UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

GEN253 - CIRCUITOS DIGITAIS

Alunos: David Fambre Mezadri / Gabriel Balbinot

Trabalho Final - Batalha Naval

Link vídeo de apresentação: https://www.youtube.com/watch?v=qLMLDFnhGKE

Estratégia aplicada

A estratégia por trás da implementação começou a partir da análise do problema, quatro entradas para determinar uma saída: saber se o disparo acertou ou não. Sabendo disso, foi analisada a codificação selecionada (nº 18) para determinar o melhor caminho para resolução do problema. Decidimos utilizar um codificador que recebe quatro entradas (os quatro bits de localização do disparo do jogador) e gera quatro saídas (os quatro bits de localização no “mar”). Tendo isso em vista, fizemos a tabela verdade para cada um dos bits de saída, nomeados S1, S2, S3 e S4, onde S1 representa o bit mais significativo da saída codificada, e S4 o menos significativo. As entradas A, B, C e D representam os bits do disparo, sendo A o mais significativo e D o menos significativo. Segue abaixo a matriz 4x4 com os valores codificados e a tabela verdade do circuito:

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

A	B	C	D	S1	S2	S3	S4
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	0	1	1
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	1	0
1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	0	1
1	1	1	1	1	0	0	1

Seguem abaixo as expressões algébricas utilizando soma de mintermos:

S1(A,B,C,D) = ∑m(0, 1, 2, 3, 5, 7, 10, 15)

S2(A,B,C,D) = ∑m(0, 1, 2, 4, 6, 9, 10, 11)

S3(A,B,C,D) = ∑m(0, 1, 3, 6, 7, 8, 11, 13)

S4(A,B,C,D) = ∑m(0, 2, 3, 4, 6, 13, 14, 15)

Segue abaixo as expressões algébricas simplificadas:

S1(A, B, C, D) = A'B' + A'D + B'CD' + BCD

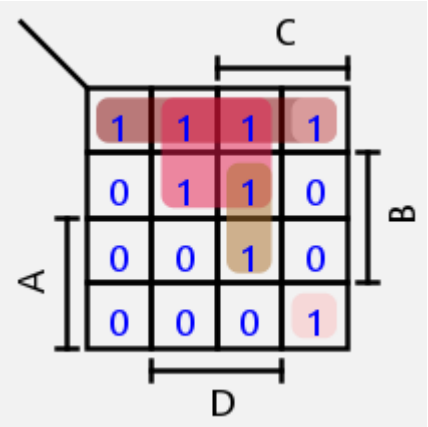
S2(A, B, C, D) = A'D' + B'C'D + AB'C

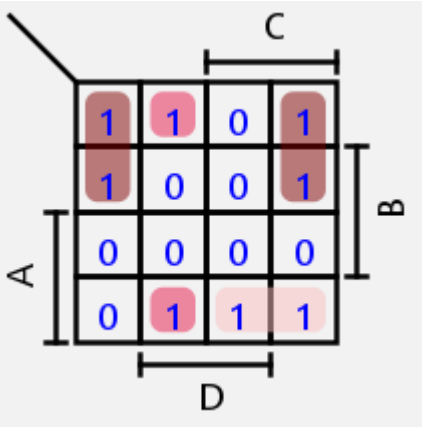
S3(A, B, C, D) = A'BC + B'C'D' + B'CD + ABC'D + A'B'C'

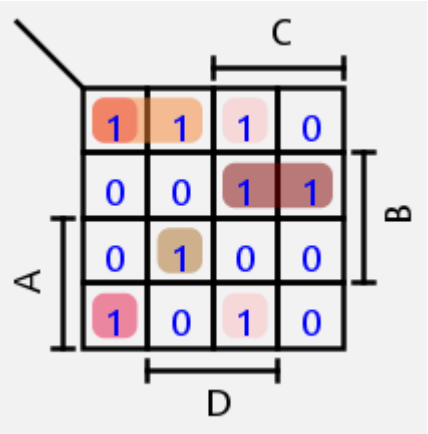
S4(A, B, C, D) = A'D' + A'B'C + ABD + BCD'

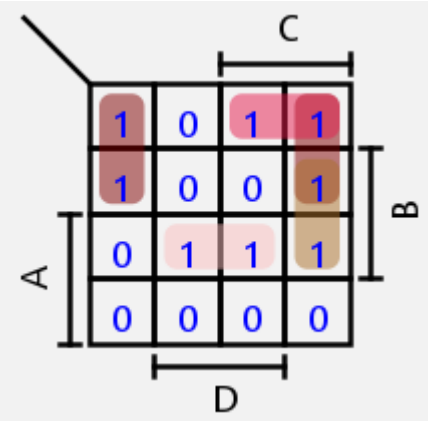
Seguem os mapas de Karnaugh:

Em cima, S1 e S2 (respectivamente); em baixo S3 e S4 (respectivamente):









Outro passo importante para a resolução é comparar cada bit codificado da entrada com os bits de posição dos navios. Nesse sentido utilizamos uma porta XNOR, visto que a saída dessa porta é 1 se ambos os bits forem iguais. Na imagem abaixo podemos ver representado uma tabela verdade de uma porta XNOR, onde Nx é o bit de um navio, Sx a saída do codificador e Cx o resultado da comparação entre os bits (x representa a posição do bit na entrada/saída). A tabela verdade abaixo representa um comparador de bits.

Nx	Sx	Cx
0	0	1
0	1	0
1	0	0
1	1	1

A utilização deste circuito comparador de bits se dá dentro de outro circuito, um circuito comparador. Ele recebe oito entradas, sendo quatro delas representando os bits do disparo, e outras quatro representando os bits da posição do navio. A saída do comparador será 1 apenas quando o disparo acertar o navio em questão. Em

outras palavras, quando os quatro comparadores de bits tiverem saída 1. Segue tabela verdade representando essa situação:

Cada entrada com começo CB representa a saída de um comparador de bit, e a saída C representa o acerto do disparo em um navio. A saída C é o resultado da expressão CB1*CB2*CB3*CB4.

cB1	cB2	cB3	cB4	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Utilizamos dois comparadores, um para cada navio. Desta forma temos uma expressão onde o acerto do disparo se dará caso acerte qualquer um dos dois navios (ambos inclusive). Caso o jogador que controla os navios coloque ambos na mesma posição, ele facilitará o jogo para o outro jogador, que precisará acertar apenas uma posição para vencer. Segue tabela verdade:

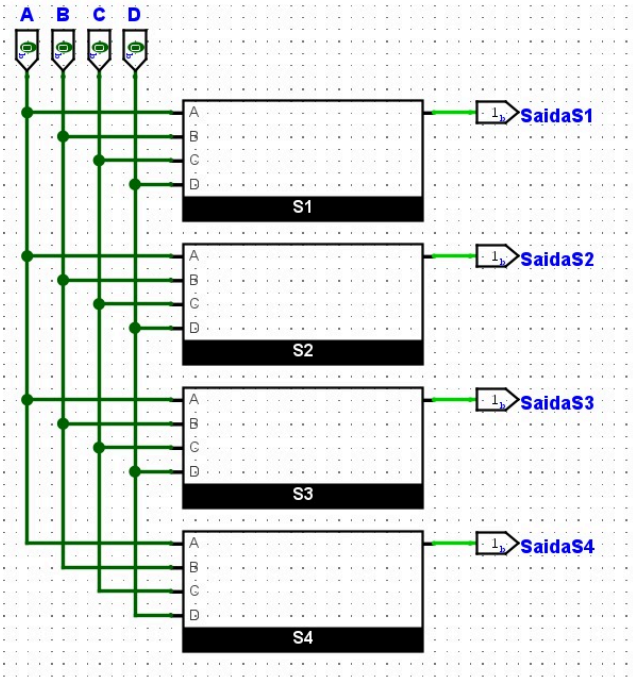
C1	C2	Acertou
0	0	0
0	1	1
1	0	1
1	1	1

O jogo se encerra quando o jogador que efetua o disparo destruir ambos os navios, isto é, acertar a localização de ambos.

Descrição logisim

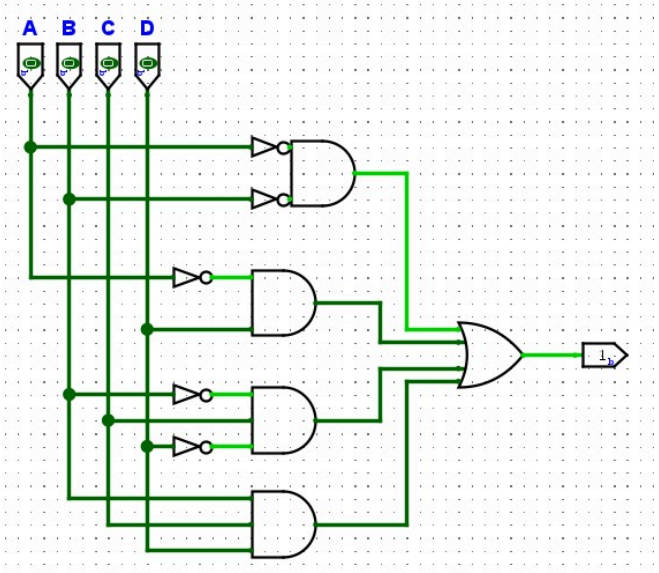
Seguindo a lógica e os circuitos apresentados, seguem imagens e descrições do circuito completo no Logisim.

Codificador



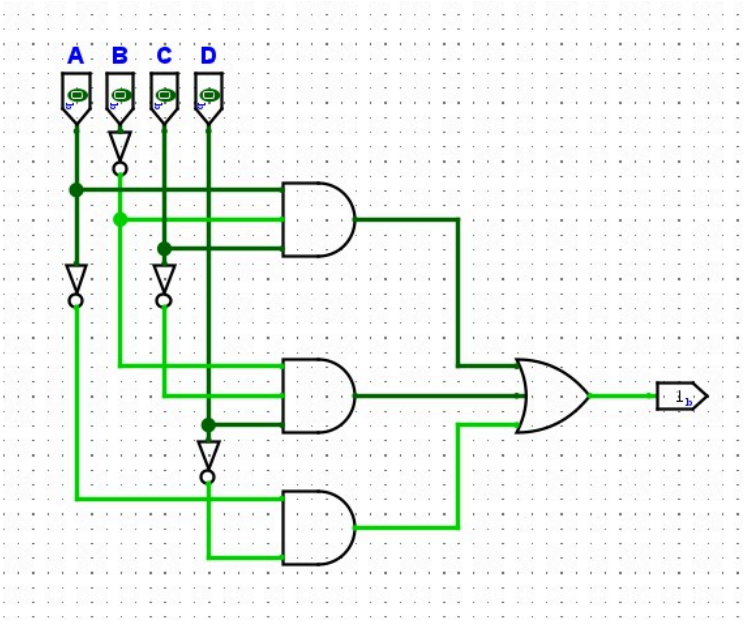
Comecemos pelo codificador: ele recebe quatro entradas. Essas mesmas entradas passam por quatro circuitos (S1, S2, S3 e S4) e cada um desses circuitos gera uma saída. Nosso codificador gera quatro saídas.

S1



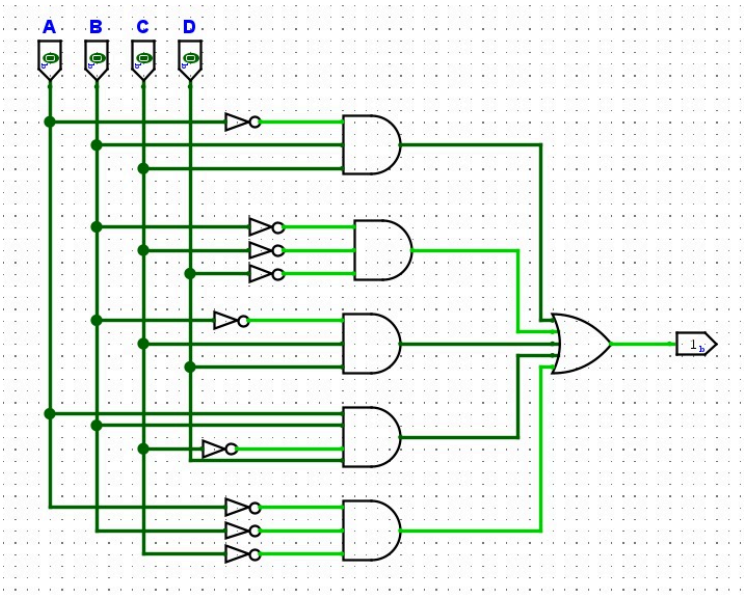
O circuito acima representa a expressão S1(A, B, C, D) = A'B' + A'D + B'CD' + BCD.

S2



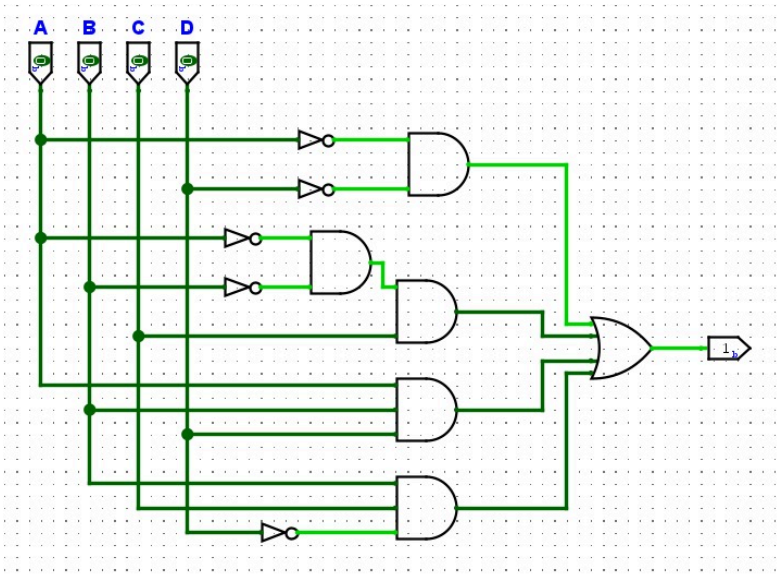
O circuito acima representa a expressão S2(A, B, C, D) = A'D' + B'C'D + AB'C.

S3



O circuito acima representa a expressão S3(A, B, C, D) = A'BC + B'C'D' + B'CD + ABC'D + A'B'C'

S4

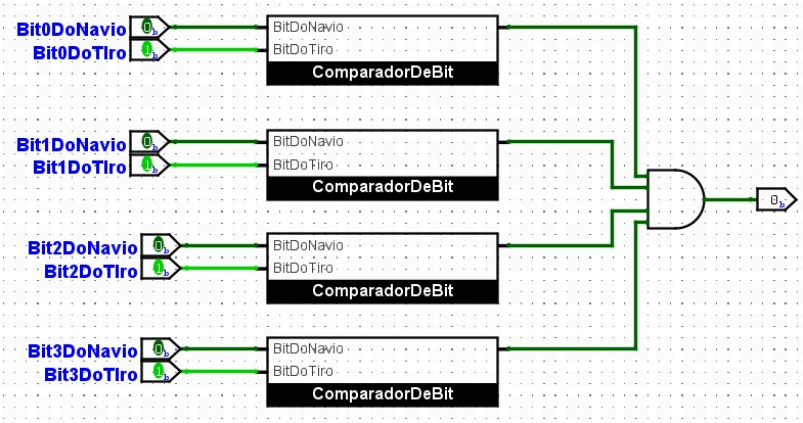


O circuito acima representa a expressão S4(A, B, C, D) = A'D' + A'B'C + ABD + BCD'

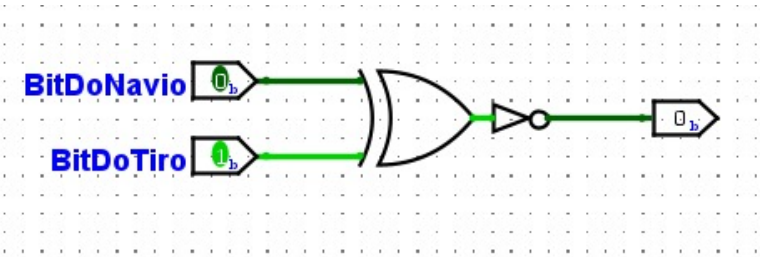
Cada uma dessas saídas (S1, S2, S3, S4) do codificador representam um bit codificado, que serão utilizados para fazer a comparação com os bits de posição de um navio.

Comparador

Cada uma das saídas do codificador servem de entrada para o comparador. As outras quatro entradas são ligadas diretamente da posição em que um navio está codificado.

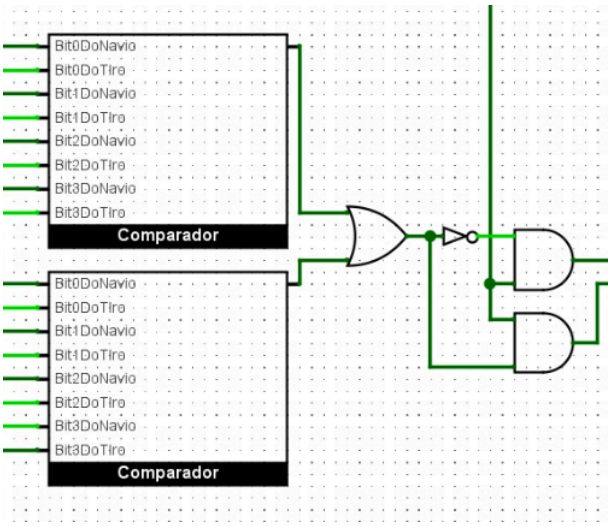


Cada bit passa por outro circuito que compara os dois bit de mesma casa (mais significativo, menos significativo, etc):



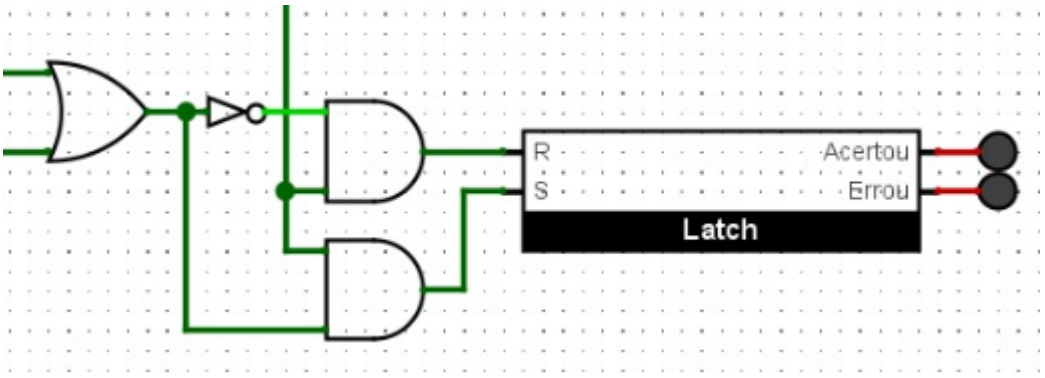
Esse circuito implementa uma XNOR, cuja saída será 1 sempre que ambos os bits forem iguais. A saída do comparador será 1 caso todas as saídas dos mini comparadores bit a bit forem 1, representando que o disparo de forma codificada acertou um navio (previamente codificado)

Como são dois navios, utilizamos dois comparadores e usamos a saída de ambos para definir se um tiro acerto ou não:



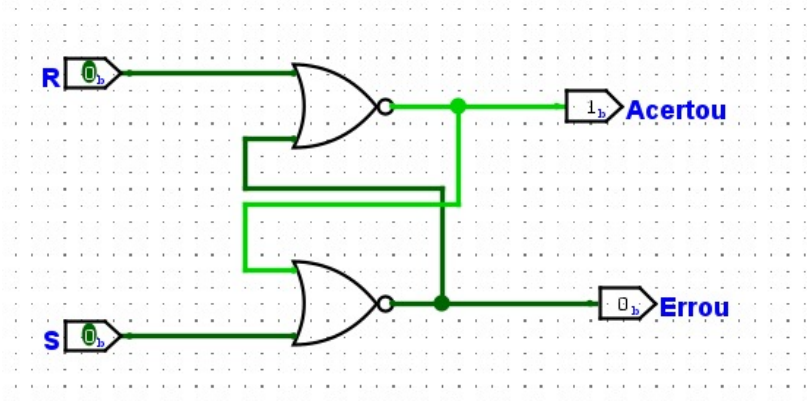
As saídas de ambos os comparadores passam por uma porta OR, indicando 1 quando acerta ao menos um navio. Podemos ver também duas portas AND, onde ambas recebem a entrada do pressionamento do botão de disparo. O que as diferencia é que uma recebe a saída da porta OR negada, enquanto a outra é ligada diretamente a saída da porta. Isso se torna importante quando analisamos o restante do circuito.

Latch RS



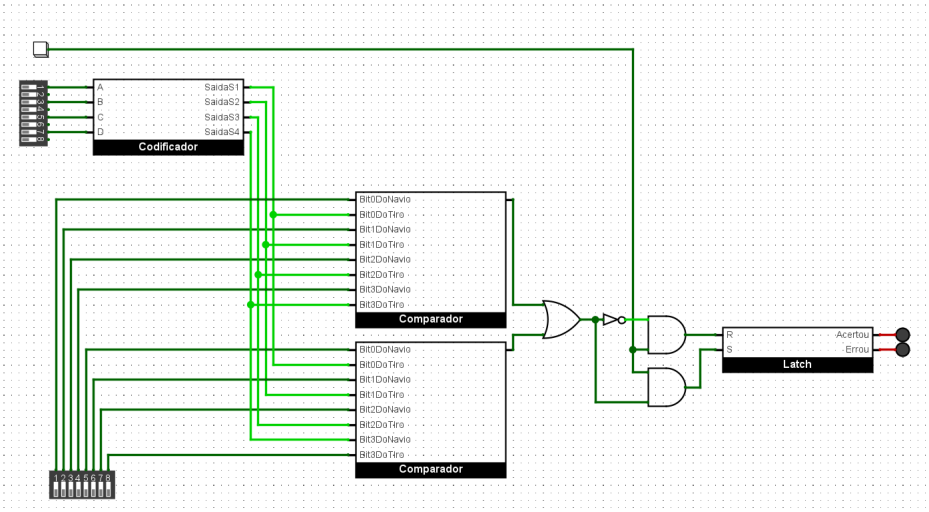
Utilizamos um circuito Latch RS para controlar qual LED estará aceso. Caso não tivéssemos usado um, o LED acenderá por um breve momento apenas. Queremos que o LED permaneça aceso, representando o resultado do último disparo feito. Seguindo a lógica de um Latch RS, sendo R para reset e S para set, o LED

vermelho acenderá quando o disparo errar, o estado do circuito será reiniciado, tendo a saída Q como 0 e a saída -Q como 1. O inverso acontece quando o disparo acertar, acionando a opção set de um latch, fazendo com que Q assuma o valor 1, e -Q o valor 0. Os LEDs são ligados diretamente às saídas do Latch RS, fazendo com que o LED acenda dadas as condições descritas acima. Segue abaixo o circuito Latch RS:



Circuito completo

Optamos por utilizar um botão para efetuar o disparo, e dois interruptores DIP, sendo um para controlar as posições dos navios e outro para a posição do disparo. O DIP que controla a posição do disparo utiliza apenas os interruptores de número ímpar. Quando alterados para ON, a entrada será 1. O outro DIP utiliza todos os interruptores, sendo os quatro primeiros (1, 2, 3, 4) para definir a posição do primeiro navio, e os quatro últimos (5, 6, 7, 8) para o segundo navio.



Descrição do trabalho no Tinkercad

https://www.tinkercad.com/things/8pO1tXC11Dj/editel?returnTo=%2Fdashboard&sharecode=Xe-DLb3KkVMzDaPlBoaStuN38nr23EfG-K6--ztLN0o

Sabendo que as entradas S1 a S4 são os bits, optamos por fazer em cada protoboard um bit a fim de manter a organização e facilitar o entendimento. Portanto, temos organizado em ordem crescente do S1 a S4.

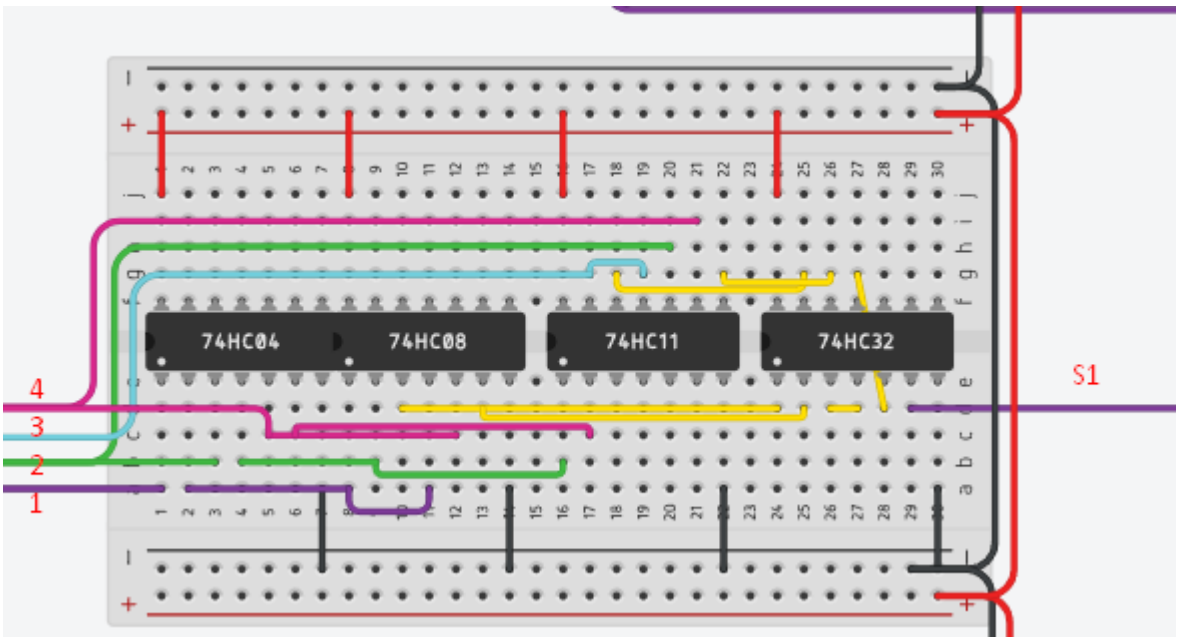
Cores:

Vermelho - VCC
Preto - GND
Roxo - Bit 1 (Mais significativo)
Verde - Bit 2 (Segundo mais significativo)
Ciano - Bit 3 (Terceiro mais significativo)
Rosa - Bit 4 (Menos significativo)
Marron - Resultado parcial, necessitando de mais somas (OR)
Amarelo - Resultado final de um mintermo ou resultado final de um bit(S)

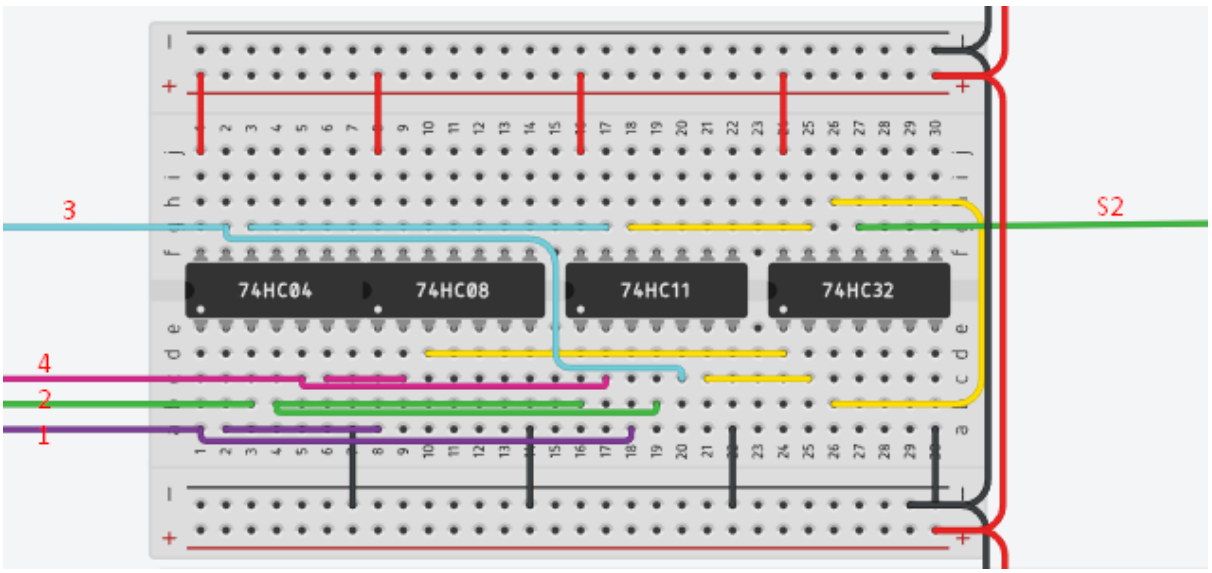
Resultados dos BITS

1. **S1(A, B, C, D) = A'B' + A'D + B'CD' + BCD**
    a. Para execução do primeiro produto (A'B'), o bit 1 e 2 passaram pelo CI 74HC04 e seus resultados foram para o CI 74HC08, tendo como resultado A'B'.
    b. O próximo produto (A'D) busca o resultado do bit 1 após passar pelo CI 74HC04 e juntamente com o bit 4 seguem para o CI 74HC08, com resultado A'D.
    c. O terceiro produto (B'CD') leva o bit 2 e 4 para o CI74HC04 e juntamente com o bit 3 caem no CI 74HC11, resultando em B'CD'.
    d. Para o último produto (BCD) temos o bit 2, 3 e 4 seguindo diretamente para o CI 74HC11, e como resultado imediato temos BCD.
    e. Por fim temos a junção do primeiro e do segundo produto, concomitantemente temos a junção dos produtos terceiro e quarto,

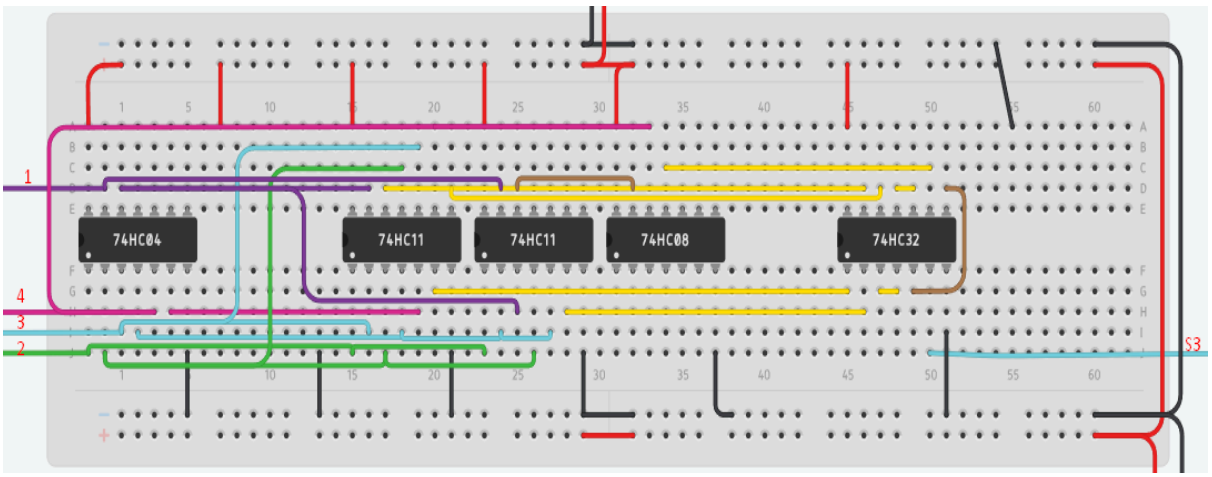
todos no CI74HC32, fazendo a soma dos mesmos. Após isso, somamos seus respectivos resultados e temos o dado final S1.



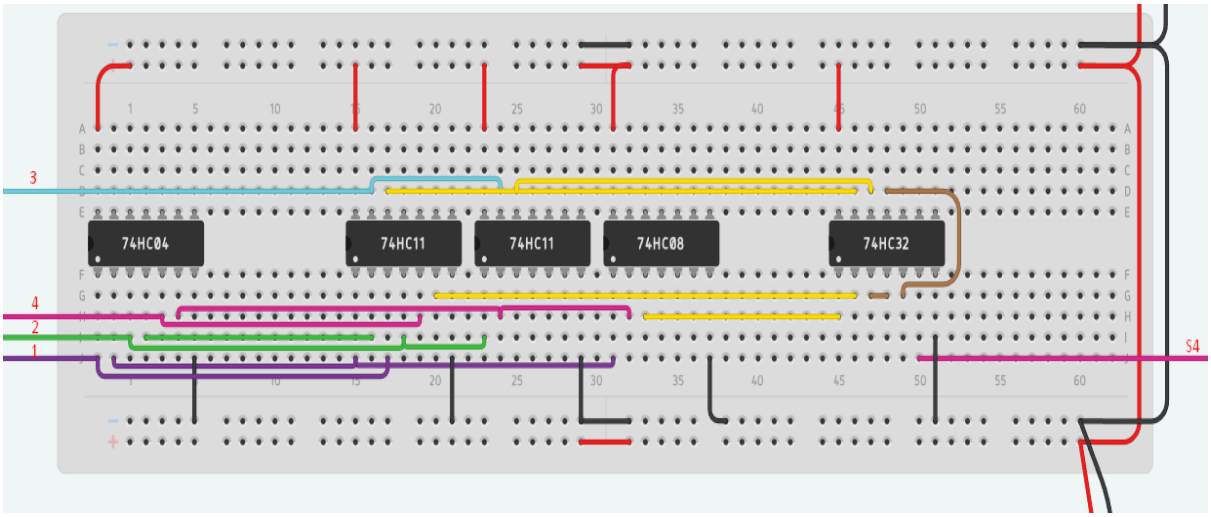
2. **S2(A, B, C, D) = A'D' + B'C'D + AB'C**
    a. Para primeiro produto (A’D’) temos o bit 1 e 4 passando pelo CI 74HC04 e seu respectivos resultado indo para o CI 74HC08 resultando em A’D’
    b. Para o segundo produto (B’C’D) temos o bit 2 e 3 passando pelo CI74HC04 indo juntamente com o bit 4 para o CI 74HC11, resultando em B’C’D
    c. Por fim, mas não menos importante temos o bit 2 passando pelo CI 74HC04 e indo juntamente com os bits 1 e 3 para o CI 74HC11, resultando em AB’C.
    d. Após isso, todos os resultados seguem para o CI 74HC32, somando-se primeiro os produtos 1 e 2.
    e. Para finalizar, o resultado do primeiro e do último produto são somados no CI 74HC32 e seu resultado é somado com o segundo produto, finalizando a soma de todos os produtos.



3. **S3(A, B, C, D) = A'BC + B'C'D' + B'CD + ABC'D + A'B'C'**
    a. Para o primeiro produto (A’BC) temos o bit 1 passando primeiro pelo CI 74HC04 e depois seguindo juntamente com os bit 2 e 3 para o CI 74HC11, retornando o resultado A’BC.
    b. Para o segundo produto (B’C’D’) temos os bits 2, 3 e 4 passando pelo CI 74HC04 e seus resultados seguindo para o CI74HC11, assim, obtemos o retorno de B’C’D’.
    c. Depois temos o bit 2 passando pelo CI 74HC04 e seguindo para o CI 74HC11 juntos com os bits 3 e 4, resultando em B’CD.
    d. Após isso temos o nosso produto mais longo (ABC’D), pois primeiramente temos nosso bit 3 passando pelo CI 74HC04 e seu resultado seguindo com os bits 1 e 2 para o CI 74HC11 seu resultando segundo para o CI 74HC08 juntamente com o bit 4, resultando em ABC’D.
    e. Depois temos nosso último produto (A’B’C’), com o resultados dos bits 1, 2 e 3 após passarem pelo CI 74HC04 e finalizando no CI 74HC11 para retornar a operação A’B’C’.
    f. Por fim, mas não menos importante, temos o resultado do primeiro e terceiro produto sendo somados e seu resultado é somado com o quarto produto, assim o resultado disso é somado com o resultado da soma do segundo e último produto, resultando na soma de todos os 5 produtos.



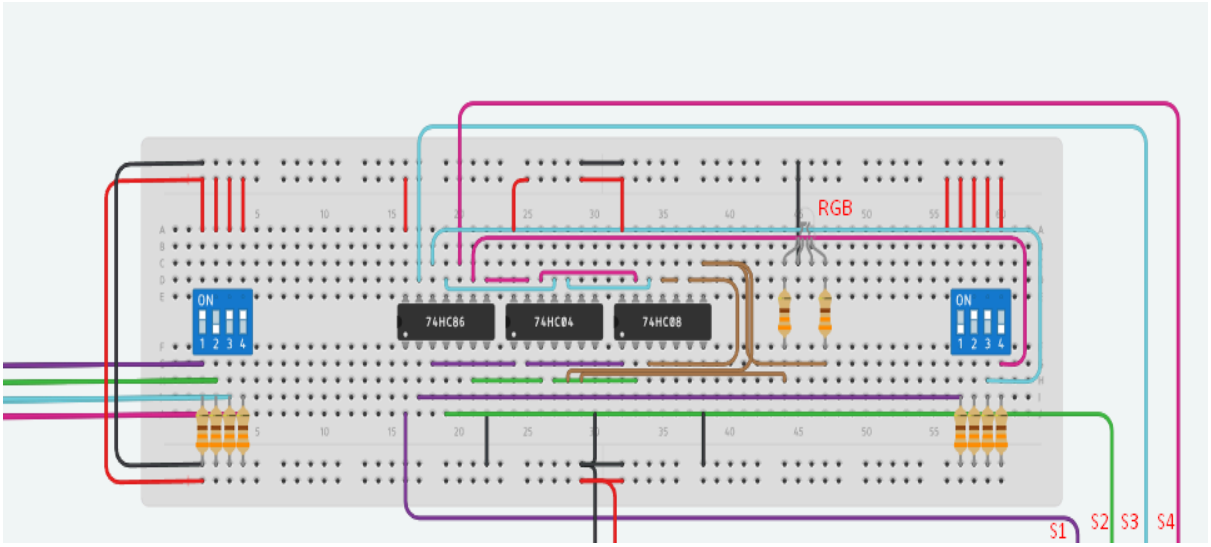
4. **S4(A, B, C, D) = A'D' + A'B'C + ABD + BCD'**
    a. Para iniciar, temos o bit 1 e 4 provenientes do botão vindo para o CI 74HC04 e seu resultado seguindo para ser multiplicado no CI 74HC08 e sua resultante é A'D'.
    b. Depois temos o bit 1 e 2 sendo negados no CI 74HC04 e seguindo para o CI 74HC11 juntamente com o bit 3, tudo isso para resultar em A'B'C.
    c. Logo após temos os bits 1, 2 e 4 seguindo somente para o CI 74HC11 e já retornando ABD.
    d. Por fim temos o bit 4 passando pelo CI 74HC04 e seguindo juntamente com os bits 2 e 3 para o CI 74HC11, para resultar em BCD'.
    e. Para terminar temos o primeiro produto somando com o terceiro e seu resultado somado com resultado somado com o resultado da soma do segundo produto com o último produto.



Os resultados da codificação retornada pelos S1, S2, S3 e S4 vão para uma porta XNOR com seus respectivos bits provenientes do interruptor da posição do navio. Contudo, no Tinkercad não temos o CI da porta XNOR então implementamos usando uma porta XOR e NOT.

Assim o resultado da passagem dos pares de bits que passaram pelo CI 74HC86 (XOR) e depois no CI 74HC04, só obteremos uma luz verde se todos os bits coincidirem, por tanto utilizamos um CI de portas ANDs e seguimos passando resultado da XNOR do Bit 1 com o Bit 2, também o Bit 3 e o 4 e por fim comparamos os dois resultados.

Por fim, se retornar o valor ligado acenderá verde, ligando a perna do green do led RGB, caso contrário esse sinal passa pela porta NOT no CI 74HC04 e é ligado à perna red, emitindo luz vermelha e indicando que houve erro ao acertar o navio.



Vale ressaltar quer os CIs são posicionas com a meia lua virada para a esquerda e ligados ao VCC no pino 14 e o GND no pino 6. Além disso, temos interruptores com o pullback para que quando desligado haja descarregamento do lixo contido no sistema resultando em bit zerado e quando ligado a corrente apta a passar pelo caminho mais fácil, ou seja, desviando da resistência.

Também é importante dizer que o LED RGB tem seus pólos cantados antecipados por uma resistência. Resistência essa igual para todos as partes do circuito, com as primeiras dois aneis de cores fixadas na laranja e o terceiro na cor

marrom, resultando em 33 *10 = 330, conforme podemos observar na imagem abaixo:

Cor	1ª Faixa	2ª Faixa	N° de zeros/multiplicador	Tolerância
Preto	0	0	0	
Marrom	1	1	1	
Vermelho	2	2	2	
Laranja	3	3	3	
Amarelo	4	4	4	
Verde	5	5	5	
Azul	6	6	6	
Violeta	7	7	7	
Cinza	8	8	8	
Branco	9	9	9	
Dourado			x0,1	
Prata			x0,01	
Sem cor				± 20%

Adição de mais um navio

Após ter terminado o projeto, visualizamos que era necessário ter dois navios para aumentar as chances de acerto do usuário. Portanto era necessário a implementação de mais 4 protoboards com a mesma lógica de codificação. Foi então que optamos por não fazer isso e utilizar a lógica do flip flop, onde se houvesse ao menos 1 acerto o CI guardaria um bit do acerto, retornando a luz verde.

Mais precisamente utilizamos o mesmo codificador porém alternando os grupos de coordenadas colocadas pelo jogador. Ligamos as saídas dos interruptores em série e as mesmas seguiam direto para o codificador, entretanto o que definia qual ia é o flip flop que alterna toda vez que se aperta o botão de tiro, portanto fica alternando entre um conjunto de coordenadas e outro. Tivemos que por diodos para que os sinais não voltassem e atrapalhasse a codificação do outro interruptor. Tivemos que por um transistor para permitir a passagem ou não dos 5v do VCC, já que se tentássemos alimentar os interruptores com o sinal vindo do CI 74HC73 o mesmo não suportava tanta carga.

Com dois botões temos 1 de tiro que deve ser apertado duas vezes para varrer as duas coordenadas e um botão inferior responsável pelo reset do game. Se a luz ficar verde após dois clique no botão superior, que o responsável pelo disparo então você acertou ao menos 1 navio, caso permaneça vermelho, não houve acerto. É interessante que mesmo que aperte o botão de tiro mais vezes, se acertou a coordenada a cor verde se mantém até o botão reset ser pressionado, isso se dá pois o bit de comparação das coordenadas entra J enquanto o K se mantém sempre em zero, portanto J = 0 E K = 0 mantém o Q = 0, se houver J = 1, o próximo

clock(apertar de botão) mudará o Q = 1, e já que o K é sempre 0, o Q não voltará a ser 0.