



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, automatyki, informatyki i inżynierii biomedycznej.

Praca dyplomowa inżynierska

*Monitoring pulsu i temperatury ciała z bezprzewodowym
powiadamianiem o anomaliiach*

*Heart rate and body temperature monitoring with wireless
notification on anomalies*

Autor:

Kierunek studiów:

Opiekun pracy:

Mikołaj Skrzyniarz

Elektrotechnika

dr inż. Grzegorz Hayduk

Kraków, 2021

Spis treści

1. Wstęp	3
1.1. Cel pracy.....	4
1.2. Zakres pracy	4
2. Elementy składowe układu.....	5
2.1. Płytką rozwojową NodeMCU V3.....	5
2.2. Czujnik MAX30102	7
2.3. Wyświetlacz OLED	10
3. Komunikacja przewodowa i bezprzewodowa.....	11
3.1. Magistrala I2C	11
3.2. Interfejs SPI.....	12
3.3. Protokół transmisji danych MQTT	13
3.3.1. Uniwersalne formaty wymiany danych	13
4. Projekt i implementacja oprogramowania	15
4.1. Język programowania Python.....	15
4.2. Środowisko programistyczne PyCharm	16
5. Opis realizacji tematu pracy	17
5.1. Projekt układu elektronicznego	17
5.2. Obsługa elementów składowych	21
5.3. Interpretacja odczytanych danych	24
5.4. Algorytm pracy oprogramowania zaimplementowanego na mikrokontrolerze	29
5.5. Projekt aplikacji zbierania danych.....	31
5.6. Komunikacja aplikacji nadzędnej z mikrokontrolerem	32
5.7. Testy wykonanego systemu.....	33
6. Podsumowanie	35
7. Bibliografia	36

1. Wstęp

Rozwój technologii mający miejsce na przestrzeni ostatnich trzydziestu lat sprawił, że możliwość zdalnej kontroli nad urządzeniem lub jego parametrami stała się czymś powszechnym. Jest ogromnym ułatwieniem dla wielu specjalności, gdzie wymagany jest stały nadzór nad pewnymi wielkościami lub procesami technologicznymi. Przykładem branży dla której możliwość bezprzewodowego podglądu była czymś przełomowym jest branża medyczna. Monitoring podstawowych parametrów życiowych stanowi nieodzowny element badań stanu pacjenta oraz jest podstawą do wykrywania roźnego rodzaju schorzeń. Przykładem takich funkcji życiowych są puls, saturacja tlenem krwi oraz temperatura ciała.

Puls w większości sytuacji rozumiany jest jako częstotliwość bicia serca. Bardziej formalna definicja mówi, że jest to wynikający ze skurczy lewej komory serca falisty ruch naczyń tętnicznych, będący w dużym stopniu zależny od ich elastyczności, a częstotliwość jest tylko jedną z jego cech charakterystycznych. Określanie pulsu jako ilości uderzeń serca na minutę nie jest jednak uznawane za błąd. Pomiar tej wielkości jest niezbędny do oceny stanu układu sercowo - naczyniowego. Badaniu takiemu bardzo często towarzyszy wyznaczenie poziomu saturacji tlenem krwi, czyli ocena nasycenia krwi tętniczej tlenem. Wartość ta pozwala wykryć czasowe niedotlenienie lub choroby układu oddechowego.

Sprawdzenie poziomu zarówno pulsu jak i saturacji krwi tlenem odbywa się przy pomocy pulsoksymetru. Działa on w oparciu o dwie zamontowane diody, emitujące różne rodzaje światła - czerwone (ang. *red*), będące wchłanianym przez znajdującą się w krwi hemoglobinę nietlenowaną oraz podczerwone (ang. *infrared, IR*), wchłaniane przez hemoglobinę utlenowaną. Dodatkowo w skład takiego urządzenia wchodzi fotodetektor pozwalający określić ilość danego światła pochloniętego przez ciało. Puls wyznaczany jest na podstawie czasu między szczytami krzywej pletryzmograficznej, będącej przebiegiem zmiany poziomu absorpcji światła jednej z diod. Saturację określa się jako względny poziom absorpcji obu światel [9].

Pulsoksymetry są powszechnie stosowane w szpitalach lub przychodniach. Mają zazwyczaj formę nakładki na palec u dłoni, ze względu na wysoką skuteczność pomiaru właśnie w tym miejscu. Aktualnie coraz więcej urządzeń codziennego użytku umożliwia dokonanie takiego pomiaru. Większość nowoczesnych smartfonów, zegarków oraz opasek sportowych posiada funkcję odpowiedzialną za określenie poziomu podstawowych parametrów życiowych, wraz informacją o ich prawidłowości oraz rejestracji - lokalnie lub w chmurze.

1.1. Cel pracy

Celem niniejszej pracy było opracowanie miniaturowego urządzenia na bazie mikrokontrolera ESP8266 odpowiedzialnego za monitoring pulsu, saturacji krwi tlenem, temperatury. Dodatkowo miało się ono komunikować z aplikacją zewnętrzną poprzez sieć WiFi. Podstawowymi zadaniami oprogramowania zaimplementowanego na mikrokontrolerze są pomiar wymienionych wyżej parametrów, wykrywanie ewentualnych odchyłek od norm, wyświetlanie bieżących odczytów wraz z znacznikiem czasowym lokalnie na wyświetlaczu OLED oraz periodyczne wysyłanie ich do aplikacji zewnętrznej. Pomiary powinny być możliwe zarówno z poziomu powierzchni palca jak i w sytuacji, gdy czujnik jest zamontowany na nadgarstku, symulując w ten sposób działanie opaski sportowej. Stworzona aplikacja zewnętrzna powinna być w stanie odebrać dane od wielu takich urządzeń, wyświetlać je oraz zapisywać do odpowiedniego pliku.

1.2. Zakres pracy

Praca jest podzielona na kilka różnych tematycznie rozdziałów. Rozdział pierwszy przedstawia charakterystykę użytych elementów wchodzących w skład układu pomiarowego. Opisane są ich parametry, specyfikacja oraz krótki opis wraz z zasadą działania. Kolejny rozdział zawiera wstęp teoretyczny na temat sposobów komunikacji użytych w pracy. W czwartym rozdziale opisano użyte podczas projektu oprogramowania narzędzia, takie jak język programowania oraz środowisko programistyczne. W przedostatnim rozdziale pracy znajduje się szczegółowy opis każdego z etapów części realizacyjnej w kolejności chronologicznej. Opisano także w krótki sposób etapy pośrednie oraz uzasadniono ich porzucenie. W skład rozdziału szóstego wchodzą podsumowanie, opis zrealizowanych celów oraz propozycja dalszego rozwoju projektu.

2. Elementy składowe układu

Drugi rozdział pracy ma na celu zaznajomienie czytelnika z elementami wchodząymi w skład układu pomiarowego. Zaprezentowane zostaną ich opis, specyfikacja oraz uzasadnienie wyboru podczas doboru elementów.

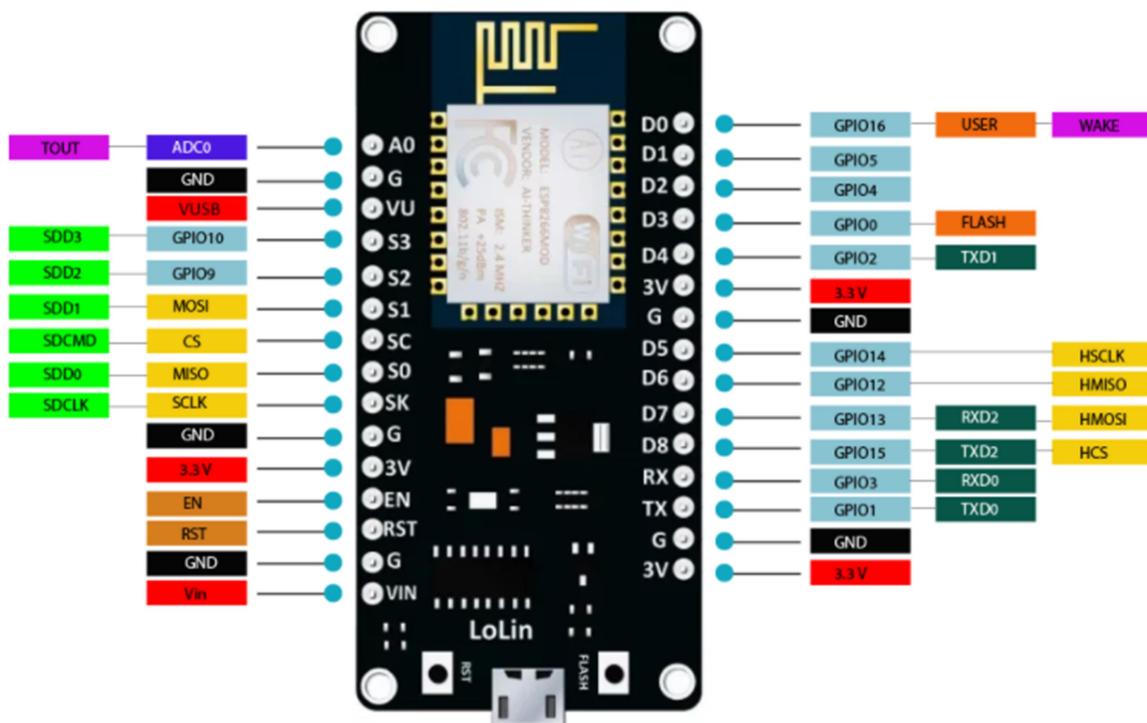
2.1. Płytki rozwojowe NodeMCU V3

NodeMCU v3 jest to niskobudżetowa płytka deweloperska na której wbudowany jest moduł WiFi ESP-12E skonstruowany w oparciu o układ ESP8266EX autorstwa chińskiego producenta Espressif Systems. Popularność zyskał jednak dopiero dzięki firmie Ai-Thinker, która zaczęła produkować różnego rodzaju moduły w bazie ESP. Układ ten znajduje szerokie zastosowanie w projektach IoT (ang. *Internet of things*) w związku z możliwością programowania go jak niezależny mikrokontroler, który jest w stanie łączyć się z siecią WiFi. Możliwe jest także dołączenie takiego modułu do skonstruowanego już urządzenia, w celu rozszerzenia jego możliwości o funkcje sieciowe.

Płytki NodeMCU jest w pełni gotowa do obsługi mikrokontrolera, z racji na zamontowane już elementy takie jak złącze microUSB, konwerter USB-UART oraz rezystory podciągające odpowiedzialne za prawidłowe zasilenie modułu. W efekcie jedyną czynnością wymaganą do zrealizowania przed przystąpieniem do programowania układu w tej formie jest wgranie odpowiedniego oprogramowania sprzętowego, zależnego od języka programowania. Specyfikacja oraz parametry wyglądają w sposób następujący:

- 32 bitowy procesor Tensilica L106 o taktowaniu 80 MHz, z możliwością zwiększenia do 160 Mhz.
- 16 wyprowadzeń generalnego przeznaczenia, z czego dla użytkownika dostępnych jest 11. Trzeba zaznaczyć, że złącze numer 16 może działać wyłącznie w trybie odczytu lub zapisu natomiast te o indeksie 6-11 są w większości modułów opartych na ESP8266 domyślnie zarezerwowane dla pamięci typu flash urządzenia. Ich użycie może skutkować zawieszeniem się programu [10-11]. Ponadto nie są one zazwyczaj zaznaczane na schematach. Grafika przedstawiająca rzut płytki wraz z widocznymi oznaczeniami złącz znajduje się na rys. 2.1.
- Obsługa UART, SPI, I2C, I2S, IRDA, SDIO 2.0, PWM oraz 1-Wire.
- Zintegrowany 10 - bitowy przetwornik analogowo cyfrowy.
- Wbudowany konwerter USB-UART CH340 wraz z złączem microUSB.
- 4 MB pamięci flash.
- 64 KB pamięci SRAM.

- Wbudowana antena PCB.
- Parametry komunikacji WiFi:
 - wsparcie dla standardów 802.11 b/g/n;
 - maksymalna prędkość transmisji 72,2 Mb/s;
 - protokoły sieciowe IPv4, TCP/UDP/HTTP/FTP;
 - możliwość pracy zarówno jako klient lub stacja.
- Możliwość obsługi elementów elektronicznych wymagających zasilania o napięciu 3,3 V oraz 5 V. Odpowiedzialne są za to końcówki o oznaczeniach odpowiednio 3V oraz VU (patrz rys. 2.1).
- Napięcie zasilania równe 5 V przy podłączeniu przez port microUSB. W przypadku zastosowania zasilania zewnętrznego zalecane jest podanie napięcia z zakresu 7 - 12 V [12]. Sam układ ESP8266 do poprawnego działania wymaga 2,5 - 3,6 V, które jest dostarczane dzięki wbudowanemu regulatorowi napięcia.
- Rozstaw sąsiednich końcówek wynosi 2,54 mm. Odległość między liniami wyprowadzeń leżącymi po przeciwnych stronach modułu jest równa 28mm. Takie ułożenie złącz sprawia, że prototypy urządzeń wykonane w oparciu o NodeMCU mogą być łączone na większości płyt stykowych dostępnych na rynku.
- Wymiary: 58x30 mm.
- Dopuszczalna temperatura pracy od -40 °C do 125 °C [3].
- Możliwość programowania w językach C, Arduino, MicroPython, LUA lub Mongoose OS.



Rys. 2.1. Oznaczenia złącz użytych w pracy NodeMCU V3 [13].

Obecnie układy z rodziny ESP pozostają bezkonkurencyjne na rynku mikrokontrolerów obsługujących komunikację WiFi. Poza modułami bazującymi na ESP8266 produkowane są także takie oparte o nowszy model - ESP32. Układ ten oferuje użytkownikowi mocniejsze parametry niż jego poprzednik oraz kilka dodatkowych funkcji, na przykład komunikację poprzez bluetooth. Moduły na bazie starszej wersji ESP nadal są jednak stosowane w tych projektach, które nie wymagają dużej mocy obliczeniowej lub podłączenia więcej niż trzech urządzeń peryferyjnych.

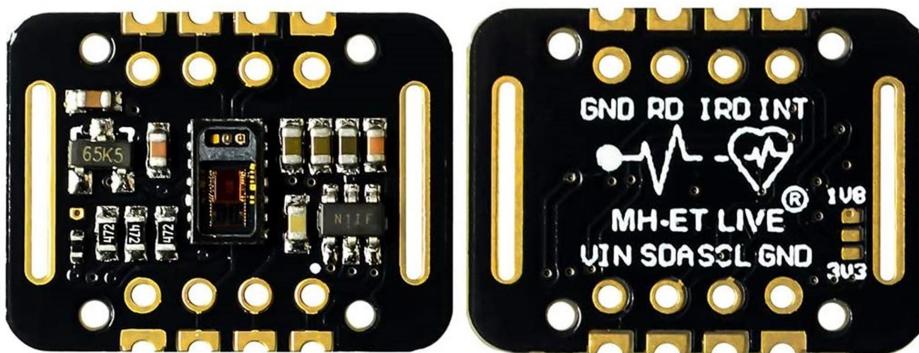
2.2. Czujnik MAX30102

MAX30102 to czujnik dzięki któremu możliwe jest wyznaczenie poziomu pulsu, saturacji tlenu we krwi oraz temperatury matrycy. Producentem jest firma Maxim Integrated Products. Składa się on z dwóch światel LED, fotodetektorów, elementów optycznych oraz układów elektronicznych niwelujących wpływ światła zewnętrznego na wartość dokonywanego pomiaru. Producent nie udostępnia szczegółowych schematów zastosowanych układów elektronicznych. MAX30102 znajduje zastosowanie w smartfonach, tabletach, opaskach sportowych lub zegarkach. Specyfikacja oraz parametry wykorzystanego modułu skonstruowanego na bazie podanego czujnika wyglądają w sposób następujący:

- Wbudowane diody LED będące źródłem światła czerwonego i podczerwonego. Długości emitowanych fal znajdują się w zakresie 650-670 nm oraz 870-900nm. Wartości te są zależne od temperatury pracy.
- Możliwość odczytu temperatury matrycy z rozdzielcością równą 0,0625 °C. Margines błędu wynosi 1 °C.
- Komunikacja za pomocą magistrali I2C.
- Zintegrowany 18-bitowy przetwornik analogowo-cyfrowy.
- Wymagane napięcie zasilania z zakresu 3,3 - 5 V. MAX30102 działa w oparciu o dwie wartości napięcia. Zasilanie diod wymaga użycia napięcia z zakresu 3,1 - 5,0 V. Reszta układu do poprawnego działania potrzebuje zasilania o wartości 1,7 - 2,0 V. Operacje te są możliwe dzięki wbudowanym regulatorom napięcia.
- Niski pobór prądu w stanie czuwania równy około 0,7 µA.
- Pobór mocy nieprzekraczający 1 mW.
- Dopuszczalna temperatura pracy od -40 °C do 85 °C.
- Rozstaw sąsiednich końcówek wynosi 2,54 mm. Odległość między liniami wyprowadzeń leżącymi po przeciwnych stronach modułu jest równa 11,4 mm.
- Wymiary: 20,6x15,5 mm.
- Czujnik pokryty warstwą szkła ochronnego.
- Otwory ułatwiające montaż przy pomocy paska.

Wykorzystany w pracy model płytki z czujnikiem oprócz złącz odpowiedzialnych za zasilanie oraz komunikacje przy użyciu I2C (patrz rys. 2.2), posiada także trzy dodatkowe wyprowadzenia:

- INT - informuje o wystąpieniu przerwania poprzez podanie stanu niskiego.
- RD, IRD - umożliwiają regulację natężenia prądu płynącego przez diody poprzez podłączanie fizycznych elementów elektronicznych. Oznaczenia odpowiadają kolejno diodom emitującym światło czerwone i podczerwone.



Rys. 2.2. Użyty w pracy moduł z czujnikiem MAX30102 [14].

Odczytywane dane zapisywane są lokalnie w buforze mogącym przechować maksymalnie 32 próbki. Struktura tego obszaru pamięci oparta jest o schemat kolejki FIFO (ang. *First-In-First-Out*). Oznacza to, że najstarsza z próbek zostanie wysłana jako pierwsza podczas odczytu przez mikrokontroler. Rozmiar jednej próbki wynosi 3 bajty dla każdej z pracujących diod. Obsługa MAX30102 ogranicza się do operacji na rejestrach. Adresy wraz opisem przedstawia tab. 2.1. Konfiguracja polega na wpisaniu odpowiedniej wartości do danego rejestru. Podgląd pomiarów odbywa się poprzez odczyt rejestru w którym przechowywane są dane z FIFO. Warto zaznaczyć, że sam czujnik nie zwraca dokładnej wartości pulsu ani nasycenia krwi tlenem, a jedynie sygnał z fotodetektora po konwersji analogowo - cyfrowej. W celu wyznaczenia poziomu wymienionych parametrów życiowych konieczna jest analiza krzywej pletryzmograficznej, którą tworzą pomiary dla obu typów światel. Wartość temperatury może zostać odczytana bezpośrednio z odpowiednich rejestrów [4].

Tab. 2.1. Adresy rejestrów MAX30102 wraz z opisem.

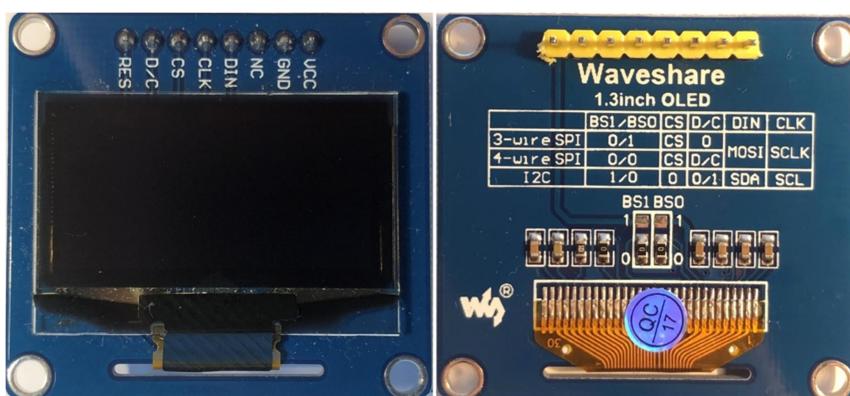
Nazwa	Adres	Opis
Status przerwań 1	0x00	Zawiera informacje o wystąpieniu przerwania w związku z: - przepełnieniem bufora FIFO, - nowymi danymi w buforze FIFO, - zaprzestaniem działania układów niwelujących wpływ światła zewnętrznego, - zasileniem modułu i gotowością do pracy.
Status przerwań 2	0x01	Zawiera informacje o wystąpieniu przerwania w związku z: - zakończeniem procesu przetwarzania A/C temperatury.

Nazwa	Adres	Opis
Wyłączenie przerwań 1	0x02	Umożliwia włączenie występowania wszystkich przerwań z statusu 1. Domyślnie wyłączone, z wyjątkiem przerwania informującego o gotowości do pracy.
Wyłączenie przerwań 2	0x03	Umożliwia włączenie występowania przerwania informującego o zakończeniu procesu przetwarzania A/C temperatury. Domyślnie wyłączone
Znacznik zapisu FIFO	0x04	Informuje o indeksie miejsca w buforze FIFO, gdzie zostanie zapisana następna próbka. Zwiększa swoją wartość automatycznie po każdym zapisie.
Licznik przepełnienia FIFO	0x05	Informuje o braku miejsca w buforze FIFO.
Znacznik odczytu FIFO	0x06	Informuje o indeksie miejsca w buforze FIFO, z którego zostanie odczytana następna próbka. Zwiększa swoją wartość automatycznie po każdym odczycie.
Bufor danych FIFO	0x07	Przechowuje zmierzone wartości.
Konfiguracja FIFO	0x08	Umożliwia zmianę ustawień FIFO takich jak: - liczba próbek, z których ma być obliczona średnia przed wpisaniem do buforu FIFO, - włączenie/wyłączenie zapisywania próbek od początku po osiągnięciu maksymalnej ilości, - liczba wolnych miejsc w buforze, która wywołuje przerwanie.
Konfiguracja trybu pracy	0x09	Umożliwia wybór trybu pracy, dostępne tryby pracy to: - <i>Heart rate</i> - aktywna dioda podczerwona, - <i>SpO2</i> - obie diody aktywne, - <i>Multi-LED</i> - obie diody aktywne.
Konfiguracja trybu <i>SpO2</i>	0x0A	Umożliwia konfiguracje parametrów takich jak: - zakres przetwornika A/C, - częstotliwość próbkowania, - szerokość impulsu powiązaną z rozdzielczością A/C.
Prąd diody podczerwonej	0x0C	Umożliwia dobór natężenia prądu diody w zakresie 0 - 51,0 mA.
Prąd diody czerwonej	0x0D	
Konfiguracja trybu <i>Multi-LED</i>	0x11 0x12	Umożliwia ustalenie kolejności kanałów.
Część całkowita temperatury	0x1F	Przechowuje część całkowitą zmierzonej temperatury.
Część ułamkowa temperatury	0x20	Przechowuje wartość pośrednią części ułamkowej zmierzonej temperatury.
Status temperatury	0x21	Umożliwia rozpoczęcie przetwarzania A/C temperatury.

2.3. Wyświetlacz OLED

Do lokalnego wyświetlania danych użyty został wyświetlacz OLED 1,3" chińskiego producenta WaveShare. Jest to najmniej unikalny element spośród zastosowanych. Móglby zostać zastąpiony przez dowolny, inny wyświetlacz OLED umożliwiający komunikację z wykorzystanym mikrokontrolerem. Wybrany model cechuje stosunkowo niska do innych tego typu urządzeń cena, wysoka dostępność oraz prostota obsługi. Specyfikacja oraz parametry wyglądają w sposób następujący:

- Skonstruowany w oparciu o sterownik SH1106.
- Domyślnie komunikacja odbywa się przy użyciu 4 - przewodowego SPI. Możliwość zmiany na 3 - przewodowe SPI lub I2C poprzez zlutowanie odpowiednich punktów na dolnej warstwie PCB modułu.
- Wymagane napięcie zasilania z zakresu 3,3 - 5V.
- Przekątna ekranu 1,3 cala.
- Rozdzielcość 128x64 pikseli.
- Wymiary: 40,5x37,5 mm.
- Maksymalny kąt widzenia około 160 °.
- Dopuszczalna temperatura pracy od -20 °C do 70 °C.



Rys. 2.3. Użyty w pracy wyświetlacz OLED. Widoczne punkty lutownicze na dolnej warstwie BS1 i BS0.

Poza złączami zasilającymi producent użył końcówek (patrz rys. 2.3) o funkcji zależnej od wybranego sposobu transmisji. Ich opis przedstawia tab. 2.2. Wyprowadzenie NC powinno zostać nieobsłużone. Stan niski złącza RES powoduje restart wyświetlacza [5].

Tab. 2.2. Funkcje złącz wyświetlacza w zależność od sposobu komunikacji.

Złącze	4 - przewodowe SPI	3 - przewodowe SPI	I2C
D/C	D/C	-	-
CS	CS	CS	-
CLK	SCK	SCK	SCL
DIN	MOSI	MOSI	SDA

3. Komunikacja przewodowa i bezprzewodowa

Trzeci rozdział pracy przedstawia podstawowe informacje na temat użytych w pracy standardów komunikacji. Ponadto opisane zostaną dwa powszechnie używane formaty wymiany danych.

3.1. Magistrala I2C

Jednym z najczęściej używanych sposobów komunikacji podczas łączenia układów elektronicznych jest magistrala I2C (ang. *Inter-Integrated Circuit*) autorstwa firmy Philips. Jest to szeregowy, dwukierunkowy sposób wymiany danych. Należy także do typów synchronicznych. Oznacza to, że dane są wysyłane w wyznaczonych momentach, które determinuje sygnał zegarowy na jednej z linii. Ten standard transmisji przewiduje istnienie:

- linii odpowiedzialnej za przesyłanie danych między układami oznaczanej jako SDA (ang. *Serial Data Line*);
- linii na której występują impulsy zegarowe synchronizujące transmisje na linii danych, oznaczanej jako SCL (ang. *Serial Clock Line*).

Dodatkowo każda z nich jest podłączona przez rezystor podciągający do zasilania. Istnienie jedynie dwóch przewodów znacząco ułatwia proces konstruowania układów elektronicznych korzystających z tego sposobu komunikacji. Pozwala to także na większą swobodę podczas projektowania płytka PCB.

Topologia interfejsu I2C oparta jest o model nadzędno - podrzędny. System składa się z dwóch typów urządzeń - nadzędnych i podrzędnych. Pierwszym z nich jest najczęściej mikrokontroler, którego zadaniem jest zarządzanie procesem transmisji poprzez wysyłanie sygnału zegarowego oraz bitów startu i stopu. Sygnał zegarowy może być wysyłany tylko przez urządzenie nadzędne. Bit startu to zmiana stanu linii danych na niski sygnalizująca początek transmisji. Po przesłaniu ciągu bitów sekwencja jest zakończona bitem stopu, czyli zmianą stanu linii danych na wysoki. Na linii zegarowej znajduje się stan wysoki podczas obu tych operacji. Drugi typ urządzenia zdefiniowanego przez wykorzystany model to urządzenie podrzędne, które jest odpowiedzialne za wysłanie danych w momencie otrzymania takiego polecenia od urządzenia nadzędnego. Podstawowy system oparty o opisywany w podrozdziale sposób transmisji przewiduje istnienie jednego urządzenia głównego oraz szeregu urządzeń podrzędnych, gdzie każde z nich ma swój własny, zdefiniowany przez producenta identyfikator. Musi on być unikalny w danym systemie. Istnieje jednak możliwość pracy w oparciu o kilka urządzeń głównych. Liczbę obsługiwanych układów ogranicza tylko maksymalna pojemność linii, która wynosi 400 pF [6].

Dane przyjmują postać ramki składającej się z 8 bitów czyli jednego bajtu. Przesyłane są począwszy bitu najbardziej znaczącego. Liczba bajtów jest zależna od specyfiki zastosowanych układów. Początkowo I2C obsługiwało adresowanie 7 - bitowe. Pierwszy bajt składa się w takiej sytuacji z 7 bitów będących adresem urządzenia podlegającego. Stan wysoki lub niski na ostatnim bicie pierwszego słowa informuje o tym, że dane są kolejno odczytywane lub wysyłane przez urządzenie główne. Nowsza wersja magistrali obsługuje adresowanie 10 - bitowe. Pierwszy bajt ma w takiej sytuacji z góry ustalone 5 początkowych bitów (11110). Ostatnie dwa bity wraz z drugim bajtem tworzą wspólnie adres urządzenia.

W realizowanym temacie pracy magistrala I2C została wykorzystana do komunikacji ESP8266 z czujnikiem MAX30102.

3.2. Interfejs SPI

Innym równie popularnym interfejsem jest SPI (ang. *Serial Peripheral Interface*) opracowany przez firmę Motorola. Standard ten oferuje zbliżone działanie do wymienionego I2C. Również bazuje na modelu nadziedzno - podlegającym oraz pozwala na szeregową, dwukierunkową oraz synchroniczną transmisję danych. Budowa składa się z co najmniej trzech przewodów mających ściśle określone funkcje, są to:

- wybór układu podlegającego, oznaczana jako CS (ang. *Chip Select*);
- synchronizacja transmisji danych poprzez wysyłanie impulsów zegarowych, oznaczana jako SCK (ang. *Serial Clock*);
- przesyłanie danych przez urządzenie nadziedzne, oznaczana jako MOSI (ang. *Master Output Slave Input*);
- przesyłanie danych przez urządzenie podlegające, oznaczana jako MISO (ang. *Master Input Slave Output*).

Możliwy jest brak jednej z linii danych w zależności od charakteru podłączonego układu. Wymieniono oznaczenia stosowane domyślnie, jednak producenci mogą korzystać z autorskich odpowiedników. Informacje przesyłane są na obu przewodach tylko w jednym kierunku. Pozwala to na jednoczesne wysyłanie i odbieranie informacji przez urządzenia. W efekcie transmisja przy użyciu interfejsu SPI odbywa się nieco szybciej, niż we wspomnianym I2C. Urządzenie główne wybiera urządzenie podlegające poprzez podanie stanu niskiego na linię CS. Istnieje możliwość podłączenia wielu układów, warunkiem jest posiadanie własnego przewodu CS przez każdy z nich. Jest to głównym czynnikiem ograniczającym liczbę urządzeń obsługiwanych przez opisywany sposób komunikacji. Interfejs SPI także pozwala na obecność w układzie wielu urządzeń nadziedznych,. Długość przesyłanej informacji może być inna dla różnych układów. Najczęściej stosuje się słowa o długości 8 i 16 bitów. W sytuacji gdy słowo składa się z dwóch bajtów możliwe jest ustawienie, który z nich ma być przesłany jako pierwszy. Transmisja odbywa się począwszy od bitu najbardziej znaczącego [1].

Interfejs SPI został w pracy wykorzystany do komunikacji wyświetlacza OLED z mikrokontrolerem. Producent zastosował autorskie oznaczenia wyprowadzeń o bliźniaczych funkcjach do wyżej wymienionych.

3.3. Protokół transmisji danych MQTT

MQTT (ang. *Message Queue Telemetry Transport*) jest to lekki protokół wymiany danych. Oparty jest o wzorzec publikacja/subskrypcja. Architektura tego systemu przewiduje istnienie trzech typów urządzeń - brokera, nadawcy i odbiorcy. Broker pełni rolę serwera pośredniczącego w transmisji danych. Jest jedynym tego typu elementem w układzie i tylko z nim kontaktują się pozostałe urządzenia. Jego zadaniem jest odbieranie danych od nadawcy i przesyłanie do odbiorcy. Dzięki takiemu rozwiązaniu urządzenia korzystające z takiego sposobu transmisji nie potrzebują znać swoich wzajemnych adresów IP - wystarczy adres brokera. Wiadomości publikowane przez nadawcę zawierają dodatkową informację o temacie. Każdy z odbiorców jest w stanie odebrać taką wiadomość subskrybując dany temat. Przekazywane informacje są danymi tekstowymi, które zazwyczaj przed wysłaniem są przekształcane do jednego z uniwersalnych formatów wymiany danych. Broker ponadto umożliwia obsługę wielu urządzeń jednocześnie [15].

Protokół ten jest bardzo często stosowany podczas tworzenia urządzeń opartych o mikrokontrolery, których zadaniem jest między innymi przesyłanie danych przez sieć. Jednym zadaniem urządzenia będącego klientem jest wybranie tematu publikowania lub subskrypcji. Operacje te nie wymagają dużej mocy obliczeniowej od takich urządzeń a pozwalają w efektywny sposób przesyłać między nimi informacje. W praktyce taki broker może zostać zrealizowany w oparciu o gotowe, dostępne w sieci rozwiązania umożliwiające utworzenie serwera na komputerze lokalnym. Jednymi z najczęściej stosowanych są Eclipse Mosquitto lub HiveMQ [16].

3.3.1. Uniwersalne formaty wymiany danych

Uniwersalne formaty wymiany danych pozwalają na przesyłanie informacji między urządzeniami i platformami o różnych specyfikacjach. Jednym wymaganiem jest możliwość rozszyfrowania danego typu zapisu przez każde z nich. Większość języków oferuje biblioteki zawierające funkcje odpowiedzialne za zapisywanie i odczytywanie danego typu. W związku z tym jeżeli celem jest tylko przesłanie danych do innego urządzenia a nie zapisanie ich do pliku w określonym formacie, to wystarczające w większości przypadków będzie wywołanie odpowiedniej funkcji oraz podanie jako argumentu treści wiadomości.

Przykładami najczęściej stosowanych formatów wymiany danych komputerowych są między innymi typy JSON (ang. *JavaScript Object Notation*) oraz

XML (ang. *Extensible Markup Language*). Oferują wyjątkową prostotę składni oraz są obsługiwane przez większość powszechnie używanych obecnie języków programowania[17-18]. Porównanie prostej wiadomości zapisanej przy użyciu obu formatów przedstawia rys. 3.1.

```
{  
    "student": [  
        {  
            "imie": "Mikolaj",  
            "nazwisko": "Skrzyniarz"  
        }  
    ]  
}
```

```
<student>  
    <imie>Mikolaj</imie>  
    <nazwisko>Skrzyniarz</nazwisko>  
</student>
```

Rys. 3.1. Dane zapisane jako JSON (lewa strona) oraz XML (prawa strona).

4. Projekt i implementacja oprogramowania

Założeniem czwartego rozdziału pracy jest przedstawienie narzędzi wykorzystanych podczas tworzenia oprogramowania. Zarówno program zaimplementowany na mikrokontrolerze jak i projekt aplikacji zewnętrznej stworzone zostały przy użyciu bliźniaczych technologii. Oparte w większości o obiektowy paradymat programowania. Opisano język programowania oraz użyte środowisko programistyczne.

4.1. Język programowania Python

Jednym językiem programowania wykorzystanym podczas realizacji części programowej tematu pracy był Python. Jest to wysokopoziomowy, interpretowany język programowania mający bardzo szerokie zastosowanie. Charakteryzuje się wysoką przejrzystością oraz czytelnością pisanej kodu, przypominającym w dużym stopniu zdania pisane w języku angielskim. Procesy takie jak zarządzanie pamięcią oraz definiowanie typów deklarowanych zmiennych odbywają się automatycznie, co jest sporym ułatwieniem zwłaszcza dla osób początkujących. W efekcie osoba pisząca kod w tym języku może przyłożyć większą uwagę do poprawności logiki pisanej programu niż do samej składni. Takie ułatwienia wymagają większego nakładu pracy przez urządzenie uruchamiające program napisany w tym języku. Sam fakt, że kod jest interpretowany a nie komplikowany sprawia, że Python jest językiem wolniejszym niż konkurencyjne C++ lub Java. Kompilacja jest procesem tłumaczenia kodu napisanego w danym języku w większości przypadków na kod maszynowy, którego odczytanie i wykonanie są możliwe przez dane urządzenie. Interpretacja polega na bieżącym wykonywaniu danych linii kodu. W efekcie te same instrukcje mogą być tłumaczone kilkukrotnie, w przeciwieństwie do procesu komplikacji. Język ten jednak nadal cechuje się dużą popularnością na rynku. Jest używany w obszarach uczenia maszynowego, analizy danych oraz tworzenia aplikacji internetowych. Duże wsparcie producenta oraz zainteresowanie społeczności powoduje ciągłe powstawanie nowych bibliotek dodatkowo poszerzających możliwości tego języka [19-20].

Python jak i większość najczęściej stosowanych języków oferuje możliwość programowania obiektowego. Jest to obecnie najpopularniejszy sposób tworzenia oprogramowania. W takim systemie program jest podzielony na obiekty wywołane na podstawie istniejących klas. Klasa jest to swego rodzaju wzorzec, zawierający atrybuty czyli zmienne zdefiniowane wewnątrz niej oraz metody czyli funkcje mogące zarządzać atrybutami. Dzięki temu możliwe jest tworzenie wielu obiektów mających te same, zadeklarowane wewnątrz rodzaje zmiennych ale różniące się wartościami. Jeżeli użytkownik ma na celu zmianę zachowywania się jakiegokolwiek z obiektów zmuszony

jest do zastosowania tych zmian dla wszystkich instancji danej klasy. Realizuje się to poprzez zmianę kodu źródłowego tej klasy. Proces manipulowania danymi obiektu odbywa się poprzez naprzemienne wywoływanie różnego rodzaju metod. Powstanie obiektowego paradygmatu programowania okazało się mieć rewolucyjne skutki. W znaczący sposób zwiększa to czytelność napisanego kodu źródłowego. Ponadto jest to podejście najbardziej zbliżone do postrzegania rzeczywistości przez ludzi. Dla większych projektów taka technika programowania pozwala także zmniejszyć ilość kodu. Możliwe jest wielokrotne odwołanie się do tych samych fragmentów poprzez utworzenie odpowiedniego obiektu lub wywołanie metody. W porównaniu do programowania proceduralnego dużym ułatwieniem jest deklarowanie zmiennych wewnętrz klasy. Ogranicza to ich ilość w stosunku do sytuacji, gdy konieczne byłoby każdorazowe stworzenie zmiennej globalnej [21].

Kod źródłowy przeznaczony na mikrokontroler bazował na dedykowanej takim zastosowaniom implementacji Python'a o nazwie MicroPython. Zawiera ona jedynie podstawowe funkcje języka macierzystego, rozszerzone o moduły pozwalające na obsługę mikrokontrolerów. Całość składni pozostaje jednak taka sama jak w języku oryginalnym. Przeprowadzona optymalizacja pozwala na pracę w oparciu tą implementację na urządzeniach mających jedynie 256 KB pamięci flash oraz 16 KB pamięci RAM. Popularność takiego zastosowania pozostaje nadal niewielka, w związku z małą liczbą obsługiwanych urządzeń [22].

4.2. Środowisko programistyczne PyCharm

PyCharm jest wieloplatformowym, zintegrowanym środowiskiem programistycznym rozwijanym przez firmę JetBrains. Z założenia dedykowany językowi Python. Udostępniane funkcje to:

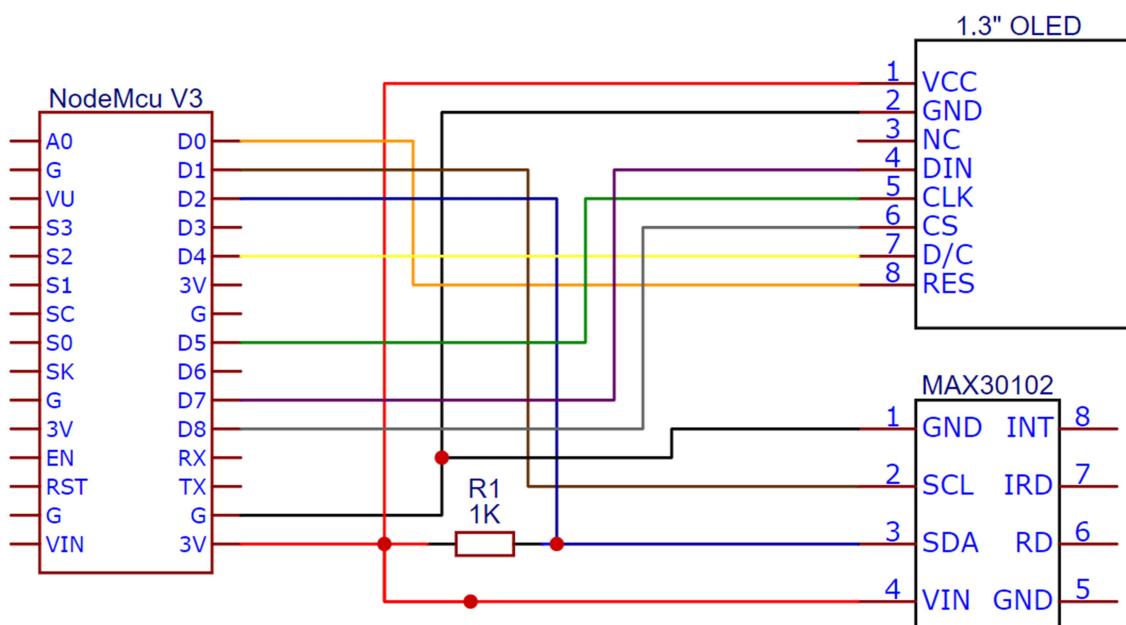
- Obsługa języków takich jak Python, JavaScript, CoffeScript, TypeScript, SQL, HTML oraz CSS.
- Wsparcie dla technologii odpowiedzialnych za tworzenie aplikacji internetowych takich jak Django, Flask, Google App Engine, Pyramid oraz web2py.
- Inteligentne podświetlanie składni, wyświetlanie podpowiedzi oraz opisów funkcji.
- Automatyczna, częściowa refaktoryzacja, czyli zmiana struktury napisanego programu bez zmiany jego funkcjonalności oraz logiki. Oparta o ogólnie przyjęte zasady pisania poprawnego, czytelnego kodu.
- Wsparcie dla systemów kontroli wersji takich jak Git, SVN oraz Mercurial.
- Zintegrowane terminal, konsola oraz debugger.
- Uproszczona nawigacja między plikami wchodząymi w skład projektu.
- Możliwość instalacji dodatkowych wtyczek oraz modułów [23].

5. Opis realizacji tematu pracy

Piąty rozdział zawiera opis procesu realizacji urządzenia zawartego w temacie pracy. Przedstawia poszczególne etapy w kolejności chronologicznej oraz efekty z nich wynikające.

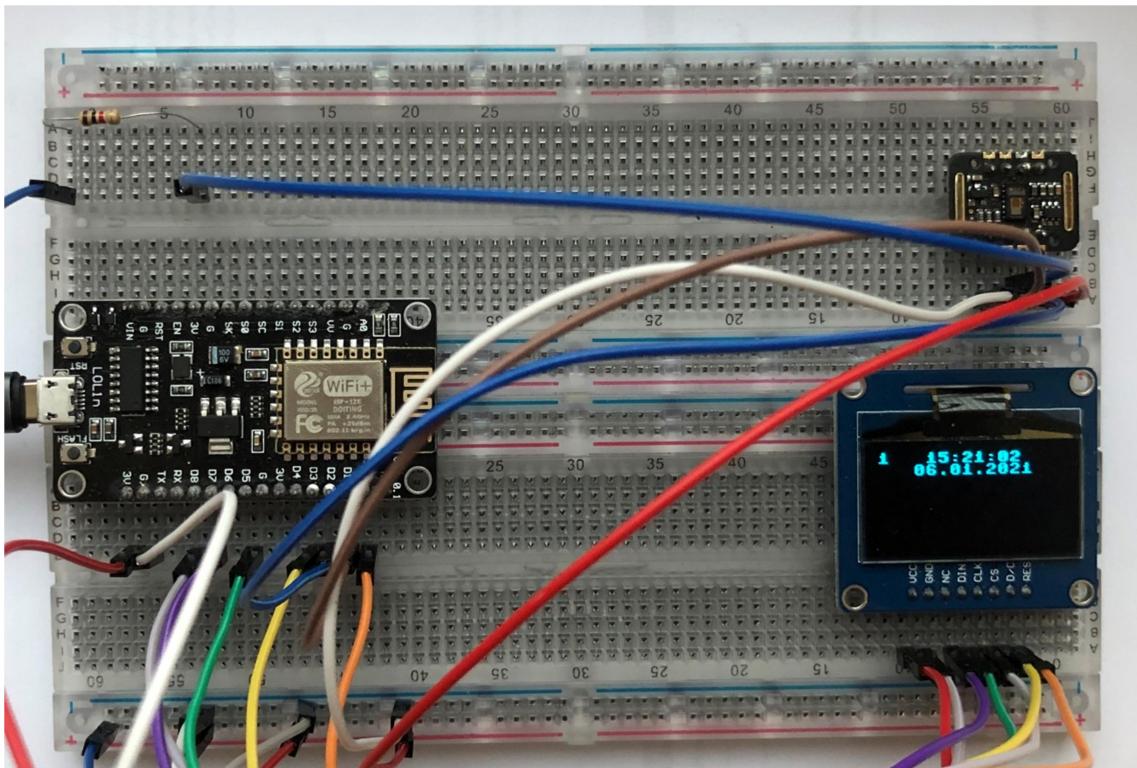
5.1. Projekt układu elektronicznego

Pierwszym etapem części praktycznej pracy była konstrukcja układu w oparciu o dobrane podzespoły. Urządzenie miało charakter prototypowy - całość połączeń wykonano na płytce stykowej. Wyświetlacz połączono przy użyciu domyślnie zastosowanego przez producenta 4 - przewodowego interfejsu SPI. Dodatkowo użyto złącza RES na którym podanie stanu niskiego skutkuje powrotem przez wyświetlacz do ustawień domyślnych. MAX30102 połączono z płytą rozwojową przy użyciu złącz dedykowanych dla magistrali I2C. Ideowy schemat wykonanych połączeń przedstawiony jest na rys. 5.1. W przypadku jednego urządzenia nadzawanego producent zaleca zastosowanie jedynie rezystora podciągającego o oporze co najmniej $500\ \Omega$ dla linii danych [4]. Wykorzystany w pracy mikrokontroler pozwala na konfigurację wyprowadzeń z wbudowanym podciąganiem, niestety producent nie podaje dokładnej wartości zastosowanej rezystancji. Autor pracy zdecydował się na użycie fizycznego rezystora o rezystancji równej $1\ k\Omega$ dla linii SDA. Użycie pozostałych wyprowadzeń czujnika nie było konieczne do poprawnego działania układu. Istnieje niewielka dowolność w kwestii doboru wyprowadzeń NodeMCU (patrz rys. 2.1), proponowane rozwiązanie uznano jednak za optymalne.



Rys. 5.1. Schemat ideowy wykonanego urządzenia.

Rzeczywisty układ przedstawia rys. 5.2. W celu prezentacji czujnik połączono bezpośrednio na płytce stykowej. W taki sposób możliwe jest wykonanie pomiarów z powierzchni palca. Domyślnie podczas projektowania oprogramowania oraz testowania czujnik był wyprowadzony przy pomocy mierzących około 60 cm przewodów i mocowany na nadgarstku. Na zdjęciu widoczny jest układ w stanie spoczynkowym. W takiej sytuacji na wyświetlaczu znajduje się informacja o identyfikatorze danego urządzenia wraz z godziną i datą zmieniającymi się w czasie rzeczywistym. Mierzone wartości oraz ewentualna wiadomość na temat stanu alarmowego pojawiają się w dolnej części ekranu.



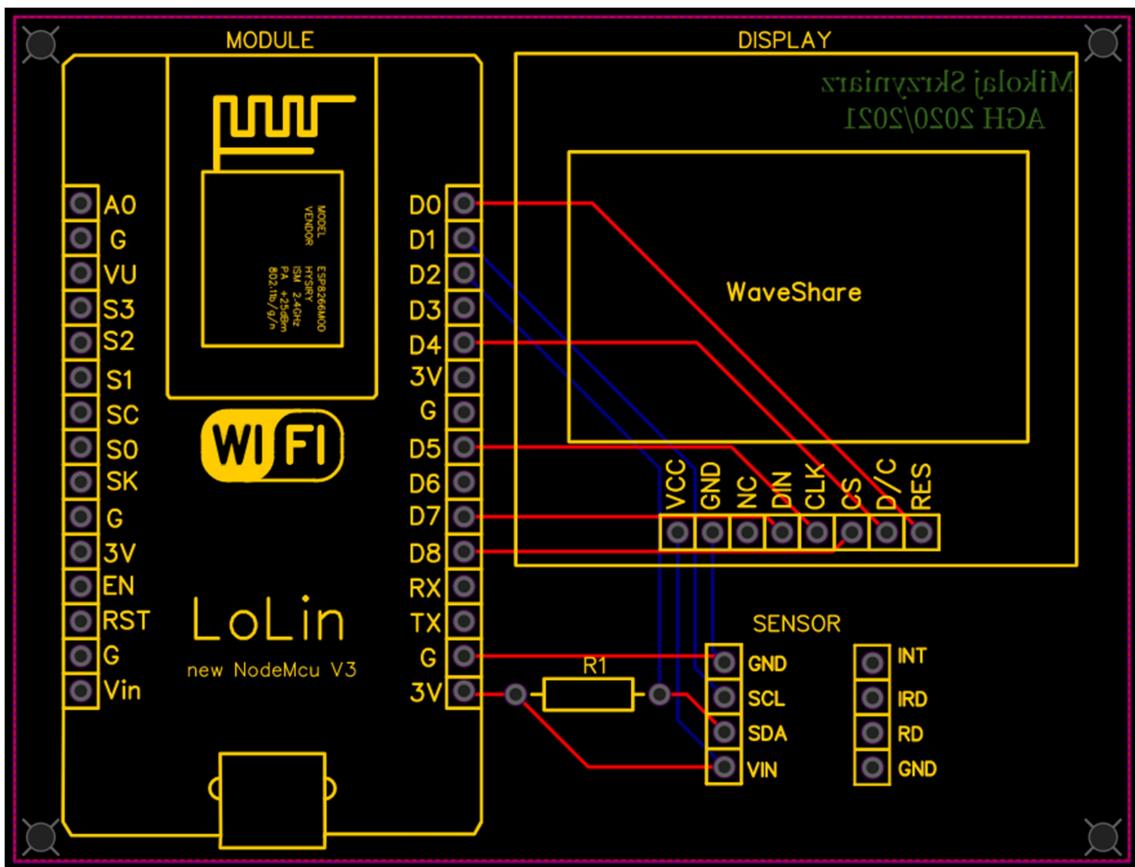
Rys. 5.2. Skonstruowane urządzenie.

Na podstawie schematu ideowego (patrz rys. 5.1) wykonano projekt płytki PCB. W tym celu należało użyć dedykowanego takim zastosowaniom oprogramowania. Autor pracy zdecydował się na skorzystanie z ogólnodostępnego, darmowego środowiska EasyEDA. Umożliwia tworzenie obwodów, ich symulacje oraz projektowanie płyt drukowanych. Udostępnia biblioteki zawierające gotowe do użycia schematy elementów i podzespołów elektronicznych, modułów oraz płyt rozwojowych. Ponadto oferuje funkcje tworzenia własnych elementów bibliotecznych oraz pozwala na korzystanie z wykonanych przez innych użytkowników. Dostęp do tego środowiska jest możliwy poprzez przeglądarkę. W efekcie zgromadziło ono duże grono użytkowników [24]. Można jednak użyć jakiegokolwiek innego oprogramowania przeznaczonego do projektowania PCB jak Eagle lub KiCad.

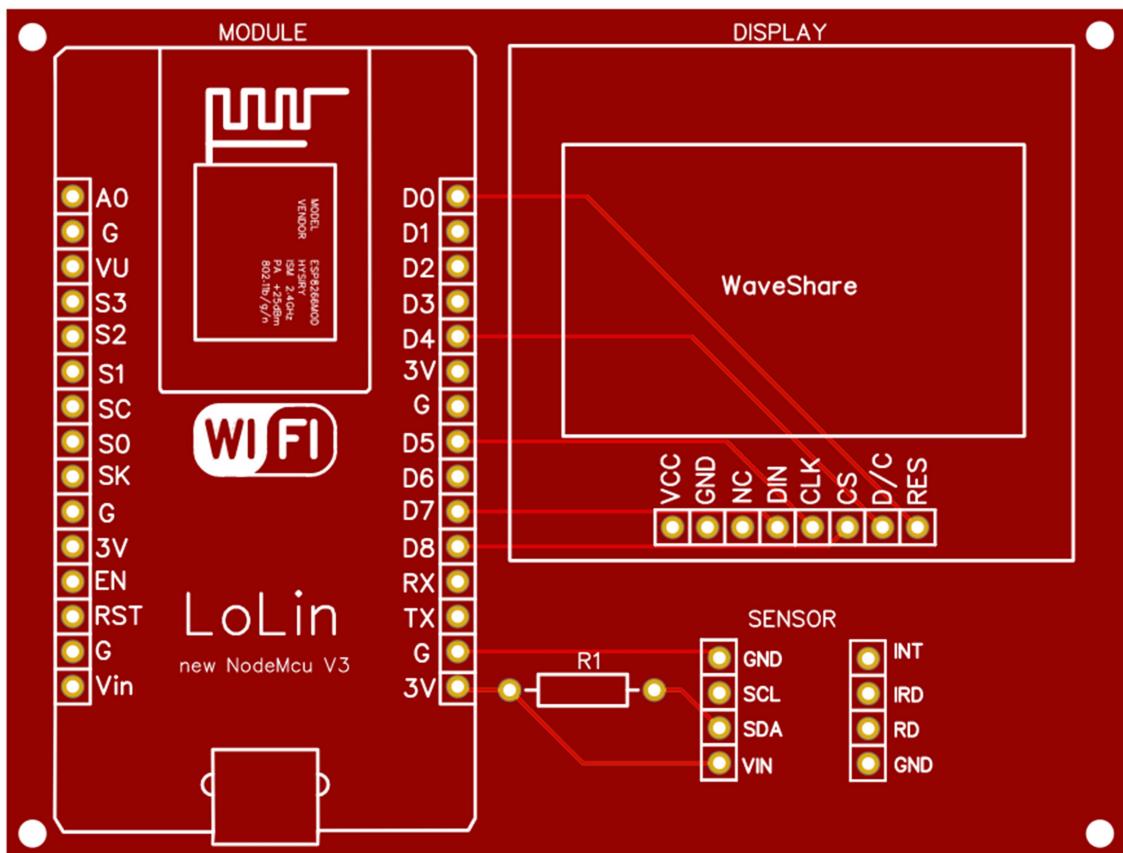
Tworzenie projektu płytki drukowanej składa się z kilku etapów. Są to kolejno wykonanie schematu ideowego, wygenerowanie na jego podstawie pliku z projektem

PCB oraz stworzenie ścieżek między odpowiednimi złączami. Podczas tworzenia projektów składających się z dużej ilości elementów konieczne jest zwrócenie na odległości między ścieżkami oraz ich grubości. Wielkości te są zależne od wartości napięć i prądów. Płytki może składać się z kilku warstw na których są prowadzone połączenia. Liczba ta jest w największym stopniu zależna od ilości elementów, ich wyprowadzeń oraz rozmiaru płytki [2].

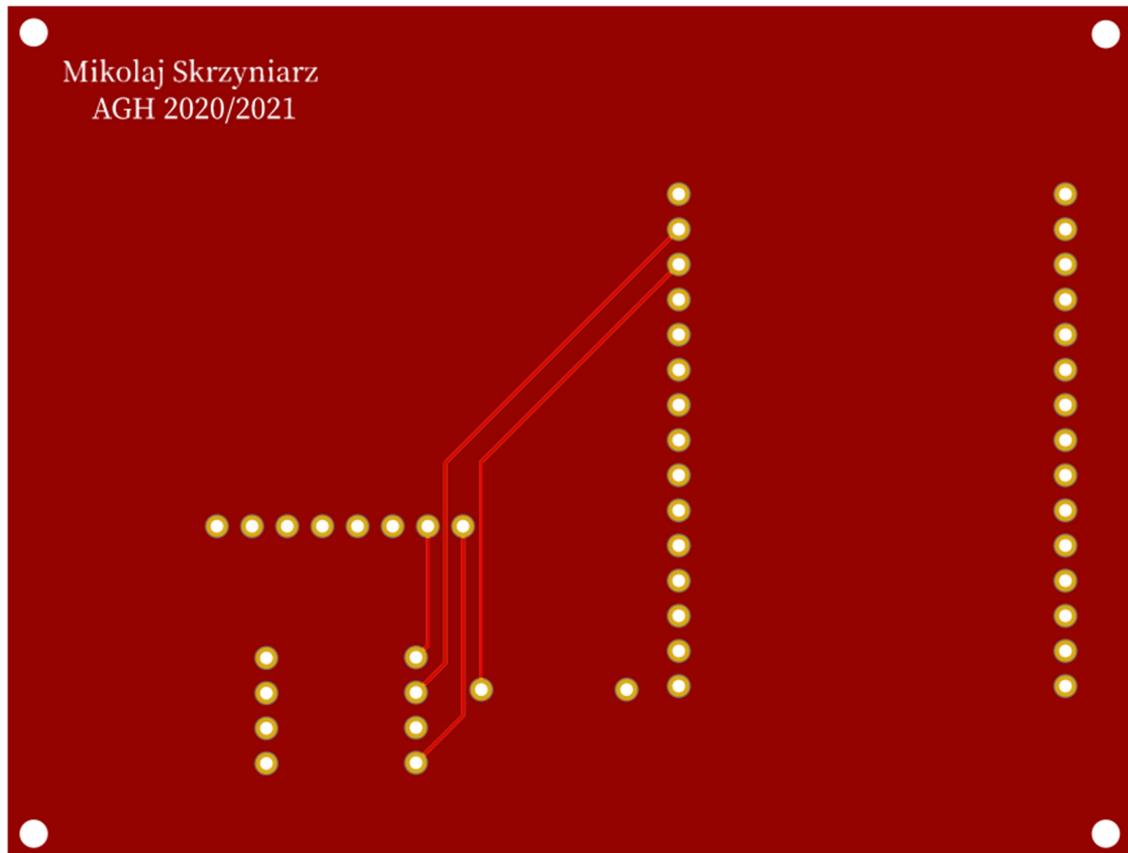
W skład układu pomiarowego wchodzą 3 podzespoły. Maksymalne napięcie wynosi 3,3 V. Pozwoliło to na większą dowolność podczas prowadzenia ścieżek. Wymiary zaprojektowanej płytki to 81x61 mm. Możliwe jest zamontowanie czujnika MAX30102 zarówno bezpośrednio na jej powierzchni jak i podłączenie go za pomocą przewodów. Podczas projektowania wzięto pod uwagę ewentualną potrzebę przymocowania płytki do obudowy. Służą do tego 4 miejsca na śruby, znajdujące się na każdym z rogów. Widok pliku projektowego z poprowadzonymi ścieżkami przedstawia rys. 5.3. Górną i dolną warstwa gotowej płytki PCB widoczne są odpowiednio na rys. 5.4 i rys. 5.5. Warto zaznaczyć, że temat pracy przewidywał jedynie wykonanie projektu płytki drukowanej. Wszystkie opisane operacje oraz przeprowadzone testy zostały wykonane przy pomocy układu prototypowego (patrz rys. 5.2). Nic to nie jednak nie zmienia w kwestii funkcjonalności skonstruowanego urządzenia.



Rys. 5.3. Projekt płytki PCB z poprowadzonymi ścieżkami.



Rys. 5.4. Górná warstwa wykonanej płytka PCB.



Rys. 5.5. Dolna warstwa wykonanej płytka PCB.

5.2. Obsługa elementów składowych

Przed przystąpieniem do jakichkolwiek operacji związanych z implementacją oprogramowania konieczne było zainstalowanie środowiska programistycznego [25] na lokalnym PC oraz oprogramowania umożliwiającego programowanie ESP8266 w języku Python [26]. Następnie można było przystąpić do programowej komunikacji elementów układu. Ograniczało się to do połączenia czujnika oraz wyświetlacza z płytą rozwojową. Całość kodu źródłowego zawartego w pracy została opisana przy użyciu komentarzy oraz ciągów dokumentacyjnych (ang. *docstrings*). Zostało to wykonane w języku angielskim, zgodnie z ogólnie panującą praktyką pisania poprawnego kodu.

Struktura projektu narzucana jest przez oprogramowanie MicroPython. Pliki o nazwie boot.py oraz main.py są charakterystyczne dla każdego projektu. Plik o nazwie boot.py uruchamia się tylko raz po uruchomieniu mikrokontrolera. Następnie uruchamiany jest plik main.py działający w pętli. Takie rozwiązanie jest powszechnie stosowane podczas programowania różnego rodzaju mikrokontrolerów.

W celu obsługi wyświetlacza stworzono element biblioteczny, bazujący na jednej z bibliotek dedykowanych ekranom opartym o sterownik sh1106. Dodano między innymi predefiniowane zmienne odpowiedzialne za oznaczenie poszczególnych wyprowadzeń. Dzięki temu do poprawnego działania wystarczające było stworzenie instancji danej klasy. Wyświetlenie tekstu na ekranie ogranicza się do wyczyszczenia jego fragmentu, ustawienia treści oraz wywołania procedury odpowiedzialnej za odświeżenie stanu wyświetlacza. Wymagane jest podanie dokładnych współrzędnych. Maksymalne wartości na obu osiach są tożsame z rozdzielcością. Punkt (0,0) jest lewym, górnym rogiem ekranu. Przykładowa funkcja wyświetlająca tekst przedstawiona jest na rys. 5.6.

```
def show_time(self, time, date, id):
    """ Show present date, time and device id on display. These are being displayed constantly. """
    # Fill the 128x16 pixels rectangle starting from 0x0 with 0s. It makes this part of display
    # completely clean. Any part of display may be cleared like this. It can be also filled with
    # 1s. 0 is dark pixel, seen as no pixel, 1 is 'blue' pixel.
    self.display.fill_rect(0, 0, 128, 16, 0)

    # Display text 'str(id)' starting from 0x0.
    self.display.text(str(id), 0, 0, 1)
    self.display.text(time, 32, 0, 1)
    self.display.text(date, 24, 8, 1)
    self.display.show()
```

Rys. 5.6. Funkcja odpowiedzialna za wyświetlenie identyfikatora urządzenia, daty oraz czasu.

Obsługa czujnika MAX30102 odbywa się przy użyciu biblioteki stworzonej na podstawie dokumentacji [4]. Tak jak w przypadku wyświetlacza oznaczenia wyprowadzeń zostały predefiniowane - do poprawnego uruchomienia konieczne jest tylko stworzenie obiektu danej klasy. Jak wspomniano konfiguracja odbywa się poprzez

wpisanie odpowiedniej wartości do rejestru. Warto zaznaczyć, że musi mieć ona formę wektora bajtowego (ang.*bytarray*). Funkcja odpowiedzialna za ten proces jest widoczna na rys. 5.7. Dodatkowym argumentem wbudowanej funkcji odpowiedzialnej za zmianę zawartości rejestrów jest adres urządzenia znajdującego się na magistrali I2C. Domyślnym adresem użytego w pracy modelu MAX30102 jest 87.

```
def write(self, reg, value):
    """ Set value named 'value' in register named 'reg'. """
    data = bytearray(1) # One byte long array.
    data[0] = value
    self.i2c.writeto_mem(address, reg, data)
```

Rys. 5.7. Funkcja zapisująca wartość do danego rejestru.

Wraz z stworzeniem instancji danej klasy wywoływana jest funkcja inicjująca. Jest ona odpowiedzialna za ustawienie wartości domyślnych wszystkich rejestrów, wyczyszczenie statusów przerwań oraz wywołania funkcji konfigurującej. Odpowiedzialna jest za ustawienie następujących parametrów:

- występowanie przerwania informującego o zakończeniu przetwarzania analogowo - cyfrowego wartości temperatury włączone;
- znacznik zapisu równa 0;
- licznik przepelnienia równy 0;
- znacznik odczytu równy 0;
- 4 próbki jako liczba próbek z których obliczana jest średnia przed wpisaniem do bufora;
- zapisywanie próbek po osiągnięciu limitu w buforze włączone;
- 17 jako liczba wolnych miejsc wywołujących przerwanie;
- praca w trybie SpO₂;
- praca w maksymalnym zakresie przetwornika A/C;
- częstotliwość próbkowania równa 100Hz;
- szerokość impulsu 411μs;
- rozdzielcość przetwornika równa 18 bitów;
- wyłączenie diody emitującej światło czerwone;
- prąd diody emitującej światło podczerwone równy 3,2 mA.

Są to domyślne warunki pracy czujnika. Prądy diod są ustawione na niskie wartości w celu oszczędzenia poboru mocy w spoczynku. Zbliżenie się ciała do czujnika powoduje wzrost odczytu z diody podczerwonej. Gdy osiągnie ustaloną wartość prądy diod ustawiane są na natężenia równe 9,4 mA.

Proces odczytu wartości mierzonych przez czujnik przebiega w kilku etapach. Pierwszym z nich jest sprawdzenie liczby dostępnych w buforze FIFO próbek. Liczba ta jest równa różnicy między wartościami wskazywanymi przez znacznik zapisu a odczytu. Następnie wywoływana jest funkcja odpowiedzialna za odczyt wartości z bufora FIFO. Wykonuje się do momentu zapisania wszystkich danych. Podczas pracy

w podanej konfiguracji jedna próbka zawierająca wartość odpowiadającą ilości światła odbitego obu diod ma rozmiar 6 bajtów. Najstarszy bajt składa się tylko z dwóch bitów, ze względu na zastosowaną rozdzielcość przetwornika. Ostatecznie rozmiar jednej próbki dla każdej z diod wynosi 18 bitów. Warto zaznaczyć, że w użytym modelu czujnika kanały są zamienione kolejnością względem dokumentacji podanej przez producenta [4]. W pierwszej kolejności odczytywane są dane zebrane przez diodę podczerwoną. Zostało to zaobserwowane podczas dobierania parametrów pracy. Funkcję odpowiedzialną za odczyt danych z bufora FIFO przedstawia rys. 5.8.

```
def read_fifo(self):
    """ Read red and ir values from the FIFO register. """
    # Clear both buffers with FIFO data read last time.
    red = None
    ir = None

    # Read 6 byte long data from the device.
    fifo_data = self.i2c.readfrom_mem(address, fifo_data_register, 6)

    # Mask bytes unused bytes[23:18]. Channels are swapped in used model of sensor.
    ir = (fifo_data[0] << 16 | fifo_data[1] << 8 | fifo_data[2]) & 0x3fff # 0x3ffff = 2^18 - 1
    red = (fifo_data[3] << 16 | fifo_data[4] << 8 | fifo_data[5]) & 0x3fff

    return red, ir
```

Rys. 5.8. Funkcja mająca za zadanie odczyt danych z bufora FIFO.

Wartość zwracana przez czujnik jest średnią z danych odczytywanych z bufora. Ma to związek z czasem trwania jednej pętli programu, który wynosi około 70 ms. W takim czasie w buforze FIFO może zostać zapisanych do dwóch próbek. Czas wykonywania pętli w tym wypadku determinuje tylko liczba wykonywanych instrukcji. Nie zastosowano dodatkowych funkcji mających na celu opóźnienia czasowe.

Podczas każdego obiegu pętli wykonywany jest także pomiar temperatury. Proces ten rozpoczyna wpisanie odpowiedniej wartości do rejestru statusowego. Inicjuje to przetwarzanie analogowo - cyfrowe wartości temperatury. Koniec tego procesu symbolizuje odpowiednia wartość w jednym z rejestrów statusowych. Gdy konwersja jest zakończona można odczytać wartości rejestrów przechowujących część całkowitą oraz liczbę kroków odpowiedzialną za część ułamkową. Odczytanie części ułamkowej czyści wspomniany wyżej register statusowy. Wyjściowa wartość temperatury obliczana jest ze wzoru:

$$T_{out} = T_{int} + T_{frac} * step_{frac} \quad (1)$$

gdzie:

- T_{out} - wyjściowa wartość temperatury [$^{\circ}\text{C}$];
- T_{int} - odczytana wartość całkowita temperatury [$^{\circ}\text{C}$];
- T_{frac} - liczba kroków odpowiedzialna za część ułamkową;
- $step_{frac}$ - wielkość kroku równa 0,0625 [$^{\circ}\text{C}$].

Manipulacja przedstawionymi funkcjami odbywa się przy użyciu jednej, wywoływanej podczas każdego obiegu pętli metody. Zwraca ona gotowe do obróbki dane odczytane przez czujnik. Jej struktura jest przedstawiona na rys. 5.9.

```
def read_values(self):
    """ Read 'samples' samples from FIFO. Return ready to process data. """
    red_buf = []
    ir_buf = []
    samples = self.get_data_samples()

    while samples > 0:
        red, ir = self.read_fifo()
        red_buf.append(red)
        ir_buf.append(ir)
        samples -= 1

    red = math.mean(red_buf)
    ir = math.mean(ir_buf)
    temperature = self.read_temperature()
    return red, ir, temperature
```

Rys. 5.9. Główna funkcja odpowiedzialna za odczyt wartości z czujnika.

5.3. Interpretacja odczytanych danych

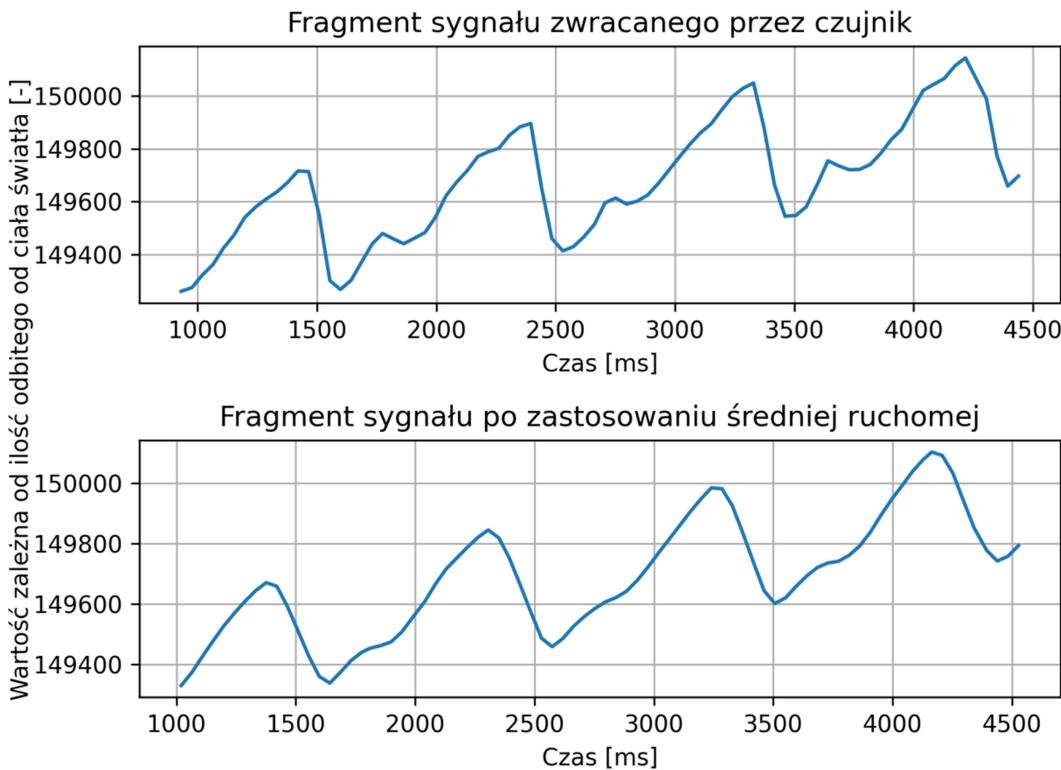
Kolejnym etapem projektu oprogramowania przeznaczonego na mikrokontroler było stworzenie algorytmu odpowiedzialnego za wyznaczanie poziomów pulsu oraz nasycenia krwi tlenem na podstawie uzyskiwanych pomiarów. Jak wspomniano wartości zwracane przez czujnik tworzą krzywą pletyzmograficzną. W celu poprawy jakości takiego sygnału zastosowano filtr typu „średnia ruchoma”. Jego działanie polega na obliczaniu średniej z wybranej liczby ostatnio uzyskanych próbek. Autor pracy zdecydował się na uśrednianie 5 próbek. Filtr ten opisuje następujący wzór [7]:

$$y(i) = \frac{1}{W} \sum_{j=i-W+1}^i u(j) \quad (2)$$

gdzie:

- y - uśredniona wartość sygnału;
- W - liczba przeszłych próbek;
- u - wartość sygnału wejściowego.

Fragment sygnału zwanego przez czujnik przedstawia rys. 5.10. Widoczny jest także wygładzony sygnał po zastosowaniu średniej ruchomej.



Rys. 5.10. Porównanie sygnału odczytanego na podstawie światła diody podczerwonej przed i po zastosowaniu średniej ruchomej.

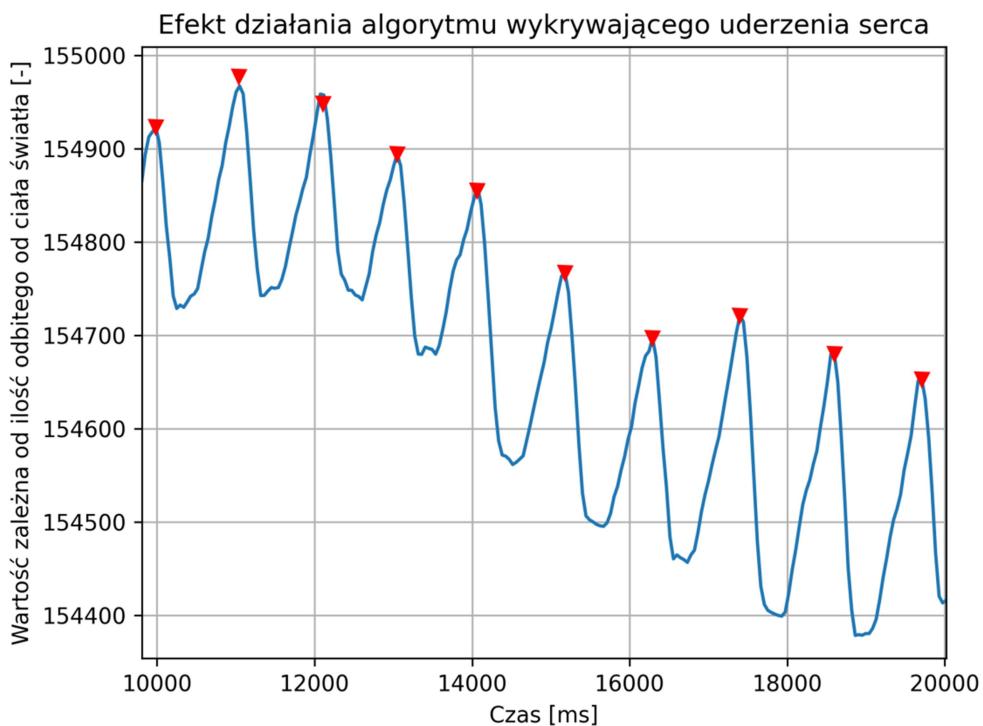
Dzięki zastosowaniu takiego filtru liczba zakłóceń została znacznie ograniczona. Zachowane zostały przy tym te same odstępy czasowe między uderzeniami serca. Przebieg tętna może się jednak różnić od warunków. Jest także indywidualny dla każdej osoby. Niewykluczone jest wystąpienie szumów mimo zastosowanych operacji. Będą one jednak znacznie mniejsze niż w przypadku domyślnie uzyskiwanego sygnału.

Poziom pulsu jest wyznaczany na podstawie sygnału generowanego przez odbite światło podczerwone. W większości przypadków jedynymi maksymami lokalnymi takiej krzywej będą jej szczyty. Każdy z nich jest skutkiem uderzenia serca, a czas między nimi determinuje częstotliwość tętna. Zaimplementowany algorytm w sposób prawidłowy lokalizował takie próbki na podstawie badanych w czasie rzeczywistym relacji między nimi. Fragment zarejestrowanego sygnału wraz z zaznaczonymi maksymami lokalnymi przedstawia rys. 5.11. W pamięci programu zapisywane są czasy wystąpienia dwóch najmłodszych szczytów. Nagły skok pulsu o 20 uderzeń jest rozpoznawany jako błędny odczyt. Takie wartości są odrzucane. Ma to na celu zniwelowanie wpływu drobnych ruchów ciała wykonywanych podczas pomiaru. Wartość pulsu wyznaczana jest na podstawie wzoru:

$$BPM = 60 * \frac{1000}{\Delta t} \quad (3)$$

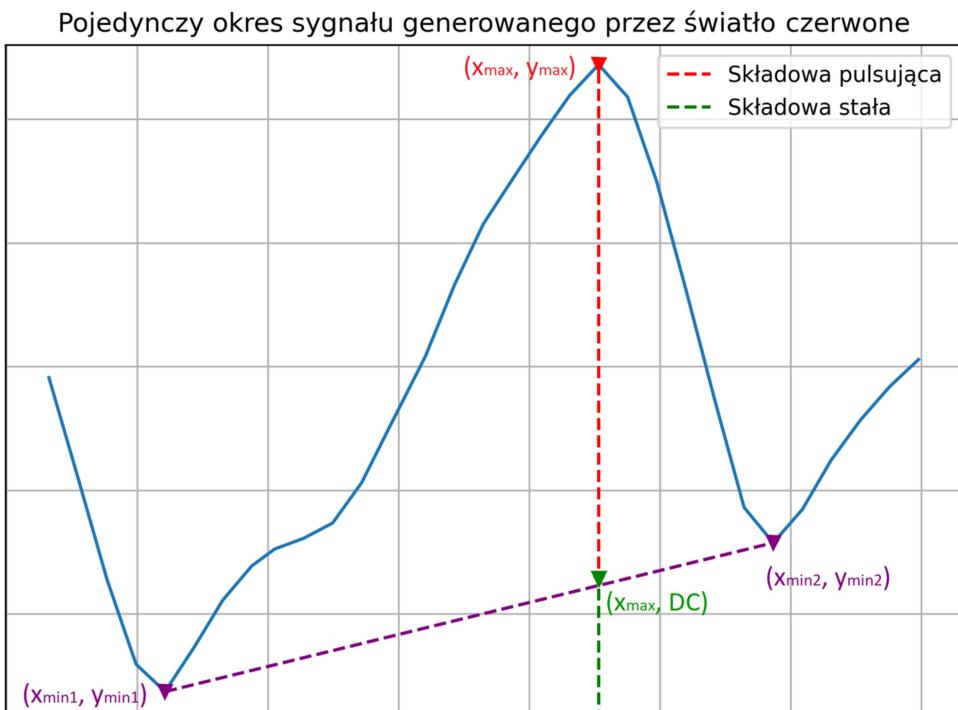
gdzie:

- BPM - ilość uderzeń serca na minutę [Hz];
- Δt - czas między dwoma najmłodszymi szczytami [ms].



Rys. 5.11. Efekt działania algorytmu odpowiedzialnego za wykrywanie uderzeń serca.

Wyznaczenie poziomu starucaji krwi tlenem wymaga analizy sygnałów generowanych na podstawie dwóch emitowanych rodzai światel. Uzyskiwane przebiegi składają się z części pulsującej oraz stałej. Do obliczeń wymagane jest poznanie wartości obu z nich. Przykład jednego okresu sygnału, wraz z oznaczeniami użytymi podczas dalszych obliczeń jest widoczny na rys. 5.12.



Rys. 5.12. Przykładowy okres sygnału generowanego przez światło czerwone, z widocznymi wielkościami potrzebnymi do wyznaczenia poziomu saturacji krwi tlenem.

Jak widać na rys. 5.12 dwa lokalne minima tworzą wspólnie funkcję liniową. Znając ich współrzędne możliwe jest opisanie jej wzorem [27]. Wartość składowej stałej jest równa wartości tej funkcji dla czasu uderzenia serca. Wyjściowy wzór, wraz z odpowiednimi oznaczeniami wygląda następująco:

$$DC = \frac{y_{min1} - y_{min2}}{x_{min1} - x_{min2}} * x_{max} + (y_{min1} - \frac{y_{min1} - y_{min2}}{x_{min1} - x_{min2}} * x_{min1}) \quad (4)$$

gdzie:

- DC - składowa stała sygnału;
- x_{min} - wartość czasu minimum znajdującego się przed maksimum;
- y_{min1} - wartość sygnału w minimum znajdującego się przed maksimum;
- x_{min2} - wartość czasu minimum znajdującego się za maksimum;
- y_{min} - wartość sygnału w minimum znajdującego się za maksimum;
- x_{max} - wartość czasu badanego maksimum.

Składowa pulsująca opisana jest następującym wzorem:

$$AC = x_{max} - DC \quad (5)$$

Stosunek między składowymi sygnału świadczy o poziomie absorpcji danego rodzaju światła. Wielkość łącząca wchłanianność obu rodzajów emitowanych światel jest opisana wzorem:

$$R = \frac{\frac{AC_{red}}{DC_{red}}}{\frac{AC_{ir}}{DC_{ir}}} \quad (6)$$

gdzie:

- R - stała absorpcji, zwana również „stosunkiem stosunków” (ang. *ratio of ratios*);
- AC_{red} - składowa pulsująca sygnału generowanego przez światło czerwone;
- DC_{red} - składowa stała sygnału generowanego przez światło czerwone;
- AC_{ir} - składowa pulsująca sygnału generowanego przez światło podczerwone;
- DC_{ir} - składowa stała sygnału generowanego przez światło podczerwone.

Wielkość ta jest wprost proporcjonalne do mierzonej wartości saturacji, która opisana jest wzorem [8]:

$$SpO_2 = 104 - 17R [\%] \quad (7)$$

Stworzony algorytm został opracowany bazując na zarejestrowanym przez czujnik sygnału, który został zapisany do pliku CSV. Praca w oparciu o plik tekstowy z gotowymi pomiarami w znacznym stopniu ułatwiła kontrolę całego procesu oraz prawidłowości obliczeń. Pomiar ten trwał około 40 sekund. Algorytm został

zaimplementowany na mikrokontrolerze dopiero, gdy udało się z powodzeniem odczytać wszystkie wartości z przebiegu testowego. Struktura funkcji głównej odpowiedzialnej za całość opisanych wyżej obliczeń, wraz z dodatkowym zapisywaniem odpowiednich próbek oraz flag jest przedstawiona na rys. 5.13.

```

def count_hr_spo(self, ir_value, red_value, time_value):
    """ Main algorithm responsible for all calculations."""
    self.new_values = False
    self.reorder_samples(ir_value, red_value, time_value)
    # Calculations are made for both of signals.
    for signal in [ir, red]:
        previous = signal - 2
        # Try to detect any edge and any extremum.
        self.detect_edge(signal)
        self.detect_extremum(signal)
        # Heartbeat is detected based on value from IR led.
        if signal == ir:
            # If true local maximum was detected and difference between it and the previous one is higher than
            # minimal declared value and higher than maximal declared value count it as a heartbeat.
            max_min_diff = self.local_max[signal] - self.local_min[signal]
            if (self.extremum[signal]) and (max_min_diff > self.min_diff) and (max_min_diff < self.max_diff):
                # Save previous time as a present beat time, then count time between present and previous beat time.
                self.beat_time[-1] = self.sample[time_previous]
                delta = self.beat_time[-1] - self.beat_time[-2]
                # Calculate the bpm based on delta time in ms.
                bpm = 60 / (delta/1000)
                # If time between beats is greater than 3 seconds then setting everything up again is needed.
                if delta >= 3000 and self.beats > 2:
                    self.setup()
                    continue
                # Anti shake condition. If difference between saved and gotten bpm is lower by value set then the
                # measure is likely valid. It counts as soon as 10 continuous beats are obtained. Every measure
                # counts if saved bpm is lower than 50
                if (self.bpm < 50) or (math.abs(bpm - self.bpm) < 20) or (self.beats < 10):
                    self.beat_time[-2] = self.beat_time[-1]
                    self.beats += 1
                    if self.beats > 1:
                        # If more than one beat collected append the bpm buffer with this value.
                        self.bpm_buf.append(bpm)
                        if len(self.bpm_buf) > 10:
                            # If more than 10 values collected, reject the oldest one.
                            self.bpm_buf = self.bpm_buf[1::]
                            # Calculate bpm value as a mean of last 2-10 values gotten. It lowers impact of incorrect
                            # values gotten. Set beat and new values flag as true.
                            self.bpm = int(math.mean(self.bpm_buf))
                            self.new_values = True
                            self.beat = True
                        else:
                            # It has to be more than one true local maximum collected to properly count the bpm.
                            pass
                    else:
                        # If difference between saved and gotten bpm is greater by offset value set then the measure is
                        # likely faulty.
                        pass
                if self.extremum[signal] == False and self.beat:
                    # If true local minimum occurred and beat was detected before then spo2 calculation can start.
                    if self.local_min[previous] != 0:
                        # It has to be two local minimums detected to calculate the spo2 value.
                        self.dc[signal] = self.get_dc_value(signal)
                        self.ac[signal] = self.local_max[signal] - self.dc[signal]
                        self.acdc_ratio[signal] = self.ac[signal]/self.dc[signal]
                        # If the processed signal is a signal from the RED LED, then the signal from the IR LED has already
                        # been processed in this cycle.
                        if signal == red:
                            try:
                                # Try to count R factor. Leave function if ZeroDivisionError occurs.
                                self.r = self.acdc_ratio[red] / self.acdc_ratio[ir]
                            except ZeroDivisionError:
                                pass
                    else:
                        self.new_values = True
            else:
                self.new_values = True
        else:
            self.new_values = True
    return self.new_values

```

Rys. 5.13a. Funkcja odpowiedzialna za wyznaczenie poziomów pulsu i saturacji.

```

        except ZeroDivisionError:
            return
        # Count spo2 value based on equation from AN6409 maxim integrated PDF.
        spo = 104 - 17 * self.r
        if 100 > spo > 60:
            # Spo can not be higher than 100. Value lower than 60 is likely unlikely.
            self.spo_buf.append(spo)
            if len(self.spo_buf) > 10:
                # If more than 10 values collected, reject the oldest one.
                self.spo_buf = self.spo_buf[1::]
            # Calculate spo2 value as a mean of last 2-10 values gotten, and round it to two decimals.
            # Set beat flag as False.
            self.spo = round(math.mean(self.spo_buf),2)
            self.beat = False
        else:
            # Two local minimums have to be detected to correctly count spo2 value.
            pass

    return self.new_values, self.bpm, self.spo

```

Rys. 5.13b. Funkcja odpowiedzialna za wyznaczenie poziomów pulsu i saturacji.

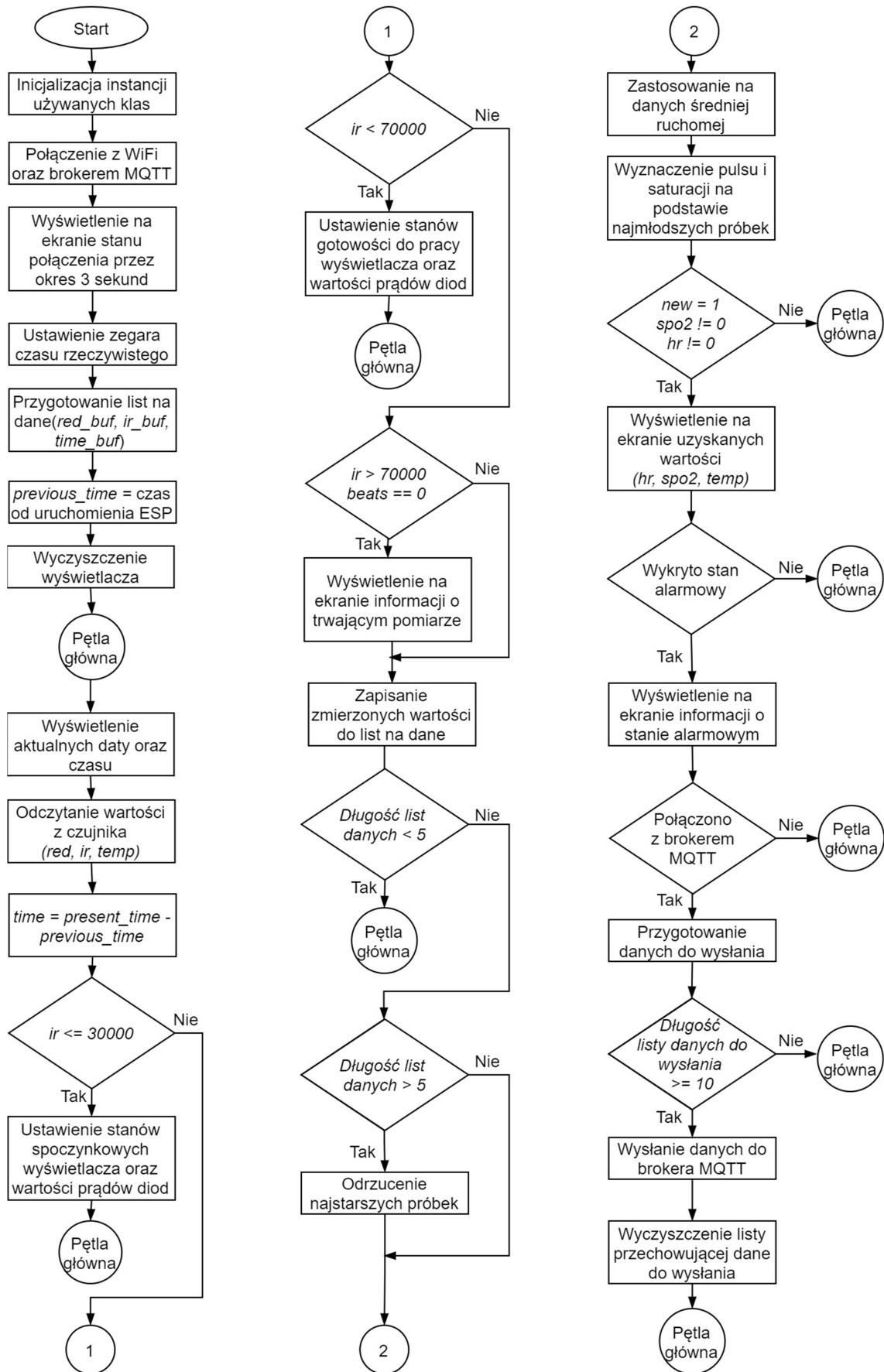
Na tym etapie dodano do urządzenia funkcję mającą za zadanie wykrywanie odchyłek mierzonych parametrów od normy. Przyjęto następujące wartości graniczne:

- dolny próg pulsu równy 50 uderzeń na minutę;
- górny próg pulsu równy 90 uderzeń na minutę;
- dolny próg saturacji równy 93 %;
- górny próg temperatury równy 37 °C.

Pomiar temperatury ma charakter pomocniczy. Zastosowanie takiej wartości granicznej powinno być optymalne biorąc pod uwagę margines błędu oraz fakt, że czujnik zwraca temperaturę matrycy. Dodatkowo warto zaznaczyć, że z uwagi na małe rozmiary wyświetlacza zwracana jest tylko wiadomość o wystąpieniu stanu alarmowego. Nie ma informacji o tym, którego parametru dotyczy.

5.4. Algorytm pracy oprogramowania zaimplementowanego na mikrokontrolerze

Po odczytem danych z czujnika i ich interpretacją oprogramowanie zaimplementowane na mikrokontrolerze zawiera szereg dodatkowych funkcji odpowiedzialnych za poszczególne działania urządzenia. Po uruchomieniu następują procesy inicjalizujące, takie jak utworzenie poszczególnych obiektów odpowiedzialnych za połączenia bezprzewodowe, obsługę urządzeń peryferyjnych, działanie wspomnianego algorytmu oraz zarządzanie danymi. Następnie zapisywany jest czas działania urządzenia w milisekundach od jego uruchomienia, względem którego liczony jest czas wykonywanych pomiarów. Funkcje takie jak wyznaczanie wartości parametrów życiowych, zmiana treści wyświetlanej na ekranie, wykrywanie stanów alarmowych oraz wysyłanie danych do brokera MQTT wywoływanie są w pętli głównej. Poglądowy schemat blokowy przedstawiający możliwe sposoby działania skonstruowanego urządzenia przedstawiony jest na rys. 5.14.



Rys. 5.14. Ideowy schemat blokowy algorytmu pracy całości oprogramowania zaimplementowanego na mikrokontrolerze.

5.5. Projekt aplikacji zbierania danych

Aplikacja zbierania danych początkowo miała mieć formę strony internetowej stworzonej przy użyciu oprogramowania Django. Jest to framework przeznaczony do tworzenia aplikacji internetowych w języku Python. Narzuca on strukturę projektu oraz oferuje wbudowane narzędzia ułatwiające zarządzanie kodem źródłowym. Natywne przeznaczenie Django różni się jednak od postawionych przez temat pracy założeń odnośnie aplikacji zewnętrznej. Stworzenie takiej aplikacji byłoby możliwe przy użyciu tej technologii, ale nie jest to rozwiązanie optymalne. Takie podejście zostało zatem szybko porzucone przez autora pracy [28].

Ostatecznie projekt aplikacji nadrzędnej wykonano w oparciu o bibliotekę tkinter. Jest ona dedykowana tworzeniu aplikacji okienkowych na komputery lokalne. Konstrukcja takiego projektu polega na umieszczaniu w oknie aplikacji widżetów mających określone działanie. Drugą ważną w procesie tworzenia aplikacji biblioteką była biblioteka o nazwie threading. Umożliwia ona programowanie asynchroniczne poprzez wywoływanie funkcji w postaci osobnych wątków. Interpreter podczas wykonywania instrukcji nie czeka na zakończenie działania takiej funkcji. Po wywołaniu wykonuje się ona w tle. Proces tworzenia wątku przedstawiony jest na rys. 5.15. Wymagane jest podanie nazwy funkcji docelowej oraz uruchomienia jej pętli. Dodatkowo konieczne jest oznaczenie takiego wątku jako daemon. W przeciwnym wypadku jego działanie uniemożliwiłoby przerwanie działania programu.

```
self.main_loop = threading.Thread(target=self.main, daemon=True)
self.main_loop.start()
```

Rys. 5.15. Tworzenie oraz uruchamianie przykładowego wątku.

Użycie wątków było zabiegiem o tyle istotnym, że dzięki temu możliwe jest korzystanie z aplikacji w sposób ciągły. Menu aplikacji pozostaje responsywne mimo równoległego działania w tle nawet kilkudziesięciu funkcji.

Aplikacja składa się z kilku plików odpowiedzialnych za poszczególne funkcje. Struktura wykonanego projektu wygląda w sposób następujący:

- main.py - główny plik projektu, uruchamiany podczas otwierania aplikacji;
- page.py - zawiera klasę odpowiedzialną za zarządzanie stronami urządzeń;
- homepage.py - zawiera klasę odpowiedzialną za operacje mające miejsce na stronie głównej;
- graph.py - odpowiedzialny za wszystkie procesy związane z wyświetlaniem przebiegów.

Całość aplikacji jest zarządzana z pliku głównego. To w nim tworzona jest okno aplikacji na bazie klasy głównej w której są zdefiniowane elementy pojawiające się na ekranie niezależnie od otworzonej strony, jak przyciski odpowiedzialne za ich wybór, dodanie lub usunięcie. Panel stworzonej aplikacji podzielony został na dwa rodzaje

stron. Pierwszy z nich to strona główna. Widoczne są na niej wartości średnie dla każdego obsługiwanej urządzenia z ostatnich 1800 pomiarów wraz z ewentualną informacją o wystąpieniu stanu alarmowego w tym czasie. Dla średniego pulsu wynoszącego około 60 uderzeń na minutę daje to czas 30 minut. Dodatkowo umieszczona została wizualizacja przedstawiająca uzyskiwane pomiary w postaci wykresu kolumnowego. Wartości na stronie głównej pochodzą z drugiego typu kart. Jest nim indywidualna strona każdego, połączonego urządzenia. Na każdej z takich stron znajduje się informacja na temat pomiarów z ostatnich 24 godzin oraz stanu alarmowego. Istnieje możliwość wyświetlenia odbieranych wartości w postaci wykresu liniowego, wyczyszczenia pola z odebranymi pomiarami oraz zmiana identyfikatora urządzenia, które jest obsługiwane przez daną kartę. Każda strona jest zaimplementowana w postaci klasy, której instancje są tworzone przy dodawaniu nowego urządzenia. Wewnątrz takich obiektów zachodzą wszystkie procesy odpowiedzialne za obsługę podłączonych urządzeń. Pętla główna danej strony uruchomiona jest jako jeden z wątków. Dodatkowo jako osobne wątki aktualizowane są dane każdego z wykresów. Pozwala to uzyskać płynniejsze działanie aplikacji. Do zdań pętli głównej każdej karty należą:

- odbiór danych z urządzenia;
- podział danych na poszczególne parametry;
- aktualizacja wyświetlanych wartości;
- zapis pomiarów w czasie rzeczywistym do pliku CSV, oraz segregacja względem daty i identyfikatora urządzenia [29-30].

5.6. Komunikacja aplikacji nadzędnej z mikrokontrolerem

Przed przystąpieniem do przesyłania danych konieczne było połączenie płytki rozwojowej NodeMCU z lokalną siecią WiFi. Do kodu źródłowego znajdującego się na mikrokontrolerze dopisano odpowiedzialną za to funkcję. W celu przesłania danych do aplikacji zewnętrznej użyto wspomnianego protokołu MQTT. Na lokalnym PC skonfigurowano własny broker przy użyciu oprogramowania Eclipse Mosquitto [31]. Następnie dodano do aplikacji funkcję, mającą za zadanie subskrypcje jednego tematu przez każdą ze stron. Jest nim "esp8266/{treść pola do wypełnienia na stronie}". Domyślnie jest to jednak "esp8266/{numer strony}". Takie rozwiązanie pozwala uniknąć wpisywania identyfikatora dla każdej ze stron osobno. Liczba urządzeń korzystających z danej sieci lokalnej jest jawną. Nie ma zatem potrzeby stosowania bardziej skomplikowanych tematów. Zdaniem autora takie rozwiązanie jest najbardziej czytelne i przejrzyste. Nic nie stoi na przeszkodzie, by treść takiego tematu zmienić na jakąkolwiek inną. Następnie skonfigurowano urządzenie w taki sposób, by publikowało mierzone wartości spakowane w paczki po 10 pomiarów. Identyfikator wykonanego prototypu to 1, więc tematem publikacji było "esp8266/1". Ze względu na prostotę dane

są przesyłane przy użyciu formatu JSON. Wiadomości są odbierane przez aplikacje periodycznie co 10 pomiarów. W skład jednej paczki wchodzą:

- data;
- czas rzeczywisty;
- czas wykonania pomiaru liczony w milisekundach od uruchomienia mikrokontrolera - ułatwia późniejszą obróbkę danych;
- wartość pulsu;
- wartość saturacji tlenem krwi;
- wartość temperatury;
- informacja na temat wystąpienia stanu alarmowego oraz jego rodzaju.

5.7. Testy wykonanego systemu

Ostatnim etapem części praktycznej tematu pracy było przeprowadzenie kompleksowych testów wykonanego systemu. W tym celu przygotowane dodatkowo 4 skrypty symulujące działanie fizycznego urządzenia. Wysyłały one wartości w różnych granicach w celu sprawdzenia działania aplikacji dla różnorodnych warunków pracy. Wszystkie dane były prawidłowo rejestrowane do plików tekstowych z zachowaniem narzuconych zasad ich segregacji. Fragment takiego pliku przedstawia tab. 5.1. Osobą badaną był mężczyzna bez stwierdzonych chorób przewlekłych mający 23 lata.

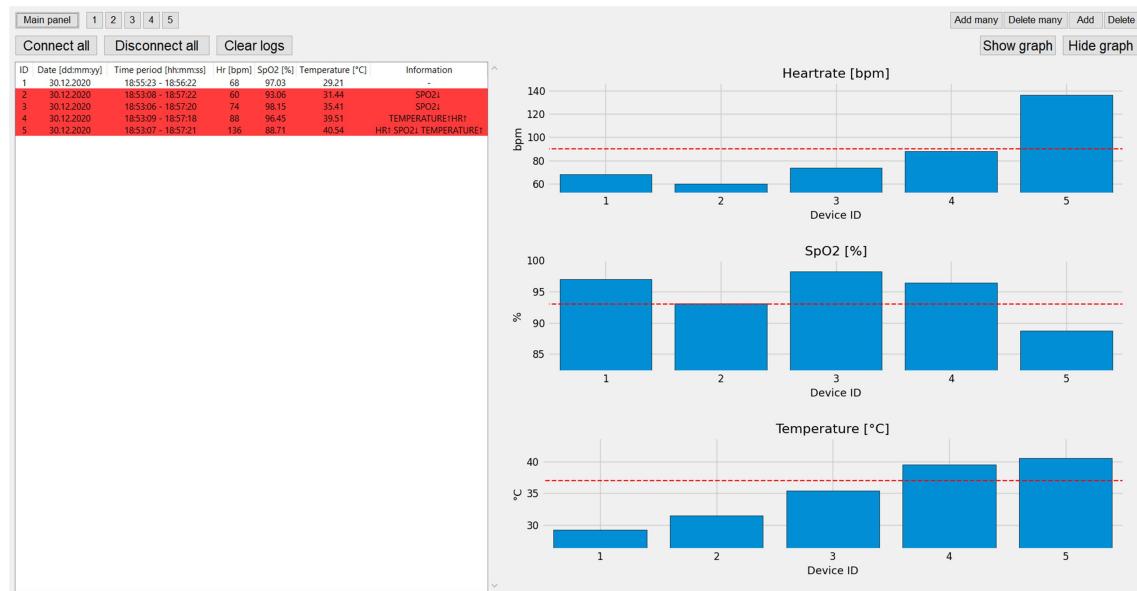
Tab.5.1. Fragment pliku z zarejestrowanymi pomiarami.

Czas [gg:mm:ss]	Puls [uderz./min]	Saturacja [%]	Temperatura [°C]	Informacja o stanie alarmowym	Czas programu [ms]
18:55:23	67	97.76	28.38	-	32718
18:55:24	67	97.66	28.19	-	33628
18:55:25	67	97.77	28.19	-	34542
18:55:26	66	97.92	28.19	-	35381
18:55:26	67	97.96	28.56	-	36217
18:55:27	68	97.96	28.62	-	37056
18:55:28	68	97.84	28.56	-	37895
18:55:29	68	97.78	28.81	-	38807
18:55:30	68	97.62	28.75	-	39647

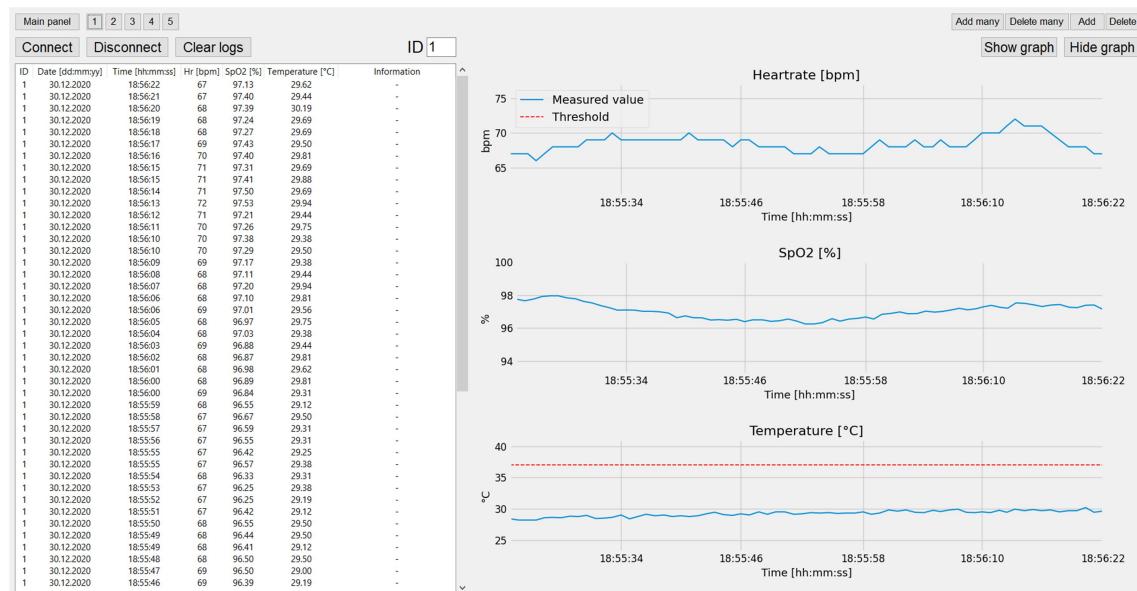
Wartości zwracane przez urządzenie pomiarowe wydają się być prawidłowe. Punktrem odniesienia było kilka aplikacji na smartfon oraz opaska sportowa znanego producenta. Wszystkie te urządzenia zwracały w danych warunkach takie same wartości. Często to urządzenie skonstruowane w temacie pracy zwracało pomiar szybciej niż pozostałe. Wynika to najprawdopodobniej z faktu, że do wyznaczenia wartości potrzebne jest wykrycie tylko trzech uderzeń serca. Zarówno opaska jak i aplikacja na telefon potrzebowały dokonać serii pomiarów trwającej około 20 sekund by określić poziom badanych parametrów

Jak widać temperatura podczas wykonywania pomiaru wynosiła około 28°C. Wynika to z faktu, że są to wartości uzyskane chwilę po zamocowaniu czujnika na nadgarstku. W efekcie matryca nie zdążyła nabrać odpowiedniej temperatury. Podczas wykonywania pomiaru trwającego około 15 minut temperatura sięgała 35,5°C. Na tej podstawie została dobrana jej górną granicę.

Stworzona aplikacja jest w stanie z powodzeniem obsłużyć większą ilość urządzeń. Zastosowane funkcje działają tak jak planowano. Na wykresie znajdują się zaznaczone wartości graniczne gdy wartość zmierzona się do nich zbliża. Informacje o alarmach wyświetlały się prawidłowo. Panel aplikacji przedstawiają rys. 5.16 oraz 5.17.



Rys. 5.16. Widok strony głównej stworzonej aplikacji podczas pracy.



Rys. 5.17. Widok strony obsługującej urządzenie o identyfikatorze równym 1.

6. Podsumowanie

Celem zaprezentowanej pracy inżynierskiej było wykonanie systemu mającego za zadanie pomiar podstawowych parametrów życiowych. Były nimi puls, saturacja krwi tlenem oraz temperatura ciała. Udało się z powodzeniem spełnić wszystkie założenia pracy.

Urządzenie zwraca prawidłowe wartości zarówno dokonując pomiaru z powierzchni palca, jak i będąc przymocowanym do nadgarstka. Jest to niewątpliwym sukcesem. Sam producent twierdzi, że pomiar wymienionych wyżej parametrów z poziomu nadgarstka przy użyciu jedynie światła czerwonego i podczerwonego może być trudny do zrealizowania [8]. Wartości są wyświetlane lokalnie oraz wysyłane w czasie rzeczywistym do aplikacji zewnętrznej. Działaniem aplikacji, które nie było jasne na początku projektu oprogramowania jest możliwość obsługi wielu urządzeń. Proces implementacji takiej funkcji również zakończono sukcesem. Liczba obsługiwanych urządzeń jest nieograniczona. Po przekroczeniu 40 kart pojawia się pole do bezpośredniego wpisania numeru strony, która ma być otwarta. Jedynym ograniczeniem jest moc obliczeniowa komputera na którym uruchomiona jest aplikacja. Warto zaznaczyć, że w takiej sytuacji możliwe jest zrezygnowanie z wizualizacji pomiarów dla poszczególnych urządzeń. Rysowanie wykresów w czasie rzeczywistym jest procesem najbardziej wymagającym pod względem mocy. Rezygnacja z tej funkcji podczas odbierania danych od wielu urządzeń znacznie poprawia płynność działania oprogramowania.

Wykonany prototyp ma spory potencjał do dalszego rozwoju. W pierwszej kolejności można rozważyć rozszerzenie projektu płytki PCB o wewnętrzne źródło zasilania. Zrealizować to można poprzez zamontowanie koszyka na baterie litowe oraz przetwornicy w celu dostarczenia napięcia o odpowiedniej wartości. Dodatkowo odpowiednim rozwiązaniem byłoby rozmieszczenie elementów na dwóch osobnych płytach drukowanych składających się na dwie platformy urządzenia. Proces ten miałby na celu zmniejszenie jego rozmiarów. Całość może być umieszczona w obudowie umożliwiającej montaż paska. Stworzenie takiej obudowy mogłoby się odbyć przy użyciu drukarki 3D. W efekcie takie urządzenie powinno przypominać swoją budową zegarek sportowy. Dodatkowo można podjąć próby poprawy stworzonego algorytmu do interpretacji wartości zwracanych przez czujnik oraz rozważyć wymianę samego czujnika na nowszy model, posiadający diodę emitującą światło zielone. Miałoby to zwiększyć poprawność wykonywanych pomiarów. Wszystkie te operacje nie są jednak wymagane, gdyż zaprezentowane urządzenie już na takim etapie konstrukcji spełnia postawione mu zadania w sposób zadowalający.

7. Bibliografia

Opracowania książkowe

- [1] Bogusz J. *Lokalne interfejsy szeregowe* Warszawa: wydawnictwo BTC, 2004
- [2] Jones D. *PCB Design Tutorial*, Revision A - June 29th 2004

Dokumenty

- [3] Espressif Systems: *ESP8266EX Datasheet*, version 6.6, 2020
- [4] Maxim Integrated Products: *High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health*, 19-7740; Rev 1; 10/18
- [5] WaveShare: *1.3inch OLED User Manual*, Rev .4, May 15th 2015
- [6] NXP Semiconductors: *I²C-bus specification and user manual*, Rev.6 - 4 April 2014
- [7] Klemiato M. *Instrukcja do ćwiczeń z przedmiotu Komputerowe projektowanie układów sterowania - Proste filtry cyfrowe*, AGH, Katedra Automatyki i Robotyki
- [8] Maxim Integrated Products: *Recommended Configurations and Operating Profiles for MAX3010/MAX30102 EV Kits*, UG6409; Rev 0; 3/18

Źródła internetowe

- [9] *Parametry życiowe*,
http://mediq.edu.pl/index.php?option=com_content&view=article&id=9&Itemid=155 (dostęp 19.12.2020).
- [10] *ESP8266 Pinout Reference: Which GPIO pins should you use?*
<https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/> (dostęp 20.12.2020).
- [11] *GPIO — ESP Easy 2.1-beta1 documentation*,
<https://espeasy.readthedocs.io/en/latest/Reference/GPIO.html> (dostęp 20.12.2020).
- [12] *NodeMCU ESP8266 Pinout, Specifications, Features & Datasheet*,
<https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet> (dostęp 20.12.2020).
- [13] *Introduction to NodeMcu V3 - The Engineering Projects*,
<https://www.theengineeringprojects.com/2018/10/introduction-to-nodemcu-v3.html> (dostęp 19.10.2020).
- [14] *Amazon.com: Heart Rate Sensor Module MAX30102 Pulse Detection Blood Oxygen Contraction Compatible for Arduino STM32*,
<https://www.amazon.com/MAX30102-Detection-Concentration-Compatible-Arduino/dp/B07ZQNC8XP> (dostęp 21.12.2020).
- [15] *MQTT - najłatwiejszy sposób przesyłania danych do chmury!*,
<https://moxa.elmark.com.pl/2019/09/19/mqtt-iothinx/> (dostęp 23.12.2020).

- [16] *Internet of Things Cloud MQTT Brokers*,
<https://svitla.com/blog/internet-of-things-cloud-mqtt-brokers> (dostęp 23.12.2020).
- [17] *JSON*, <https://www.json.org/json-en.html> (dostęp 23.12.2020).
- [18] *XML Introduction*, https://www.w3schools.com/xml/xml_whatis.asp
(dostęp 23.12.2020).
- [19] *3.9.1 Documentation*, <https://docs.python.org/3/> (dostęp 25.12.2020).
- [20] *Kompilator a interpreter*, <http://student.agh.edu.pl/~michalus/> (dostęp 25.12.2020).
- [21] *PO Obiektowe modelowanie dziedziny - Studia Informatyczne*,
http://wazniak.mimuw.edu.pl/index.php?title=PO_Obiektowe_modelowanie_dziedziny%C2%A3.C4.99.C5.9Bciowy_model_dziedziny_dla_gry_w_Monopol (dostęp 28.12.2020).
- [22] *MicroPython - Python for microcontrollers*, <http://micropython.org/>
(dostęp 25.12.2020).
- [23] *Features - PyCharm*, <https://www.jetbrains.com/pycharm/features/> (dostęp 26.12.2020).
- [24] *EasyEDA - Symulator obwodów i konstrukcja PCB Online*, <https://easyeda.com/pl>
(dostęp 28.12.2020).
- [25] *Download PyCharm: Python IDE for Professional Developers by JetBrains*,
<https://www.jetbrains.com/pycharm/download/#section=windows> (dostęp 15.05.2020).
- [26] *1. Getting started with MicroPython on the ESP8266 - MicroPython 1.1 documentation*
<http://docs.micropython.org/en/latest/esp8266/tutorial/intro.html#intro>
(dostęp 08.09.2020).
- [27] *Równanie prostej przechodzącej przez dwa punkty*, <https://www.matemaks.pl/rownanie-prostej-przechodzacej-przez-dwa-punkty.html> (dostęp 23.10.2020).
- [28] *Czym jest Django? - HonKit*, <https://tutorial.djangogirls.org/pl/django/>
(dostęp 30.12.2020).
- [29] *TkDocs Home*, <https://tkdocs.com/index.html> (dostęp 23.11.2020).
- [30] *threading - Thread-based parallelism - Python 3.9.1*,
<https://docs.python.org/3/library/threading.html> (dostęp 25.11.2020).
- [31] *Download | Eclipse Mosquitto*, <https://mosquitto.org/download/> (dostęp 09.12.2020).
- [32] *Pull up resistors - ESP8266 Developer Zone*,
<https://bbs.espressif.com/viewtopic.php?t=1079> (dostęp 06.01.2020).