Politechnika Śląska

PRACA MAGISTERSKA

Energooszczędny czujnik zalania STM-WiFi z funkcją powiadamiania

Adam SKÓBEL
Nr albumu: 282109

Kierunek: Teleinformatyka
Specjalność: Teleinformatyczne Systemy Autonomiczne

PROWADZĄCY PRACĘ
dr inż. Andrzej Kukiełka
KATEDRA Telekomunikacji i Teleinformatyki
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2023

Tytuł pracy

Energooszczędny czujnik zalania STM-WiFi z funkcją powiadamiania

Streszczenie

Celem pracy magisterskiej było zaprojektowanie, skonstruowanie oraz uruchomienie autonomicznego i energooszczędnego czujnika zalania, wykorzystującego mikrokontroler STM oraz moduł WiFi. Urządzenie powinno być wyposażone we własne źródło zasilania.

Praca wymagała także wykonania oprogramowania umożliwiającego zdalne nadzorowanie pracy czujnika oraz generowanie raportów w sytuacjach alarmowych poprzez prostą aplikację mobilną.

Urządzenie powinno mieć mały rozmiar, aby umożliwić dogodny montaż w pomieszczeniu o podwyższonym ryzyku zalania oraz posiadać obudowę zapewniającą bezpieczną pracę.

Słowa kluczowe

czujnik zalania, mikrokontroler STM, moduł WiFi, aplikacja mobilna, energooszczędność

Thesis title

Energy-saving STM-WiFi flood sensor with notification function

Abstract

The aim of the thesis was to design, build and commission a standalone and energy-efficient flooding sensor using an STM microcontroller and a WiFi module. The device should be equipped with its own power supply.

The work also required the development of software to remotely monitor sensor performance and generate reports in alarm situations via a simple mobile app.

The unit should be small in size to allow convenient installation in a room with a high risk of flooding and have an enclosure to ensure safe operation.

Key words

flood sensor, STM microcontroller, WiFi module, mobile app, energy efficiency

Spis treści

1 Wstęp **1**

1.1 Wprowadzenie . . . 1

1.2 Założenia projektowe . . . 2

1.2.1 Charakterystyka urządzenia . . . 2

1.2.2 Architektura systemu . . . 3

1.2.3 Oprogramowanie . . . 3

1.2.4 Uwzględnienie bibliotek zewnętrznych . . . 3

1.3 Podział pracy . . . 4

2 Analiza rynkowa **5**

2.1 Istniejące rozwiązania czujników zalania . . . 5

2.2 Możliwości zarządzania IoT . . . 6

2.3 Porównanie wybranych urządzeń . . . 6

2.4 Porównanie czujników zalania . . . 7

3 Infrastruktura Techniczna oraz Narzędzia Projektowe **9**

3.1 Zastosowane podzespoły . . . 9

3.2 Użyte urządzenia testowe . . . 14

3.3 Użyte oprogramowanie . . . 14

3.4 Użyte narzędzia konstrukcyjne . . . 17

3.5 Tabela kosztów . . . 18

4 Opis programu **19**

4.1 Program urządzenia . . . 19

4.1.1 wifi.c . . . 19

4.1.2 main.c . . . 26

4.2 Program aplikacji mobilnej . . . 31

4.3 Wygląd aplikacji mobilnej . . . 34

4.4 Schemat blokowy algorytmu urządzenia . . . 37

4.5 Optymalizacja oprogramowania . . . 39

5 Budowa i działanie **41**
5.1 Schemat blokowy 41
5.2 Schemat elektryczny 42
5.3 Schemat PCB 44
5.4 Instrukcja obsługi 46
5.5 Prototyp urządzenia 47
5.6 Urządzenie docelowe 48
5.6.1 Bez obudowy 48
5.6.2 Z obudową 49

6 Badania **51**
6.1 Badanie zasięgu sygnału WiFi 51
6.2 Badanie transferu danych modułu ESP-12E 53
6.3 Badanie bilansu energetycznego 54

7 Podsumowanie **57**

Bibliografia **61**

Spis skrótów i symboli **63**

Lista dodatkowych plików uzupełniających tekst pracy **65**

Spis rysunków **67**

Spis tabel **67**

Spis wzorów **68**

Rozdział 1

Wstęp

1.1 Wprowadzenie

Współczesny rozwój technologii bezprzewodowej, mikrokontrolerów oraz aplikacji mobilnych otwiera nowe perspektywy dla automatyzacji i zdalnego monitoringu różnorodnych procesów i zdarzeń. W perspektywie zarządzania i ochrony infrastruktury, a także dla zastosowań w inteligentnych systemach, jednym z kluczowych zadań jest detekcja i szybka reakcja na potencjalne zagrożenia, takie jak zalanie pomieszczeń. Dlatego rośnie zapotrzebowanie na rozwijanie efektywnych i autonomicznych rozwiązań, które umożliwiają wykrycie zalania oraz powiadomienie o nim użytkownika lub systemu zarządzania w czasie rzeczywistym.

Współczesne domy, biura oraz przemysł coraz częściej korzystają z inteligentnych systemów monitorujących, które pozwalają na automatyczną kontrolę i reakcję na różnego rodzaju sytuacje. Czujniki zalania stanowią istotny element takich systemów, umożliwiając szybką reakcję na wycieki wody, co może skutkować minimalizacją szkód materialnych oraz zagrożeń dla zdrowia i bezpieczeństwa pracowników bądź mieszkańców. Wspomniany rozwój technologiczny (przy równocześnie rosnącej świadomości ekologicznej), wprowadza popyt na energooszczędne urządzenia, które pozostają w pełni funkcjonalne przy minimalnym zużyciu energii elektrycznej.

Celem niniejszej pracy magisterskiej jest zaprojektowanie, konstrukcja oraz uruchomienie autonomicznego, energooszczędnego czujnika zalania, wykorzystującego mikrokontroler STM oraz moduł WiFi. Praca polega na opracowaniu dogodnego rozwiązania, które umożliwi zdalne monitorowanie stanu czujnika, a także generowanie alarmów w przypadku wykrycia zalania.

W ramach pracy magisterskiej zostaną przeprowadzone następujące czynności: badania, projektowanie oraz implementacja zarówno sprzętowa, jak i programowa - w celu osiągnięcia działającego prototypu czujnika zalania. Realizacja pracy będzie wymagała poruszenia zagadnień związanych z elementami elektroniki, programowania mikrokontro-

lerów, technologii bezprzewodowej oraz interfejsu użytkownika. Osiągnięcie celu pracy magisterskiej przyczyni się do poszerzenia wiedzy w obszarze automatyzacji oraz Internetu Rzeczy (IoT).

W kolejnych rozdziałach pracy zostaną omówione teoretyczne i praktyczne aspekty projektowania i implementacji autonomicznego czujnika zalania. Rozdziały te obejmą m.in. analizę istniejących rozwiązań, wybór odpowiednich komponentów, szczegółowy opis projektowanego układu, strategię oszczędzania energii, implementację oprogramowania oraz testy funkcjonalne. Celem końcowym jest realizacja działającego urządzenia, które spełni założone wymagania oraz pomoże w rozwoju inteligentnych systemów monitorujących.

1.2 Założenia projektowe

W tym podrozdziale zostaną przedstawione kluczowe założenia, na których polega projektowanie, konstrukcja oraz implementacja autonomicznego, energooszczędnego bezprzewodowego czujnika zalania. Cele te stanowią podstawę dla dalszych prac nad opracowaniem prototypu oraz realizacją postawionych wyzwań.

1.2.1 Charakterystyka urządzenia

Urządzenie będzie zdolne do samodzielnego wykrywania wycieków wody w monitorowanym obszarze. Podstawowe cechy projektowanego urządzenia obejmują:

- **Autonomiczność:** urządzenie będzie posiadać własne źródło zasilania, co pozwoli na pracę niezależnie od dostawcy zasilania zewnętrznego.
- **Energooszczędność:** projekt będzie kłaść duży nacisk na minimalizację zużycia energii w celu przedłużenia żywotności baterii i zwiększenia efektywności energetycznej.
- **Bezprzewodowa komunikacja:** urządzenie będzie wyposażone w moduł WiFi, umożliwiający bezprzewodową komunikację z innymi urządzeniami oraz zdalne generowanie raportów.
- **Mikrokontroler STM:** urządzenie będzie działało z wykorzystaniem mikrokontrolera STM, co pozwoli na implementację odpowiedniej logiki sterującej.
- **Prosta aplikacja mobilna:** do nadzorowania pracy urządzenia zostanie opracowana prosta aplikacja mobilna, umożliwiająca użytkownikowi zdalny monitoring pracy czujnika.

1.2.2 Architektura systemu

Architektura projektowanego systemu składać się będzie z pięciu głównych komponentów:

- **Czujnik zalania:** odpowiedzialny za wykrywanie wycieków wody w monitorowanym obszarze. Czujnik ten będzie generował sygnał alarmowy w przypadku wykrycia zalania.
- **Mikrokontroler STM:** stanowi centralny element systemu, integrujący działanie czujnika zalania oraz modułu WiFi. Będzie odpowiedzialny za zarządzanie urządzeniem, komunikację z aplikacją mobilną oraz generowanie alarmów.
- **Moduł WiFi:** umożliwia bezprzewodową komunikację z aplikacją mobilną oraz innymi urządzeniami dostępnymi w sieci.
- **Sekcja zasilania:** umożliwia zasilanie układu z sieci oraz zapewnia pracę z użyciem zapasowego źródła zasilania w przypadku przerwania działania zasilania głównego.
- **System alarmowy:** zapewnia dostępność lokalnego sygnału alarmowego w przypadku wykrycia zalania. Będzie to sygnał dźwiękowy generowany przez głośnik.

1.2.3 Oprogramowanie

Oprogramowanie będzie stanowić ważną część pracy, umożliwiającą monitorowanie działania urządzenia oraz obsługę sytuacji alarmowych. Główne funkcjonalności oprogramowania obejmują:

- monitorowanie stanu czujnika zalania oraz generowanie sygnałów alarmowych w przypadku wykrycia zalania,
- komunikację z innymi urządzeniami oraz aplikacją mobilną poprzez moduł WiFi,
- umożliwienie zdalnego monitorowania pracy urządzenia i dostęp do historii zdarzeń (alarmów).

1.2.4 Uwzględnienie bibliotek zewnętrznych

Zgodnie z założeniami będzie możliwe wykorzystanie dostępnych bibliotek zewnętrznych zgodnie z ich licencją. Wszystkie wykorzystane komponenty lub biblioteki będą opatrzone odpowiednimi odniesieniami w tekście pracy oraz uznaniem ich autorstwa.

1.3 Podział pracy

Praca została podzielona na 7 rozdziałów:

- Pierwszy rozdział stanowi wstęp tematyczny, określenie wymogów oraz przedstawienie celu, oraz zakresu prac.
- W drugim rozdziale przedstawiono analizę rynkową, czyli skonfrontowanie pomiędzy sobą rozwiązań rynkowych.
- Kolejny rozdział to przedstawienie bazy sprzętowej oraz środowisk systemowych, tj. zastosowanych komponentach, narzędziach programistycznych i testowych potrzebnych do realizacji pracy. Opisano tu najważniejsze funkcje służące wyborowi odpowiednich komponentów, programów i narzędzi.
- Czwarty rozdział zawiera opis działania programu urządzenia i aplikacji mobilnej. Przedstawia również wygląd samej aplikacji, prezentuje schemat blokowy algorytmu programu oraz opisuje proces optymalizacyjny.
- W piątym rozdziale zaprezentowano schematy, prototyp oraz rozwiązanie docelowe czujnika zalania. Dodatkowo przedstawiono tutaj sposób działania oraz instrukcję obsługi urządzenia wraz z aplikacją.
- Kolejny rozdział to opis badań przeprowadzonych z pomocą urządzenia docelowego. Rozdział zawiera także opis bilansu energetycznego i podsumowanie związane z energooszczędnością urządzenia oraz czasu pracy z zastosowaniem zasilania awaryjnego.
- Ostatni rozdział to podsumowanie związane z przebiegiem prac oraz przedstawienie ewentualnych wniosków, wynikających z uzyskanych rezultatów.

Podstawową literaturą wykorzystaną do realizacji pracy były następujące książki: 'Mikrokontrolery STM32 dla początkujących' [1] (dotycząca podstaw programowania mikrokontrolerów), 'Wydanie II języka C dla mikrokontrolerów AVR' [2] (dotycząca zagadnień przydatnych podzczas konfiguracji poszczególnych funkcjonalności takich jak np. przetworniki ADC) oraz 'Android Studio Podstawy tworzenia aplikacji' [3] (poruszająca temat wykonania swojej pierwszej aplikacji mobilnej).

Rozdział 2

Analiza rynkowa

W rozdziale opisano analizę dostępnych na rynku rozwiązań dotyczących czujników zalania wodą oraz ich zintegrowanych możliwości zarządzania (poprzez technologie Internetu Rzeczy (IoT)). Przedstawione zostaną wybrane produkty, ich charakterystyki oraz funkcje związane z detekcją wycieków wody oraz zdalnym zarządzaniem.

2.1 Istniejące rozwiązania czujników zalania

Na rynku dostępne są różnorodne rozwiązania służące do wykrywania wycieków wody i monitorowania sytuacji alarmowych. Przedstawione zostaną przykładowe produkty, w tym:

- **Sieciowe czujniki zalania:** sieciowe czujniki zalania czerpią energię elektryczną z gniazda sieciowego oraz są wyposażone w odpowiednio długą sondę pomiarową, która reaguje na kontakt z wodą. W momencie wykrycia wilgoci czujnik generuje lokalny sygnał alarmowy (np. dźwiękowy lub świetlny), informując również domową sieć IoT o wykryciu zagrożenia.
- **Bezprzewodowe czujniki podłogowe:** istnieje również rozwiązanie bezprzewodowe, które sprawdza się w przypadku, gdy użytkownik chce zapewnić ochronę infrastruktury lokalnej w miejscu trudno dostępnym, bez łatwego dostępu do sieciowego gniazda zasilającego.

Ta część pracy będzie dotyczyła budowy rozwiązania sieciowego czujnika zalania z możliwością pracy z użyciem zasilania awaryjnego. Ma to na celu zapewnienie możliwości powiadomienia użytkownika o utracie zasilania głównego.

2.2 Możliwości zarządzania IoT

Współczesne technologie IoT otwierają nowe możliwości zarządzania i monitorowania urządzeń zdalnie. Przedstawione zostaną cechy i funkcje związane z IoT w kontekście czujników zalania:

- **Zdalne sterowanie i konfiguracja:** rozwiązania kierujące się IoT umożliwiają zarządzanie czujnikami, takie jak zdalne aktywowanie i deaktywowanie alarmów, dostosowanie czułości czujnika i otrzymywanie powiadomień.
- **Integracja z aplikacjami mobilnymi:** czujniki wyposażone w IoT mogą być integrowane z dedykowanymi aplikacjami mobilnymi, umożliwiając użytkownikom monitorowanie stanu czujników, otrzymywanie powiadomień w czasie rzeczywistym oraz zarządzanie nimi.
- **Automatyczne generowanie raportów:** rozwiązania kierujące się IoT mogą zbierać dane z czujników zalania i automatycznie generować raporty dotyczące historii alarmów oraz zdarzeń.

2.3 Porównanie wybranych urządzeń

W tabeli nr 2.1 znajduje się porównanie kilku wybranych rozwiązań komercyjnych dostępnych na rynku. Przedstawiono tutaj m.in. funkcjonalności związane z zdalnym zarządzaniem poprzez IoT (w połączeniu z aplikacją mobilną), rodzaj zasilania czy cenę.

Tabela nr 2.1: Porównanie urządzeń rynkowych.

Rozwiązanie	Zdalne zarządzanie IoT	Integracja z aplikacją mobilną	Zasilanie	Cena
Eve [4]	Tak (Apple HomeKit)	Tak	Sieciowe	520 zł
Sonoff [5]	Tak (Kompatybilne)	Tak	Sieciowe	109 zł
Aqara Leak [6]	Tak (Home Kit)	Tak	Bateryjne	89 zł
Eurolook [7] TU-013-WLS	Tak (Tuya Smart)	Tak	Bateryjne	79 zł
ZigBee Tuya [8]	Tak (Tuya Smart)	Tak	Bateryjne	70 zł
Xiaomi [9]	Tak (Mi Smart)	Tak	Bateryjne	65 zł

Podsumowując, większość rozwiązań komercyjnych pracuje wyłącznie z wykorzystaniem zasilania bateryjnego, umożliwiającego ciągłą pracę w okresie ok. roku. Wszystkie z wymienionych urządzeń współpracują z domową siecią IoT oraz posiadają aplikację mobilną umożliwiającą zdalny nadzór lub zarządzanie pracą urządzenia w czasie rzeczywistym. Aplikacje te są w różnym stopniu zaawansowania (niektóre ograniczają się tylko do opcji włączania i wyłączania pomiaru).

2.4 Porównanie czujników zalania

W celu zaprojektowania systemu detekcji zalania wybór odpowiedniego czujnika ma kluczowe znaczenie dla skuteczności, niezawodności oraz dostosowania do konkretnych wymagań. Dla dokonania właściwego wyboru, warto dokładnie przeanalizować różne aspekty techniczne oferowane przez dostępne na rynku rozwiązania.

W tabeli nr 2.2 zestawiono ze sobą trzy różne czujniki, które umożliwiają realizację pracy. Każdy z nich oferuje unikalne cechy i funkcje: czujnik opadów deszczu YL-83 [10] oraz czujnik poziomu cieczy - Iduino SE045 [11] i Waveshare 9525 [12]. Głównym celem tego porównania jest ukazanie wyraźnych różnic w zakresie zasady wykrywania, zasilania, poboru prądu, wymiarów, możliwości kalibracji oraz integracji. To zestawienie ma także na celu przedstawić czym należy się kierować przy projektowaniu czujnika zalania.

Tabela nr 2.2: Porównanie dostępnych czujników zalania.

Cecha Techniczna	YL-83	Iduino SE045	Waveshare 9525
Sposób Wykrywania	Obecność wody - sygnał cyfrowy, Siła deszczu - sygnał analogowy	Poziom cieczy - sygnał analogowy	Poziom cieczy - sygnał analogowy
Zasilanie	5V DC	5V DC	2V-5V DC
Pobór Prądu	maks. 100 mA	20mA	20mA
Wymiary	54 x 40 mm	60 x 20 mm	19 x 63 mm
Możliwość Kalibracji	Tak	Tak	Tak
Integracja	Dla obszarów otwartych	Do projektów ogólnych	Do projektów ogólnych

Podsumowując informacje przedstawione w tabeli nr 2.2, głównymi różnicami mającymi wpływ na zastosowanie czujnika jest rozmiar komponentu oraz pobór prądu.

Wszystkie wymienione w tabeli urządzenia umożliwiają kalibrację. Czujniki Iduino oraz Waveshare można zastosować w obszarze otwartym oraz zamkniętym, natomiast czujnik YL-83 nadaje się tylko dla zastosowań w obszarze otwartym.

Biorąc pod uwagę sposób wykrywania zagrożenia, to czujnik YL-83 stanowi najlepsze rozwiązanie. Urządzenie to posiada wyprowadzenie cyfrowe określające w sposób jednoznaczny obecność wody w obszarze pomiaru. Pozostałe urządzenia umożliwiają wykrycie obecności zagrożenia przy jednoczesnym określeniu poziomu wody. Obecność dodatkowej funkcjonalności jest natomiast zbędna z perspektywy tematyki pracy, która ma polegać wyłącznie na wykrywaniu wystąpienia zalania. Na niekorzyść podzespołu YL-83 działa jednak fakt wysokiego poboru prądu w trakcie pracy.

Podsumowując, podczas prac związanych z budową czujnika zalania należy w pierwszej kolejności kierować się niskim poborem prądu oraz małymi rozmiarami. Może wystąpić tutaj możliwość kalibracji urządzenia, a integracja ma dotyczyć wyłącznie obszaru zamkniętego. Sposób wykrywania zagrożenia ma polegać na jednoznacznym określeniu wystąpienia cieczy w zasięgu pomiaru urządzenia.

Rozdział 3

Infrastruktura Techniczna oraz Narzędzia Projektowe

3.1 Zastosowane podzespoły

Biorąc pod uwagę analizę rynkową rozwiązań komercyjnych, dokonano wyboru podzespołów niezbędnych do realizacji pracy:

1. **STM NUCLEO-L432KC** [13]

   Dla projektu budowy czujnika zalania, wybór odpowiedniego mikrokontrolera stanowi bardzo ważny element, determinujący wydajność, niezawodność oraz elastyczność działania całego systemu. Mikrokontroler NUCLEO-L432KC, wyprodukowany przez firmę STMicroelectronics, wyróżnia się serią cech i funkcji, które sprawiają, że jest on odpowiednim wyborem:

   - **Wydajność w kompaktowej formie:** mikrokontroler NUCLEO-L432KC jest zrealizowany z użyciem rdzenia ARM Cortex-M4 o zegarze taktującym do 80 MHz. Zapewnia to wystarczającą moc obliczeniową do szybkiej i skutecznej reakcji na wykrycie zalania oraz przetwarzania danych związanych z czujnikiem. Dodatkowo kompaktowa forma płytki umożliwia łatwe zintegrowanie mikrokontrolera w obudowie czujnika.
   - **Niskie zużycie energii:** czujnik zalania musi działać wykorzystując zasilanie sieciowe. Mikrokontroler NUCLEO-L432KC charakteryzuje się niskim zużyciem energii, co pozwala na zmniejszenie kosztów związanych z poborem prądu. Zwiększa to także czas działania urządzenia z użyciem awaryjnego źródła zasilania.
   - **Wsparcie dla wielu interfejsów komunikacyjnych:** mikrokontroler ten oferuje różnorodne interfejsy komunikacyjne, takie jak I2C, SPI, UART, które

są istotne w przypadku czujnika zalania. Dzięki tym interfejsom mikrokontroler może łatwo komunikować się z innymi urządzeniami, przekazując informacje o wykryciu zalania lub odbierając polecenia związane z zarządzaniem czujnikiem.

- **Duża ilość dostępnych wyprowadzeń:** projekt czujnika zalania może wymagać podłączenia różnych elementów, takich jak czujniki wilgotności, styki wody czy diody LED. Mikrokontroler NUCLEO-L432KC posiada wystarczającą ilość styków GPIO (general-purpose input/output), co umożliwia swobodne podłączenie wymaganych komponentów.
- **Wsparcie narzędziowe:** STMicroelectronics zapewnia obszerną dokumentację, zestaw narzędzi programistycznych i bibliotek, co ułatwia rozwój oprogramowania na bazie mikrokontrolera. To z kolei przyspiesza proces implementacji i testowania kodu, umożliwiając skoncentrowanie się na samym projekcie czujnika zalania.
- **Wsparcie społeczności:** dzięki wspierającej społeczności i dostępności źródeł oraz projektów z otwartym dostępem do kodu, można skorzystać z gotowych rozwiązań i pomysłów. Przyspiesza to rozwój i poprawę funkcjonalności czujnika zalania.

Mikrokontroler NUCLEO-L432KC wydaje się być dobrze dostosowany do wymagań projektu budowy czujnika zalania. Jego wydajność, oszczędność energii, wsparcie dla różnorodnych interfejsów, dostępność wielu wyprowadzeń GPIO oraz dostępne narzędzia programistyczne budują solidną podstawę dla niezawodnego i efektywnego systemu detekcji zalania. Mikrokontroler ten cechuje się także relatywnie niską ceną w porównaniu z resztą dostępnych na rynku rozwiązań, co czyni go bardziej atrakcyjnym w kontekście rozwiązania.

2. **ESP-12E** [14]

   Dla budowy autonomicznego, energooszczędnego, bezprzewodowego czujnika zalania, wybór odpowiedniego modułu WiFi ma równie duże znaczenie dla zapewnienia niezawodnej komunikacji, efektywnego zarządzania oraz interakcji z użytkownikiem poprzez prostą aplikację mobilną. Moduł WiFi ESP-12E, produkowany przez Espressif Systems, jest odpowiednim wyborem ze względu na swoje charakterystyki i funkcjonalności:

   - **Wsparcie dla Internetu Rzeczy (IoT):** moduł ESP-12E został specjalnie zaprojektowany z myślą o aplikacjach związanych z Internetem Rzeczy. Jego zdolność do bezprzewodowej komunikacji za pośrednictwem WiFi umożliwia łatwe wdrożenie urządzenia w sieci i zdalne zarządzanie nim. Jest to kluczowe w kontekście autonomiczności oraz zdalnego raportowania zdarzeń.

- **Niska cena i dostępność:** moduł ESP-12E jest dostępny w przystępnej cenie, co jest istotne z punktu widzenia budżetu. Jego popularność i szeroka dostępność na rynku oznaczają, że można go łatwo zdobyć oraz znaleźć wsparcie w postaci dokumentacji, poradników i projektów społecznościowych.
- **Niskie zużycie energii:** energooszczędność była jednym z głównych celów pracy. Moduł ESP-12E charakteryzuje się relatywnie niskim zużyciem energii w trybie niskiego poboru mocy (deep sleep), co pozwala na efektywne zarządzanie baterią lub innym źródłem zasilania.
- **Prosta integracja z mikrokontrolerem STM:** projekt wymaga wykorzystania dowolnego mikrokontrolera STM. Moduł ESP-12E łatwo integruje się z mikrokontrolerem poprzez interfejsy UART lub SPI. To umożliwia płynną komunikację między czujnikiem zalania, a mikrokontrolerem.
- **Możliwości komunikacyjne:** moduł ESP-12E nie tylko obsługuje standardy WiFi, ale także umożliwia komunikację za pomocą interfejsów UART, SPI i I2C. To otwiera drzwi do komunikacji z innymi urządzeniami lub czujnikami, które mogą być wykorzystane w projekcie czujnika zalania.
- **Wsparcie dla oprogramowania:** Espressif Systems oferuje bogatą dokumentację, narzędzia programistyczne oraz środowiska, które ułatwiają implementację oprogramowania na bazie modułu ESP-12E. Dzięki temu programowanie i rozwijanie funkcji związanych z komunikacją i zarządzaniem urządzenia jest znacznie ułatwione.

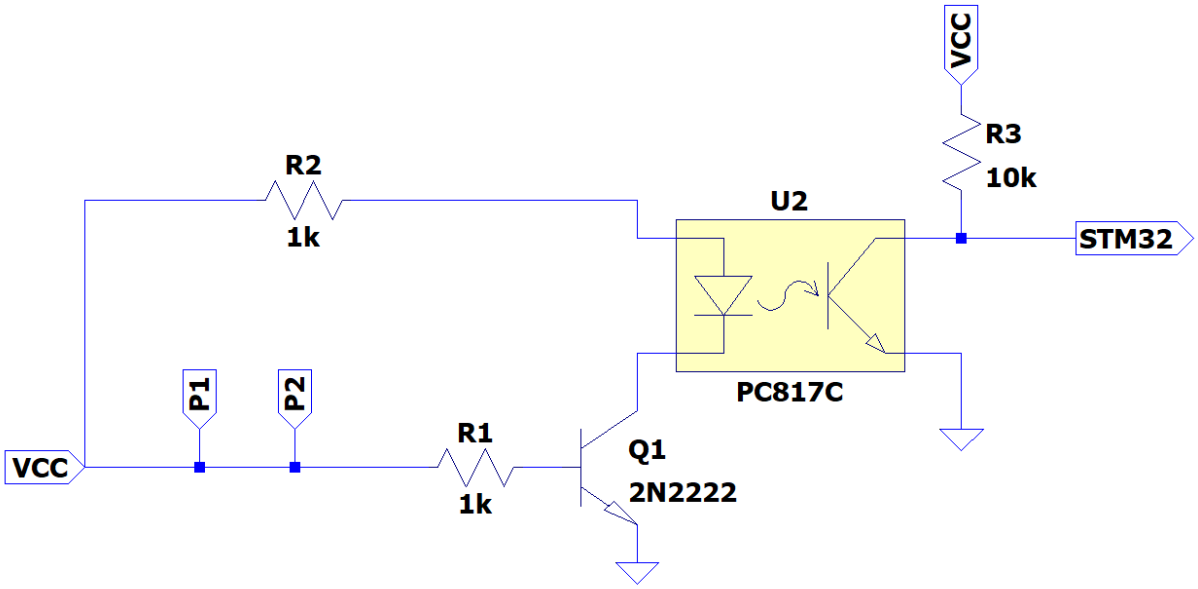
Moduł WiFi ESP-12E jest idealnie dopasowany do wymagań projektu czujnika zalania. Jego zdolność do bezprzewodowej komunikacji, niskie zużycie energii, obszerne możliwości komunikacyjne, dostępność narzędzi programistycznych i wsparcia dokumentacyjnego budują solidną podstawę dla wdrożenia i efektywnego funkcjonowania urządzenia.

Dodatkowym atutem jest fakt, że dla modułu WiFi dostępny jest specjalny adapter. Zapewnia on dodatkową stabilność zasilania oraz ułatwia dostęp do wyprowadzeń modułu - co jest równoznaczne z uproszczeniem montażu.

3. **Czujnik zalania**

   Podczas prac związanych z budową własnego czujnika zalania, posłużono się sondą pomiarową dostępną dla urządzenia YL-83, przedstawionego w tabeli nr 2.2. Sonda wykorzystana została jako zakończenie wyjść pomiarowych czujnika zwiększając tym samym powierzchnię pomiaru.

Schemat ideowy czujnika zaprojektowanego z użyciem programu LTSpice [15] przedstawiono na rysunku nr 3.1.



Rysunek nr 3.1: Schemat ideowy czujnika

Czujnik odznacza się małymi rozmiarami oraz niskim poborem prądu (5.3mA w trybie pracy oraz 450pA w trybie czuwania).

Do budowy czujnika wykorzystano następujące elementy elektroniczne:

- **PC817C:** [16] układ transoptora wykorzystywany do izolacji elektrycznej między dwoma obwodami. Składa się z diody emiterowej i fototranzystora umieszczonych w jednej obudowie. Właściwości izolacyjne transoptora pozwalają na przekazywanie sygnału świetlnego przez barierę izolacyjną, co umożliwia sterowanie obwodem wyjściowym na podstawie sygnału wejściowego. Zastosowanie transoptora pozwoliło na realizację połączenia galwanicznego pomiędzy mikrokontrolerem STM, a wyjściem z czujnika.
- **2N2222:** [17] tranzystor bipolarny NPN 2N2222 to popularny element elektroniczny, który znajduje zastosowanie w szerokim zakresie projektów elektronicznych. Tranzystor ten należy do rodziny tranzystorów bipolarnych i jest często wykorzystywany jako przełącznik lub wzmacniacz w układach analogowych oraz cyfrowych.
- **Rezystory 1kΩ**: rezystory węglowe o tolerancji ilościowej ±5%.

Układ jest zasilany za pomocą źródła 3.3V. Elementem sprawczym czujnika są dwa wyprowadzenia: P1 oraz P2.
W przypadku, gdy pomiędzy tymi wyprowadzeniami dojdzie do zwarcia (tak jak na rysunku nr 3.1), tranzystor Q1 zacznie przewodzić. Spowoduje to uaktywnienie

diody transoptora U2. W ten sposób na wejściu mikrokontrolera STM pojawi się stan niski, który wyzwoli sygnał alarmowy. Wyprowadzenia P1 oraz P2 przeznaczone zostaną do aplikacji sondy pomiarowej, należącej do czujnika YL-83.

4. **TP4056** [18]

   W szeregu komponentów, które wpłyną na funkcjonalność i wydajność urządzenia, szczególnie istotnym aspektem jest odpowiedni wybór ładowarki akumulatora. W tym kontekście ładowarka TP4056 odznacza się jako opcja uzasadniona i adekwatna.

   W zaprojektowanym urządzeniu zapasowym źródłem zasilania będzie akumulator Li-Pol Akyga 1200mAh [19], który spełnia wymogi trwałości i niezawodności działania. Jest to istotne dla wymogu autonomiczności urządzenia, dlatego wybór ładowarki TP4056 nabiera jeszcze większego sensu. Element ten zapewnia parę przydatnych funkcjonalności:

   - **Jednoczesne ładowanie i zabezpieczenie przed przeładowaniem:** ładowarka TP4056 jest zaprojektowana tak, aby umożliwić jednoczesne ładowanie akumulatora oraz kontrolowanie tego procesu w celu uniknięcia przeładowania. Ta właściwość jest szczególnie istotna w projekcie, gdzie autonomiczność oraz długa żywotność baterii są krytycznymi założeniami.
   - **Prosta implementacja:** ładowarka TP4056 jest dostępna w formie niewielkiego układu scalonego, co ułatwia integrację z projektem. Posiada także niewielką liczbę wymaganych zewnętrznych komponentów, co przyspiesza proces konstrukcji.

   Warto zwrócić uwagę na kolejny istotny element, który został umieszczony tuż za wyjściem z ładowarki - jest nim stabilizator napięcia LF33CV [20]. Funkcją tego komponentu będzie zapewnienie stabilizacji napięcia wyjściowego z ładowarki na poziomie 3.3V. Dzięki temu możliwe będzie zasilenie różnych układów z uwzględnieniem tych wyjątkowo wrażliwych na zmiany napięcia zasilającego. Stabilizacja ma istotne znaczenie dla zapewnienia niezawodnej pracy oraz precyzyjnych parametrów działania całego urządzenia.

   Warto zaznaczyć, że zastosowanie stabilizatora napięcia przyniesie również korzyści związane z uproszczeniem konstrukcji. Brak takiego stabilizatora powoduje konieczne skorzystanie z dzielników napięcia w celu obniżenia wartości napięcia wejść zasilających. Wykorzystanie dedykowanego stabilizatora eliminuje tę potrzebę, co przyczynia się do efektywniejszego projektowania i redukcji potencjalnych źródeł błędów czy zakłóceń w układzie.

5. **Głośnik z generatorem FY248** [21]

   W celu zastosowania lokalnego systemu alarmowania użytkownika o wystąpieniu zalania użyto głośnika z generatorem FY248. Umożliwia to zastosowanie mechanizmu PWM, który zmniejszy zużycie energii przez głośnik w trakcie jego pracy.

6. **Przełącznik bistabilny ON/OFF** [22]

   Przełącznik bistabilny posłużył jako wyłącznik urządzenia. W momencie, gdy użytkownik postanowi wyłączyć urządzenie, należy użyć przełącznika w celu odcięcia akumulatora zapasowego.

3.2 Użyte urządzenia testowe

W procesie testowania oraz badań nad projektem skorzystano również z modułu konwertera UART-USB CH343 [23]. W użyciu z programem RealTerm posłużył jako element umożliwiający przetestowanie komunikacji pomiędzy mikrokontrolerem, a modułem łączności bezprzewodowej.

Komunikacja ta odbyła się za pomocą symulacji, gdzie mikrokontroler zastąpiono stacją roboczą PC. Umożliwiło to ręczne wysyłanie komunikatów do podzespołu ESP-12E z równoczesną obserwacją efektów.

3.3 Użyte oprogramowanie

W procesie realizacji autonomicznego czujnika zalania, wykorzystanie odpowiedniego oprogramowania odgrywa znaczną rolę w osiągnięciu celów oraz efektywnej implementacji funkcjonalności. W ramach tego podrozdziału zostaną przedstawione użyte narzędzia programistyczne oraz środowiska, które odegrały rolę w implementacji zarówno oprogramowania mikrokontrolera STM, jak i aplikacji mobilnej:

1. **CubeIDE i CubeMX** [24],[25]

   Zastosowanie mikrokontrolera STM jako głównego komponentu procesorowego w projekcie skłoniło do wykorzystania narzędzi programistycznych, które umożliwiają efektywną pracę oraz optymalne wykorzystanie możliwości tej platformy. W związku z tym zdecydowano się na użycie CubeIDE oraz CubeMX:

   - **CubeIDE** to zintegrowane środowisko programistyczne dedykowane dla mikrokontrolerów STM, które zapewnia zaawansowane narzędzia do projektowania, programowania oraz tzw. 'debugowania'. Dzięki interfejsowi graficznemu oraz możliwości generowania kodu dla różnych rodzajów aplikacji, CubeIDE dostarcza efektywnego środowiska dla programowania mikrokontrolera STM.

- **CubeMX** to narzędzie umożliwiające konfigurację parametrów mikrokontrolera, generowanie kodu źródłowego oraz inicjalizację peryferii. Ułatwia ono pracę poprzez automatyczne generowanie kodu inicjalizującego, co znacznie przyspiesza proces budowy oprogramowania.

W pracy postanowiono skorzystać z języka programowania C [26]. Wybór ten wynikał z faktu, że język C jest powszechnie stosowany w programowaniu mikrokontrolerów i mikroprocesorów, oferując efektywną kontrolę nad sprzętem oraz możliwość optymalizacji kodu. Ponadto, ogromny zasób bibliotek, poradników i gotowych rozwiązań dostępnych w języku C, szczególnie w kontekście platformy STM, przyczynia się do skrócenia czasu implementacji oraz zapewnienia niezbędnych funkcji.

2. **Android Studio** [27]

   W celu umożliwienia zdalnej interakcji z autonomicznym czujnikiem zalania, postanowiono opracować dedykowaną aplikację mobilną, która umożliwi użytkownikowi monitorowanie oraz odbieranie powiadomień związanych z działaniem urządzenia. W tym celu zaadoptowano Android Studio, uznawane za główne środowisko programistyczne służące do budowy aplikacji na platformę systemową Android. Zamiast tradycyjnego języka Java [28], postawiono na język Kotlin [29], który zyskał popularność w środowisku programistycznym Androida jako alternatywa dla Javy. Decyzja o wykorzystaniu Kotlin w projekcie była podyktowana kilkoma czynnikami:

   - **Zwięzłość:** Język Kotlin oferuje bardziej zwięzłą i klarowną składnię w porównaniu do Javy, co przekłada się na krótszy i bardziej czytelny kod. Dzięki temu praca nad implementacją aplikacji mobilnej staje się bardziej efektywna i zwiększa czytelność kodu.
   - **Interoperacyjność z Javą:** Język Kotlin jest w pełni interoperacyjny z Javą, co oznacza, że istniejące moduły napisane w Javie mogą być łatwo integrowane z kodem Kotlin. To pozwoliło na wykorzystanie istniejących bibliotek oraz narzędzi z ekosystemu Androida.
   - **Nowoczesność:** Kotlin wprowadza nowoczesne koncepcje programowania, takie jak właściwości (properties), czy rozszerzenia funkcji (extension functions), które przyspieszają proces budowy aplikacji i pozwalają na bardziej modułowy i zwięzły kod programu.

3. **Fusion360 i EAGLE** [30],[31]

   W ramach procesu projektowania schematów ideowych oraz płytek PCB, skorzystano z zaawansowanego oprogramowania Fusion 360, które posiada wbudowany program EAGLE. Ta kombinacja dostępna jest w ramach darmowej licencji edukacyjnej, co pozwala na wykorzystanie profesjonalnych narzędzi bez konieczności ponoszenia kosztów licencyjnych.

   Program Fusion 360 zintegrowany z programem EAGLE dostarczył szeroki zakres narzędzi umożliwiających kompleksową pracę nad projektem. Pozwolił na swobodną konstrukcję schematów ideowych, gdzie każdy element mógł zostać starannie umieszczony i połączony. Ponadto, program umożliwił zaprojektowanie płytki PCB, pozwalającej na zbudowanie urządzenia docelowego.

4. **RealTerm** [32]

   Program RealTerm pełni funkcję terminala szeregowego, czyli aplikacji służącej do komunikacji z urządzeniami poprzez port szeregowy. Jest to przydatne narzędzie w wielu dziedzinach, takich jak np. programowanie mikrokontrolerów, czy diagnostyka i analiza sprzętu. Aplikacja ta umożliwiła m.in. na podgląd komunikatów zwrotnych przychodzących z modułu WiFi.

5. **ThingSpeak** [33]

   W zakresie opracowywanego autonomicznego czujnika zalania, nasuwa się potrzeba gromadzenia oraz analizy danych związanych z wykrywaniem wycieków wody oraz obecnym stanem urządzenia. W odpowiedzi na tę potrzebę zdecydowano się na wykorzystanie platformy ThingSpeak. Względem realizowanej pracy, baza danych ThingSpeak odgrywa podwójną rolę jako baza danych oraz narzędzie analityczne. Pozwala to na efektywne gromadzenie, wizualizację oraz analizę danych zgromadzonych z czujnika zalania.

   Strona ThingSpeak stanowi rozwiązanie dedykowane Internetowi Rzeczy (IoT), które umożliwia agregację, przechowywanie oraz analizę danych pochodzących z różnych źródeł, w tym wypadku z czujnika zalania. Wykorzystanie tej platformy umożliwia centralizację danych oraz zapewnia możliwość przeprowadzenia dokładnej analizy zachowań, trendów oraz ewentualnych niestandardowych zdarzeń.

   Istotnym aspektem wyboru bazy danych ThingSpeak jest dostępność darmowej licencji, która oferuje atrakcyjne możliwości. Ta darmowa wersja umożliwia ładowanie na serwer aż trzech milionów próbek danych rocznie w interwałach wynoszących co najmniej 15 sekund. W praktyce oznacza to, że przepustowość darmowego łącza wynosi cztery próbki na minutę. To istotne, zwłaszcza w kontekście autonomicznego czujnika zalania, gdzie rejestrowanie danych w interwałach czasowych jest niezbędne dla monitorowania stanu oraz ewentualnych wycieków.

Możliwe jest zakupienie pakietu subskrypcji bazy danych ThingSpeak zwiększając tym samym maksymalną liczbę dostępnych próbek do 33 milionów rocznie oraz zwiększając przepustowość łącza do 60 próbek na minutę. Oznacza to transfer jednej próbki na sekundę. Koszt subskrypcji różni się w zależności od wybranej licencji:

- FREE: darmowa licencja ThingSpeak przeznaczona do użytku prywatnego (niekomercyjnego). W ramach tej licencji występuje m.in. ograniczenie przepustowości łącza, zmniejszenie maksymalnej ilości kanałów, itd.
- Student (492zł/rok): studencka licencja ThingSpeak jest przeznaczona do użytku prywatnego (niekomercyjnego).
- Home (670zł/rok): domowa licencja ThingSpeak jest przeznaczona do użytku prywatnego (niekomercyjnego).
- Academic (2370zł/rok): licencja akademicka ThingSpeak jest przeznaczona do badań akademickich i nauczania.
- Standard (3800zł/rok): standardowa licencja ThingSpeak jest przeznaczona do użytku komercyjnego.

W celu realizacji pracy magisterskiej postanowiono skorzystać z darmowej licencji bazy danych ThingSpeak.

3.4 Użyte narzędzia konstrukcyjne

W trakcie realizacji pracy magisterskiej, w celu konstrukcji precyzyjnego czujnika zalania, skorzystano z różnorodnych narzędzi. Wśród nich ważną rolę odegrała stacja lutownicza Zhaoxin 936D. To urządzenie, charakteryzujące się kontrolą temperatury, umożliwiło zlutowanie elementów elektronicznych. Dzięki jego funkcjom zapewniono trwałość i dokładność połączeń pomiędzy podzespołami.

W trakcie montażu elementów oraz podczas konstrukcji mechanicznych połączeń wykorzystano kombinerki oraz obcęgi.

W procesie testowania oraz kalibracji urządzenia wykorzystano uniwersalny multimetr DPM DT9205A [34]. Dzięki jego funkcjom pomiarowym możliwy był dokładny monitoring parametrów elektrycznych oraz przeprowadzenie niezbędnych pomiarów.

W trakcie wszystkich faz projektowania oraz montażu, niezastąpiona okazała się suwmiarka. To narzędzie pomiarowe pozwoliło na precyzyjne określenie wymiarów oraz dopasowanie elementów, zapewniając jednocześnie dokładność konstrukcji.

Do wykonania elementów mocujących podzespoły w wodoszczelnej obudowie wykorzystano drukarkę 3D. Umożliwiła ona precyzyjne wykonanie elementów oraz wdrożenie niezbędnych korekt w konstrukcji obudowy.

3.5 Tabela kosztów

Na podstawie wskazanych wcześniej elementów, podzespołów i narzędzi, sporządzono tabelę podsumowującą koszty związane z pracą. Koszty te przedstawiono w tabeli nr 3.1.

Tabela nr 3.1: Tabela kosztów

Element	Cena
STM NUCLEO-L432KC	64.50zł
ESP-12E	29.80zł
Adapter ESP-12E	5.80zł
TP4056	6.30zł
Akyga Li-Po	18.90zł
PC817	0.82zł
2N2222	4.90zł
FY248	4.30zł
YL-83	9.90zł
LF33CV	8.63zł
CH343	17.90zł
Dwustronna płytka miedzi	6zł
Przełącznik bistabilny ON/OFF	2.50zł
Gniazda żeńskie 1x10, 1x2, 1x16	10zł
Obudowa	34zł
Suma:	224.25zł

Jak widać na liście zakupowej znalazło się sporo pozycji. Koszt pracy magisterskiej wyniósł ostatecznie 224.25zł.

W celu zmniejszenia kosztów całkowitych pracy można było zrezygnować z użycia sondy pomiarowej czujnika YL-83 i zamiast niej pozostawić dwa wyprowadzenia pomiarowe (rozwiązanie wielu urządzeń komercyjnych). Zmniejszyłoby to jednak powierzchnię pomiarową urządzenia, co miałoby bezpośredni wpływ na zmniejszenie prawdopodobieństwa wykrycia zagrożenia. Ponadto, zamiast użycia mikrokontrolera można pokusić się o użycie samego mikroprocesora. Można tym sposobem oszczędzić na rozmiarach urządzenia oraz koszcie końcowym, ale skomplikowałoby to proces konstrukcji urządzenia.

Rozdział 4

Opis programu

W tym rozdziale skupiono się na opisie programów napisanych w celu zarządzania zarówno urządzeniem docelowym, jak i aplikacją mobilną.

4.1 Program urządzenia

4.1.1 wifi.c

Podstawą działania modułu WiFi-12E jest zestaw instrukcji AT (AT commands), które pozwalają na interakcję z modułem poprzez port szeregowy UART (Universal Asynchronous Receiver-Transmitter). Komendy AT to standardowy sposób komunikacji pomiędzy hostem (np. mikrokontrolerem), a modułem przez wysyłanie i odbieranie instrukcji tekstowych. W praktyce oznacza to, że za pomocą zdefiniowanych komend można kontrolować zachowanie modułu WiFi-12E, np. nawiązywać połączenia sieciowe, przesyłać i odbierać dane oraz zarządzać parametrami sieciowymi.

Na podstawie dostępnych komend AT napisano autorską bibliotekę (wifi.c) kontrolującą pracę modułu WiFi wykorzystując do tego interfejs komunikacyjny UART, łączący moduł WiFi i mikrokontroler STM.

Początek pliku otwiera definicja bibliotek oraz zmiennych pomocniczych pokazana na rysunku nr 4.1.

```
#include "tim.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <wifi.h>

// WiFi module UART buffer
#define WIFI_BUFFER_SIZE 512
uint8_t wifi_buffer[WIFI_BUFFER_SIZE];

// Bool flags
bool OK_init_status = false;     // Init status flag
bool OK_connected = false;       // Connect to network flag
bool OK_disconnected = false;    // Disconnect from network flag
bool OK_got_rtc_data = false;    // Get RTC data from server flag
bool OK_wifi_works = false;      // WiFi works fine flag
bool TS_flag = false;            // ThingSpeak buffer check flag

volatile char* RTC_temp = NULL; // Helps hold RTC string fragment
```

Rysunek nr 4.1: Zdefiniowanie bibliotek i zmiennych pomocniczych

Wśród bibliotek można zauważyć m.in. 'tim.h' oraz <stdbool.h>. Pierwsza zawiera realizację zegara TIM1 nadzorującego wypełnienie i okres dla sygnału dźwiękowego, pochodzącego z głośnika FY248. Druga biblioteka odpowiada za możliwość użycia w projekcie zmiennych typu bool.

Poniżej można znaleźć inicjalizację rozmiaru bufora (512 bajtów), który będzie używany do przechowywania danych otrzymanych poprzez interfejs UART od modułu WiFi.

Następnym elementem jest deklaracja flag sygnalizujących pewne stany podczas działania programu. Na przykład flaga `OK_init_status` może oznaczać, że etap inicjalizacji przebiegł pomyślnie, a `OK_connected_to_network` czy udało się pomyślnie połączyć z siecią.

Na końcu znajduje się zmienna wskaźnikowa `RTC_temp` typu char oznaczona jako volatile, co oznacza, że jej wartość może zmieniać się niezależnie od wykonywanego programu. Wskazuje ona na fragment ciągu znaków związanego z RTC (Real-Time Clock), który będzie odbierany przez moduł WiFi.

Kolejnym fragmentem są dwie podstawowe funkcje przedstawione na rysunku nr 4.2. Określono je jako podstawowe, gdyż są one podstawą działania każdej innej funkcji zrealizowanej w dalszej części pliku.

```
// ==================================================================
// ------------------------ Basic functions ------------------------
// ==================================================================

// Clear buffer
void wifi_clear_buffer(void)
{
    HAL_UART_AbortReceive(&huart2);
    memset(wifi_buffer, 0, sizeof(wifi_buffer));
    HAL_UART_Receive_IT(&huart2, wifi_buffer, sizeof(wifi_buffer));
}

// Send a command
void wifi_send_command(const char* command)
{
    HAL_UART_Transmit(&huart2, (uint8_t*)command, strlen(command),
                                                    HAL_MAX_DELAY);
}
```

Rysunek nr 4.2: Funkcje podstawowe

Pierwsza z nich odpowiada za zatrzymanie odbierania przez mikrokontroler STM informacji na interfejsie UART, wyczyszczenie bufora podręcznego `wifi_buffer` i wznowienie odbioru. Druga odpowiada za wysłanie danych przez interfejs UART do modułu WiFi.

Kolejnym elementem jest inicjalizacja modułu WiFi.

```
// ==================================================================
// --------------------------- Initial ----------------------------
// ==================================================================
void wifi_init(void)
```

Rysunek nr 4.3: Inicjalizacja modułu WiFi

Funckja `wifi_init()` odpowiada za wysłanie czterech poleceń AT, tj.: `AT`, `AT+RST`, `AT+CWQAP` oraz `AT+CWMODE=1`. Poniżej opisano właściwości przedstawionych poleceń:

- **AT:** to podstawowa komenda, która inicjuje komunikację z modułem. Po wysłaniu tej komendy moduł odpowiada, zazwyczaj wysyłając 'OK'. Umożliwia to potwierdzenie, że komunikacja pomiędzy urządzeniem nadzorującym, a modułem przebiega prawidłowo.

- **AT+RST:** ta komenda służy do zresetowania modułu ESP-12E. Po jej wysłaniu moduł zostanie zresetowany i ponownie uruchomiony. Często używane jest to po

zmianie konfiguracji lub w przypadku wystąpienia problemów z działaniem samego modułu.

- **AT+CWQAP:** ta komenda jest używana do rozłączania się z aktualnym połączeniem Wi-Fi. Po jej wysłaniu moduł przestaje być połączony z punktem dostępu (Access Point, AP), do którego był podłączony. Komenda ta jest przydatna w momencie, gdy zamiarem jest zmiana połączenia Wi-Fi lub zrezygnowanie z połączenia.

- **AT+CWMODE=1:** to polecenie jest używane do ustawiania trybu pracy modułu Wi-Fi. Wartość '1' oznacza tryb klienta, czyli moduł będzie działał jako klient Wi-Fi, łącząc się z dostępnymi sieciami. Istnieją dwa dodatkowe tryby pracy: tryb punktu dostępowego (AT+CWMODE=2) oraz tryb punktu dostępowego i klienta (AT+CWMODE=3).

Po zakończeniu inicjalizacji wyzwalany jest krótki sygnał dźwiękowy sygnalizujący bezbłędne zakończenie tego fragmentu programu.

Kolejną funkcją zrealizowaną wewnątrz pliku wifi.c jest funkcja odpowiedzialna za skanowanie i zapisywanie w tabeli dostępnych w zasięgu modułu sieci WiFi. Każda z wyznaczonych dostępnych w zasięgu sieci posiada dodatkową informację o wartości mocy sygnału RSSI, co pozwala określić jakość sygnału. Kod przedstawiono na rysunku nr 4.4.

```
// ================================================================
// ---------------------- Scanning Networks ----------------------
// ================================================================

// MAX Networks number [rows number]
#define MAX_NETWORKS 10

// Networks tab struct
typedef struct {..

// Function variables
WiFiNetwork wifi_networks[MAX_NETWORKS];
uint8_t num_networks = 0;

// Scan Networks function + create table
void wifi_scan_networks(void)..
```

Rysunek nr 4.4: Skanowanie sieci

Należy tutaj wspomnieć, że funkcja ta została wykorzystana wyłącznie do badań mających na celu określenie czułości modułu. Wyniki badań przedstawione zostały w rozdziale pt. "Badania".

Kolejnymi elementami pliku są dwie funkcje odpowiedzialne za połączenie się modułu z wybraną siecią WiFi (podając przy tym SSID oraz hasło dostępu do wybranej sieci) oraz rozłączenie z aktualnie używaną siecią. Przedstawione one zostały na rysunku nr 4.5.

```
// ==================================================================
// -------------------- Connecting to Network --------------------
// ==================================================================

// Connect to WiFi Specified Network
void wifi_connect_to_network(const char* ssid, const char* password)

// ==================================================================
// ----------------- Disconnecting from Network ------------------
// ==================================================================

// Disconnect from current AP
void wifi_disconnect_from_network(void)
```

Rysunek nr 4.5: Funkcje łączenia i rozłączenia z siecią WiFi

Działanie funkcji `wifi_connect_to_network()` polega na użyciu polecenia `AT+CWJAP`. Po wysłaniu tego polecenia moduł będzie próbował nawiązać połączenie z podaną siecią Wi-Fi. Jeśli uda się połączyć, modułowi zostanie przydzielony adres IP, co skutkować będzie możliwością komunikowania się z innymi urządzeniami w obrębie sieci. Jeśli połączenie z siecią nie powiedzie się, moduł może próbować ponownie zainicjalizować połączenie lub podjąć inne akcje zgodnie z konfiguracją. Jednym z etapów pełnej inicjalizacji modułu WiFi jest połączenie z wybraną siecią. Po pomyślnym połączeniu się, wyzwalane są dwa krótkie sygnały dźwiękowe.

Użycie elementu `wifi_disconnect_from_network()` polega na użyciu wspomnianego wcześniej polecenia `AT+CWQAP`, które spowoduje rezygnację z aktualnie nawiązanego połączenia.

Głównym elementem pliku wifi.c i podstawą agregacji danych używanych później przez aplikację mobilną jest funkcja przedstawiona na rysunku nr 4.6.

```
// ==================================================================
// ------------------------- ThingSpeak -------------------------
// ==================================================================

// Connection with ThingSpeak
void wifi_thingspeak_send_data(const char* API_KEY, uint8_t value)
```

Rysunek nr 4.6: Inicjalizacja połączenia z bazą danych ThingSpeak

Funkcja `wifi_thingspeak_send_data()` służy do nawiązania tymczasowego połączenia TCP z bazą danych ThingSpeak w celu przesyłania danych pomiarowych. Na początku przyjmuje ona dwa argumenty: `API_KEY` jako wskaźnik do klucza API platformy ThingSpeak oraz `value` jako wartość pomiaru, która ma zostać przesłana. Funkcja rozpoczyna nawiązywanie połączenia TCP z serwerem ThingSpeak. Warto zauważyć, że po nawiązaniu połączenia, moduł utrzymuje je przez pewien okres, jednak aby je utrzymać, konieczne jest cykliczne przesyłanie danych w określonym przedziale czasowym.

Następnie dane do przesłania są przygotowywane poprzez zbudowanie odpowiedniej komendy GET zawierającej klucz API oraz wartość pomiaru. Warto podkreślić, że platforma ThingSpeak umożliwia przesyłanie danych w formie pól, a maksymalna liczba pól w jednym przesłaniu to 8, przy czym każde pole może zawierać maksymalnie 255 znaków.

Po przygotowaniu danych funkcja inicjuje przesyłanie poprzez wysłanie komendy `AT+CIPSEND`, która określa długość danych, jakie zostaną przesłane. Następnie przesyła właściwe dane i oczekuje krótką chwilę na potwierdzenie. Po zakończeniu przesyłania funkcja czyści bufor komunikacyjny `wifi_buffer`.

Warto także nadmienić fakt, że istnieje również polecenie AT zamykające połączenie TCP: `AT+CIPCLOSE`. Nie jest ono jednak wymagane, gdyż po wysłaniu danych do bazy ThingSpeak, połączenie TCP zostaje automatycznie zakończone. Z powodu wykorzystania darmowej subskrypcji ThingSpeak, możliwość nawiązania kolejnego połączenia pojawia się dopiero po 15 sekundach od czasu zakończenia ostatniego.

Przedostatnim elementem pliku jest fragment odpowiedzialny za wprowadzanie modułu WiFi w tryb uśpienia oraz wybudzanie go. Uśpienie oraz wybudzenie zrealizowano w dwóch osobnych funkcjach przedstawionych na rysunku nr 4.7.

```
// ==================================================================
// ------------------------- Active/Sleep Mode -------------------
// ==================================================================

// Sleep Mode
void wifi_sleep_mode(void)

// Active Mode
void wifi_active_mode(void)
```

Rysunek nr 4.7: Usypianie i wybudzanie modułu WiFi

W funkcji `wifi_sleep_mode()` użyta została komenda `AT+GSLP` do ustawienia modułu w tryb głębokiego uśpienia (deep sleep), w którym moduł zużywa minimalną ilość energii. Parametr '0' w komendzie oznacza, że moduł zostanie obudzony z tego trybu za pomocą sygnału zewnętrznego.

Funkcja `wifi_active_mode()` jest odpowiedzialna za przejście modułu Wi-Fi z trybu uśpienia lub głębokiego uśpienia do trybu aktywnego. Użyta jest tutaj pętla while do czasowego zastosowania napięcia na styku resetującym moduł Wi-Fi przez określony czas. Powoduje to wyjście modułu z trybu głębokiego uśpienia. Po aktywowaniu modułu styk resetujący jest ponownie ustawiany w stan niski.

Ostatnim elementem biblioteki modułu WiFi jest funkcja przedstawiona na rysunku nr 4.8.

```
// ==================================================================
// ----------------------------- RTC ----------------------------
// ==================================================================

// Get actual RTC data
void wifi_get_rtc(int* hours, int* minutes, int* seconds)
```

Rysunek nr 4.8: Funkcja pobrania czasu rzeczywistego

Funkcja `wifi_get_rtc()` ma na celu uzyskanie aktualnej daty i czasu z serwera NTP. Moduł łączy się z serwerem NTP o adresie 'time.nist.gov' na porcie 13, który jest znanym portem dla tego protokołu. Funkcja oczekuje na odpowiedź od serwera, sprawdzając, czy bufor WiFi zawiera oczekiwaną sekwencję '+IPD,51:'. Gdy odpowiedź zostanie otrzymana, jest ona kopiowana do dynamicznie alokowanej pamięci `(RTC_temp)`, a zmienna `OK_got_rtc_data` jest ustawiana na wartość true. Dane czasowe są odczytywane z uzyskanego fragmentu odpowiedzi.

W zależności od miesiąca, funkcja dostosowuje wartość aktualnego czasu dla strefy czasowej Polski. W okresie od kwietnia do października czas jest przesuwany o dwie godziny (czas letni). W pozostałych miesiącach czas przesuwany jest o godzinę (czas standardowy) [35].

Dodatkowo funkcja sprawdza status połączenia modułu WiFi, aby upewnić się, że połączenie z serwerem NTP zostało zakończone. Oczekuje, aż bufor WiFi będzie zawierać sekwencję 'STATUS:4', co będzie wskazywać na poprawne zakończenie połączenia. Po pomyślnym wykonaniu kodu następuje jeden dłuższy sygnał dźwiękowy.

4.1.2 main.c

Początek pliku głównego rozpoczyna definicja wybranych bibliotek w tym wspomniana wcześniej biblioteka <stdbool.h>, potrzebna do zdefiniowania zmiennych typu bool.

W sekcji odpowiedzialnej za definiowanie zmiennych znalazły się zmienne wraz z komentarzami, potrzebne do realizacji pętli głównej. Sekcja ta przedstawiona została na rysunku nr 4.9.

```
// Flood sensor detector flag
volatile bool flood = false;

// UPS power flag
volatile bool UPS_info = false;

// RTC falg
volatile bool RTC_reminder = false;

// ADC input voltage variable
uint32_t voltage_in;

// Correction used in voltage calculation
// [correction = battery voltage - measured voltage]
float correction = -0.5f;

// ThingSpeak channels description:
// "Allerts"
// 1 - RTC allert that everything works fine
// 10 - RTC flood appeared allert
// 20 - RTC UPS is ON allert
// 100 - flood appeared
// 110 - flood is over
// 200 - UPS is ON (+ sending actual battery charge status)
// 210 - UPS is OFF

// "Battery coverage"
// actual battery charge status in %
```

Rysunek nr 4.9: Zmienne pliku głównego

Rysunek nr 4.9 przedstawia następujące elementy:

1. Deklarowane są zmienne flagowe (typu bool), oznaczające różne stany:
   - `flood`: informacja o wykryciu zalania,
   - `UPS_info`: informacja o stanie zasilania z zasilacza UPS,
   - `RTC_reminder`: informacja o przypomnieniu z zegara czasu rzeczywistego (RTC).
2. Deklarowana jest zmienna `voltage_in` typu uint32t, która będzie przechowywać wartość napięcia z wejścia ADC (konwertera analogowo-cyfrowego).
3. Deklarowana jest zmienna `correction` typu float, używana w obliczeniach związanych z napięciem. Ta zmienna pełni rolę korekty (różnica między napięciem wyjściowym ładowarki TP4056, a napięciem odczytanym na wyprowadzeniu ADC mikrokontrolera STM).
4. Opisane są kanały komunikacyjne w bazie danych ThingSpeak:
   - 'Allerts':
     - 1: powiadomienie o poprawnym funkcjonowaniu RTC,
     - 10: powiadomienie o wystąpieniu zalania według RTC,
     - 20: powiadomienie o włączonym zasilaniu UPS według RTC (wraz z aktualnym stanem naładowania baterii),
     - 100: powiadomienie o wystąpieniu zalania,
     - 110: powiadomienie o zakończeniu zalania,
     - 200: powiadomienie o włączonym zasilaniu UPS (wraz z aktualnym stanem naładowania baterii),
     - 210: powiadomienie o wyłączonym zasilaniu UPS.
   - 'Battery coverage': wartość aktualnego stanu naładowania baterii wyrażona w procentach.

Kolejnym elementem pliku jest fragment odpowiedzialny za aplikowanie funkcji napisanych przez użytkownika. Na rysunku nr 4.10 przedstawiono dwie funkcje napisane na potrzeby działania urządzenia.

```
// Battery chre status map (%)
float map(float v, float x_min, float x_max, float y_min, float y_max)..

// Function to initialize proper app work
void TS_app_init(void)..
```

Rysunek nr 4.10: Dodatkowe funkcje

Pierwsza z nich ma na celu przemapowanie wartości `v` z przedziału `x_min - x_max` na przedział `y_min - y_max` z zachowaniem proporcji. Ta funkcja mapująca użyta jest w późniejszym etapie programu do wyznaczania wartości procentowego zużycia baterii.

Element `TS_app_init()` odpowiedzialny jest za poprawne wyświetlanie danych w aplikacji mobilnej po uruchomieniu urządzenia. Gwarantuje ona wysłanie testowych sygnałów do bazy ThingSpeak, upewniając przy tym użytkownika, że aplikacja oraz urządzenie działają poprawnie.

Następnym fragmentem pliku jest przestrzeń opisana w środowisku CubeIDE jako 'USER CODE BEGIN 2', która przedstawiona została na rysunku nr 4.11. Znajdują się w niej funkcje opisane wcześniej dla pliku wifi.c, obsługującego działanie modułu WIFi.

```
while(HAL_GPIO_ReadPin(UPS_ON_GPIO_Port, UPS_ON_Pin) == GPIO_PIN_RESET)
{
    // Wait for power from grid
}

// Start PWM Buzzer timer
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

// Hard WiFi Reset
wifi_active_mode();

// ----- WiFi Module init -----
wifi_init();

// -- Scan in-range networks --
//  wifi_scan_networks();

// ----- Connect to Wi-Fi -----
wifi_connect_to_network("ID", "haslo");

// ----- TS App Data Init -----
TS_app_init();

// ------ Update the RTC ------
int rtc_hours, rtc_minutes, rtc_seconds;
wifi_get_rtc(&rtc_hours, &rtc_minutes, &rtc_seconds);

RTC_TimeTypeDef sTime = {0};
sTime.Hours = rtc_hours;
sTime.Minutes = rtc_minutes;
sTime.Seconds = rtc_seconds;
HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN);

// Stop PWM Buzzer timer
HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
```

Rysunek nr 4.11: Pełna inicjalizacja peryferiów

W przedstawionym na rysunku nr 4.11 kodzie można zauważyć, że w pierwszej kolejności program czeka na ustawienie pożądanego stanu wyprowadzenia portu `UPS_ON`. Bez niskiego stanu logicznego na tym wyprowadzeniu program zapętli swoje działanie do momentu, aż warunek zostanie spełniony. Warunek ten ulega spełnieniu, gdy urządzenie działa w trybie zasilania sieciowego.

Dalej widoczne jest uruchomienie zegara obsługującego działanie głośnika informującego użytkownika o etapach inicjalizacji i poprawnym wykonaniu programu. Kolejnym fragmentem kodu są elementy opisane już wcześniej. Elementy te to inicjalizacja modułu WiFi oraz skanowanie dostępnych w zasięgu urządzenia sieci WiFi. Następnym elementem, który można zauważyć jest połączenie się modułu WiFi z siecią o nazwie 'ID' oraz wykorzystując nawiązane połączenie, poprzez funkcję `TS_app_init()` inicjalizację warunków dla aplikacji mobilnej. Ostatnim etapem tego fragmentu kodu jest pobranie z serwera NTP informacji o aktualnym czasie odpowiednim dla regionu Polski i załadowania tych informacji do odpowiednich zmiennych. Następnie pojawia się inicjalizacja pracy wbudowanego w mikrokontroler STM zegara RTC (wykorzystując do tego zdefiniowane zmienne). Zegar TIM1 ulega wyłączeniu w celu oszczędzenia energii elektrycznej. Kolejnym powodem dezaktywacji zegara TIM1 jest to, że proces inicjalizacji dobiegł końca i sygnały dźwiękowe do czasu wykrycia zagrożenia w postaci zalania cieczą nie są potrzebne do pracy urządzenia.

Końcowym elementem pliku jest program zawarty w przerwaniach oraz pętli głównej. Obsługa przerwań umiejscowiona jest w dyrektywie `HAL_GPIO_EXTI_Callback` (obsługa sygnałów GPIO) oraz `HAL_RTC_AlarmAEventCallback` (obsługa przerwania RTC). Dyrektywy te zostały pokazane na rysunku nr 4.12.

```
// ==================================================================
//  --------------------- GPIO Interrupts -----------------------
// ==================================================================
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

// ==================================================================
//  ------------------------ RTC Reminder -------------------------
// ==================================================================
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
```

Rysunek nr 4.12: Obsługa przerwań

Program przerwania odpowiedzialnego za obsługę sygnałów GPIO polega na dwóch wyprowadzeniach:

- `SENSOR_Pin:` przerwanie zostaje wyzwolone w momencie, gdy wyprowadzenie osiągnie niski stan logiczny. Jest to spowodowane zasadą działania czujnika zalania, który domyślnie podaje wyjściowy wysoki stan logiczny w momencie, gdy zalanie nie występuje.

- `UPS_ON_Pin:` przerwanie zostaje wyzwolone, gdy stan logiczny na styku osiągnie niski stan logiczny, co będzie oznaczać przerwanie zasilania sieciowego.

W obu tych obsługach preambułą jest wyjście mikrokontrolera oraz modułu WiFi z trybu uśpienia, oraz zaaplikowanie odpowiedniego sygnału dźwiękowego (w przypadku `SENSOR_Pin`) oraz użycie odpowiednich flag. Ten opis dotyczy także przerwania RTC, z wyróżnieniem, że przerwanie RTC następuje codziennie o godzinie 14:30 i służy sprawdzeniu poprawności pracy urządzenia. Poniżej przedstawiono funkcjonalność fragmentów kodu opisanego w pętli głównej programu:

- `Flood While Interrupt:` w tym fragmencie programu jest realizowana pętla, która działa wtedy, gdy flaga flood jest ustawiona na stan wysoki. Gdy czujnik wykryje zalanie, wysyłane są dane na platformę ThingSpeak, informując o wystąpieniu zalania. W przypadku, gdy czujnik przestanie wykrywać wodę wysyłana jest informacja o zakończeniu zalania, po czym urządzenie wchodzi w stan uśpienia.

- `Power Problem While Interrupt:` pętla while jest wykonywana, dopóki zmienna `UPS_info` jest równa stanowi wysokiemu. Po wykryciu zmiany wartości flagi, program wysyła informację na platformę ThingSpeak o utracie zasilania głównego i włączeniu zasilania awaryjnego UPS. Następnie odczytywana jest wartość liczbowa napięcia z wyprowadzenia ADC, która zapisywana jest później w zmiennej `voltage_in`.
  Z uwagi na 12-bitowość przetwornika ADC, wartość odczytywana z wyprowadzania jest liczbą z zakresu od 0 do 4096.

  Liczba ta wykorzystywana jest później w obliczeniu rzeczywistej wartości napięcia. Obliczanie wartości napięcia przedstawiono we wzorze (4.1).

$$U_r = (\frac{ADC_{in}}{ADC_{max}} + correction) * 2 \tag{4.1}$$

  gdzie:

  U_r - obliczona wartość napięcia rzeczywistego,

  ADC_{in} - wartość liczbowa zmierzona przez konwerter analogowo-cyfrowy,

ADC_{max} - maksymalna wartość liczbowa danych przetwornika ($2^{12} - 1$),

correction - wartość korygująca, która jest różnicą napięcia zmierzonego pomiędzy wyjściami ładowarki TP4056, a napięciem zmierzonym dla wyprowadzenia ADC mikrokontrolera STM.

Dodatkowo (z powodu użycia dzielnika napięcia przed stykiem przetwornika), należy pomnożyć otrzymaną wartość przez liczbę 2.

Następnie wynik jest mapowany za pomocą wspomnianej wcześniej funkcji `map()` do postaci wartości procentowej od 0 do 100. Z uwagi na możliwe przekłamania pomiarowe, wartość ta jest później kalibrowana, a następnie wysyłana do wyznaczonego kanału ThingSpeak. Gdy zasilanie główne wznowi działanie, do bazy danych zostanie przesłana odpowiednia wartość oznaczająca przywrócenie poprawnego działania urządzenia. Na koniec urządzenie wchodzi w stan uśpienia.

- `RTC While Reminder:` pętla uruchamiana jest codziennie o godzinie 14:30 poprzez zmianę stanu flagi `RTC_reminder` na true. Wewnątrz funkcji wykonywane jest sprawdzenie stanu wyprowadzeń GPIO. Jeśli przerwanie nie wykryje żadnych odstępstw od stanu pożądanego, tj. logicznego stanu niskiego na stykach `SENSOR_Pin` oraz `UPS_ON_Pin`, do bazy danych ThingSpeak wysyłana jest informacja, że wszystko działa prawidłowo. W przypadku wystąpienia zalania bądź utracenia zasilania głównego (a nie zostało to wykryte podczas pracy normalnej) przesłane zostaną do bazy odpowiednie wartości informujące o konkretnym zdarzeniu. Po przeprowadzeniu pełnego sprawdzenia podzespołów urządzenie przechodzi w stan uśpienia.

4.2 Program aplikacji mobilnej

Plikiem odpowiedzialnym za strukturę i wygląd aplikacji jest `activity_main.xml`, a głównym elementem realizującym program aplikacji jest aktywność `MainActivity.kt`. Opis obu tych plików przedstawiono poniżej:

1. `MainActivity.kt`

   Plik składa się z dwóch głównych klas:

   - **ThingSpeakService()** służy do pobierania danych z platformy ThingSpeak, przetwarzania tych danych i komunikacji z główną aktywnością w celu aktualizacji interfejsu użytkownika oraz generowania powiadomień. Działa jako pośrednik między aplikacją Android a źródłem danych na platformie ThingSpeak. ThingSpeakService jest odpowiedzialny za:
     - `Komunikację z platformą ThingSpeak:` klasa ta zawiera funkcje, które pozwalają na pobieranie danych z kanałów ThingSpeak poprzez protokół

HTTP. W kodzie używane są dwa różne adresy URL, które wskazują dwa różne kanały na platformie ThingSpeak, jeden związany z powiadomieniami (alertChannelUrl) i drugi związany ze stanem baterii (batteryChannelUrl).

  - `Asynchroniczne pobieranie danych:` w celu uniknięcia blokowania interfejsu użytkownika podczas pobierania danych z Internetu, ThingSpeakService wykorzystuje mechanizm AsyncTask. Dzięki temu operacje pobierania danych odbywają się w tle, a główny wątek aplikacji pozostaje responsywny.
  - `Przetwarzanie danych:` po pobraniu danych z bazy danych ThingSpeak, klasa ThingSpeakService przetwarza te dane i wywołuje odpowiednie metody tzw. 'listenera' (interfejs ThingSpeakListener), które zostały zaimplementowane w głównej aktywności (MainActivity).
  - `ThingSpeakListener:` jest to interfejs, który pozwala na komunikację między ThingSpeakService a MainActivity. ThingSpeakService wywołuje metody zdefiniowane w tym interfejsie, aby przekazać dane lub informacje o błędach, a MainActivity te metody implementuje, aby obsłużyć te informacje.

- `MainActivity : AppCompatActivity(), ThingSpeakListener()` to główna aktywność aplikacji Android, która obsługuje interfejs użytkownika oraz komunikację z ThingSpeakService poprzez implementację interfejsu ThingSpeakListener. Dzięki dziedziczeniu po AppCompatActivity aktywność ma dostęp do funkcji interfejsu użytkownika Android, a implementacja interfejsu ThingSpeakListener umożliwia jej reakcję na wywołanie funkcji związanej z pobieraniem danych z ThingSpeak. Poniżej opisano funkcjonalności zawarte w tej klasie:
  - `Companion Object i CHANNEL_ID:` zdefiniowany wewnątrz klasy, Companion Object pozwala na definiowanie stałych i właściwości statycznych dla danej klasy. W tym przypadku `CHANNEL_ID` jest stałą reprezentującą identyfikator kanału powiadomień.
  - `Handler i fetchRunnable:` handler umożliwia planowanie i wykonywanie zadań w określonych interwałach. fetchRunnable jest obiektem typu Runnable, który definiuje, co ma się wydarzyć w określonym interwale czasu. W tym przypadku fetchRunnable inicjuje powiadomienie oraz uruchamia pobieranie danych z ThingSpeak co 10 sekund.
  - `createNotificationChannel():` funkcja ta implementuje kanał powiadomień za pomocą klasy NotificationChannel dostępnej od Androida 8.0. Implementacja kanału jest konieczna, aby wyświetlać powiadomienia. Opis kanału i jego priorytet są ustawiane w tej funkcji.

- `showNotification()`: funkcja ta buduje i wyświetla powiadomienie z odpowiednimi parametrami, takimi jak tytuł, treść i ikona. W zależności od dostępności odpowiednich uprawnień powiadomienie jest budowane i wyświetlane za pomocą NotificationCompat.Builder.
- `Zmienne do przechowywania pobranych danych`: zdefiniowane zmienne firstAlertData, secondAlertData, firstAlertDate, secondAlertDate, lastBatteryData oraz batteryDate służą do przechowywania danych pobranych z ThingSpeak, takich jak informacje o alarmach i stanie baterii oraz ich daty.
- `onCreate()`: to jest funkcja, która zostaje wywołana, gdy aktywność jest inicjowana. Tutaj ustawiany jest interfejs użytkownika poprzez przypisanie odpowiedniego layoutu `(R.layout.activity_main)` za pomocą setContentView(). Ponadto, uruchamiany jest mechanizm cyklicznego pobierania danych za pomocą handler.post(fetchRunnable).
- `onDataFetched()`: ta funkcja jest częścią implementacji interfejsu ThingSpeakListener. Otrzymuje ona pobrane dane z ThingSpeakService w postaci dwóch obiektów JSON: alertChannelData i batteryChannelData. Dane te są analizowane, przetwarzane i przypisywane do odpowiednich zmiennych.
- `processData()`: funkcja ta przetwarza pobrane dane w celu aktualizacji interfejsu użytkownika. W zależności od warunków, obrazki i teksty w interfejsie są aktualizowane oraz wyświetlane powiadomienia.
- `correctAndFormatTime()`: ta funkcja konwertuje datę i czas w formacie UTC na lokalny czas strefy czasowej ("Europe/Warsaw"). Jest to używane do poprawnego formatowania i wyświetlania dat w interfejsie.
- `onDataFetchFailed()`: ta funkcja obsługuje sytuację, gdy pobieranie danych z ThingSpeak się nie powiedzie. W obecnej implementacji ta funkcja jest pusta, co oznacza, że nie wykonuje żadnych działań.
- `onDestroy()`: funkcja ta jest wywoływana, kiedy aktywność jest niszczona. W tym przypadku usuwa zaplanowane zadania związane z pobieraniem danych, aby uniknąć wycieków pamięci.

2. `activity_main.xml`: definiuje wygląd interfejsu użytkownika (ang. UI) dla aktywności MainActivity w aplikacji Android.

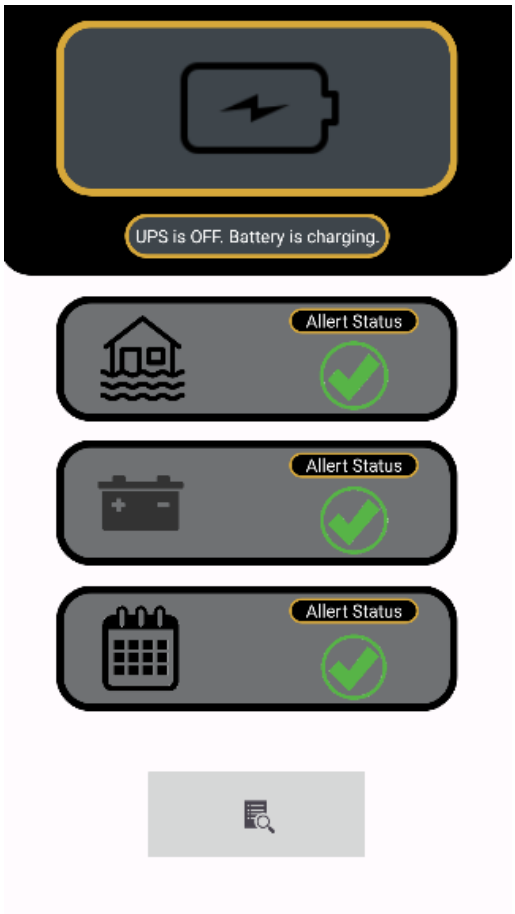
Aby zapewnić aplikacji możliwość łączności z siecią Internet oraz możliwość użycia powiadomień należało zmodyfikować odpowiednio plik manifestu. Plik manifestu (AndroidManifest.xml) to plik konfiguracyjny w systemie Android, który jest nieodzownym elementem każdej aplikacji Android. Ten plik jest obowiązkowy i znajduje się w głów-

nym katalogu projektu aplikacji. Manifest zawiera informacje o aplikacji, takie jak nazwa pakietu, komponenty (aktywności, serwisy, odbiorniki itp.), uprawnienia, konfiguracje sprzętowe i inne ważne informacje, które system Android wykorzystuje do zarządzania i uruchamiania aplikacji.

Ostatnim elementem wchodzącym w skład plików odpowiedzialnych za wizualny aspekt aplikacji są autorskie pliki zmieniające zaokrąglenie krawędzi bloków.

4.3 Wygląd aplikacji mobilnej

Finalny wygląd domyślny aplikacji pokazany jest na rysunku nr 4.13.



Rysunek nr 4.13: Wygląd domyślny aplikacji mobilnej

W zależności od wymiarów wyświetlacza platformy sprzętowej, wielkość interfejsu aplikacji ulega dopasowaniu. Widoczny na rysunku nr 4.13 interfejs aplikacji znajduje się w stanie domyślnym (tuż po inicjalizacji sprzętowej urządzenia). Interfejs ten można podzielić na dwa bloki:

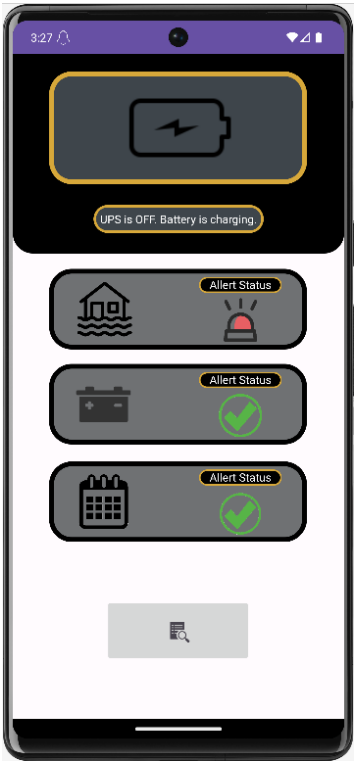
- **Górny:** informacja dotycząca stanu naładowania akumulatora UPS. Obecna ikona oraz napis sugerują, że UPS jest w stanie ładowania i nie jest aktualnie używany. W przypadku zmiany trybu zasilania na awaryjny aplikacja podmieni ikonę baterii

na obraz odpowiadający konkretnemu procentowemu przedziałowi zgromadzonego ładunku: 100 - 80, 80 - 50, 50 - 1. Informacją zwrotną będzie także liczbowa wartość naładowania wyświetlona w rubryce przeznaczonej na tekst.

- **Dolny:** w tej sekcji znajdują się 3 rubryki informujące o nieprawidłowościach i zagrożeniach jakie następują w czasie rzeczywistym. Ikony znajdujące się na początku każdej z rubryk nie ulegają zmianie. Są one wizualną reprezentacją, jakiego typu wydarzenia dotyczy ostrzeżenie. Pierwszą rubryką jest informacja dotycząca wystąpienia zalania. W przypadku wystąpienia zagrożenia ikona statusu znajdująca się po prawej stronie zmieni wygląd na ikonę alarmową. Tak samo nastąpi w przypadku drugiej rubryki (środkowa) ze wskazaniem, że dotyczy ona ostrzeżenia o awarii zasilania głównego. Ostatnia rubryka, której ikoną główną jest kalendarz odpowiada za podmianę ikony statusu na ikonę zalania lub akumulatora w przypadku awarii jednego z tych podzespołów. Ostatnim elementem segmentu dolnego jest przycisk służący jako hiperłącze do strony internetowej z przeglądem i wizualizacją historii zdarzeń.

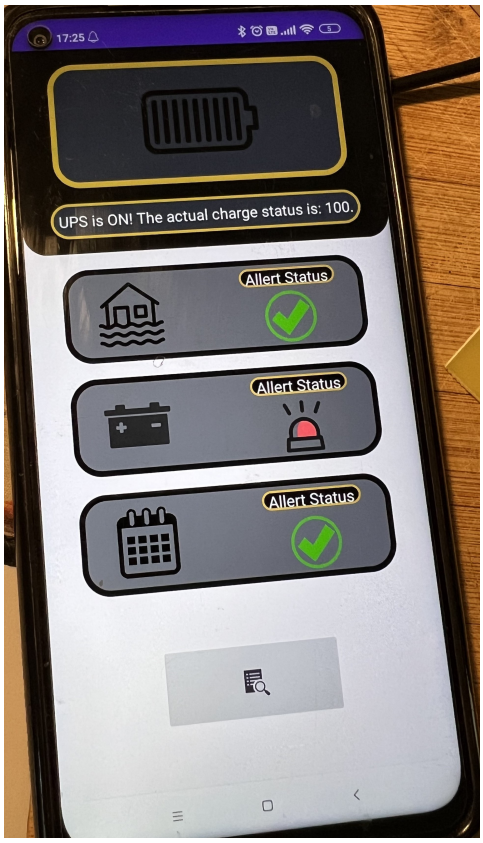
Widok ten pokazuje, że: zalanie nie nastąpiło, nie ma problemu z zasilaniem, kontrola RTC nie wykazała nieprawidłowości.

Zdjęcie nr 4.14 pokazuje symulację zrealizowaną w programie AndroidStudio. W przedstawionej sytuacji zauważyć można widok aplikacji w przypadku wystąpienia zalania. Widać tutaj zmianę ikony statusu oraz wysłanie powiadomienia oznaczonego jako dzwonek.



Rysunek nr 4.14: Symulacja zalania

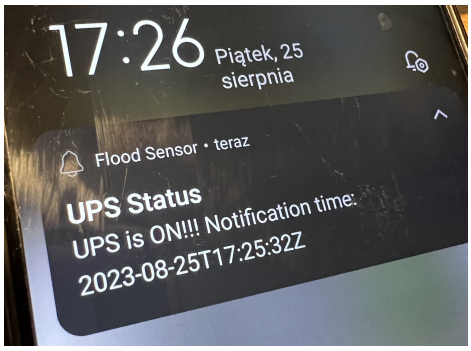
Dodatkowo na rysunku nr 4.15 przedstawiono wygląd aplikacji na telefonie osobistym.



Rysunek nr 4.15: Aplikacja na rzeczywistym urządzeniu

Na rysunku nr 4.15 można zaobserwować sytuację opisującą utratę zasilania sieciowego przez urządzenie. Sygnalizowane jest to poprzez zmianę ikony statusu dla drugiej rubryki oraz zmianę ikony baterii w górnej warstwie interfejsu aplikacji. Dodatkowym opisem zdarzenia jest informacja tekstowa przedstawiająca aktualny stan zużycia akumulatora awaryjnego. W tym przypadku stan naładowania akumulatora wynosi 100% co pokazane jest także przez ikonę baterii.

Po wystąpieniu zdarzenia wysłane zostało także powiadomienie, którego widok przedstawia rysunek nr 4.16.



Rysunek nr 4.16: Widok powiadomienia

Po wyświetleniu powiadomienia można zauważyć następującą treść:

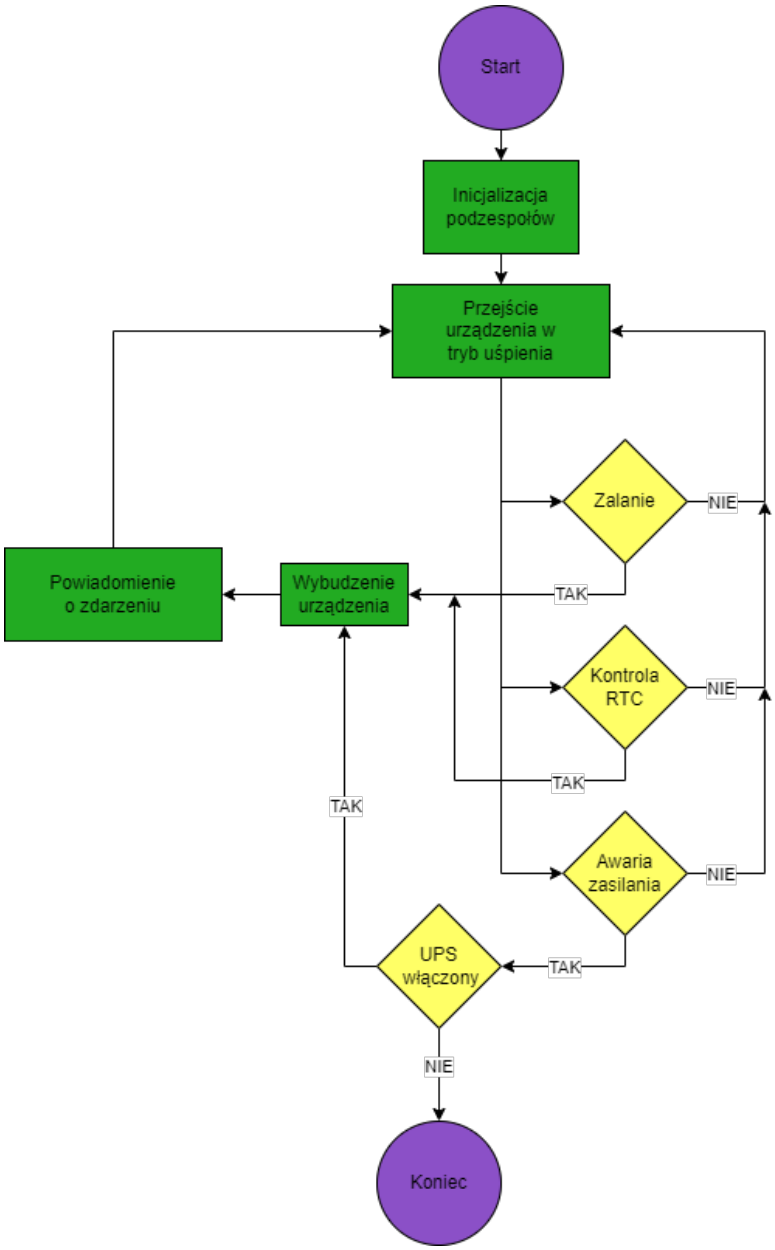
- **UPS Status**: jest to wskaźnik, jakiego typu zdarzenie nastąpiło. W tym przypadku zdarzenie dotyczy stanu zasilania.
- **UPS is ON!!!**: jest to treść komunikatu. Mówi ona, że nastąpiła zmiana źródła zasilania na awaryjne.
- **Notification time**: to informacja dotycząca czasu wystąpienia zdarzenia.

Informacje zawarte w treści powiadomienia są odpowiednie dla typu zdarzenia.

4.4 Schemat blokowy algorytmu urządzenia

Na rysunku nr 4.17 przedstawiono schemat blokowy algorytmu programu urządzenia. Poniżej opisano poszczególne etapy algorytmu:

- **START:** uruchomienie urządzenia poprzez użycie zasilania sieciowego.
- **Inicjalizacja podzespołów:** inicjalizacja pracy modułu WiFi, mikrokontrolera STM oraz zapewnienie poprawnej pracy aplikacji mobilnej.
- **Przejście urządzenia w tryb uśpienia:** zmniejszenie poboru mocy urządzenia poprzez uśpienie mikrokontrolera STM i modułu WIFi.
- **Zalanie:** gdy urządzenie znajduje się już w trybie niskiego poboru mocy, a czujnik wykryje obecność cieczy, to następuje wybudzenie urządzenia.
- **Kontrola RTC:** gdy urządzenie znajduje się w trybie niskiego poboru mocy, a czujnik wykona rutynową/codzienną kontrolę to następuje wybudzenie urządzenia.
- **Awaria Zasilania:** gdy urządzenie znajduje się w trybie niskiego poboru mocy, a urządzenie wykryje awarię zasilania, to sprawdzany jest stan trybu pracy modułu UPS. Gdy UPS jest dezaktywowany następuje wyłączenie zasilania i urządzenie kończy pracę (koniec pętli). W przeciwnym wypadku następuje wybudzenie urządzenia.
- **Powiadomienie o zdarzeniu:** w przypadku wykrycia zdarzenia i wybudzenia urządzenia, następuje zastosowanie odpowiedniego lokalnego sygnału alarmowego oraz wysłanie informacji o nastąpieniu konkretnego zdarzenia do bazy danych ThingSpeak. Po aktualizacji danych w bazie aplikacja mobilna jest w stanie odpowiednio zaalarmować użytkownika. Po zakończeniu zagrożenia urządzenie przechodzi ponownie w stan uśpienia.



Rysunek nr 4.17: Schemat blokowy algorytmu urządzenia

W odróżnieniu od algorytmu opisanego przed chwilą, algorytm aplikacji mobilnej jest mniej zaawansowany.

Algorytm aplikacji mobilnej polega na cyklicznym odświeżaniu pamięci aplikacji w interwałach co 10 sekund. Podczas każdorazowej aktualizacji danych program sprawdza dwie ostatnie wartości, które przechowuje baza danych ThingSpeak. W zależności od kombinacji tych wartości aplikacja wysyła odpowiednie powiadomienie i zmienia wskazaną ikonę interfejsu.

Do realizacji schematu blokowego aplikacji przysłużył się program Draw.io [36]. Tego samego programu użyto także podczas późniejszego przedstawienia schematu blokowego urządzenia.

4.5 Optymalizacja oprogramowania

Przed optymalizacją kod urządzenia zajmował 45.31KB pamięci FLASH oraz 3.28KB pamięci RAM. Po optymalizacji kodu urządzenia pod względem zajętości udało się uzyskać wynik 37.19KB zużycia pamięci FLASH oraz niecałych 3KB pamięci RAM.

Przyczyniło się do tego m.in. użycie własnej biblioteki obsługi modułu WiFi zamiast użycia już gotowej, zawierającej nadprogramowe funkcjonalności zbędne dla działania urządzenia docelowego. Dla biblioteki ESP-12E zrezygnowano także z obsługi bazy danych Firebase od firmy Google na rzecz bazy ThingSpeak. Spowodowane było to brakiem potrzeby stosowania dodatkowych bibliotek obsługujące transmisję danych do bazy ThingSpeak.

Szacuje się, że od pierwszej wersji oprogramowania do końcowej, zmniejszono zajętość programu mikrokontrolera o ok. 30%. Zrezygnowano z fragmentów odpowiedzialnych m.in. za oczekiwanie na odpowiedzi od modułów, czy też zrezygnowano z pomysłu użycia protokołu MQQT jako głównego protokołu komunikacji z bazą ThingSpeak (na wcześniejszym etapie prac również z bazą Firebase).

Aplikacja mobilna zawiera tylko niezbędne funkcje i klasy, a także jest zoptymalizowana tak, aby osiągać minimalną zajętość kodu przy maksymalnej wydajności. W początkowej fazie pisania programu i generowania wstępnego interfejsu postanowiono zrealizować dwa okna aplikacji. Pierwsze okno posłużyłoby jako podgląd zdarzeń w czasie rzeczywistym, a drugie jako tabela historii wystąpienia zagrożeń. Zrezygnowano jednak z tego pomysłu i zrealizowano interfejs w jednym oknie. Pozwoliło to zaoszczędzić czas realizacji pracy oraz pamięć programu.

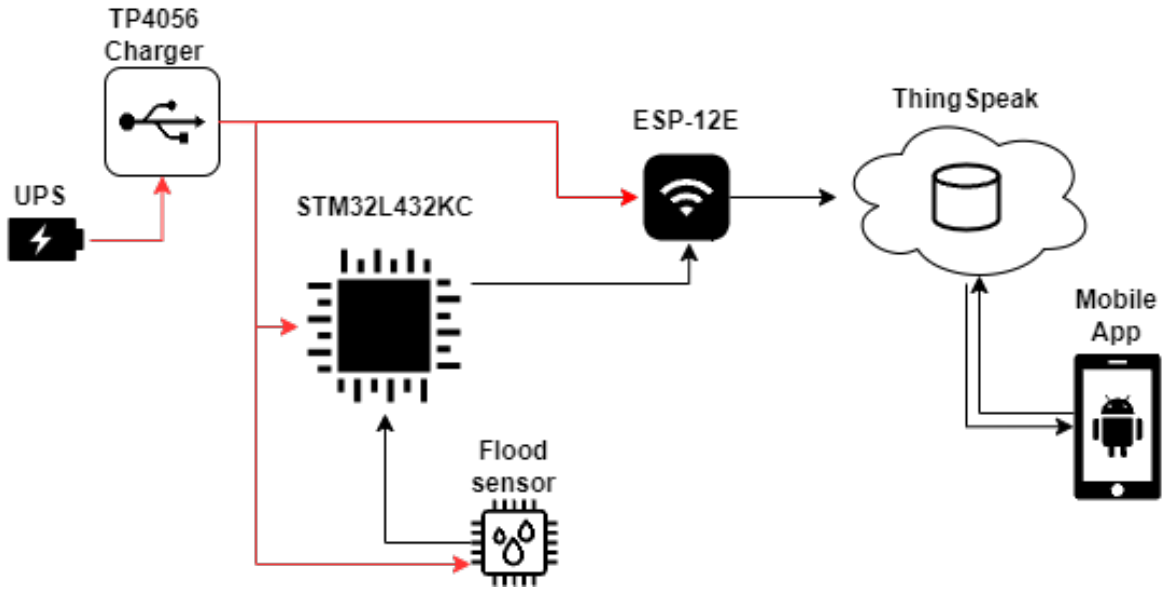
Program Android Studio umożliwia na sprawdzenie wydajności napisanej aplikacji za pomocą wbudowanego narzędzia 'Profiler'. Dane pomiarowe wskazują na zużycie 62MB pamięci urządzenia, a pobór energii przez aplikację jest zdefiniowany jako 'bardzo niski'.

Rozdział 5

Budowa i działanie

5.1 Schemat blokowy

Po określeniu podstawowych wymagań dotyczących budowy i działania czujnika zalania należało w kolejnym kroku sporządzić schemat blokowy urządzenia. Służy on wizualizacji wstępnych połączeń pomiędzy poszczególnymi zastosowanymi podzespołami oraz w minimalnym stopniu wizualizuje rozmieszczenie elementów na przygotowanej podstawie (w postaci układu stykowego). Schemat blokowy przedstawiono na rysunku nr 5.1.



Rysunek nr 5.1: Schemat blokowy

Zaczynając od lewej strony pierwszym elementem jest ładowarka TP4056 wraz z modułem UPS. Ładowarka zapewnia zasilanie pozostałych modułów układu, wykorzystując do tego zasilanie sieciowe. W momencie utraty zasilania głównego moduł UPS zapewni tymczasową pracę urządzenia umożliwiając tym samym wysłanie do bazy danych informacji o wydarzeniu. Głównym elementem układu jest mikrokontroler STM. Przetwarza on dane otrzymane z czujnika zalania oraz układu zasilającego i wysyła te dane do modułu

WIFi ESP-12E w odpowiedniej formie. Moduł WiFi łączy się z bazą danych ThingSpeak i przekazuje jej informacje powierzone przez mikrokontroler. Baza danych przechowuje i segreguje otrzymane informacje, które w późniejszym czasie zostaną wykorzystane przez aplikację mobilną wykorzystującą system operacyjny Android. Aplikacja w równych odstępach czasu będzie skanowała zawartość bazy danych i generowała odpowiednie komunikaty w czasie rzeczywistym.

5.2 Schemat elektryczny

Na rysunku nr 5.2 przedstawiono schemat elektryczny urządzenia docelowego. W lewym górnym rogu zauważyć można moduł ładowarki TP4056. Posiada ona wejście zasilające `TP_IN+/PWR_IN_GND`. Sygnał zasilający znajdujący się w tym miejscu pomaga modułowi mikrokontrolera NUCLEOL432KC zdiagnozować poprawność zasilania. Wyprowadzenie, które obserwuje zmianę w napięciu wejściowym to styk PB6. Dla bezpieczeństwa użyto tutaj dodatkowo dzielnik napięcia, zmniejszający poziom napięcia na wskazanym styku GPIO.

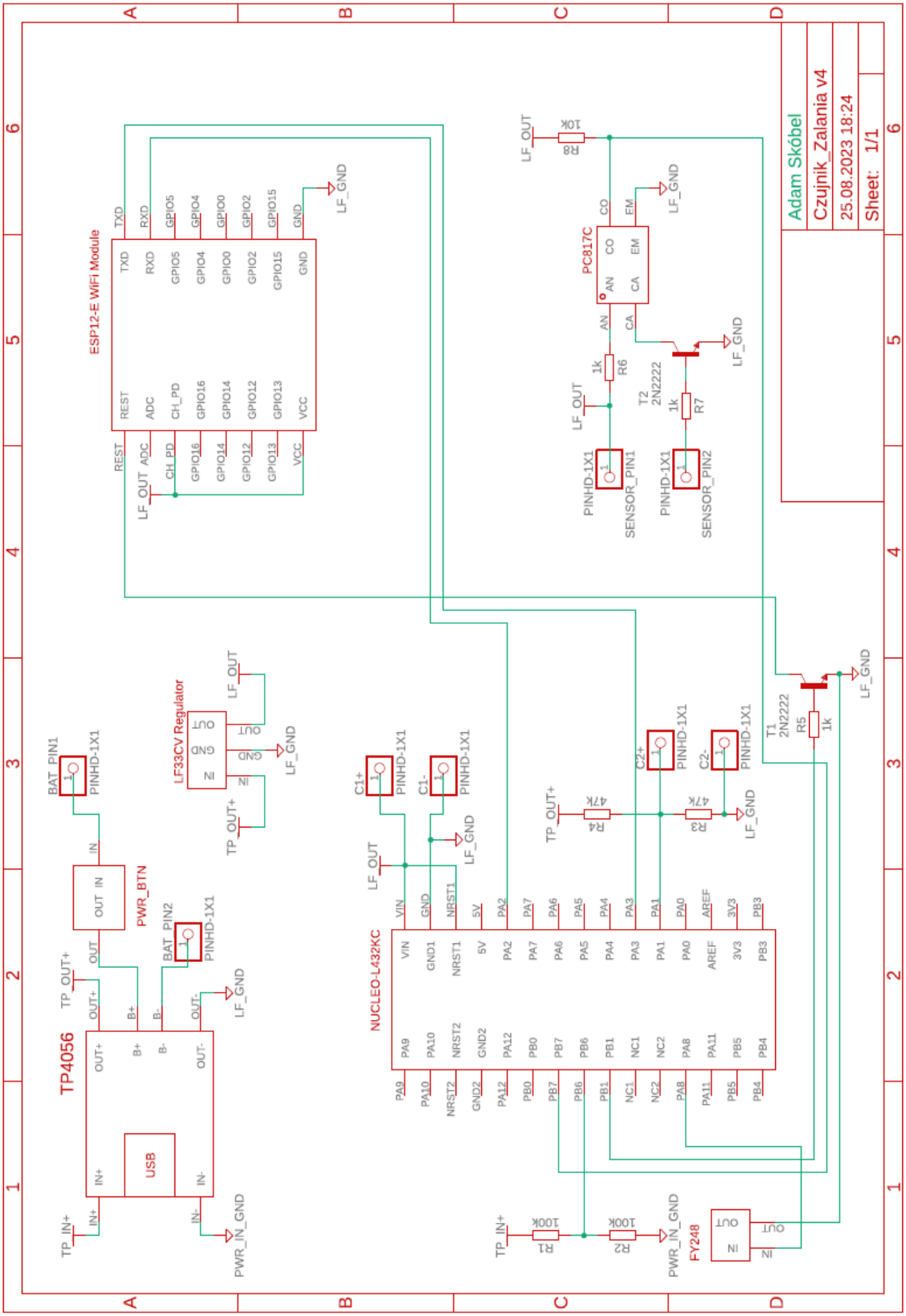
Moduł TP4056 posiada cztery wyjścia składające się na dwie pary wyprowadzeń. Pierwsza para `B+/B-` odpowiada za zapewnienie zasilania awaryjnego za pomocą dodatkowego modułu UPS. Zasilanie awaryjne można wyłączyć poprzez odciśnięcie przycisku `PWR_BTN`. Wyłączenie zasilania awaryjnego umożliwi wyłączenie urządzenia po odłączeniu go od zasilania głównego.

Kolejna para wyprowadzeń to `OUT+/OUT-`. Sygnał znajdujący się na tym wyprowadzeniu poprowadzony jest na stabilizator napięcia 3.3V LF33CV. Element ten zapewnia stabilny sygnał o napięciu 3.3V gwarantujący podzespołom czułym na poziom napięcia zasilania stabilne i bezawaryjne działanie.

Centralnym punktem schematu jest mikrokontroler STM NUCLEOL432KC wraz z widocznymi wyprowadzeniami. Mikrokontroler zasilany jest poprzez wspomniane już wcześniej wyprowadzenie stabilizatora napięcia. Pomiędzy stykiem zasilania oraz masy mikrokontrolera (`C1+/C1-`) ze względów bezpieczeństwa umiejscowiony jest kondensator elektrolityczny 47µF/25V. W przypadku pracy mikrokontrolera z użyciem zasilania zewnętrznego niezbędnym było zasilenie wejścia RESET napięciem 3.3V. Styk PA1 służy jako pomiar aktualnego napięcia na wyjściu z modułu TP4056 za pomocą przetwornika ADC. Można tutaj zaobserwować użycie styków `C2+/C2-`. Są one przeznaczone na zastosowanie kondensatora ceramicznego, użytego w celu zabezpieczenia przetwornika ADC przed zmianami napięcia wejściowego.

Kolejnym funkcjonalnym stykiem mikrokontrolera STM jest PB7, który otrzymuje informacje od czujnika zalania o wystąpieniu lub zakończeniu zalania.

Wyprowadzenie PB8 służy natomiast do nadzoru pracy głośnika, który w przypadku wystąpienia zalania generuje lokalnie dźwiękowy sygnał alarmowy.



Rysunek nr 5.2: Schemat elektryczny

Wyprowadzenia służące do komunikacji z modułem WiFi to:

- PB1: styk służący do wybudzania modułu WiFi poprzez zastosowanie tranzystora bipolarnego T1,
- PA2/PA3 (TX/RX): wymiana wiadomości mikrokontrolera z modułem WiFi poprzez port szeregowy UART.

Przedostatnim elementem widocznym na rysunku nr 5.2 jest czujnik zalania. Składa się on z rezystorów ograniczających prąd w obwodzie, fototranzystora PC817C zapewniający połączenie galwaniczne z mikrokontrolerem STM oraz tranzystora T2 zamykającego układ w przypadku wykrycia cieczy. Na schemacie można zauważyć także elementy o nazwach `SENSOR_PIN1` oraz `SENSOR_PIN2`. Służą one jako sondy pomiarowe czujnika zalania. Gdy nastąpi zwarcie tych dwóch wyprowadzeń, czujnik rozpocznie generowanie sygnału alarmowego.

Ostatnim elementem rysunku jest moduł WiFi ESP-12E. Dla poprawnego działania modułu należało zasilić styki `CH_PD` oraz `VCC` napięciem o wartości 3.3V, oraz uziemić styk `GND` do reszty układu.

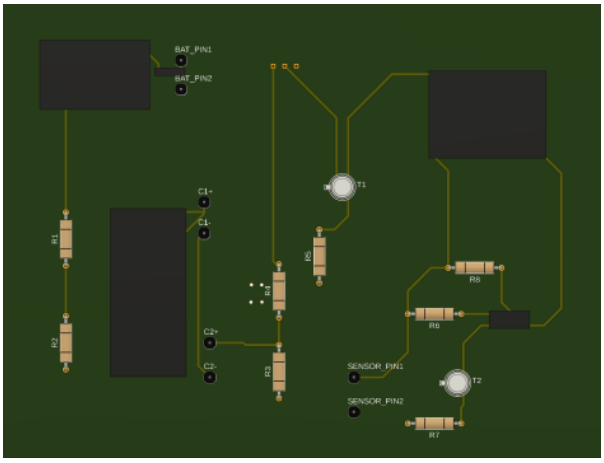
5.3 Schemat PCB

Na podstawie schematu elektrycznego został zbudowany także projekt układu drukowanego PCB. Układ modelowany był na podstawie układu dwustronnego, zawierającego wyprowadzenia dla złącza zasilania akumulatora, kondensatorów oraz sond czujnika zalania. Projekt układu drukowanego PCB przedstawiony został na rysunku nr 5.3. Rzeczywiste wymiary układu to 132mm szerokości i 100mm długości.

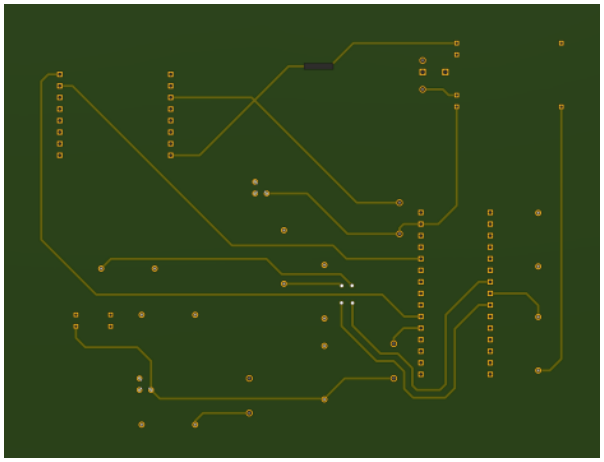
Rysunek nr 5.3: Schemat PCB

Warto wspomnieć, że zarówno plan elektryczny, jak i projekt płytki drukowanej PCB, zostały wykonane przy użyciu aplikacji EAGLE, która jest zintegrowana z programem Fusion360. Aplikacja ta posiada licencję umożliwiającą zastosowanie edukacyjne. Wszystkie biblioteki elementów użyte podczas procesu projektowania schematu oraz układu drukowanego bazują na wewnętrznych zbiorach programu EAGLE lub zostały wykonane ręcznie, zachowując oryginalne wymiary i układ wyprowadzeń oraz otworów montażowych użytych podzespołów. Do określania parametrów, takich jak wymiary komponentów czy rozmieszczenie otworów do montażu, wykorzystano informacje z katalogów dostarczonych przez producenta oraz dokonano pomiarów własnoręcznie, aby zestawić wartości dostarczone przez producenta z rzeczywistymi wymiarami.

Używając programu Fusion360 udało się także otrzymać widok 3D zaprojektowanego urządzenia. Wyniki działań przedstawiono na rysunkach nr 5.4 oraz nr 5.5 (rysunki nie odzwierciedlają rzeczywistych wymiarów).



Rysunek nr 5.4: Górna warstwa płytki



Rysunek nr 5.5: Dolna warstwa płytki

Projekt zamodelowanego urządzenia posłuży jako referencja podczas budowy podstawy pracy w postaci własnoręcznie wykonanej płytki PCB.

5.4 Instrukcja obsługi

Po podłączeniu urządzenia do zasilania rozpocznie się proces inicjalizacji działania czujnika zalania oraz modułu WIFi.

Proces inicjalizacji podzielony jest na trzy etapy, gdzie zakończenie każdego z osobna jest zasygnalizowane poprzez unikatowy sygnał dźwiękowy. Sygnały te opisano poniżej:

- 1 krótki: poprawne połączenie mikrokontrolera STM z ESP-12E oraz pozytywne ustawienie modułu WiFi w trybie klienta,
- 2 krótkie: połączenie się urządzenia z siecią WiFi,
- 1 długi: pobranie czasu RTC i załadowanie zmiennych programowych.

Po wygenerowaniu ostatniego sygnału dźwiękowego urządzenie przechodzi w tryb uśpienia, co pozwoli na oszczędzenie zużycia energii.

Istnieją 3 zdarzenia, które wybudzają urządzenie:

- **1:** Urządzenie wykrywa zalanie pomieszczenia cieczą, wydając przy tym wysoki, ciągły sygnał dźwiękowy. Wydarzenie to jest rozpoznawane przez aplikację mobilną, która wysyła powiadomienie użytkownikowi o zaistniałym zdarzeniu. Interfejs aplikacji również ulega zmianie. W rubryce oznaczonej przez symbol domu pojawi się ikona ostrzeżenia. W momencie, gdy czujnik wykryje brak wody (rozwarcie styków czujnika), sygnał dźwiękowy będzie funkcjonował jeszcze przez 15 sekund. Po tym czasie czujnik wyłączy sygnał dźwiękowy, przejdzie w stan uśpienia, a aplikacja mobilna powiadomi użytkownika o zniknięciu zagrożenia.
- **2:** Urządzenie wykrywa brak zasilania sieciowego. W takim przypadku urządzenie natychmiastowo przełącza się zasilanie awaryjne (jeśli jest włączone) i poprzez aplikację mobilną powiadamia o tym użytkownika. Należy także nadmienić fakt, że przycisk znajdujący się na obudowie urządzenia służy do ustawiania dostępności zasilania awaryjnego. Gdy przycisk ten jest odciśnięty, będzie powodować to deaktywację zasilania awaryjnego. Wciśnięcie spowoduje natomiast odwrotny skutek. Informację o utracie zasilania sieciowego wyświetli aplikacja mobilna. Zasygnalizowane to będzie poprzez powiadomienie oraz zmianę ikony ostrzeżenia w środkowej rubryce. Dodatkową informacją będzie aktualny stan naładowania zapasowego akumulatora. Aktualny stan zużycia jest wyświetlany w górze ekranu za pomocą obrazka baterii oraz napisu, podającego liczbową (procentową) wartość stanu naładowania. W przypadku wznowienia zasilania głównego urządzenie przejdzie w stan uśpienia, a aplikacja mobilna powiadomi użytkownika o przywróceniu normalnego trybu pracy.

- **3:** Codziennie o godzinie 14:30 czasu polskiego, urządzenie sprawdza poprawność działania swoich peryferiów. Podczas codziennej kontroli sprawdzany jest stan czujnika zalania oraz tryb zasilania. W przypadku, gdyby czujnik zalania lub układ nadzoru pracy zasilania działał niepoprawnie, codzienna kontrola mogłaby wykryć nieprawidłowości. Zależnie od otrzymanych informacji aplikacja mobilna informuje użytkownika o efektach kontroli. Informacja ta jest w postaci powiadomienia oraz zmiany ikony w trzeciej rubryce. W przypadku problemu z czujnikiem zalania ikona zmienia się na na symbol domu, a w przypadku problemów z układem zasilającym symbol zmienia się na akumulator.

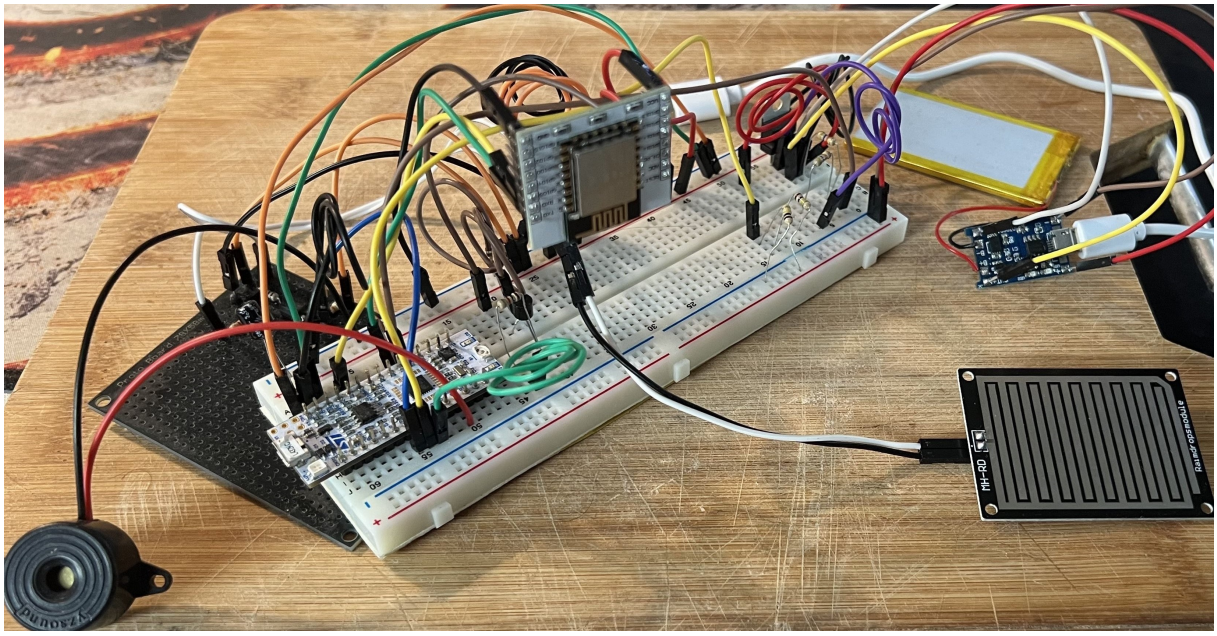
W przypadku zaistnienia potrzeby wyłączenia urządzenia należy deaktywować układ odpowiedzialny za dostępność zasilania awaryjnego poprzez odciśnięcie przycisku znajdującego się na boku obudowy. W przeciwnym razie urządzenie zamiast wyłączyć się zacznie przesyłać informację o włączeniu zasilania awaryjnego.

Ostatnią funkcjonalnością, na którą należy zwrócić uwagę jest element znajdujący się w dole aplikacji mobilnej. Służy on do wyświetlenia strony internetowej bazy danych ThingSpeak, gdzie możliwy jest podgląd historii zdarzeń. Zdarzenia opisane są za pomocą liczb, których znaczenie zamieszczono w opisie kanału ThingSpeak.

W razie braku funkcjonowania powiadomień w aplikacji mobilnej, należy sprawdzić uprawnienia, jakie system telefonu komórkowego przyznał aplikacji.

5.5 Prototyp urządzenia

Na rysunku nr 5.6 przedstawiono wygląd prototypu urządzenia.



Rysunek nr 5.6: Prototyp urządzenia

Można tutaj zauważyć, że podstawą łączącą wszystkie wymienione wcześniej w pracy elementy elektroniczne oraz podzespoły jest płytka stykowa. Zapewniła ona wystarczającą liczbę wejść, aby umożliwić przejrzysty wygląd koncepcji rozwiązania. Na rysunku widoczny jest również moduł WiFi, a zaraz pod nim mikrokontroler STM.

Po lewej stronie znaleźć można moduł głośnika oraz zaprojektowany czujnik zalania, którego podstawą konstrukcyjną jest dwustronna płytka prototypowa.

W prawej części rysunku odszukać można sondę pomiarową łączącą styki czujnika zalania przedstawione wcześniej na rysunku nr 5.4. Jest to sonda pomiarowa czujnika opadów deszczu YL-83. Za sondą pomiarową dostrzec można moduł ładowarki TP4056 wraz z podłączonym do niej akumulatorem litowo-polimerowym. Służy on jako zapasowe źródło zasilania.

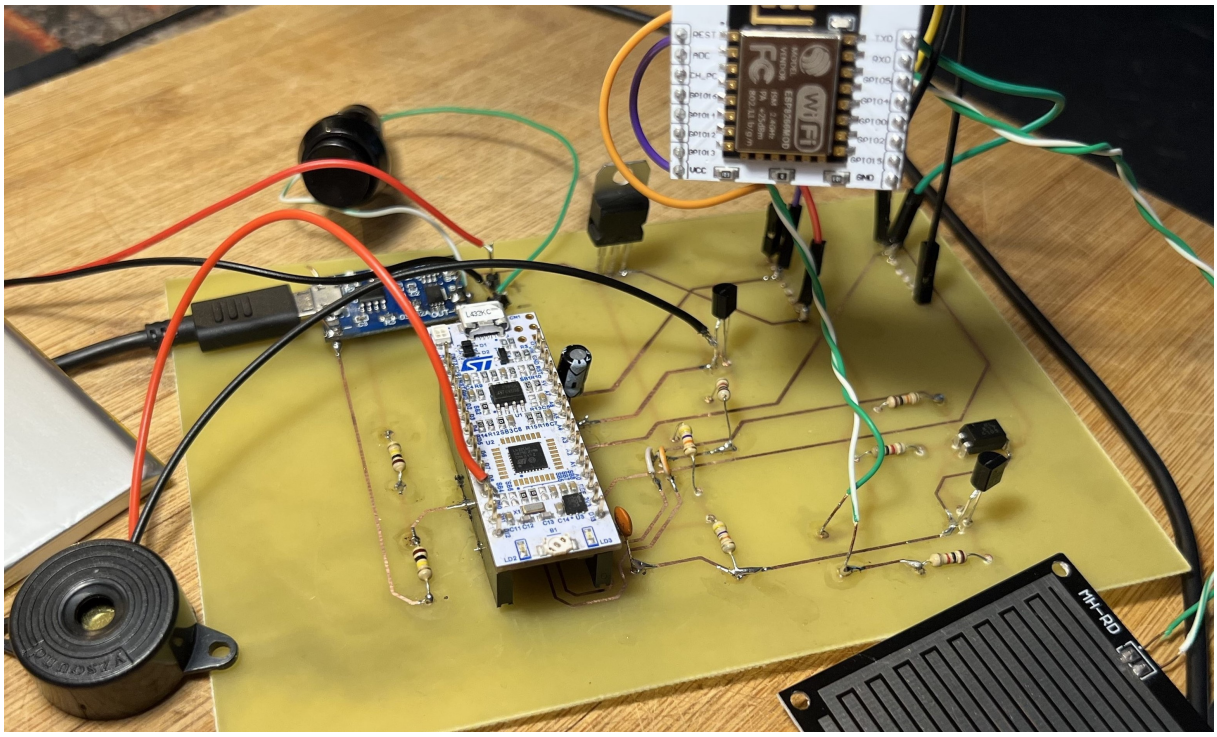
Na rysunku widoczne są także podstawowe elementy elektroniczne takie, jak kondensatory zabezpieczające, czy też rezystory. Słabo widocznymi elementami są stabilizator napięcia LF33CV oraz przycisk służący jako włącznik układu UPS.

Z uwagi na użycie płytki stykowej prototyp urządzenia posiada mankament w postaci możliwości wystąpienia niedokładności połączeń.

5.6 Urządzenie docelowe

5.6.1 Bez obudowy

Na rysunku nr 5.7 zauważyć można urządzenie przeniesione na płytkę PCB.



Rysunek nr 5.7: Urządzenie bez obudowy

Urządzenie po przeniesieniu na nową platformę w postaci płytki PCB działa prawidłowo i realizuje wszelkie funkcjonalności zawarte w planie pracy.

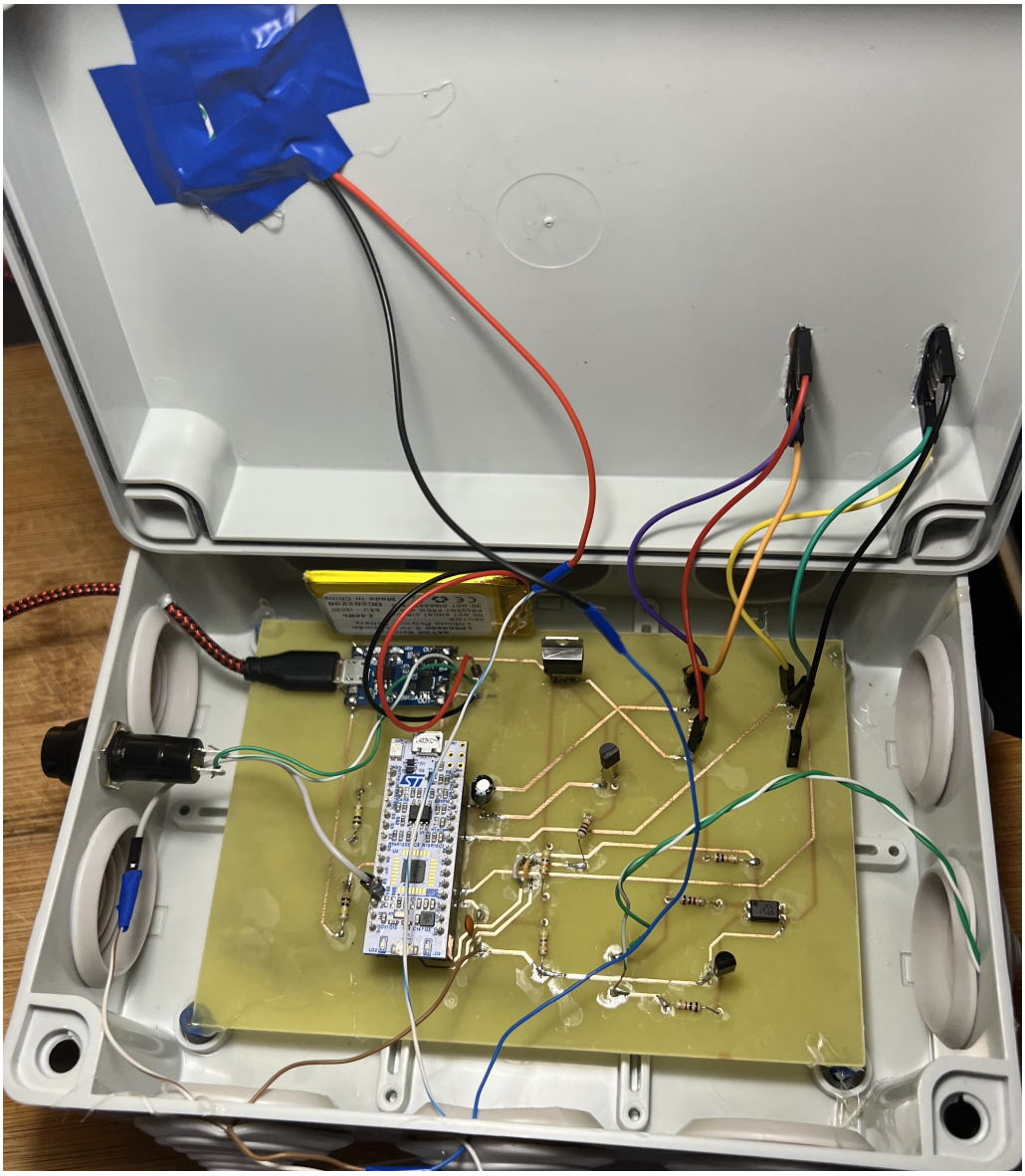
5.6.2 Z obudową

Urządzenie końcowe umieszczone w przygotowanej obudowie przedstawiono na rysunku nr 5.8.

Rysunek nr 5.8: Urządzenie w obudowie

Obudowa została dostosowana do aplikacji elementów sterujących, głośnika oraz modułu WiFi. Dodatkowo obudowa posiada certyfikat IP65, który świadczy o zabezpieczeniu obudowy przed wpływem wody i wilgoci. Można zauważyć tutaj, że moduł WiFi oraz głośnik został wystawiony na zewnątrz obudowy. Dzięki temu dźwięk alarmowy będzie mniej tłumiony przez otoczenie, co zwiększy czytelność powiadomienia o zagrożeniu. Dodatkowo moduł ESP-12E będzie miał lepsze warunki łączności z domową siecią WiFi, co zwiększy wydajność urządzenia.

Na rysunku nr 5.9 przedstawiono także wygląd urządzenia w obudowie po zdjęciu przykrycia górnej warstwy.



Rysunek nr 5.9: Urządzenie w obudowie bez przykrycia

Podstawę urządzenia w postaci płytki PCB umieszczono na wspornikach i przytwierdzono na stałe do dołu obudowy. Akumulator za pomocą dodatkowego wspornika został przyklejony do boku obudowy nad płytką PCB. Wywiercono specjalne otwory na wyprowadzenia modułu WiFi, głośnik oraz przełącznik bistabilny.

Specjalne otwory zawarte po bokach obudowy przeznaczono na wyprowadzenie głównego przewodu zasilającego oraz na ułożenie i zabezpieczenie przewodów prowadzących do sondy pomiarowej wykrywającej zalanie. Przewody te zostały zabezpieczone taśmą izolacyjną oraz "gorącym klejem". Sama sonda pomiarowa przytwierdzona została do spodu obudowy. Wszelkie elementy plastikowe potrzebne do montażu zostały wykonane za pomocą druku 3D.

Rozdział 6

Badania

W tym rozdziale przedstawiono kilka dodatkowych elementów, które ze względu na swoje walory dydaktyczne i eksperymentalne podległy analizie badawczej. Zaakcentowano tutaj m.in. aspekt badawczy dotyczący pomiaru zasięgu domowej sieci WiFi (określenie jakości sygnału w stosunku do odległości), aspekt badawczy dotyczący efektywnego zakończenia transmisji TCP, czy aspekt badawczy dotyczący określenia zużycia energii przez urządzenie w dostępnych trybach pracy.

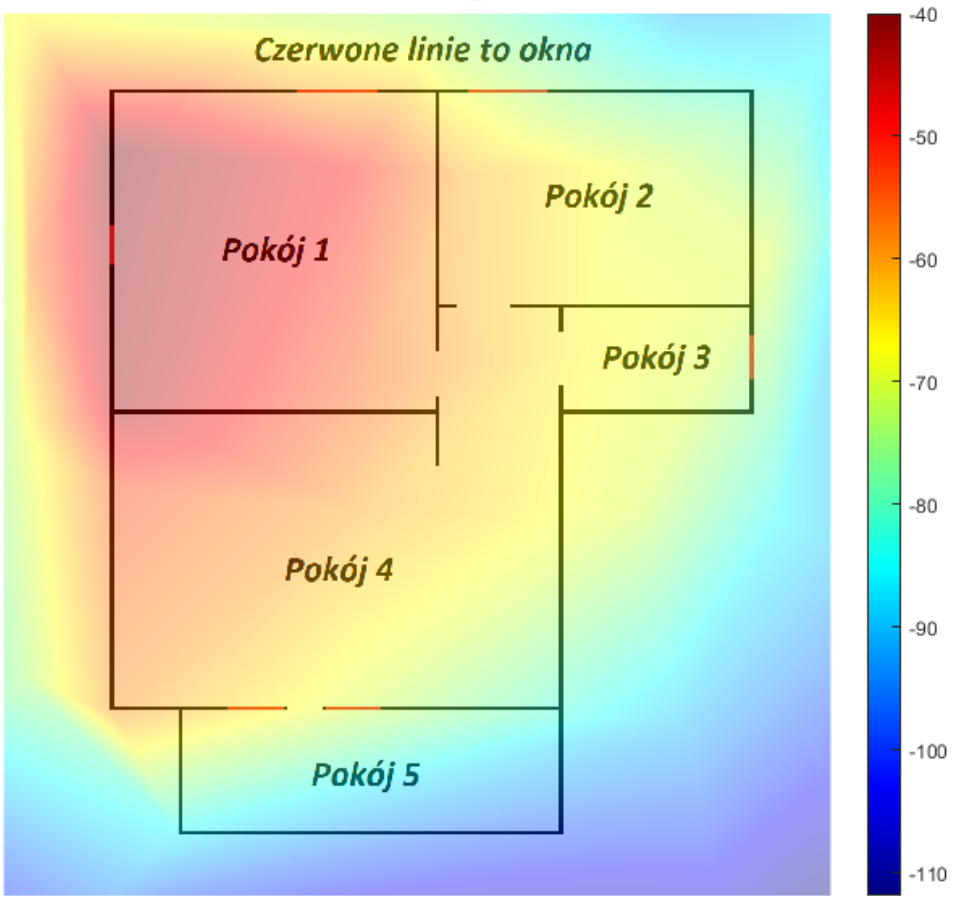
6.1 Badanie zasięgu sygnału WiFi

Pierwszym elementem, jaki podlega weryfikacji jest siła sygnału domowej sieci WiFi oraz jej rozkład w pomieszczeniach, gdzie potencjalnie może wystąpić wyciek. Jest to aspekt badawczy pracy, który wskaże najlepsze miejsce na umieszczenie czujnika. Wskaże on też miejsca, w których umieszczenie urządzenia będzie skutkować zmniejszeniem wydajności połączenia, co przełoży się na pogorszenie funkcjonalności.

Na rysunku nr 6.1 przedstawiono widok pomieszczeń, w których możliwe jest wystąpienie wycieku. Pomieszczenia te mieszczą się na pierwszym piętrze domu jednorodzinnego, a łączna powierzchnia wynosi $92m^2$.

Na widok ten nałożona została także mapa siły sygnału WiFi. Z prawej strony znajduje się oś wartości przydzielająca dane liczbowe mocy wyrażone w dBm do konkretnych kolorów i odcieni.

Dodatkową informacją jest fakt, że źródło sygnału znajduje się w pokoju nr 1. Można to stwierdzić, ponieważ właśnie w tym pomieszczeniu sygnał osiąga najlepsze wartości, co sprawia, że urządzenie można umieścić w dowolnym miejscu.



Rysunek nr 6.1: Mapa siły sygnału WiFi

Przedziały wartości mocy odpowiadają konkretnej jakości sygnału:

- $\leq 66dBm$ - doskonały,
- $66dBm \div 77dBm$ - dobry,
- $77dBm \div 82dBm$ - słaby,
- $82dBm \div 86dBm$ - zły,
- $86dBm \div 90dBm$ - bardzo zły.
- $> 90dBm$ - brak sygnału

Pokoje nr 2, nr 3 i nr 4 równie dobrze nadają się na umiejscowienie czujnika, co pokój nr 1. Moc sygnału w tych miejscach minimalnie spada do wartości ok. 70dBm.

Widoczną zmianą sygnału jest przejście pomiędzy pokojem nr 4, a pokojem nr 5. Utrata jakości sygnału w tym miejscu jest drastyczna. Spowodowane jest to rozproszeniem sygnału w wolnej przestrzeni, gdyż pokój nr 5 jest balkonem.

W wolnej przestrzeni wokół domu najlepszy sygnał można zaobserwować w pobliżu pomieszczenia nr 1. Jest to spowodowane dogodnym umiejscowieniem okien, które przepuszczają sygnał WiFi. Krótko po przekroczeniu bariery szkła sygnał ulega stopniowemu utraceniu.

Szczególnymi miejscami, które są najbardziej podatne na wyciek wody to pokój nr 3 oraz pokój nr 4. Pokój nr 3 to w rzeczywistości łazienka, w której zdarzają się wycieki z powodu nieszczelnego brodzika od prysznica. Pokój nr 4 to natomiast salon połączony z sekcją kuchenną. Jest to równie bardzo podatny na zalanie punkt z powodu uszkodzonej zmywarki oraz nieszczelnych rur. Pozostałe pokoje z wykluczeniem pomieszczenia nr 5 zagrożone są nieszczelnościami kaloryferów.

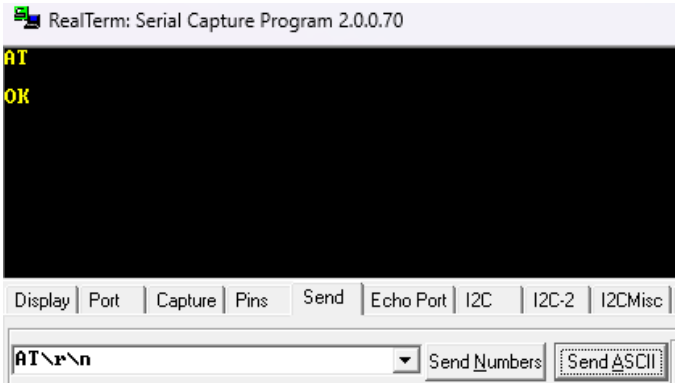
Na podstawie widocznych efektów, można stwierdzić, że powodem utraty siły sygnału jest materiał budowlany, oddzielający przedstawione pomieszczenia oraz odległość od źródła sygnału. Przykładowo sygnał w miejscu oddalonym od źródła o 9m (dwa pokoje) jest słabszy o 37dBm. Jest to znaczna różnica w mocy sygnału, jednak nie stanowi ona jeszcze poważnego problemu i zagrożenia dla zachowania ciągłości połączenia.

Przykładowym rozwiązaniem problemu utraty siły sygnału byłoby zastosowanie wzmacniacza sygnału, który zregeneruje jakość i przedłuży skuteczny zasięg sieci. Innym rozwiązaniem byłaby zmiana miejsca położenia routera.

6.2 Badanie transferu danych modułu ESP-12E

Za pomocą programu RealTerm oraz konwertera UART-USB zbadano sposób przesyłania komend AT pomiędzy mikrokontrolerem, a modułem WiFi.

Na rysunku nr 6.2 przedstawiono wygląd konsoli programu RealTerm oraz przykładową komendę użytą do przetestowania połączenia.



Rysunek nr 6.2: Program RealTerm

W tym przypadku jest to komenda `"AT\r\n"`. Jak widać na załączonym rysunku, odpowiedzią modułu ESP-12E jest `"OK"` co każe sugerować, że połączenie funkcjonuje poprawnie.

W podobny sposób sprawdzono także możliwość nawiązania połączenia z bazą danych ThingSpeak za pośrednictwem protokołu TCP. W tym celu należało użyć kolejno pięć poleceń:

- `AT+CIPMUX=0` - ustawienie modułu w trybie pojedynczego połączenia,
- `AT+CIPSTART="TCP","api.thingspeak.com",80` - połączenie się z interfejsem API ThingSpeak przy użyciu protokołu TCP,
- `AT+CIPSEND=51` - informacja dla kanału TCP odnośnie długości wysyłanych danych,
- `GET /update?api_key=<API>&field1=255` - informacja dotycząca klucza identyfikującego konkretny kanał ThingSpeak, numer pola oraz wartość przesyłana do bazy.

Aby zakończyć transmisję TCP należy użyć polecenia `AT+CIPCLOSE`. Testy jednak wykazały, że nie ma potrzeby używania polecenia kończącego transmisję TCP, gdyż po wysłaniu danych do kanału ThingSpeak, połączenie zamyka się automatycznie w chwili poprawnego wysłania danych.

6.3 Badanie bilansu energetycznego

Urządzenie podczas ciągłej pracy (tj. ciągłe wysyłanie informacji do bazy danych i lokalne alarmowanie o zalaniu) pobiera z sieci ok. 110mA. Wartość tą zmierzono wykorzystując multimetr wspomniany w rozdziale nr 3. W tabeli nr 6.1 przedstawiono zużycie energii poszczególnych podzespołów urządzenia z podziałem na dane katalogowe oraz dane zmierzone multimetrem.

Tabela nr 6.1: Wykaz zużycia energii

Nr	Element	Pobór zmierzony	Pobór deklarowany
1	Czujnik zalania	5.35mA (400pA w trybie 'sleep')	brak noty katalogowej (autorski czujnik)
2	Głośnik	225µA (z PWM)	15mA (bez PWM)
3	ESP-12E	70mA (10µA w trybie 'deep sleep')	120mA (10µA w trybie 'deep sleep')
4	STM NUCLEO L432KC	38mA (36mA w trybie 'sleep')	100mA (wszystkie piny GPIO włączone)
Suma:		110mA (37mA w trybie 'sleep')	235mA (115mA w trybie 'sleep')

Wartość poboru prądu na poziomie ok. 37mA przez urządzenie w trybie uśpienia jest zadowalająca. Różnica pomiędzy poborem całkowitym mający podstawę w notach katalogowych znacząco różni się od zmierzonego. Jest to spowodowane użyciem technik takich jak PWM, czy też wprowadzenie modułów w stan uśpienia.

Zużycie energii przez mikrokontroler można jeszcze bardziej obniżyć poprzez wylutowanie z układu programatora `ST_LINK` oraz diod sygnalizacyjnych. Przy aktualnym oprogramowniu mikroprocesora, zużycie prądu mogłoby oscylować w granicach 0.5mA.

Po uruchomieniu zasilania awaryjnego czas pracy urządzenia ogranicza się do ok. 11 godzin. Teoretyczny czas (T_t) obliczono za pomocą wzoru (6.1).

$$T_t = \frac{Pojemność\ akumulatora}{Pobór\ prądu} = \frac{1200mAh}{110mA} \approx 10.91h \approx 10h\ 54min \tag{6.1}$$

Czas ten obliczany jest jednak dla pełnego rozładowania akumulatora.

Urządzenie traci wszelkie funkcjonalności w momencie, gdy napięcie za stabilizatorem LF33CV osiągnie wartość 3.1V. Spowodowane jest to właściwością sekcji zasilania modułu WiFi, który potrzebuje do pracy napięcia o nominale przynajmniej 3.1V (wg. noty katalogowej producenta). Sekcja zasilania mikrokontrolera pozwala na użycie napięcia o wartości minimalnej 1.71V, lecz reszta elementów układu zawyża minimalną wartość napięcia, jakiej należy użyć, aby zapewnić działanie urządzenia.

Z tego powodu należało nanieść poprawkę na wzór obrazujący rzeczywisty czas działania urządzenia na zasilaniu awaryjnym. Obliczenie rzeczywistego czasu pracy przedstawiono we wzorze (6.2).

$$T_r = \frac{U_{max} - U_{min}}{U_{min}} * T_t = \frac{4.2 - 3.1}{4.2} * 10.91h \approx 2.86h \approx 2h\ 52min \tag{6.2}$$

gdzie:

U_{max} - maksymalne napięcie ładowania

U_{min} - minimalne dostępne napięcie zasilania

T_t - teoretyczny maksymalny czas pracy na zasilaniu awaryjnym

Rzeczywisty czas pracy urządzenia z użyciem zasilania awaryjnego wynosi w przybliżeniu 2 godziny i 52 minuty. W tym czasie użytkownik jest alarmowany o utracie zasilania głównego.

Po wykonania testu mającego na celu całkowite rozładowanie akumulatora awaryjnego wykazano, że urządzenie zakończyło pracę po okresie 2 godzin i 38 minut. Jest to zbieżne z czasem obliczonym we wzorze (6.2).

W celu przybliżenia czasu działania dla rozwiązań rynkowych należałoby wylutować programator z modułu mikrokontrolera co znacząco przedłużyłoby czas pracy urządzenia na zasilaniu awaryjnym. Zmniejszyłoby to także pobór energii podczas pracy normalnej.

Pomiar czasu pracy urządzenia z wykorzystaniem zasilania awaryjnego jest ważnym aspektem badawczym. Badaniu podległ także aspekt energooszczędności urządzenia.

Rozdział 7

Podsumowanie

Efekt końcowy pracy spełnia w 100% założenia projektowe. Urządzenie dokonuje detekcji wskazanego w celach pracy założenia w postaci zalania oraz powiadamia o tym fakcie użytkownika za pomocą aplikacji mobilnej (w czasie rzeczywistym).

Udało się także zaimplementować parę dodatkowych funkcjonalności, takich jak zapewnienie pracy w trybie zasilania awaryjnego z powiadomieniem o utracie zasilania głównego i wskaźnikiem naładowania akumulatora. Dodatkową funkcją wprowadzoną do rozwiązania jest również codzienny raport poprawności działania urządzenia wykorzystujący do odmierzania czasu i wybudzania urządzenia zegar RTC. W ramach aplikacji możliwy jest także podgląd całkowitej historii zdarzeń wraz z ich opisem.

Analiza aspektów badawczych pracy rozbija się na wiele obszarów. Pierwszym z nich jest rozdział nr 2, w którym skupiono się na analizie rozwiązań komercyjnych. Rozdział nr 3 poświęcony został przedstawieniu powodów wyboru poszczególnych modułów użytych do budowy urządzenia. Wspomniano tutaj m.in o aspekcie funkcjonalności oraz energooszczędności tychże modułów. Dalsze badania w tym rozdziale skupiły się na analizie dostępnego oprogramowania pod kątem użyteczności oraz opisie optymalizacji kodu programu aplikacji mobilnej i platformy sprzętowej. W podrozdziale nr 6.1 przedstawiono badania dotyczące zasięgu sygnału domowej sieci WiFi oraz podano powody jego utraty i sposoby regeneracji. Podrozdział 6.2 przeznaczony został na analizę transferu informacji modułu WiFi ESP-12E. Zbadano tutaj m.in. pojęcie działania komend AT oraz określono, w jaki sposób można efektywnie zakończyć transmisję danych TCP. Ostatnim elementem badawczym pracy jest podrozdział 6.3, w którym opisano pobór energii urządzenia w różnych trybach pracy oraz wyznaczono czas działania z użyciem zasilania awaryjnego.

Pewną niedoskonałością urządzenia jest fakt, że mimo użycia mikrokontrolera STM w wersji L (co symbolizuje serię o niskim poborze prądu), całkowity pobór prądu w trybie uśpienia nie jest do końca zadowalający. Mikrokontroler nie umożliwia łatwej możliwości odłączenia programatora od reszty modułu, co zmniejszyłoby pobór prądu. W przypadku użycia innego mikrokontrolera STM spełniającego wymogi wydajnościowe oraz wymogi związane z oszczędnością energii można wziąć pod uwagę użycie elementu NUCLEO-

F303RE [37]. Cechuje się on zastosowaniem zworki, po usunięciu której mikrokontroler drastycznie obniża zużycie prądu z poziomu ok. 50mA do 1.6mA. Jest to zdecydowanie bardziej wygodne rozwiązanie niż ingerencja w konstrukcję modułu celem dezaktywacji programatora. Niestety założenia pracy związane z wielkością urządzenia oraz ceną całkowitą uniemożliwiają wybranie podanego rozwiązania zastępczego.

Kolejną niedogodnością możliwą do poprawy jest użycie darmowej licencji bazy danych ThingSpeak. Użycie darmowego rozwiązania w zupełności wystarcza do poprawnego działania urządzenia, lecz wprowadza niepotrzebne opóźnienia w systemie alarmownia użytkownika o zdarzeniu. Problem ten można rozwiązać poprzez zakupienie jednej z licencji opisanych na stronie nr 17.

Podczas implementacji oprogramowania dla mikrokontrolera STM nastąpiła aktualizacja środowiska CubeIDE. Wprowadziła ona szereg błędów w bibliotekach, które m.in. uniemożliwiły pracę programatora. Spowodowało to brak możliwości wgrania nowego programu dla mikrokontrolera. Po kilku dniach CubeIDE wprowadził kolejną aktualizację, która naprawiła część błędów. Pozostał jednak problem, który polega na losowym braku funkcji wykrywania połączenia z mikrokontrolerem, co powoduje obniżenie funkcjonalności środowiska do minimum.

Praca magisterska przysłużyła się m.in. do zapoznania z budową oraz działaniem infrastruktury domowej sieci IoT, sposobem centralizacji obsługi domowych urządzeń nadzoru i monitoringu, a także budową aplikacji mobilnych. Szczególnie ważnym aspektem, jaki poruszyła budowa aplikacji było zaplanowanie interfejsu użytkownika w taki sposób, aby obsługa urządzenia była przystępna i czytelna.

Dodatkowym walorem edukacyjnym okazał się być proces wytrawiania ścieżek w dwustronnej płytce miedzi. Mimo zastosowania w programie projektowym odpowiedniej grubości ścieżek efekt końcowy procesu wygrzewania płytki nie był zadowalający. Z powodu użycia zbyt małej szerokości ścieżek, atrament nie odbijał się na laminacie, przez co proces należało powtarzać wielokrotnie. Dopiero zwiększenie szerokości ścieżek poprawiło efekt. Dodatkowe użycie proszku do prania w trakcie chłodzenia płytki sprawiło, że papier kredowy sam odszedł od warstw, a ścieżki odwzorowane zostały bez większych strat.

Możliwym kierunkiem rozwoju jest wprowadzenie zdalnej aktywacji i dezaktywacji zasilania awaryjnego z użyciem aplikacji mobilnej. Aplikacja w stanie aktualnym nie umożliwia także zdalnego wyłączania alarmu dźwiękowego. Zaaplikowanie takiej funkcji znacznie zwiększyłoby funkcjonalność urządzenia. Kolejnym możliwym rozwojem pracy byłby wybór przez użytkownika sygnału dźwiękowego, jaki włączałby się w momencie zagrożenia.

Do pracy dołączono dodatkowe pliki takie jak: streszczenie pracy, zdjęcia wykonane podczas prac, filmy (skompresowane darmowym programem Pazera [38]) zawierające testy urządzenia, kod źródłowy programu, schematy, instrukcję obsługi, wykorzystane biblioteki, kopię „bibliografii internetowej”, pliki oprogramowania, noty katalogowe oraz licencje do oprogramowania oraz bibliotek. Pliki te wchodzą w skład płyty wirtualnej [39].

Bibliografia

[1] Aleksander Kurczyk. *Mikrokontrolery STM32 dla początkujących*. Legionowo: BTC, 2019.

[2] Tomasz Francuz. *Język C dla mikrokontrolerów AVR, Wydanie II*. Gliwice: Helion, 2015.

[3] Andrzej Stasiewicz. *Android Studio Podstawy tworzenia aplikacji*. Gliwice: Helion, 2015.

[4] URL: https://www.evehome.com/en/eve-water-guard (dostęp: 12.03.2023).

[5] URL: https://allegro.pl/oferta/sonoff-wifi-czujnik-poziomu-cieczy-zalania-13459024148 (dostęp: 12.03.2023).

[6] URL: https://www.aqara.com/us/product/water-sensor/ (dostęp: 12.03.2023).

[7] URL: https://eurolook.com.pl/inteligentne-urzadzenia (dostęp: 12.03.2023).

[8] URL: https://allegro.pl/oferta/czujnik-sensor-zalania-woda-tuya-wifi-10165823413 (dostęp: 12.03.2023).

[9] URL: https://allegro.pl/oferta/xiaomi-wykrywacz-powodzi-czujnik-wycieku-wody-13533341280?utm_feed=aa34192d-eee2-4419-9a9a-de66b9dfae24&utm_source=google&utm_medium=cpc&utm_campaign=_dio_budownictwo-dd_pla_pmax&ev_campaign_id=17967348437&gclid=Cj0KCQjwldKmBhCCARIsAP-0rfz9sYyY0Ek78QmwPUtFRHyf95c57Mu4ZEMAFgNY7rQK59CbrA5Z_c0aAlv8EALw_wcB (dostęp: 12.03.2023).

[10] URL: https://urolakostapk.files.wordpress.com/2016/10/yl-83-rain-detector-datasheet_low.pdf (dostęp: 18.03.2023).

[11] URL: https://cdn-reichelt.de/documents/datenblatt/A300/SE045.pdf (dostęp: 18.03.2023).

[12] URL: https://www.waveshare.com/w/upload/7/78/Liquid-Level-Sensor-UserManual.pdf (dostęp: 18.03.2023).

[13] URL: https://www.st.com/resource/en/datasheet/stm32l432kc.pdf (dostęp: 18.03.2023).

[14] URL: https://components101.com/sites/default/files/2021-09/ESP12E-Datasheet.pdf (dostęp: 19.03.2023).

[15] URL: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html (dostęp: 19.03.2023).

[16] URL: https://www.alldatasheet.com/datasheet-pdf/pdf/43368/SHARP/PC817.html (dostęp: 19.03.2023).

[17] URL: https://www.alldatasheet.com/datasheet-pdf/pdf/15067/PHILIPS/2N2222.html (dostęp: 19.03.2023).

[18] URL: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf (dostęp: 05.05.2023).

[19] URL: https://download.kamami.pl/p584866-aky0622.pdf (dostęp: 05.05.2023).

[20] URL: https://pdf1.alldatasheet.com/datasheet-pdf/download/22819/STMICROELECTRONICS/LF33CV.html (dostęp: 05.05.2023).

[21] URL: https://www.alldatasheet.com/datasheet-pdf/pdf/332814/DUBILIER/FY24.html (dostęp: 05.05.2023).

[22] URL: https://taggsm.ru/pdf/catalog/704972.pdf (dostęp: 10.02.2023).

[23] URL: https://www.waveshare.com/w/upload/a/a3/CH343DS1-en.PDF (dostęp: 05.05.2023).

[24] URL: https://www.st.com/resource/en/data_brief/stm32cubeide.pdf (dostęp: 10.02.2023).

[25] URL: https://www.st.com/resource/en/data_brief/stm32cubemx.pdf (dostęp: 10.02.2023).

[26] URL: https://en.wikipedia.org/wiki/C_(programming_language) (dostęp: 10.02.2023).

[27] URL: https://developer.android.com/studio.

[28] URL: https://en.wikipedia.org/wiki/Java_(programming_language).

[29] URL: https://en.wikipedia.org/wiki/Kotlin_(programming_language).

[30] URL: https://www.autodesk.com/products/fusion-360/personal (dostęp: 23.06.2023).

[31] URL: https://www.autodesk.com/products/eagle/free-download (dostęp: 23.06.2023).

[32] URL: https://realterm.sourceforge.io (dostęp: 12.08.2023).

[33] URL: https://thingspeak.com (dostęp: 20.08.2023).

[34] URL: https://home.agh.edu.pl/~ggora/PE/hardware/Multimetr_DT9205A.pdf (dostęp: 10.02.2023).

[35] URL: https://www.gum.gov.pl/pl/aktualnosci/komunikaty/473,Informacja-dotyczaca-czasu-urzedowego.html (dostęp: 10.02.2023).

[36] URL: https://app.diagrams.net (dostęp: 10.02.2023).

[37] URL: https://www.st.com/resource/en/data_brief/nucleo-f303re.pdf (dostęp: 10.02.2023).

[38] URL: http://www.pazera-software.pl/products/free-audio-video-pack/ (dostęp: 10.02.2023).

[39] URL: https://polslpl-my.sharepoint.com/:f:/g/personal/adamsko563_student_polsl_pl/EsIjrBol7PJMk06twd5mkjEBNSmoJKYlHkBzguhriYl5vw (dostęp: 26.08.2023).

Dodatki

Spis skrótów i symboli

WiFi technologai sieci bezprzewodowej (ang. *Wireless Fidelity*)

UPS zasilacz awaryjny (ang. *Uninterruptible Power Supply*)

IoT Internet Rzeczy (eng. *Internet of Things*)

ARM zaawansowana maszyna RISC (ang. *Advanced RISC Machine*)

LED dioda elektroluminescencyjna (ang. *Light-Emitting Diode*)

PWM modulacja szerokości impulsów (ang. *pulse-width modulation*)

UART uniwersalny asynchroniczny nadajnik-odbiornik (ang. *Universal Asynchronous Receiver-Transmitter*)

UTC uniwersalny czas koordynowany (ang.*Coordinated Universal Time*)

SPI szeregowy port urządzeń peryferyjnych (ang. *Serial Peripherial Interface*)

I2C szeregowa, dwukierunkowa magistrala (ang. *Inter-Integrated Circuit*)

GPIO wejście/wyjście ogólnego przeznaczenia (ang. *General-Purpose Input/Output*)

RX dana odbierana (ang. *Receiving*)

TX dana wysyłana (ang. *Trasmitting*)

PCB obwód drukowany (ang. *Printed Circuit Board*)

AT zestaw komend do obsługi modemów (eng. *AT commands*)

UI interfejs uzytkownika (eng. *User Interface*)

RTC zegar czasu rzeczywistego (ang. *Real-Time Clock*)

RSSI wskaźnik mocy sygnału odbieranego (ang. *Received Signal Strength Indication*)

PC komputer osobisty (ang. *Personal COmputer*)

FLASH pamięć flash (ang. *Flash Memory*)

RAM pamięć o dostępie swobodnym (ang. *Random-Access Memory*)

MQQT protokół transmisji danych (eng. *MQ Telemetry Transport*)

NTP protokół synchronizacji czasu (ang. *Network Time Protocol*)

TCP protokół sterowania transmisją (ang. *Transmission Control Protocol*)

USB uniwersalna magistrala szeregowa (ang. *Universal Serial Bus*)

Li-Ion akumulator litowo-jonowy

dBm logarytmiczna jednostka mocy odniesiona do mocy 1 mW

Lista dodatkowych plików uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- streszczenie pracy,
- zdjęcia urządzenia (w obudowie i bez obudowy),
- filmy przedstawiające pracę urządzenia,
- kod źródłowy,
- schemat blokowy urządzenia,
- schemat PCB,
- instrukcję obsługi,
- biblioteki (+ licencje),
- oprogramowanie (+ licencje),
- kopię bibliografii internetowej,
- noty katalogowe.

Spis rysunków

3.1 Schemat ideowy czujnika 12

4.1 Zdefiniowanie bibliotek i zmiennych pomocniczych 20
4.2 Funkcje podstawowe 21
4.3 Inicjalizacja modułu WiFi 21
4.4 Skanowanie sieci 22
4.5 Funkcje łączenia i rozłączenia z siecią WiFi 23
4.6 Inicjalizacja połączenia z bazą danych ThingSpeak 24
4.7 Usypianie i wybudzanie modułu WiFi 25
4.8 Funkcja pobrania czasu rzeczywistego 25
4.9 Zmienne pliku głównego 26
4.10 Dodatkowe funkcje 27
4.11 Pełna inicjalizacja peryferiów 28
4.12 Obsługa przerwań 29
4.13 Wygląd domyślny aplikacji mobilnej 34
4.14 Symulacja zalania 35
4.15 Aplikacja na rzeczywistym urządzeniu 36
4.16 Widok powiadomienia 36
4.17 Schemat blokowy algorytmu urządzenia 38

5.1 Schemat blokowy 41
5.2 Schemat elektryczny 43
5.3 Schemat PCB 44
5.4 Górna warstwa płytki 45
5.5 Dolna warstwa płytki 45
5.6 Prototyp urządzenia 47
5.7 Urządzenie bez obudowy 48
5.8 Urządzenie w obudowie 49
5.9 Urządzenie w obudowie bez przykrycia 50

6.1 Mapa siły sygnału WiFi 52
6.2 Program RealTerm 53

Spis tabel

2.1 Porównanie urządzeń rynkowych. 6
2.2 Porównanie dostępnych czujników zalania. 7

3.1 Tabela kosztów 18

6.1 Wykaz zużycia energii 54

Spis wzorów

4.1 Rzeczywista wartość napięcia wyjściowego ładowarki TP4056 30
6.1 Teoretyczny czas pracy z użyciem zasilania awaryjnego 55
6.2 Rzeczywisty czas pracy z użyciem zasilania awaryjnego 55