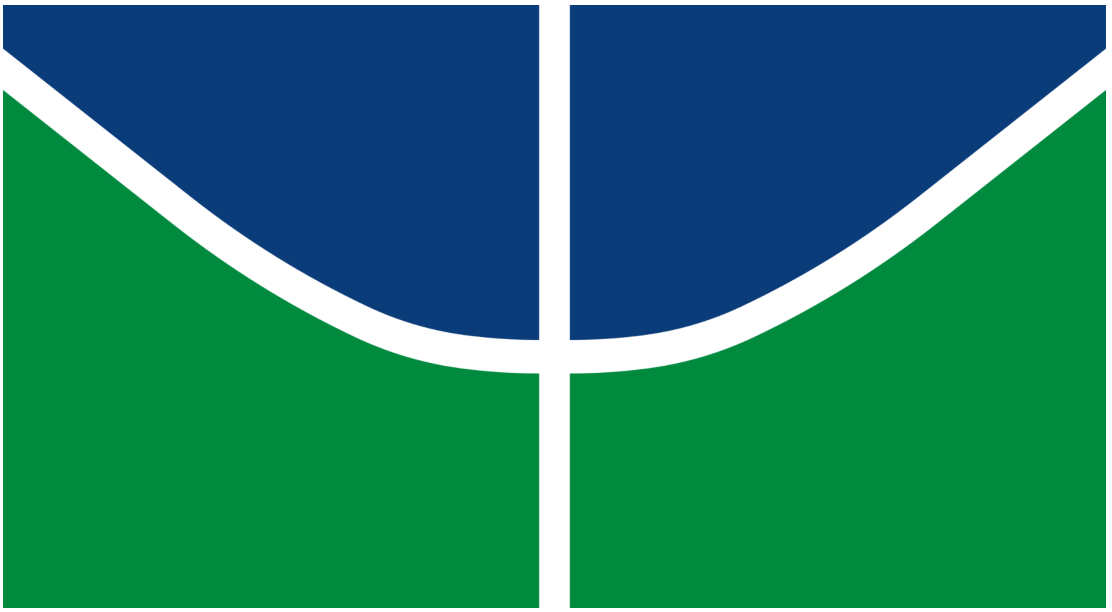
SIMULADOR PROTOCOLOS BINÁRIO, MANCHESTER E BIPOLAR

Eduardo Marques dos Reis - 190012358
João Paulo Marcondes D'Oliveira - 170069923
Kayran Vieira de Oliveira - 190031891



Universidade de Brasília
Teleinformática e Redes 1 - Turma A - 2021/1
Professor Marcelo Antônio Marotta

1. Introdução

Em se tratando de redes de dispositivos, é de extrema importância o uso de protocolos de banda base, pois enquanto diversos dispositivos têm a capacidade de interpretar bits, os cabos transmitem sinais analógicos. Essa troca de informações entre os dispositivos e os cabos não poderia ser efetivada caso esses protocolos não fossem utilizados para converter as mensagens trocadas.

Nesse sentido, para entendermos, praticamente, o funcionamento de alguns desses protocolos, usaremos a linguagem de programação C + + para implementar uma pequena simulação da camada física.

1.1. Codificação Binária

Na Codificação Binária, o dispositivo envia um sinal que é traduzido em um sinal binário, no qual o bit 1 representa a presença de voltagem no sinal, já o bit 0 quer dizer ausência de voltagem.

1.2. Codificação Manchester

Quando queremos transmitir um sinal, é necessário fazermos uma sincronização entre os dispositivos transmissor e receptor, visto que precisamos saber a extensão do sinal tanto para enviar, quanto para receber. Nesse sentido, usamos um ‘clock’, juntamente com o sinal, então para enviar o sinal do ‘clock’ junto com a informação que queremos passar usamos a Codificação Manchester.

Essa codificação junta o sinal do clock com a informação que é enviada usando a operação ‘Ou exclusivo’ (XOR).

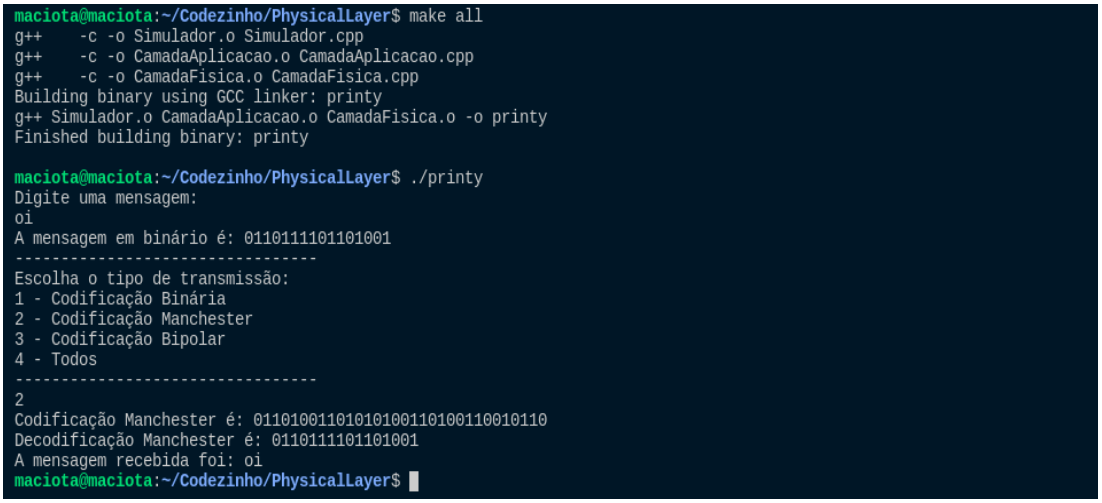
1.3. Codificação Bipolar

Por fim, a Codificação Bipolar utiliza não apenas a ideia de presença e ausência da voltagem no sinal, mas também utiliza voltagem negativa, balanceando, assim, o sinal. Um sinal balanceado não necessita do componente elétrico corrente contínua e isso é vantajoso, pois esse componente faz com que a comunicação com os componentes mais básicos seja dificultada. Assim sendo, a vantagem dessa codificação se dá porque ela trabalha melhor com os componentes eletrônicos do que as outras.

2. Implementação

Para rodar a aplicação é bem simples, primeiramente, escrevemos o comando “**make all**” no terminal, esse comando executa um arquivo makefile que foi criado por acharmos que seria mais prático ter um arquivo que já compile o programa com todos os componentes e com diversas flags, a ter que escrever um comando gigante toda vez que quiséssemos compilar o programa. Após a execução do comando, caso tudo ocorra bem, teremos um arquivo “**printy**” criado, esse arquivo é o nosso executável, para rodar ele, é necessário executar o comando “**./printy**”.

Ao executar o arquivo “printy” aparecerá a opção de digitar uma mensagem, ao digitarmos a mensagem aparecerá a codificação dessa mensagem em binário e a opção para escolhermos entre as codificações Binária, Manchester, Bipolar e todas juntas, então, após escolher a codificação, aparecerá um texto mostrando como fica a mensagem na codificação escolhida, um texto que mostra o resultado após a decodificação escolhida, ou seja, se tivermos escolhido a Codificação Bipolar, nessa etapa será mostrada o resultado depois de decodificarmos e o resultado estará em bits. Por fim, a última etapa consiste na decodificação da mensagem em bits para a mensagem normal, novamente.



2.1. Codificação Binária

A codificação binária foi a mais simples, visto que, ao transformar a string em binário, já tínhamos, basicamente, toda essa codificação pronta. Nesse sentido, foi possível reutilizar a função “StringToBit” que pega a mensagem inicial e, por meio do bitset<8>, transforma em binário convertido para string. Depois, fazemos uma verificação de 0’s e 1’s e, dependendo do valor, guardamos dentro do vetor quadro.

```
vector<int> StringToBit(string mensagem){
 vector<int> quadro;
 string mensagemBit;

 //cout << mensagem << endl;
```

```
 // Para fazer a conversão, utilizamos bitset para conseguir
transformar nossa mensagem em um binário no formato de string.

 for(int i = 0; i<mensagem.size(); i++) {
   mensagemBit += std::bitset<8>(mensagem[i]).to_string();
 }

 // Aqui mostramos o resultado da conversão.
 int tamanho = mensagemBit.length();
 cout << "A mensagem em binário é: " << mensagemBit << endl;

 // Nesse for, estamos comparando cada item da String com seu
valor ASCII. Sendo 0 = 48 e 1 = 49. De acordo com o valor,
colocamos 0 ou 1 no vetor quadro.
 for(int i = 0; i < tamanho; i++){
   if (mensagemBit[i] == 48){
     quadro.push_back(0);
   }

   if (mensagemBit[i] == 49){
     quadro.push_back(1);
   }
 }
 return quadro;
}
```

2.2. Codificação Manchester

Para implementar a Codificação Manchester, foi preciso simular um clock no programa, ou seja, uma uma variável que muda de valor de 0 para 1 e vice-versa, sendo 1 representa a subida do clock, já o 0 a descida. Essa mudança do clock é necessária para montar a codificação, pois essa codificação utiliza os dados recebidos junto com o valor atual do clock e simula uma porta XOR que é muito utilizada em circuitos elétricos. Quando o valor dos dois dados recebidos são iguais (0 e 0 ou 1 e 1), o resultado desse XOR é 0, quando os valores são diferentes (0 e 1 ou 1 e 0) a codificação retorna o valor 1. Outrossim, foi preciso fazer um loop para percorrer os dados recebidos e em cada iteração desse loop eram simuladas a subida e a descida do clock, isso retorna dois valores, um que é pego pelo vetor de dados recebidos e o outro do clock, com os dados obtidos é realizada uma comparação obedecendo a porta XOR. Após todas as iterações, realizadas o resultado dessas comparações retorna o vetor resultante da Codificação Manchester.

```
vector<int>
CamadaFisicaTransmissoraCodificacaoManchester(vector<int> quadro){
 vector<int> manchester;
 int i, clock = 0;
 for(i = 0; i < quadro.size(); i++){
   clock = 0;  // Simulando um clock
   CodManchester(quadro, manchester, clock, i);
   clock = 1;  // Simulando um clock
   CodManchester(quadro, manchester, clock, i);
 }
 return manchester;
}
```

2.3. Codificação Bipolar

Para implementar a Codificação Bipolar, é preciso receber apenas os dados que serão codificados e com o fluxo de bits recebido são estabelecidas algumas regras, os bits 0's tem o valor de 0, os bits 1's são substituídos por mais(+) ou menos(-). Algum valor(v) determinado (o primeiro 1 do vetor recebido recebe o valor de +v, o segundo 1 recebe o valor de -v, e assim sucessivamente). Para implementar o código foi definido um valor constante de v (igual a 1), e foi realizado um loop que percorre todo o vetor de dados recebidos, e para cada iteração era verificado o valor do bit obtido com as regras citadas acima, cada iteração me retorna um valor de 0 ou v, e após todas as iterações se obtém o valor da codificação Bipolar.

```
vector<int> CamadaFisicaTransmissoraCodificacaoBipolar(vector<int>
quadro){
 int i, aux = 0;
 for(i = 0; i < quadro.size(); i++){
   if(quadro[i] == 1 && aux == 0){
     quadro[i] = valorBipolar;
     aux = 1;
   }else if(quadro[i] == 1 && aux == 1){
     quadro[i] = -valorBipolar;
     aux = 0;
   }
 }
 return quadro;
}
```

3. Membros

Cada membro do trabalho fez o seguinte: Eduardo criou tanto as funções de Codificação Manchester e Bipolar quanto as suas decodificações, também foi responsável por refatorar e revisar o código e contribuiu na criação do relatório. João ajudou a identificar e resolver alguns erros que tivemos na parte inicial do programa, principalmente porque ele é quem tinha mais noção da linguagem C++ no grupo, ajudou com o relatório e ajudou bastante na configuração base do projeto. Já o Kayran, criou os códigos para codificar a string inicial para binário e para decodificar de binário para String, criou o repositório e gerenciou o Kanban com as issues, depurou algumas partes do código que deram problema e escreveu o relatório.

4. Conclusão

Tendo implementado todas as codificações, foi possível ter uma noção do funcionamento da Camada Física de forma prática. O fato do grupo ter conseguido codificar e decodificar as mensagens fazendo com que nada se perdesse, foi bastante importante, principalmente, no âmbito do aprendizado.