C# .NET Aplikacja transmisji szeregowej RS232

1. Wstęp

Celem zadania 2 jest wykorzystanie interfejsów dostępnych w komputerze PC do komunikacji pomiędzy dwoma komputerami. Wykorzystując dowolnie wybrany język programowania należy napisać aplikację symulującą transmisję szeregową zgodną ze standardem RS232. Z założenia, interfejs powinien w sposób interaktywny wykorzystywać do komunikacji z operatorem monitor ekranowy, klawiaturę oraz myszkę.

Do realizacji posłużyliśmy się programem Microszoft *Visual Studio*, *Windows Forms App (.NET Framework)* w języku *C#*, co umożliwia na wykorzystanie techniki graficznego tworzenia interfejsu użytkownika.

2. Założenia

Założyliśmy, że wykorzystanie aplikacji będzie przebiegało w następujący sposób:

1) W oknie nadajnika *Wejście* wpisujemy tekst do nadania w postaci ciągu znaków ASCII.
2) Wpisany tekst jest formatowany do formatu bit startu (log. ‘1’), bity znaku od LSB do MSB i dwa bity stopu (log. ‘1’, ‘1’) i zapisywany do łańcucha, przy czym każdy znak ASCII posiada własny bit startu i bity stopu.
3) Łańcuch binarny zostanie wyświetlony w osobnym polu *Konwersja* na ekranie monitoru.
4) Łańcuch jest przesyłany do odbiornika - okna *Wyjście*, którym jest drugie okno tego samego programu.
5) W odbiorniku łańcuch danych zostaje poddany dekodowaniu, to znaczy, usunięciu ramek i dekompozycji szeregowo równoległej oraz zamianie postaci binarnej na ciąg znaków ASCII.

Ponadto elementem aplikacji będzie dbałość o czystość języka. Postanowiliśmy to zrealizować w postaci słownika „grubiaństw”. Napotkanie w nadawanym tekście „grubego” słowa ze słownika będzie skutkowało zastąpieniem wszystkich jego liter ciągiem gwiazdek ‘*’.

3. Postęp pracy

3.1. Wykonanie interfejsu graficznego

Na początku zbudowaliśmy prosty interfejs graficzny. Do tego posłużyliśmy się rozszerzeniem pliku *.cs* który pozwala na tworzenie aplikacji w sposób graficzny. Dodaliśmy trzy przyciski:

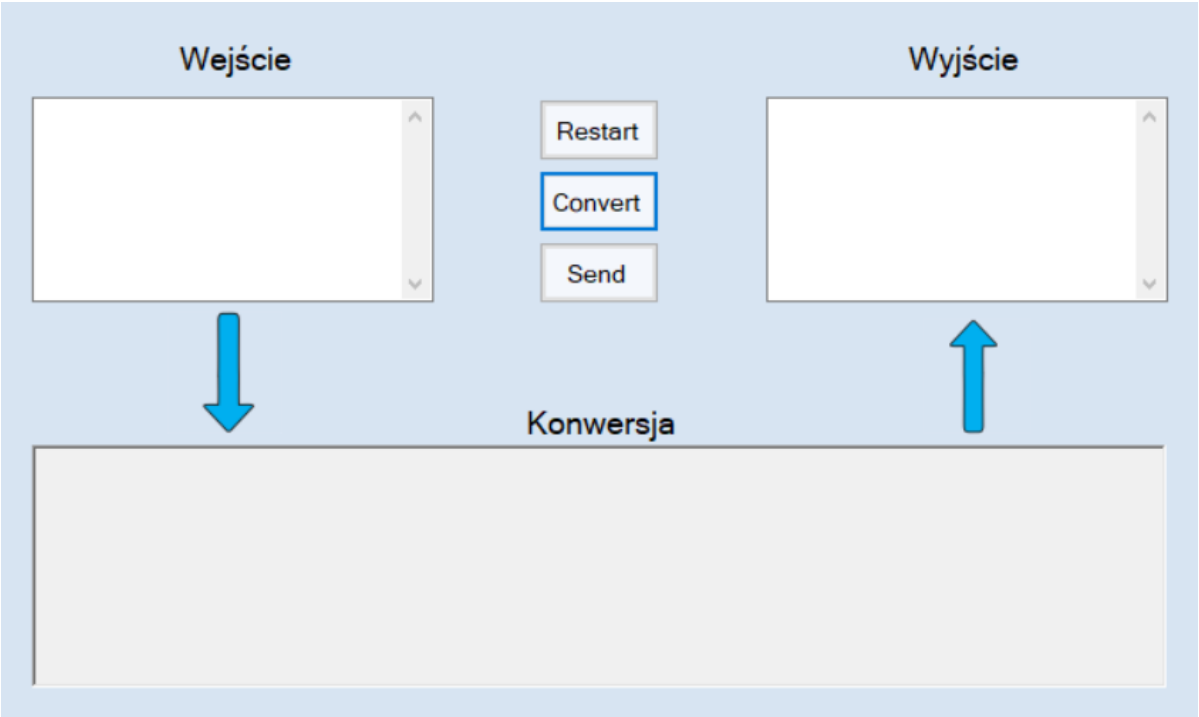
- Restart - przycisk do odświeżania programu;
- Convert - przycisk do poboru danych z wejścia oraz konwersji ciągu znaków do postaci binarnej z uwzględnieniem bitów startu i stopu;
- Send - do dekonwersji danych binarnych do postaci tekstowej oraz wyświetlania wyniku na wyjsciu;

oraz trzy okna tekstowe:

- Wejście (TextBox) - do wpisywania danych;

- Konwersja (RichTextBox) - do wyświetlania bitowej postaci;
- Wyjście (TextBox) - do wyświetlania przesłanego wyniku.

Na danym etapie plik *Form1.cs [Design]* wyglądał następująco:



Rys.1 - Zbudowanie interfejsu graficznego

3.2. Podstawowe funkcje kodu

Aby program działał według założeń (przesyłał i nadawał informację w odpowiedni sposób), stworzyliśmy podstawowe funckje, których zadaniem jest:

1. odczyt danych tekstowych z wejścia;
2. konwersja danych do postaci binarnej;
3. dodawanie bitu startu i bitów stopu;
4. przesłanie binarnego ciągu danych do okienka pośredniego;
5. odczyt danych z okienka pośredniego, dekonwersja danych i wyświetlenieje ich w okienku wyjściowym jako ciąg znaków.

Odczytanie danych z okenka *Wejście* odbywa się po kliknięciu na przycisk ***Convert.*** Po kliknięciu danego przycisku za pomocą funkcji ConvertClick() pobieramy dane z wejścia, tworząc jeden ciąg danych *string input* oraz konwertując go na ciąg *char[] inputchar.*

```
private void ConvertClick(object sender, EventArgs e)
{
    BinDisplay.Text = "";
    OutDisplay.Text = "";

    string input = InDisplay.Text; // to co napisaliśmy w okienku, wpisujemy jednym ciągiem do stringa
    char[] inputchar = InDisplay.Text.ToCharArray(); // tworzymy ciąg charów, który będzie potrzebny
                                                     // przy sprawdzeniu słownika grubiaństw
    sprawdz_slowniczek(inputchar);
    string nowyinput = new string(inputchar); // po sprawdzeniu "grubiaństw" i ewentualnym cenzurowaniu
                                              // otrzymamy inne dane niż te które na początku były na wejściu

    byte[] ASCIIvalues = Encoding.ASCII.GetBytes(nowyinput); // konwertacja ASCII, na wyjściu uzyskujemy w postać DEC
```

Rys.2 - Początek funkcji ConvertClick() pobierający dane z okienka wejściowego i realizujący konwersję danych

wejściowych do postaci binarnej

W kolejnym kroku sprawdzamy, czy w odczytanych danych są “grubieństwa” - za to odpowiada funcja *sprawdz_slowniczek* (więcej o niej jest w punkcie 3.3 Słowniczek “grubiaństw”). W przypadku, jeśli wykryto grubiaństwo, w *inputchar* litery cenzurowanego słowa zostaną zamienione na gwiazdki ‘*’. Po wykonaniu funkcji *sprawdz_slowniczek* tworzymy nowy ciąg danych *string nowyinput* (ponieważ niektóre słowa zostały cenzurowane), który później zostaje poddany kodowaniu do postaci byte za pomocą funcji *Encoding.ASCII.GetBites().*

Dalej dodajemy ramkę do każdej danej - bit startu (log. ‘1’) i bity stopu (log. ‘1’, ‘1’) odpowiednio przed i po binarnej postaci danej (8 bitów). Ostatecznie przesłanie każdej danej to 11 bitów. Aby łatwiej było zobaczyć poprawność działania programu postanowiliśmy wyróżnić bity startu i stopu kolorem czerwonym. Do tego użyliśmy funkcji SelectColor. Pętla *foreach()* konwersuje postać ASCII *byte* do binarnej postać danej, dodaje bity startu i stopu oraz koloruje je na wybrany kolor. Otrzymany ciąg z 11 bitów wyświetla na display’u *Konwersja.*

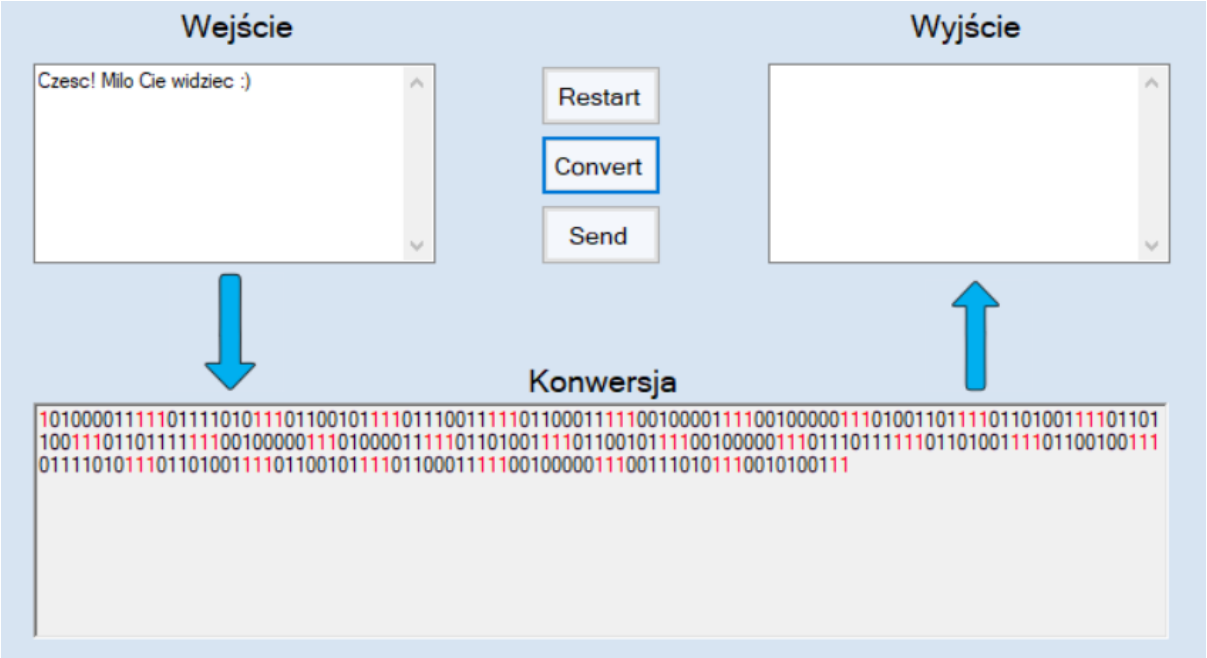
```
int m = 0;
foreach (var value in ASCIIvalues)
{
    resultBinIn = System.Convert.ToString(value, 2).PadLeft(8, '0'); // konwertacja DEC
                                                                      // do BIN pojedynczego znaku
    string start = "1"; // dodajemy z przodu bit startu = 1, i dwa bity stopu = 11
    string stop = "11";

    BinDisplay.AppendText(start);
    BinDisplay.Select(m*11, start.Length + m*11);
    BinDisplay.SelectionColor = Color.Red;

    BinDisplay.AppendText(resultBinIn);
    BinDisplay.Select(start.Length + m * 11, resultBinIn.Length + m * 11);
    BinDisplay.SelectionColor = Color.Black;

    BinDisplay.AppendText(stop);
    BinDisplay.Select(resultBinIn.Length+1 + m * 11, stop.Length + m * 11);
    BinDisplay.SelectionColor = Color.Red;
    m++;
    // Console.WriteLine(result);
}
```

Rys.3 - Część funkcji ConvertClick() odpowiadająca za dodawanie bitów startu i stopu oraz za ich kolorowanie



Rys.4 - Przykład działania przycisku Convert

Na końcu funkcji ConvertClick() pobieramy dane, takie jak długość ciągu binarnego *resultbinLength* oraz ilość znaków na wejściu *ile_znakow*, które przydadzą się w przyszłości podczas dekonwersji danych.

```
resultbinchar = BinDisplay.Text.ToCharArray(); //to co uzyskaliśmy w okienku konwertacji,
                                               // wkładamy do tabeli charów (do nośnika danych)
int resultbinLength = resultbinchar.Length;
ile_znakow = resultbinLength / 11; // mówi ile mamy zadekodowanych znaków w binarnej tabeli charów
```

Rys.5 - Koniec funkcji ConvertClick() - istotne dane do pobrania

Po wyświetleniu ciągu bitów w okienku *Konwersja*, można nacisnąć przycisk ***Send,*** aby przesłać dane do okienka *Wyjście*. Funkcja *SendClick()* służy do dekonwersji ciągu binarnego, to znaczy, do usunięcia ramek i dekompozycji szeregowo równoległej oraz zamiany postaci binarnej na ciąg znaków ASCII. W danej funkcji po kolei usuwamy bity startu i stopu, konwertujemy otrzymany ciąg (8 bitów) do postaci *int decimalValue* (do tego służy pętla *while*), oraz z postaci *integer* do *byte* za pomocą funkcji *BitConverter.GetBytes(decimalValue)*. Otrzymaną postać pozostało jedynie poddać dekonwersji używając funkcji *Encoding.ASCII.GetString()*, aby uzyskać zakodowane znaki, które zostały podane na *Wejściu*.

```
private void SendClick(object sender, EventArgs e)
{
    OutDisplay.Text = "";
    int resultBinOut;

    int zlicz = 1;
    for (int i=0; i<ile_znakow; i++)
    {
        string resultstring = null;

        for (int j=0; j<8;j++)
        {
            resultstring = resultstring + System.Convert.ToString(resultbinchar[j + zlicz]);
        }
        resultBinOut = Int32.Parse(resultstring); // konwertacja binarnego stringa na int

        int decimalValue = 0; // konwertacja binarnego inta na decymalny int
        int base1 = 1;
        while (resultBinOut > 0)...

        byte[] intBytes = BitConverter.GetBytes(decimalValue); // konwertacja dec int na byte dec
        Array.Reverse(intBytes);
        byte[] result = intBytes;

        byte[] bytes = BitConverter.GetBytes(decimalValue);

        var newText = Encoding.ASCII.GetString(bytes); // konwertacja z byte do znaków za pomocą ASCII

        OutDisplay.Text = OutDisplay.Text + newText.ToString();
        zlicz = 8 + zlicz + 3;
    }
}
```

Rys.6 - Funkcja SendClick() odpowiadająca za dekonwersję ciągu binarnego i wyświetlanie znaków w okienku wyjściowym

3.3. Słowniczek “grubiaństw”

Do wykrycia grubiaństwa służy funkcja *sprawdz_slowniczek().* Jej podstawą jest słownik grubiaństw, do którego możemy dodać słowa, cenzurowane przez program oraz pętla *while* (ostatnia linijka). Dana pętla zamieszcza w sobie różnego rodzaju ograniczenia oraz warunki, pozwalające na wykrycie grubieństw ze słowniczka.

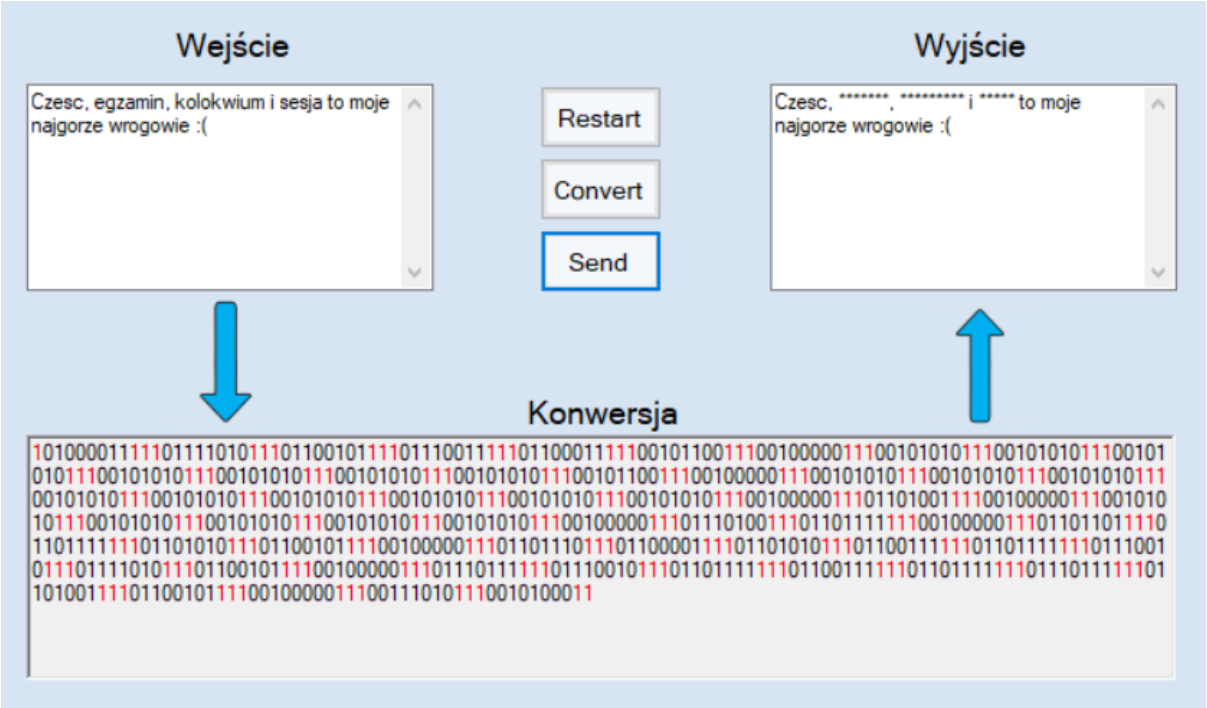
```
private void sprawdz_slowniczek(char[] inputchar)
{
    string[] slowniczekString =
        { "test", "piwo", "sesja", "egzamin", "kolokwium", "zaliczenie" };

    char[][] slowniczekChar = slowniczekString.Select(item => item.ToArray()).ToArray();
    int slowniczekLength = slowniczekString.Length;
    int inputCharLength = inputchar.Length;

    for (int s=0; s < slowniczekLength; s++) // szuka wszystkie grubieństwa zamieszczone w słowniku
    {
        bool znaleziono_slowo = false;
        bool czysto = false;
        bool znaleziono_litere = false;
        int slowkoLength = slowniczekString[s].Length;

        while (znaleziono_slowo == false && czysto == false)...
    }
}
```

Rys.7 -Funkcja sprawdz_slowniczek() pełniąca rolę mechanizmu cenzurującego



Rys.8 - Ostateczne działanie programu - wyświetlanie tekstu na wyjściu po kliknięciu przycisku Send

4. Wyniki pracy

Wyniki pracy można uznać za pomyślne, gdyż aplikacja dobrze pełni swoją funkcję symulowania transmisji szeregowej, zgodnej ze standardem RS232. Aplikacja umożliwia użytkownikowi przesył danych zgodny ze standardem RS232, przy czym jest uwzględniona "czystość języka"; interfejs graficzny oraz sam program jest prosty, czytelny i uwzględniający wszystkie funkcje przyjęte w założeniach.

Można uznać, że zaletą tej aplikacji jest jej prostota budowy oraz łatwość obsługi, natomiast wadą jest to, że przy przesyłaniu większej ilości danych może wystąpić problem z efektywnym wykorzystaniem dostępnej pamięci. Wynika to z tego, że przesył danych jest dokonywany za pomocą stringów, a nie wartości logicznych, co, niestety, nie jest efektywnym rozwiązaniem przy przesłaniu dużej ilości danych. Aby to rozwiązać, przykładowo, należałoby zamienić ciąg stringów na postać tabeli z wartościami logicznymi (tabeli bool), jednak to wymagałoby napisania bardziej zaawansowanego algorytmu przetwarzania takiego typu danych.

Najwięcej trudności doświadczyliśmy podczas dekonwersji danych z ciągu binarnego do ponownej postaci tekstowej. To wymagało pozbycia się ramki (bitów startu i stopu) każdego ze znaków oraz odpowiednie przekształcenie binarnej postaci na znak ASCII. Również nie było łatwe wymyślenie sposobu, w jaki program będzie cenzurował grubiaństwa. Jednak po przeprowadzeniu analizy oraz poszerzeniu wiedzy na dany temat udało nam się zaprojektować w pełni funkcjonalny mechanizm filtracji niecenzuralnych słów.