

**Programowanie w środowisku Windows**

**Oprogramowanie modemu**

**EVALKIT-ST7580**

Mateusz Balcer

Informatyka Stosowana, grupa 4

Spis treści

[1. Wstęp 3](#_Toc30511853)

[2. Interfejs aplikacji 3](#_Toc30511854)

[3. Struktura projektu 4](#_Toc30511855)

[4. Opis najważniejszych funkcji 5](#_Toc30511856)

[4.1. Otwieranie / zamykanie portu 5](#_Toc30511857)

[4.2. Odbieranie wiadomości 6](#_Toc30511858)

[4.3. Wysyłanie wiadomości 8](#_Toc30511859)

[4.4. Reset modemu 9](#_Toc30511860)

[4.5. Protokół komunikacyjny 10](#_Toc30511861)

[4.5.1. Przetwarzanie ramki 10](#_Toc30511862)

[4.5.2. Opakowywanie danych w ramkę 11](#_Toc30511863)

[4.5.3. Pobranie bajtów z ramki 12](#_Toc30511864)

[4.6. Przełączanie trybów PHY/DL 12](#_Toc30511865)

[4.7. Ustawienie trybu modulacji 13](#_Toc30511866)

[4.8. Konfiguracja 14](#_Toc30511867)

[4.8.1. Ładowanie konfiguracji 14](#_Toc30511868)

[4.8.2. Zapisywanie konfiguracji 15](#_Toc30511869)

[5. Przykłady działania programu podczas pracy z modemem 16](#_Toc30511870)

[5.1. Reset modemu 16](#_Toc30511871)

[5.2. Zmiana trybu na PHY 17](#_Toc30511872)

[5.3. Wysłanie wiadomości 17](#_Toc30511873)

[5.4. Odbiór wiadomości 18](#_Toc30511874)

[5.5. Ładowanie konfiguracji 19](#_Toc30511875)

[6. Wnioski 20](#_Toc30511876)

1. Wstęp

Celem projektu było zaimplementowanie programu do obsługi modemu EVALKIT – ST7580. Oprogramowanie powinno posiadać podstawowe funkcje takie jak:

* Łączenie się z portem na którym jest połączony modem
* Reset modemu
* Wysyłanie / Odbiór wiadomości
* Wybór rodzaju modulacji
* Przełączanie trybów PHY/DL

Program powinien posiadać przyjazny interfejs użytkownika do obsługi wymienionych funkcji. Dodatkowo należało zaimplementować protokół komunikacyjny Host-Modem podczas wysyłania / odbioru wiadomości.

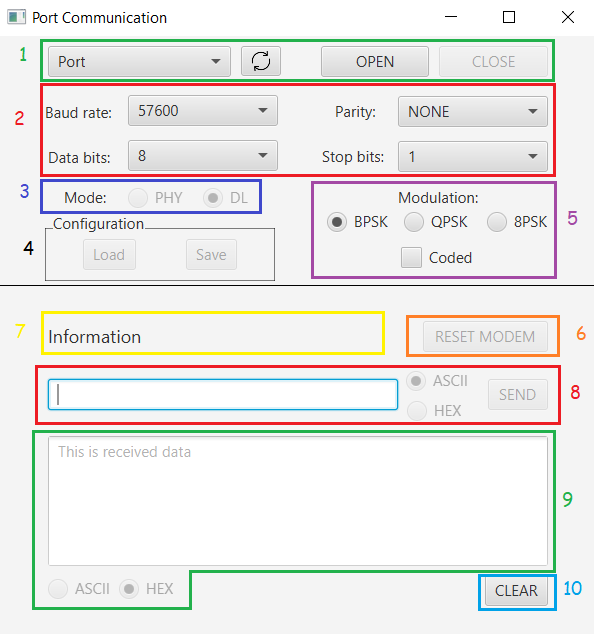
W dokumentacji przedstawiam najważniejsze fragmenty kodu. Kod całego programu znajduje się na: <https://github.com/mbalcer/PSW-project>

1. Interfejs aplikacji

Aplikacja złożona jest z jednego okna widoku na którym znajdują się wszystkie niezbędne elementy. Okno podzielone jest na kilka części:

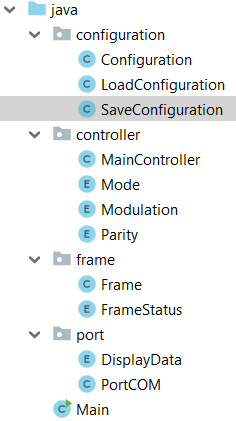
1. Otwieranie / zamykanie portów (autodetekcja dostępnych portów) + odświeżanie listy portów
2. Konfiguracja parametrów połączenia (prędkość transmisji, bity danych, bity stopu, parzystość)
3. Zmiana trybu na PHY/DL
4. Zapisywanie / Ładowanie konfiguracji
5. Ustawienie modulacji
6. Przycisk do resetowanie modemu
7. Etykieta wyświetlające informacje dla użytkownika
8. Wysyłanie wiadomości (możliwość wysyłania jako ASCII lub jako HEX)
9. Element wyświetlający odebrane wiadomości (możliwość zmiany wyświetlanych danych jako ASCII lub HEX)
10. Przycisk do czyszczenia okien wysyłania i odbioru

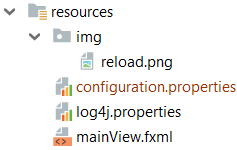
Wygląd aplikacji oznaczony poszczególnymi elementami:



1. Struktura projektu

Struktura projektu podzielona jest na kilka klas. Główną klasą jest MainController, który odpowiada za komunikację z GUI. PortCOM odpowiada za port komunikacyjny. Implementacja klasy Frame jest odpowiedzialna za protokół komunikacyjny. Klasy Configuration przedstawiają zapisywanie / ładowanie konfiguracji. Dodatkowo w projekcie jest zaimplementowane kilka Enumów ułatwiających pisanie kodu. Natomiast w katalogu resources znajduje się plik fxml, który odpowiada za stworzenie okna aplikacja. Okno zostało zaprojektowane w programie Scene Builder.

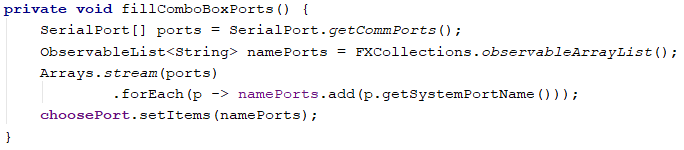


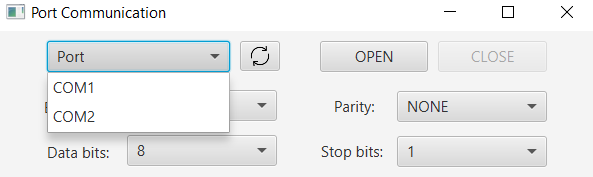


1. Opis najważniejszych funkcji
   1. Otwieranie / zamykanie portu

Podczas uruchamiania aplikacji porty, który są dostępne do połączenia wyświetlane są w polu do wyboru. Gdy użytkownik chce zaaktualizować listę dostępnych portów może użyć przyciska „Refresh”.

Kod odpowiedzialny za autodetekcję portów:

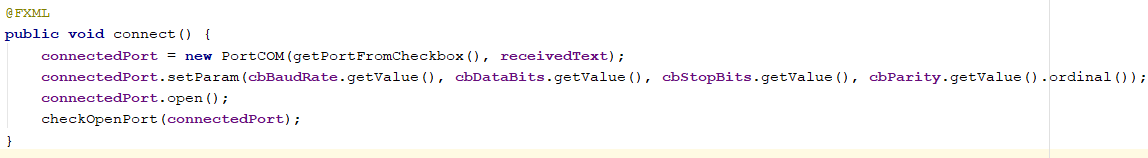


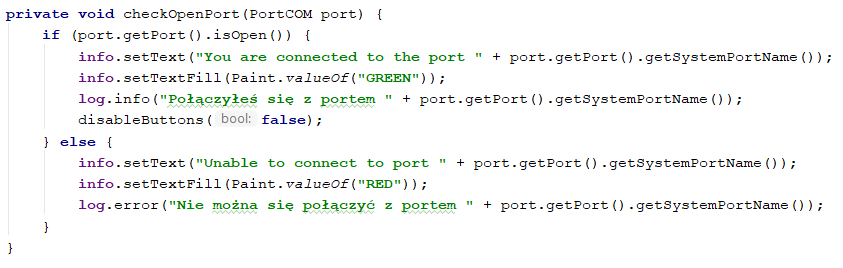


Po wybraniu z listy portu użytkownik może wybrać konfigurację połączenia z modemem. Program podpowiada domyślną konfigurację dla modemu tzn:

* Baud rate: 57600
* Partity: none
* Data bits: 8
* Stop bits: 1

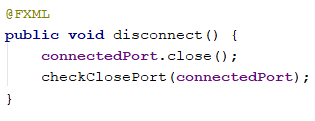
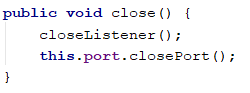
Kolejnym krokiem jest otworzenie portu. Tworzona jest instancja klasy PortCOM, ustawiane parametry połączenia i otwierany jest port. Podczas otwierania portu aktywowany jest również nasłuchiwacz, który przechwytuje odebrane wiadomości. Opis funkcji aktywującej nasłuchiwacz jest w pkt. **4.2** Następnie sprawdzane jest czy port został otwarty za pomocą specjalnej funkcji. Jeśli tak się stało to wyświetlany jest odpowiedni komunikat i odblokowane zostają elementy tj. wysyłanie wiadomości czy zmiana trybu modulacji.





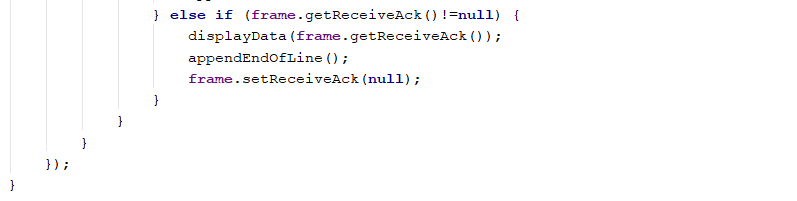
Po otwarciu odblokowywany jest również przycisk do zamknięcia portu. Po jego naciśnięciu usuwany jest nasłuchiwacz do odbioru wiadomości i zamykany jest port.

Klasa MainController: Klasa PortCOM:

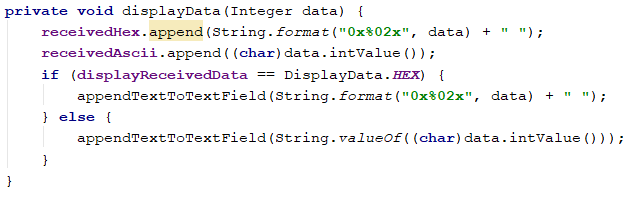


* 1. Odbieranie wiadomości

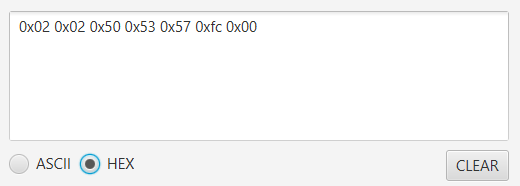
Odbieranie komunikatów od modemu jest realizowane przez włączenie nasłuchiwacza w klasie PortCOM. Komunikaty są odbierane po jednym bajcie za pomocą kolejki. Bajt który został odebrany jest przekazywany do metody processFrame w klasie Frame gdzie jest analizowany (opis tej funkcjonalności w pkt ). Następnie w zależności od tego czy przekazana ramka jest poprawna to wyświetlane są bajty ramki w polu tekstowym do odbioru wiadomości (oznaczonym nr 9 w pkt **2**). Jest tutaj również kod odpowiedzialny za wysłanie ACK jeśli ramka zaczyna się od bajtu 0x02.

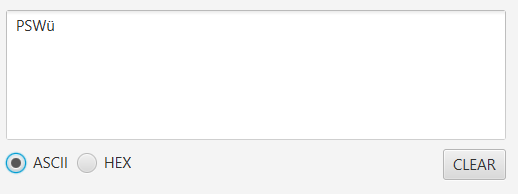
 

Dane są odpowiednio konwertowane w zależności od zaznaczonego trybu wyświetlania. Dane które zostały już wyświetlona można zmieniać na ASCII <-> HEX.



Odebrana ta sama wiadomość w HEX i ASCII

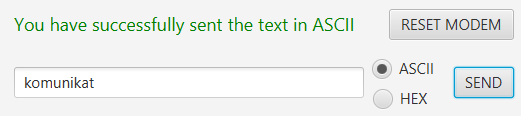




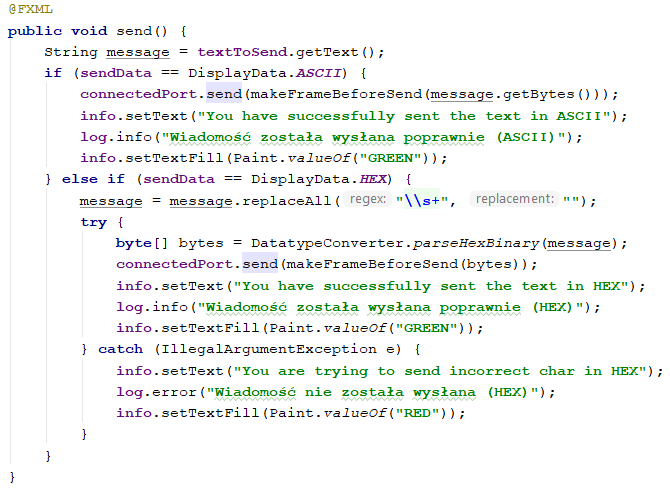
* 1. Wysyłanie wiadomości

Aby wysłać komunikat do modemu należy wpisać dane w polu tekstowym (w ASCII lub HEX) a następnie kliknąć przycisk. Dane opakowywane są w ramkę (dokładny opis w jaki sposób są opakowywane w pkt **4.5.2**). Przed wysłaniem ustawiane są linie RTS i DTR, które po wysłaniu komunikatu są czyszczone.

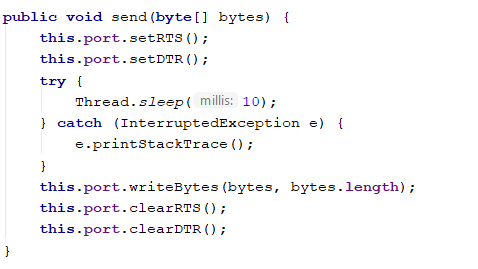
Wysłanie wiadomości:



Klasa MainController:

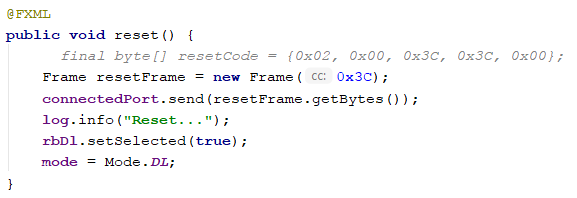


Klasa PortCom:



* 1. Reset modemu

Reset jest wywoływany po kliknięciu w przycisk oznaczony nr 6 w pkt **2**. Kod resetu (0x3C) opakowywany jest w ramkę (opakowanie w ramkę opisane w pkt **4.5.2**). Bajty ramki są pobierane i wysyłane przez funkcję send (z pkt **4.3**). Po resecie ustawiany jest również tryb DL (jeśli przed resetem był ustawiony tryb PHY).



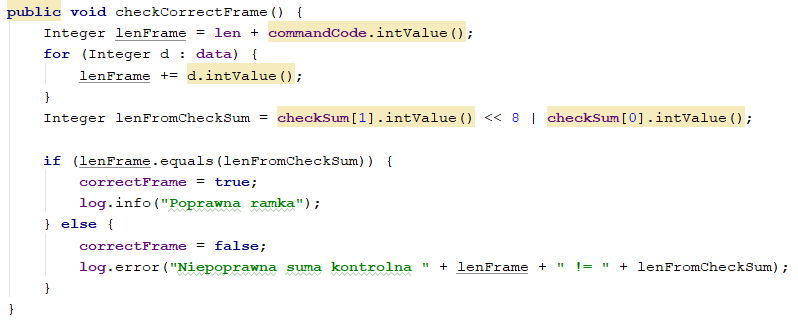
* 1. Protokół komunikacyjny
     1. Przetwarzanie ramki

Podczas odbioru komunikaty bajty przesyłane są do metody processFrame. W ciele metody sprawdzany jest aktualny stan ramki, a następnie w zależności od stanu sprawdzane są kolejne warunki np. jeśli ramka posiada stan BEGIN to sprawdzane jest czy odebrany bajt jest 0x02 lub 0x03. Jeśli bajt jest 0x02 to jednocześnie po odbiorze całej ramki zostaje wysłane ACK. Stan ramki zmieniany jest na LEN i ramka jest dalej analizowana.

Część kodu metody processFrame w klasie Frame:

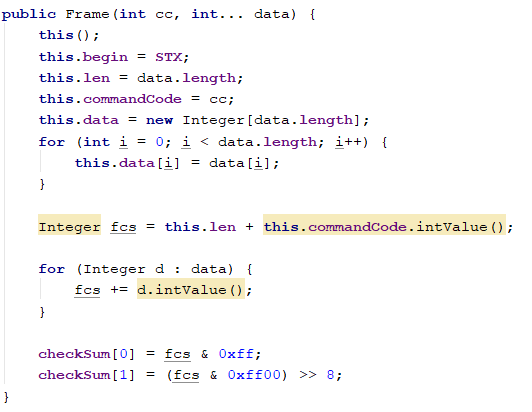


Jeśli ramka dojdzie do ostatniego stanu tzn. SECOND\_FCS to stan ramki odbierającej jest resetowany. Kolejnym elementem jest sprawdzanie sumy kontrolnej ramki. Odpowiada za to metoda checkCorrectFrame. W ciele metody dodawane są wszystkie bajty ramki (długość, command code i dane) a następnie jest to sprawdzane z sumą kontrolną wpisaną w FIRST\_FCS i SECOND\_FCS.



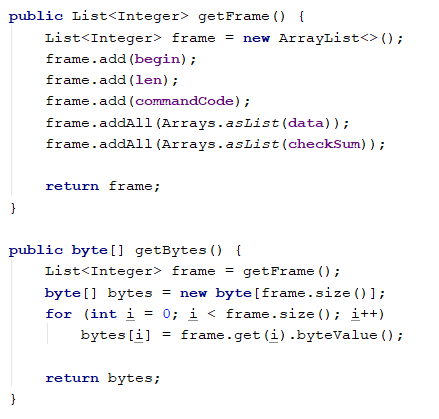
* + 1. Opakowywanie danych w ramkę

Realizowane jest poprzez konstruktor klasy Frame. Przyjmuje on jako argumenty command code oraz opcjonalnie dane. W ciele konstruktora ustawiane są kolejne bajty ramki tzn. początek, długość, command code, dane, a na końcu liczona jest suma kontrolna poprzez dodanie wszystkich bajtów. Sumę kontrolną należy ustawić na dwóch bajtach, więc konieczna jest operacja AND z odpowiednią maską i przesunięcie bitowe. Tak przygotowaną ramkę można pobrać przez metodę opisaną w pkt 4.5.3



* + 1. Pobranie bajtów z ramki

Po opakowanie danych aby wysłać ramkę konieczne jest pobranie wszystkich bajtów z klasy Frame poprzez metodę getBytes(). Metoda ta pobiera listę bajtów ramki a następnie jest ona konwertowana na tablicę. Konieczne jest przekonwertowanie listy na tablicę ponieważ jest ona potrzebna do wysłania danych (nie można wysłać listy).

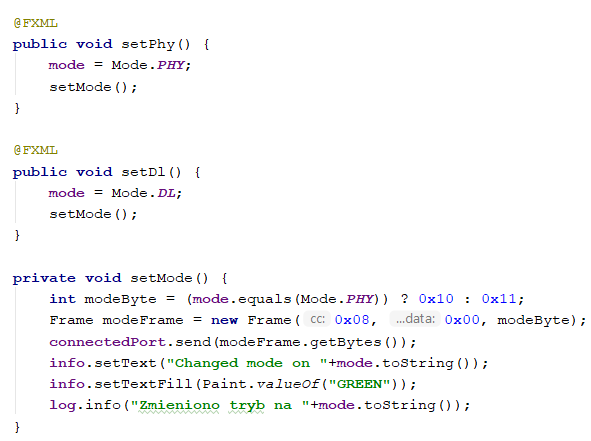


* 1. Przełączanie trybów PHY/DL

Aby zmienić tryb na PHY lub DL należy wysłać ramkę z kodem komendy 0x08 oraz odpowiednimi danymi

* Dla PHY: 0x00 0x10
* Dla DL: 0x00 0x11

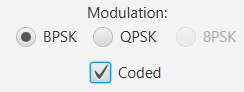
Skrypt odpowiedzialny za zamianę trybu wysyła przygotowaną ramkę do modemu.



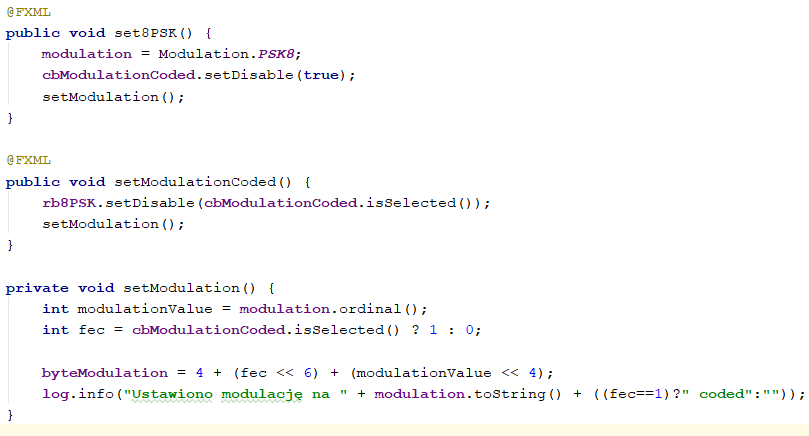
* 1. Ustawienie trybu modulacji

Aby ustawić modulację na konkretny tryb trzeba dodać bajt kontrolny przed wysłanymi danymi, a jako kod komendy dać 0x24 (PHY) 0x50 (DL). W moim programie modulację można zmienić na:

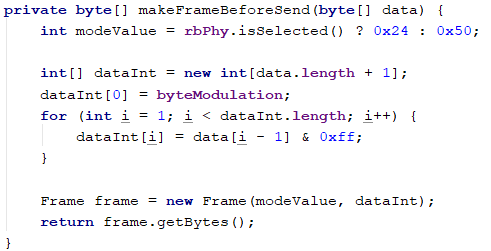
* BPSK
* QPSK
* 8PSK
* BPSK coded
* QPSK coded



W kontrolerze głównym znajduje się zmienna bajt modulacji, która jest zmieniana za każdym razem, kiedy użytkownik zmienia modulację.

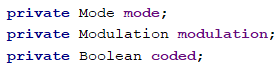


Podczas wysłania danych ten sam bajt jest wstawiany jako pierwszy bajt danych a następnie przygotowywana jest ramka do wysłania.



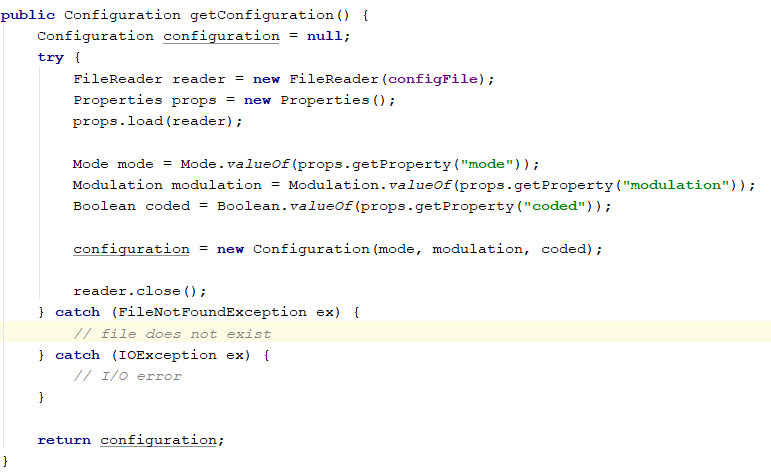
* 1. Konfiguracja

Użytkownik ma możliwość również ładowania / zapisywania konfiguracji połączenia tzn.: modulacji i trybu PHY/DL. W GUI służą do tego dwa przyciski „Load” i „Save”, które po kliknięciu każą wybrać lokalizację pliku do ładowania / zapisania. Do łatwiejszego przechowywania danych konfiguracji została stworzona klasa modelu Configuration, która posiada pola:

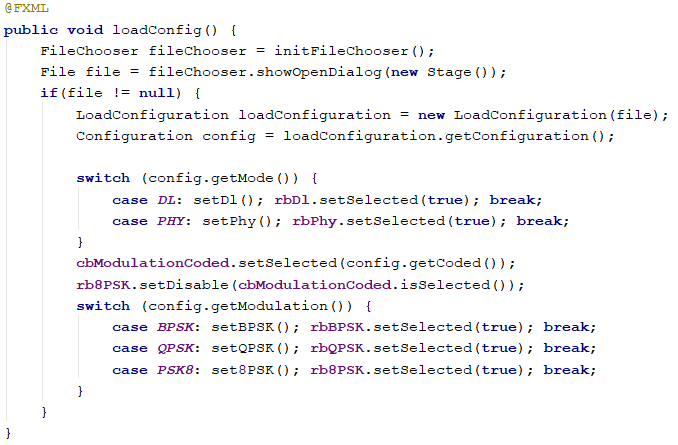


* + 1. Ładowanie konfiguracji

Do ładowania konfiguracji została utworzona klasa LoadConfiguration posiadająca jedną metodę. Do klasy przekazywany jest plik konfiguracji i odczytywane są z niego potrzebne wartości. Metoda zwraca obiekt konfiguracji.

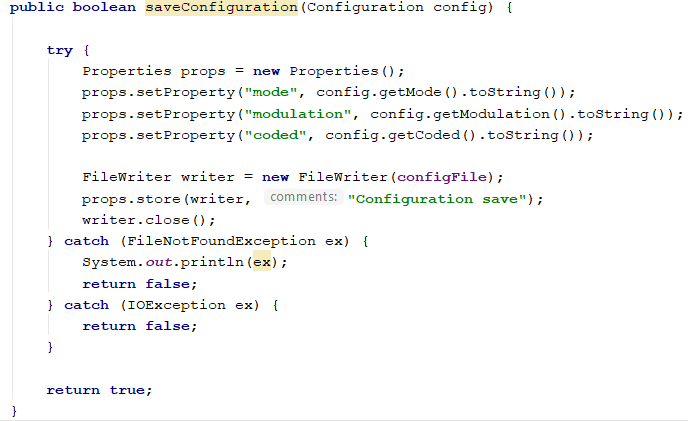


W klasie MainController po kliknięciu przycisku otwierane jest okno dialogowe do wybrania pliku. Po wybraniu pliku odczytywana jest konfiguracja a następnie ustawiane odpowiednie wartości w GUI oraz przy połączeniu z portem.

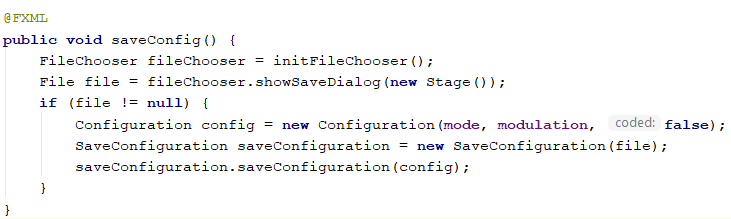


* + 1. Zapisywanie konfiguracji

Analogicznie jak w przypadku ładowania konfiguracji do zapisu została stworzona klasa SaveConfiguration, posiadająca metodę save. Metoda przyjmuje obiekt konfiguracji a w ciele metody zapisywana jest do pliku wskazanego przy wyborze.

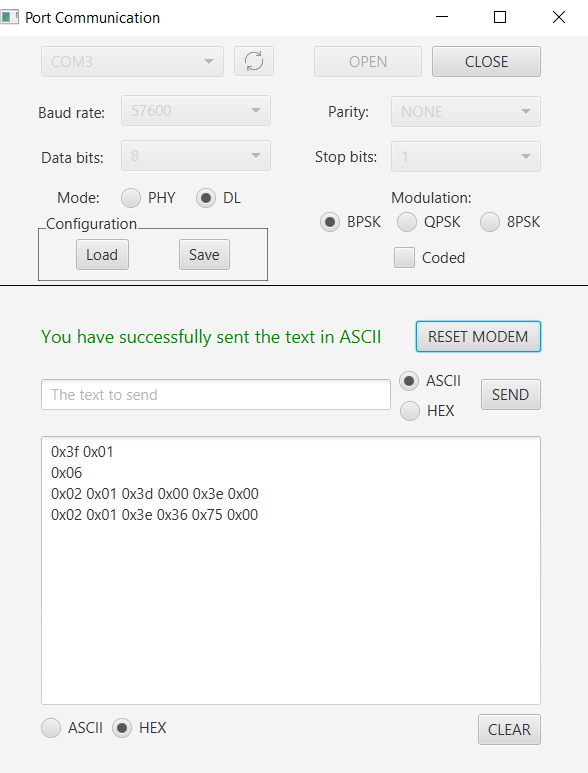


W klasie MainController wyświetlane jest okno dialogowe do zapisu pliku. Tworzony jest obiekt konfiguracji z aktualnymi parametrami i przekazywany do metody save z klasy SaveConfiguration.



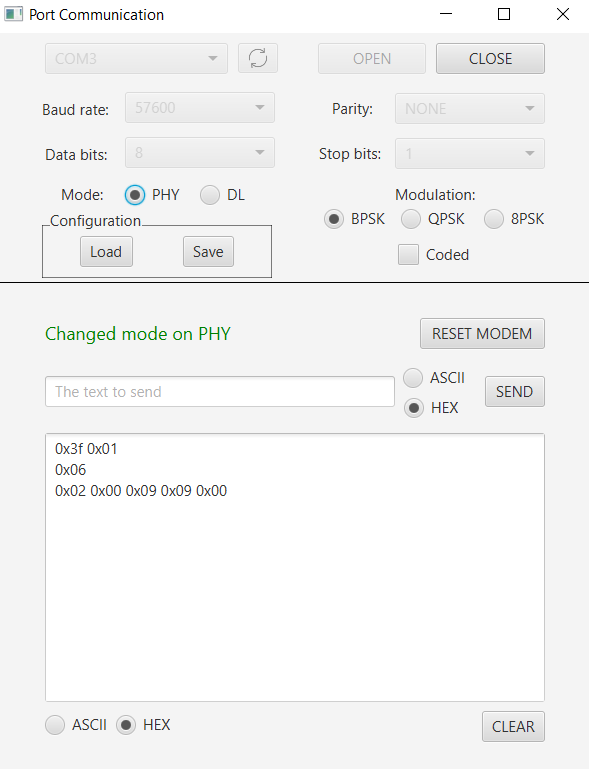
1. Przykłady działania programu podczas pracy z modemem
   1. Reset modemu

Efekt zwracanych danych po resecie modemu z poziomu aplikacji:



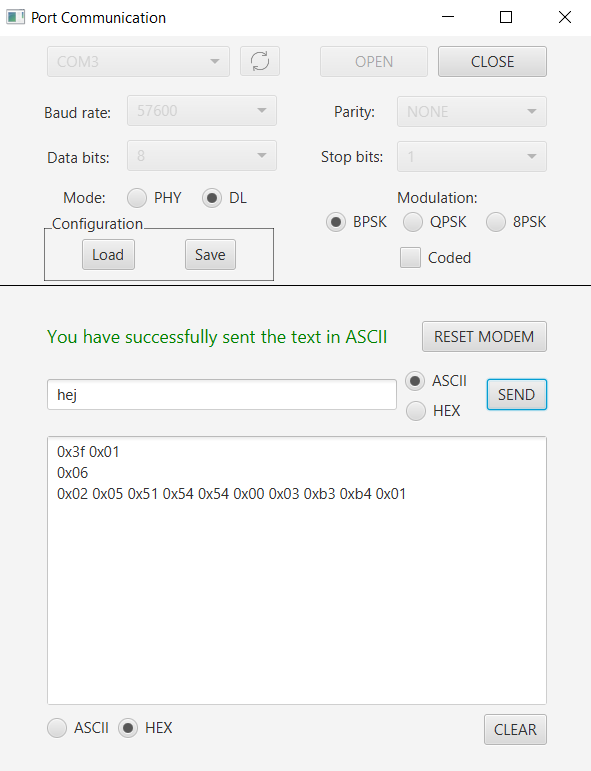
* 1. Zmiana trybu na PHY

Efekt działania aplikacji po zmianie trybu na PHY:

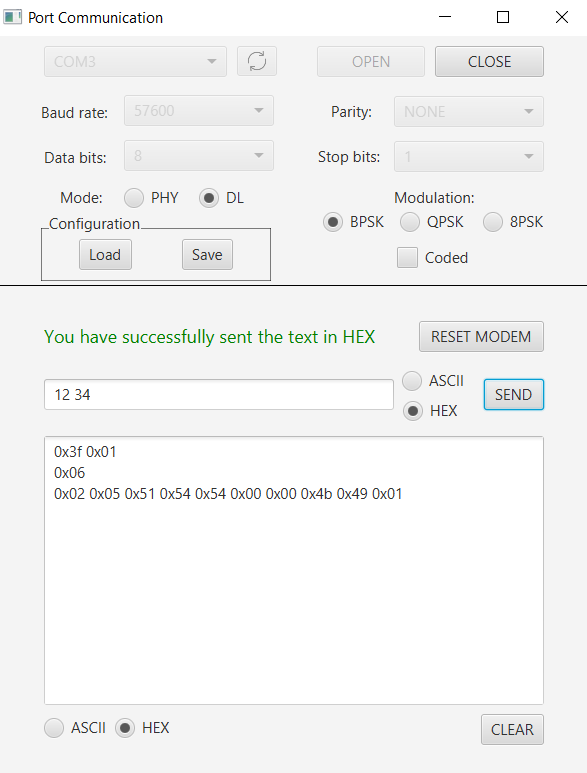


* 1. Wysłanie wiadomości

Wysłanie wiadomości przez modem w ASCII

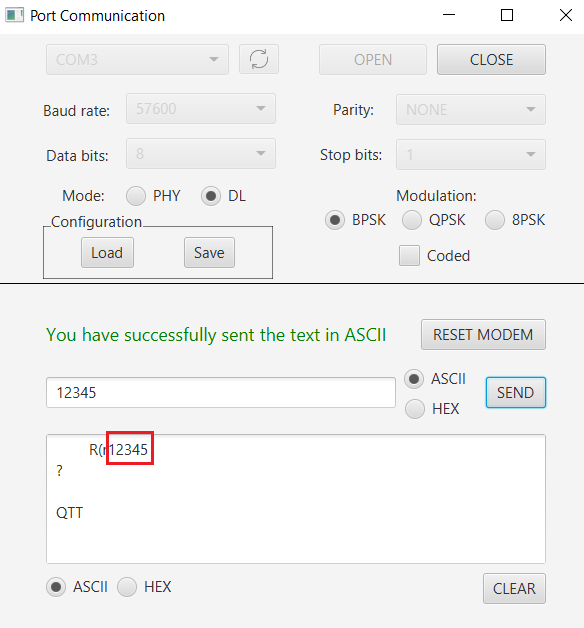


Wysłanie wiadomości w HEX



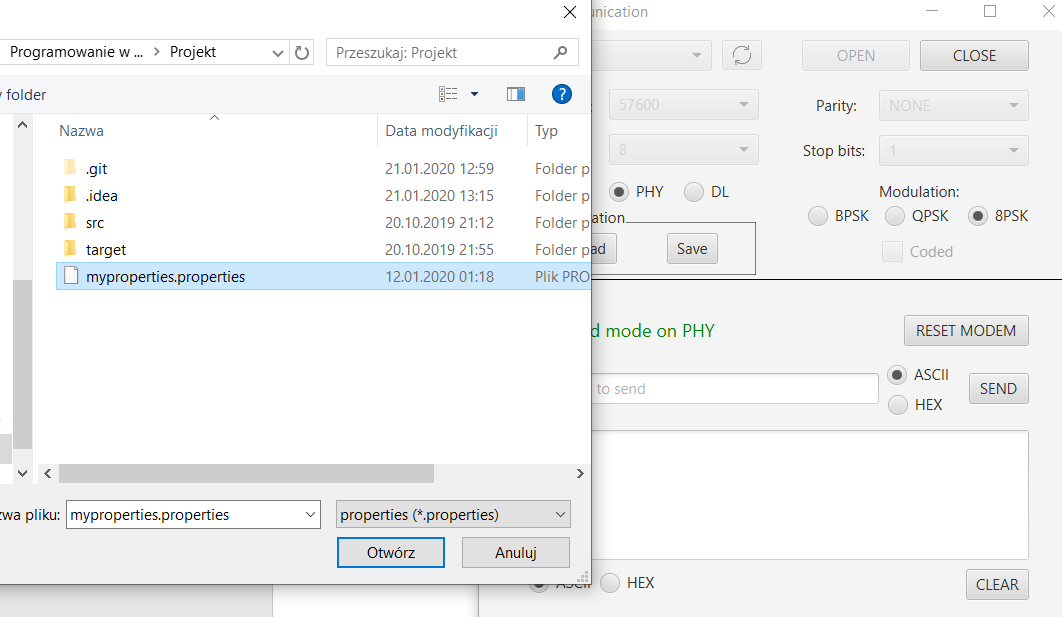
* 1. Odbiór wiadomości

Podczas testowania aplikacji na zajęciach kolega wysyłał wiadomość „12345”, która została odebrana przez moją aplikację. Efekt widać poniżej:

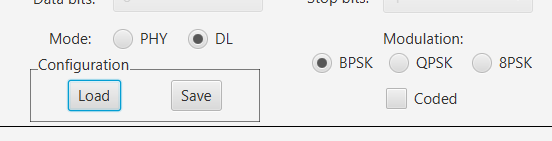


* 1. Ładowanie konfiguracji

Przed załadowaniem:



Po załadowaniu:



1. Wnioski

Aplikacja, którą musiałem napisać w ramach projektu początkowo wydawała się bardzo dużym wyzwaniem. Jednak dzięki regularnym zajęciom projektowym wszelkie niejasności były rozwiązywane na bieżąco.

Podczas realizowania projektu nie obyło się bez problemów. Głównym problemem był brak dostępu do modemu w domu, przez co testować program można było jedynie na zajęciach. Często program na zajęciach nie działał według oczekiwań, a podczas zajęć było zbyt mało czasu na naprawienie problemu.

W początkowej fazie projektu miałem problem podczas wysłania komendy resetu do modemu. Przed wysłaniem jakichkolwiek danych należało ustawić linię RTS i DTR a po wysłaniu należało obie linie wyczyścić. Kolejnym problemem na jaki napotkałem to podczas odbioru wiadomości, komunikaty zapętlały się. Było to spowodowane brakiem wysłania ACK po otrzymaniu wiadomości.

Napisany przeze mnie program dał mi wiele satysfakcji, szczególnie podczas końcowego testowania efektów pracy, kiedy razem z kolegami wysyłaliśmy sobie wiadomości przez modem.