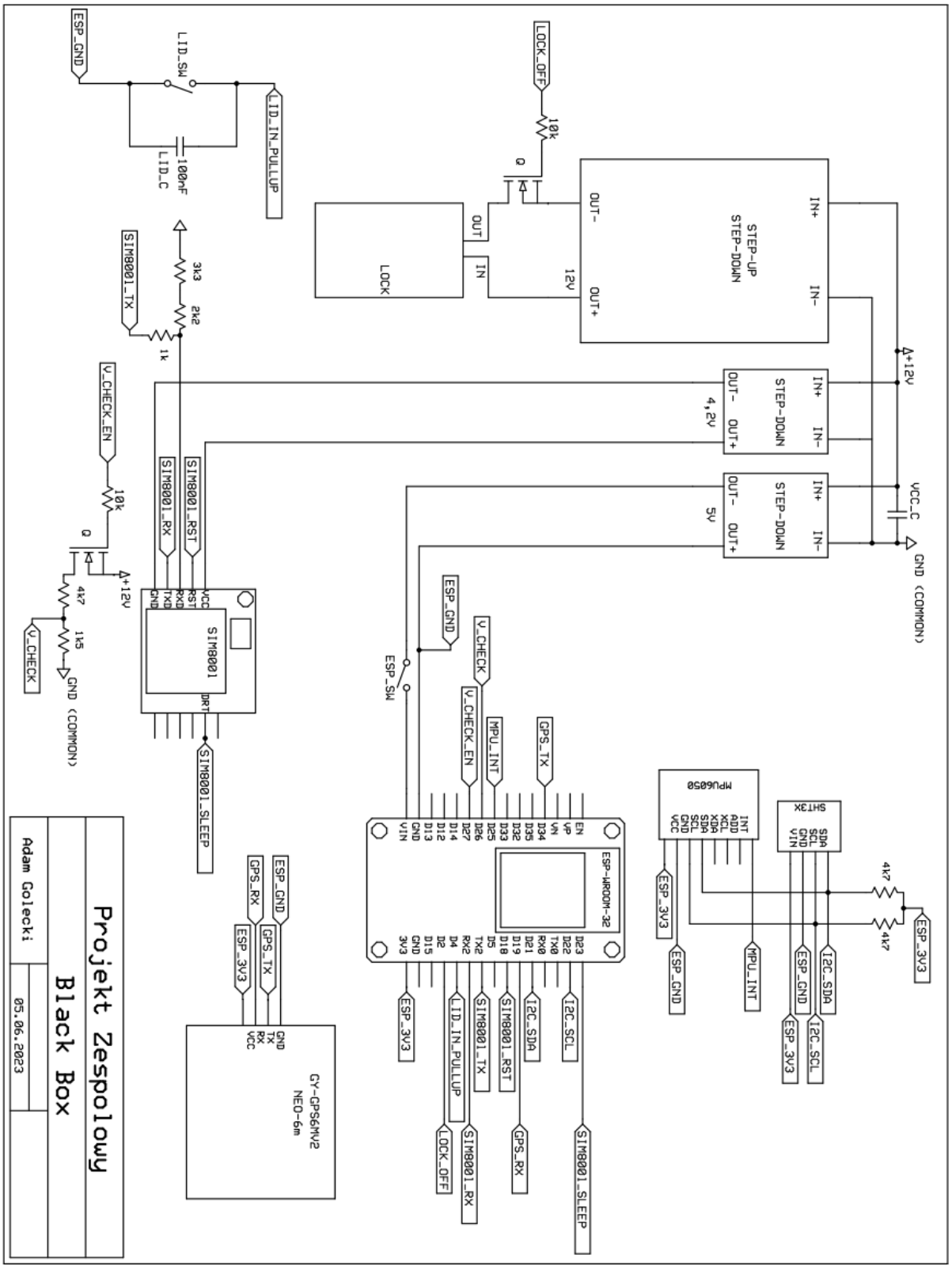
Rysunek 12. Przetwornica MPI581, źródło Botland.com.pl

15) Dwa tranzystory N-THT IRLB8721 H241/R AA03
- Służące otwieraniu elektrozamka oraz wyłączaniu dzielnika napięcia służącego badaniu napięcia baterii.
- Maksymalny prąd drenu Id: 36 A.
- Maksymalne napięcie VDSS: 100 V.
- Rezystancja kanału Rdson: 0,044 Ω.
- Obudowa: TO-220.

16) Płytka drukowana, uniwersalna 12x8 cm

6.3.3.2. Schemat podłączenia

Poniżej zamieszczono schemat wykonanego urządzenia elektronicznego. Schemat znajduje się dodatkowo w lepszej rozdzielczości w plikach do sprawozdania. Schemat wykonano w programie ExpressSCH.



Rysunek 13. Schemat urządzenia elektronicznego

Wszelkie przyciski dołączone do płytki są używane razem z kondensatorami mającymi za zadanie zminimalizować drgania styków. Pomiędzy przetwornicą a ESP32 znajduje się przełącznik mający za zadanie odcięcie zasilania bateryjnego na czas debuggingu lub wgrywania nowego programu na mikrokontroler, kiedy ESP zasilane jest przez dołączone do komputera USB.

6.3.4. Software

Jako SDK użytego do programowania ESP32 użyto polecanego przez producenta ESP-IDF. SDK używa lekkiego systemu czasu rzeczywistego FreeRTOS dedykowanego dla mikrokontrolerów. Niestety nie ma możliwości programowania ESP32 bez użycia FreeRTOS (sam ESP-IDF jest używany nawet podczas programowania ESP32 w Arduino IDE (Arduino jest tylko nakładką na ESP-IDF), tylko bez wiedzy użytkownika; dodatkowo używa ona starszej wersji ESP-IDF v4.3, kiedy dostępna jest wersja v5.0.2), dlatego zdecydowano się w pełni używać zalet płynących z FreeRTOS.

6.3.4.1. Struktura projektu

Struktura projektu jest zgodna z zaleceniami producenta, przykład na stronie z dokumentacją:

```
- myProject/
        - CMakeLists.txt
        - sdkconfig
        - components/ - component1/ - CMakeLists.txt
                                    - Kconfig
                                    - src1.c
                      - component2/ - CMakeLists.txt
                                    - Kconfig
                                    - src1.c
                                    - include/ - component2.h
        - main/       - CMakeLists.txt
                      - src1.c
                      - src2.c


        - build/
```

Główny folder projektu jest podzielony na trzy foldery. Folderu „*build*” nie należy edytować. W folderze „*components*” znajdują się biblioteki pobrane lub napisane przez programistę – a zatem również wszystkie wymienione wcześniej pobrane biblioteki. Każdy folder w „*components*” zawiera napisany przez programistę plik *CMakeLists.txt*, przykładowy plik *CMakeLists.txt* w module obsługującym pracę SIM800l:

```
set(COMPONENT_REQUIRES driver)
set(COMPONENT_PRIV_REQUIRES )

set(COMPONENT_SRCS "Sim800l.cpp" "Sim800lBB.cpp" "Sim800lESP.cpp")
set(COMPONENT_ADD_INCLUDEDIRS "include")

register_component()
```

W folderze „*main*” znajduje się plik z pętlą główną oraz plik *CMakeLists.txt* łączący wszystkie używane przez projekt komponenty:

```
idf_component_register(SRCS "main.cpp"
                    INCLUDE_DIRS "."
                    REQUIRES sim800l
                        driver
                        esp_timer
                        esp_system
                        nvs_flash
                        hal
                        I2Cdev
                        hmc5883lUni
                        bbFormats
                        gps
                        MPU6050
                        MPU6050Parser
                        SHT30
                        jsonData
                        lid
                        boxMessage
                        bbConsole)
```

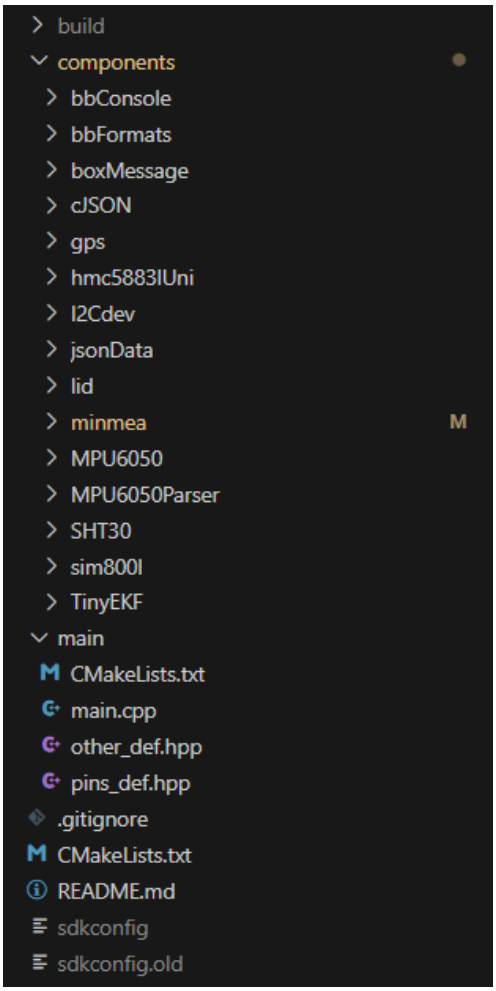
W folderze głównym projektu znajduje się plik *CMakeLists.txt* definiujący cały projekt:

```
cmake_minimum_required(VERSION 3.5)


include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(sim800l_test LANGUAGES CXX C)
```

Oraz plik *sdkconfig* zawierająca ustawienia ESP-IDF. Ustawienia te można również zmieniać w trakcie działania mikrokontrolera w programie.

Zatem struktura projektu jest następująca:



Rysunek 14. Struktura projektu

Używane komponenty:

1) bbConsole
   - dostęp konsolowy do urządzenia,
   - zmiana hasła i loginu administratora urządzenia, adresu IP serwera itp.,
2) boxMessage
   - struktura wiadomości wysyłanej do i z urządzenia,
3) cJSON
4) gps
   - obsługa modułu GPS,
5) I2Cdev
6) jsonDATA
   - funkcje używające biblioteki cJSON w celu generowania i odbierania wiadomości,
7) lid
   - obsługa pokrywy,
8) minmea
9) MPU6050
   - obsługa MP6050,
10) MPU6050Parser
   - obsługa danych przychodzących z modułu MPU6050,
11) SHT30
   - Obsługa czujnika temperatury i wilgotności SHT30,

12) sim800l
- obsługa modułu komunikacyjnego GSM/GPRS SIM800l,
- dane zwracane przez moduł są zwracane z dużymi opóźnieniami, dlatego odbieranie danych od modułu jest wykonane na zasadzie przerwań.

6.3.4.2. Program główny

Działanie mikrokontrolera polega na wykonaniu komunikacji z serwerem (wysłanie telemetrii, pobranie aktualnej wiadomości; dokładna struktura wiadomości omówiona jest w punkcie 6.2.4.2), sprawdzeniu poprawności działania pojemnika (sprawdzenie zamknięcia pokrywy itp.), a następnie przejściu w stan głębokiego uśpienia. Pobrana z serwera wiadomość zapisywana jest w pamięci trwałej urządzenia (NVS), jak również login i hasło administratora oraz adres IP serwera.

Przy pierwszym uruchomieniu użytkownik pojemnika może dostać się do jego konsoli w trakcie pierwszych 10 sekund od uruchomienia. Dostępne polecenia:

1) set url xxxxxxxxx
   - zmiana adresu serwera na xxxxxxxxx,
2) set login xxxxxxxxx
   - zmiana loginu na xxxxxxxxx,
3) set password xxxxxxxxx
   - zmiana hasła na xxxxxxxxx,
4) exit
   - wyjście z trybu konsolowego i powrót do zwykłego działania pojemnika.

W trakcie działania programu mikrokontroler odbiera przerwania z MPU6050, aby od razu otrzymać informację o przekroczeniu progu przyspieszenia; z przycisku użytkownika wymuszającego wykonanie zapytania na serwer, które pobierze z niego wiadomość; z krańcówki o informacji o zmianie stanu zamknięcia pokrywy. Po odebraniu przerwania mikrokontroler decyduje jaką czynność wykonać, np. przy aktualnej wiadomości „*open*" równej *false* nastąpi wysłanie alarmu oraz telemetrii do serwera; jeżeli jednak wartość „*open*" równa jest *true*, wówczas nastąpi jedynie sprawdzenie czy pokrywa jest zamknięta czy otwarta. Jeżeli jest otwarta, mikrokontroler zamknie dopływ prądu do zamka; jeżeli zamknięta otworzy go.

Uśpienie następuje tylko wtedy gdy pokrywa urządzenia jest zamknięta, a zatem pojemnik jest w stanie transportu. Jeżeli pokrywa jest otwarta, wówczas włączony jest timer, który czuwa nad wysyłaniem wiadomości do serwera w odpowiednim czasie. Uśpienie nie może również nastąpić w krótkim czasie po wystąpieniu alarmu – mikrokontroler blokuje na czas działania timera (10 s) przerwania danego podzespołu. Po tym czasie przerwania od podzespołu są odblokowywane w przerwaniu danego timera.

W stanie uśpienia mikrokontroler również odbiera takie przerwania, które mają za zadanie wybudzić go i jak najszybciej zareagować na dane zdarzenie. Po wybudzeniu, inicjalizacji (uruchomieniu podzespołów), pobraniu danych z pamięci trwałej, nastąpi reakcja na wiadomość na takiej samej zasadzie jak przy przerwaniach bez uśpienia.

Jeśli w trakcie głębokiego uśpienia nie nastąpiło żadne zewnętrzne przerwanie, wówczas mikrokontroler wybudza się po zadanym czasie (10 s). wykonywane są dwa takie wybudzenia, podczas których następuje pobranie danych z czujników, po czym mikrokontroler znowu przechodzi w stan uśpienia. Dane z czujników i numer wybudzenia zapisywane są w pamięci RTC, która nie jest czyszczona przy przejściu w stan głębokiego snu (atrybut ***RTC_DATA_ATTR***). Za trzecim razem mikrokontroler wysyła zapytanie na serwer.

6.3.4.3. Energooszczędność

W celu oszczędzania energii, mikrokontroler korzysta z mechanizmu uśpienia, podczas czego pobiera ~10 μA. Aby umożliwić takie zachowanie i jednocześnie być w stanie otrzymać informacje o alarmach wdrożono mechanizmy przerwań od przycisku użytkownika, krańcówki oraz akcelerometru również w trakcie głębokiego snu.

Wymagało to kalibracji akcelerometru za pomocą rejestrów oraz włączenia w nim wysyłania przerwań do mikrokontrolera, jeżeli wartość określona w rejestrze zostanie przekroczona. Akcelerometr w takiej konfiguracji pobiera ~5.2 mA.

Na czas uśpienia oraz w trakcie dwóch wybudzeń, gdy mikrokontroler nie używa modułu komunikacyjnego SIM800l zostaje uśpiony i pobiera ~1 mA. Moduł pobiera najwięcej prądu tylko podczas pierwszego włączenia, gdy łączy się z siecią GSM.

W module SHT30 nie trzeba było wdrażać specjalnych opcji. Moduł domyślnie jest ustawiony w tryb nieregularnych odczytów („*single shot mode*”) i po każdorazowym wysłaniu danych do mikrokontrolera i przechodzi w stan uśpienia. Wówczas pobiera 2 μA.

Niestety z powodu podrabianego/wadliwego modułu nie można było wysłać żadnych komend do modułu GPS, dlatego nie przechodzi on w stan uśpienia. W domyślnej konfiguracji wykonywane jest co 1 min odczyt lokalizacji.