



AKADEMIA  
NAUK STOSOWANYCH  
W ŁOMŻY

**Wydział Nauk Informatyczno-Technologicznych**

Kierunek studiów: **Informatyka II stopnia**

Ścieżka rozwoju: **Systemy mobilne**

**Damian Modzelewski**

8419

**M2P W SYSTEMIE IOT NA PRZYKŁADZIE ANALIZY  
PARAMETRÓW PORUSZAJĄCEGO SIĘ SAMOCHODU.**

Praca dyplomowa magisterska napisana pod  
kierunkiem:

**dr inż. Grzegorza Rubina**

.....

Podpis promotora

Łomża 2022



AKADEMIA  
NAUK STOSOWANYCH  
W ŁOMŻY

**Faculty of Computer Science and Technology**

Field of study: **Computer Science Master's Degree**

Specialization path: **Mobile systems**

**Damian Modzelewski**

8419

**M2P IN IOT SYSTEM WITH AN EXAMPLE OF  
ANALYZING THE PARAMETERS OF A MOVING CAR**

Supervisor:

**dr inż. Grzegorz Rubin**

.....

Supervisor sign

Lomza 2022



## SPIS TREŚCI

<b>1. Wstęp .....</b>	<b>1</b>
<b>2. Cel i uzasadnienie wyboru tematu .....</b>	<b>2</b>
<b>3. Komputery pokładowe oraz interfejsy komunikacyjne wykorzystywane w pojazdach .....</b>	<b>3</b>
3.1. ECU .....	3
3.2. System ABS .....	4
3.2.1. System TCS .....	4
3.2.2. System ESP .....	5
3.3. FIS .....	5
3.4. Interfejsy komunikacyjne .....	6
3.4.1. Magistrala CAN .....	6
3.4.2. Protokół SENT .....	8
3.4.3. Protokół KWP .....	9
3.4.4. Magistrala LIN .....	10
<b>4. Użyte rozwiązania oraz technologie .....</b>	<b>12</b>
4.1. Interfejs OBD .....	12
4.2. Interpreter ELM327 .....	14
4.3. ESP8266 oraz ESP32 .....	16
4.4. Działanie sieci ESP-NOW oraz MQTT .....	17
4.5. Komunikacja IoT .....	18
4.6. Raspberry PI oraz SQLite .....	20
4.6.1. Mikrokomputery Raspberry PI .....	21
4.6.2. Bazy danych SQLite .....	24
4.7. Nextion HMI oraz jego możliwości .....	25



4.7.1. Program Nextion Editor .....	25
4.7.2. Interfejs HMI Nextion .....	27
<b>5. Projekt urządzenia .....</b>	<b>29</b>
5.1. Wymagania funkcjonalne i нефункционалне urządzenia, specyfikacja techniczna oraz wykorzystane biblioteki. ....	34
5.1.1. Wymagania funkcjonalne .....	34
5.1.2. Wymagania нефункционалне .....	35
5.1.3. Specyfikacja techniczna urządzeń .....	35
5.1.4. Użyte biblioteki .....	35
<b>6. Utworzenie systemu .....</b>	<b>37</b>
6.1. Znaczenie otrzymywanych specyficznych danych .....	37
6.2. Kod oraz objaśnienie odbiornika danych z modułu ELM327 na bazie ESP32 ..	40
6.3. ESP8266 jako mikrokomputer powiązany z Nextion .....	48
6.3.1. ESP8266 odpowiedzialny za obróbkę danych dla interfejsu .....	49
6.3.2. Utworzenie struktury wyświetlacza Nextion HMI .....	52
6.4. ESP8266 jako „Czarna skrzynka” .....	53
6.4.1. ESP8266 przesyłające dane pomiędzy ESP32 a Raspberry PI .....	54
6.4.2. Raspberry PI zawierające bazę danych SQLite .....	55
<b>7. Pomiary .....</b>	<b>64</b>
7.1. Pomiary uzyskane w czasie rzeczywistym z wyświetlacza .....	64
7.2. Pomiary gromadzone w bazie danych .....	68
<b>8. Podsumowanie .....</b>	<b>74</b>
<b>Literatura .....</b>	<b>76</b>
<b>Spis rysunków .....</b>	<b>78</b>
<b>Spis Listingów .....</b>	<b>79</b>
<b>Spis Załączników .....</b>	<b>80</b>



## 1. Wstęp

Wszystkie pojazdy aktualnie posiadają jakiegoś rodzaju czujniki czy komputery które pozwalają na dokładną analizę zachodzących akcji oraz reakcji. Najpopularniejszym pojazdem z takimi elementami właśnie jest samochód który mimo ogromnych zmian na przestrzeni lat jego zastosowanie nie zmieniło się. Jedyne elementy jakie zostały rozwijane od jego wynalezienia to zwiększenie komfortu użytkownika, zmniejszanie awaryjności elementów w nich stosowanych oraz moc silnika wraz z jej możliwością przeniesienia.

Aktualne pomiary uzyskujemy poprzez zastosowanie całej gamy czujników rozłożonych po całym pojeździe począwszy od monitorowania pracy silnika a skończywszy na takich elementach jak widoczność poza pojazdem jak i również parametry niezwiązane z pracą pojazdu. Co powoli pozwala na eliminację większości czynności wykonywanych przez użytkownika, oraz niestety coraz bardziej zmniejsza jego odruchy automatyczne na zdarzenia losowe które występują podczas jazdy.

Jak wiadomo osoba prowadząca samochód dostaje takie informacje jak prędkość pojazdu, obroty silnika, błędy które powstały podczas jazdy i szereg podstawowych informacji o stanie płynów. Aczkolwiek tych parametrów jest o wiele, wiele więcej. Przykładowo dostajemy informację o ilościach obrotu silnika, lecz w tego skład wchodzi dawka podawanego paliwa, zmiana kąta zapłonu czy też taki element jak ilość podawanego powietrza względem dawki paliwowej. Jak widać ilość parametrów rośnie, a co za tym idzie zwiększa się niedoinformowanie osoby posiadającej pojazd mechaniczny. Z pomocą przychodzą nam wykorzystane w tych pojazdach komputery które posiadają stałą komunikację między sobą wliczając w to stały dostęp do wszystkich potrzebnych nam informacji w szczególności dla mechaników diagnostów oraz tunerów.



## 2. Cel i uzasadnienie wyboru tematu

Głównym celem prowadzonej pracy jest zbudowanie systemu komunikacji pomiędzy człowiekiem a samochodem wykorzystując technologie IoT (ang. Internet of things) oraz stworzenie urządzenia umożliwiającego przekazanie zebranych informacji w przystępnej cenie. Specyfiką dostarczonych nam danych jest brak ich dostępności w normalnych warunkach użytkowania pojazdu. Producenci starają się, aby coraz mniej istotnych informacji zostało podawanych użytkownikowi nie zważając na ich znaczenie w całokształcie korzystania z nich. Dokładne pomiary oraz dane uzyskane w warunkach spoczynku pojazdu, jak i podczas jego obciążenia znacząco się różnią od siebie i ich analiza może pozwolić na uniknięcie usterek oraz dodatkowych kosztów w przypadku wcześniejszego wykrycia nieprawidłowości działania któregoś z komponentów. Powyższe stwierdzenia pozwalają postawić tezę pracy: Możliwe jest możliwe zbudowanie urządzenia zdolnego do odczytu informacji z komputera samochodu poprzez wykorzystanie tanich technologii IoT do szybkiej ich analizy.

W normalnych warunkach osoba posiadająca samochód i niemająca dostępu do szeregu dodatkowych informacji diagnostycznych dowiaduje się o nie prawidłowości działania poprzez odmienny dźwięk czy zachowanie któregoś z podzespołów, lecz przeważnie jest niemożliwe wcześniejsze przeciwdziałanie, aby wyeliminować problem lub go załagodzić muszą one być monitorowane poprzez wizyty u mechaników.

Kiedy mamy szczegółowe informacje oraz stale je monitorujemy możemy znaleźć odmienne zachowania, zanim znajdą one skutek w postaci uszkodzeń. Przykładem może być spalanie stukowe silnika gdzie zapłon występuje w niechcianych oraz nieprzewidywalnych momentach, nie wiedza na ten temat prowadzi do poważnych zniszczeń w wyniku zaistniałych naprężeń użytych komponentów w silniku, w ostateczności brak szybkiej reakcji ze strony użytkownika może skutkować śmiercią jednostki napędowej.

Wybór tematu pracy ma na celu uzmysłowienie posiadaczom pojazdów o braku dostarczanych im dodatkowych informacji oraz ich dostępność w nieinwazyjny sposób bez konieczności nadmiernej ilości okablowania wykorzystując do tego łączność bezprzewodową w systemie IoT, co poszerza horyzont możliwych rozwiązań.



Dodatkowo ma na celu analizę możliwości komunikacji pomiędzy człowiekiem a maszyną, a także interpretację uzyskanych danych w taki sposób, aby były one czytelne i zrozumiałe dla użytkownika systemu.

### **3. Komputery pokładowe oraz interfejsy komunikacyjne wykorzystywane w pojazdach**

#### **3.1. ECU**

Sterownik silnika ECU (ang. *Engine Control Unit*) jest odpowiedzialny za działanie całego silnika wraz z jego osprzętem takim jak pompy czy urządzenia potrzebne do kontroli działania jednostki napędowej. Komputer ten bazuje na serii informacji począwszy od ciśnień w układzie paliwowym oraz dolotowym, jak i na przykład parametrach spalin wychodzących z samochodu, ponieważ to dzięki nim liczony jest AFR (ang. *Air to Fuel Ratio*).

Polega to na odczycie spalin poprzez sondę Lambda i na jej współczynniku. Jeżeli mamy wartość poniżej 1 oznacza to, że w mieszance paliwowo-powietrznej jest mniej tlenu niż jest to potrzebne do wykorzystania w pełni ilości paliwa (Mieszanka jest za bogata). Podobnie jest w przypadku kiedy lambda jest wyższa niż wartość 1 oznacza to ubogą mieszankę w zależności powietrza jest więcej niż paliwa. Idealna mieszanka występuje w wąskim przedziale wartości sondy lambda która wynosi od 0,997 do 1,003 to właśnie w tym przedziale występuje spalanie stechiometryczne (Polega to na mieszance paliwowej w składzie 1 kg spalonego paliwa przypada 14.7 kg powietrza, jest to skład idealny wykorzystujący całość możliwości paliwa). Jest to bardzo istotny element, ponieważ w większości nowszych samochodów jest on wykorzystywany do analizy i interpretacji zaistniałych problemów z brakiem mocy silnika czy ewentualnym jej podniesieniem.

Dajmy na to, że silnik w pewnym przedziale obrotów potrzebuje większej mocy, ECU pobiera wtedy informacje jaki jest stan AFR, jeżeli jest lekko odchylony silnik spróbuje dobrać odpowiednią dawkę, następnie sprawdzi jaki jest kąt wyprzedzenia zapłonu oraz kąty na wałkach wylotowych i ssących. Dodatkowo jeżeli silnik posiada



turbinę stara się odpowiednio zbić powietrze dochodzące z zewnątrz na cel wykorzystania spalin z turbiny. Oczywiście w skład całej operacji wchodzi regulacja ciśnień układu chłodzącego, jak i pompy olejowej dla jak najmniejszego naprężenia bloku silnika.

ECU posiada również dodatkowe wykorzystanie w nowych samochodach. Chodzi tutaj o system ASG (ang. Automated Shift Gearbox). Jest to wszechstronny system sterujący pracą skrzyni biegów, działa ona podobnie jak manualna skrzynia, lecz posiada ona cechy skrzyni automatycznej. W skład komputerów wykorzystywanych w tym systemie możemy wliczyć ECU oraz Komputer ABS (ang. Anti-lock braking system) czy ESP (ang. Electronic stability program). Dzięki wykorzystaniu ich możemy używać samochodu który nie posiada pedału sprzęgła, ale możemy zmieniać manualnie biegi według naszego uznania (Oczywiście jest system zabezpieczający przed nadmiernym przeciążeniem skrzyni i w tym przypadku skrzynia zmieni sama bieg na bardziej jej dogodny). Wiele podstawowych informacji dostarczają nam publikacje WKŁ [21] oraz [22] ponieważ zawierają opisy podstawowych akcji i reakcji zachodzących w komorze silnika.

## **3.2. System ABS**

Komputer ABS pierwotnie został stworzony do systemu ABS, lecz znajduje on wiele innych zastosowań w przeróżnych systemach takich jak ASR (niem. *Antriebsschlup-fregelung*), ESP czy systemy Start-stop. Praktycznie wszystkie elementy elektroniczne które próbują wpłynąć na działanie hamulców biorą część funkcjonalności z komputera ABS. A jest on odpowiedzialny za kompensację ciśnień w przewodach hamulcowych w taki sposób, aby zminimalizować poślizg kół podczas hamowania w celu zwiększenia kontroli użytkownika nad pojazdem. Szczegółowe informacje możemy przeczytać w książce J. Haynesa [16].

### **3.2.1. System TCS**

Jest to bliźniaczego przeznaczenia system opracowany na podstawie systemu ABS. On również nie dopuszcza do nadmiernego poślizgu kół, lecz w tym przypadku nie podczas hamowania a przyspieszania pojazdu w celu zwiększenia panowania nad





samochodem, kiedy napotykamy oblodzoną nawierzchnię. Oczywiście system ten znalazł również zastosowanie w szybszych samochodach, ponieważ działa on na suchych nawierzchniach i równomiernie rozprowadza energię na koła tam, gdzie jest to potrzebne więc co za tym idzie zwiększa się kontrola pojazdów podczas wykonywania znacznego przyspieszenia na suchej nawierzchni. Został on zaadaptowany przez różne marki które przeróżnie je nazywały począwszy od TCS (ang. *Traction control system*) po ASR aż do DSC (ang. *Dynamic stability control*) które wykorzystuje marka BMW (niem. *Bayerische Motoren Werke*).

### 3.2.2. System ESP

Jest to kolejny system z dziedziny układu hamulcowego i jest on bardzo podobny w działaniu jak system TCS, lecz jego zadaniem jest nie branie pod uwagę śliskich nawierzchni a ostrych zakrętów czy śliskich zakrętów. ESP (ang. *Electronic stability control*) w odróżnieniu od innych nie jest stosowany w motosporcie, ponieważ zmniejsza on znacznie prędkości podczas wyjścia z zakrętu oraz nie pozwala on na kontrolowaną jazdę nadsterowną w poślizgu. Podobnie jak w TCS jest wiele odmian nazewnictwa tego systemu, ponieważ różne marki inaczej je nazywały ze względów komercyjnych bardziej niż inne systemy wspomagające jazdę. W kręgach fachowców jego nazwa przyjęta została jako ESC (ang. *Electronic stability control*).

### 3.3. FIS

FIS (niem. *Fahrer Information System*) został przyjęty jako oddzielny komputer występujący wraz z elektroniką zegarów w samochodach grupy VAG (niem. *Volkswagen AG*). W jego wyposażenie wchodzi takie elementy jak ekran, w zależności od wersji 1/2 FIS lub FULL FIS na którym możemy podejrzeć elementy dodatkowe jak aktualne błędy czy nieprawidłowości, niedomknięte drzwi, różne odmiany spalania, temperatura na zewnątrz czy komunikaty z nawigacji. Drugim elementem w zestawie jest specjalnie przystosowana manetka do obsługi całości systemu, ponieważ jest on niezależny od całości samochodu i jedynie uzyskuje dane poprzez zapytania. Na manetce posiadamy przyciski plus oraz minus, a także reset. Jego nowszy system posiada również połączenie z interfejsem MMI (ang. *Multi Media Interface*) co pozwala na jeszcze szersze spektrum zastosowań tego rozwiązania.



### 3.4. Interfejsy komunikacyjne

Dość bardzo istotnym elementem całości systemów jest komunikacja pomiędzy wszystkimi komputerami w sieci pojazdu. W większości przypadków jest to komputer ECU spięty z modułem od hamowni zależności jaki znajduje się w samochodzie, dodatkowo posiadamy komputer zwany „modułem komfortu” który odpowiada za elementy związane z czynnościami zwiększającymi komfort osób przebywających w pojeździe.

Istnieje jednak wiele metod komunikacji podzespołów w pojazdach, ponieważ rozwijały się one na przestrzeni lat. Używany do dziś i najstarszy z nich wszystkich jest protokół LIN (ang. *Local interconnect network*). Został on opracowany w 1 lipca 1999 roku a ostatnia odsłona została wprowadzona 24 grudnia 2006 roku. Jego najnowszym bratem wykorzystywanym w nowoczesnej komunikacji jest standard CAN (ang. *Controlled area network*). Każdy standard komunikacji nie jest kompatybilny z innym w pełni. Do działania równoległego potrzebują interpreterów.

#### 3.4.1. Magistrala CAN

Jest to szeregową magistrala komunikacji która powstała oraz została opracowana we wczesnych latach 80 przez firmę która aktualnie nazywa się BOSCH. Od samego początku brała na cel dziedzinę motoryzacji w celu usprawnienia komunikacji pomiędzy wszystkimi elementami. Aczkolwiek system ten tak się dostosował do rynku, że jest wykorzystywany nie tylko w dziedzinie motoryzacji, ale także w takich działach jak medycynie oraz w kolejnictwie. System ten w przeciwieństwie do innych nie działa na wydzielonej jednostce nadrzędnej co nadaje mu klasę multi-master (Każdy element sieci jest swoim panem przez co nie ma urządzeń niewolniczych).

Komunikacja odbywa się na zasadzie radia gdzie każdy z elementów sieci jest jako nadajnik i odbiornik, jednocześnie nadając swoje pakiety informacji oraz odbierając wiadomości z innych elementów. Magistrala oczywiście posiada swoje granice których nie może przekroczyć a jest nią liczba elementów wpiętych jako nadajniki. W aktualnym stanie jesteśmy zdolni wykorzystać 30 jednostek, lecz należy pamiętać, że „technologia która dziś jest przyszłością jutro może zostać przeżytkiem”.



System przesyłu jest oparty o 8 bitów danych, a cała komunikacja odbywa się na 12 bitach podzielonych na 7 pól. Są to następująco: początek, arbitraż, sterowanie, dane, suma kontrolna, potwierdzenie oraz koniec. Szyna CAN posiada również zaletę w postaci „czystszej” okablowania, ponieważ wymaga tylko dwóch przewodów do połączenia elementów gdzie jest CAN wysoki oraz niski. Daje to stosunkowo większą swobodę producentom sprzętu do bezpiecznego ułożenia przewodów, bez obaw o przepełnienie pojazdu zbędną ilością instalacji.

Działanie szyny CAN odbywa się na przedziale od 0 do 5 volt gdzie 2.5 volta jest złotym środkiem. System ten potrafi działać na dwa sposoby, szybki i wolny. Działanie w trybie szybkim odbywa się poprzez podzielenie linii CAN wysokiego i niskiego tak, aby nigdy nie przekraczały wartości 2.5 volta i nie mieszały się. Dzięki temu możemy uzyskać szybki odczyt. Zbliżenie się do granicy obu linii daje wartość logiczną 1 zaś oddalenie 0. Działanie w trybie wolnym odbywa się w zupełnie inny sposób CAN niski w stanie 1 znajduje się w przedziale 5 volt i analogicznie stan wysoki znajduje się około 0 woltów. Stan logiczny 0 dla tego rodzaju występuje tylko wtedy kiedy oba te stany przekroczą granice 2.5 volta i wymieniają się miejscami. Tryb ten nazywa się wolnym, ponieważ wymaga od systemu większego nakładu energii do wykorzystania w celu zmiany stanu. Gdzie w systemie Szybkim są to tylko skoki napięcia w trybie wolnym jest to próba utrzymania się poza granicą wartości.

Każdy komputer czy element wykorzystujący szynę CAN jest wyposażony w kontroler tego protokołu który łączy się z nadajnikiem szyny CAN, niestety bez danego połączenia urządzenie podłączone do sieci CAN nie zadziała i nie będzie wspierane przez ten system, ponieważ cała komunikacja oraz wpłatanie danych w sieć oparta jest na synchronizacji płynących pakietów, jeżeli dajmy na to element A chce wysłać wiadomość, ale zrobił to element B to A musi poczekać na bit końca, aby wpleść swoją sekwencję do sieci (Opiera się to również o pulę adresową gdzie mniejszy numer ma pierwszeństwo). Występuje to w dwóch przypadkach jednym z nich jest skończenie się przekazywanego ciągu danych, drugi zachodzi, wtedy kiedy któryś element przestanie nadawać dajmy na to poprzez złą sumę kontrolną. Szyna nie czeka na wypełnienie reszty bloku tylko dostaje zielone światło na wpuszczenie nowego pakietu danych,



Elementy w sieci CAN posiadają swój unikatowy identyfikator pozwalający na decyzję o przeprowadzeniu kolejki wpisu danych w transmisję. Przykładowo jeżeli element o numerze 15 oraz 17 chcą wysłać wiadomość w tej samej chwili pierwsza wiadomość zostanie wysłana przekazana od numeru 15, lecz jeżeli podczas oczekiwania pomiędzy pojawi się numer 16 będzie on miał prawo pierwszeństwa. Więcej na dany temat możemy przeczytać w książce [1] od strony 59 lub też w [8] czy w [15] gdzie proces komunikacji jest bardziej szczegółowo opisany. Rozwinięcie standardu komunikacji szyny CAN przedstawia książka [26].

### 3.4.2. Protokół SENT

Protokół ten nosi pełną nazwę SAE J2716 SENT (ang. *Single Edge Nibble Transmission*) jest to protokół o schemacie od punkt do punktu, znaczy to, że element nadający posiada tylko jedno połączenie z elementem końcowy i w tym samym czasie może wykonywać się jedno połączenie. Z pomocą przychodzi infrastruktura wykorzystana w tym protokole, polega ona na wykorzystaniu trzech przewodów gdzie kolejno pierwszy jest odpowiedzialny za linię sygnałową (Może ona pracować w dwóch trybach, niskim oraz wysokim jest to zależne od użytego napięcia w niskim wykorzystujemy mniej niż 0,5 V zaś w trybie wysokim wartości zaczynają się od 4.1V), drugi odpowiada za linie zasilania 5 V, natomiast ostatni za masę. SENT wykorzystuje sygnał PWM (ang. *Puls width modulation*) aby zakodować 4 bity.

Standardem w tym protokole jest długość sygnałów danych pomiędzy 3 a 90 mikrosekundami to właśnie w tych odstępach czasowych występuje przesył informacji na podstawie podniesienia napięcia prądu. W zależności od obranej strategii możemy podzielić przesył danych na 32, 24, 20, 12 oraz 4 bitowe. Gdzie jeden puls napięcia daje nam znać jaką wartość posiada i wynosi ona 4 bity. Biorąc to pod uwagę wartość całej wiadomości mieści się maksymalnie w 8 znakach (pulsach napięcia). Dane są interpretowane jako długość występowania spadku i ponownego skoku napięcia, interpretując te wartości otrzymujemy Półbajty w ich najczęstszej odmianie, czyli oktetch. Półbajt ma długość 4 bitów dzięki czemu może przybrać 16 różnych wartości, co prowadzi nas do zapisu systemu szesnastkowego.



Protokół SENT podczas przechodzenia pomiędzy pakietami danych posiada dwa dodatkowe elementy jednym z nich jest puls przerwania zaś drugim puls synchronizacji, gdzie w pierwszym nie posiadamy ściśle określonej reguły przez jaki czas on powinien być w stanie wysokim tak w pulsie synchronizacji mamy ściśle określony czas który wynosi pomiędzy 671.5 do 673 mikrosekund.

Dodatkowo protokół ten posiada również powolny kanał dla wartości diagnostycznych które posiadają większe ilości danych, lecz nie są one stale potrzebne. Wysyłanie danych polega na tej samej zasadzie, niemniej jednak jest tych danych znacznie więcej. W szybkim kanale maksymalny zasób informacji jaki możemy wysłać to 8 półbajtów wszak, w kanale wolnym możemy wysłać nawet ich 16, co za tym idzie kosztem prędkości odczytu informacji zyskujemy ich objętość.

### **3.4.3. Protokół KWP**

KWP lub inaczej KWP2000 (ang. Keyword Protocol) jest jednym z częściej wspieranych protokołów wykorzystywanych przez producentów samochodów, a to dlatego, że jest on w stanie działać również w standardzie magistrali CAN, albowiem również wykorzystuje do komunikacji podzespołów tylko dwa przewody, lecz w przeciwieństwie do niego musi on zostać lekko zmodyfikowany przez interpreter, aby komunikacja mogła się odbywać bez zakłóceń i przepelnienia niepotrzebnymi informacjami. Nie działa jednak to w stronę przeciwną, nie możemy w żaden sposób sprawić, aby magistrala CAN działała na zasadach protokołu KWP.

Przewody działają w trochę inny sposób niż w przypadku magistrali CAN oraz opisane są jako linia K wysoka oraz niska. Linia wysoka przenosi wszystkie informacje w takim samym standardzie jak działają ówczesne sieci internetowe, linia niska w tym przypadku jest wykorzystywana jako element który wzbudza komunikację pomiędzy dwoma elementami, kiedy jej stan jest wysoki. Przesył informacji jest dość szybki i oscyluje pomiędzy 1200 a 10400 Bd (Baud - jednostka transmisji).



### 3.4.4. Magistrala LIN

Szeregowa magistrala LIN (ang. *Local internet network*) została opracowana w 1999 r. Dzięki temu, że w tamtych czasach większe koncerny motoryzacyjne zebrały się, aby stworzyć firmę odpowiedzialną za stworzenie jednolitej i uniwersalnej komunikacji pomiędzy pojazdami a mechanikami. Od samego początku była ona przystosowana do zastosowań motoryzacyjnych w celu zastąpienia magistrali CAN w miejscach, gdzie występował przerost formy nad treścią. W przypadku magistrali CAN możemy osiągnąć przesył danych w wysokości 1Mb/s, lecz wymaga ona dość kosztownych komponentów do działania, inaczej jest w magistrali LIN gdzie maksymalny przesył to 20kb/s, za to w jej topologii nie potrzebne są rezonatory dla modułów pracujących w trybie nasłuchu oraz rozszerzenie sieci nie wymaga zmiany modułów i programu na reszcie urządzeń. Dodatkowo moduł odpowiedzialny za działanie nadrzędne sieci LIN może być jednocześnie elementem sieci CAN.

Działanie tej magistrali odbywa się na ramkach danych, ponieważ system działa na zasadzie stanu wysokiego i niskiego system ten posiada ściśle określone reguły zachodzących operacji. Ramka LIN jest złożona z nagłówka i odpowiedzi.

#### **Do nagłówka zaliczamy:**

- SYNCH BREAK
- SYNCH FIELD
- IDENT FIELD

#### **Zaś dla odpowiedzi posiadamy 2 pola:**

- DATA FIELD
- CHECKSUM FIELD

To właśnie dzięki tym nagłówkom system interpretuje odpowiednie informacje. Przykładowo IDENT FIELD (Pole identyfikacji) składa się z szeregu zmiany stanu na przemian z 0 na 1 dzięki czemu uzyskujemy wartość np. 0x55. Ramka takiej informacji może wyglądać następująco:



Dostajemy dokładną informację który element nadaje może to być element o numerze PID (ang. *Process Identifier*) 0x15 który wysłał dane o wartości 0x34, 0x55, 0x67, 0x97. Suma kontrolna będzie liczona na zasadzie dodania tych wartości i sprawdzenia, czy są mniejsze czy większe niż największa wartość puli, jeżeli jest większa wtedy odejmujemy tą wartość od sumy a na koniec dodajemy wartość PID. W tym przypadku dodanie wszystkich będzie skutkowało uzyskaniem wartości powyżej 256 (0x100) ponieważ suma wyniesie 0x187 co za tym idzie od tej wartości odejmujemy 255 (0xFF) daje to nam 0x88, następnie dodanie wartości PID i uzyskujemy finalną sumę kontrolną 0x9D dla wartości wysłanych danych z elementu o PID 0x15.

Dodatkowo system ten cechuje się funkcją oszczędzania zasobów na zasadzie uśpienia magistrali w przypadku nie wysyłania wiadomości przez określony czas (Jest on zależny od ustawienia szybkości przesyłu magistrali). Aby wzbudzić magistralę w danym punkcie musi zostać nadana ramka WAKE UP o wartości 0x80 ,a jest ona niczym więcej niż chwilowym zwarciem do masy.

Więcej szczegółowych informacji dostarczyć może odnośnik [10], książka Michała Kardasia [15] lub książka Andreasa Grzemby [17].



## 4. Użyte rozwiązania oraz technologie

### 4.1. Interfejs OBD

Interfejs OBD (ang. *On Board Diagnostic*) powstał w 1980 jako system, lecz jego załóżki sięgają roku 1968 gdzie Volkswagen użył pierwszego komputera wbudowanego w samochód do analogowego sterowania wtryskiwaczy paliwa. OBD samo w sobie wynalazł General Motors i składał się on z systemu 5 wyjściowego ALDL (ang. *Assembly line diagnostic link*) gdzie bardzo szybko doznał uznania wśród osób naprawiających samochody oraz lubiącym z nimi eksperymentować. W tamtych czasach całość pracowała na zasadach działania sygnałów PWM. W kolejnych latach gniazdo zmieniło nie tylko swój wygląd, ale również swoje działanie. Gniazdo OBD w standardzie pierwszym pozwalało wyłącznie na prace w standardzie KWP oraz składało się z czterech wyjść masy, zasilania, linii K oraz L.

Aktualne gniazda OBD są bowiem w standardzie drugim, a cały system zmienił nazwę na OBD 2. Obsługuje on oraz umożliwia wykorzystanie praktycznie każdego standardu komunikacji z podzespołami samochodu. Obsługuje on takie standardy jak:

- SAE j1850 PWM (ang. *Pulse width modulation*)
- SAE j1850 VPW (ang. *Variable pulse Width*)
- ISO 9141-2 (Jest to protokół asynchroniczny samochodów ze stajni Chrysler, samochodów europejskich oraz wypuszczanych na rynek azjatycki)
- ISO 14230 KWP2000 (ang. *Keayword Protocol 2000*)
- ISO 15765 CAN (ang. *Controlled area network*)



Schemat wyprowadzeń oraz ich standardy w złączu OBDII wygląda następująco:



Rys. 4.1. Schemat wyjść gniazda OBDII

*Źródło: [1]*

Dzięki nim OBD 2 stało się jednym z najczęściej używanych systemów diagnostycznych pojazdów na całym świecie. Co najciekawsze wszystkie inne systemy diagnostyczne dają się przerobić w prosty sposób, aby działały one na gniazdach OBD II a jest to zależne od gniazda wykorzystywanego do diagnostyki. BMW bowiem wykorzystuje również protokoły OBD, lecz mimo tego, że jego gniazdo nie wygląda tak samo jak ono, do działania potrzebna jest przejściówka z okrągłego gniazda na gniazdo OBDII. Taki sam trik wykorzystują inne koncerny zmieniając kształt gniazda na swoje upodobanie, lecz wszystkie one i tak schodzą się do standardu OBDII.

Do analizy tego systemu wykorzystuje się przeważnie komputera wyposażonego w przejściówkę dekodującą odpowiednie sygnały dochodzące z gniazda OBD, dodatkowo są one wyposażone w analizatory jaki jest potrzebny schemat dekodowania danych wykorzystywany przez samochód. Czasami możemy również znaleźć wariant gdzie to komputer oraz oprogramowanie dostarczone do niego pełni funkcję dekodera danych.

Na rynku można również uświadczyc specjalnie przygotowane bezprzewodowe interpretatory które łączą się z samochodem we własnym zakresie i działają one w pętli

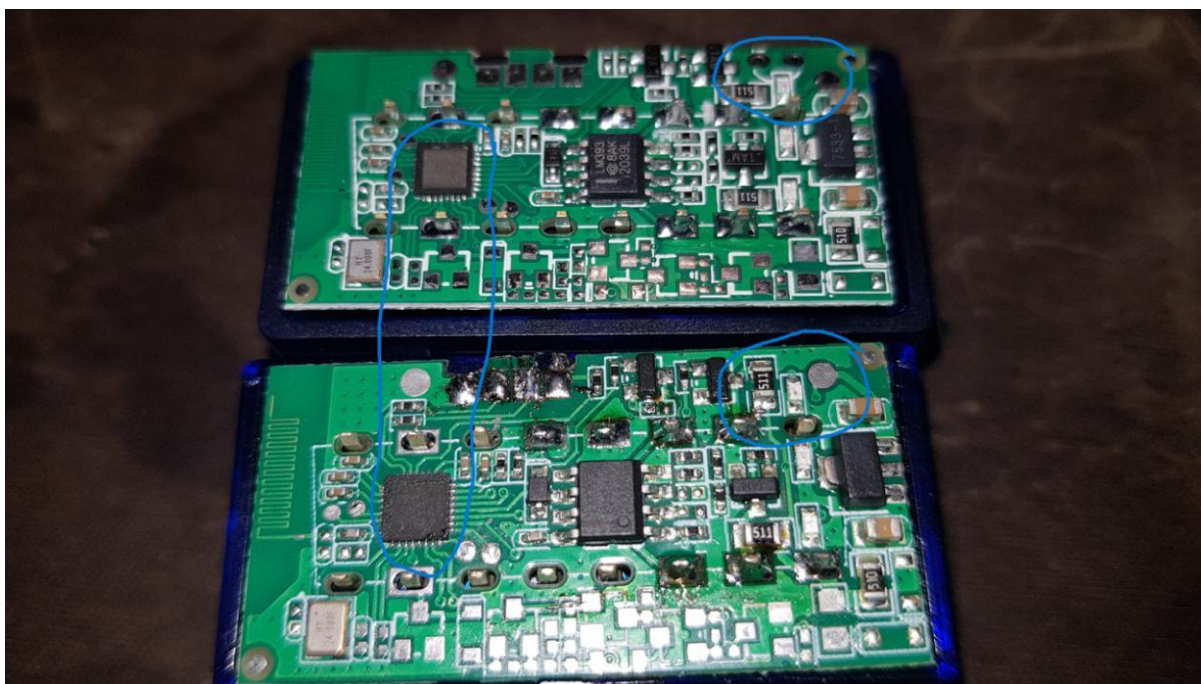


aż do uzyskania sygnału o potrzebie wykorzystania części informacji. Jednym z takich produktów jest interfejs ELM327 który posłużył do wykonania urządzenia testującego możliwości interpretacji danych samochodów oraz ich ograniczenia.

## 4.2. Interpreter ELM327

Mikrokontroler ELM327 został stworzony przez firmę ELM Electronics dla translacji standardu OBD dla wszystkich możliwych nowoczesnych samochodów. Jest on jednym z najbardziej rozpoznawalnych na rynku takich produktów i prawie każda jego odłona została skopiowana i wykorzystana w inny sposób. Oryginał został zaprojektowany na układzie PIC18F2480 który oferuje wiele możliwości oraz prostą komunikację UART (ang. *universal asynchronous receiver-transmitter*). Powstały 3 odmiany tego układu, dwa warianty po 28 wyjść oraz jeden z 40 wyjściami pozwalający na poszerzenie układu oraz komunikację z drugim podobnym układem lub mniejszym.

Większość aktualnie dostępnych produktów jest to ich podróbka, lecz niekiedy może nam się poszczęścić i trafić oryginalny układ. Dodatkowo oryginał od podróbki możemy rozpoznać, że posiada dwie płytki PCB (ang. *printed circuit board*), natomiast podróbki tylko jedną. Przykładami mogą być zakupione do testów egzemplarze od sprzedawców z różnych portali które wyglądają następująco:



Rys. 4.2. Nieprawidłowości zamówionych układów

*Źródło: Opracowanie własne*

Ciekawym dodatkowym elementem jest to, że podrabiane interfejsy posiadają zeszlifowaną górną część mikrokontrolerów w taki sposób, aby nie dało się odczytać oznaczeń (Opcjonalnie są one zalane twardą czarną żywicą) możemy to zauważyć na powyższym zdjęciu, dodatkowo płytki nie różnią się prawie niczym oprócz prawej górnej strony gdzie inaczej jest zaprojektowana sekcja zasilania układu. Druga z płytek w oryginalnym ELM327 jest przeznaczona na układ od komunikacji bezprzewodowej, przeważnie znajduje się na niej zmodyfikowany układ HC-05 lub HC-06 w taki sposób, aby były możliwe połączenia Bluetooth oraz BLE (ang. *Bluetooth Low-Energy*). Wszystkie informacje odnośnie do oryginalnego urządzenia dostarcza producent pod danym odnośnikiem [8]. Moduły te są zdolne do komunikacji z układami rodziny ESP które zostały wykorzystane w budowie urządzenia.



### 4.3. ESP8266 oraz ESP32

Microchipy ESP zaprojektował Chiński koncern Espressif Systems od którego zawdzięczają nazwę. Zostały one bardzo szybko spopularyzowane przez miłośników elektroniki jako tanie rozwiązania sieci Wi-Fi, ponieważ posiadają wbudowany system TCP/IP (ang. *Transmission Control Protocol*). Na skutek tego powstały mikrokontrolery ESP8266 oraz ESP32, które później zostały zaadaptowane przez innych dystrybutorów, gdzie nie rzadko były one wzbogacane o nowe funkcje takie jak obsługi baterii lub pakietów bateryjnych do zastosowań mobilnych, odczyty danych GPS (ang. *Ground positioning system*) czy też wyposażenie w ekran LCD (ang. *Liquid crystal display*). Dzięki swoim kompaktowym rozmiarom ich wykorzystanie jest możliwe w przeróżnych projektach wymagających jak najmniejszej przestrzeni.

Model ESP8266 został stworzony jako pierwszy i posiada tylko możliwości łączenia się przez sieć. Wiele informacji na temat tego układu oraz jego zastosowań możemy znaleźć w książce [2]. Drugi model to ESP32 który nie tylko ma możliwość łączenia się z sieciami bezprzewodowymi, ale również i z urządzeniami posiadającymi Bluetooth lub BLE. Mimo tego, że ten pierwszy nie charakteryzuje się możliwością łączenia z Bluetooth dalej znajduje swoich zwolenników ze względu na cenę, co pozwala na tańsze rozwiązania modelu sieci, a więcej na ten temat możemy przeczytać w [24]. Zaletą tych układów jest możliwość połączenia ich poprzez bibliotekę ESP-NOW która pozwala na stworzenie dowolnego schematu sieci bez konieczności podłączania któregośkolwiek z nich do Internetu. Tworzymy wtedy izolowaną sieć IoT (ang. *Internet of things*) bez możliwości zdalnego dostępu do niej.

Wiele informacji o programowaniu oraz zastosowaniu modułów Arduino i ESP możemy znaleźć w książce Arduino Cook [13] oraz publikacji Neila Camerona [23].



#### 4.4. Działanie sieci ESP-NOW oraz MQTT

Utworzone urządzenie zostało oparte o bibliotekę Arduino wykorzystującą oba typy mikrokontrolerów ESP, a jest nią właśnie ESP-NOW. Pozwala ona na stworzenie bezpołączeniowej sieci p2p (ang. *Peer to peer*). Protokół ten powstał dzięki Espressif, który utworzył całą listę reguł oraz połączeń polegających wyłącznie na adresach MAC (ang. *Media access control*). To właśnie dzięki nim możemy dostarczać chciane przez nas informacje do wybranych urządzeń, a co najciekawsze bez konieczności uwierzytelniania ze strony urządzenia nadającego (Choć opcja ta jest jak najbardziej do wykorzystania). ESP-NOW może działać w wielu topologiach sieci takich jak topologia gwiazdy, pierścienia czy też topologia hierarchiczna. Zaletą tych połączeń jest ich trwałość, przeprowadzone testy przez wydawcę biblioteki ukazują, że w środowisku o średniej urbanizacji i mało zakłócającym środowisku możemy spodziewać się połączenia na odległości nawet do 140 metrów bez utraty prędkości. Całość systemu pracuje w paśmie 2.4 GHz. Większa porcja informacji znajduje się na [9].

Dodatkowym atutem jest to, że może ona działać podobnie jak pierwotna w zamyśle do wykorzystania komunikacja, czyli sieć typu MQTT (ang. *MQ Telemetry Transport*). Jest to lekki protokół transmisji danych oparty o publikacje i subskrypcje z dodatkowym elementem brokera danych. Posiadając brokera danych możemy dowolnie formować sieć, ponieważ działa on jak dyspozytornia, dysponując wszystkimi danymi które do niego trafiają. Broker posiada również funkcje zapisu każdej wysłanej informacji do urządzenia w formie subskrybenta.

Minusem bądź plusem jest to, że każdy z elementów tej sieci musi być spięty do Internetu w celu poprawnego działania co daje możliwość odczytu z każdego miejsca na ziemi gdzie jest dostęp do Internetu, lecz naraża to osobę korzystającą na wyciek informacji poprzez brokera danych. Na rynku najbardziej rozpoznawalnymi brokerami danych są Mosquitto, RabbitMQ, HiveMQ, IBM MessageSignt, EMQ X czy też VerneMQ. Serwisy te mają za zadanie dbać o bezpieczeństwo danych przesyłanych tą metodą. ESP-NOW posiada funkcje umożliwiającą również połączenie do niektórych z podanych brokerów danych. Wszystkie z wymienionych sposobów komunikacji łączy to, że każda z nich jest w jakimś stopniu połączona z komunikacją IoT.



## 4.5. Komunikacja IoT

IoT, czyli Internet rzeczy - koncepcja wedle której wszystkie elementy znajdujące się w jednej części projektu są ze sobą połączone by bezpośrednio gromadzić, przetwarzać oraz wymieniać dane za pośrednictwem sieci komputerowej. Koncepcja ta znajduje zastosowanie w przemyśle przetwórczym, zarządzaniu miastami oraz nawet drobnymi projektami. Dzięki wykorzystaniu sieci komputerowej możemy w prosty i szybki sposób wysłać dane z punktu A do punktu B.

**Zagadnienie to obejmuje elementy takie jak:**

- Systemy osadzone
- Urządzenia inteligentne
- Jednostki MCU (ang. Master Control Unit)
- Jednostki MPU (ang. Memory Protection Unit)
- Urządzenia nieprzetwarzające danych
- Czujniki

### **Bramy w systemie IoT:**

Bramy są częścią IoT, dzięki nim można łączyć urządzenia IoT z chmurą danych. Chociaż nie wszystkie urządzenia IoT wymagają bram, mogą służyć do ustanawiania komunikacji między urządzeniami lub łączyć urządzenia, które nie są oparte na protokole IP i nie mogą łączyć się bezpośrednio z chmurą. Dane wysyłane z urządzeń są przerzucane przez bramę, wstępnie przetwarzane, a następnie wysyłane do chmury.

Dzięki korzystaniu z bram można zmniejszyć czas potrzebny do wysłania i otrzymania danych oraz ich rozmiar. Umożliwiają również łączenie urządzeń, które nie mają dostępu do Internetu, oraz zapewniają dodatkową ochronę danych przesyłanych w obu kierunkach.



## Typy sieci IoT

Istnieją dwa typy sieci IoT są to sieci krótkiego zasięgu o małej mocy oraz sieci rozległe małej mocy

### 1. Sieci krótkiego zasięgu oraz małej mocy:

- **Bluetooth** - Komunikacja krótkiego zasięgu o stosunkowo szybkim przesyłaniu danych
- **NFC** - Komunikacja zbliżeniowa która działa w maksymalnej odległości 4 cm od nadajnika.
- **Wi-Fi/802.11** - Aktualny standard komunikacyjny wykorzystywany w większości domów oraz biur.
- **Z-Wave** - Sieć w topologii siatki używająca fal radiowych o niskiej energii do komunikowania się pomiędzy urządzeniami.
- **Zigbee** - Komunikacja oparta o niskiej częstotliwości falach radiowych. Wykorzystuje ona małe nadajniki do stworzenia osobistej sieci lokalnej.

### 2. Sieci rozległe małej mocy:

- **4G LTE IoT** - Wysoka pojemność i małe opóźnienia - te sieci są stworzone do scenariuszy IoT, które wymagają informacji w czasie rzeczywistym.
- **5G IoT** - Standard ten jest powoli wprowadzany do użytku. Jest on rozwojem wcześniejszego 4G LTE, w tym przypadku zwiększono przepustowość oraz zasięg sieci tego standardu
- **Cat-0** - Te sieci oparte o standard LTE są najtańszą opcją. Tworzą one podstawę dla funkcjonowania standardu Cat-M.
- **Cat-1** - Ten standard sieci komórkowej IoT zastąpił technologię 3G wprowadzając nowy wariant sieci 4G LTE. Sieci Cat-1 są łatwe do konfigurowania i oferują doskonałe rozwiązanie dla aplikacji.
- **LoRaWAN** - System komunikacyjny bezprzewodowej dalekiego zasięgu, przeznaczony on jest do połączeń pomiędzy urządzeniami IoT. Maksymalnym





zasięgiem pomiędzy urządzeniami sieci LoRaWAN (ang. *Long Range Wide Area Network*) a stacjami bazowymi wynosi średnio od 10 do 15 kilometrów, lecz w roku 2020 został ustanowiony nowy rekord odległości komunikacji który wynosił 832 kilometry, do działania na takim dystansie potrzebne były odpowiednie warunki propagacji atmosferycznej.

- **LTE Cat-M1** - Te sieci są w pełni zgodne z sieciami LTE. Optymalizują koszty i moc w drugiej generacji mikroukładów LTE zaprojektowanych specjalnie dla aplikacji IoT
- **Sigfox** - To globalny dostawca sieci IoT oferuje on urządzenia sieciowe o możliwościach ciągłej komunikacji średniej prędkości, lecz o wysokiej żywotności i odporności na zakłócenia.

Wszystko to i o wiele więcej przedstawia książka Francisa daCosta [5] oraz publikacja [27] wraz z odnośnikiem [12].

#### 4.6. Raspberry PI oraz SQLite

Mikrokomputer znacząco różni się od mikrokontrolera gdzie ten drugi wymaga do działania jedynie prądu co znaczy, że jest w pełni autonomiczny. Komputery takiego typu jak Raspberry PI wymagają o wiele więcej elementów, w skład elementów bazowych możemy zaliczyć takie rzeczy jak stabilizatory zasilania, układy wejścia wyjścia, mikroprocesor oraz Pamięci RAM i ROM. Całość systemu mikroprocesora i komponentów komunikuje się przez różnego rodzaju szyny o różnym przeznaczeniu. Działają one jak przepusty danych które pracują bezkolizyjnie w obu kierunkach. Raspberry PI nie tylko przedstawia cechy mikrokomputera, ale również mikrosystemu. Cechuje się on możliwością instalacji specjalnie spreparowanych wersji systemów opartych w początkowych wersjach produktu wyłącznie o strukturę Linuxa, lecz w późniejszych odsłonach możemy również ustrzec systemy takie jak Windows 10 IoT Core lub też Android TV, co oznacza, że system jest zdolny do operowania na dość dużej rozpiętości systemów.



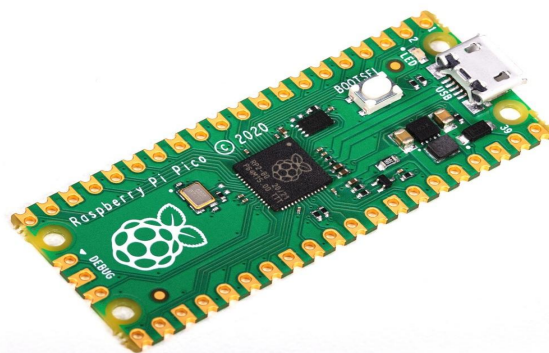


#### 4.6.1. Mikrokomputery Raspberry PI

Raspberry PI zostało utworzone, aby wspierać naukę podstaw informatyki (Podobnie jak Arduino). Jego pierwsza odsłona była skromna, lecz na tamte czasy oferowała bardzo dużo, ponieważ miała ona premierę w 2014 roku, a już wtedy układ ten posiadał pełno niespodzianek. Pierwszą z nich był dostępny specjalnie spreparowany system dla tego mikrokomputera, posiadał on szatę graficzną i możliwość wyświetlenia obrazu nie tylko przez złącze HDMI (ang. *High Definition Multimedia Interface*), ale również przez złącze RCA (ang. *Radio Corporation of America*) czy też złącze Jack. Dodatkowo na płycie możemy również odnaleźć złącze zdolne do wyświetlenia obrazu wykorzystując do tego taśmę, co stanowi kolejnym świetnym rozwiązaniem dla kompaktowych projektów.

Sam na swoim pokładzie nie posiada dysku twardego, lecz jest wyposażony w złącze karty MicroSD (ang. *Secure Digital*) które służy za nośnik danych (Starsze wersje posiadały również złącze kart SD), co jest wielką zaletą w przypadkach chęci tworzenia kopii zapasowych systemu czy innych danych (Ma to również zastosowanie w tworzeniu kontrolerów macierzy). Kolejnymi „Asami w rękawie” były złącza GPIO (ang. *general-purpose input/output*), szyna SPI (ang. *Serial Peripheral Interface*) oraz prosta obsługa Sieci z komunikacją UART.

Do tej pory opracowano 3 różne modele Raspberry PI które znacząco się od siebie różnią nie tylko podzespołami, ale również gabarytami. Najmniejsze z nich noszą miano „Pico” oraz „Zero” oba te warianty są dość unikatowe. Raspberry PI Pico przypomina bardziej coś w stylu układów ESP czy Arduino, ponieważ jest to mikrokontroler, a nie jak w reszcie przypadków komputer SBC (ang. *Single-Board Computer*), wymaga on zasilania 5 V przy pomocy Micro USB (ang. *Universal Serial Bus*) lub też zasilania w wysokości od 1.85 do 5 V (Granica błędu jest tutaj 0.05V) podanego na pin zasilający. Układ ten prezentuje następujące zdjęcie:

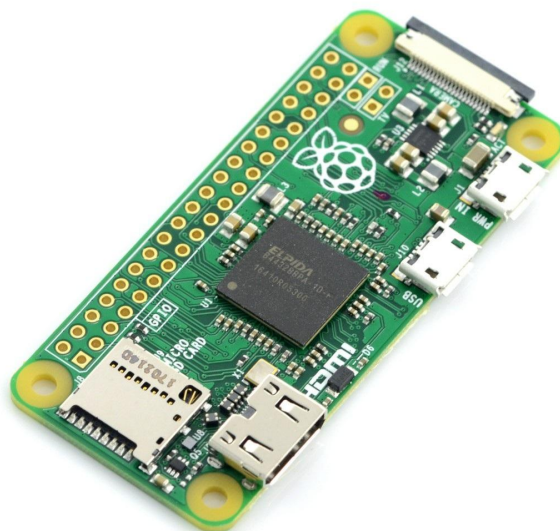


Rys. 4.3. Raspberry PI Pico

*Źródło: [2]*

Obsługuje on takie języki programowe jak C lub C++, Micro oraz Circuit Python. Jest to bardzo kompaktowa wersja Raspberry, ponieważ jego wymiary wynoszą 5,1cm na 2,1cm oraz jego waga wynosi około 6 gram, co stanowi idealne rozwiązanie dla większych projektów wymagających dużej mocy obliczeniowej przy stosunkowo ograniczonym miejscu.

Wariant Zero jest to miniaturyzacja wersji SBC, posiada on wszystkie dostępne elementy co w innych wersjach, lecz jego wymiary są zbliżone do wersji Pico. Niestety w tym miejscu napotykamy jeden z minusów, Raspberry Zero jest wyposażone w mikroprocesor co jest jednocześnie zaletą ze strony wydajności, lecz i ogromną wadą ze względu konsumpcji energii oraz wydzielanego ciepła (W przypadku ciepła wszystko zależy od złożoności odpalanych na nim aplikacji oraz czy istnieje miejsce na zainstalowanie radiatora). Minimalne napięcie potrzebne do uruchomienia urządzenia i prowadzenia stabilnej pracy musi znajdować się na poziomie przynajmniej 1500mA. Wygląda on następująco:



Rys. 4.4. Raspberry PI Zero

*Źródło: [3]*

Raspberry Zero przeszło rewizję a jego nowa odmiana została oznaczona numerem 2. W jej skład wchodzi nowy układ Wi-Fi (Gdzie poprzednia wersja musiała być z dopiskiem „W” na końcu, aby obsługiwała sieć Wi-Fi) oraz komunikacją Bluetooth. Posiada on takie same wymiary co poprzednik, lecz jego zapotrzebowanie na energię diametralnie się zmienia, w tej wersji potrzebujemy co najmniej 3000mA, aby urządzenie pracowało stabilnie, jest to dwukrotna ilość pobieranego prądu w stosunku do pierwszego wydania, a co za tym idzie ciepło generowane przez te płyty potrafi być problematyczne.

Większe wersje Raspberry PI mają zdolność szerszego zastosowania pod względem wydajności, posiadają nie tylko w standardzie obsługę Wi-Fi oraz Bluetooth (Oprócz wersji pierwszej i drugiej) ale również złącze Ethernet, dodatkem w wersji 4 jest komunikacja Bluetooth w technologii BLE. Co jest ciekawe szyna UART jest dzielona pomiędzy portami GPIO a Bluetooth więc w razie chęci wykorzystania, GPIO do przesyłu danych musimy wejść do menu konfiguracyjnego danego systemu i ustawić chęć przesyłu danych poprzez to złącze, lecz w tym przypadku całkowicie tracimy możliwość komunikowania się urządzeniem z innymi systemami w standardzie UART przez Bluetooth (Peryferia które wykorzystują Bluetooth w większości dają się połączyć bez szyny UART, lecz zdarzają się przypadki gdzie ona jest wymagana do prawidłowego działania). Układy te wymagają dość sporego zapasu energii do stabilnego działania,

przeważnie 3000mA wystarczy do pracy przy wysokim obciążeniu, lecz niektóre modele najlepiej pracują, kiedy mają trochę więcej zapasu. Wszystkie modele oraz sposoby ich wykorzystania wraz z opisami i potrzebnymi informacjami do wygodnego korzystania możemy przeczytać w książce [3].

Przeważnie posiadają one zbliżony do siebie wygląd i formę która prezentuje się jak ta podana na przedstawionym zdjęciu:



Rys. 4.5. Raspberry PI 4 B

Źródło: [4]

#### 4.6.2. Bazy danych SQLite

Bazy danych SQLite cechują się w stosunkowo małym zapotrzebowaniem na wykorzystanie innych podzespołów niż RAM (ang. *Random Access Memory*) ponieważ istnieje możliwość załadowania całej bazy danych właśnie do tej pamięci, dlatego są to najczęściej wybierane bazy danych stosowane w programach wykorzystujące Raspberry PI. Biblioteka implementuje silnik SQL (ang. *Structured Query Language*), co prowadzi do utworzenia i możliwości używania bazy danych bez konieczności uruchamiania procesu RDBMS (ang. *Relational Database Management System*). Sam w sobie jest dość elastycznym systemem zarządzania bazą danych, ponieważ obsługuje większość



dostępnych języków programowania. Przestrzeń bazodanowa stanowi jeden plik w którego wchodzi struktura drzewa, dzięki czemu bazy te są kompaktowe i pozornie zajmują mało miejsca. Prawda jest trochę inna niż się wydaje, plik przechowujący bazę danych może się rozrosnąć nawet do 280 terabajtów, co może spowodować trudności w przenoszeniu baz danych biorąc pod uwagę to, że nie mamy fizycznego dostępu do zawartości struktury drzew.

SQLite obsługuje również elementy takie jak zapytania zagnieżdżone, widoki, klucze, transakcje, definiowane własne funkcje oraz częściowo wyzwalacze. Ciekawostką jest to, że ten system mimo to iż jest jak najbardziej kompaktowy cechuje się bardzo rozbudowaną strukturą elementów do wykorzystania. W porównaniu z MySQL w wersji niższej niż 5.0 (Aktualna wersja to 8.0, lecz wymaga ona już o wiele większych zasobów) która wymaga podobnej liczby dostępnych zasobów, by pracować bez zakłóceń, nie posiada obsługi transakcji. Ponadto wydajność SQLite znacząco przewyższa bazy danych MySQL. Informacje na temat używania oraz tworzenia baz SQLite możemy przeczytać pod [11] czy też [20], lecz jeżeli interesują nas bardziej podstawy zapytań SQL dobrym wyborem może być również [19].

## **4.7. Nextion HMI oraz jego możliwości**

### **4.7.1. Program Nextion Editor**

Program Nextion Editor jest wszechstronnym interfejsem programistycznym zawierającym elementy graficzne, jak i elementy wymagające wstępnego zakodowania. Jego zadaniem jest umożliwienie użytkownikowi dopasowaną edycję panelu dotykowego oraz pomoc w obsłudze zdarzeń z tym związanych. Ciekawym elementem całego środowiska jest to, że każdy z elementów zastosowanych podczas tworzenia projektu w tym edytorze posiada własny blok kodu, to właśnie poprzez nie musimy programować wszystkiego na maszynie zewnętrznej. Część operacji arytmetycznych i matematycznych może zostać wykonana poprzez dodanie nowej funkcji z dostępnej palety, lecz jeżeli jest to dla nas niewystarczające możemy również dodać własny algorytm w polu do tego przeznaczonym.



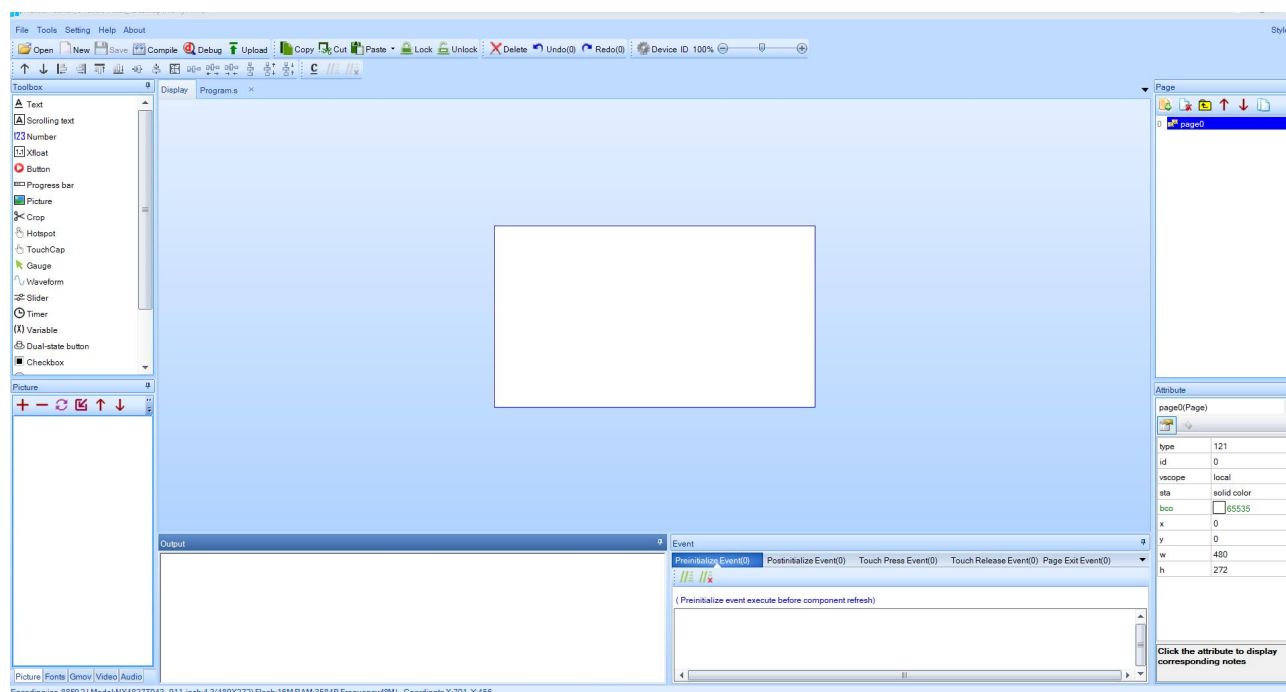
Elementy jakie możemy wykorzystać w naszym projekcie znajdują się w bocznym pasku Toolbox, znajdziemy tam składniki takie jak podzespoły do wyświetlenia na interfejsie HMI (ang. Human-Machine Interface) oraz struktury pasywne takie jak Waveform który powoduje generowanie przebiegu zmiennego, Audio gdzie jest odtwarzany dźwięk podczas podania mu odpowiednich warunków oraz Timer z wbudowanego zegara w urządzeniu.

Edytor pozwala również rozbudowanie interfejsu o dodatkowe strony po których możemy się płynnie poruszać oraz dodawać im warunki otworzenia.

Na samym dole edytora możemy zauważyć pole Event gdzie możemy podać warunki oraz kalkulacje przy odpowiednich zachowaniach, a także rozbudować niedołączone funkcje dla naszych kontrolerek oraz stron interfejsu.

Dla każdego elementu znajdującego się w obszarze roboczym możemy ustawić odpowiednie atrybuty ustawiając elementy typu id, pozycje, szerokości oraz wartości.

Cały Program prezentuje następujący obraz.



Rys. 4.6. Widok Aplikacji Nextion Editor

*Źródło: Opracowanie własne*



Program posiada również możliwość emulowania pracy urządzenia oraz prowadzenia dziennika zdarzeń, dzięki czemu możemy monitorować cały okres pracy bez konieczności na nowo wgrywania programu do pamięci interfejsu. Emulator posiada również takie funkcje jak test przeciążenia który celowo obciąża interfejs w taki sposób danymi, aby była możliwa analiza słabych punktów algorytmów zapisanych nie tylko na tym urządzeniu, ale również na stacjach pomocniczych. Dodatkowo możemy podać emulatorowi układ sterujący który jest docelowy i to właśnie z niego są wtedy brane dane testowe, do tego wykorzystujemy przejściówkę UART do USB, dzięki czemu możemy podłączyć Mikrokontroler bez konieczności ciągłego wgrywania nowego oprogramowania na interfejs poprzez kartę MicroSD. Więcej interesujących nas treści możemy znaleźć w podręczniku dostarczonym od producenta pod linkiem [7].

#### **4.7.2. Interfejs HMI Nextion**

Nextion wprowadził na rynek nowatorskie rozwiązanie w postaci interfejsu graficznego łatwego zarówno w użytkowaniu, jak i w programowaniu. Jego kompaktowa budowa sprawia, że jest prawie że idealny do wszelakiego rodzaju projektów. Możemy go dostać w wielu różnych modelach oraz rozmiarach począwszy od 2.4 cala do nawet 10 cali. Pracuje on na technologii TTL co daje możliwość zastosowania go we wszelakiego rodzaju projektach bez konieczności korzystania z obszernych części kodu. Inne rozwiązania typu HMI przedstawia publikacja [18] zawierająca większą pulę informacji o rozwiązaniach konkurencyjnych oraz ich zastosowania.

Sam interfejs jest wyposażony w dotykowy ekran działający na zasadzie styku pól na nim znajdujących się. Jednocześnie jest to wada, ale i również zaleta. Wadą jest to, że ekran potrzebuje dość pokaźnej siły nacisku, żeby wykryć wciśnięcie pola, lecz z drugiej strony ekran może być używany wszystkim, co mamy pod ręką, a nie tylko ludzką dłońią jak jest to w przypadku ekranów dotykowych działających na indukcji pomiędzy polami.





Jego wygląd przedstawia następujący rysunek:



Rys. 4.7. Interfejs Nextion HMI 2.4 Cala

*Źródło:[5]*

Urządzenie jest wyposażone w elementy takie jak MCU/16 MB Pamięci oraz SRAM co daje nam bardzo pokaźne możliwości wykorzystania go w projektach. Dodatkowo ekran posiada 4 punkty mocowania co przydaje się w przypadkach używania go w ręcznie robionych obudowach lub na statywach. Dodatkowym atutem jest autorski program producenta do programowania tych interfejsów, ponieważ różne serie produktu posiadają dodatkowe funkcje które program uwzględnia.

Niestety podczas tworzenia urządzenia okazało się, że nie ma możliwości stworzenia lustrzanego odbicia ekranu z poziomu edytora Nextion ani z poziomu kodu dostarczanego do niego. Udało się znaleźć sposób przerobienia ekranu tak, aby ta funkcja działała, ale jest to zabieg bardzo inwazyjny i istnieje możliwość nieodwracalnego uszkodzenia interfejsu co zdyskwalifikowało ten zabieg z wykonania na poczet projektu.





## 5. Projekt urządzenia

W celu sprawdzenia możliwości komunikacji oraz jej jakości z samochodem powstał układ urządzeń połączonych razem za pomocą rozwiązań IoT. Głównym elementem komunikacyjnym w tym zestawieniu jest układ ELM327 z którym od samego początku były dość spore problemy z powodu ilości i jakości podróbek oryginału tego układu. Po zakupieniu trzeciej sztuki w końcu udało się nawiązać komunikację pomiędzy nim a mikrokontrolerem ESP32, lecz to nie był koniec problemów napotkany na drodze do utworzenia tego systemu. Okazało się, że mimo iż kontroler OBD potrafił odszukać jaki protokół komunikacji był potrzebny do działania, to nie potrafił odnaleźć z jakiej puli danych możemy korzystać. Pojazdem testowym było Audi A6 C5 z silnikiem benzynowym o pojemności 2.4 litra z 2002 roku. Szybko wyklarowało się, że większość systemów była odseparowana od ECU i nie był możliwy dostęp do ich danych oraz nie pozwoliło na wykorzystanie w projekcie informacji na temat systemu ABS oraz innych rozwiązań z dziedziny układu hamulcowego. Oczywiście pula dostępnych informacji jest bardzo zależna od roku produkcji samochodu oraz jaki dany samochód wykorzystuje protokół komunikacyjny. Najwięcej informacji aktualnie na rynku możemy uzyskać z samochodów wyposażonych w szyny CAN. Niektóre modele pojazdów również pozwalają na uzyskanie informacji z panelu FIS. Projekt urządzenia niestety musiał zostać przystosowany do dostępnych środków z czego nie ma możliwości usuwania kodów DTC (ang. *Diagnostic Trouble Code*).

W Systemie został wykorzystany dotykowy interfejs graficzny typu HMI w celu czytelnej interpretacji danych. Przesył danych pomiędzy poszczególnymi modułami odbywa się za pomocą połączenia bez internetowego. ELM327 łączy się z ESP32 za pomocą Bluetooth, następnie ESP32 łączy się poprzez lokalnie utworzoną sieć Wi-Fi do ESP8266 za pomocą MAC adresów, to właśnie dzięki nim odpowiednie dane zostają wysłane do odpowiednich mikrokontrolerów.

Przesył danych pomiędzy ESP8266 a interfejsem Nextion odbywa się za pomocą szyny UART co pozwala na szybki i niezakłócony dostęp do danych. Niestety i w tym przypadku napotykałyśmy problem bowiem biblioteka dostarczona przez producenta musi



zostać nieco zmodyfikowana, aby działała ona poprawnie. W pliku konfiguracyjnym możemy znaleźć odniesienia do uruchomienia trybu serwisowego co znaczy, że do poprawnego działania musimy zmienić parę linii kodu.

W pierwszej kolejności otwieramy plik konfiguracyjny NexConfig.h aby za komentować dwie linie kodu zawierające odniesienie do funkcji „DEBUG SERIAL”, następnym krokiem jest ustawienie seryjnego „Serial2” na „Serial” ponieważ jest to odniesienie do drugiej szyny UART w mikrokontrolerach posiadających 2 szyny. Dodatkową opcją konfiguracji jest lekkie przerobienie kodu pliku NexHardware.cpp oraz NexHardware.h znajdują się bowiem tam odniesienia do inicjalizacji połączenia z interfejsem. W pierwszym z nich szukamy części kodu o nazwie nexInit który nie zawiera parametrów wejściowych. Dodajemy do niego parametr w taki sposób, aby jego nazwa oraz zawartość wyglądała w następujący sposób:

```
bool nexInit(int i)
{
    bool ret1 = false;
    bool ret2 = false;

    dbSerialBegin(i);
    nexSerial.begin(i);
}
```

Rys. 5.1 Przeprogramowanie zdefiniowanej funkcji

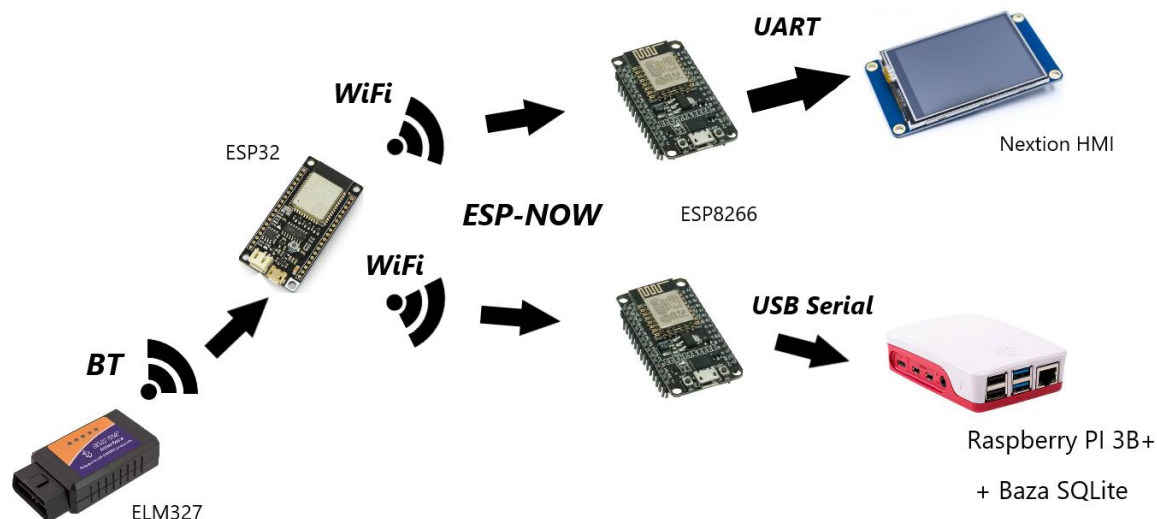
*Źródło: Opracowanie własne*

Dodatkowo w pliku z końcówką „.h” musimy znaleźć odwołanie do tego bloku i również dodać w jego implementacji „(int i)”. Dzięki tej drobnej modyfikacji nie tylko rozszerzyliśmy funkcjonalność biblioteki o wybór szybkości przesyłu danych, ale również możemy go deklarować i zmieniać w kodzie według upodobań od 2400 do katalogowo 115200 (Podstawowo jest on ustawiony na 9600), lecz internauci podają również możliwość lekkiej przeróbki urządzenia, aby działał on również z prędkością 250000. Są również wersje systemu Nextion w wersji Intelligent, oznaczone one są literą „P”, posiadają one o wiele większe pole zastosowań ze względu poszerzoną opcje konfiguracji oraz o stosunkowo szybszy przesył informacji pomiędzy urządzeniem a interfejsem (Z wykorzystaniem „Foca Max” prędkość możemy zwiększyć do 2.5Mbps,



niestety ten moduł rozszerzeń jest tylko dostępny dla wersji z oznaczeniem „P”). Więcej informacji na temat efektywnej pracy z językiem C przedstawia książka [28].

Całość systemu utworzonego do analizy danych pozyskanych z ECU przedstawia następujący schemat poglądowy Rys.5.2.



Rys. 5.2. Schemat poglądowy utworzonego systemu wraz z opisami połączeń oraz nazwami elementów

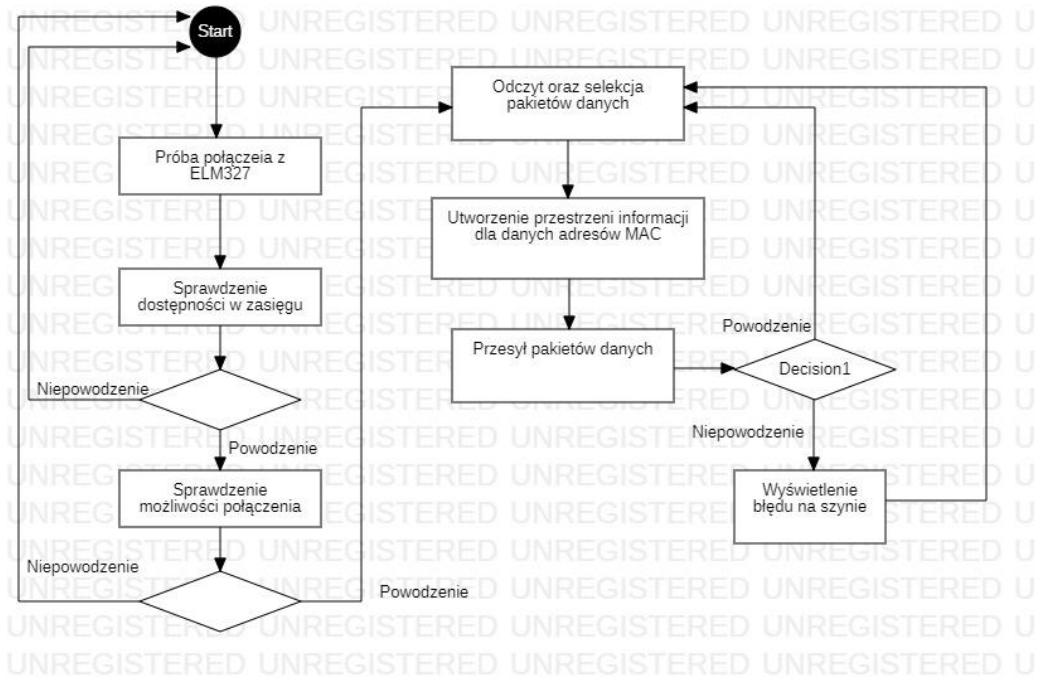
*Źródło: Opracowanie własne*

Jak możemy zauważyć posiadamy dwa urządzenia wyjściowe połączone z ESP8266 jako końcowe elementy oraz ELM327 połączony bezprzewodowo z ESP32 który nadaje do wszystkich ESP w zasięgu, w tym przypadku ten sam pakiet danych. Do komunikacji na elementach końcowych wykorzystano technologie niekolidujące z komunikacją bezprzewodową (W tym przypadku UART oraz USB Serial pozwoliły na najszybszy dostęp do informacji zaraz po przesłaniu danych z nadajnika).

Nadajnik jest wyłącznie zależny od modułu ELM327 i do działania nie potrzebuje informacji zwrotnych z odbiorników w celu kontynuacji pracy, dało to możliwość stworzenia niezależnych sekcji urządzeń i opcjonalne użycie elementów końcowych w zależności od potrzeb. Dodatkowo uzyskujemy możliwość prostej rozbudowy o kolejne urządzenia końcowe na przykład takie jak bazy danych znajdujące się w chmurze czy też urządzenia korygujące pracę samochodu.

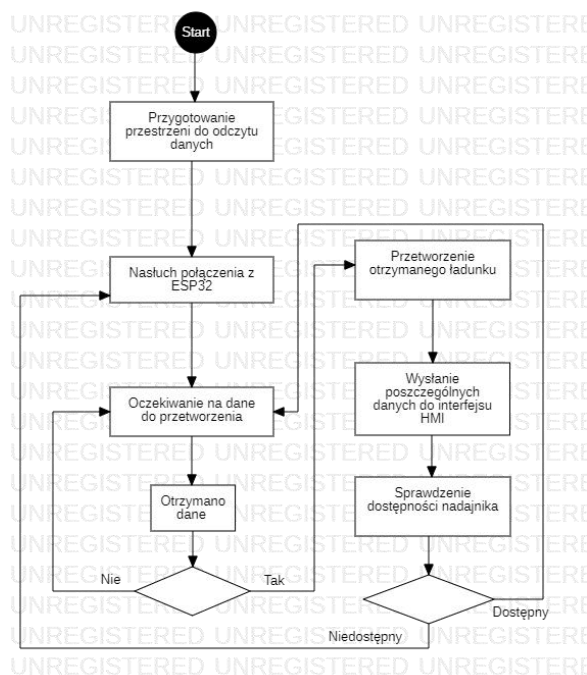


Pracę poszczególnych elementów przedstawiają następujące schematy utworzone w programie StarUML:



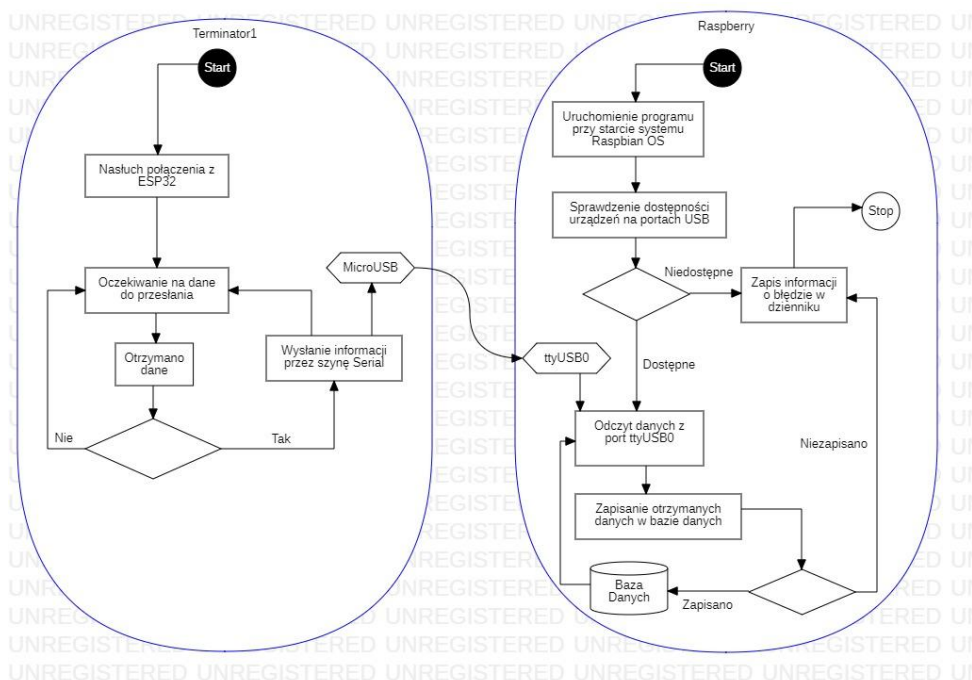
Rys. 5.3. Schemat działania nadajnika ESP32 pobierającego informacje od ELM327

*Źródło: Opracowanie własne z wykorzystaniem programu StarUML*



Rys. 5.4. Schemat działania odbiornika ESP8266 połączonego z interfejsem Nextion

Źródło: Opracowanie własne z wykorzystaniem programu StarUML



Rys. 5.5. Działanie odbiornika ESP8266 połączonego z Raspberry Pi oraz Bazą SQLite

Źródło: Opracowanie własne z wykorzystaniem programu StarUML



Na rysunkach 5.3, 5.4 oraz 5.5 przedstawione zostały schematy działania z poszczególnymi etapami pracy urządzeń. Zaczynając od nadajnika który po nawiązaniu połączenia oraz otrzymania informacji z ECU nie posiada momentu zatrzymania. Podobnie jest w przypadku elementów końcowych, jednakże one zaczynają pełną swoją pracę wyłącznie w przypadku połączenia z nadajnikiem oraz kończą razem z nim. Jedynym elementem który posiada dziennik zdarzeń gdzie zapisywane są błędy pracy jest program napisany na urządzeniu Raspberry Pi. Program podczas uruchomienia zapisuje swój stan na bieżąco dzięki wykorzystaniu planogramu „croncat” oraz właśnie z racji tego program uruchamiany jest przy starcie systemu po załadowaniu wszystkich niezbędnych zasobów systemowych do jego poprawnego działania.

## **5.1. Wymagania funkcjonalne i нефunkcjonalne urządzenia, specyfikacja techniczna oraz wykorzystane biblioteki.**

### **5.1.1. Wymagania funkcjonalne**

- Odczyt prędkości samochodu.
- Odczyt temperatury płynu chłodniczego.
- Odczyt obrotów silnika.
- Odczyt obciążenia silnika.
- Odczyt kąta wyprzedzenia zapłonu.
- Interakcja dotykowa z Interfejsem HMI.
- Wyświetlenie zgromadzonych danych.
- Połączenie bezprzewodowe pomiędzy odbiornikami a nadajnikiem.



### 5.1.2. Wymagania niefunkcjonalne

- Budowa urządzenia oparta o platformy ESP oraz Raspberry PI.
- Uzyskanie dostępu do danych znajdujących się w ECU samochodu.
- Interakcja użytkownika poprzez panel HMI.
- Dowolność w wyborze urządzeń końcowych (Odbiorników).

### 5.1.3. Specyfikacja techniczna urządzeń

- ELM327 zasilany poprzez złącze OBDII.
- Zasilanie wszystkich ESP w zakresie 5 V USB / 5 V port VIN.
- Zasilanie Raspberry PI 5 V USB minimalnym prądem 1,7 A.
- Możliwość szybkiej implementacji kolejnych modułów końcowych.
- Interfejs HMI zasilany jest poprzez złącze na ESP8266.
- Odbiornik ESP8266 jest zasilany przez złącze USB w Raspberry PI.

### 5.1.4. Użyte biblioteki

- Arduino (ESP32):
  - „BluetoothSerial.h” - Jest to biblioteka która pozwala na komunikacje z innymi urządzeniami obsługującymi technologię bluetooth.
  - „ELMduno.h” - Biblioteka ta jest wykorzystywana w celach komunikacyjnych z modułami ELM, zawiera ona zapytania oraz interpretacje dla różnych standardów komunikacyjnych.
  - „esp\_now.h” - Biblioteka jest wykorzystywana w celu wykorzystania różnych szyn komunikacyjnych, posiada wykorzystanie wyłącznie dla ESP32.



- „WiFi.h” - Zawiera ona wszystkie potrzebne protokoły do komunikacji bezprzewodowej w standardzie Wi-Fi.
- Arduino (ESP8266):
  - „espnw.h” - Jest to praktycznie ta sama biblioteka co „esp\_now.h” lecz posiada on zastosowanie wyłącznie dla modułów ESP8266
  - „ESP8266WiFi.h” - Podobnie jak w „espnw.h” ta biblioteka również jest bliźniacza z „WiFi.h” oraz tak samo jest tylko do wykorzystania na układach ESP8266.
  - „Nextion.h” - Biblioteka ta jest wykorzystywana, aby zminimalizować ilość kodu w programie oraz ułatwić komunikację pomiędzy układem a interfejsem HMI.
- Python:
  - „Serial” - Biblioteka ta umożliwia komunikację programu z portami szeregowymi USB.
  - „sqlite3” - Jest ona odpowiedzialna za możliwość związania programu z bazą danych w taki sposób, aby była możliwość wykonywania poleceń.
  - Elementy „datetime” oraz „timezone” z biblioteki „datetime” - cała biblioteka jest dość obszerna a Raspberry Pi posiada dość małą pulę zasobów więc użyte zostało rozwiązanie wyodrębnienia elementów z biblioteki w taki sposób, aby nie była ładowana w całości. Biblioteka odnosi się do aktualnego stanu czasu pobieranego z urządzenia. Ważne jest to aby komputer miał prawidłowo ustawiony zegar.





## 6. Utworzenie systemu

Oba mikrokontrolery zostały zaprogramowane w środowisku Arduino. Dało to dostęp do szerokiej gamy bibliotek pomagających zautomatyzowanie pewnych procesów. System jest w większości kompatybilny z samochodami powyżej 2004 roku. Poniżej możemy znaleźć się w sytuacji gdzie samochód nie będzie dostarczał nam informacji.

### 6.1. Znaczenie otrzymywanych specyficznych danych

W pierwszej kolejności musimy zrozumieć istotę obciążenia silnika (Jest ona podawana w procentach), kąta wyprzedzenia zapłonu oraz temperatury płynu chłodniczego.

Pierwszy z nich odpowiada za obciążenie wywołane przez użytkownika na silnik poprzez zmianę ilości podawanej mieszanki paliwowej, jest to również zależne od przeniesienia mocy silnika na skrzynię biegów. Ważnym elementem jest to, że samochody które nie były w żaden sposób modyfikowane nie są w stanie wygenerować pełnego obciążenia silnika (Przeważnie jego granica sięga od 75 procent do 85). Stosunkowo na obciążenie silnika wpływają elementy napędzane przez niego takie jak pompy oraz generatory. W wykonanych testach stale była włączona pompa klimatyzacji co dodatkowo obciążało silnik. Nieprawidłowości związane z tą wartością występują w przypadku braku odpowiedniej kompresji na cylindrach silnika co może skutkować poważną usterką jednostki. Niewykorzystanymi parametrami związanymi z pracą układu hamulcowego również możemy uzasadnić natężenie obciążenia silnika i wartości jakie uzyskujemy, ponieważ wywieranie nacisku na dźwignię hamulca daje znak o zwiększeniu ciśnienia w układzie wspomagania hamowni co za tym idzie część siły napędowej jest przekierowywana na nią. Prosty test działania pompy wspomagania układ hamulcowego jest ciągle wciskanie pedału aż zostanie utworzone ciśnienie (proces ten należy wykonać na zgaszonym silniku), jeżeli podczas odpalenia samochodu będziemy trzymać dźwignię hamulca i pocujemy, że w pierw nabierze odrobinę większego ciśnienia oznacza to o poprawności działania pompy wspomagania hamulca.



Drugim element odpowiada za dobranie odpowiedniego momentu zapłonu mieszanki paliwowej w zależności od obciążenia silnika, stosunku powietrza do paliwa, możliwej kompresji mieszanki, ciśnienia maksymalnego zadeklarowanego przez producenta jakie może występować podczas kompresji mieszanki oraz jak w przypadku przeprowadzonych testów synchronizacja bloków w silnikach o strukturze widlastej. Kąt wyprzedzenia zapłonu jest istotny do wykorzystania w pełni mocy dawki paliwowej podawanej do silnika. Wartość ta jest liczona w stopniach i jest możliwość pojawienia się wartości ujemnych w przypadku nagłego zmniejszenia dawki paliwowej lub konieczności zminimalizowania „bocznego zapłonu” lub inaczej „spalania stukowego” paliwowa która mogłaby spowodować wybuch niekontrolowany mieszanki co skutkowałoby powolnym wypalaniem bloku oraz powstawaniem wżerów na powierzchni cylindrów. Ciekawostką jest to, że nowsze samochody posiadają czujniki spalania stukowego które informują poprzez interfejs FIS użytkownika o występowaniu owego zachowania zapłonu.

Jak widać oba te elementy mają znaczący wpływ na bezpieczeństwo jednostki napędowej samochodu i mają swoje skrajne punkty które informują nas o nieprawidłowości działania. W przypadku obciążenia silnika punktem który daje nam informacje, że coś się dzieje nie w taki sposób jak powinno jest moment w którym na postoju silnik dostaje obciążenia bez dodawania gazu przeważnie wartość tą można określić do 10% ze względu na klimatyzację która potrafi wygenerować pokaźne obciążenie w zależności od zużycia pompy. Kolejnym punktem krytycznym jest otrzymywanie znacznych różnic pomiędzy obrotami silnika a jego obciążeniem można to zaobserwować podczas jazdy ze stałą prędkością, jeżeli jesteśmy dajmy na to w połowie obrotomierza a wskaźnik obciążenia wskazuje nam na przykład 70% obciążenia może to znaczyć o zachodzących tarciach pomiędzy blokiem a tłokiem, lub też pomiędzy silnikiem a skrzynią.

W przypadku kąta zapłonu posiadamy dość duże spektrum danych i jest ciężiej wskazać element winny nieprawidłowości, lecz daje nam to informacje, kiedy musimy podjąć działania, zanim zaczniemy słyszeć specyficzne metaliczne uderzenia silnika przy każdym obrocie wału. Punktem krytycznym w tym przypadku jest praca silnika na biegu jałowym. Jeżeli podczas postoju zauważamy znaczne odchylenia kąta zapłonu świadczy



to o zły synchronizacji świec zapłonowych z kompresją mieszanki paliwowej. Przykładowo jeżeli kąt zapłonu na postoju znajduje się w wartościach od -8 do 15 jest to normalna praca jednostki z uwagi na jakość paliwa oraz ilość powietrza potrzebnego do kompresji, lecz kiedy wartości te są powyżej tych wartości może to świadczyć o powstawaniu zapłonu niekontrolowanego, a zarazem może oznaczać początki wypalania się pierścieni uszczelniających oraz początki powstawania wżerów na powierzchni cylindrów.

Ostatnim elementem który potrzebuje omówienia jest temperatura płynu chłodniczego. Wartość ta jest istotna w dłuższych trasach, ale również posiada dodatkowe zastosowania. W przypadku dłuższych tras płyn chłodniczy nie może przekroczyć wartości 100 stopni, ponieważ doprowadza to do jego wrzenia (Oczywiście układy wytrzymują lekkie wrzenia płynu za pomocą zaworów zwrotnych ciśnieniowych umiejscowionych w najwyższych punktach układu w przypadku powstania nadciśnienia. Oddzielną kwestią są również stosowane płyny chłodnicze, ponieważ niektóre nie są na bazie wody a płynów o większej temperaturze wrzenia co eliminuje problemy z powstawaniem dodatkowego ciśnienia w układzie). Jeżeli temperatura przekroczy wartość 100 stopni jest to sygnał, że nasz układ chłodniczy nie działa prawidłowo co może być skutkiem przegrzania oleju silnikowego oraz stworzenia sytuacji w której blok silnika zostanie uszkodzony.

Dodatkową funkcją temperatury płynu jest identyfikowanie poprawności działania układu zasilanego gazem LPG (ang. *Liquefied Petroleum Gas*) a w zasadzie możliwości aktywowania zasilania pojazdu gazem lotnym. Temperatura początkowa jest istotna w tym przypadku, ponieważ jeżeli będzie ona za niska do bloku będzie trafiać za mała ilość paliwa co za tym idzie jednostka nie będzie działała prawidłowo. Parownik który ma na celu zmienić gaz z formy stałej na lotną jest podłączony bezpośrednio z układem chłodniczym w celu jak najszybszego zwiększenia temperatury parownika oraz zwiększenia jego wydajności. W większości pojazdów skala temperatury zaczyna się od 60 stopni, lecz do sprawdzenia poprawności działania musimy mieć informację o dokładnej temperaturze płynu w danej chwili (Przeważnie temperatura przełączenia ustawiana jest w okolicy 30 stopni w lecie a 35 stopni zimą w celu lepszego dogrzania mieszanki). Wszystkie dotychczasowe elementy opisujące zdarzenia wpływających



elementów silnika na siebie przedstawia książka edytowana przez Davida Crolla z 2009 roku [14] oraz publikacja WKŁ [25].

## 6.2. Kod oraz objaśnienie odbiornika danych z modułu ELM327 na bazie ESP32

Całość napisanego kodu możemy podzielić na 4 części pierwszą sekcją jest stworzenie pul do przesyłu informacji oraz komunikacji z ELM327.

```
1.  #include "BluetoothSerial.h"
2.  #include "ELMduino.h"
3.  #include <esp_now.h>
4.  #include <Wi-Fi.h>
5.  BluetoothSerial SerialBT;
6.  #define ELM_PORT    SerialBT
7.  #define DEBUG_PORT Serial
8.
9.  typedef enum {ENG_RPM, COOLANT_TEMP, ENGINE_LOAD_PER, T
    IMING,SPEED} obd_pid_states;
10. typedef struct struct_message { char PaySend[150]; } st
    ruct_message;
11. uint8_t address[6]  = {0x01, 0x23, 0x45, 0x67, 0x89, 0x
    BA};
12. String payload  = "";
13. uint8_t broadcastAddress[] = {0x84, 0xF3, 0xEB, 0x89, 0
    xB1, 0xED};
14. uint8_t broadcastAddressBalckBox[] = {0x84, 0xF3, 0xEB,
    0x59, 0xE9, 0x07};
```

Listing 6.1. Deklaracja struktury przesyłu i adresacja połączeń w ESP32

Na początku mamy implementacje bibliotek wykorzystanych w danym urządzeniu po kolei z nich BluetoothSerial jest odpowiedzialny za komunikację esp32 z ELM327



bez konieczności używania okablowania. Łącze jest ustawione na 115200 baudrate. Następnie ELMduino jest to biblioteka polecana przy pracy z ELM327 dzięki niej możemy w odrobinę prostszy sposób otrzymywać dane oraz łączyć się z interpreterem, niestety posiada ona ograniczoną ilość odwołań do danych więc niekiedy musimy sami napisać niestandardowe zapytanie do interpretera. Biblioteki esp\_now oraz Wi-Fi są w pewnym stopniu ze sobą połączone, ponieważ pierwsza z nich potrzebuje inicjalizacji Wi-Fi z tej drugiej. Kolejne linie kodu są to po prostu przedefiniowania nazw zmiennych używanych w kodzie.

Dochodząc do deklaracji definicji enumeratora, czyli modułu wyliczającego po kolei odpowiednie elementy znajdujące się w kategorii „obd\_pid\_states”, w naszym przypadku znajduje się on w bibliotece ELMduino. Kolejną definicją jest utworzenie struktury w której będziemy wysyłać naszą wiadomość na zasadzie ciągu znaków. Na samym końcu danego bloku definiujemy jakie adresy MAC posiadają kolejno ELM327 oraz ESP8266 do komunikacji bezprzewodowej.

Kolejna sekcja przedstawiona na Listingu 13.2. odpowiada za czynności wykonane tylko raz przy włączeniu rządu, znajdują się w niej elementy połączeniowe z oba modułami oraz ustawienia ich dotyczące.

```
1. void setup()
2. {
3.     DEBUG_PORT.begin(115200);
4.     ELM_PORT.begin("ArduHUD", true);
5.     SerialBT.setPin("1234");
6.     Wi-Fi.mode(WI-FI_STA);
7.     if(esp_now_init() != ESP_OK){
8.         Serial.println("ESP CONNECTION ERROR (ESP_NOW)");
```



```
9.         return;
10.     }
11.     esp_now_register_send_cb(OnDataSent);
12.     peerInfo.channel = 0;
13.     peerInfo.encrypt = false;
14.     memcpy(peerInfo.peer_addr, broadcastAddressBalckBox,
6);
15.     if(esp_now_add_peer(&peerInfo) != ESP_OK) {
16.         Serial.println("Peer Error");
17.         return;
18.     }
19.     memcpy(peerInfo.peer_addr, broadcastAddress, 6);
20.     if(esp_now_add_peer(&peerInfo) != ESP_OK) {
21.         Serial.println("Peer Error");
22.         return;
23.     } if (!ELM_PORT.connect(address))
24.     {
25.     DEBUG_PORT.println("Connection error ELM327 Phase 1");
26.     resetFunc();
27.         while(1);
28.     }
29.     if (!myELM327.begin(ELM_PORT))
30.     {
31.     Serial.println("Connection error ELM327 Phase 2");
32.     resetFunc();
33.         while (1);
```



```
34.    }  
35.    Serial.println("Connected to ELM327");  
36.    }
```

Listing 6.2. Sekcja kodu ESP32 zawierająca elementy inicjalizacji

W przedstawionej części kodu możemy zaobserwować sekcję odpowiedzialną za ustawienia dotyczące przesyłu informacji poprzez „Serial” czy też definicję odpowiedzialne za komunikację całego systemu. Przypisanie „SerialBT.setPin(„1234”)” jest zależne od naszego ELM327 w niektórych przypadkach możemy uświadczyc inaczej ustawione hasło do modułu Bluetooth (Standardowo „1234”) a czasami i tak potrzebujemy nadać odniesienie do zmiany używanego hasła, bo jest to zależne również od modułu z jakiego korzystamy.

Pierwszy z warunków sprawdza, czy połączenie pomiędzy ESP32 a ESP8266 zostało nawiązane a w razie problemów na wyjściu podaje błąd połączenia. Kolejnym etapem jest przypisanie co ma zrobić ESP w przypadku uzyskania danych do przesyłu. Kolejne linie kodu to konfiguracja przesyłu informacji typu na jakim kanale one mają zostać wysłane i czy mają one być zaszyfrowane. Pętla która zawiera element „esp\_now\_add\_peer” pobiera informację od dodanym adresie ESP który przechowuje w pamięci oraz dodatkowo sprawdza, czy warunek dodania został spełniony.

Kolejne pętle warunkowe odpowiadają za sprawdzenie połączenia ESP32 z ELM327 w pierwszej kolejności jest nawiązywane połączenie z interpreterem zaś w drugiej nawiązywane jest połączenie z ECU pojazdu. Jeżeli wszystkie warunki zostaną spełnione program przejdzie do trzeciego etapu w którym pracuje bez końca, a wygląda on następująco:

```
1. void loop()  
2. {  
3.    StateCaseSwitch();  
4.    if(rpmget==true && coolantget == true && eloadget ==
```



```
    true && timingget == true && speedget == true){  
5.      char Buff[150];  
6.      payload = rpm_catcher + "/" + coolant_catcher + "/"  
          + eload_catcher + "/" + timing_catcher + "/" + speed_cat  
          cher + "/";  
7.      payload.toCharArray(Buff,150);  
8.      strcpy(myData.PaySend,Buff);  
9.      Serial.println(payload);  
10.     esp_err_t result = esp_now_send(0, (uint8_t *) &m  
        yData, sizeof(myData));  
11.     if (result == ESP_OK) {  
12.         Serial.println("success");  
13.         digitalWrite(LED_BUILTIN, LOW);  
14.         ResetMacro();  
15.     }  
16.     else {  
17.         Serial.println("Error ESP data");  
18.         digitalWrite(LED_BUILTIN, LOW);  
19.     }  
20. }  
21. }
```

Listing 6.3. Sekcja kodu ESP32 zawierająca główną pętlę wykonawczą

Sekcja ta rozpoczyna się od uruchomienia czwartej części kodu polegająca na wyszukiwaniu potrzebnych elementów ze stosu dostarczanych danych. Po niej została zadeklarowany warunek w postaci: „Jeżeli masz informacje na każdy z elementów składowych wyślij informację do ESP8266”. Wewnątrz powstała również funkcja sklejająca wszystkie elementy w jeden ciąg gdzie ukośnik służy za oddzielenie poszczególnych informacji. Następnie całość jest deklarowana jako „CharArray” czyli ciąg znaków, ponieważ ta konwersja jest potrzebna, aby można było wysłać dane bez





obawy o przepełnienie bufora danych. Następnie jako wynik działania ESP-NOW używamy wysłania ciągu. Dodatkowo jest wprowadzony warunek sprawdzający poprawność przesyłu danych, jeżeli dane udało się wysłać wywoływana jest klasa o nazwie „ResetMacro();” która ma za zadanie wyzerować wszystkie dane znajdujące się w warunkach pętli potrzebnych do wysłania i odczytu danych.

Wracając do omówienia części czwartej w której jest wykonywana część wyszukująca jaki typ danych aktualnie jest podawany przez ECU przedstawia Listing 6.4.

```
1. void StateCaseSwitch() {
2.     switch (obd_state)
3.     {
4.         case ENG_RPM:
5.         {
6.             rpm = myELM327.rpm();
7.             if (myELM327.nb_rx_state == ELM_SUCCESS)
8.             {
9.                 Serial.print("rpm: ");
10.                 Serial.println(rpm);
11.                 rpm_catcher = String(rpm,2);
12.                 rpmget = true;
13.                 obd_state = COOLANT_TEMP;
14.             }
15.             break;
16.         }
17.         case COOLANT_TEMP:
18.         {
19.             coolant = myELM327.engineCoolantTemp();
```



```
20.         if (myELM327.nb_rx_state == ELM_SUCCESS)
21.         {
22.             Serial.print("Coolant: ");
23.             Serial.print(coolant);
24.             Serial.println("*");
25.             coolant_catcher = String(coolant,2);
26.             coolantget = true;
27.             obd_state = ENGINE_LOAD_PER;
28.         }
29.         break;
30.     }
31.     case ENGINE_LOAD_PER:
32.     {
33.         eload = myELM327.engineLoad();
34.         if (myELM327.nb_rx_state == ELM_SUCCESS)
35.         {
36.             Serial.print("EngineLoad: ");
37.             Serial.print(eload);
38.             Serial.println("%");
39.             eload_catcher = String(eload,2);
40.             eloadget = true;
41.             obd_state = TIMING;
42.         }
43.         break;
44.     }
45.     case TIMING:
```



```
46.     {
47.         timing = myELM327.timingAdvance();
48.         if (myELM327.nb_rx_state == ELM_SUCCESS)
49.         {
50.             Serial.print("TA: ");
51.             Serial.print(timing);
52.             Serial.println("*");
53.             timing_catcher = String(timing,2);
54.             timingget = true;
55.             obd_state = SPEED;
56.         }
57.         break;
58.     }

59.     case SPEED:
60.     {
61.         speed = myELM327.kph();
62.         if (myELM327.nb_rx_state == ELM_SUCCESS)
63.         {
64.             Serial.print("Speed: ");
65.             Serial.print(speed);
66.             Serial.println("km/h");
67.             Serial.println("");
68.             speed_catcher = String(speed,2);
69.             speedget = true;
70.             obd_state = ENG_RPM;
```



```
71.      }  
72.      break;  
73.      }  
74.      }  
75.  }
```

Listing 6.4. Deklaracja przełącznika wybierającego typ danych otrzymanych z ECU

Krótko opisując zasadę działania tego „Przełącznika”, posiadamy parę działów do wyboru i szukamy w całym tym stosie które zapytanie da nam wartość uzyskania wiadomości, w tym przypadku jest to stan „ELM\_SUCCESS” jeżeli nie uda nam się na przykład znaleźć obrotów silnika przełącznik automatycznie przechodzi do następnej wartości i sprawdza, czy uzyskana wiadomość to np. temperatura płynu chłodzącego, i tak w koło aż znajdzie wszystkie komponenty (oczywiście może zdarzyć się sytuacja, że na przykład komputer zdąży 3 razy wysłać obroty, ale nie zdąży wysłać nam temperatury to właśnie po to są warunki do przesłania wiadomości takie jak rpmget czy coolantget. Dzięki nim zbieramy komplet danych) przy każdym kolejnym wykonaniu się tego przełącznika gromadzone dane są zapisywane w zmiennych i mogą być nadpisywane w każdej chwili aż do momentu przesłania gdzie zostają one wyzerowane. Dodatkowo możemy sprawdzić poprawność działania wykorzystując wyłącznie komputer i esp32 połączony z elm327, ponieważ esp32 wyświetla wszystkie zaistniałe sytuacje w postaci wpisów na szynie „Serial”.

### 6.3. ESP8266 jako mikrokomputer powiązany z Nextion

W tym przypadku wykorzystujemy ESP jako odbiornik który nasłuchuje wiadomości które do niego trafiają. Nie sprawdza on żadnej zawartości czy nie dekoduje informacji, jego zadaniem jest nasłuch wszystkiego, co jest nadawane oraz późniejsza obróbka danych, aby były one w pełni kompatybilne dla interfejsu HMI. Jego zaletą jest to, że nie wymaga autoryzacji stacji wysyłającej informacje do niego, ponieważ to ona zawiera MAC adres w które miejsce powinny trafić informacje, co za tym idzie kod połączenia jest stosunkowo niewielkich rozmiarów w porównaniu, do naszego nadajnika.



### 6.3.1. ESP8266 odpowiedzialny za obróbkę danych dla interfejsu

Najważniejszą częścią kodu, a raczej metodą jest tutaj element „OnDataRecv” który odpowiada za przetworzenie utrzymanego ciągu danych w taki sposób, aby odseparować odpowiednie informacje i przyporządkować je zmiennym, które na sam koniec zostają wysłane jako wartości do interfejsu HMI poprzez protokół UART co znaczy, że pin znajdujący się na ESP8266 z podpisem Tx trafia do wejścia Rx oraz analogicznie pin Rx do Tx wyświetlacza. Kod zapisany na mikrokontrolerze przedstawia następujący Listing:

```
1. void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
2.     memcpy(&myData, incomingData, sizeof(myData));
3.     byte index = 0;
4.     ptr = strtok(myData.PaySend, "/");
5.     while (ptr != NULL) {
6.         strings[index] = ptr;
7.         index++;
8.         ptr = strtok(NULL, "/");
9.     }
10.    rpm = atof(strings[0]);
11.    coolant = atof(strings[1]);
12.    eload = atof(strings[2]);
13.    timing = atof(strings[3]);
14.    speedval = atof(strings[4]);
15.    delay(2000);
```



```
16.   Obroty.setValue(rpm);
17.   Temp.setValue(coolant);
18.   Kat.setValue(timing*10);
19.   Obci.setValue(eload);
20.   KMH.setValue(speedval);
21.   rpmpb = rpm/80;
22.   pb1.setValue(rpmpb);
23.   pb2.setValue(coolant);
24.
25.   if(rpmpb < 35){
26.       pb1.Set_font_color_pco(7489);
27.   }
28.   else if(rpmpb<75){
29.       pb1.Set_font_color_pco(43840);
30.   }
31.   else{
32.       pb1.Set_font_color_pco(43136);
33.   }
34.   if(coolant<90){
35.       pb2.Set_font_color_pco(7489);
36.   }
37.   else{
38.       pb2.Set_font_color_pco(43136);
39.   }
40. }
```

Listing 6.5. Sekcja odbiornika ESP8266 przesyłającego dane do Nextion HMI



Zaczynając od początku metoda ta jest utworzona w celu odbioru i przesyłu danych, na wstępie deklarujemy warunki w metodzie gdzie „mac” będzie informacją zwrotną od elementu nadajnika jaki posiada ona MAC adres. „Incoming data” odnosi się do ładunku jaki jest przesyłany pomiędzy ESP za pomocą tej biblioteki, natomiast na samym końcu mamy długość owego ładunku. Kolejna linia kodu ma na celu utworzenia kopii przesyłanych informacji w celu zminimalizowania opóźnień w przesyłaniu danych pomiędzy ESP.

Pętla „while” przetwarza tablice poszatkowanego ładunku w taki sposób, aby każdy z elementów w ładunku został przypisany do odpowiedniej zmiennej jak jest to w naszym przypadku (Trzeba mieć na uwadze, aby wszystkie informacje pokrywały się pomiędzy urządzeniami, ponieważ może to spowodować przepełnienie buforu danych a w następnej kolejności cykliczne resetowanie się układów ESP). Ważnym elementem jest w tym przypadku „delay(2000)” ponieważ zapobiega on przepełnieniu danych wejściowych względem operacji prowadzonych na danych.

Po wykonaniu konwersji danych z typu ciągu znaków na wartość zmiennoprzecinkową wszystkie zmienne zostają przesłane w zamienionej formie, oprócz wartości sterujących poziomem suwaków odpowiedzialnych za wyświetlanie temperatury i obrotów silnika. Obroty silnika zostały obrane jako maksymalna wartość wynosi 8000 Rpm (ang. *Revolutions per minute*) dla jednostki benzynowej, w tym przypadku musimy podzielić wartość otrzymaną przez 80, aby wynik stał się miarą procentową dla naszego wskaźnika. Dodatkowo podczas zmiany tej wartości, jak i temperatury płynu chłodzącego zostaje zmieniony kolor paska kolejno od zielonego po pomarańczowy aż do czerwonego w celu przejrzystej wizualizacji oraz łatwej interpretacji danych bez konieczności skupiania uwagi na wartości.

### 6.3.2. Utworzenie struktury wyświetlacza Nextion HMI

Wykorzystując środowisko Nextion Editor zostały utworzone dwie strony wyświetlające informacje. Jedna z nich przedstawia bardziej czytelny obraz zawierający czyste dane, drugi zaś skonstruowany został w myśli utworzenia kopii elementów deski rozdzielczej z dodatkowymi funkcjami. Projekt ekranu zawierający dane w czystej formie wygląda następująco:

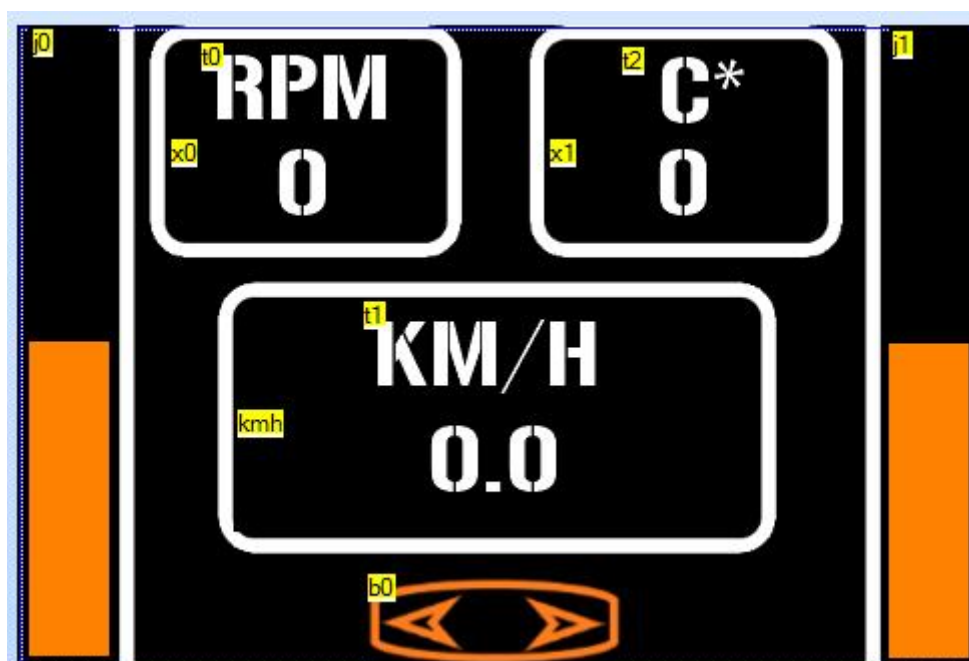


Rys. 6.1. Strona pierwsza zaprojektowana w Nextion Editor

*Źródło: Opracowanie własne*

Jak możemy zauważyć każdy z elementów wykorzystany w projekcie ma swoją unikatową nazwę, ponieważ to dzięki nim komunikujemy się pomiędzy interfejsem a mikrokontrolerem (Ciekawostką jest to, że biblioteka dostarczona przez producenta uwzględnia w przesyle danych takie rzeczy jak numer strony lub jej nazwę oraz konieczność podania unikatowej nazwy elementu, lecz nie wiadomo czemu biblioteka omija numer strony i po prostu szuka elementów o danej nazwie, co prowadzi do konieczności odmiennego nazywania poszczególnych elementów na stronach, ponieważ może dojść do sytuacji gdzie ten sam pakiet danych będzie przesyłany w dwa różne miejsca z czego jedno z nich niekoniecznie będzie tym właściwym).





Rys. 6.2. Strona pierwsza zaprojektowana w Nextion Editor

*Źródło: Opracowanie własne*

Drugi z ekranów który przedstawia rysunek zawiera podobne elementy co ten pierwszy, lecz został on stworzony na potrzeby użytkownika, aby nie była potrzebna zmiana skupienia z drogi na analizę informacji podawanych przez interfejs. Dodatkowym elementem występującym tutaj są paski danych zaprojektowane w taki sposób, aby mogły informować użytkownika w bardziej przyjazny sposób o krytycznych oraz odpowiednich strefach. Przyciski na obu stronach zostały zaprogramowane w interfejsie, a co za tym idzie nawet bez dostępu do danych działają one prawidłowo.

#### 6.4. ESP8266 jako „Czarna skrzynka”

W tym przypadku posiadamy trzy główne elementy. Są to: Esp8266, Raspberry PI oraz znajdująca się na nim baza danych SQLite. Połączenie pomiędzy ESP8266 a Raspberry odbywa się poprzez szynę serial po wybraniu odpowiedniego portu USB. Niestety stwarza to sytuację w której nie możemy odłączyć ESP, a następnie podłączyć go ponownie, ponieważ Raspberry przypisze ESP nie do portu ttyUSB0, lecz kolejnej jego iteracji, co nie pozwala na kontynuację pracy programu napisanego w języku Python.



Próby stworzenia pętli z kolejnymi iteracjami sprawdzającymi dostępność portów USB niestety nie powiodła się ze względów technicznych i ograniczeń sprzętowych. W tym przypadku program nie, posiadający podłączonego żadnego portu USB wchodził w pętle która cały czas działała oraz zawieszała lub resetowała mikrokomputer, wniosku z tym plan został porzucony, ponieważ i tak urządzenie przewidywało stałe połączenie pomiędzy ESP a Raspberry.

#### 6.4.1. ESP8266 przesyłające dane pomiędzy ESP32 a Raspberry PI

Kod powstały w tym przypadku jest prawie kopią jeden do jednego jak było to w przypadku ESP połączonego z interfejsem HMI jedyną różnicą było to, że nie potrzebował on wysyłać dodatkowych wartości podzielonych z ciągu znaków tylko przekazywał on bezpośrednio otrzymaną wartość do Raspberry PI. Prezentuje to następujący listing

```
1. void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {  
2.     memcpy(&myData, incomingData, sizeof(myData));  
3.     Serial.println(myData.PaySend); }  
4. void setup() {  
5.     Serial.begin(115200);  
6.     Wi-Fi.mode(WI-FI_STA);  
7.     if(esp_now_init() != 0){  
8.         return;  
9.     }  
10.    esp_now_register_recv_cb(OnDataRecv);  
11. }  
12. void loop() {  
13. }
```

Listing 6.6. Kod zawarty w ESP8266 połączonego z Raspberry PI



Jak możemy zobaczyć głównym elementem wysyłającym dane jest funkcja „Serial.println(myData.PaySend)” która wysyła dane przez dostępne wyjścia typu Serial (W tym znajdują się wyjścia typu SPI, USB oraz UART), w tym przypadku wykorzystany został port USB.

#### 6.4.2. Raspberry PI zawierające bazę danych SQLite

Na mikrokomputerze Raspberry PI został zainstalowany system Raspbian OS, jest on odmianą dystrybucji Linuksa oparta o Debian oraz przystosowana do pracy z mikroprocesorem znajdującym się na wszystkich wersjach Raspberry, dodatkowo jego licencja działa na zasadach wolnego i otwartego oprogramowania.

Całość systemu została wzbogacona w bazę danych SQLite która ma za zadanie prowadzić dziennik zdarzeń na podstawie otrzymanych danych, całość jednak wymaga odpowiedniej początkowej konfiguracji urządzenia.

Pierwszym krokiem jest aktualizacja wszystkich komponentów Raspberry PI, ponieważ nie tylko korzystać będziemy z najnowszej wersji jądra, ale też usprawni to działanie wersji Pythona 3. A wykonujemy to w taki sposób jak na przedstawionych zdjęciach:

```
pi@raspberrypi: ~  
File Edit Tabs Help  
Most used commands:  
list - list packages based on package names  
search - search in package descriptions  
show - show package details  
install - install packages  
remove - remove packages  
autoremove - Remove automatically all unused packages  
update - update list of available packages  
upgrade - upgrade the system by installing/upgrading packages  
full-upgrade - upgrade the system by removing/installing/upgrading packages  
edit-sources - edit the source information file  
  
See apt(8) for more information about the available commands.  
Configuration options and syntax is detailed in apt.conf(5).  
Information about how to configure sources can be found in sources.list(5).  
Package and version choices can be expressed via apt_preferences(5).  
Security details are available in apt-secure(8).  
This APT has Super Cow Powers.  
pi@raspberrypi:~$ sudo apt update  
Hit:1 http://archive.raspberrypi.org/debian stretch InRelease  
Get:2 http://raspbian.raspberrypi.org/raspbian stretch InRelease [15.0 kB]  
Fetched 15.0 kB in 3s (4,738 B/s)  
Reading package lists... Done  
Building dependency tree... 50%
```

Rys. 6.3. Aktualizacja systemu Raspbian.

*Źródło: Opracowanie własne.*



```
Setting up vlc (3.0.12-0+deb9u1) ...
pi@raspberrypi:~$ sudo apt full-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  coinor-libipopt1v5 libmicrodns0 libmumps-seq-4.10.0 libraw15 libreadline5
  realpath
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  python-colorzero python3-asttokens python3-colorzero uuid-dev
The following packages will be upgraded:
  avahi-daemon bind9-host cifs-utils dpkg-dev erlang-base erlang-crypto
  erlang-syntax-tools libavahi-core7 libavahi-glib1 libavahi-gobject0
  libbind9-140 libbun2 libcupscimage2 libdcs-export162 libdcs162 libdcs-perl
```

Rys. 6.4. Aktualizacja wszystkich podprogramów.

*Źródło: Opracowanie własne.*

```
sudo: snap: command not found
pi@raspberrypi:~$ sudo apt install snapd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  coinor-libipopt1v5 libmicrodns0 libmumps-seq-4.10.0 libraw15 libreadline5
  realpath
Use 'sudo apt autoremove' to remove them.
```

Rys. 6.5. Instalacja programu do aktualizacji jądra systemu.

*Źródło: Opracowanie własne.*

```
File Edit Tabs Help
pi@raspberrypi:~$ sudo snap install core
core 16-2.56.2 from 'canonical' installed
pi@raspberrypi:~$
```

Rys. 6.6. Instalacja najnowszego dostępnego rodzaju jądra systemu.

*Źródło: Opracowanie własne.*

```
Processing triggers for libc-bin (2.24-11+deb9u4) ...
pi@raspberrypi:~$ sudo apt install sqlite3
Reading package lists... Done
Building dependency tree
Reading state information... Done
sqlite3 is already the newest version (3.16.2-5+deb9u3).
The following packages were automatically installed and are no longer required:
```

Rys. 6.7. Instalacja środowiska SQLite.

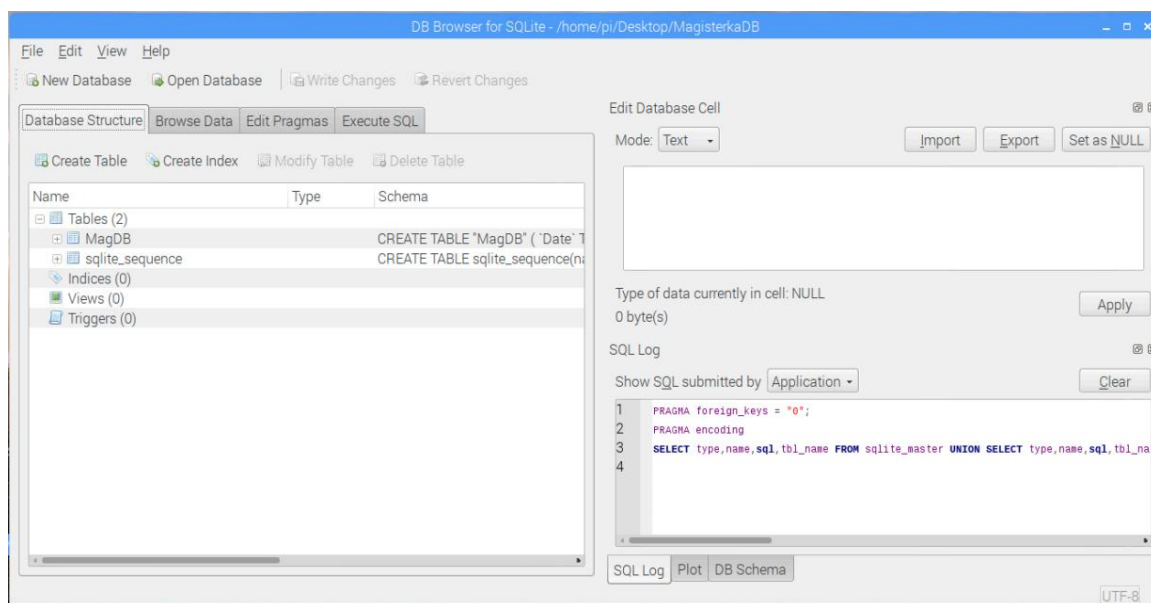
*Źródło: Opracowanie własne.*



Po kolei objaśniając Rys. 6.3. i 6.4. odpowiadają za przygotowanie środowiska do obsługi operacji jakie chcemy wykonać na nim. Aktualizacje są dość istotne w systemach opartych o strukturę Linuxa, ponieważ często są one powiązane z innymi programami. Dajmy na, to jeżeli mamy wersję Python 3.4 a program współpracuje z wersją 3.7 i wyżej system zapyta się nas czy chcemy zaktualizować dany program, ale po drodze może być jeszcze więcej aktualizacji, tak więc lepiej zrobić to na samym początku i cieszyć się z bezproblemowych instalacji programów.

Rys. 6.5. przedstawia komendę instalacji oprogramowania „Snap” wraz z serwisem automatycznego wykrywania aktualizacji oraz błędów. Dzięki temu mamy możliwość obsługi oprogramowania prosto od jego producenta oraz zawsze mamy aktualną wersję oprogramowania dzięki automatycznym sprawdzeniom aktualizacji. Przy wykorzystaniu tego oprogramowania aktualizujemy również jądro systemu co daje nam stałą aktualną wersję jądra, komenda potrzebna do danej operacji jest przedstawiona na Rys. 6.6.

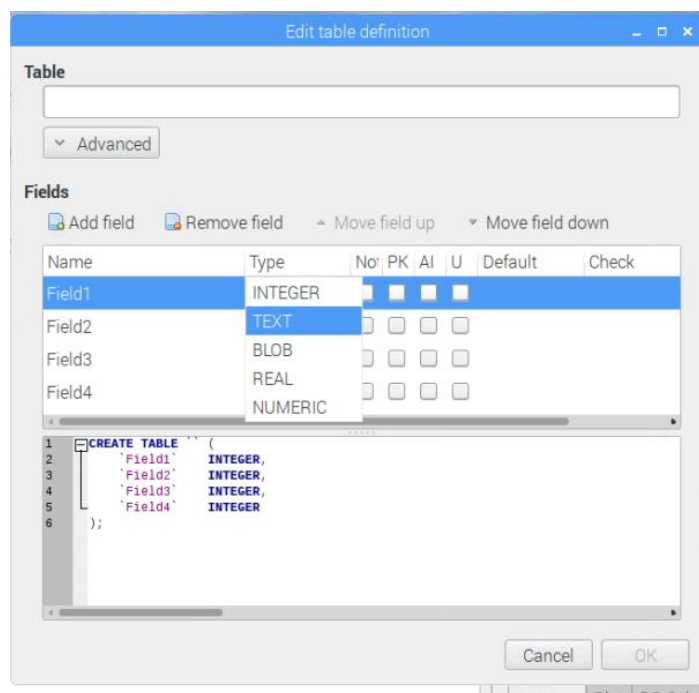
Na samym końcu instalujemy SQLite w wersji trzeciej co przedstawia Rys. 6.7. Dobrym wyborem jest również instalacja dodatkowego oprogramowania do obsługi baz danych SQLite. W tym przypadku padło na „Sqlitebrowser” instalowany również przez oprogramowanie „Snap”. Komenda wygląda następująco: „sudo snap install sqlitebrowser”. Po instalacji mamy dostęp do kreatora oraz edytora baz danych SQLite. Jego wygląd przedstawia Rys. 6.8.:



Rys. 6.8. Środowisko SQLitebrowser.

*Źródło: Opracowanie własne.*

Środowisko to pozwala na utworzenie baz danych w sposób „Point and click”, co stosunkowo przyspiesza cały proces i pozwala na szybsze zrozumienie architektury SQLite. Głównym problemem w tym środowisku jest struktura danych która jest w dużym stopniu inna od standardowego SQL, przedstawia to następujące zdjęcie:



Rys. 6.9. Instalacja środowiska SQLite.

*Źródło: Opracowanie własne.*

Jak możemy zaobserwować na Rys. 6.9. system ten ogranicza się do 5 rodzajów typów danych. Oznaczają one kolejno

- INTEGER - przeznaczony dla liczb całkowitych maksymalnie przechowywanych na 8 bitach.
- TEXT - są to ciągi znaków.
- BLOB - odpowiada to za dane binarne lub inaczej „czyste”
- REAL - służy do przechowywania danych zmiennoprzecinkowych.
- NUMERIC - służy on do konwersji danych przychodzących w zależności od przeznaczenia dostosowując się on pomiędzy INTEGER a REAL.

W oparciu o pozyskane informacje została stworzona baza danych o nazwie MagisterkaDB zaś w niej tabela o nazwie MagDB zawierająca kolumny: Date, Rpm, Coolant, EngineLoad, EngineTiming oraz KMH. Przedstawia to Rys. 6.10. (Dane zapisane na zdjęciu nie są danymi testowymi).





	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
	Filter	Filter	Filter	Filter	Filter	Filter
81	2022-08-01 18:53:16.277769+0...	2051.25	60	9.8	45	0
82	2022-08-01 18:53:17.437021+0...	2455.75	61	10.2	48	0
83	2022-08-01 18:53:18.578713+0...	2637	61	10.2	-4	0
84	2022-08-01 18:53:19.736553+0...	1824.75	61	3.92	19.5	0
85	2022-08-01 18:53:20.896989+0...	1075.75	61	3.53	12.5	0
86	2022-08-01 18:53:22.041062+0...	863.75	61	3.53	12.5	0
87	2022-08-01 18:53:23.187862+0...	772.75	61	3.14	15	0
88	2022-08-01 18:53:24.349519+0...	746.25	62	3.14	15.5	0
89	2022-08-01 18:53:25.437452+0...	753.75	62	3.14	12	0
90	2022-08-01 18:53:26.567822+0...	752.25	62	3.14	15.5	0
91	2022-08-01 18:53:27.715733+0...	749.25	62	3.14	12.5	0
92	2022-08-01 18:53:28.896989+0...	753.75	62	3.14	12.5	0

Rys. 6.10. Struktura tabeli SQLite.

*Źródło: Opracowanie własne.*

Po utworzeniu bazy danych SQLite został stworzony program odpalany przy starcie systemu przy wykorzystaniu pliku „shell” utworzonego w celu rozruchu poprzez „crontab”, a jest to nic innego jak cykliczna lista zadań, w zależności od potrzeb docelowego programu lub skryptu (W naszym przypadku będzie to skrypt powłoki „Shell”). Dany plik wykonawczy powłoki wygląda następująco:

```
1. #!/bin/sh
2. # launcher.sh
3. cd
4. cd /home/pi/Desktop
5. sudo python3 USBSERIAL.py
6. cd
```

Listing 6.7. Plik wykonawczy powłoki.





Jego zadaniem w całym założeniu jest przekazanie powłocie odpowiednich instrukcji do odpalenia programu „SERIALUSB.py” który wygląda następująco:

```
1. import serial
2. import sqlite3
3. from datetime import datetime, timezone
4. if __name__ == '__main__':
5.     ser = serial.Serial("/dev/ttyUSB0", 115200, timeout=
    1)
6.     ser.reset_input_buffer()
7.     while True:
8.         if ser.in_waiting > 0:
9.             line = ser.readline().decode('utf-8').rstrip
    ()
10.             lineDecoder = line.split("/")
11.             rpm = lineDecoder[0]
12.             coolant = lineDecoder[1]
13.             eload = lineDecoder[2]
14.             timing = lineDecoder[3]
15.             speedval = lineDecoder[4]
16.             date = datetime.now()
17.             date = date.replace(tzinfo=timezone.utc)
18.             try:
19.                 sqliteConnection = sqlite3.connect(r"/h
    ome/pi/Desktop/MagisterkaDB")
20.                 cursor = sqliteConnection.cursor()
21.                 print("Successfully Connected")
22.                 sqlite_insert_query = """INSERT INTO Ma
    gDB VALUES ('{}', {}, {}, {}, {}, {})""".format(date, rpm, cool
```



```
ant, eload, timing, speedval)

23.             conut = cursor.execute(sqlite_insert_query)
               ery)

24.             sqliteConnection.commit()

25.             print("Record inserted", cursor.rowcount)
               t)

26.             cursor.close()

27.             except sqlite3.Error as error:

28.             print("Failed to insert data into table
               ", error)

29.             finally:

30.             if sqliteConnection:

31.             sqliteConnection.close()

32.             print("The SQLite con closed")
```

Listing 6.8. Kod programu obsługującego bazę danych SQLite w języku Python

Składnia języka Python jest dość specyficzna i wymaga zachowania odpowiednich wcięć odnośnie do linii, ponieważ to dzięki nim interpretuje on przynależność do pętli czy innych obiektów. Kolejnym z odmiennych elementów tego języka jest brak konieczności deklaracji jakiego typu ma być zmienna. Więcej specyficznych elementów oraz informacji o środowisku dostarcza książka [4].

Przedstawiony program zaczyna się od sformułowania „if \_\_name\_\_ == „\_\_main\_\_”” ma to na zadaniu zapewnienia programu, aby działał on we własnej przestrzeni bez udzielania dostępu do innych programów zewnętrznych. Kolejne dwie linie przedstawiają utworzenie komunikacji Raspberry PI z ESP8266 za pomocą protokołu Serial. W tym przypadku odnajdujemy urządzenie które zostało przypisane do pliku ttyUSB0 w folderze systemu „dev”.

Program z góry zakłada, że ESP jest podpięte do niego i jest jedynym urządzeniem znajdującym się w porcie USB. Dalszy warunek sprawdza, czy w szynie Serial znajdują się informacje, a kolejne linie kodu dekodują oraz zostają przypisane zmiennym dane



zawarte w przesyłanym sygnale. Dodatkowo utworzona zostaje nowa zmienna zawierająca aktualną datę i czas w celu zapisu dokładnego stanu z danego momentu.

Kolejny blok decyzyjny próbuje utworzyć połączenie z lokalną bazą danych poprzez dostęp do pliku, jeżeli się to nie uda błąd jest zapisywany w pliku na dysku Raspberry PI w postaci dziennika zdarzeń, podobnie jest, kiedy w którymś momencie program nie może funkcjonować poprawnie.

Jeżeli połączenie się uda wykonywane są dwie fazy programu. W pierwszej program stara się umieścić dane pozyskane z wcześniejszego bloku do zapytania SQL za pomocą utworzonej zmiennej, a następnie zostaje wysłane zapytanie w postaci umieszczenia danych na odpowiednie miejsca. Dodatkowo jeżeli program zostanie uruchomiony w konsoli mamy możliwość podejrzenia działania aplikacji ze względu umieszczenia wyświetlenia powiadomień o krytycznych elementach. Jeżeli proces nie powiedzie się program zapisze problem w pliku „Cronlog”.

Na samym końcu zostaje zamknięte połączenie do bazy danych oraz zostaje wysłana wiadomości w celu weryfikacji powodzenia.

Po utworzeniu bazy danych oraz programu do jej obsługi, jesteśmy zdolni do zbierania danych które nie są dostępne dla zwykłego użytkownika samochodu. Dodatkowo posiadamy coś w rodzaju „Czarnej skrzynki” Znajdującej się w samolotach, jeżeli coś się stanie podczas jazdy będzie możliwa analiza danych do ustalenia zachowań pojazdu a co za tym idzie zachowań użytkownika. Na przykład czy gaz samochodu w danym przypadku nie był agresywnie dodawany daje nam to informacja o obciążeniu silnika oraz obroty. Czy prędkość w danym miejscu nie była za wysoka, czy obroty silnika nie wskazywałyby na to, że silnik zgaś co skutkowałoby odcięciem układu wspomagania? Jak widać wiele z tych czynników pomaga w interpretacji co się działo podczas jazdy.

Dodatkowym atutem jest to, że możemy zbierać informacje na temat pracy silnika przez dłuższy okres, czasu a później interpretować je i porównywać z wynikami które uzyskaliśmy na przykład rok temu, może to pozwolić na wykrycie nieprawidłowości oraz ominięcie nieoczekiwanych kosztów naprawy.

## 7. Pomiary

### 7.1. Pomiary uzyskane w czasie rzeczywistym z wyświetlacza

Pierwszy pomiar który przedstawia następujące zdjęcie został wykonany w momencie postoju samochodu.



Rys. 7.1. Pomiar podczas postoju

*Źródło: Opracowanie własne*

Jak widzimy na przedstawionym zdjęciu silnik był odpowiednio dogrzany oraz obroty silnika wynosiły 721 co dla tej jednostki napędowej jest w normie obrotów jałowych (Norma dla silnika benzynowego 2.4 v6 wynosi od 680 do 810). Obciążenie silnika występuje, ponieważ podczas prowadzonych testów pompa klimatyzacji pracowała nieprzerwanie ze względu na panujące warunki atmosferyczne.

Warto wspomnieć, że obciążenie silnika jest wysoce zależne od jakości i stopnia zużycia pompy klimatyzacji, jeżeli pompa była bardzo eksploatowana jej wydajność może się nie zmniejszyć, lecz silnik odczuje to jako potrzebę większego nakładu energii do jej napędzenia co za tym idzie większej dawki paliwa. Kąt wyprzedzenia zapłonu w tym przypadku mieści się w granicach poprawnego działania jednostki i nie wskazuje na potencjalne zagrożenie ze względu bocznego zapłonu.



Pomiar drugi został przeprowadzony w normalnych warunkach spokojnej jazdy bez gwałtownych akcji (Dodawania lub ujmowania gazu).



Rys. 7.2. Pomiar przy ciągłej pracy w okolicach 100 km/h na ekranie danych szczegółowych

*Źródło: Opracowanie własne*



Rys. 7.3. Pomiar przy ustabilizowanej prędkości na ekranie uproszczonym

*Źródło: Opracowanie własne*



Jak widzimy na Rys. 7.2. obciążenie silnika wynosi 36% z czego obroty stanowią 34% z maksymalnej wartości obrotów a klimatyzacja oraz reszta osprzętu silnika dodatkowe 2% co się zgadza z otrzymanym wynikiem i ukazuje prawidłowość działania jednostki pod tym względem. Możemy również zauważyć, że kąt zapłonu został ustawiony na 22.5 stopnia co może wskazywać na ubogą mieszankę paliwową a bardziej na niedotlenienie jednostki względem paliwa (W tym przypadku LPG), ponieważ samochód próbuje przyspieszyć moment zapalenia mieszanki.

Inaczej wygląda to w przypadku gwałtownego odjęcia gazu wtedy komputer będzie próbował opóźnić zapalenie mieszanki, ponieważ ilość paliwa która trafia do bloku jest stosunkowo mniejsza i musi zgrać wszystkie tłoki w taki sposób, aby nie doszło do kolizji. Przedstawia to następujące zdjęcie:



Rys. 7.4 Spowolniony kąt zapłonu

*Źródło: Opracowanie własne*

Jak możemy zauważyć ECU zareagowało na gwałtowne odjęcie gazu regulując kąt zapłonu w taki sposób, aby go spowolnić i zsynchronizować z resztą elementów. Dodatkowo odjęcie gazu sprawia, że silnik nie posiada żadnego obciążenia względem obrotów a jedynie jest zależne od osprzętu który się na nim znajduje. Obroty w tym momencie były w tendencji spadkowej.



Odmienny wynik uzyskujemy, kiedy potrzebujemy gwałtownie przyspieszyć wykorzystując do tego zmianę biegu o jeden w dół, powoduje to gwałtowny wzrost obciążenia silnika a co za tym idzie kąt zapłonu musi odpowiednio zostać dostosowany, co ukazuje następujące zdjęcie:



Rys. 7.5. Gwałtowne wywołanie wyższych obrotów poprzez zmianę biegu o jeden w dół

*Źródło: Opracowanie własne*

Jak możemy zauważyć schemat obciążenia silnika zostaje zachowany, czyli 55% jego obciążenia to przeniesienie obrotów na skrzynie za to reszta odnosi się do osprzętu silnika, lecz w tym przypadku kąt wyprzedzenia zapłonu jest stosunkowo wyższy niż było to w przypadku stabilnej jazdy przy prędkości 100 km/h, a dzieje się to za sprawą potrzeby dostarczenia większej ilości dawki paliwowej co wymaga odpowiedniego zwiększenia odstępu wywołania iskry zapłonu względem położenia tłoka silnika. Na drugiej stronie ekranu wyniki prezentowały się następująco:



Rys. 7.6. Wyniki wyświetlone na stronie uproszczonej przy zrzuceniu biegu w dół

*Źródło: Opracowanie własne*

Jak możemy zaobserwować kolor paska po lewej odpowiedzialnego za obroty silnika zmienił swój kolor na pomarańczowy co informuje kierowcę o przekroczeniu ekonomicznej oraz zdrowej granicy prędkości obrotowej silnika, kolejną strefą jest strefa czerwona która informuje nas o konieczności zmniejszenia obrotów w celach bezpieczeństwa jednostki napędowej, gdyż może ona ulec zniszczeniu. Podobny element znajduje się na pasku po prawej stronie który jest zależny od temperatury silnika. Posiada on dwa stany gdzie zielona strefa zaczyna się od 0 stopni a kończy na 99, a czerwona informuje nas o zbyt wysokiej temperaturze płynu chłodniczego który przekroczył 100 stopni i może spowodować wrzenie płynu oraz rozszczelnienie układu.

## 7.2. Pomiary gromadzone w bazie danych

Badania przeprowadzone w tej sekcji opierały się o dane zebrane podczas przejazdów próbnych oraz postoju pojazdu. Dzięki temu możemy w prosty sposób przeanalizować dane pod kątem odchyłeń i nieprawidłowości mogących prowadzić do usterek pojazdu. Całość analizy będzie się opierać jedynie o zapytania wykonywane na bazie danych utworzonej wewnątrz Raspberry Pi.





Otwierając bazę danych SQLite w wykorzystanym przez nas edytorze mamy możliwość podejrzenia danych znajdujących się w niej oraz jesteśmy w stanie używać zapytań SQL do odpowiedniego odfiltrowywania informacji w taki sposób, aby były one przyjazne do interpretacji. Widok podglądu danych przedstawia następujące zdjęcie:

	Date	Rpm	Coolant	EngineLoad	Engine
437	2022-08-13 ...	725.5	97	3.92	9.5
438	2022-08-13 ...	1369	98	7.84	27.5
439	2022-08-13 ...	1494	98	6.27	33.5
440	2022-08-13 ...	1571.25	97	5.49	33
441	2022-08-13 ...	1134.25	97	3.92	29
442	2022-08-13 ...	1907.5	96	12.94	39.5
443	2022-08-13 ...	3250.5	96	3.92	37.5
444	2022-08-13 ...	1873	96	8.63	39.5
445	2022-08-13 ...	1882.5	96	7.06	38

Rys. 7.7 Widok zapisanych danych w bazie danych SQLite z przeprowadzonych badań

*Źródło: Opracowanie własne*

Kolejnym etapem do przeprowadzenia analizy danych jest utworzenie odpowiednich zapytań które będą odpowiednio filtrowały informacje jakie chcemy uzyskać. Pierwszym zastosowanym filtrem będzie sprawdzenie jakie dane uzyskaliśmy konkretnego dnia o konkretnej godzinie bez podawania w której sekundzie pomiar miałby zostać wykonany.



The screenshot shows a SQLite database window with a query editor and a results table. The query is: `select * from MagDB where MagDB.Date like '%2022-08-17 18:19%'`. The results table contains 12 rows of data, including timestamps, RPM, coolant levels, engine load, engine timing, and speed (KMH).

	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
1	2022-08-17 18:19:30.046047+00:00	767	60	3.53	19.5	0
2	2022-08-17 18:19:31.749208+00:00	800.5	60	4.31	10.5	0
3	2022-08-17 18:19:32.604911+00:00	749.5	60	4.31	8	0
4	2022-08-17 18:19:33.922532+00:00	788.75	60	4.31	6.5	0
5	2022-08-17 18:19:35.283914+00:00	788.75	60	4.31	6	0
6	2022-08-17 18:19:37.733505+00:00	768.5	60	3.92	10.5	0
7	2022-08-17 18:19:37.894273+00:00	798.5	60	3.92	12	0
8	2022-08-17 18:19:39.189391+00:00	765.5	60	3.53	12.5	0
9	2022-08-17 18:19:40.562053+00:00	760.25	60	3.92	12.5	0
10	2022-08-17 18:19:41.887130+00:00	775.25	60	3.92	10.5	0
11	2022-08-17 18:19:43.234888+00:00	761.75	60	3.92	12.5	0
12	2022-08-17 18:19:44.553679+00:00	773.25	60	3.92	11	0

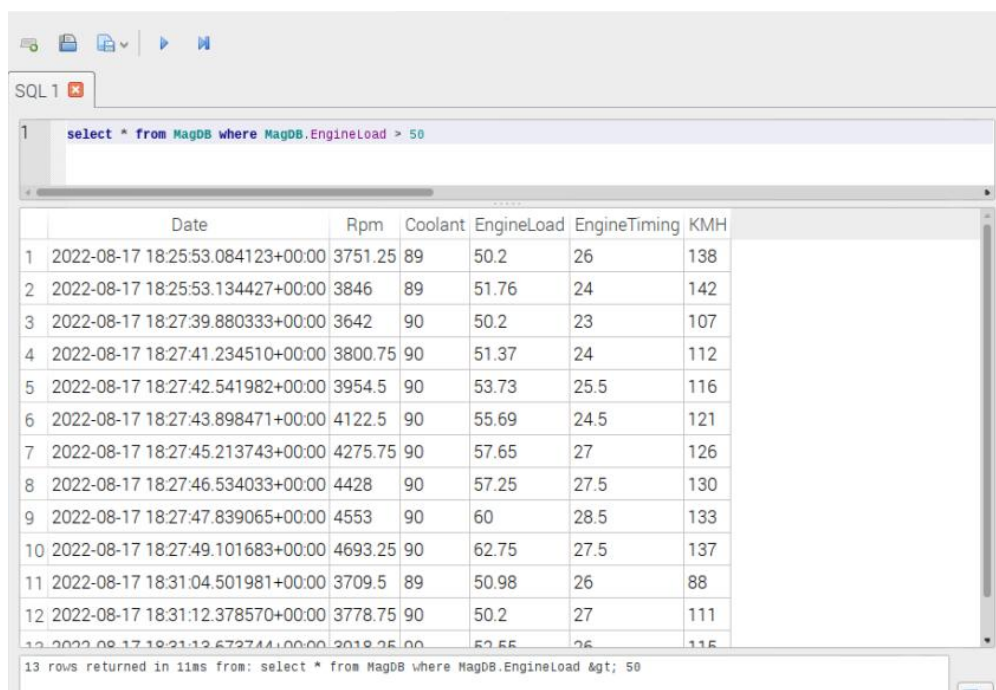
23 rows returned in 19ms from: select \* from MagDB where MagDB.Date like '%2022-08-17 18:19%'

Rys. 7.8. Filtr daty oraz godziny zastosowany na bazie SQLite

*Źródło: Opracowanie własne*

Jak możemy zauważyć filtr zawiera instrukcje „Like” co oznacza, że silnik bazy ma przefiltrować dane i znaleźć o takim samym początku zapisywanej daty, lecz informujemy go również o braku istoty co ma się znajdować po podanych wartościach za pomocą znaku procentu. W taki sposób znaleźliśmy wszystkie wykonane pomiary wykonane 17 sierpnia o godzinie 18:15.

Kolejnym etapem analizy pomiarów jest sprawdzenie poprawności działania pojazdu i czy występowały jakieś nieprawidłowości w wynikach podczas wykonywania testów systemu. Elementy te przedstawiają następujące rysunki:



SQL 1

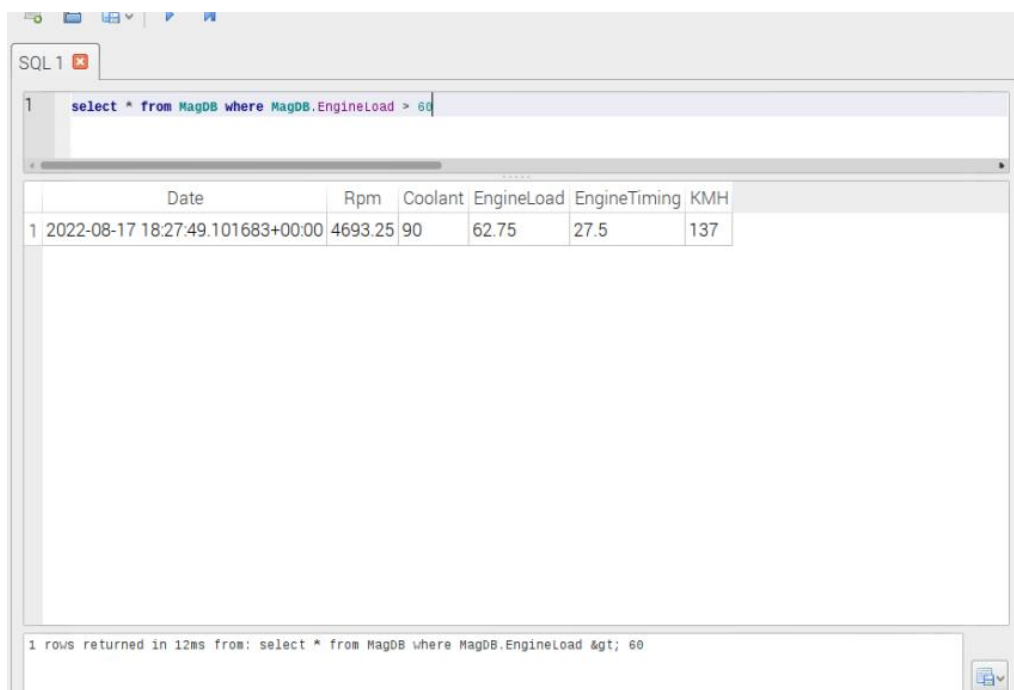
```
1 select * from MagDB where MagDB.EngineLoad > 50
```

	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
1	2022-08-17 18:25:53.084123+00:00	3751.25	89	50.2	26	138
2	2022-08-17 18:25:53.134427+00:00	3846	89	51.76	24	142
3	2022-08-17 18:27:39.880333+00:00	3642	90	50.2	23	107
4	2022-08-17 18:27:41.234510+00:00	3800.75	90	51.37	24	112
5	2022-08-17 18:27:42.541982+00:00	3954.5	90	53.73	25.5	116
6	2022-08-17 18:27:43.898471+00:00	4122.5	90	55.69	24.5	121
7	2022-08-17 18:27:45.213743+00:00	4275.75	90	57.65	27	126
8	2022-08-17 18:27:46.534033+00:00	4428	90	57.25	27.5	130
9	2022-08-17 18:27:47.839065+00:00	4553	90	60	28.5	133
10	2022-08-17 18:27:49.101683+00:00	4693.25	90	62.75	27.5	137
11	2022-08-17 18:31:04.501981+00:00	3709.5	89	50.98	26	88
12	2022-08-17 18:31:12.378570+00:00	3778.75	90	50.2	27	111
13	2022-08-17 18:31:13.672744+00:00	3018.25	90	52.55	26	115

13 rows returned in 11ms from: select \* from MagDB where MagDB.EngineLoad > 50

Rys. 7.9. Zapytanie szukające obciążnika

*Źródło: Opracowanie własne*



SQL 1

```
1 select * from MagDB where MagDB.EngineLoad > 60
```

	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
1	2022-08-17 18:27:49.101683+00:00	4693.25	90	62.75	27.5	137

1 rows returned in 12ms from: select \* from MagDB where MagDB.EngineLoad > 60

Rys. 7.10. Znalezienie momentu największego obciążenia silnika

*Źródło: Opracowanie własne*



1 `select * from MagDB where MagDB.EngineTiming < 0`

	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
1	2022-08-01 18:53:18.578713+00:00	2637	61	10.2	-4	0
2	2022-08-17 18:22:11.442299+00:00	2779	80	4.31	-7	47
3	2022-08-17 18:22:22.032854+00:00	2261	81	3.92	-7	51
4	2022-08-17 18:22:23.338038+00:00	2213.5	81	3.92	-7	50
5	2022-08-17 18:23:29.161171+00:00	2796.75	84	3.92	-2.5	56
6	2022-08-17 18:24:08.707142+00:00	2723.25	87	4.31	-1	77
7	2022-08-17 18:24:17.864434+00:00	1167	87	2.75	-2.5	71
8	2022-08-17 18:24:19.164570+00:00	1919.25	87	3.53	-7	68
9	2022-08-17 18:24:20.533464+00:00	1862.75	87	3.14	-7	66
10	2022-08-17 18:24:21.829920+00:00	1765.5	87	3.14	-6	62
11	2022-08-17 18:24:23.128776+00:00	1661	87	2.75	-5.5	60
12	2022-08-17 18:24:24.426530+00:00	1626	87	2.75	-5.5	58
13	2022-08-17 18:24:26.246676+00:00	2160.25	87	2.52	-7	77

55 rows returned in 9ms from: select \* from MagDB where MagDB.EngineTiming < 0

Rys. 7.11. Wyszukanie momentów ujemnego kąta zapłonu

*Źródło: Opracowanie własne*

1 `select * from MagDB where MagDB.KMH < 60 AND MagDB.Rpm > 3000`

	Date	Rpm	Coolant	EngineLoad	EngineTiming	KMH
1	2022-08-13 16:42:27.368246+00:00	3250.5	96	3.92	37.5	0
2	2022-08-17 18:23:13.507440+00:00	3010	87	34.12	27	48
3	2022-08-17 18:23:14.810304+00:00	3419.75	87	36.47	29	55
4	2022-08-17 18:23:23.981727+00:00	3886.5	85	11.76	48	58
5	2022-08-17 18:23:25.279441+00:00	3845	84	12.16	48.5	58
6	2022-08-17 18:23:26.572143+00:00	3819.5	84	12.16	48.5	57
7	2022-08-17 18:23:27.861931+00:00	3777.75	84	9.02	8	57

7 rows returned in 13ms from: select \* from MagDB where MagDB.KMH < 60 AND MagDB.Rpm > 3000

Rys. 7.12. Odnalezienie momentów większych obrotów przy prędkości niższej niż 60 km/h

*Źródło: Opracowanie własne*



Zaczynając od zdjęcia które przedstawia Rys. 7.9. możemy zauważyć, że jednostka napędowa nie posiada odchyień on norm obciążenia silnika, a udowadnia to obliczenie stosunku obrotów silnika do ich maksymalnej wartości wraz z obciążeniami występującymi na, wskutek wykorzystania części jego mocy do zasilenia dodatkowego osprzętu na nim znajdującego się. Biorąc pod uwagę pierwszy wynik oraz to, że maksymalne obroty silnika podawane przez producenta to 8000 Rpm możemy obliczyć, ponieważ obciążenie jakie wywierają obroty wynosi 46,8%, co daje nam 3,4% jako moc przekazywana na elementy napędzane przez silnik, znaczy to o poprawności działania jednostki napędowej.

Kontynuując Rys. 7.10. ukazuje moment w którym silnik przekracza 60% obciążenia i analogicznie do poprzednio wykonanych kalkulacji możemy dowiedzieć się, że udział obrotów w obciążeniu wynosił 58,6% a osprzętu 1,4%. Niskie obciążenie osprzętu może być zinterpretowane jako moment w którym sprzęgło elektroniczne rozłączyło pompę klimatyzacji wskutek uzyskania komfortowej temperatury w kabinie.

Inną kwestię porusza Rys. 7.11. Który przedstawia momenty w których silnik doznał negatywnego kąta zapłonu. Jak możemy zauważyć żadna z wartości ujemnych nie przekracza progu -8 stopni co informuje nas o poprawności działania świec zapłonowych, modułu zapłonu oraz kompresji zachodzącej w komorze spalania. Dodatkowo analizując wyniki kąta zapłonu wraz z obciążeniem silnika widoczne jest to, że wartości te nie występowały podczas przekazywania momentu obrotowego na skrzynię biegów, co tylko utwierdza nas w tym, iż jednostka napędowa działa prawidłowo.

Ostatnią analizą danych było sprawdzenie możliwości odczytu parametrów podczas gwałtownych przyspieszeń na co wskazują obroty silnika powyżej 3000 oraz prędkość poniżej 60 km/h. Przedstawione warunki prezentuje Rys. 7.12. Jak możemy zauważyć wartość pierwsza i ostatnia nie przedstawia warunków znaczącego przyspieszania a jedynie podwyższonych obrotów oraz hamowania sprzęgłem. Jednakże reszta wyników wykazuje się cechami gwałtownego przyspieszenia. Pomiary te mogą zostać wykorzystane jako informacja istotna w sprawie kolizji lub również niewiadomej usterki silnika.



## 8. Podsumowanie

Cel tej pracy został zdefiniowany aby potwierdzić tezę, czy jest możliwe utworzenie urządzenia zdolnego do odczytu informacji z komputera samochodu poprzez wykorzystanie tanich technologii IoT do szybkiej ich analizy.

W niniejszej pracy opisano rodzaje i zastosowania komputerów pokładowych oraz interfejsów komunikacyjnych takich jak ECU w pojazdach które posiadają element głównego komputera sterującego pracą silnika, FIS wykorzystywany w większości aut importowanych z Europy Zachodniej oraz Protokoły SENT i KWP wraz z magistralami LIN i CAN stosowanymi w komunikacji pomiędzy komponentami sieci wewnętrznej pojazdów.

Opisane zostały wymagania funkcyjne i нефункционаłne systemu oraz utworzona została specyfikacja wymagań dla realizowanego projektu, umoralniając jego spójniejszą i efektywniejszą realizację.

Na podstawie wymagań funkcjonalnych i specyfikacji systemu utworzono projekt nieinwazyjnego urządzenia pozwalającego na uzyskiwanie oraz gromadzenie szeregu wielu informacji niedostępnych dla przeciętnego posiadacza samochodu w czasie rzeczywistym. Projekt urządzenia jest oparty o stosunkowo tanie rozwiązania technologiczne IoT takie jak mikroprocesorowe moduły ESP, Raspberry Pi oraz interpreter ELM327 w izolowanej sieci wymiany informacji.

Zgodnie z opisany projektem zostało utworzone urządzenie na które składa się system z elementem nadajnika oraz punktami końcowymi w postaci odbiorników a sam system został przygotowany do możliwej rozbudowy o kolejne elementy rozszerzające jego funkcjonalności.

Dodatkowo utworzono interfejs typu człowiek maszyna do wizualizacji informacji pozyskiwanych prosto z pojazdu w oparciu o programowalny wyświetlacz dotykowy Nextion HMI o ciekłokrystalicznej matrycy TFT (ang. *Thin Film Transistor*).

W ramach pracy dokonano także pomiarów uzyskanych w czasie rzeczywistym które były dostępne podczas pracy pojazdu z równoczesnym zapisem bieżącym



w wykreowanej bazie SQLite dla utworzenia możliwości analizy przejazdów wykonanych podczas działania systemu.

Podsumowując, cel pracy udało się osiągnąć mimo problemów napotkanych na drodze ze względów sprzętowych i dostępności elementów. System działa prawidłowo przekazując dane specyficzne niedostępne w normalnych warunkach dla użytkownika oraz wykazuje wiele możliwości rozwoju o dodatkowe elementy. Całość pracuje na zasadach sieci wewnętrznej IoT, odpornej na ataki zewnętrzne co zwiększa bezpieczeństwo osoby korzystającej z urządzenia podpiętego do samochodu. Mając na uwadze powyższe teza pracy została potwierdzona.





## Literatura

- [1] T. Denton, *Advanced Automotive Fault Diagnosis*, Wydawnictwo Taylor & Francis, 2020.
- [2] N. Kolban, *Kolban's Book on ESP8266*, Wydawnictwo EspressIf, Shanghai 2016.
- [3] G. Halfacree, *The official Raspberry Pi Beginner's Guide*, Wydawnictwo Raspberry Pi Press, 2020.
- [4] M. Lutz, *Lerning Python*, Wydawnictwo O'Reilly Media, 2013.
- [5] F. daCosta, *Rethinking the Internet of Things A Scalable Approach to Connecting Everything*, Wydawnictwo Apress Open, 2013.
- [6] [https://wiki.dfrobot.com/FireBeetle\\_ESP32\\_IOT\\_Microcontroller\(V3.0\)\\_Supports\\_Wi-Fi\\_&\\_Bluetooth\\_SKU\\_DFR0478](https://wiki.dfrobot.com/FireBeetle_ESP32_IOT_Microcontroller(V3.0)_Supports_Wi-Fi_&_Bluetooth_SKU_DFR0478) (Stan z dnia 14.08.2022r.)
- [7] [https://nextion.tech/editor\\_guide/](https://nextion.tech/editor_guide/) (Stan z dnia 14.08.2022r.)
- [8] <https://www.elmelectronics.com/products/ics/obd/> (Stan z dnia 14.08.2022r.)
- [9] <https://www.espressif.com/en/news/ESP-NOW> (Stan z dnia 14.08.2022r.)
- [10] <https://ep.com.pl/files/11117.pdf> (Stan z dnia 14.08.2022r.)
- [11] <https://it.dru.ac.th/o-bookcs/pdfs/15.pdf> (Stan z dnia 15.08.2022r.)
- [12] <https://ep.com.pl/files/11457.pdf> (Stan z dnia 15.08.2022r.)
- [13] M. Margolis, *Arduino Cookbook*, Wydawnictwo O'Reilly Media, 2011.
- [14] D. Crolla, *Automotive Engineering: Powertrain, Chassis System and Vehicle Body*, Wydawnictwo Butterworth-Heinemann, 2009
- [15] M. Kardaś, *Magistrala Can/Lin Od A Do Z. Diagnostyka i programowanie*, Wydawnictwo ATNEL, 2020





- [16] J. Haynes, *Automotive Anti-lock Brake Systems (ABS)*, Wydawnictwo Motorbooks INTL, 2000
- [17] A. Grzemba, *LIN-BUS: Systeme, Protokolle, Tests von LIN-Systemen, Tools, Hardware. Applikationen (Wersja tłumaczona)*, Wydawnictwo Franzis, 2005
- [18] L. Bee, *PLC and HMI Development with Siemens TIA Portal*, Wydawnictwo Packt Publishing, 2022
- [19] A. Molinaro oraz R. de Graaf, *SQL Cookbook*, Wydawnictwo O'Reilly Media, 2020
- [20] J. Feiler, *Introducing SQLite for Mobile Developers*, Wydawnictwo Apress, 2015
- [21] K. Trzeciak, *Diagnostyka samochodów osobowych*, Wydawnictwo WKŁ, 2013
- [22] M. Doległo, *Podstawy elektrotechniki i elektroniki*, Wydawnictwo WKŁ, 2016
- [23] N. Cameron, *Electronics Projects with the ESP8266 and ESP32*, Wydawnictwo Apress, 2020
- [24] V. Ozan Oner, *Developing IoT Projects with ESP32*, Wydawnictwo Packt Publishing, 2021
- [25] G. Schneehage, *Elementy wykonawcze układu sterowania silnika w praktyce warsztatowej*, Wydawnictwo WKŁ, 2019
- [26] L. Gabriel, *Development and Formal Verification of TTCAN*, Wydawnictwo OmniScriptum GmbH & Co. KG, 2010
- [27] Dr. Lucy Rogger oraz Dr. Andy Standford-Clark, *Wiring the IoT*, Wydawnictwo O'Reilly Media, 2017
- [28] R. C. Seacord, *Effective C*, Wydawnictwo No Starch Press US, 2020



## Spis rysunków

Rys. 4.1. Schemat wyjść gniazda OBDII .....	13
Rys. 4.2. Nieprawidłowości zamówionych układów .....	15
Rys. 4.3. Raspberry PI Pico .....	22
Rys. 4.4. Raspberry PI Zero .....	23
Rys. 4.5. Raspberry PI 4 B .....	24
Rys. 4.6. Widok Aplikacji Nextion Editor .....	26
Rys. 4.7. Interfejs Nextion HMI 2.4 Cala .....	28
Rys. 5.1 Przeprogramowanie zdefiniowanej funkcji .....	30
Rys. 5.2. Schemat poglądowy utworzonego systemu wraz z opisami połączeń oraz nazwami elementów .....	31
Rys. 5.3. Schemat działania nadajnika ESP32 pobierającego informacje od ELM327 ....	32
Rys. 5.4. Schemat działania odbiornika ESP8266 połączonego z interfejsem Nextion...	33
Rys. 5.5. Działanie odbiornika ESP8266 połączonego z Raspberry Pi oraz Bazą SQLite33	
Rys. 6.1. Strona pierwsza zaprojektowana w Nextion Editor .....	52
Rys. 6.2. Strona pierwsza zaprojektowana w Nextion Editor .....	53
Rys. 6.3. Aktualizacja systemu Raspbian .....	55
Rys. 6.4. Aktualizacja wszystkich podprogramów. ....	56
Rys. 6.5. Instalacja programu do aktualizacji jądra systemu. ....	56
Rys. 6.6. Instalacja najnowszego dostępnego rodzaju jądra systemu. ....	56
Rys. 6.7. Instalacja środowiska SQLite. ....	56
Rys. 6.8. Środowisko SQLitebrowser. ....	58
Rys. 6.9. Instalacja środowiska SQLite. ....	59
Rys. 6.10. Struktura tabeli SQLite. ....	60
Rys. 7.1. Pomiar podczas postoju .....	64



Rys. 7.2. Pomiar przy ciągłej pracy w okolicach 100 km/h na ekranie danych szczegółowych .....	65
Rys. 7.3. Pomiar przy ustabilizowanej prędkości na ekranie uproszczonym .....	65
Rys. 7.4 Spowolniony kąt zapłonu .....	66
Rys. 7.5. Gwałtowne wywołanie wyższych obrotów poprzez zmianę biegu o jeden w dół .....	67
Rys. 7.6. Wyniki wyświetlone na stronie uproszczonej przy zrzuceniu biegu w dół .....	68
Rys. 7.7 Widok zapisanych danych w bazie danych SQLite z przeprowadzonych badań .....	69
Rys. 7.8. Filtr daty oraz godziny zastosowany na bazie SQLite .....	70
Rys. 7.9. Zapytanie szukające obciążnika .....	71
Rys. 7.10. Znalezienie momentu największego obciążenia silnika .....	71
Rys. 7.11. Wyszukanie momentów ujemnego kąta zapłonu .....	72
Rys. 7.12. Odnalezienie momentów większych obrotów przy prędkości niższej niż 60 km/h .....	72

## Spis Listingów

Listing 6.1. Deklaracja struktury przesyłu i adresacja połączeń w ESP32 .....	40
Listing 6.2. Sekcja kodu ESP32 zawierająca elementy inicjalizacji .....	43
Listing 6.3. Sekcja kodu ESP32 zawierająca główną pętlę wykonawczą .....	44
Listing 6.4. Deklaracja przełącznika wybierającego typ danych otrzymanych z ECU .....	48
Listing 6.5. Sekcja odbiornika ESP8266 przesyłającego dane do Nextion HMI .....	50
Listing 6.6. Kod zawarty w ESP8266 połączonego z Raspberry PI .....	54
Listing 6.7. Plik wykonawczy powłoki .....	60
Listing 6.8. Kod programu obsługującego bazę danych SQLite w języku Python .....	62



## Spis Załączników

- [1] <https://www.fixya.com/uploads/images/obd2%20diagnostic%20port-4lfvybuxsukahru2dxa4xzsn-5-0.jpg> (Stan z 03.08.2022)
- [2] <https://cdn3.botland.com.pl/88800/raspberry-pi-pico-rp2040-arm-cortex-m0.jpg> (Stan z 03.08.2022)
- [3] <https://botland.com.pl/moduly-i-zestawy-raspberry-pi-zero/5215-raspberry-pi-zero-v13-512mb-ram-5904422368715.html> (Stan z 03.08.2022)
- [4] <https://cdn1.botland.com.pl/68232/raspberry-pi-4-model-b-Wi-Fi-dualband-Bluetooth-4gb-ram-15ghz.jpg> (Stan z 03.08.2022)
- [5] <https://elty.pl/userdata/public/gfx/b6c86cc2e364302e6a9c4e014b3ae111.jpg> (Stan z 03.08.2022)