Simulação da camada física usando os protocolos: Binário, Manchester e Bipolar em C++

- Introdução
- Camada Física e Camada de Aplicação
- Protocolo Binário
- Protocolo Manchester
- Protocolo Bipolar
- Implementação do simulador em C++
- Visão geral

Introdução

Os protocolos binários, Manchester e bipolares são cruciais para a transmissão de dados pelas redes e simulá-los com precisão nos permite entender seu comportamento e otimizar seu desempenho, além de ajudar a entender seu comportamento.

No simulador será demonstrado o funcionamento da camada física, onde o usuário a partir da camada de aplicação poderá interagir com a interface gráfica e escolher a codificação que deseja, podendo ser elas: Binária, Manchester ou Bipolar.

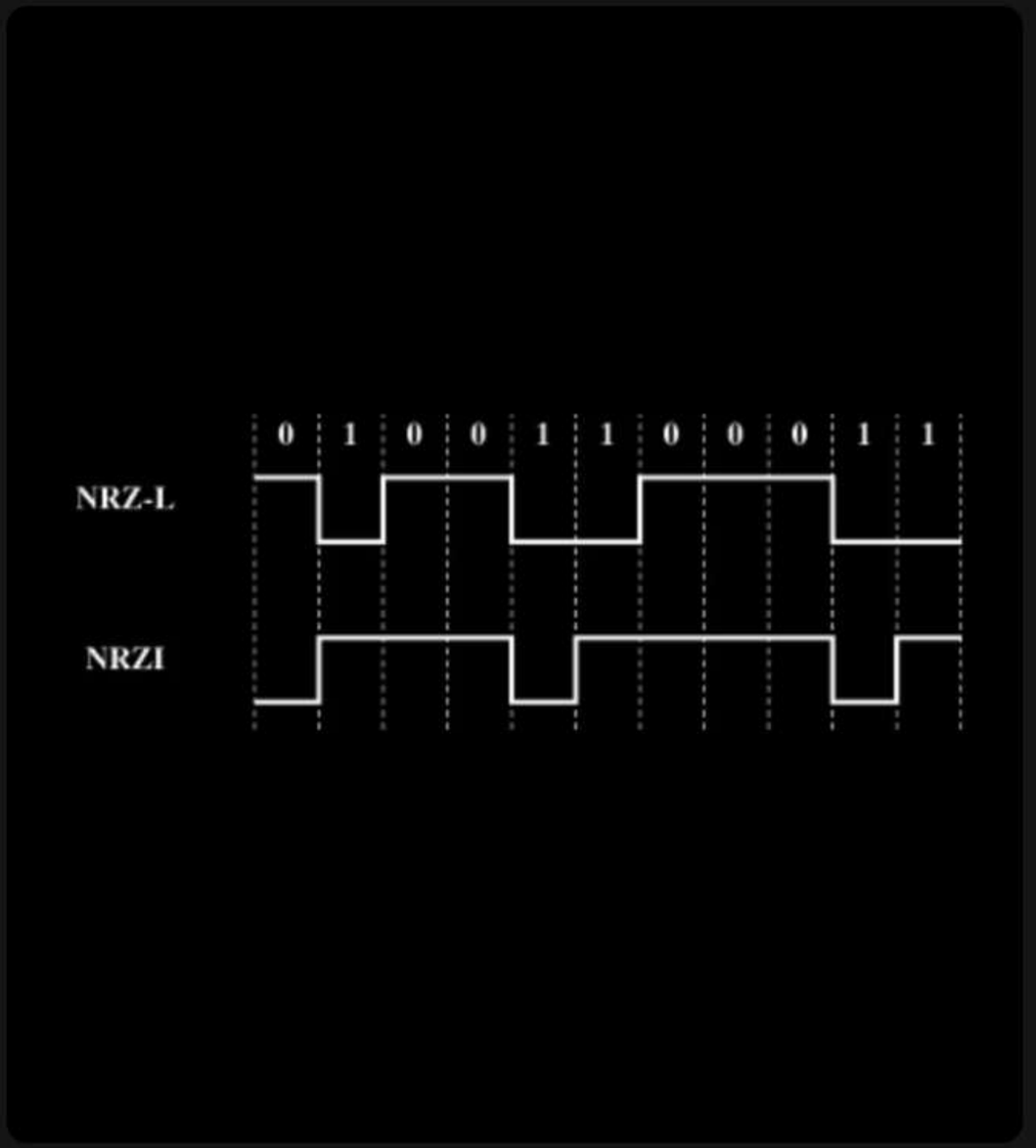
Camadas (Física e de Aplicação)

No simulador, serão abordadas duas camadas fundamentais do modelo de interconexão de sistemas abertos que é o padrão de comunicação entre os sistemas de computadores e redes (OSI): sendo elas, a camada física e a camada de aplicação.

O protocolo binário NRZI-M (Non-Return-To-Zero Inverted - Mark)

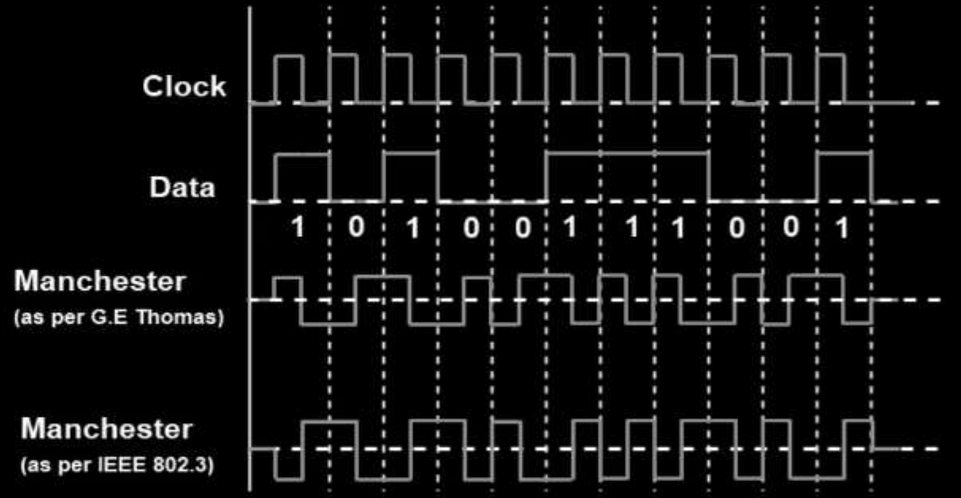
É uma técnica decodificação usada na camada física para transmitir os dados digitais. Neste protocolo, cada bit é representado por um nível de tensão e a codificação usa uma mudança no nível de tensão para representar um bit 1 e a ausência de mudança é representada por um bit 0.

É importante lembrar que o NRZI-M comparado ao NRZI convencional, inverte o sinal para representar um bit 1 ao invés de manter o sinal constante. Isso evita problemas de sincronização que podem surgir com longas sequências de bits 0.



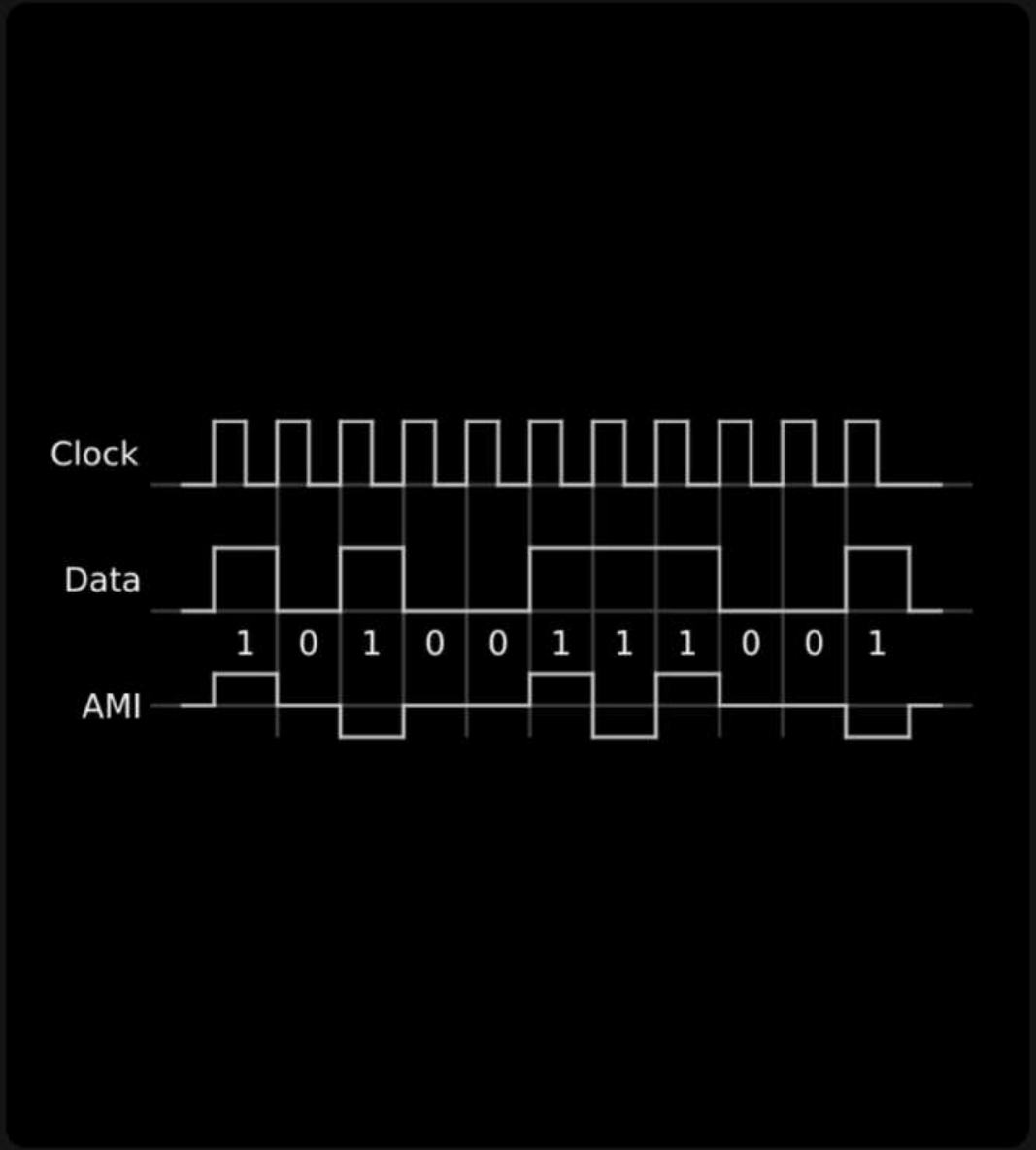
Protocolo Manchester

Também é uma técnica de codificação que utiliza transições de sinal para representar bits de dados. Sendo assim, cada bit é representado em dois períodos de tempo iguais, o primeiro período representa o bit 1 e o segundo período representa o bit 0. Durante o primeiro período, acontece uma transição do sinal de nível alto para o nível baixo, e durante o segundo, de nível baixo para o nível alto. Essas transições constantes de sinal, permitem a recuperação do relógio (clock) do sinal no receptor, o que ajuda a sincronizar o transmissor e o receptor.



Protocolo Bipolar (AMI - Alternate Mark Inversion)

É uma técnica de codificação onde cada bit de dados é representado por um sinal elétrico que pode ter três níveis possíveis: positivo, negativo e zero. A codificação é construída de forma que os bits 0 são representados por um nível de tensão 0, já os bits 1 são representados por alternâncias de polaridade. O sinal positivo representa o bit 1, o sinal negativo representa o bit 1 invertido e o sinal 0 representa o bit 0. A característica principal deste protocolo é sua técnica de inversão de polaridade, onde cada bit 1 é representado por uma alternância de polaridade em relação ao bit anterior, isso garante que a média da tensão do sinal seja zero em longas sequências de zeros consecutivos, o que ajuda a prevenir problemas de sincronização.



Implementação do simulador em C++

O simulador foi desenvolvido com base nestes protocolos e suas particularidades de codificação e decodificação no que tange o processo de transmissão e recepção da mensagem inserida pelo usuário partindo do pressuposto que contém apenas caracteres ASCII.

A codificação do simulador é modularizada e as funções implementadas no arquivo **camadaFisica.cpp** na camada física estão declaradas no arquivo de cabeçalho **camadaFisica.hpp**

```cpp
// Funcao de Codificacao Binaria (NRZI-M)
vector<bool> camadaFisicaTransmissoraCodificacaoBinaria(vector<bool> quadro) {
    // dois sinais (0 == V-, 1 == V+)

    int tamanho = (int) quadro.size();
    vector<bool> onda(tamanho*2, 0);

    for(int i=0; i<tamanho; i++) {
        // mantendo o sinal
        if (i > 0) { // o primeiro bit é sempre 0
            onda[2*i] = onda[2*i-1];
        }

        // invertendo o sinal caso seja um bit 1
        if (quadro[i] == 1) {
            onda[2*i+1] = !onda[2*i];
        }
        else {
            onda[2*i+1] = onda[2*i];
        }
    }

    imprimirSinal(quadro, 1, "Quadro a ser codificado com a codificacao Binaria (NRZI-M)");

    return onda;
}
```

```cpp
// Funcao de Decodificacao Binaria (NRZI-M)
vector<bool> camadaFisicaReceptoraDecodificacaoBinaria(vector<bool> onda) {
    // dois sinais (0 == V-, 1 == V+)

    int tamanho = (int) onda.size();
    vector<bool> quadro(tamanho/2, 0);

    for(int i=0; i<tamanho/2; i++) {
        // há transição (bit 1)
        if (onda[2*i] ^ onda[2*i+1]) {
            quadro[i] = 1;
        }
        // não há transição (bit 0)
        else {
            quadro[i] = 0;
        }
    }

    imprimirSinal(quadro, 1, "Quadro gerado a partir da decodificacao Binaria (NRZI-M)");
    return quadro;
}
```

```cpp
// Funcao de Codificacao Manchester (IEEE convention)
vector<bool> camadaFisicaTransmissoraCodificacaoManchester(vector<bool> quadro) {
    // dois sinais (0 == V-, 1 == V+)

    int tamanho = (int) quadro.size();
    vector<bool> clock(tamanho*2, 0), onda(tamanho*2, 0);

    // inicializa o clock
    for(int i=0; i<tamanho; i++)
        clock[2*i+1] = 1;

    // realiza o xor do sinal com o clock caso o bit seja 1
    for(int i=0; i<tamanho; i++) {
        onda[2*i] = clock[2*i] ^ quadro[i];
        onda[2*i+1] = clock[2*i+1] ^ quadro[i];
    }

    imprimirSinal(quadro, 1, "Quadro a ser codificado com a codificacao Manchester (IEEE)");
    return onda;
}
```

```cpp
// Funcao de Decodificacao Manchester
vector<bool> camadaFisicaReceptoraDecodificacaoManchester(vector<bool> onda) {
    // dois sinais (0 == V-, 1 == V+)

    int tamanho = (int) onda.size();
    vector<bool> quadro(tamanho/2, 0);

    for(int i=0; i<tamanho/2; i++) {
        // houve inversao pelo xor (bit 1)
        if (onda[2*i] == 1 and onda[2*i+1] == 0) {
            quadro[i] = 1;
        }
        // nao houve inversao pelo xor (bit 0)
        else {
            quadro[i] = 0;
        }
    }

    imprimirSinal(quadro, 1, "Quadro gerado a partir da decodificacao Manchester (IEEE)");
    return quadro;
}
```

```cpp
// Funcao de Codificacao Bipolar (AMI)
vector<bool> camadaFisicaTransmissoraCodificacaoBipolar(vector<bool> quadro) {
    // tres sinais (0 == Terra, 1 == V+, 2 == V-)

    int tamanho = (int) quadro.size();

    // flag booleano que cria o comportamento alternado do bipolar
    bool polaridade = 1; // 1 == V+, 0 == V-

    vector<bool> onda(tamanho*2, 0);

    for(int i=0; i<tamanho; i++) {
        // sinal com bit 1 (V = V+ ou V-)
        if (quadro[i]) {
            if (polaridade) {
                onda[2*i] = 0;
                onda[2*i+1] = 1;
            }
            else {
                onda[2*i] = 1;
                onda[2*i+1] = 0;
            }
            polaridade = !polaridade;
        }
        // sinal com bit 0 (V = 0)
        else {
            onda[2*i] = 0;
            onda[2*i+1] = 0;
        }
    }

    imprimirSinal(quadro, 1, "Quadro a ser codificado com a codificacao Bipolar (AMI)");
    return onda;
}
```

```cpp
// Funcao de Decodificacao Bipolar
vector<bool> camadaFisicaReceptoraDecodificacaoBipolar(vector<bool> onda) {
    // tres sinais (0 == Terra, 1 == V+, 2 == V-)

    int tamanho = (int) onda.size();
    vector<bool> quadro(tamanho/2, 0);

    for(int i=0; i<tamanho/2; i++) {
        // Sinal sem polariade -> Bit 0
        if (onda[2*i] == 0 and onda[2*i+1] == 0) {
            quadro[i] = 0;
        }
        // Sinal com polariade -> Bit 1
        else {
            quadro[i] = 1;
        }
    }

    imprimirSinal(quadro, 1, "Quadro gerado a partir da decodificacao Bipolar (AMI)");
    return quadro;
}
```

Visão geral

Essas simulações podem ajudar pesquisadores e engenheiros a entender melhor como esses protocolos funcionam, suas vantagens e desvantagens e como eles podem ser otimizados para diferentes aplicações.

O simulador modularizado em C++ facilita a simulação de diferentes protocolos e a experimentação de vários parâmetros.