UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Tecnologia - FT

Grupo: "Programadores"
Gabriel Domingues Ferreira. RA: 216207
Giovanni Bassetto. RA: 216968
Larissa Benevides Vieira. RA: 200805

Projeto "Verificação de Paridade Bidimensional"

CAMPINAS
2018

Introdução

A paridade bidimensional é uma técnica derivada da paridade de bloco único, a qual possui melhor eficiência, devido sua maior capacidade de detecção de erros.

A lógica estrutural aplicada neste método, pode ser melhor descrita por Tenenbaum (2003, p. 209), cujo diz que:

> *"As disparidades poderão ser consideravelmente melhoradas se cada bloco for enviado como uma matriz retangular com **n** bits de largura e **k** bits de altura [...] Um bit de paridade é calculado separadamente para cada coluna e afixado à matriz como sua última linha. Em seguida a matriz é transmitida uma linha de cada vez."*

A Figura 3, demostra de maneira simples, como pode ser interpretada a matriz de paridade bidimensional, para o processamento dos bits de paridade, referentes ao bloco de verificação, o qual é afixado como última linha da matriz, e no conceito do projeto, o qual deve ser comparada com a entrada fornecida pelo usuário no 3 Byte. Do ponto de vista prático, primeiramente deve ser processado o bit de paridade que será adicionado ao final de cada pacote, gerando uma última coluna, denominada de verificação de redundância vertical (VRC - *Vertical redundancy check*). Posteriormente são processados são calculados os bits de paridade para cada coluna, incluindo os da coluna VRC, que integrarão um último pacote de verificação, geralmente intitulado de verificação de redundância longitudinal (LRC- *Longitudinal redundancy check*), ou paridade horizontal. Após tais procedimentos, os dados já estarão codificados e prontos para a comparação com os bytes fornecidos pelo usuário.

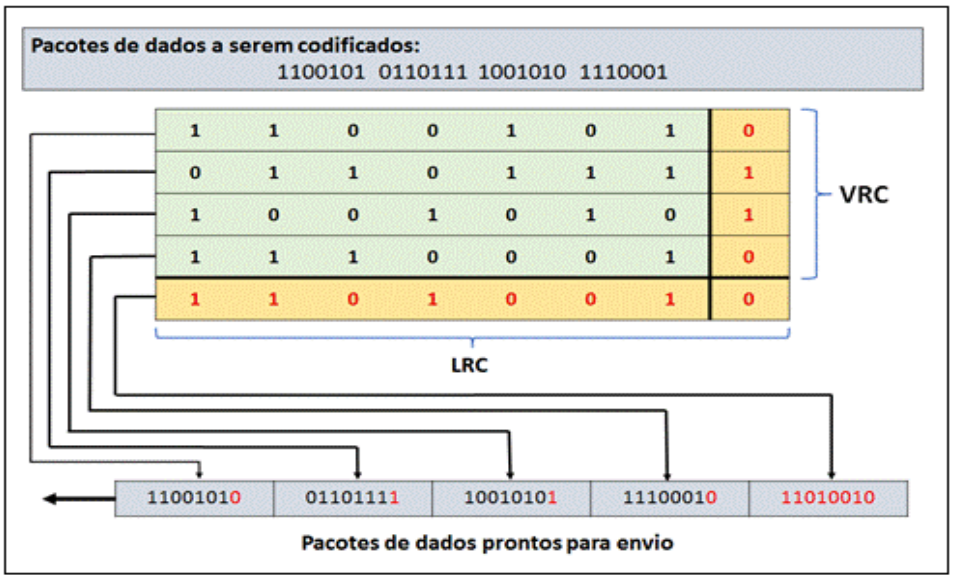
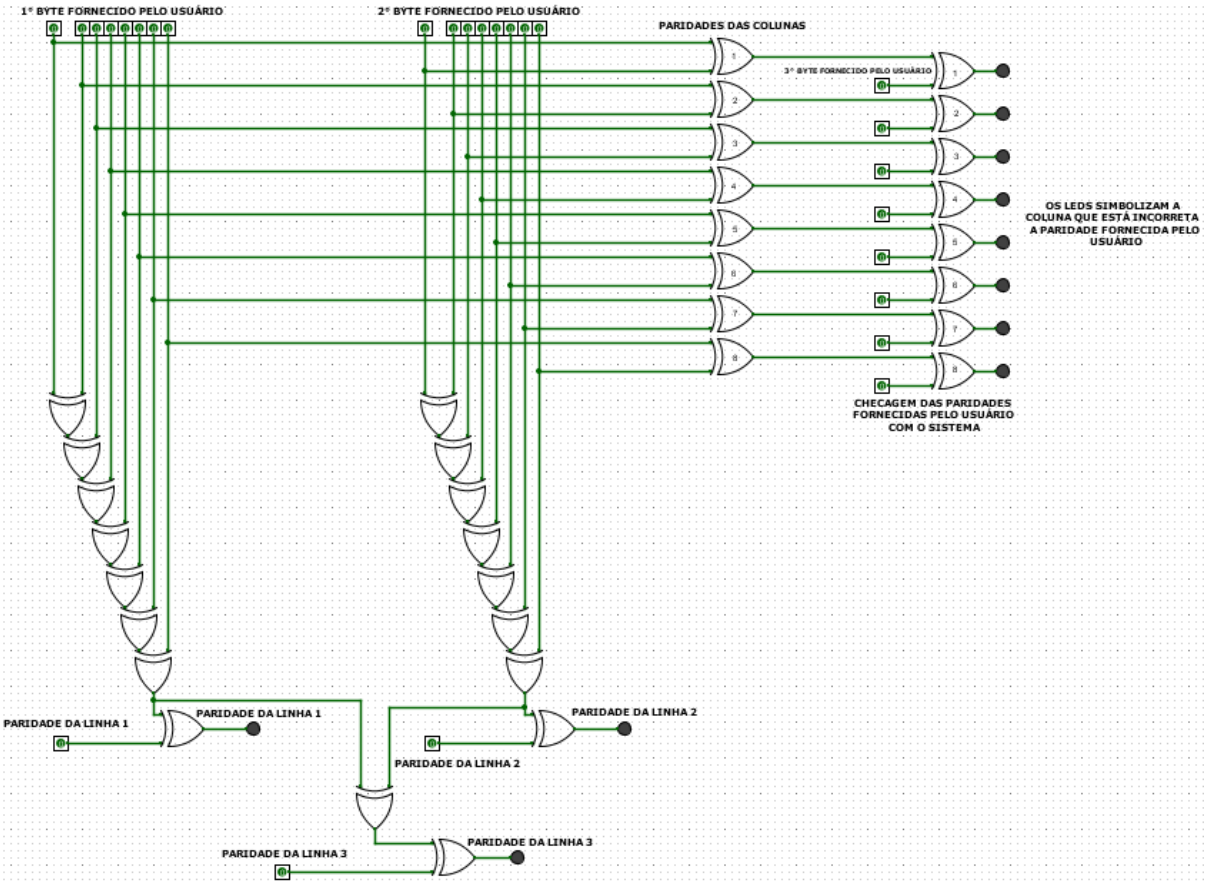


Figura 3: Matriz de paridade par bidimensional

O circuito combinacional completo implementado com o software Logisim



As funções lógicas para cada saída do circuito combinacional que calcula os bits de paridade

No nosso circuito existem três funções lógicas, duas delas tem a finalidade de calcular a paridade da primeira, da segunda e da terceira linha separadamente (1° e 2° bytes). Sua função pode ser escrita da seguinte forma:
Paridade da 1ª linha = X9.
Paridade da 2ª linha = Y9.
Paridade da 3ª linha = Z9.

Paridade da 1ª linha = ((((((X1⊕X2)⊕X3)⊕X4)⊕X5)⊕X6)⊕X7)⊕X8).

Paridade da 2ª linha = ((((((Y1⊕Y2)⊕Y3)⊕Y4)⊕Y5)⊕Y6)⊕Y7)⊕Y8).

Paridade da 3ª linha=
((((((X1⊕X2)⊕X3)⊕X4)⊕X5)⊕X6)⊕X7)⊕X8)⊕(((((((Y1⊕Y2)⊕Y3)⊕Y4)⊕Y5)⊕Y6)⊕Y7)⊕Y8).

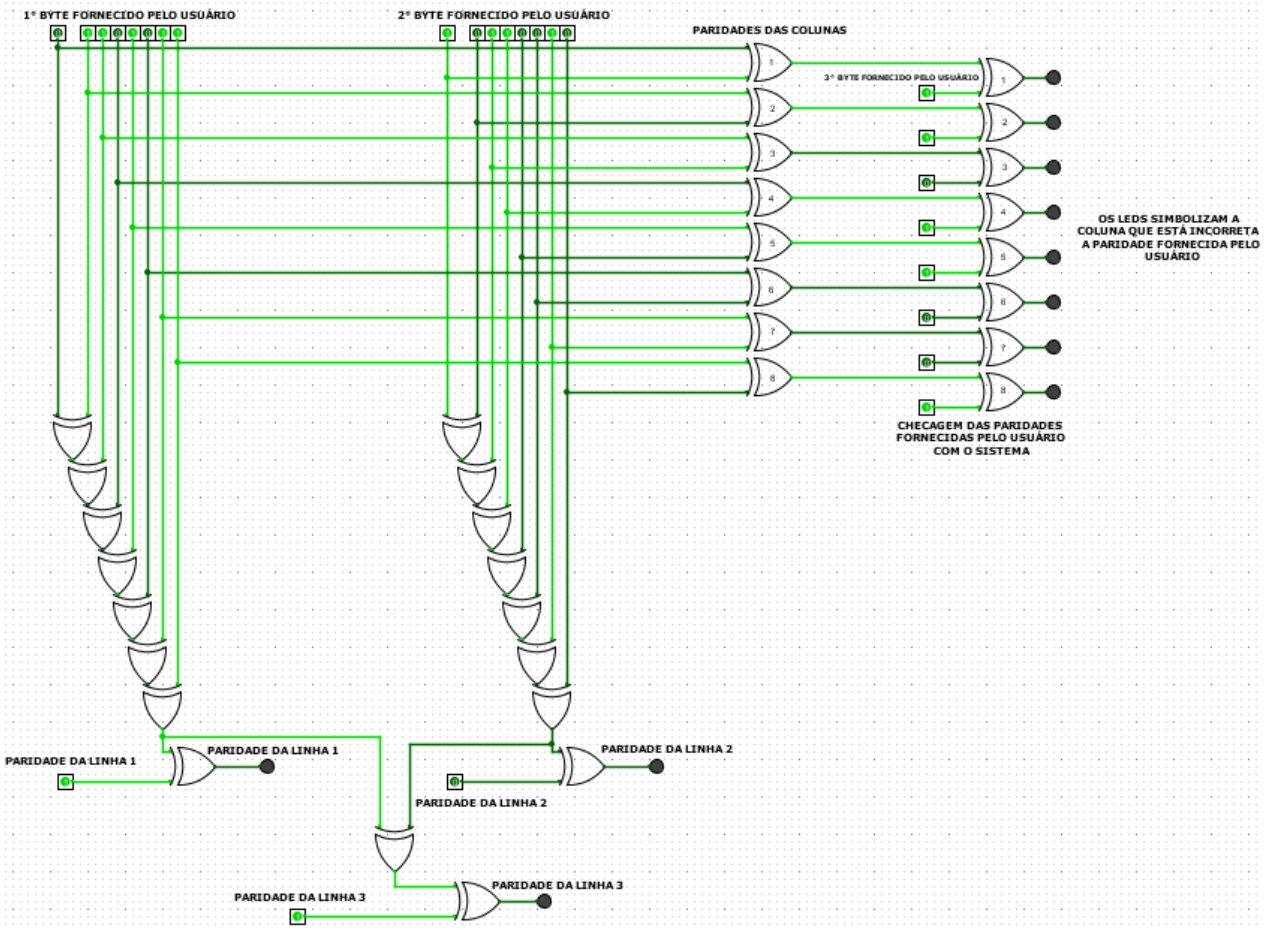
A terceira função lógica é responsável por calcular as paridades de todas as colunas e comparar o resultado obtido pelo circuito com a entrada inserida pelo usuário. Com isso, se houver diferença dos valores, o LED acenderá, indicando exatamente em qual coluna e linha se encontra o erro.

Usando-se as variáveis X1, X2, X3, X4 … X9 como os bits da linha 1, Y1, Y2, Y3 …. Y9 como os bits da linha 2, calculamos a paridade das colunas, da seguinte maneira:

((X1⊕Y1)) = Paridade da coluna 1
((X2⊕Y2)) = Paridade da coluna 2
((X3⊕Y3)) = Paridade da coluna 3
((X4⊕Y4)) = Paridade da coluna 4
((X5⊕Y5)) = Paridade da coluna 5
((X6⊕Y6)) = Paridade da coluna 6
((X7⊕Y7)) = Paridade da coluna 7
((X8⊕Y8)) = Paridade da coluna 8
((X9⊕Y9)) = Paridade da coluna 9

Exemplificando, o nosso circuito possui 3 entradas de dados, nas quais, o 1º e o 2º byte são inseridos para que o circuito calcule todas as paridades. A paridade da 9ª coluna é a mesma paridade da 3ª linha, sendo os dois a mesma função, os quais são calculados a partir da paridade da 1ª linha e da 2ª linha. O 3º byte é a comparação entre o resultado do circuito e as paridades que o usuário acha que estão corretas. O circuito compara esses dados e se houver diferença, o respectivo LED referente a coluna X e linha Y acenderá, denunciando o erro.

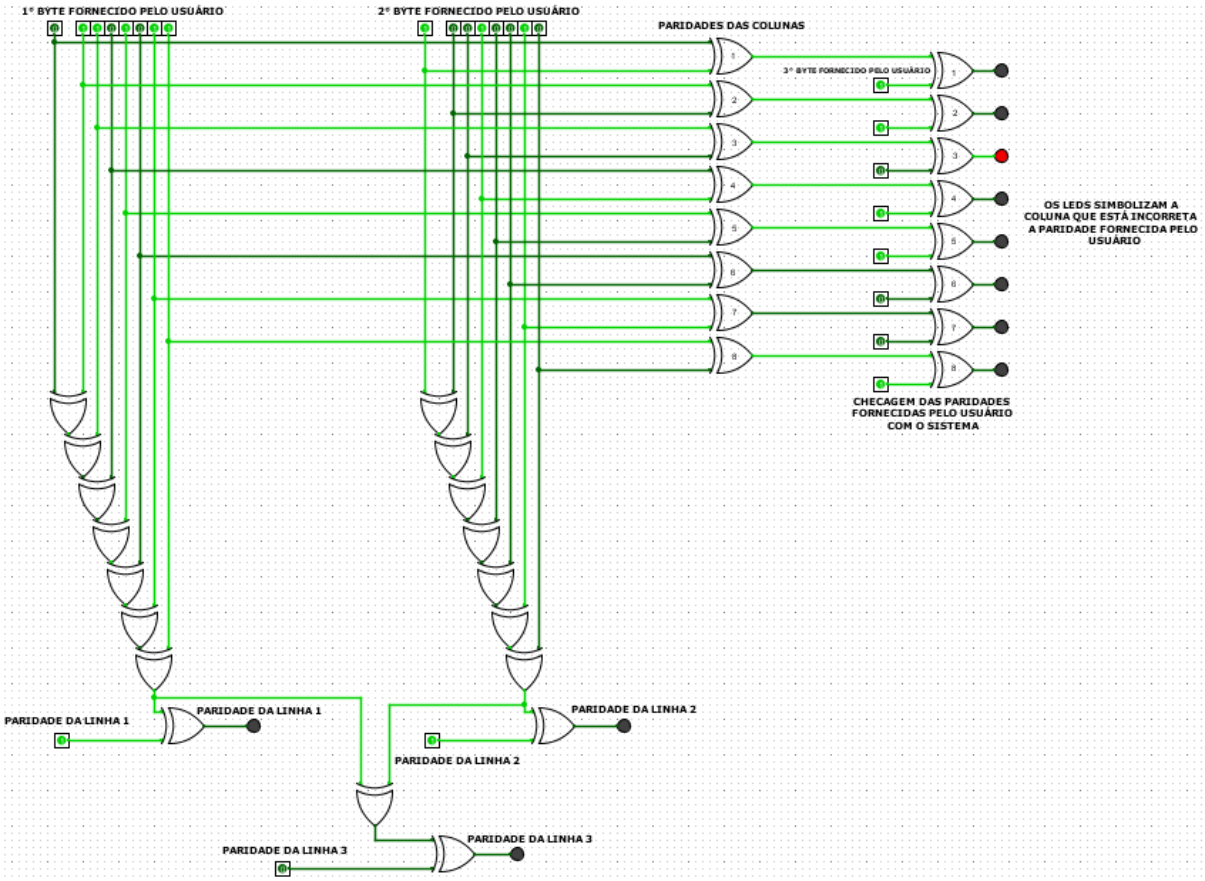
Demonstração da funcionalidade do circuito para os exemplos do texto do projeto e outros exemplos escolhidos.



(Exemplo de sequência não corrompida do texto do Projeto)

Exemplo de sequência não corrompida

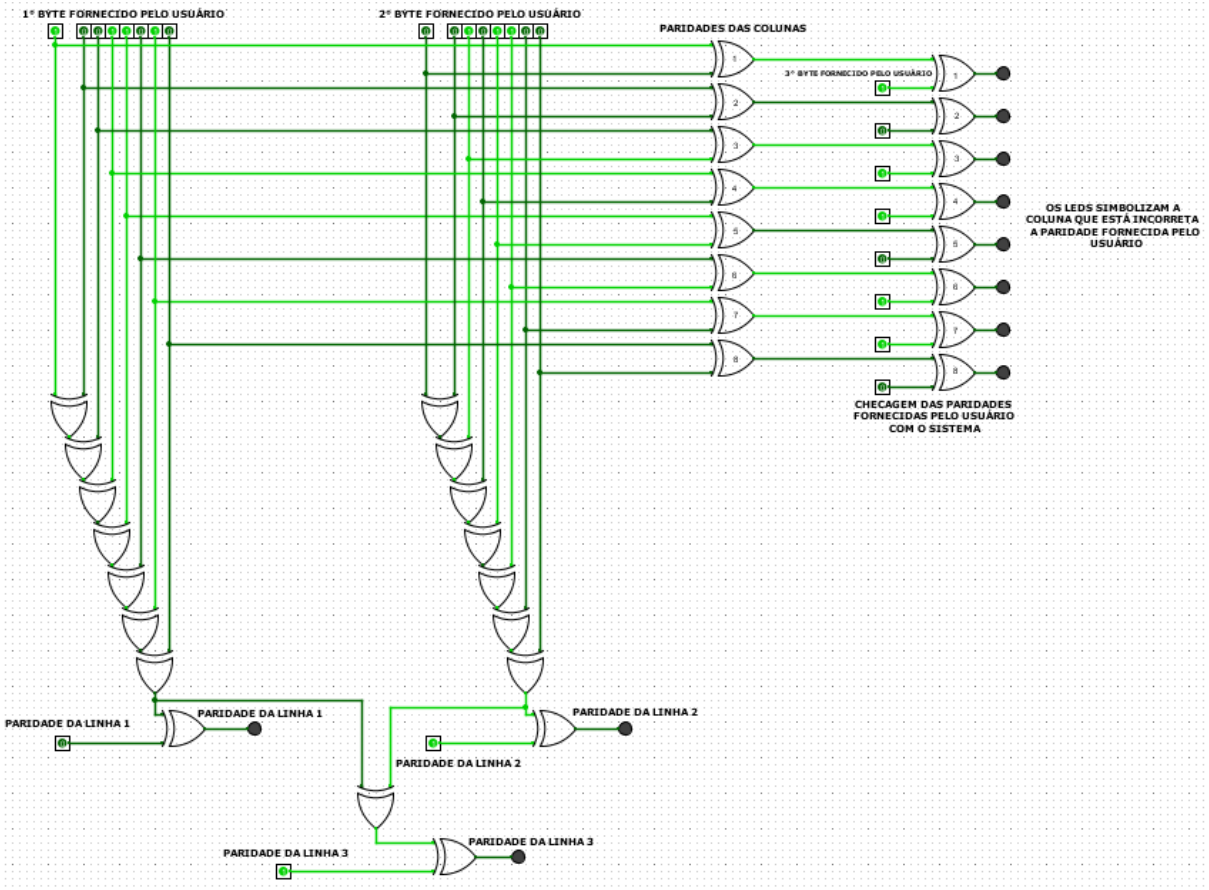
0	1	1	0	1	0	1	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	1	0	0	1	1



(Exemplo de sequência corrompida do texto do Projeto)

Exemplo de sequência corrompida

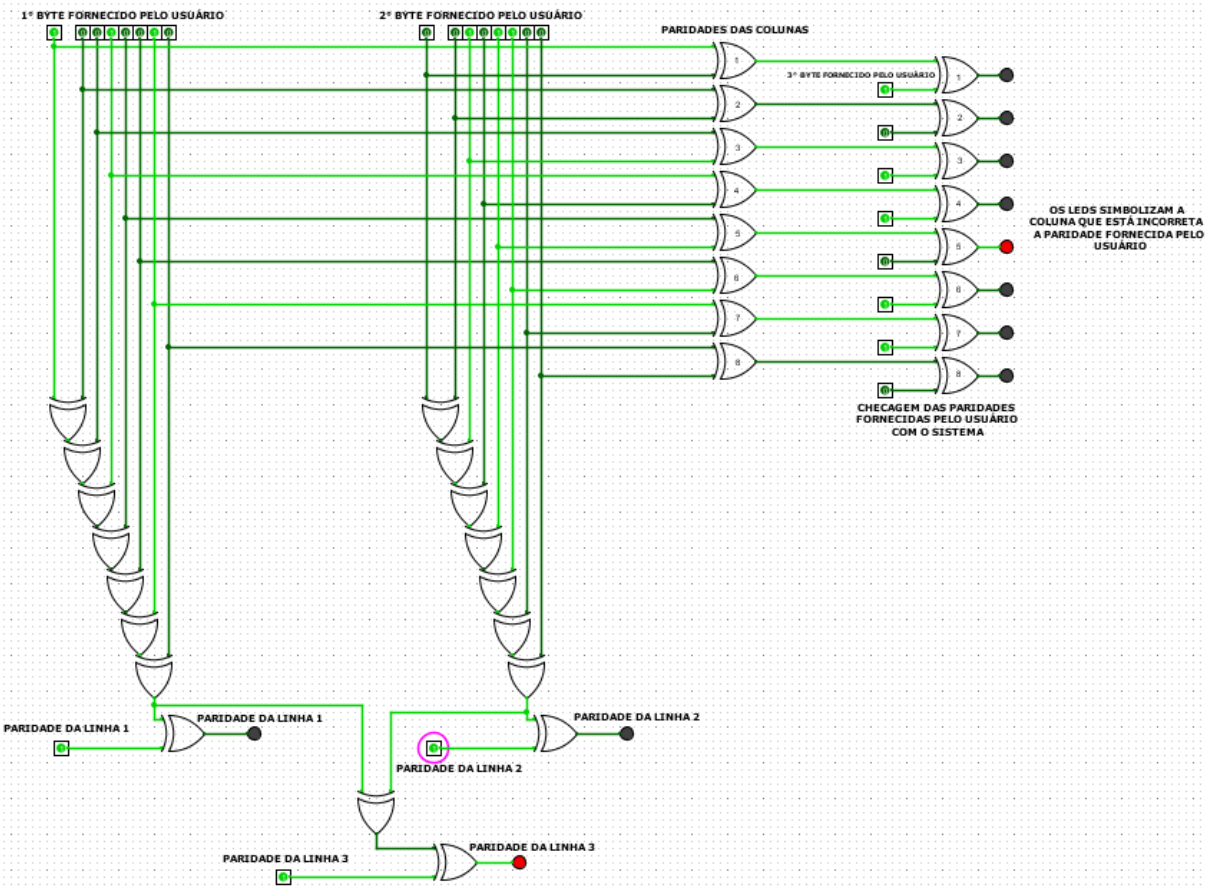
0	1	1	0	1	0	1	1	1
1	0	0	1	0	0	1	0	1
1	1	0	1	1	0	0	1	0



Exemplo 1 (Sequência não corrompida)

Exemplo 1: Sequência não corrompida

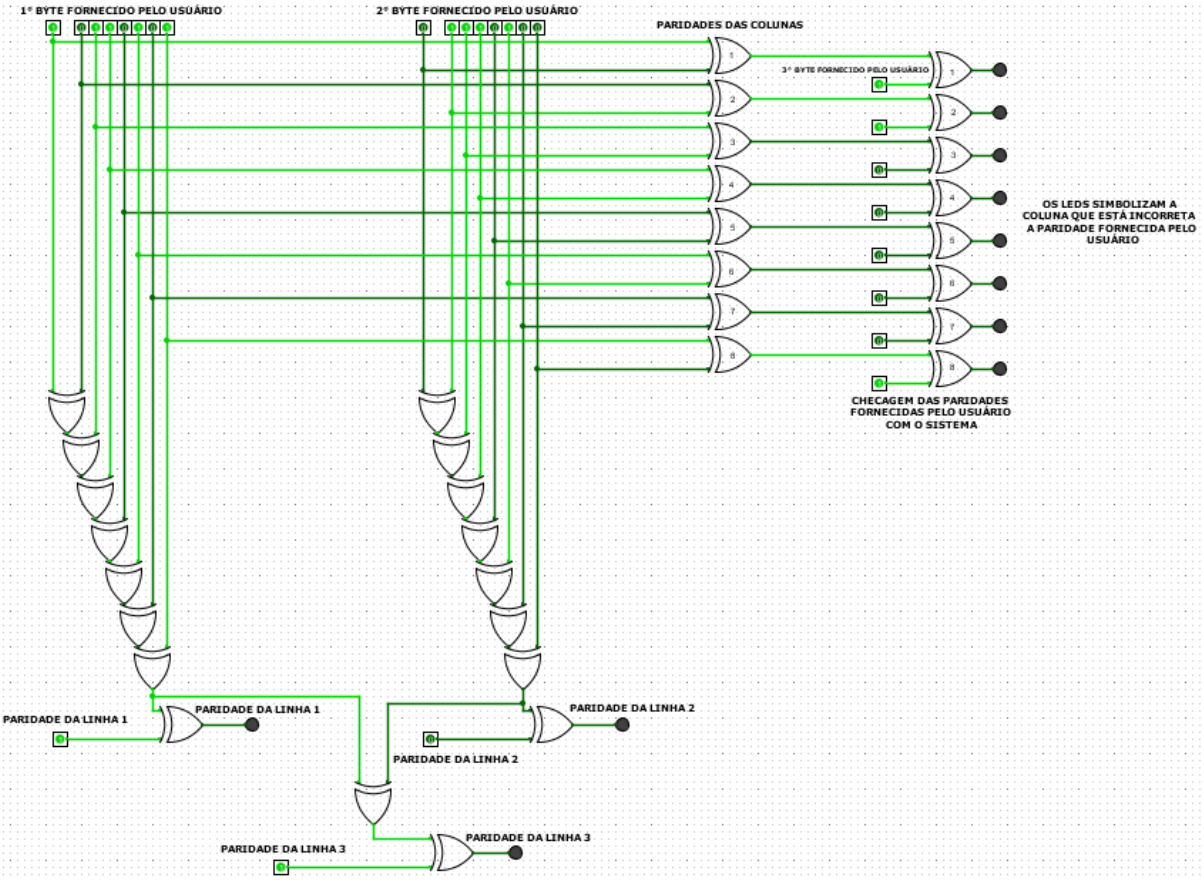
1	0	0	1	1	0	1	0	0
0	0	1	0	1	1	0	0	1
1	0	1	1	0	1	1	0	1



Exemplo 1 (Sequência corrompida)

Exemplo 1: Sequência corrompida na coluna 5 e na paridade da linha 3

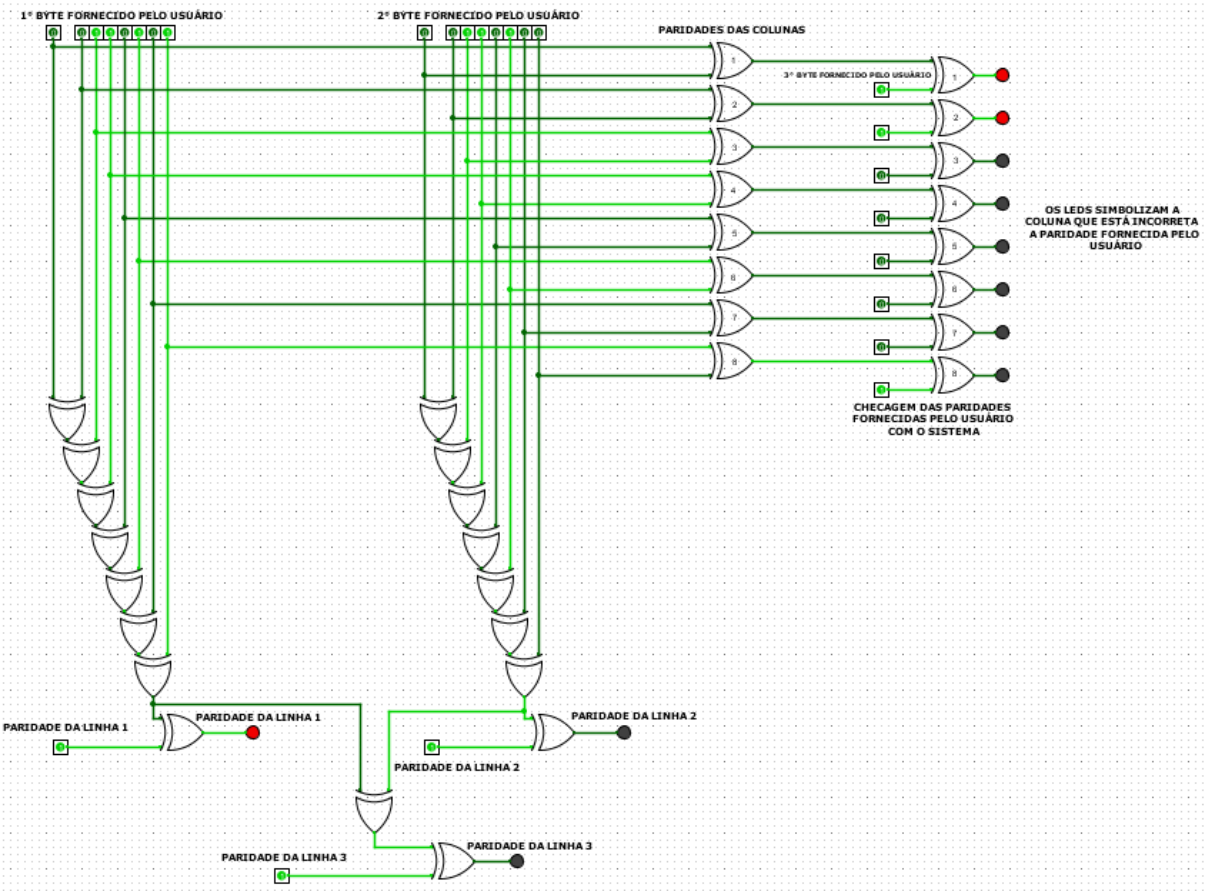
1	0	0	1	0	0	1	0	1
0	0	1	0	1	1	0	0	1
1	0	1	1	0	1	1	0	1



Exemplo 2 (Sequência não corrompida)

Exemplo 2: Sequência não corrompida

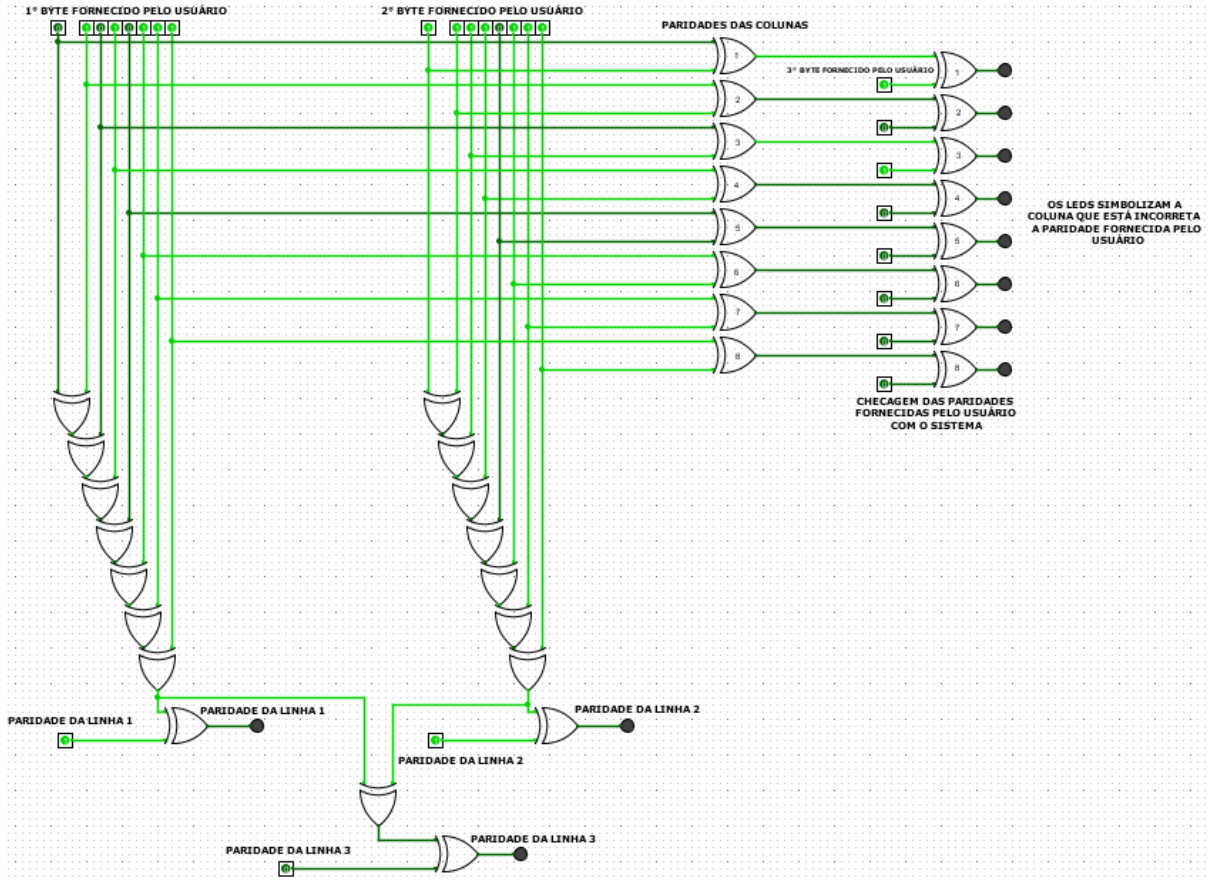
1	0	1	1	0	1	0	1	1
0	1	1	1	0	1	0	0	0
1	1	0	0	0	0	0	1	1



Exemplo 2 (Sequência corrompida)

Exemplo 2: Sequência corrompida na 1ª e 2ª coluna e na paridade da 1ª linha

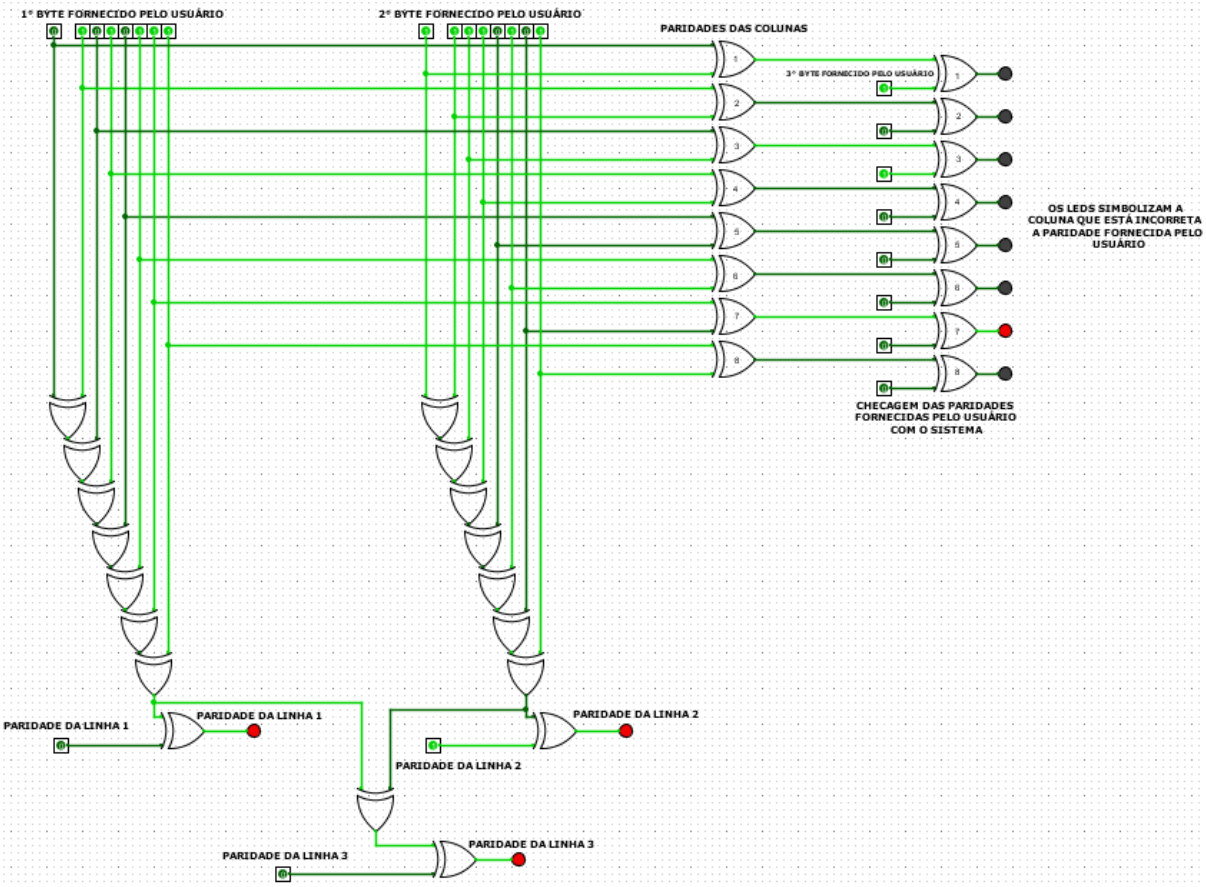
0	0	1	1	0	1	0	1	1
0	0	1	1	0	1	0	0	1
1	1	0	0	0	0	0	1	1



Exemplo 3 (Sequência não corrompida)

Exemplo 3: Sequência não corrompida

0	1	0	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1
1	0	1	0	0	0	0	0	1



Exemplo 3 (Sequência corrompida)

Exemplo 3: Sequência corrompida na 7ª coluna e na paridade da 1ª, 2ª e 3ª linha

0	1	0	1	0	1	1	1	0
1	1	1	1	0	1	0	1	1
1	0	1	0	0	0	0	0	1