Universidade Federal da Fronteira Sul - Campus Chapecó

# Trabalho Final de Circuitos Digitais: Relatório - Batalha Naval

Aluno: Douglas Kosvoski
Matrícula: 1911100022
CCR: Circuitos Digitais 2º Fase - Vespertino
Professores: Luciano L. Caimi e Adriano S. Padilha

12 de dezembro de 2019

- Apresentação

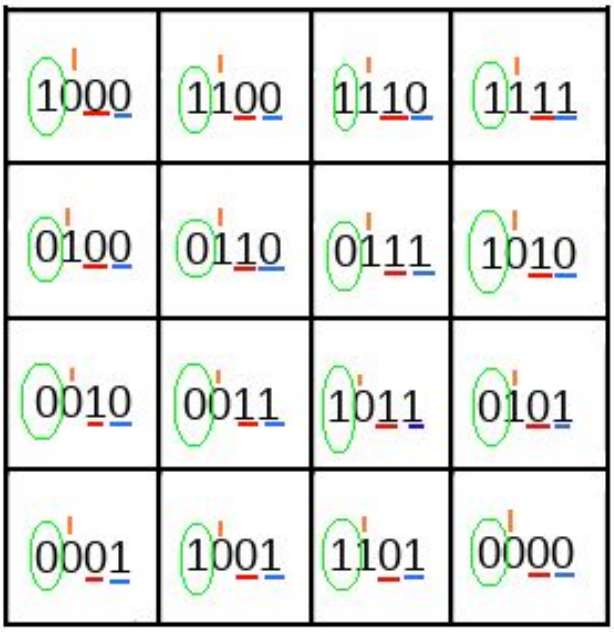
O trabalho final da disciplina Circuitos Digitais consiste em codificar e implementar simulações do jogo Batalha Naval, jogo de tabuleiro de dois jogadores, no qual os jogadores têm de adivinhar em que quadrados estão os *navios* do oponente. Seu objectivo é derrubar os barcos do oponente, ganha quem derrubar todos os navios adversários primeiro.

Formado por uma matriz 4x4 (16 posições, tendo cada posição uma codificação associada) com dois barcos dispostos sobre ela, tendo a solução implementada em diversas plataformas, entre elas estão: TinkerCad, Logisim, Quartus Web Edition e Altera DE1.

Os comandos se baseiam em 2 conjuntos de chaves codificadas para os barcos (4 *Keys* para cada barco), 1 conjunto de chaves sem codificação para o adversário (4 *Keys*), bem como um *switch* de disparo (o disparo é realizado na posição real da matriz de acordo com o *input* do adversário). Tendo como retorno *LED's* de coloração verde (em nível lógico alto) para quando o disparo acertar o alvo e *LED's* de tom avermelhado (em nível lógico alto) para quando o alvo escapar ileso do disparo.

- Descrição da Solução

A solução encontrada e utilizada em todo o desenvolvimento do trabalho consiste em criar uma codificação qualquer (que já não estivesse sendo utilizada por outros grupos) para cada posição da matriz, e em seguida a transformar em expressões lógicas utilizando Álgebra de Boole levando em consideração a codificação a partir da posição de cada caracter.




| C1 | C2 | C3 | C4 | A | B | C | D |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| CODIFICADAS | | | | ORIGINAIS | | | |

Percebe-se que, cada posição da matriz tem uma codificação distinta associada e cada dígito da posição da matriz está associada a uma sequência de módulos (conjunto de codificações) e com uma tabela-verdade respectivamente.

Exemplo:

A posição 1001 na matriz (posição original) para a ser considerada, a partir da codificação, a posição 0011 (posição codificada), de acordo com o seguinte passo a passo:

Todos os dígitos de 1001, ou seja, A,B, C e D participam em todos os módulos de codificação, portanto, A B C e D entram no primeiro codificador passam por diversas portas lógicas e seus resultados nos retornam o valor C1 (primeiro dígito da posição codificada, dígito 0 da posição 0011), e assim por diante também com os outros

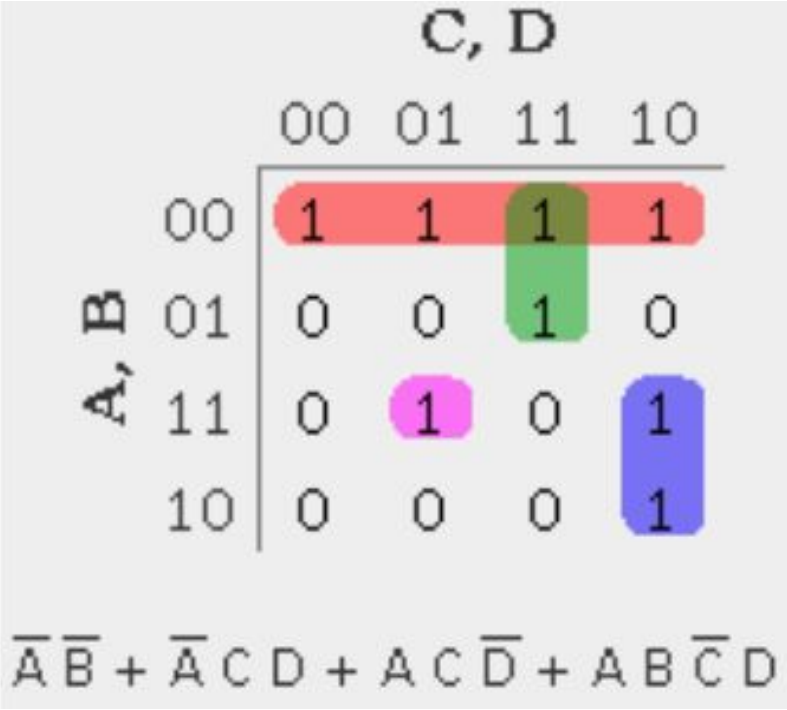
codificadores, nos retornando também os valores de C2, C3 e C4. Resultados que quando colocados lado a lado nos mostram a posição 0011 (codificada) que originalmente foi inserida 1001, ou seja, entra 1001 no circuito e sai 0011. E esse procedimento remete a qualquer posição inserida para os barcos, lembrando que apenas as posições dos barcos são codificadas, no que se refere as chaves do usuário apenas são feitas comparações com as chaves codificadas dos barcos.

- Procedimento interno de cada módulo de codificação

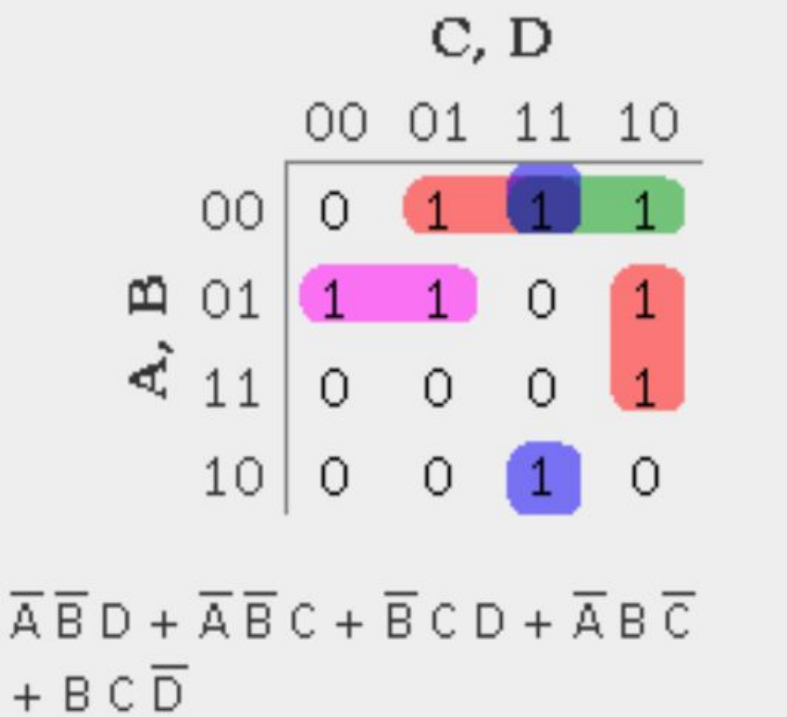
Ao pegarmos respectivamente todos os dígitos A, B, C e D de cada posição da matriz e fazermos uma tabela-verdade por coluna temos a tabela anterior, agora por Soma de Produtos, sequencialmente pegamos todos os valores de nível lógico alto, ou seja, 1 e aplicamos o Mapa de Karnaugh, com a ajuda do programa Logisim para então fazermos as simplificações necessárias.

Temos por tanto, os mapas de Karnaugh dos módulos de codificação juntamente com a expressão lógica resultante, vemos em seguida C1, C2, C3 e por fim C4.

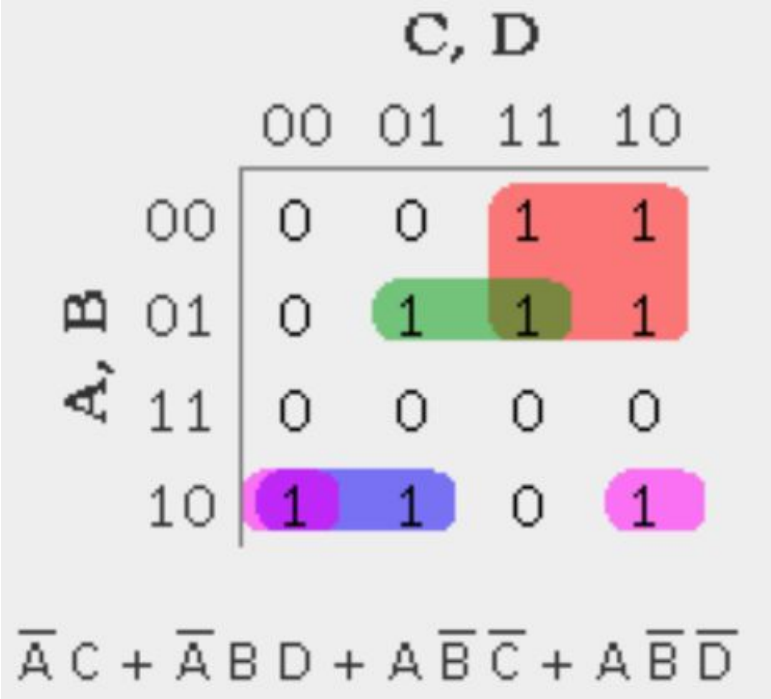
Mapa C1 - minitermos {0,1,2,3,7,10,13,14}



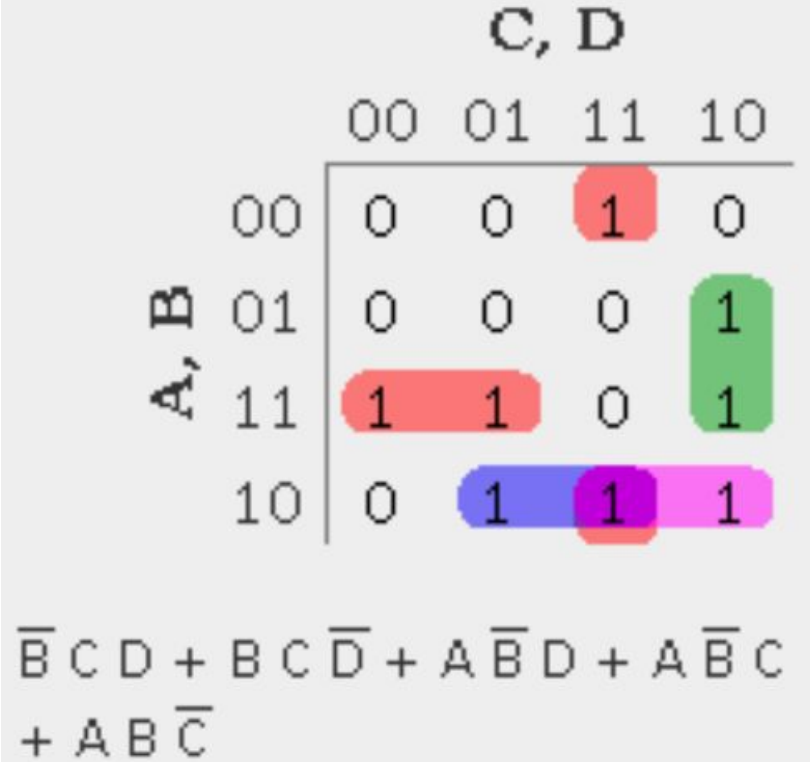
Mapa C2 - minitermos {1,2,3,4,5,6,11,14}




Mapa C3 - minitermos {2,3,5,6,7,8,9,10}



Mapa C4 - minitermos {3,6,9,10,11,12,13,14}




Visto que agora temos as Tabelas-Verdades, Mapas de *Karnaugh* com as Simplificações e Equações Lógicas, podemos aplicar as equações nas diversas plataformas.

Vamos começar por aplicar no programa *Logisim,* em seguida *TinkerCad* e por fim mas não menos importante *Quartus* juntamente com a placa *Altera DE1.*

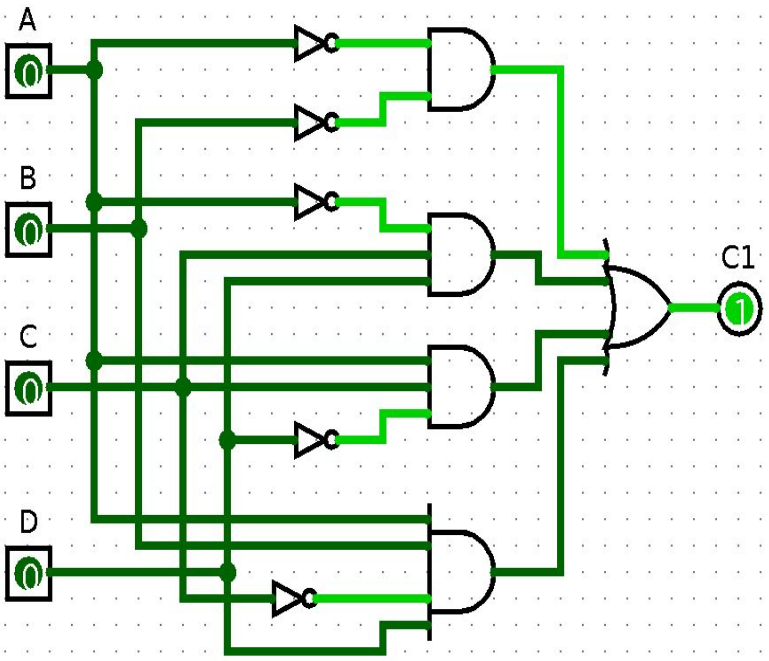
## *Logisim*

*O programa Logi*sim se mostra extremamente fácil e intuitivo de se utilizar, em questão de segundos a partir da escolha de como será gerado o circuito, seja por meio da inserção da tabela-verdade ou da equação lógica, e da escolha das variáveis de entrada e saída, nos é apresentado de prontidão o circuito resultante (pessoalmente escolhi inserir individualmente as tabelas-verdades de cada codificador, tanto para me assegurar do resultado final, quanto para escolher em qual arquivo cada codificador seria salvo, no meu projeto também escolhi por salvar cada módulo em um arquivo separado).

Vamos a inserção das tabelas-verdades no programa.

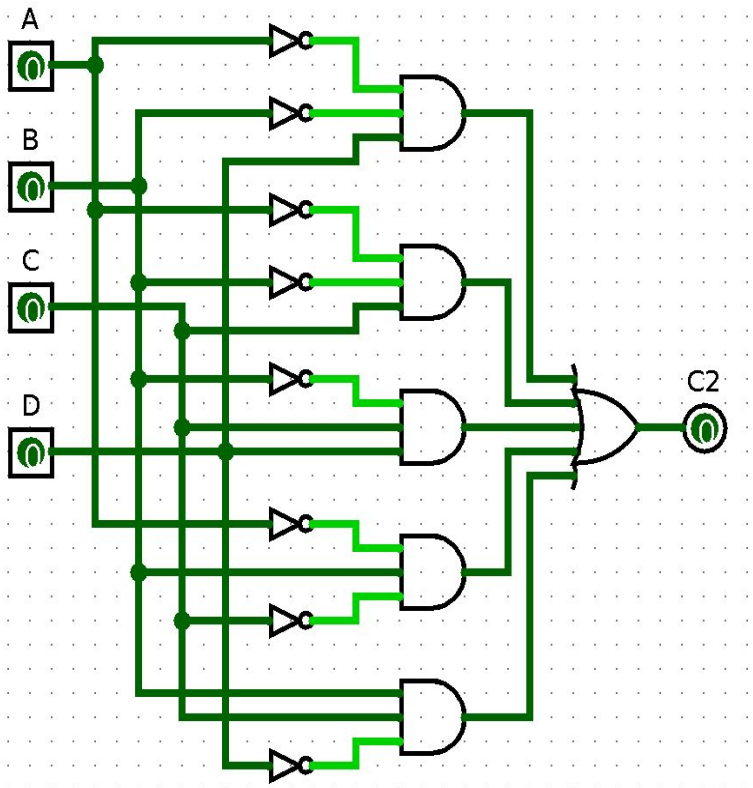
Ao inserirmos a tabela do **primeiro** dígito e atribuímos a variável de saída como sendo **C1**, temos o circuito codificador **C1**.

| A | B | C | D | C1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |



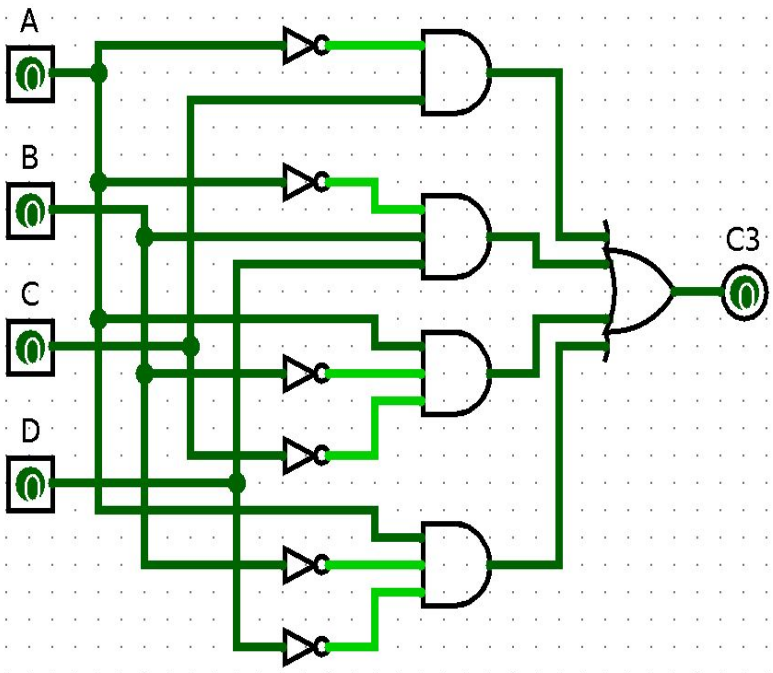
Ao inserirmos a tabela do **segundo** dígito e atribuímos a variável de saída como sendo **C2**, temos o circuito codificador **C2**.

| A | B | C | D | C2 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |



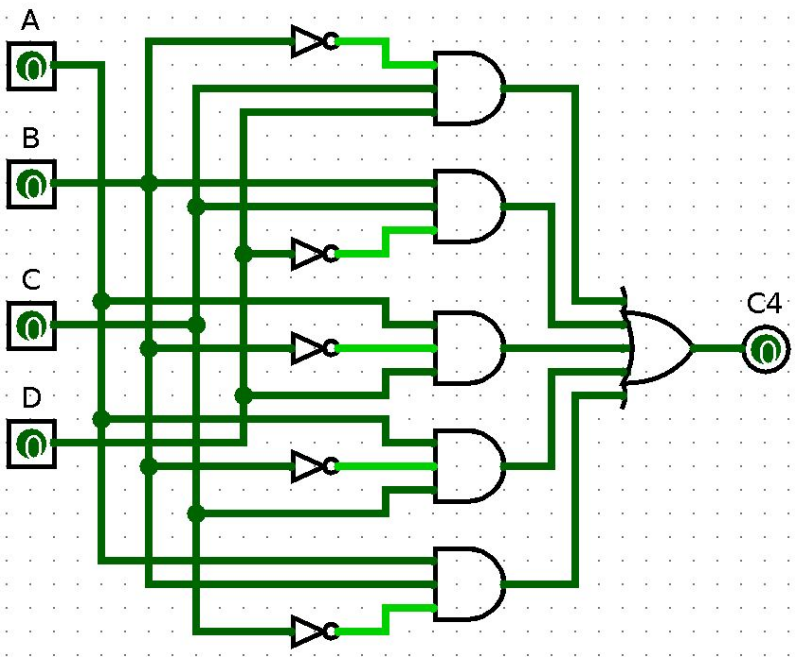
Ao inserirmos a tabela do **terceiro** dígito e atribuímos a variável de saída como sendo **C3**, temos o circuito codificador **C3**.

| A | B | C | D | C3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |



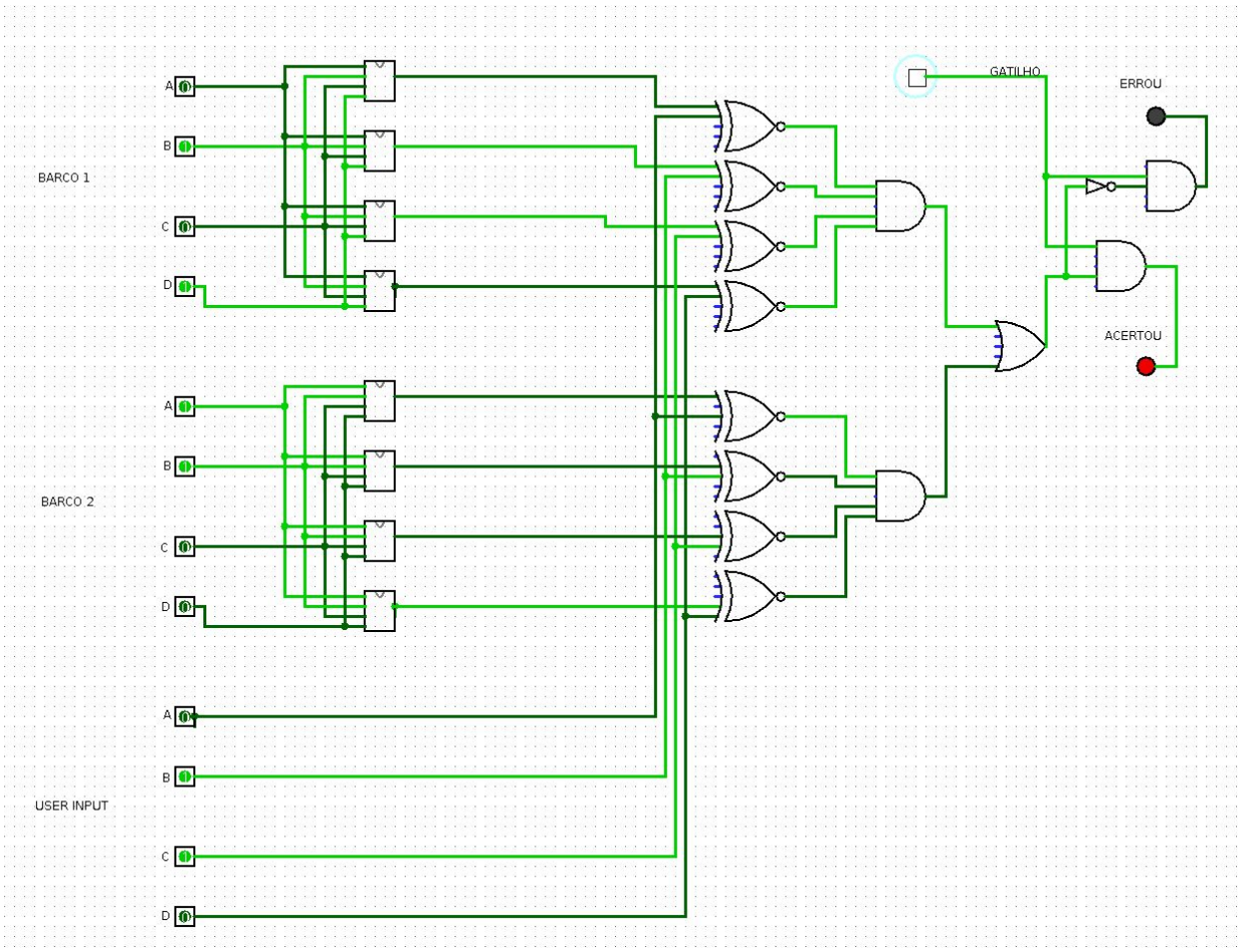
Ao inserimos a tabela C4 temos o codificador C4

| A | B | C | D | C4 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |




E com cada módulo (C1, C2, C3 e C4) podemos então junto com o *INPUT* do adversário formar o arquivo/estrutura *Main* (Junção dos módulos de codificação, entrada do adversário e comparador entre o resultado da codificação e *inputs).*

*Main*




Explicação:

BARCO 1 possui 4 entradas, cada uma com 2 opções de entrada (nível lógico baixo ou alto), assim como o BARCO 1, o BARCO 2 possui 4 entradas binárias (0 ou 1), essas 4 entradas correspondem ao A, B, C e D individuais dos barcos, que entram em cada um dos codificadores (que foram importados a partir do circuito gerado das tabelas-verdades anteriores).

Ainda em relação ao barcos, a saída desses codificadores são ligadas em uma XNOR (XOR negada) juntamente com a entradas do usuário (que tentará adivinhar as posições dos barcos por meio de quatro entradas de valores ou 0 ou 1) (C1 xnor A, C2 xnor B, C3 xnor C, C4 xnor D) o resultado dessas 4 portas XNOR é então enviado para uma AND, a AND é responsável por assegurar que as entradas do usuário estejam completamente corretas e não parcialmente (confirma que os quatro dígitos estejam

corretos). O mesmo procedimento de codificar as entradas do BARCO 1 e comparar com as entradas do usuário por meio de XNOR e AND, acontecem com o BARCO 2.

Por seguinte, temos uma porta OR que verifica se tanto o usuário adivinhou as posições do BARCO 1, quanto do BARCO 2, a mesma porta OR envia o sinal lógico agora para uma AND que analisará se o botão de **Gatilho** foi acionado, isso é de extrema importância, visto que, esse sistema é assíncrono, ou seja, independe de CLK (*clock*), o gatilho é portanto nossa entrada manual de sinal de clock, responsável por dar prosseguimento no circuito.

Se o resultado da OR for 1 (nível lógico alto) e o botão de disparo for acionado, reflete que o usuário descobriu a posição de nosso barco e por consequência aciona um LED de retorno ao usuário sinalizando seu acerto, porém se a saída da porta lógica OR for 0 (nível lógico baixo) e o gatilho for acionado um outro LED será acionado, agora sinalizando um disparo sem vítimas. Enquanto o disparo não for acionado nenhum LED será aceso.

## # Observação

Na imagem acima o comparador do retorno do codificador com a entrada do usuário está representada por uma XNOR de 5 entradas, porém devido a minha falta de familiaridade com o software acabei não encontrando uma XNOR de 2 entradas, por isso, quero deixar claro que, se implementado na vida real o desempenho do circuito não estaria garantido, visto que as entradas sem utilização poderiam vir a afetar seriamente o comportamento da porta lógica, acarretando em um resultado errôneo e impreciso, um ajuste temporário poderia ser ligar a entradas sem utilização da XNOR no GND visto que o nível lógico baixo é o elemento neutro da porta e não afetaria seu resultado.

O mesmo acontece na porta OR (que compara se qualquer um dos barcos foi acertado) e na porta AND (que verifica o acionamento do disparo), sendo necessário respectivamente que suas entradas sem utilização fossem ligadas em nível lógico baixo (OR) e alto (AND).

Na imagem acima podemos também analisar um exemplo de teste.

BARCO 1 tem suas entradas A, B, C e D em 0101 e a saída de seus codificadores em 0110 respectivamente para C1, C2, C3 e C4, por outro lado, o BARCO 2 recebe suas variáveis de entrada como sendo 1100 e saídas 0001.

Por sua vez o usuário insere a posição 0110 em suas chaves de entrada, que através do circuito podemos perceber que quando comparadas com a posição codificada do BARCO 2 não há acerto, no entanto, quando comparada com a posição codificada do BARCO 1, nos é retornado por meio do LED “ACERTOU”, que o usuário descobriu e efetuou o disparo corretamente no BARCO 1.

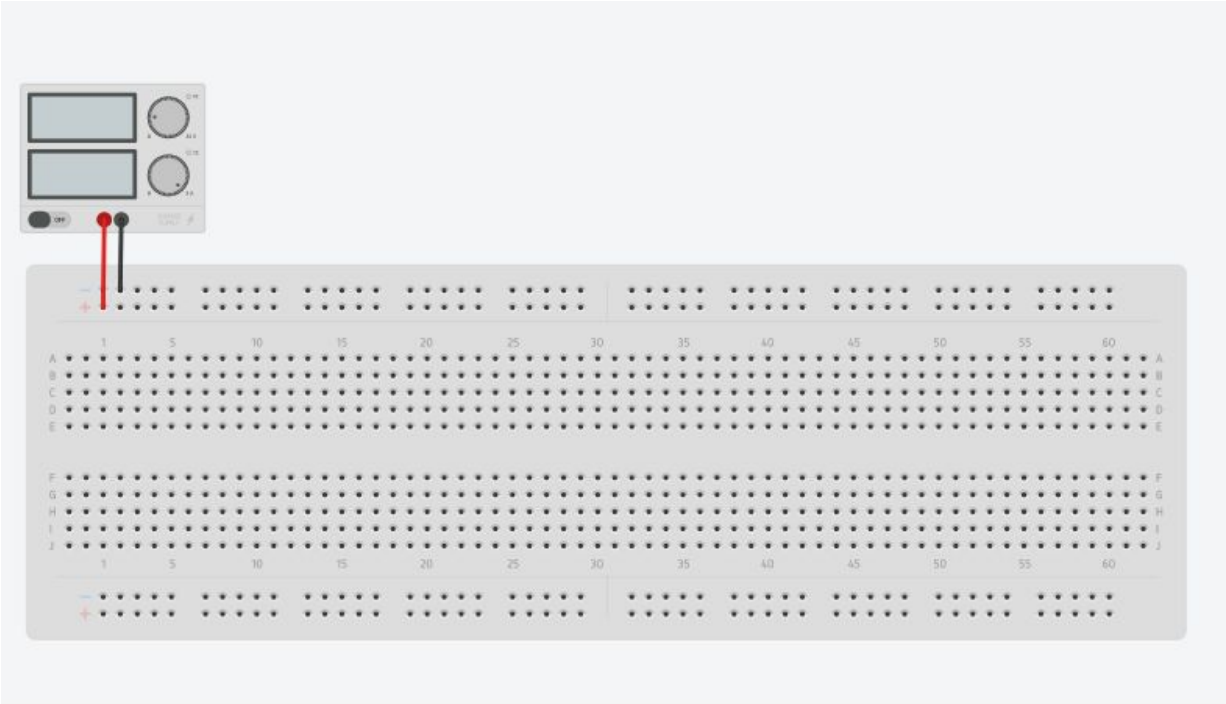
- *TinkerCad*

O *TinkerCad* é um *Website*, o qual simula circuitos e projetos 3D puramente online. Neste projeto utilizaremos suas funções de circuito, o qual nos permite emular o comportamento de CIs (Circuitos Integrados), placas de prototipação (*BreadBoard*), resistores, fontes de energia e entre outros recursos.

Antes de iniciarmos, gostaria de lembrar que a especificação da aplicação do projeto na plataforma, exigia a simulação dos codificadores de apenas um barco dos dois barcos.

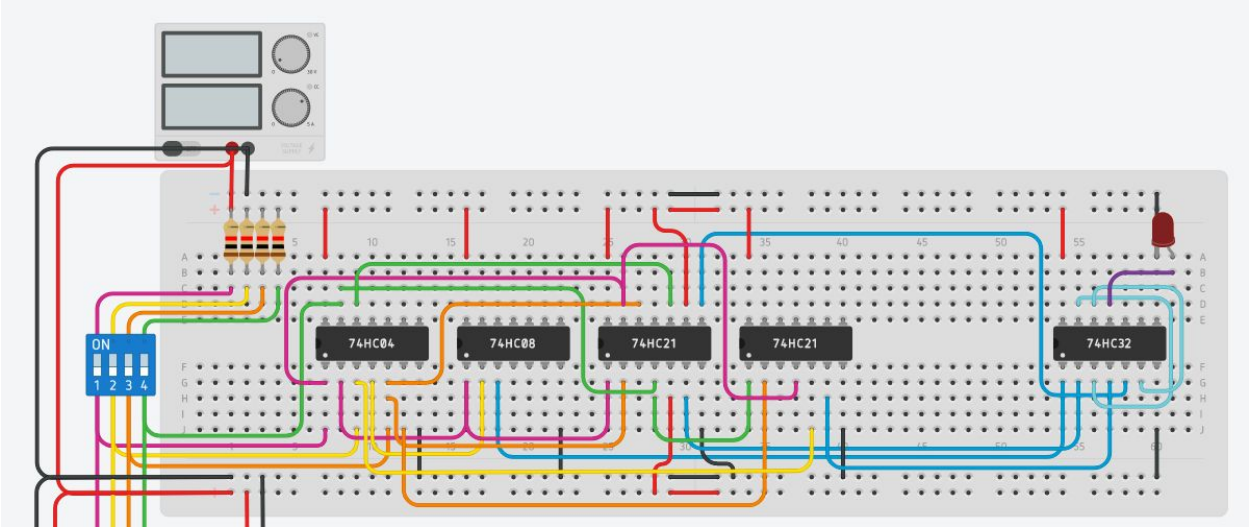
Primeiramente iniciemos por adicionar elementos fundamentais para nosso circuito.

A fonte de energia e nosso primeiro módulo.



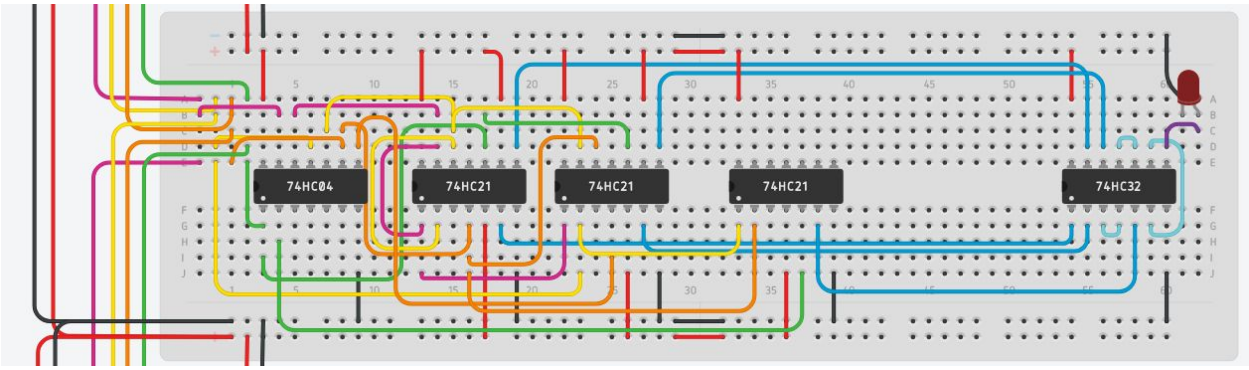
Em seguida adicionaremos todos os componentes necessários para a criação de nosso primeiro codificador e faremos as devidas ligações lógicas e elétricas, seguindo suas respectivas equações lógicas.

C1 = $\overline{A}\,\overline{B} + \overline{A}\,C\,D + A\,C\,\overline{D} + A\,B\,\overline{C}\,D$

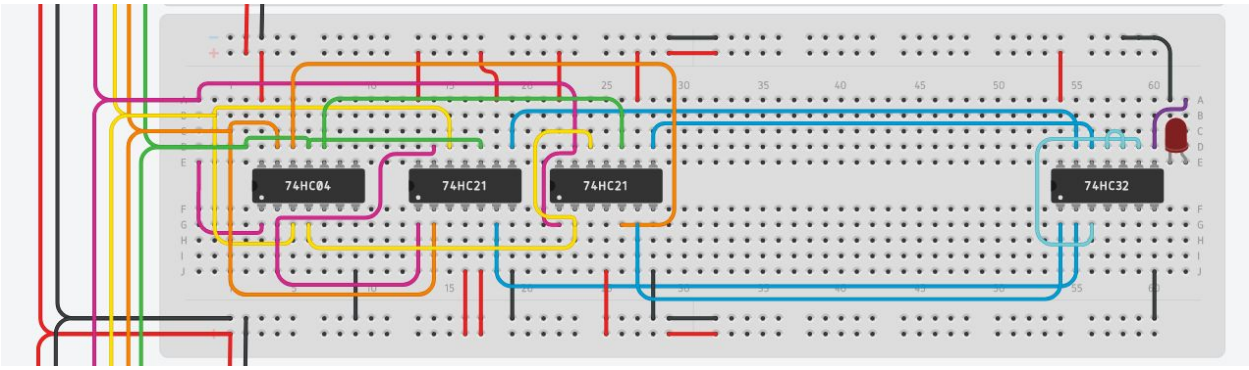



E assim por diante com os demais codificadores.

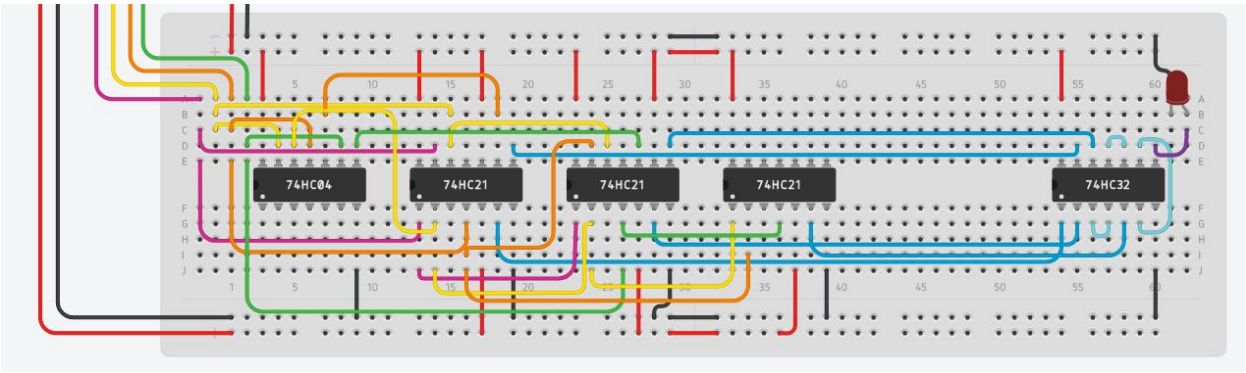
C2 = $\overline{A}\,\overline{B}\,D + \overline{A}\,\overline{B}\,C + \overline{B}\,C\,D + \overline{A}\,B\,\overline{C} + B\,C\,\overline{D}$



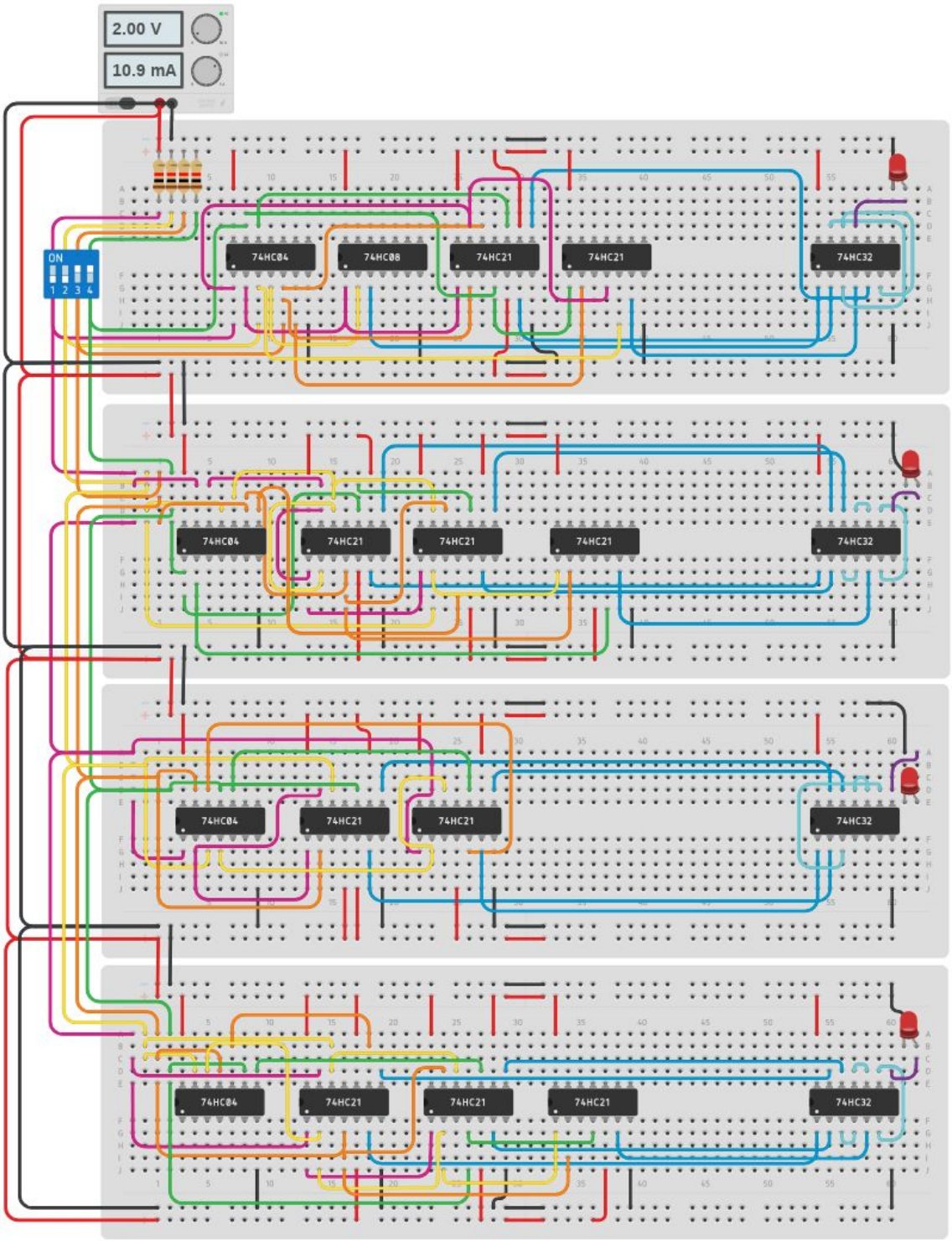
C3 = $\overline{A}\,C + \overline{A}\,B\,D + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{D}$




C4 = $\overline{B}\,C\,D + B\,C\,\overline{D} + A\,\overline{B}\,D + A\,\overline{B}\,C + A\,B\,\overline{C}$



Por fim, temos os circuito completo (Módulos C1, C2, C3 e C4) implementados na plataforma TinkerCad.




O circuito anterior pode ser testado no seguinte link (o link estará ativo até às 16h00m do dia 23/12/2019).

O comportamento de cada codificador reflete ao estudado anteriormente no programa Logisim.

Temos o BARCO o qual possui 4 entrada (cada uma com duas opções, nível lógico alto ou baixo), essas 4 entradas correspondem ao A,B,C e D que serão codificados, a partir das portas lógicas NOT (74HC04), AND (74HC08, 74HC21), OR (74HC32). E terão seus sinais exibidos a partir de um LED de coloração avermelhada, o qual acenderá se o nível lógico do sinal for alto ou ficará apagado caso o nível lógico do sinal for baixo.

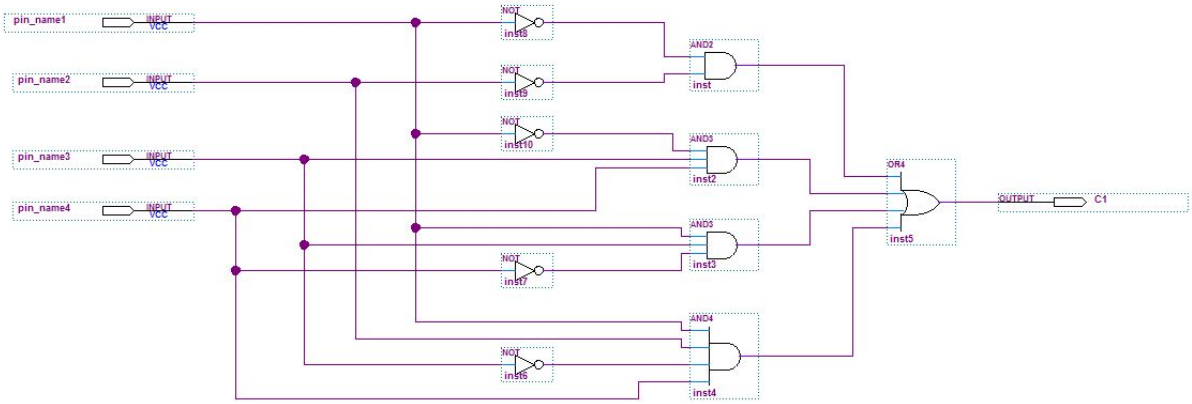
Podemos verificar que as configurações estão corretas ao analisarmos as entradas e saídas de cada codificador. Entramos 0011 e saem do codificador 1111 respectivamente, ou seja, todos os LEDs acesos.

- Quartus e Altera DE1

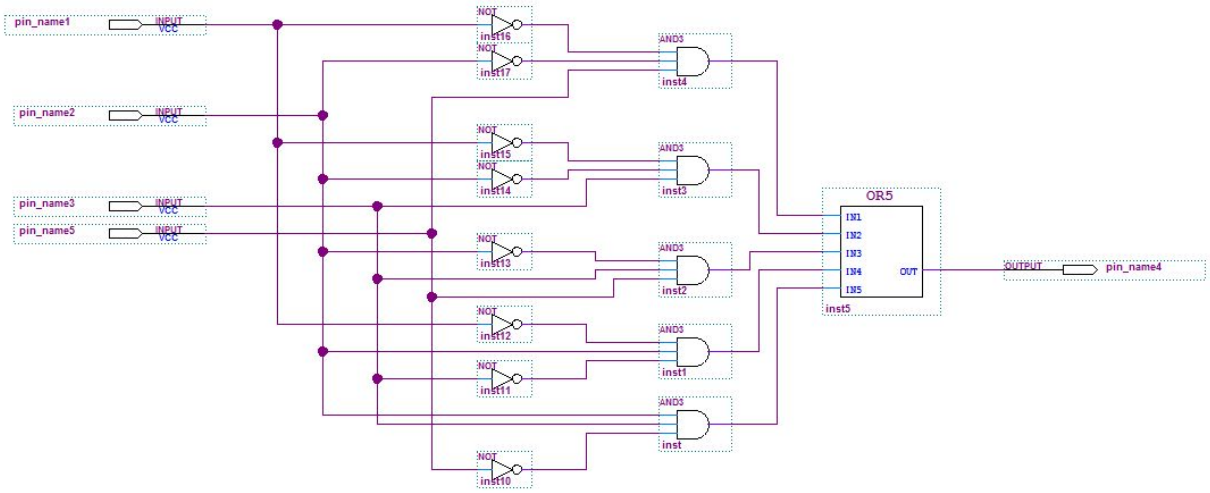
O Quartus é um ambiente de desenvolvimento integrado (IDE) no qual podemos construir nossa simulação do jogo de Batalha Naval, compilar e transferir diretamente para a placa FPGA, Cyclone 2 - EP2C2F484C7, onde serão feitos testes práticos.

No Quartus Web Edition iniciaremos um novo projeto, e a partir dele será possível criar quantos arquivos e codificadores quisermos, nesse projeto vamos precisar de apenas quatro arquivos (Diagrama de Blocos/Arquivo Esquemático), visto que, utilizaremos apenas 4 codificadores diferentes.

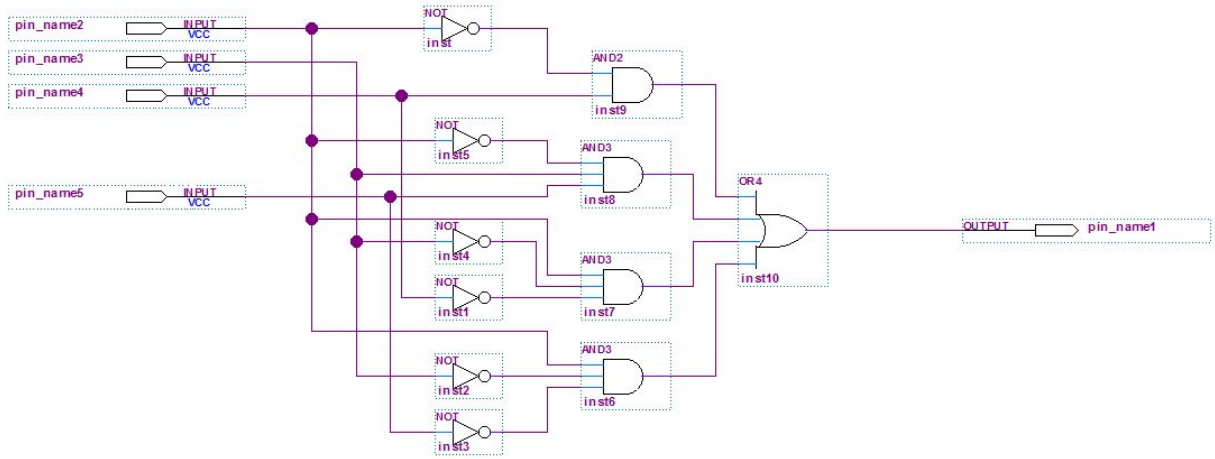
Primeiro codificador (C1)



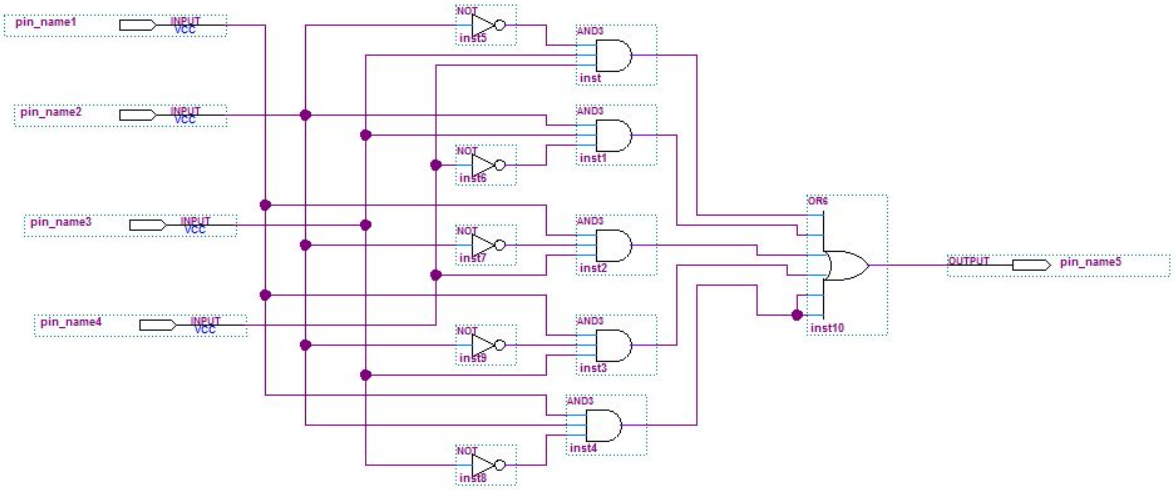

Segundo codificador (C2)



Terceiro codificador (C3)




Quarto codificador (C4)

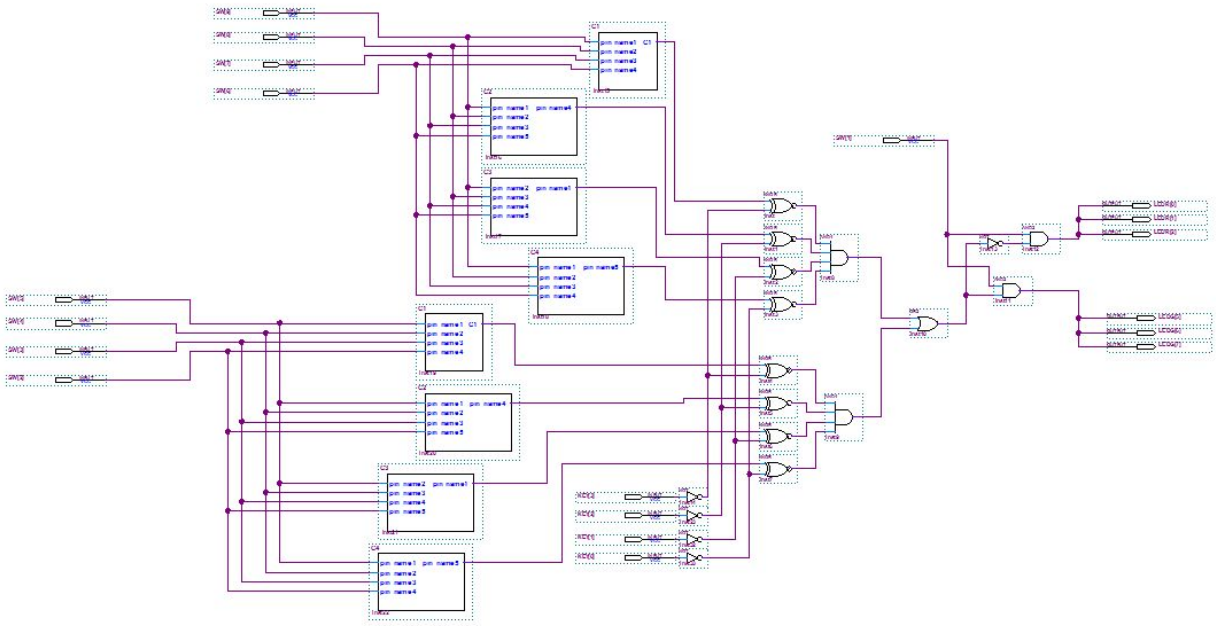


Por fim, com todos os codificadores prontos com seus devidos *inputs* e *outputs* associados, podemos criar a *Main* (arquivo no qual todos os codificadores, entradas e saídas convergem).

Nele temos todos as entradas dos barcos, entradas {SW[9], SW[8], SW[7], SW[6]} para o primeiro barco e as entradas {SW[5], SW[4], SW[3], SW[2]} para o segundo barco. As entradas do usuário, por sua vez, são formadas pelas 4 chaves {KEY[3], KEY[2], KEY[1], KEY[0]}. Conforme o DE1 Pin Assignments.

Podemos também, importar nossos 4 codificadores, cada um como sendo um bloco de símbolos individual, isso nos permite utilizá-los no circuito como se fossem uma porta lógica (mas com um comportamento customizado). E neste projeto utilizaremos 4 codificadores para cada barco (visto a matriz 4x4).

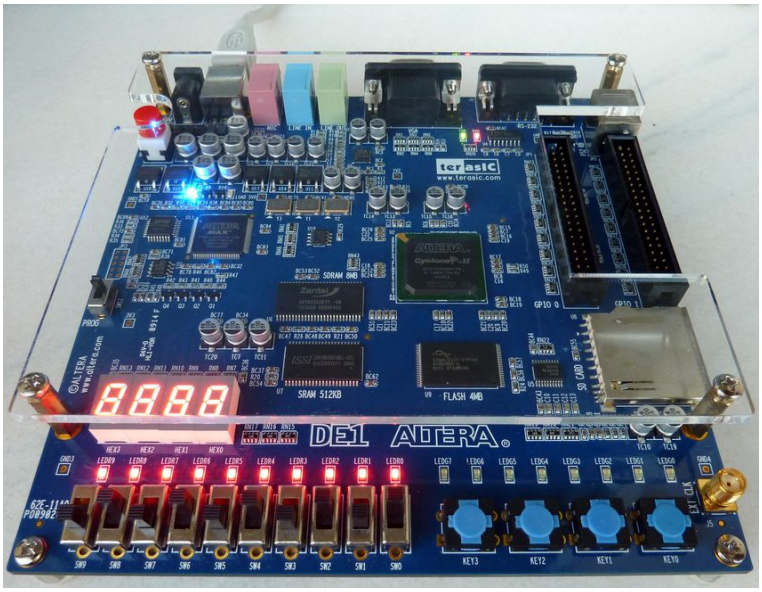
Após os codificadores serem inseridos no circuito e terem suas respectivas entradas e saídas ligadas, podemos comparar suas saídas com as entradas do usuário através de portas XNOR, que por sua vez tem seu sinal enviado a uma OR e por consequência até uma AND, a qual irá verificar o botão do **Disparo**, que acionará LEDs verdes {LEDG[5], LEDG[6], LEDG[7]} caso o alvo seja acertado ou vermelhos {LEDR[0], LEDR[1], LEDR[2]} caso a posição do usuário não seja a mesma posição da codificada dos barcos 1 ou 2.



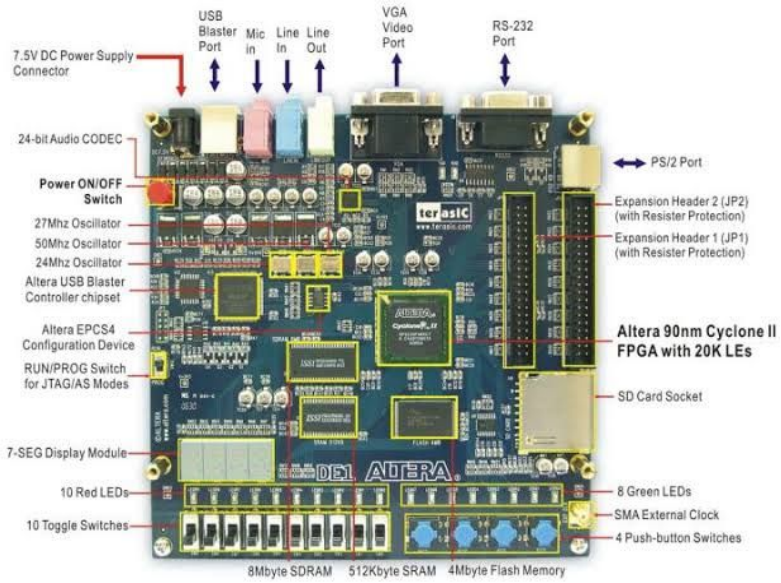
Se analisar, verá que as entradas do usuário são as únicas entradas que estão negadas, isso ocorre devido as entradas de nome “KEY[ ]” serem naturalmente de pull-down, ou seja, são ativadas em nível lógico baixo, mas para mantermos o mesmo padrão de entradas para todas as portas, decidimos negá-las a fim de as transformar em pull-up, ativando-as em nível lógico alto.

Tudo pronto, podemos compilar e passar nosso projeto para a placa, no entanto, primeiro temos que colocar nossa Main no topo da hierarquia ou “set as *top level”,* feito isso agora sim podemos compilar e passar o nosso circuito ou “programação” para a placa programável.

Eis uma imagem da placa com o nosso circuito rodando.




E uma imagem ilustrativa de seus componentes.




12 de dezembro de 2019

- Conclusão

Esse trabalho como um todo, para mim foi extremamente vantajoso. Pude não só aprender conteúdo novos, como também reforçar, praticar e me divertir (o que pode ser massante e/ou entediante para alguns, me trouxe certa diversão). Ficar conectando fios, portas lógicas e breadboards me parecia muito com um puzzle complexo.

Criando equações lógicas, mapas de Karnaugh, tabelas-verdades como meios de resolver desafios me enriqueceram tanto para a disciplina quanto para o dia a dia (em disciplina alguma jamais pensei que poderia aplicar tanto o conhecimento e o aprendizado como foi nessa de Circuitos Digitais), aprender como muitos de nossos eletrônicos hoje em dia funcionam e se comportam a nível lógico foi deveras interessante.

Quanto aos softwares utilizados todos me surpreenderam de diferentes formas. Logisim através de seu incrível poder e simplicidade, TinkerCad através de seu simulador de circuitos e Quartus por meio da sincronização do circuito para com a placa.
No entanto, nem tudo é uma maravilha, o vilão da história TinkerCad, a primeira vista extremamente fluído, responsivo e agradável; por outra problemático, travado e com um tempo de resposta absurdamente irritante, travadas bruscas em suas simulações (tanto com o projeto rodando quanto ainda por adicionar fios e portas lógicas) trazia dores de cabeça e irritação a qualquer um que tentasse utilizá-lo após a adição de 2 ou mais placas de ensaio. Porém com paciência quase tudo se resolve e após o circuito ser concluído no TinkerCad foi só felicidade, por que eu não via o trabalho como trabalho, mas sim como “estou fazendo um jogo e jogos são divertidos”.

Pude concluir o projeto em relativamente pouco tempo e assim auxiliar meus colegas que tinham dificuldades em alguma parte.