Dominik Piątkowski

Zespół Szkół im. ks. S. Staszica w Tarnobrzegu

PRZYKŁADY KODOWANIA DANYCH W LINIACH TRANSMISYJNYCH

Streszczenie

Praca pokazuje różne typy kodowania danych, które są wykorzystywane do transmisji danych oraz implementację oprogramowania kodującego.

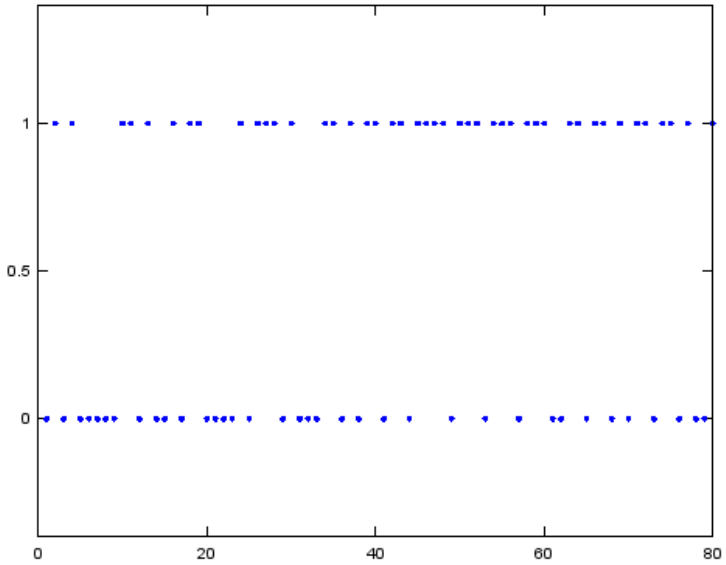
1. WSTĘP

Dane w komputerze są zapisane w systemie binarnym, lecz nie można przesyłać danych zakodowanych w ten sposób. Istnieje szereg komplikacji związanych z transmisją danych, wymagane jest użycie innego typu kodowania. W związku z budową karty sieciowej i zastosowanej tam separacji galwanicznej w postaci kilku transformatorów konieczne jest nadawanie w sposób niewprowadzający składowej stałej, więc oscylujący od dodatnich wartości po ujemne, rozłożone w sposób jak najbardziej zbliżony do równomiernego. Występuje również problem synchronizacji zegara, ponieważ transmisja danych przeprowadzana jest w sposób asynchroniczny, czyli bez użycia linii z sygnałem zegarowym. Wprowadza to ryzyko utraty danych przy transmisji wielu zer lub jedynek pod rząd, gdyż możemy odczytać ich ilość powiększoną lub pomniejszoną o 1, co powoduje kompletną utratę danych od momentu wystąpienia takiego zjawiska. Zegar synchronizuje się przy zmianie stanu w linii transmisyjnej, więc, podsumowując, sposób transmisji musi być taki, aby stan obecny w liniach transmisyjnych zmieniał się możliwie często oraz był równomiernie rozłożony nad i pod zerem, aby nie wprowadzać składowej stałej.

2. KODOWANIE

Do transmisji danych wykorzystuje się następujące sposoby kodowania:
Manchester, NRZ, CMI
2B1Q
4b -> 5b, a następnie MLT-3
8b -> 10b, a następnie PAM-5
AMI
HDB-3

Pierwszym krokiem jest zamiana danych na zera i jedynki, użyjemy do tego kodowania ASCII. Naszą informacją będzie moje nazwisko, tak więc wyraz Piatkowski w kodzie ASCII będzie brzmiał następująco:
01010000 01101001 01100001 01110100 01101011 01101111 01110111 01110011 01101011 01101001. Tekst zawiera 10 znaków, co oznacza, że w kodzie ASCII zajmuje on 80 bitów.



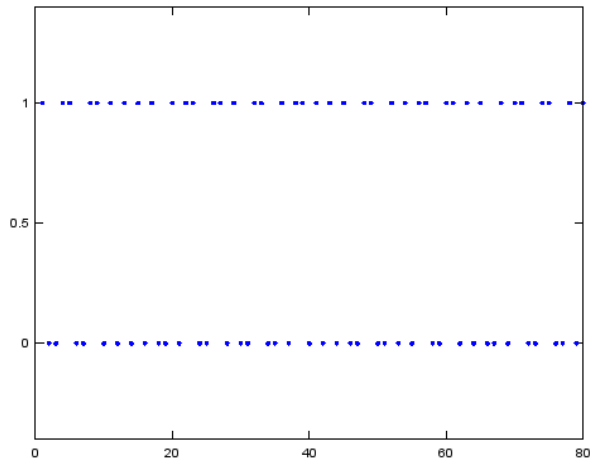
Rysunek 1- Przebieg czasowy danych w kodzie ASCII

2.1 Kodowanie Manchester, NRZ i CMI

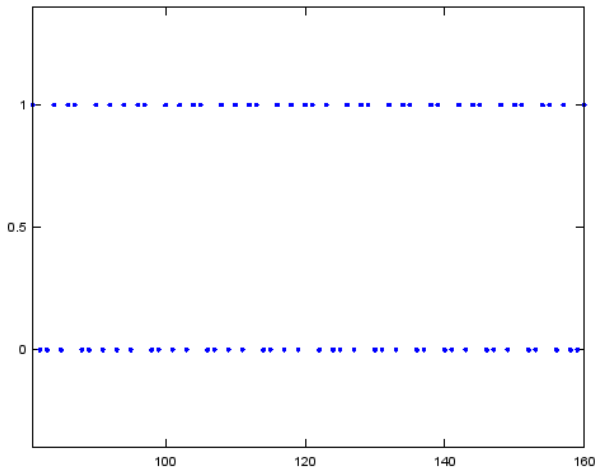
Kodowanie Manchester to kodowanie stosowane w transmisji danych z prędkością 10 Mbit/s.
Stosując się do konwencji IEEE 802.3, wiemy, że logicznemu „0” odpowiada sekwencja 10, natomiast logicznemu „1” – sekwencja 01. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string ManchesterEncoding(string ASCII)
        {
            ASCII = ASCII.Replace("0", "x");
            ASCII = ASCII.Replace("1", "01");
            ASCII = ASCII.Replace("x", "10");
            return ASCII;
        }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1
Otrzymamy dane zakodowane kodem Manchester:
10 01 10 01 10 10 10 10 10 01 01 10 01 10 10 01 10 01 01 10 10 10 10 01 10 01 01 01 10 01 10 10 10 01 01 10 01 10 01 01 10 01 01 10 01 01 01 01 10 01 01 01 10 01 01 01 10 01 01 01 10 10 01 01 10 01 01 10 01 10 01 01 10 01 01 10 01 10 10 01
Kiedy używamy tego kodowania, każdy bit zostaje przedstawiony za pomocą dwóch, co daje współczynnik nadmiarowości równy 50%. Możemy zatem zauważyć, iż aby przesłać 10 Mbit informacji, potrzebne jest przesłanie przez sieć Ethernet ilości dwukrotnie większej, czyli 20 Mbit. Ilość tych informacji narzuca korzystanie ze skrętek stosowanych do przesyłu potrafiących przesłać sygnał o częstotliwości 10 MHz.



Rysunek 2 - Przebieg czasowy danych w kodzie Manchester - część 1

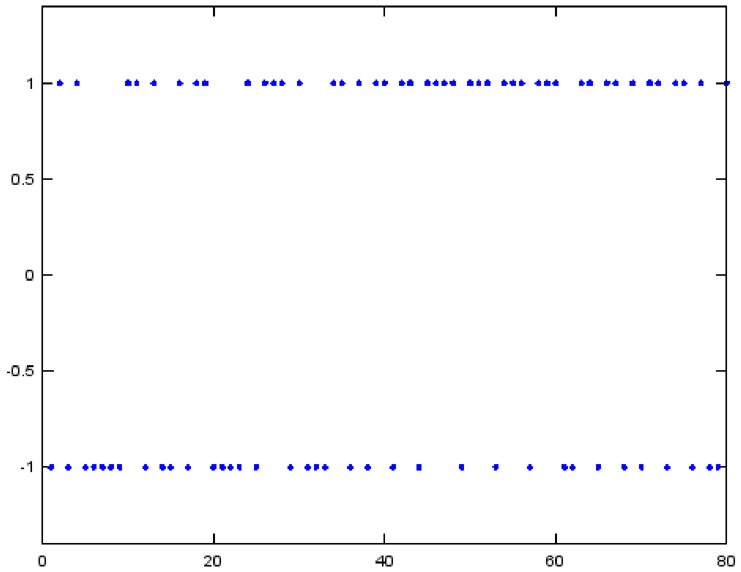


Rysunek 3 - Przebieg czasowy danych w kodzie Manchester- część 2

Kodowanie NRZ (non-return-to-zero) to kodowanie dwustanowe. Za logiczną jedynkę przyjmuje się dodatnie napięcie, natomiast za logiczne zero – ujemne. W tym kodowaniu nie występuje stan 0V na linii transmisyjnej. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string NRZEncoding(string ASCII)
        {
            ASCII = ASCII.Replace("0", "x");
            ASCII = ASCII.Replace("1", "1");
            ASCII = ASCII.Replace("x", "-1");
            return ASCII;
        }
```

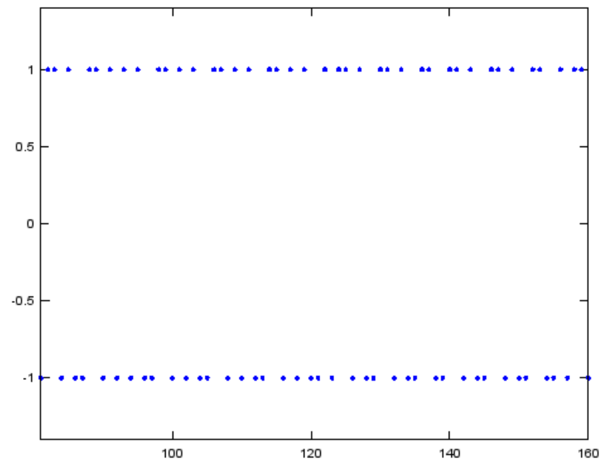
Tym oto sposobem podając następujący ciąg zer i jedynek:
0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1
Otrzymamy dane zakodowane kodem NRZ:
-1 1 -1 1 -1 -1 -1 -1 -1 1 1 -1 1 -1 -1 1 -1 1 1 -1 -1 -1 -1 1 -1 1 1 1 -1 1 -1 -1 -1 1 1 -1 1 -1 1 1 -1 1 1 -1 1 1 1 1 -1 1 1 1 -1 1 1 1 -1 1 1 -1 -1 1 -1 -1 1 1 -1 1 1 -1 1 -1 1 1 -1 1 1 -1 1 -1 -1 1
Kodowanie CMI (Coded Mark Inversion) to kodowanie dwustanowe. Jest ono modyfikacją kodu Manchester i jest wykorzystywane do transmisji danych o prędkości 140 Mbit/s. Kodowanie to kompletnie nie wprowadza składowej stałej. Wartość logicznej jedynki jest przedstawiona jako stan 1 zmieniający się na -1, natomiast logiczne zero jest przedstawione jako stan -1 zmieniający się na 1. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:



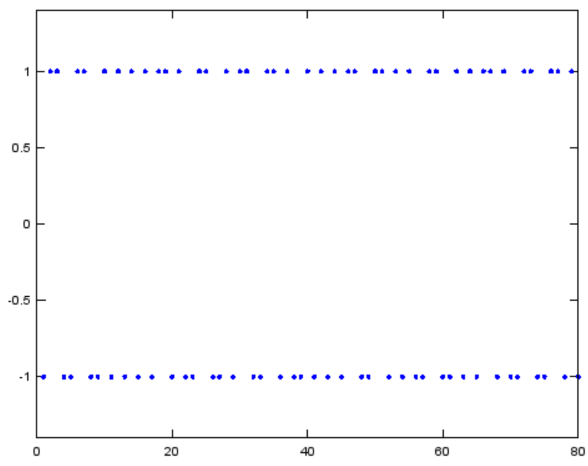
Rysunek 4: Przebieg czasowy danych w kodzie NRZ

```
string CMIEncoding(string ASCII)
    {
        ASCII = ASCII.Replace("0", "x");
        ASCII = ASCII.Replace("1", "1 -1");
        ASCII = ASCII.Replace("x", "-1 1");
        return ASCII;
    }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1
Otrzymamy dane zakodowane kodem CMI:
-1 1 1 -1 -1 1 1 -1 -1 1 -1 1 -1 1 -1 1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1 -1 1 -1 1 -1 1 1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 1 -1 -1 1 -1 1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 -1 1 -1 1 1 -1



Rysunek 5 - Przebieg czasowy danych w kodzie CMI - część 1



Rysunek 6- Przebieg czasowy danych w kodzie CMI - część 2

2.2 Kodowanie 2B1Q

Kodowanie 2B1Q to jedna z metod kodowania stosowana w modemach ISDN. Opiera się na wykorzystaniu 4 stanów:

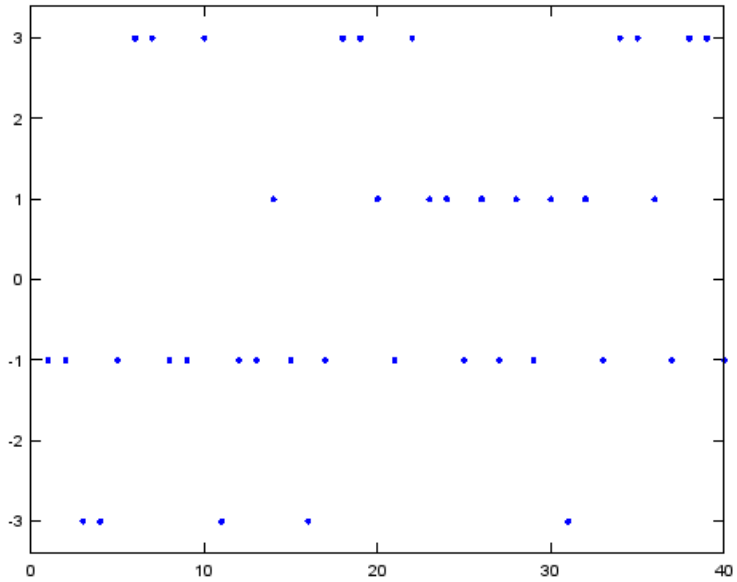
Bity	Poziom napięcia
00	-3V
01	-1V
10	+3V
11	+1V

Tabela 1 - Poziomy napięć w kodowaniu 2B1Q

To kodowanie podwaja efektywność transmisji. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string _2B1QEncoding(string ASCII)
    {
        ASCII = ASCII.Replace("00", "a");
        ASCII = ASCII.Replace("01", "b");
        ASCII = ASCII.Replace("10", "c");
        ASCII = ASCII.Replace("11", "d");

        ASCII = ASCII.Replace("a", "-3");
        ASCII = ASCII.Replace("b", "-1");
        ASCII = ASCII.Replace("c", "3");
        ASCII = ASCII.Replace("d", "1");

        return ASCII;
    }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
01 01 00 00 01 10 10 01 01 10 00 01 01 11 01 00 01 10 10 11 01 10 11 11 01 11 01 11 01 11 00 11 01 10 10 11 01 10 10 01
Otrzymamy dane zakodowane kodem 2B1Q:
-1 -1 -3 -3 -1 3 3 -1 -1 3 -3 -1 -1 1 -1 -3 -1 3 3 1 -1 3 1 1 -1 1 -1 1 -1 1 -3 1 -1 3 3 1 -1 3 3 -1



Rysunek 7- Przebieg czasowy danych w kodzie 2B1Q

2.3 Kodowanie 4b -> 5b oraz MLT-3

Kodowanie 4b -> 5b, a następnie MLT-3 to kodowanie stosowane w transmisji danych z prędkością 100 Mbit/s. Na początku, musimy zakodować dane za pomocą kodowania 4b -> 5b.

Nazwa	4b	5b	Wartość/Opis
0	0000	11110	0
1	0001	01001	1
2	0010	10100	2
3	0011	10101	3
4	0100	01010	4
5	0101	01011	5
6	0110	01110	6
7	0111	01111	7
8	1000	10010	8
9	1001	10011	9
A	1010	10110	A
B	1011	10111	B
C	1100	11010	C
D	1101	11011	D
E	1110	11100	E
F	1111	11101	F
Q	-brak-	00000	Quiet (sygnał utracony)
I	-brak-	11111	Idle
J	-brak-	11000	Start #1
K	-brak-	10001	Start #2
T	-brak-	01101	End
R	-brak-	00111	Reset
S	-brak-	11001	Set
H	-brak-	00100	Halt

Tabela 2 - Tabela kodowania z 4b na 5b.

Konieczne jest pogrupowanie danych po 4 bity i zamiana ich na odpowiedniki z tabeli powyżej. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

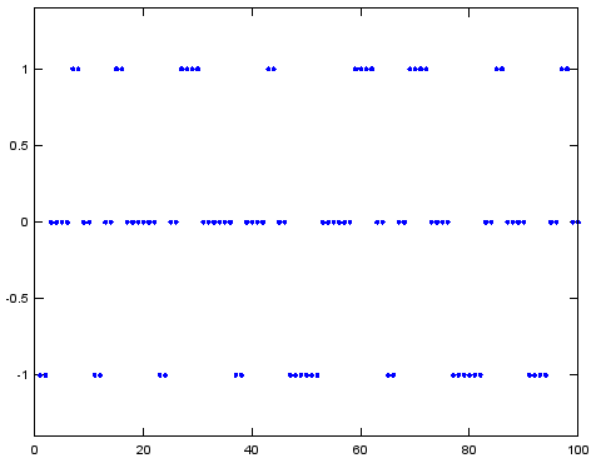
```
string _4b5bEncoding(string ASCII)
        {
            ASCII = ASCII.Replace("0000", "a");
            ASCII = ASCII.Replace("0001", "b");
            ASCII = ASCII.Replace("0010", "c");
            ASCII = ASCII.Replace("0011", "d");
            ASCII = ASCII.Replace("0100", "e");
            ASCII = ASCII.Replace("0101", "f");
            ASCII = ASCII.Replace("0110", "g");
            ASCII = ASCII.Replace("0111", "h");
            ASCII = ASCII.Replace("1000", "i");
            ASCII = ASCII.Replace("1001", "j");
            ASCII = ASCII.Replace("1010", "k");
            ASCII = ASCII.Replace("1011", "l");
            ASCII = ASCII.Replace("1100", "m");
            ASCII = ASCII.Replace("1101", "n");
            ASCII = ASCII.Replace("1110", "o");
            ASCII = ASCII.Replace("1111", "p");
            ASCII = ASCII.Replace("a", "11110");
            ASCII = ASCII.Replace("b", "01001");
            ASCII = ASCII.Replace("c", "10100");
            ASCII = ASCII.Replace("d", "10101");
            ASCII = ASCII.Replace("e", "01010");
            ASCII = ASCII.Replace("f", "01011");
            ASCII = ASCII.Replace("g", "01110");
            ASCII = ASCII.Replace("h", "01111");
            ASCII = ASCII.Replace("i", "10010");
            ASCII = ASCII.Replace("j", "10011");
            ASCII = ASCII.Replace("k", "10110");
            ASCII = ASCII.Replace("l", "10111");
            ASCII = ASCII.Replace("m", "11010");
            ASCII = ASCII.Replace("n", "11011");
            ASCII = ASCII.Replace("o", "11100");
            ASCII = ASCII.Replace("p", "11101");
            return ASCII;
        }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0101 0000 0110 1001 0110 0001 0111 0100 0110 1011 0110 1111 0111 0111 0111 0011 0110 1011 0110 1001
Otrzymamy:
01011 11110 01110 10011 01110 01001 01111 01010 01110 10111 01110 11101 01111 01111 01111 10101 01110 10111 01110 10011. Następnym krokiem jest zakodowanie otrzymanego ciągu kodem MLT-3. Jest to kod trójstanowy, czyli posiada 3 poziomy sygnału: 0, 1 oraz -1. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

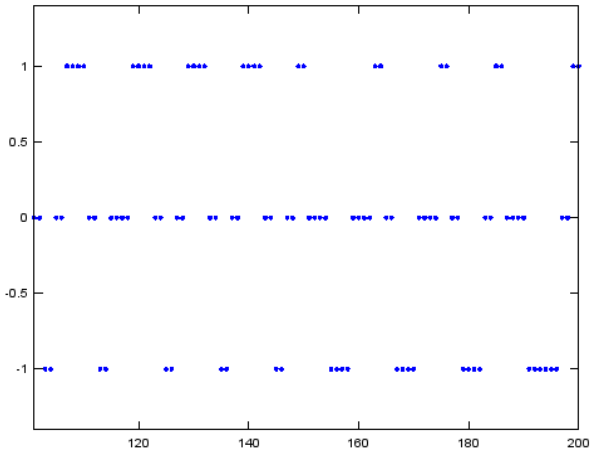
```
string MLT_3_Encoding(string _4b5b)
    {
        string result = "";
        bool rising = true;
        int current = -1;
        for (int i = 0; i < _4b5b.Length; i++)
        {
            if (_4b5b.Substring(i, 1) == "1")
            {
                if (rising)
                {
                    switch (current)
                    {
                        case -1:
                            {
                                current = 0;
                                break;
                            }
                        case 0:
                            {
                                current = 1;
                                break;
                            }
                        case 1:
                            {
                                current = 0;
                                rising = false;
                                break;
                            }
                    }
                }
                else
                {
```

```
                switch (current)
                {
                    case 1:
                        {
                            current = 0;
                            break;
                        }
                    case 0:
                        {
                            current = -1;
                            break;
                        }
                    case -1:
                        {
                            current = 0;
                            rising = true;
                            break;
                        }
                }
            }
        }

        result += current;
        result += " ";
    }
    return result;
}
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0101111110011101001101110010010111101010011101011101110111010111011110111101010111010111011101010011
Otrzymamy:
-1 0 0 1 0 -1 0 1 0 0 0 -1 0 1 1 0 0 0 -1 0 0 1 0 -1 -1 -1 0 0 0 1 1 0 -1 0 1 1 0 0 -1 -1 -1 0 1 0 0 -1 -1 0 1 0 0 -1 0 1 1 0 -1 0 0 1 1 0 -1 0 1 1 0 -1 0 1 1 0 -1 0 1 0 0 -1 -1 0 0 1 0 -1 -1 0 0 1 0 -1 -1 0 1 0 0 -1 -1 -1 0 1
Gdy kodujemy dane za pomocą tego kodowania, każdy blok złożony z 4 bitów zostaje przedstawiony za pomocą 5 bitów, co daje współczynnik nadmiarowości równy 25%. Możemy zatem zauważyć, iż aby przesłać 100 Mbit informacji, potrzebne jest przesłanie przez sieć Ethernet ilości o 25% większej większej, czyli 125 Mbit. Ilość tych informacji narzuca korzystanie ze skrętek stosowanych do przesyłu potrafiących przesłać sygnał o częstotliwości 31,25 MHz. Jest on przesyłany na 2 parach skrętek – 1 do transmisji, 1 do nasłuchu.



Rysunek 8 - Przebieg czasowy danych w kodzie MLT-3 – część 1



Rysunek 9 - Przebieg czasowy danych w kodzie MLT-3 – część 2

2.4 Kodowanie 8b -> 10b, a następnie PAM-5

Kodowanie 8b -> 10b, a następnie PAM-5 to kodowanie stosowane w transmisji danych z prędkością 1000 Mbit/s, czyli 1 Gbit/s. Na początku, musimy zakodować dane za pomocą kodowania 8b -> 10b. Aby to uczynić, musimy podzielić nasze dane najpierw na bloki po 8 bitów, a następnie rozbić każdy z nich na 2 kolejne: zawierające 5 bitów i 3 bity. Jest to konieczne, gdyż kodowanie 8b -> 10b możemy w prosty sposób zrealizować, przeprowadzając kodowania 5b -> 6b oraz 3b -> 4b. Do realizacji powyższego zadania posłużymy się poniższą tabelą (użyjemy RD = 1):

Nazwa	5b	6b	Nazwa	3b	4b
D.00	00000	011000	D.x.0	000	0100
D.01	10000	100010	D.x.1	100	1001
D.02	01000	010010	D.x.2	010	0101
D.03	11000	110001	D.x.3	110	0011
D.04	00100	001010	D.x.4	001	0010
D.05	10100	101001	D.x.5	101	1010

D.06	01100	011001	D.x.6	011	0110
D.07	11100	000111	D.x.P7	111	0001
D.08	00010	000110	D.20	00101	001011
D.09	10010	100101	D.21	10101	101010
D.10	01010	010101	D.22	01101	011010
D.11	11010	110100	D.23	11101	000101
D.12	00110	001101	D.24	00011	001100
D.13	10110	101100	D.25	10011	100110
D.14	01110	011100	D.26	01011	010110
D.15	11110	101000	D.27	11011	001001
D.16	00001	100100	D.28	00111	001110
D.17	10001	100011	D.29	10111	010001
D.18	01001	010011	D.30	01111	100001
D.19	11001	110010	D.31	11111	010100

Tabela 3 - Tabela kodowania z 5b na 6b oraz 3b na 4b

Konieczne jest pogrupowanie danych w bloki naprzemiennie zawierające 5 i 3 bity i zamiana ich na odpowiedniki z tabeli powyżej. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string _8b10bEncoding(string ASCII)
        {
            ASCII = ASCII.Replace("00000", "a");
            ASCII = ASCII.Replace("10000", "b");
            ASCII = ASCII.Replace("01000", "c");
            ASCII = ASCII.Replace("11000", "d");
            ASCII = ASCII.Replace("00100", "e");
            ASCII = ASCII.Replace("10100", "f");
            ASCII = ASCII.Replace("01100", "g");
            ASCII = ASCII.Replace("11100", "h");
            ASCII = ASCII.Replace("00010", "i");
            ASCII = ASCII.Replace("10010", "j");
            ASCII = ASCII.Replace("01010", "k");
            ASCII = ASCII.Replace("11010", "l");
            ASCII = ASCII.Replace("00110", "m");
            ASCII = ASCII.Replace("10110", "n");
            ASCII = ASCII.Replace("01110", "o");
            ASCII = ASCII.Replace("11110", "p");
            ASCII = ASCII.Replace("00001", "q");
            ASCII = ASCII.Replace("10001", "r");
            ASCII = ASCII.Replace("01001", "s");
            ASCII = ASCII.Replace("11001", "t");
            ASCII = ASCII.Replace("00101", "u");
            ASCII = ASCII.Replace("10101", "v");
            ASCII = ASCII.Replace("01101", "w");
            ASCII = ASCII.Replace("11101", "x");
            ASCII = ASCII.Replace("00011", "y");
            ASCII = ASCII.Replace("10011", "z");
            ASCII = ASCII.Replace("01011", "A");
            ASCII = ASCII.Replace("11011", "B");
            ASCII = ASCII.Replace("00111", "C");
            ASCII = ASCII.Replace("10111", "D");
            ASCII = ASCII.Replace("01111", "E");
            ASCII = ASCII.Replace("11111", "F");
            ASCII = ASCII.Replace("000", "G");
            ASCII = ASCII.Replace("100", "H");
            ASCII = ASCII.Replace("010", "I");
            ASCII = ASCII.Replace("110", "J");
            ASCII = ASCII.Replace("001", "K");
            ASCII = ASCII.Replace("101", "L");
            ASCII = ASCII.Replace("011", "M");
            ASCII = ASCII.Replace("111", "N");
            ASCII = ASCII.Replace("a", "011000");
            ASCII = ASCII.Replace("b", "100010");
            ASCII = ASCII.Replace("c", "010010");
            ASCII = ASCII.Replace("d", "110001");
            ASCII = ASCII.Replace("e", "001010");
            ASCII = ASCII.Replace("f", "101001");
            ASCII = ASCII.Replace("g", "011001");
            ASCII = ASCII.Replace("h", "000111");
            ASCII = ASCII.Replace("i", "000110");
            ASCII = ASCII.Replace("j", "100101");
            ASCII = ASCII.Replace("k", "010101");
            ASCII = ASCII.Replace("l", "110100");
            ASCII = ASCII.Replace("m", "001101");
            ASCII = ASCII.Replace("n", "101100");
            ASCII = ASCII.Replace("o", "011100");
            ASCII = ASCII.Replace("p", "101000");
            ASCII = ASCII.Replace("q", "100100");
            ASCII = ASCII.Replace("r", "100011");
            ASCII = ASCII.Replace("s", "010011");
            ASCII = ASCII.Replace("t", "110010");
```

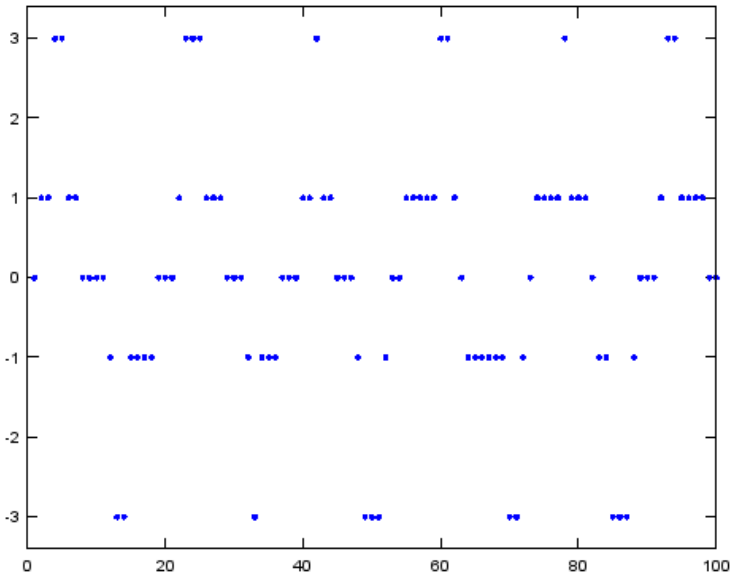
```
            ASCII = ASCII.Replace("u", "001011");
            ASCII = ASCII.Replace("v", "101010");
            ASCII = ASCII.Replace("w", "011010");
            ASCII = ASCII.Replace("x", "000101");
            ASCII = ASCII.Replace("y", "001100");
            ASCII = ASCII.Replace("z", "100110");
            ASCII = ASCII.Replace("A", "010110");
            ASCII = ASCII.Replace("B", "001001");
            ASCII = ASCII.Replace("C", "001110");
            ASCII = ASCII.Replace("D", "010001");
            ASCII = ASCII.Replace("E", "100001");
            ASCII = ASCII.Replace("F", "010100");
            ASCII = ASCII.Replace("G", "0100");
            ASCII = ASCII.Replace("H", "1001");
            ASCII = ASCII.Replace("I", "0101");
            ASCII = ASCII.Replace("J", "0011");
            ASCII = ASCII.Replace("K", "0010");
            ASCII = ASCII.Replace("L", "1010");
            ASCII = ASCII.Replace("M", "0110");
            ASCII = ASCII.Replace("N", "0001");

            return ASCII;
        }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
01010 000 01101 001 01100 001 01110 100 01101 011 01101 111 01110 111 01110 011 01101 011 01101 001
Otrzymamy:
010101 0100 011010 0010 011001 0010 011100 1001 011010 0110 011010 0001 011100 0001 011100 0110 011010 0110 011010 0010. Powyższy otrzymany ciąg podajemy do kolejnego programu napisanego w C#:

```
string PAM_5_Encoding(string _8b10b)
    {
        string result = "";
        bool rising = true;
        int current = 0;
        for (int i = 0; i < _8b10b.Length; i++)
        {
            if (_8b10b.Substring(i, 1) == "1")
            {
                if (rising)
                {
                    switch (current)
                    {
                        case -3:
                            {
                                current = -1;
                                break;
                            }
                        case -1:
                            {
                                current = 0;
                                break;
                            }
                        case 0:
                            {
                                current = 1;
                                break;
                            }
                        case 1:
                            {
                                current = 3;
                                break;
                            }
                        case 3:
                            {
                                current = 1;
                                rising = false;
                                break;
                            }
                    }
                }
                else
                {
                    switch (current)
                    {
                        case 3:
                            {
                                current = 1;
                                break;
                            }
                        case 1:
```

```
                        {
                            current = 0;
                            break;
                        }
                    case 0:
                        {
                            current = -1;
                            break;
                        }
                    case -1:
                        {
                            current = -3;
                            break;
                        }
                    case -3:
                        {
                            current = -1;
                            rising = true;
                            break;
                        }
                }
            }
        }
        result += current;
        result += " ";
    }
    return result;
}
```

Po czym otrzymujemy:
0 1 1 3 3 1 1 0 0 0 0 -1 -3 -3 -1 -1 -1 -1 0 0 0 1 3 3 3 1 1 1 0 0 0 -1 -3 -1 -1 -1 0 0 0 1 1 3 1 1 0 0 0 -1 -3 -3 -3 -1 0 0 1 1 1 1 1 3 3 1 0 -1 -1 -1 -1 -1 -1 -3 -3 -1 0 1 1 1 1 3 1 1 1 0 -1 -1 -3 -3 -3 -1 0 0 0 1 3 3 1 1 1 1 0 0



Rysunek 10- Przebieg czasowy danych w kodzie PAM-5

Gdy kodujemy dane za pomocą tego kodowania, każdy blok złożony z 8 bitów zostaje przedstawiony za pomocą 10 bitów, co daje współczynnik nadmiarowości równy 25%. Możemy zatem zauważyć, iż aby przesłać 1000 Mbit informacji, potrzebne jest przesłanie przez sieć Ethernet ilości o 25% większej większej, czyli 1250 Mbit. Ilość tych informacji narzuca korzystanie ze skrętek stosowanych do przesyłu potrafiących przesłać sygnał o częstotliwości 78,125 MHz. Jest on przesyłany na 4 parach skrętek – 2 do transmisji, 2 do nasłuchu.

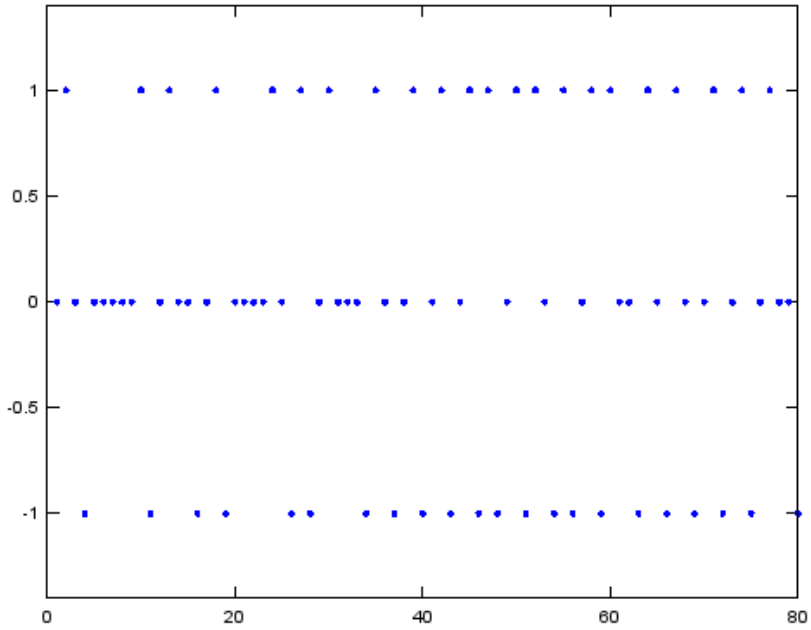
2.5 Kodowanie AMI

Kodowanie AMI to kodowanie używane w modemach ISDN.
Zero logiczne jest zerem w tym kodowaniu, natomiast jedynka logiczna to jeden spośród 2 stanów: 1 oraz -1, występujących naprzemiennie. Posiada jednak wadę w postaci możliwego zjawiska rozsynchronizowania zegara z powodu „długiego zera”.
Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string AMIEncoding(string ASCII)
        {
            int direction = 1;
            string result = "";
            foreach (char c in ASCII)
            {
                if (c == '1')
```

```
                {
                    result += direction;
                    result += " ";
                    if (direction == 1) direction = -1;
                    else direction = 1;
                }
                else
                {
                    result += "0";
                    result += " ";
                }
            }
            return result;
        }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0101000001101001011000010111010001101011011011110111011101110011011010110110100 1
Otrzymamy:
0 1 0 -1 0 0 0 0 0 1 -1 0 1 0 0 -1 0 1 -1 0 0 0 0 1 0 -1 1 -1 0 1 0 0 0 -1 1 0 -1 0 1 -1 0 1 -1 0 1 -1 1 -1 0 1 -1 1 0 -1 1 -1 0 1 -1 1 0 0 -1 1 0 -1 1 0 -1 0 1 -1 0 1 -1 0 1 0 0 -1



Rysunek 11- Przebieg czasowy danych w kodzie AMI

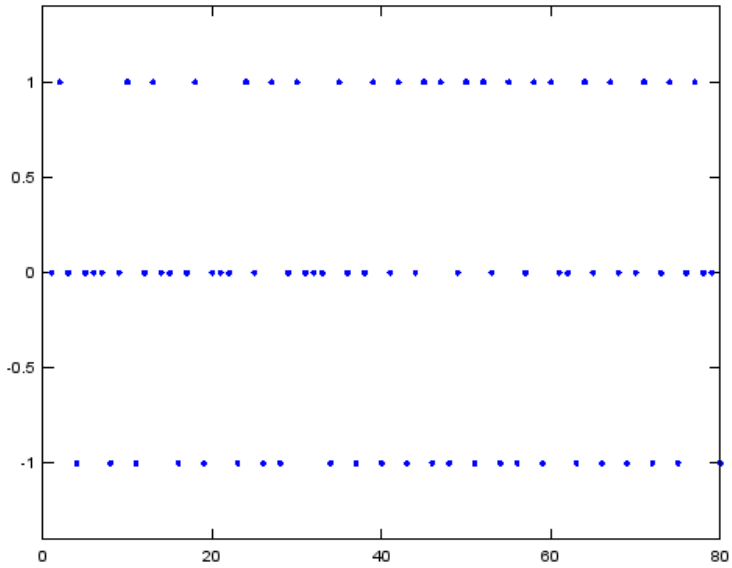
2.6 Kodowanie HDB-3

Kodowanie HDB-3 to zmodyfikowany kod AMI, który przy występowaniu 4 zer logicznych ustawia odpowiedni, inny stan (1 lub -1, w zależności od kilku czynników), aby zachować synchronizację zegara. Do przeprowadzenia sprawnego zakodowania możemy użyć prostego programu napisanego w języku C#:

```
string HDB_3_Encoding(string ASCII)
        {
            int direction = 1;
            string result = "";
            int count_zeros = 0;
            foreach (char c in ASCII)
            {
                if (c == '1')
                {
                    result += direction;
                    result += " ";
                    if (direction == 1) direction = -1;
                    else direction = 1;
                    count_zeros = 0;
                }
                else
                {
                    count_zeros++;
                    if (count_zeros == 4)
                    {
                        if (direction == 1)
                        {
                            result += "-1";
                            result += " ";
                        }
```

```
                    else
                    {
                        result += "1";
                        result += " ";
                    }
                    count_zeros = 0;
                }
                else
                {
                    result += "0";
                    result += " ";
                }
            }
        }
        return result;
    }
```

Tym oto sposobem podając następujący ciąg zer i jedynek:
0101000001101001011000010111010001101011011011110111011101110011011010110110 1001
Otrzymamy:
0 1 0 -1 0 0 0 -1 0 1 -1 0 1 0 0 -1 0 1 -1 0 0 0 -1 1 0 -1 1 -1 0 1 0 0 0 -1 1 0 -1 0 1 -1 0 1 -1 0 1 -1 1 -1 0 1 -1 1 0 -1 1 -1 0 1 -1 1 0 0 -1 1 0 -1 1 0 -1 0 1 -1 0 1 -1 0 1 0 0 -1



Rysunek 12- Przebieg czasowy danych w kodzie HDB-3

3. ZAKOŃCZENIE

Podsumowując, możemy stwierdzić, że wraz z postępem techniki pojawiły się nowe sposoby przesyłu danych, jak również ich kodowania. Niektóre są dzisiaj powszechnie stosowane, inne uznaje się za kompletnie archaiczne. Na przykład, kodowanie AMI oraz HDB-3 było tak naprawdę globalnie powszechne w modemach ISDN, których już praktycznie nie używamy.

BIBLIOGRAFIA

[1]] Bogusław Rzeszut, Paweł Kiciński: *Metody kodowania sygnałów sieci*, ZS im. ks. S. Staszica w Tarnobrzegu 2012

DATA CODING EXAMPLES USED IN TRANSMISSION LINE

Summary

This text is about different data encoding types, and shows how to use them with an example and provides reusable C# source code for testing these encodings on own examples