POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

KIERUNEK AUTOMATYKA I ROBOTYKA

Praca dyplomowa magisterska

Badanie komunikacji bezprzewodowej z inteligentnymi elektrodami jonoselektywnymi

Autor: Artur Piernikarczyk

Kierujący pracą: dr inż. Józef Wiora

Gliwice, wrzesień 2015

Spis treści

1 Wstęp **6**

1.1 Cel pracy 6

1.2 Zakres pracy 6

1.3 Streszczenie rozdziałów 7

2 Pomiary jonoselektywne **8**

2.1 Potencjometria 8

2.2 Elektrody wskaźnikowe 11

2.3 Elektrody odniesienia 12

2.4 Metody analityczne 13

2.5 Źródła błędów 14

2.6 Zalety elektrod jonoselektywnych 14

3 Przegląd technologii bezprzewodowych **15**

3.1 IrDA 15

3.2 ZigBee 15

3.3 LwMesh 16

4 Projekt interfejsu elektronicznego **17**

4.1 Założenia projektowe 18

4.2 Oprogramowanie Eagle Light Edition 18

4.3 Moduł bezprzewodowy 19

4.4 Moduł koordynatora 21

4.5 Moduł urządzenia końcowego 24

4.5.1 Opis części analogowej 25

4.5.2 Opis czujnika temperatury 36

4.5.3 Podłączenie modułu 37

5 Oprogramowanie interfejsu elektronicznego **39**

5.1 Założenia projektowe 39

5.2 Środowisko programistyczne Atmel Studio 40
5.3 Programator KamPROG 41
5.4 Oprogramowanie 42
5.4.1 Obsługa modułu koordynatora 44
5.4.2 Obsługa modułu urządzenia końcowego 52
5.4.3 Zaprogramowanie układów 66

6 Oprogramowanie użytkowe PC 67
6.1 Założenia projektowe 68
6.2 Wybór oprogramowania 69
6.2.1 Python 69
6.2.2 PyGTK 69
6.2.3 Matplotlib 69
6.2.4 pySerial 69
6.3 Instalacja oprogramowania 70
6.3.1 Glade 71
6.3.2 pyCharm 72
6.4 Aplikacja 73
6.4.1 run_ISE_application.sh 73
6.4.2 ISE_lib.py 73
6.4.3 main.py 77
6.5 Dystrybucja LiveCD 98
6.5.1 Tworzenie dystrybucji 99
6.6 Instrukcja obsługi 100
6.6.1 Podłączenie 100
6.6.2 Konfiguracja 101
6.6.3 Kalibracja 103
6.6.4 Pomiar 104
6.6.5 Lista komend 106

7 Testy i pomiary bezprzewodowego systemu pomiarowego 107
7.1 Pomiar zasięgu 107
7.2 Pomiar poboru prądu 109
7.3 Test interfejsu I2C 110
7.4 Pomiar zakłóceń 111
7.4.1 Zakłócenia sieci elektrycznej 111
7.4.2 Zakłócenia wyeliminowane 115
7.5 Wyznaczenie niepewności pomiarowych 116

7.5.1 Pomiar potencjału elektrody jonoselektywnej ERH-115 116
7.5.2 Pomiar napięcia z generatora 124
7.6 Parametry techniczne bezprzewodowego systemu pomiarowego 130

8 Podsumowanie **131**

Bibliografia **132**

Dodatek A - Artykuł: „The Monte Carlo method in determining the measurement uncertainty of electronic measurement device” **134**

Rozdział 1

Wstęp

1.1 Cel pracy

Celem pracy jest zbudowanie interfejsu umożliwiającego wyposażenie elektrod jonoselektywnych w układ pomiarowy oraz moduł transmisji bezprzewodowej. Ponadto należy napisać oprogramowanie zarządzające transmisją danych. Należy wykonać badania nad działaniem łącza bezprzewodowego, a także oddziaływaniem zakłóceń pochodzących od transmisji na wynik pomiaru wielkości mierzonej. W pracy należy rozważyć użycie standardu radiowego ZigBee, optycznego (IR) i ewentualnie innego.

Niniejsza praca jest kontynuacją projektu inżynierskiego autora pt. „Bezprzewodowy pomiar napięcia za pomocą transmisji ZigBee i układu ATZB-24".

1.2 Zakres pracy

- studium literatury dot. pomiarów jonoselektywnych i transmisji bezprzewodowych;
- projekt, wykonanie i uruchomienie części elektronicznej;
- napisanie oprogramowania umożliwiającego komunikację z urządzeniem;
- wykonanie pomiarów określających parametry transmisji oraz wpływ zakłóceń;
- przygotowanie manuskryptu artykułu opisującego wyniki prac.

1.3 Streszczenie rozdziałów

- Rozdział 2 zawiera informacje teoretyczne dotyczące pomiarów jonoselektywnych. Przybliża budowę elektrod, wymienia metody analityczne oraz najczęstsze źródła błędów popełnianych podczas pomiarów.
- Rozdział 3 to krótki przegląd technologii bezprzewodowych takich jak IrDA, ZigBee oraz LwMesh. Zawiera opis wad i zalet tych technologii.
- Rozdział 4 dostarcza informacji o budowie elementów interfejsu elektronicznego i zasady działania poszczególnych podzespołów. Zawiera schematy oraz zdjęcia gotowych modułów.
- Rozdział 5 opisuje oprogramowanie interfejsu elektronicznego. Przedstawia zastosowane algorytmy oraz rozwiązania programistyczne użyte podczas implementacji protokołu transmisji bezprzewodowej.
- Rozdział 6 zawiera opis oprogramowania użytkowego PC. Zawiera także instrukcję obsługi aplikacji dla użytkownika.
- Rozdział 7 to testy i pomiary bezprzewodowego systemu pomiarowego. Opracowane pomiary oraz liczne zdjęcia pozwalają na zaznajomienie się z parametrami systemu pomiarowego.
- Rozdział 8 podsumowuje projekt.
- Dodatek A zawiera manuskrypt artykułu w języku angielskim, gdzie opisano sposób szacowania niepewności pomiarowych elektronicznego urządzenia pomiarowego za pomocą metody Monte Carlo.

Rozdział 2

Pomiary jonoselektywne

2.1 Potencjometria

Potencjometria to jedna z metod elektrochemicznych polegająca na pomiarze siły elektromotorycznej SEM ogniwa pomiarowego umieszczonego w badanym roztworze. Ogniwo pomiarowe składa się z dwóch elektrod: wskaźnikowej (pomiarowej) oraz odniesienia. Wartość potencjału elektrody odniesienia jest stała względem roztworu, natomiast potencjał elektrody wskaźnikowej zależy od wartości aktywności jonów w roztworze na które elektroda jest czuła.

Równanie Nikolskiego-Eisenmana opisuje zależność siły elektromotorycznej SEM ogniwa pomiarowego od wartości aktywności jonów badanego roztworu [1, s. 11]:

$$E_m = E_m^0 + \frac{RT}{z_m F} \cdot \ln\left(a_m + \sum_{m \neq n} K_{m,n} \cdot a_s^{\frac{z_m}{z_s}}\right) = E_m^0 + S \cdot \log_{10}\left(a_m + \sum_{m \neq n} K_{m,n} \cdot a_s^{\frac{z_m}{z_s}}\right) \tag{2.1}$$

gdzie:
E_m^0 - potencjał standardowy [mV]
R - uniwersalna stała gazowa 8,314 $[\frac{\mathrm{J}}{\mathrm{K \cdot mol}}]$
T - temperatura bezwzględna [K]
F - stała Faraday'a 96485$[\frac{\mathrm{C}}{\mathrm{mol}}]$
a_m - aktywność jonu oznaczanego $[\frac{\mathrm{mol}}{\mathrm{l}}]$
a_s - aktywność jonu przeszkadzającego
z_m - ładunek jonu oznaczanego
z_s - ładunek jonu przeszkadzającego
K_m - współczynnik selektywności na dany jon zakłócający
S - czułość elektrody jonoselektywnej [mV]

Czułość idealnej elektrody jonoselektywnej w temperaturze 25^oC wynosi 59,19 [mV] dla jonów jednowartościowych. W praktyce czułość elektrody jest mniejsza a charakterystyka względem logarytmu aktywności jonu nie jest liniowa w całym zakresie. Wartości współczynników selektywności na jony zakłócające powinny być jak najmniejsze, aby elektroda charakteryzowała się wysoką selektywnością. Istnieje granica wykrywalności elektrody, czyli parametr określający dolną granicę mierzonych wartości aktywności jonów.

Aktywność i stężenie jonu powiązane są zależnością [1, s. 4]:

$$a_m = \gamma_c \cdot c_m \tag{2.2}$$

gdzie:
a_m - aktywność jonu
γ_c - współczynnik aktywności molowej (w skali molowej)
c_m - stężenie molowe jonu $[\frac{\text{mol}}{\text{l}}]$

Współczynnik aktywności nie jest wartością stałą, dla niskich stężeń jest bliski jedności, natomiast dla wysokich najpierw maleje, a potem rośnie (nawet powyżej jedności). Prawidłowy pomiar stężenia jonu możliwy jest wtedy, gdy zapewniona jest stała wartość współczynnika aktywności. Do tego celu wykorzytuje się stabilizatory siły jonowej, zapewniające dużą siłę jonową badanej próbki. Zmiana stężenia oznaczanego jonu w roztworach o dużej sile jonowej nie wpływa na wartość współczynnika aktywności. Siła jonowa roztworu opisana jest zależnością [1, s. 4]:

$$I = 0,5 \cdot \sum_i c_i \cdot z_i^2 \tag{2.3}$$

gdzie:
c_i - stężenie molowe i-tego jonu w roztworze $[\frac{\text{mol}}{\text{l}}]$
z_i - ładunek i-tego jonu w roztworze

Warto wyrazić stężenie w skali pX, czyli ujemnym logarytmie z wartości stężenia:

$$pX = -\log_{10}(c) \tag{2.4}$$

gdzie:
c - stężenie molowe jonu $[\frac{\text{mol}}{\text{l}}]$

Zakładając pomijalnie mały wpływ jonów przeszkadzających oraz stały współczynnik aktywności molowej w badanych roztworach możliwe jest uproszczenie równania Nikolskiego-Eisenmana:

$$E_m = E_m^0 - S \cdot pX \tag{2.5}$$

gdzie:
E_m^0 - potencjał standardowy [mV]
pX - stężenie molowe jonu wyrażone w skali pX
S - czułość elektrody jonoselektywnej $[\frac{\text{mV}}{\text{pX}}]$

Równanie to opisuje zależność siły elektromotorycznej SEM ogniwa pomiarowego od wartości stężenia jonu w badanym roztworze.

2.2 Elektrody wskaźnikowe

Elektrody wskaźnikowe można sklasyfikować ze względu na mechanizm reakcji wymiany ładunków [1, 4]:

- Elektroda pierwszego rodzaju - odwracalna względem kationu lub anionu. Jest wykonana z metalu zanurzonego w roztworze zawierającym jony tego metalu. Wymiana ładunku następuje między jonami w roztworze i fazie elektrody.
  Elektrody tego typu to: srebrowa $Ag|Ag^{+}$, wodorowa $Pt|H_2|H^{+}$, cynkowa $Zn|Zn^{++}$ i chlorowa $Pt|Cl_2|Cl^{-}$.

- Elektroda drugiego rodzaju - odwracalna względem wspólnego anionu. Jest wykonana z metalu pokrytego cienką warstwą trudnorozpuszczalnej soli tego metalu, który zanużony jest w roztworze zawierającym wspólny anion z solą. Wartość SEM elektrody zależy od aktywności anionu w roztworze.
  Elektrody tego typu to: kalomelowa $Hg|Hg_2Cl_2|KCl$ i chlorosrebrowa $Ag|AgCl|KCl$.

- Elektroda trzeciego rodzaju - odwracalna względem wspólnego kationu. Jest wykonana na bazie elektrody drugiego rodzaju. Na powierzchni posiada trudnorozpuszczalną sól ze wspólnym anionem elektrody drugiego rodzaju i jest zanużona w roztworze zawierającym kation tej soli. Wartość SEM elektrody zależy od aktywności kationu w roztworze.
  Elektrody tego typu to: wapniowa $Pb|PbCO_3|Ca^{+}$ lub $Hg|Hg_2C_2O_4|Ca^{+}$.

- Elektroda redoks - zbudowana z metalu obojętnego np. złota, platyny, umieszczonego w roztworze zawierającym jony w formie utlenionej i zredukowanej. Wartość SEM elektrody zależy od stopnia utlenienia jonów w roztworze.
  Elektrody tego typu to: chinhydronowa, Fe^{3+} oraz $Pt|Fe^{2+}$.

- elektroda tlenkowa - zawiera metal pokryty warstwą tlenku tego metalu np. $Sb|Sb_2O_3|H^{+}$.

Elektrody wskaźnikowe klasyfikuje się także ze względu na rodzaj membrany [1, s. 6]:

- Elektroda z membraną stałą - istnieje kilka typów elektrod:
  - Elektroda z membraną szklaną - ma postać szklanej bańki wypełnionej elektrolitem z umieszczoną wewnątrz elektrodą wyprowadzającą. Selektywność na konkretne jony uzyskuje się za pomocą zmiany składu szkła.
  - Elektroda z membraną monokrystaliczną - zawiera membranę w postaci monokryształu np. LaF_3.
  - Elektroda z membraną homogeniczną - zawiera membranę w postaci spieku polikrystalicznego np. spiek kryształów Ag_2S.
  - Elektroda z membraną heterogeniczną - zawiera membranę zbudowaną z hydrofobowego tworzywa sztucznego, w którym zdyspergowany jest materiał aktywny np. AgCl czy $BaSO_4$.
- Elektroda z membraną ciekłą - membrana jest substancją aktywną tzw. jonoforem, będącym nośnikiem jonów np. walinomycyna, nonaktyna i monaktyna.
- Elektroda membranowa gazowa - zbudowana z membrany przepuszczalnej dla gazów i oddzielającej wewnętrzny elektrolit od badanej próbki. Zmiana SEM elektrody powstaje w wyniku przedostawania się gazu do elektrolitu.
- Elektroda membranowa enzymatyczna - membrana posiada enzym rozkładający substancję organiczną na jony, które powodują zmianę potencjału elektrody.

2.3 Elektrody odniesienia

Elektroda odniesienia zamyka ogniwo pomiarowe, dlatego powinna charakteryzować się stałością i odtwarzalnością potencjału oraz niskim oporem elektrycznym. Podstawową elektrodą odniesienia jest standardowa elektroda wodorowa posiadająca znakomitą powtarzalność, bias mniejszy niż 10 uV i potencjał równy 0 V będący punktem odniesienia w elektrochemii. Jest ona zbudowana z platynowej blaszki pokrytej czernią i zanużonej w jednomolowym roztworze HCl. Blaszka jest omywana gazowym wodorem o ciśnieniu 1013,25 hPa i uzyskiwany jest potencjał standardowy. Elektroda ta jest niewygodna w użyciu nawet w warunkach laboratoryjnych, dlatego powszechnie stosuje się elektrody odniesienia drugiego rodzaju np. elektrodę chlorosrebrową Ag|AgCl|KCl oraz kalomelową $Hg|Hg_2Cl_2|KCl$.

2.4 Metody analityczne

Istnieje kilka metod wyznaczenia stężenia substancji [8, 9]

- Metoda krzywej wzorcowej - wykonywany jest pomiar SEM elektrody dla serii roztworów wzorcowych o rosnącym stężeniu. Otrzymana krzywa wzorcowa pozwala odczytać stężenie dla badanej próbki. Metoda jest wrażliwa na substancje zakłócające (efekt matrycowy).

- Metoda roztworów ograniczających - gdy znany jest zakres stężeń w jakim próbka się znajduje, można wykonać pomiar SEM i przyporządkować go pomiędzy dwa punkty krzywej wzorcowej.

- Metoda jednokrotnego dodatku wzorca - wykonywany jest pomiar badanej próbki oraz pomiar po dodaniu do niej określonej objętości roztworu wzorcowego. Otrzymany wynik jest dokładniejszy od metody krzywej wzorcowej.

- Metoda kilkukrotnego dodatku wzorca - roztwór wzorcowy dodawany jest kilkukrotnie do badanej próbki, a stężenie wyznaczane jest metodą graficzną. Metoda stosowana dla bardzo niskich stężeń próbek lub skomplikowanej matrycy.

- Miareczkowanie potencjometryczne - metoda pośrednia, polegająca na pomiarze SEM elektrody i dodawaniu titranta do badanego roztworu. Titrant reaguje z wskazywanym jonem i stężenie jonu w roztworze zmienia się. Wykreśla się charakterystykę SEM w funkcji objętości titranta i określa stężenie jonu w badanym roztworze.

2.5 Źródła błędów

Główne źródła błędów w pomiarach potencjometrycznych to [4, s. 35]

- Temperatura - pomiary powinny być wykonywane w stałej temperaturze, gdyż zmiana temperatury powoduje zmianę właściwości roztworu, napięcia dyfuzyjnego oraz SEM elektrody.
- Niska impedancja wejściowa przyrządu pomiarowego - elektroda wskaźnikowa posiada bardzo wysoką rezystancję (np. elektroda szklana EH-01 firmy Hydromet posiada rezystancję $100 \div 200$ MΩ), więc istotna jest wysoka impedancja wejściowa przyrządu pomiarowego. Przepływ prądu przez ogniwo pomiarowe wpływa na stan równowagi chemicznej między elektrodą a roztworem.
- Niewłaściwe ekranowanie układu pomiarowego - w przewodach doprowadzeniowych elektrody indukują się zakłócenia spowodowane działaniem pól elektromagnetycznych i elektrostatycznych. Ważne jest prawidłowe ekranowanie przewodów oraz ich jak najmniejsza długość.
- Zbyt szybki odczyt pomiaru - ustalenie się równowagi chemicznej ogniwa pomiarowego dla niskich stężeń trwa dosyć długo. Odczyt pomiaru przed stanem ustalonym obarcza ten pomiar błędem.
- Nieaktualna krzywa wzorcowa - należy często przeprowadzać kontrolę krzywej kalibracji, gdyż pod wpływem czasu elektrody zmieniają swoje właściwości.
- Niepoprawnie przygotowany roztwór wzorcowy - roztwór wzorcowy powinien być czysty i posiadać dużą siłę jonową.
- Niepoprawne pH roztworu badanego - elektrody wskaźnikowe działają poprawnie tylko w pewnym zakresie pH. Istnieje zjawisko łączenia się jonów wodorowych lub wodorotlenowych z jonami oznaczanymi, powodujące błąd pomiaru.

2.6 Zalety elektrod jonoselektywnych

Elektrody jonoselektywne są wykorzystywane w laboratoriach, ośrodkach medycznych, zakładach przemysłowych i wielu innych obszarach. Wykonane z półprzewodnika lub wypełnione żelem charakteryzują się trwałością i wytrzymałością, co umożliwia ich zastosowanie w trudnych warunkach przemysłowych. Bezpośredni pomiar substancji w jeziorze czy ściekach jest łatwy i szybki. Służą zarówno do informacji o przekroczeniu pewnych poziomów stężenia, jak i do ciągłego monitorowania zmian stężenia.

Rozdział 3

Przegląd technologii bezprzewodowych

Podczas prac nad projektem należało dokonać wyboru technologii bezprzewodowej. W związku z tym dokonano krótkiego przeglądu trzech technologii: IrDA, ZigBee oraz LwMesh.

3.1 IrDA

IrDA jest standardem bezprzewodowej transmisji danych, wykorzystującym promieniowanie podczerwone. Przeznaczona jest głównie do tworzenia tymczasowych sieci typu punkt-punkt. Standard ten był bardzo popularny na początku XXI wieku, szeroko stosowany w komputerach, telefonach komórkowych, drukarkach i aparatach cyfrowych. Standard jest prosty i tani w implementacji jednak maksymalny zasięg to tylko 1 metr przy prędkości transmisji 115,2 kb/s (IrDA 1.0). Jakakolwiek przeszkoda pomiędzy urządzeniami powoduje przerwanie transmisji. Wady standardu wykluczyły użycie go w niniejszym projekcie.

3.2 ZigBee

ZigBee jest specyfikacją protokołów transmisji danych bazującą na standardzie IEEE 802.15.4. Umożliwia tworzenie sieci bezprzewodowych 2,4 GHz o niewielkim zasięgu (do 100 m) i prędkości do 250 kbit/s. Sieć zbudowana w oparciu o moduły radiowe może być łatwo rozszerzana, ponieważ każde z urządzeń może pełnić rolę routera np. w topologii siatki (mesh). Wyróżnia się trzy typy urządzeń: koordynator (węzeł początkowy do którego przyłączają się pozostałe urządzenia), router (umożliwia przesyłanie danych dalej, do innych urządzeń) oraz urządzenie końcowe (przesyła dane do routera, może

być usypiane). Technologia ZigBee jest stosowana głównie w sieciach personalnych, automatyce domowej i systemach alarmowych.

3.3 LwMesh

LwMesh jest protokołem sieciowym firmy Atmel, zaprojektowanym dla modułów radiowych tej firmy. Zapewnia on podstawową obsługę sieci bezprzewodowej 2,4 GHz. Wyróżnia się dwa typy urządzeń: urządzenia z routingiem (są rdzeniem sieci, przesyłają dane między urządzeniami) i bez routingu (znajdują się na końcach sieci, mogą być usypiane). Protokół LwMesh posiada format ramki zgodny ze standardem IEEE 802.15.4, jednak sam protokół nie jest zgodny z tym standardem.

Główną zaletą tego protokołu jest prostota implementacji oraz mała ilość zajmowanej pamięci w mikrokontrolerze. Znajduje zastosowanie w systemach zdalnego sterowania, systemach alarmowych, automatyce domowej, gdyż posiada możliwości zbliżone do specyfikacji ZigBee. W związku z powyższym zdecydowano się na użycie technologii LwMesh.

Rozdział 4

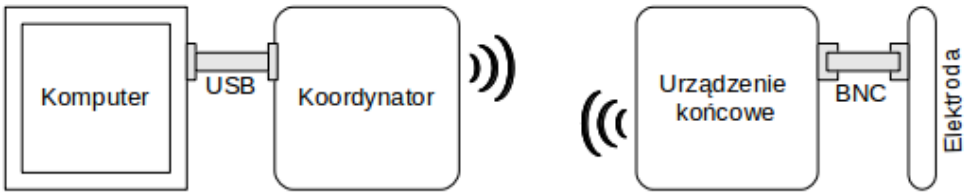
Projekt interfejsu elektronicznego

Interfejs elektroniczny (rysunek 4.1) można podzielić na dwa moduły komunikujące się ze sobą za pomocą transmisji bezprzewodowej:

- Moduł koordynatora - podłączony do komputera PC. Służy do wysyłania komend i odbierania danych pomiarowych z modułu urządzenia końcowego.
- Moduł urządzenia końcowego - podłączony do elektrody jonoselektywnej. Służy do wykonywania pomiarów potencjału elektrody jonoselektywnej oraz pomiarów temperatury badanego roztworu.

Każdy z modułów jest wyposażony w moduł bezprzewodowy (zaprojektowany i wykonany na potrzeby projektu inżynierskiego).

Przestudiowanie literatury dotyczącej pomiarów jonoselektywnych pozwoliło na doprecyzowanie wymagań stawianych elektronicznym urządzeniom pomiarowym. Wymagania te zostały uwzględnione w założeniach projektowych.



Rysunek 4.1: Schemat blokowy interfejsu pomiarowego

4.1 Założenia projektowe

Założenia przyjęte podczas prac nad interfejsem elektronicznym:

- wysoka rozdzielczość pomiaru napięcia - potencjał elektrody wynosi kilkaset miliwolt, pomiar napięć rzędu pojedynczych miliwoltów jest wskazany;
- wysoka impedancja wejściowa wzmacniacza pomiarowego - elektroda posiada dużą rezystancję wewnętrzną, dlatego ważne jest, aby impedancja wejściowa wzmacniacza nie obciążała prądowo elektrody;
- odporność na zakłócenia elektromagnetyczne i elektrostatyczne - odpowiedni sposób projektowania obwodu drukowanego oraz użycie filtru dolnoprzepustowego minimalizuje wpływ zakłóceń na pracę interfejsu;
- niski pobór prądu - możliwość zasilania bateryjnego;
- niewielkie wymiary - możliwość umieszczenia w obudowie.

4.2 Oprogramowanie Eagle Light Edition

Projekt interfejsu elektronicznego został wykonany przy pomocy Eagle Light Edition. Jest to oprogramowanie do wspomagania projektowania obwodów elektronicznych. Zawiera edytor schematów ideowych i PCB, rozbudowaną bibliotekę elementów oraz funkcje sprawdzania poprawności schematu z regułami projektowymi (tzw. Electrical Rule Check i Design Rule Check).

Wersja darmowa posiada kilka ograniczeń:

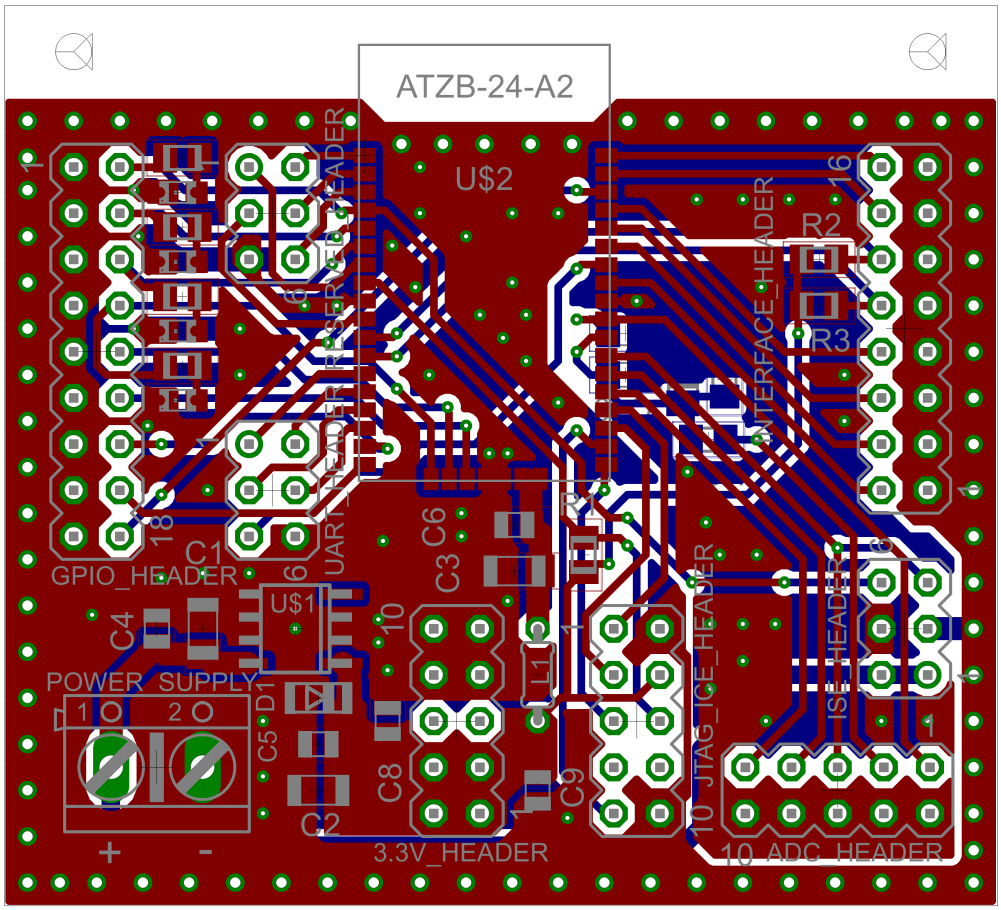
- maksymalny rozmiar PCB to 100 x 80 mm,
- tylko dwie warstwy (Top i Bottom),
- edytor schematów może utworzyć tylko jeden dokument.

4.3 Moduł bezprzewodowy

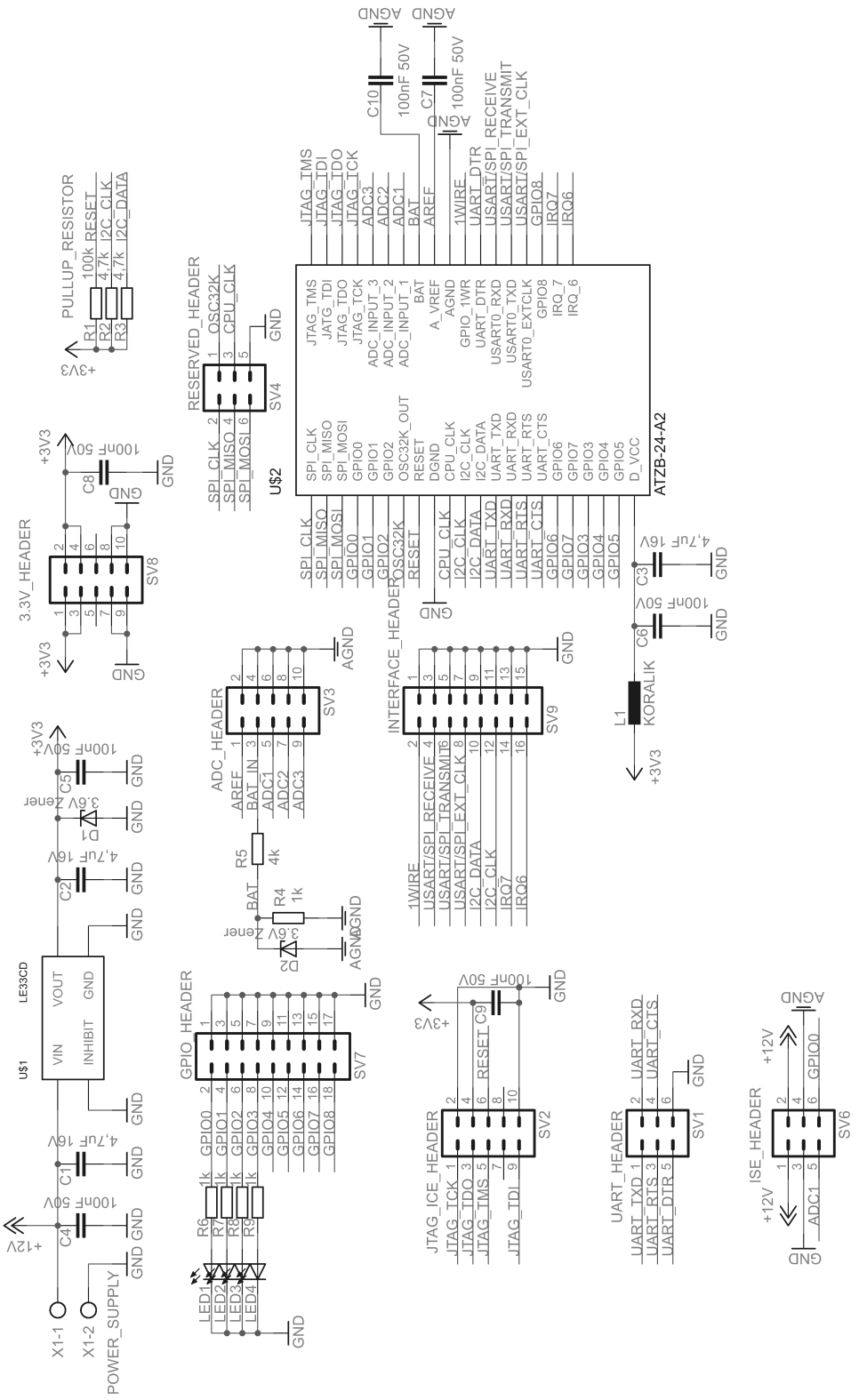
W podrozdziale zawarte są podstawowe informacje o module bezprzewodowym. Więcej informacji można znaleźć w projekcie inżynierskim [5].

Moduł bezprzewodowy zbudowany jest na bazie układu radiowego ATZB-24-A2, który zawiera mikroprocesor ATmega1281 i transceiver AT86RF230. Posiada interfejsy takie jak: UART, I2C, SPI i GPIO. Pełni rolę jednostki sterującej zarówno w module koordynatora jak i urządzenia końcowego. Napięcie zasilania modułu to 5 ÷ 15 V, a poziom napięć logicznych to 3.3 V.

Schematy: ideowy (rysunek 4.3) oraz montażowy (rysunek 4.2), są przydatne podczas podłączania programatora oraz tworzenia modułu koordynatora i modułu urządzenia końcowego.



Rysunek 4.2: Schemat montażowy modułu bezprzewodowego

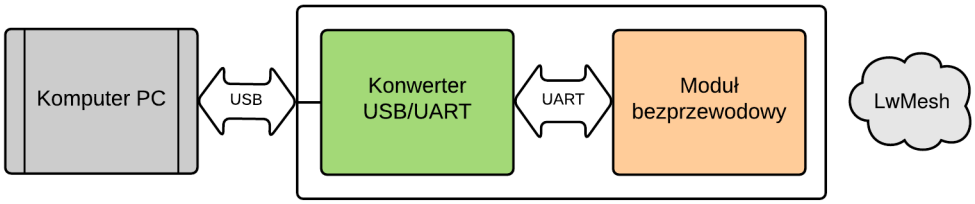


Rysunek 4.3: Schemat ideowy modułu bezprzewodowego

4.4 Moduł koordynatora

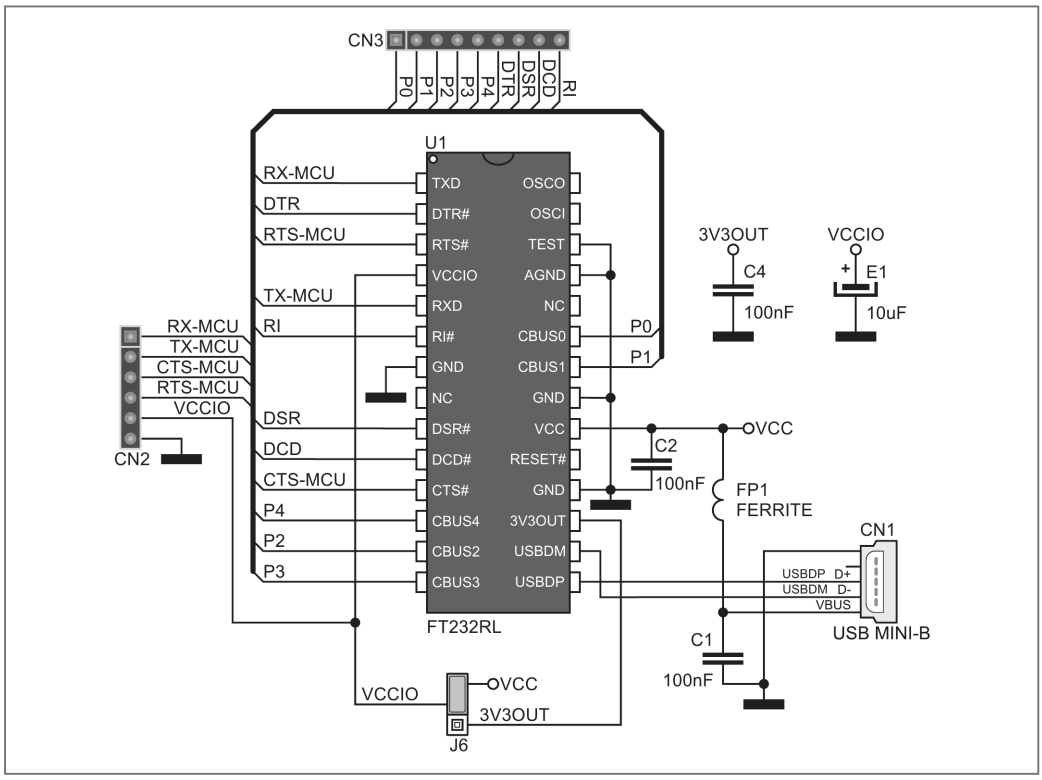
Moduł koordynatora (rysunek 4.4) składa się z dwóch części: konwertera USB/UART oraz modułu bezprzewodowego. Za pomocą konwertera USB/UART możliwa jest dwukierunkowa transmisja danych pomiędzy komputerem PC a modułem bezprzewodowym. Konwerter jest widziany w systemie operacyjnym komputera jako port szeregowy. Moduł bezprzewodowy służy do dwukierunkowej transmisji danych pomiędzy interfejsem UART a modułem urządzenia końcowego za pośrednictwem sieci bezprzewodowej LwMesh.

Szczegółowe informacje o zaimplementowanych funkcjach modułu można znaleźć w podrozdziale 6.4.



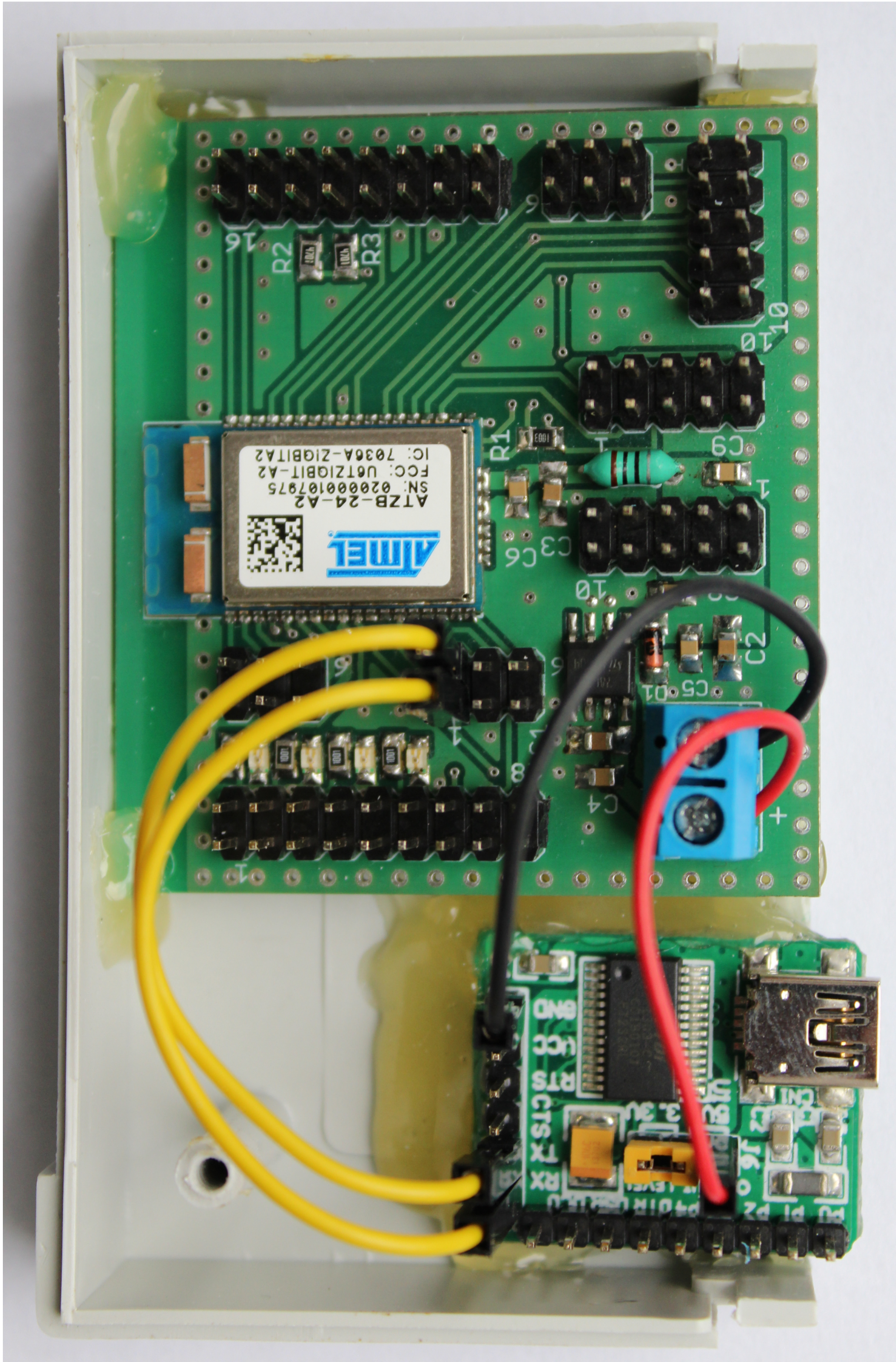
Rysunek 4.4: Schemat blokowy modułu koordynatora

Na rysunku 4.5 przedstawiono schemat ideowy konwertera USB/UART firmy MikroElektronika (symbol mikroe483). Za pomocą zworki J6 można dokonać wyboru poziomu napięć logicznych na 5 V oraz 3,3 V. Ustawiono poziom napięć logicznych na 3,3 V w celu kompatybilności z poziomem napięć logicznych modułu bezprzewodowego. Napięcie 5 V z konwertera zostało doprowadzone do gniazda zasilania modułu bezprzewodowego, co pozwoliło na jego zasilanie bezpośrednio z USB. Linie RX i TX konwertera podłączono do wyprowadzeń UART_TXD i UART_RXD modułu bezprzewodowego (złącze UART_HEADER). Schemat pochodzi z dokumentacji technicznej producenta [32, s. 2].



Rysunek 4.5: Schemat ideowy konwertera

Moduł koordynatora umieszczono w plastikowej obudowie, co znacznie zmniejszyło wymiary urządzenia (rysunek 4.6).

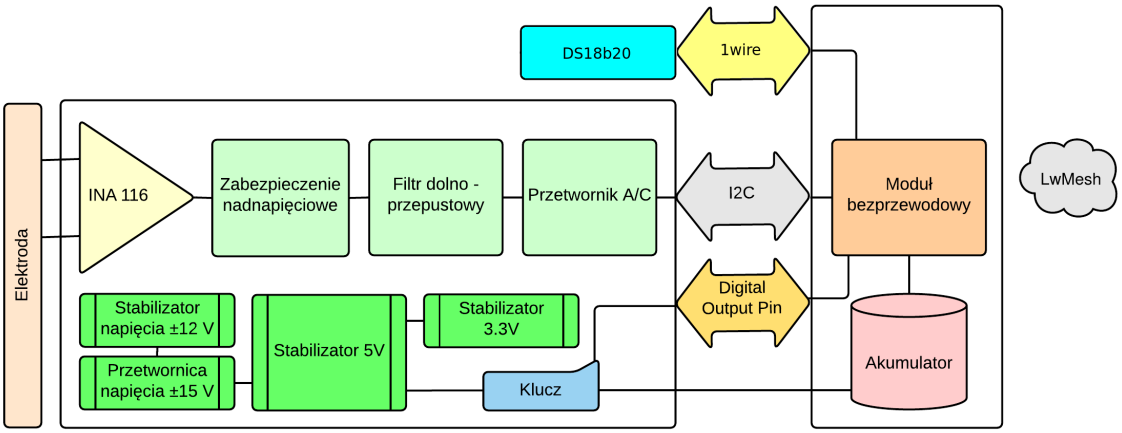


Rysunek 4.6: Moduł koordynatora w obudowie

4.5 Moduł urządzenia końcowego

Moduł urządzenia końcowego (rysunek 4.7) składa się z trzech części: części analogowej, czujnika temperatury oraz modułu bezprzewodowego. Część analogowa służy do wykonywania pomiarów napięcia. Za pomocą magistrali I2C wyniki pomiarów są odczytywane przez moduł bezprzewodowy. Czujnik temperatury wykonuje pomiary temperatury i przesyła je interfejsem 1-Wire do modułu bezprzewodowego. Moduł bezprzewodowy odbiera komendy oraz wysyła dane pomiarowe do modułu koordynatora za pośrednictwem sieci LwMesh. Za pomocą klucza może on zdalnie sterować zasilaniem części analogowej.

Szczegółowe informacje o zaimplementowanych funkcjach modułu można znaleźć w podrozdziale 6.4.



Rysunek 4.7: Schemat blokowy modułu urządzenia końcowego

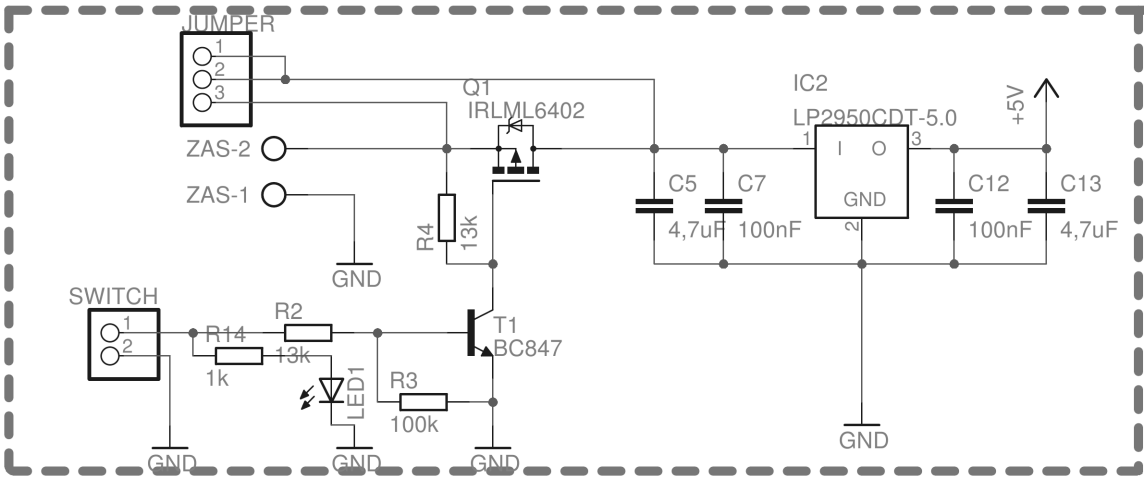
4.5.1 Opis części analogowej

Część analogowa modułu urządzenia końcowego realizuje pomiar potencjału elektrody jonoselektywnej. Na schemacie blokowym (rysunek 4.7) wyróżnić można kilka bloków części analogowej, które zostaną kolejno omówione wraz z przedstawieniem schematów ideowych.

Klucz + Stabilizator 5 V

Złącze SWITCH, tranzystor bipolarny T1, MOSFET Q1 typu p oraz elementy bierne (R2, R3, R4) tworzą klucz umożliwiający włączanie i wyłączanie napięcia zasilania, które jest podawane na zaciski złącza ZAS. Dioda LED1 sygnalizuje obecność sygnału sterującego. Za pomocą złącza JUMPER można bezpośrednio podać napięcie zasilające na stabilizator IC2 (z pominięciem klucza). MOSFET IRLML6402 (Q1) charakteryzuje się bardzo niską rezystancją dren-źródło R_{DSon} wynoszącą 0,135Ω przy napięciu bramka-źródło V_{GS} równym -2,5 V. Oznacza to, że rezystancja klucza jest niska, a sterowanie kluczem jest możliwe dla niskich napięć zasilania np. 5 V.

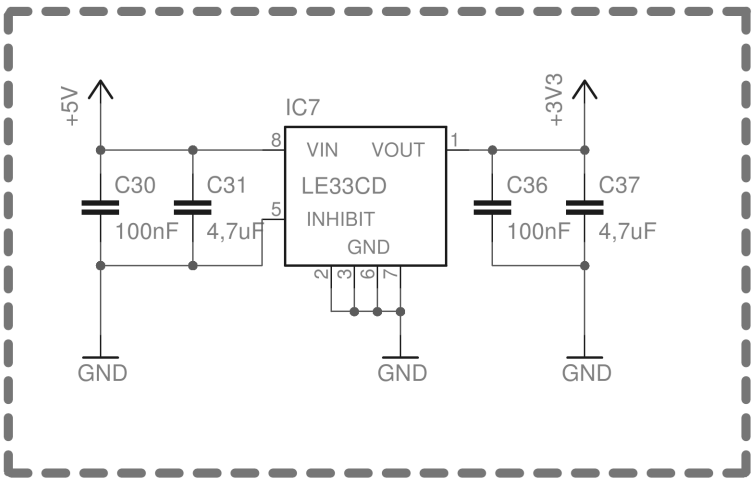
Napięcie zasilające jest stabilizowane za pomocą stabilizatora LDO LP2950CDT-5.0 (IC2), który charakteryzuje się niskim spadkiem napięcia (Dropout voltage) i umożliwia pracę układu już od napięcia zasilania 5,5 V. Prąd maksymalny układu to 100 mA, a w momencie przekroczenia prądu maksymalnego włączane jest zabezpieczenie przeciwzwarciowe oraz termiczne. Napięciem 5 V zasilana jest przetwornica napięcia ±15 V, część analogowa przetwornika ADC, układu napięcia odniesienia, układu zabezpieczenia nadnapięciowego oraz układu filtru dolnoprzepustowego.



Rysunek 4.8: Układ stabilizatora napięcia 5 V z kluczem sterującym

Stabilizator 3,3 V

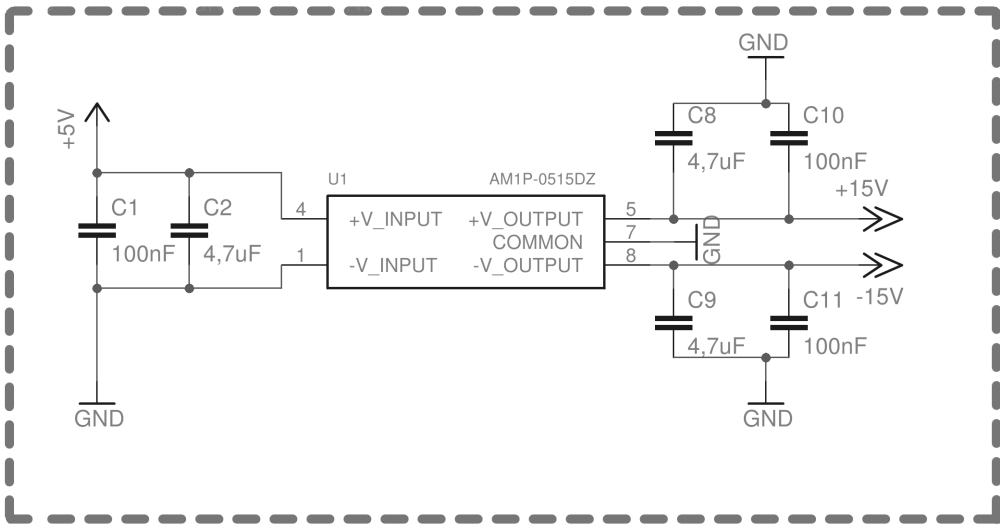
Napięcie 3,3 V jest wymagane do zasilania części cyfrowej przetwornika analogowo-cyfrowego. Obniżenie napięcia 5 V na 3,3 V jest realizowane przez układ LE33CD (IC7). Jest to stabilizator LDO (low dropout) charakteryzujący się niskim spadkiem napięcia, typowo 0,2 V dla prądu 100 mA.



Rysunek 4.9: Układ stabilizatora napięcia 3,3 V

Przetwornica napięcia ±15 V

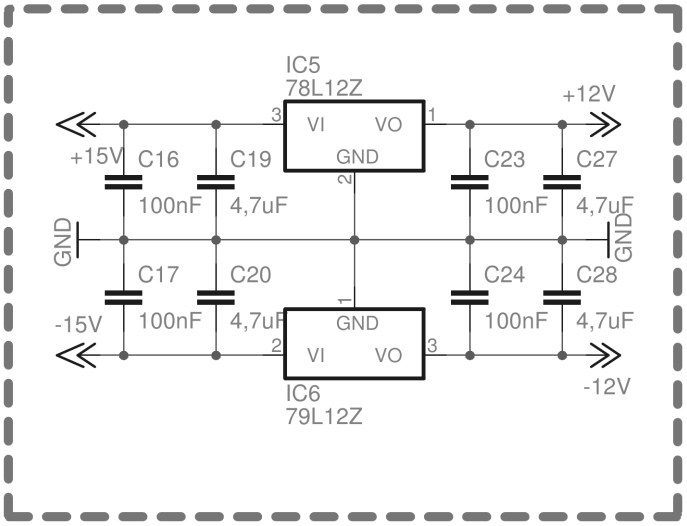
Wzmacniacz pomiarowy INA116 wymaga zasilania symetrycznego, dlatego konieczne jest wykorzystanie przetwornicy napięcia AM1P-0515DZ (U1). Przetwornica zasilana jest napięciem 4,5 ÷ 5 V, a na wyjściu dostarcza napięcia symetrycznego ±15 V. Maksymalny prąd wynosi ±34 mA, maksymalne obciążenie pojemnościowe ±100 uF, a sprawność dochodzi do 81%. Warto wspomnieć, że częstotliwość pracującej przetwornicy to 80 kHz a tętnienia napięcia wyjściowego 100 mV peak-peak.



Rysunek 4.10: Układ przetwornicy napięcia ±15 V

Stabilizator napięcia ±12 V

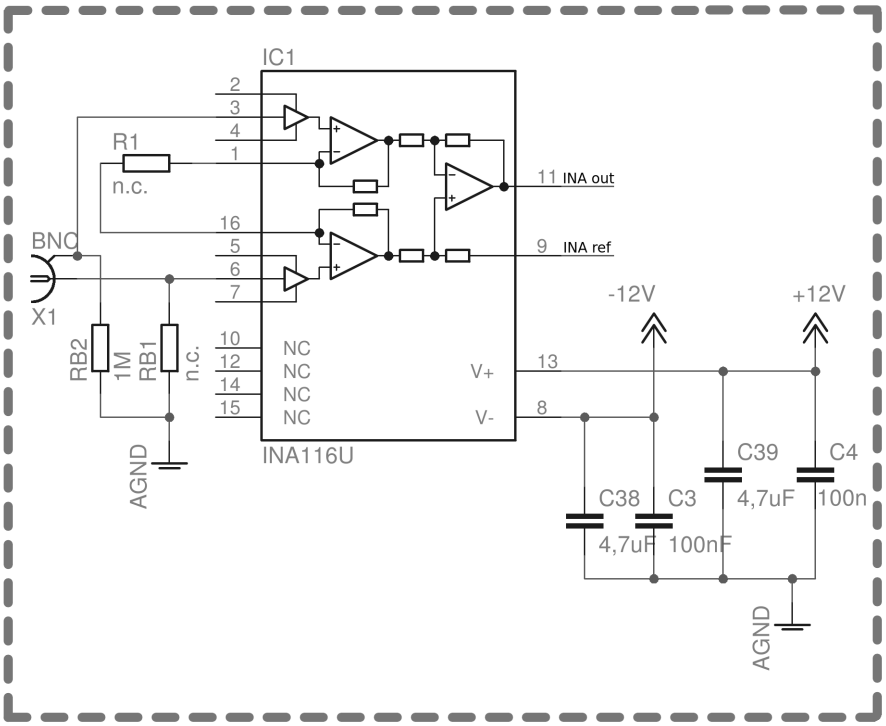
W celu eliminacji tętnień napięcia symetrycznego z przetwornicy zastosowano symetryczny stabilizator napięcia. Stabilizatory napięcia dodatniego (IC5) oraz ujemnego (IC6) posiadają maksymalny prąd wyjściowy 100 mA. Zmiana prądu obciążenia powoduje zmianę napięcia wyjściowego układów. Współczynnik stabilizacji prądowej (load regulation) jest zadowalający i wynosi 0,01% $\frac{V_o}{mA}$.



Rysunek 4.11: Układ stabilizatora napięcia ±12 V

INA116

W pomiarach potencjału elektrod jonoselektywnych wymagane jest, aby impedancja wejściowa wzmacniacza pomiarowego była bardzo wysoka. Jeżeli warunek ten nie jest spełniony to impedancja wzmacniacza obciąża prądowo elektrodę, co skutkuje spadkiem wartości jej potencjału i w efekcie dużym błędem pomiaru.
W celu zapewnienia wysokiej impedancji wejściowej układu pomiarowego, zdecydowano się na zastosowanie wzmacniacza pomiarowego INA116 (IC1), którego impedancja wejściowa wynosi ponad $10^{15}\Omega$. Wejście wzmacniacza wyposażono w gniazdo BNC (X1), do wygodnego podłączania elektrod jonoselektywnych. Rezystorem R1 ustala się wartość wzmocnienia, brak rezystora oznacza wzmocnienie równe 1. Do nóżki 9 układu (INA ref) podłączono napięcie referencyjne 1,25 V służące do przeniesienia zakresu napięć wejściowych z ±1,25 V (użyteczny zakres napięć wejściowych) do zakresu napięć 0 - 2,5 V (użyteczny zakres napięć wyjściowych dla przetwornika ADC) dostępnych na nóżce 11 (INA out). Rezystor RB2 służy do prawidłowego spolaryzowania wejść prądem wejściowym (input bias current). Bez niego wartość napięcia wejściowego mogłaby przekroczyć dopuszczalny zakres napięć wejściowych, co w efekcie spowodowałoby nasycenie wzmacniacza.



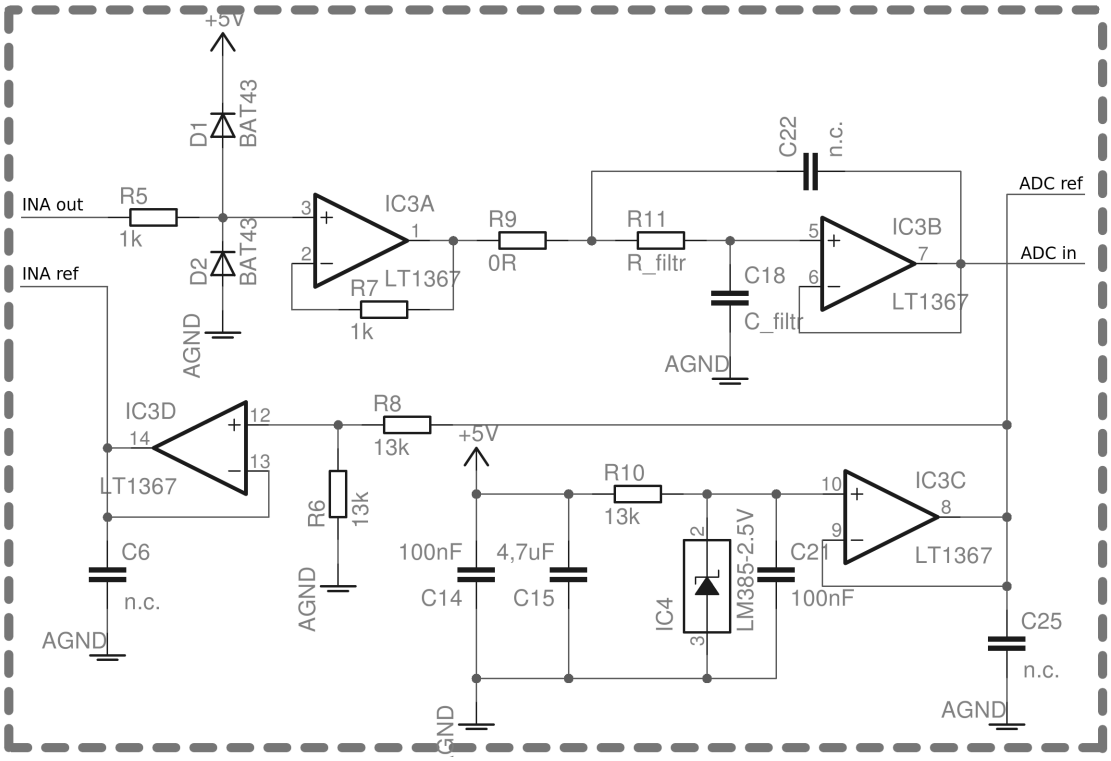
Rysunek 4.12: Układ wzmacniacza pomiarowego

Zabezpieczenie nadnapięciowe + Filtr dolnoprzepustowy + Napięcie odniesienia

Elementy R5, R7, D1, D2 i IC3A tworzą zabezpieczenie przed zbyt wysokim napięciem wyjściowym z wzmacniacza pomiarowego (INA out). Jeśli napięcie przekroczy wartość 5 V + 0,3 V (spadek napięcia na diodzie Schottkiego D1), to dioda D1 zacznie przewodzić powodując przepływ prądu przez rezystor R5 i wytworzenie się spadku napięcia na tym rezystorze. Ponieważ spadek napięcia na R5 jest ujemny względem napięcia z układu wzmacniacza pomiarowego, to wypadkowe napięcie pojawiające się na wejściu nieodwracającym wtórnika (IC3A) będzie zawsze wynosiło 5 V + 0,3 V. Rezystor R7 kompensuje spadek napięcia na rezystorze R5 pochodzący od przepływu prądu wejściowego wtórnika. Niestety nie można skompensować spadku napięcia na R5 pochodzącego od przepływu wypadkowej prądów wstecznych diod D1 i D2.
Tego typu zabezpieczenie jest niezbędne, gdyż w przypadku pozostawienia niepodłączonego gniazda BNC wzmacniacz pomiarowy może się nasycić i utrzymywać na wyjściu wartości napięć ±12 V.

Elementy R9, R11, C18, C22 oraz wzmacniacz operacyjny IC3B tworzą układ aktywnego filtru RC. W zależności od potrzeby można zrealizować filtr pierwszego rzędu (zwora zamiast rezystora R9, brak kondensatora C22), lub drugiego rzędu w topologii Sallen-Key (montowane wszystkie elementy).
Filtr dolnoprzepustowy służy do tłumienia zakłóceń wysokoczęstotliwościowych pojawiających się na wejściu wzmacniacza pomiarowego. Źródłem tych zakłóceń mogą być urządzenia elektryczne pracujące w pobliżu układu, sieć elektryczna, pola elektromagnetyczne i elektrostatyczne indukujące zakłócenia w przewodach doprowadzeniowych. Aby uzyskać częstotliwość graniczną filtru wynoszącą 0,34 Hz, należy wmontować R11 o wartości 100 kΩ oraz C18 o wartości 4,7 uF.

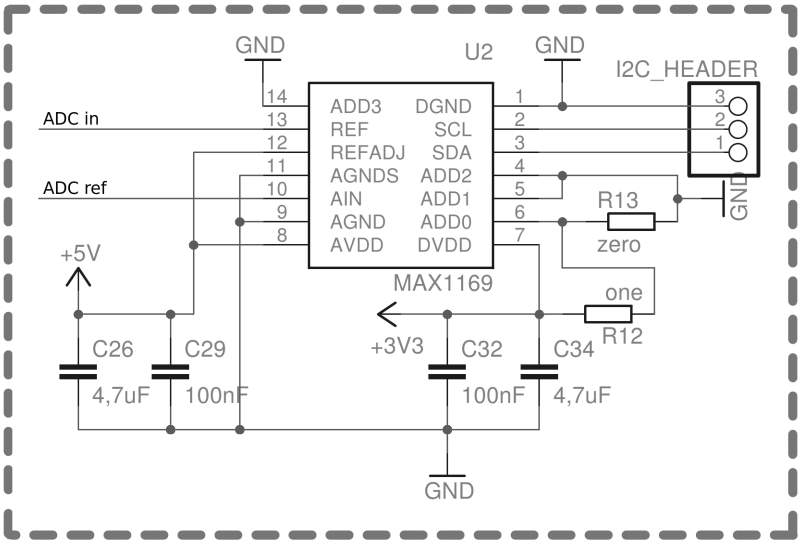
Element IC4 jest scalonym źródłem napięcia odniesienia 2,5 V o tolerancji 1% oraz temperaturowym współczynniku zmian napięcia 80 $\frac{\text{ppm}}{^{\circ}\text{C}}$. Źródło to działa stabilnie dla niewielkich prądów obciążenia, dlatego zastosowano wtórniki napięcia IC3C i IC3D, aby nie obciążać prądowo źródła. Wtórnik IC3C dostarcza napięcie referencyjne (2,5 V) dla przetwornika ADC (ADC ref), a wtórnik IC3D dostarcza połowę wartości tego napięcia (1,25 V) dla wzmacniacza pomiarowego (INA ref).
Kondensatorów C6 oraz C25 nie należy montować, gdyż obciążają one pojemnościowo wyjścia wzmacniaczy, co może powodować ich niestabilną pracę (więcej w podrozdziale 8.4.2).



Rysunek 4.13: Układ zabezpieczenia nadnapięciowego z filtrem dolnoprzepustowym

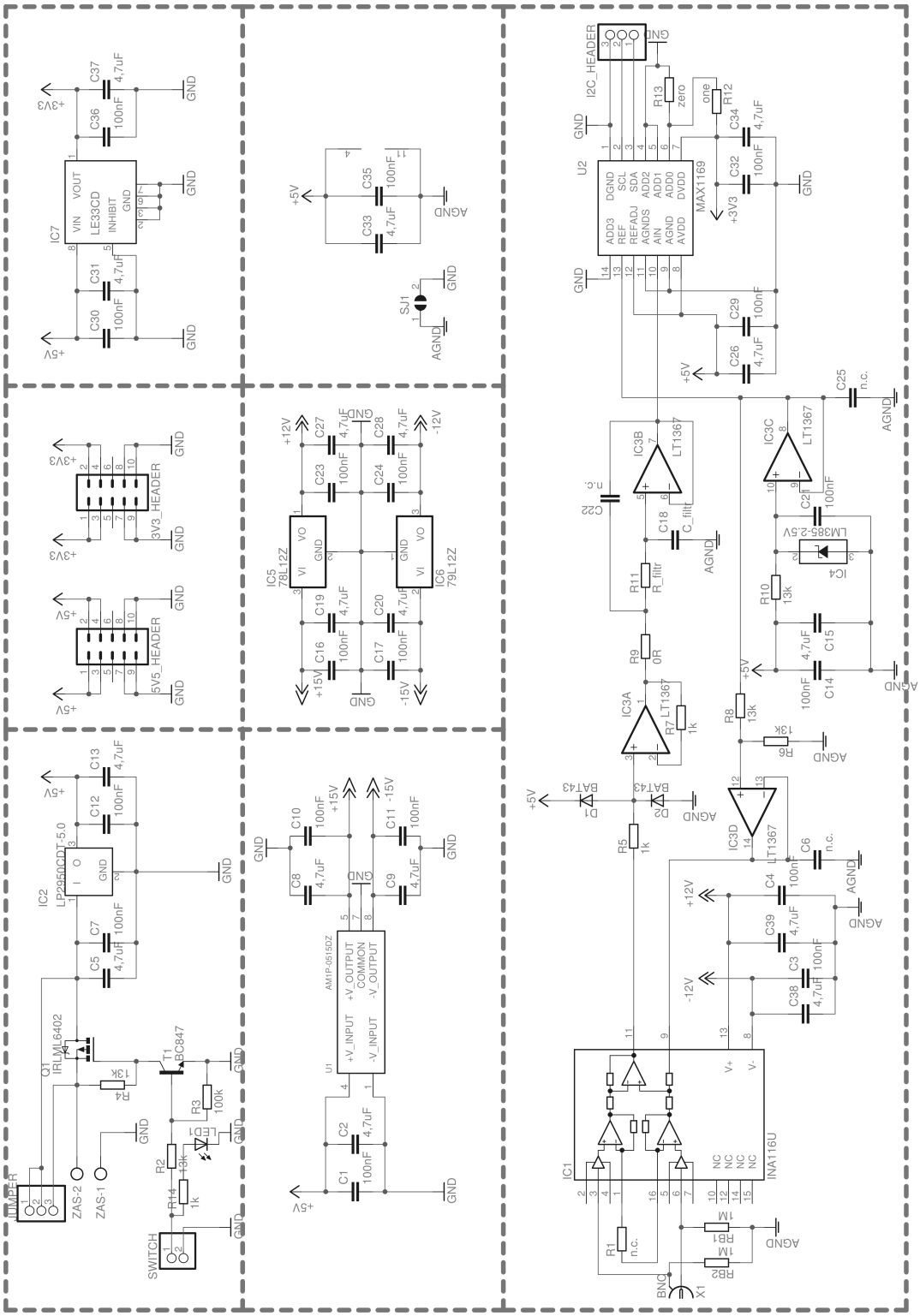
Przetwornik A/C

Napięcie mierzone przez wzmacniacz pomiarowy, po przefiltrowaniu w układzie filtra dolnoprzepustowego trafia na wejście przetwornika analogowo-cyfrowego MAX1169 (ADC ref). Jest to przetwornik 16-bitowy z sukcesywną aproksymacją. Część analogowa zasilana jest napięciem 5 V, a część cyfrowa napięciem 3,3 V. Pozwala to na podłączenie do części cyfrowej układu mikroprocesorowego, który odczytuje wartości mierzonych napięć za pomocą interfejsu cyfrowego I2C wyprowadzonego na złączu I2C_HEADER. Za pomocą rezystorów R12 i R13 możliwe jest ustawienie adresu slave'a.



Rysunek 4.14: Układ przetwornika analogowo-cyfrowego

Schemat ideowy



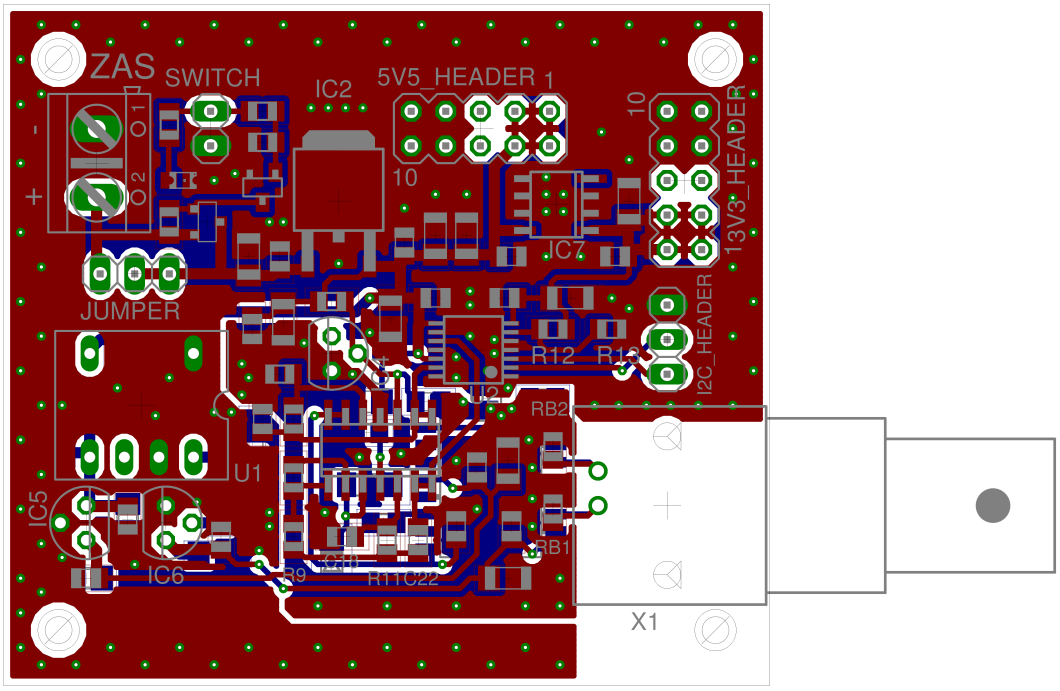
Rysunek 4.15: Schemat ideowy części analogowej

Schemat montażowy

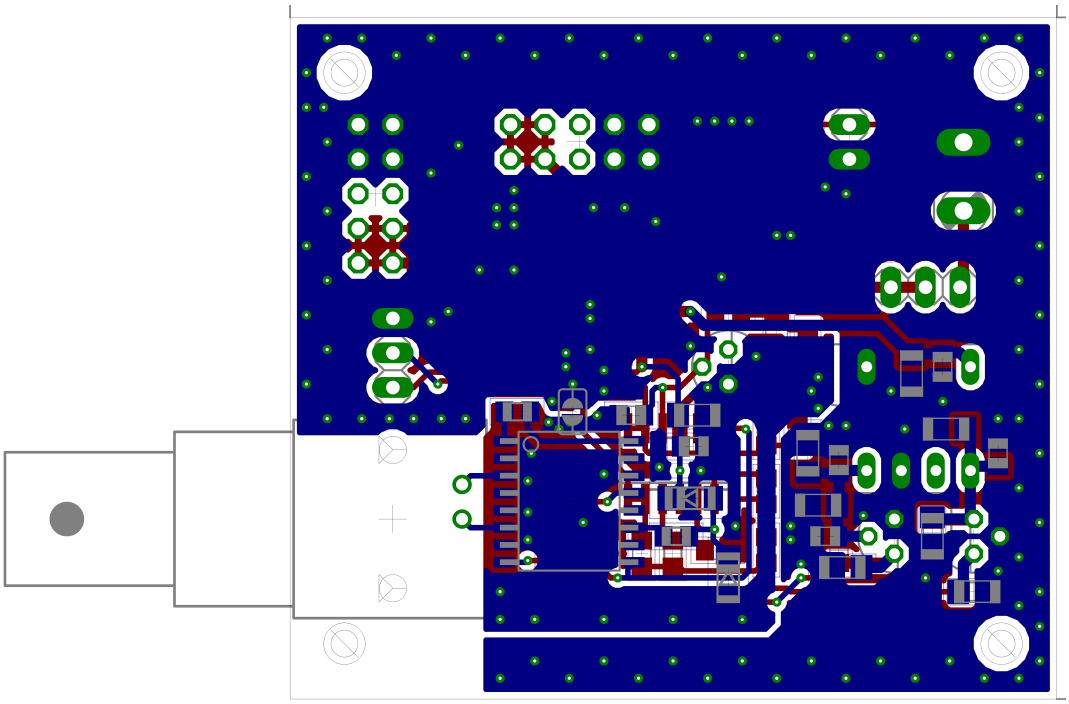
Projekt płytki drukowanej (rysunek 4.16 i 4.17) został wykonany w programie Eagle Light Edition. Zastosowanie technologii dwuwarstwowej oraz elementów SMD pozwoliło na zmniejszenie wymiarów urządzenia do rozmiarów ok. 50 mm x 56 mm. Podczas projektowania płytki drukowanej kierowano się zasadami projektowania:

- unikanie pętli masy, powodującej nierównomierny rozpływ prądów w ścieżkach i płaszczyznach (polygonach);
- tworzenie możliwie krótkich połączeń między elementami, co zmniejsza rezystancję ścieżek;
- odsprzęganie zasilania, czyli dodawanie kondensatorów blokujących (100nF) w celu eliminacji sprzężeń w obwodzie;
- zmniejszanie zakłóceń elektromagnetycznych w obwodzie, zarówno emitowanych przez urządzenie jak i zewnętrznych, poprzez zastosowanie płaszczyzn mas (polygonów) oraz przelotek na brzegach płytki.

Poniżej przedstawiono schematy montażowe części analogowej, warstwę Top i Bottom.



Rysunek 4.16: Schemat montażowy części analogowej - warstwa Top



Rysunek 4.17: Schemat montażowy części analogowej - warstwa Bottom

Podczas projektowania schematu montażowego skorzystano z wygodnego narzędzia jakim jest DRC (Design Rule Check). Jest to narzędzie, które automatycznie sprawdza poprawność projektowanego obwodu, biorąc pod uwagę następujące parametry:

- minimalna odległość między warstwami sygnałowymi (ścieżkami, padami oraz przelotkami);
- minimalny dystans pomiędzy warstwami sygnałowymi a otworami i krawędziami płytki;
- minimalna szerokość ścieżki;
- minimalna średnica i otwór przelotki.

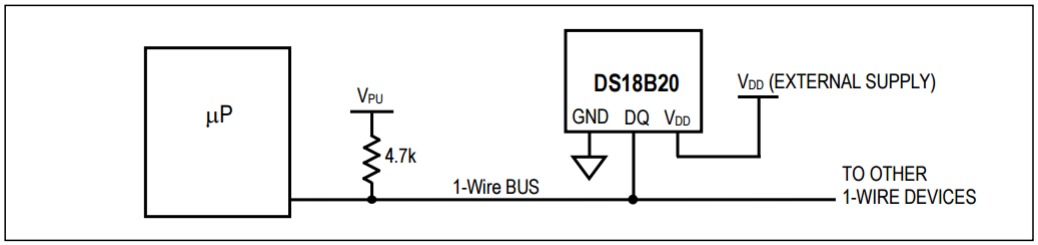
Narzędzie te pozwala wykryć błędy w prowadzeniu ścieżek, zwarcia, nachodzące na siebie elementy, brak połączenia i wiele innych błędów projektowych.

Montaż urządzenia

Montaż elementów na płytce PCB należy rozpocząć od elementów najmniejszych (rezystory, kondensatory), kończąc na elementach największych (układy scalone, złącza). W celu łatwego i szybkiego procesu lutowania, bez ryzyka przegrzania padów i elementów, należy ustawić temperaturę stacji lutowniczej na 350°C i często oczyszczać grot lutownicy za pomocą czyścika. Należy korzystać z pasty lutowniczej, która zapobiega utlenianiu się powierzchni lutowanych i przyśpiesza proces lutowania. Zmontowaną płytkę PCB należy przemyć za pomocą alkoholu izopropylowego, w celu usunięcia zanieczyszczeń oraz pozostałości pasty lutowniczej.

4.5.2 Opis czujnika temperatury

Czujnik temperatury DS18b20 pozwala na pomiar temperatury w zakresie od -10°C do +85°C z dokładnością ±0,5^{o}C. Użytkownik może zmieniać rozdzielczość pomiaru w zakresie 9-12 bitów. Dla domyślnej rozdzielczości 12 bitów czas pomiaru wynosi 750 ms. Czujnik może być zasilany napięciem w zakresie 3 ÷ 5,5 V.
Czujnik posiada linie zasilające (Vdd, GND), które należy podłączyć do napięcia zasilania. Linia DQ jest linią danych i musi ona być podciągnięta do zasilania za pomocą rezystora podciągającego o wartości 4,7 kΩ. Na rysunku 4.18 przedstawiono schemat podłączenia czujnika. Schemat pochodzi z dokumentacji technicznej producenta [29].

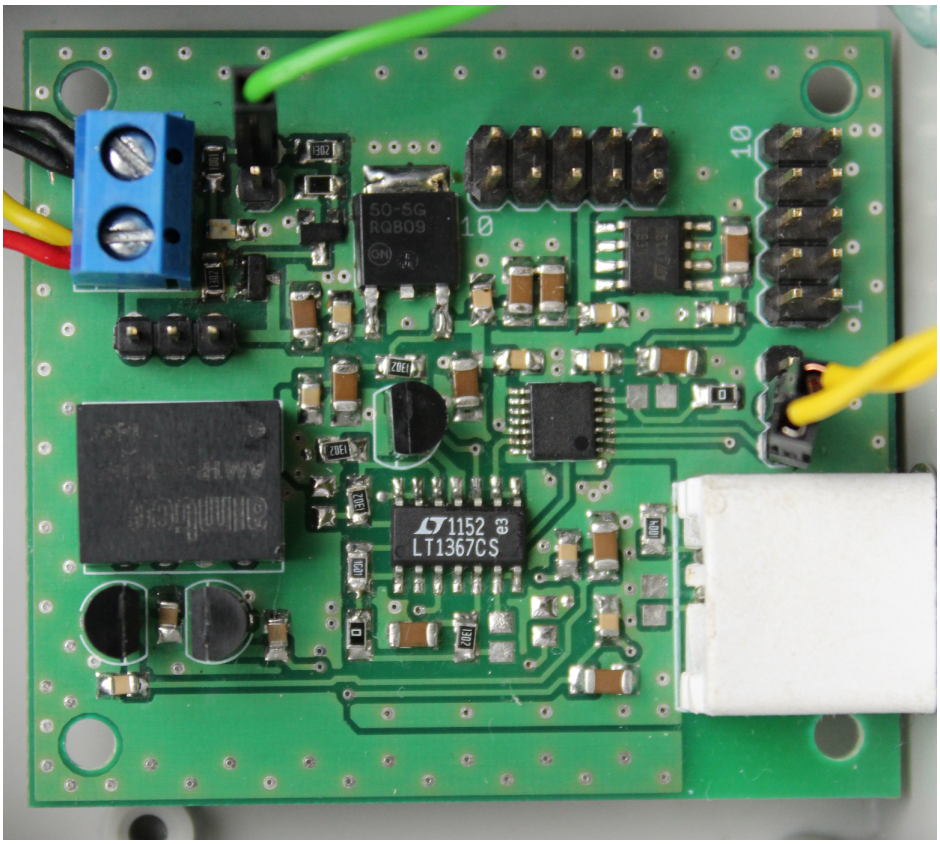


Rysunek 4.18: Schemat podłączenia czujnika DS18b20

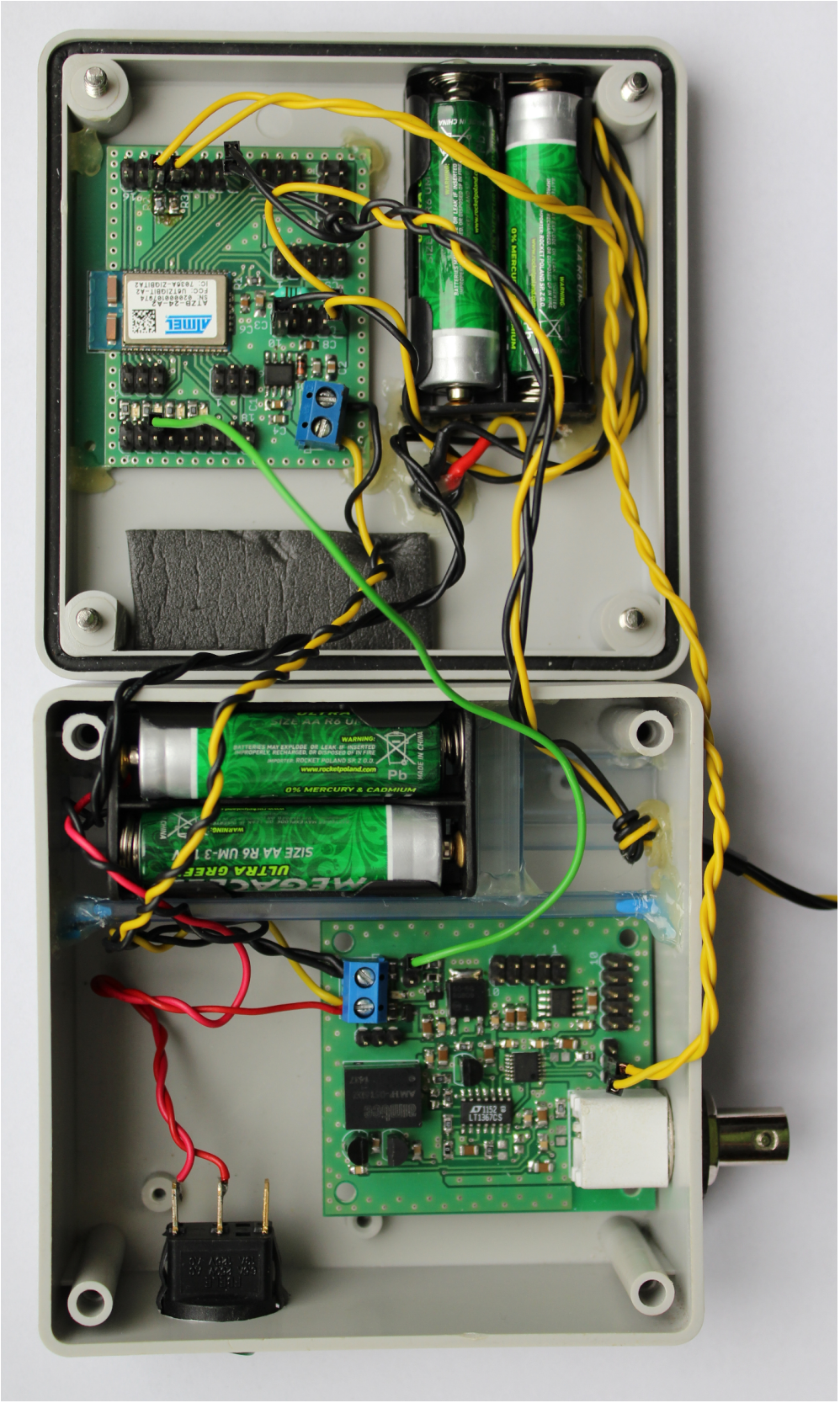
4.5.3 Podłączenie modułu

Część analogowa (rysunek 4.19) oraz moduł bezprzewodowy są zasilane tym samym napięciem zasilania, pochodzącym od pakietu baterii 6xAA. Złącze SWITCH części analogowej sterujące kluczem należy połączyć z pinem 6 (GPIO2) złącza GPIO_HEADER modułu bezprzewodowego. Złącze I2C_HEADER części analogowej będące wyprowadzeniem magistrali I2C należy połączyć z pinem 10 (I2C_DATA) i pinem 11 (I2C_CLK) złącza INTERFACE_HEADER modułu bezprzewodowego. Czujnik temperatury zasila się napięciem 3,3 V dostępnym na złączu 3,3 V_HEADER modułu bezprzewodowego, a linię DQ podłącza się do pinu 2 (1WIRE) złącza INTERFACE_HEADER. Należy pamiętać o dodaniu rezystora podciągającego o wartości 4,7 kΩ.

Moduł urządzenia końcowego (rysunek 4.20) umieszczono w plastikowej obudowie. Wykonano otwór dla gniazda BNC oraz przewodu czujnika temperatury. Dodano przełącznik zasilania oraz diodę LED sygnalizującą pracę układu. Kompaktowa obudowa chroni urządzenie przed zanieczyszczeniami i wilgocią.



Rysunek 4.19: Moduł urządzenia końcowego



Rysunek 4.20: Moduł urządzenia końcowego w obudowie

Rozdział 5

Oprogramowanie interfejsu elektronicznego

Oprogramowanie interfejsu elektroniczego bazuje na protokole sieciowym (stosie komunikacyjnym) Atmel Lightweight Mesh (LwMesh), który umożliwia utworzenie sieci bezprzewodowej z wykorzystaniem transceiverów Atmel IEEE 802.15.4. Protokół charakteryzuje się łatwością implementacji oraz niewielkim rozmiarem zajmowanym w pamięci Flash oraz RAM mikrokontrolera.

5.1 Założenia projektowe

Po zapoznaniu się z dokumentacją techniczną przystąpiono do sprecyzowania założeń projektowych:

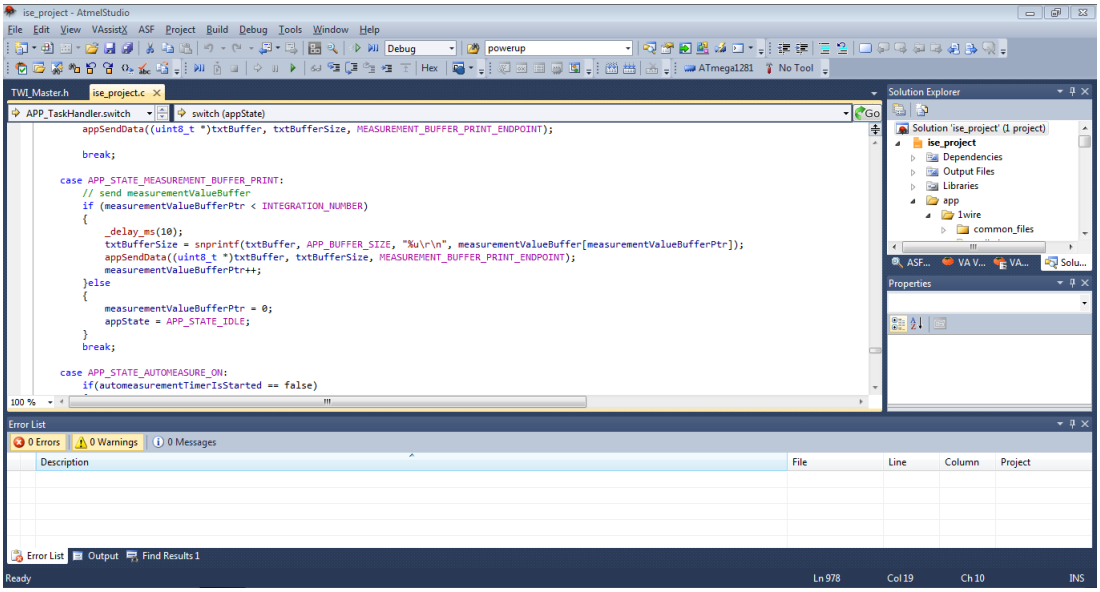
- moduł koordynatora wysyła dane odebrane z portu szeregowego (komendy) do modułu urządzenia końcowego;
- moduł koordynatora odbiera dane z modułu urządzenia końcowego (dane pomiarowe i diagnostyczne) i wysyła je do portu szeregowego;
- moduł urządzenia końcowego odbiera dane z modułu koordynatora (komendy) i podejmuje akcje takie jak: pomiar, zmiana trybu pomiaru, zmiana interwału czasowego pomiaru lub wysłanie informacji diagnostycznych;
- możliwe jest zdalne włączanie i wyłączanie zasilania części analogowej oraz włączanie i wyłączanie nadajnika radiowego w celu realizacji energooszczędnego trybu pomiaru;
- oba moduły wyposażone są w watchdoga, jeżeli program się zawiesi to układ resetuje się po 8 sekundach.

5.2 Środowisko programistyczne Atmel Studio

Oprogramowanie interfejsu elektronicznego zostało napisane w języku C w środowisku Atmel Studio, które jest dostarczane przez producenta mikrokontrolerów AVR - firmę Atmel. Środowisko dostarcza gotowych do użycia narzędzi takich jak: kompilator języka C dla mikrokontrolerów AVR, debugger, refaktoryzacja kodu, symulator i wiele innych. Praca w tym zintegrowanym środowisku programistycznym jest wygodna i szybka.
Dzięki kreatorowi projektu możliwa jest jego konfiguracja krok po kroku:

- wybrać File -> New -> Project;
- wybrać rodzaj szablonu jako „GCC C Executable Project”;
- w polu „Name” wpisać nazwę projektu i nacisnąć przycisk „OK”;
- pojawi się okno „Device Selection”, gdzie z przewijanej listy należy wybrać „ATmega1281” (taki mikrokontroler jest umieszczony w module radiowym ATZB-24-A2), nacisnąć przycisk „OK”;
- pojawi się szablon nowego projektu.

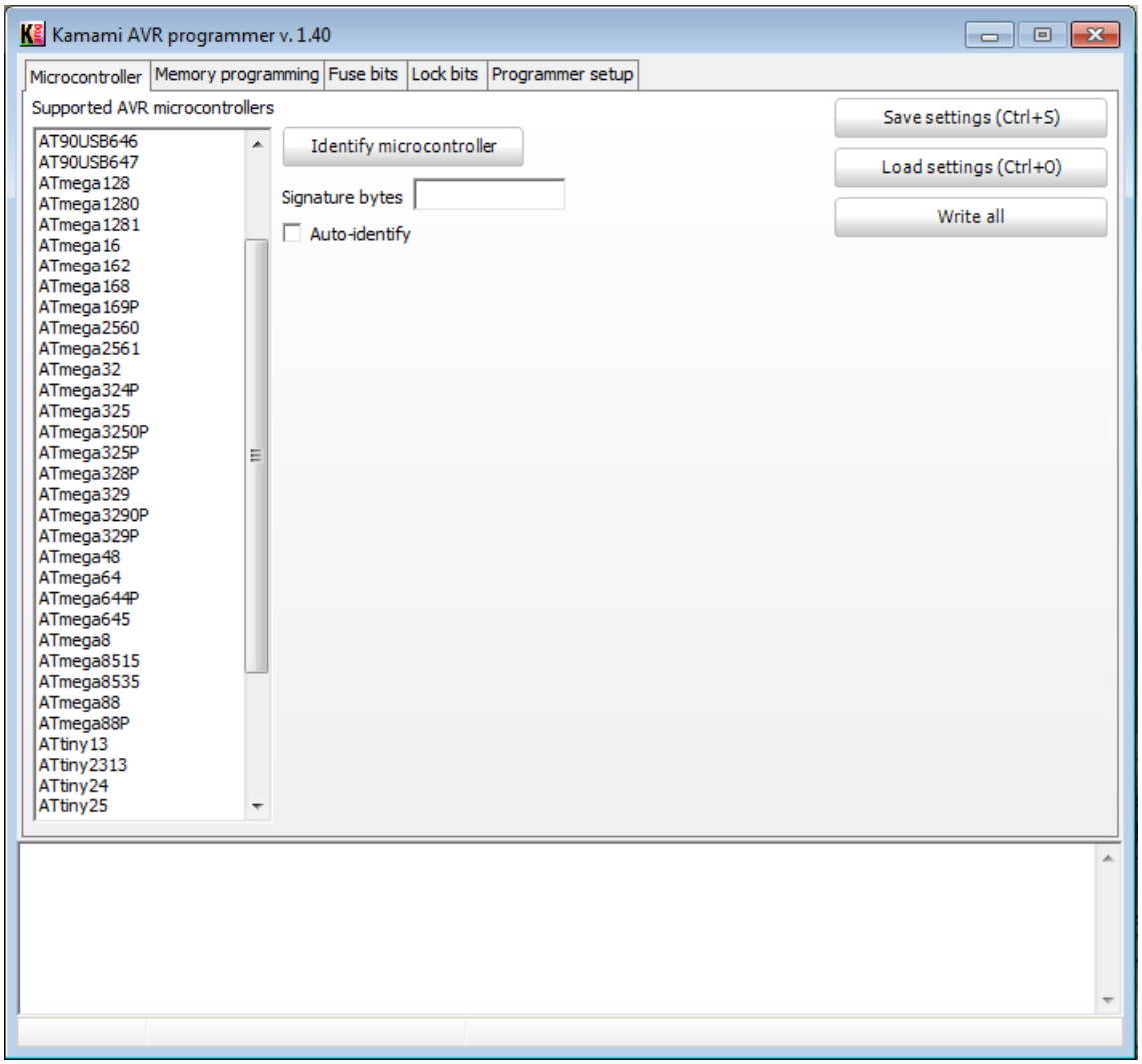
W razie potrzeby deklaracji stałych np. „F_CPU=8000000UL”, należy je zadeklarować w oknie Project Properties -> Toolchain -> AVR/GNU C Compiler -> Symbols.



Rysunek 5.1: Widok okna środowiska Atmel Studio

5.3 Programator KamPROG

Skompilowany wsad należy wgrać do mikrokontrolerów modułu koordynatora i urządzenia końcowego. Do tego celu wykorzystano programator ISP (In-system programming) KamPROG firmy Kamami. Obsługuje on wiele mikrokontrolerów z rodziny AVR, programuje pamięć Flash, EEPROM, Fuse Bity i Lock Bity.



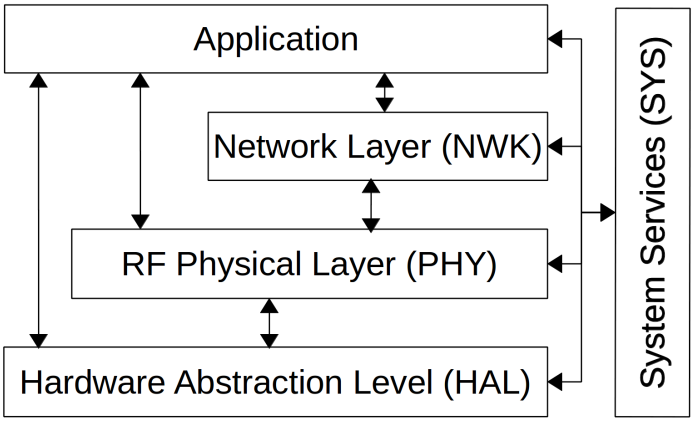
Rysunek 5.2: Widok okna aplikacji Kamami AVR programmer

5.4 Oprogramowanie

Pobrano i rozpakowano archiwum stosu komunikacyjnego Atmel Lightweight Mesh. Wyróżnić można kilka folderów:

- apps - zawiera przykładowe aplikacje wykorzystujące stos;
- doc - zawiera pliki pdf z dokumentacją stosu;
- hal - zawiera pliki źródłowe stosu dla warstwy HAL (Hardware Abstraction Layer) oraz dostarcza podstawowych funkcjonalności zależnych od sprzętu takich jak: obsługa timerów, kontrola trybu uśpienia, dostęp GPIO dla interfejsu radiowego;
- nwk - zawiera pliki źródłowe stosu dla warstwy NWK (Network layer) oraz zapewnia podstawową funkcjonalność stosu;
- phy - zawiera pliki źródłowe stosu dla warstwy PHY (Radio physical layer) oraz dostarcza funkcji obsługujących nadajnik radiowy takich jak: wybór kanału, detekcja mocy sygnału, żądanie wysyłania danych i sygnalizacja odebrania danych;
- sys - zawiera pliki źródłowe współdzielone przez wszystkie warstwy m.in. definicje typów, stałych, funkcje realizujące programowe timery i domyślne konfiguracje parametrów stosu;

Architekturę stosu przedstawiono na rysunku 5.3.



Rysunek 5.3: Architektura stosu LwMesh

Oprogramowanie posiada następującą strukturę folderów i plików:

- stack - folder zawierający stos LwMesh;
- app - folder zawierający biblioteki producenta do obsługi interfejsu 1-Wire oraz I2C;
- config.h - plik konfiguracyjny zawierający definicje stałych wykorzystywanych w oprogramowaniu np. typ urządzenia, adres urządzenia, rozmiar kolejki fifo dla portu szeregowego, czas trwania interwałów czasowych timerów itp.;
- ise_project.c - plik główny oprogramowania bazujący na przykładowej aplikacji Peer2Peer.c, którą można znaleźć w katalogu apps.

W celu skrócenia mniej istotnych linii kodu posłużono się wielokropkiem. Kontynuacja linii kodu korzysta ze znaku „\". Plik główny zawiera oprogramowanie zarówno dla modułu koordynatora jak i urządzenia końcowego. Wybór typu urządzenia odbywa się w pliku config.h, a dzięki użyciu dyrektyw preprocesora możliwe jest pominięcie kompilacji pewnych „niewspólnych" fragmentów kodu.

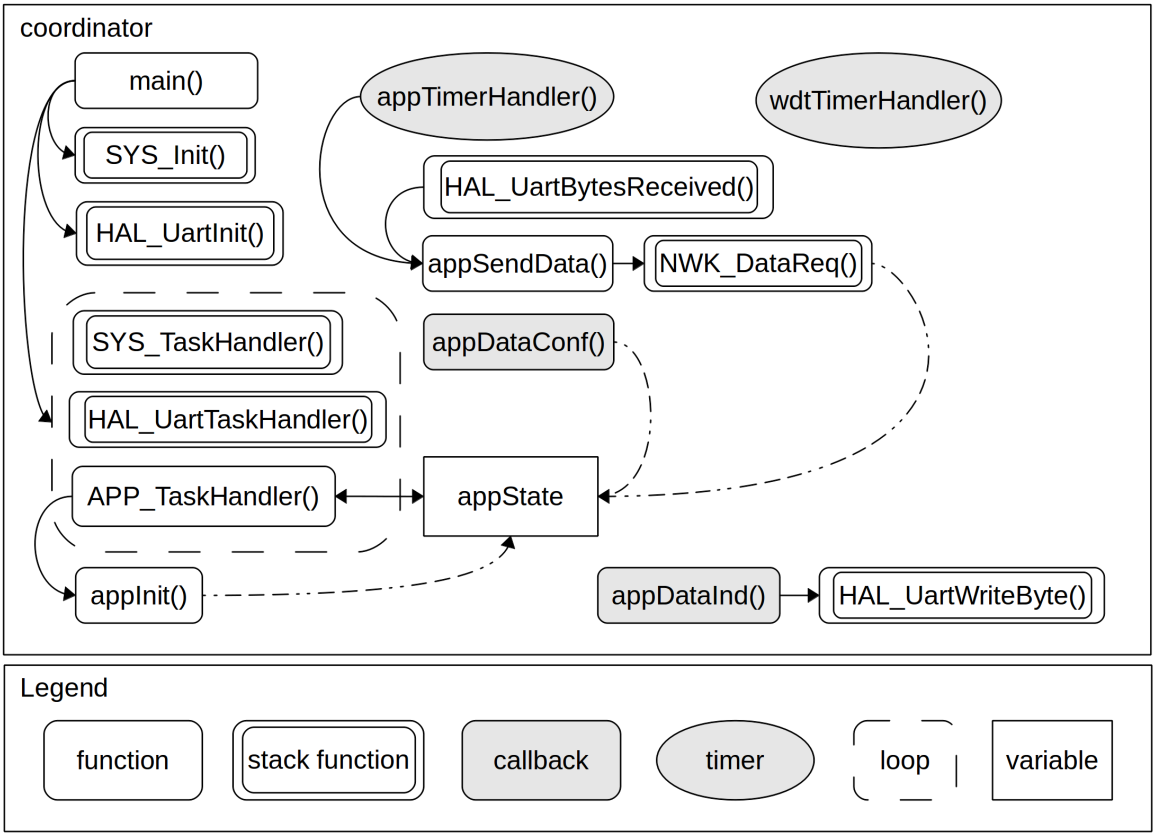
Listing 5.1: Fragment pliku config.h

```
1  //#define DEVICE_MODE 0 // Coordinator
2  #define DEVICE_MODE    1 // End device
3  ...
4  #if DEVICE_MODE == 0
5    #define APP_ADDR     0
6  #else
7    #define APP_ADDR     1
8  #endif
9  #define APP_PANID 0x4567
10 ...
11 #define HAL_UART_TX_FIFO_SIZE 200
12 ...
13 #define AUTOMEASUREMENT_TIMER_INTERVAL  5000 //ms
14 ...
```

5.4.1 Obsługa modułu koordynatora

W celu łatwiejszej analizy kodu źródłowego przedstawiono schemat blokowy oprogramowania koordynatora. Wyróżniamy bloki takie jak:

- function - funkcja napisana przez użytkownika;
- stack function - funkcja dostarczana przez stos LwMesh;
- callback - funkcja obsługująca zdarzenie;
- timer - funkcja wywoływana cyklicznie co określony interwał czasowy;
- loop - pętla wykonująca się tak długo, dopóki warunek pętli jest prawdziwy;
- variable - zmienna.



Rysunek 5.4: Schemat blokowy oprogramowania koordynatora

main()

Funkcja uruchamiana jako pierwsza. Linia 3 wywołuje niskopoziomową inicjalizację sprzętową a linia 4 inicjalizuje port UART mikrokontrolera. Nieskończona pętla (linie 5-10) służy do wywoływania funkcji odpowiedzialnych za obsługę wielozadaniowości. Funkcja SYS_TaskHandler() uruchamia zadania wewnętrzne stosu. Poprawne działanie portu UART zapewnione jest przez funkcję HAL_UartTaskHandler(). Funkcja APP_TaskHandler() jest maszyną stanów i wykonuje zadania zaimplementowane przez użytkownika. W tej funkcji zawarty jest algorytm obsługi modułu koordynatora.

Listing 5.2: Fragment pliku config.h

```
1  int main(void)
2  {
3    SYS_Init();
4    HAL_UartInit(38400);
5    while (1)
6    {
7      SYS_TaskHandler();
8      HAL_UartTaskHandler();
9      APP_TaskHandler();
10   }
11 }
```

APP_TaskHandler()

Funkcja może przyjmować trzy stany w zależności od wartości zmiennej appState. Stan APP_STATE_INITIAL jest stanem domyślnym, w którym po uruchomieniu urządzenia wywoływana jest funkcja inicjalizująca urządzenie appInit().
Stan APP_STATE_IDLE to stan bezczynności, a stan APP_STATE_WAIT_CONF to stan oczekiwania na potwierdzenie odebrania danych wysłanych bezprzewodowo.

Listing 5.3: Funkcja APP_TaskHandler()

```
1  static void APP_TaskHandler(void)
2  {
3    switch (appState)
4    {
5      case APP_STATE_INITIAL:
6        appInit();
7        appState = APP_STATE_IDLE;
8        break;
9      case APP_STATE_IDLE:
10       break;
11     case APP_STATE_WAIT_CONF:
12       break;
13   }
14 }
```

appInit()

Funkcja inicjalizująca urządzenie. Linia 3 włącza watchdoga i ustawia go na 8 sekund. Jeżeli aplikacja się zawiesi to po upływie tego czasu nastąpi reset mikroprocesora. Linie 6-8 ustawiają adres urządzenia, identyfikator sieci i kanał. Linia 11 rejestruje funkcję, która uruchamiana będzie w momencie zajścia zdarzenia odebrania danych. Kolejne instrukcje ustawiają wartość interwału czasowego timerów (linie 13 i 18), tryb pracy (linie 14 i 19) oraz powiązują timery z funkcjami obsługującymi (linie 15 i 20). W celu uruchomienia timerów wywoływana jest funkcja SYS_TimerStart().

Listing 5.4: Funkcja appInit()

```
1  static void appInit(void)
2  {
3    wdt_enable(WDTO_8S); //enable watchdog timer 8s
4    wdt_reset();
5
6    NWK_SetAddr(APP_ADDR);
7    NWK_SetPanId(APP_PANID);
8    PHY_SetChannel(APP_CHANNEL);
9    PHY_SetRxState(true);
10   ...
11   NWK_OpenEndpoint(APP_ENDPOINT, appDataInd);
12   ...
13   wdtTimer.interval = WDT_TIMER_INTERVAL;
14   wdtTimer.mode = SYS_TIMER_PERIODIC_MODE;
15   wdtTimer.handler = wdtTimerHandler;
16   SYS_TimerStart(&wdtTimer);
17
18   appTimer.interval = APP_FLUSH_TIMER_INTERVAL;
19   appTimer.mode = SYS_TIMER_PERIODIC_MODE;
20   appTimer.handler = appTimerHandler;
21   SYS_TimerStart(&appTimer);
22 }
```

HAL_UartBytesReceived()

Kiedy port szeregowy UART otrzyma nowe dane, uruchamiana jest funkcja HAL_UartBytesReceived(). W pętli for (linie 3-15) odczytywane są kolejne bajty danych (linia 5). Bajty te zapisywane są do buforu appUartBuffer (linie 13-14), a jeżeli jest on całkowicie zapełniony to dane z bufora są wysyłane bezprzewodowo do modułu urządzenia końcowego (linie 6-12) i możliwe jest dalsze zapełnianie bufora nowymi danymi.

Listing 5.5: Funkcja HAL_UartBytesReceived()

```
1  void HAL_UartBytesReceived(uint16_t bytes)
2  {
3    for (uint16_t i = 0; i < bytes; i++)
4    {
5      uint8_t byte = HAL_UartReadByte();
6      if (appUartBufferPtr >= (sizeof(appUartBuffer)/sizeof( \
7      appUartBuffer[0])) )
8      {
9        appSendData(appUartBuffer, appUartBufferPtr, \
10       UART_ENDPOINT);
11       appUartBufferPtr = 0;
12     }
13     if (appUartBufferPtr < sizeof(appUartBuffer))
14       appUartBuffer[appUartBufferPtr++] = byte;
15   }
16 }
```

Timer appTimerHandler()

Za cykliczne wysyłanie danych z portu szeregowego UART do modułu urządzenia końcowego odpowiedzialna jest funkcja appTimerHandler().

Listing 5.6: Timer appTimerHandler()

```
1  static void appTimerHandler(SYS_Timer_t *timer)
2  {
3    appSendData(appUartBuffer, appUartBufferPtr, UART_ENDPOINT);
4    appUartBufferPtr = 0; //buffer empty
5    (void)timer;
6  }
```

appSendData()

Funkcja realizująca wysyłanie danych drogą bezprzewodową. Jej postać w oprogramowaniu dla modułu urządzenia końcowego jest identyczna. Funkcja przyjmuje jako argumenty: wskaźnik na dane do wysłania, rozmiar danych, numer lokalny endpoint (numer zależny od miejsca lokalnego wywołania funkcji). Linia 4 sprawdza czy urządzenie nie jest zajęte oraz czy dane mają niezerowy rozmiar. W linii 7 następuje przekopiowanie danych do bufora appDataReqBuffer, który przechowuje dane do wysłania. Aby wysłać dane wymagana jest specjalna struktura typu NWK_DataReq_t (zmienna appDataReq), która posiada pola takie jak: adres docelowy (linia 9), numer endpoint urządzenia docelowego, numer lokalny endpoint, wskaźnik na bufor z danymi (linia 12), rozmiar danych oraz wskaźnik na funkcję potwierdzającą odebranie wysłanych danych (linia 14). Funkcja NWK_DataReq() jest funkcją stosu i realizuje wysłanie wcześniej przygotowanych danych do urządzenia docelowego.

Listing 5.7: Funkcja appSendData()

```
1  static void appSendData( uint8_t *data, uint8_t size, \
2  uint8_t srcEndpoint)
3  {
4    if (appDataReqBusy || 0 == size)
5      return;
6    HAL_LedToggle(LED_TRANSMIT);
7    memcpy(appDataReqBuffer, data, size);
8    appDataReqBufferSize = size;
9    appDataReq.dstAddr = 1-APP_ADDR;
10   appDataReq.dstEndpoint = APP_ENDPOINT;
11   appDataReq.srcEndpoint = srcEndpoint;
12   appDataReq.data = appDataReqBuffer;
13   appDataReq.size = appDataReqBufferSize;
14   appDataReq.confirm = appDataConf;
15   NWK_DataReq(&appDataReq);
16   appDataReqBusy = true;
17   appState = APP_STATE_WAIT_CONF;
18 }
```

appDataConf()

Po wysłaniu danych do modułu urządzenia końcowego, uruchamiana jest funkcja potwierdzająca wysłanie danych appDataConf(). W przypadku modułu koordynatora, niezależnie od miejsca lokalnego wywołania funkcji appSendData(), przechodzi on zawsze w stan bezczynności (linie 3-11). Jeżeli dane zostały wysłane poprawnie to do portu szeregowego UART wysyłany jest tekst „_sys_ok_” (linie 17-20), jeżeli wystąpił błąd do wysyłana jest informacja „_sys_error_” (linie 21-25).

Listing 5.8: Funkcja appDataConf()

```
1  static void appDataConf(NWK_DataReq_t *req)
2  {
3    switch(req->srcEndpoint)
4    {
5      case UART_ENDPOINT:
6        appState = APP_STATE_IDLE; break;
7      case DEFAULT_IDLE:
8        appState = APP_STATE_IDLE; break;
9      default:
10       appState = APP_STATE_IDLE; break;
11   }
12   appDataReqBusy = false;
13
14   char ok[] = "_sys_ok_\r\n";
15   char error[] = "_sys_error_\r\n";
16   // frame was sent successfully
17   if (NWK_SUCCESS_STATUS == req->status)
18   {
19     for (uint8_t i = 0; i < 9; i++)
20       HAL_UartWriteByte(ok[i]);
21   }else // some error happened
22   {
23     for (uint8_t i = 0; i < 12; i++)
24       HAL_UartWriteByte(error[i]);
25   }
26   (void)req;
27 }
```

appDataInd()

Odebranie danych z modułu urządzenia końcowego powoduje wywołanie funkcji appDataInd(). Jest to sygnalizowane zmianą stanu diody LED (linia 3). Dane umieszczone są w strukturze typu NWK_DataInd_t (zmienna ind). Linie 4-5 wysyłają dane do portu szeregowego UART, aby możliwe było odebranie ich w komputerze PC.

Listing 5.9: Funkcja appDataInd()

```
1 static bool appDataInd(NWK_DataInd_t *ind)
2 {
3   HAL_LedToggle(LED_RECEIVE);
4   for (uint8_t i = 0; i < ind->size; i++)
5     HAL_UartWriteByte(ind->data[i]);
6   return true;
7 }
```

Timer wdtTimerHandler()

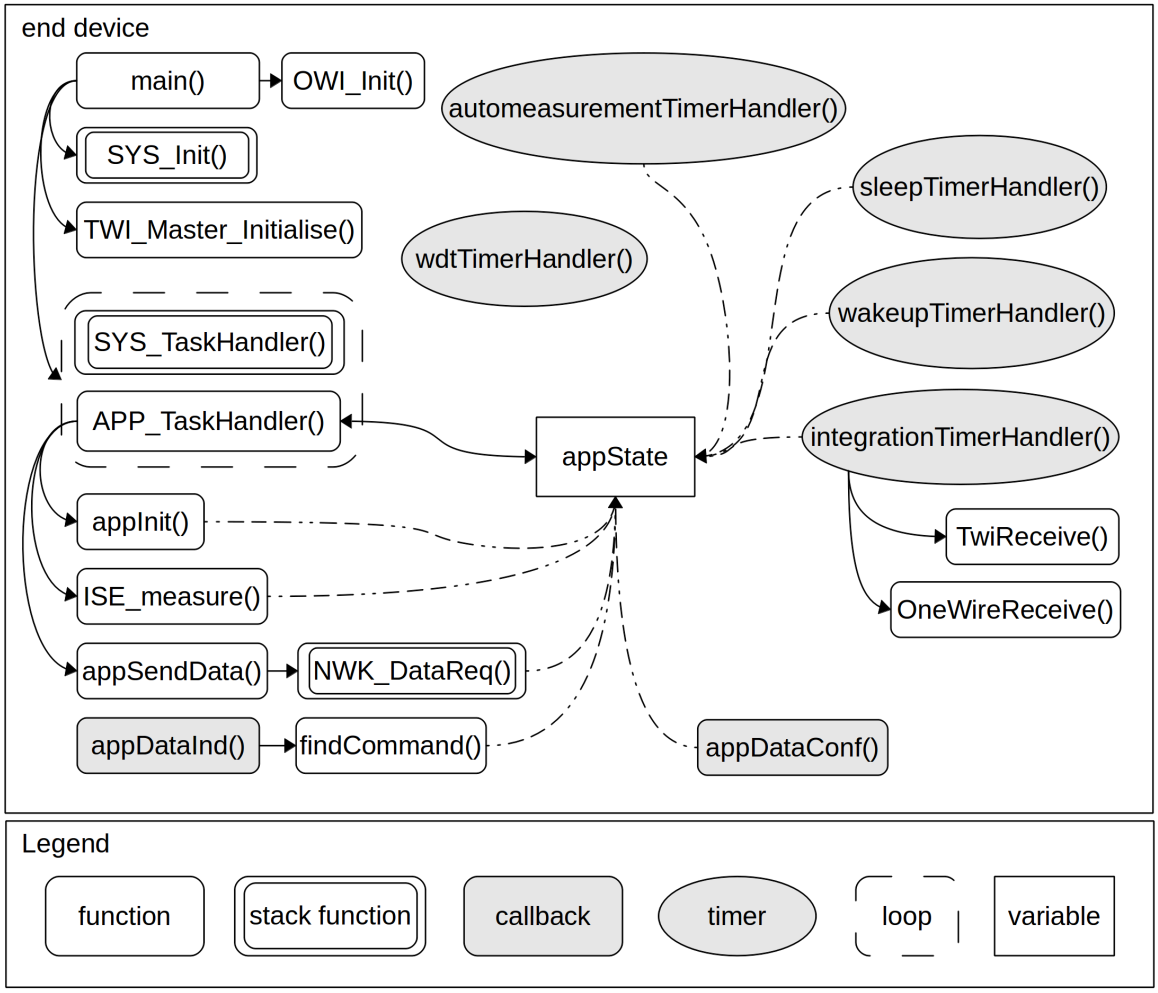
Postać tej funkcji w oprogramowaniu dla modułu urządzenia końcowego jest identyczna. Funkcja obsługuje timer o interwale 1 sekunda, którego zadaniem jest cykliczne resetowanie watchdoga.

Listing 5.10: Timer wdtTimerHandler()

```
1 static void wdtTimerHandler(SYS_Timer_t *timer)
2 {
3   wdt_reset(); //reset watchdog timer
4   (void)timer;
5 }
```

5.4.2 Obsługa modułu urządzenia końcowego

Poniżej przedstawiono schemat blokowy oprogramowania urządzenia końcowego. Funkcje wspólne takie jak appSendData() oraz wdtTimerHandler() zostały omówione we wcześniejszym podrozdziale 6.4.1. Oprogramowanie te posiada nowe funkcje oraz rozszerzony kod źródłowy już istniejących, w porównaniu do oprogramowania modułu koordynatora.



Rysunek 5.5: Schemat blokowy oprogramowania urządzenia końcowego

main()

Oprócz inicjalizacji stosu (linia 3) funkcja przygotowuje do działania magistralę I2C w trybie Master (linia 4) oraz uruchamia programową obsługę interfejsu 1-Wire (linia 5). W pętli wywoływane są funkcje obsługujące stos (linia 8) oraz wykonujące algorytm użytkownika (linia 9).

Listing 5.11: Funkcja main()

```
1  int main(void)
2  {
3     SYS_Init();
4     TWI_Master_Initialise();
5     OWI_Init(BUSES);
6     while (1)
7     {
8        SYS_TaskHandler();
9        APP_TaskHandler();
10    }
11 }
```

appInit()

Różnica pomiędzy tą funkcją, a funkcją w oprogramowaniu modułu koordynatora, polega na ustawieniu innego adresu urządzenia (linia 6) oraz konfiguracji dodatkowych timerów. Timer sleepTimer (linia 22) posiada tryb SYS_TIMER_INTERVAL_MODE, co oznacza jednokrotne wywołanie funkcji obsługującej (linia 24), po upływie zdefiniowanego czasu (linia 22). Kod źródłowy funkcji przedstawiono na listingu 5.12.

Listing 5.12: Funkcja appInit()

```
1  static void appInit(void)
2  {
3    wdt_enable(WDTO_8S); //enable watchdog timer 8s
4    wdt_reset();
5
6    NWK_SetAddr(APP_ADDR);
7    NWK_SetPanId(APP_PANID);
8    PHY_SetChannel(APP_CHANNEL);
9    PHY_SetRxState(true);
10   ...
11   NWK_OpenEndpoint(APP_ENDPOINT, appDataInd);
12   ...
13   wdtTimer.interval = WDT_TIMER_INTERVAL;
14   wdtTimer.mode = SYS_TIMER_PERIODIC_MODE;
15   wdtTimer.handler = wdtTimerHandler;
16   SYS_TimerStart(&wdtTimer);
17
18   wakeupTimer.interval = WAKEUP_TIMER_INTERVAL;
19   wakeupTimer.mode = SYS_TIMER_INTERVAL_MODE;
20   wakeupTimer.handler = wakeupTimerHandler;
21
22   sleepTimer.interval = sleepTimerInterval;
23   sleepTimer.mode = SYS_TIMER_INTERVAL_MODE;
24   sleepTimer.handler = sleepTimerHandler;
25
26   integrationTimer.interval = 1; // 1ms
27   integrationTimer.mode = SYS_TIMER_PERIODIC_MODE;
28   integrationTimer.handler = integrationTimerHandler;
29
30   automeasurementTimer.interval = automeasurementTimerInterval;
31   automeasurementTimer.mode = SYS_TIMER_PERIODIC_MODE;
32   automeasurementTimer.handler = automeasurementTimerHandler;
33 }
```

appDataInd()

Odebrane dane od modułu koordynatora są kopiowane do zmiennej receivedCommand (linia 8). Dane te mogą zawierać komendy użytkownika, dlatego wywoływana jest funkcja findCommand() (linia 12), aby sprawdzić tą ewentualność. Pobierane są także informacje o wskaźnikach jakości sygnału RSSI (linia 6) oraz jakości transmisji LQI (linia 7), wykorzystywane w celach diagnostycznych.

Listing 5.13: Funkcja appDataInd()

```
1  static bool appDataInd(NWK_DataInd_t *ind)
2  {
3    //clear flag that indicates first run of device
4    powerUpFlag = false;
5    HAL_LedToggle(LED_RECEIVE);
6    rssi_lastCommand = ind->rssi;
7    lqi_lastCommand = ind->lqi;
8    memcpy(receivedCommand, ind->data, ind->size);
9    receivedCommandSize = ind->size;
10   //add null terminator to string
11   receivedCommand[receivedCommandSize] = 0;
12   findCommand();
13   return true;
14 }
```

findCommand()

Funkcja sprawdza czy odebrany tekst jest komendą. Jeżeli tak, to zmienia stan aplikacji (zmienna appState), co powoduje podjęcie działań takich jak pomiar, zmiana trybu pomiaru, wysłanie informacji diagnostycznych itp.. Jeśli komenda nie zostanie odnaleziona, to urządzenie odeśle tekst z powrotem do modułu koordynatora (tzw. echo, linia 21).

Funkcja strcmp() porównuje dwa łańcuchy znaków i zwraca wartość 0 jeżeli łańcuchy są identyczne.

Listing 5.14: Funkcja findCommand()

```
1  static void findCommand(void)
2  {
3    if(strcmp((char *)receivedCommand, "ISE:sleep:on") \
4    == 0){
5      appState = APP_STATE_SLEEP_ON;
6    }else if(strcmp((char *) receivedCommand, "ISE:sleep:off") \
7    == 0){
8      appState = APP_STATE_SLEEP_OFF;
9    }else if(strcmp((char *) receivedCommand, "ISE:auto:on") \
10   == 0){
11     appState = APP_STATE_AUTOMEASURE_ON;
12   }else if(strcmp((char *) receivedCommand, "ISE:auto:off") \
13   == 0){
14     appState = APP_STATE_AUTOMEASURE_OFF;
15   }else if(strcmp((char *) receivedCommand, "ISE:meas") \
16    == 0){
17     appState = APP_STATE_MEASURE;
18   }else if
19   ...
20   }else{
21     appState = APP_STATE_ECHO;
22   }
23 }
```

APP_TaskHandler()

Funkcja APP_TaskHandler() jest maszyną stanów, która może przyjmować 24 stany. Omówienie wszystkich stanów nie jest konieczne dla zrozumienia zasady działania maszyny, dlatego zdecydowano się na zaprezentowanie działania maszyny dla trybu pomiaru sleep.

Listing 5.15: Funkcja APP_TaskHandler()

```
1  static void APP_TaskHandler(void)
2  {
3    switch (appState)
4    {
5      case APP_STATE_INITIAL:
6      case APP_STATE_IDLE:
7      case APP_STATE_ISE_MEASURE_START:
8      case APP_STATE_ISE_MEASURE_COMPLETE:
9      case APP_STATE_INTEGRATION:
10     case APP_STATE_SLEEP_ON:
11     case APP_STATE_WAKEUP_AND_SLEEP:
12     case APP_STATE_PREPARE_TO_SLEEP:
13     case APP_STATE_SLEEP:
14     case APP_STATE_SLEEP_OFF:
15     case APP_STATE_WAIT_CONF:
16     case APP_STATE_ECHO:
17     case APP_STATE_MEASURE:
18     case APP_STATE_MEASURE_NOWAIT:
19     case APP_STATE_MEASUREMENT_BUFFER_PRINT:
20     case APP_STATE_AUTOMEASURE_ON:
21     case APP_STATE_AUTOMEASURE:
22     case APP_STATE_AUTOMEASURE_OFF:
23     case APP_STATE_SLEEP_MODE:
24     case APP_STATE_AUTOMEASURE_MODE:
25     case APP_STATE_DIAGNOSE:
26     case APP_STATE_DIAGNOSE_2:
27     case APP_STATE_DIAGNOSE_3:
28     default:
29   }
30 }
```

Stan APP_STATE_SLEEP_ON

Stan przygotowujący urządzenie do przejścia w tryb pomiaru sleep. Jeżeli tryb pomiaru auto jest włączony to następuje jego wyłączenie (linia 11-17). Wyświetlana jest także informacja o włączeniu trybu pomiaru sleep (linie 7-10). Jak widać w linii 9-10, numer lokalny endpoint przechowywany jest w zmiennej SLEEP_ON_ENDPOINT.

Listing 5.16: Stan APP_STATE_SLEEP_ON

```
1  case APP_STATE_SLEEP_ON:
2    if(isSleepModeActive == false)
3    {
4      //it is first measurement
5      sleepFirstMeasurement = true;
6      // _sys_sleep_on_
7      txtBufferSize = snprintf(txtBuffer, APP_BUFFER_SIZE, \
8      "%s\r\n", "_sys_sleep_on_\r\nSleep mode ON" );
9      appSendData((uint8_t *)txtBuffer, txtBufferSize, \
10     SLEEP_ON_ENDPOINT);
11     if (automeasurementTimerIsStarted == true)
12     {
13       automeasurementTimerIsStarted = false;
14       SYS_TimerStop(&automeasurementTimer);
15       //disable power on measurement device
16       HAL_LedOff(LED_MEASURE);
17     }
18   }else
19   {
20     txtBufferSize = snprintf(txtBuffer, APP_BUFFER_SIZE, \
21     "%s\r\n", "Already enabled !");
22     appSendData((uint8_t *)txtBuffer, txtBufferSize, \
23     DEFAULT_SLEEP);
24   }
25   break;
```

appDataConf()

W związku z powyższym, gdy nastąpi potwierdzenie wysłania danych i wywołana zostanie funkcja appDataConf(), stan aplikacji (zmienna appState) zmieni się na APP_STATE_WAKEUP_AND_SLEEP.

Listing 5.17: Funkcja appDataConf()

```
1  static void appDataConf(NWK_DataReq_t *req)
2  {
3    switch(req->srcEndpoint)
4    {
5      case SLEEP_ON_ENDPOINT:
6        appState = APP_STATE_WAKEUP_AND_SLEEP; break;
7      case WAKEUP_AND_SLEEP_ENDPOINT:
8        appState = APP_STATE_IDLE; break;
9      case SLEEP_OFF_ENDPOINT:
10       appState = APP_STATE_IDLE; break;
11     ...
12   }
13 }
```

Stan APP_STATE_WAKEUP_AND_SLEEP

Stan, w którym wykonywany jest pomiar napięcia oraz temperatury (linie 2-3), wybudzany jest nadajnik radiowy (linia 7) oraz wysyłane są dane pomiarowe (linie 13-19). Następnie uruchamiany jest timer odmierzający czas (5 sekund) po którym nadajnik radiowy zostanie uśpiony (timer zmienia stan aplikacji na APP_STATE_PREPARE_TO_SLEEP).

Listing 5.18: Stan APP_STATE_WAKEUP_AND_SLEEP

```
1  case APP_STATE_WAKEUP_AND_SLEEP:
2    if(ISE_measure() == false)
3      break;
4    //no wakeup request if first measurement
5    if(sleepFirstMeasurement == false)
6    {
7      NWK_WakeupReq();
8    }else
9    {
10     sleepFirstMeasurement = false;
11   }
12   // _sys_meas_ voltage temperature
13   txtBufferSize = snprintf(txtBuffer, APP_BUFFER_SIZE,\
14   "_sys_meas_ %u %d\r\nMeasured voltage: %u [bits]\r\n \
15   Measured temperature: %d [*C]\r\n", measurementMeanValue, \
16   measurementTemperature, measurementMeanValue, \
17   measurementTemperature);
18   appSendData((uint8_t *)txtBuffer, txtBufferSize, \
19   WAKEUP_AND_SLEEP_ENDPOINT);
20   SYS_TimerStart(&wakeupTimer);
21   break;
```

ISE_measure()

Pomiar jest wykonywany w dwóch etapach. Najpierw wywoływana jest funkcja ISE_measure() (listing 5.19), która zapisuje kopię bierzącego stanu aplikacji (linia 13), zmienia ten stan na APP_STATE_ISE_MEASURE_START (linia 15) oraz zwaraca wartość false (linia 16). W ten sposób aplikacja przerywa wykonywanie bierzącego kodu (linia 3, listing 5.18).

Listing 5.19: Funkcja ISE_measure()

```
1  static bool ISE_measure(void)
2  {
3    //measurement complete
4    if(appState_measurement_backup == \
5    APP_STATE_ISE_MEASURE_COMPLETE)
6    {
7      //ready for next measurement
8      appState_measurement_backup = APP_STATE_ISE_MEASURE_START;
9      return true;
10   }else //measurement not started
11   {
12     //store app state
13     appState_measurement_backup = appState;
14     //make measurement
15     appState = APP_STATE_ISE_MEASURE_START;
16     return false;
17   }
18 }
```

Stan APP_STATE_ISE_MEASURE_START

Aplikacja znajduje się teraz w stanie APP_STATE_ISE_MEASURE_START (listing 5.20). Dla trybu pomiaru sleep niezbędne jest włączenie zasilania części analogowej modułu urządzenia końcowego (linia 8). Prawidłowe odczyty pomiarów możliwe są po odczekaniu 3 minut od włączenia zasilania (linie 9-13), przez ten okres czasu nie jest możliwa komunikacja z urządzeniem. Następnie uruchamiany jest timer integrationTimer, odpowiedzialny za uśrednianie wyników pomiaru napięcia (linia 16).

Listing 5.20: Stan APP_STATE_ISE_MEASURE_START

```
1  case APP_STATE_ISE_MEASURE_START:
2    if(automeasurementTimerIsStarted == false || \
3    automeasurementFirstMeasurement == true)
4    {
5      if (measurementNoWaitFlag == false)
6      {
7        //enable power on measurement device
8        HAL_LedOn(LED_MEASURE);
9        for(uint8_t i=0; i<= (MEASURE_POWER_ON_DELAY/1000); i++)
10       {
11         _delay_ms(1000); //delay time, 1s
12         wdt_reset();
13       }
14     }
15   }
16   SYS_TimerStart(&integrationTimer);
17   appState = APP_STATE_INTEGRATION;
18   break;
```

Timer integrationTimerHandler()

Na listingu 5.21 przedstawiono kod źródłowy timera. Timer ten wywoływany cyklicznie co 1 milisekundę wykonuje pomiar napięcia za pomocą funkcji TwiReceive() (linia 23). Napięcie jest następnie sumowane pod warunkiem, że nie wystąpił błąd pomiaru (linia 31). Gdy wewnętrzny licznik zliczy 1000 pomiarów to następuje obliczenie średniego napięcia (linie 9-11). Timer zostaje zatrzymany (linia 19) a stan aplikacji zmieniony (linia 20).

Listing 5.21: Funkcja integrationTimerHandler()

```
1  static void integrationTimerHandler(SYS_Timer_t *timer)
2  {
3    uint16_t temp_val = 0;
4    if (integrationCounter >= INTEGRATION_NUMBER)
5    {
6      //if is not error when integrate
7      if(integrationError == false)
8      {
9        measurementMeanValue = (uint16_t) \
10       ((measurementIntegralValue + (INTEGRATION_NUMBER/2)) \
11       / INTEGRATION_NUMBER);
12     }else
13     {
14       measurementMeanValue = TWI_ERROR;
15     }
16     measurementIntegralValue = 0;
17     integrationCounter = 0;
18     integrationError = false;
19     SYS_TimerStop(&integrationTimer); //stop integration timer
20     appState = APP_STATE_ISE_MEASURE_COMPLETE;
21   }else
22   {
23     temp_val = TwiReceive();
24     measurementValueBuffer[integrationCounter] = temp_val;
25     integrationCounter += 1;
26     if (temp_val == TWI_ERROR)
27     {
28       integrationError = true;
29     }else
30     {
31       measurementIntegralValue += temp_val;
32     }
33   }
34   (void)timer;
35 }
```

Stan APP_STATE_ISE_MEASURE_COMPLETE

W tym stanie zostaje wykonany pomiar temperatury za pomocą funkcji OneWireReceive() (linia 2). Przywrócony zostaje stan aplikacji sprzed wywołania funkcji ISE_measure() (linia 7).
W ten sposób w stanie APP_STATE_WAKEUP_AND_SLEEP możliwe jest kontynuowanie wykonywania przerwanego kodu.

Listing 5.22: Stan APP_STATE_ISE_MEASURE_COMPLETE

```
1 case APP_STATE_ISE_MEASURE_COMPLETE:
2   measurementTemperature = OneWireReceive();
3   if(automeasurementTimerIsStarted == false)
4     //disable power on measurement device
5     HAL_LedOff(LED_MEASURE);
6   //restore app state
7   appState = appState_measurement_backup;
8   appState_measurement_backup = APP_STATE_ISE_MEASURE_COMPLETE;
```

Stan APP_STATE_PREPARE_TO_SLEEP

Po wykonaniu pomiarów i odczekaniu 5 sekund (okno czasowe dzięki któremu możliwa jest komunikacja z modułem urządzenia końcowego) aplikacja zmienia swój stan (listing 5.24), gdzie następuje uśpienie nadajnika radiowego (linia 5). Uruchomiony zostaje timer odmierzający czas przebywania w trybie sleep. W tym momencie nie jest możliwa komunikacja z modułem urządzenia końcowego.
Aplikacja przechodzi w stan APP_STATE_SLEEP, który jest po prostu procesem bezczynności. Po odmierzeniu określonego czasu, timer sleepTimer wybudza urządzenie, zmieniając stan aplikacji na APP_STATE_WAKEUP_AND_SLEEP i wykonywany jest kolejny pomiar. Widać więc, że algorytm ten jest zapętlony.

Listing 5.23: Stan APP_STATE_SLEEP

```
1 case APP_STATE_SLEEP:
2   break;
```

Listing 5.24: Stan APP_STATE_PREPARE_TO_SLEEP

```
1  case APP_STATE_PREPARE_TO_SLEEP:
2    if (!NWK_Busy())
3    {
4      isSleepModeActive = true;
5      NWK_SleepReq();
6      SYS_TimerStart(&sleepTimer);
7      appState = APP_STATE_SLEEP;
8    }else
9    {
10     isSleepModeActive = false;
11     // _sys_sleep_off_
12     txtBufferSize = snprintf(txtBuffer, APP_BUFFER_SIZE, \
13     "%s\r\n", "_sys_sleep_off_\r\nSleep mode OFF" );
14     appSendData((uint8_t *)txtBuffer, txtBufferSize, \
15     SLEEP_OFF_ENDPOINT);
16   }
17   break;
```

Stan APP_STATE_SLEEP_OFF

Wyłączenie trybu pomiaru sleep następuje w stanie APP_STATE_SLEEP_OFF. Wyłączony zostaje timer odmierzający czas do ponownego uśpienia nadajnika radiowego (linia 5) oraz wyświetlana jest stosowna informacja (linie 7-10).

Listing 5.25: Stan APP_STATE_SLEEP_OFF

```
1  case APP_STATE_SLEEP_OFF:
2    if(isSleepModeActive == true)
3    {
4      isSleepModeActive = false;
5      SYS_TimerStop(&wakeupTimer);
6      // _sys_sleep_off_
7      txtBufferSize = snprintf(txtBuffer, APP_BUFFER_SIZE, \
8      "%s\r\n", "_sys_sleep_off_\r\nSleep mode OFF" );
9      appSendData((uint8_t *)txtBuffer, txtBufferSize, \
10     SLEEP_OFF_ENDPOINT);
11   }else{
12   ...
13   }
```

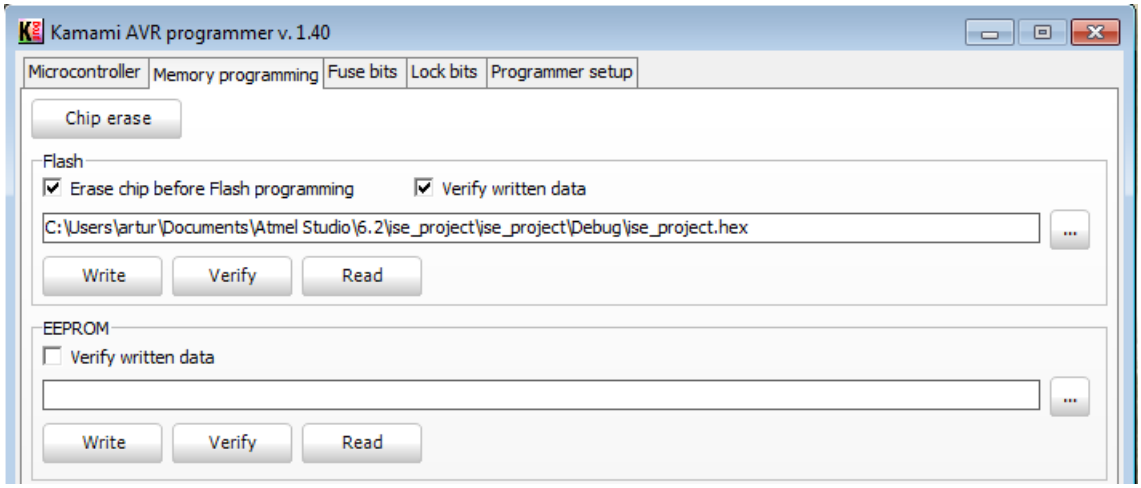
5.4.3 Zaprogramowanie układów

Sposób podłączenia programatora KamPROG do wyprowadzeń ISP modułu bezprzewodowego przedstawiono poniżej.

Tablica 5.1: Sposób podłączenia programatora do modułu

Moduł		Programator
Pin	Złącze	Programator
USART/SPI_RECEIVE	INTERFACE_HEADER	MOSI
USART/SPI_TRANSMIT	INTERFACE_HEADER	MISO
SPI_CLK	RESERVED_HEADER	SCK
RESET	JTAG_ICE_HEADER	RST
VCC	3.3V_HEADER	+3.3 V
GND	3.3V_HEADER	GND

Po podłączeniu programatora do gniazda USB komputera należy uruchomić aplikację Kamami AVR Programmer i przycisnąć przycisk „Identify microcontroller". Mikrokontroler zostanie znaleziony, następnie w zakładce „Memory programming" w sekcji „Flash" wybrać ścieżkę do skompilowanego wsadu. Zaznaczyć opcję „Erase chip before Flash programming" oraz „Verify written data" a następnie nacisnąć przycisk „Write". UWAGA: Fabrycznie nowe mikrokontrolery ATmega1281 mają domyślnie wyłączonego watchdoga oraz częstotliwość zegara podzieloną przez 8. Należy w zakładce „Fuse bits" zaznaczyć pole „WDTON" oraz odznaczyć „CKDIV8".

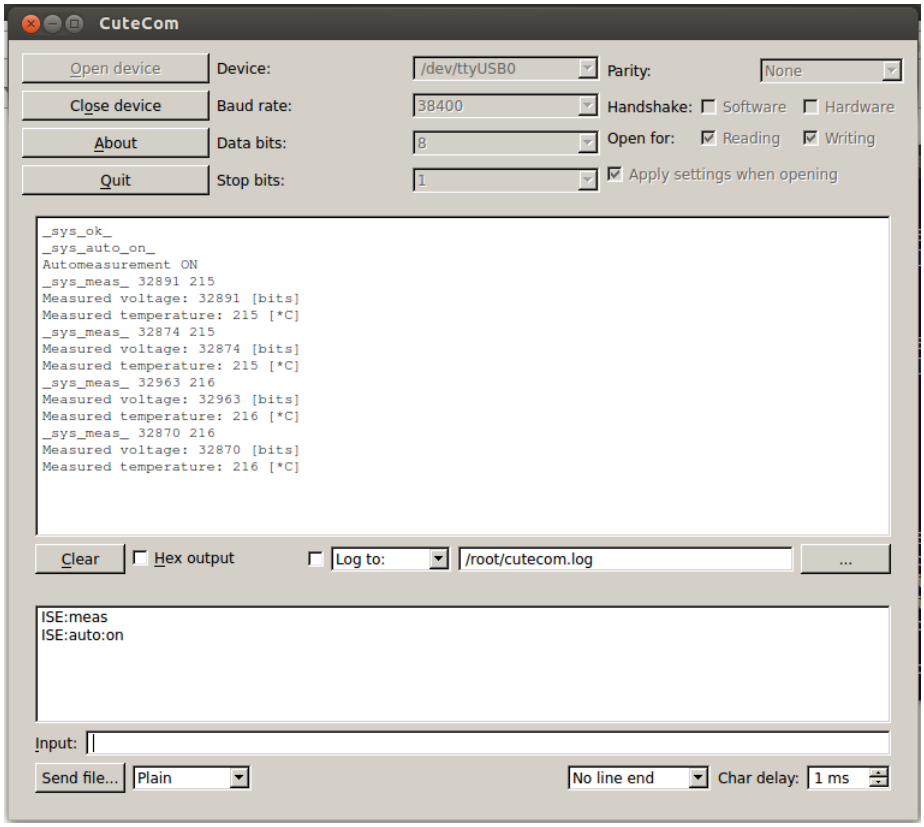


Rysunek 5.6: Widok okna aplikacji Kamami AVR programmer

Rozdział 6

Oprogramowanie użytkowe PC

Dane z modułu koordynatora można odbierać za pomocą terminala portu szeregowego zarówno w systemiach opartych na Linuxie jak i w systemach Windows. Wysyłając komendy tekstowe (szczegółowo opisane w sekcji 7.5.4) możliwe jest konfigurowanie i diagnostyka modułu urządzenia końcowego oraz otrzymywanie wyników pomiarów w postaci tekstowej.



Rysunek 6.1: Widok odbieranych danych za pomocą terminala CuteCom

Wartość zmierzonego napięcia przedstawiona jest w postaci szesnastobitowego rejestru, a wartość temperatury jest liczbą całkowitą. Przetwarzanie tych wartości na napięcie w woltach i temperaturę w stopniach Celsjusza nie odbywa się po stronie modułu urządzenia końcowego.

Tablica 6.1: Przykładowe wartości otrzymane z modułu koordynatora

Wartość odebrana		Wartość po przetworzeniu	
Napięcie	Temperatura	Napięcie	Temperatura
16384	225	-0,625 V	22,5°C
32768	250	0,000 V	25,0°C
43691	333	0,417 V	33,3°C

W związku z powyższym stworzono oprogramowanie użytkowe, którego głównym celem jest przetwarzanie i wizualizacja odebranych danych.

6.1 Założenia projektowe

W czasie prac nad oprogramowaniem użytkowym przyjęto założenia:

- graficzny interfejs użytkownika (GUI),
- łatwa konfiguracja za pomocą dostępnych widżetów,
- wyświetlanie informacji diagnostycznych,
- przetwarzanie odebranych danych i ich wizualizacja w postaci wykresów,
- zapisywanie logów pomiarowych i wykresów,
- wbudowany terminal portu szeregowego,
- możliwość wyznaczenia krzywej wzorcowej elektrody jonoselektywnej (kalibracja),
- wyświetlanie tekstu pomocy dla użytkownika,
- kompatybilność oprogramowania dla wielu systemów operacyjnych.

6.2 Wybór oprogramowania

Główne zasady jakimi kierowano się podczas wyboru elementów oprogramowania były: licencja wolnego oprogramowania, kompatybilność z wieloma systemami operacyjnymi, łatwość i wygoda implementacji.

6.2.1 Python

Spośród wielu języków programowania wybrano język Python ze względu na szereg zalet:

- zwięzła i przejrzysta składnia,
- dynamiczny system typów i automatyczne zarządzanie pamięcią,
- obiektowość i obsługa wątków,
- rozbudowana biblioteka standardowa,
- interpretery Python dostępne są na wiele systemów operacyjnych.

6.2.2 PyGTK

Biblioteka GTK+ służy do tworzenia interfejsu graficznego dla programów komputerowych. Została napisana w języku C, dlatego aby móc korzystać z niej za pomocą języka Python należy skorzystać z nakładki PyGTK. GTK+ jest dostępna na wiele systemów operacyjnych.

6.2.3 Matplotlib

Matplotlib jest biblioteką Pythona służącą do tworzenia wielu rodzai wykresów np. wykresów liniowych, wykresów słupkowych, wykresów 3D, histogramów. Korzystanie z oferowanych funkcji możliwe jest za pomocą skryptów Pythona, a także w interaktywnych powłokach Pythona np. ipython (na podobieństwo korzystania z MATLAB'a).

6.2.4 pySerial

pySerial jest biblioteką Pythona, która umożliwia łatwą obsługę portu szeregowego. Działa na systemach Windows, Linux, BSD i innych.

6.3 Instalacja oprogramowania

Prace nad aplikacją rozpoczęto od aktualizacji istniejących i instalacji brakujących pakietów (Python, PyGTK, Matplotlib, pySerial):

Listing 6.1: Aktualizacja istniejących i instalacja brakujących pakietów

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python python-gtk2
sudo apt-get install python-pip python-matplotlib
sudo pip install --upgrade pyserial
```

Zainstalowano program Glade, który jest kreatorem graficznego interfejsu użytkownika.

Listing 6.2: Instalacja programu Glade

```
sudo apt-get install glade
```

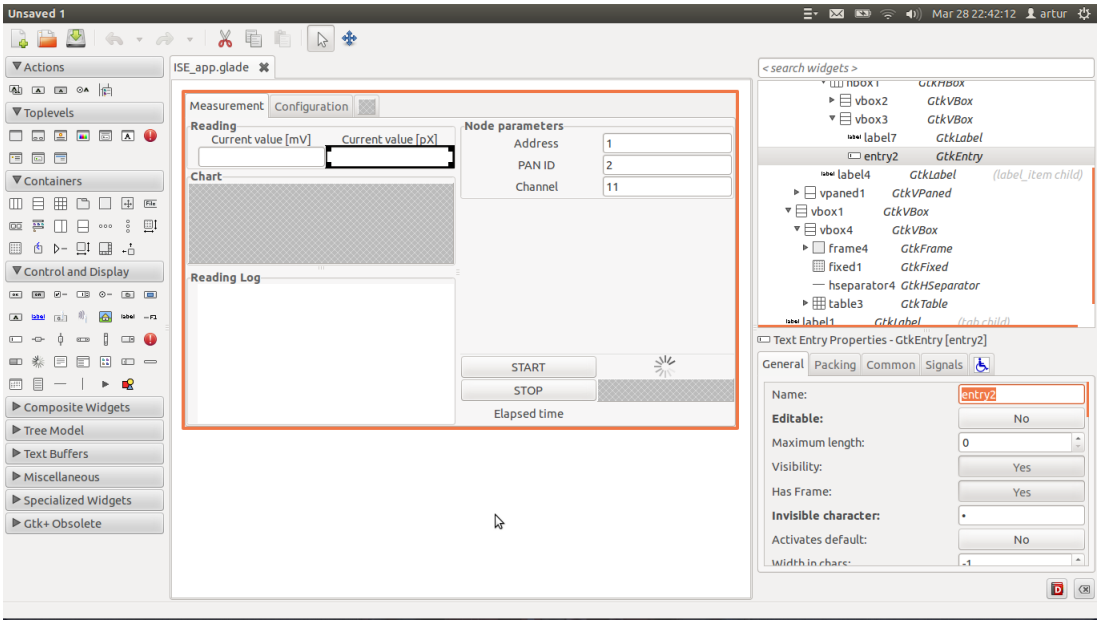
Instalator środowiska programistycznego pyCharm należy pobrać ze strony: *https://www.jetbrains.com/pycharm/download/index.html*

Listing 6.3: Instalacja programu pyCharm

```
tar xfz pycharm-*.tar.gz
pycharm.sh
```

6.3.1 Glade

W celu wizualnego zaprojektowania graficznego interfejsu użytkownika posłużono się programem Glade. Program posiada duży wybór widżetów, których parametry można modyfikować za pomocą okna edycji. Kreowanie wyglądu interfejsu odbywa się za pomocą metody „przeciągnij i upuść". Projekt GUI zapisywany jest w postaci pliku XML, który można załadować do aplikacji za pomocą biblioteki GTK+. Warto wspomnieć, że program jest dostępny na licencji wolnego oprogramowania GPL.



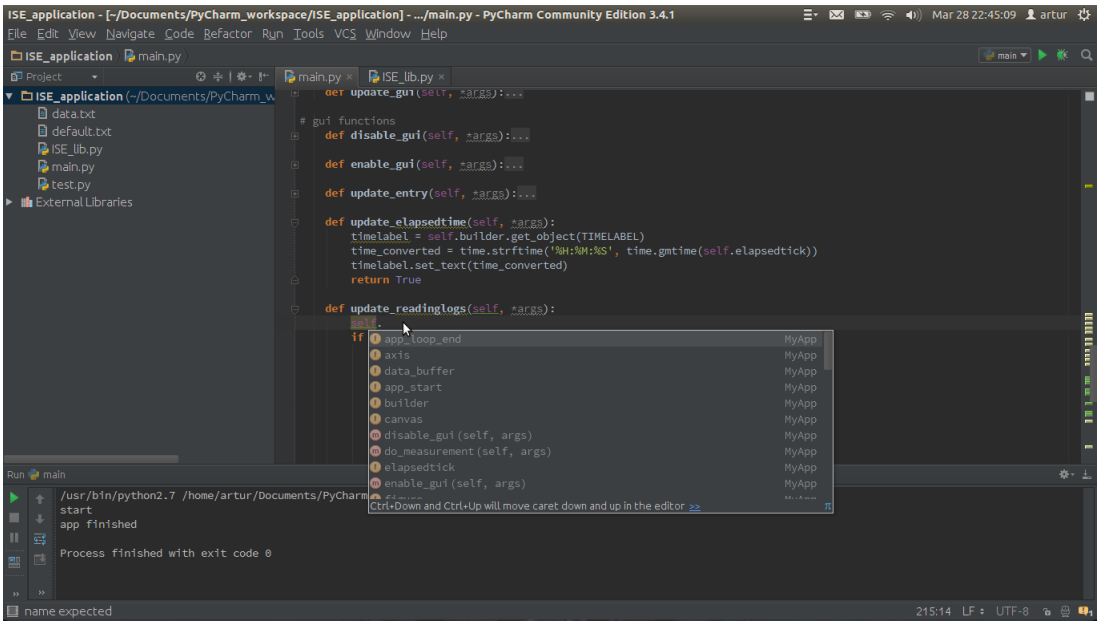
Rysunek 6.2: Widok kreatora Glade

6.3.2 pyCharm

Środowisko programistyczne pyCharm umożliwia programowanie w języku Python, dostarczając przy tym wiele udogodnień:

- edycja i analiza kodu źródłowego,
- refaktoryzacja kodu źródłowego,
- debugger,
- autouzupełnianie tekstu.

Wersja Community Edition tego środowiska dostępna jest na licencji Apache 2.



Rysunek 6.3: Widok środowiska pyCharm

6.4 Aplikacja

Aplikacja składa się z trzech plików. W celu skrócenia mniej istotnych linii kodu posłużono się wielokropkiem. Kontynuacja linii kodu korzysta ze znaku „\".

6.4.1 run_ISE_application.sh

Plik jest prostym skryptem uruchamiającym interpreter Pythona i ładującym do niego plik main.py, co skutkuje uruchomieniem aplikacji. Utworzenie skryptu zwalnia użytkownika od samodzielnego wpisywania poniższego polecenia w konsoli.

Listing 6.4: Skrypt uruchamiający aplikację

```
sudo python PyCharm_workspace/ISE_application/main.py
```

6.4.2 ISE_lib.py

Plik jest biblioteką napisaną w języku Python, zawierającą klasę ISE() będącą wirtualnym odpowiednikiem elektrody jonoselektywnej podłączonej do modułu urządzenia końcowego. Rzeczywiste zmierzone wartości napięcia, temperatury oraz dane diagnostyczne są przepisywane do odpowiednich pól klasy po uprzednim przetworzeniu.

Listing 6.5: Zawartość biblioteki ISE_lib.py

```
class ISE(object):
  def __init__(self)
  def configure(self, ele_ref, ele_slope)
  def get_value_mv(self)
  def get_value_px(self)
  def get_temperature(self)
  def get_address(self)
  def get_panid(self)
  def get_channel(self)
  def get_rssi(self)
  def get_lqi(self)
  def get_measinterval(self)
  def get_sleepinterval(self)
  def is_new_data(self)
  def is_error(self)
  def binary2mv(self, value, resolution, reference)
  def mv2px(self, value, slope, reference)
  def unpack(self, data)
```

Przetworzenie ciągu znaków odebranych za pomocą portu szeregowego (patrz tabela 6.2) następuje w metodzie unpack(). Metoda ta z danego łańcucha znaków wyłuskuje informacje o mierzonym napięciu, temperaturze oraz danych diagnostycznych.
Poniżej (listing 6.6) zamieszczono skrócony kod metody unpack(). W lini 2-3 następuje zastąpienie znaków CR (carriage-return, znak powrotu karetki) znakiem LF (line-feed, znak końca linii). Następnie w linii 4 za pomoca funkcji split() dokonywane jest rozdzielenie ciągu znaków na linie zapisywane do listy. Linia zawierająca informacje o mierzonych wartościach oraz informacje diagnostyczne poprzedzona jest słowem kluczowym „_sys_", dlatego w linii 7 za pomoca funkcji find() oraz instrukcji warunkowej dokonuje się odnalezienie tej linii.
Wyszukana linia posiada wartości liczbowe oddzielone od siebie znakami spacji.
W związku z tym, aby wydobyć te wartości należy ją podzielić - linia 9. Wynikiem tej instrukcji jest lista zawierająca wartości liczbowe pomiarów lub informacji diagnostycznych. W kolejnych liniach 11-25 następuje przepisanie listy do odpowiednich pól klasy, oraz sprawdzana jest poprawność danych.

Listing 6.6: Metoda unpack()

```
1  def unpack(self, data):
2    sp_data = data.replace('\r\n', '\n')
3    sp_data = sp_data.replace('\r', '\n')
4    sp_data = sp_data.split('\n')
5    print sp_data
6    for line in sp_data:
7      if line.find('_sys_') == 0:
8        if line.find(SYS_MEAS) == 0:
9          part = line.split(' ')
10         try:
11           adc = float(part[1])
12           temp = float(part[2])
13           if adc == TWI_ERROR:
14             self.adc_result = float('nan')
15           else:
16             self.adc_result = adc
17           if temp == ONEWIRE_ERROR:
18             self.temperature = float('nan')
19           else:
20             self.temperature = temp/10.0
21           #convert value from adc to voltage and px
```

```
22            self.value_mv = self.binary2mv(self.adc_result, \
23            self.adc_resolution, self.adc_reference)
24            self.value_px = self.mv2px(self.value_mv, \
25            self.electrode_slope, self.electrode_reference)
26            #new data available
27            self.new_data = True
28          except Exception:
29            pass
30        elif line.find(SYS_DIAG) == 0:
31          part = line.split(' ')
32          try:
33            self.address = part[1]
34            self.panid = part[2]
35            self.channel = part[3]
36            self.rssi = int(part[4])
37            self.lqi = int(part[5])
38            self.measinterval = int(part[6])
39            self.sleepinterval = int(part[7])
40          except Exception:
41            pass
42        ...
43        ...
44        ...
45        elif line.find(SYS_ERROR) == 0:
46          self.error = True
47          pass
48    return True
```

W liniach 22-25 następuje konwersja wartości napięcia z przetwornika analogowo-cyfrowego (liczba dwubajtowa) na wartośc napięcia w miliwoltach oraz wartość stężenia w skali pX. Kod źródłowy obu metod przedstawiono poniżej.

Listing 6.7: Metody konwertujące

```
1 def binary2mv(self, value, resolution, reference):
2   return (reference * 1000) / (2**resolution) \
3    * (value - (2**resolution)/2)
4
5 def mv2px(self, value, slope, reference):
6   return (value - reference) / slope
```

Od wartości napięcia odczytanego z przetwornika analogowo-cyfrowego (liczba dwubajtowa) należy odjąć połowę wartości napięcia źródła referencyjnego (także liczba dwubajtowa), ze względu na przesunięcie napięcia w torze analogowym modułu urządzenia końcowego (więcej w rozdziale 5).

$$result = \frac{reference \cdot 1000}{2^{resolution}} \cdot \left(value - \frac{2^{resolution}}{2} \right) \tag{6.1}$$

gdzie:
result - wynik, napięcie mierzone [mV]
$reference$ - napięcie odniesienia przetwornika ADC [V]
resolution - rozdzielczość przetwornika ADC [bits]
value - wartość odczytana przez przetwornik ADC [16bit value]

Wartości stężenia wyrażone w skali pX zależą od dwóch parametrów: slope i reference. Są to parametry regresji liniowej obliczonej dla krzywej wzorcowej elektrody jonoselektywnej. Za pomocą funkcji odwrotnej możliwe jest obliczenie stężenia w skali pX na podstawie znajomości wartości napięcia.

$$result = \frac{value - reference}{slope} \tag{6.2}$$

gdzie:
result - wynik, stężenie mierzone w skali pX
$reference$ - przesunięcie [mV]
slope - nachylenie $[\frac{\text{mV}}{\text{pX}}]$
value - wartość napięcia mierzonego [mV]

Tablica 6.2: Przykładowy ciąg znaków przed i po przetworzeniu

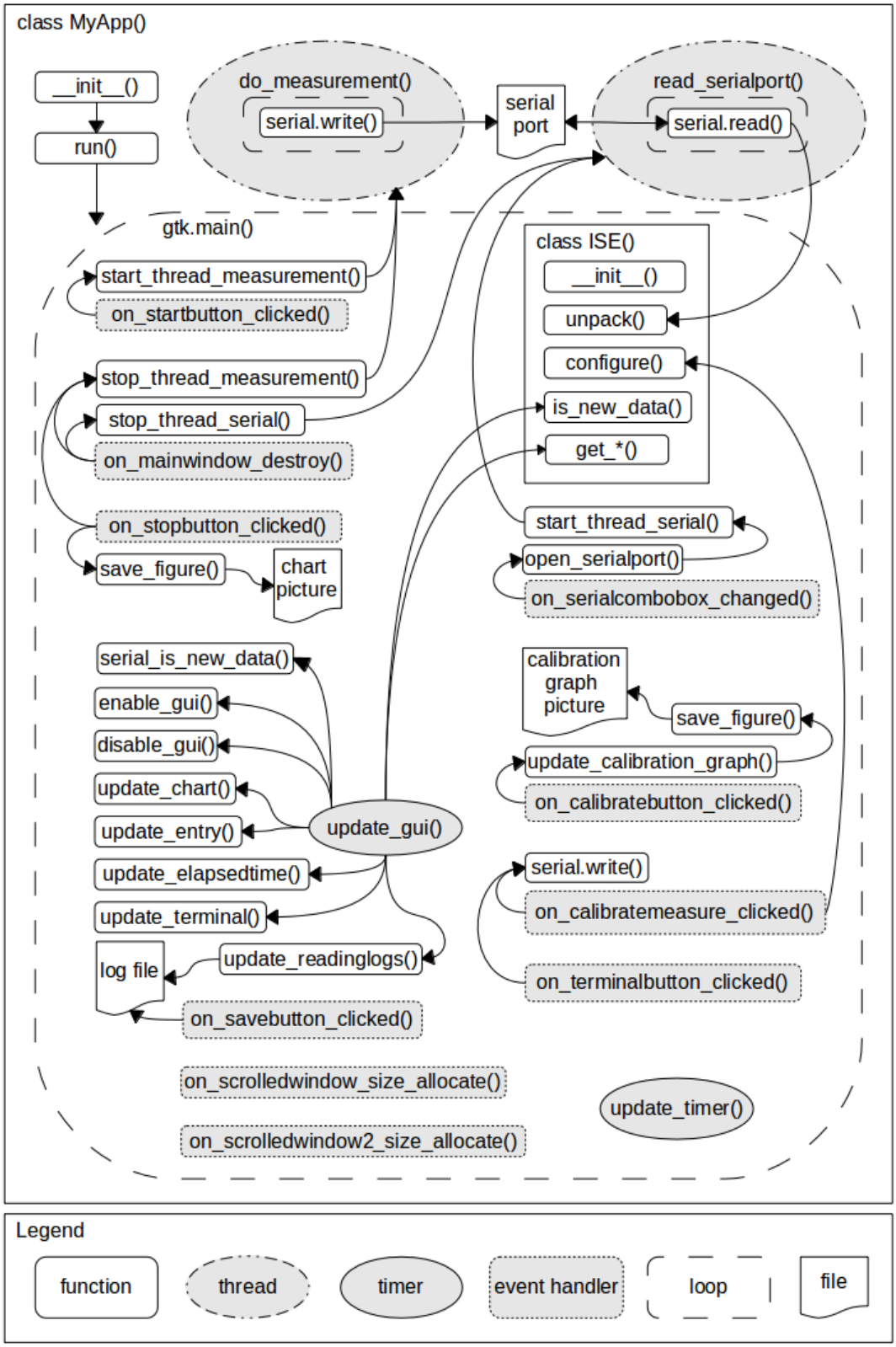
Przed przetworzeniem
[‘_sys_meas_ 32891 215\r\nMeasured voltage: 32891 [bits]\r\nMeasured temperature: 215 [*C]\r\n’]
Przetworzenie linia 2-3
[‘_sys_meas_ 32891 215\nMeasured voltage: 32891 [bits]\nMeasured temperature: 215 [*C]\n’]
Przetworzenie linia 4
[‘_sys_meas_ 32891 215’, ‘Measured voltage: 32891 [bits]’, ‘Measured temperature: 215 [*C]’]
Przetworzenie linia 7
[‘_sys_meas_ 32891 215’]
Przetworzenie linia 9
[‘_sys_meas_’, ‘32891’, ‘215’]

6.4.3 main.py

Plik główny aplikacji odpowiedzialny za tworzenie graficznego interfejsu oraz interakcję z użytkownikiem. Wykorzystuje bibliotekę GTK+ do wyświetlania okien graficznych, bibliotekę Matplotlib do rysowania wykresów oraz bibliotekę pySerial do komunikacji z portem szeregowym. Dla łatwiejszego zrozumienia jak działa aplikacja utworzono schemat blokowy przedstawiony na rysunku 6.4.

Wyróżnić można kilka rodzai bloków:

- function - funkcja, wydzielona część kodu realizująca określone działanie np. odświeżanie widżetów, zapis do pliku, rysowanie wykresu, przetwarzanie danych.
- thread - wątek, wydzielona część kodu wykonywana współbieżnie do wątku głównego.
- timer - funkcja wywoływana cyklicznie co określony interwał czasowy.
- event handler - funkcja wywoływana w momencie zajścia określonego wydarzenia np. kliknięcie przycisku, wybór opcji, zmiana zawartości rozwijanej listy.
- loop - pętla wykonywana dopóki spełniony jest warunek jej działania.
- file - plik, przechowuje tekst lub grafikę.



Rysunek 6.4: Schemat blokowy aplikacji

___init()___

W momencie uruchomienia aplikacji, tworzona jest instancja klasy MyApp(). Wywołana zostaje metoda ___init()___, która wykonuje czynności takie jak:

- ładowanie pliku XML do aplikacji. W pliku tym zawarte są parametry i informacje o rozmieszczeniu widżetów graficznego interfejsu użytkownika. W linii 6 plik XML zostaje dodany do obiektu klasy gtk.Builder. Linia 7 łączy zdarzenia z event handler'ami obsługującymi te zdarzenia.

Listing 6.8: Ładowanie pliku XML do aplikacji

```
1 GLADE_APP = "/Glade_workspace/ISE_app.glade"
2 ...
3 class MyApp (object):
4   def __init__(self):
5     self.builder = gtk.Builder()
6     self.builder.add_from_file(GLADE_APP)
7     self.builder.connect_signals(self)
8     ...
```

- tworzenie wątków i blokad. Linia 2 tworzy obiekt Lock() służący do blokowania dostępu innych wątków do określonego zasobu (zmiennej). Kolejna linia tworzy pole, które zostanie później przypisane do wątku - obiektu Thread(). Ostatnia linia tworzy obiekt Event(), reprezentujący wewnętrzną flagę, która może być ustawiana i zerowana, co jest przydatne podczas synchronizacji wątków.

Listing 6.9: Tworzenie wątków i blokad

```
1 ...
2 self.lock_thread_measurement = threading.Lock()
3 self.thread_measurement = None
4 self.alive_thread_measurement = threading.Event()
5 ...
```

- tworzenie FigureCanvas, czyli obszaru rysowania wykresu. Oprócz określenia rozmiaru i rozdzielczości wykresu (linia 2), dodano także nazwy osi pionowej i poziomej (linia 5, 7 i 8) oraz określono kolor linii wykresu (linia 9 i 10). Linia 11 i 12 tworzy bufory przeznaczone dla przechowywania wykreślanych wartości.

Listing 6.10: Tworzenie FigureCanvas

```
1  ...
2  self.figure = Figure(figsize=(3, 2), dpi=60)
3  self.canvas = FigureCanvas(self.figure)
4  self.axis1 = self.figure.add_subplot(211)
5  self.axis1.set_ylabel("px (-log(c))")
6  self.axis2 = self.figure.add_subplot(212)
7  self.axis2.set_ylabel("Temperature (C)")
8  self.axis2.set_xlabel("Time (s)")
9  self.line1, = self.axis1.plot([], [], 'r-o')
10 self.line2, = self.axis2.plot([], [], 'b-o')
11 self.data_chart_buffer1 = [[], []]
12 self.data_chart_buffer2 = [[], []]
13 ...
```

- tworzenie obiektu klasy ISE(). To ten obiekt dostarcza metod pozwalających na przetwarzanie danych odebranych z portu szeregowego.

Listing 6.11: Tworzenie obiektu klasy ISE()

```
1  ...
2  self.ise_object = ISE_lib.ISE()
3  ...
```

- tworzenie listy dostępnych urządzeń znakowych. Linia 4 pobiera z obiektu gtk.Builder widżet typu „ComboBox", rozwijalną listę z pozycjami do wyboru. Linia 5 tworzy obiekt gtk.ListStore(), będący listą do której w pętli for (linia 11) dodawane są kolejne nazwy urządzeń znakowych. Urządzenia znakowe dostępne w systemie wyszukiwane są za pomocą funkcji serial.tools.list_ports.comports(). Następnie w linii 13 do widżetu typu „ComboBox" dodawana jest lista z urządzeniami znakowymi. Linia 17 ustawia pierwszy element listy wyboru jako aktywny.

Listing 6.12: Tworzenie listy dostępnych urządzeń znakowych

```
1  ...
2  self.serialports = ["N/A"]
3  self.serialports_string = []
4  serialcombobox =self.builder.get_object(SERIALPORTCOMBOBOX)
5  self.serialstore = gtk.ListStore(gobject.TYPE_STRING)
6  self.serialstore.append(["N/A"])
7  for n, (portname, desc, hwid) in \
8  enumerate(sorted(serial.tools.list_ports.comports())):
9    port_s = [ portname + " (" + desc + " [" + hwid + "])" ]
10   self.serialports_string.extend(port_s)
11   self.serialstore.append(port_s)
12   self.serialports.append(portname)
13 serialcombobox.set_model(self.serialstore)
14 self.serialcell = gtk.CellRendererText()
15 serialcombobox.pack_start(self.serialcell, True)
16 serialcombobox.add_attribute(self.serialcell, 'text', 0)
17 serialcombobox.set_active(0)
18 ...
```

run()

Po utworzeniu klasy MyApp() wywoływana jest metoda run(), która uruchamia graficzny interfejs użytkownika (GUI). Linia 2 pobiera referencję okna głównego aplikacji, aby w linii 11 i 12 zmaksymalizować okno i pokazać wszystkie widżety w nim umieszczone. Linie 4-7 są odpowiedzialne za umieszczenie obszaru rysowania wykresów „Canvas" wewnątrz kontenera klasy gtk.Vbox. Linia 8 ustawia pierwszą zakładkę aplikacji okienkowej jako bieżącą, a linia 10 ukrywa pozostałe zakładki. W linii 13-14 rejestrowane są timery (funkcje wywoływane cyklicznie). Argumenty jakie przyjmuje funkcja glib.timeout_add() to interwał czasowy w milisekundach oraz nazwa funkcji wywoływanej. Linia 15 uruchamia pętlę główną biblioteki GTK.

Listing 6.13: Metoda run()

```
1  def run(self):
2      window = self.builder.get_object(MAINWINDOW)
3      #add graph widget
4      vbox = self.builder.get_object('vbox8')
5      vbox.pack_start(self.canvas, True, True)
6      vbox2 = self.builder.get_object('vbox16')
7      vbox2.pack_start(self.canvas2, True, True)
8      self.builder.get_object(NOTEBOOK).set_current_page(1)
9      #lock notebook tabs
10     self.builder.get_object(NOTEBOOK).set_show_tabs(False)
11     window.maximize()
12     window.show_all()
13     glib.timeout_add(TIMER_CYCLE, self.update_timer)
14     glib.timeout_add(UPDATE_CYCLE, self.update_gui)
15     gtk.main()
```

Wątek do_measurement()

Metoda start_thread_measurement() tworzy wątek (linia 2-3) i ustawia jego typ jako „daemon" co powoduje, że wątek kończy swoje działanie, gdy aplikacja (wątek główny) zostaje wyłączona. Flaga alive_thread_measurement typu Event() (zadeklarowana w metodzie ___init()___) jest teraz ustawiana za pomocą metody set(). Linia 6 uruchamia wątek.
Metoda stop_thread_measurement() sprawdza czy wątek istnieje. Jeśli tak - zeruje flagę alive_thread_measurement() informując o chęci zakończenia wątku.

Listing 6.14: Metody start() i stop() wątku do_measurement()

```
1  def start_thread_measurement(self):
2    self.thread_measurement = threading.Thread( \
3    target=self.do_measurement)
4    self.thread_measurement.daemon = True
5    self.alive_thread_measurement.set()
6    self.thread_measurement.start()
7
8  def stop_thread_measurement(self):
9    if self.thread_measurement is not None:
10     #clear alive event for thread
11     self.alive_thread_measurement.clear()
```

Wątek do_measurement (listing 6.15 służy do wysyłania komend do modułu urządzenia końcowego w celu włączenia / wyłączenia automatycznych pomiarów, konfiguracji i diagnostyki.
Za pomocą instrukcji warunkowej sprawdzane jest, czy został wybrany tryb pomiaru „auto" (linia 2) lub „sleep" (linia 21). Obsługa tych dwóch trybów jest zbliżona, więc zaprezentowano tylko obsługę trybu „auto". W liniach 3-8, w zależności od wybranej wartości interwału czasowego pomiarów, wysyłana jest komenda konfigurująca. Następnie wysyłana jest komenda diagnostyczna, w celu otrzymania informacji o konfiguracji modułu urządzenia końcowego. W linii 13 uruchamiany zostaje pomiar automatyczny, po czym wątek przechodzi do pętli (linia 16). Każdorazowo po otrzymaniu przez port szeregowy nowych danych pomiarowych, wysyłana zostaje komenda diagnostyczna (linie 18-20). W momencie gdy użytkownik zażąda wyłączenia pomiarów, warunek pętli (linia 16) staje się fałszywy i następuje opuszczenie pętli. Linia 25 czeka na ostatnią wartość pomiarową z modułu urządzenia końcowego po czym wyłącza pomiar automatyczny (linia 27-31), usuwa wątek (linia 32) i zeruje flagę app_start informującą o tym, czy aplikacja wykonuje pomiary.

Listing 6.15: Wątek do_measurement()

```
def do_measurement(self, *args):
  if self.measurement_mode == 0: #auto mode
    if self.auto_interval == 0:
      self.serial.write("ISE:auto:mode:0")
    elif self.auto_interval == 1:
      self.serial.write("ISE:auto:mode:1")
    else:
      self.serial.write("ISE:auto:mode:2")
    time.sleep(1)
    self.serial.write("ISE:diag") #get diagnostic info
    time.sleep(1)
    #enable automeasurement mode
    self.serial.write("ISE:auto:on")
    time.sleep(1)
    #main loop
    while self.alive_thread_measurement.isSet():
      time.sleep(SERIAL_TIMEOUT)
      if self.serial_new_data == True:
        self.serial.write("ISE:diag") #get diagnostic info
        time.sleep(1)
  else: #sleep mode
    ...
  time.sleep(1)
  #disable measurement and stop thread
  while self.serial_is_new_data() == False:
    time.sleep(SERIAL_TIMEOUT)
  if self.measurement_mode == 0: #auto mode
    #disable automeasurement mode
    self.serial.write("ISE:auto:off")
  else: #sleep mode
    self.serial.write("ISE:sleep:off") #disable sleep mode
  self.thread_measurement = None
  self.app_start = False
  return True
```

Wątek read_serialport()

Podobnie jak dla wątku do_measurement(), tak dla wątku read_serialport() zostały napisane metody start() i stop(). Istotną różnicą w metodzie stop_thread_serial() jest użycie metody join() (linia 13), która powoduje blokowanie pozostałych wątków do momentu, aż wątek read_serialport() nie zostanie zakończony.
Metoda serial_is_new_data() zwraca wartość True jeśli port szeregowy odebrał nowe dane, po czym ustawia flagę serial_new_data jako False.

Listing 6.16: Metody wątku read_serialport()

```
1  def start_thread_serial(self):
2    self.thread_serial = threading.Thread( \
3    target=self.read_serialport)
4    self.thread_serial.daemon = True
5    self.alive_thread_serial.set()
6    self.thread_serial.start()
7
8  def stop_thread_serial(self):
9    if self.thread_serial is not None:
10     #clear alive event for thread
11     self.alive_thread_serial.clear()
12     #wait until thread has finished
13     self.thread_serial.join()
14     self.thread_serial = None
15
16 def serial_is_new_data(self):
17   if self.serial_new_data == True:
18     self.serial_new_data = False
19     return True
20   else:
21     return False
```

Praca wątku polega na odczytywaniu znaków przechowywanych w buforze portu szeregowego. Za pomocą instrukcji serial.read(300) odczytywanych jest 300 znaków lub ich mniejsza ilość w czasie określonym parametrem „timeout", który wynosi 200 milisekund. Linia 8 wysyła dane do przetworzenia przez obiekt klasy ISE(). W celu zabezpieczenia współdzielonych zasobów przed jednoczesnym dostępem kilku wątków używana jest blokada lock_thread_serial. Za pomocą metody acquire() (linia 10) wątek blokuje innym wątkom dostęp do zasobu, lub jeżeli zasób jest już zablokowany - czeka do momentu odblokowania zasobu za pomocą metody release() (linia 16). W linii 12 odebrany ciąg znaków jest zapisywany do bufora.

Listing 6.17: Wątek read_serialport()

```
1  def read_serialport(self):
2    #loop while alive event is true
3    while self.alive_thread_serial.isSet():
4      text = self.serial.read(300) #get data
5      if text:
6        #unpack incoming data
7        #store in ise_object variables
8        self.ise_object.unpack(text)
9        #lock shared resource
10       self.lock_thread_serial.acquire()
11       try:
12         self.serial_raw_buffer += text
13         self.serial_new_data = True
14       finally:
15         #unlock shared resource
16         self.lock_thread_serial.release()
```

Procedura obsługi zdarzenia on_serialcombobox_changed()

W momencie wyboru portu szeregowego z listy dostępnych portów za pomocą rozwijanej listy „ComboBox" wywoływana jest funkcja on_serialcombobox_changed(). Sprawdza ona czy port szeregowy nie został wcześniej otwarty, jeżeli nie to wywołuje metodę open_serialport().
Linia 3 pobiera indeks wybranego elementu z listy portów. Za pomocą klauzuli „try-except" następuje próba otwarcia portu szeregowego (linia 5-7) wraz z obsługą wyjątku. Jeżeli próba przebiegnie pomyślnie uruchamiany jest wątek obsługi portu szeregowego (linia 17) i odblokowane zostają zakładki aplikacji (linia 19). W przypadku gdy próba otwarcia portu zakończy się niepowodzeniem, następuje wyświetlenie tekstu pomocy (linia 10-11) oraz ustawienie flagi informującej o otwartym porcie na wartość False.

Listing 6.18: Metoda open_serialport()

```
1  def open_serialport(self):
2    combo = self.builder.get_object(SERIALPORTCOMBOBOX)
3    portnumber = combo.get_active()
4    try:
5      self.serial = serial.Serial(self.serialports[portnumber], \
6      baudrate=38400, parity=serial.PARITY_NONE, \
7      timeout=SERIAL_TIMEOUT)
8    except Exception:
9      self.builder.get_object(NOTEBOOK).set_current_page(1)
10     self.builder.get_object(HELPLABEL).set_text( \
11     "Select correct serial port in Serial port settings!")
12     self.serial_is_open = False
13     return False
14   else:
15     self.serial_is_open = True
16     #start serial thread
17     self.start_thread_serial()
18     #unlock notebook tabs
19     self.builder.get_object(NOTEBOOK).set_show_tabs(True)
20     ...
21   return True
```

Procedura obsługi zdarzenia on_startbutton_clicked()

Na listingu 6.19 przedstawiono skrócony kod źródłowy procedury obsługi zdarzenia on_startbutton_clicked(). Użytkownik po naciśnięciu przycisku „START" uruchamia w aplikacji tryb wykonywania pomiarów. Metoda sprawdza czy tryb ten nie został już wcześniej włączony i czy wątek obsługi portu szeregowego jest właczony (linia 2). Klauzula „try-except" (linie 4-12) służy do pobrania parametrów krzywej wzorcowej elektrody jonoselektywnej (współczynnik nachylenia i przesunięcia), które potrzebne są do konfiguracji obiektu typu ISE() (linia 14). Jeśli podane parametry są błędne to wyświetlany jest tekst pomocy (linia 8-10), a funkcja kończy działanie.
W liniach 17-21 sprawdzane jest, czy zaznaczony został tryb „auto" lub „sleep", a w liniach 22-30 sprawdzona zostaje wybrana wartość interwału czasowego pomiarów. Aby zapobiec nadpisaniu plików zawierających logi pomiarowe, użytkownik musi podać nową nazwę pliku, kiedy wykonuje nowy pomiar (linia 34). Następnie zmieniana jest treść etykiety przycisku (linia 38), ustawiana flaga działania trybu pomiaru (linia 39) oraz uruchamiany jest wątek wykonywania pomiarów (linia 43). W sytuacji gdy nazwa pliku nie jest poprawna, pojawia się okienko błedu z wskazówką jak rozwiązać błąd (linie 44-51).

Listing 6.19: Metoda on_startbutton_clicked()

```
1  def on_startbutton_clicked(self, *args):
2    if self.app_start is False and self.thread_serial is not None:
3      #configure electrode
4      try:
5        ele_ref = float(self.builder.get_object( \
6        ELECTRODE_REFERENCEENTRY).get_text())
7      except Exception:
8        self.builder.get_object(HELPLABEL).set_text( \
9        "Enter correct value in Electrode settings! \
10       Use dot for decimals!")
11       self.builder.get_object(NOTEBOOK).set_current_page(1)
12       return
13     ...
14     self.ise_object.configure(ele_ref, ele_slope)
15     ...
16     #select measurement mode
17     if self.builder.get_object( \
18     MEASUREMENTMODEBUTTON).get_active() is True:
19       self.measurement_mode = 0 #auto mode
20     else:
```

```
21          self.measurement_mode = 1 #sleep mode
22        #select measurement timer interval
23        if self.builder.get_object( \
24        AUTOMODEBUTTON0).get_active() is True:
25          self.auto_interval = 0
26        elif self.builder.get_object( \
27        AUTOMODEBUTTON1).get_active() is True:
28          self.auto_interval = 1
29        else:
30          self.auto_interval = 2
31        #select sleep timer interval
32        ...
33        #check if entered file name
34        if self.savefilepath != "default.txt" and \
35        self.file_change is True:
36          self.file_change = False
37          #change startbutton label
38          self.builder.get_object(STARTBUTTON).set_label("Working")
39          self.app_start = True
40          #clear ISE data flag
41          self.ise_object.is_new_data()
42          #start thread
43          self.start_thread_measurement()
44        else:
45          save_errordialog = gtk.MessageDialog( \
46          type=gtk.MESSAGE_ERROR, buttons=gtk.BUTTONS_OK)
47          save_errordialog.set_markup( \
48          "You must specify the new file name in \
49          'Configuration' -> 'Save settings' ! ")
50          save_errordialog.run()
51          save_errordialog.destroy()
52          ...
```

Procedura obsługi zdarzenia on_stopbutton_clicked()

Zakończenie działania aplikacji możliwe jest po naciśnięciu przycisku „STOP”. Linia 3 zmienia treść etykiety przycisku, następnie do pliku zapisywane są wykresy pomiarowe i wyłączony zostaje wątek wykonywania pomiarów. W linii 7 zerowana jest flaga informująca o odebranych nowych danych pomiarowych.

Listing 6.20: Metoda on_stopbutton_clicked()

```
1 def on_stopbutton_clicked(self, *args):
2   #change startbutton label
3   self.builder.get_object(STARTBUTTON).set_label("Wait")
4   self.save_figure()
5   self.stop_thread_measurement()
6   #clear ISE data flag
7   self.ise_object.is_new_data()
```

Metoda save_figure() ustawia wielkość wykresów w calach (linia 3-4) i zapisuje je do plików w formacie .png (linia 5-8).

Listing 6.21: Metoda save_figure()

```
1 def save_figure(self):
2   #save matplotlib figure
3   self.figure.set_size_inches([8, 6])
4   self.figure2.set_size_inches([8, 6])
5   self.figure.savefig(self.savefilepath + \
6   "_measuring_chart.png")
7   self.figure2.savefig(self.savefilepath + \
8   "_calibration_graph.png")
```

Procedura obsługi zdarzenia on_calibratebutton_clicked()

Użytkownik może wprowadzić punkty mierzonej krzywej wzorcowej elektrody jonoselektywnej, po to aby otrzymać współczynniki tej krzywej korzystając z regresji liniowej. Punkty pomiarowe są dodawane do list za pomocą metody append() (linie 9-15). W pętli for (linie 16-27) sprawdzana jest poprawność wprowadzanych punktów. Jeżeli dane są poprawne to są dodawane do bufora (linie 23-24), a następnie wywoływana jest metoda update_calibration_graph().

Listing 6.22: Metoda on_calibratebutton_clicked()

```
1  def on_calibratebutton_clicked(self, *args):
2    data_px = []
3    data_mv = []
4    flag = False
5    index = 0
6    #remove items from graph buffer
7    self.data_graph_buffer = [[], []]
8    #entries with px values
9    data_px.append(self.builder.get_object("entry14").get_text())
10   ...
11   data_px.append(self.builder.get_object("entry22").get_text())
12   #entries with voltage values
13   data_mv.append(self.builder.get_object("entry15").get_text())
14   ...
15   data_mv.append(self.builder.get_object("entry23").get_text())
16   for x in data_px:
17     try:
18       float(x)
19       float(data_mv[index])
20     except ValueError:
21       pass
22     else:
23       self.data_graph_buffer[0].extend([float(x)])
24       self.data_graph_buffer[1].extend([float(data_mv[index])])
25       flag = True
26     finally:
27       index += 1
28   if flag is True:
29     self.update_calibration_graph()
```

Aby wyświetlić punkty pomiarowe i prostą regresji liniowej należy najpierw posortować dane znajdujące się w buforze (linie 3-5). Obliczenie współczynników regresji zrealizowane jest za pomocą metody polyfit(), która jest dostępna w bibliotece numpy (linia 7). W linii 8 tworzona jest funkcja obliczonej prostej, następnie obliczane są nowe wartości (linia 9). Wartość błędu regresji jest obliczona jako maksimum z modułu różnic pomiędzy wartościami rzeczywistymi a estymowanymi (linia 10). Wartości współczynników są zapisywane do widżetów typu „text entry" (linie 12-15).
W liniach 17-18 rysowane są punkty pomiarowe, a w dwóch kolejnych prosta regresji liniowej. Pozostałe metody służą do ustawienia siatki wykresu (linia 21), przeskalowania osi (linie 22-23) oraz odświeżenia wykresu (linia 24).

Listing 6.23: Metoda update_calibration_graph()

```
1  def update_calibration_graph(self):
2    #sort data in correct order
3    order = numpy.argsort(self.data_graph_buffer[0])
4    xs = numpy.array(self.data_graph_buffer[0])[order]
5    ys = numpy.array(self.data_graph_buffer[1])[order]
6    #calculate regression coefficient
7    fit = numpy.polyfit(xs, ys, 1)
8    fit_fn = numpy.poly1d(fit)
9    yfit = fit_fn(xs)
10   max_error = numpy.max(numpy.abs(ys - yfit))
11   #set electrode settings
12   self.builder.get_object(ELECTRODE_REFERENCEENTRY).set_text( \
13   str(fit[1]))
14   self.builder.get_object(ELECTRODE_SLOPEENTRY).set_text( \
15   str(fit[0]))
16   # plot
17   self.line3.set_ydata(ys)
18   self.line3.set_xdata(xs)
19   self.line4.set_ydata(yfit)
20   self.line4.set_xdata(xs)
21   self.axis3.grid(True)
22   self.axis3.relim()
23   self.axis3.autoscale_view()
24   self.canvas2.draw()
25   return True
```

Timer update_gui()

Metoda ta wywoływana jest cyklicznie co 500 milisekund i służy to odświeżania widżetów aplikacji. Jeżeli tryb pomiaru jest uruchomiony (linia 6), to następuje zablokowanie wybranych widżetów (linia 7). Metoda sprawdza, czy są nowe dane oczekujące na wyświetlenie (linia 3 i 9), jeśli tak to wywołuje metody aktualizujące poszczególne widżety.

Listing 6.24: Timer update_gui()

```
1  def update_gui(self, *args):
2    if self.thread_serial is not None:
3      if self.serial_is_new_data() is True:
4        self.update_terminal()
5      ...
6      if self.app_start is True:
7        self.disable_gui()
8        self.update_elapsedtime()
9        if self.ise_object.is_new_data() is True:
10         self.update_entry()
11         self.update_readinglogs()
12         self.update_chart()
13     else:
14       self.enable_gui()
15       self.builder.get_object(STARTBUTTON).set_label("START")
16   return True
```

Na listingu 6.25 pokazano metodę aktualizującą widżet typu gtk.TextView, będący wielowierszowym polem tekstowym. Najpierw pobrano referencję buforu przechowującego tekst widżetu (linia 2). Zasób serial_raw_buffer, w którym przechowywane są dane odebrane z portu szeregowego, jest zasobem współdzielonym przez wątek główny oraz wątek read_serialport(). Niezbędne jest więc zastosowanie blokady zasobu (linia 4) na czas przekopiowania zawartości zasobu do bufora widżetu (linia 7) oraz opróżnienia zawartości zasobu (linia 8). Finalnie zasób zostaje odblokowany (linia 11).

Listing 6.25: Metoda update_terminal()

```
1  def update_terminal(self, *args):
2    termbuffer = self.builder.get_object(TERMINALBUFFER)
3    #lock shared resource
4    self.lock_thread_serial.acquire()
5    try:
6      term_iter = termbuffer.get_end_iter()
7      termbuffer.insert(term_iter, self.serial_raw_buffer)
8      self.serial_raw_buffer = "" #clear buffer
9    finally:
10     #unlock shared resource
11     self.lock_thread_serial.release()
12   return True
```

Metoda update_elapsedtime() służy do wyświetlania czasu pomiaru. Metoda strftime() jest dostępna w standardowej bibliotece Pythona - time. Linia 3-4 konwertuje upływający czas wyrażony w sekundach na czas w formacie: godziny, minuty, sekundy.

Listing 6.26: Metoda update_elapsedtime()

```
1  def update_elapsedtime(self, *args):
2    timelabel = self.builder.get_object(TIMELABEL)
3    time_converted = time.strftime('%H:%M:%S', time.gmtime( \
4    self.elapsed_time))
5    timelabel.set_text(time_converted)
6    return True
```

Na listingu 6.27 przedstawiono metodę update_entry(), która aktualizuje pola tekstowe z danymi pomiarowymi i diagnostycznymi. Pobieranie danych odbywa się za pomocą metod get_*() należących do obiektu typu ISE(). Niektóre z danych są wartościami zmiennoprzecinkowymi (linia 2), inne całkowitoliczbowymi (linia 5), więc należy je przekonwertować za pomocą funkcji str() na łańcuch znakowy, aby możliwa była aktualizacja pola tekstowego (metoda set_text()).
Siła sygnału radiowego RSSI jest prezentowana za pomocą widżetu typu gtk.ProgressBar, czyli tzw. paska postępu. Ponieważ widżet ten przyjmuje wartości w zakresie 0-1, to należy wartość RSSI znormalizować (linia 13) i zabezpieczyć przed przekroczeniem zakresu (linie 14-17). Obliczona wartość jest przekazywana jako argument metody set_fraction() obiektu typu gtk.ProgressBar.

Listing 6.27: Metoda update_entry()

```
1  def update_entry(self, *args):
2    self.builder.get_object(READENTRY_MV).set_text(str("%.3f" % \
3    self.ise_object.get_value_mv()))
4    ...
5    self.builder.get_object(TEMPERATUREENTRY).set_text(str( \
6    self.ise_object.get_temperature()))
7
8    self.builder.get_object(ADDRESENTRY).set_text( \
9    self.ise_object.get_address())
10   ...
11   self.builder.get_object(RSSIENTRY).set_text(str( \
12   self.ise_object.get_rssi()))
13   strength = (self.ise_object.get_rssi() + 100) / 100.0
14   if strength < 0:
15     strength = 0
16   elif strength > 1:
17     strength = 1
18   self.builder.get_object(RSSIPROGRESSBAR).set_fraction( \
19   strength)
20   ...
21   return True
```

Aktualizacja i zapis logów pomiarowych do pliku możliwy jest dzięki metodzie update_readinglogs(). Linie 2-6 tworzą łańcuch znakowy zawierający datę, czas, wartość mierzonego napięcia, stężenia oraz temperatury. Otrzymany łańcuch jest zapisywany jako kolejny wiersz widżetu typu gtk.TextView (linia 10) oraz zostaje dodany do pliku otwartego w trybie binarnym (linie 12-16).

Listing 6.28: Metoda update_readinglogs()

```
1  def update_readinglogs(self, *args):
2    logline = (datetime.datetime.now().strftime("%Y-%m-%d") + \
3    "\t" + datetime.datetime.now().strftime("%H:%M:%S.%f")[:-3] \
4     + "\t" + str(self.ise_object.get_value_mv()) + \
5    "\t" + str(self.ise_object.get_value_px()) + \
6    "\t" + str(self.ise_object.get_temperature()) + "\n")
7
8    logsbuffer = self.builder.get_object(LOGSBUFFER)
9    log_iter = logsbuffer.get_end_iter()
10   logsbuffer.insert(log_iter, logline)
11   #save to file
12   fileobject = open(self.savefilepath, "ab")
13   try:
14     fileobject.write(logline)
15   finally:
16     fileobject.close()
17   return True
```

Nowe wartości pomiarowe są dołączane do list (linie 3-8), wykorzystując do tego celu metodę extend(). Listy te zawierają punkty pomiarowe (wartość, czas), które są wykreślane na wykresach za pomocą metod set_ydata() oraz set_xdata(). Aby wykresy były widoczne należy odświeżyć obszar rysowania wykresów tzw. Canvas, używając polecenia draw() (linia 21).

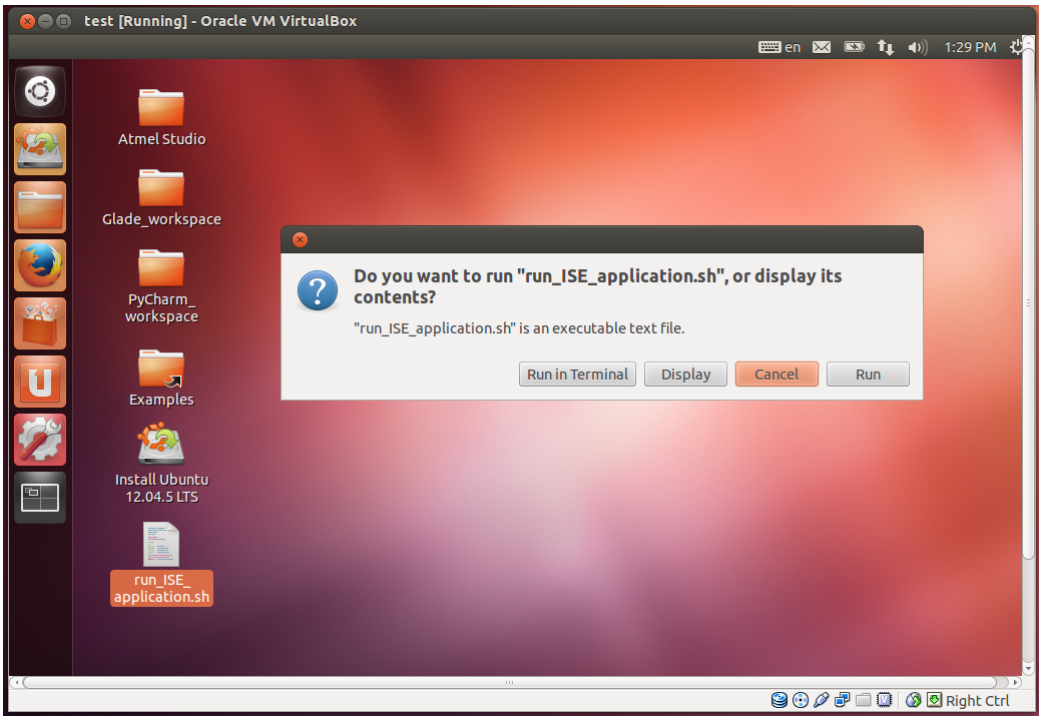
Listing 6.29: Metoda update_chart()

```
1  def update_chart(self, *args):
2    #write values to data_buffer for drawing chart
3    self.data_chart_buffer1[0].extend( \
4    [self.ise_object.get_value_px()])
5    self.data_chart_buffer1[1].extend([self.elapsed_time])
6    self.data_chart_buffer2[0].extend( \
7    [self.ise_object.get_temperature()])
8    self.data_chart_buffer2[1].extend([self.elapsed_time])
9    #first plot
10   self.line1.set_ydata(self.data_chart_buffer1[0])
11   self.line1.set_xdata(self.data_chart_buffer1[1])
12   self.axis1.grid(True)
13   self.axis1.relim()
14   self.axis1.autoscale_view()
15   #second plot
16   self.line2.set_ydata(self.data_chart_buffer2[0])
17   self.line2.set_xdata(self.data_chart_buffer2[1])
18   self.axis2.grid(True)
19   self.axis2.relim()
20   self.axis2.autoscale_view()
21   self.canvas.draw()
22   return True
```

6.5 Dystrybucja LiveCD

Do poprawnego działania aplikacji PC wymagane są elementy (zainstalowane w systemie operacyjnym): Python, PyGTK, Matplotlib i PySerial. Podczas korzystania z aplikacji wskazane jest użycie systemów opartych na Linuxie np. Ubuntu, gdyż praca w systemie Windows może być problematyczna. Dla ułatwienia pracy z aplikacją, bez konieczności instalowania i konfigurowania systemu, zdecydowano się stworzyć dystrybucję Ubuntu typu LiveCD. Jest to wersja systemu bezpośrednio uruchamiana z płyty CD lub pamięci USB (instalacja jest opcją).

Po uruchomieniu systemu, użytkownik podłącza kabel USB modułu koordynatora do komputera i uruchamia skrypt *run_ISE_application.sh* w celu włączenia aplikacji. Należy pamiętać, aby podczas wskazywania miejsca zapisu logów określić lokalizację, do której jest możliwy zapis np. dysk, pamięć USB.



Rysunek 6.5: Widok pulpitu dystrybucji LiveCD

6.5.1 Tworzenie dystrybucji

Pobrano obraz systemu *ubuntu-12.04.5-desktop-i386.iso* z oficjalnej strony *http://releases.ubuntu.com/12.04/*.
Zainstalowano program *uck* - Ubuntu Customization Kit, służący do tworzenia własnych dystrybucji systemu:

Listing 6.30: Instalacja Ubuntu Customization Kit

```
sudo apt-get install uck
```

Po jego uruchomieniu należy wskazać plik obrazu systemu i poczekać, aż program rozpakuje plik .iso oraz zdekompresuje system plików SquashFS. Program automatycznie zmienia katalog główny (chroot) i uruchamia konsolę za pomocą której można zainstalować brakujące pakiety:

Listing 6.31: Instalacja brakujących pakietów

```
sudo apt-get install python-gtk2 python-matplotlib
sudo apt-get install python-pip cutecom
sudo pip install --upgrade pyserial
```

Po instalacji należy pozbyć się osieroconych pakietów i tych zalegających w pamięci cache:

Listing 6.32: Oczyszczanie systemu ze zbędnych pakietów

```
sudo apt-get clean
sudo apt-get autoremove
```

Do katalogu /etc/skel (będzie to pulpit systemu) należy skopiować katalog z napisanym programem i ewentualnie inne pliki, które mogą być przydatne podczas pracy w systemie. W momencie zamknięcia konsoli program Ubuntu Customization Kit kompresuje system plików i tworzy obraz płyty .iso. Obraz można nagrać na płytę CD lub otworzyć w programie do wirtualizacji np. VirtualBox.

6.6 Instrukcja obsługi

Zaznajomienie się z poniższą instrukcją obsługi jest wskazane. Informacje w niej zawarte dotyczą zarówno pracy aplikacji PC jak i technikalii bezprzewodowego systemu pomiarowego.

6.6.1 Podłączenie

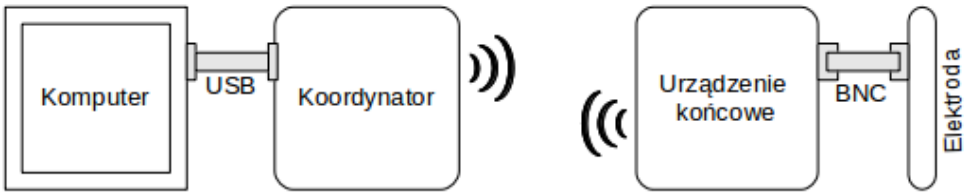
Moduł urządzenia końcowego

Moduł urządzenia końcowego należy zaopatrzyć w sprawne baterie (6xAA). Do gniazda BNC podłączamy przewód elektrody kombinowanej i zanurzamy ją w badanym roztworze, do którego zanurzamy także czujnik temperatury. Następnie włączamy zasilanie modułu, co powinno być sygnalizowane świeceniem czerwonej diody LED na obudowie.

Moduł koordynatora

Moduł koordynatora należy podłączyć do komputera za pomocą kabla USB. We wnętrzu obudowy koordynatora znajduje się konwerter USB-UART. W celu uzyskania jak najlepszego zasięgu i mocy sygnału skierować oba moduły na siebie, unikając przeszkód mogących znajdować się na drodze sygnału radiowego (ściany, drzewa itp.).

Schemat podłączenia



Rysunek 6.6: Schemat podłączenia bezprzewodowego systemu pomiarowego

6.6.2 Konfiguracja

Po uruchomieniu skryptu run_ISE_application.sh użytkownikowi ukazuje się okno aplikacji z widoczną zakładką konfiguracji. Poszczególne pozycje w tej zakładce to:

- Save settings - przycisk zapisu logów pomiarowych i wykresów do pliku o podanej przez użytkownika nazwie. Podczas wskazywania miejsca zapisu plików należy pamiętać, aby było to urządzenie fizyczne np. dysk twardy, dysk wymienny USB. Wskazanie lokalizacji znajdującej się w wirtualnym systemie plików dystrybucji LiveCD spowoduje, że po wyłączeniu systemu, pliki ulegną utracie (gdyż są przechowywane w pamięci operacyjnej).

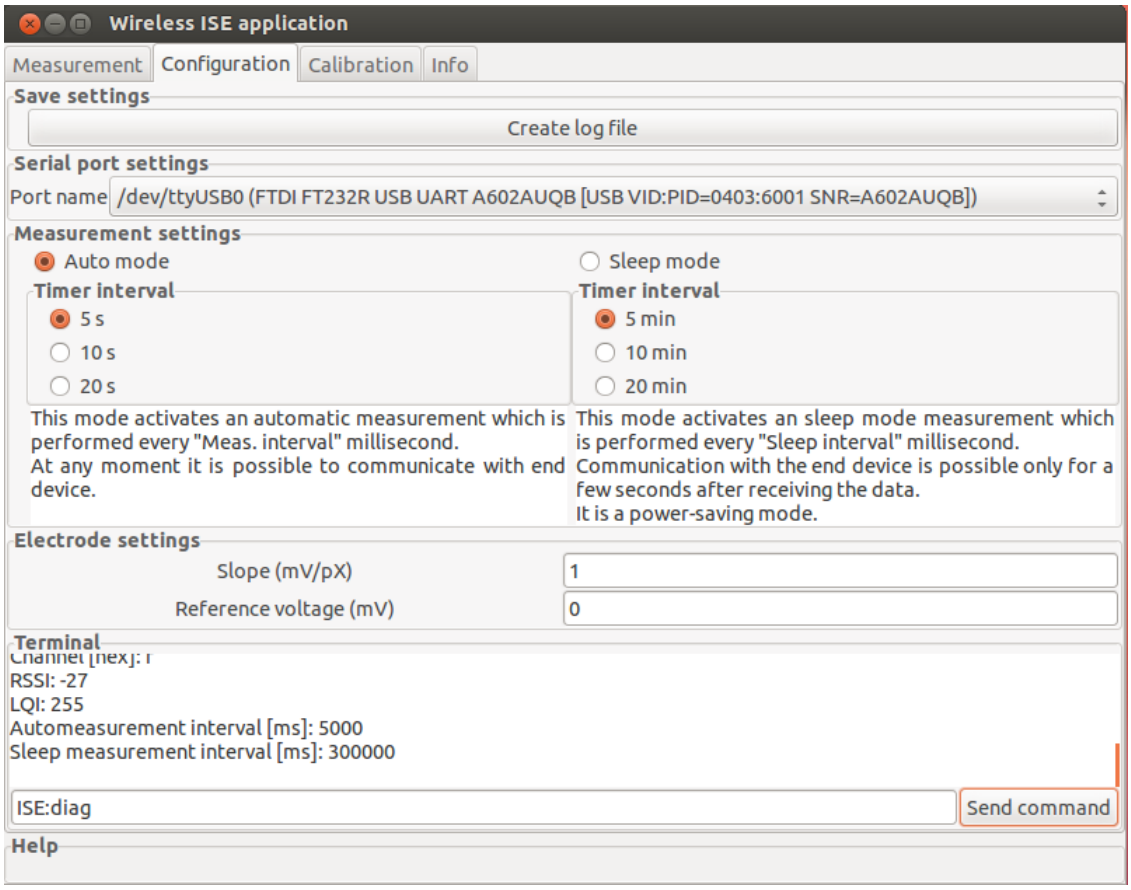
- Serial port settings - z rozwijalnej listy należy wyszukać podłączony do komputera konwerter USB-UART.

- Measurement settings - wybór trybu pomiaru. Dostępne są dwa tryby: auto i sleep. W trybie auto nadajnik radiowy oraz część analogowa urządzenia końcowego są włączone przez cały czas. W trybie sleep nadajnik radiowy uaktywniony zostaje tylko na czas przesłania danych pomiarowych, a część analogowa włącza się na trzy minuty przed dokonaniem pomiaru (czas ten potrzebny jest, aby układy wzmacniające osiągnęły swój stan ustalony). Po wysłaniu danych nadajnik radiowy jest aktywny jeszcze przez 5 sekund, co umożliwia wyłączenie trybu sleep.

- Electrode settings - do pól tekstowych można wprowadzić współczynniki krzywej wzorcowej elektrody jonoselektywnej: nachylenie (slope) oraz przesunięcie (reference). Pozostawienie wartości domyślnych skutkuje otrzymaniem takich samych wartości mierzonego napięcia i stężenia.

- Terminal - okno terminala, do którego można wysyłać komendy tekstowe. Przydatne podczas rozwiązywania problemów lub testowania bezprzewodowego systemu pomiarowego.

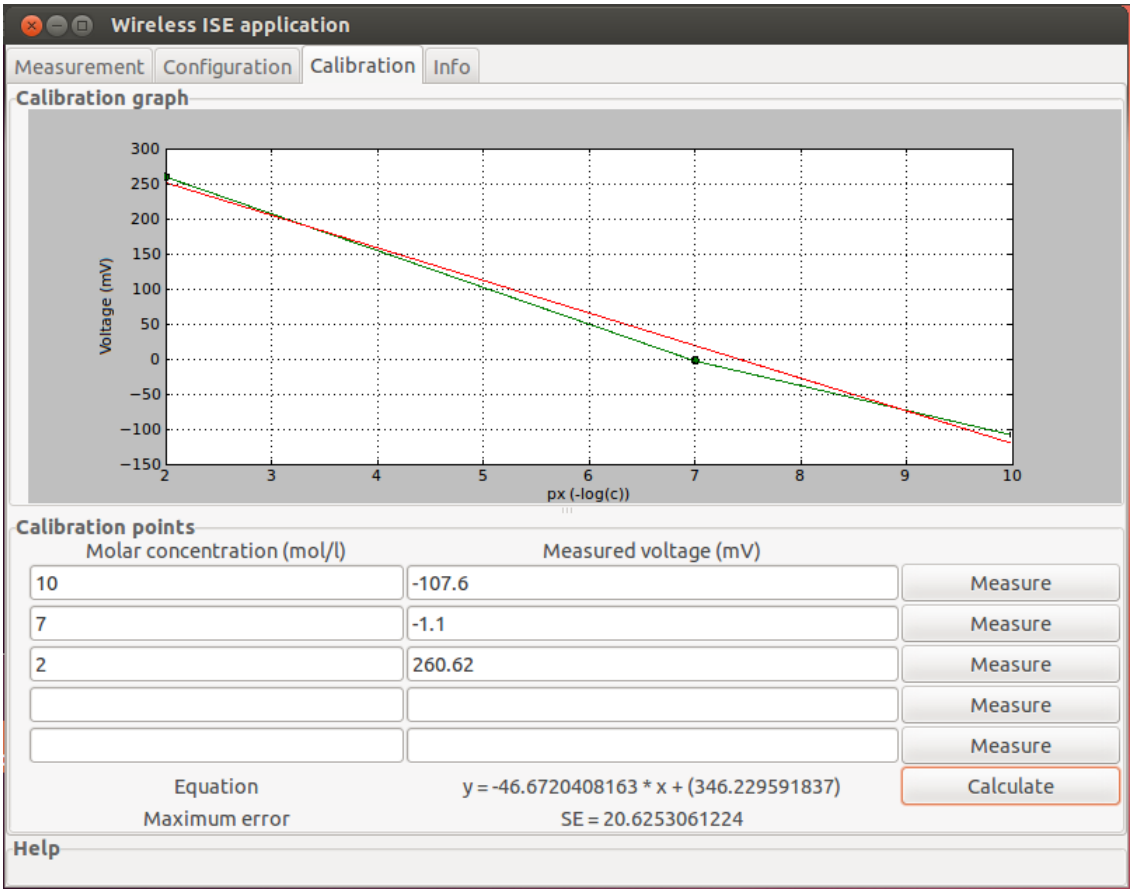
- Help - tekst pomocy wyświetlający się w zależności od działania użytkownika.



Rysunek 6.7: Zakładka konfiguracji

6.6.3 Kalibracja

Za pomocą zakładki kalibracji możliwe jest obliczenie krzywej wzorcowej elektrody jonoselektywnej. Można to zrobić na dwa sposoby. Pierwszy sposób polega na wprowadzeniu wartości stężeń w jednostkach pX oraz odpowiadających im wartości napięcia mierzonego i naciśnięciu przycisku „Calculate". Drugi sposób polega na rzeczywistym pomiarze napięcia dla próbek o określonym stężeniu. Należy więc dla każdego wprowadzonego stężenia nacisnąć odpowiadający mu przycisk „Measure". Pomiar trwa trzy minuty, a wartość pomiaru automatycznie zostanie wpisana w odpowiednie pole tekstowe. Wynikiem obu metod jest wzór zawierający współczynniki krzywej oraz błąd regresji.



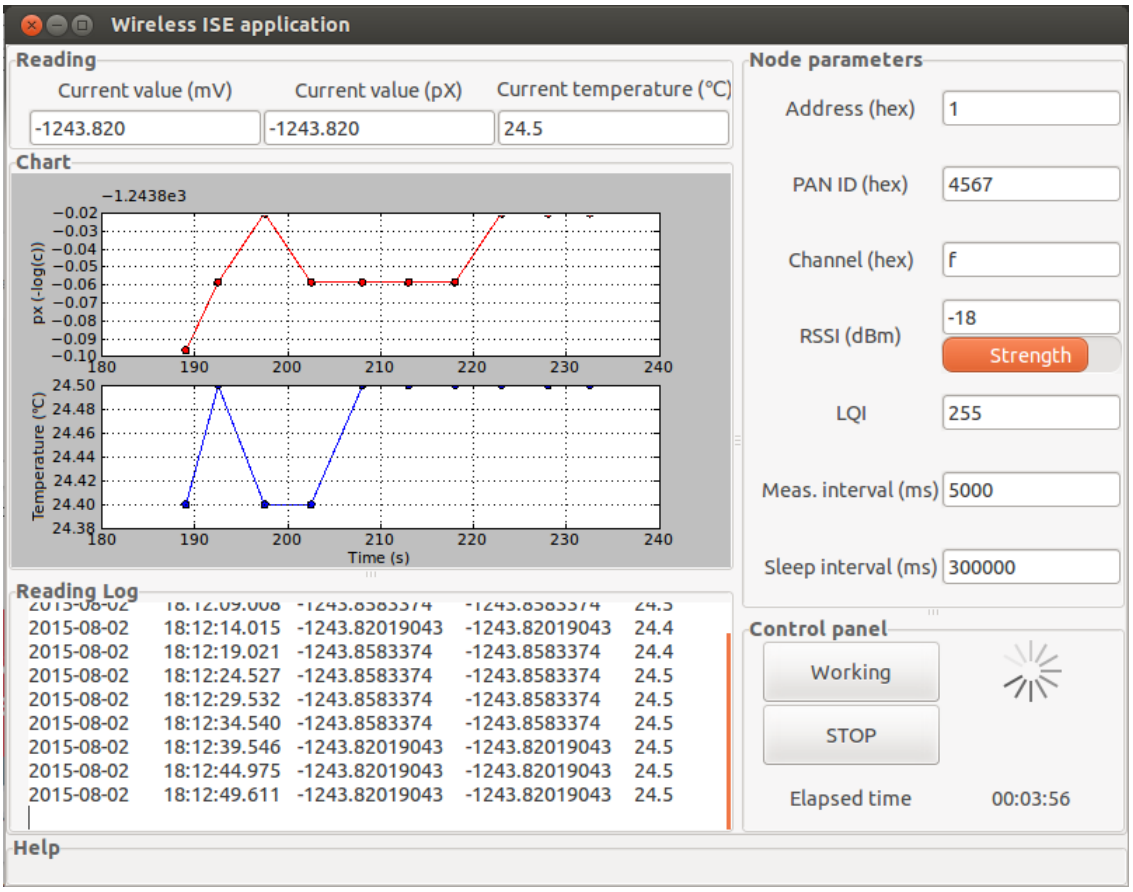
Rysunek 6.8: Zakładka kalibracji

6.6.4 Pomiar

Zakładka pomiaru służy do obserwacji wyników pomiarowych oraz danych diagnostycznych. Można wyróżnić kilka pozycji:

- Reading - trzy pola tekstowe informujące o bierzącej wartości napięcia, stężenia i temperatury.
- Chart - pole wykresu na którym naniesione są wartości mierzonego stężenia i temperatury.
- Reading Log - wielowierszowe pole tekstowe z logami pomiarowymi. Pole to można przewijać za pomocą suwaka znajdującego się po prawej stronie.
- Node parameters - parametry diagnostyczne modułu urządzenia końcowego takie jak: adres, identyfikator PAN ID, kanał, RSSI, LQI, interwał czasowy dla trybu auto i sleep. RSSI - to wartość wskaźnika mocy sygnału radiowego odbieranego przez moduł urządzenia końcowego. Przyjmuje wartości od -90 dBm wzwyż (-90 dBm oznacza najsłabszą moc). LQI - Link Quality Indicator to wskaźnik jakości połączenia radiowego. Przyjmuje wartości od 0 do 255 (0 oznacza najgorszą jakość).
- Control panel - zawiera przyciski „START” i „STOP” do uruchamiania i zatrzymywania pomiarów. Widoczny jest także czas trwania pomiarów.

Po naciśnięciu przycisku „START” wysyłana jest komenda rozpoczynająca pomiary modułu urządzenia końcowego. Pierwszy wynik pomiaru jest dostępny po upływie 3 minut, ponieważ tyle czasu jest potrzebne, aby część analogowa modułu ustabilizowała swoje parametry tzw. „warm-up time”.



Rysunek 6.9: Zakładka pomiaru

6.6.5 Lista komend

Tablica 6.3: Lista dostępnych komend tekstowych

Komenda	Opis
ISE:meas	wykonaj jednokrotny pomiar
ISE:diag	wyświetl dane diagnostyczne
ISE:auto:on	uruchom tryb auto
ISE:auto:off	wyłącz tryb auto
ISE:auto:mode:0	wykonuj pomiary co 5 sekund
ISE:auto:mode:1	wykonuj pomiary co 10 sekund
ISE:auto:mode:2	wykonuj pomiary co 20 sekund
ISE:sleep:on	uruchom tryb sleep
ISE:sleep:off	wyłącz tryb sleep
ISE:sleep:mode:0	wykonuj pomiary co 5 minut
ISE:sleep:mode:1	wykonuj pomiary co 10 minut
ISE:sleep:mode:2	wykonuj pomiary co 20 minut
Specjalna komenda ISE:meas:nowait	wykonaj pomiar natychmiast i prześlij 1000 próbek

Rozdział 7

Testy i pomiary bezprzewodowego systemu pomiarowego

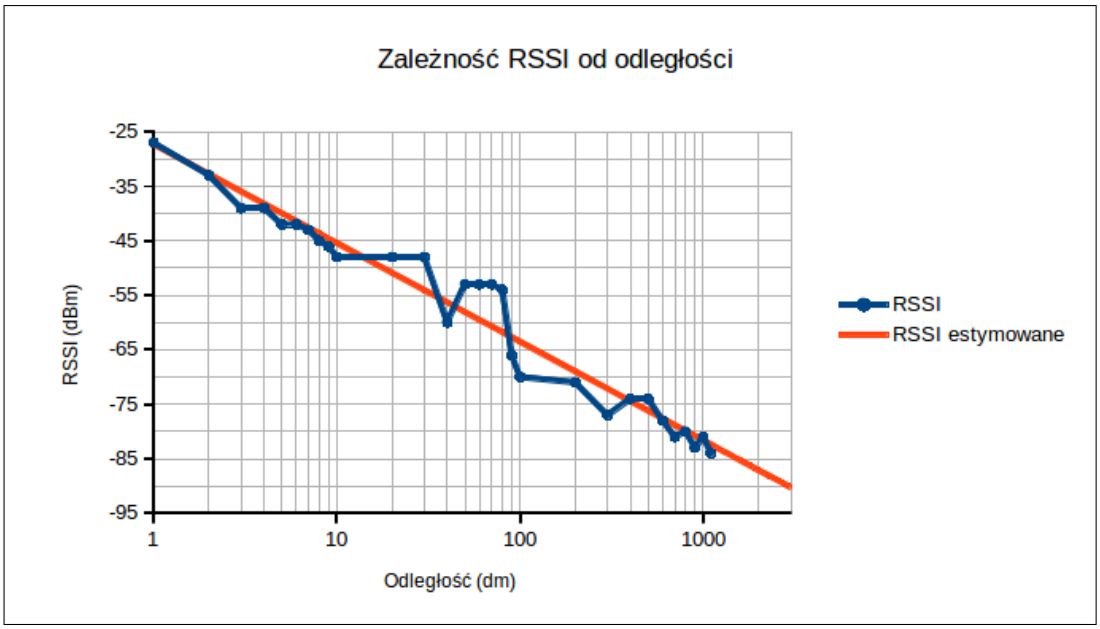
Bezprzewodowy system pomiarowy poddano testom i pomiarom, aby wykryć nieprawidłowości w : komunikacji radiowej, pomiarach napięcia i temperatury, działaniu aplikacji PC. Określono sposób pomiaru konkretnych wielkości, wymagane przyrządy pomiarowe oraz sposób obliczenia i prezentacji wyników pomiarowych.

7.1 Pomiar zasięgu

Wykonano pomiar wartości wskaźnika mocy odbieranego sygnału radiowego (RSSI) w zależności od odległości między nadajnikami. Wskaźnik ten przyjmuje wartości od -90 dBm wzwyż. Gdy moc sygnału jest najmniejsza (-90 dBm) to nadajniki radiowe tracą ze sobą łączność. Pomiary wykonano na otwartej przestrzeni ustawiając moduł urządzenia końcowego 1,5 metra nad ziemią. Następnie za pomocą miary elastycznej odmierzano odległość, kierowano moduł koordynatora w stronę modułu urządzenia końcowego i odczytywano wartość wskaźnika RSSI. Zależność RSSI od odległości przedstawiono na rysunku 7.1 w skali logarytmicznej. Można zauważyć, że 10-krotne zwiększenie odległości powoduje spadek RSSI o około 18 dBm. Wyznaczono także prostą regresji, aby oszacować wartości RSSI dla odległości od 1200 dm to 3000 dm. Teoretyczny zasięg wynosi więc 300 metrów, niestety możliwości techniczne nie pozwoliły na sprawdzenie zasięgu na tak dalekim dystansie.
Równanie prostej regresji to:

$$RSSI = -18,11 \cdot \log_{10}(l) - 27,3 \tag{7.1}$$

gdzie:
l - odległość [dm]



Rysunek 7.1: Zależność RSSI od odległości

odległość (dm)	RSSI	RSSI estymowane	odległość (dm)	RSSI	RSSI estymowane
1	-27	-27	700	-81	-79
2	-33	-33	800	-80	-80
3	-39	-36	900	-83	-81
4	-39	-38	1000	-81	-82
5	-42	-40	1100	-84	-82
6	-42	-41	1200		-83
7	-43	-43	1300		-84
8	-45	-44	1400		-84
9	-46	-45	1500		-85
10	-48	-45	1600		-85
20	-48	-51	1700		-86
30	-48	-54	1800		-86
40	-60	-56	1900		-87
50	-53	-58	2000		-87
60	-53	-59	2100		-87
70	-53	-61	2200		-88
80	-54	-62	2300		-88
90	-66	-63	2400		-89
100	-70	-64	2500		-89
200	-71	-69	2600		-89
300	-77	-72	2700		-89
400	-74	-74	2800		-90
500	-74	-76	2900		-90
600	-78	-78	3000		-90

Rysunek 7.2: Arkusz pomiarowy zależności RSSI od odległości

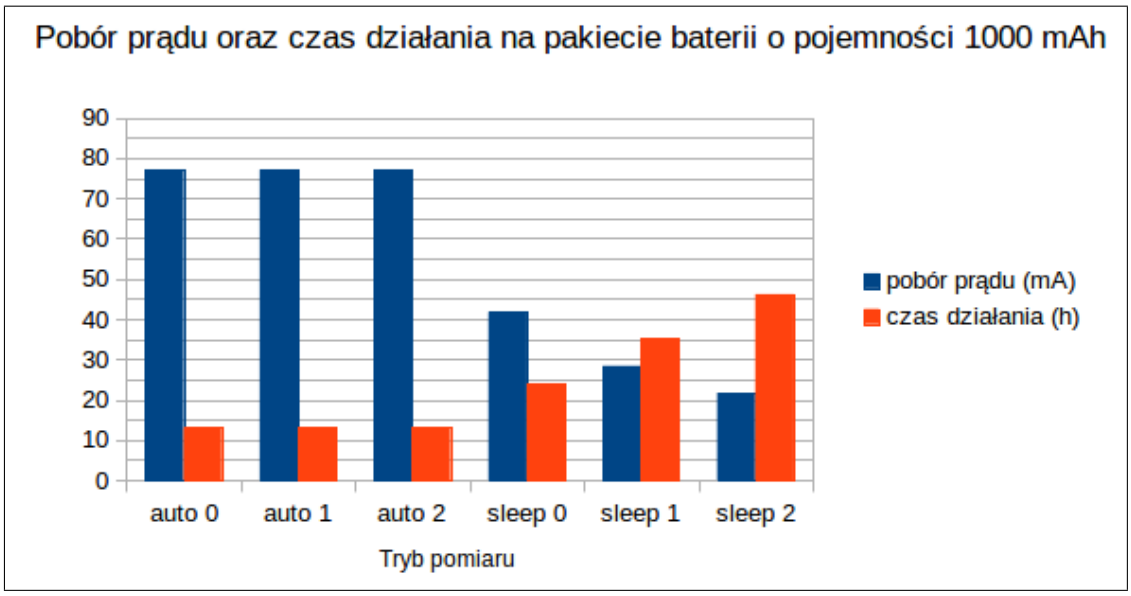
7.2 Pomiar poboru prądu

Zmierzono pobór prądu poszczególnych elementów modułu urządzenia końcowego. Największą wartość prądu pobiera część analogowa urządzenia, co jest zgodne z oczekiwaniami, ponieważ zawiera wiele elementów elektronicznych takich jak: stabilizatory, rezystory, wzmacniacze, przetwornica DC/DC oraz przetwornik ADC.

Tablica 7.1: Pobór prądu poszczególnych elementów

Element	Prąd (mA)
Mikroprocesor	15
Nadajnik	18
Część analogowa	44

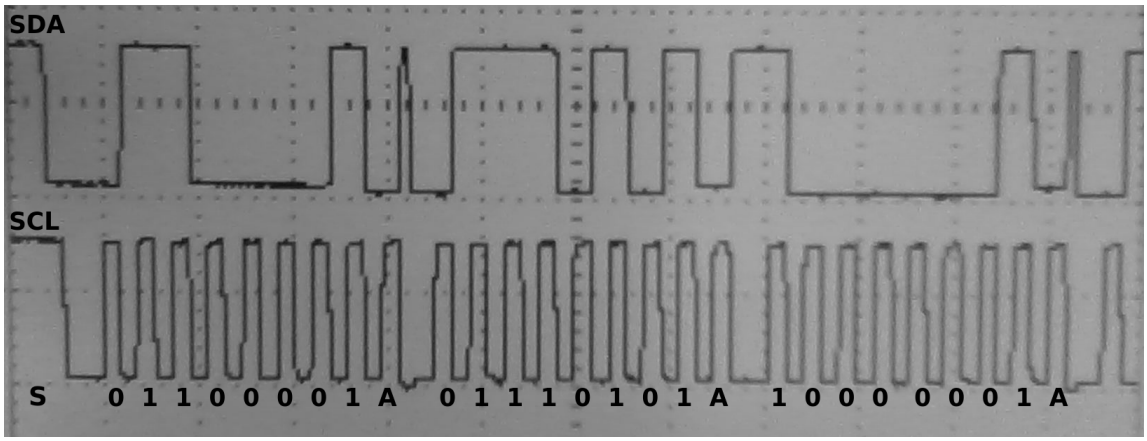
Zmierzono pobór prądu w zależności od trybu pomiaru. Dla trzech trybów auto pobór prądu jest identyczny, ponieważ zarówno nadajnik jak i część analogowa jest włączona przez cały czas. Oszczędność energii możliwa jest w trybie sleep. Im dłuższy okres między pomiarami tym mniejszy pobór prądu.
Wykonano oszacowanie czasu działania modułu na jednym pakiecie baterii o pojemności 1000 mAh (rysunek 7.3).



Rysunek 7.3: Pobór prądu w zależności od trybu pomiaru

7.3 Test interfejsu I2C

Za pomocą oscyloskopu zbadano sygnały na magistrali I2C (rysunek 7.4). Pierwszym bajtem jest wysyłany adres urządzenia slave, kolejne dwa bajty to 16-bitowy wynik pomiaru napięcia. Częstotliwość sygnału zegarowego wynosi 55,5 kHz. Komunikacja działa poprawnie, sygnały nie są obarczone zakłóceniami. Pojemność linii nie jest wysoka, gdyż czas narastania zboczy jest krótki.



Rysunek 7.4: Sygnały na magistrali I2C

7.4 Pomiar zakłóceń

Źródłem zakłóceń wpływających na wartość sygnału mierzonego są: sieć elektryczna oraz pracujący moduł urządzenia końcowego.
Zakłócenia pochodzące od pracującego modułu urządzenia końcowego mają charakter impulsowy i są sumą zakłóceń od takich elementów jak: nadajnik radiowy pracujący z częstotliwością 2,4 Ghz, mikroprocesor pracujący z częstotliwością 8 Mhz i przetwornica DC/DC pracująca z częstotliwością 80 kHz. Niestety pomiar tych zakłóceń nie jest możliwy za pomocą zwykłego oscyloskopu. Pomiary takie można wykonać korzystając z analizatora widma sygnału.

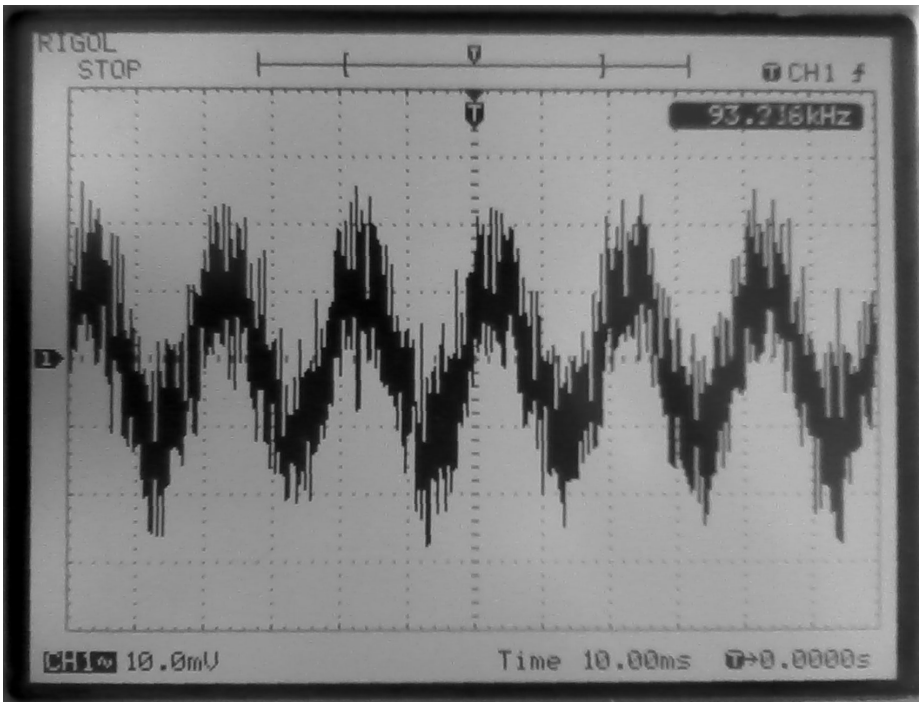
7.4.1 Zakłócenia sieci elektrycznej

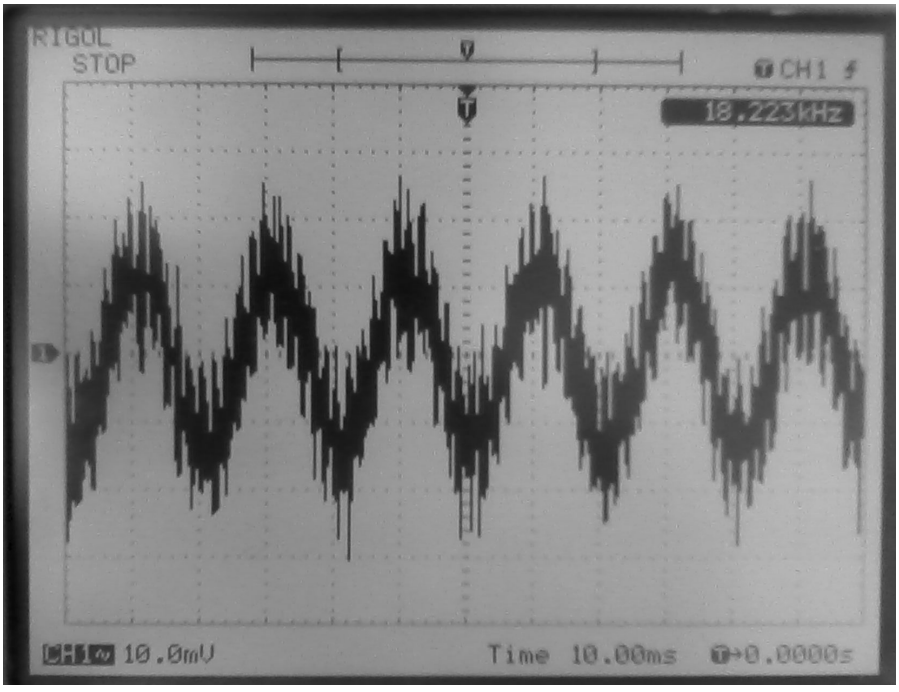
Zakłócenia pochodzące od sieci elektrycznej są łatwo rozpoznawalne, charakteryzują się sinusoidalnym kształtem i częstotliwością 50 Hz. Powstają one w wyniku indukcji elektromagnetycznej w długich przewodach doprowadzeniowych łączących elektrodę jonoselektywną z modułem urządzenia końcowego. Ich pomiar jest możliwy za pomocą oscyloskopu.

W celu zminimalizowania wpływu zakłóceń sieci elektrycznej zastosowano filtr dolnoprzepustowy (więcej w rozdziale 5) o częstotliwości granicznej 0,34 Hz w torze analogowym modułu. W poniższych tabelach 7.2, 7.3 i 7.4 zaprezentowano porównanie poziomu zakłóceń, przed i za filtrem oraz na zasilaniu, dla trybu auto i sleep.

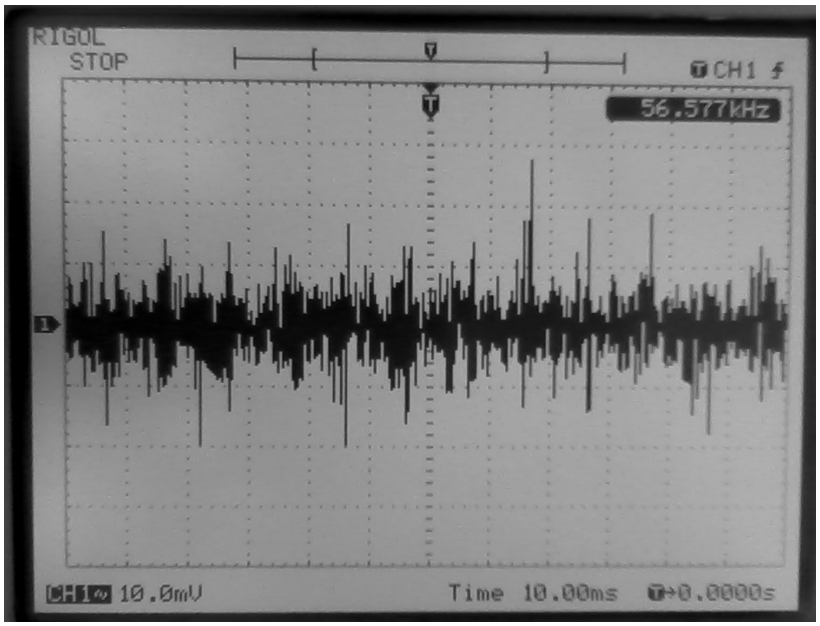
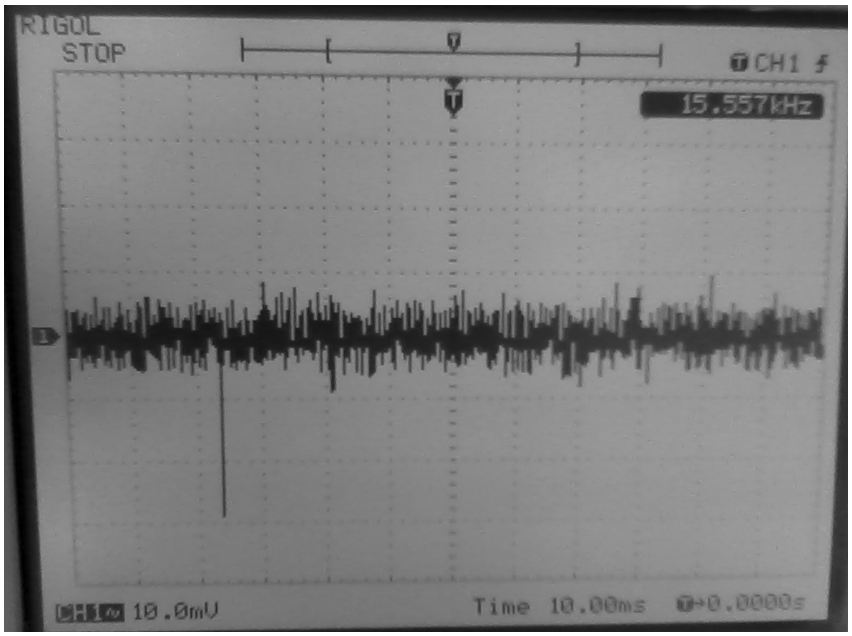
Poziom zakłóceń przed filtrem jest praktycznie jednakowy dla obu trybów. Widać wyraźnie, że zakłócenia nie są czystą sinusoidą, ale zawierają składowe o wyższych częstotliwościach. Dopiero odfiltrowanie sygnału mierzonego za pomoca filtru dolnoprzepustowego powoduje, że sygnał ten jest obarczony mniejszym poziomem zakłóceń. Wartość międzyszczytowa zakłóceń dla trybu sleep jest dwukrotnie mniejsza w porównaniu do trybu auto, ponieważ nadajnik radiowy jest wyłączony. Charakter impulsowy zakłóceń pochodzących od pracującego modułu urządzenia końcowego, w postaci pionowych szpilek napięcia, można zauważyć w tabelach 7.3 i 7.4.

Tablica 7.2: Porównanie poziomu zakłóceń przed filtrem dla trybu auto i sleep

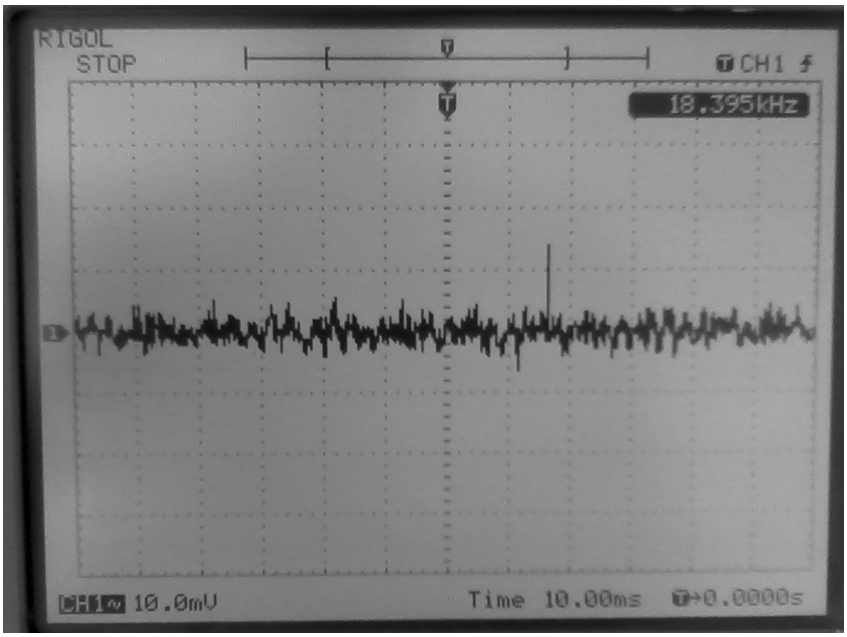
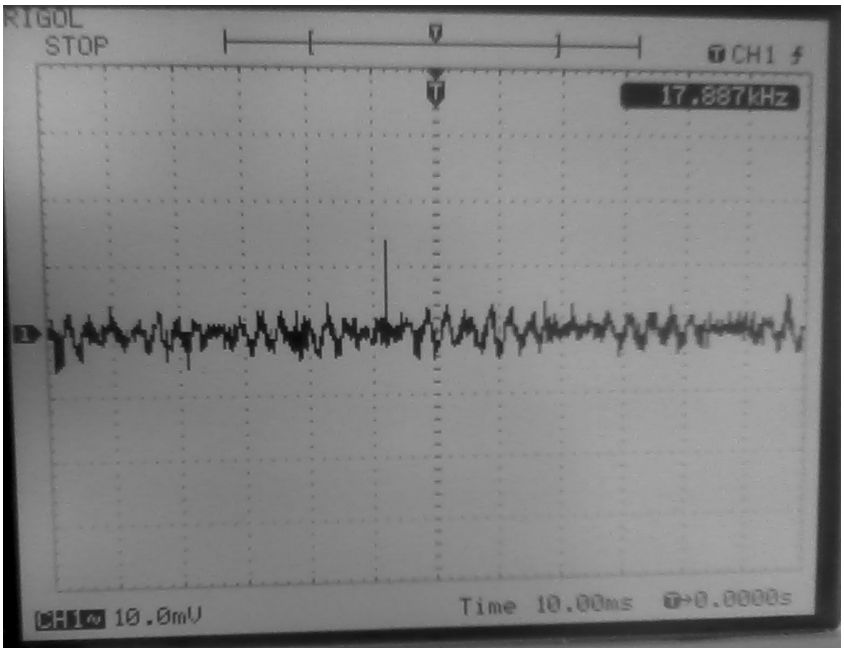
Tryb auto
Wartość międzyszczytowa: 50 mV


Tryb sleep
Wartość międzyszczytowa: 50 mV


Tablica 7.3: Porównanie poziomu zakłóceń za filtrem dla trybu auto i sleep

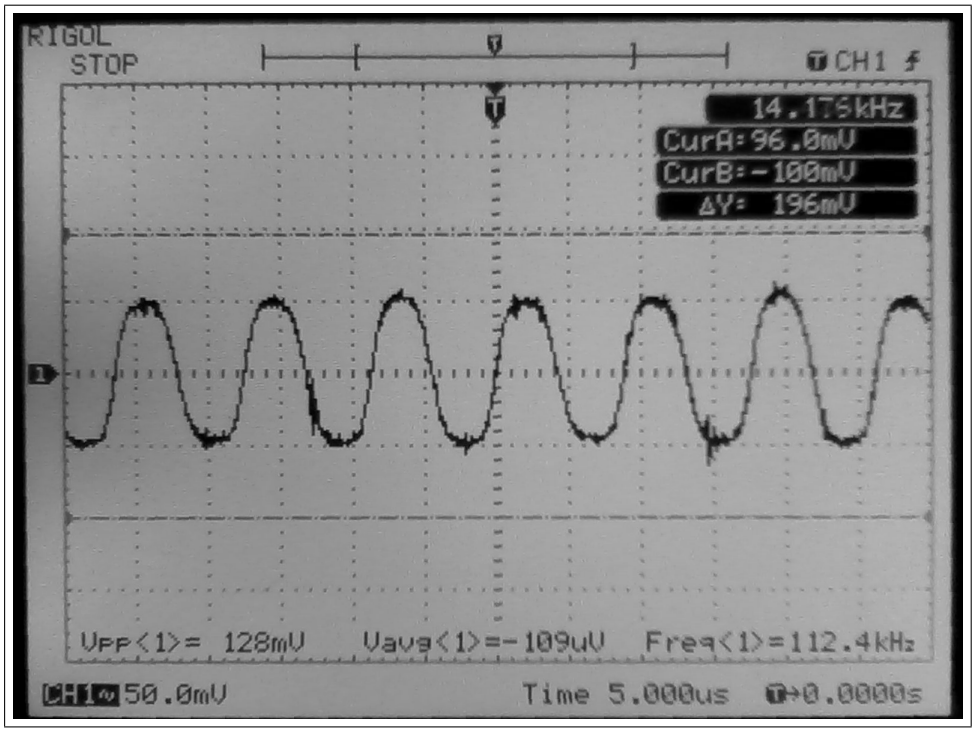
Tryb auto
Wartość międzyszczytowa: 40 mV 
Tryb sleep
Wartość międzyszczytowa: 20 mV 

Tablica 7.4: Porównanie poziomu zakłóceń na zasilaniu dla trybu auto i sleep

Tryb auto
Wartość międzyszczytowa: 10mV 
Tryb sleep
Wartość międzyszczytowa: 10 mV 

7.4.2 Zakłócenia wyeliminowane

Podczas badania oscyloskopem poszczególnych sygnałów na płytce elektronicznej modułu urządzenia końcowego, zaobserwowano interesujące zjawisko. Zjawisko to polegało na samoistnym generowaniu się napięcia sinusoidalnego na wyjściu wtórnika napięciowego (symbol IC3D, patrz podrozdział 5.3). Przyczyną samowzbudzania się wtórnika było umieszczenie na jego wyjściu kondensatora 100 nF, który obciążał go pojemnościowo i powodował utratę stabilności wzmacniacza. Po wymontowaniu kondensatora zjawisko zostalo zlikwidowane.



Rysunek 7.5: Samowzbudzenie się wtórnika napięciowego

7.5 Wyznaczenie niepewności pomiarowych

Wartość napięcia mierzonego za pomocą modułu urządzenia końcowego jest obarczona pewną niepewnością pomiarową.
Dokładność pomiaru zależy od wielu czynników:

- Temperatury - zmiana temperatury powoduje zmianę napięcia odniesienia dla wzmacniacza INA116 i przetwornika ADC. Wzmocnienie wzmacniaczy operacyjnych i ich napięcie offsetu także zmienia się pod wpływem temperatury.
- Pól elektromagnetycznych i elektrostatycznych - indukują one dodatkowe napięcia i prądy zakłócające w przewodach doprowadzeniowych.
- Czasu - parametry elementów elektronicznych zmieniają się w czasie.

7.5.1 Pomiar potencjału elektrody jonoselektywnej ERH-115

Porównano wartość potencjału elektrody jonoselektywnej ERH-115 mierzoną przez jonometr laboratoryjny ORION 930 z wartością mierzoną przez moduł urządzenia końcowego. Elektroda ERH-115 jest elektrodą kombinowaną z płaską membraną, przeznaczoną do pomiarów pH. Jej zaletą jest pomiar roztworów o małych objętościach (powyżej 0,1 ml).

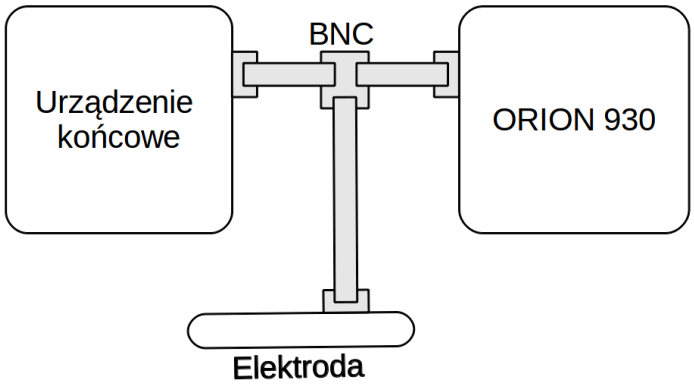
Tablica 7.5: Dane techniczne elektrody ERH-115

Zakres pomiarowy	0...14 pH
Zakres temperatury	5...60°C
Rezystancja membrany	300...1000 MΩ
Punkt zerowy	7,5 ± 0,25 pH
Półogniwo odniesienia	Ag/AgCl, 3 M KCl

Tablica 7.6: Dane techniczne jonometru ORION 930

Impedancja wejściowa	$>10^{12}\Omega$
Zakres napięc wejściowych	-1600,0 mV ... 1600,0 mV
Rozdzielczość	0,1 mV
Dokładność	±0,2 mV

Sygnał z elektrody jonoselektywnej rozdzielono za pomocą trójnika BNC i połączono z jonometrem ORION 930 oraz modułem urządzenia końcowego (schemat 7.6). W ten sposób możliwe było jednoczesne porównanie wyników pomiarów. Moduł urządzenia końcowego posiada impedancję wejściową 1000-krotnie wyższą od jonometru ORION 930, dlatego wspólne podłączenie do jednej elektrody nie miało wpływu na wartość jej potencjału.
W celu zmiany potencjału elektrody do naczynia z badanym roztworem o pH = 10 stopniowo dolewano stężony roztwór o ph = 2. Uzyskano w ten sposób potencjał elektrody w zakresie od ok. -100 mV do ok 280 mV.
Pomiary potencjału elektrody jonoselektywnej są czasochłonne. Po wymieszaniu roztworu i umieszczeniu w nim elektrody należy odczekać co najmniej kilka minut, aby wartość potencjału ustabilizowała się. Minimalny dryft potencjału był zauważalny nawet wtedy.



Rysunek 7.6: Schemat podłączenia podczas pomiarów potencjału

Wykonano 24 serie pomiarowe, po 20 pomiarów potencjału każda. Arkusze pomiarowe przedstawiono w tabelach 7.7, 7.8 i 7.9. Pomiary w rubryce ORION 930 pochodzą od jonometru, a w rubryce ISE od modułu urządzenia końcowego.

	mV		mV		mV		mV		mV		mV		mV	
Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE	
1	-105.5		-109.406	2		-103.6		-109.253	3		-100.2		-108.032	4		-94.2	-93.803
	-105.5		-109.367			-103.8		-109.253			-100.2		-108.299			-94.2	-94.528
	-105.5		-109.406			-103.7		-109.253			-100.2		-108.643			-94.2	-93.536
	-105.5		-109.367			-103.7		-109.253			-100.3		-108.528			-94.2	-94.147
	-105.5		-109.406			-103.8		-109.253			-100.3		-108.070			-94.3	-94.261
	-105.5		-109.406			-103.7		-109.253			-100.3		-108.566			-94.3	-93.880
	-105.5		-109.367			-103.8		-109.253			-100.3		-108.643			-94.3	-94.414
	-105.5		-109.367			-103.8		-109.253			-100.3		-108.528			-94.3	-95.329
	-105.5		-109.406			-103.8		-109.253			-100.4		-108.337			-94.3	-94.948
	-105.5		-109.367			-103.9		-109.253			-100.4		-108.681			-94.4	-94.643
	-105.5		-109.406			-103.9		-109.253			-100.4		-108.681			-94.4	-94.337
	-105.5		-109.367			-103.9		-109.253			-100.4		-108.719			-94.4	-94.109
	-105.5		-109.406			-103.9		-109.291			-100.4		-108.643			-94.4	-94.490
	-105.5		-109.406			-103.9		-109.253			-100.5		-108.795			-94.4	-94.681
	-105.5		-109.367			-103.9		-109.291			-100.5		-108.490			-94.4	-94.528
	-105.5		-109.406			-103.9		-109.291			-100.5		-108.452			-94.4	-93.956
	-105.5		-109.367			-103.9		-109.253			-100.5		-108.681			-94.4	-95.482
	-105.5		-109.406			-103.9		-109.253			-100.5		-108.643			-94.5	-95.291
	-105.5		-109.406			-103.9		-109.253			-100.5		-108.833			-94.5	-95.253
	-105.5		-109.406			-103.9		-109.291			-100.6		-108.795			-94.5	-95.291

	mV		mV		mV		mV		mV		mV		mV	
Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE		Numer serii	ORION 930	ISE	
5	-86.7		-89.149	6	-75.4		-84.648	7	-61.7		-69.962	8	-48.4		-51.041
	-86.7		-89.035		-75.5		-84.114		-61.8		-69.962		-48.4		-51.041
	-86.7		-89.111		-75.5		-84.991		-61.8		-70.000		-48.4		-51.041
	-86.7		-89.111		-75.5		-84.991		-61.9		-70.000		-48.5		-51.041
	-86.7		-89.073		-75.6		-85.564		-61.9		-70.000		-48.5		-51.041
	-86.8		-89.111		-75.7		-85.144		-61.9		-70.000		-48.6		-51.041
	-86.8		-89.111		-75.5		-85.334		-61.9		-70.000		-48.6		-51.041
	-86.8		-89.149		-75.8		-85.182		-62.0		-70.038		-48.7		-51.041
	-86.8		-89.073		-75.8		-85.373		-62.0		-70.000		-48.7		-51.041
	-86.9		-89.149		-75.9		-85.450		-62.1		-70.000		-48.8		-51.041
	-86.9		-89.149		-75.9		-85.678		-62.1		-70.000		-48.8		-51.041
	-86.9		-89.111		-76.0		-85.754		-62.1		-70.038		-48.8		-51.079
	-86.9		-89.149		-76.0		-85.716		-62.2		-70.038		-48.8		-51.041
	-86.9		-89.188		-76.0		-85.640		-62.2		-70.000		-48.9		-51.041
	-86.9		-89.111		-76.1		-85.983		-62.3		-70.038		-48.9		-51.079
	-87.0		-89.149		-76.1		-85.564		-62.3		-70.038		-49.0		-51.079
	-87.0		-89.149		-76.1		-85.640		-62.4		-70.038		-49.0		-51.079
	-87.0		-89.188		-76.1		-85.564		-62.4		-70.038		-49.1		-51.079
	-87.0		-89.111		-76.2		-85.907		-62.4		-70.038		-49.1		-51.079
	-87.1		-89.149		-76.2		-86.021		-62.4		-70.038		-49.2		-51.079

Rysunek 7.7: Arkusz pomiarowy mierzonych potencjałów elektrody

Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE
9	-23.3	-29.945	10	7.0	7.401	11	22.3	14.572	12	40.0	31.509
	-23.4	-29.907		6.9	7.706		22.3	14.915		40.0	31.128
	-23.4	-30.022		6.8	7.629		22.2	14.420		40.0	31.471
	-23.6	-29.869		6.8	7.477		22.2	14.458		39.9	31.586
	-23.6	-29.984		6.7	7.210		22.2	14.801		39.8	31.433
	-23.7	-30.174		6.6	7.286		22.2	14.572		39.8	31.509
	-23.8	-30.060		6.5	7.172		22.2	14.725		39.7	31.548
	-23.8	-30.174		6.5	6.714		22.1	14.343		39.7	31.052
	-23.9	-30.212		6.4	6.790		22.0	14.572		39.6	31.395
	-24.0	-30.098		6.3	6.638		22.0	14.343		39.6	31.433
	-24.1	-30.098		6.2	7.095		22.0	14.420		39.5	31.013
	-24.1	-30.174		6.1	6.523		21.9	14.153		39.5	30.899
	-24.2	-30.212		6.0	6.485		21.8	13.695		39.4	30.899
	-24.3	-30.212		5.9	6.447		21.8	13.847		39.3	30.052
	-24.4	-30.251		5.9	5.646		21.7	13.924		39.2	30.975
	-24.4	-30.212		5.8	6.256		21.7	14.458		39.1	30.632
	-24.5	-30.251		5.7	6.332		21.6	13.733		39.1	30.078
	-24.6	-30.251		5.5	6.180		21.6	13.657		39.0	30.899
	-24.6	-30.289		5.3	5.112		21.5	13.809		38.8	30.327
	-24.7	-30.251		5.2	5.760		21.5	13.542		38.7	30.518

Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE	Numer serii	mV ORION 930	mV ISE
13	59.4	49.744	14	78.5	69.656	15	117.7	109.100	16	151.7	163.574
	59.3	49.820		78.4	69.962		117.6	108.757		151.3	163.574
	59.3	49.667		78.3	68.817		117.5	108.185		151.1	163.536
	59.3	49.744		78.3	69.313		117.4	108.604		150.9	163.460
	59.3	49.438		78.2	69.084		117.3	107.880		150.5	163.460
	59.3	49.820		78.1	68.893		117.2	108.757		150.1	163.307
	59.3	49.896		78.1	68.855		117.1	108.681		149.9	163.040
	59.3	49.744		78.0	68.169		117.1	108.299		149.7	162.888
	59.2	49.553		77.9	68.703		117.0	107.994		149.4	163.002
	59.2	49.896		77.8	68.550		116.9	107.689		149.2	162.888
	59.2	49.533		77.7	68.474		116.8	108.070		149.0	162.849
	59.1	49.362		77.6	68.245		116.6	107.727		148.8	162.468
	59.1	49.324		77.5	68.054		116.5	107.727		148.6	162.354
	59.0	49.477		77.5	67.940		116.4	107.231		148.3	162.315
	59.0	49.095		77.4	67.863		116.3	107.346		148.0	162.239
	58.9	49.095		77.3	67.863		116.2	107.193		147.8	162.086
	58.9	49.019		77.2	67.673		116.1	107.155		147.5	162.048
	58.8	49.057		77.2	67.902		116.0	107.002		147.3	161.896
	58.8	48.981		77.0	67.825		115.9	107.117		147.1	161.705
	58.7	49.133		77.0	67.635		115.8	106.812		146.8	161.858

Rysunek 7.8: Arkusz pomiarowy mierzonych potencjałów elektrody

	mV	mV			mV	mV			mV	mV			mV	
Numer serii	ORION 930	ISE	Numer serii		ORION 930	ISE	Numer serii		ORION 930	ISE	Numer serii		ORION 930	ISE
17	188.1	183.334		18	205.1		196.190	19	218.3		213.470	20	231.4	
	187.9	183.372			205.0		195.999		218.2		212.135		231.4	
	187.7	183.182			204.9		195.961		218.2		212.708		231.3	
	187.5	183.182			204.7		195.847		218.1		212.250		231.3	
	187.3	183.067			204.5		195.541		218.0		212.669		231.3	
	187.0	182.838			204.4		195.541		217.9		212.173		231.3	
	186.8	182.419			204.2		195.198		217.9		212.288		231.2	
	186.5	182.495			204.0		195.274		217.8		212.212		231.2	
	186.1	181.999			203.8		195.274		217.7		211.105		231.2	
	185.9	181.618			203.7		194.626		217.6		211.105		231.2	
	185.7	181.541			203.5		195.007		217.5		211.525		231.2	
	185.5	181.732			203.4		194.473		217.4		211.258		231.2	
	185.3	181.274			203.1		194.550		217.3		210.190		231.1	
	185.1	181.198			203.0		194.244		217.3		210.075		231.1	
	184.9	180.702			202.9		194.397		217.2		210.419		231.1	
	184.7	180.550			202.7		194.359		217.1		209.885		231.1	
	184.6	180.206			202.6		194.206		217.0		210.190		231.1	
	184.4	180.390			202.4		194.092		216.9		209.732		231.0	
	183.9	179.634			202.3		194.054		216.8		209.732		231.0	
	183.7	178.757			202.1		194.054		216.7		209.389		231.0	

	mV	mV			mV	mV			mV	mV			mV	
Numer serii	ORION 930	ISE	Numer serii		ORION 930	ISE	Numer serii		ORION 930	ISE	Numer serii		ORION 930	ISE
21	239.0	237.387		22	255.9		248.260	23	268.5		261.955	24	279.5	
	239.0	237.465			255.9		248.375		268.5		261.933		279.3	
	239.1	237.236			255.9		248.337		268.4		261.955		279.1	
	239.0	236.511			255.9		248.260		268.4		261.993		278.9	
	239.0	237.961			255.8		247.765		268.4		261.955		278.8	
	239.1	237.427			255.8		248.108		268.3		261.955		278.6	
	239.0	237.198			255.8		248.070		268.3		261.917		278.3	
	239.0	236.969			255.8		248.337		268.3		261.879		278.2	
	239.0	237.350			255.8		248.498		268.2		261.955		278.0	
	239.0	237.855			255.8		247.803		268.2		261.955		277.9	
	239.0	237.617			255.8		247.726		268.2		261.955		277.7	
	239.0	237.427			255.8		248.260		268.2		261.955		277.6	
	239.0	237.503			255.8		248.032		268.1		261.955		277.5	
	239.0	237.427			255.8		248.032		268.1		261.955		277.4	
	239.0	236.549			255.8		248.337		268.0		261.917		277.3	
	238.9	236.244			255.8		247.726		268.0		261.917		277.1	
	238.9	236.244			255.8		248.032		268.0		261.917		277.0	
	238.9	237.579			255.7		247.688		267.9		261.917		276.9	
	238.9	237.236			255.7		247.803		267.9		261.917		276.7	
	238.8	237.007			255.7		247.993		267.9		261.917		276.6	

Rysunek 7.9: Arkusz pomiarowy mierzonych potencjałów elektrody

Wyniki pomiarów zostały opracowane. Obliczono średnią wartość potencjału oraz oszacowano niepewność tej średniej dla każdej z serii pomiarowych. Następnie wyznaczono średni błąd dla każdej serii.
Każda seria to 20 pomiarów, dlatego aby oszacować niepewność (odchylenie standardowe) należy skorzystać z rozkładu Studenta.

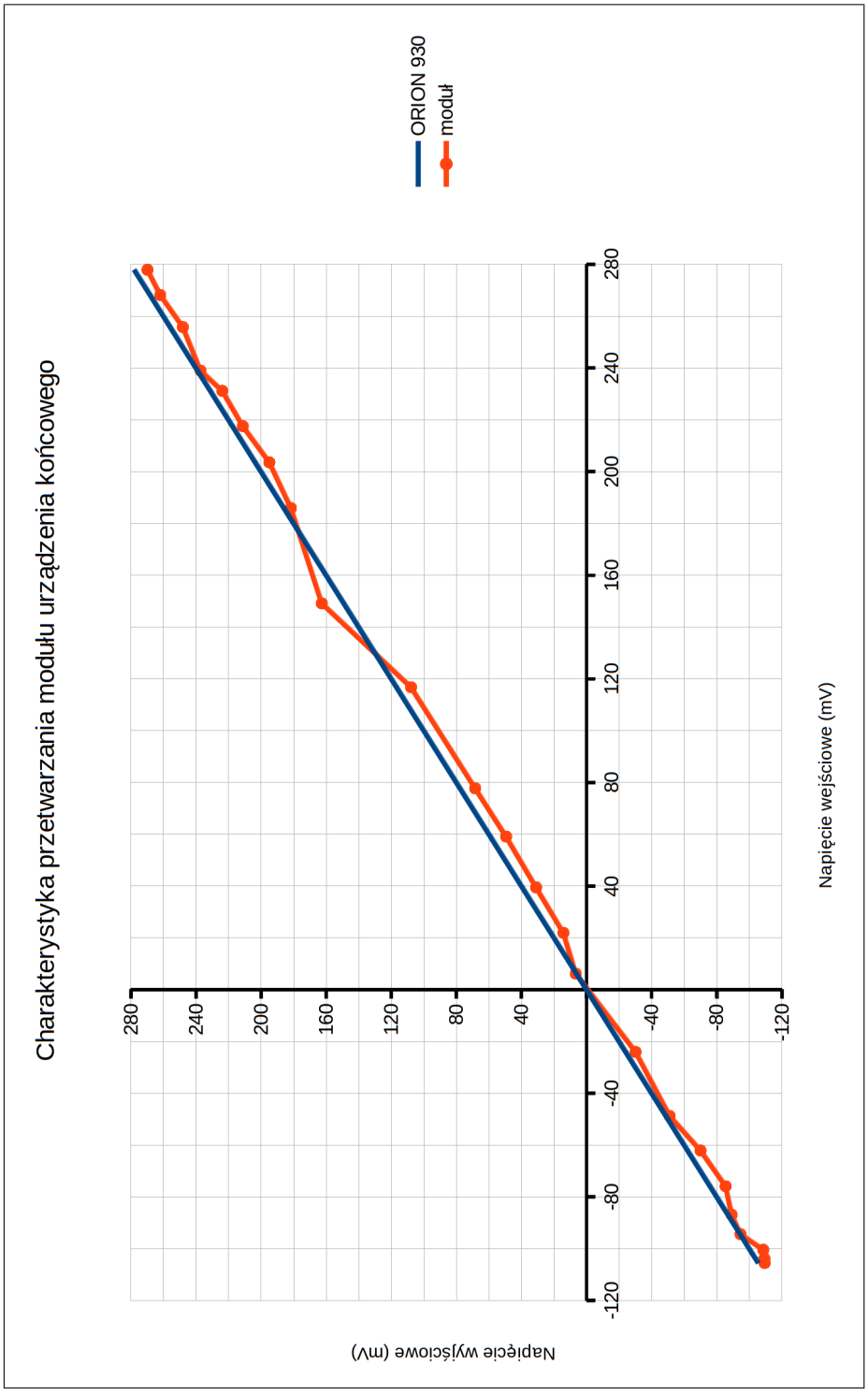
$$u(\bar{x}) = t_n \cdot \sqrt{\frac{\sum (x_i - \bar{x})^2}{n \cdot (n-1)}} \tag{7.2}$$

gdzie:
t_n - wartość krytyczna rozkładu Studenta
t_n = 1,7291 dla poziomu istotności a = 0,1 i próby n = 20
x_i - wartość pomiarowa
$\bar{x}$ - średnia wartość
n - liczba pomiarów

Seria	Średnia ORION 930	Niepewność ORION 930	Średnia ISE	Niepewność ISE	Średni błąd
1	-105.500	0.000	-109.390	0.008	-3.890
2	-103.830	0.036	-109.261	0.006	-5.431
3	-100.385	0.046	-108.553	0.086	-8.168
4	-94.350	0.039	-94.545	0.220	-0.195
5	-86.860	0.048	-89.126	0.015	-2.266
6	-75.845	0.103	-85.413	0.181	-9.568
7	-62.090	0.088	-70.013	0.010	-7.923
8	-48.760	0.096	-51.054	0.007	-2.294
9	-24.020	0.169	-30.132	0.049	-6.112
10	6.205	0.207	6.693	0.269	0.488
11	21.940	0.105	14.248	0.162	-7.692
12	39.485	0.154	31.018	0.189	-8.467
13	59.120	0.081	49.470	0.123	-9.650
14	77.750	0.185	68.474	0.261	-9.276
15	116.770	0.231	107.866	0.263	-8.904
16	149.150	0.574	162.727	0.246	13.577
17	185.930	0.528	181.675	0.517	-4.256
18	203.615	0.371	194.944	0.280	-8.671
19	217.545	0.193	211.226	0.470	-6.320
20	231.185	0.047	223.858	0.036	-7.327
21	238.980	0.027	237.210	0.189	-1.770
22	255.805	0.023	248.072	0.098	-7.733
23	268.190	0.076	261.939	0.010	-6.251
24	277.920	0.343	269.823	0.418	-8.097

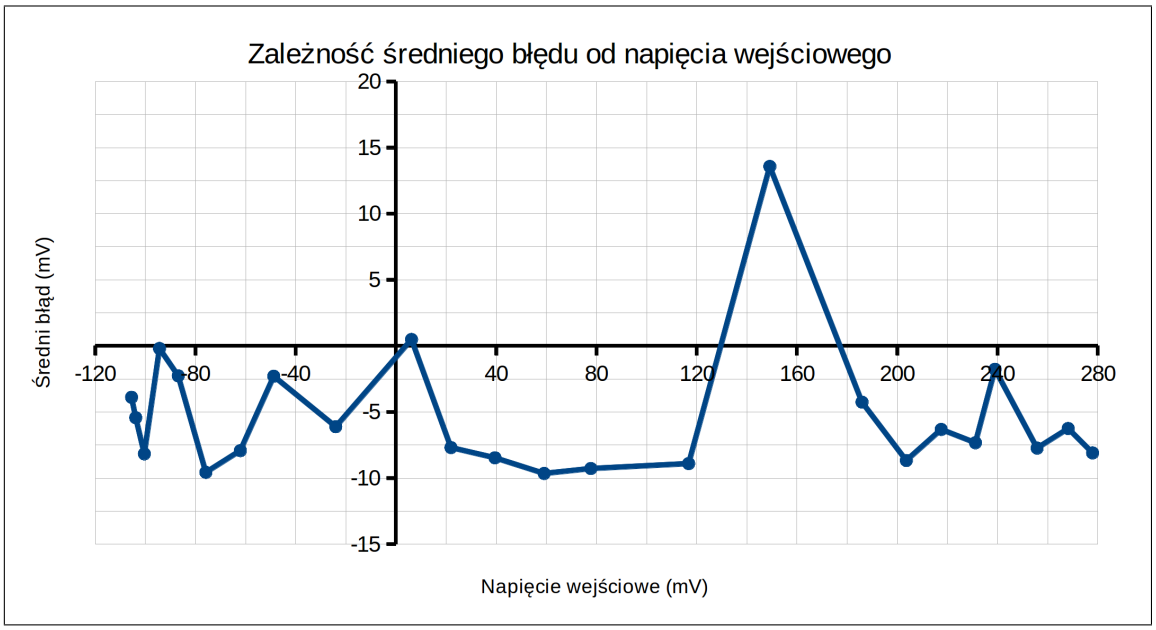
Rysunek 7.10: Arkusz z uśrednionymi pomiarami potencjału

Wykreślono charakterystykę przetwarzania modułu urządzenia końcowego. Jako punkty charakterystyki idealnej przyjęto pomiary wykonane przez jonometr ORION 930.



Rysunek 7.11: Charakterystyka przetwarzania modułu urządzenia końcowego

Charakterystyka nie jest liniowa, widoczne są błedy przetwarzania, które przedstawiono na rysunku 7.12.



Rysunek 7.12: Błąd przetwarzania modułu urządzenia końcowego

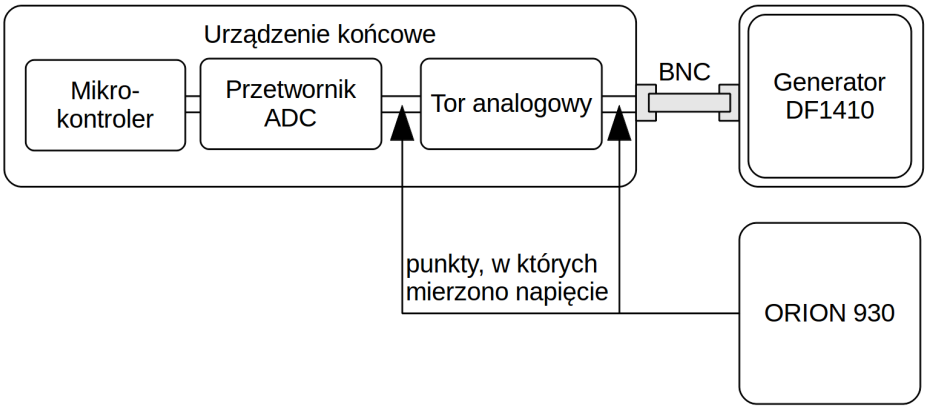
Gdyby średni błąd był niezależny od napięcia wejściowego, wtedy byłby on jak najbardziej akceptowalny. Taki błąd byłby po prostu napięciem przesunięcia (offsetu) wprowadzonym przez elementy toru analogowego modułu. Niewielka zmiana nachylenia charakterystyki przetwarzania świadczyłaby o niejednostkowym wzmocnieniu takich układów jak wzmacniacz pomiarowy INA116 lub wzmacniacz operacyjny kondycjonujący sygnał.

Wyniki pomiarów świadczą jednak o nieliniowym błędzie, a wartość tego błędu jest wielokrotnie większa od nieliniowości całkowej przetwornika analogowo-cyfrowego, która wynosi INL = ±2 LSB, czyli około ±76 uV.

Przyczyną tak dużej wartości błędów pomiarowych mógł być, albo przetwornik analogowo-cyfrowy, albo jakiś element toru analogowego modułu (wzmacniacz pomiarowy INA116, układ kondycjonujący, źródło napięcia odniesienia, przetwornica DC/DC). Wykonano stosowne pomiary opisane w kolejnym podrozdziale.

7.5.2 Pomiar napięcia z generatora

W celu sprawdzenia czy źródłem błędów jest tor analogowy urządzenia końcowego wykonano pomiary napięcia na wejściu toru (gniazdo BNC) oraz na jego wyjściu (w miejscu gdzie sygnał podawany jest na wejście przetwornika analogowo-cyfrowego). Napięcie podawano z generatora DF1410, a mierzono je za pomocą jonometru ORION 930 ustawionego w tryb pomiaru napięcia.



Rysunek 7.13: Schemat podłączenia podczas pomiarów toru analogowego

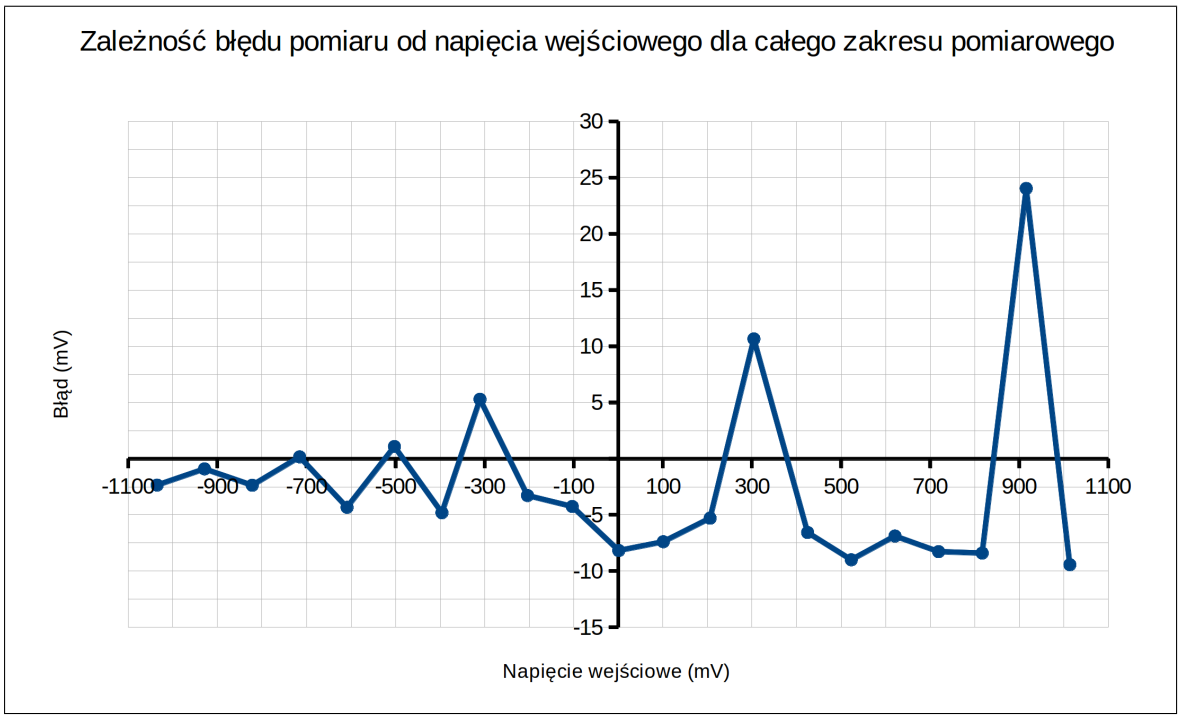
Wykonano pomiar w całym zakresie napięć wejściowych modułu urządzenia końcowego tzn. od -1000 mV do 1000 mV, co 100 mV. Wyniki pomiarów przedstawiono w arkuszu 7.14. Błąd toru analogowego jest różnicą między napięciem na wejściu BNC a napięciem na wejściu przetwornika analogowo-cyfrowego. Widać wyraźnie, że błąd toru analogowego wynosi ok. -1 mV i nie zależy od wartości napięcia wejściowego. Tor analogowy nie jest więc źródłem dużych błędów.

Napięcie na wejściu BNC modułu urządzenia końcowego	Napięcie na wejściu przetwornika analogowo-cyfrowego	Wynik pomiaru modułu urządzenia końcowego	Błąd toru analogowego	Błąd pomiaru modułu urządzenia końcowego
1013.8	1012.8	1004.372	-1.00	-9.428
915.9	914.9	939.941	-1.00	24.041
817.0	815.9	808.601	-1.10	-8.399
719.1	718.1	710.831	-1.00	-8.269
621.2	620.1	614.281	-1.06	-6.879
523.1	522.0	514.107	-1.10	-8.993
425.0	423.8	418.434	-1.20	-6.566
304.4	303.4	315.056	-1.00	10.656
206.1	205.1	200.806	-1.00	-5.294
101.0	99.9	93.613	-1.10	-7.387
1.0	0.0	-7.172	-1.00	-8.172
-103.4	-104.4	-107.651	-1.00	-4.251
-203.9	-204.9	-207.176	-1.00	-3.276
-310.5	-311.6	-305.214	-1.10	5.286
-396.0	-397.0	-400.810	-1.00	-4.810
-502.6	-503.6	-501.518	-1.00	1.082
-609.0	-610.0	-613.327	-1.00	-4.327
-715.4	-716.3	-715.239	-0.90	0.161
-821.8	-822.7	-824.165	-0.90	-2.365
-929.0	-930.0	-929.909	-1.00	-0.909
-1035.4	-1036.3	-1037.750	-0.90	-2.350

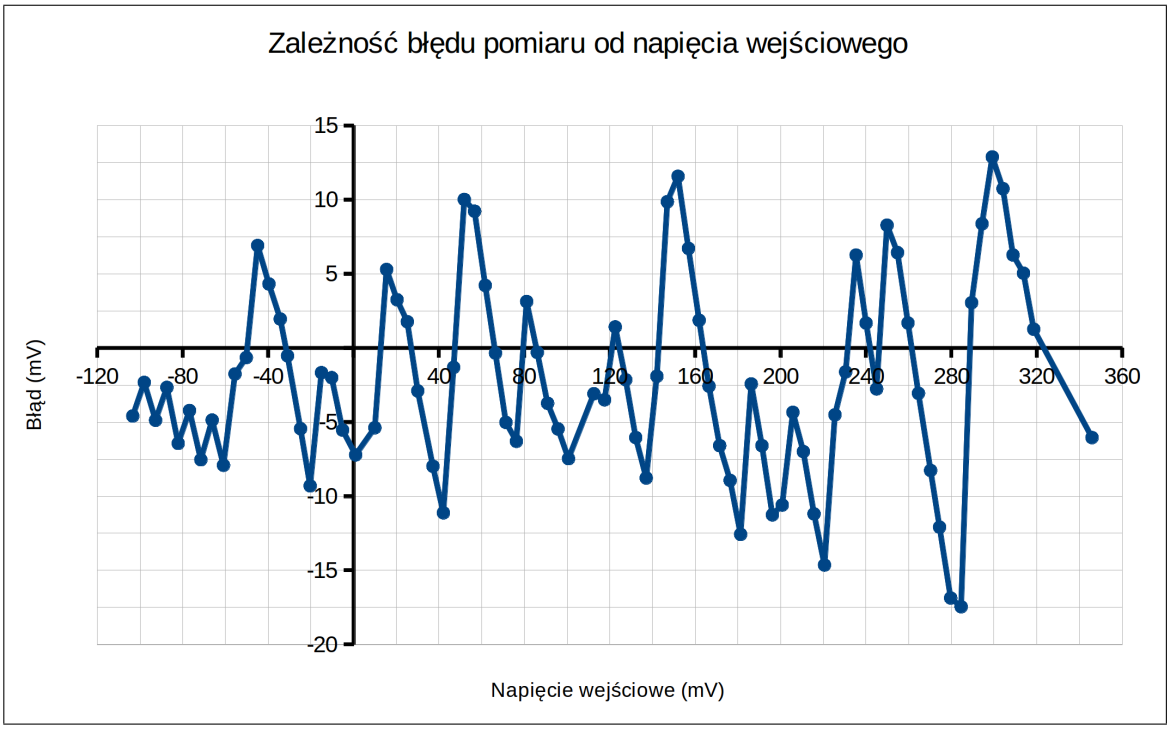
Rysunek 7.14: Arkusz z pomiarami toru analogowego dla całego zakresu pomiarowego

Wykreślono błąd pomiaru dla całego zakresu pomiarowego modułu urządzenia końcowego (wykres 7.15). Zaobserwowano, że wartość bezwzględna tego błędu rośnie wraz ze wzrostem napięcia wejściowego. Tak duża niedokładność pomiaru jest spowodowana wadliwym działaniem przetwornika analogowo-cyfrowego. Analizując jego 16-bitowy wynik pomiaru uznano, że drugi bajt (najmniej znaczący) nigdy nie przyjmuje wszystkich wartości z zakresu 0-255, a jedynie kilkadziesiąt wartości np. 31, 39, 47, 60, 175, 180, 199, 207, 255. Bardzo prawdopodobne jest, że rejestr sukcesywnej aproksymacji SAR, jest uszkodzony.

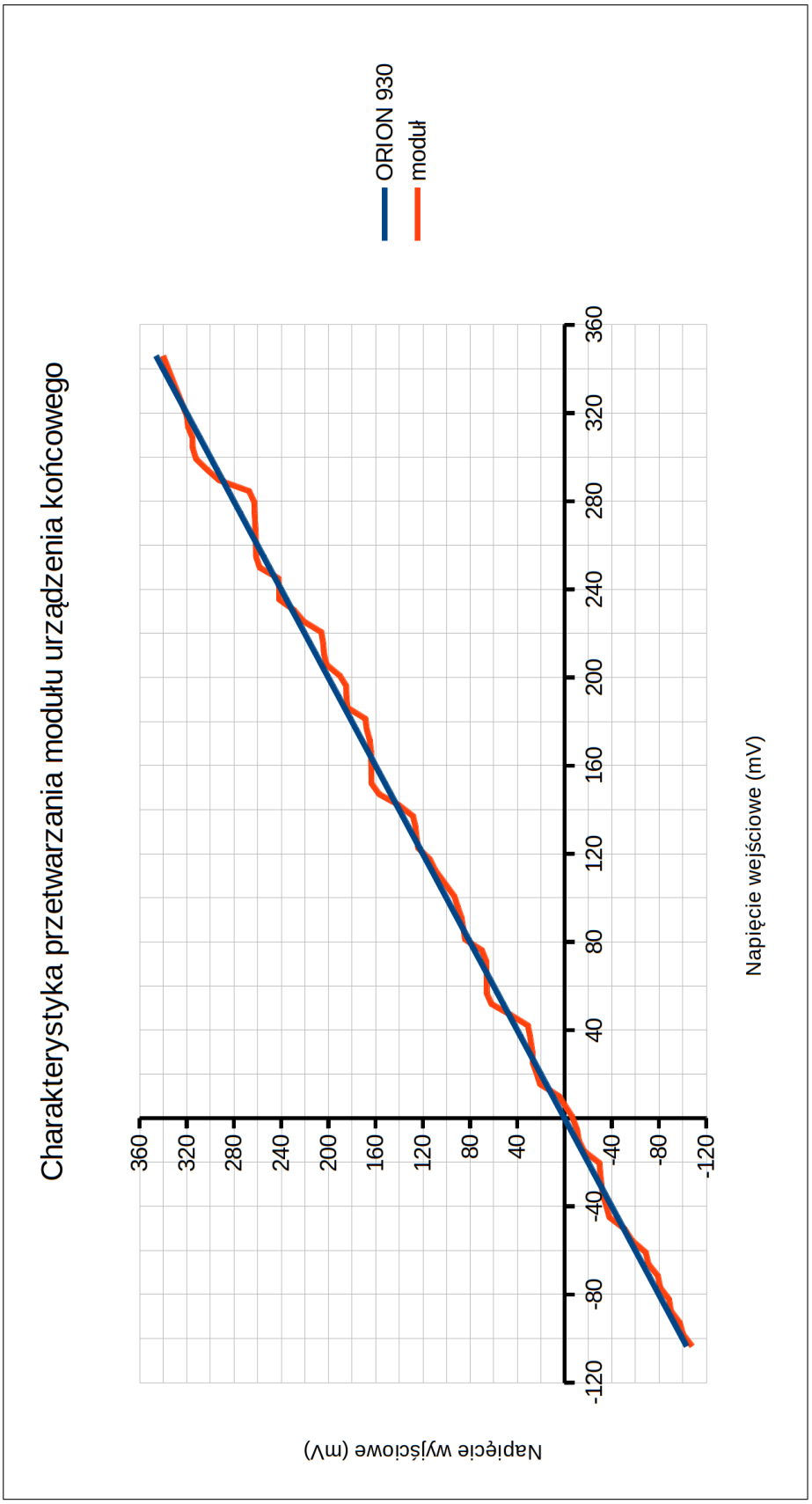
Wymieniono więc układ elektroniczny na nowy. Wykonano pomiar napięcia z generatora w zakresie napięć typowych dla elektrody jonoselektywnej ERH-115. Używając generatora przyśpieszono proces pomiaru i unikniono konieczności przygotowywania roztworów o określonych stężeniach. Wyniki pomiarów przedstawiono w arkuszu 7.18. Wykreślono błąd pomiaru (rysunek 7.16) oraz charakterystykę przetwarzania modułu urządzenia końcowego (rysunek 7.17).



Rysunek 7.15: Błąd pomiaru dla całego zakresu pomiarowego



Rysunek 7.16: Błąd pomiaru po wymianie ADC



Rysunek 7.17: Charakterystyka przetwarzania po wymianie ADC

Napięcie na wejściu BNC modułu urządzenia końcowego	Wynik pomiaru modułu urządzenia końcowego	Błąd pomiaru modułu urządzenia końcowego	Napięcie na wejściu BNC modułu urządzenia końcowego	Wynik pomiaru modułu urządzenia końcowego	Błąd pomiaru modułu urządzenia końcowego
345.9	339.851	-6.049	117.6	114.098	-3.502
318.6	319.862	1.262	112.6	109.506	-3.094
313.8	318.841	5.041	100.7	93.231	-7.469
308.9	315.170	6.270	95.8	90.332	-5.468
304.2	314.941	10.741	90.9	87.166	-3.734
299.2	312.080	12.880	86.1	85.793	-0.307
294.4	302.773	8.373	81.1	84.229	3.129
289.5	292.549	3.049	76.3	70.000	-6.300
284.6	267.143	-17.457	71.4	66.376	-5.024
279.7	262.833	-16.867	66.5	66.147	-0.353
274.5	262.413	-12.087	61.7	65.918	4.218
270.3	262.032	-8.268	56.7	65.918	9.218
264.6	261.536	-3.064	51.9	61.913	10.013
259.7	261.383	1.683	46.9	45.586	-1.314
254.8	261.230	6.430	42.1	30.975	-11.125
249.9	258.179	8.279	37.2	29.221	-7.979
245.0	242.233	-2.767	30.1	27.199	-2.901
240.1	241.776	1.676	25.2	26.970	1.770
235.4	241.661	6.261	20.4	23.651	3.251
230.5	228.882	-1.618	15.5	20.790	5.290
225.5	220.985	-4.515	10.0	4.616	-5.384
220.6	205.956	-14.644	1.0	-6.218	-7.218
215.7	204.506	-11.194	-5.1	-10.643	-5.543
210.7	203.705	-6.995	-10.2	-12.207	-2.007
205.8	201.454	-4.346	-15.0	-16.670	-1.670
200.8	190.201	-10.599	-20.3	-29.602	-9.302
196.2	184.937	-11.263	-24.8	-30.251	-5.451
191.3	184.708	-6.592	-30.9	-31.433	-0.533
186.3	183.868	-2.432	-34.3	-32.349	1.951
181.3	168.724	-12.576	-39.6	-35.286	4.314
176.4	167.465	-8.935	-44.9	-37.994	6.906
171.5	164.909	-6.591	-50.2	-50.850	-0.650
166.5	163.918	-2.582	-55.5	-57.259	-1.759
161.9	163.765	1.865	-60.9	-68.817	-7.917
156.9	163.612	6.712	-66.2	-71.068	-4.868
152.0	163.574	11.574	-71.5	-79.041	-7.541
147.0	156.860	9.860	-76.8	-81.024	-4.224
142.1	140.190	-1.910	-82.1	-88.539	-6.439
137.1	128.326	-8.774	-87.4	-90.065	-2.665
132.2	126.152	-6.048	-92.7	-97.580	-4.880
127.5	125.351	-2.149	-98.0	-100.327	-2.327
122.6	124.016	1.416	-103.4	-107.994	-4.594

Rysunek 7.18: Arkusz z pomiarami napięcia po wymianie ADC

Niestety wymieniony przetwornik analogowo-cyfrowy zachowuje się w podobny sposób, jego dokładność jest bardzo niska. Według danych technicznych przetwornik posiada rozdzielczość 16 bit, zwraca wynik w postaci 16-bitowej liczby, jednak w rzeczywistości jego dokładność jest zbliżona do przetwornika 6 lub 7-bitowego. Wszystko wskazuje na to, że obydwa układy pochodzą z tej samej wadliwej serii produkcyjnej.

Z powodu późnego wykrycia usterki (w tym samym czasie implementowano aplikację PC oraz rozwijano oprogramowanie modułów bezprzewodowych), nie było możliwe sprowadzenie na czas nowych układów MAX1169 od producenta.
Fakt późnego wykrycia usterki spowodowany był także brakiem precyzyjnego woltomierza w czasie wstępnych testów zmontowanego modułu urządzenia końcowego. Nie spodziewano się również wadliwego działania nowych układów scalonych.

W związku z powyższym, za dokładność pomiaru modułu urządzenia końcowego przyjęto jego maksymalny błąd wynoszący ± 25 mV.

7.6 Parametry techniczne bezprzewodowego systemu pomiarowego

Parametry bezprzewodowego systemu pomiarowego wyznaczone zostały bazując na: danych technicznych elementów elektronicznych wchodzących w skład modułów oraz wynikach pomiarów przeprowadzonych w celu określenia zasięgu, poboru prądu i niepewności pomiarowych.

Tablica 7.7: Parametry techniczne modułu koordynatora

Napięcie zasilania	5 V USB
Pobór prądu	33 mA
Zasięg (dla RSSI -85 dBm)	110 m
Prędkość transmisji szeregowej	38400 baud

Tablica 7.8: Parametry techniczne modułu urządzenia końcowego

Napięcie zasilania	7,2 ÷ 9 V 6xAA
Pobór prądu	
Tryb auto	77 mA
Tryb sleep 0	42 mA
Tryb sleep 1	28 mA
Tryb sleep 2	22 mA
Czas pracy na pakiecie baterii 1000 mAh	
Tryb auto	13 h
Tryb sleep 0	24 h
Tryb sleep 1	35 h
Tryb sleep 2	46 h
Zasięg (dla RSSI -85 dBm)	110 m
Zakres napięć mierzonych	-1000 mV ÷ 1000 mV
Impedancja wejściowa	$10^{15}\Omega$
Rozdzielczość pomiaru	38 uV
Dokładność pomiaru	±25 mV

Rozdział 8

Podsumowanie

Cele projektu zrealizowano w całości:

- wykonano elektroniczny interfejs pomiarowy dla elektrod jonoselektywnych,
- napisano oprogramowanie do komunikacji bezprzewodowej interfejsu,
- napisano aplikację na komputer PC dla użytkownika interfejsu,
- wykonano pomiary określające parametry interfejsu.

Podczas prac nad projektem wykorzystano wiedzę nabytą w trakcie studiów z zakresu: programowania, elektroniki oraz systemów pomiarowych.

Projekt z powodzeniem może być wykorzystany jako stanowisko pomiarowe do badania stężenia jonów w roztworach, ponieważ oprócz obszernej dokumentacji posiada krótką i zwięzłą instrukcję obsługi. Interfejs pomiarowy został umieszczony w obudowach, co zapewnia odporność na uszkodzenia i pracę w warunkach laboratoryjnych.

Możliwości rozwoju projektu jest wiele, na przykład:

- poprawa dokładności pomiaru napięcia poprzez wymianę przetwornika analogowo-cyfrowego na nowy,
- rozbudowa oprogramowania interfejsu oraz aplikacji PC w celu stworzenia sieci urządzeń końcowych wysyłających dane do koordynatora,
- dalsze zwiększenie energooszczędności modułu urządzenia końcowego.

Bibliografia

[1] Kozyra Andrzej: Pomiary w Biotechnologii. Laboratorium. Pomiary potencjometryczne, jonoselektywne, Gliwice 2005.

[2] Laboratorium Miernictwa Przemysłowego, pod red. Jerzego Frączka i Stanisława Walusia, Wydawnictwo Politechniki Śląskiej, Gliwice 2002, ISBN 83-7335-146-9.

[3] Chris C Rundle: Beginners Guide to Ion-Selective Electrode Measurements, London 2011.

[4] Wasik Andrzej, Konieczka Piotr: Chemia Analityczna. Wybrane metody elektroanalityczne (potencjometria, konduktometria, kulometria). Materiały do ćwiczeń, Gdańsk 2002, wydanie II uzupełnione.

[5] Piernikarczyk Artur: Bezprzewodowy pomiar napięcia za pomocą transmisji ZigBee i układu ATZB-24, Gliwice 2014.

[6] Lundh Fredrik. Python thread synchronization. [artykuł][dostęp z dnia 6 sierpień 2015]
http://effbot.org/zone/thread-synchronization.htm

[7] Turczak Bogdan. Analiza niepewności pomiarowych. [artykuł][dostęp z dnia 13 sierpień 2015]
http://labor.zut.edu.pl/e/er7.html

[8] Podział metod elektrochemicznych [hasło][dostęp z dnia 4 stycznia 2015]
http://www.pg.gda.pl/chem/Dydaktyka/Analityczna/ELECTRO/ech_wyk.pdf

[9] Membranowe elektrody jonoselektywne i ich zastosowanie w analityce [hasło][dostęp z dnia 4 stycznia 2015]
http://www.technologia.gda.pl/dydaktyka/index/l/tchmemb_tch/pdf_z/cw4ElemtrodyJSE.pdf

[10] Potencjometryczna metoda oznaczania chlorków z zastosowaniem elektrody jonoselektywnej [hasło][dostęp z dnia 4 stycznia 2015]

http://www.umb.edu.pl/photo/pliki/WF_jednostki/zaklad-chemii-fizycznej/cwiczenie_5_an_-_chlorki.pdf

[11] IrDA [hasło][dostęp z dnia 24 sierpień 2015]
https://pl.wikipedia.org/wiki/Infrared_Data_Association

[12] ZigBee [hasło][dostęp z dnia 24 sierpień 2015]
https://en.wikipedia.org/wiki/ZigBee

[13] Python [hasło][dostęp z dnia 3 sierpień 2015]
https://pl.wikipedia.org/wiki/Python

[14] GTK [hasło][dostęp z dnia 3 sierpień 2015]
https://pl.wikipedia.org/wiki/GTK%2B

[15] Matplotlib [oprogramowanie][dostęp z dnia 3 sierpień 2015]
http://matplotlib.org/

[16] pySerial [oprogramowanie][dostęp z dnia 3 sierpień 2015]
http://pyserial.sourceforge.net/

[17] Glade [oprogramowanie][dostęp z dnia 3 sierpień 2015]
https://glade.gnome.org/

[18] pyCharm [oprogramowanie][dostęp z dnia 3 sierpień 2015]
https://www.jetbrains.com/pycharm/

[19] Atmel Studio [oprogramowanie][dostęp z dnia 20 sierpień 2015]
http://www.atmel.com/tools/atmelstudio.aspx

[20] ATZB-24-A2 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.atmel.com/images/atmel-8226-zigbit-2.4ghz-wireless-modules-atzb-24-a2b0_datasheet.pdf

[21] LP2950 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.ti.com/lit/ds/symlink/lp2950-n.pdf

[22] IRLML6402 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.irf.com/product-info/datasheets/data/irlml6402.pdf

[23] LE33CD [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.farnell.com/datasheets/1811320.pdf

[24] AM1P-0515DZ [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.aimtec.com/site/Aimtec/files/Datasheet/HighResolution/AM1P-Z.pdf

[25] LM78L12 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.ti.com/lit/ds/symlink/lm78l05.pdf

[26] LM79L12 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.ti.com/lit/ds/symlink/lm79l05.pdf

[27] INA116 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://www.ti.com/lit/ds/symlink/ina116.pdf

[28] LT1367 [dokumentacja techniczna] [dostęp z dnia 19 sierpień 2015]
http://cds.linear.com/docs/en/datasheet/1366fb.pdf

[29] DS18b20 [dokumentacja techniczna] [dostęp z dnia 20 sierpień 2015]
http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[30] KamPROG [dokumentacja techniczna] [dostęp z dnia 20 sierpień 2015]
http://kamami.com/programmers/50609-kamprog-for-avr.html

[31] LwMesh [dokumentacja techniczna] [dostęp z dnia 20 sierpień 2015]
http://www.atmel.com/tools/lightweight_mesh.aspx

[32] USB UART Manual, MikroElektronika [dokumentacja techniczna] [dostęp z dnia 20 sierpień 2015]
http://www.mikroe.com/downloads/get/1560/usb_uart_manual_v100.pdf

Dodatek A - Artykuł: „The Monte Carlo method in determining the measurement uncertainty of electronic measuring device”