INSTYTUT AUTOMATYKI
I INŻYNIERII INFORMATYCZNEJ
WYDZIAŁ ELEKTRYCZNY
POLITECHNIKA POZNAŃSKA

INŻYNIERSKA PRACA DYPLOMOWA

SYSTEM WSPOMAGAJĄCY KIEROWANIE POJAZDEM Z UŻYCIEM ZŁĄCZA DIAGNOSTYCZNEGO

Mateusz BARTOSZ

Promotor:
Dr inż. Konrad URBAŃSKI

Poznań 2017

Spis treści

Streszczenie

Abstract

1 Wstęp

1.1 Wybór tematu

Głównym czynnikiem decydującym o wyborze tematu było zainteresowanie rozwiązaniami elektronicznymi stosowanymi we współczesnej motoryzacji oraz chęć podjęcia próby zbudowania układu opartego o własną koncepcję pracującego jako komputer pokładowy w samochodzie osobowym. Kolejnym czynnikiem było umożliwienie cyklicznego badania i kontroli parametrów pracy poszczególnych układów pojazdu, w celu uniknięcia lub wczesnego wykrycia potencjalnych usterek. Dodatkową motywacją była chęć zbudowania układu, który mógłby być w przyszłości praktycznie wykorzystywany w samochodach niewyposażonych w wbudowany komputer pokładowy.

1.2 Cel i zakres pracy

Celem pracy było zaprojektowanie oraz wykonanie układu przyłącza do gniazda diagnostycznego w samochodzie osobowym Seat Cordoba III oraz zbudowanie interfejsu użytkownika umożliwiającego wizualizację odczytywanych parametrów, kontrolę wartości granicznych, a także przechowanie ich w celach dalszej diagnostyki. Dodatkowym celem było określenie możliwości wykorzystania odbieranych parametrów do opracowania algorytmów umożliwiających wspomaganie kierującego pojazdem, aby zoptymalizować jazdę, zwiększyć bezpieczeństwo podróży oraz zminimalizować ryzyko wystąpienia usterek.

1.3 Założenia i wymagania

Założeniem pracy jest opracowanie kompleksowego układu umożliwiającego odczytywanie, wizualizację oraz kontrolę parametrów odbieranych przez złącze diagnostyczne w samochodzie osobowym Seat Cordoba III, a także określenie możliwości wykorzystania tych parametrów do opracowania algorytmów wspomagających prowadzenie pojazdu.

2 Zagadnienia wprowadzające

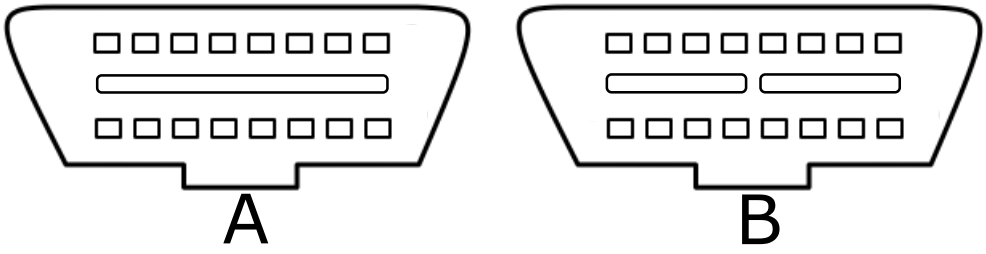
We współczesnej motoryzacji wyraźnie można zauważyć tendencję automatyzacji procesu prowadzenia pojazdu oraz kontroli stanu jego parametrów. W samochodach dostępnych na rynku można spotkać bardzo wiele różnych protokołów komunikacyjnych. Cześć z nich służy do komunikacji pomiędzy urządzeniami wewnętrznymi pojazdu, inne do komunikacji z użytkownikiem, w celu zwiększenia komfortu jazdy, a jeszcze inne wykorzystywane są w diagnostyce stanu poszczególnych układów samochodu. Te ostatnie, wyprowadzone są do gniazda diagnostycznego(ang. On-Board Diagnosdic - OBD). W zależności od producenta, modelu oraz roku produkcji pojazdu do dyspozycji są różne protokoły. Umożliwiają one między innymi odczytywanie aktualnych wskazań niektórych czujników, kontrolę zużywania się elementów eksploatacyjnych czy detekcję błędów silnika. W niniejszym rozdziale omówione zostało złącze diagnostyczne w wersji drugiej(ODB2) wraz z udostępnianymi przez nie protokołami komunikacyjnymi.

2.1 Złącze diagnostyczne

Historia złącza diagnostycznego używanego w motoryzacji sięga końcówki lat sześćdziesiątych dwudziestego wieku. Pierwsze komputery pokładowe wprowadzone zostały w samochodach marki Volkswagen w 1968 roku. Dziesięć lat później za sprawą marki Nissan komputery pokładowe pojawiły się w pojazdach konsumenckich. Kolejnym krokiem było wprowadzenie protokołu ALDL przez General Motors w 1980 roku. Był to pierwszy standard zbliżony do obecnie stosowanego w gniazdach OBD2. Występował w trzech wersjach: dwunasto, dziesięcio i pięciopinowej. Pierwsze wersje były jednokierunkowe i umożliwiały przesyłanie 160 bodów danych. W późniejszych wersjach wprowadzono dwukierunkową transmisję danych o zwiększono szybkości transmisji do 8192 bodów. W 1991 roku agencja California Air Resources Board zażądała, aby każdy nowy pojazd sprzedawany w Kalifornii posiadał wyprowadzenie diagnostyczne. Stało się to impulsem do opracowania i wprowadzenia standardu OBD-I, choć nazwa ta została wprowadzona dopiero po opracowaniu kolejnego standardu OBD w wersji drugiej. Złącze ODB-I nie zostało ściśle ustandaryzowane i każdy producent samochodów mógł wykonać je w swojej wersji. Główną motywacją do wprowadzenia tego standardu było zachęcenie producentów pojazdów do zaprojektowania systemów kontroli emisji spalin. Często spotykana wersja tego złącza umożliwiała odczytywanie kodów błędów poprzez analizę migania diody znajdującej się przy złączu. Miganie diody reprezentowało odpowiednią liczbę dwucyfrową, która była interpretowana jako odpowiedni kod błędu pojazdu. W 1994 roku na bazie poprawionej i uzupełnionej specyfikacji OBD-I powstał standard ODB-II. Jest to najpopularniejszy, aktualnie używany system diagnostyki samochodowej. Od roku 1996 wszystkie nowe samochody sprzedawane w Stanach Zjednoczonych muszą być wyposażone właśnie w gniazdo ODB w wersji drugiej. W 2001 roku standard OBD-II pod nazwą EOBD został wprowadzony jako obowiązkowy w samochodach benzynowych produkowanych w Unii Europejskiej, a w 2003 roku również w samochodach z silnikami wysokoprężnymi.

2.1.1 Złącze diagnostyczne OBD-II

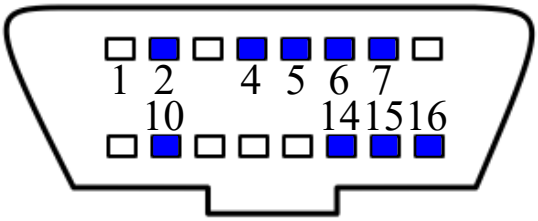
Złącze diagnostyczne OBD w wersji drugiej jest aktualnie stosowanym w motoryzacji wyprowadzeniem umożliwiającym wykrywanie błędów poszczególnych układów pojazdu. Opis budowy tego złącza zawarty jest w normie SAE J1962[[9]]. Standard OBD-II zawiera specyfikację dotyczącą budowy złącza, a także opis udostępnianych przez nie protokołów komunikacyjnych. Zgodnie z normą, złącze występuje w dwóch wersjach: A oraz B, oba szesnastopinowe (2x8), żeńskie. Na Rys. 2.1 przedstawiono budowę obu typów złączy.



Rys. 2.1: Porównanie złącza diagnostycznego w wersji A oraz B.

Złącze typu A stosowane jest w pojazdach wyposażonych w akumulator o napięciu 12V, natomiast złącze typu B w pojazdach wyposażonych w akumulator o napięciu 24V.

Norma SAE J1962 definiuje protokoły komunikacyjne udostępniane przez złącze diagnostyczne. Dostępność poszczególnych protokołów komunikacyjnych może być różna w zależności od producenta pojazdu oraz roku produkcji. Na Rys. 2.2 oraz w tabeli 2.1 przedstawione zostały potencjalnie dostępne protokoły komunikacyjne wraz z numerem pinu, do którego powinny być podłączone.



Rys. 2.2: Opis wyprowadzeń złącza diagnostycznego. Wykorzystywane wyprowadzenia zostały zaznaczone kolorem niebieskim oraz odpowiednim numerem. Opis wyprowadzeń znajduje się w Tab 2.1

W złączu diagnostycznym udostępnionych jest 5 różnych protokołów. Dodatkowo rozróżnione są dwie masy: masa podwozia oraz masa sygnałowa. W praktyce najczęściej wyprowadzenia te są ze sobą zwarte. Zazwyczaj w pojazdach udostępniony jest tylko jeden z opisanych poniżej protokołów.

Tabela 2.1: Opis wyprowadzeń złącza diagnostycznego OBD-II.

Numer wyprowadzenia	Przeznaczenie wyprowadzenia
2	Linia dodatnia protokołu SAE J1850 PWM oraz VPW
4	Masa podwozia
5	Masa sygnałowa
6	Linia wysoka magistrali CAN
7	Linia K protokołu ISO 9141-2 oraz ISO 14230-4
10	Linia ujemna protokołu SAE J1850 PWM
14	Linia niska magistrali CAN
15	Linia L protokołu ISO 9141-2 oraz ISO 14230-4
16	Zasilanie 12V/4A dla złącza typu A 24V/2A dla złącza typu B

2.1.2 Protokoły komunikacyjne dostępne w złączu diagnostycznym

Zgodnie z normą SAE J1962 do złącza diagnostycznego doprowadzone może być pięć różnych magistral danych które udostępniają następujące protokoły komunikacji:

- SAE J1850 PWM,
- SAE J1850 VPW,
- CAN,
- ISO 9141,
- ISO 14230 - KWP2000.

SAE J1850 to magistrala wykorzystywana głównie w samochodach marek amerykańskich. Występuje w dwóch odmianach. Pierwsza z nich to J1850 PWM (Pulse width modulation). Jest to technologia komunikacji wykorzystująca modulację szerokością impulsu do kodowania logicznych stanów (0 oraz 1). Każdy bit zaczyna się zmianą stanu niskiego na wysoki. Następnie, w zależności od wysyłanego bitu danych, zmienia się wypełnienie impulsu, dla logicznego „0” stan wysoki utrzymuje się przez 2/3 czasu impulsu, natomiast dla „1” stan wysoki to 1/3 impulsu. Czas trwania impulsu jest stały. Magistrala jest dwuprzewodowa, kodowanie różnicą pomiędzy potencjałami obu kanałów. Prędkość transmisji wynosi 41,6 kb/s. Głównym koncernem wykorzystującym magistralę J1850 PWM jest Ford.

Drugą odmianą magistrali J1850 jest wersja VPW (variable pulse width). W tej wersji każdy nowy bit sygnalizowany jest za pomocą zmianą stanu. W związku z tym, aby rozpoznać logiczne zmiany, w zależności od wartości poprzedniego bitu kolejny bit kodowany jest za pomocą długości impulsu, nie zaś stanem wysokim lub niskim. W związku z tym, że długości bitów są różne, nie można jednoznacznie określić prędkości transmisji, przyjmowane jest średnio 10,4 kbit/s. Jest to magistrala jednoprzewodowa, używana przez General Motors.

Magistrala CAN (Controller Area Network), zastosowana pierwszy raz przez koncern Mercedes w roku 1992, umożliwia komunikację pomiędzy urządzeniami oraz mikrokontrolerami bez używania urządzenia nadrzędnego. Początkowo używany w branży motoryzacyjnej, został również z powodzeniem wprowadzony do automatyki przemysłowej. Transmisja sygnału odbywa się za pomocą dwóch przewodów, na każdym występują dwa stany. Jeden stan jest zgodny (dla linii wysokiej oraz niskiej potencjał jest ten sam), oznacza logiczne 0, drugi stan sygnalizowany jest za pomocą wysokiego stanu na linii „High”, a niskim na linii „Low”.

Tańszym odpowiednikiem CAN stosowanym we wcześniejszych pojazdach jest magistrala oparta na normie ISO 9141, LIN (Local Interconnect Network). Komunikacja opiera się na architekturze master/slave. Komunikacja wywoływana jest poprzez zapytanie wysyłane przez mastera, przez co nie ma konieczności wykrywania kolizji. Jest to sieć stworzona jednoprzewodowo lub dwuprzewodowo, w zależności od zastosowania, pozwalająca na podłączenie maksymalnie 16 urządzeń do jednej sieci. Maksymalna prędkość przesyłania to 20 kbit/s.

2.2 Struktura zapytań złącza diagnostycznego

Głównym zastosowaniem złącza diagnostycznego miało być nadzorowanie systemów, które mają znaczenie w emisji spalin. Specyfikacja na temat transmisji danych diagnostycznych dostępne są w normie ISO 15031 oraz bliźniaczej SAE J1979. Jako systemy podległe normy dopuszczają sieć K-Line z KWP 200 oraz jej poprzednika ISO 9141-2 CARB, KWP 2000 z CAN oraz stosowaną głównie w pojazdach amerykańskich producentów magistralę SAE J1850 w wersji VPW i PWM.

Komunikacja z OBD odbywa się w postaci żądanie - odpowiedź(and request - response) i ma formę protokołu KWP 2000 [1]. Komunikat żądania podzielony jest na trzy części. Pierwszą z nich stanowi identyfikator serwisu (ang. Service ID - SID), który w normie amerykańskiej nazywane jest trybem testowym lub po prostu trybem (ang. test modes). Tryby o numerach 01-09 są nazywane trybami podstawowymi, natomiast tryby powyżej numeru 09 trybami rozszerzonymi[2]. W tabeli 2.2 przedstawione zostały dostępne podstawowe identyfikatory serwisu.

Tabela 2.2: Tryby pracy złącza diagnostycznego[1].

SID (hex)	Opis
01	Pytania o wartości danych w sterowniku. Dane są identyfikowane za pomocą kodu PID
02	Pytanie o dane dotyczące warunków otoczenia dla kodów usterek zapisanych w pamięci.
03	Odczytywanie z pamięci błędów mających wpływ na emisję spalin zakwalifikowanych jako „wykryte bezzasadnie”.
04	Kasowanie pamięci błędów (kodów błędów, warunków otoczenia, statusu różnych testów)
05	Nadzorowanie sondy lambda
06	Nadzorowanie katalizatorów, układów wylotowych, ogrzewania sondy lambda i katalizatorów, układów wtrysku paliwa i zapłonu.
07	Podobnie jak dla SID=03, tutaj są odsyłane tylko błędy, które zostały zakwalifikowane jako „wykryte przejściowo”.
08	Test poziomu paliwa
09	Wyświetlanie informacji o pojeździe

Opis trybów rozszerzonych można znaleźć w literaturze [2].

Kolejną częścią wysyłanego żądania jest identyfikator parametru - PID (ang. Parameter Identifier). Zajmuje on również 1 bajt oraz kodowany jest szesnastkowo. Ostatnią częścią zapytania są parametry żądania, które mogą zająć maksymalnie 6 bajtów. Najczęściej jednak wykorzystuje się żądania bez dodatkowych parametrów.

W złączu diagnostycznym pojazdu nie muszą być udostępnione wszystkie tryby pracy, a także nie muszą być udostępnione wszystkie kody zapytań z udostępnionego trybu. Ogólna struktura kodów zapytań ma postać:

- AA BB CC CC CC CC CC,

gdzie AA to numer trybu testowego , BB to identyfikator z wybranego trybu natomiast CC dodatkowe parametry. Kody BB oraz CC są opcjonalne, natomiast kod AA musi być zawarty w każdym zapytaniu.

Struktura odpowiedzi złącza diagnostycznego na zadany kod jest podobna do zapytania. Pierwszy bajt to kod SID zapytania, ale po wykonaniu (AA|40), czyli po dodaniu do kodu SID zapytania liczby 40 kodowanej szesnastkowo. Drugi bajt jest powtórzeniem identyfikatora parametru żądania. Następnie w odpowiedź może się znaleźć do sześciu bajtów parametrów odpowiedzi. Ogólna struktura kodów odpowiedzi ma zatem postać:

- AA|40 BB CC CC CC CC CC.

Ilość odebranych danych, a także ich interpretacja jest różna w zależności od identyfikatora parametru. Strukturę oraz interpretację odpowiedzi można przeanalizować na przykładzie. Wysyłając do złącza diagnostycznego żądanie 01 00 w odpowiedzi zwrócona zostanie informacja o udostępnionych identyfikatora parametru trybu 01 z zakresu 01-20:

- 01 00
- 41 00 BE 1F B8 10

Odpowiedź - po usunięciu z niej nagłówka - należy zapisać w postaci binarnej, w której wartość 1 oznacza że dany kod jest udostępniony. Sposób dekodowanie odpowiedzi przedstawiony zostały w tabeli 2.3

Tabela 2.3: Interpretacja odpowiedzi na kod zapytania 01 00

Bajt odpowiedzi (hex)	B				E				.	.
Postać binarna	1	0	1	1	1	1	1	0	.	.
Kod PID	01	02	03	04	05	06	07	08	.	.

Wartości identyfikatora parametru powtarzające się co wartość 20 w kodzie szesnastkowym(00, 20, 40) zarezerwowane są do odczytywania udostępnionych przez dany sterownik kodów PID i dekoduje się je zawsze tak jak przedstawiono w przykładzie.

Najczęściej jednak żadnie dotyczy konkretnego parametru pojazdu. Odpowiedź otrzymana na żądanie jest wtedy dekodowana na podstawie wzoru:

$$x_{fiz} = (x_{hex}) \times (2^n - 1)^{-1} \times (V_{max} - V_{min}) + V_{min}, \tag{2.2.1}$$

gdzie: x_{fiz} - obliczona wartość fizyczna, x_{hex} - odebrana wartość w postaci szesnastkowej, n - liczba bitów odpowiedzi, V_{max} - wartość maksymalna zakresu zmiennej, V_{min} - wartość minimalna zakresu zmiennej.

2.3 Struktura kodów błędów

Podstawową funkcjonalnością złącza diagnostycznego jest raportowanie błędów poszczególnych układów za pomocą diagnostycznych kodów błędów(ang. Diagnostic Trouble Codes - DTC). Są one zapisywane w sterowniku jako 16-bitowe liczby heksadecymalne, a następnie przetwarzane przez tester diagnostyczny to postaci 5-znakowego słowa alfanumerycznego. Aby uzyskać informację o zapisanych kodach należy wysłać zapytanie 03 do złącza diagnostycznego, ale najpierw zdefiniować ile kodów jest obecnych w pamięci. W tym celu należy wysłać komendę:

- 01 01.

Kod odpowiedzi powinien mieć postać zbliżoną do:

- 41 01 81 07 65 04.

Wartości 41 01 to nagłówek odpowiedzi. Właściwa odpowiedź zaczyna się od dwóch pary bajtów 81, w których zapisana jest informacja o ilości przechowywanych kodów błędów pojazdu. Wartość 81 to liczba zapisana w postaci szesnastkowej, której wartość w systemie dziesiętnym to 129. Nie jest to jednak wprost liczba kodów błędów. W kodzie tym zawarta jest również informacja o stanie diody kontrolnej silnika, który zapisany jest na najbardziej znaczącym bicie, co oznacza, że odczytaną wartość należy pomniejszyć o 128 lub 80 w systemie szesnastkowym. Wynika z tego że w zaprezentowanym przykładzie przechowywany jest tylko jeden kod błędu. Podany przykład ilustruję sytuację, w której pojazd udostępnia tylko jeden moduł, który raportuje kody błędów. Jeżeli takich modułów byłoby więcej w odpowiedzi na instrukcję 01 01 wysłana zostanie odpowiedź podobna do przedstawionej dla każdego modułu. W celu sprawdzenia z którego modułu wysłana została odpowiedź należy włączyć wysyłanie nagłówka odpowiedzi.

Po ustaleniu liczby przechowywanych błędów można przejść do odczytywania ich kodów poprzez wysłanie instrukcji 03. Odpowiedź powinna mieć postać zbliżoną do:

- 43 01 33 00 00 00 00.

Wartość 43 to nagłówek odpowiedzi na kod 03. Kolejnych 6 bajtów powinno być odczytywanych parami. Każda para reprezentuje jeden kod błędu. W powyższym przypadku przechowywany jest tylko jeden kod 0133, natomiast zgodnie z standardem SAE odpowiedź została uzupełniona bajtami 00, które nie są interpretowane jako kody diagnostyczne. Pierwsza cyfra w odczytanym kodzie zawsze przechowuje zakodowaną informację o źródle błędu. Sposób dekodowania źródła wystąpienia błędu przedstawiony został w tabeli 2.4.

Tabela 2.4: Kodowanie źródła błędu [4].

Pierwsza cyfra odpowiedzi	**Zakodowany identyfikator**	**Opis**
0	P0	Kody układu napędowego zdefiniowane w normie SAE
1	P1	Kody zdefiniowane przez producenta
2	P2	Kody zdefiniowane w normie SAE
3	P3	Kody zdefiniowane przez producenta
4	C0	Kody podwozia zdefiniowane w normie SAE
5	C1	Kody zdefiniowane przez producenta
6	C2	Kody zdefiniowane przez producenta
7	C3	Kody zdefiniowane w normie SAE
8	B0	Kody nadwozia zdefiniowane w normie SAE
9	B1	Kody zdefiniowane przez producenta
A	B2	Kody zdefiniowane przez producenta
B	B3	Kody zdefiniowane w normie SAE
C	U0	Kody sieci zdefiniowane w normie SAE
D	U1	Kody zdefiniowane przez producenta
E	U2	Kody zdefiniowane przez producenta
F	U3	Kody zdefiniowane w normie SAE

Kod błędu z przykładowej odpowiedzi to 0133. Pierwszą cyfrę należy zastąpić wartością z tabeli, a pozostałe trzy cyfry pozostawić bez zmiany. W przedstawionym przykładzie daje to wartość P0133 co oznacza wystąpienie błędu wolnej odpowiedzi obwodu czujnika tlenu. Analogicznie po otrzymaniu odpowiedzi w postaci D016 odczytanym kodem jest U1016. Znaczenie wszystkich kodów błędów można odnaleźć w dokumentacji normie [10] lub w literaturze [4].

2.4 Interfejs komunikacyjny ELM 327

Do komunikacji ze złączem diagnostyczny można wykorzystać popularny interfejs ELM327 firmy ELM Electronics. Dostępny jest w dwóch wariantach: RS232 lub bluetooth. Jądrem urządzenia jest mikroprocesor PIC18F2480 firmy Microchip Technology. W niniejszej pracy wykorzystany został urządzenie udostępniające komunikację bluetooth pokazane na Rys. 2.3.

Rys. 2.3: ELM 327 w wersji bluetooth.

Interfejs ELM327 umożliwia współpracę ze wszystkimi udostępnionymi w złączu diagnostycznym protokołami. Komunikacja modułu z komputerem odbywa się poprzez wirtualny port szeregowy. W celu umożliwienia poprawnej komunikacji należy skonfigurować port zgodnie z następującymi parametrami:

- prędkość transmisji : 15200 bit/s,
- liczba bitów danych : 8,
- parzystość : żaden,
- liczba bitów stopu : 1.

Z modułem można się komunikować za pomocą komend AT zdefiniowanych w dokumentacji producenta [11]. Żeby żądanie zostało wysłane do interfejsu każdą instrukcję należy zakończyć znakiem powrotu karetki. Przykładowa komenda z odpowiedzią do wyłączania echa instrukcji w odpowiedzi wraz ma następującą postać:

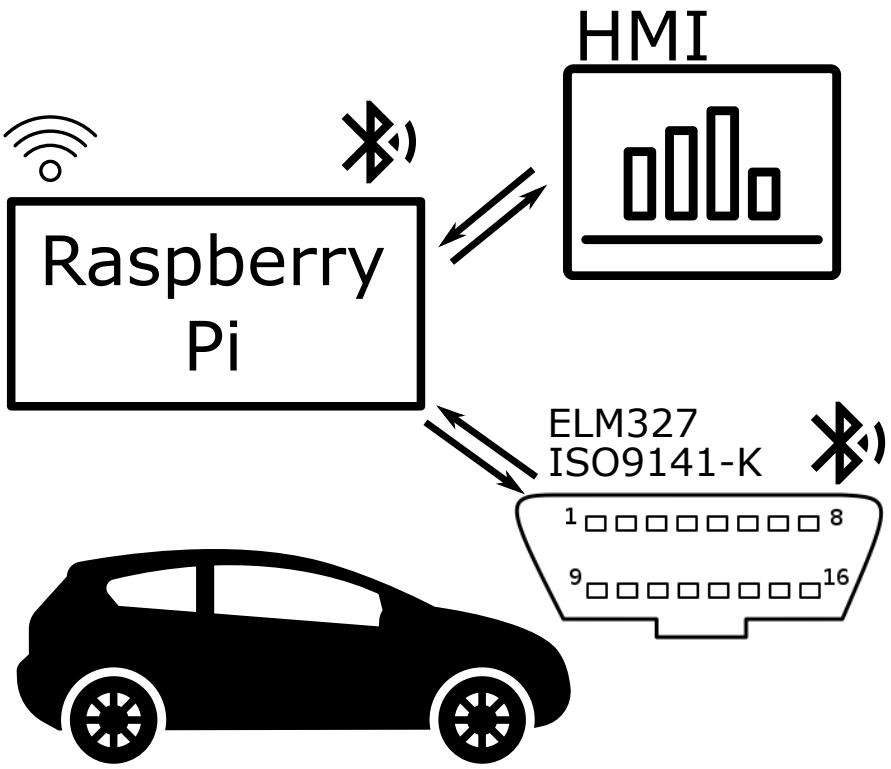
- ATE0
- ATE0OK

3 Struktura projektu

Zrealizowany projekt układu do komunikacji ze złączem diagnostycznym składa się z trzech głównych modułów:

- Moduł komunikacji ze złączem diagnostycznym
- Oprogramowanie umożliwiające komunikację z komputera z wymienionym wyżej modułem oraz zapisywanie danych pomiarowych
- Oprogramowanie wizualizujące odczytane parametry

Struktura zbudowanego układu przedstawiona jest na Rys. 3.1.



Rys. 3.1: Schemat układu. Opis w tekście.

W pierwszej fazie projektu planowano wykonać autorski moduł do komunikacji ze złączem diagnostycznym. W tym celu zaprojektowany został układ oparty o mikroprocesor STM32F103CBT6 opisany w rozdziale 4. Niestety ze względu na zbyt wysoką cenę wykonania tylko jednej sztuki prace zakończono na projekcie schematu elektrycznego oraz obwodu drukowanego. Jako zamiennik autorskiego układu w pracy wykorzystano popularne urządzenie ELM327 v2.1 udostępniające komunikację bezprzewodową bluetooth.

Jako moduł główny w projekcie zdecydowano się wykorzystać komputer Raspberry Pi 3 ze względu na jego dobry stosunek jakości do ceny. Przy wyborze tego układu ważna również była możliwość podłączenia zewnętrznego ekranu, a także dostępność komunikacji bluetooth. Oprogramowanie obsługujące funkcjonalność napisane zostało w języku Java SE 8 z wykorzystaniem dodatkowej biblioteki pi4j umożliwiającej dostęp dostęp do funkcji wbudowanych komputera Raspberry Pi, takich jak komunikacja bluetooth, czy zarządzanie portami GPIO. Dodatkowo oprogramowanie wykorzystuje bibliotekę jbidibc-rxtx-2.2 umożliwiającą wykorzystanie portów szeregowych komputera, które były niezbędne do komunikacji z urządzeniem ELM327. Dokładny opis funkcjonalności stworzonego oprogramowania znajduje się w rozdziale 5.1.

Ostatni moduł - interfejs graficzny użytkownika - stworzony został w języku Java SE 8 z wykorzystaniem nowego frameworku do tworzenia zaawansowanych interfejsów graficznych JavaFX 8. Interfejs użytkownika dostosowany został do wykorzystywania na urządzeniach wyposażonych w ekran dotykowy o rozdzielczości 1024x600 pikseli. Do umożliwienia zapisywania danych w celu sporządzania dalszych statystyk stworzona została relacyjna baza danych wykorzystująca darmowy serwer MySQL Community Server w wersji 5.7 firmy Oracle. Dokładny opis funkcjonalności stworzonego oprogramowania wraz z przykładowymi widokami panelu użytkownika znajduje się w rozdziale 5.2.

4 Projekt układu do komunikacji ze złączem diagnostycznym

4.1 Zastosowane elementy

W projekcie układu do komunikacji ze złączem diagnostycznym wykorzystane zostały układy scalone wymienione poniżej(w nawiasie znajduje się symbol odpowiadający danemu elementowi na Rys. 4.1.

- Mikroprocesor F103CBT6 firmy STM32(U1). W tabeli 4.1 przedstawione zostały jego najważniejsze w kontekście omawianego układu dane katalogowe.

Tabela 4.1: Najważniejsze parametry STM32F103CBT6 [12].

Napięcie zasilania	**Częstotliwość Taktowania**	**Pamięć Flash**	**Pamięć SRAM**
3.3V	72MHz	128kB	20kB
Liczba timerów	**Interfejsy**	**Obudowa**	**Montaż**
4, 16bit	CAN, I2C x2, LIN SPI x2, USART x2	LQFP48	SMD

- Moduł komunikacji bluetooth RN4020-V/RM firmy Microchip(U2). W tabeli 4.2 przedstawione zostały jego najważniejsze dane katalogowe.

Tabela 4.2: Najważniejsze parametry RN4020-V/RM [13].

Napięcie zasilania	**Standard**	**Komunikacja**	**Protokół**
3.0-3.6V	4.1	UART	ASCII AT Commands
Programowalne GPIO	**Częstotliwość pracy**	**Wbudowana antena**	**Montaż**
7 cyfrowych 3 analogowe	2.402-2.480GHz	Tak	SMD

- Interfejs protokołu ISO9141-2, L9637D firmy STMicroelectronics(U3). Dane katalogowe tego układu znaleźć można w odnośniku [14].
- Regulator napięcia 5V MC7805BDTRK(U4). Dane katalogowe tego układu znaleźć można w odnośniku [15].
- Regulator napięcia 3V MCP1703T-3302E/CB(U5). Dane katalogowe tego układu znaleźć można w odnośniku [16].

W tabeli 4.3 znajduje się pełny wykaz wykorzystanych elementów.

Tabela 4.3: Wykaz wykorzystanych elementów wraz z opisem obudowy, rodzaju montażu oraz warstwy na projekcie obwodu drukowanego.

Oznaczenie	**Nazwa/wartość**	**Obudowa**	**Montaż**	**Ilość**	**Warstwa**
U1	STM32F103CBT6	LQFP48	SMD	1	Górna
U2	RN4020-V/RM	RN4020-ALT	SMD	1	Dolna
U3	L9637D	SO8	SMD	1	Górna
U4	MC78M05 BDTRK	TO-252-3	SMD	1	Górna
U5	MCP1703T-3302 E/CB	SOT-23-3	SMD	1	Górna
R1, R2, R3	330Ω	0402	SMD	3	Dolna
R4, R5	330Ω	0402	SMD	2	Górna
R7	510Ω	0402	SMD	1	Górna
R8	12kΩ	0402	SMD	1	Górna
R9	22kΩ	0402	SMD	1	Górna
R10	1MΩ	0402	SMD	1	Dolna
R11	10kΩ	0402	SMD	1	Dolna
C1, C2, C12	1μF 16V	0402	SMD	3	Górna
C3	1μF 25V	0603	SMD	1	Górna
C5	100nF 16V	0402	SMD	1	Dolna
C6	100nF 16V	0402	SMD	1	Górna
C7, C8	22pF 50V	0402	SMD	2	Dolna
C9, C10	10pF 50V	0402	SMD	2	Dolna
C11	4.7μF 10V	0402	SMD	1	Dolna
D1, D2, D3	Red diode	0402	SMD	3	Dolna
D4, D5	Red diode	0402	SMD	2	Górna
D8	1N4007	DO-41	THT	1	Górna
Q1	32.768kHz	3.2x1.5x0.9	SMD	1	Dolna
Q2	8MHz	HC49U	THT	1	Dolna
P1	Header	HDR1X4	THT	1	Górna
P3	Header	HDR1X5	SMD	1	Górna
P4	Header	HDR1X3	THT	1	Górna
S1	Button PS-03B	SPST-NO	THT	1	Górna
CB1	Button PS11ABK	SPST-NO	THT	1	Górna

4.2 Schemat elektryczny

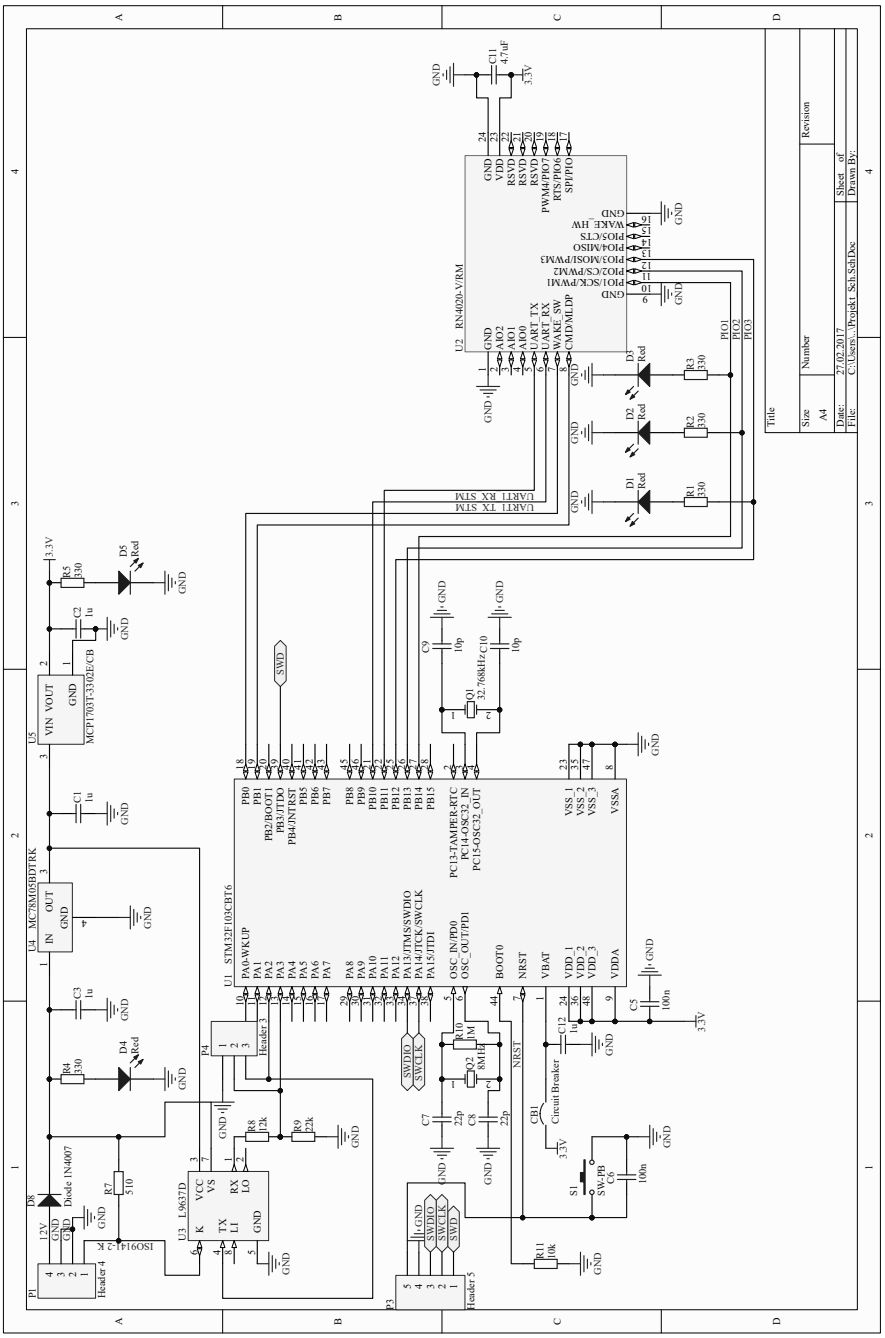
Celem zaprojektowanego układu było umożliwienie komunikacji z samochodowym komputerem pokładowym poprzez złącze diagnostyczne. Głównym elementem układu jest mikroprocesor U1, którego zadaniem jest odbieranie danych ze złącza diagnostycznego oraz przesyłanie odebranych danych za pomocą modułu bluetooth do układu wizualizacji danych. Mikroprocesor zasilany jest napięciem 3.3V. Układ został zaprojektowany do obsługi tylko jednego protokołu komunikacyjnego udostępnianego przez złącze diagnostyczne - ISO 9141-2. Wynika to z potrzeby minimalizacji całego układu, a także minimalizacji ceny jego wykonania. Sygnał protokołu komunikacyjnego doprowadzony jest do złącza P1 wraz z napięciem 12V - którego źródłem jest akumulator - oraz wyprowadzeniami masy układu. Sygnał zasilania złącza P1 doprowadzony jest to dwóch regulatorów napięcia U4 oraz U5, dzięki którym w układzie dostępne są napięcia zasilające odpowiednio 5V oraz 3.3V. W obwodzie wejściowym oraz wyjściowym umieszczone zostały kondensatory C1, C2 i C3 zgodnie z zaleceniami producenta [15][16]. W obwodzie zasilania znajduje się dioda prostownicza D8 oraz dodatkowo dwie czerwone diody LED D4 i D5 wraz z odpowiadającymi im rezystorami R4 i R5, których zadaniem jest sygnalizowanie obecności napięcia.

Sygnał linii k protokołu ISO 9141-2 wychodzący ze złącza P1 doprowadzony jest do układu interfejsu magistrali ISO 9141. Układ ten zasilany jest napięciem 5V, a jego zadaniem jest przetwarzanie sygnału wyprowadzonego ze złącza diagnostycznego. Zastosowanie takiego rozwiązania umożliwia wygodne programowanie komunikacji ze złączem diagnostycznym przy użyciu interfejsu UART, które jest bezpośrednio dostępne w zastosowanym mikroprocesorze. Ze względu na różnicę napięć pracy układu L9637D oraz mikroprocesora do odbierania danych z wyprowadzenia Rx interfejsu ISO należy dodatkowo obniżyć napięcie sygnału. Do tego celu wykorzystano rezystancyjny dzielnik napięcia R8 i R9. Wysyłanie danych z mikroprocesora może odbywać się bezpośrednio, ponieważ napięcie 3.3V jest wystarczające do zmiany stanu na wejściu Tx układu U3. W obwodzie komunikacji pomiędzy modułami umieszczono dodatkowo złącze umożliwiające bezpośrednie odczytywanie danych na liniach Rx oraz Tx, które może być wykorzystywane w celach diagnostycznych lub jako wyprowadzenie do dodatkowego mikroprocesora. Dokładny opis układu można znaleźć w nocie katalogowej producenta [14].

Układ U2 znajdujący się na schemacie to moduł komunikacyjny bluetooth. Jego zadaniem jest odbieranie wstępnie opracowanych wyników ze złącza diagnostycznego do układu wizualizacji danych. Moduł charakteryzuje się małym poborem prądu w stanie pracy(12mA), bardzo małym w stanie spoczynku(<0.5A), a także dużą szybkością przesyłania danych(do 1Mbps). Dokładną specyfikację urządzenia można znaleźć w nocie katalogowej producenta [13]. Układ zasilany jest napięciem 3.3V oraz komunikuje się z mikroprocesorem wykorzystując magistralę UART. Dodatkowo do modułu U2 doprowadzony jest sygnał wybudzający ze stanu czuwania(7:WAKE_ SW) oraz sygnał wejścia resetującego układ(8:CMD/MLDP). Wyprowadzenia PIO1, PIO2, PIO3 wykorzystane zostały do sygnalizowania stanu komunikacji układu. Dodatkowo moduł bluetooth posiada wyprowadzenia magistrali SPI, które w zaprojektowanym układzie nie zostały wykorzystane.

W układzie znajdują się również elementu obsługi mikroprocesora. Złącze P3 umożliwia programowanie mikroprocesora z wykorzystaniem zewnętrznego programatora dostępnego w zestawach STM32 NUCLEO lub DISCOVERY lub programatora ST-LINK/V2. Dodatkowo w układzie znajdują się dwa rezonatory kwarcowe(Q1, Q2) wraz kondensatorami(C7, C8, C9, C10) zgodne z zaleceniami producenta, a także przycisk umożliwiający resetowanie układu(S1).

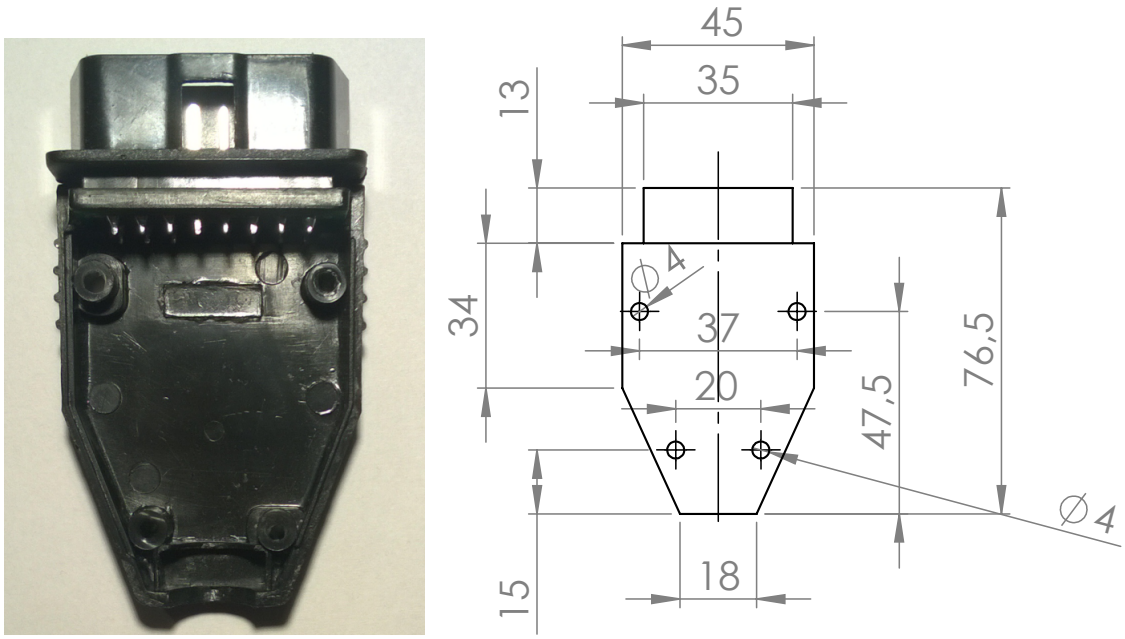
Rys. 4.1 przedstawia schemat zaprojektowanego układu. Schemat w większym rozmiarze dodany został również do załączników na końcu dokumentu.



Rys. 4.1: Schemat obwodu układu do komunikacji ze złączem diagnostycznym. Opis w tekście.

4.3 Projekt obwodu drukowanego

Na podstawie zaprojektowanego schematu elektrycznego zaprojektowany został obwód drukowany. Przy projektowaniu istotnym aspektem był rozmiar oraz kształt wykonanej płytki, aby układ mógł zostać umieszczony w obudowie, której widok oraz przekrój poprzeczny wraz z wymiarami przedstawiony został na Rys. 4.2.



(a) (b)

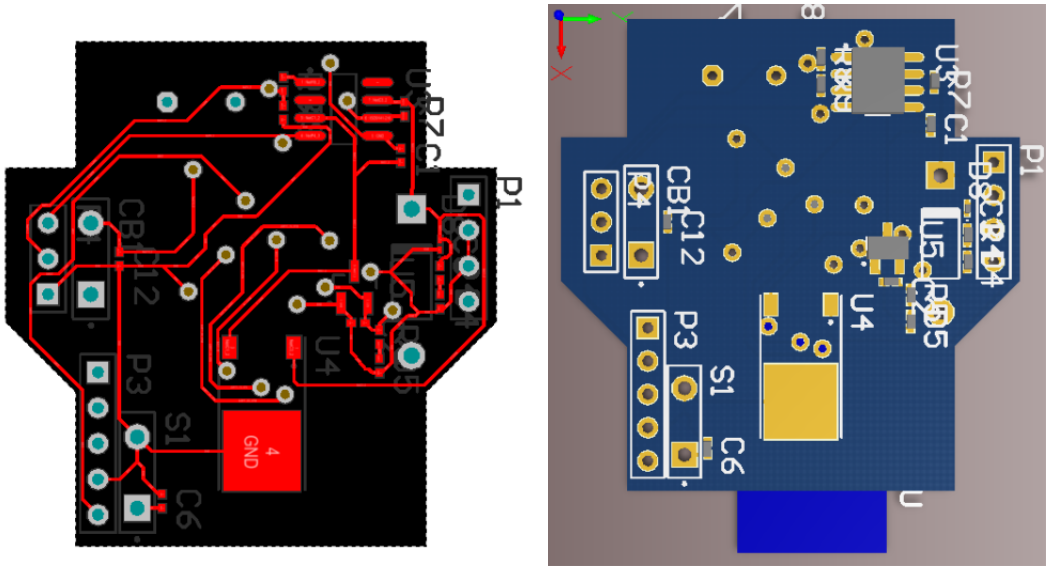
Rys. 4.2: Widok (a) oraz przekrój poprzeczny z wymiarami (b) obudowy układu do komunikacji ze złączem diagnostycznym.

Projekt obwodu drukowanego został wykonany w technologii dwustronnej. W tabeli 4.4 znajduje się opis poszczególnych warstw zaprojektowanej płytki.

Tabela 4.4: Opis wykorzystanych warstw.

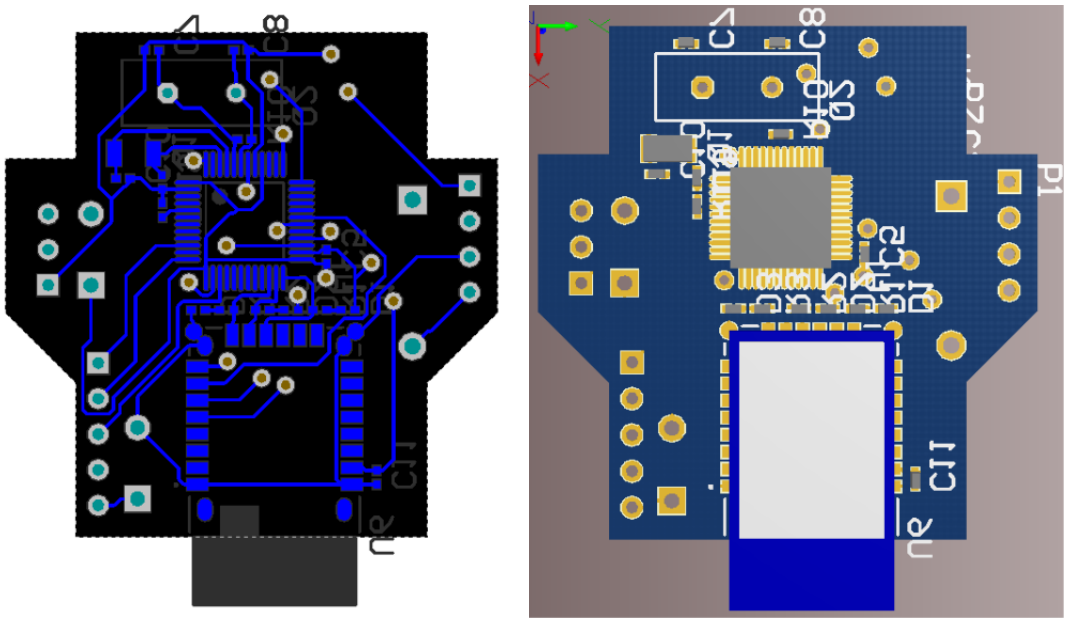
Nazwa/Typ	**Materiał**	**Grubość [mm]**	**Stała dielektryczna**
Pokrycie górne			
Górna soldermaska	Poliamid	0.01016	3.5
Górna warstwa sygnałowa	Miedź	0.03556	
Warstwa dielektryczna	Dielektryk FR-4	0.32004	4.8
Dolna warstwa Sygnałowa	Miedź	0.03556	
Dolna soldermaska	Poliamid	0.01016	3.5
Pokrycie dolne			

W tabeli 4.3 znajduje się wykaz zastosowanych elementów wraz z warstwą, na której zostały umieszczone. Na Rys. 4.3 4.4 przedstawione zostały schematy obwodów obu warstw, a także widoki modelu.



(a) (b)

Rys. 4.3: Schemat obwodu (a) oraz widok modelu (b) górnej warstwy płytki.

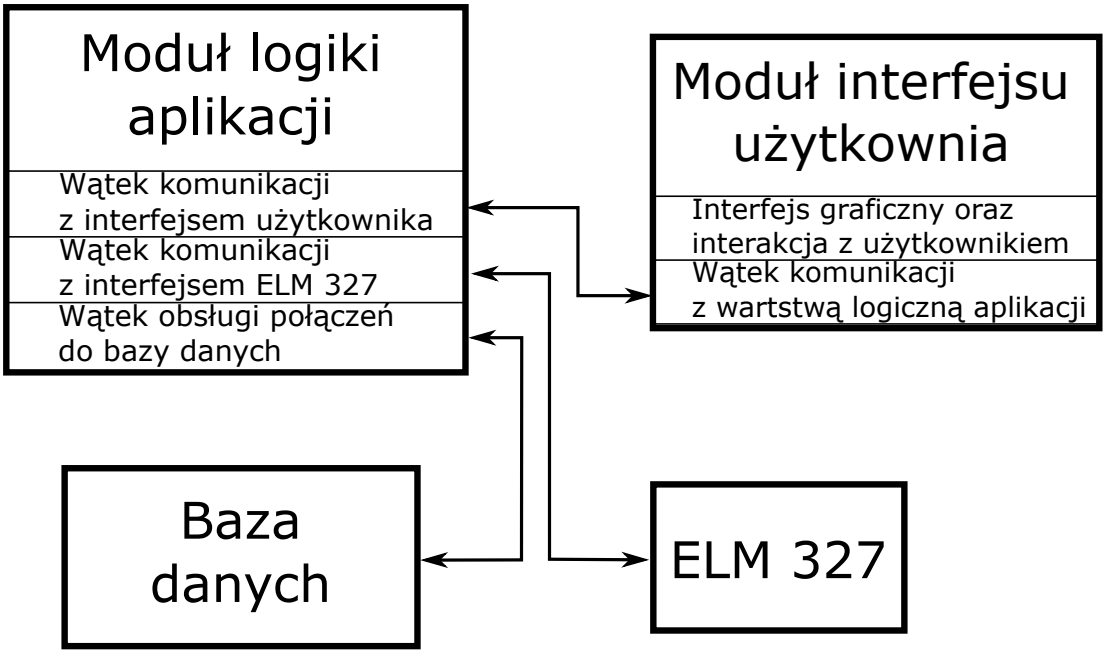


(a) (b)

Rys. 4.4: Schemat obwodu (a) oraz widok modelu (b) dolnej warstwy płytki.

5 Oprogramowanie

W ramach pracy stworzono oprogramowanie umożliwiające komunikację ze złączem diagnostycznym poprzez urządzenie ELM327, zapisywanie danych pomiarowych oraz wizualizację odczytywanych parametrów. Zgodnie z przyjętymi standardami dobrego wytwarzania oprogramowania, aplikacja podzielona została na dwa osobne moduły. Moduł pierwszy odpowiedzialny jest za zarządzanie logiką aplikacji, drugi natomiast za komunikację z użytkownikiem za pomocą graficznego interfejsu. Na Rys. 5.1 przedstawiony został schemat ideowy aplikacji.



Rys. 5.1: Schemat ideowy aplikacji.

W kolejnych dwóch podrozdziałach znajduje się dokładny opis zastosowanych rozwiązań.

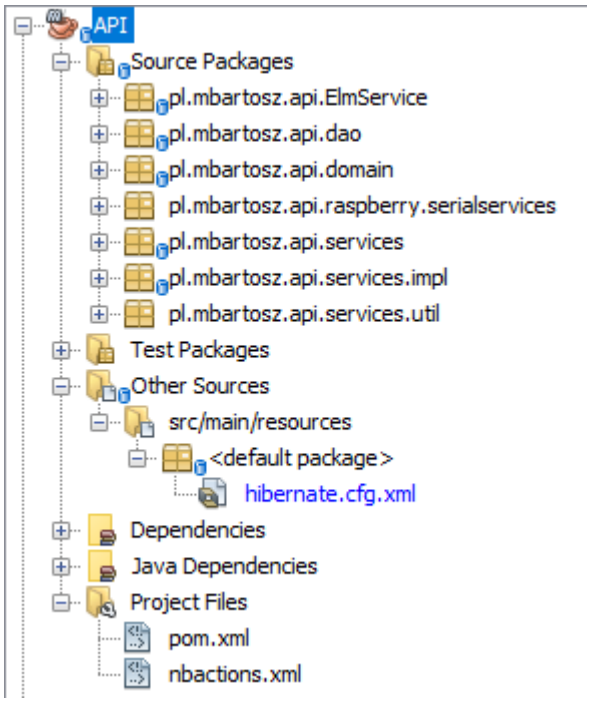
5.1 Moduł komunikacji i przetwarzania danych

5.1.1 Struktura aplikacji

Zadaniem omawianego w tym rozdziale modułu jest dostarczenie aplikacji usług niezbędnych do komunikowania się z interfejsem ELM 327 oraz przechowywania odczytywanych wyników pomiarów. Aplikacja napisana została w języku Java 8 kompilowana z wykorzystaniem narzędzia Maven z wykorzystaniem dodatkowych bibliotek:

- do obsługi portu szeregowego: jbidibc-rxtx-2.2 1.6.0 implementująca bibliotekę rxtx 2.1.7,
- do obsługi loggera aplikacji: log4j 1.2.17
- do obsługi połączenia z bazą danych: hibernate-core 5.2.6 oraz mysql-connector-java 5.1.38.

Moduł został podzielony na pakiety zgodnie z przedstawionym na Rys. 5.2.



Rys. 5.2: Schemat ideowy aplikacji.

W folderze Source Packages znajdują się wszystkie pakiety aplikacji:

- pl.mbartosz.api.ElmService - zawiera wszystkie klasy obsługujące komunikację z interfejsem ELM 327,
- pl.mbartosz.api.dao - zawiera klasy interfejsów operujących na bazie danych,
- pl.mbartosz.api.domain - zawiera klasy encji bazy danych. Klasy te generują automatycznie tabele w bazie danych jeżeli jeszcze nie istnieją,

- pl.mbartosz.api.raspberry.serialservices - zawiera klasy obsługujące konfigurację, połączenie i komunikację z wykorzystaniem portu szeregowego,
- pl.mbartosz.api.services - zawiera interfejsy pośredniczące w wymianie danych pomiędzy klasami operującymi bezpośrednio na bazie danych, a klasami aplikacji,
- pl.mbartosz.api.services.impl - zawiera klasy implementujące interfejsy z pakieto *.services,
- pl.mbartosz.api.services.util - zawiera klasę konfiguracyjną połączenia z bazą danych.

Folder Test Packages zawiera klasy automatycznych testów. Do aplikacji nie zostały napisane żadne testy, dlatego folder ten jest pusty. Kolejny folder - Other Sources - zawiera plik konfiguracyjny biblioteki hibernate umożliwiający połączenie aplikacji z bazą danych. W folderze Project Files znajdują się pliki konfiguracyjne dla narzędzia Maven, oraz środowiska NetBeans.

5.1.2 Funkcjonalność aplikacji

Aplikacja udostępnia trzy funkcjonalności: komunikację z portem szeregowym, komunikacje z ELM 327 wykorzystująca port szeregowy oraz obsługę zapisywania i pozyskiwania danych z bazy. Obsługa komunikacji z portem szeregowym zawarta w pakiecie pl.mbartosz.api.raspberry.serialservices umożliwia połączenie z portem wykorzystując wykorzystując zdefiniowane przez użytkownika parametry, a także wysyłanie danych do portu oraz odbieranie informacji zwrotnych w trybie zapytania i oczekiwania na odpowiedź. W pakiecie znajdują się trzy klasy:

- SerialPortComm - zawiera funkcje umożliwiające wyszukiwanie dostępnych w systemie portów, podłączenie oraz rozłączenie od portu szeregowego,
- SerialPortCongurationParameters - zawiera obiekty konfiguracyjne dla połączenia z portem,
- AtCommandPort - zawiera funkcje umożliwiające komunikację z wykorzystaniem portu szeregowego.

Najistotniejsze z punktu widzenia funkcjonalności aplikacji są metody odpowiedzialne za wysyłanie instrukcji do portu szeregowego oraz odbieranie informacji zwrotnych. Funkcja wysyłająca dane do portu zaprezentowana została na Listingu 1. Przyjmuje ona jeden argument w postaci łańcucha znaków. Podawana jest tutaj instrukcja, która ma zostać wysłana do portu. Zmienna responseData służy do przechowywania otrzymanej odpowiedzi w postaci listy tablic typu byte. Na początku następuje wyzerowanie zmiennej przechowującej liczbę otrzymanych odpowiedzi, a następnie do portu wysyłana jest żądana komenda po konwersji to tablicy typu byte. W kolejnym kroku wykonywana jest pętla, powtarzająca się dopóki nie liczba otrzymanych odpowiedzi nie jest równa liczbie oczekiwanych odpowiedzi. Takie rozwiązanie umożliwia wysłanie jednej komendy i otrzymanie kilku linii odpowiedzi co jest wykorzystywane przy wysyłaniu niektórych instrukcji do OBD-II. Po wysłaniu instrukcji wątek przechodzi w stan oczekiwania na odpowiedź.

```
1   public List<byte[]> putAtCommand(String cmd) throws IOException,
2           InterruptedException {
3       responseData = new ArrayList<>();
4       receivedResponses = 0;
5       out.write(cmd.getBytes());
6       while (receivedResponses != commandObject.
            getExpectedResponseLines()) {
7           responseSync.wait();
8       }
9       return responseData;
10  }
```

Listing 1: Funkcja wysyłająca dane do portu szeregowego. Opis w tekście.

Funkcja odpowiadająca za odbieranie danych z portu szeregowego implementuje interfejs SerialPortEventListener, dzięki czemu możliwa jest wykonanie określonych instrukcji dokładnie w momencie pojawienia się danych na porcie. Wysyłanie danych oraz ich odbieranie synchronizowane jest poprzez obiekt responseSync. W każdym wywołaniu wydarzenia na porcie odbierana jest tylko jedna linia danych i zapisywana do listy tablic typu byte. Następnie zwalniana jest blokada obiektu responseSync i program wraca do linii siódmej funkcji wysyłającej dane do portu. Obsługa odbierania danych przestawiona została na Listingu 2.

```
1   private class Callback implements SerialPortEventListener {
2
3       @Override
4       public void serialEvent(SerialPortEvent spe) {
5           byte[] responseLine = new byte[256];
6           synchronized (responseSync) {
7               if (spe.getEventType() == SerialPortEvent.
                    DATA_AVAILABLE) {
8                   try {
9                       int av = in.available();
10                      int read = in.read(responseLine, 0, av);
11                      responseData.add(responseLine);
12                      receivedResponses++;
13                      responseSync.notifyAll();
14                  } catch (IOException ioe) {
15                      LOGGER.error("Cannot read data from port " +
                            ioe);
16                  }
17              }
18          }
19      }
20  }
```

Listing 2: Funkcja odbierająca dane z portu szeregowego. Opis w tekście.

Realizacja komunikacji z interfejsem ELM 327 zawarta jest w pakiecie pl.mbartosz.api.ElmService, który zawiera następujące klasy:

- AT_ COMMANDS - jest to klasa wyliczeniowa, w której zdefiniowane są komendy AT, ich opis, a także oczekiwana odpowiedź dla każdej komendy,
- CommandObject - obiekt przechowujący komendę, oraz oczekiwaną liczbę linii odpowiedzi,
- ResponseObject - obiekt przechowujący listę odpowiedzi oraz ich status,
- CommunicationMode - klasa wyliczeniowa, w której zdefiniowane są tryby pracy z urządzeniem : AUTO, MANUAL i WAIT,
- CommunicationState - klasa wyliczeniowa, w której zdefiniowany jest stan połączenia z urządzeniem: CONNECTED, DISSCONNCTED,
- ElmMessageService - klasa udostępniająca metody do wysyłania i odbierania danych z urządzenia.

Większość klas w tym pakiecie to obiekty przechowujące parametry pracy aplikacji. Najważniejsza z punktu widzenia funkcjonalności jest klasa ElmMessageService, która zarządza wątkiem wysyłającym instrukcje do interfejsu ELM 327, generuje instrukcje do wysłania, a także przetwarza odebrane informacje. W celu optymalizacji pracy aplikacji komunikacja przeniesiona jest do osobnego wątku, który widocznym jest na Listingu 3.

```
1     @Override
2     public void run() {
3         LOGGER.debug("Communication thread started");
4         while (true) {
5             if (getCommunicationMode() == CommunicationMode.AUTO) {
6                 synchronized (atCommandPort.responseSync) {
7                     sendToPort(commandObject);
8                 }
9             } else if (getCommunicationMode() == CommunicationMode.
                  MANUAL) {
10                synchronized (atCommandPort.responseSync) {
11                    setCommunicationMode(CommunicationMode.WAITING);
12                    sendToPort(commandObject);
13                }
14            } else {
15                try {
16                    Thread.sleep(10);
17                } catch (InterruptedException ex) {
18                    java.util.logging.Logger.getLogger(
                          ElmMessageService.class.getName()).log(Level.
                          SEVERE, null, ex);
19                }
20            }
21        }
22    }
```

Listing 3: Wątek komunikacji z ELM 327. Opis w tekście.

Metoda run() jest implementacją interfejsu Runnable pozwalającego na uruchamianie nowych wątków. Po uruchomieniu wątku wykonywana jest instrukcja nieskończonej pętli, której działanie może zakończyć tylko wyłączenie programu. W pętli tej sprawdzane są warunki stanu aplikacji. Jeżeli aplikacja jest w trybie pracy automatycznej to znaczy w przypadku wyświetlania panelu głównego lub zaznaczenia odpowiedniej opcji w ustawieniach aplikacji(rozdział 5.2) do urządzenia wysyłana jest instrukcja w trybie ciągłym, synchronizowanym obiektem responseSync. W przypadku gdy aplikacja pracuje w trybie ręcznym lista instrukcji zostaje wysłana do urządzenia tylko raz. Następnie aplikacja zmienia stan pracy na oczekiwanie. W trybie oczekiwania wątek jest wstrzymywany na 10ms, po czym następuje ponowne sprawdzenie warunków wykonywania instrukcji wątku.

Za generowanie oraz wysyłanie instrukcji do urządzenia odpowiada funkcja sendToPort, której struktura przedstawiona na Listingu 4. W metodzie tej najważniejszą funkcję pełni pętla for. Ponieważ pojedyncze zapytanie do urządzenia może zawierać kilka instrukcji, pętla ma za zadanie pobierać z listy instrukcji po kolei każdą z nich i wysyłania jej metodą putAtCommand. Funkcja ta zwraca obiekt typu ResponseObject, który następnie dodawany jest do listy otrzymanych odpowiedzi. Kiedy wykonane zostają wszystkie instrukcje z zapytania i otrzymano odpowiedzi od urządzenia wywoływana jest metoda sprawdzająca poprawność odebranych informacji executeResponse.

```
1     public List<ResponseObject> sendToPort(CommandObject commandObject)
          {
2         List<ResponseObject> responseObject = new ArrayList<>();
3         LOGGER.debug("Sending message " + commandObject.getCmd());
4         LOGGER.debug("Expected response lines " + commandObject.
              getExpectedResponseLines());
5         try {
6             atCommandPort.setCommandObject(commandObject);
7             for (int i = 0; i < commandObject.getCmd().size(); i++) {
8                 ResponseObject co = new ResponseObject(atCommandPort.
                      putAtCommand(commandObject.getCmd().get(i).getCmd()
                      ));
9                 responseObject.add(co);
10            }
11        } catch (IOException | InterruptedException ex) {
12            LOGGER.error("Cannot send cmd to port : " + ex);
13            responseObject.add(new ResponseObject());
14        }
15        if (responseObject.size() > 0) {
16            executeResponse(responseObject, commandObject);
17        }
18        return responseObject;
19    }
```

Listing 4: Funkcja generująca instrukcje do wysłania. Opis w tekście.

Zadaniem metody executeResponse jest sprawdzenie poprawności otrzymanych odpowiedzi dla każdej instrukcji oraz wymuszenie odświeżenia komponentów interfejsu użytkownika. Metoda ta w pętli sprawdza w pierwszej kolejności czy flaga poprawności w każdym odebranym obiekcie jest zaznaczona. Następnie w kolejnej pętli porównuje wszystkie odebrane odpowiedzi dla jednego obiektu responseObject z oczekiwanymi odpowiedziami. Odpowiedzialna jest za to funkcja checkResponse, która jako argument przyjmuje dwa łańcuchy znaków. Pierwszy argument to otrzymana odpowiedź, drugi natomiast to odpowiedź oczekiwana w postaci wyrażenia regularnego zdefiniowanego dla każde instrukcji w klasie AT_ COMMANDS. Po sprawdzeniu poprawności odpowiedzi następuje wymuszenie odświeżenia interfejsu użytkownika. Omawiane funkcje przedstawione są na Listinigu 5.

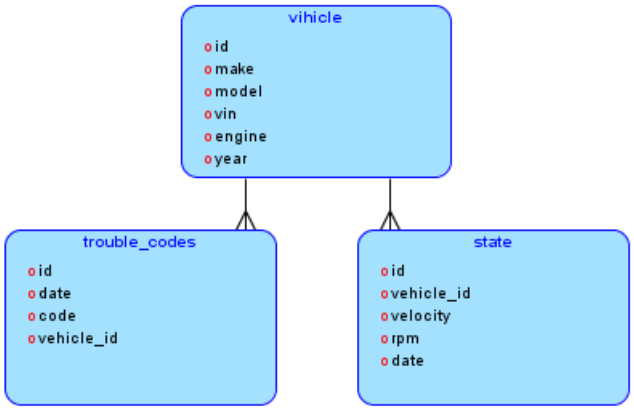
```
1     private void executeResponse(List<ResponseObject> responseObject,
          CommandObject commandObject) {
2         for (int i = 0; i < responseObject.size(); i++) {
3             if (responseObject.get(i).isResponseOk()) {
4                 LOGGER.debug("Received response " + responseObject.get(
                      i).getResponse());
5                 for (String response : responseObject.get(i).
                      getResponse()) {
6                     if (checkResponse(response, commandObject.getCmd().
                          get(i).getExpectedResponseRegex())) {
7                         if (getCommunicationMode() != CommunicationMode
                              .AUTO) {
8                             refreshTextArea(commandObject.getCmd().get(
                                  i).getCmd(), response);
9                         } else {
10                            refreshLabels(commandObject, responseObject
                                  );
11                        }
12                    }
13                }
14            }
15        }
16    }
17
18    private boolean checkResponse(String response, String
          expectedResponse) {
19        Pattern expectedResponsePattern = Pattern.compile(
              expectedResponse);
20        Matcher m = expectedResponsePattern.matcher(response);
21        return m.matches();
22    }
```

Listing 5: Funkcja sprawdzająca poprawność odpowiedzi i wymuszająca odświeżenie interfejsu użytkownika. Opis w tekście.

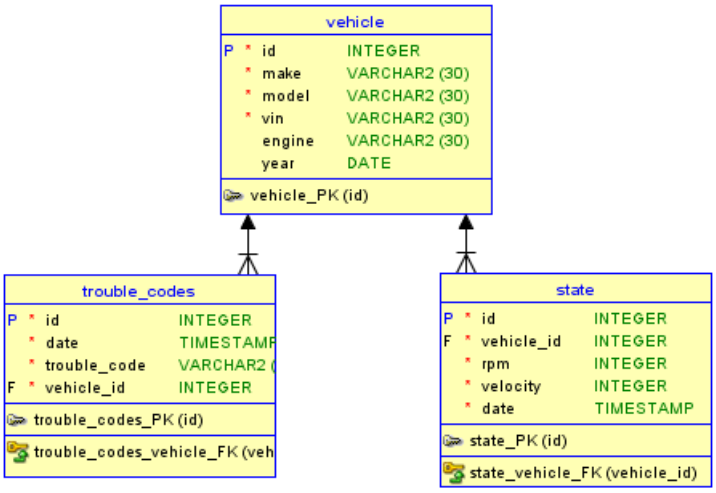
Pozostałe pakiety wymienione w podrozdziale 5.1.1 odpowiadają za komunikację aplikacji z bazą danych i są w istocie implementacją framworku Hibernate, dlatego ich dokładny opis został w pracy pominięty.

5.1.3 Baza danych

Aplikacja połączona jest z relacyjną bazą danych MySQL umożliwiającą zapisywanie odczytywanych wyników w celu ich dalszej analizy. Na Rys. 5.3 przedstawiony został model logiczny oraz fizyczny bazy danych.



(a)



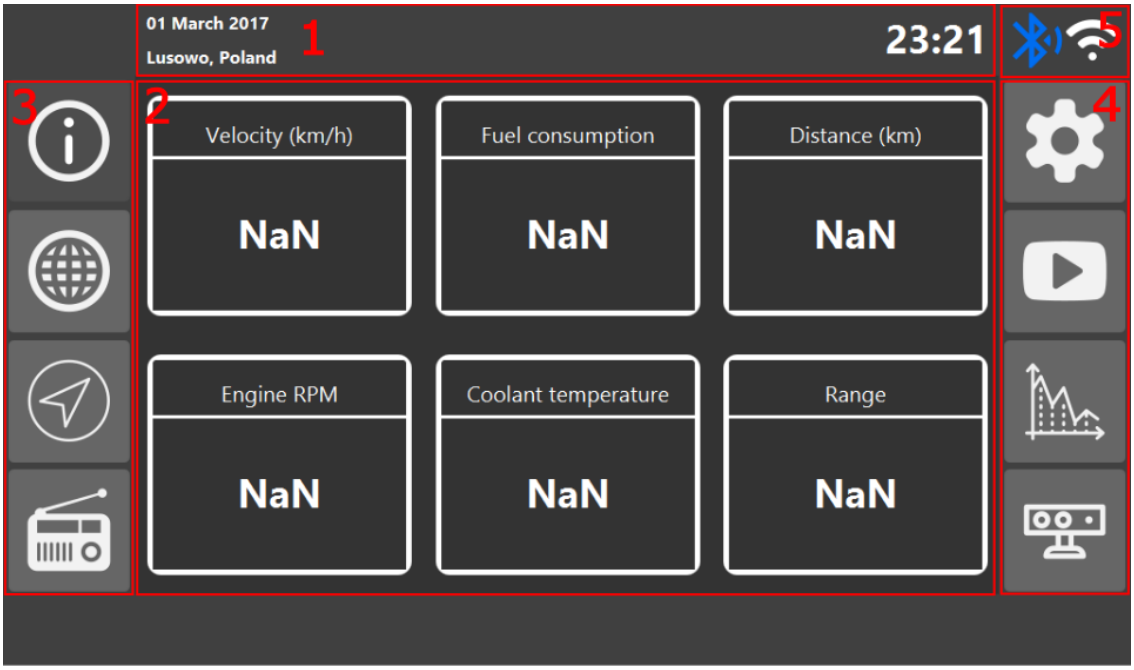
(b)

Rys. 5.3: Model logiczny (a) oraz fizyczny (b) bazy danych. Opis w tekście.

Baza danych składa się z trzech tabel. Tabela vehicle zawiera podstawowe informację o pojeździe. Tabela state przechowuje informacje o odczytanych prędkościach i obrotach silnika pojazdu. Tabela trouble_ codes odczytanych kodach błędów pojazdu. Tabela vehicle jest połączona relacją jeden do wielu z pozostałymi tabelami.

5.2 Moduł interfejsu użytkownika

Moduł interfejsu użytkownika napisany został w języku Java z wykorzystaniem najnowszego framweworku do tworzenia interfejsów graficznych JavaFX 8. Wygląda oraz obsługa aplikacji zostały dostosowane ekranu o rozdzielczości 1024x600 pikseli. Po uruchomieniu użytkownikowi ukazuje się ekran przedstawiony na Rys. 5.6.



Rys. 5.4: Panel główny interfejsu użytkownika.

Na zaprezentowanym widoku przedstawiony jest panel główny z zaznaczonymi poszczególnymi sekcjami:

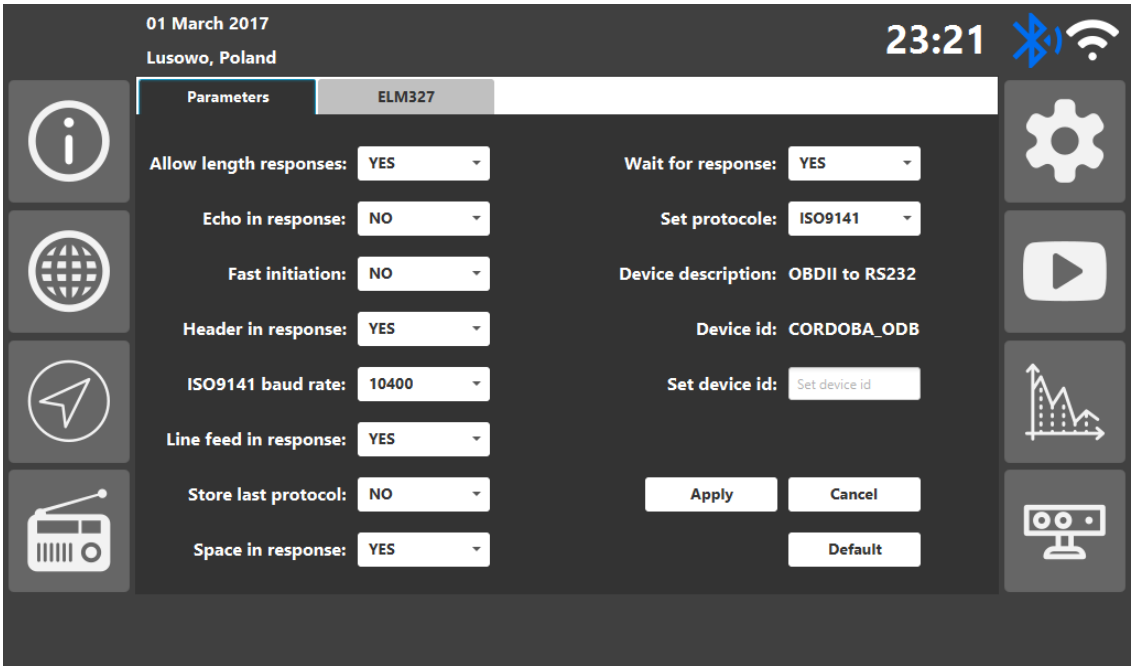
1. Nagłówek okna. Widoczne są tutaj data, lokalizacja oraz aktualny czas.
2. Okno główne. Prezentuje treść odpowiednią do wybranej opcji z menu. Widok na Rys. 5.6 prezentuje domyślne okno główne.
3. Lewy panel przycisków menu. Znajdują się tutaj kolejno od góry:
   - Przycisk informacyjny. Po jego naciśnięciu w oknie głównym pokazany jest widok domyślny.
   - Przycisk przeglądarki. W oknie pokazana jest przeglądarka. Nieobsłużony.
   - Przycisk nawigacji. W oknie pokazana jest nawigacja. Nieobsłużony.
   - Przycisk radia. W oknie pokazana jest panel sterowania radiem. Nieobsłużony.
4. Prawy panel przycisków menu. Znajdują się tutaj kolejno od góry:
   - Przycisk ustawień. Po jego naciśnięciu w oknie głównym jest panel ustawień.

- Przycisk multimediów. W oknie pokazana jest panel zarządzania multimediami. Nieobsłużony.
- Przycisk statystyk. W oknie pokazana jest panel prezentujący statystki zebranych danych. Nieobsłużony.
- Przycisk czujników. W oknie pokazana jest stan czujników parkowania. Nieobsłużony.

5. Panel statusu. Informuje o podłączeniu urządzenia ELM 327 oraz dostępie do sieci.

Przełączanie widoków okna polega na zmianie zawartości okna głównego. Panele przycisków nie zmieniają swojej zawartości. Widok przycisków zmienia się tylko w przypadku najechania na przycisk lub jego naciśnięcia. W panelu statusu kolorem niebieskim zaznaczone jest uzyskanie połączenia, natomiast białym brak połączenia. Wartości widoczne na widoku NaN oznaczają, że dane nie są odbierane od urządzenia ELM 327. Kiedy aplikacja pracuje w trybie automatycznym dane te są automatycznie odświeżane po odebraniu kolejnych wyników.

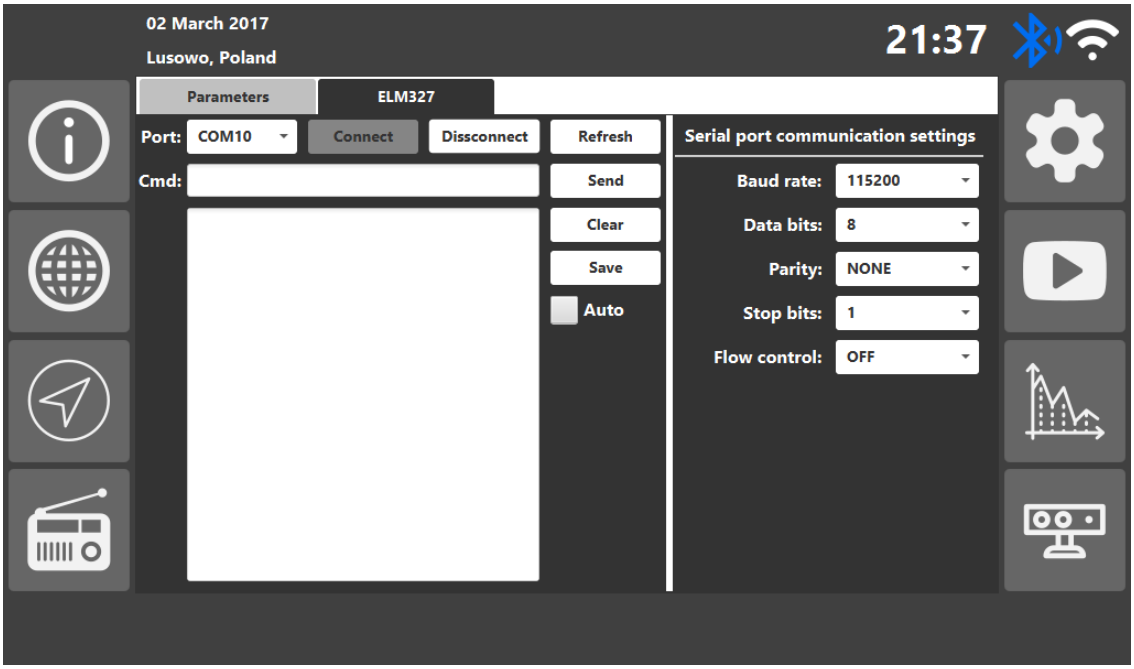
Po naciśnięciu przycisku ustawień okno główne zmienia zawartość i ukazuje się widok przedstawiony na Rys. 5.6



Rys. 5.5: Panel główny interfejsu użytkownika.

W panelu ustawień dostępne są dwie zakładki. Pierwsza z nich umożliwia skonfigurowanie urządzenia ELM 327, a także prezentuje opis oraz identyfikator podłączonego urządzenia. W celu wysłania ustawień do interfejsu należy nacisnąć przycisk Apply. Naciśnięcie przycisku Cancel powoduje powrót wszystkich kontrolek do poprzedniego stanu natomiast naciśnięcie przycisku Default powoduje przywrócenie wszystkich domyślnych ustawień.

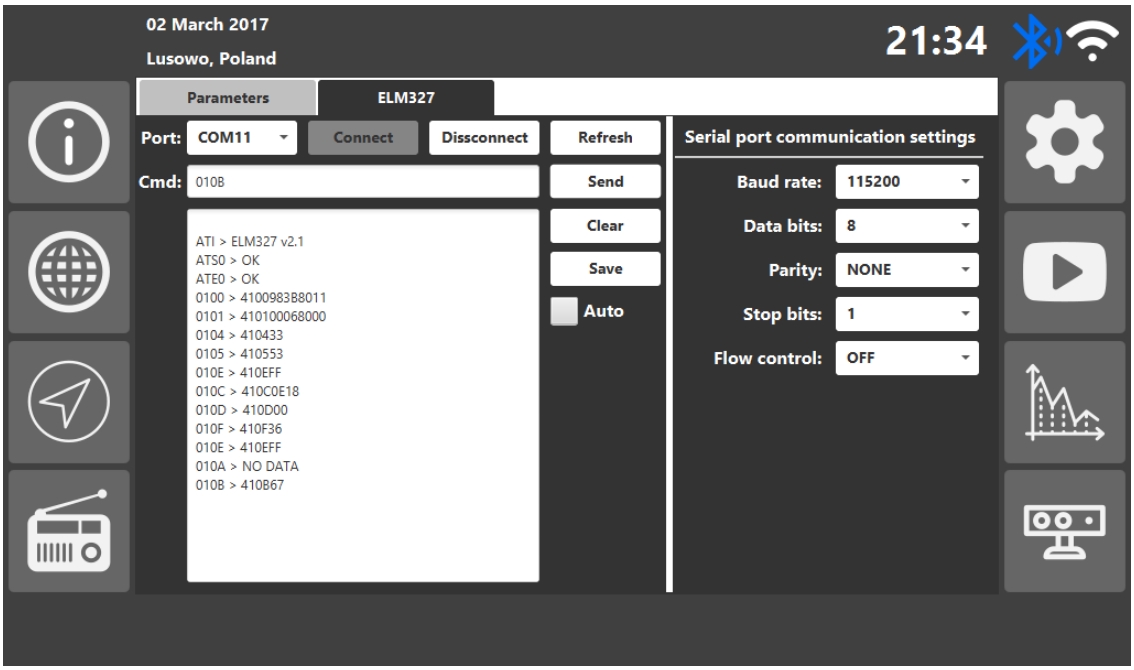
Druga zakładka dostępna w panelu ustawień umożliwia podgląd odbieranych danych od urządzenia ELM, a także pracę w trybie ręcznego wysyłania żądań. Zakładka ta podzielona jest na dwie części. Strona lewa odpowiada za wybór portu szeregowego do którego aplikacja ma się podłączyć. Przyciski Connect i Dissconnect odpowiadają za ustanawianie połączenia lub jego zakończenie. Przycisk Refresh wywołuje funkcję wyszukującą wszystkie dostępne w systemie porty szeregowe. W drugim wierszu znajduje się pole tekstowe Cmd w które należy wpisać instrukcję do wysłania do interfejsu. Przycisk Send wywołuje metodę wysyłającą instrukcję poprzez ustawienie stanu aplikacji na ręczny. Poniżej znajduje się pole tekstowe wyświetlające odpowiedzi urządzenia ELM. Pole to można wyczyścić naciskając na przycisk Clear lub zapisać jego zawartość do pliku poprzez naciśnięcie przycisku Save. W panelu znajduje się dodatkowo kontrolka Auto, której zaznaczenie przełącza aplikację w tryb pracy automatycznej. W panelu po prawej można skonfigurować parametry połączenia szeregowego. Dostępne opcje to: szybkość transmisji, liczba bitów danych, parzystość, liczba bitów stopu oraz kontrola przepływu. Aby nowe ustawienia zostały wprowadzone należy zakończyć bieżące połączenie z portem i zainicjować je na nowo. Widok drugiej zakładki panelu ustawień widoczny jest na Rys. 5.6.



Rys. 5.6: Panel główny interfejsu użytkownika.

6 Wyniki testów układu

Stworzone oprogramowanie poddane zostało testom praktyczny w samochodzie Seat Cordoba II. Połączenie zostało skonfigurowane tak, aby w odpowiedzi nie odsyłane było echo żądania oraz znaki białe. Z odpowiedzi dodatkowo usunięty jest nagłówek, ale dopiero po sprawdzeniu czy spełnia ona warunek podany w wyrażeniu regularnym. Sprawdzone zostały oba tryby pracy: automatyczny i ręczny. W pierwszy kroku wykonany został test trybu ręcznego pracy aplikacji. Wyniki kilku przykładowych zapytań przedstawiono na Rys. 6.1.



Rys. 6.1: Wyniki testu ręcznego trybu pracy programu.

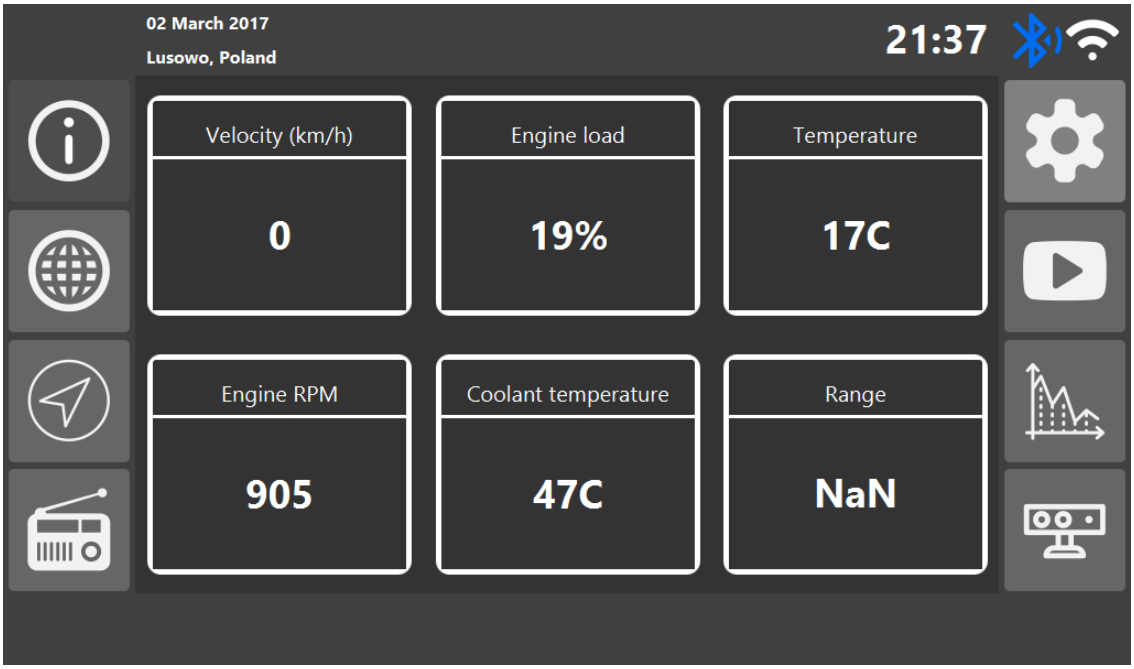
Pierwsza instrukcja wysłana to zapytanie o identyfikator urządzenia. Następnie wysłane zostały dwie instrukcje konfigurujące strukturę odpowiedzi. Instrukcja 0100 to pytanie o dostępne kody PID z zakresu 01-20. Na podstawie odpowiedzi wyznaczono dostępne kody dla testowanego samochodu.

Tabela 6.1: Kodu PID udostępnione w samochodzie testowym.

9				8				3				B			
1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	1
01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
8				0				1				1			
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20

Kolejna instrukcja 0101 to instrukcja sprawdzająca stan diody kontrolnej silnika. Następnie wysłanych zostało kilka instrukcji o bieżące parametry pracy samochodu. Na podstawie przeprowadzonych testów można powiedzieć, że tryb pracy ręcznej zachowuje się poprawnie.

W kolejnym kroku wykonano test automatycznego trybu pracy aplikacji dla parametrów udostępnionych w testowym pojeździe. Mierzonymi parametrami były: obciążenie silnika podawane w (0 - 100%), temperatura cieczy chłodzącej (-40 - 215°C), prędkość obrotowa silnika (0 - 16383 obr/min), prędkość silnika (0 - 255 km/h) oraz temperatura powietrza zasysanego (-40 - 215°C). Otrzymane rezultaty zaprezentowane zostały na Rys. 6.2



Rys. 6.2: Wyniki testu automatycznego trybu pracy programu.

Na podstawie obserwacji pracy programu można powiedzieć, że automatyczny tryb pracy programu również zachowuje się poprawnie.

7 Dyskusja potencjalnych zastosowań

Podstawowym zastosowanie układu omawianego w niniejszej pracy mogłoby być wczesne informowanie o wystąpieniu błędu w pojeździe. W tym celu należałoby cyklicznie odpytywać komputer pokładowy czy wystąpił jakieś błędy. Takie odpytywanie mogłoby się odbywać na przykład przy każdym uruchomieniu pojazdu. W tym celu należałoby dodatkowo doprowadzić do komputera sterującego programem sygnał zapłonu. Jeżeli w pojeździe zostałby wykryty błąd to na wyświetlaczu pojawiłby się stosowny napis z informacją o błędzie oraz jego źródle.

Innym zastosowaniem mogłoby być obliczanie wartości średniego spalania czy zasięgu pojazdu. W tym celu można wykorzystać następujące komendy:

- 01 5E - zwraca wartość aktualnego przepływu paliwa w l/h,
- 01 0D - zwraca wartość aktualnej prędkości pojazdu w km/h,
- 01 2F - zwraca wartość ilości paliwa w zbiorniku jako wartość procentowa pojemności zbiornika.

Pobierając wartość aktualnego przepływu paliwa dostatecznie często przez określony okres czasu można z dobrą dokładnością wyliczyć spalanie średnie korzystając z wzoru:

$$Q_{sr} = \frac{\sum Q_{ch}}{n}\left[\frac{l}{h}\right], \tag{7.0.1}$$

gdzie: Q_{sr} to średnia wartość przepływu paliwa w l/h, Q_{ch} to wartość chwilowa wartość przepływu paliwa w l/h, n to liczba próbek. Następnie w analogiczny sposób można obliczyć średnią prędkość pojazdu. Korzystając z obliczonych wartości można wyznaczyć wartość średniego spalania na kilometr oraz na każde 100 kilometrów:

$$C_{sr} = \frac{Q_{sr}}{v_r}\left[\frac{l}{km}\right], \tag{7.0.2}$$

gdzie: C_{sr} to średnia spalanie w l/km, c_{sr} obliczona średnia prędkość pojazdu. Wartość tę można w łatwy sposób przeliczyć na spalanie pojazdu w litrach na 100 kilometrów.

Wykorzystując obliczoną wartość średniego spalania samochodu na 100 kilometrów można wyznaczyć zasięg samochodu. Pobierając wartość ilości paliwa w zbiorniku i znając pojemność maksymalną zbiornika można wykorzystać wzory:

$$d = \frac{\frac{V_a}{100} * V_{max}}{C_{sr}}[km], \tag{7.0.3}$$

gdzie: d - obliczony zasięg pojazdu, V_a - aktualna ilość paliwa w zbiorniku jako wartość procentowa pojemności zbiornika.

8 Podsumowanie

Literatura

[1] Schmidgall R., Zimmermann W., *Magistrale Danych w Pojazdach. Protokoły i Standardy*, Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 2008.

[2] Merkisz J., Mazurek St., *Pokładowe systemy diagnostyczne pojazdów samochodowych*, Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 2002.

[3] Sitek K., Syta S., *Pojazdy Samochodowe. Badania stanowiskowe i diagnostyka*, Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 2011.

[4] White C., Randall M., *Kody usterek*, Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 2006.

[5] Fryśkowski B., Grzejszczyk E., *Systemy transmisji danych*, Wydawnictwa Komunikacji i Łączności WKŁ, Warszawa, 10.

[6] Upton E., Halfacree G., *Raspberry Pi. Przewodnik użytkownika. Wydanie III*, Wydawnictwo Helion, Gliwice, 2015.

[7] https://www.raspberrypi.org/documentation/

[8] Schildt H., *Java. Kompendium programisty. Wydanie IX*, Wydawnictwo Helion, Gliwice, 2015.

[9] https://law.resource.org/pub/us/cfr/ibr/005/sae.j1962.2002.pdf (14.02.2017)

[10] https://law.resource.org/pub/us/cfr/ibr/005/sae.j1979.2002.pdf (14.02.2017)

[11] https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf (14.02.2017)

[12] http://www.st.com/content/ccc/resource/technical/document/datasheet/33/d4/6f/1d/df/0b/4c/6d/CD00161566.pdf/files/CD00161566.pdf/jcr:content/translations/en.CD00161566.pdf (27.02.2017)

[13] http://ww1.microchip.com/downloads/en/DeviceDoc/50002279A.pdf (27.02.2017)

[14] http://www.st.com/content/ccc/resource/technical/document/datasheet/4a/80/83/26/e0/78/4d/18/CD00000234.pdf/files/CD00000234.pdf/jcr:content/translations/en.CD00000234.pdf (27.02.2017)

[15] http://pdf.datasheetcatalog.com/datasheet2/9/0p4t1g1lw0spoo5ukh2s1uxchzky.pdf (27.02.2017)

[16] http://www.mouser.com/ds/2/268/22049a-51817.pdf (27.02.2017)

[17] http://docs.oracle.com/javase/8/javafx/api/toc.htm (14.02.2017)

[18] https://docs.oracle.com/javase/8/docs/api/index.html (14.02.2017)

[19] http://docs.jboss.org/hibernate/orm/5.2/javadocs/ (14.02.2017)

[20] http://docs.jboss.org/hibernate/jpa/2.1/api/ (14.02.2017)

[21] http://pi4j.com/apidocs/ (14.02.2017)

[22] https://maven.apache.org/guides/ (14.02.2017)

Spis rysunków

Spis tabel