

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ ELEKTRONIKI



PRACA DYPLOMOWA

**Bezprzewodowy rozproszony system pomiaru
warunków środowiskowych**

inż. Piotr Gajcy

ELEKTRONIKA I TELEKOMUNIKACJA

SYSTEMY TELEKOMUNIKACYJNE

NIESTACJONARNE STUDIA DRUGIEGO STOPNIA – MAGISTERSKIE

ppłk dr inż. Tadeusz Sondej

WARSZAWA 2021

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ ELEKTRONIKI
INSTYTUT SYSTEMÓW ŁĄCZNOŚCI

"AKCEPTUJĘ"
DZIEKAN
WYDZIAŁU ELEKTRONIKI

prof. dr hab. inż. Ryszard SZPLET

dnia 5. MAR. 2021 r.

ZADANIE do pracy dyplomowej

Wydane studentowi

inż. Piotr GAJCY

(stopień, tytuł zawodowy, imiona i nazwisko)

ELEKTRONIKA I TELEKOMUNIKACJA

(kierunek studiów)

NIESTACJONARNE STUDIA DRUGIEGO STOPNIA - MAGISTERSKIE

(forma i rodzaj studiów)

I. Temat pracy: **Bezprzewodowy rozproszony system pomiaru warunków środowiskowych**

II. Treść zadania:

1. Sensory do pomiaru warunków środowiskowych.
2. Sposoby komunikacji w rozproszonych systemach pomiarowych.
3. Projekt i wykonanie modelu systemu pomiarowego.
4. Opracowanie oprogramowania sterującego i monitorującego.
5. Wykonanie badań testowych.
6. Opracowanie wniosków. Redakcja tekstu pracy.

III. W rezultacie wykonania pracy należy przedstawić:

- a/ notatkę objaśniającą z obliczeniami
- b/ wykresy: niezbędne do realizacji pracy
- c/ rysunki: niezbędne do realizacji pracy

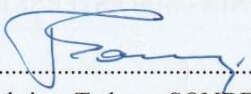
IV. Konsultant:

V. Opiekun merytoryczny: prof. dr hab. Ryszard SZPLET

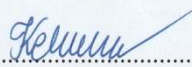
VI. Termin zdania przez studenta ukończonej pracy: 26.05.2021r.

VII. Data wydania zadania: 06.03.2021r.

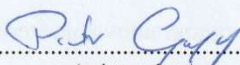
PROMOTOR PRACY DYPLOMOWEJ


.....
ppłk dr inż. Tadeusz SONDEJ

DYREKTOR
INSTYTUTU SYSTEMÓW ŁĄCZNOŚCI


.....
ppłk dr hab. inż. Jan KELNER

Zadanie otrzymałem/am dnia 27.04.2021 r.


.....
(podpis studenta)

Spis treści

1. Wstęp	7
2. Sensory do pomiaru warunków środowiskowych	8
2.1. Wstęp.....	8
2.2. Czujniki ciśnienia.....	8
2.3. Czujniki temperatury.....	10
2.3.1. Czujniki rezystancyjne	10
2.3.2. Termopara.....	11
2.3.3. Czujniki bimetaliczne	11
2.4. Czujniki wilgotności	12
2.4.1. Higrometry włosowe	13
2.4.2. Higrometry oparte na zasadzie przewodnictwa cieplnego	14
2.4.3. Higrometry pojemnościowe	14
3. Sposoby komunikacji w rozproszonych systemach pomiarowych.....	15
3.1. Wstęp.....	15
3.2. Interfejsy przewodowe	15
3.2.1. Interfejs RS-232.....	15
3.2.2. Interfejs CAN	16
3.2.3. Interfejs I ² C	17
3.3. Interfejsy bezprzewodowe	18
3.3.1. Interfejs IrDA	18
3.3.2. Radio.....	19
3.3.3. Interfejs ZigBee	21
3.3.4. Sieć komórkowa GSM	22
3.3.5. Interfejs WiFi.....	23
3.3.6. Interfejs LoRa	24
3.3.7. Interfejs Bluetooth	25
4. Projekt i wykonanie modelu systemu pomiarowego	30
4.1. Wstęp.....	30
4.2. Projekt urządzenia	30
4.2.1. Moduł ESP32 DevKit ESP-WROOM-32 V2.....	30
4.2.2. Czujnik BME280.....	32
4.2.3. Zasilanie	32
4.2.4. Budowa sensora i centralki	33
4.2.5. Montaż elementów systemu pomiarowego	35
5. Opracowanie oprogramowania sterującego i monitorującego.....	37

5.1. Wstęp.....	37
5.2. Sterowniki sensora.....	38
5.3. Sterownik centralki.....	40
5.4. Wyświetlanie i prezentacja pomiarów.....	43
6. Wykonanie badań testowych.....	48
6.1. Zużycie energii sensora	48
6.2. Badania z użyciem danych symulowanych	48
6.3. Akwizycja danych rzeczywistych	51
6.4. Analiza danych	51
7. Podsumowanie	63
Bibliografia.....	65

1. Wstęp

Celem niniejszej pracy była budowa bezprzewodowego rozproszonego systemu pomiaru warunków środowiskowych monitorującego temperaturę, ciśnienie oraz wilgotność. Tego typu systemy mogą być podstawą do budowania domowych stacji pogodowych lub mogą służyć jako elementy wielkoskalowego układu monitorowania parametrów pogodowych, używanych w badaniach meteorologicznych.

W rozdziale „Sensory do pomiaru warunków środowiskowych” przedstawiono podstawy teoretyczne pomiarów wybranych parametrów oraz budowę i działanie czujników używanych do ich mierzenia. Ze względu na konieczność przetworzenia wielkości mierzonych (na przykład temperatury) na wartości elektryczne (napięcie lub natężenie prądu) czujniki te wykorzystują różnorodne procesy fizyczne.

W rozdziale „Sposoby komunikacji w rozproszonych systemach pomiarowych” przedstawiono problematykę łączności pomiędzy urządzeniami pomiarowymi i opisano różne protokoły oraz systemy komunikacji przewodowej i bezprzewodowej. W różnych zastosowaniach ważne są różne aspekty typu komunikacji – czasem istotna jest energooszczędność, podczas gdy innym razem nacisk położony jest na zasięg lub szybkość transmisji.

W następnym rozdziale opisano projekt i sposób wykonania autorskiego urządzenia, schematy (zarówno blokowe, jak i elektryczne), a także przedstawiono wygląd wykonanego urządzenia. Zaprojektowane urządzenie korzysta z dostępnych w sklepach czujników oraz mikrokontrolerów.

W kolejnym rozdziale zawarto kody programów sterujących zarówno stroną nadawczą (sensorem) urządzenia, jak również jego stroną odbiorczą (centrałą). W rozdziale tym zawarto również kod służący do analizy i wizualizacji uzyskanych danych. Kody sterujące napisano z użyciem języka C++ (w wersji używanej w Arduino IDE). Natomiast aplikację analizującą wyniki pomiarów napisano w środowisku MATLAB.

W kolejnym rozdziale przedstawiono proces badań testowych, czyli uzyskania pomiarów oraz ich analizy. Opisano umiejscowienie sensorów, warunki pomiarów a także ogólne działanie zbudowanego urządzenia.

Ostatnią część niniejszej pracy poświęcono podsumowaniu oraz wnioskowi z przeprowadzonych działań.

2. Sensory do pomiaru warunków środowiskowych

2.1. Wstęp

W niniejszym rozdziale przedstawiono podstawowe typy czujników służących do pomiarów warunków środowiskowych. Pokrótce opisano czujniki temperatury, wilgotności oraz ciśnienia. Przy pisaniu niniejszego rozdziału korzystano z następujących źródeł: [1-5].

2.2. Czujniki ciśnienia

Ciśnienie p jest wielkością skalarną równą granicy ilorazu siły F normalnej (prostopadłej) do pola powierzchni S , na które działa ta siła:

$$p = \lim_{\Delta S \rightarrow 0} \frac{\Delta F}{\Delta S} = \frac{dF}{ds}. \quad (2.1)$$

W każdej metodzie pomiaru ciśnienia dokonuje się pomiaru względem ciśnienia odniesienia. Jeżeli ciśnieniem odniesienia jest ciśnienie próżni, wówczas jest to pomiar ciśnienia bezwzględnego. Pomiar ciśnienia atmosferycznego są pomiarami bezwzględnymi. Ciśnieniem odniesienia jest w tym przypadku ciśnienie próżni.

Jednostką ciśnienia w układzie SI jest paskal (Pa).

$$1 \text{ Pa} = \frac{N}{m^2} \quad (2.2)$$

Ponieważ 1Pa jest małą jednostką, w praktyce używa się jej wielokrotności:

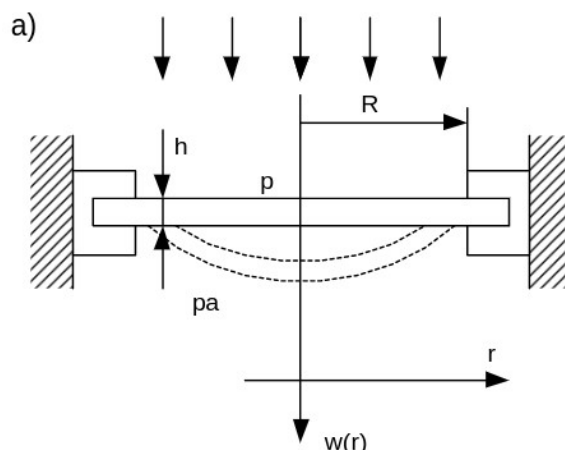
1 hPa = 10^2 Pa oraz 1 MPa = 10^6 Pa.

Używa się również innych jednostek ciśnienia. Ich wzajemne relacje przedstawia poniższa tabela:

Tabela 2.1. Porównanie jednostek ciśnienia

Przeliczenie na:		kPa	bar	mmHg (Tor)	mmH ₂ O
Jednostka:	kPa	1	0,01	7,5006	101,973
	bar	100	1	750,06	10197,3
	mmHg (Tor)	0,13332	$1,3332 \cdot 10^{-3}$	1	13,5951
	mmH ₂ O	0,09806	$98,06 \cdot 10^{-6}$	0,07355	1
	Pounds per square inch PSI	6,8948	$68,948 \cdot 10^{-3}$	51,715	703,09
	In. H ₂ O	0,2491	0,002491	1,8683	25,400
	In. Hg	3,3864	0,033864	25,400	345,32
	atm	101,325	1,01325	760	10332,3
	at	98,0665	0,980665	735,559	10000

Obecnie stosowane czujniki ciśnienia budowane są przy użyciu płaskiej sprężystej membrany, która poddawana jest ciśnieniu gazu z jednej lub z obu stron. Różnica ciśnień powoduje odkształcenie membrany, które można zmierzyć. Zakładając, że sztywno zamocowaną płaską kolistą membranę o promieniu R podda się różnicy ciśnień działających na obie powierzchnie membrany, to ulegnie ona odkształceniu w sposób przedstawiony na rys. 2.1.:



Rys. 2.1. Membranowy czujnik ciśnienia

Można wyprowadzić wzór na odkształcenie membrany u w odległości r od jej środka:

$$u(r) = (p - p_0) \cdot \left(1 - \frac{r^2}{R^2}\right) \cdot F_m \quad (2.3)$$

Gdzie F_m jest stałą zależną od membrany, a R to jej promień.

Rodzaje czujników ciśnienia różnią się materiałem wykonania membrany oraz sposobem pomiaru jej odkształcenia. Jako materiały do wykonania membrany obecnie stosuje się metale, krzem lub ceramikę.

Do pomiaru odkształcenia membrany stosowane są światłowodowe czujniki przesunięcia, indukcyjne czujniki przesunięcia i tym podobne.

Współcześnie produkowane czujniki ciśnienia zwykle wykonywane są w jednym kryształce krzemu, w którym wytrawiono krzemową membranę wraz z mierzącymi odkształcenie przetwornikami tensometrycznymi oraz układem elektronicznym wzmacniającym i wstępnie przetwarzającym sygnał pomiarowy. Materiały takie jak krzem wykazują bardzo dużą odporność na zmęczenie, więc membrana krzemowa może być odkształcana praktycznie nieskończenie długo. Taki czujnik jest bardzo trwały. Na wyjściu czujnika otrzymuje się analogowy lub cyfrowy sygnał, który najczęściej zależy liniowo od zmierzonego ciśnienia.

2.3. Czujniki temperatury

Podstawową jednostką temperatury jest Kelwin [K], zdefiniowany poprzez przyjęcie ustalonej wartości liczbowej stałej Boltzmanna k , wynoszącej $1,380649 \cdot 10^{-23}$, wyrażonej w jednostce J/K , która jest równa $\frac{kg \cdot m^2}{s^2 K}$. Zero K to najniższa temperatura możliwa do osiągnięcia we Wszechświecie. Innymi jednostkami temperatury stosowanymi powszechnie na świecie są:

- $1^\circ C$ – stopnie Celsjusza – $0^\circ C$ to temperatura zamarzania wody w warunkach normalnych (273.15 K). Przyrost temperatury o $1^\circ C$ równa się przyrostowi o jeden K.
- $1^\circ F$ – stopnie Fahrenheita – $0^\circ C$ odpowiada $32^\circ F$, a $100^\circ C$ to $212^\circ F$.

Do pomiarów temperatury najczęściej używa się czujników rezystancyjnych, czujników półprzewodnikowych, termopar, czujników bimetalicznych.

2.3.1. Czujniki rezystancyjne

Termorezystory to elementy elektroniczne, które zmieniają swoją rezystancję wraz ze zmianą temperatury otoczenia. Zmiany te są opisywane przez temperaturowy współczynnik rezystancji oznaczany symbolem α . Znając α oraz rezystancję termorezystora R_P w ustalonej temperaturze T_P , można wyznaczyć jego rezystancję R w dowolnej temperaturze T :

$$R = R_p \cdot (1 + \alpha(T - T_p)) \quad (2.4)$$

Wykorzystywane zjawisko polega na tym, że wraz ze wzrostem temperatury zwiększa się amplituda drgań atomów w sieci krystalicznej rezystora, co zwiększa prawdopodobieństwo zderzeń wolnych elektronów z atomami metalu, w konsekwencji prowadząc do wzrostu rezystancji elementu.

Drugim rodzajem czujników rezystancyjnych są termistory. Wytwarzane są zwykle z tlenków, siarczków i krzemianów metali takich jak nikiel, kobalt, miedź. Dla termistorów zależność rezystancji od temperatury przedstawia się następująco:

$$R = A \cdot \exp\left(\frac{B}{T}\right) \quad (2.5)$$

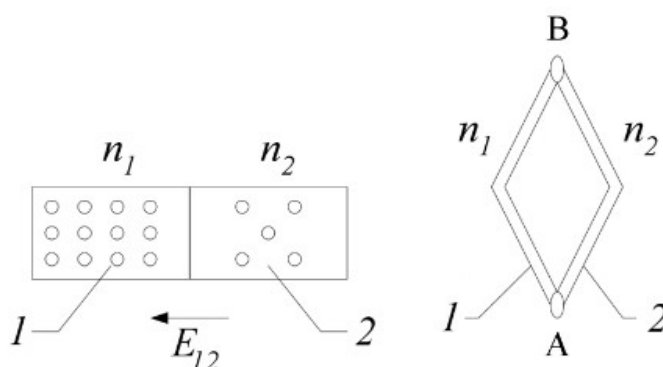
Gdzie B – stała materiałowa, A – rezystancja dla temperatury dążącej do nieskończoności.

Wyróżnia się dwa typy termistorów PTC (rezystancja wzrasta wraz ze wzrostem temperatury) oraz NTC (rezystancja spada wraz ze wzrostem temperatury).

Aby dokonać odczytu z rezystancyjnego czujnika temperatury, należy zadać prąd pomiarowy i odczytać napięcie na elemencie. Używa się prądu o takim natężeniu, by nie nagrzewać elementu, a jednocześnie takiego, który wywoła mierzalny spadek napięcia.

2.3.2. Termopara

Działanie termopary opiera się na zjawisku Seebecka, polegającym na powstawaniu siły elektromotorycznej (a więc i przepływu prądu) w obwodzie zamkniętym, w miejscu zetknięcia dwóch metali o różnych temperaturach. Na styku tych dwóch metali pojawia się różnica potencjałów (rys. 2.2.).



Rys. 2.2. Istota działania termopary

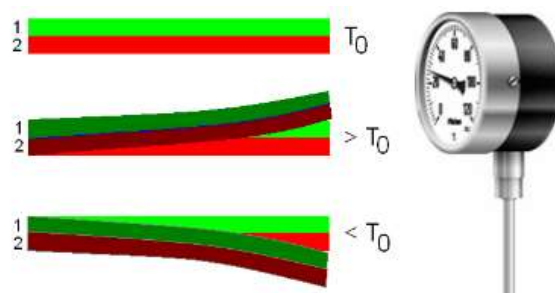
Powstająca siła termoelektryczna w obwodzie zamkniętym przedstawionym na rysunku 2.2. wynosi

$$E = \frac{k_B}{e} \cdot \ln \frac{n_1}{n_2} \cdot (T_A - T_B) = C \cdot (T_A - T_B) \quad (2.6)$$

gdzie e to ładunek elektronu, n_1 i n_2 to koncentracja swobodnych elektronów w metalach, T_A i T_B to temperatury bezwzględne styków, C to stała zależna od wybranych metali, a k_B to stała Boltzmanna. T_B jest temperaturą złącza odniesienia, które jest niezbędne przy użyciu termopary. Istnieją elektryczne odpowiedniki złącza odniesienia.

2.3.3. Czujniki bimetaliczne

Bimetal to dwa połączone trwale na całej swojej powierzchni metale. Jeżeli użyte metale będą miały różne współczynniki rozszerzalności cieplnej, to pod wpływem zmiany temperatury składniki bimetalu zmieniają rozmiar o różne wartości, a cały bimetal zacznie się wyginać (rys. 2.3.).



Rys. 2.3. Zasada działania termometru bimetalicznego

Oznaczając T_0 temperaturę równowagi bimetalu – czyli temperaturę, w której oba składniki mają równą długość, a T to temperatura otoczenia, można zapisać, że:

- Jeżeli $T > T_0$ – materiał 2 ma większy współczynnik rozszerzalności cieplnej i zwiększy swoją długość bardziej niż metal 1.
- Jeżeli $T < T_0$ – materiał 2 ma większy współczynnik rozszerzalności cieplnej i zmniejszy swoją długość bardziej niż metal 1.

Taki bimetal można zwinąć w kształt spirali i umocować na jednym z jego końców wskazówkę – powstanie wtedy prosty termoskop. Czujniki bimetaliczne znajdują zastosowanie w klimatyzacji, przemyśle spożywczym, grzejnictwie i wentylacji.

2.4. Czujniki wilgotności

Jeżeli rozważamy mieszaninę gazów, to można założyć, że każdy ze składników mieszaniny zachowuje się niezależnie od pozostałych. W związku z tym powietrze można traktować jako mieszaninę pary wodnej oraz innych suchych składników. Zawartość pary wodnej w powietrzu nie może wzrastać nieograniczenie – istnieje granica nazywana stanem nasycenia. Jeżeli zawartość pary wodnej dalej zwiększałaby się, to spowoduje to skraplanie się wody do otoczenia. Temperatura, przy której to następuje, nazywa się temperaturą punktu rosy.

Skoro składniki powietrza traktowane są oddzielnie, to ciśnienie atmosferyczne można potraktować jako sumę ciśnień pary wodnej i składników suchych:

$$p = p_{po} + p_w \quad (2.7)$$

Gdzie p to ciśnienie powietrza, p_{po} – ciśnienie składników suchych, p_w – ciśnienie pary wodnej. Ciśnienie pary jest związane z panującą temperaturą:

$$p_w = \begin{cases} 610.5 \cdot \exp\left(\frac{17.269 \cdot T}{237.3 + T}\right), & \text{dla } T > 0^\circ\text{C} \\ 610.5 \cdot \exp\left(\frac{21875 \cdot T}{265.5 + T}\right), & \text{dla } T \leq 0^\circ\text{C} \end{cases} \quad (2.8)$$

Teraz można zdefiniować pojęcie wilgotności. Wilgotność względna (RH , ang. Relative Humidity), to iloraz ciśnienia pary wodnej w powietrzu oraz ciśnienia pary nasyconej p_n w danej temperaturze powietrza:

$$RH = \varphi = \frac{p_w}{p_n} \cdot 100\% \quad (2.9)$$

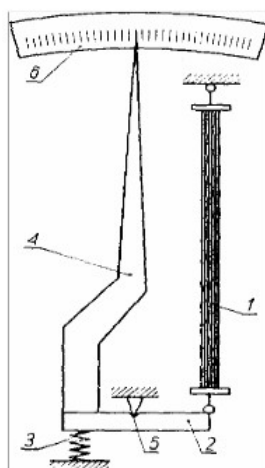
Natomiast wilgotność bezwzględna to ilość wody m_w w jednostce objętości V powietrza:

$$H = \frac{m_w}{V} \left[\frac{g}{m^3} \right] \quad (2.10)$$

Przyrządy do badania wilgotności nazywane są higrometrami. Pokróćce zostaną omówione higrometry włosowe, oparte na zasadzie przewodnictwa cieplnego oraz pojemnościowe.

2.4.1. Higrometry włosowe

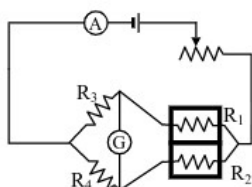
Ich działanie wykorzystuje właściwość zmian długości włosów (ludzkich lub zwierzęcych) lub włókien syntetycznych pod wpływem zmian wilgotności powietrza. Są powszechnie stosowane ze względu na prostotę konstrukcji. Potrafią mierzyć wilgotność w zakresie 30% – 100% RH. Ich dokładność sięga $\pm 3\%$, jednakże mają też wady. Wymagają okresowej regeneracji włosów a także dokładnego ich odłuszczenia. Stosować je można tylko do temperatury 50 °C. Wymagają okresowej regulacji i konserwacji. Budowę higrometru włosowego przedstawiono na rys. 2.4.



Rys. 2.4. Higrometr włosowy: 1 – wiązka włosów odłuszczonych, 2 – dźwignia, 3 – sprężyna, 4 – wskazówka, 5 – oś dźwigni, 6 – skala

2.4.2. Higrometry oparte na zasadzie przewodnictwa cieplnego

Ten typ higrometru zbudowany jest na podstawie elektrycznego mostka Wheatstone'a. W jego skład wchodzi dwie pary rezystorów, takie, że $R_1=R_2$ i $R_3=R_4$ (rys. 2.5.). Jeżeli rezystory 1 i 2 znajdują się w takich samych warunkach, to mostek znajduje się w stanie równowagi i galwanometr wskaże zero. Jeżeli wilgotność wokół jednego z tych rezystorów się zmieni, to zmieni się również jego rezystancja – co prowadzi do nierównowagi mostka.



Rys. 2.5. Schemat higrometru opartego na mostku Wheatstone'a

2.4.3. Higrometry pojemnościowe

Ich działanie opiera się na pomiarach pojemności elektrycznej kondensatora zbudowanego z porowatej warstwy higroskopijnej. Są niewrażliwe na skoki temperatury, a także mają małą bezwładność wskazań.

3. Sposoby komunikacji w rozproszonych systemach pomiarowych

3.1. Wstęp

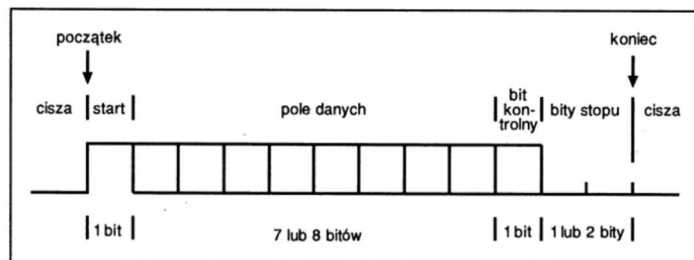
W niniejszym rozdziale przedstawiono różne sposoby komunikacji w systemach pomiarowych – od interfejsów przewodowych, przez kilka różnych standardów łącz bezprzewodowych, aż po wykorzystany w niniejszej pracy interfejs Bluetooth Low Energy. Przy pisaniu niniejszego rozdziału korzystano z następujących źródeł: [6-16].

3.2. Interfejsy przewodowe

3.2.1. Interfejs RS-232

Nazwa tego interfejsu pochodzi od słów Recommended Standard (rekomendowany standard). Wersja RS-232 została wprowadzona przez Electronic Industries Association (EIA) w 1962 roku w celu normalizacji komunikacji pomiędzy urządzeniami DTE (ang. Data Terminal Equipment), czyli urządzeń końcowych danych (np. komputer) oraz urządzeń DCE (ang. Data Communication Equipment), czyli urządzeń komunikacji danych (np. modem). Zrewidowana wersja RS-232C wprowadzona w 1969 roku pozwalała na szeregową transmisję bit po bicie na niewielkie odległości z niedużą prędkością (do 20 kbit/s na odległości do około 15 m).

Transmisja danych może odbywać się w trybie asynchronicznym, w którym przesyłane są pojedyncze znaki o ściśle określonym formacie: bit startu, pole danych, bit kontrolny i na koniec jeden lub dwa bity stopu. Poniższy rysunek (rys. 3.1.) ilustruje budowę ramki interfejsu RS-232:

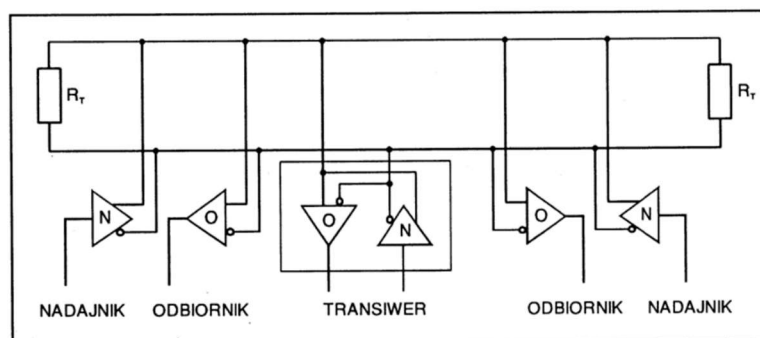


Rys. 3.1. Budowa ramki interfejsu RS-232

Prawidłowość przesyłanych danych określa się na podstawie bitu kontrolnego, który najczęściej jest bitem parzystości. Jego wartość jest ustawiana na podstawie zliczenia „jedynek”.

W trybie synchronicznym przesyłane są duże bloki danych w postaci ramek, których kolejne bity nadawane są zgodnie z taktowaniem nadawania. Na początku i końcu każdego bloku występują specjalne znaki synchronizacji, natomiast znaki wewnątrz bloku następują bezpośrednio jeden po drugim. Transmisja synchroniczna jest szybsza niż asynchroniczna, jest jednak trudniejsza w realizacji.

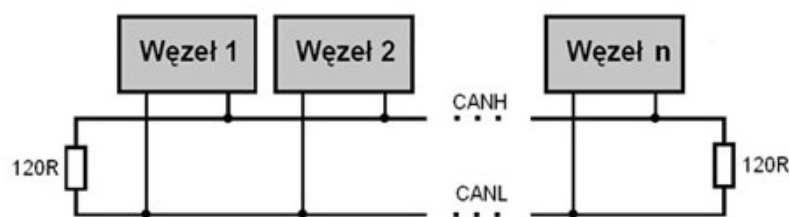
Rozwinięciem standardu RS-232C jest standard RS-485, który określa symetryczny, zrównoważony system transmisji danych złożony z różnicowego nadajnika, dwuprzewodowego zrównoważonego toru przesyłowego oraz odbiornika. Standard ten dopuszcza istnienie zarówno wielu odbiorników jak i nadajników na jednej linii. Model takiej linii przedstawiono na rysunku 3.2.



Rys. 3.2. Model interfejsu RS-485

3.2.2. Interfejs CAN

CAN, czyli Controller Area Network, to standard przemysłowej sieci transmisyjnej, stworzonej w odpowiedzi na potrzeby przemysłu motoryzacyjnego. Został zaprojektowany i wdrożony przez firmę Bosch na początku lat osiemdziesiątych XX wieku. Jest magistralą typu rozgłoszeniowego, to znaczy, że każdy węzeł „słyszy” całą krążącą po sieci transmisję. Transmisja odbywa się za pomocą dwóch przewodów, na których nadawane są przeciwstawne napięcia – jest to tak zwana metoda różnicowa. Strukturę magistrali CAN przedstawiono na rysunku 3.3.

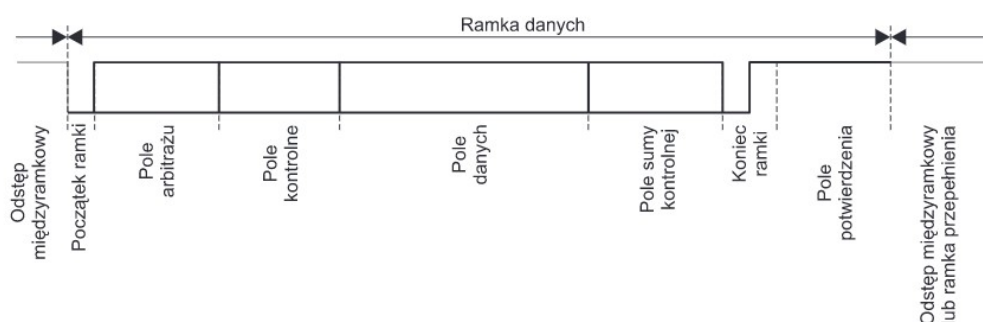


Rys. 3.3. Struktura magistrali CAN

Standard ten charakteryzuje się:

- wysokim bezpieczeństwem i odpornością na błędy transmisji,
- prędkością transmisji do 1 Mbps (na odległościach do kilkunastu metrów),
- architekturą multi-master (każdy z węzłów sieci ma identyczne prawa rozpoczynania transmisji),
- transmisją rozgłoszeniową.

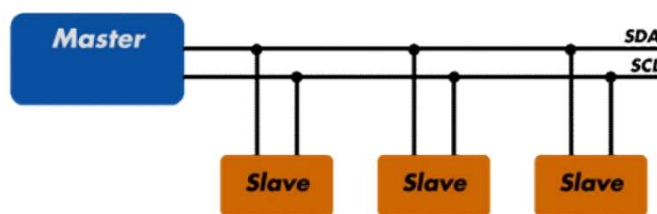
Ogólny format ramki danych standardu CAN przedstawiono na rysunku 3.4.



Rys. 3.4. Format ramki danych interfejsu CAN

3.2.3. Interfejs I²C

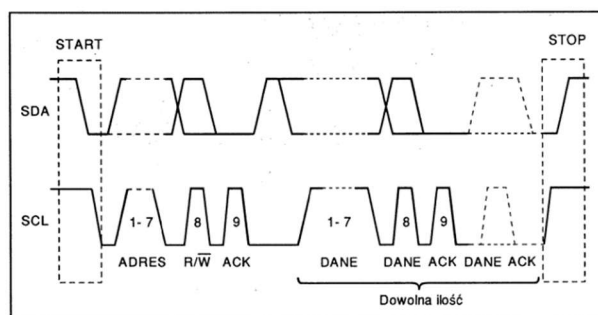
Standard ten został zaproponowany przez firmę Philips na początku lat osiemdziesiątych XX wieku. Jest to standard połączenia szeregowego, synchronicznego, dwu-kierunkowego o prędkości transmisji od 400 kbps do 3,4 Mbps. Korzysta z dwóch równoległych linii: SDA (Serial Data) oraz SCL (Serial Clock) (rys. 3.5.).



Rys. 3.5. Interfejs I²C

Magistrala ta jest kontrolowana przez urządzenie nadrzędne (tak zwany master), które zapewnia zegar (na linii SCL) synchronizujący przesyłanie danych. Każde urządzenie obecne na magistrali identyfikowane jest przez indywidualny adres. W warstwie fizycznej obie linie podciągnięte są do napięcia zasilania za pomocą rezystorów (zwyczajowo od 2 k Ω do 10 k Ω).

Rozpoczęcie transmisji oznaczone jest znakiem startu (w czasie wysokiego poziomu zegara następuje zmiana linii sygnałowej z HIGH na LOW), a jej koniec znakiem końca (w czasie wysokiego poziomu zegara następuje zmiana linii sygnałowej SDA z LOW na HIGH). Dane przesyłane magistralą grupowane są w bajty po 8 bitów. Natomiast liczba bajtów przesyłanych w trakcie jednej transmisji jest nieograniczona. Na rysunku 3.6. przedstawiono pełny transfer danych na magistrali I²C:



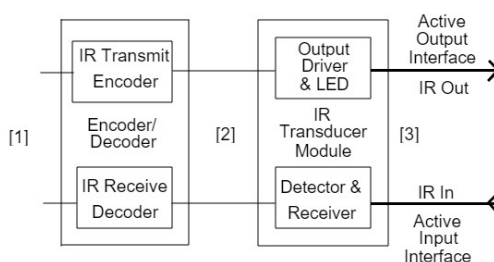
Rys. 3.6. Pełny transfer danych na magistrali I²C

3.3. Interfejsy bezprzewodowe

3.3.1. Interfejs IrDA

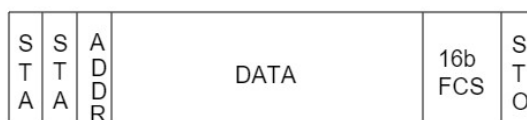
Standard IrDA przeznaczony jest przede wszystkim do tworzenia sieci tymczasowych, w których komunikują się urządzenia przenośne takie jak telefony komórkowe. Do przesyłania danych wykorzystywane jest światło podczerwone (to znaczy o długości fali powyżej 700 nm). Urządzenia łączą się połączeniami bezpośrednimi typu punkt-punkt, a sam standard charakteryzuje się prostą i taną implementacją oraz małym poborem mocy. Ceną za te zalety jest konieczność zachowania linii bezpośredniej widoczności między urządzeniami, a także wolny

i podatny na przerwanie transfer danych. Na rysunku 3.7. przedstawiono schemat blokowy systemu IrDA:



Rys. 3.7. Schemat blokowy standardu IrDA

Jak widać są tu dwie drogi sygnału: wejściowa i wyjściowa, które fizycznie realizowane są poprzez dwie diody podczerwone, odbiornik i nadajnik. Dalej jest układ przetwarzający sygnał świetlny na elektryczny oraz dekodery i enkodery sygnału. Typowy format ramki standardu przedstawiono na rysunku 3.8.



Rys. 3.8. Ramka interfejsu IrDA

STA: flaga startu (01111110)

ADDR: 8 bitów adresu

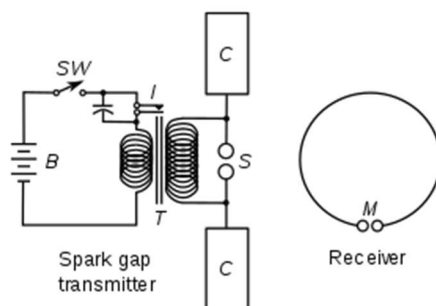
DATA: pole danych

FCS: 16 bitowa suma kontrolna

STO: flaga końca, identyczna z flagą startu

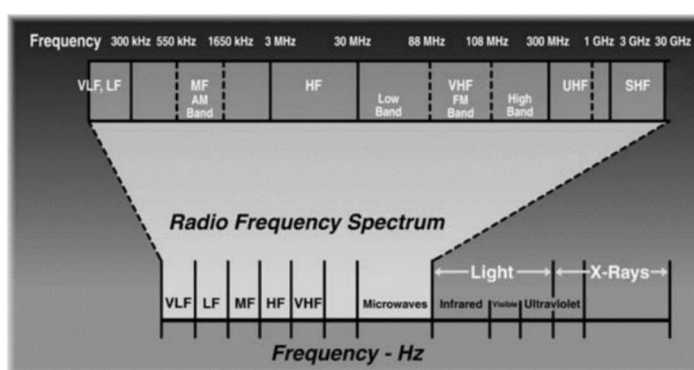
3.3.2. Radio

Pierwszy raz w historii fale radiowe zostały odebrane w 1888 roku przez Heinricha Hertza, który za pomocą zbudowanego przez siebie generatora i odbiornika (rys. 3.9.) wykazał istnienie fal radiowych postulowanych przez Maxwella. Fale odebrane jego aparaturą wykazywały właściwości i zachowanie zgodne z teorią Maxwella.



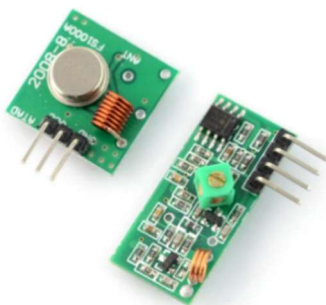
Rys. 3.9. Aparatura Hertza

Dalsze badania włoskiego badacza Marconiego pozwoliły po raz pierwszy nawiązać łączność z użyciem fal radiowych. Szybko fale radiowe stały się uniwersalnym medium komunikacji na odległość, zarówno na odległości tysięcy kilometrów (choćby używając fal radiowych odbitych od jonosfery), jak i na odległości kilku do kilkudziesięciu metrów (na przykład samochody zdalnie sterowane). Fale radiowe obejmują spektrum od około 30 kHz do 30 GHz (rys. 3.10.)



Rys. 3.10. Zakres fal radiowych

Do celów komunikacji krótkiego zasięgu zwykle używana jest częstotliwość 433 MHz znajdująca się w nielicencjonowanym paśmie. Typowo dostępne moduły łączności radiowej, jak na przykład FS100A z odbiornikiem (rys. 3.11.), mogą zapewnić zasięg łączności do około 100 metrów.



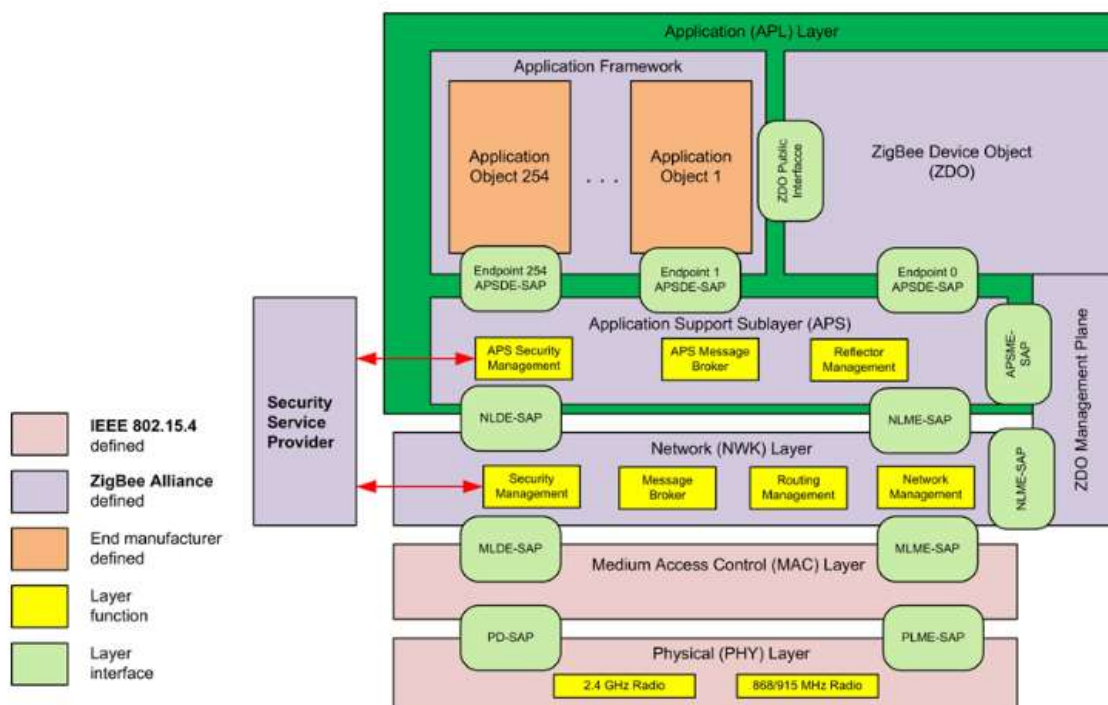
Rys. 3.11. Przykład modułu łączności radiowej

3.3.3. Interfejs ZigBee

ZigBee to protokół transmisji bazujący na samoorganizacji i topologii kraty (ang. mesh). Sieci oparte na tym standardzie charakteryzują się niewielkim poborem energii, małymi prędkościami przesyłu danych (do około 250 kbps) oraz zasięgiem między węzłami rzędu 100 m. Architekturę standardu przedstawiono na rysunku 3.12. Parametry używanego w standardzie toru radiowego opisuje standard IEEE802.15.4 (komunikacja odbywa się najczęściej w paśmie 2,4 GHz). Urządzenia ZigBee można podzielić na 3 typy:

- Koordynator – jest węzłem początkowym sieci, do którego mogą się podłączać pozostałe urządzenia. Zazwyczaj pełni rolę urządzenia zbierającego dane. W danej sieci może istnieć tylko jeden koordynator.
- Router – przekazuje dane dalej do koordynatora lub innych routerów.
- Urządzenie końcowe – źródło danych, które przesyła je do routera.

W sieci ZigBee liczba urządzeń końcowych to teoretycznie 65536 (16-bitowe adresowanie), jednak zazwyczaj liczba urządzeń końcowych nie przekracza 256. Technologia ZigBee jest rozwijana przez The ZigBee Alliance, organizację założoną w 2002 roku. Zrzesza ponad 150 firm z całego świata w celu wspólnego rozwoju specyfikacji ZigBee.



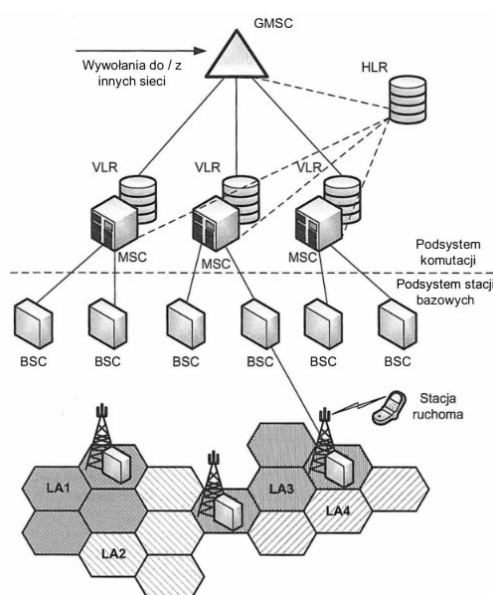
Rys. 3.12. Architektura interfejsu ZigBee

3.3.4. Sieć komórkowa GSM

Sieć GSM można podzielić na trzy zasadnicze części (rys. 3.13.):

- stacja ruchoma (MS, ang. Mobile Station),
- podsystem stacji bazowych (BSS, ang. Base Station Subsystem),
- podsystem komutacyjny (SSS, ang. Switching Subsystem).

Stacja ruchoma (czyli telefon GSM lub telefon komórkowy) używana jest przez abonentów do komunikowania się z innymi użytkownikami sieci telekomunikacyjnych. W skład podsystemu stacji bazowych wchodzi stacja bazowa BTS (ang. Base Transceiver Station) oznaczana też w skrócie BS oraz sterownik stacji bazowych BSC (ang. Base Station Controller).



Rys. 3.13. Architektura sieci GSM

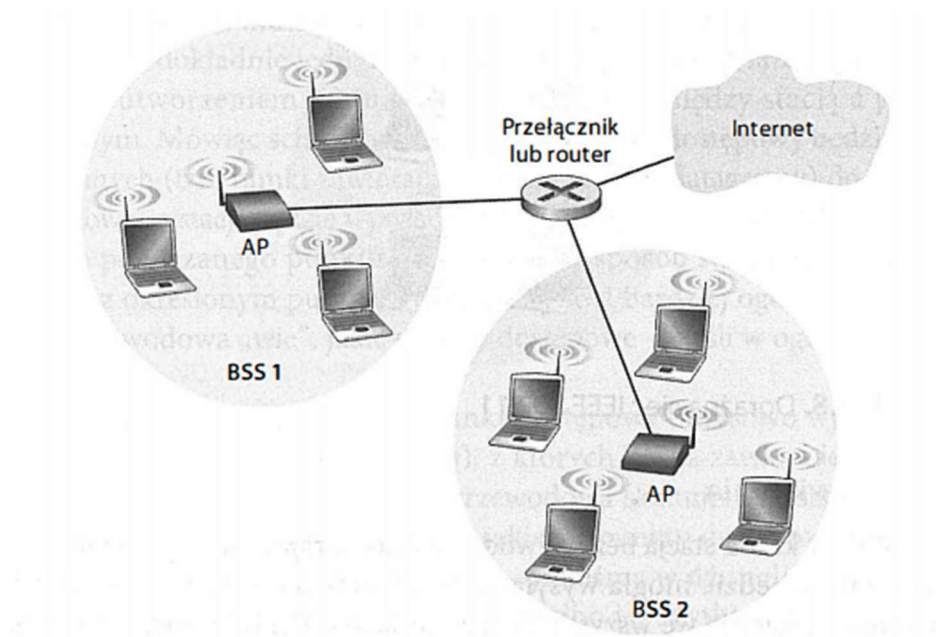
BSS zapewnia funkcje transmisyjne i zarządzające systemem transceiverów BTS, niezbędne do poprawnej pracy części systemu telekomunikacyjnego operującego w interfejsie radiowym na obsługiwanym przez dany BSS obszarze. Podsystem komutacji SSS odpowiada za funkcje komutacyjne przy zestawianiu połączeń oraz administruje zasobami sieci.

Stacje bazowe zapewniają zasięg radiowy na obszarze jednej komórki sieci GSM. Do przesyłania danych używa się dwóch różnych częstotliwości. Jedna z nich służy do przesyłania danych w kierunku od stacji ruchomej do sieci (ang. uplink), a druga – w stronę przeciwną (ang. downlink). Przy przemieszczaniu abonenta pomiędzy komórkami niezbędne jest użycie mechanizmów przełączania do innej stacji (ang. handover).

Po wyposażeniu czujników w odpowiedni moduł komunikacyjny wraz z kartą SIM, sieć GSM może zapewnić dobre medium komunikacyjne dla rozproszonego systemu pomiarowego. Wadą takiego rozwiązania są koszty dostępu, zaletą natomiast – ogólnosiwiatowy zasięg.

3.3.5. Interfejs WiFi

Na rysunku 3.14. przedstawiono model sieci lokalnej zbudowanej przy użyciu standardu WiFi, znanego również jako bezprzewodowa sieć lokalna IEEE 802.11.



Rys. 3.14. Sieć lokalna IEEE 802.11

Do działania sieci 802.11 niezbędny jest co najmniej jeden punkt dostępowy, do którego może podłączyć się stacja bezprzewodowa, by wysyłać lub odbierać dane. Obszar działania sieci WiFi zależy od charakterystyki lokalizacji, w której działa sieć i wynosi do kilkunastu metrów (w zamkniętych pomieszczeniach) do kilkuset metrów na terenie otwartym.

Sieć WiFi działa w paśmie ISM, w zakresie częstotliwości od 2,4 do 2,485 GHz oraz 5 GHz. W tym paśmie (o szerokości 85 MHz) zdefiniowano 11 częściowo nakładających się kanałów. Każdy punkt dostępowy standardu okresowo wysyła ramki identyfikacyjne (ang. beacon frames), zawierające jego identyfikator SSID (ang. Service Set Identifier) i adres MAC.

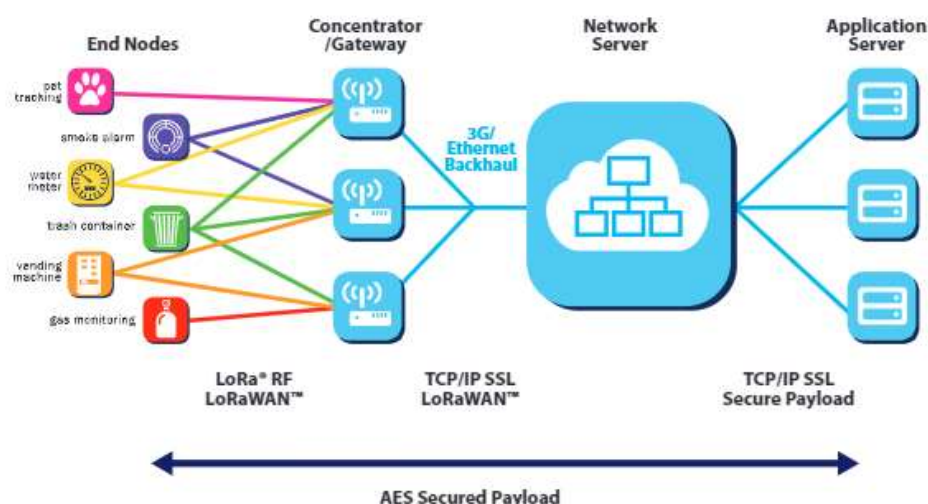
WiFi wykorzystuje protokół CSMA z unikaniem kolizji (ang. CSMA with collision avoidance – CSMA/CA). Każda stacja w sieci przed transmisją bada kanał i wstrzymuje transmisję, jeśli kanał jest zajęty.

WiFi doskonale nadaje się do budowy sieci lokalnych w domach lub biurach. Jedną z wad tego standardu jest wykorzystanie pasma ISM, w którym pracują również między innymi urządzenia Bluetooth, a także duże zużycie energii.

3.3.6. Interfejs LoRa

LoRa korzysta z technologii CSS – Chirp Spread Spectrum. Jest to sposób modulacji fal elektromagnetycznych używany w aplikacjach wojskowych i astronautyce. Jej kluczowa zaleta to odporność na interferencje oraz możliwość uzyskania dużego zasięgu transmisji – maksymalne odległości między urządzeniami korzystającymi ze standardu a stacjami bazowymi wynoszą 10 do 15 km. System LoRa używa nielicencjonowanych pasm częstotliwości (433 MHz, 868 MHz, a także 915 MHz). Urządzenia korzystające z tego standardu łączą się używając topologii gwiazdy. Architektura LoRa (rys 3.15.) składa się z 4 głównych komponentów:

- urządzeń końcowych (ang. end nodes),
- bramek (ang. gateways),
- serwera sieciowego (ang. network server),
- serwera aplikacyjnego (ang. application server).



Rys. 3.15. Architektura sieci LoRa

Węzły w sieci LoRa pracują asynchronicznie. Komunikują się z bramkami wtedy, kiedy mają gotowe do wysłania dane. Dzięki temu oszczędzana jest energia, którą inne sieci (jak na przykład GSM) zużywać muszą na synchronizację urządzeń w sieci.

Dzięki używanej modulacji transmisji, sygnały dochodzące do bramki są właściwie ortogonalne (jeżeli mają różne parametry rozproszenia). Przy zmianie rozproszenia zmienia się również prędkość przesyłu danych – dzięki temu bramka może na tym samym kanale odbierać kilka różnych transmisji o różnej prędkości. Kolejną korzyścią jest to, że jeżeli węzeł znajduje się blisko i ma dobre połączenie z bramką, może przestawić się na wyższą prędkość transmisji tak, aby zajmować łącze tylko na ile to jest rzeczywiście konieczne. System może też dostosowywać moc nadajnika i szybkość transmisji do aktualnie panujących warunków propagacyjnych.

Na rysunku 3.16. przedstawiono przykład modułu komunikacyjnego wykorzystującego standard LoRa. Moduł ten pracuje na częstotliwości 868 MHz i według danych producenta, może osiągnąć zasięg komunikacji do 2 km.



Rys. 3.16. Transmitter radiowy Feather 32u4 RFM95 LoRa

3.3.7. Interfejs Bluetooth

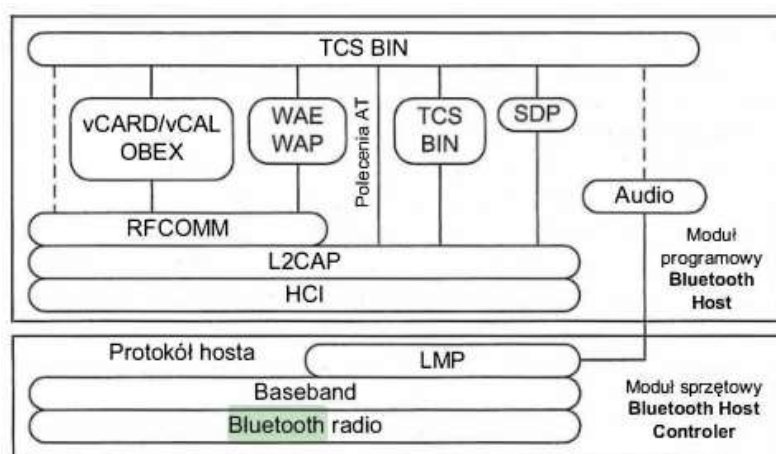
Standard IrDA wymagał, aby pomiędzy dwoma komunikującymi się urządzeniami była bezpośrednia widoczność. Była to znacząca wada tego standardu. Dlatego w 1998 roku przedstawiono pomysł na nowy bezprzewodowy standard komunikacji urządzeń elektronicznych – Bluetooth¹, wykorzystujący fale radiowe.

Systemy Bluetooth pracują w paśmie częstotliwości ISM 2,4 GHz. W większości krajów jest to zakres od 2400 do 2483,5 MHz. Całe dostępne pasmo podzielone jest na 79 kanałów o szerokości 1 MHz. Urządzenia w celu uniknięcia zakłóceń okresowo zmieniają wykorzystywany kanał. Jest to tak zwana metoda rozpraszania widma FHSS (ang. Frequency-Hopping Spread Spectrum) zmiany kanału następują 1600 razy na sekundę, zgodnie z pięcioma różnymi sekwencjami zamiany

¹ Nazwa standardu „bluetooth” wzięła się od przydomka duńskiego króla Haralda Sinozębnego, który w 970 roku podporządkował sobie Norwegię, przyczyniając się do zjednoczenia wrogich plemion z Danii i Norwegii.

kanałów. Metoda ta sprawia, że pasmo może być współdzielone przez wiele sieci jednocześnie, jednak ceną za to jest niska wydajność, długotrwałe nawiązywanie połączenia oraz generowanie silnych zakłóceń.

Standard Bluetooth definiuje także protokoły komunikacyjne, których architekturę przedstawiono na rys. 3.17.



Rys. 3.17. Stos protokołu Bluetooth

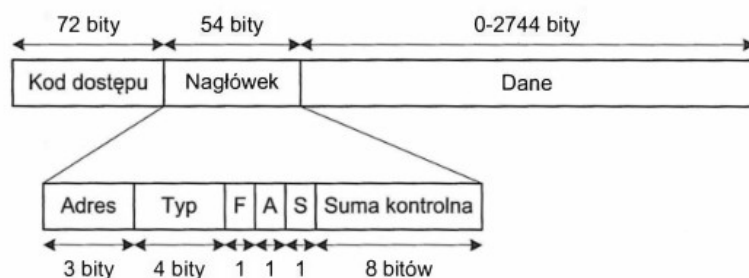
W modelu wyróżniamy moduł sprzętowy oraz programowy. Poniżej opisano warstwy modułu sprzętowego.

Najniższa warstwa – radiowa (odpowiednik warstwy fizycznej modelu OSI) – definiuje fizyczne parametry nadawania, takie jak używane częstotliwości, mechanizmy rozpraszania widma, typ modulacji (GFSK), a także moc nadawczą (decydującą o zasięgu komunikacji).

Następna warstwa – baseband (odpowiednik warstwy fizycznej oraz podwarstwy sterowania dostępu do medium) – realizuje zestawianie połączeń wewnątrz pikosieci (w standardzie Bluetooth oznacza to połączenie do ośmiu urządzeń), adresowanie urządzeń, kontroluje format wysyłanych pakietów oraz steruje poziomem mocy urządzeń.

Budowę pakietu baseband przedstawiono na rys. 3.18. Ramka rozpoczyna się polem kodu dostępu (ang. Access Code) określającym pikosieć, w której jest fizyczny kanał transmitowanego pakietu. Kod ten zmienia się z kolejnymi połączeniami i tylko właściwy odbiornik ma jego aktualną wartość. Pole to służy również procesowi synchronizacji. Następnym polem jest nagłówek. Składa się on z 18 bitów sterujących, które zakodowane są na 54 bitach z użyciem korekcji błędów FEC 1/3. Rozpoczyna się trzema bitami adresu wskazującego urządzenie podrzędne biorące udział w komunikacji. Pole „Typ” określa rodzaj pakietu. Bit sterowania przepływem

F (ang. Flow) steruje przepływem pakietów ACL. Bit A służy do określenia czy pakiet ACL został odebrany poprawnie.



Rys. 3.18. Pakiet Bluetooth

Bit S (SEQN) służy numeracji pakietów w sekwencji – jego wartość jest negowana dla każdego kolejnego pakietu. Ostatnim polem nagłówka jest pole kontroli błędów HEC. W polu danych przenoszone są wiadomości protokołów LMP i L2CAP.

Warstwa LMP (ang. Link Manager Protocol) odpowiada za konfigurację połączenia, realizację ustawień sprzętowych oraz procedurę weryfikacji i szyfrowania.

Poniżej opisano warstwy modułu programowego.

Najniższą warstwą jest interfejs sterujący hosta HCI (ang. Host Control Interface) zarządzający fizycznymi parametrami połączenia. Izoluje niższe warstwy modelu od portów komunikacyjnych takich jak USB, RS-232 lub UART. Dzięki temu wyższe warstwy nie muszą znać szczegółów sprzętowej implementacji standardu, a i tak uzyskują dostęp do żądanych urządzeń.

Warstwa L2CAP (ang. Logical Link Control and Adaptation Layer Protocol – warstwa łączy logicznych i protokołów adaptacyjnych) przystosowuje dane przesyłane z wyższych warstw do możliwości transmisyjnych urządzeń Bluetooth.

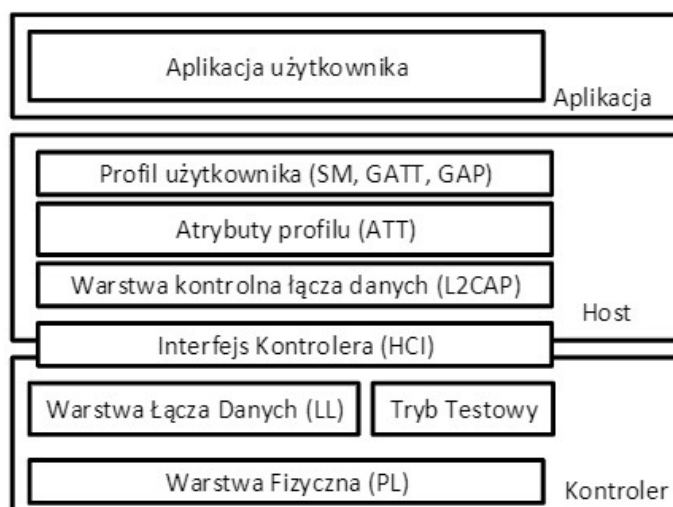
RFCOMM emuluje porty szeregowo RS-232 w połączeniu Bluetooth. Z jej usług korzystają moduły OBEX (ang. Object Exchange Protocol) oraz WAE/WAP (ang. Wireless Application Environment/Wireless Application Protocol). Pierwszy z nich (pierwotnie zdefiniowany dla technologii IrDA) implementuje protokół warstwy sesji służący do wymiany danych binarnych. Drugi określa zasady obsługi WML i HTML przez łącza TCP/IP, czyli umożliwia korzystania z bezprzewodowego dostępu do Internetu.

Zarządzanie połączeniami telefonicznymi (odbieranie, zawieszanie, wznawianie, rozłączanie) umożliwia moduł TCS BIN (ang. Telephony Control Protocol Specification – Binary).

Do wykrywania usług realizowanych przez urządzenia w obszarze pikosieci służy protokół SDP (ang. Service Discovery Protocol).

W 2010 roku opracowano standard Bluetooth Low Energy, który pozwolił znacząco zmniejszyć pobór mocy dodatkowo zwiększając znacznie zasięg. Mimo wielu podobieństw oba standardy: Bluetooth Classic oraz Low Energy nie są ze sobą kompatybilne.

Na rysunku 3.19. przedstawiono stos protokołów standardu BLE – obejmujący warstwę fizyczną (ang. Physical Layer), warstwę łącza danych (ang. Link Layer) oraz HCI (ang. Host Controller Interface).



Rys. 3.19. Stos protokołów interfejsu BLE

W warstwach wyższych zdefiniowano podwarstwy:

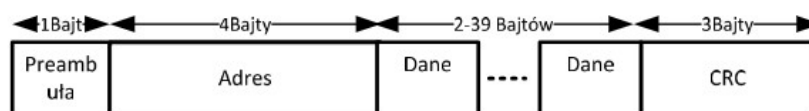
- L2CAP – interfejs pomiędzy warstwami wyższymi i warstwami transmisyjnymi,
- ATT (ang. Attribute Protocol) – warstwa realizująca proces komunikacji pomiędzy węzłami pełniącymi funkcję klienta i serwera za pośrednictwem danego kanału L2CAP,
- SM (ang. Security Manager) – warstwa odpowiada za generację, zarządzanie oraz przechowywanie kluczy szyfrujących,
- GATT (ang. Generic Attribute Profile) – opisuje procedury korzystania z ATT oraz profile Bluetooth,
- GAP (ang. Generic Access Profile) – interfejs między aplikacją, a profilami BLE, odpowiada również za wyszukiwanie urządzeń i usług, nawiązywanie i konfigurowanie połączeń oraz bezpieczeństwo transmisji.

Główne różnice pomiędzy opisywanymi wersjami protokołu dotyczą warstwy fizycznej. Bluetooth Low Energy transmituje dane w paśmie ISM 2,4-2,4835 GHz, używając modulacji GFSK (ang. Gaussian Frequency Shift Keying) oraz mechanizmu sekwencyjnej zmiany częstotliwości (ang. Frequency Hopping – FH). Pasma podzielone zostało na 40 kanałów o szerokości 2 MHz każdy. Przepustowość transmisji to około 300 kbps w warstwie aplikacji. Niezawodność transmisji wzrosła, a jej zasięg zwiększono do ponad 100 m w otwartej przestrzeni dzięki zwiększeniu współczynnika modulacji. Porównanie parametrów transmisyjnych obu standardów przedstawiono w tabeli 3.1:

Tabela 3.1. Porównanie protokołów

	Bluetooth	Bluetooth Low Energy
Technika Modulacji	FH	FH
Modulacja	GFSK	GFSK
Współczynnik modulacji	0.35	0.5
Szerokość kanału	1MHz	2MHz
Nominalna prędkość transmisji	1-3Mbps	1Mbps
Przepustowość	0.7-2.1Mbps	<0.3Mbps
Liczba węzłów	7	nielimitowana
Zabezpieczenie	56-128 bitowe	AES 128-bitowy

W celu ograniczenia zużycia energii zmieniono między innymi procedurę parowania urządzeń. Urządzenie, które chce nadawać dane, wysyła pakiet inicjujący w kanale zgłoszeniowym (ang. Advertising Channel). Jeśli pakiet zostanie odebrany przez urządzenie docelowe, to wtedy wysyła ono pakiet żądania przyłączenia. Pakiet ten zawiera parametry, z jakimi zrealizowana ma zostać transmisja. Urządzenie to staje się *Masterem* nowopowstałej sieci. Urządzenie zgłaszające dane wysyła potwierdzenia odebrania żądania i automatycznie uzyskuje status *Slave* w tej sieci. O tego momentu urządzenia mogą wymieniać ze sobą dane. Dane są transmitowane w postaci ramek o polu danych zmiennej długości. Na rysunku 3.20. przedstawiono format ramki protokołu BLE.



Rys. 3.20. Format ramki protokołu BLE

Głównym celem protokołu są cykliczne transmisje niewielkich pakietów danych (z minimalnym odstępem między ramkami $\geq 150 \mu s$). Natomiast pomiędzy transmisjami urządzenia zostają uśpione, redukując w ten sposób pobór prądu do zaledwie kilku μA .

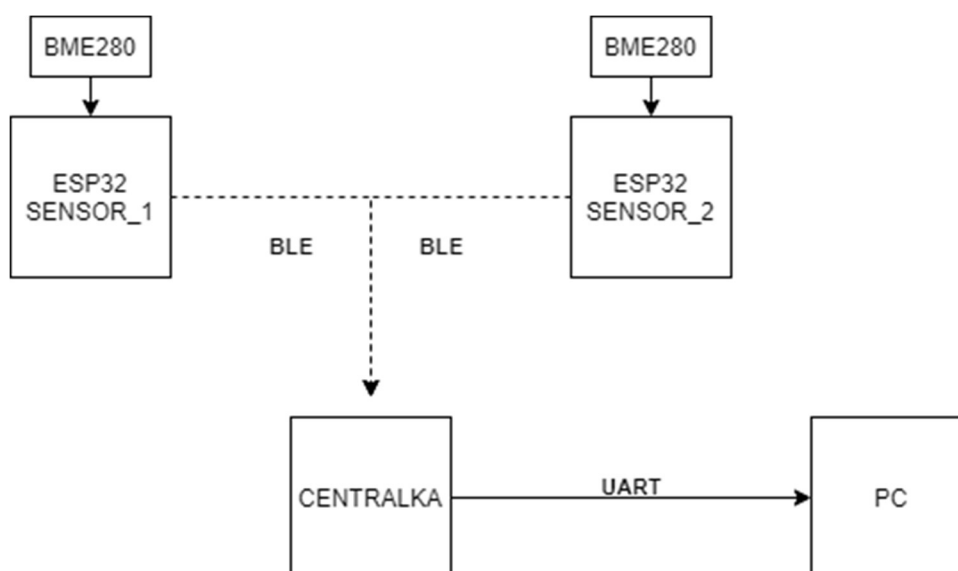
4. Projekt i wykonanie modelu systemu pomiarowego

4.1. Wstęp

Celem niniejszej pracy jest budowa i przetestowanie bezprzewodowego, rozproszonego systemu pomiaru warunków środowiskowych. Model systemu przewiduje dwa sensory zbierające dane (ciśnienie atmosferyczne, temperaturę powietrza oraz wilgotność względną) oraz moduł centralny zbierający odczyty i przesyłający je do komputera celem analizy i wizualizacji. Przesyłanie danych pomiędzy sensorami, a centralką zrealizowano za pomocą protokołu Bluetooth Low Energy.

4.2. Projekt urządzenia

Schemat blokowy zaproponowanego urządzenia pokazano na rysunku 4.1.



Rys. 4.3. Schemat blokowy planowanego urządzenia

4.2.1. Moduł ESP32 DevKit ESP-WROOM-32 V2

Do wykonania sensorów wybrano moduł ESP32 DevKit ESP-WROOM-32 V2 z chipsetem z serii ESP-32 (rys. 4.1.). Taki sam moduł posłużył jako centralka urządzenia.



Rys. 4.1. Moduł ESP32 DevKit ESP-WROOM-32 V2

Specyfikacja modułu:

- napięcie zasilania: 5 V – z microUSB,
- mikrokontroler Dual Core Tensilica LX6 240 MHz,
- pamięć SRAM: 520 KB,
- pamięć Flash: 4 MB,
- wbudowany układ WiFi 802.11BGN HT40,
- zabezpieczenia WiFi: WEP, WPA/WPA2, PSK/Enterprise, AES / SHA2 / Elliptical Curve Cryptography / RSA-4096,
- wbudowany moduł Bluetooth BLE,
- zintegrowany czujnik Halla
- 30 wyprowadzeń GPIO w tym:
 - 3x UART,
 - 3x SPI,
 - 2x I2C (2x I2S),
 - 12-kanałowy przetwornik ADC,
 - 2-kanałowy przetwornik DAC,
 - wyjścia PWM,
 - interfejs karty SD,
- wymiary: 55 x 28 x 8 mm.

Moduł ten wybrano ze względu na kompaktowość. Na jednej niewielkiej płytce znajduje się moduł BLE, interfejsy UART i SPI, a także mikrokontroler pozwalający na bezpośrednią obsługę czujnika środowiskowego. Wykorzystano trzy takie moduły. Po jednym do dwóch sensorów i jeden jako centralkę systemu.

4.2.2. Czujnik BME280

Parametry środowiskowe rejestrowane są przez czujnik BME280 zainstalowany na płytce testowej (rys. 4.2.). Pozwala on na odczyty zarówno temperatury, jak i ciśnienia oraz wilgotności.



Rys. 4.2. Moduł czujnika BME280

Specyfikacja modułu:

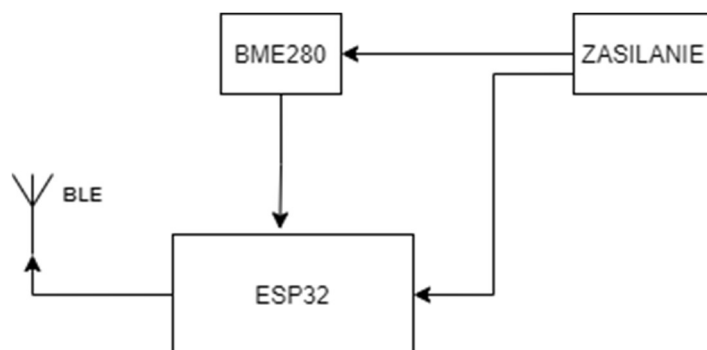
- napięcie pracy: 3,3 V lub 5 V,
- interfejs: I2C / SPI,
- temperatura:
 - zakres pomiarowy: od -40 °C do 85 °C,
 - rozdzielczość: 0,01 °C,
 - dokładność: ± 1 °C,
- wilgotność:
 - zakres pomiarowy: od 0 % do 100 % RH,
 - rozdzielczość: 0,008 % RH,
 - dokładność: ± 3 % RH,
 - czas odpowiedzi: 1 s,
 - opóźnienie: ≤ 2 % RH,
- ciśnienie:
 - zakres pomiarowy: od 300 hPa do 1100 hPa,
 - rozdzielczość: 0,18 Pa,
 - dokładność: ± 1 hPa,
- pobór prądu: do 0,1 mA @ 1 Hz,
- wymiary: 27 x 20 mm.

4.2.3. Zasilanie

Każdy z modułów sensorów zasilany jest przez zasilacz sieciowy 12 V. Następnie umieszczone na płytkach stabilizatory liniowe (wraz z pomocniczymi kondensatorami) obniżają napięcie do 3,3V.

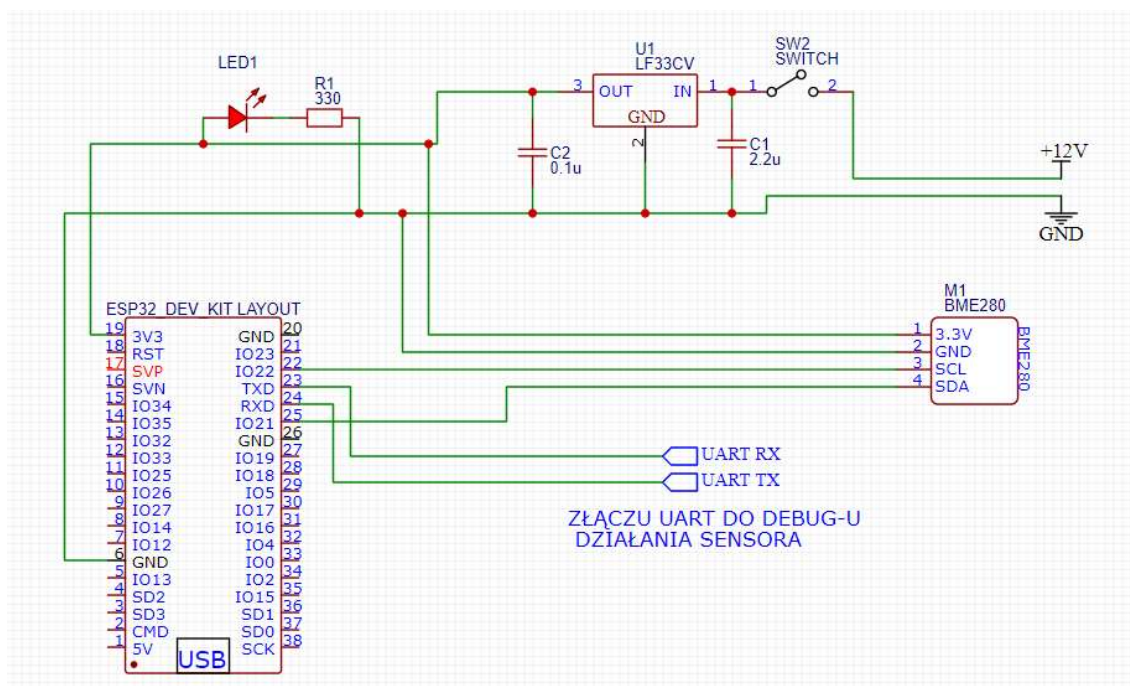
4.2.4. Budowa sensora i centralki

Schemat blokowy modułu sensora BME280 połączonego z modulem ESP32 DevKit wraz z zasilaniem pokazano na rysunku 4.4.



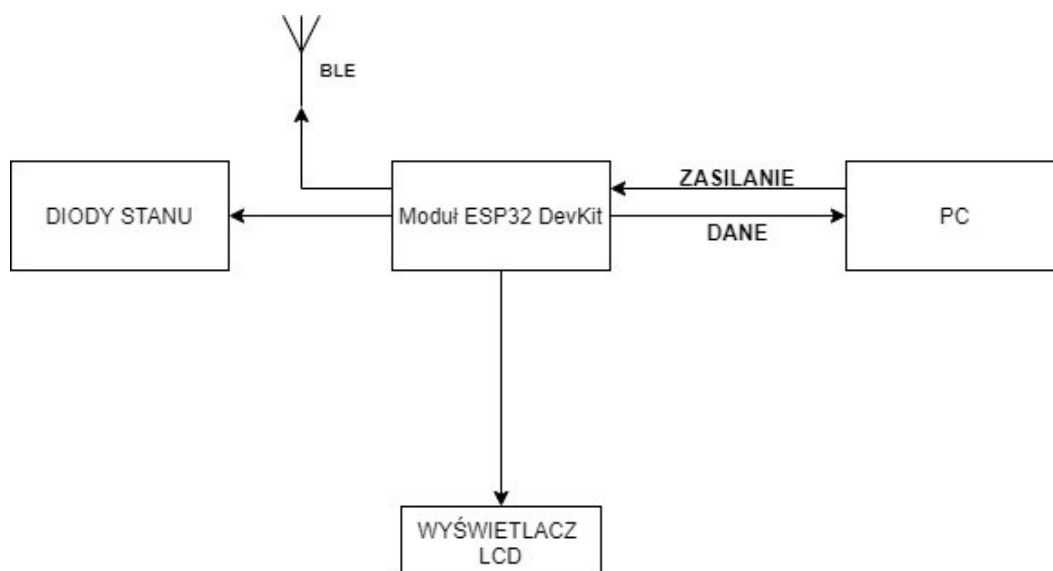
Rys. 4.4. Schemat blokowy modułu sensora

Poniżej przedstawiono schemat elektryczny modułu sensora (rys. 4.5.):



Rys. 4.5. Schemat elektryczny modułu sensora

Moduł centralny składa się z modułu ESP32 DevKit, wyświetlacza LCD oraz elementów pomocniczych (diody LED i rezystory). Zasilany jest poprzez przewód USB z komputera PC – przez ten sam przewód prowadzona jest komunikacja z PC. Schemat blokowy centralki pokazano na rysunku rys. 4.6.



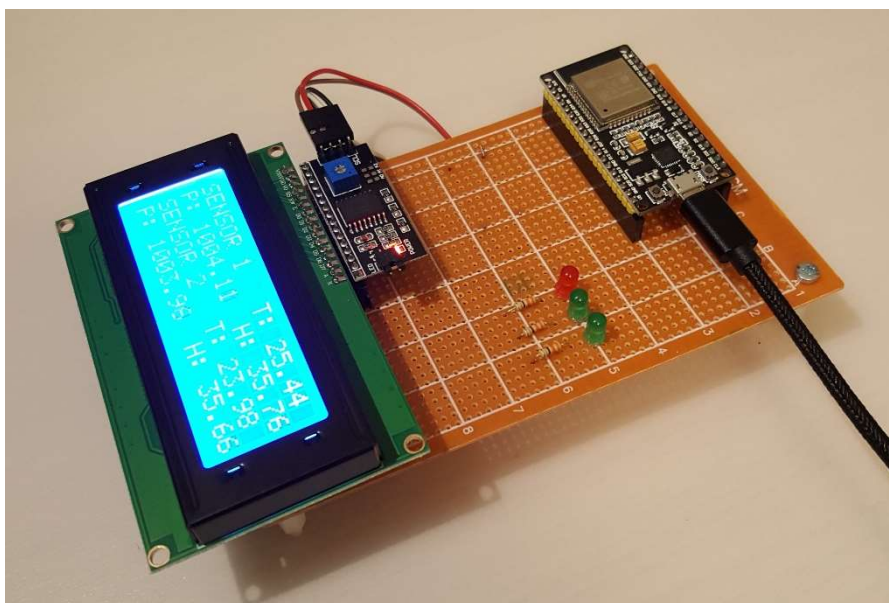
Na poniższym rysunku (rys. 4.7.) przedstawiono schemat elektryczny modułu centralki:

Diody LED oznaczone na schemacie jako S1 i S2 zapalają się w momencie zestawiania połączenia do danego sensora, a następnie gasną. Dalej sygnalizują moment odczytu danych z danego sensora. Dioda ERR zapala się, jeżeli połączenie z którymś z sensorów jest zerwane. W celu ułatwienia pracy z wyświetlaczem LCD, do układu włączono konwerter I²C, dzięki któremu wyświetlaczem sterować można poprzez magistralę I²C.

4.2.5. Montaż elementów systemu pomiarowego

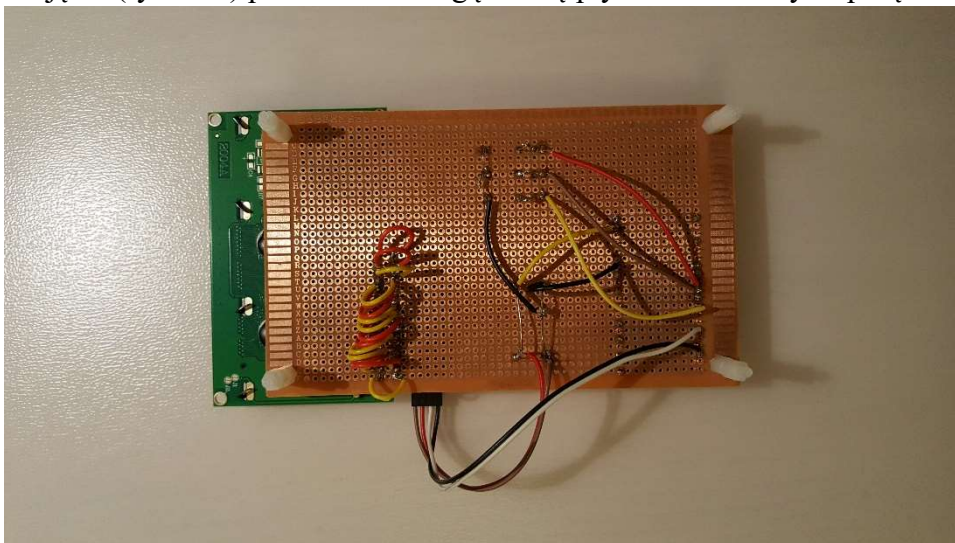
Poszczególne moduły zostały przetestowane z użyciem płytek stykowych, jednak aby układ był trwały, zlutowano elementy według przedstawionych schematów na płytkach uniwersalnych. W celu zagwarantowania możliwości powtórnego wykorzystania modułów nie lutowano ich bezpośrednio do płytek, lecz użyto gniazd kołkowych, w których osadzono moduły. Poniższe zdjęcia przedstawiają zlutowane układy.

Zdjęcie (rys. 4.8.) przedstawia moduł centralny wraz z podłączonym przewodem USB zapewniającym zarówno zasilanie, jak i łączność z komputerem za pomocą portu szeregowego UART.



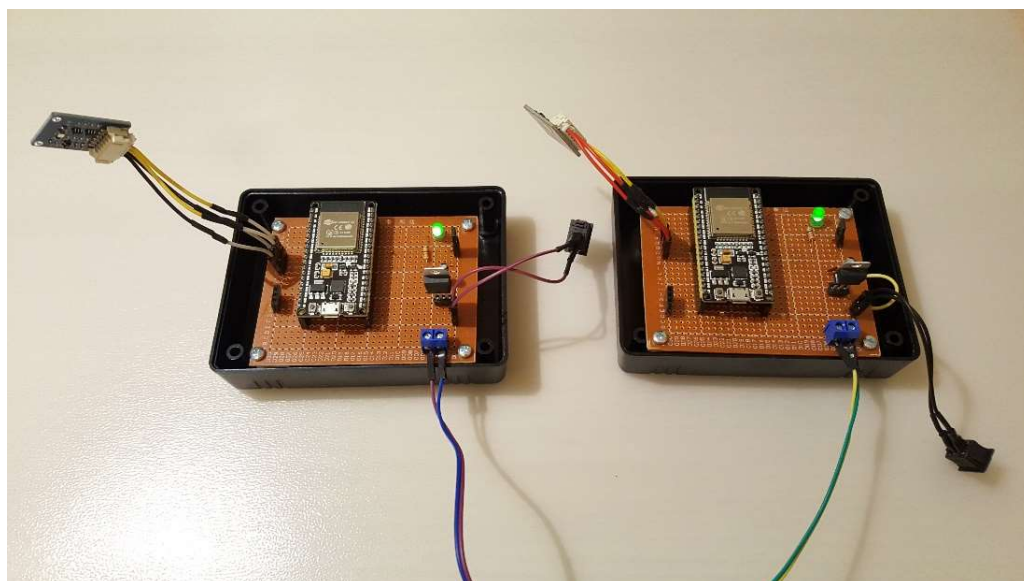
Rys. 4.8. Widok zmontowanej centrali

Zdjęcie (rys. 4.9.) przedstawia drugą stronę płytki z widocznymi połączeniami:



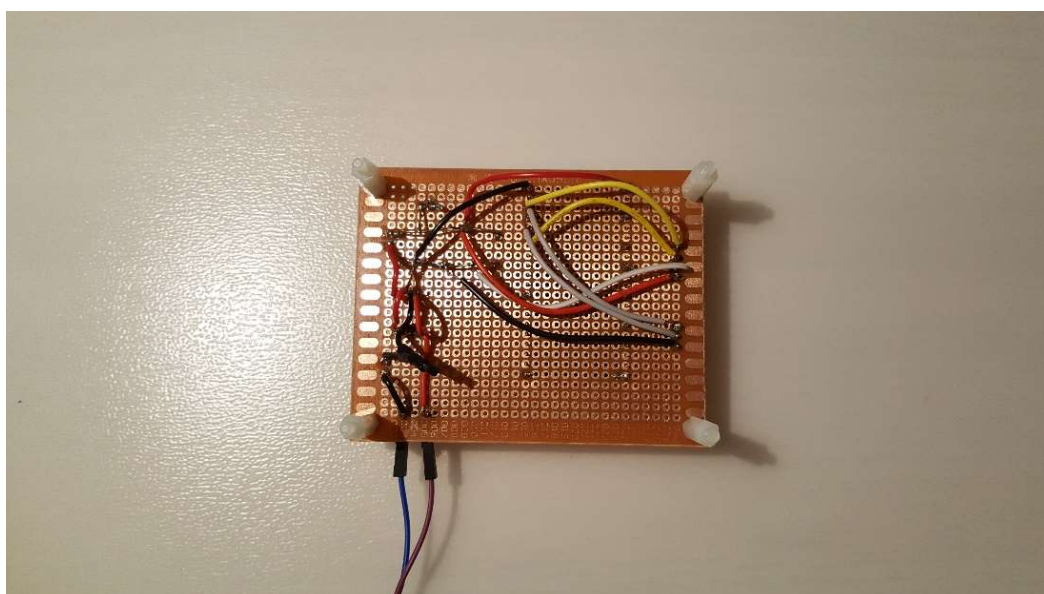
Rys. 4.9. Widok zmontowanej centrali (spód)

Zdjęcie (rys. 4.10.) przedstawia płytki sensorów:



Rys. 4.10. Widok zmontowanych modułów sensorów

Zdjęcie (rys. 4.11.) przedstawia drugą stronę płytki sensora z widocznymi połączeniami:



Rys. 4.11. Widok zmontowanych modułów sensorów (spód)

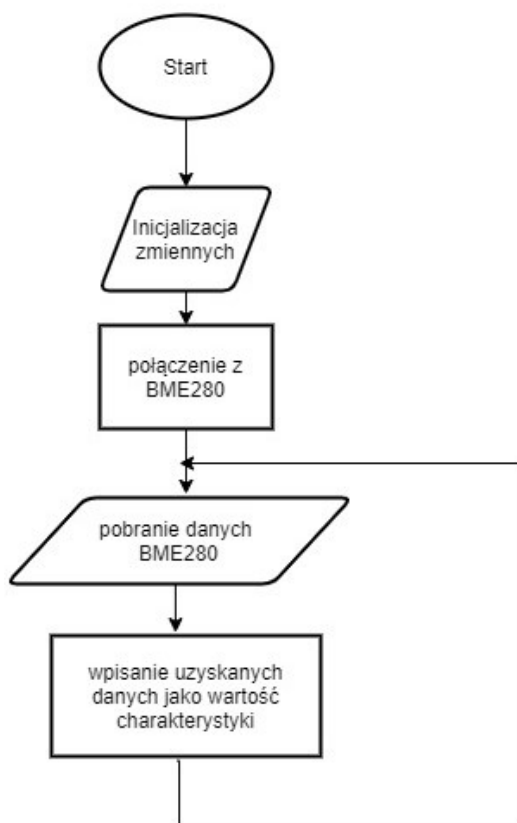
5. Opracowanie oprogramowania sterującego i monitorującego

5.1. Wstęp

Po przygotowaniu części sprzętowej przygotowano programy, które będą nią zarządzać. Oprogramowanie napisane na potrzeby tej pracy składa się z dwóch części:

- Programów sterujących płytkami ESP32 (moduły sensorów i moduł centralny) – powstało ono w środowisku Arduino IDE. Do napisania tych programów użyto bibliotek ESP32 BLE² wspomagających implementację protokołu Bluetooth Low Energy. Z kolei za obsługę czujników BME280 odpowiadają sterowniki firmy Adafruit³. Arduino IDE korzysta z języków C oraz C++.
- Program analizujący zebrane dane. Napisany został w środowisku MATLAB, które udostępnia różnorodne narzędzia analizy i wizualizacji danych.

Algorytm działania sensora jest prosty i został przedstawiony na rysunku 5.1.

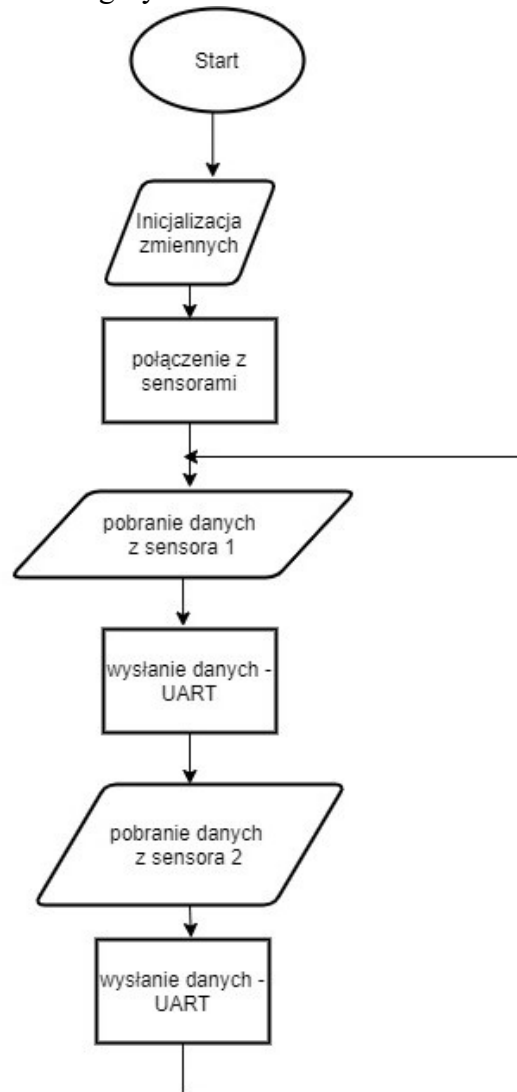


Rys. 5.1. Algorytm działania sensora

² https://github.com/nkolban/ESP32_BLE_Arduino

³ https://github.com/adafruit/Adafruit_BME280_Library

Na rysunku 5.5. przedstawiono algorytm działania centralki.



Rys. 5.2. Algorytm działania centralki

W trakcie pisania niniejszego rozdziału korzystano ze źródeł: [16-17].

5.2. Sterowniki sensora

Do sterowania modułem sensora przygotowano kod przedstawiony poniżej:

```
#include <Wire.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"

Adafruit_BME280 bme; // I2C

unsigned long delayTime;

void setup() {
```



```

Serial.begin(9600);
while (!Serial);

unsigned status;
status = bme.begin();
if (!status) {
    Serial.println("Czujnik BME280 nie wykryty!");
    while (1) delay(10);
}
delayTime = 1000;
delay(delayTime * 3);
}

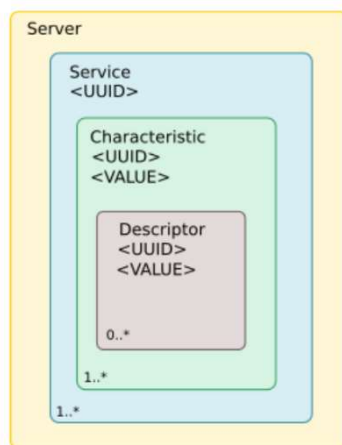
void loop() {
    float temperature, pressure, humidity;
    BLEDevice::init("Sensor_1");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);
    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ |
        BLECharacteristic::PROPERTY_WRITE
    );
    pCharacteristic->setValue("test sensora 1");
    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    pAdvertising->setScanResponse(true);
    pAdvertising->setMinPreferred(0x06);
    pAdvertising->setMinPreferred(0x12);
    while (1) {
        printValues(); //for debug
        pAdvertising->stop();
        temperature = bme.readTemperature();
        pressure=bme.readPressure() / 100.0F;
        humidity=bme.readHumidity();
        char buffer_data[24];
        snprintf(buf2, 24, "%.2f, %.2f, %.2f\r\n", temperature, pressure, humidity);
        pCharacteristic->setValue(buffer_data);
        pCharacteristic->notify();
        BLEDevice::startAdvertising();
        delay(delayTime);
    }
    delay(delayTime);
}

void printValues() {
    Serial.print(bme.readTemperature());
    Serial.print(", ");
    Serial.print(bme.readPressure() / 100.0F);
    Serial.print(", ");
    Serial.print(bme.readHumidity());
    Serial.println();
}

```

Poniżej omówiono najważniejsze fragmenty kodu:

- `bme.begin()` – funkcja ta inicjalizuje czujnik BME280
- `BLEDevice::init("Sensor_1")` – inicjalizacja protokołu BLE
- `BLEServer *pServer = BLEDevice::createServer();`
 - `BLEService *pService = pServer->createService(SERVICE_UUID);`
 - `BLECharacteristic *pCharacteristic = pService->createCharacteristic(`
 - `CHARACTERISTIC_UUID,`
 - `BLECharacteristic::PROPERTY_READ |`
 - `BLECharacteristic::PROPERTY_WRITE);` – ten fragment identyfikuje urządzenie jako serwer (w nomenklaturze BLE urządzenie nadające dane nazywa się serwerem) oraz uruchamia usługę wraz z charakterystyką, która będzie zawierać wyniki pomiarów (patrz rys. 5.3.)



Rys. 5.3. Schemat usługi BLE

- `pService->start();` – uruchamia utworzoną usługę
- `BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();`
 - o `pAdvertising->addServiceUUID(SERVICE_UUID);` – uruchamia „advertising”, co sprawia, że usługa i jej charakterystyka będą widoczne dla innych urządzeń
- `temperature = bme.readTemperature();`
 - o `pressure=bme.readPressure() / 100.0F;`
 - o `humidity=bme.readHumidity();`
 - o `char buffer_data[24];`
 - o `snprintf(buffer_data, 24, "%.2f, %.2f, %.2f\r\n", temperature, pressure, humidity);`
 - o `pCharacteristic->setValue(buffer_data);` – w tym fragmencie odczytywane są pomiary temperatury, ciśnienia i wilgotności, a następnie tworzona jest zmienna znakowa, do której są one wpisywane. Zmienna `buffer_data` jest następnie wpisywana jako wartość charakterystyki serwera.

5.3. Sterownik centrali

Do sterowania modułem centrali przygotowano kod przedstawiony poniżej:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEAddress.h>
#include <LiquidCrystal_I2C.h>

int lcdColumns = 16;
int lcdRows = 2;
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
static BLEUUID charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");

static int timesRead = 0;
long int freeMem;
const int howManySensors = 2;
static std::string validSensors[howManySensors] = {"7c:9e:bd:f8:a9:c2",
"7c:9e:bd:f9:f9:fa"};

const byte led_gpio[howManySensors] = {16, 17};
const byte led_error = 18;

std::string value;
float temperature, pressure, humidity;

static bool connection_status[howManySensors] = {false, false};
static byte failed_connections[howManySensors] = {0, 0};
static bool skipLoop = false;
```

```

static BLERemoteCharacteristic *pRemoteCharacteristics[howManySensors];
static BLERemoteService *pRemoteServices[howManySensors];
static BLEAddress *pServerAddress[howManySensors];
static BLEClient *pClients[howManySensors];

class MyClientCallback : public BLEClientCallbacks {
    void onConnect(BLEClient* pclient) {
        Serial.println("Połączono");
    }

    void onDisconnect(BLEClient* pclient) {
        Serial.println("Rozłączono");
        delay(3000);
        digitalWrite(led_error, HIGH);
    }
};

static MyClientCallback *clientCallback;

void setup() {
    Serial.begin(115200);
    lcd.init(); lcd.backlight(); lcd.clear();
    BLEDevice::init("Cent_Dev");
    clientCallback = new MyClientCallback();

    for (int i = 0; i < howManySensors; i++) {
        pinMode(led_gpio[i], OUTPUT);
        digitalWrite(led_gpio[i], LOW);
    }
    pinMode(led_error, OUTPUT);
    digitalWrite(led_error, LOW);

    for (int i = 0; i < howManySensors; i++) {
        Serial.print("Sensor nr: "); Serial.println(i + 1);
        pServerAddress[i] = new BLEAddress(validSensors[i]);
        pClients[i] = BLEDevice::createClient();
        delay(100);
        pClients[i]->setClientCallbacks(clientCallback);
        delay(100);
        Serial.print("SETUP ENDED for sensor nr: "); Serial.println(i + 1);
    }
}

void loop() {
    for (int i = 0; i < howManySensors; i++) {
        failed_connections[i] = 0;
    }
    digitalWrite(led_error, LOW);

    for (int i = 0; i < howManySensors; i++) {
        Serial.print("BEFORE CONNECTION: SENSOR "); Serial.println(i + 1);
        if (connection_status[i] == false) {
            pClients[i]->connect(*pServerAddress[i]);
            delay(100);
        }
        connection_status[i] = pClients[i]->isConnected();
        Serial.print("Wynik isConnected(): "); Serial.println(connection_status[i]);
        if (connection_status[i] == true) {
            digitalWrite(led_gpio[i], HIGH);
            delay(100);
            Serial.print("AFTER CONNECTION: SENSOR "); Serial.println(i + 1);
            delay(100);
            pRemoteServices[i] = pClients[i]->getService(serviceUUID);
            pRemoteCharacteristics[i] = pRemoteServices[i]->getCharacteristic(charUUID);
            delay(2000);
            digitalWrite(led_gpio[i], LOW);
        } else {
            Serial.print("SENSOR "); Serial.print(i + 1); Serial.println(" NOT FOUND");
        }
    }
    delay(300);

    while (1) {
        for (int i = 0; i < howManySensors; i++) {
            if (connection_status[i] == true && pRemoteCharacteristics[i]->canRead()) {
                digitalWrite(led_gpio[i], HIGH);
                value = pRemoteCharacteristics[i]->readValue();
            }
        }
    }
}

```



```

char * strtokIndx;
strtokIndx = strtok((char *)value.c_str(), ",");
temperature = atof(strtokIndx);
strtokIndx = strtok(NULL, ",");
pressure = atof(strtokIndx);
strtokIndx = strtok(NULL, ",");
humidity = atof(strtokIndx);
Serial.print(i); Serial.print(", "); Serial.print(temperature); Serial.print(", 
"); Serial.print(pressure); Serial.print(", "); Serial.println(humidity);
Serial.println();

lcd.setCursor(0, i);
lcd.print(temperature); lcd.print(", "); lcd.print(pressure); lcd.print(", ");
lcd.println(humidity);
digitalWrite(led_gpio[i], LOW);
delay(200);
//pClients[i]->disconnect();
} else {
    Serial.print(i); Serial.print(", "); Serial.print("00"); Serial.print(", ");
Serial.print("00"); Serial.print(", "); Serial.println("00");
    Serial.println();
    digitalWrite(led_error, HIGH);
    delay(1000);
    if (connection_status[i] == true) {
        pClients[i]->disconnect();
    }
    if (failed_connections[i]++ >= 5) skipLoop = true;
}
}
Serial.print("Obiegi: "); Serial.println(timesRead++);
delay(350);
if (skipLoop == true) {
    skipLoop = false;
    Serial.println("WYSKAKUJE!");
    break;
}
} //while
}

```

Najważniejsze punkty kodu:

- `BLEDevice::init("Cent_Dev");` – inicjalizacja protokołu BLE
- `pClients[i] = BLEDevice::createClient();` – stworzenie instancji klienta BLE. Ponieważ do „obsłużenia” są dwa serwery (sensory), ich adresy wpisano do tablicy adresów (`pServerAddress[i] = new BLEAddress(validSensors[i]);`), z którymi następnie kolejno się połączono (`pClients[i]->connect(*pServerAddress[i])`).
- `pRemoteServices[i] = pClients[i]->getService(serviceUUID);`
 - `pRemoteCharacteristics[i] = pRemoteServices[i]->getCharacteristic(charUUID);` – następnie podano identyfikator żądanej usługi i jej charakterystyki
- `value = pRemoteCharacteristics[i]->readValue();`
`char * strtokIndx;`
`strtokIndx = strtok((char *)value.c_str(), ",");`
`temperature = atof(strtokIndx);`
`strtokIndx = strtok(NULL, ",");`
`pressure = atof(strtokIndx);`
`strtokIndx = strtok(NULL, ",");`
`humidity = atof(strtokIndx);` – tu następuje odczytanie wartości zdalnej charakterystyki i rozbitcie odczytanego stringu na trzy zmienne zmiennoprzecinkowe
- `Serial.print(i); Serial.print(", "); Serial.print(temperature);`
`Serial.print(", "); Serial.print(pressure); Serial.print(", ");`
`Serial.println(humidity);` – następuje przesłanie odczytów przez port szeregowy do komputera celem zapisania i analizy

5.4. Wyświetlanie i prezentacja pomiarów

Dane odczytywane są na komputerze przez program *SerialPlot*⁴, posiadający możliwość zapisu odczytywanych danych do pliku tekstowego. Uzyskane wyniki zostaną zaimportowane do programu *data_analyser* napisanego w środowisku MATLAB, który posłuży do ich analizy i wizualizacji.

```
function data_analyser

format bank %format - dwa miejsca dziesiętne
disp("Welcome!")

%-----Pobranie danych-----
filename="pomiar.txt"; % plik z danymi
if isfile(filename)
    % File exists.
    disp("Znaleziono plik... Analizuję...");
else
    warningMessage = sprintf('Warning: file does not exist:\n%s', filename);
    uiwait(msgbox(warningMessage));
    return;
end

B=readmatrix(filename); % wczytanie pliku
[rows, ~]=size(B);%rozmiar wczytanego pliku

sensor1_data=zeros(rows,3); % alokacja pamięci na dane z serwerów
indexS1=1; % indeks pomocniczy do rozdzielania danych
sensor2_data=zeros(rows,3);
indexS2=1;

%-----Rozdzielenie danych-----
for i = 1:1:rows
    if B(i,1) == 0.00
        sensor1_data(indexS1, 1)= B(i, 2);
        sensor1_data(indexS1, 2)= B(i, 3);
        sensor1_data(indexS1, 3)= B(i, 4);
        indexS1=indexS1+1;
    elseif B(i,1) == 1.00
        sensor2_data(indexS2, 1)= B(i, 2);
        sensor2_data(indexS2, 2)= B(i, 3);
        sensor2_data(indexS2, 3)= B(i, 4);
        indexS2=indexS2+1;
    else
        disp("Błąd parsowania linii:\n")
        disp(B(i, :));
    end
end

sensor1_data( ~any(sensor1_data,2), : ) = []; %usuwanie rzędów wypełnionych samymi zerami
sensor2_data( ~any(sensor2_data,2), : ) = []; %

%-----Analiza danych-----

analyseData(sensor1_data, 1);
analyseData(sensor2_data, 2);
compareSensors(sensor1_data, sensor2_data);
end

function compareSensors(s1_data, s2_data)
    disp('Compare sensors')
    temp1=s1_data(:,1);
    temp2=s2_data(:,1);
    press1=s1_data(:,2);
    press2=s2_data(:,2);
    hum1=s1_data(:,3);
    hum2=s2_data(:,3);
```

⁴ <https://hackaday.io/project/5334-serialplot-realtime-plotting-software>

```

figure('Name', 'Temperatura S1 & S2', 'NumberTitle','off');
plot(temp1, 'DisplayName', 'Sensor 1');
hold on;
plot(temp2, 'DisplayName', 'Sensor 2');
title("porównanie temperatury");
xlabel('Czas pomiaru [hh:mm]');
xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
xtickangle(45);
legend('show');
ylabel(['temperatura [' char(176) 'C]']);
hold off;

figure('Name', 'Ciśnienie S1 & S2', 'NumberTitle','off');
plot(press1, 'DisplayName', 'Sensor 1');
hold on;
plot(press2, 'DisplayName', 'Sensor 2');
title("porównanie ciśnienia");
xlabel('Czas pomiaru [hh:mm]');
xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
xtickangle(45);
legend('show');
ylabel(['ciśnienie [kPa]']);
hold off;

figure('Name', 'Wilgotność S1 & S2', 'NumberTitle','off');
plot(hum1, 'DisplayName', 'Sensor 1');
hold on;
plot(hum2, 'DisplayName', 'Sensor 2');
title("porównanie wilgotności");
xlabel('Czas pomiaru [hh:mm]');
xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
xtickangle(45);
legend('show');
ylabel(['wilgotność [%]']);
hold off;

end
function analyseData(data, nmb_of_sensor)
    [data_rows, ~] = size(data);
    temp=data(:,1);
    avg_temp=mean(temp,'omitnan');
    max_temp=max(temp);
    min_temp=min(temp);
    std_temp=std(temp,'omitnan');

    pressure=data(:,2);
    avg_press=mean(pressure,'omitnan');
    max_press=max(pressure);
    min_press=min(pressure);
    std_press=std(pressure,'omitnan');

    humidity=data(:,3);
    avg_hum=mean(humidity,'omitnan');
    max_hum=max(humidity);
    min_hum=min(humidity);
    std_hum=std(humidity,'omitnan');

    x = linspace(0,1,data_rows);

    plot_title=sprintf("Sensor %d", nmb_of_sensor);
    figure('Name',plot_title, 'NumberTitle','off');
    line_spec= {'r-', 'LineWidth', 0.5};

    subplot(2,2,1)
    plot(temp, line_spec{:});
    hold on;
    yline(avg_temp, 'b:', 'LineWidth', 2);
    title("Sensor " + nmb_of_sensor+ " - temperatura");

```

```

        xlabel('Czas pomiaru [hh:mm]');
        xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
    xtickangle(45);
    ylabel(['temperatura [' char(176) 'C']]);
    hold off;

    subplot(2,2,2)
    plot(pressure, line_spec{:});
    hold on;
    yline(avg_press, 'b:', 'LineWidth',2);
    title("Sensor " + nmb_of_sensor+ " - ciśnienie");
    xlabel('Czas pomiaru [hh:mm]');
    xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
    xtickangle(45);
    ylabel('ciśnienie [kPa]');
    hold off;

    subplot(2,2,3)
    plot(humidity, line_spec{:});
    hold on;
    yline(avg_hum, 'b:', 'LineWidth',2);
    title("Sensor " + nmb_of_sensor+ " - wilgotność");
    xlabel('Czas pomiaru [hh:mm]');
    xticks(linspace(0,41409,12));

set(gca,'XTickLabel',{'12:00','13:00','14:00','15:00','16:00','17:00','18:00','19:00','20:00','21:00','22:00','23:00'});
    xtickangle(45);
    ylabel('wilgotność względna [%]');
    hold off;

    analysis_params = {sprintf('Średnia temperatura: %.2f°C\n', avg_temp,char(176)),
    ...
        sprintf('Temperatura zawiera się w przedziale: %.2f°C do %.2f°C\n', min_temp, char(176), max_temp,char(176)), ...
        sprintf('Średnie ciśnienie: %.2f kPa\n', avg_press), ...
        sprintf('Ciśnienie zawiera się w przedziale: %.2f kPa do %.2f kPa\n', min_press, max_press), ...
        sprintf('Średnia wilgotność: %.2f %%\n', avg_hum), ...
        sprintf('Wilgotność zawiera się w przedziale: %.2f%% do %.2f%%\n', min_hum, max_hum), ...
        sprintf(['Odchylenie standardowe:\n ' ...
        'Temperatura: %.2f°C Ciśnienie: %.2f kPa Wilgotność: %.2f%%'],
std_temp, char(176), std_press, std_hum),
        };

    subplot(2,2,4)
    text(0,0.50,analysis_params);
    set(gca,'XTick',[], 'YTick', [1]);
    set(gca,'visible','off')

%-----HISTOGRAMY-----

draw_gauss=0;

plot_title=sprintf("Sensor %d - histogramy", nmb_of_sensor);
figure('Name',plot_title, 'NumberTitle','off');

subplot(2,2,1)
data_to_draw=temp;
current_mean=avg_temp;
current_std=std_temp;
current_label='temperatura';

current_histogram=histogram(data_to_draw);
current_histogram.DisplayName='dane';
title("Sensor " + nmb_of_sensor+ " - " + current_label);
xlabel(['current label ' [' char(176) 'C']]);
ylabel('Liczebność');
hold on;

```

```

    plot(conv(current_histogram.BinEdges, [0.5 0.5], 'valid'),
current_histogram.BinCounts, 'DisplayName', 'test', 'LineWidth', 2);
    xline(current_mean-current_std, '-g', {'SR-sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean-current_std));
    xline(current_mean, '-k', {'SR'}, 'LineWidth', 2, 'DisplayName',
num2str(current_mean));
    xline(current_mean+current_std, '-g', {'SR+sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean+current_std));
    hold off;

    subplot(2,2,2)
    data_to_draw=pressure;
    current_mean=avg_press;
    current_std=std_press;
    current_label='ciśnienie';
    current_histogram=histogram(data_to_draw);
    current_histogram.DisplayName='dane';
    title("Sensor " + nmb_of_sensor+ " - " + current_label);
    xlabel([current_label ' [kPa]']);
    ylabel('Liczebność');
    hold on;

    plot(conv(current_histogram.BinEdges, [0.5 0.5], 'valid'),
current_histogram.BinCounts, 'DisplayName', 'test', 'LineWidth', 2);
    xline(current_mean-current_std, '-g', {'SR-sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean-current_std));
    xline(current_mean, '-k', {'SR'}, 'LineWidth', 2, 'DisplayName',
num2str(current_mean));
    xline(current_mean+current_std, '-g', {'SR+sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean+current_std));
    hold off;

    subplot(2,2,3)
    data_to_draw=humidity;
    current_mean=avg_hum;
    current_std=std_hum;
    current_label='wilgotność';
    current_histogram=histogram(data_to_draw);
    current_histogram.DisplayName='dane';
    title("Sensor " + nmb_of_sensor+ " - " + current_label);
    xlabel([current_label ' [%]']);
    ylabel('Liczebność');
    hold on;

    plot(conv(current_histogram.BinEdges, [0.5 0.5], 'valid'),
current_histogram.BinCounts, 'DisplayName', 'test', 'LineWidth', 2);
    xline(current_mean-current_std, '-g', {'SR-sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean-current_std));
    xline(current_mean, '-k', {'SR'}, 'LineWidth', 2, 'DisplayName',
num2str(current_mean));
    xline(current_mean+current_std, '-g', {'SR+sigma'}, 'LineWidth', 1, 'DisplayName',
num2str(current_mean+current_std));
    hold off;

    analysis_params = {sprintf('Temperatura: (%.2f %c %.2f)%c \n ',avg_temp, char(177),
std_temp, char(176)), ...
        sprintf('Ciśnienie: (%.2f %c %.2f) kPa \n', avg_press,
char(177), std_press), ...
        sprintf('Wilgotność: (%.2f %c %.2f) %% \n', avg_hum, char(177),
std_hum), ...
        };

    subplot(2,2,4)
    text(0,0.50,analysis_params);
    set(gca,'XTick',[], 'YTick', []);
    set(gca,'visible','off')

end

```

Aby przetestować powyższy kod przygotowano testowy zestaw danych (w postaci przebiegów piłokształtnych) za pomocą następującego skryptu:

```

function random_data_gen
Tmin=21.00;
Tmax=30.00;
Pmin=1000.00;
Pmax=1020.00;
Hmin=0;
Hmax=100;

nmb_to_gen=75;
file_to_gen="rnd_data.txt";

fid = fopen( file_to_gen, 'wt' );
s=0;

temp_val=[linspace(Tmin, Tmax, nmb_to_gen/3),linspace(Tmin, Tmax,
nmb_to_gen/3),linspace(Tmin, Tmax, nmb_to_gen/3)];
pres_val=[linspace(Pmin, Pmax, nmb_to_gen/3),linspace(Pmin, Pmax,
nmb_to_gen/3),linspace(Pmin, Pmax, nmb_to_gen/3)];
hum_val=[linspace(Hmin, Hmax, nmb_to_gen/3),linspace(Hmin, Hmax,
nmb_to_gen/3),linspace(Hmin, Hmax, nmb_to_gen/3)];

for i = 1:1:nmb_to_gen

    fprintf(fid, "%d,%.2f,%.2f,%.2f\n", s,temp_val(i),pres_val(i),hum_val(i));
    if (s==0)
        s=1;
    elseif (s==1)
        s=0;
    end
end

fclose('all');
end

```

6. Wykonanie badań testowych

6.1. Zużycie energii sensora

W czasie pracy urządzenia mierzono również natężenie zużywanego prądu (za pomocą multimetru DT830B). W czasie obserwacji wskazania miernika wahały się pomiędzy 63,7 mA a 80,9 mA. Średnio moduł sensora zużywał więc 72,3 mA.

Zakładając, że urządzenie zasilane byłoby baterią 9V typu 6LR61, która posiada zazwyczaj pojemność na poziomie około 550 mAh, moduł sensora mógłby pracować przez:

$$t_{6LR61} = \frac{550 \text{ mAh}}{72,3 \text{ mA}} \cong 7,61 \cong 7 \text{ h } 36 \text{ m} \quad (6.1)$$

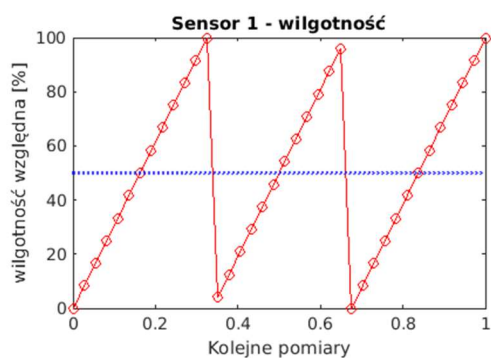
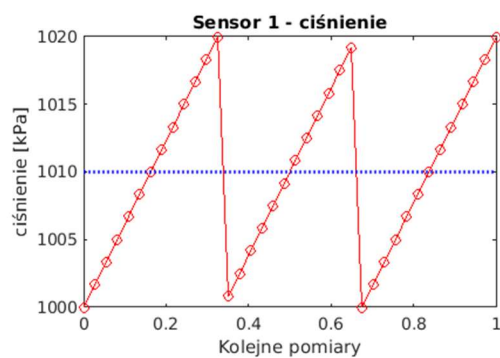
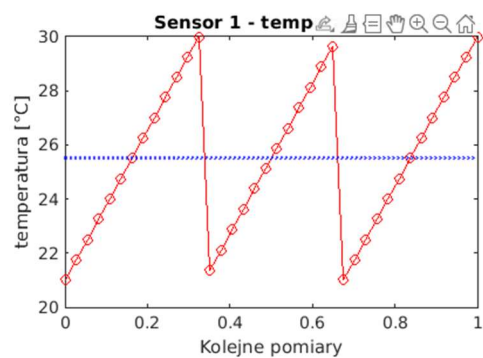
Jeżeli natomiast użyto by baterii typu 6F22, która jest baterią cynkowo-węglową o pojemności około 400 mAh, to czas pracy wyniósłby:

$$t_{6F22} = \frac{400 \text{ mAh}}{72,3 \text{ mA}} \cong 5,53 \cong 5 \text{ h } 31 \text{ m} \quad (6.2)$$

Dlatego w czasie testów zdecydowano się na zasilanie w postaci zasilaczy sieciowych (12 V), które umożliwiły ciągłą pracę przez 11 godzin.

6.2. Badania z użyciem danych symulowanych

Przygotowany program analizujący dane przetestowano za pomocą wygenerowanych danych symulowanych (przebiegów piłokształtnych). Na rysunkach 6.1 oraz 6.2 zawarto działanie analizatora na danych testowych.



Średnia temperatura: 25.50°C

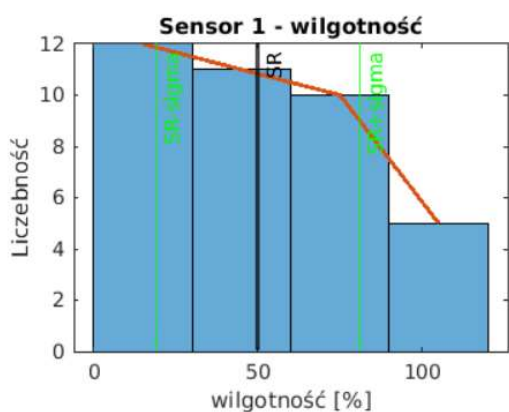
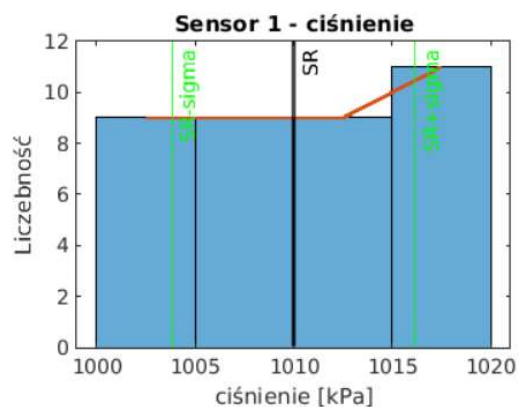
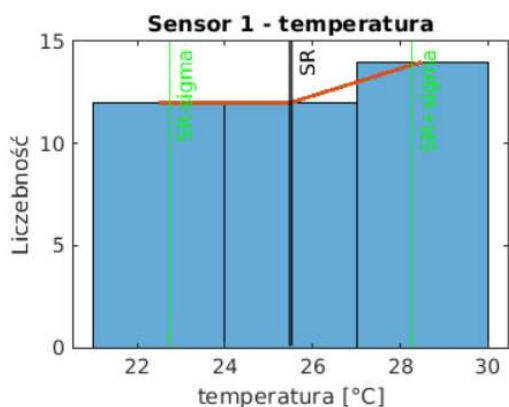
Temperatura zawiera się w przedziale: 21.00°C do 30.00°C

Średnie ciśnienie: 1010.00 kPa

Ciśnienie zawiera się w przedziale: 1000.00 kPa do 1020.00 kPa

Średnia wilgotność: 50.00 %

Wilgotność zawiera się w przedziale: 0.00% do 100.00%

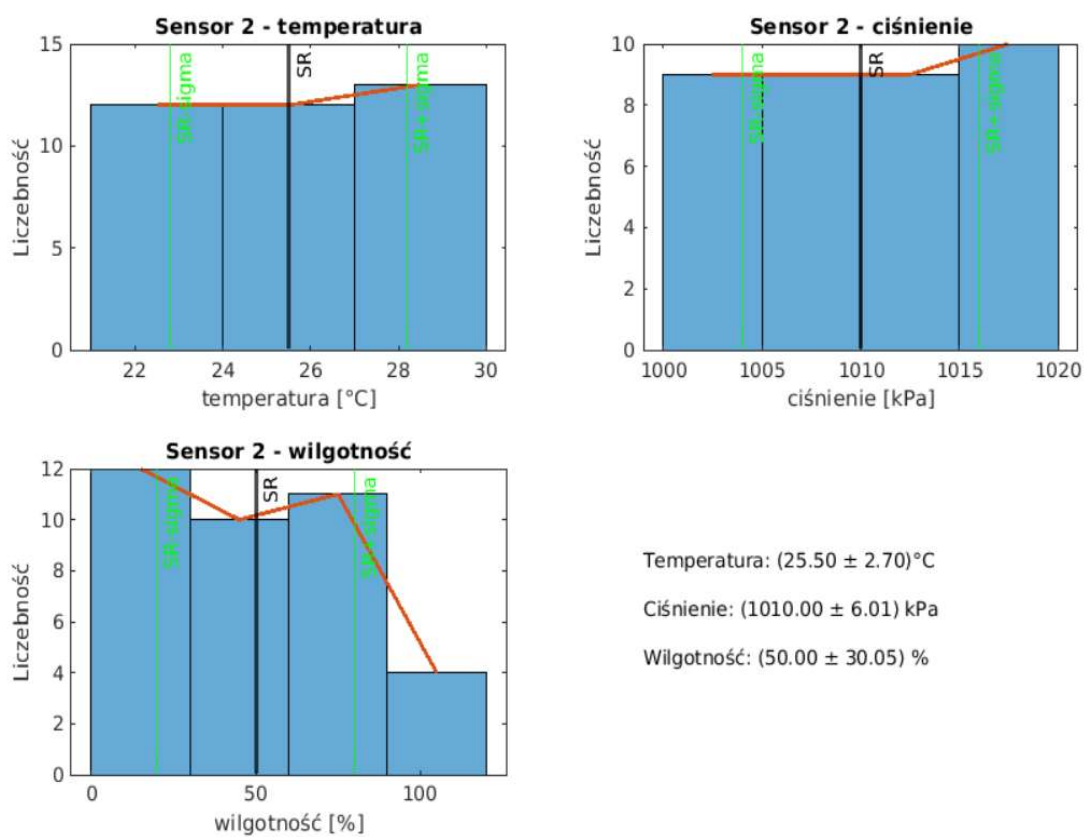
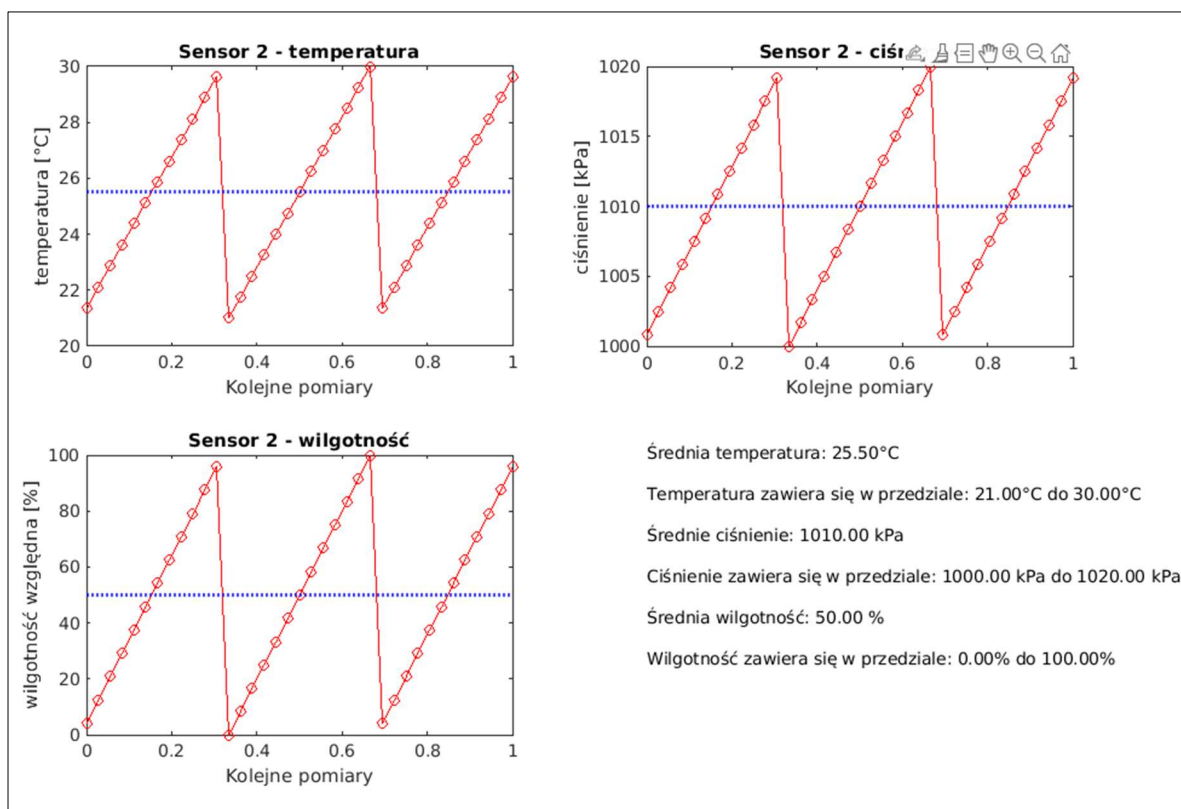


Temperatura: $(25.50 \pm 2.78)^{\circ}\text{C}$

Ciśnienie: $(1010.00 \pm 6.17) \text{ kPa}$

Wilgotność: $(50.00 \pm 30.85) \%$

Rys. 6.1. Analiza danych testowych – sensor 1



Rys. 6.2. Analiza danych testowych – sensor 2

6.3. Akwizycja danych rzeczywistych

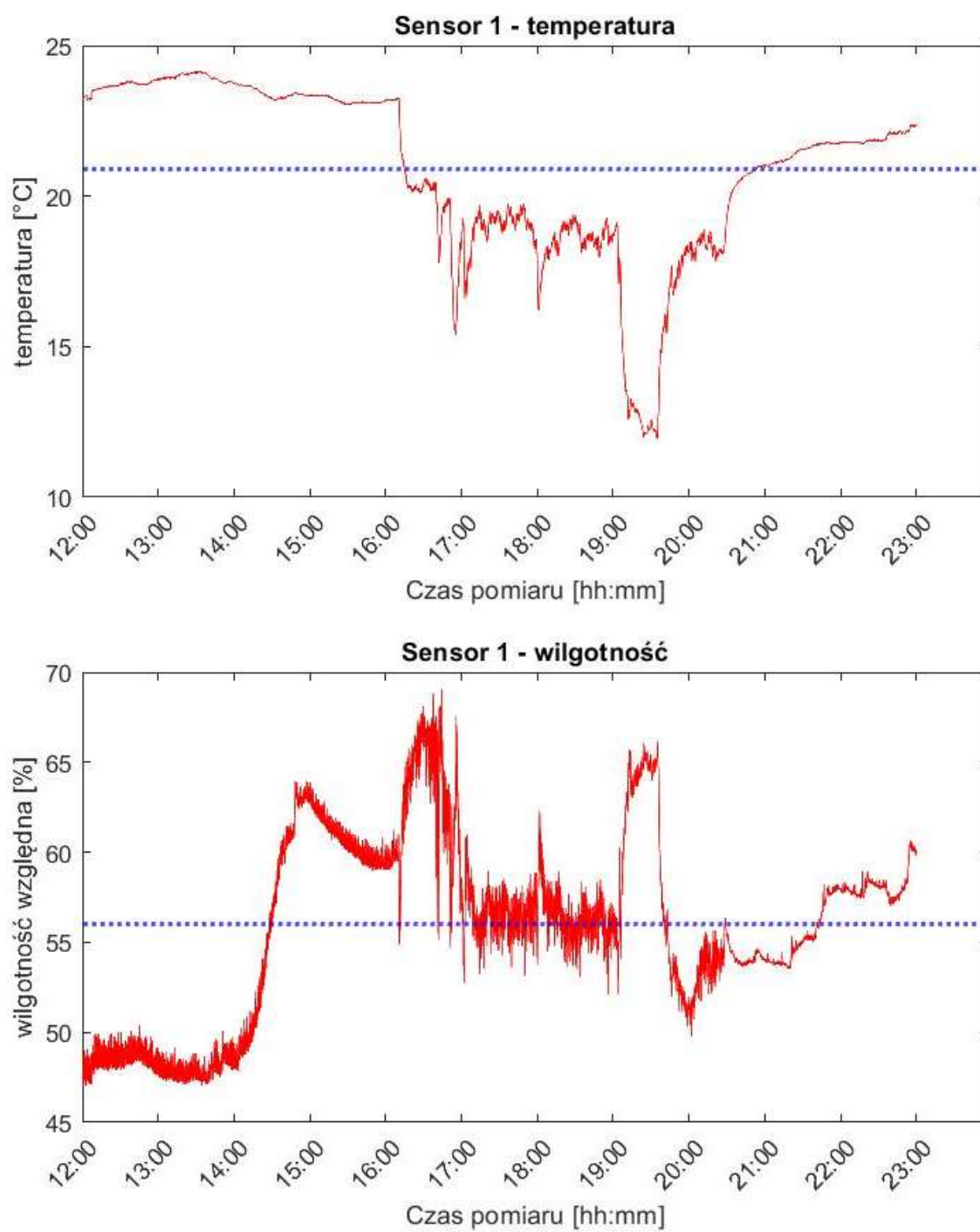
Badania testowe wykonano w dniu 01.05.2021 roku, między godzinami 12:00 a 23:00 (11 godzin). Czujniki umieszczono w jednym pomieszczeniu. Jeden – na parapecie okna, drugi – na środku pomieszczenia. Jedenaście godzin pomiarów zapisane zostało do pliku tekstowego *pomiary.txt*. Kilka przykładowych rekordów:

```
...  
1.000,24.630,1002.560,46.230  
0.000,23.240,1002.510,47.240  
1.000,24.630,1002.590,46.590  
0.000,23.250,1002.530,47.300  
1.000,24.640,1002.590,48.080  
0.000,23.250,1002.550,47.370  
...
```

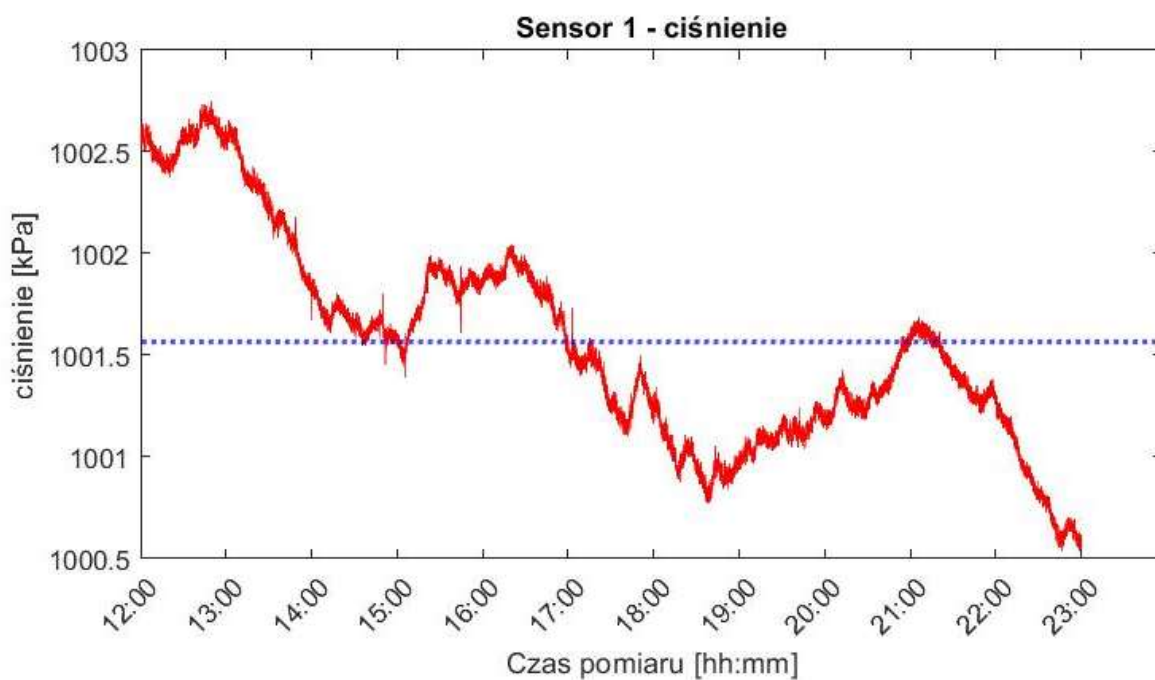
Jak widać centralka odbierała dane na przemian z obu sensorów (początkowa liczba to numer sensora: „0” – pierwszy sensor, „1” – drugi sensor). W ciągu badań testowych odebrano 82818 linii pomiarów (po 41409 na sensor), co oznacza 2,29 MB (czyli 2 401 753 bajtów) uzyskanych danych.

6.4. Analiza danych

Następnie uzyskane dane zaimportowano do programu *data_analyser*. Wynikiem jego działania są poniższe wykresy (rys. 6.3. i 6.4.):



Rys. 6.3a. Analiza danych pomiarowych – sensor 1



Średnia temperatura: 20.89°C

Temperatura zawiera się w przedziale: 11.94°C do 24.14°C

Średnie ciśnienie: 1001.56 kPa

Ciśnienie zawiera się w przedziale: 1000.54 kPa do 1002.75 kPa

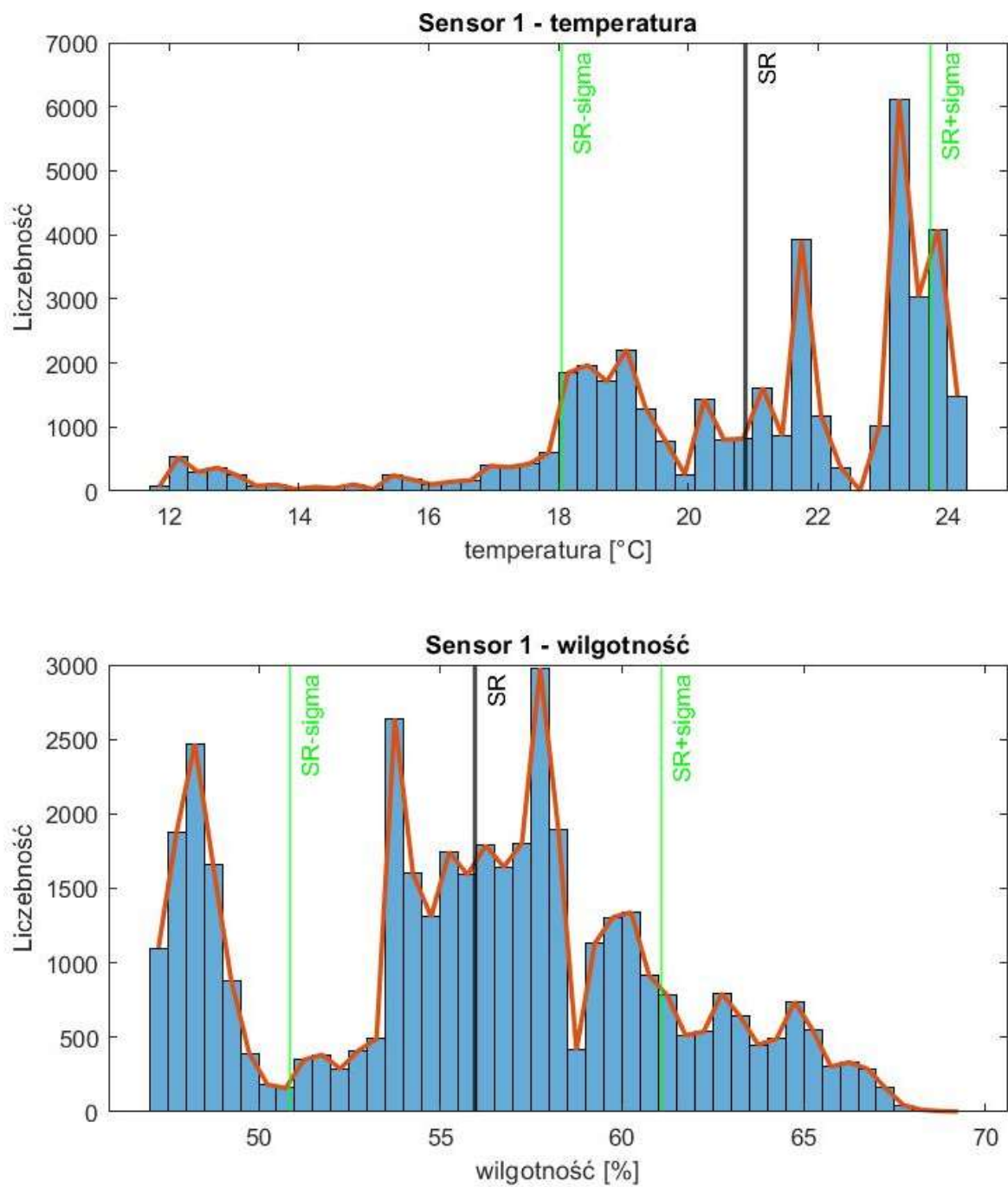
Średnia wilgotność: 55.98 %

Wilgotność zawiera się w przedziale: 47.04% do 69.05%

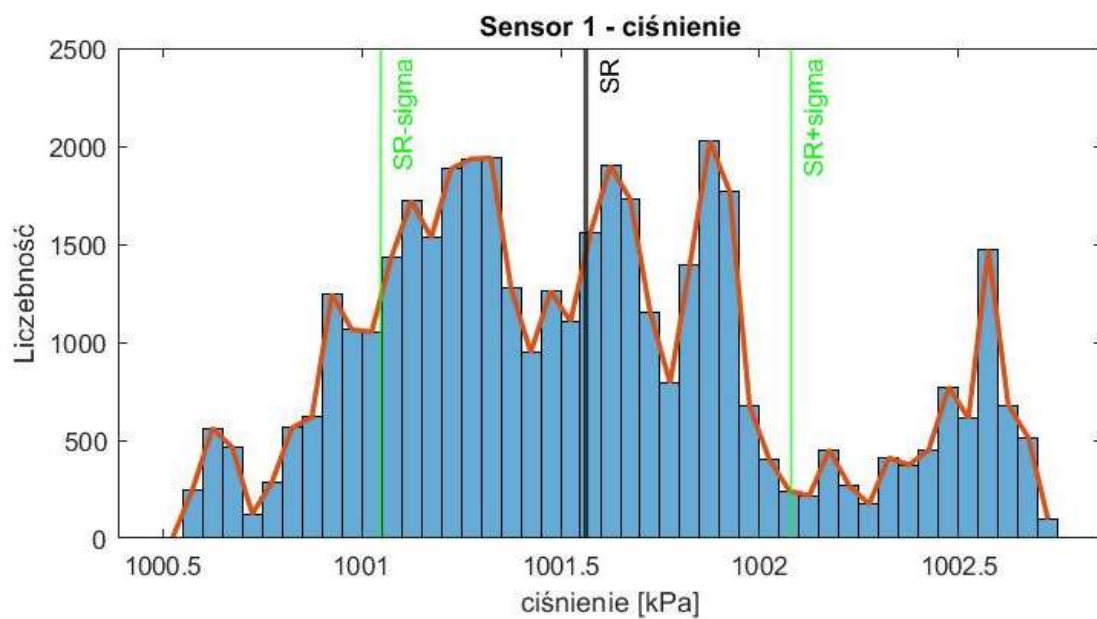
Odchylenie standardowe:

Temperatura: 2.84°C Ciśnienie: 0.52 kPa Wilgotność: 5.11%

Rys. 6.3b. Analiza danych pomiarowych – sensor 1



Rys. 6.3c. Analiza danych pomiarowych – sensor 1

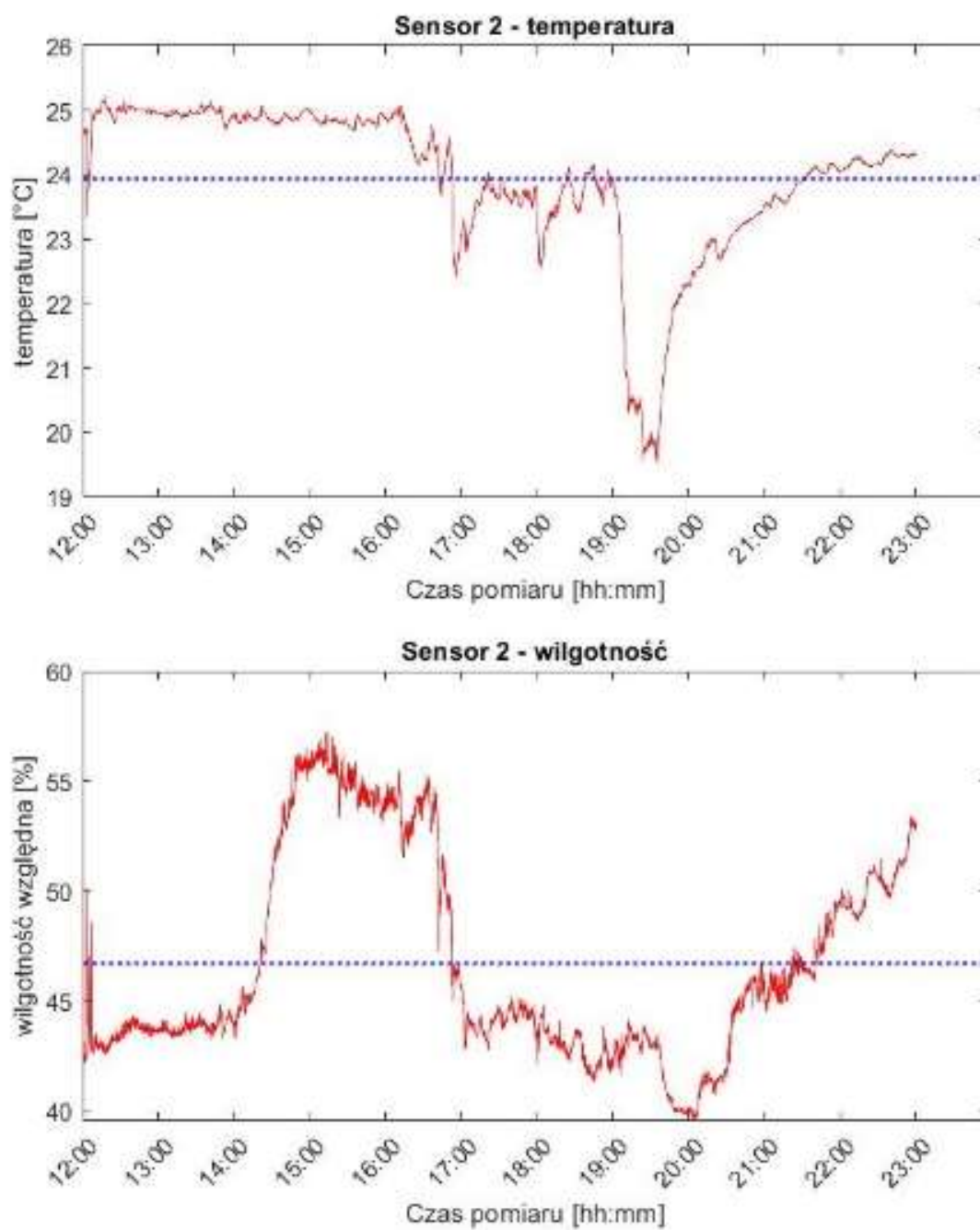


Temperatura: $(20.89 \pm 2.84)^{\circ}\text{C}$

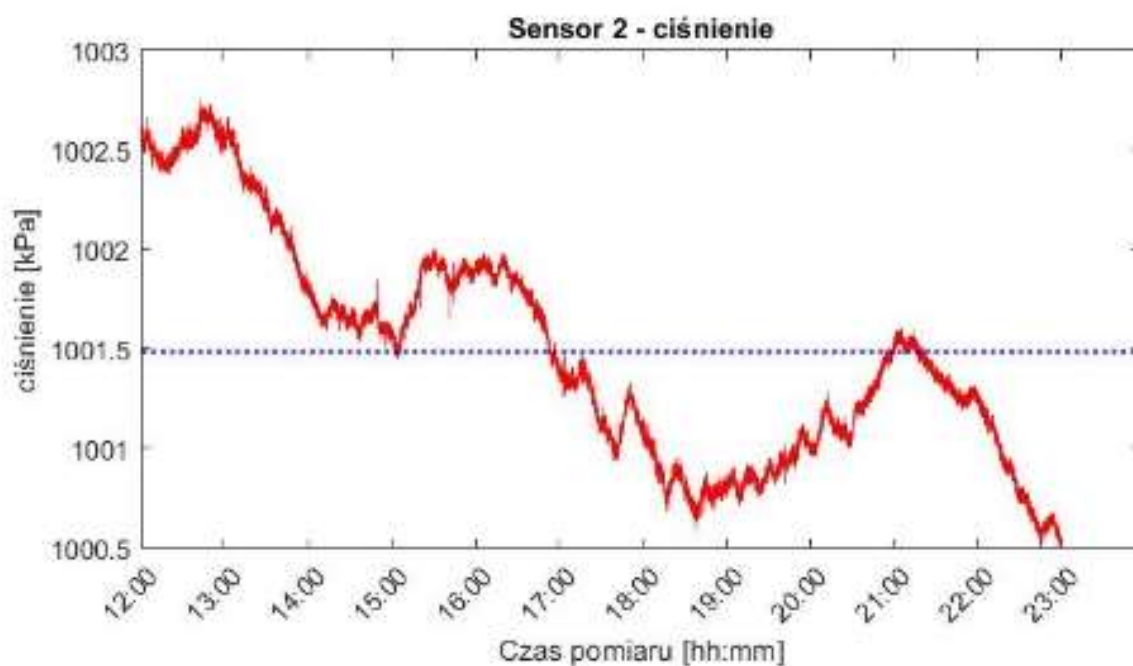
Ciśnienie: $(1001.56 \pm 0.52) \text{ kPa}$

Wilgotność: $(55.98 \pm 5.11) \%$

Rys. 6.3d. Analiza danych pomiarowych – sensor 1



Rys. 6.4a. Analiza danych pomiarowych – sensor 2



Średnia temperatura: 23.94°C

Temperatura zawiera się w przedziale: 19.52°C do 25.22°C

Średnie ciśnienie: 1001.49 kPa

Ciśnienie zawiera się w przedziale: 1000.50 kPa do 1002.75 kPa

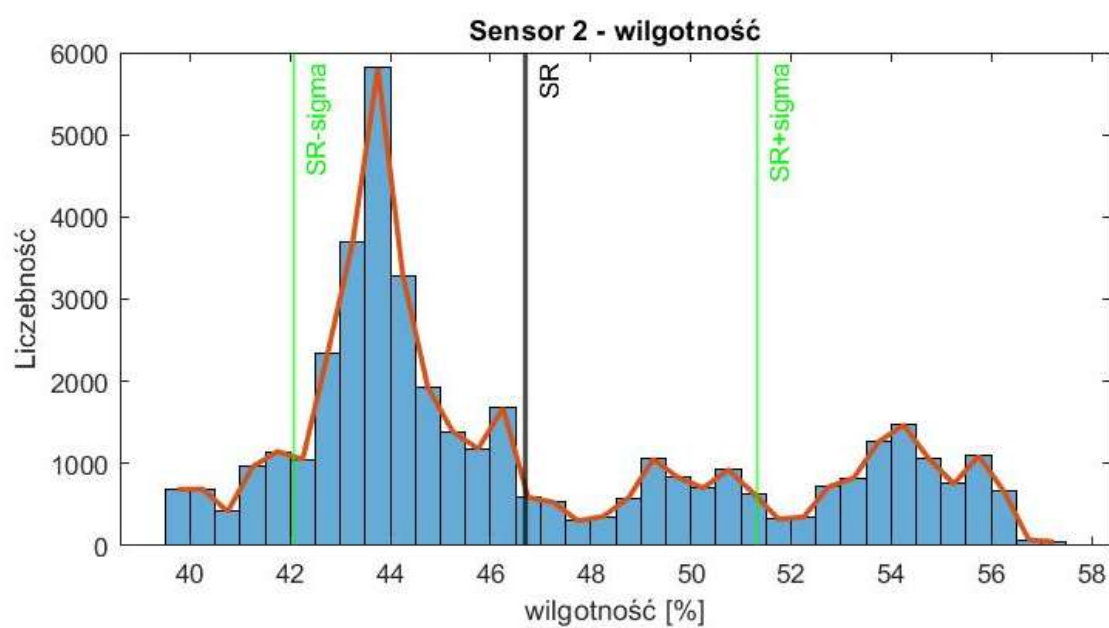
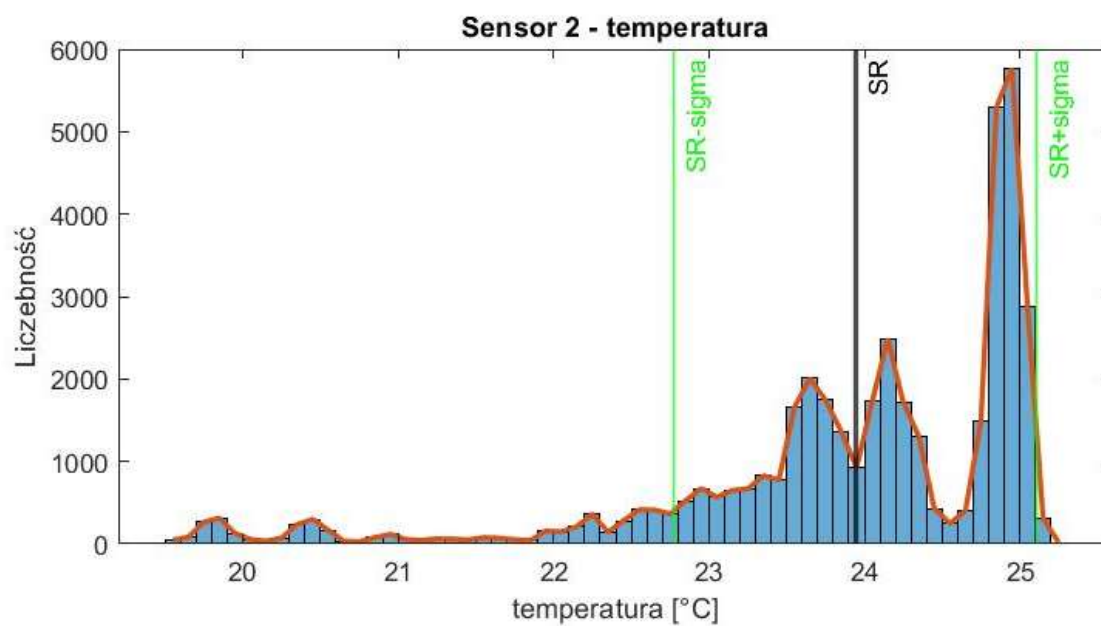
Średnia wilgotność: 46.70 %

Wilgotność zawiera się w przedziale: 39.55% do 57.25%

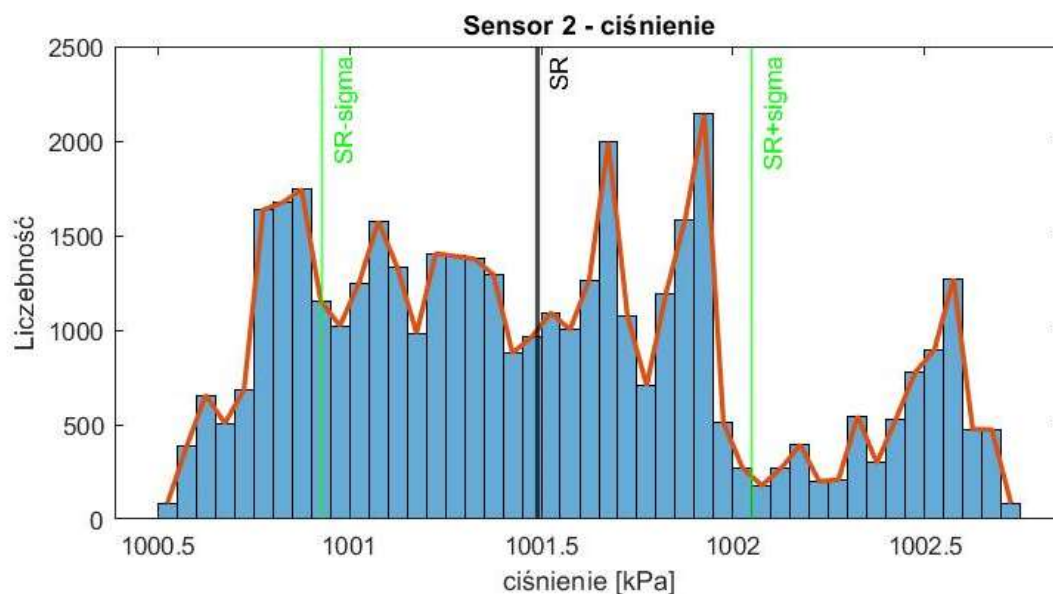
Odchylenie standardowe:

Temperatura: 1.17°C Ciśnienie: 0.56 kPa Wilgotność: 4.63%

Rys. 6.4b. Analiza danych pomiarowych – sensor 2



Rys. 6.4c. Analiza danych pomiarowych – sensor 2



Temperatura: $(23.94 \pm 1.17)^{\circ}\text{C}$

Ciśnienie: $(1001.49 \pm 0.56) \text{ kPa}$

Wilgotność: $(46.70 \pm 4.63) \%$

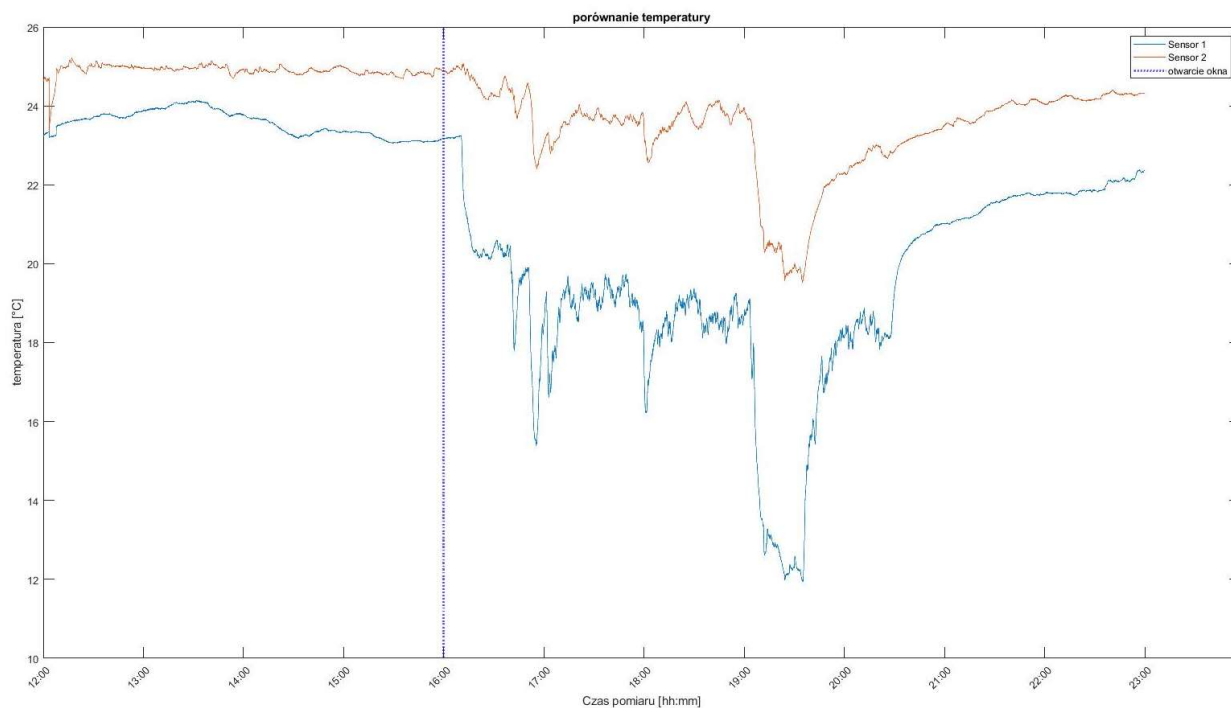
Rys. 6.4d. Analiza danych pomiarowych – sensor 2

System działał poprawnie bez przerw przez 11 godzin. Dłuższy czas pracy nie był testowany. Uzyskano pomiary temperatury, ciśnienia i wilgotności z obu sensorów. W poniższej tabeli zawarto wybrane parametry statystyczne z uzyskanych pomiarów:

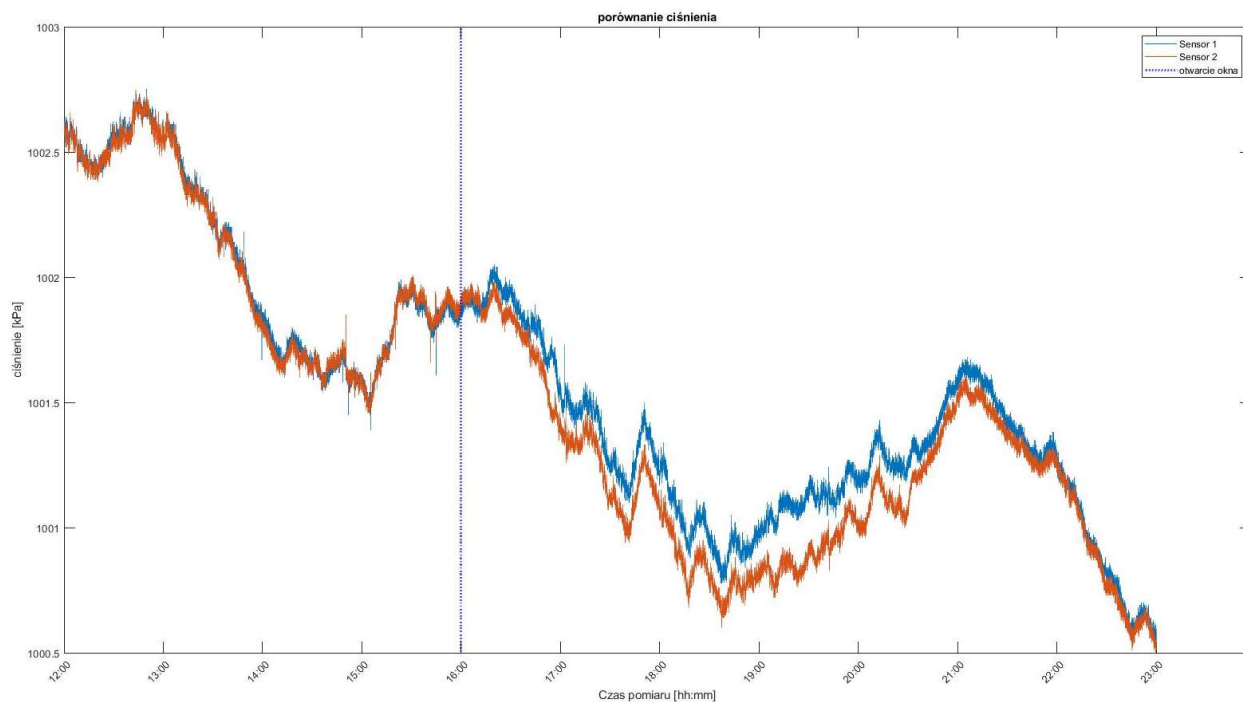
Tabela 6.1. Parametry statystyczne pomiarów

	Średnia temperatura [°C]	Średnie ciśnienie [kPa]	Średnia wilgotność [%]	Odchylenie standardowe temperatury [°C]	Odchylenie standardowe ciśnienia [kPa]	Odchylenie standardowe wilgotności [%]
Sensor 1 (na parapecie)	20,89	1001,56	55,98	2,84	0,52	5,11
Sensor 2 (na środku pomieszczenia)	23,94	1001.49	46,70	1,17	0,56	4,63

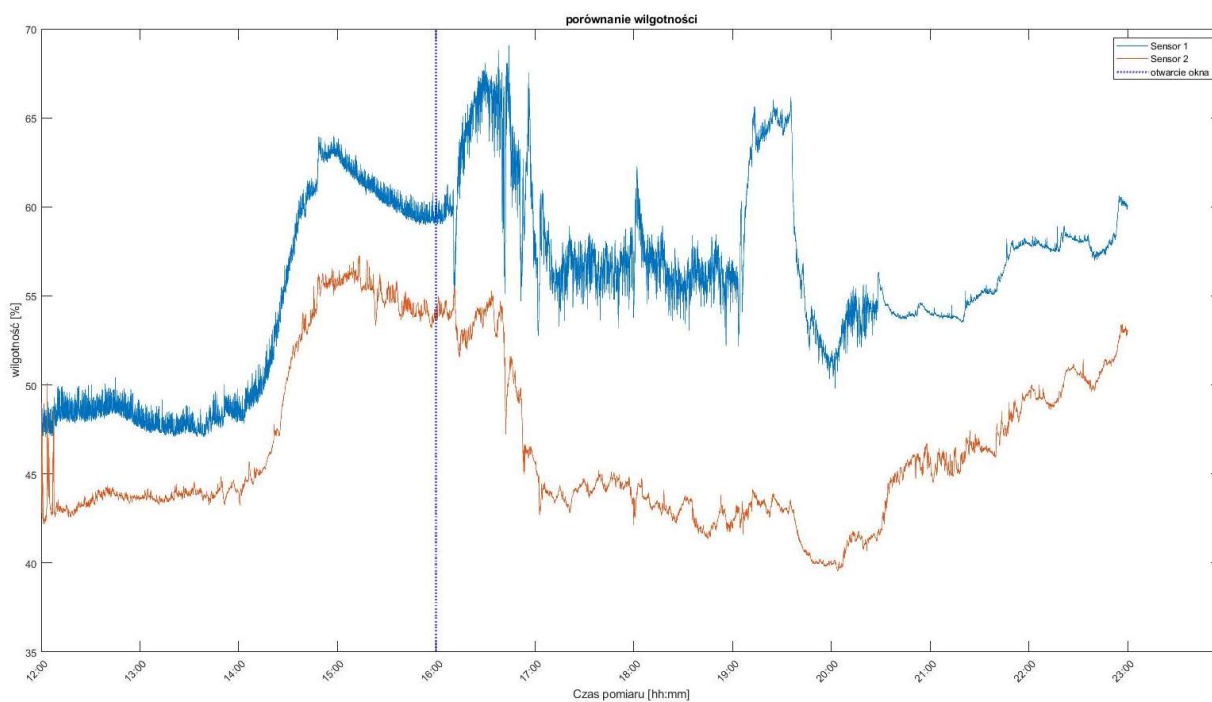
Dodatkowo program *data_analyser* wyrysował porównanie uzyskanych przez oba sensory pomiarów dla każdego z mierzonych parametrów (rys. 6.5. do do 6.7.):



Rys. 6.5. Porównanie wskazań temperatury



Rys. 6.6. Porównanie wskazań ciśnienia



Rys. 6.7. Porównanie wskazań wilgotności

Około godziny 16:00 w pomieszczeniu, w którym znajdowały się sensory, zostało otwarte okno. Urządzenie prawidłowo to odnotowało rejestrując spadek temperatury (wywołany napłynięciem chłodnego powietrza z zewnątrz) – warto też zauważyć, że wskazania temperatury obu sensorów mimo różnicy poziomów zachowują bardzo podobny charakter zmian. Otwarcie okna rozpoznać można też na wykresie przebiegu ciśnienia – wyraźny spadek nastąpił również około godziny 16:00.

Ciekawsza sytuacja wystąpiła na wykresie porównującym pomiary wilgotności. O ile do chwili otwarcia okna wskazania wilgotności wykazywały podobną charakterystykę zmian, to po otwarciu okna sensor umieszczony na parapecie okiennym wskazał nagły skok wilgotności (wywołany napłynięciem chłodnego i wilgotnego powietrza z zewnątrz, powodując jednocześnie ruch powietrza w pomieszczeniu). Poziom wilgotności rejestrowany przez czujnik umieszczony przy oknie wskazywał następnie dużą zmienność, która była wywołana tym, że jego wskazania podlegały ruchom powietrza z zewnątrz. Natomiast po otwarciu okna powietrze wewnątrz pomieszczenia zmniejszyło poziom wilgotności – spowodowane jest to tym, że wilgotne powietrze zaczęło uchodzić z pomieszczenia. Po zamknięciu okna poziom wilgotności zaczął się zwiększać, co można zaobserwować na wskazaniach obu sensorów.

W trakcie analizy zostały również przygotowane histogramy mierzonych wartości (rys. 6.1b. oraz 6.2b.). Rozkład wartości na uzyskanych histogramach nie przedstawia żadnego „klasycznego” rozkładu – nie widać tu na przykład rozkładu normalnego, ani jednostajnego. Można jednak zauważyć to, że na histogramach temperatury większość pomiarów leży powyżej wartości średniej (co oznacza, że typowa temperatura oscylowała wobec górnego krańca zmierzonego spektrum). W histogramach ciśnienia widać, że wartości skupione są zasadniczo wokół średniej – poza małym obszarem (1002,0 – 1002,5 kPa), w którym jest widocznie mniej wystąpień niż w innych obszarach. Histogram wilgotności sensora numer 2 (leżącego na środku pomieszczenia) wskazuje większość pomiarów w okolicach 43%, natomiast dla sensora numer 1 nie da się wskazać jasno jednego maksimum.

7. Podsumowanie

Na potrzeby niniejszej pracy zbudowano bezprzewodowy rozproszony system pomiaru warunków środowiskowych. Przedstawione w poprzednich rozdziałach urządzenie spełnia wszystkie zadane kryteria: bezprzewodowość (dzięki użyciu komunikacji Bluetooth Low Energy (BLE) centralka otrzymuje dane z sensorów), rozproszenie elementów (sensory można ustawić w dowolnym miejscu niezależnie od siebie) i w końcu pomiar ciśnienia, temperatury oraz wilgotności (dzięki użytym czujnikom BME280).

Do komunikacji sensorów z centralką użyto protokołu Bluetooth Low Energy. Użycie tego protokołu możliwe było dzięki modułom ESP32 DevKit ESP-WROOM-32 V2, które w stosunkowo łatwy sposób pozwalają nawiązać komunikację z użyciem BLE. Do pomiarów warunków środowiskowych użyto czujników BME280 zapewniających szybkie i dokładne pomiary. Gotowe urządzenie zmontowano na płytkach prototypowych i zasilono zasilaczami sieciowymi 12V (sensory) oraz 5V (centralka).

W czasie przeprowadzonych badań trwających 11 godzin uzyskano 82818 linii pomiarów (po 41409 na sensor), co oznacza 2,29 MB (dokładnie 2 401 753 bajtów) zarejestrowanych danych. Dane zanalizowano i zobrazowano za pomocą autorskiego programu *data_analyser*. Omówiono uzyskane wyniki i zaprezentowano podstawowe dane statystyczne.

Przeprowadzone badania potwierdzają prawidłowe działanie zbudowanego urządzenia. Wykorzystane moduły (ESP32 DevKit ESP-WROOM-32 V2 oraz BME280) sprawdziły się i mogą być stosowane do podobnych zadań. Protokół Bluetooth Low Energy wykazał swoją przydatność w zestawieniu komunikacji pomiędzy dwoma elementami systemu – prawidłowo przysyłał pakiety danych. Zbudowanie opisywanego urządzenia wykazało zasadność użycia omówionych modułów i elementów do budowy niewielkiej sieci czujników.

Mimo, że zbudowane urządzenie spełnia swoją funkcję, są możliwości dalszego usprawnienia lub jego rozbudowania, np.:

- zmniejszenie zużycia energii – korzystając z bardziej zaawansowanych funkcji protokołu BLE można spróbować zaimplementować tryb uśpienia (sleep mode) w taki sposób aby sensory uaktywniały moduł nadawczy tylko i wyłącznie na krótką chwilę niezbędną na wysłanie danych,

- automatyzacja liczby czujników – w obecnej wersji moduł centralny rozpoznaje moduły sensorów po adresach MAC i łączy się tylko z urządzeniami na stałe wpisanymi w kod programu,
- wprowadzenie dwustronnej komunikacji sensorów z centralką – moduły sensora mogłyby reagować na rozkazy nadesłane z modułu centralnego,
- zaprojektowanie dedykowanej płytki PCB – użyte w urządzeniu płytki prototypowe pozwalają zmontować podzespoły w sposób trwały, jednak dedykowane płytki PCB pozwoliłyby zwiększyć estetykę urządzenia przy jednoczesnym zmniejszeniu jego rozmiarów,
- zawarcie podstawowych funkcji analizy danych w samej centralce – jeżeli wzbogacić moduł centralny o obsługę wyświetlacza graficznego (na przykład OLED), to sama centralka mogłaby rysować przebieg wybranego parametru oraz podawać parametry statystyczne takie jak odchylenie standardowe.

Powyższe punkty to oczywiście tylko przykłady pomysłów na rozwój zbudowanego urządzenia – dostępne w sprzedaży moduły oraz dokumentacja i przewodniki dostępne w sieci Internet pozwalają na nieograniczony rozwój systemu. Przygotowanie urządzenia było źródłem cennej wiedzy zarówno programistycznej jak i dotyczącej szeroko pojmowanej elektroniki.

Bibliografia

1. A. Chwaleba, M. Poniński, A. Siedlecki, Metrologia elektryczna, Wydawnictwa Naukowo-Techniczne, 2003 Warszawa
2. Pomiary wilgotności – instrukcja laboratoryjna, Politechnika Wrocławska, Wydział Elektryczny, Katedra Maszyn, Napędów i Pomiarów Elektrycznych (http://kmmnpe.pwr.edu.pl/files/prv/id35/zp-lab/pase/pomiary_przem/cw10_air_przemysl_wilgotnosc.pdf)
3. Pomiar Wilgotności Powietrza – instrukcja laboratoryjna, Politechnika Szczecińska, Wydział Mechaniczny, Katedra Techniki Ciepłej (http://kmc.zut.edu.pl/fileadmin/dydaktyka/instrukcje/cwiczenie_9.pdf)
4. Czujniki ciśnienia i pomiary ciśnienia – instrukcja laboratoryjna, Politechnika Lubelska, Katedra Automatyki i Metrologii (<http://elektron.pol.lublin.pl/dljl24/pwn/cw5.pdf>)
5. Pomiary temperatury – instrukcja laboratoryjna, Politechnika Wrocławska, Laboratorium Metrologii (http://www.w12.pwr.wroc.pl/metrologia/instrukcje/Met_2014_8.pdf)
6. J. Kurose, K. Ross, Sieci komputerowe, Helion 2010
7. Harris Corporation 2005, Radio Communications in the Digital Age, Volume One: HF Technology
8. W. Kabaciński, M. Żal, Sieci telekomunikacyjne, WKŁ 2008
9. A. Mielczarek, Szeregowe interfejsy cyfrowe, Helion 1993
10. Infrared Data Association, Serial Infrared Physical Layer Specification, 2001
11. A technical overview of LoRa and LoRaWAN, LoRa Alliance, 2015
12. Zigbee Specification, ZigBee Alliance, 2015
13. P. Mol, Sieci CAN, Elektronika Praktyczna 7/2005, str. 84-88
14. J. Stępień, J. Kołodziej, W. Machowski, Niskoenergetyczne bezprzewodowe personalne sieci sensorowe, Przegląd Elektrotechniczny 2017-2
15. C. Gomez, J. Oller, J. Paradells, Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology, Sensors 2012
16. N. Kolban, Kolban's book on ESP32, <https://leanpub.com/kolban-ESP32>
17. MATLAB Documentation (<https://www.mathworks.com/help/matlab/>)