TR1 - Trabalho de Camada Física

José Vinícius Garreto Costa
viniciusgarreto@gmail.com
18/0123734

Luiz Carlos Schonarth Junior
luiz.junior@aluno.unb.br
19/0055171

Hyago Gabriel Oliveira Figueiredo
hyagogabriel@aluno.unb.br
17/0105067

Abstract

Simulate the operation of the physical link by implementing binary, Manchester and bipolar encodings.

Palavras-chave: *Physical Layer; Manchester; Bipolar; Binary;*

Resumo

Simular o funcionamento do enlace físico por meio da implementação das codificações binárias, de Manchester e bipolar.

Palavras-chave: *Camada Física; Manchester; Bipolar; Binário;*

1. Introdução

Neste relatório será descrito o processo de comunicação de uma pilha de protocolos fictícia, composta apenas pelas camadas de aplicação e física e um meio de comunicação sem modulação/demodulação.

Para contextualizar, faremos uma breve explicação sobre camada física, descrevendo, em suma, o caminho que uma informação passa entre um gerador e um receptor. Daremos atenção as codificações binárias, em especial os protocolos de Manchester, Bipolar (AMI) e Binária (NRZ unipolar).

2. Fundamentação Teórica

2.1. Camada Física

A camada física é a camada mais próxima ao meio físico, que tem como objetivo transcrever os dados em sinais elétricos em meios física cabeáveis, sem fios etc.

Informação aqui será tratada como um trem de bits que será considerado uma mensagem, que podem assumir valores binários de um ou zero. Essa mensagem será encaminhada para um transmissor que ira codifica-la, transformando em símbolos elétricos. Para isso precisamos de um canal de comunicação, que atualmente utilizam propagação de ondas eletromagnéticas como sinais elétricos. Esse sinal sofrerá vários fenômenos físicos, chamados de ruídos, que chegarão a um receptor que irá decodificar o sinal e volta-lo ao seu formato de trem de bits.

Para a conversão dos bits em sinais elétricos, ocorrerá um processamento do sinal. Para processarmos o sinal utilizaremos a série de Fourier, sendo assim o sinal elétrico pode ser analisado como uma composição de somas de senos e cossenos.

Para representar um trem de bits em um sinal, podemos representá-lo em um sinal digital. Utilizando a série de Fourier transformamos o sinal digital em um sinal analógico.

Como o sinal pode ser composto em tempo e frequência, temos que esse canal tem uma capacidade. Como o canal é ruidoso temos alterações no canal de comunicação. Utilizaremos o teorema de Shannon para calcular o ruído, e assim conseguimos calcular a capacidade do canal ruidoso.

Dentre os vários meios de comunicação, temos uma clara divisão, meios guiados e meios não guidados. Os meios guiados são aqueles que são cabeados, quer dizer que a energia é contida nesse meio físico, no geral, gerando menos ruído. Já os meios não guiados temos sinal se propagando no ar, sem fio. Nesse caso o meio é compartilhado, onde temos bem mais pertubações, porém as estruturas são bem mais simples, no geral antenas.

Dependendo do meio temos diferentes velocidades de propagação, que é influenciada pela constante dielétrica do meio, consequentemente influenciando a largura máxima de banda do meio.

Como tanto os canais com fio e sem fio precisam transportar sinais analógicos, para enviar informações digitais precisamos de sinais digitais, e para isso utilizaremos modulação digital. Podemos utilizar canal de banda base

ou canal de banda passante.

O canal de banda base é o processo do receptor para interpretar um sinal. Ele irá determinar o nível do sinal (zero ou um) e fará a detecção de erros. Existem diversos códigos de banda de base que permitem a modularização, falaremos dessas codificações nas sessões a seguir.

2.2. Codificações

2.2.1 Binária

A *codificação Binária*, ou NRZ unipolar, é uma codificação em que o valor lógico 1 (”um”) é representado por um nível DC (convencionalmente positivo) e o valor lógico 0 (”zero”) é representado pela ausência deste nível. É uma das codificações mais simples, sendo também conhecida como *”on-off keying”*, pelo fato da DC estar presente somente quando o *bit* está ”ligado” (valor lógico 1).

Entre as desvantagens do sinal NRZ unipolar é importante ressaltar que longas cadeias de zeros ou uns são possíveis, o que torna a sincronização difícil.

2.2.2 Manchester

A *codificação Manchester* é um código para linha de transmissão que gera um sinal *self-clocking*, o que significa que não é necessário um sinal de *clock* separado para manter sincronismo entre um receptor e seu transmissor.

A codificação é realizada combinando-se o a informação de sincronização – o *clock* – com os dados a serem transmitidos. A frequência do *clock* é 2 vezes maior do que a frequência dos *bits* de dados.

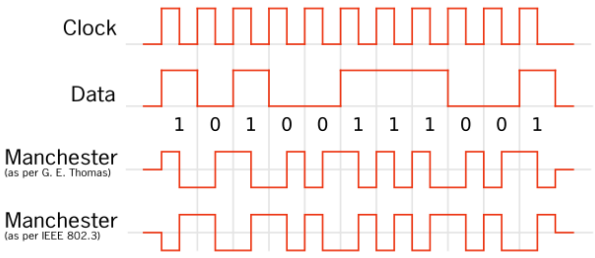

Figura 1. Exemplo de codificação Manchester.

A combinação entre o *clock* e os *bits* de dados é realizada por uma operação XOR *bit* a *bit*, de tal forma que para cada valor binário dos dados, são gerados dois valores no sinal, ou, em outras palavras, para cada valor binário dos dados, é gerada uma transição. Isso implica que um sinal codificado usando o código Manchester exige duas vezes mais largura de banda do que um codificação unipolar (NRZ), por exemplo.

2.2.3 Bipolar

A *codificação bipolar* é classificada como um RZ (retorno a zero) e um exemplo do seu uso é na codificação AMI, que utiliza os 3 níveis de tensão para representar os dados. De forma extensa, a tensão em 0 representa o zero binário, onde a linha é mantida em um nível ”zero” constante, e a tensão +V, ou -V, representa o *bit* 1. Assim, a linha sempre retorna ao nível ”zero”, seja para denotar opcionalmente uma separação de bits, seja para denotar ociosidade da linha.

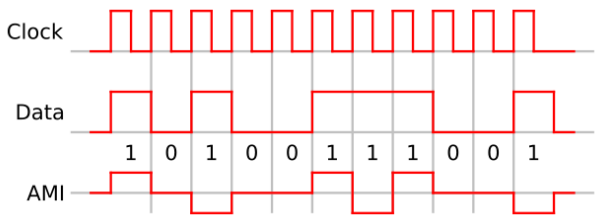

Figura 2. Exemplo de codificação AMI (Bipolar).

Neste tipo de protocolo a leitura é, também, facilitada, pois os *bits* ”0” (zero) são codificados sem pulso e o *bit* ”1” (um) são representados por pulsos positivos ou negativos em alternância, ou seja, nunca haverá dois pulsos positivos ou negativos consecutivos, mesmo existindo pulsos zero entre eles.

Dado as alternâncias de marcas, é possível garantir a ausência de nível DC no sinal final já codificado, porém, se houver a ocorrência de uma sequência longa de zeros, o sinal codificado permanece por muito tempo sem transições na linha, o que torna muito mais difícil a obtenção do *clock* de sincronismo.

Comentado o problema de relógio, é interessante salientar que, por ser imune a inversões de polaridade e possuir uma boa eficiência espectral, este protocolo de codificação ainda pode ser considerado relevante.

3. Implementação do simulador

O simulador foi escrito na linguagem de programação C++ e implementa uma pilha de protocolos fictícia, sendo composta apenas pelas camadas físicas e de aplicação.

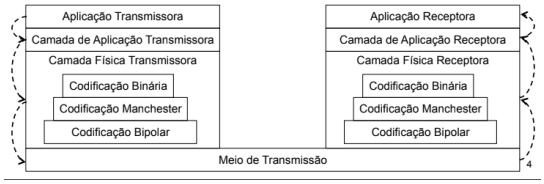

Figura 3. Pilha de protocolos implementada.

Neste cenário fictício, a comunicação é estabelecida entre um *transmissor* e um *receptor*. O transmissor é composto pela camada de aplicação transmissora, e a camada

física transmissora. O receptor é construído de forma recíproca. O transmissor e o receptor são, por fim, ligados por um *meio de comunicação* por meio das camadas físicas transmissora e receptora.

No simulador, a comunicação foi implementada de forma a ser compatível com os três tipos de codificação descritos na sessão anterior. No código, foram utilizadas *classes* como forma de abstração de cada componente das camadas. O diagrama abaixo ilustra melhor a arquitetura usada para implementar as diferente codificações:

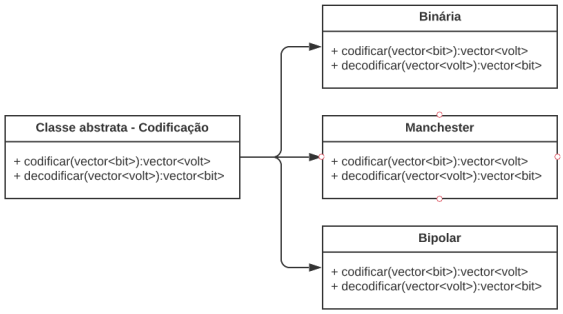

Figura 4. Diagrama de classes representando as codificações disponíveis no simulador.

Uma classe abstrata *Codificação* foi usada para criar uma interface que deve ser implementada pelas classes derivadas *Binária*, *Manchester* e *Bipolar*, sobrescrevendo os métodos virtuais e, consequentemente, implementando seus próprios métodos *codificar* e *decodificar*. Esse paradigma foi escolhido para que o usuário possa, em tempo real, selecionar o tipo de codificação desejado sem ter que alocar uma instância para cada tipo de código. Em vez disso, existe apenas uma instância de *Codificação* que armazena uma das classes derivadas por vez.

Para implementar a pilha de protocolos em si, isto é, as camadas físicas e de aplicação transmissoras e receptoras, também foram usadas classes e hierarquias, como pode-se observar nos diagramas a seguir:

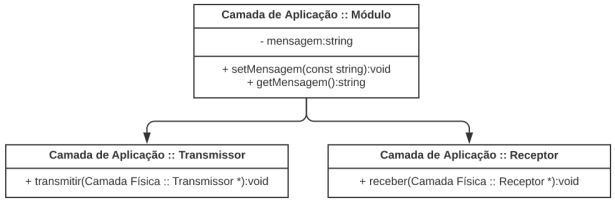

Figura 5. Diagrama de classes representando os módulos transmissor e receptor da camada de aplicação.

Na implementação da camada de aplicação, a classe base *Módulo* implementa métodos e armazena o atributo compartilhados entre as classes *Transmissor* e *Receptor*, como *setter* e *getter* para o atributo privado *mensagem* da classe

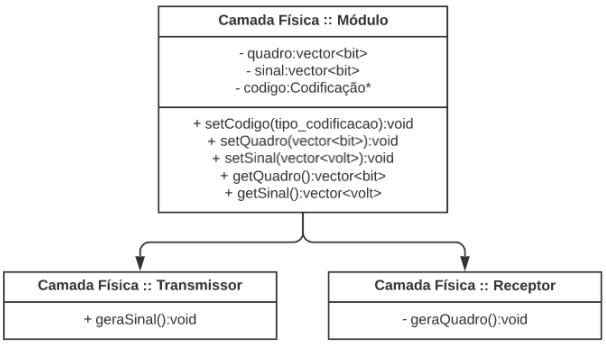

Figura 6. Diagrama de classes representando os módulos transmissor e receptor da camada física.

base. As classes *Transmissor* e *Receptor*, por sua vez, implementam os métodos *transmitir* e *receber*, respectivamente.

O método *transmitir* é responsável por gerar o quadro de bits correspondente à mensagem recebida pela *setter* do *Módulo* e inserir na instância de *CamadaFísica::Transmissor* recebida como parâmetro. O método *receber*, por sua vez, faz o caminho contrário: é responsável por gerar a mesagem a partir do quadro armazenada na instância de *CamadaFísica::Receptor* recebida como parâmetro.

Na camada física, a classe base *Módulo* implementa métodos e armazena atributos compartilhados entre as classes *Transmissor* e *Receptor*, como *setters* e *getters* para os atributos protegidos da classe base. As classes *Transmissor* e *Receptor*, por sua vez, implementam os métodos *geraSinal* e *geraQuadro*, responsáveis, respectivamente, por usar os métodos *codificar/decodificar* do atributo *codigo* para gerar o *sinal/quadro* e inserí-los nos atributos protegidos correspondentes.

Com os módulos das camadas físicas e de aplicação implementados, é necessário estabelecer a arquitetura do *meio de comunicação* para fazer os transmissores e receptores se comunicarem na simulação. A seguir está a arquitetura de *meio de comunicação* implementada nesse simulador:

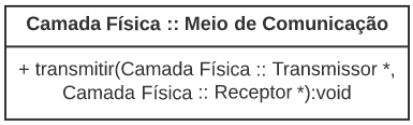

Figura 7. Diagrama de classes representando os meio de comunicação.

O método *transmitir* da classe *Meio de Comunicação* é responsável por transmitir o sinal gerado pelo *Transmissor* para o *Receptor*, ambos passados ao método como parâmetro.

Para implementar os passos da simulação seguindo o

paradigma OOP que foi estabelecido no projeto, foi implementada uma *super classe* que governa todos os módulos usados no processo de comunicação (transmissores e receptores das camadas física e de aplicação e meio de comunicação). O diagrama abaixo apresenta a estrutura da super classe *"Pilha"* descrita:

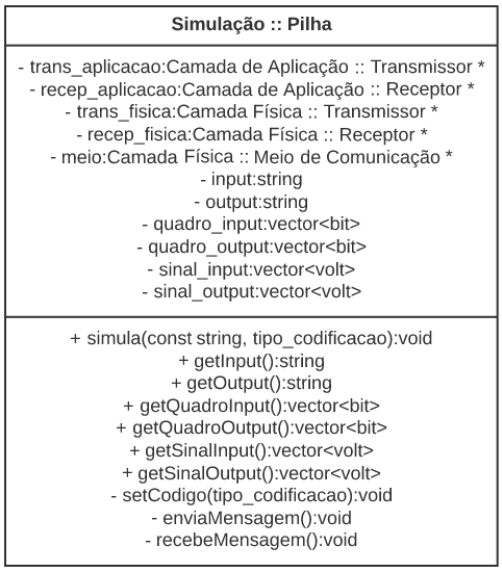

Figura 8. Diagrama de classes da super classe *Pilha*.

A super classe *Pilha* é composta majoritariamente por métodos *getter* e atributos privados que armazenam o resultado da simulação para cada estágio da comunicação (mensagem transmitida/recebida, quadro recebido/transmitido e sinal transmitido/recebido).

Os métodos mais importantes, porém, são os métodos *simula* e *enviaMensagem/recebeMensagem*. Estes últimos são métodos auxiliares que geram o quadro e enviam para a camada física; e recebem o quadro da camada física e traduz a mensagem codificada no quadro, respectivamente. O método *simula*, por sua vez, implementa a simulação de fato, seguindo o fluxograma da figura 9.

Para fim de facilitar o uso do simulador, uma interface gráfica foi desenvolvida usando as bibliotecas externas *Dear ImGui* e *ImPlot*. Internamente, essas bibliotecas usam *SDL2* e *OpenGL* para manipular as janelas, manipular os contextos gráficos e renderizar a interface.

A *GUI* implementada apresenta um campa para o usuário escrever a mensagem que quer transmitir entre o *Transmissor* e o *Receptor*, e um seletor de codificação a ser usada. O quadros recebidos/transmitidos e o sinal gerado são, então, plotados para o usuário visualizar a codificação e decodificação da mensagem.

A renderização dé feita em tempo real, permitindo que o usuário observe o quadro aumentar de tamanho enquanto a mensagem é escrita no input. A figura 10 demonstra a

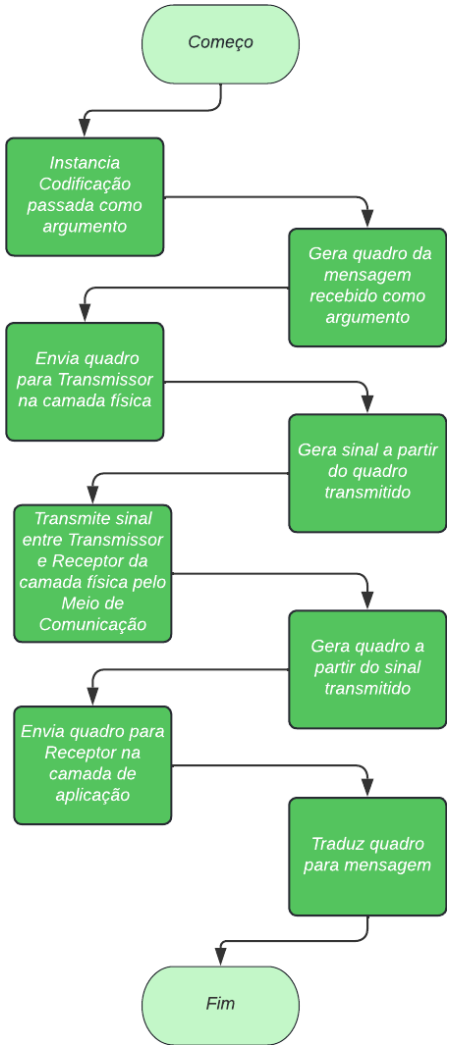

Figura 9. Fluxograma do método *simula* da classe *Pilha*.

interface gráfica criada.

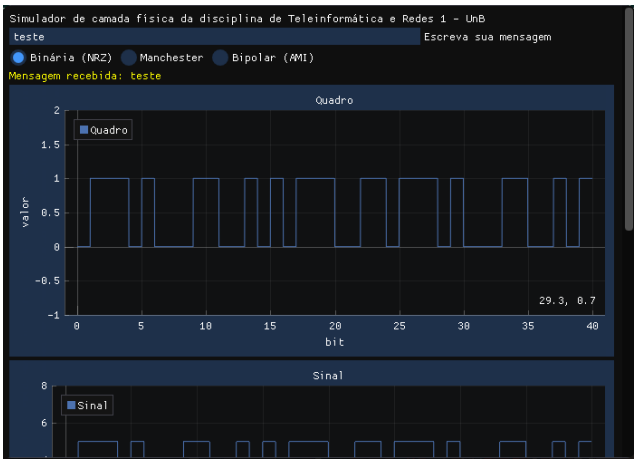

Figura 10. Demonstração da interface gráfica do simulador.

4. Membros

A seguir se encontra uma lista das atividades desenvolvidas por cada membro do grupo:

- **José Vinícius Garreto Costa:** Estrutura do relatório e fundamentação teórica.
- **Hyago Gabriel Oliveira Figueiredo:** Estrutura do relatório e fundamentação teórica.
- **Luiz Carlos Schonarth Junior:** Implementação do simulador e descrição da implementação.

5. Conclusão

A implementação do simulador foi simples após determinar o paradigma a ser seguido e contrução dos diagramas de classe. O maior problema durante a implementação foi, na realidade, a construção da interface gráfica.

Escolher um biblioteca externa para construção de uma interface gráfica não é uma tarefa simples, quando se tenta considera compatibilidade entre sistemas operacionais. Mas as bibliotecas selecionadas não decepcionam no quesito de compatibilidade e simplicidade de uso.

Outro problema durante o desenvolvimento foi lidar com *leaks* de memória alocado, especialmente aqueles que ocorrem dentro das bibliotecas externas, que não podem ser facilmente resolvidos.