UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY

im. Jana i Jędrzeja Śniadeckich w Bydgoszczy

WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI
I ELEKTROTECHNIKI

Interfejs komunikacyjny host-modem ST7580 wraz z GUI

Mateusz Kucaj

Semestr: V
Kierunek: IS
Grupa: IV

Numer albumu: 110953
Rok akademicki: 2019/2020

1. **Otwieranie portu**

   Po naciśnięciu przycisku „Open” pobierana jest wybrana nazwa portu z listy, po czym ustawiane są parametry BitRate, DataBit, StopBit oraz Parity, kolejny krok to ustawienie timeoutów tzn. czas, po którym będzie możliwe odczytanie lub zapisanie bajtu. Ustawione jest to w trybie Read_SEMI_BLOCKING|WRITE_BLOCKING co oznacza, że jeżeli będzie możliwość odczytania minimum 1 bajtu lub gdy minie określona ilość czasu to wykona się read(), a zapisanie bajtu możliwe tylko co określony okres czasu. Następnie RTS ustawiany jest na 0, po czym otwiera się port za pomocą openPort().

```
con.setOnMouseClicked(mouseEvent -> {
    try {
        actualPort = SerialPort.getCommPort(coms.getSelectionModel().getSelectedItem());
        actualPort.setComPortParameters( newBaudRate: 57600,  newDataBits: 8,  newStopBits: 1,  newParity: 0);
        actualPort.setComPortTimeouts( newTimeoutMode: SerialPort.TIMEOUT_READ_SEMI_BLOCKING | TIMEOUT_WRITE_BLOCKING,  newReadTimeout: 0,  newWriteTimeout: 0);
        actualPort.clearRTS();
        actualPort.openPort();
```

2. **Odczytywanie i analizowanie bajtów ramki**

   Jeżeli na otwartym porcie flaga LISTENING_EVENT_DATA_AVAILABLE ustawi się na true to znaczy, że są dostępne dane do odebrania. W pętli, dopóki dostępne dane są w liczbie większej od 0 odczytywany jest jeden bajt i zapisywany do zmiennej actualbyte, który dodawany jest do dwóch StringBuilderów, do jednego w formie heksadecymalnej a do drugiego w formie znaku.

   Analizowanie ramki zaczyna się od Begin i każdy bajt zapisywany jest do odpowiednich zmiennych w zależności od statusu. W przypadku, gdy przyjdzie bajt 0x02 jako pierwszy to ACK wysyłane jest do modemu po odebraniu całej ramki. Odebrane bajty są wyświetlane w zależności od wybranego formatu HEX/ASCII.

```
actualPort.addDataListener(new SerialPortDataListener() {
    @Override
    public int getListeningEvents() { return SerialPort.LISTENING_EVENT_DATA_AVAILABLE; }

    @Override
    public void serialEvent(SerialPortEvent serialPortEvent) {

        if (serialPortEvent.getEventType() != SerialPort.LISTENING_EVENT_DATA_AVAILABLE)
            return;
        i = 0;
        System.out.println("Event detected");
        //  actualPort.setComPortTimeouts(SerialPort.TIMEOUT_SCANNER, 0, 0);
        InputStream input = actualPort.getInputStream();
        try {
            byte[] actualbyte = new byte[1];
            while (input.available() > 0) {
                int numBytes = input.read(actualbyte,  off: 0,  len: 1);
                receivedTextHEX.append(String.format("%02x", actualbyte[0]));
                receivedTextASC.append((char) actualbyte[0]);
                switch (status) {
                    case Begin:
                        if (actualbyte[0] == 0x02) {
                            sendack = true;
                            status = Status.Len;
                            System.out.println(String.format("begin: %02x ", actualbyte[0]));
                        }
                        if (actualbyte[0] == 0x03) {
                            status = Status.Len;
                            System.out.println(String.format("begin: %02x ", actualbyte[0]));
                        } else if (actualbyte[0] == 0x06) {
                            ack = actualbyte[0];
```

```
                ack = actualbyte[0];
                System.out.println(String.format("ACK: %02x ", actualbyte[0]));
            } else if (actualbyte[0] == 0x15) {
                nak = actualbyte[0];
                System.out.println(String.format("NAK: %02x ", actualbyte[0]));
            }
            break;
        case Len:
            len = actualbyte[0];
            len2 = len;
            System.out.println(String.format("len: %02x ", len));
            dataBytes = new byte[len];
            status = Status.Command;
            break;
        case Command:
            commandByte = actualbyte[0];
            System.out.println(String.format("command: %02x ", commandByte));
            status = Status.Data;
            if (len == 0) {
                status = Status.FCS_1;
            }
            break;

        case Data:
            System.out.println("data");
            dataBytes[dataIndex++] = actualbyte[0];
            dataText.append(String.format("%02x", actualbyte[0]));
                status = Status.FCS_1;
                dataIndex = 0;
                break;
            }

            break;
        case FCS_1:
            fcs1Byte = actualbyte[0];
            System.out.println(String.format("fcs1: %02x ", fcs1Byte));
            status = Status.FCS_2;
            break;

        case FCS_2:
            fcs2Byte = actualbyte[0];
            System.out.println(String.format("fcs2: %02x ", fcs2Byte));
            checkFCS(fcs1Byte, fcs2Byte, len2, commandByte, dataBytes);
            if (sendack) {
                sendACK(new byte[]{0x06});
                sendack = false;
            }
            System.out.print("Data: " + dataText);
            System.out.println(String.format(", Command: %02x, FCS_1 : %02x, FCS_2: %02x", commandByte, fcs1Byte, fcs2Byte));
            dataText.delete(0, dataText.length());
            status = Status.Begin;
            break;
    }

}
start = false;
} catch (IOException e) {
e.printStackTrace();
            }
            area.setVisible(true);
            receivedTextHEX.append("\n");
            receivedTextASC.append("\n");
            if (formatStatus == FormatStatus.HEX)
                area.setText(receivedTextHEX.toString().trim());
            else if (formatStatus == FormatStatus.ASCII)
                area.setText(receivedTextASC.toString().trim());
        }
});
```

Sprawdzanie odebranej sumy kontrolnej odbywa się w metodzie checkFCS. Drugi bajt przesuwany jest o 8 bitów w lewo, ponieważ suma kontrolna wysyłana jest z modemu najpierw najmłodszy bajt a potem najstarszy.

```
private void checkFCS(byte fcs1, byte fcs2, int len, byte command, byte[] data) {
    short decodedFCS1 = Short.parseShort(String.format("%02x",fcs1Byte), radix 16);
    short decodedFCS2 = Short.parseShort(String.format("%02x",fcs2Byte), radix 16);
    short moveByte = (short)(decodedFCS2<<8);
    short decodedFCS = (short)(moveByte+decodedFCS1);
    short fcsTest = 0;
    fcsTest += len;
    fcsTest += command;
    for (int i = 0; i < data.length; i++) {
        fcsTest += data[i];
    }
    if (fcs2 != 0x00) {
        if(fcsTest==(decodedFCS))
            System.out.println("Suma kontrolna prawidłowa");
    }
    else if(fcsTest==fcs1)
        System.out.println("Suma kontrolna prawidłowa");

}
```

Wysyłanie ACK – odbywa się bez ustawiania RTS na wysoki stan.

```
    private void sendACK(byte[] mess) {
        actualPort.setDTR();
        actualPort.writeBytes(mess, mess.length);
        actualPort.clearDTR();
    }
```

Zmiana formatu wyświetlania bajtów

```
    hex.setOnMouseClicked(mouseEvent13 -> {
        formatStatus = FormatStatus.HEX;
        area.setText(receivedTextHEX.toString().trim());
    });
    asc.setOnMouseClicked(mouseEvent14 -> {
        formatStatus = FormatStatus.ASCII;
        area.setText(receivedTextASC.toString().trim());
    });
```

Czyszczenie pola odebranych danych

```
clearRx.setOnMouseClicked(mouseEvent -> {
    area.clear();
    receivedTextHEX.delete(0, receivedTextHEX.length());
    receivedTextASC.delete(0, receivedTextASC.length());
});
```

3. **Wysyłanie danych**

Po naciśnięciu przycisku „Send” w zależności od wybranego formatu wpisywanych bajtów są wykonywane inne instrukcje. W przypadku wybranego formatowania HEX dwa wpisane znaki są łączone ze sobą w StringBuilderze po czym stworzony String zostaje zamieniony na jeden bajt i jest dodawany do tablicy bajtów, krok ten powtarzany jest dla wszystkich znaków, które mają zostać wysłane. W przypadku wybranego ASCII wszystkie wpisane znaki od razu są spakowane do ramki i wysyłane za pomocą metody sending().

```
send.setOnMouseClicked(mouseEvent1 -> {
    if(formatTextToSend.getSelectionModel().getSelectedItem()=="HEX") {
        int size = textsend.getText().length() / 2;
        byte[] bytesToSend = new byte[size];
        for (int i = 0; i < textsend.getText().length(); i++) {
            switch (textsend.getText().charAt(i)) {

                default:
                    if (hexBytes) {
                        firstByte = textsend.getText().charAt(i);
                        System.out.println("mam pierwszego bajta " + firstByte);
                        hexBytes = false;
                        boolSecond = true;
                        break;

                    }
                    if (boolSecond) {
                        secondByte = textsend.getText().charAt(i);
                        StringBuilder buildByte = new StringBuilder();
                        buildByte.append(firstByte);
                        buildByte.append(secondByte);
                        try
                        {
                            Byte byteToSend = (byte) Integer.parseInt(buildByte.toString(),  radix: 16);
                            System.out.println(String.format("%02x", byteToSend));
                            bytesToSend[index] = byteToSend;
                            index++;
                            hexBytes = true;
                            boolSecond = false;
                        }
                        catch(NumberFormatException e){
                            System.out.println("Niepoprawną wartość");
                            break;
                        }

                    }
                    break;
            }

        }

        byte[] toSend = packToFrame(bytesToSend);
        index = 0;
        sending(toSend);
    }
    else{
        byte[] toSend = packToFrame(textsend.getText().getBytes());
        sending(toSend);
    }

});
```

Metoda packtoFrame()
Przyjmuje w argumencie dane do wysłania, na początku stworzona zostaje tablica bajtów, w której będą zapisywane wszystkie bajty w odpowiedniej kolejności (tworzenie ramki). Na pierwszym miejscu ustawiany jest bajt 0x02, następnie długość danych użytkownika plus bajt modulacji, kolejny bajt to bajt komendy, w zależności od trybu ustawiany jest jako 0x24 dla PHY lub 0x50 dla DL. Następne miejsce to bajt modulacji, różni się w zależności od wybranej modulacji. Następnie wpisywane do tablicy są dane wpisane przez użytkownika. Ostatnie 2 bajty to bajty sumy kontrolnej. Najpierw wpisany do tablicy jest najmłodszy bajt a później najstarszy. Metoda ta zwraca całą tablice bajtów (ramke).

```
private byte[] packToFrame(byte[] message) {
    byte[] frame = new byte[message.length + 6]; // +6 couse start,len,cc,bajtMOD,fcs,fcs2
    int frameIndex = 4;
    short fcsframe = 0;
    int lenData = message.length;
    frame[0] = 0x02;
    if (bpskB.isSelected() || qpskB.isSelected() || epskB.isSelected()) {
        lenData = lenData+1;
        frame[1] = (byte) (lenData);
        fcsframe += modByte;
        frame[3] = (byte) modByte;
    } else {
        frame[1] = (byte) lenData;
    }
    if (phyb.isSelected()) {
        frame[2] = 0x24;
        fcsframe += 0x24;
    } else if (dlb.isSelected()) {
        frame[2] = 0x50;
        fcsframe += 0x50;
    }
    for (int i = 0; i < message.length; i++) {
        fcsframe += message[i];
        frame[frameIndex++] = message[i];
    }
    fcsframe+=lenData;
       if (fcsframe > 0xff) {
            frame[message.length + 4] = (byte) (fcsframe & 255);
            frame[message.length + 5] = (byte) (fcsframe >> 8);
       }
       else if (fcsframe < 0xff) {
            frame[message.length + 4] = (byte) fcsframe;
            frame[message.length + 5] = 0x00;
       }
       return frame;
  }
```

Wysyłanie do modemu odbywa się za pomocą metody sending().

```
private void sending(byte[] bytes) {
    actualPort.setRTS();
    actualPort.setDTR();
    System.out.println("Sending...");
    try {
        Thread.sleep( millis: 10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    actualPort.writeBytes(bytes, bytes.length);
    actualPort.clearRTS();
    actualPort.clearDTR();
}
```

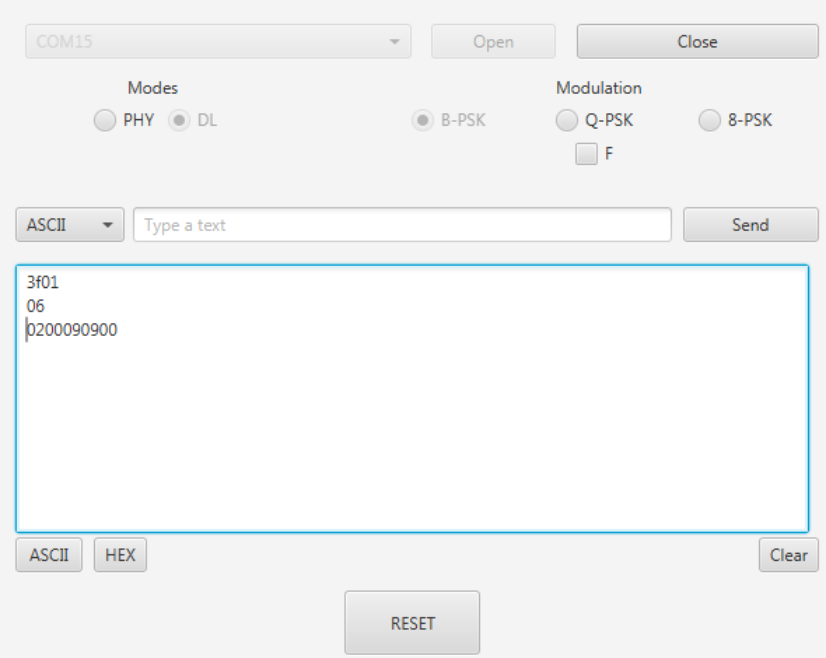
4. Zmiana trybu PHY/DL

Zmiana trybu odbywa się poprzez wysłanie odpowiedniej ramki do modemu

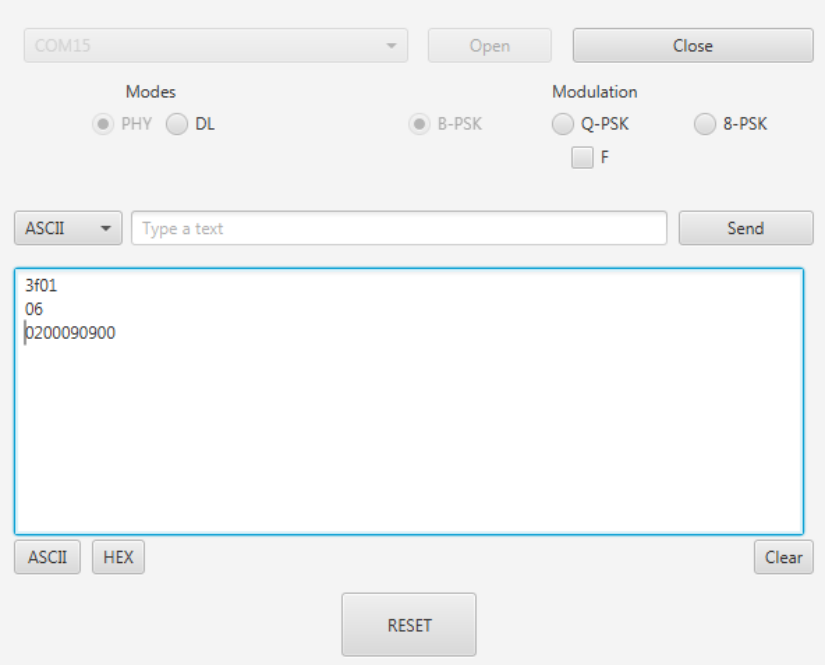
```
        });
        phyb.setOnAction(actionEvent -> {
            dlb.setSelected(false);
            dlb.setDisable(false);
            phyb.setDisable(true);
            sending(phy);
            System.out.println("Tryb PHY włączony");
        });
        dlb.setOnAction(actionEvent -> {
            phyb.setSelected(false);
            dlb.setDisable(true);
            phyb.setDisable(false);
            sending(dl);
            System.out.println("Tryb DL włączony");
        });
        fcb.setOnAction(actionEvent ->
```

```
private final byte[] phy = new byte[]{0x02, 0x02, 0x08, 0x00, 0x10, 0x1A, 0x00};
private final byte[] dl = new byte[]{0x02, 0x02, 0x08, 0x00, 0x11, 0x1B, 0x00};
```

Zmiana na DL



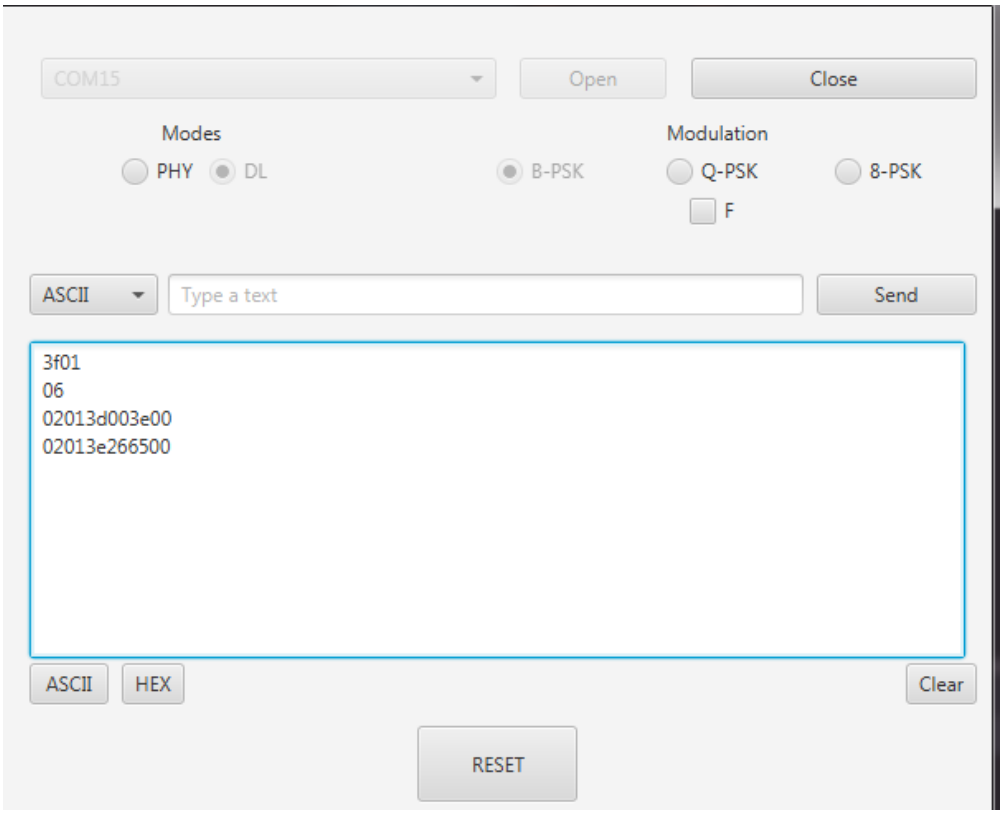
Zmiana na PHY



5. **Reset modemu**

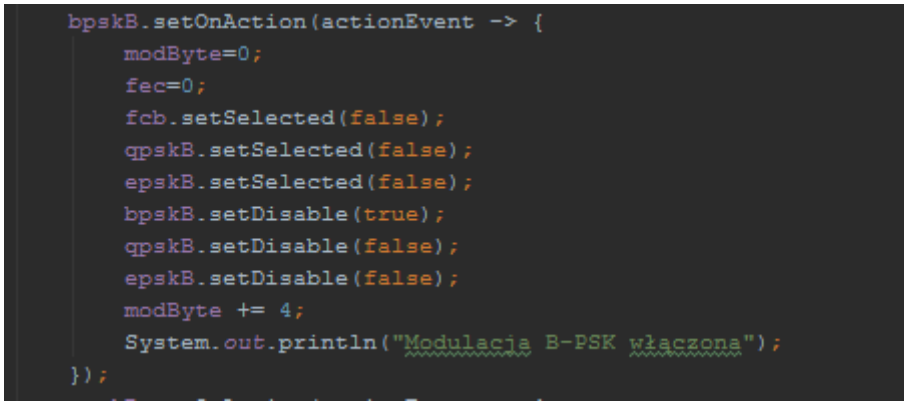
Po naciśnięciu przycisku „RESET” do modemu wysyłana jest odpowiednia ramka

```
        Reset.setOnMouseClicked(mouseEvent12 -> {
            actualPort.setRTS();
            actualPort.setDTR();
            System.out.println("reset...");
            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            actualPort.writeBytes(reset, reset.length);

            actualPort.clearRTS();
            actualPort.clearDTR();

        });
private final byte[] reset = new byte[]{0x02, 0x00, 0x3C, 0x3C, 0x00};
```

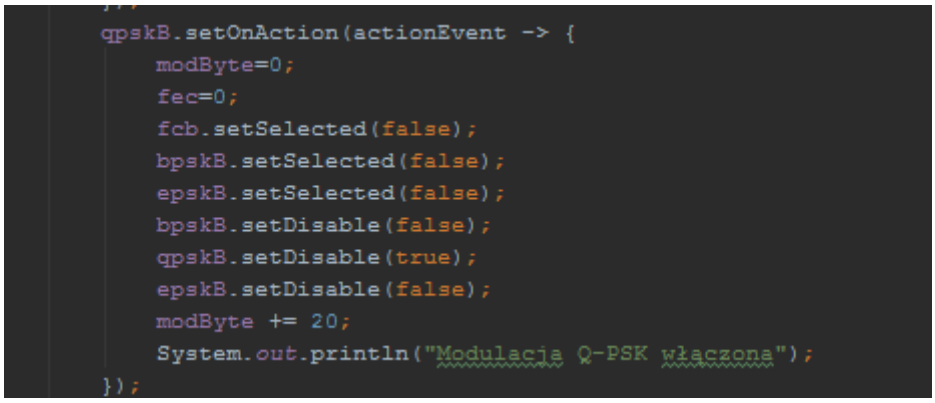


6. Modulacja

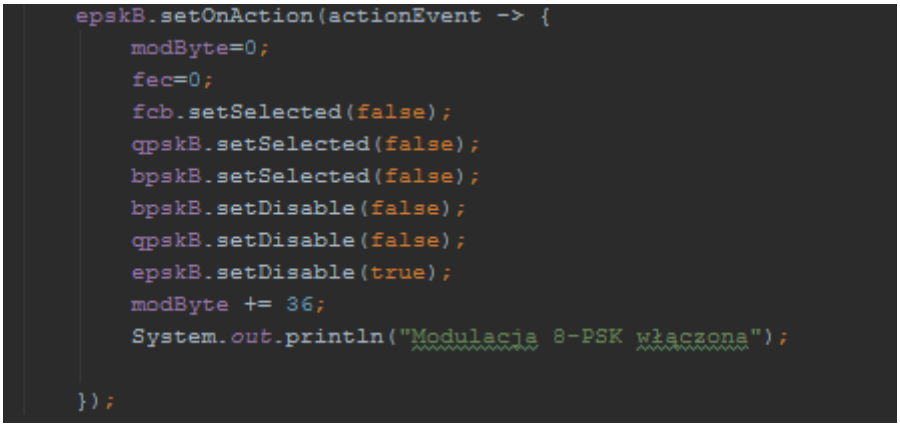
Za modulacje odpowiada jeden bajt, który jest dodawany do ramki przed danymi
Wybranie modulacji BPSK - modByte przyjmuje wartość 4, ponieważ ta modulacja przyjmuje wartość 0 na 5 bicie licząc od najmłodszego, ale ustawiamy na 3 bicie stan wysoki, aby ustawić częstotliwość (ten bit w przypadku każdej modulacji ma wartość 1).



Wybranie modulacji QPSK - modByte przyjmuje wartość 20, ponieważ ta modulacja przyjmuje wartość 1 na 5 bicie licząc od najmłodszego, czyli decymalnie jest to liczba 16 + 4 (częstotliwość)



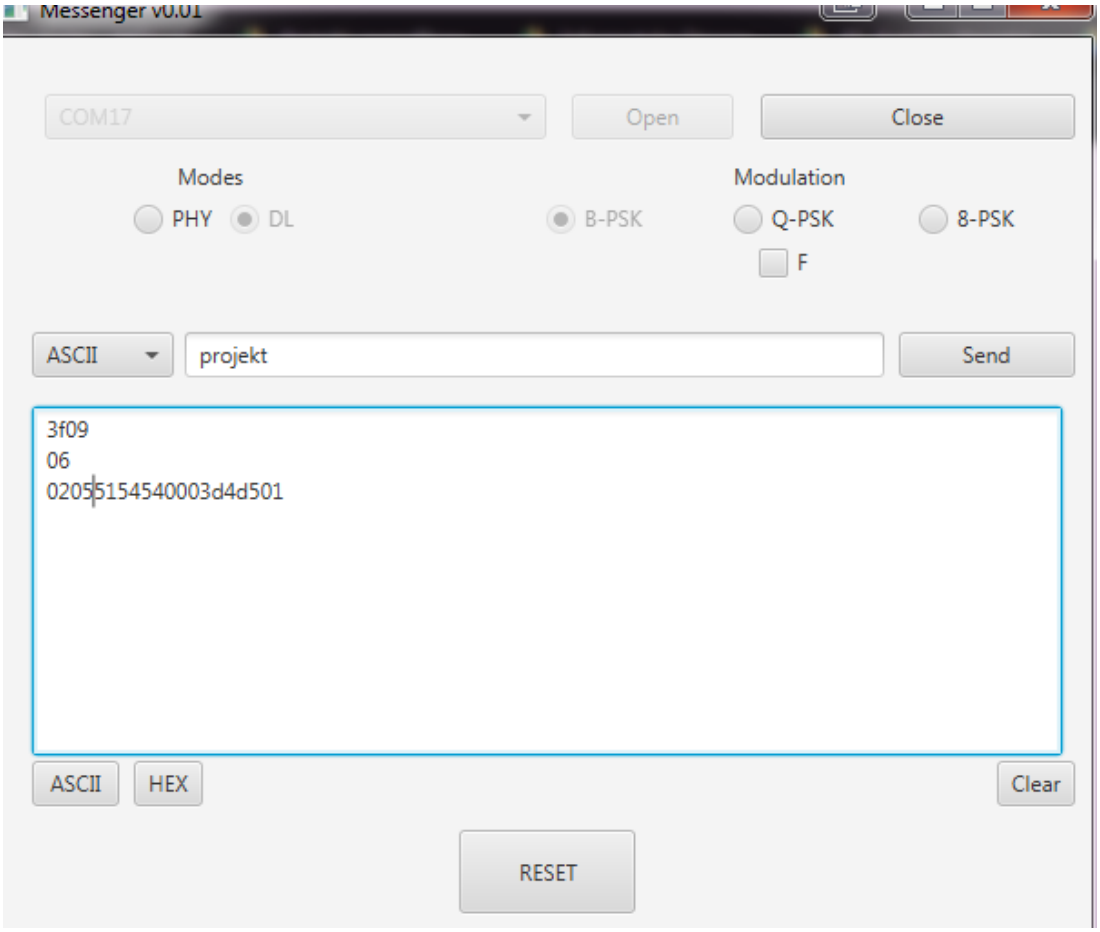
Wybranie modulacji 8-PSK - modByte przyjmuje wartość 36, ponieważ ta modulacja przyjmuje wartość 1 na 6 bicie licząc od najmłodszego, czyli decymalnie jest to liczba 32 + 4 (częstotliwość)



Jeżeli zostanie zaznaczony checkBox „f” to do bajtu modulacji zostaje dodane 64, ponieważ ustawiany jest 7 bit na stan wysoki – zmienia to modulacje na Coded.

```
fcb.setOnAction(actionEvent ->
{
    if (fec == 0) {
        modByte+=64;
        fec=1;
        System.out.println("F checked");
    }
    else if (fec == 1) {
        modByte-=64;
        fec=0;
        System.out.println("F unchecked");
    }
});
```

Wysłanie danych z modulacją B-PSK
Otrzymano DataConfirm



7. Zamykanie portu

```
disc.setOnMouseClicked(mouseEvent -> {
    try {
        actualPort.closePort();
        con.setDisable(false);
        disc.setDisable(true);
        coms.setDisable(false);
        send.setDisable(true);
        phyb.setDisable(true);
        dlb.setDisable(true);
        fcb.setDisable(true);
        bpskB.setDisable(true);
        qpskB.setDisable(true);
        epskB.setDisable(true);
        area.setDisable(true);
        Reset.setDisable(true);
    } catch (Exception e) {
        e.printStackTrace();
    }
});
```

Podsumowanie

Cały program został napisany w języku JAVA przy użyciu biblioteki o nazwie jSerialComm jest to alternatywa dla RxTx i (przestarzałego) Java Communications API, o zwiększonej łatwości użytkowania, ulepszonej obsłudze timeoutów i możliwości otwierania wielu portów jednocześnie. GUI zostało napisane w bibliotece JavaFX przy użyciu zewnętrznego programu SceneBuilder.

Najbardziej czasochłonne okazało się zrozumienie jak działa komunikacja Host-Modem, co oznaczają poszczególne bajty w ramce i jak na nie odpowiadać, szczególnie przy wysyłaniu ramki resetującej modem. Kolejną rzeczą było napisanie kolejki blokującej, problem pojawił się przy wysyłaniu ACK po ramce, której bajt startowy to 0x02 – ACK było wysyłane po ustawieniu RTS a nie należy tego robić.

Docs: https://github.com/Fazecast/jSerialComm/wiki

Project: https://github.com/matkuc003/ProjectPSW.git