Politechnika Śląska

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Oprogramowanie Systemów Pomiarowych

Raport końcowy

Autorzy:

Chodura Mateusz

Czerwieński David

Kandzia Kamil

Szromek Dawid

Rok III, semestr VI, grupa III TI

Gliwice, lipiec 2018

Spis treści

1. Cel i zakres

Głównym celem projektu było napisanie oprogramowania w Labview, które pozwoli na komunikację z pojazdem jak np. samochód i odczyt podstawowych parametrów pracy silnika. Aplikacja miała służyć głównie do diagnostyki samochodowej, do odczytu błędów silnika, a także pomóc w lokalizacji usterki pojazdu. Oprócz tego dodatkowym celem było nauczenie się czegoś nt. programowania w Labview oraz akwizycji danych.

Pomimo już dostępnych na rynku aplikacji tego typu, wybraliśmy ten temat, ponieważ chcieliśmy stworzyć aplikację darmową, prostą w użyciu, a przede wszystkim taką, która będzie posiadała dokładnie takie funkcje, jakie potrzebujemy i będzie bardzo łatwa w rozbudowie w przyszłości.

Jednymi z większych problemów, które wynikły już na początku była kwestia testowania naszej aplikacji, jakby nie patrzeć aplikacja służyła do komunikacji z samochodem, także, aby testować to co już udało nam się napisać musieliśmy godzinami przesiadywać w samochodzie. Aby rozwiązać ten problem zdecydowaliśmy o napisaniu aplikacji symulującej nasz samochód, dzięki czemu mogliśmy rozbudowywać aplikację w bardziej wygodnym i przystosowanym do tego miejscu.

2. Symulator samochodu

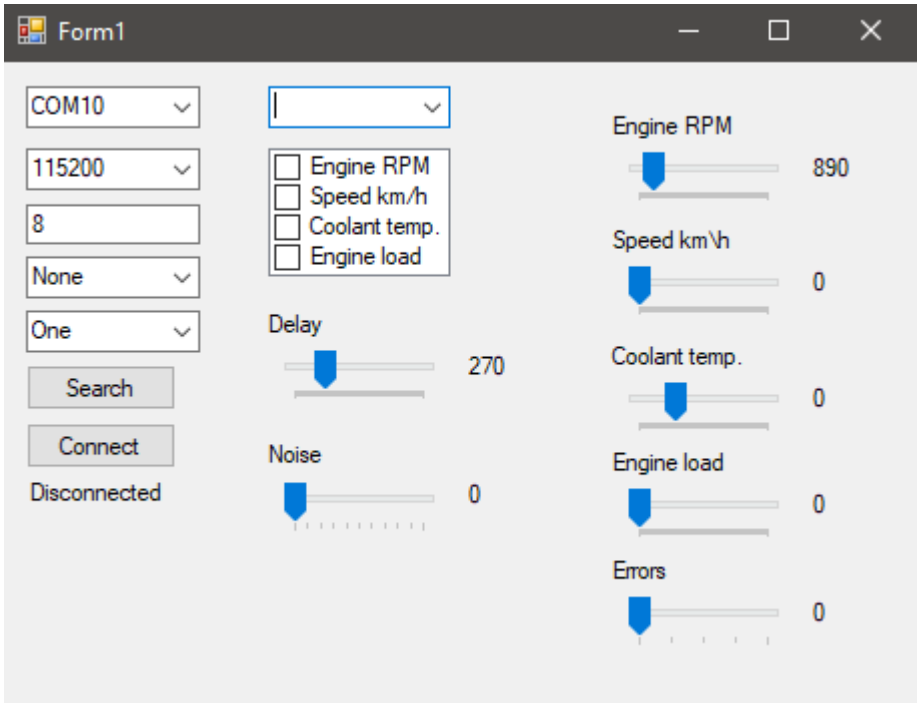
2.1 Technologia

Symulator został napisany w języku C# w środowisku Visual Studio jakoaplikacja okienkowa. Pozwoliło to na szybkie stworzenie GUI dla użytkownika, oraz na szybkie pisanie programu. Z symulatorem możemy się połączyć poprzez port COM.

Stworzenie symulatora poprzedziło długie testowanie połączenia z samochodem, zapisy log'ów z wymiany danych, testowanie różnych ustawień portów oraz czasów transmisji i czasów, jakie należy czekać na otrzymanie odpowiedzi zwrotnej.

2.2 Funkcjonalność

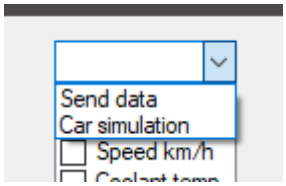
Okno główne programu:



a) Ustawienia komunikacji

Okno głównym po lewej stronie pozwala nam na wybranie ustawień portu COM, domyślnie wpisywane są tam ustawienia, jakich używamy do naszego zastosowania, jednak możliwość zmiany tych ustawień sprawia, że aplikacja jest bardziej uniwersalna.

b) Wybór trybu pracy:



Ta lista rozwijana pozwala na wybór trybu pracy:

Send data – w tym trybie program wysyła dane ciągiem wg. ustawień (wysyłane są dane zaznaczone w checkbox'u pod listą)

Car simulation–ten tryb symuluje dokładne zachowane auta, tzn., że symulator odpowiada tylko i wyłącznie z na zapytania, które przychodzą do niego poprzez port COM.

c) Ustawienia

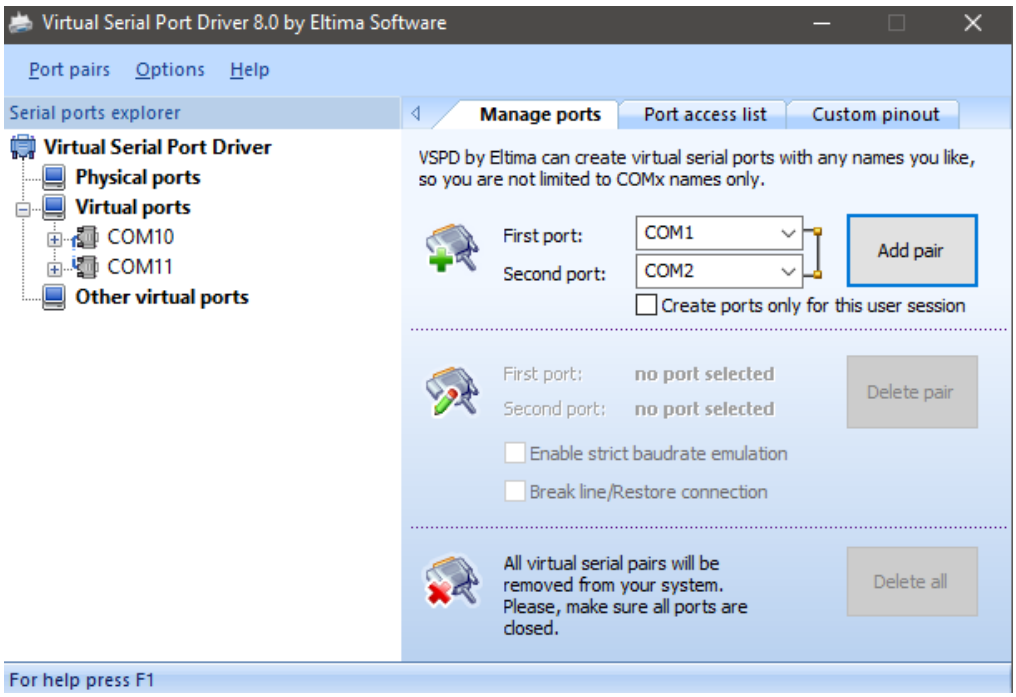
- Delay - dane wysyłane są cookreślony czas (ms),
- Noise- służy do generowania sztucznych zakłóceń w transmisji danych, (0 – brak),
- Engine RPM – (0-8000), ustawia wartość obrotów silnika,
- Speed km/h – (0-220), ustawia wartość prędkości pojazdu,
- Coolant temp. – (-40 - 100), ustawia wartość płynu chłodniczego silnika,
- Engine load – (0 - 100), ustawia wartość obciążenia silnika,
- Errors – (0 - 4), ustawia ilość błędów.

3. Rozwiązania techniczne

3.1 Wirtualne porty COM

Aby w pełni wykorzystać funkcjonalność naszego symulatora musieliśmy jakoś połączyć nasze obie aplikacje. W tym celu użyliśmy programu firmy Eltima Software o nazwie Virtual Serial Port Driver 8.0. Pozwala on na stworzenie wirtualnych portów COM, które następnie możemy łączyć w pary zarówno z innym portem wirtualnym, jak i z portem fizycznym.

Dzięki takim możliwościom można np. podglądać komunikacje na wybranym porcie, lub tak jak w naszym przypadku, stworzyć parę portów wirtualnych, które podłączyliśmy do naszych aplikacji, aby te mogły się między sobą komunikować.



3.2 Interfejs OBDII

W celu komunikacji z samochodem zakupiliśmy interfejs OBDII o nazwie iCar2.

Interfejs ten obsługuję popularny układ ELM327, oraz posiada komunikację przez bluetooth, dzięki czemu nie potrzeba specjalnych sterowników na komputery, a dodatkowo można go użyć także z telefonem. Sam interfejs nie jest z górnej półki jednak działa bardzo stabilnie i nie ma żadnych kłopotów, aby odczytać dane, które potrzebujemy. Zaletą takiej obudowy jest możliwość zostawienia go na stałe w samochodzie, ponieważ układ wyłącza się sam po 30 min nie używania.

W tym miejscu zaznaczyć należy także, że nasze oprogramowanie dla Labview zostało napisane tak, aby współpracowało z każdym interfejsem wyposażonym, lub symulującym układ ELM327, które można podłączyć do portu COM komputera.

4. Aplikacja Labview

4.1 Opis ogólny

Nasza aplikacja ma za zadanie komunikować się z interfejsem OBDII stosowanym zarówno w samochodach osobowych jak i w ciężarowych. Jako, że interfejs ten w najbardziej powszechnej wersji komunikuje się bezpośrednio ze sterownikiem silnika, możemy za jego pomocą odczytać parametry jego pracy, jak np. prędkość pojazdu, temperatura cieczy chłodzącej, obroty silnika i wiele innych.

Gniazdo OBDII posiada również bardzo często wyprowadzone linie interfejsu CAN, dlatego też projekt można w bardzo prosty sposób rozbudować o komunikację z całym pojazdem za pomocą tego właśnie interfejsu.

Sama aplikacja została napisana zgodnie z architekturą producent-konsument oraz pozwala użytkownikowi na komunikację z samochodem na dwa sposoby:

- pierwszy pozwala ręcznie wpisywać komendy i odczytywać komunikat zwrotny w postaci wartości hexadecymalnych, czyli w postaci kodów AT zgodnych ze standardem interfejsu ELM327,
- drugi pozwala na ciągłe odczytywanie 5 parametrów (obroty silnika, prędkość, temperatura cieczy chłodzącej, obciążenie silnika oraz błędy) oraz ich bieżące wyświetlanie na wykresach.

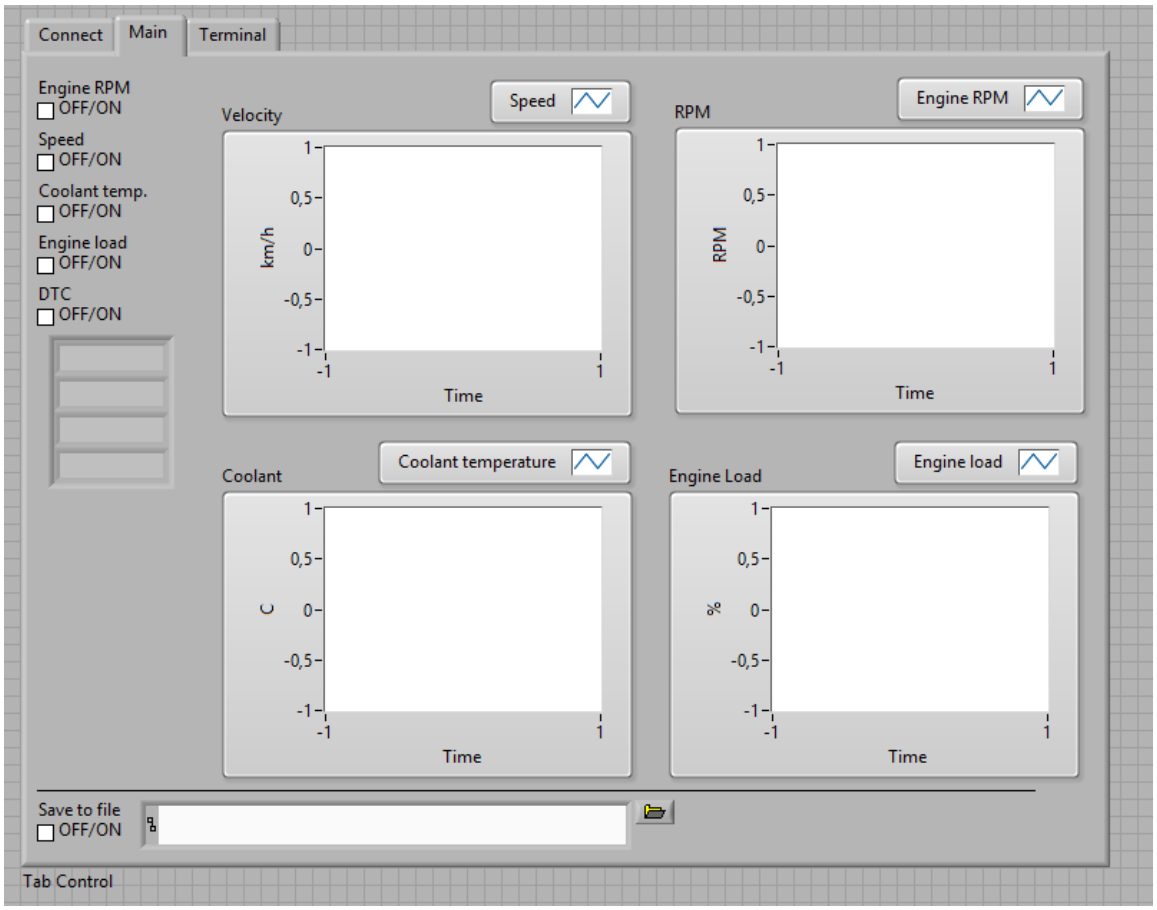
Aplikacja została podzielona na 3 oddzielne karty, a każda z nich pełni inną funkcję, dzięki temu program stał się bardziej przejrzysty oraz intuicyjny dla użytkownika, który spotyka się z nim pierwszy raz. Nazwy poszczególnych kart pochodzą od głównego zadania, jakie pełnią.

4.2 Opis kart aplikacji

a) Main

Jest to główna karta naszej aplikacji, w której przeciętny użytkownik będzie spędzał najwięcej czasu. Znajdują się w niej pola wyboru parametru (określają, które parametry samochodu będą monitorowane), wykresy, na których są wyświetlane parametry oraz pole zapisu pliku, gdzie możemy określić jego ścieżkę i włączyć zapis do pliku.

Wybór, które parametry chcemy aktualnie czytać z pojazdu jest ważny ze względu na ograniczone możliwości sprzętowe zarówno sterownika silnika jak i zakupionego interfejsu. Ponieważ jeden odczyt dowolnej wartości trwa w naszym przypadku ok 260 ms, oznacza to, że szybkość odczytu to ok 4 odczyty/sekundę. Nie jest to wynik oszałamiający, lecz nie jesteśmy w stanie osiągnąć nic więcej z aktualnym sprzętem. Dlatego też, gdy np. wybierzemy do odczytu wszystkie parametry, aktualizacja każdego parametru będzie przebiegała, co ok 1.3 s. Natomiast, gdy wybierzemy np. tylko obroty silnika, wartość ta będzie aktualizowana, co ok. 260 ms.

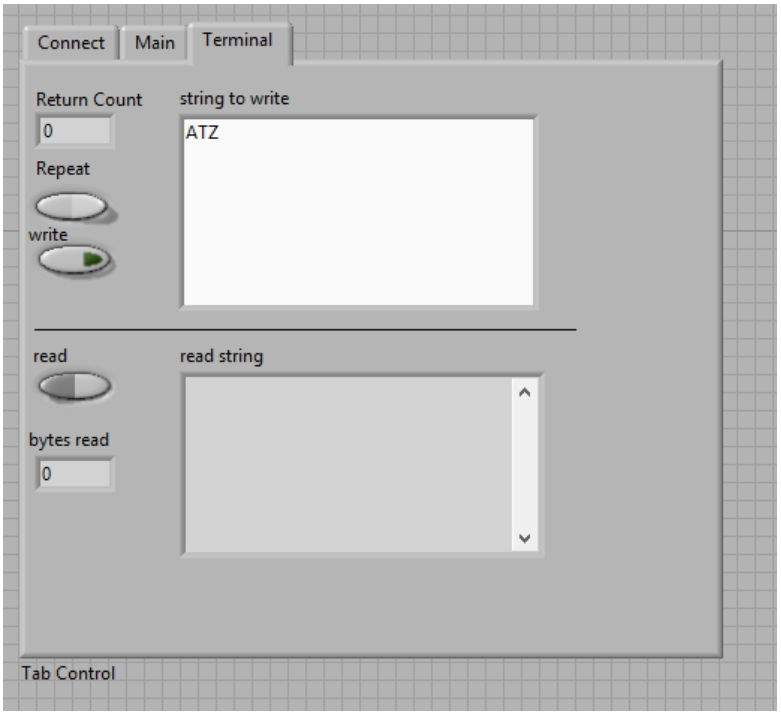


b) Terminal

Karta terminalu pozwala zaawansowanemu użytkownikowi wprowadzać komendy, które zostaną wysłane do samochodu, jako odpowiedź, użytkownik dostaje ciąg znaków z zapisanymi wartościami hexadecymalnymi, są to po prostu informacje oraz dane zapisane zgodnie ze standardem AT interfejsów ELM.

Dzięki temu panelowi, użytkownik jest w stanie odczytać ze swojego samochodu parametry i informacje, które nie są zaimplementowane w karcie głównej. Pozwala ona nie tylko na komunikację ze sterownikiem silnika, ale także na komunikację za pomocą linii CAN z całym samochodem.

Niestety, aby wykorzystać pełnię możliwości tej karty trzeba być obeznanym ze standardem komunikacji OBDII oraz w razie potrzeby należy mieć dostęp do często ukrywanych przez producentów samochodów poleceń CAN.

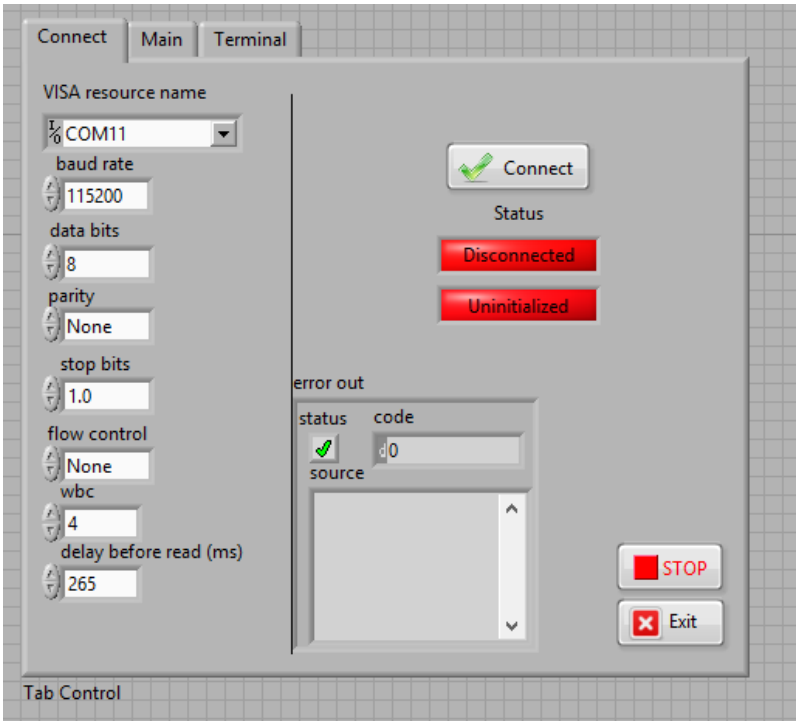


c) Connect

Domyślna karta naszej aplikacji gdyż przez nią użytkownik nawiązuje połączenie z autem. Aplikacja była tworzona z myślą o uniwersalności, dlatego też w tym panelu możemy ustawiać takie parametry jak:

- Port COM, (zależny od urządzenia i komputera, pozwala skorzystać zarówno z interfejsów przewodowych jak i bezprzewodowych)
- Szybkość transmisji, (parametr zależny bezpośrednio od zakupionego interfejsu)
- Bity danych, (jw.)
- Parzystość, (jw.)
- Bity stopu, (jw.)
- Kontrole przepływu, (jw.)
- Czas w ms pomiędzy kolejnymi bitami, (parametr zależny od samochodu i częściowo od interfejsu)
- Odstęp między wysyłaniem zapytań (zależna od auta oraz modułu komunikacyjnego, zapobiega gubieniu informacji).

Okno Error powiadamia nas o błędach w komunikacji, natomiast dwie lampki z napisem informują nas o stanie połączenia, np., że program połączył się z interfejsem, lecz nie zainicjalizował jeszcze połączenia z pojazdem.



4.3 Opis SubVI

a) Prepare_Error_Array(SubVI).vi

Context Help

Prepare_Error_Array(SubVI).vi

SubVI that update string array containing error list that i displayed on main window.

b) Array_extraction(SubVI).vi

Context Help

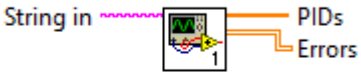
Array_extraction(SubVI).vi

SubVI that extracts values from **Array** into three double values.
Info - Contain information about what is in array(Velocity,RPM,Error etc.).
Value - Contain value of read parameter.
Iteration - Contain number of iteration in which data was read.

c) convert(SubVI).vi

Context Help

convert(SubVI).vi



SubVI that convert string of hexadecimal values into one of two tables depending of its content.
PIDs contain number that specify value type (Velocity, RPM,Engine Load etc.) and its value.
Errors table contain three arrors at a time, first row inform that it contains error and identifier of module in which that error occured , second row contain value of error and last row is always 0.

d) data_array_creation(SubVI).vi

Context Help

data_array_creation(SubVI).vi

SubVI that replace **Index** element of array with new **Value**

e) Empty_queue(SubVI).vi

Context Help

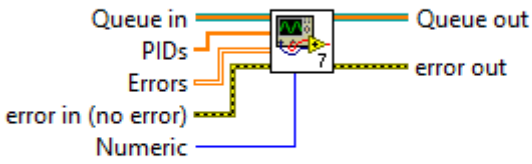
Empty_queue(SubVI).vi

SubVI that prevent queue from being closed when not empty

f) Enqueue_data(SubVI).vi

Context Help

Enqueue_data(SubVI).vi

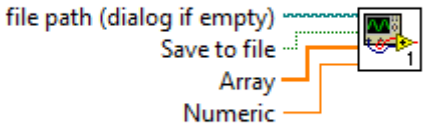


SubVI that enque data to queue.
It can only take one array in at a time, either PIDs or Errors.
PIDs table is extended of information in which loop that table was obtained then its enqueued while Errors is divided into 3 subarrays that are enqueued to queue.

g) File_save(SubVI).vi

Context Help

File_save(SubVI).vi

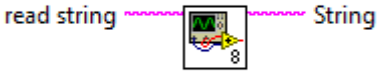


SubVi that save data from **Array** into file when **Save to File** is true and **Numeric** is equal to 4

h) String_prepare(SubVI).vi

Context Help

string_prepare(SubVI).vi

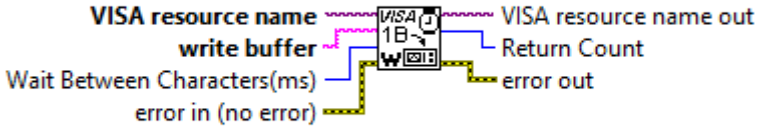


SubVI to initially prepare string read from car.

i) 1-at-a-time(SubVI).vi

Context Help

1-at-a-time(SubVI).vi

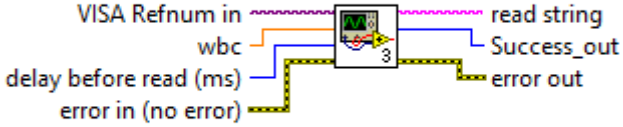


For Instruments that require characters to be written one at a time. It breaks down the string into individual bytes and transfers one at a time.

j) Serial_Init(SubVI).vi

Context Help

Serial_Init(SubVI).vi

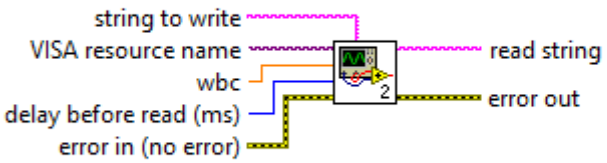


SubVI that initialize and prepare whole communication with a car. VI gets COM port from **VISA refnum in**. Everythink works thine if **Success_out** is equal to numeric 9.

k) One_Command(SubVI).vi

Context Help

One_Command(SubVI).vi



This SubVI is used to send just one command saved in **string to write** to the car and to return received messeage with **read string**

5. Napotkane problemy

5.1 Komunikacja z pojazdem

Pierwszym problemem, na jaki się natknęliśmy na samym początku była komunikacja z pojazdem. Aby poprawnie odczytywać dane oraz aby osiągnąć jak najlepsze rezultaty musieliśmy metodą prób i błędów dobierać ustawienia komunikacyjne do zakupionego przez nas interfejsu. Oprócz tego problemem okazał się sam sterownik silnika, którego czas odpowiedzi okazał się bardzo długi jak na dzisiejsze standardy. Większość prób skierowanych na zmniejszenie tego czasu skutkowała gubieniem przez interfejs informacji oraz danych. Wiemy jednak, że czas ten może w znaczący sposób różnić się w zależności od użytego samochodu, dlatego zostawiliśmy w opcjach połączenia możliwość zmiany domyślnej wartości.

5.2 Konieczność sprawdzania aplikacji na obiekcie rzeczywistym

Powiązany z poprzednim problemem. Jest to pomniejszy problem, który jednak potrafił bardzo utrudnić życie.

Jako, że przesiadywanie cały dzień w samochodzie i rozładowywanie akumulatora nie było najlepszym rozwiązaniem, postanowiliśmy napisać symulator samochodu, który zachowuje się identycznie jak jego fizyczny odpowiednik i łączy się z naszą aplikacją poprzez wirtualny port COM.

5.3 Zapis do pliku

Jednym z problemów napotkanych podczas tworzenia naszej aplikacji był zapis do pliku.

Na jego potrzeby musieliśmy stworzyć dodatkowy rejestr przesuwny, do którego są zapisywane poszczególne wartości w pętli a następnie po odczytaniu ostatniej wartości są one konwertowane do tablicy typu string, która jest zapisywana bezpośrednio do pliku.

6. Podsumowanie

Po wielu godzinach spędzonych nad projektem, obie aplikacje udało się doprowadzić do końca. Komunikacja z pojazdem i z symulatorem działa bez zarzutu, program posiada wbudowane kilka parametrów do odczytu, a dzięki modułowej budowie aplikacji, bardzo łatwo będzie ją w przyszłości rozbudować.

Oprócz standardowych parametrów, aplikacja posiada również terminal do manualnego wpisywania komend. Pozwala on na bardziej zaawansowane użycie komunikacji z pojazdem.

Program został wyposażony w ciągły zapis do pliku, dzięki czemu nie musimy na bieżąco monitorować parametrów, ale możemy zrobić jazdę testową i dopiero później sprawdzić, w jakich okolicznościach wystąpił błąd.

Podczas pracy nad projektem nauczyliśmy się bardzo wielu ważnych umiejętności związanych z programowaniem w LabView, zbieraniem i zapisywaniem danych, programowaniem w C#, a także związanych z połączeniami pomiędzy urządzenie-program oram program-program.