Introdução

Um transmissor fará o envio de dados à um receptor. Porém, ao longo do caminho, o dado enviado receberá ou não algum ruido, interferindo em sua integridade. Nosso objetivo é implementar um código que faça o cálculo do CRC e mescle-o ao dado original no transmissor. Já o receptor irá receber o dado e devemos implementar um código que verifique sua integridade e decodifique-o caso não haja erros. Caso contrário, o receptor deverá requisitar a retransmissão da informação ao transmissor.

Implementação

A implementação do nosso código baseia-se na lógica do cálculo do CRC que envolve operações XORs dos bits a serem enviados.

O polinômio escolhido para fazer as operações foi “1011”.

Após o dado ser convertido em um vetor de oito posições (oito bits ou um byte), é feito a chamada da função “dadosBitsCRC()” que retorna o dado a ser enviado junto ao CRC. Para o funcionamento da função, é feito a adição dos zeros ao dado original pela função “inserirZeros()”, que cria um novo vetor de tamanho igual a soma do tamanho do vetor do dado mais a quantidade de zeros a ser anexados. Essa função retornará um novo vetor que será utilizado na função “xorDadoModificado()”.

A função “xorDadoModificado()” é responsável pelos cálculos XORs para encontrar o CRC. Para isso, é criado um novo vetor chamado “bitsXOR[]” de tamanho igual a quantidade de bits do polinômio (no nosso caso, quatro). Nele é armazenado os primeiros bits do vetor do dado. Para realizar as operações XORs, utilizamos um “for” sendo a condição de início a posição do próximo bit a ser inserido ao resto do cálculo (“bitIndicador”) e a de parada sendo o valor do “bitIndicador” ser maior que o tamanho do vetor do dado modificado. Dentro do “for” é feita a verificação do primeiro bit do vetor “bitsXOR[]”. Caso seja ‘1’, o cálculo é feito comparando bit-a-bit do vetor “bitsXOR[]” com os bits do polinômio, resultando em um novo valor que é armazenado no próprio “bitsXOR[]”. Caso seja ‘0’ ou o cálculo anterior seja finalizado, o algoritmo irá deslocar os bits do vetor “bitsXOR[]” uma posição para esquerda utilizando a função “deslocarBits()”, que elimina o zero da primeira posição, realoca os outros bits do vetor e pega o próximo bit a “descer” do dado modificado (“bitIndicador”) e o coloca na última posição do “bitsXOR[]”. Por fim, a função “xorDadoModificado()” retornará os “bitsXOR” contendo o CRC.

Após feitos esses processos, a função “mesclarBits()” mescla os bits do dado original com o CRC encontrado em “xorDadoModificado()” copiando os dois vetores em um novo, sendo seu tamanho a soma dos tamanhos dos dois vetores. Logo após, é gerado ou não interferência neste novo vetor.

O receptor ao receber o dado com o CRC e com ou sem ruído verifica se o mesmo está integro. Para isso, dentro da função “decodificarDadoCRC()” é feito os mesmos cálculos XORs presente no transmissor por meio da função “xorDadoRecebido()” que retornará um vetor de quatro bits e que será armazenado no vetor “bitsResto[]”. Logo após, é verificado os bits deste vetor. Caso todos sejam ‘0’, o dado recebido está integro e a função retorna true, caso contrário, existiu alguma interferência no dado recebido e a função retorna false.

Se retornado false, o receptor requisita a retransmissão do dado ao transmissor. Ele fará isso até o dado recebi estiver integro. Se retornado true, é retirado o CRC do dado a ser decodificado por meio da função “dividirBitsCRC()” e a conversão é feita pela função “decodificarDado()”.

Conclusão

No geral, o trabalho foi divertido, contudo desafiador. Enfrentamos algumas dificuldades na implementação da lógica do cálculo do CRC e na correção de erros do código.