UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CIÊNCIA DA COMPUTAÇÃO - 2ª FASE
DISCIPLINA: CIRCUITOS DIGITAIS
PROFESSOR LUCIANO LORES CAIMI
TRABALHO 1

ANDERSON HENRIQUE GROSSELLI TABALDI
RENAN CARLOS LOEWENSTEIN

O seguinte trabalho simula o jogo “Batalha Naval”, em que o mar é formado por uma matriz de 4x4 posições e são colocados dois navios dentro da área escolhida pelo jogador 1, enquanto o jogador 2 controla, através de código binário, e escolhe a posição que deseja atirar e tentar acertar o navio adversário, por meio da disposição de 4 chaves para informar o alvo e um botão para solicitar o disparo.

A seguir será descrita:

Codificação: A partir da tabela de codificação, optamos por tomar os quatro níveis lógicos como variáveis A, B, C e D. Utilizamos o símbolo de “negado (~)” para o nível lógico 0 (zero) e no nível lógico 1 as variáveis não levavam esse símbolo.

Abaixo comparamos as posições da parte codificada e do código normal binário:

1111	1110	1101	1011
0101	1000	0111	1010
0010	0100	1100	0110
0000	0011	0001	1001

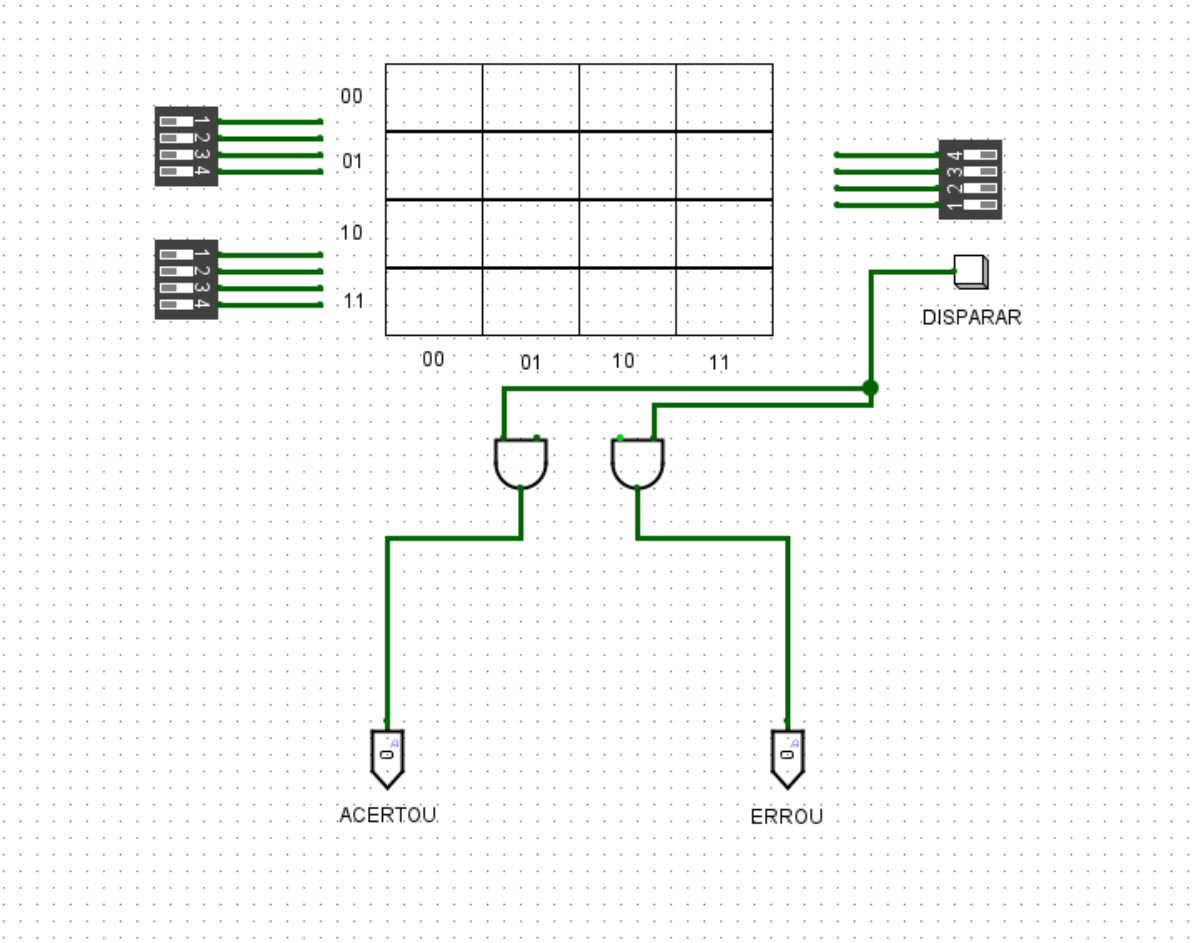
0 = A . B . C . D	0000 = ~A . ~B . ~C . ~D
1 = ~A . B . ~C . D	0001 = ~A . ~B . ~C . D
2 = ~A . ~B . C . ~D	0010 = ~A . ~B . C . ~D
3 = ~A . ~B . ~C . ~D	0011 = ~A . ~B . C . D
4 = A . B . C . ~D	0100 = ~A . B . ~C . ~D
5 = A . ~B . ~C . ~D	0101 = ~A . B . ~C . D
6 = ~A . B . ~C . ~D	0110 = ~A . B . C . ~D
7 = ~A . ~B . C . D	0111 = ~A . B . C . D
8 = A . B . ~C . D	1000 = A . ~B . ~C . ~D
9 = ~A . B . C . D	1001 = A . ~B . ~C . D
10 = A . B . ~C . ~D	1010 = A . ~B . C . ~D
11 = ~A . ~B . ~C . D	1011 = A . ~B . C . D
12 = A . ~B . C . D	1100 = A . B . ~C . ~D
13 = A . ~B . C . ~D	1101 = A . B . ~C . D
14 = ~A . B . C . ~D	1110 = A . B . C . ~D
15 = A . ~B . ~C . D	1111 = A . B . C . D

Tabela-verdade:

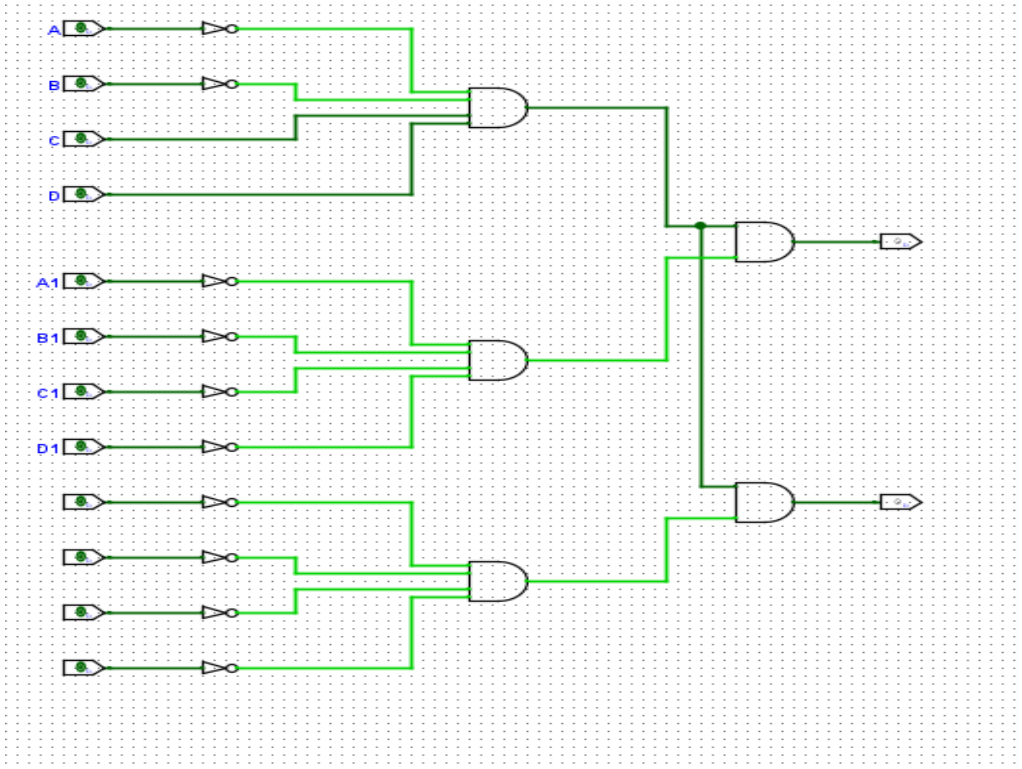
A	B	C	D	S
1	1	1	1	1
0	1	0	1	0
0	0	1	0	0
0	0	0	0	0
1	1	1	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	1	0
1	1	0	1	0
0	1	1	1	0
1	1	0	0	0
0	0	0	1	0
1	0	1	1	0
1	0	1	0	0
0	1	1	0	0
1	0	0	1	0

Circuito utilizando portas lógicas no LogiSim:

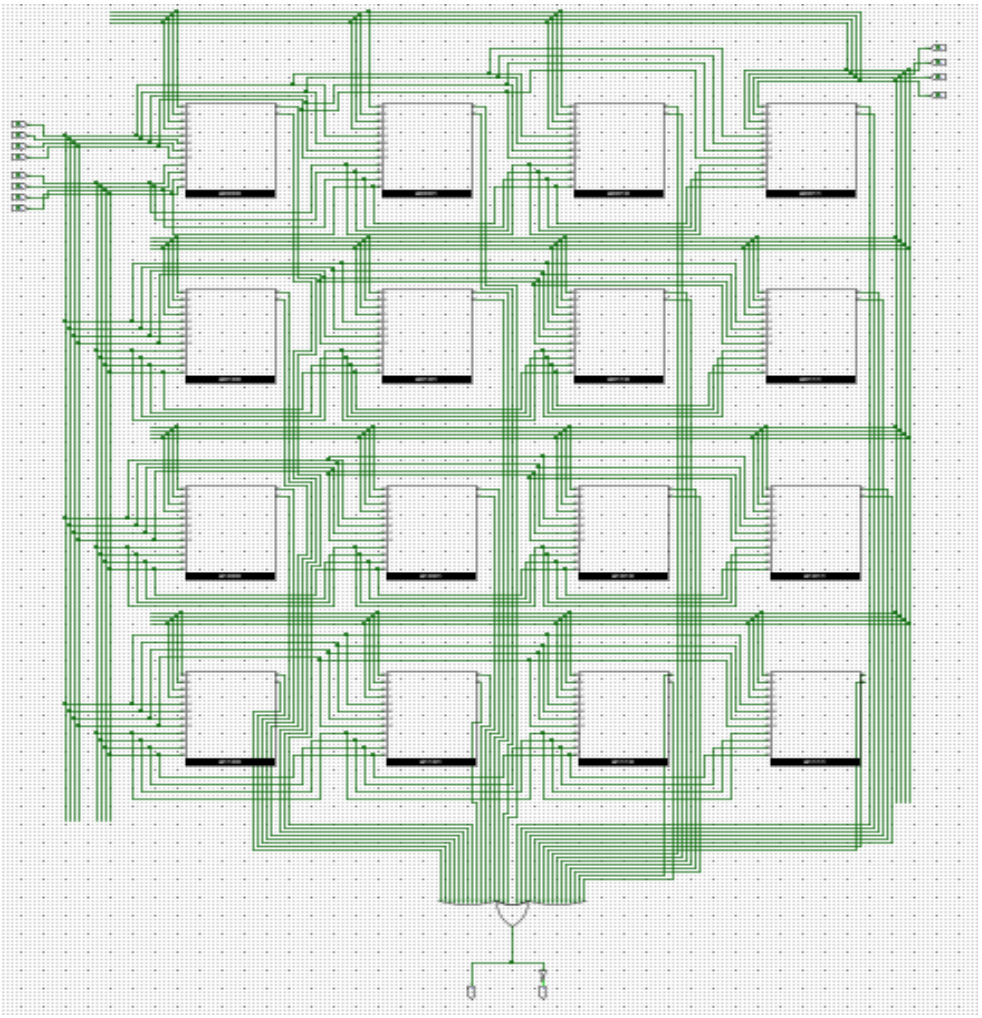
Menu principal, parte em que os dois jogadores irão destravar a Batalha Naval: Nas chaves da esquerda são definidas as posições dos navios 1 e 2, à direita é definida a posição em que o jogador 2 deseja realizar o disparo. Caso o disparo seja efetuado exatamente na posição codificada do navio, a saída “acertou” vai para nível lógico 1, em caso de erro, a saída “errou” muda para 1.



Em cada expressão da tabela formamos um circuito como este abaixo, ao todo são 16 circuitos, simulando as posições no mar. Utilizamos a lógica de que a expressão quando ativadas simultaneamente ao mesmo código, elevam para nível lógico 1, portanto, a porta “and” faz essa seleção dos acertos e erros, e o circuito de baixo é para a segunda posição do navio.



Parte “TODOS”, em que estão todas as entradas e combinações possíveis do circuito: Utilizamos as 8 chaves da esquerda para inserir as posições do navio, e as entradas da direita para simular a jogada do usuário. Caso acerte, a saída do mesmo será igual a 1, chegando na porta “and”, tornando o resultado como assertivo, negando outra saída para representar a situação de erro.



Circuitos utilizando CIs no TinkerCad:

Link para acessar o circuito:
https://www.tinkercad.com/things/0FJVbkAe5Aq-fantabulous-jaban/editel?sharecode=Upr_Uc1ooMVYOTtuvgFu2OuRqgmGdeFY5eRcJ0t8r9Y

Conclusão:

Baseamos nossa prática em detalhar as expressões em 16 circuitos, comparando a parte codificada com o normal binário, onde a assertividade era dada por uma porta “and”. Agrupamos os circuitos e nos casos de entradas conectamos com um uma codificação geral para todos, onde obtivemos a saída de cada circuito, reunindo todas as saídas em uma “or”, assim, conquistamos o resultado 1 para caso de assertividade.

No decorrer do trabalho, a tabela verdade não nos pareceu explícita no programa para com aquele circuito, portanto, tivemos dificuldade na parte de extrair a tabela verdade. Além disso, se aproximando do fim do circuito, o método de “disparar” que nos era proposto, nos pareceu complicado, pois, imaginávamos o "botão" sendo como um interruptor, porém descobrimos que a função dele era “mandar” 1 em sua saída, então o conectamos a uma porta “or”, sendo assim, se a saída já era verdadeira, 1 vezes 1, permaneceria como “acertou”, em caso contrário, multiplicado por 0, o resultado se conservava.