

**WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA
„Copernicus” we Wrocławiu**

WYDZIAŁ INFORMATYKI

Kierunek studiów: **Informatyka**

Poziom studiów: **Studia pierwszego stopnia-inżynierskie**

Specjalność: **Systemy i sieci komputerowe**

PRACA DYPLOMOWA INŻYNIERSKA

Paweł Koryciński

**Symulator sterownika do RET-a implementujący
protokół AISG 2.0**

Ret driver simulator for AISG 2.0 Device

Ocena pracy:
(ocena pracy dyplomowej, data, podpis promotora)

.....
(pieczętka uczelni)

Promotor:

dr Grzegorz Debita

WROCŁAW 2020

Spis treści

Wykaz skrótów i symboli	4
1 Wstęp	6
1.1 Wprowadzenie	6
1.2 Cel pracy	6
1.3 Motywacja	6
1.4 Zakres	7
1.4.1 Baza danych	7
1.4.2 Back-end	7
1.4.3 Front-end	8
1.4.4 Testowanie	8
Część Przeglądowa	9
2 RET	10
2.1 Prezentacja urządzenia	10
2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej	11
3 Protokół AISG 2.0	13
3.1 Warstwy	13
3.1.1 1-sza - Fizyczna	13
3.1.2 2-ga - Łącza danych	13
3.1.3 7-ma - Aplikacji	13
3.2 HDLC	14
3.2.1 Struktura ramki	14
3.2.2 Typy ramek	14
3.3 Typy ramek HDLC	15
3.3.1 Ramka XID	15
3.3.2 Ramka U	16
3.3.3 Ramka I	16
4 SOLID	18
4.1 SRP	18
4.2 OCP	18
4.3 LSP	18

4.4 ISP	18
4.5 DIP	19
Część Praktyczna	20
5 Wzorce projektowe	21
5.1 Behawioralne	21
5.1.1 Komenda	21
5.1.2 Obiekt pusty	23
5.1.3 Metoda szablonowa	24
5.1.4 Strategia	24
5.1.5 Wstrzykiwanie zależności	27
5.1.6 Inicjowanie przy pozyskaniu zasobu	27
5.2 Kreacyjne	27
5.2.1 Budowniczy	27
5.2.2 Fabryka	28
6 Wymagania funkcjonalne	30
6.1 Opis wymagań dla warstwy fizycznej oraz łącza danych wraz z nazwą komendy	30
6.1.1 Ustanowienie prędkości połączenia - SetLinkSpeed	30
6.1.2 Negocjacja roli - AddressAssignment	31
6.1.3 Negocjacja parametrów HDLC - HDLCParameters	31
6.1.4 Ustanowienie synchronicznego trybu komunikacji - LinkEstablishment	32
6.1.5 Negocjacja parametrów HDLC - 3GPPReleaseID	32
6.1.6 Negocjacja parametrów HDLC - AISGProtocolVersion	33
6.2 Opis wymagań dla warstwy aplikacyjnej wraz z nazwą komendy	33
6.2.1 Kalibracja - Calibrate	33
7 Analiza nawiązanej komunikacji	34
7.1 Ustanowienie prędkości połączenia	35
7.2 Skanowanie urządzeń	35
7.3 Żadanie adresacji	35
7.4 Ponowne skanowanie urządzeń	35
7.5 Negocjacje parametrów HDLC	36
7.6 Przejście na normalny tryb komunikacji	37
7.7 Kalibracja	37

8 Logi z wykonania	40
8.1 Program właściwy	40
8.2 Test jednostkowe	40
8.3 Test modułowe	42
9 Wymagania niefunkcjonalne	43
9.1 Środowisko uruchomieniowe	43
10 Plan dalszego rozwoju	44
10.1 Porażki odnośnie planu początkowego	44
10.1.1 Weryfikacja wiadomości pod względem obecności zarezerwowanych znaków	44
10.1.2 Interwały czasowe	45
10.1.3 Ustalanie maski	45
10.2 Rozbudowa aplikacji o nowe funkcjonalności	45
11 Podsumowanie	47
Bibliografia	48
Spis rysunków	50
Spis listingów	51
Zawartość płyty DVD	52
Załączniki	53
Oświadczenie autorskie	53
Oświadczenie o udostępnieniu pracy	54

Wykaz skrótów i symboli

3GPP — organizacja normalizująca rozwój telefonii komórkowej;

5G — standard sieci komórkowej, następca LTE;

AISG 2.0 — protokół bazujący na komunikacji half duplex oraz protokole HDLC;

Back-end — warstwa oprogramowania obsługująca niskopoziomową logikę biznesową;

bajt — najmniejsza adresowalna jednostka informacji pamięci komputerowej;

bit — najmniejsza jednostka informacji w odniesieniu do sprzętu komputerowej;

broadcast — rozgłoszeniowy tryb transmisji danych;

C++ — język programowania;

CRC — Cyclic Redundancy Check - system sum kontrolnych;

CRC-16 — 16-bitowy cykliczny kod nadmiarowy;

DB — database - baza danych;

debugowanie — proces analizy programu pod kątem zaistniałych błędów;

Debug — priorytet logowania wskazujący informacje potrzebne podczas debugowania;

delete — słowo kluczowe języka C++, którego użycie wywołuje destruktor obiektu;

driver — sterownik - program obsługujący urządzenie podłączone do komputera;

enkapulacja — opakowanie danych z wyższej warstwy w warstwie niższej;

execute — metoda interfejsu klasy ICommand służąca do uruchomienia komendy;

Error — priorytet logowania informujący o błędny wykonaniu programu;

Front-end — warstwa oprogramowania obsługująca odbiór danych od użytkownika;

gniazdo — dwukierunkowy punkt końcowy połączenia sieciowego;

GSM — Global System for Mobile Communications - standard telefonii komórkowej;

half duplex — połączenie w którym naprzemienne jest przesyłanie i odbieranie informacji;

HDLC — High-Data Link Control - protokół warstwy łącza danych;

HDLC body — ramka HDLC bez bajtów startu, stopu oraz sumy CRC;

h — hour - godzina;

Info — priorytet logowania informujący o ważnych etapach w wykonaniu programu;

klasa finalna — klasa po której nie można zdefiniować dziedziczenia;

little endian — cienkokoncowość;

LTE — Long Term Evolution - standard bezprzewodowego przesyłu danych;

Wykaz skrótów i symboli

metoda — funkcja należąca do klasy;

min — minute - minuta;

model OSI — model odniesienia łączenia systemów otwartych;

ms — milisecond - milisekunda;

N(R) — receive sequence number, numer porządkowy odebranej ramki;

N(S) — send sequence number, numer porządkowy wysłanej ramki;

null — wartość przypisywana do wskaźnika równa 0;

numer portu — jeden z parametrów gniazda, identyfikujący proces nim zarządzający;

OFDMA — metoda zwielokrotnienia w dziedzinie częstotliwości;

P/F bit — Pool/Final bit - obliczany podczas kalkulacji bajtu kontrolnego;

payload — fragment wiadomości zawierający jedynie istotne informacje;

POSIX Regexp — standard zapisu wyrażeń regularnych;

RAII — Resource acquisition is initialization - inicjowanie przy pozyskaniu zasobu;

ramka — pakiet danych;

RET — Remote Electrical Tilt - urządzenie zmieniające elektryczny kąt wiązki anteny;

RS-485 — standard transmisji szeregowej;

std::any — klasa reprezentująca dowolny typ danych;

std::string — klasa reprezentująca łańcuch znaków;

std::vector — klasa reprezentująca dynamiczną tablicę;

STL — Standard Template Library - biblioteka C++ w przestrzeni nazw std:::

s — second - sekunda;

Trace — priorytet logowania najniższej rangi;

UI — User Interface - interfejs użytkownika;

UMTS — Universal Mobile Telecommunications System - standard telefonii komórkowej;

UniqueId — złożenie numeru seryjnego wraz z kodem producenta;

USB — Universal Serial Bus - uniwersalna magistrala szeregowa;

Warning — priorytet logowania informujący o wykonaniu mogącym powodować błędy;

WCDMA — technika związana z dostępem do sieci radiowej;

1. Wstęp

1.1 Wprowadzenie

Stacja nadawcza to zintegrowany system składający się z modułu systemowego, modułu rozszerzeniowego, radia oraz fizycznej anteny. Ma ona na celu dostarczenie jak największej liczbie ludzi sygnału telekomunikacyjnego w celu nawiązania połączenia głosowego, wysłania wiadomości tekstowej czy skorzystania ze skrzynki mailowej. Głównym problemem jest to, że stacja nadawcza może być umieszczona w jednym miejscu, natomiast odbiorcy, mogą przemieszczać się, co niesie ze sobą problem efektywnego pokrycia obszaru zasięgiem sieci operatora. W celu modyfikacji charakterystyki sygnału, anteny dipolowe usprawniane są o dodatkowe urządzenie o nazwie *RET*, które jest szeroko stosowane w technologiach *GSM*, *WCDMA* czy *LTE*.

1.2 Cel pracy

Jako cel obrałem zaznajomienie się z protokołem komunikacyjnym AISG 2.0, którego użycie możemy zaobserwować na drodze pomiędzy radio modułem a wzmacniaczem antenowym czy RET-em. Jest to krok obowiązkowy przed rozpoczęciem zapoznawania się z AISG 3.0. Implementacja protokołu będzie wymagała wysokich umiejętności programowania w języku C++, tworzenia testów, wdrażania wzorców projektowych oraz wiedzy z zakresu systemu kontroli wersji czy inżynierii oprogramowania.

1.3 Motywacja

W przyszłości podłączenie RET-a do komputera (pomijając radio moduł) przy pomocy adaptera *RS-485 -> USB*, w celu skrócenia czasu testowania urządzenia.

1.4 Zakres

Praca obejmuje implementację sterownika, którego analizę możemy podzielić na:

1.4.1 Baza danych

- Implementacja:
 - `std::vector<std::string, std::any>`
 - Programowanie optymalne dla cache-u procesora
- Klucz:
 - Generowanie unikalnego;
 - Walidacja - Extended POSIX Regexp: `,/[a-z]+_[1-9]+”;`
- Operacje:
 - Dodaj;
 - Usuń;
 - Aktualizuj;
 - Pobierz wartość na podstawie : konkretnego/typu klucza;

1.4.2 Back-end

- Realizacja wzorców projektowych:
 - Komenda;
 - Metoda szablonowa;
 - Obiekt pusty;
 - Budowniczy;
 - *RAII*;
 - Wstrzykiwanie zależności;
- Implementacja:
 - L1 - Warstwy fizycznej;
 - L2 - Warstwy łącza danych;
 - L7 - Warstwy aplikacyjnej;
- Logowanie ruchu aplikacji:
 - `<h:min::s::ms> priorytet [nazwaPliku::nazwaFunkcji:numerLinii]` komunikat;
 - Obsługiwane priorytety: Trace < Debug < Info < Warning < Error
 - Filtrowanie w zależności od wybranego minimalnego priorytetu
 - Rezultat przekazywany na wyjście standardowe oraz do pliku tekstowego

1.4.3 Front-end

- Konsolowy interfejs użytkownika umożliwiający wydawanie komend:
 - Kontrolera sterownika;
 - Bazy danych;
- Realizacja wzorców projektowych:
 - Komenda;
 - Metoda szablonowa;
- Przekazywanie logów aplikacji z Back-endu;
- Walidacja komend wpisanych przez użytkownika:
 - Odrzucanie nieznanych komend;
 - Podpowiedź odnośnie wartości argumentów komend już znanych;

1.4.4 Testowanie

- Front-end
 - Jednostkowe
 - Modułowe
- Back-end
 - Jednostkowe
 - * HDLC
 - * HDLC Body
 - * DB
 - Modułowe
 - * Happy Path
 - * Sad Path
 - Integracja
 - * UI + DB
 - * UI + DB + Back-end Controller
- Komponentowe

W celu weryfikacji działania symulatora sterownika od początku do końca zalecane jest uruchomienie całego komponentu jako czarną czarną skrzynkę oraz operowanie na nim przy pomocy zaimplementowanego interfejsu. W tym celu zaimplementowany został również symulator urządzenia, który odbiera wiadomości oraz odpowiada na nie.

Część Przeglądowa

2. RET

2.1 Prezentacja urządzenia

Na poniższym rysunku przedstawiono RET-a czyli urządzenie dla którego zaimplementowano symulator sterownika.[2]



Rysunek 2.1. RET - Miejsce na antenę.
(Zdjęcie własne)



Rysunek 2.2. RET - podłączonym kablem RS-485 oraz włożoną atrapą anteny.
(Zdjęcie własne)

2.2 Zmiana szerokości głównej wiązki fali elektromagnetycznej



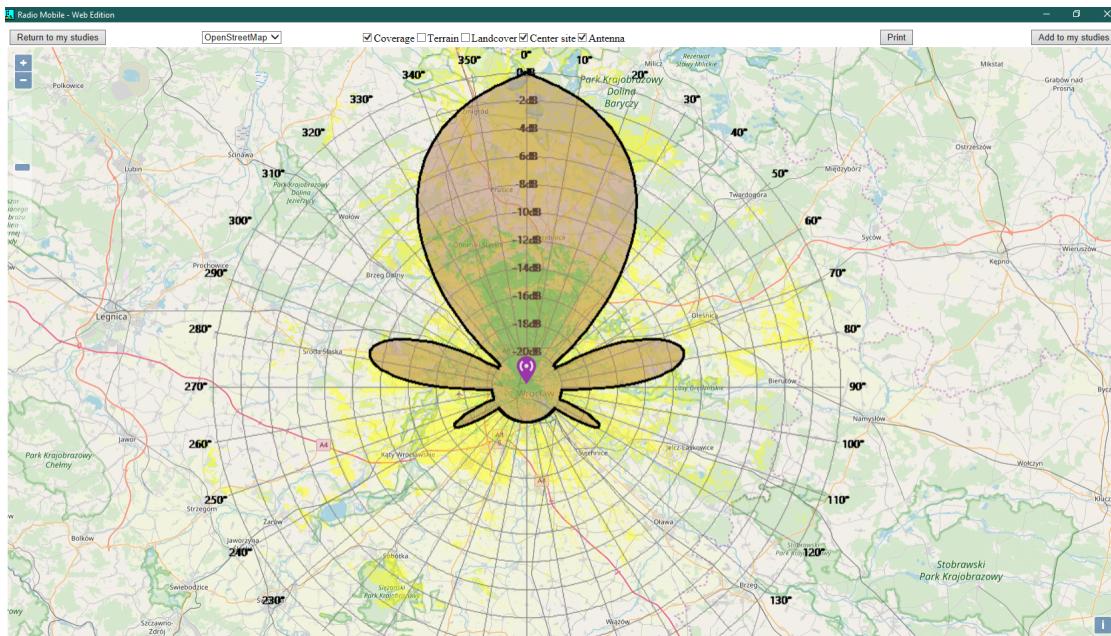
Rysunek 2.3. Rysunek przedstawiający kąt modyfikowany przy pomocy RET-a.
(Opracowanie własne)

Poniższe rysunki wygenerowano dzięki programowi Radio Mobile[5], Przedstawiono na nich szerokość wiązki promieni sygnału radiowego w osi poziomej[1] zmieniającą się dzięki modyfikacji elektrycznego kąta na wartość 40-tu stopni. Podczas symulacji użyto antenę Yagi, która aktualnie nie jest stosowana w technologii mobilnej z racji wspieranych przez nią częstotliwości, gdyż są one znacznie niższe aniżeli te wymagane przez WCDMA, LTE czy 5G, aczkolwiek bardzo dobrze odzwierciedla działanie tych rzeczywistych ze względu na swoją charakterystykę kierunkową.

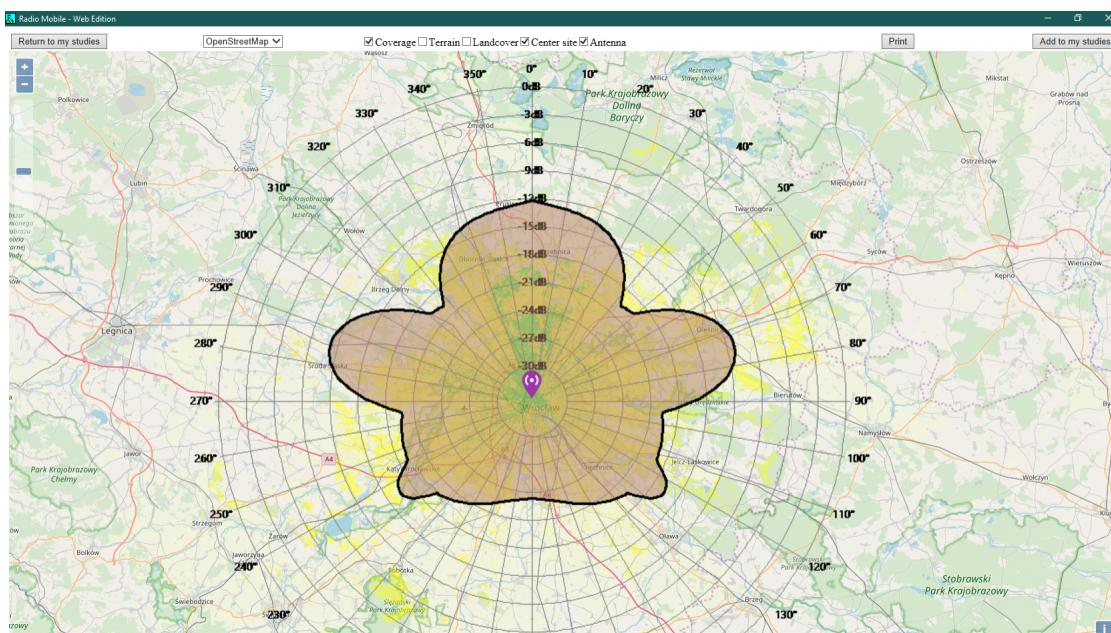
Antenę umieszczono przy WSIZ Copernicus.

Pomarańczowym kolorem przedstawiono charakterystykę zysku promieniowania anteny, które w przypadku kąta 0 stopni jest wydłużone oraz węższe, co pozwala pokryć śladowym zasięgiem nawet dalekie obszary, lecz do terenów bliżej zlokalizowanych dostarczony jest słabszy sygnał względem tego, który można otrzymać, zmieniając elektryczny kąt anteny na 40 stopni. Dzięki RET-owi, można skoncentrować wiązkę na mniejszym obszarze, oferując znacznie wyższe prędkości transmisji danych.

2. RET



Rysunek 2.4. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 0 stopni.
(Opracowanie własne przy pomocy programu Radio Mobile)



Rysunek 2.5. Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kąt elektryczny 40 stopni.
(Opracowanie własne przy pomocy programu Radio Mobile)

3. Protokół AISG 2.0

Symulator sterownika ma za zadanie odebrać od użytkownika polecenie w postaci komendy wieloargumentowej, która zawierała będzie nazwę procedury rozpoznawanej przez odpowiednie warstwy protokołu komunikacyjnego *AISG v2.0* realizującego podzbiór *modelu OSI*.

3.1 Warstwy

3.1.1 1-sza - Fizyczna

Realizacja pracy w kierunku emulacji fizycznego połączenia do urządzenia niesie ze sobą pewne zmiany na tej warstwie.

Należy pominąć obszar mechaniczny i elektryczny[1], a skupić się na obszarze funkcjonalnym oraz proceduralnym.

3.1.2 2-ga - Łącza danych

Rola tej warstwy jest:

- Enkapsulacja[BIB] ramki *HDLC Body* do ramki *HDLC[BIB]*
 - Dodanie *bitów startu i stopu*;
 - Obliczenie oraz walidacja sumy *CRC[BIB]*
- Budowa ramki typu *XID* podczas procedur negocjacji:
 - Rozmiaru ramki;
 - Unikalnego identyfikatora urządzenia, na które składa się numer seryjny oraz kod producenta;
 - Wersji *3GPP* oraz *AISG* urządzenia podległego;
 - Adresu;
- Budowa ramki typu *U* w celu ustanowienia normalnego trybu komunikacji;

3.1.3 7-ma - Aplikacji

- Budowa ramki typu *I*;
- Rozpoznawanie wysokopoziomowej komendy kalibruj;

3.2 HDLC

Z języka angielskiego High-Level Data Link Control. Protokół warstwy łącza danych modelu OSI. Standard HDLC opisuje norma ISO, lecz szeroko stosuje się także implementację CISCO. HDLC jest stosowany w technologii WAN, ponieważ obsługuje zarówno połączenia dwupunktowe, jak i wielopunktowe. Jest protokołem o orientacji bitowej oraz jest przezroczysty informacyjnie.[3] W przypadku, jeśli przesyłana wartość jest wielobajtowa, zastosowane jest podejście *little endian*. Głównym powodem, dla którego protokół utworzony w roku 1979 roku wciąż znajduje zastosowanie jest fakt, że jego implementacja pozwala w maksymalnym stopniu wyeliminować możliwość utraty przesyłanej informacji dzięki mechanizmowi validacji sumy *CRC* oraz algorytmowi ewaluacji *bajtu kontrolnego*. Dzięki temu urządzenie nadzędne może nawet żądać ponownego przesłania wcześniej otrzymanej wiadomości.

3.2.1 Struktura ramki

1. Flaga startu - 0x7E;
2. Adres stacji docelowej;
3. Sterowanie - określa typ ramki oraz jej parametry w zależności od typu;
4. Dane;
5. Suma kontrolna FCS (dwubajtowa) - na przykład CRC-16;
6. Flaga stopu - 0x7E;

3.2.2 Typy ramek

- Ramka I - Informacyjna;
- Ramka U - Nienumerowana;
- Ramka S - Nadzorująca;
- Ramka XID - Identyfikująca urządzenia;

3.3 Typy ramek HDLC

3.3.1 Ramka XID

Jej nazwa pochodzi z języka angielskiego „exchange identification”. Służy do przekazania urządzeniu podrzdnemu wiedzy na temat możliwości oraz charakterystyki komunikacji na warstwie łącza danych. [4] Odpowiadając na tego typu wiadomości, urządzenie podrzędne najczęściej zwraca tę samą wartość w przypadku zgodności, bądź najwyższą wspieraną, jeśli żądana jest zbyt duża. Szereg wysłanych i odebranych wiadomości XID nazywamy XID negocjacją. Tej ramki użyto również podczas ustalania prędkości komunikacji, co jest częścią procedury zestawienia warstwy fizycznej. Bajt kontrolny dla wiadomości przesyłanej ma zawsze wartość 0xBF.

Bajty budujące ramkę:

- Adresu;
- Kontrolny;
- Identyfikujący format;
- Identyfikujący grupę;
- Długości grupy;
- Parametrów HDLC:
 - Identyfikujący parametr;
 - Długości parametru;
 - Wartości parametru;

Wspomniano o parametrach HDLC, gdyż wiadomość negocjująca ich wartości może zawierać zarówno jeden jak i więcej parametrów.

3.3.2 Ramka U

Jej nazwa pochodzi z języka angielskiego „Unnumbered” co oznacza nienumerowana[4]. Wzięło się to z tego, że wielokrotnie wysłana wiadomość tego typu, zawsze posiada tą samą wartość bajtu kontrolnego. Służy ona do zarządzania warstwą łącza danych, a czasami również do przesyłania pewnych informacji.

Bajty budujące ramkę:

- Adresu;
- Kontrolny;

Poszczególne wiadomości przesyłane przy pomocy tej ramki identyfikowane są dzięki charakterystycznej wartości bajtu kontrolnego.

3.3.3 Ramka I

Jej nazwa pochodzi z języka angielskiego „Information” co oznacza informacyjna[4]. Dzięki niej można żądać od urządzenia podległego wykonania wysokopoziomowej operacji zdefiniowanej przez warstwę aplikacyjną, a nawet przesłać najnowszą wersję oprogramowania urządzenia. Jej długość definiowana jest podczas XID negocjacji w trakcie zestawiania warstwy łącza danych.

Bajty budujące ramkę:

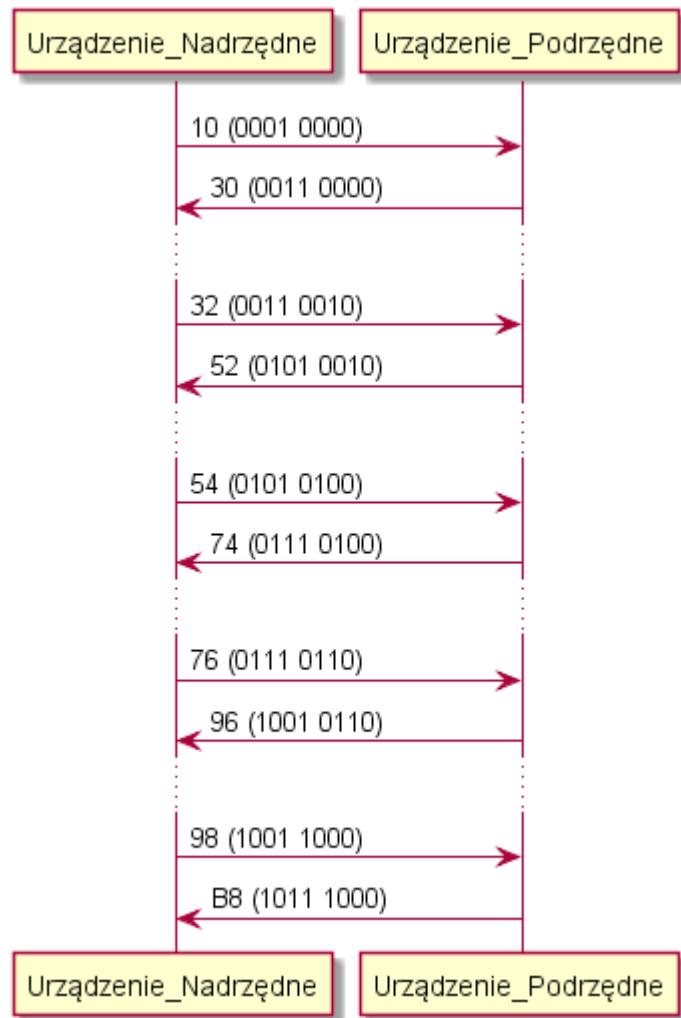
- Adresu;
- Kontrolny;
- Kodu procedury;
- Dodatkowe wartości procedury/raportujące proces wykonania procedury;

Wiadomości tego typu przy wielokrotnym zwołaniu żądania wykonania tej samej procedury posiadają zmienną wartość bajtu kontrolnego, co różni ją od ramki U czy XID.

Proces ten w przypadku bitu P/F zlokalizowanego na pozycji o indeksie 4 bajtu kontrolnego:

- Poinformowania urządzenia podległego, że urządzenie nadzorcze żąda odpowiedzi na właśnie przesyłaną wiadomość;
- Poinformowania urządzenia nadzorczego o zakończeniu nadawania odpowiedzi na wiadomość;

Wartość bitu na pozycji 0 jest zawsze równa 0. Bit od piątego do siódmego włącznie definiują nam licznik wiadomości odebranych przez urządzenie nadzorcze, czyli $N(R)$. Bit od pierwszego do trzeciego włącznie definiują nam licznik wiadomości wysłanych przez urządzenie nadzorcze, czyli $N(S)$. Na poniższym diagramie przedstawiono ewaluację bajtu kontrolnego dla kolejnych wiadomości warstwy aplikacyjnej.



Rysunek 3.1. Ramka informacyjna - ewaluacja bajtu kontrolnego.
(Opracowanie własne)

4. SOLID

Projektując sterownik, podjęto próbę podążania zgodnie z dobrymi praktykami programowania, powszechnie znanyymi jako SOLID. Pomyślna ich realizacja sprawia, że korzystanie, rozbudowywanie, jak i utrzymywanie powstałego kodu źródłowego przypomina przyjemne poruszanie się po świecie gry aniżeli walkę z napotkałymi przeszkodami. Tak powstały program umożliwiające ponowne używanie wcześniej powstałego kodu. Poniższe reguły czasami bywają trudne w realizacji, zwłaszcza kiedy tworzony system osiągnie duże rozmiary, dlatego warto myśleć o nich już na początkowym etapie. Trzeba też pamiętać o tym, że nie jest rozsądne stosowanie którejkolwiek z reguł SOLID, jeśli nie ma ku temu wyraźnych powodów, dlatego jedynie doświadczenie oraz intuicja pozwoli nam wyczuć moment podjęcia decyzji projektowej. Poniżej przedstawiono prawa budujące zasadę SOLID wraz z ich głównymi założeniami.

4.1 SRP

Z angielskiego Single Responsibility Principle, a więc zasada pojedynczej odpowiedzialności. Definiuje ona, że powód modyfikacji klasy powinien być tylko jeden. [6, 103] Realizacja tej reguły uchroni nas przed niechcianą modyfikacją definicji dużej liczby klas w przypadku zmiany logiki tylko jednej z nich.

4.2 OCP

Z angielskiego Open/Closed Principle, a więc zasada otwarte-zamknięte. Definiuje ona, że encje programowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji. [6, 117] Jeśli ta zasada zostanie właściwie zastosowana, to nowe zmiany uzyskuje się poprzez dodanie nowego kodu, aniżeli zmiane istniejącego.

4.3 LSP

Z angielskiego Liskov Substitution Principle, a więc zasada podstawiania Liskov. Definiuje ona, że musi być możliwość podstawienia typów pochodnych za ich typy bazowe. [6, 127] Trzeba pamiętać o tej regule w trakcie definiowania relacji dziedziczenia pomiędzy klasami.

4.4 ISP

Z angielskiego Interface Segregation Principle, a więc zasada segregacji interfejsów. Definiuje ona, że klienci nie powinny być zmuszone do zależności od metod, których nie używają.

[6, 151] Dzięki postępowaniu zgodnie z tą zasadą, utworzone interfejsy abstrakcyjnych klas bazowych będą zawierały minimalną liczbę funkcji czysto wirtualnych.

4.5 DIP

Z angielskiego Dependency Inversion Principle, a więc zasada odwracania zależności. Definiuje ona, że moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. I jedne, i drugie powinny zależeć od abstrakcji. [6, 141] Kolejnym postulatem jest to, że abstrakcje nie powinny zależeć od szczegółów. To szczegóły powinny zależeć od abstrakcji. [6, 141]

Część Praktyczna

5. Wzorce projektowe

W latach, kiedy programowanie obiektowe stawało się bardziej popularne, wiele osób natrafiało na grupę problemów pewnej kategorii. Podczas wielokrotnych prób ich rozwiązywania, różne osoby dochodziły do wspólnych wniosków odnośnie do architektury kodu źródłowego tworzonego w celu ich rozwiązań. Napotykane wyzwania rozpoczęto dzielić według trzech kategorii: behawioralne, strukturalne oraz kreacyjne. Tak oto to powstały wzorce projektowe. Obszerna znajomość tej niemałej dziedziny informatyki pozwala spojrzeć na kolejne zadania w znacznie bardziej zaawansowany sposób aniżeli wcześniej. Można je porównać do znanych przekształceń oraz wzorów matematycznych w rachunku całkowym, gdyż zostały one ściśle zdefiniowane, a osiągniecie efektu końcowego różni się zastosowaniem innych parametrów wejściowych. Podczas tworzenia wzorców usilnie trzymano się zbioru kolejnych zasad znanych pod nazwą SOLID.

Poniżej przedstawiono użyte podczas projektowania sterownika wzorce wraz z fragmentami kodu źródłowego.

5.1 Behawioralne

5.1.1 Komenda

Dzięki utworzeniu interfejsu komendy, zrealizowano regułę otwarcia na modyfikacje a zamknięcia na zmiany. W przypadku poprawnej definicji funkcjonalności, która powinna zostać zrealizowana, dodajemy jedynie implementację tego interfejsu, przez co cała regresja pozostaje bez zmian, a dzięki kontrolerowi, który kolejkuje komendy można pokusić się o realizację kompozytu oraz wprowadzenie systemu wag bądź poleceń terminujących aktualnie zaplanowane zadania. Podczas implementacji sterownika zastosowano połączenie trzech wzorców projektowych a to wszystko dzięki potężnemu podejściu do organizacji struktury kodu jakim jest użycie komend. Zestawiając implementację interfejsu komendy oraz budowania konkretnej wiadomości dzięki fabryce pozwala na zdefiniowanie *klas finalnych* zawierających charakterystyczne dla każdej z komend wywołań.

```
1 class ICommand
2 {
3     public:
4         virtual void execute() = 0;
5         virtual std::string handleResponse() = 0;
6         void registerResponseHandler(std::function<void(void)> responseHandler
7 );
8     virtual ~ICommand();
9 protected:
```

5. Wzorce projektowe

```
9     ICommand();
10    using AlmagControllerInformer = boost::signals2::signal<void(void)>;
11    AlmagControllerInformer informControllerAboutResult_;
12 };
```

Listing 5.1. Interfejs komendy

```
1 void DeviceScan::execute()
2 {
3     executeImpl();
4     informControllerAboutResult_();
5 }
6
7 HDLCFrameBodyPtr DeviceScan::getFrameBody() const
8 {
9     return hdlcFrameBodyFactory_->get_FrameXID_DeviceScan();
10 }
11
12 void DeviceScan::executeImpl()
13 {
14     hdlcCommunicator_->communicate(validatedUserInput_[IDX_OF_ADDRESS_],
15                                         getFrameBody());
16 }
17
18 std::string DeviceScan::handleResponse()
19 {
20     return constraints::almag::L2::DEVICE_SCAN + DELIMITER;
21 }
```

Listing 5.2. Definicja klasy konkretnej komendy używającej fabryki oraz strategii

5.1.2 Obiekt pusty

Znaną przypadłością w językach obiektowych jest wykonywanie dalszej akcji w zależności od stanu pewnego z obiektów. W przypadku korzystania ze wskaźników dobrą praktyką jest każdorazowa weryfikacja czy jego wartość nie jest równa *null*. Wykorzystanie klasy która implementuje interfejs kontrolera, w sposób neutralny sprawia, że zawsze bezpieczne będzie wywołanie na jej obiekcie instancjonującym którykolwiek z metod.

```
1 void AlmagControllerNull::addCommands(const StringsMatrix&
    validatedUserInput)
2 { LOG(debug); }
3
4 bool AlmagControllerNull::executeCommand()
5 { LOG(debug); return true; }
6
7 void AlmagControllerNull::handleCommandsResult()
8 { LOG(debug); }
9
10 std::string AlmagControllerNull::getFinalresultCode()
11 { return defaultVals::FOR_STRING; }
```

Listing 5.3. Definicja klasy dla obiektu pustego

```
1 ReturnCode CMenu::interpretControllerCommand(const Strings& userInput)
2 {
3     LOG(debug) << "Start";
4     if (const auto validatedUserInput = almagCmdValidationMgr_->perform(
5         userInput))
6     {
7         almagCtrl_->addCommands({*validatedUserInput});
8         return almagCtrl_->executeCommand();
9     }
10    LOG(warning) << "Validation rejected the command";
11    return true;
11 }
```

Listing 5.4. Przykład użycia obiektu pustego

5.1.3 Metoda szablonowa

Wprowadzenie systemu walidacji argumentów podanych przez użytkownika gwarantuje bezpieczne wykonywanie akcji niskopoziomowych takich jak wysłanie wiadomości do urządzenia, bez zbędnego umieszczanie logiki weryfikacji w dalszym etapie. Zastosowanie tego wzorca umożliwi w przyszłości wdrożenie dodatkowych mechanizmów. Jednym z nich jest automatycznie pobranie długiego adresu portu na który ma zostać wysłana komenda, w przypadku podaniu klucza obiektu w bazie danych. Natomiast następnym jest funkcjonalność automatycznej korekty w przypadku zaistniałego błędu syntaktycznego we wprowadzanej przez użytkownika komendzie. Obecność komendy „execute” pozwala połączyć wywołanie powyższych funkcji za pomocą jednego polecenia, nie posiadając wiedzy o tym jakiego typu procedury weryfikacji sterownik będzie próbował dokonać. Powyższą funkcjonalność osiągnięto dzięki efektownemu wykorzystaniu funkcji wirtualnych.

```

1 class ICommandValidation
2 {
3     public:
4         virtual ~ICommandValidation() = default;
5         MaybeStrings execute(Strings userInput);
6     protected:
7         virtual MaybeStrings validateInputCorrectness(Strings userInput) = 0;
8         virtual MaybeStrings modifyIfRequired(Strings validatedUserInput) = 0;
9 }
```

Listing 5.5. Plik nagłówkowy dla metody szablonowej walidacji komendy

```

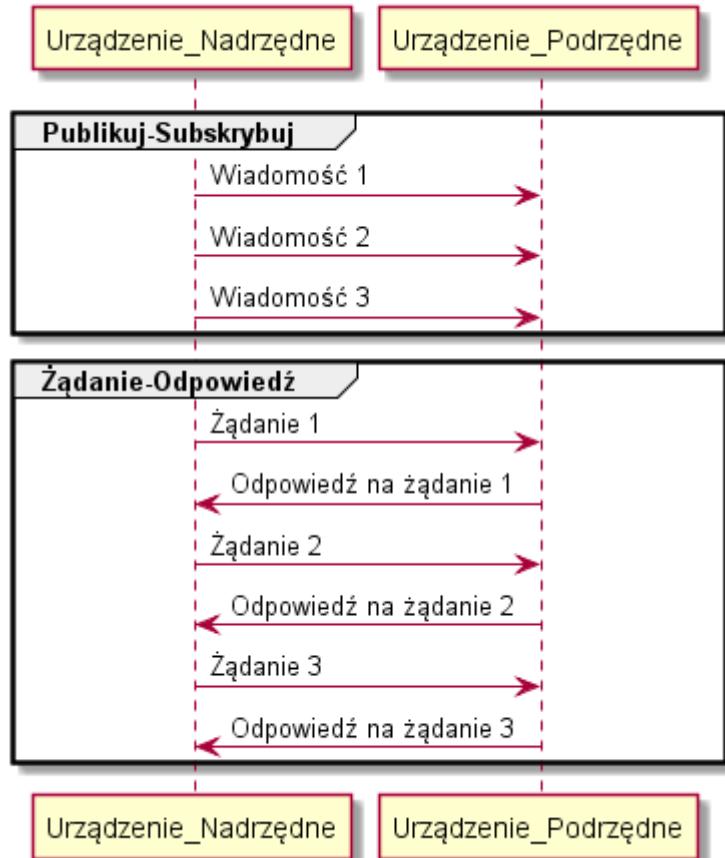
1 MaybeStrings ICommandValidation::execute(Strings userInput)
2 {
3     if (auto successfullyValidatedInput =
4         validateInputCorrectness(userInput))
5         return modifyIfRequired(*successfullyValidatedInput);
6     return boost::none;
7 }
```

Listing 5.6. Metoda szablonowa - Wywołanie metod wirtualnych z poziomu innej metody

5.1.4 Strategia

Znanych jest wiele wzorców komunikacji pomiędzy komponentami aczkolwiek w projekcie użyto dwa najbardziej znane: Publikuj-Subskrybuj oraz Żadanie-Odpowiedź.

Pierwszy z nich efektownie realizuje zasadę odwrócenia zależności, ponieważ sterownik urządzenia nadzorujące na początkowym etapie nie musi znać ilości podłączonych do linii urządzeń. Drugi natomiast pozwala zrealizować podejście do transmisji z urządzeniem znane jako półduplex, wymagane podczas wywoływania komend zestawiających warstwę łączącej danych oraz



Rysunek 5.1. Przepływ wiadomości dla wzorca Publikuj-Subskrybij oraz Żądanie-Odpowiedź.
(Opracowanie własne)

aplikacyjnej. Głównym problem jest to, że obydwa wzorce wymagają innego zestawu komend w celu konfiguracji połączenia ze slotami systemowymi. W projekcie zaimplementowano jeden kontroler, który zarządza zestawianiem wymaganych warstw OSI w trakcie ciągłego uruchomienia sterownika, co osiągnięto dzięki dynamicznemu podmianie strategii komunikacji z Publikuj-Subskrybij na Żądanie-Odpowiedź. Zaobserwowano pewne złe następstwo nieoprawnego użycia wzorca strategii polegające na prewencyjnemu rzuceniu wyjątku w przypadku gdyby programista wywołał niepoprawną metodę. W związku z tym, podczas przyszłej rozbudowy programu, zastosowany zostanie inny wzorzec o nazwie most.

```
1 void ZMqReqRepPrimaryStrategy::setupSend(const std::string& address)
2 {
3     tcpPortAddress = tcpPortAddressHeader + address;
4     socket_.connect(tcpPortAddress);
5 }
6
7 void ZMqReqRepPrimaryStrategy::setupReceive(const std::string& address)
8 { throw std::runtime_error("Redundant function"); }
9
10 bool ZMqReqRepPrimaryStrategy::send(const std::string &address,
11     HDLCFrameBodyPtr frame)
12 {
13     const std::string sentMessage = toString(frame->build());
14     return s_send(socket_, sentMessage);
15 }
16 HDLCFramePtr ZMqReqRepPrimaryStrategy::receive(const std::string &address
17 )
18 {
19     std::string message = s_recv(socket_);
20     auto recFrame{
21         std::make_shared<HDLCFrame>(HDLCFrameBodyInterpreter().apply(
22             message));
23     }
24     return recFrame;
25 }
26 HDLCFramePtr ZMqReqRepPrimaryStrategy::communicate(const std::string&
27     address, HDLCFrameBodyPtr frame)
28 {
29     send(tcpPortAddress, frame);
30     std::this_thread::sleep_for(1s);
31     receive(tcpPortAddress);
32     return nullptr;
33 }
```

Listing 5.7. Strategia komunikacji Żadanie-Odpowiedź dla urządzenia nadziednego

5.1.5 Wstrzykiwanie zależności

Klasa „HDLCCCommand” bezpośrednio dziedziczy po klasie „Command”. Dzięki implementacji interfejsu „execute” zrealizowany kontroler posiada możliwość, przyszłej rozbudowy nawet o komendy typu „włącz muzykę”. W przypadku obsługi urządzenia implementującego protokół AISG, zaobserwowano zapotrzebowanie na dodatkową klasę abstrakcyjną, która posiadała będzie wskaźnik na obiekt implementujący interfejs fabryki ramek oraz wzorca komunikacji. Podejście programowania sterowanego testami umożliwiło przedstawienie programisty przed koniecznością przekazania powyższych obiektów z poziomu testu, w celu wyeliminowania wymagania podłączania fizycznego urządzenia bądź uruchamiania aplikacji symulującej urządzenie oraz skrócenia czasu regresji programu, dzięki korzystaniu z atrap. Ten sposób zarządzania kolejnością tworzenia oprogramowania naturalnie wymusił realizację wstrzykiwania zależności polegającej na przekazywaniu obiektów z zewnątrz podczas wywoływanego konstruktora, aniżeli utworzeniu konstruktora zeroparametrowego, który uniemożliwia dalszą modernizację elementów składowych systemu.

```
1 HDLCCCommand(  
2     IHDLCFrameBodyFactoryPtr frameBodyFactoryPtr,  
3     IHDLCCommunicatorPtr hdlcCommunicator,  
4     Strings userInput  
5 );
```

Listing 5.8. Konstruktor zrealizowany podejściem wstrzykiwania zależności

5.1.6 Inicjowanie przy pozyskaniu zasobu

Do implementacji pracy użyto kompilatora GCC wspierającego język C++ w wersji 11-tej, który posiada wbudowany mechanizm inteligentnych wskaźników. „shared_ptr<T>” oraz „unique_ptr<T>” zmieniły sposób korzystania z dynamicznie alokowanej pamięci w sposób diametralny. Do tej pory dealokacja pamięci wskazywanej przez wskaźnik należała do obowiązków programisty. Dzięki podejściu tzw. RAII dla „shared_ptr<T>”, w przypadku destrukcji wszystkich wskaźników odnoszących się do wcześniej zaalokowanego obszaru pamięci, kompilator przy pomocy destruktora wywołuje komendę „delete” automatycznie, dzięki czemu programista jest ostrzeżony przed niepożądanym wyciekiem pamięci.

5.2 Kreacyjne

5.2.1 Budowniczy

Często zdarza się, że obiekt klasy wymaga modernizacji wielu z jego pól a realizacja konstruktora posiadającego trzy bądź więcej parametrów, uznawana jest za niepoprawną implementacje oraz antywzorzec. W tej sytuacji z pomocą pojawia się wzorzec budowniczego, który pod-

czas wywoływania metody zmieniającej stan obiektu, dokonuje zamierzonego celu, po czym zwraca referencję na obiekt na rzecz którego została wywołana *metoda*. Umożliwia to szeregowe wywoływanie kolejnych modyfikatorów co znacznie oczyściło i zwiększyło czytelność programu. Istnieje wiele interpretacji tego wzorca projektowego, w których dodana jest metoda finalizująca budowanie obiektu.

```
1 HDLCFrameBodyPtr HDLCReqFrameBodyFactory::get_FrameI_Calibrate() const
2 {
3     const auto retFrame = FrameI()
4         .setAddressByte(0x03)
5         .setControlByte(frameI::BYTE_CONTROL::CALIBRATE_REQ)
6         .setProcedureCode(PROCEDURE_CODE::CALIBRATE_SRET)
7         .setParameterLength(Hexes{ZERO, ZERO});
8     return std::make_shared<FrameI>(retFrame);
9 }
```

Listing 5.9. Budowniczy wraz z Fluent API podczas budowania ramki I - Kalibruj

5.2.2 Fabryka

Istnieje pewien wzorzec, który owiany jest złą sławą. Programiści w momencie usłyszenia o nim dostają ciarek, gdyż sądzą, że pod tym słowem, kryje się obiekt, który potrafi zachować się w nieprzewidziany sposób podczas każdorazowego wywoływanego jego metod. Z drugiej strony poprawna jego realizacja, umożliwia dynamiczną zmianę wartości zwracanych przed system, a w przypadku sterownika dała możliwość uwspólnienie kodu wraz z symulatorem urządzenia, na poziomie 90%. Mowa o fabryce. W przypadku nieprzechowywania jakiegokolwiek stanu w jej instancji oraz zapoznania się w całości z realizowanym problemem komunikacji pomiędzy urządzeniem podlegającym i nadającym, prawdziwe jest to, że zaledwie na jedną komendę sterownik nie oczekuje odpowiedzi, a odpowiednie nazwanie metod interfejsu fabryki pozwoli zaobserwować, że po obu stronach trzeba obsłużyć komunikację oraz budowę wiadomości charakterystyczną na przykład dla polecenia „kalibruj” wprowadzając jedynie niewielkie modyfikacje.

```
1 class HDLCReqFrameBodyFactory : public IH DLCFrameBodyFactory
2 {
3     public:
4         HDLCFrameBodyPtr get_FrameI_Calibrate() const override;
5         HDLCFrameBodyPtr get_FrameU_LinkEstablishment() const override;
6         HDLCFrameBodyPtr get_FrameXID_3GPPReleaseId() const override;
7         HDLCFrameBodyPtr get_FrameXID_AddressAssignment() const override;
8         HDLCFrameBodyPtr get_FrameXID_AISGProtocolVersion() const override;
9         HDLCFrameBodyPtr get_FrameXID_DeviceScan() const override;
10        HDLCFrameBodyPtr get_FrameXID_DummyScan() const override;
11        HDLCFrameBodyPtr get_FrameXID_HDLCParameters() const override;
```

5. Wzorce projektowe

```
12     virtual ~HDLCReqFrameBodyFactory() = default;  
13 };
```

Listing 5.10. Fabryka budowniczych dla sterownika urządzenia nadziednego

6. Wymagania funkcjonalne

6.1 Opis wymagań dla warstwy fizycznej oraz łącza danych wraz z nazwą komendy

Częstą praktyką modelowania etapu łączenia z RET-em jest użycie przy maszynie stanowej w związku z czym, z poniższych funkcjonalności należy korzystać zgodnie z zadaną kolejnością. Dla każdej z poniższych ramek wyznaczono część wspólną składającą się z czterech bajtów. Pierwszym z nich jest bajt flagi startu, która oznacza początek wiadomości i ma wartość 0x7E. Koleje bajty budują ramkę XID, I, U czy S. Następnie dodaje się dwubajtową sumę CRC oraz na samym końcu flagę stopu, która jest tej samej wartości co flaga startu i oznacza koniec nadawania wiadomości.

6.1.1 Ustanowienie prędkości połączenia - SetLinkSpeed

Jako użytkownik chcę ustanowić prędkość transmisji na 9.6 kbps.

Jest to komenda wysyłana do urządzenia podlegającego przy pomocy ramki XID.

Pierwsze dwa bajty o wartości 0xFF odpowiadają za zdefiniowanie adresu urządzenia docelowego. W tym przypadku celem jest synchronizacja prędkości połączenia ze wszystkimi urządzeniami na lini, które nie są zaadresowane, a więc jest to wiadomość typu textitbroadcast;

Kolejne dwa bajty kontrolne o wartości 0xBF są charakterystyczne dla ramki XID.

Następne dwa bajty o wartości 0x81 identyfikują format wiadomości XID, co oznacza, że jest to wiadomość z kategorii przypisania adresu.

Potem napotkano kolejne dwa bajty o wartości 0xF0, które są identyfikatorem grupy, co definiuje nam jakie kolejne bajty dopuszczalne są w dalszej części wiadomości.

Urządzenie podlegające w celu umożliwienia mu zdekodowania wiadomości, otrzymuje w wiadomości długość grupy, czyli dwa bajty w tym przypadku o wartości 0x08 i jest to liczba bajtów która pojawi się następnie, aż do pierwszego bajta sumy CRC.

Przechodząc do przesyłanych parameterów, jako pierwszy będzie to unikalny identyfikator urządzenia. Rozpoznano go po identyfikatorze parametru o wartości 0x01. Następnie podajemy liczbę bajtów zajmowanych przez UniqueID czyli 0x02. Kolejnie podajemy wartość unikalnego identyfikatora. Z racji tego, że nie oczekujemy odpowiedzi na tę wiadomość, są to dowolne wartości. Drugim przesyłanym parametrem jest maska danych z identyfikatorem o wartości 0x03, potem podajemy liczbę bajtów budujących przesyłaną przez nas maskę czyli 0x02, a w wartości parametru umieszczały 4-ry bajty o wartości obu 0xFF. Jest to najbardziej ogólna maska urządzenia.

6.1.2 Negocjacja roli - AddressAssignment

Jako użytkownik chcę zaadresować urządzenie typu SingleRET identyfikujące się unikalnym identyfikatorem $UniqueID == \{0x4E, 0x4B, 0x34, 0x36, 0x35, 0x30, 0x30, 0x30, 0x30\}$ adresem 3, przy pomocy komendy „AddressAssignment”.

Budowa ramki wysyłanej przez tę komendę jest bardzo podobna do powyższej z racji tego, że powyższą można nazwać atrapą skanu a tą skanowaniem celowanym. Z racji tego, że różna jest liczba oraz wielkość poszczególnych parametrów HDLC, bajt rozmiaru grupy posiada wartość 0x11.

Pierwszy parametr identyfikuje się przy pomocy wartości 0x01 czyli jest to unikalny identyfikator urządzenia, który ma zostać zaadresowany. Kolejny bajt przypomina o rozmiarze parametru i w tym przypadku jest on równy 0x09. W powyższym opisie przedstawiono unikalny identyfikator urządzenia, który w przypadku równejści z rzeczywistym, pozwala urządzeniu podzielić się na przyjęcie żądanie adresacji.

Następny parametr posiada wartość 0x02 i oznacza on, że w jego wartości urządzenie nadziedne zdefiniowało adres urządzeniu wewnętrznemu, dzięki któremu będzie identyfikowane oraz z racji tego, że długość parametru ma wartość 0x01, to wartość wynosi 0x03. W przypadku obecnej pracy, nie będzie miał on większego znaczenia aczkolwiek w momencie kiedy urządzenia typu Ret połączone są ze sobą szerogowo, potrzeba jest możliwość wysłania wiadomości do na przykład drugiego w łańcuchu.

Wartość ostatniego parametru równa 0x04, definiuje typ urządzenia podzielnego, z którym inicjalizowana jest komunikacja. Długość wynosi 0x01 a wartość to 0x01. Oznacza ona, że podejmowana jest próba nawiązania komunikacji z pojedynczym RET-em. Dla porównania istnieją również urządzenia takie jak MultiRET, dzięki któremu można zmieniać wartość nachylania kąta głównej wiązki anteny na wielu osiach czy płaszczyznach.

6.1.3 Negocjacja parametrów HDLC - HDLCParameters

Jako użytkownik chcę ustalić maksymalny rozmiar nadawanej jak i odbieranej ramki typu I oraz maksymalną liczbę ramek jednocześnie możliwych do wysłania bądź odebrania.

Powyższą wiadomość można rozbić na 4-ry osobne aczkolwiek połączono je z racji podobnych funkcjonalności realizowanych podczas jej wysłania. Protokół AISG bazuje na HDLC, które używane jest w wielu miejscach gdzie potrzebne jest połączenie połączenie o wysokiej gwarancji otrzymania wiadomości bez utraty informacji. Istnieje nawet jego realizacja służąca do komunikacji satelit kosmicznych i różni się ona od AISG głównie ustalonimi wiadomościami HDLC, gdzie potrzeba wysłać znacznie większą liczbę ramek na raz aniżeli tylko jedna. Pierwszy bajt adresu może zostać zmieniony z wartości 0xFF na 0x03, gdyż w poprzedniej wiadomości ustanowiono adres urządzenia podzielnego. Ta wartość będzie aplikowana do każdej

6. Wymagania funkcjonalne

poniższej wiadomości a więc pominięto jej wspominanie, gdyż potraktowano ją jako stałą. Następnym bajtem jest identyfikator formatu, który dla wiadomości zawierającej parametry HDLC jest równy 0x81.

Kolejny bajt czyli identyfikator grupy ma wartość 0x80, a następny (rozmiar grupy) jest równy 0x12.

Pierwszym parametrem HDLC który jest negocjowany to maksymalny rozmiar *payloadu* dla ramki informacyjnej nadawanej przez urządzenie nadrzędne. Identyfikatorem w tym przypadku jest 0x05. Rozmiar payloadu jest równy 0x04 a wartościami są { 0xF0, 0x2D, 0x00, 0x00 }.

Kolejny bajty natomiast budują negocjowany maksymalny rozmiar payloadu ramki informacyjnej do wiadomości otrzymywanej przez urządzenie nadrzędne. Identyfikator ma wartość 0x06 a pozostały bajty są tożsame jak dla poprzedniej negocjacji.

Następnym parametrem jest maksymalna liczba ramek wysłanych pod rząd przez urządzenie nadrzędne. Identyfikator ma wartość 0x07 a zarówno długość jak i wartość to 0x01.

Ostatnim parametrem negocjowanym podczas tej wiadomości jest maksymalna liczba ramek otrzymanych pod rząd poprzez urządzenie nadrzędne. Wartość identyfikatora jest równa 0x08 a długość oraz wartość znów posiadają wartości 0x01.

6.1.4 Ustanowienie synchronicznego trybu komunikacji - LinkEstablishment

Jako użytkownik chcę ustawić synchroniczny tryb komunikacji z urządzeniem.

Bajt kontrolny posiada wartość 0x93 i oznacza żądanie ustalenia sposobu komunikacji z urządzeniem podlegającym na tryb normalny, a więc taki w którym wysyła ono ramkę jedynie jako odpowiedź na wcześniej nadaną przez urządzenie nadrzędne.

6.1.5 Negocjacja parametrów HDLC - 3GPPReleaseID

Jako użytkownik chcę wynegocjować parametr HDLC urządzenia, jakim jest wspierana wersja 3GPP.

Teraz można przejść do coraz bardziej szczegółowych parametrów jak ustanowienie numeru wersji standardu 3GPP. Podczas tworzenia tej wiadomości znów skorzystano z ramki XID a więc należy zdefiniować identyfikator formatu, grupy oraz jej długość. Pierwsze dwie wartości są równe tym z komendy „AddressAssignment” a długość ma wartość 0x03. Identyfikatorem parametru jest tym razem wartość 0x05, długością 0x01 natomiast wartością parametru 0x08. Jest to wersja standardu w wersji 8-mej czyli pionierska dla technologii *LTE*. Wprowadzono w niej wparcie dla interfejsu radiowego opartego o *OFDMA*, którego wykorzystanie można zaobserwować również przy technologii 5G.

6.1.6 Negocjacja parametrów HDLC - AISGProtocolVersion

Jako użytkownik chcę wynegocjować parametr HDLC urządzenia, jakim jest wspierana wersja protokołu AISG.

Z racji tego, że implementowana wersja protokołu AISG 2.0 jest jedną z wielu (istnieją jeszcze dwie wcześniejsze: 1.0, 1.1 oraz najnowsza 3.0), potrzeba zdefiniować według którego standardu budowane są wiadomości przez urządzenie nadzędne oraz rozpoznawane przez podrzędne. Bajt identyfikujący parametr dla tej wiadomości ma wartość 0x14, długość parametru to 0x01 a jej wartość to 0x02.

6.2 Opis wymagań dla warstwy aplikacyjnej wraz z nazwą komendy

Po pomyślnym zestawieniu warstwy fizycznej posiadamy zdefiniowany wszystkie charakterystyki połączenia, które pozwolą w jednoznaczny sposób porozumieć się z urządzeniem oraz żądać odpowiedzi na wysłaną wiadomość. Kolejne komendy są już żądaniami wysokozłożomowymi, które definiują użyteczne dla klienta polecenia służąca do zarządzania RET-em. Jest ich znacznie więcej, aczkolwiek obranym celem pracy inżynierskiej było pomyślne przeprowadzenie kalibracji urządzenia, co pozwala w dalszym etapie ustawić zadany kąt odchylenia głównej wiązki sygnału z anteny.

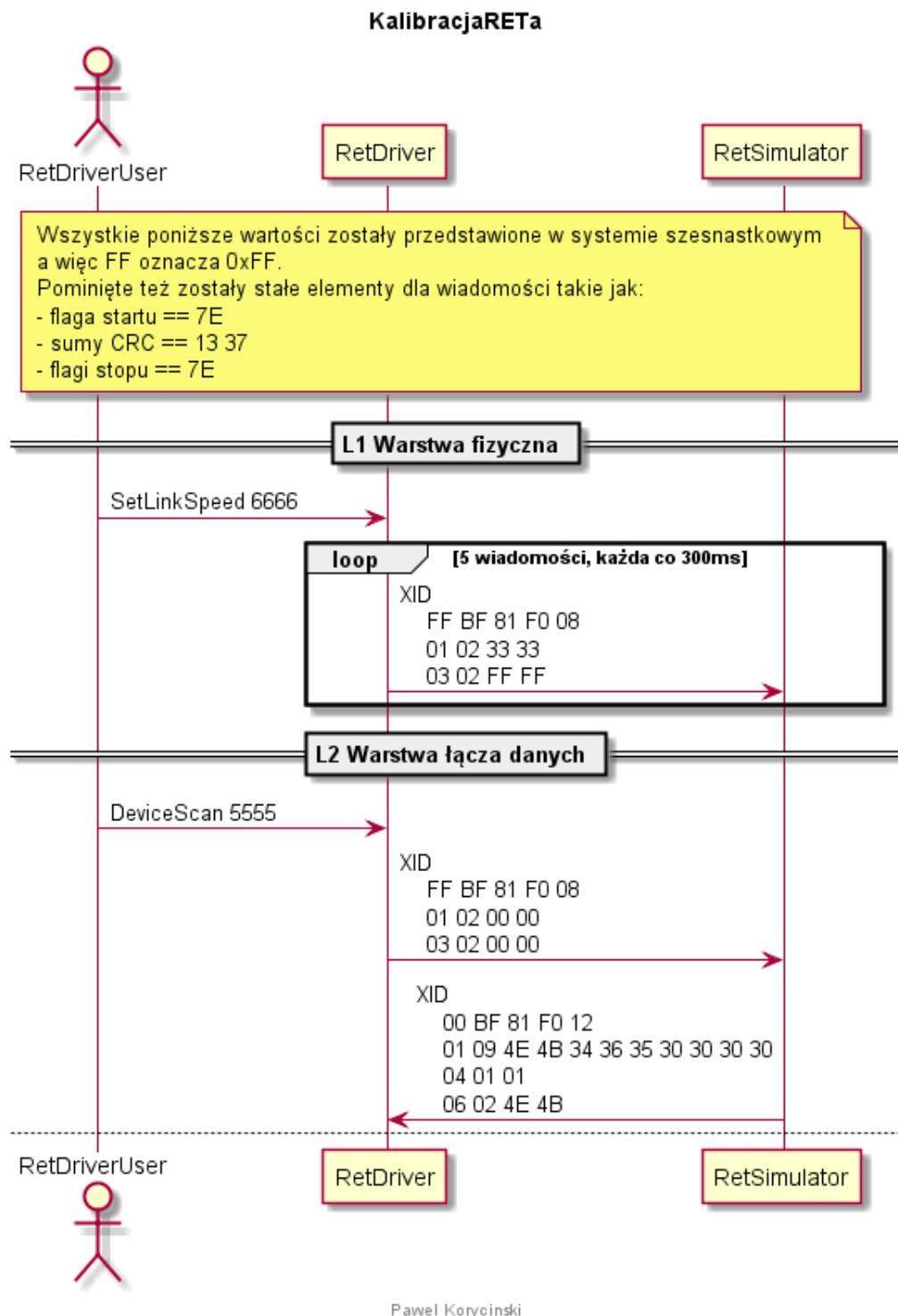
6.2.1 Kalibracja - Calibrate

Jako użytkownik chcę skalibrować urządzenie.

Z racji tego, że wiadomości tej warstwy korzystają z ramki informacyjnej, inną rolę pełni bajt kontrolny. Poza stałymi wartości bitów w bajcie każdorazowe wysłanie i odebranie wiadomości zmienia wartość sekcji P/F wcześniej wspomnianej. Dzięki temu można w łatwy sposób identyfikować czy wiadomość, którą otrzymujemy jest ta której oczekiwaliśmy. W związku z tym dla pierwszej wiadomości wysłanej przez urządzenie nadzędne poprawną wartością będzie 0x10. Kolejnym bajtem jest kod procedury, który ma wartość 0x31, co oznacza, że oczekujemy kalibracji urządzenia będącego zwykłym pojedynczym RET-em.

Z racji tego, że podczas tej komendy nie jest przesyłana żadna wartość dla tej procedury, dwa bajty zaalokowane dla długości parametru mają wartość {0x00, 0x00}

7. Analiza nawiązanej komunikacji



Rysunek 7.1. Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń/
(Opracowanie własne)

Na diagramach sekwencji przedstawiono komendy wywoływanie przez użytkownika symulatora wraz z zawartościami ramek pochodzących z urządzenia nadziednego oraz wartościami ramek przychodzących z urządzenia podziednego. W celu analizy procesu komunikacji, w opisie skupiono się na cechach charakterystycznych dla poszczególnej ramki czy komendy, o których nie wspomniano we wcześniejszych rozdziałach.

7.1 Ustanowienie prędkości połączenia

Parametrem tej komendy jest adres portu 6666, z którego korzysta sterownik do nawiązania połączenia typu tcp na adresie 127.0.0.1 wraz z symulatorem urządzenia przy zastosowaniu wzorca Publish-Subscribe. Podczas tego połączenia protokół AISG 2.0 nie zakłada oczekiwania na odpowiedź od urządzenia podziednego oraz użyta biblioteka ZeroMQ również nie udostępnia możliwości wysłania odpowiedzi na taką wiadomość.

7.2 Skanowanie urządzeń

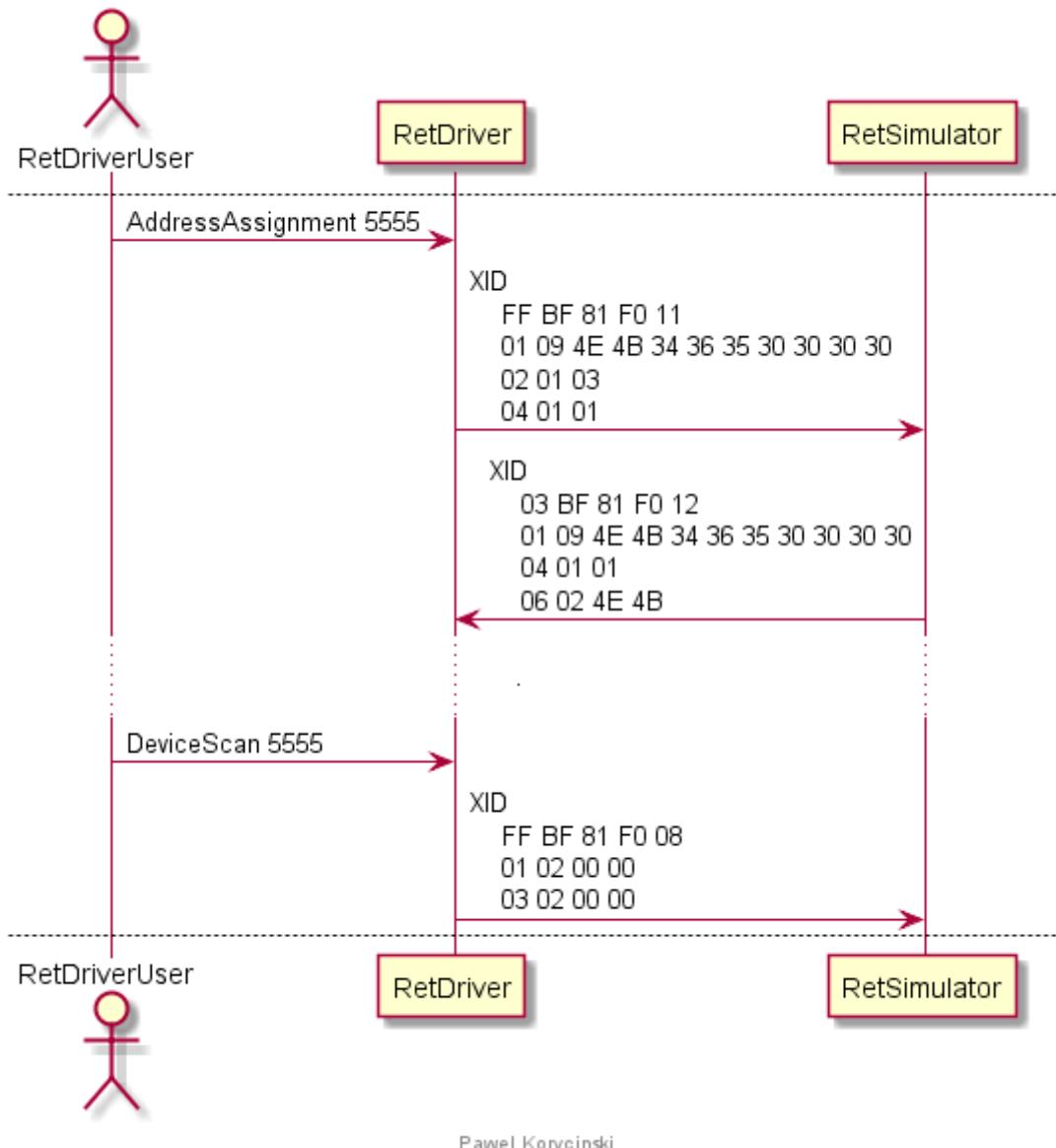
Parametrem tej komendy jest adres portu 5555, z którego korzysta sterownik do nawiązania połączenia typu tcp na adresie 127.0.0.1 wraz z symulatorem urządzenia, przy zastosowaniu wzorca Żadanie-Odpowiedź. Jest to wzorzec, który zakłada otrzymanie odpowiedzi na każdą wyslaną wiadomość, zanim kolejna zostanie nadana. Opisany mechanizm komunikacji aplikowalny jest również do wszystkich poniższych komend. Ciekawym elementem tej wiadomości jest to, że otrzymano kod producenta zarówno w osobnym parametrze jak i jako składową unikalnego identyfikatora urządzenia.

7.3 Żadanie adresacji

Tutaj po raz pierwszy można zaobserwować zmienioną wartość pola adresu dla wiadomości przychodzącej. Oznacza to, że urządzenie podziedne zaakceptowało żadanie adresacji oraz identyfikuje się w trakcie rozmowy z urządzeniem nadziednim pod adresem 0x03 co jest prawdą dla każdej następnej wiadomości.

7.4 Ponowne skanowanie urządzeń

W zakresie tej pracy, komunikacja nawiązywana jest z jednym urządzeniem, a więc dla czego ponownie wysłano wiadomość skanowania? Otóż na tę wiadomość urządzenie nadziedne nie powinno otrzymać odpowiedzi, gdyż jedynie urządzenia niezaadresowane mogą na nie odpowiedzieć, co jest podstawową metodą dodatkowej weryfikacji powodzenia procedury adresowania.



Pawel Korycinski

Rysunek 7.2. Żadanie adresacji oraz dodatkowe skanowanie urządzeń.
(Opracowanie własne)

7.5 Negocjacje parametrów HDLC

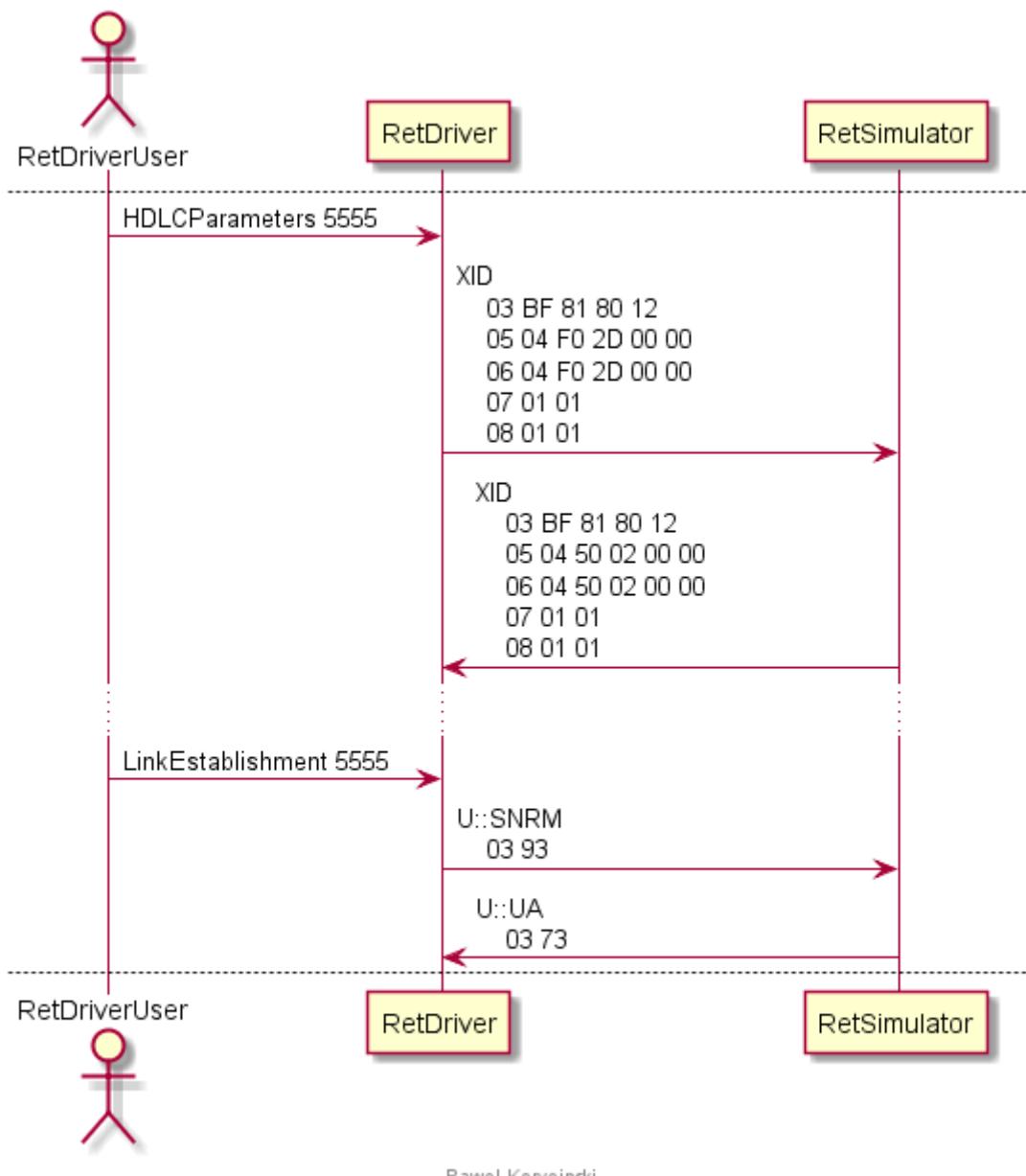
Cechą negocjacji parametrów przy pomocy ramek XID jest to, że jeśli żądana wartość jest wspierana przez urządzenie podrzędne, to odpowie ono wiadomością zawierającą te same parametry oraz te same wartości. W przeciwnym wypadku otrzymanymi wartościami parametrów będą największe możliwe przez nie wspierane. Zaobserwowano to zjawisko w przypadku negocjacji wielkości payloadu dla wysłanej oraz otrzymanej ramki informacyjnej. Urządzenie nadzorujące próbowało ustanowić dopuszczalną liczbę bitów na {0xF0, 0x2D, 0x00, 0x00} co daje wartość 61485, lecz urządzenie podrzędne odpowiedziało {0x50, 0x02, 0x00, 0x00} co po konwersji na system dziesiętny daje 20482 bity.

7.6 Przejście na normalny tryb komunikacji

Jako odpowiedź na tę wiadomość urządzenie otrzyma ramkę, która zawiera wartość 0x73, co oznacza że potwierdza ono żądane oczekiwanie.

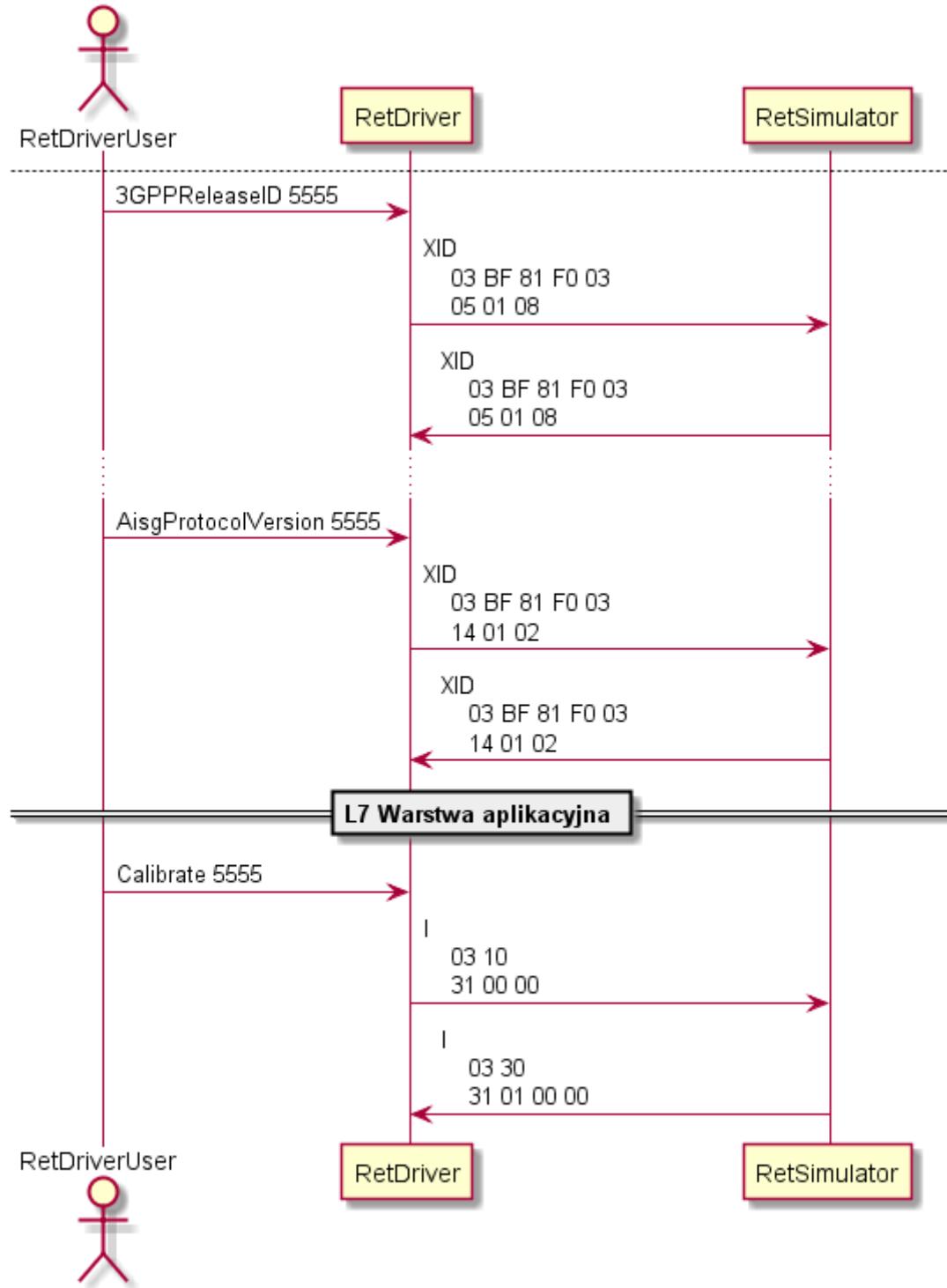
7.7 Kalibracja

W związku z tym, że oczekiwanie kalibracji przesłano przy pomocy ramki informacyjnej, bajt kontrolny posiada charakterystyczną metodę jego ewaluacji przedstawioną wcześniej. Diagram sekwencji potwierdza to, gdyż wiadomość odebrana przez urządzenie nadrzędne posiada wartość równą 0x30. Ramka odebrana, na wyznaczenie liczby bajtów budujących odpowiedź posiada zarezerowane dwa bajty. Ciekawą obserwacją jest to, że odpowiedź równą 0x00 czyli OK można by zapisać przy pomocy jedynie jednego bajta, lecz optymalizacja pamięci nie jest tutaj zastosowana.



Rysunek 7.3. Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji.
(Opracowanie własne)

Pawel Koryciński



Pawel Korycinski

Rysunek 7.4. Negocjacja pozostałych parametrów HDLC oraz żadanie kalibracji urządzenia.
(Opracowanie własne)

8. Logi z wykonania

8.1 Program właściwy

```
1 <01:40:22.701547> info [ZMqPubSubPrimaryStrategy :: setupSend:22] on tcp://127.0.0.1:6666
2 <01:40:22.701887> info [ZMqReqRepPrimaryStrategy :: setupSend:25] on tcp://127.0.0.1:5555
3 <01:40:22.701970> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
4 <01:40:56.584982> info [DummyScan :: execute:27]
5 <01:40:56.585129> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
6 <01:40:56.888292> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
7 <01:40:57.194233> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
8 <01:40:57.506573> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
9 <01:40:57.807764> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
10 <01:40:58.121425> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
11 <01:40:58.422405> info [ZMqPubSubPrimaryStrategy :: send:35] 7e ff bf 81 f0 8 1 2 33 33 3 2 ff ff 13 37 7e
12 <01:40:58.765170> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
13 <01:41:00.160840> info [CMenu :: runImpl:68] {
14 SetLinkSpeed
15 AddressAssignment
16 AISGProtocolVersion
17 DeviceScan
18 HDLCPParameters
19 LinkEstablishment
20 3GPPReleaseID
21 Calibrate
22 }
23 <01:41:00.160936> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
24 <01:41:05.459650> info [AddressAssignment :: execute:21]
25 <01:41:05.459844> info [ZMqReqRepPrimaryStrategy :: send:37] 7e ff bf 81 f0 11 1 9 4e 4b 34 36 35 30 30 30 30 2 1 3 4 1 1
   13 37 7e
26 <01:41:06.460786> debug [ZMqReqRepPrimaryStrategy :: receive:46] 7e 3 bf 81 f0 12 1 9 4e 4b 34 36 35 30 30 30 30 4 1 1 6 2
   4e 4b 13 37 7e
27 <01:41:06.461099> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
28 <01:41:11.479561> info [AISGProtocolVersion :: execute:22]
29 <01:41:11.479719> info [ZMqReqRepPrimaryStrategy :: send:37] 7e 3 bf 81 f0 3 14 1 2 13 37 7e
30 <01:41:12.485479> debug [ZMqReqRepPrimaryStrategy :: receive:46] 7e 3 bf 81 f0 3 14 1 2 13 37 7e
31 <01:41:12.485829> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
32 <01:41:17.741647> info [DeviceScan :: execute:23]
33 <01:41:17.741806> info [ZMqReqRepPrimaryStrategy :: send:37] 7e ff bf 81 f0 8 1 2 0 0 3 2 0 0 13 37 7e
34 <01:41:18.743370> debug [ZMqReqRepPrimaryStrategy :: receive:46] 7e 0 bf 81 f0 12 1 9 4e 4b 34 36 35 30 30 30 4 1 1 6 2
   4e 4b 13 37 7e
35 <01:41:18.743554> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
36 <01:41:23.806075> info [LinkEstablishment :: execute:20]
37 <01:41:23.806209> info [ZMqReqRepPrimaryStrategy :: send:37] 7e 3 93 13 37 7e
38 <01:41:24.812150> debug [ZMqReqRepPrimaryStrategy :: receive:46] 7e 3 73 13 37 7e
39 <01:41:24.812412> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
40 <01:41:29.058188> info [Calibrate :: execute:20]
41 <01:41:29.058310> info [ZMqReqRepPrimaryStrategy :: send:37] 7e 3 10 31 0 0 13 37 7e
42 <01:41:30.100991> debug [ZMqReqRepPrimaryStrategy :: receive:46] 7e 3 30 31 1 0 0 13 37 7e
43 <01:41:30.101131> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
44 <01:41:31.998110> info [CMenu :: runImpl:75] asd - not supported, please check command constraints, paste 'help' or 'exit'
45 <01:41:31.998169> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
46 <01:41:34.861291> error [DummyScanValidationStrategy :: validateInputCorrectness:8] Requires path of port as second
   argument
47 <01:41:34.861349> warning [CMenu :: interpretControllerCommand:87] Validation rejected the command
48 <01:41:34.861380> info [UserCommandParser :: receiveAndLex:34] Please pass your command ( Pascal Case )
```

Listing 8.1. Interfejs komendy

8.2 Test jednostkowe

```
1 [ 3%] Built target gtest
2 [ 6%] Built target gmock
3 [ 13%] Built target gmock_main
4 [100%] Built target KorytkoMag_RetDriver_UT
```

8. Logi z wykonania

```
5 [=====] Running 28 tests from 5 test suites.
6 [-----] Global test environment set-up.
7 [-----] 7 tests from DBShould
8 [ RUN    ] DBShould.getObj_NotExisting_IOPaths2_template
9 [ OK     ] DBShould.getObj_NotExisting_IOPaths2_template (1 ms)
10 [ RUN   ] DBShould.updateObj_NotExisting_IOPaths2
11 [ OK    ] DBShould.updateObj_NotExisting_IOPaths2 (1 ms)
12 [ RUN   ] DBShould.createObj_WithGenerationOfUK_FromUK
13 [ OK    ] DBShould.createObj_WithGenerationOfUK_FromUK (4 ms)
14 [ RUN   ] DBShould.createObj_WithGenerationOfUK_ThreeTimes
15 [ OK    ] DBShould.createObj_WithGenerationOfUK_ThreeTimes (1 ms)
16 [ RUN   ] DBShould.updateObj_Existing_IOPaths2
17 [ OK    ] DBShould.updateObj_Existing_IOPaths2 (1 ms)
18 [ RUN   ] DBShould.delete_Existing_IOPaths1
19 [ OK    ] DBShould.delete_Existing_IOPaths1 (2 ms)
20 [ RUN   ] DBShould.delete_NotExisting_IOPaths2
21 [ OK    ] DBShould.delete_NotExisting_IOPaths2 (1 ms)
22 [-----] 7 tests from DBShould (11 ms total)
23
24 [-----] 1 test from HDLCFrameTests
25 [ RUN   ] HDLCFrameTests.Transceive_XID_DummyScan
26 [ OK    ] HDLCFrameTests.Transceive_XID_DummyScan (0 ms)
27 [-----] 1 test from HDLCFrameTests (0 ms total)
28
29 [-----] 3 tests from UserCommandParserShould
30 [ RUN   ] UserCommandParserShould.Accept_OnInput_StartPooling
31 [ OK    ] UserCommandParserShould.Accept_OnInput_StartPooling (0 ms)
32 [ RUN   ] UserCommandParserShould.RejectUnknownCommand_OnInput_startPooling
33 [ OK    ] UserCommandParserShould.RejectUnknownCommand_OnInput_startPooling (0 ms)
34 [ RUN   ] UserCommandParserShould.Shutdown_OnInput_Shutdown
35 [ OK    ] UserCommandParserShould.Shutdown_OnInput_Shutdown (0 ms)
36 [-----] 3 tests from UserCommandParserShould (0 ms total)
37
38 [-----] 9 tests from HDLCReqFrameBodyTests/HDLCReqFrameBodyTests
39 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/0
40 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/0 (0 ms)
41 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/1
42 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/1 (0 ms)
43 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/2
44 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/2 (0 ms)
45 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/3
46 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/3 (0 ms)
47 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/4
48 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/4 (0 ms)
49 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/5
50 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/5 (0 ms)
51 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/6
52 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/6 (0 ms)
53 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/7
54 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/7 (0 ms)
55 [ RUN   ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/8
56 [ OK    ] HDLCReqFrameBodyTests/HDLCReqFrameBodyTests.BuiltFrame/8 (0 ms)
57 [-----] 9 tests from HDLCReqFrameBodyTests/HDLCReqFrameBodyTests (1 ms total)
58
59 [-----] 8 tests from HDLCFrameInterpreterTests/HDLCFrameInterpreterTests
60 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/0
61 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/0 (0 ms)
62 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/1
63 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/1 (0 ms)
64 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/2
65 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/2 (0 ms)
66 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/3
67 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/3 (0 ms)
68 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/4
69 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/4 (0 ms)
70 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/5
71 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/5 (1 ms)
72 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/6
73 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/6 (0 ms)
74 [ RUN   ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/7
75 [ OK    ] HDLCFrameInterpreterTests/HDLCFrameInterpreterTests.InterpretFrame/7 (0 ms)
76 [-----] 8 tests from HDLCFrameInterpreterTests/HDLCFrameInterpreterTests (1 ms total)
77
78 [-----] Global test environment tear-down
79 [=====] 28 tests from 5 test suites ran. (14 ms total)
80 [ PASSED ] 28 tests.
```

Listing 8.2. Interfejs komendy

8.3 Test modułowe

```

1 [ 2%] Built target gtest
2 [ 4%] Built target gmock
3 [ 9%] Built target gmock_main
4 [100%] Built target KorytkoMag_RetDriver_MT
5 [=====] Running 18 tests from 3 test suites.
6 [-----] Global test environment set-up.
7 [-----] 5 tests from UIAndDatabaseIntegrationShould
8 [ RUN    ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Once
9 [      OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Once (5 ms)
10 [ RUN   ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Twice
11 [      OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByOT_BufferToSend_Twice (2 ms)
12 [ RUN   ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend
13 [      OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend (2 ms)
14 [ RUN   ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_EqualKey
15 [      OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_EqualKey (1 ms)
16 [ RUN   ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_DifferentKey
17 [      OK ] UIAndDatabaseIntegrationShould.Put_IOPaths_ByUK_BufferToSend_Twice_DifferentKey (2 ms)
18 [-----] 5 tests from UIAndDatabaseIntegrationShould (12 ms total)
19
20 [-----] 8 tests from BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar
21 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/0
22 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/0 (2104 ms)
23 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/1
24 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/1 (2 ms)
25 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/2
26 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/2 (2 ms)
27 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/3
28 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/3 (2 ms)
29 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/4
30 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/4 (2 ms)
31 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/5
32 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/5 (4 ms)
33 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/6
34 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/6 (1 ms)
35 [ RUN   ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/7
36 [      OK ] BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar.Execute_Command_Expect_ResultCode/7 (2 ms)
37 [-----] 8 tests from BaseFixtureWithDBAndHDLC/KorytkoMagControllerShouldPar (2119 ms total)
38
39 [-----] 5 tests from BaseFixtureWithDB/UI_Controller_RoundTripHDLC
40 [ RUN   ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/0
41 [      OK ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/0 (2 ms)
42 [ RUN   ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/1
43 [      OK ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/1 (1 ms)
44 [ RUN   ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/2
45 [      OK ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/2 (2 ms)
46 [ RUN   ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/3
47 [      OK ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/3 (8 ms)
48 [ RUN   ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/4
49 [      OK ] BaseFixtureWithDB/UI_Controller_RoundTripHDLC.ExecuteCommandAndExpectSentFrame/4 (2 ms)
50 [-----] 5 tests from BaseFixtureWithDB/UI_Controller_RoundTripHDLC (15 ms total)
51
52 [-----] Global test environment tear-down
53 [=====] 18 tests from 3 test suites ran. (2146 ms total)
54 [ PASSED ] 18 tests.

```

Listing 8.3. Interfejs komendy

9. Wymagania niefunkcjonalne

9.1 Środowisko uruchomieniowe

- System operacyjny Linux
 - Dystrybucje:
 - * Manjaro 17.01;
 - Zależności:
 - * CMake 3.15.4;
 - * Boost C++ Libraries 1.71;
 - * Git -> Pobranie kodu produkcyjnego, GTest, GMock;
 - * GCC 9.2;
- Podłączenie do internetu w celu pobrania repozytorium oraz komplikacji programu uruchomieniowego;
- Komendy wywoływanie w pierwszym oknie konsoli, w celu uruchomienia symulatora urządzenia:
 1. „git clone https://github.com/trunksBT/KorytkoMag_RetSimulator.git --recursive”
 2. „cd KorytkoMag_RetSimulator”
 3. „make .”
 4. „bash runBinary.sh”
- Komendy wywoływanie w drugim oknie konsoli, w celu uruchomienia sterownika urządzenia:
 1. „git clone https://github.com/trunksBT/KorytkoMag_RetDriverSimulator.git --recursive”
 2. „cd KorytkoMag_RetDriverSimulator”
 3. „make .”
 4. „bash runBinary.sh”
 5. Wywołanie komend zgodnie z diagramem sekwencji zatytułowanym „Kalibracja RETa”
 6. Aby zakończyć trzeba wpisać „exit”

10. Plan dalszego rozwoju

10.1 Porażki odnośnie planu początkowego

W celu przesłania wiadomości zapisanej w systemie heksadecymalnym przy pomocy interfejsu USB <-> RS-485 należy dokonać konwersji na postać binarną. Zarówno wiadomość odebrana, jak i wysłana może w trakcie transmisji ulec wybrakowaniu bądź zmianie zawartości. W tym celu ramka HDLC protokołu AISG 2.0 posiada dwa bajty przeznaczone na validację takowej wiadomości a jest to wykonywane dzięki wyliczeniu sumy CRC-16. W związku z tym, że istnieje wiele implementacji algorytmu wyliczania sumy CRC, a każda z nich może różnić się od siebie zarówno: definicją wyrażenia wielomianowego, jego postacią oraz wartością początkową, próbowałem na podstawie „podsłuchanej” wiadomości przy pomocy inżynierii wstępnej zdefiniować brakującą wiedzę, lecz aby tego dokonać potrzebowałem więcej informacji na temat zarówno endianowości jak i uporządkowania bitów, których na moment opracowywania algorytmu nie posiadałem. Pomimo tego, że dla komputera jest to prosta operacja, to weryfikacja przez człowieka wartości sumy CRC dla wiadomości o długości 16-tu bajtów na kartce zajmuje naprawdę dużo czasu. W dodatku powszechnym problemem jest to, że producenci urządzeń linii antenowej pomimo tego, że deklarują pełną implementację protokołu AISG 2.0, to w rzeczywistości okazuje się, że wcale tak nie jest. Chcąc zapoznać się z protokołem komunikacyjnym oraz zrealizować projekt inżynierski podczas którego sprawdę umiejętności testowania, wdrażania wzorców projektowych czy też posługiwania się językiem C++, zamiast walczyć z urządzeniem i jego możliwymi błędami, postanowiłem utworzyć symulator urządzenia, z którym sterownik będzie komunikował się przy pomocy biblioteki ZeroMQ, a samą wartość sumy wyznaczyłem na {0x13, 0x37}. Dzięki zaznajomieniu się z wieloma wzorcami projektowymi jak fabryka, komenda, budowniczy czy nakładanie obostrzeń na program dzięki listowaniu obsługiwanych komend, wspomniany wcześniej symulator urządzenia powstał niemalże za darmo, co uznaję za bardzo duży sukces z dziedziny projektowania architektury oprogramowania. Kolejną zaletą utworzenia symulatora urządzenia jest umożliwienie testów komponentowych realizując regułę czarnej skrzynki, co pozwoli w przyszłości znacznie skrócić czas testowania sterownika pod względem błędów logicznych.

10.1.1 Weryfikacja wiadomości pod względem obecności zarezerwowanych znaków

Z języka angielskiego „Byte stuffing”. Procedura polega na tym, że zarówno dla flagi startu jak i stopu zarezerwowana jest wartość 0x7E. Niestety urządzenie podrzędne może odpowiedzieć na przykład podczas procedur XID negocjacji, wiadomością zawierającą bajt 0x7E. Nieobsłużone takie zjawisko spowoduje, że urządzenie nadziedne zinterpretuje ten bajt jako bajt

stopu, co jest niepożądane[5]. W tym celu standard AISG definiuje procedurę do jakiej należy się zastosować w takiej sytuacji, z którą nie zaznajomiłem się.

10.1.2 Interwały czasowe

Protokół AISG 2.0 definiuje szereg interwałów których obecność można zaobserwować podczas komunikacji z urządzeniem. Jednym z nich jest utrzymanie urządzenia w stanie za-adresowanym. W przypadku kiedy urządzenie podrzędne nie otrzyma jakiekolwiek wiadomości w ciągu 3 minut od jego zaadresowania, przechodzi ono w stan niezaadresowany, po czym ponownie należy zestawić warstwę fizyczną oraz łącza danych wraz z całą procedurą XID negocjacji. Kolejnymi zależnościami czasowymi są odstępy pomiędzy wysłaniem wiadomości a rozpoczęciem nasłuchiwanego na odpowiedź. Problemem mógłby być scenariusz w którym wysyłamy wiadomość do urządzenia podrzędnego, a jest ono na przykład w trakcie procedury kalibracji która może trwać ok 2 minuty. Jako odpowiedź możemy wtedy otrzymać ramkę typu S której stan nazwano RNR czyli z angielskiego „Receiver Not Ready”. W tej sytuacji należy odczekać pewien kwant czasu, a następnie ponówić komunikację.

10.1.3 Ustalanie maski

RetSimulator jest urządzeniem które często posiada wbudowane wyjście RS-485, co oznacza że można podłączyć dwa bądź więcej urządzeń szeregowo. W takiej sytuacji na wiadomość pochodząą z urządzenia nadrzędne zaadresowaną wartością 0xFF, każde z nich wyśle swoją odpowiedź, co niesie ze sobą wiele problemów. Po stronie urządzenia nadrzędne zaobserwujemy to w ten sposób, że nasłuchiwanego w czasie standardowego interwału czasu bajty pochodzące będą naprzemian od dwóch bądź większej liczby urządzeń. Algorytm definiowania maski skutecznie rozwiązuje ten problem. Podczas tej procedury dążymy do ustalenia unikalnego identyfikatora każdego z urządzeń, w celu wysłania wiadomości z zadaniem zaadresowania, na którą odpowie tylko jedno urządzenie. Realizacja powyższego problemu często korzysta z algorytmów przeszukiwania binarnego, w trakcie którego zadaniem jest ustalenie unikalnego identyfikatora urządzenia podrzędnego. Z racji tego, że podczas pracy inżynierskiej przewidywano uruchomienie instancji sterownika dla każdego urządzenia osobno, pominięto implementację wyżej wymienionej logiki.

10.2 Rozbudowa aplikacji o nowe funkcjonalności

Dzięki elastycznej i skalowej architekturze oprogramowania, bez większego problemu powinna być możliwość zaimplementowania rzeczywistej warstwy fizycznej. Pozwoli to na podłączenie prawdziwego urządzenia w celu wykonywania zabiegów testowych na nim. Po zrealizowaniu tego etapu planowane jest dodanie kolejnych komend serwowanych przez protokół AISG 2.0 takich jak: aktualizacja oprogramowania, reset twardy czy miękki oraz usta-

10. Plan dalszego rozwoju

wienie kąta. Wdrożenie do aplikacji mechanizmu wielowątkowości, podłączenie więcej niż jednego urządzenia również będzie osiągalne. Warstwy dynamicznego budowania wiadomości pozwalają również zmniejszyć czas wdrożenia najnowszej wersji protokołu AISG w wersji 3.0.

11. Podsumowanie

Podsumowanko

Bibliografia

- [123] *Control interface for antenna line devices*, (2016)
<https://aisg.org.uk/files/AISG-v2.0.pdf>
- [1] *UTRAN Iu-ant interface: General aspects and principles*, (2012)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125460/11.00.00_60/ts_125460v110000p.pdf
- [2] *UTRAN Iu-ant interface: Layer 1*, (2016)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125461/13.00.00_60/ts_125461v130000p.pdf
- [3] *UTRAN Iu-ant interface: Signalling transport*, (2016)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125462/06.05.01_60/ts_125462v060501p.pdf
- [4] *UTRAN Iu-ant interface: Remote Electrical Tilting (RET) antennas Application Part (RETAP) signalling (2017) 125 463 V7.5.0*, (2007)
https://www.etsi.org/deliver/etsi_ts/125400_125499/125463/07.05.00_60/ts_125463v070500p.pdf
- [5] *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*, (2002)
<https://www.iso.org/standard/37010.html>

Bibliografia

- [1] Beamwidth. <http://www.itcmp.pwr.wroc.pl/~jwach/Techniczny%20EN-PL.htm>. (27.01.2020).
- [2] Elektryczny tilt. <https://www.kathreinusa.com/support/ret-products/>. (27.01.2020).
- [3] High-level data link control. https://pl.wikipedia.org/wiki/High-Level_Data_Link_Control. (25.01.2020).
- [4] High-level data link control. https://en.wikipedia.org/wiki/High-Level_Data_Link_Control. (25.01.2020).
- [5] Radio mobile. https://www.ve2dbe.com/rmonline_s.asp. (27.01.2020).
- [6] RC Martin. *Zwinne wytwarzanie oprogramowania*. 2015.

Spis rysunków

2.1	RET - Miejsce na antenę. (Zdjęcie własne)	10
2.2	RET - podłączonym kablem RS-485 oraz włożoną atrapą anteny. (Zdjęcie własne)	10
2.3	Rysunek przedstawiający kat modyfikowany przy pomocy RET-a. (Opracowanie własne)	11
2.4	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kat elektryczny 0 stopni. (Opracowanie własne przy pomocy programu Radio Mobile)	12
2.5	Pokrycie obszaru sygnałem radiowym dzięki antenie Yagi, kat elektryczny 40 stopni. (Opracowanie własne przy pomocy programu Radio Mobile)	12
3.1	Ramka informacyjna - ewaluacja bajtu kontrolnego. (Opracowanie własne)	17
5.1	Przepływ wiadomości dla wzorca Publikuj-Subskrybuj oraz Żądanie-Odpowiedź. (Opracowanie własne)	25
7.1	Ustanowienie prędkości połączenia wraz z początkowym skanowaniem urządzeń/ (Opracowanie własne)	34
7.2	Żadanie adresacji oraz dodatkowe skanowanie urządzeń. (Opracowanie własne)	36
7.3	Negocjacja rozmiaru okna oraz payloadu ramki informacyjnej oraz ustanowienie normalnego trybu komunikacji. (Opracowanie własne)	38
7.4	Negocjacja pozostałych parametrów HDLC oraz żadanie kalibracji urządzenia. (Opracowanie własne)	39

Spis listingów

5.1	Interfejs komendy	21
5.2	Definicja klasy konkretnej komendy używającej fabryki oraz strategii	22
5.3	Definicja klasy dla obiektu pustego	23
5.4	Przykład użycia obiektu pustego	23
5.5	Plik nagłówkowy dla metody szablonowej walidacji komendy	24
5.6	Metoda szablonowa - Wywołanie metod wirtualnych z poziomu innej metody .	24
5.7	Strategia komunikacji Żadanie-Odpowiedź dla urządzenia nadziednego	26
5.8	Konstruktor zrealizowany podejściem wstrzykiwania zależności	27
5.9	Budowniczy wraz z Fluent API podczas budowania ramki I - Kalibruj	28
5.10	Fabryka budowniczych dla sterownika urządzenia nadziednego	28
8.1	Interfejs komendy	40
8.2	Interfejs komendy	40
8.3	Interfejs komendy	42

Zawartość płyty DVD

1. Praca inżynierska w formacie .tex
2. Praca inżynierska w formacie .tex
3. Kod źródłowy w postaci plików .cpp, .hpp, .txt
4. Adresy repozytorium z zamieszczonym kodem źródłowym
5. Skonfigurowany obraz systemu Manjaro Linux, do użycia w programie VirtualBox
6. Skrypt instalujący potrzebne zależności na systemie Manjaro Linux

Wrocław, dnia 2020-02-15

Wydział Informatyki

Kierunek: informatyka

Paweł Koryciński

(imię i nazwisko studenta)

6749

(nr albumu)

O ŚWIADCZENIE AUTORSKIE

Oświadczam, że niniejszą pracę dyplomową pod tytułem: Symulator sterownika do RET-a implementujący protokół AISG 2.0 napisałem/am samodzielnie. Nie korzystałem/am z pomocy osób trzecich, jak również nie dokonałem/am zapożyczeń z innych prac.

Wszystkie fragmenty pracy takie jak cytaty, rycinę, tabele, programy itp., które nie są mojego autorstwa, zostały odpowiednio zaznaczone i zamieszczono w pracy źródła ich pochodzenia. Treść wydrukowanej pracy dyplomowej jest identyczna z wersją pracy zapisaną na przekazywanym nośniku elektronicznym.

Jednocześnie przyjmuję do wiadomości, że jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdzi popełnienie przeze mnie plagiatu, skutkować to będzie niedopuszczeniem do dalszych czynności w sprawie nadania mi tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz złożenie zawiadomienia o podejrzeniu popełnienia przestępstwa.

.....
(podpis studenta)

Wrocław, dnia 2020-02-15

Wydział Informatyki

Kierunek: informatyka

Paweł Koryciński

.....
(imię i nazwisko studenta)

6749

.....
(nr albumu)

OŚWIADCZENIE O UDOSTĘPNIANU PRACY DYPLOMOWEJ

Tytuł pracy dyplomowej:

Symulator sterownika do RET-a implementujący protokół AISG 2.0

Wyrażam zgodę (nie wyrażam zgody)¹ na udostępnianie mojej pracy dyplomowej.

.....
(podpis studenta)

¹ Niepotrzebne skreślić