

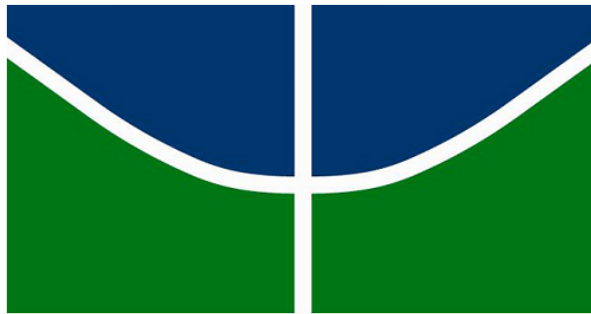
# Simulador dos Protocolos Binário, Manchester e Bipolar

Diogo Luis Teixeira de Macedo Sampaio - 16/0056489

Gabriel Carvalho Moretto - 15/0154917

Pedro Henriques Nogueira - 14/0065032

Teleinformática e Redes 1 - Turma A - 2020/2



Universidade de Brasília  
Departamento de Ciência da Computação  
Professor Geraldo Pereira Rocha Filho



## 2 Implementação

A aplicação é feita pelo terminal, e quando ela é aberta, é pedido do usuário que ele escreva uma mensagem, que representa a aplicação transmissora, depois o programa pede para que o usuário escolha um dos três tipos de codificação (Binário, Manchester e Bipolar), através de uma numeração de zero a dois. Uma vez que o tipo de codificação é escolhido, tanto a mensagem quanto a escolha da codificação é enviado para a parte do programa responsável pela camada de aplicação transmissora que, para efeitos do trabalho, transforma a mensagem de uma string para um vetor de inteiros que corresponde a mensagem escrita em ASCII.

Depois dessa conversão, o vetor é enviado para a parte do código responsável pela camada física transmissora, onde ela passa por um dos protocolos dependendo da escolha que o usuário fez na parte do código da aplicação transmissora (zero para codificação binária, um para codificação manchester e dois para codificação bipolar). Depois da codificação, a mensagem em ASCII é replicada para a parte do código que diz respeito ao meio de comunicação, para que em seguida ela seja decodificada pela parte do código responsável pela camada física receptora, que decodifica de acordo com o método que ela foi transformada, para que por sua vez ela seja convertida de ASCII para uma string na parte da camada de aplicação receptora para que finalmente ela chegue na aplicação da parte receptora no código. O resultado do código consiste na mesma mensagem que foi enviada pela aplicação transmissora, no que qualifica um sucesso, já que os protocolos não modificaram os dados.

### 2.1 Protocolo Binário

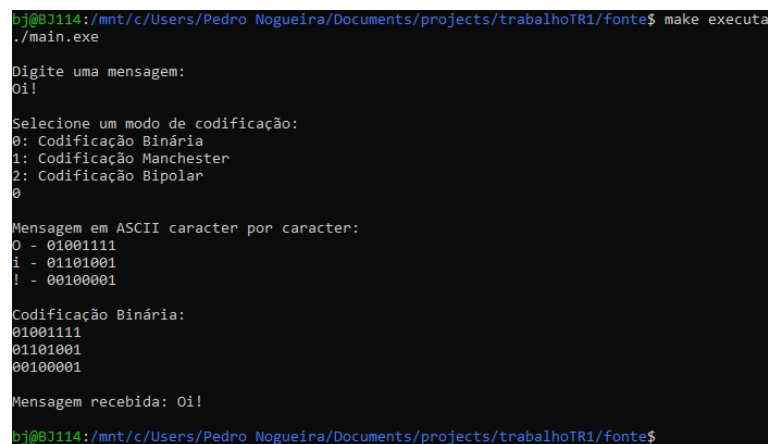
Para a implementação desse protocolo não necessária fazer nenhum tipo de alteração no vetor de inteiros, já que o próprio formato do código em ASCII consegue representar fielmente o resultado do protocolo binário

```
1 vector<int> CamadaFisicaTransmissoraCodificacaoBinaria(  
    vector<int> quadro) {  
3     return quadro;  
}
```

Uma vez que não houve alteração no vetor para transmissão, também não houve necessidade de decodificá-lo na recepção do sinal

```
vector<int> CamadaFisicaReceptoraCodificacaoBinaria(  
2     vector<int> fluxoBrutoDeBits) {  
    return fluxoBrutoDeBits;  
4 }
```

Abaixo temos a tela do terminal quando o protocolo binário é escolhido com a mensagem de exemplo “Oi!”.



```
bj@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$ make executa  
./main.exe  
  
Digite uma mensagem:  
Oi!  
  
Selecione um modo de codificação:  
0: Codificação Binária  
1: Codificação Manchester  
2: Codificação Bipolar  
0  
  
Mensagem em ASCII caracter por caracter:  
O - 01001111  
i - 01101001  
! - 00100001  
  
Codificação Binária:  
01001111  
01101001  
00100001  
  
Mensagem recebida: Oi!  
bj@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$
```

## 2.2 Protocolo Manchester

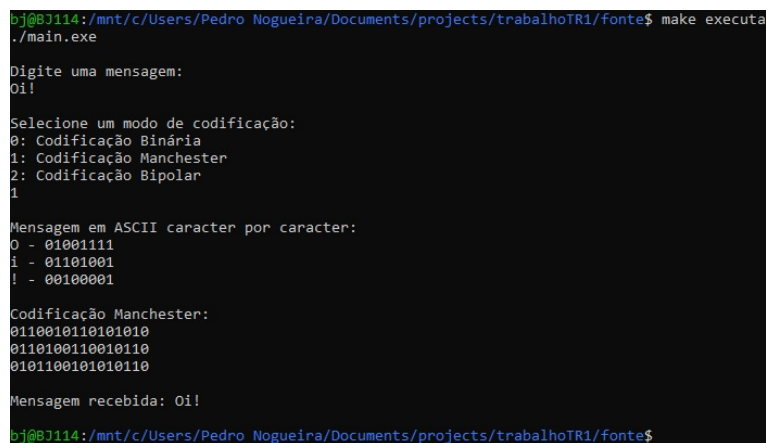
Na implementação, nesse caso usamos de uma iteração que percorre o vetor do sinal para que possamos fazer duas operações em cada bit, que é a operação XOR (ou exclusivo) com os bits zero e um vindas do clock.

```
vector<int> CamadaFisicaTransmissoraCodificacaoManchester(vector<int> quadro) {
2   vector<int> manchesterBits;
   for (unsigned i = 0; i < quadro.size(); i++) {
4       manchesterBits.push_back(quadro.at(i) ^ CLOCK_MANCHESTER.at(0));
       manchesterBits.push_back(quadro.at(i) ^ CLOCK_MANCHESTER.at(1));
6   } // CLOCK_MANCHESTER = [0,1] e so existe para padronizar a codificacao
   return manchesterBits;
8 }
```

Para a decodificação do sinal usamos um método bem simples: iteramos o sinal codificado de dois em dois para recuperar o sinal original, pois foi observado que o sinal codificado possui o sinal original nesse intervalo sem qualquer tipo de perda ou modificação, já que em qualquer caso, a operação XOR de qualquer tipo com o zero resulta no mesmo bit.

```
vector<int> CamadaFisicaReceptoraCodificacaoManchester(
2   vector<int> fluxoBrutoDeBits) {
   vector<int> bitsDecodificados;
4   for (unsigned i = 0; i < fluxoBrutoDeBits.size(); i += 2) { // Sinal original
       bitsDecodificados.push_back(fluxoBrutoDeBits.at(i)); // esta na xor com 0
6   }
   return bitsDecodificados;
8 }
```

Abaixo temos a tela do terminal quando o protocolo manchester é escolhido com a mensagem de exemplo “Oi!”.



```
h3@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$ make executa
./main.exe

Digite uma mensagem:
Oi!

Selecione um modo de codificação:
0: Codificação Binária
1: Codificação Manchester
2: Codificação Bipolar
1

Mensagem em ASCII character por character:
O - 01001111
i - 01101001
! - 00100001

Codificação Manchester:
0110010110101010
0110100110010110
0101100101011010

Mensagem recebida: Oi!

h3@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$
```

## 2.3 Protocolo Bipolar

Neste caso, nós percorremos o vetor de inteiros de modo a achar os bits no sinal em ASCII que sejam iguais a um e mudar o polo no nosso sinal de transmissão de modo que o sinal tenha dois polos, negativo e positivo.

```
vector<int> CamadaFisicaTransmissoraCodificacaoBipolar(vector<int> quadro) {
2   bool umNegativo = false; // Meu bipolar começa com 1 positivo
   for (unsigned i = 0; i < quadro.size(); i++) {
4       if (quadro.at(i) == 1) {
           if (umNegativo) {
6               quadro.at(i) = -1;
           }
       }
   }
```

```

8         umNegativo = !umNegativo;
9     }
10 }
11 return quadro;
12 }

```

Para recuperarmos o nosso sinal codificado na implementação, percorremos o vetor para recuperarmos o modulo de cada bit, no que resulta no sinal original.

```

vector<int> CamadaFisicaReceptoraCodificacaoBipolar(
2     vector<int> fluxoBrutoDeBits) {
3     for (unsigned i = 0; i < fluxoBrutoDeBits.size(); i++) { // So faz modulo
4         fluxoBrutoDeBits.at(i) = fabs(fluxoBrutoDeBits.at(i)); // de tudo
5     }
6     return fluxoBrutoDeBits;
7 }

```

Abaixo temos a tela do terminal quando o protocolo bipolar é escolhido com a mensagem de exemplo “Oi!”.

```

bj@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$ make executa
./main.exe

Digite uma mensagem:
Oi!

Selecione um modo de codificação:
0: Codificação Binária
1: Codificação Manchester
2: Codificação Bipolar
2

Mensagem em ASCII character por character:
O - 01001111
i - 01101001
! - 00100001

Codificação Bipolar:
O 1 0 0 -1 1 -1 1
i -1 1 0 -1 0 0 1
! 0 0 -1 0 0 0 1

Mensagem recebida: Oi!

bj@BJ114:/mnt/c/Users/Pedro Nogueira/Documents/projects/trabalhoTR1/fonte$ _

```

### 3 Membros

A distribuição do trabalho se deu pela seguinte forma: Pedro implementou o código em sua completude enquanto Gabriel e Diogo ficaram responsáveis pelo relatório.

### 4 Conclusão

A implementação satisfaz o esperado: ela recebe uma mensagem e passa pelo código de forma a codificar e decodificar uma mensagem sem modificá-la, no que configura um sucesso. De forma prática, a aplicação em C++ não possui fins práticos, porém o fim didático por conta do trabalho feito teve um impacto extremamente positivo no processo de aprendizado, já que os membros do grupo possuem familiaridade com a área da tecnologia da informação.