| Nome: | RA: | Email: |
| --- | --- | --- |
| Giovanna Vendramini | 173304 | giovannavendra@hotmail.com |
| Thiago Dong Chen | 187560 | thiagodchen@gmail.com |


# Relatório - Jogo 2048
# MC613 A - Grupo 17

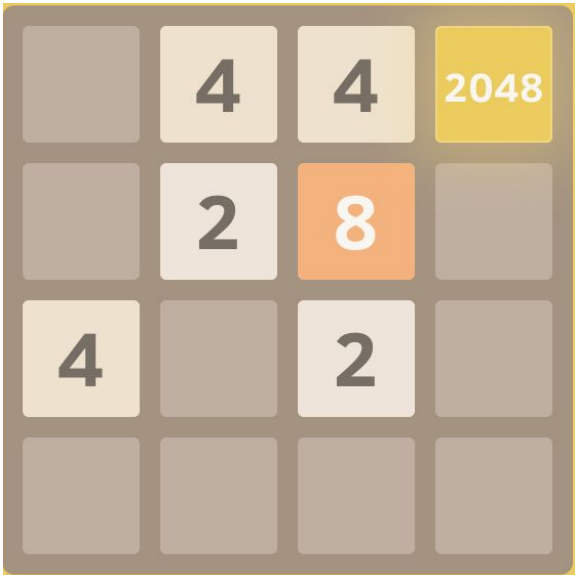


**Fig.1:** Jogo 2048.

Data de entrega: 18/06

# I. Introdução

O jogo 2048 é um jogo de raciocínio que foi criado em março de 2014 por Gabriele Cirulli, um desenvolvedor italiano. O jogo consiste em um tabuleiro 4x4 e peças cujos valores são potências de 2, e seu intuito é combinar valores iguais para que se obtenha peças com o dobro do valor, até que se alcance o 2048.

# II. Resumo

O jogo se inicia com o tabuleiro vazio e somente duas peças de valor 2 em posições aleatórias. A cada rodada, o jogador pode escolher uma das quatro direções, de modo que todo o conjunto de peças irá se movimentar de acordo com a direção escolhida enquanto não colidir com uma borda do tabuleiro ou com outra peça. Caso duas peças de mesmo valor colidam, elas se fundem dando origem a uma peça com o dobro do valor no lugar, não sendo possível que haja uma nova fusão da peça resultante na mesma rodada. Ao final de cada rodada, surge uma nova peça de valor 2 em uma posição aleatória que estava vazia no tabuleiro.

O fim do jogo ocorre quando o jogador consegue juntar duas peças de 1024 e dar origem a uma peça de 2048, ou quando o tabuleiro está cheio, não sendo possível fazer mais nenhum movimentação ou fusão de peças.

# III. Introdução Teórica

## A. Máquina de Estados Finitos

Uma máquina de estados finitos consiste em uma máquina com uma determinada quantidade de estados, só podendo assumir um estado por vez. A transição de estados ocorre em decorrência a uma entrada (sinal) e é sincronizada com o clock.

As máquinas de estados finitos se diferem em máquinas de Moore e Mealy, ambas são dependentes do estado atual, entretanto, a de Mealy depende de um sinal de entrada também.

## B. Process

O comando Process é utilizado em VHDL para que se execute de forma sequencial uma lista de ações. Dentro de um Process, a atribuição de variáveis ocorre de maneira imediata, enquanto que atribuições a sinais são feitas apenas ao finalizar o bloco de ações. Ainda no comando, após a palavra “process” há a lista de sensibilidade do processo, isso é, um conjunto de sinais e eventos para os quais o process é retomado.

## C. Vgacon [1]

A componente vgacon, disponibilizada no Moodle da disciplina, realiza a sincronização horizontal e vertical da tela e associa à placa DE1. É uma componente que facilita a escrita na tela de um monitor com entrada analógica por ser responsável pela atribuição aos pinos de saída, sendo 4 bits para as cores azul, verde e vermelho, além de dois pinos para sincronização. Uma descrição mais detalhada do funcionamento da entidade é fornecido na seção V e na referência [1].

## IV. Diagrama de Blocos

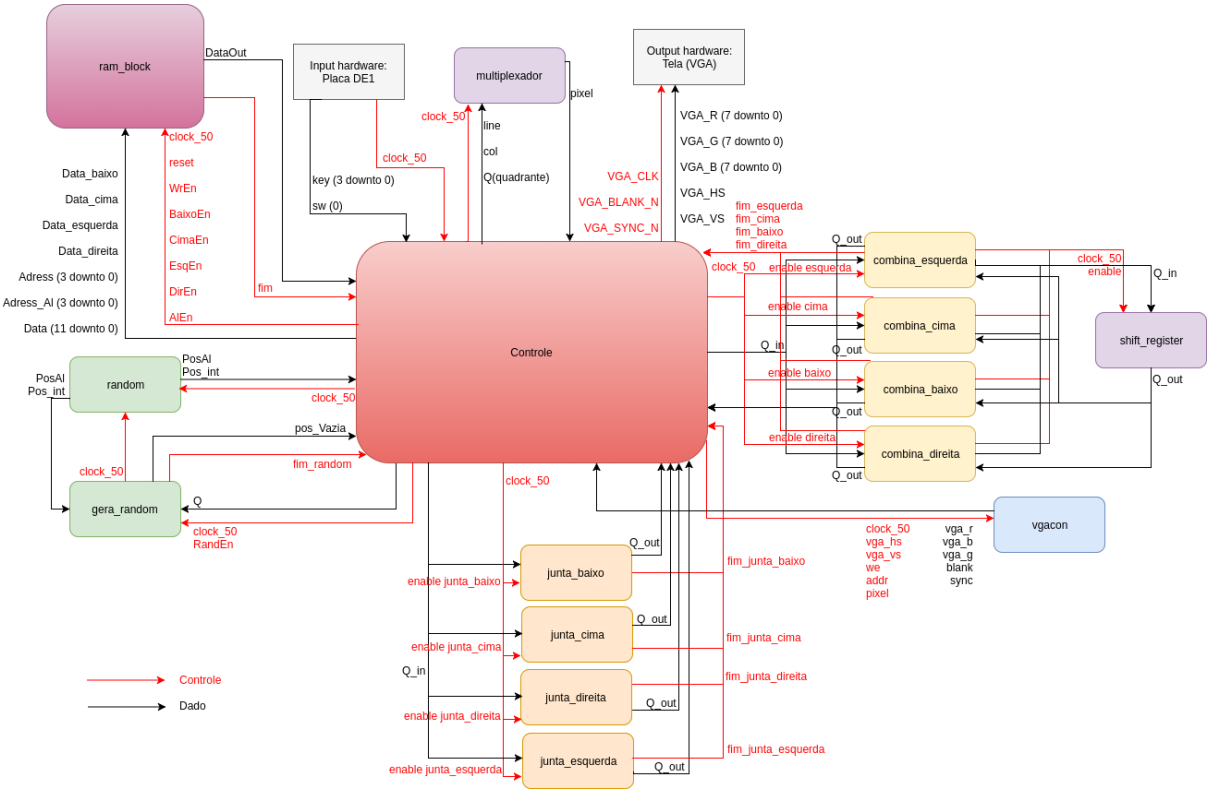


**Fig.2:** Diagrama de Blocos entidade top-level.

## V. Descrição do sistema

**Input Hardware (Placa DE1):** Clique do jogador nas keys da placa para movimentar as peças do tabuleiro nas direções (← ↑ ↓ →) ou dar reset no jogo utilizando SW(0). A implementação da movimentação das peças poderia ter sido feita utilizando as setas do teclado, o que tornaria mais intuitiva a jogada, no entanto, optou-se por focar na implementação da lógica em detrimento dessa possível melhoria.

**Package Common:** Declara o tipo Q_Array como um array de 0 até 15 de vetores de 12 bits. Ou seja, cria uma matriz com os valores das posições da memória de 0 a 15. Com isso, para escrevermos ou lermos o valor de uma das posições da memória (referentes aos 16 quadrados da tela) acessamos diretamente utilizando o índice. A forma escolhida para representar os valores foi através de vetores de 12 bits, para que se obtivessem diretamente os valores, representando eles já através dos bits. Essa implementação pode facilmente ser utilizada combinada com o shift já implementado para o laboratório anterior, o que tornou simples realizar o acréscimo das peças combinadas. Uma outra alternativa, porém, teria sido representar apenas com o valor do expoente da potência de dois equivalente ao número e somando um cada vez que houvesse combinação de peças, tendo como vantagem menor consumo de memória.

**Controle:** Entidade top-level que comunica com os Inputs (Key e SW) e outputs (VGA e HEX). O controle instancia a maior parte dos componentes (random, ram_block, multiplexador, junta_cima, junta_baixo, junta_esquerda, junta_direita, combina_cima,

combina_baixo, combina_esquerda, combina_direita, gera_random). Ela também inicializa o tabuleiro. Para o controle e sincronização das componentes, essa entidade possui uma máquina de estados de Mealy que controla a ativação de cada componente. Os estados que ela possui são:

- **inicio:** reseta o tabuleiro, timers, e contadores de linha e colunas.
- **inicializa_tab:** reseta o tabuleiro e permite a escrita de dois ‘2’ em posições aleatórias.
- **constroi_quadro:** ativa a escrita na memória do vga e espera um comando do jogador.
- **est_junta_cima, est_combina_cima, est_junta_baixo, est_combina_baixo, est_junta_esquerda, est_combina_esquerda, est_junta_direita, est_combina_direita:** ativam os componentes de junção e combinação, respectivamente, das peças.
- **est_gera_random**: ativa a componente de gerar random até achar uma posição vazia.
- **est_escreve_rand:** escreve na memória, o número ‘2’ na posição sorteada. Sendo seu próximo estado novamente o “constroi_quadro”.

Algo interessante de notar é que, durante cada jogada, as peças são juntadas, depois combinadas e juntadas novamente. Essa ordem é necessária pois, após ser feita a combinação, podem surgir espaços vazios entre as peças que foram combinadas (duas peças que originaram uma de maior valor, sendo que uma das peças deixou de existir) e é necessário garantir que essas peças serão novamente aglomeradas em uma das laterais.

**ram_block:** Memória Ram em que cada posição do quadrado é representado pelo seu endereço (0 a 15) e o dado representa o valor do quadrado. Esse recebe o clock para a sincronização da escrita, reset e os enables com seus respectivos dados. Essa entidade irá retornar todos os valores nas posições da memória num Q_Array. Um exemplo de armazenamento e leitura do dado pode ser visto da figura 4.

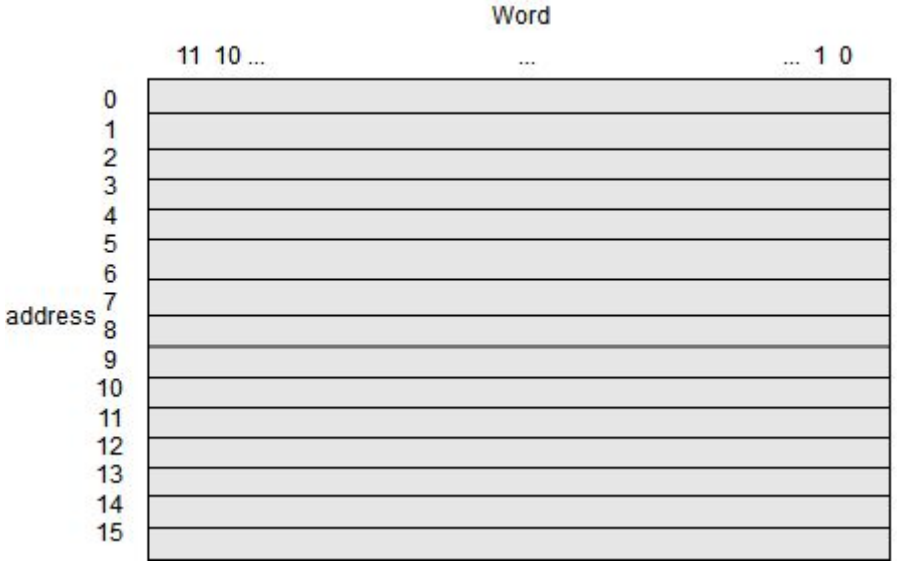


**Fig.3:** Representação do mapa de memória.

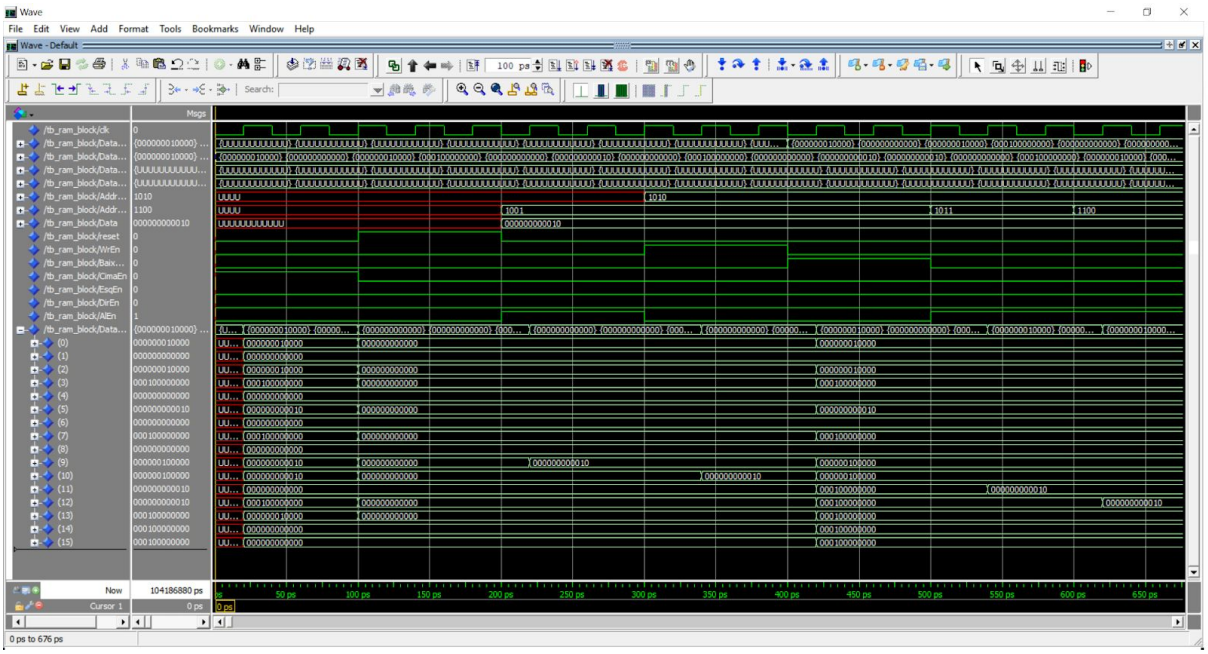

**Fig.4:** Simulação da forma de onda da entidade ram_block.vhd.

**random:** Recebe o clock, gera e retorna um número aleatório de 0 a 15, referente às posições do tabuleiro, na forma de binário (PosAl) e convertido para inteiro (Pos_int). A implementação dessa entidade foi feita de forma que há um contador nela que incrementa a cada subida do clock. O seu valor está limitado entre 0 a 15 por esse contador possuir apenas 4 bits e, se houver um novo incremento quando seu valor é 15, ocorre overflow e o seu valor retorna para 0.

**gera_random:** Recebe o vetor Q do tipo Q_Array que fornece todos os valores de todas as posições da memória que podem ser acessados através do índice. Para o controle dessa entidade, a entidade Controle envia para gera_random o clock_50 e o RandEn, sincronizando as atribuições com o clock e definindo quando o número aleatório poderá ser criado. Além disso, dentro dessa entidade é instanciada a entidade random, enviando para ela o sinal de clock_50 e recebendo os valores aleatórios criados. Posteriormente é feita a verificação se a posição aleatória retornada está vazia e, se estiver, tal posição é enviada para a entidade controle como Pos_Vazia e fim_random é atualizado para 1. Dessa forma, a geração de números aleatórios é interrompida e a máquina de estados do controle pode passar para o próximo estado. Essa entidade pode ser facilmente testada através de forma de onda que pode ser vista na figura 5.

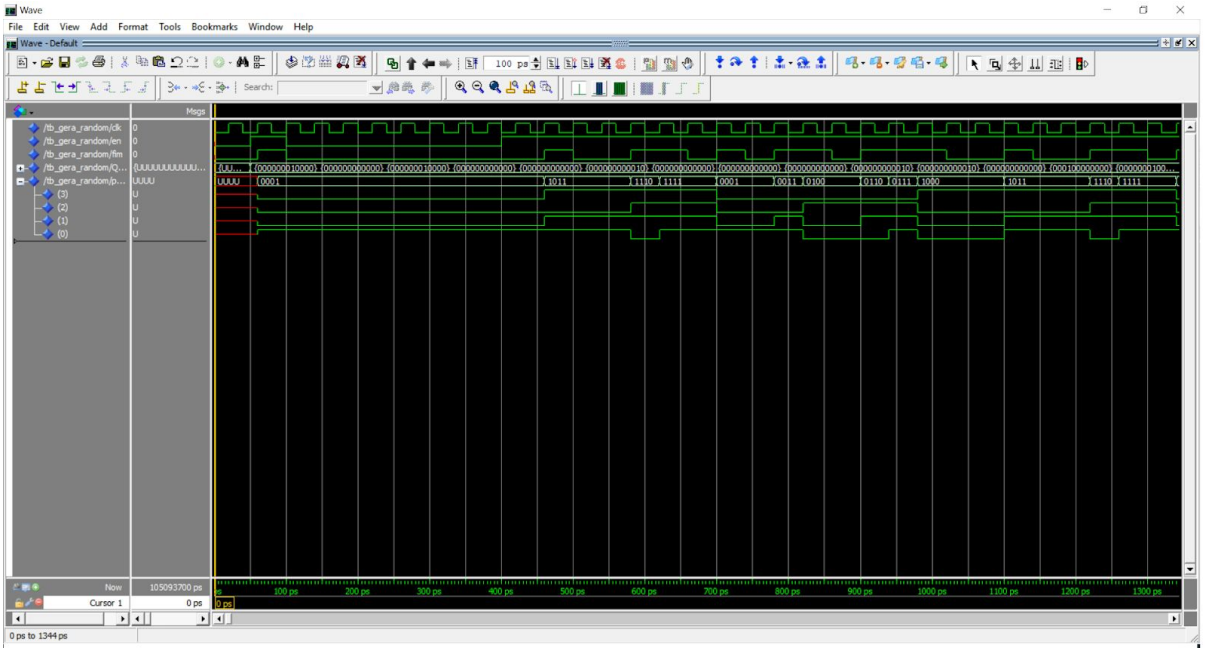


**Fig. 5:** Simulação da forma de onda da entidade gera_random.vhd.

**junta_baixo:** A entidade junta_baixo recebe o clock_50 para a sincronização e o enable para habilitar ou desabilitar as suas alterações. Seu enable estará ativo quando a máquina de estados da entidade Controle estiver no estado referente a junta_baixo. Recebe também Q_in do tipo Q_Array que fornecerá à entidade os valores em cada posição do tabuleiro, bem como as posições vazias. Essa entidade irá fazer a verificação iniciando das posições mais inferiores até as superiores a fim de, se encontrar uma posição vazia, permutar com a próxima posição ocupada se houver. Para isso, são feitas comparações dentro de um process, realizando as alterações do mapa de memória primeiro em uma matriz de variáveis, para que só depois de realizadas todas as modificações, passe esses novos valores para o sinal de saída. Com isso, se garante alterações imediatas do valor para que a próxima iteração já detecte o novo valor e realize as modificações se baseando nele.

Ao fim da entidade se garante que todos os valores diferentes de 0 estão aglomerados na parte inferior do tabuleiro. Por fim, a entidade irá retornar o estado das novas posições do tabuleiro e seus novos valores, no Q_Array através do signal Q_out e um signal fim_junta_baixo indicando que essas alterações já foram realizadas e habilitando a troca de estado na máquina de estados do Controle.

Exemplo de uma instância de junta_baixo:
Antes:

| | | | |
|---|---|---|---|
| 0 | 4 | 16 | 0 |
| 2 | 0 | 8 | 0 |
| 0 | 256 | 0 | 16 |
| 2 | 0 | 0 | 16 |

Depois:

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 2 | 4 | 16 | 16 |
| 2 | 256 | 8 | 16 |

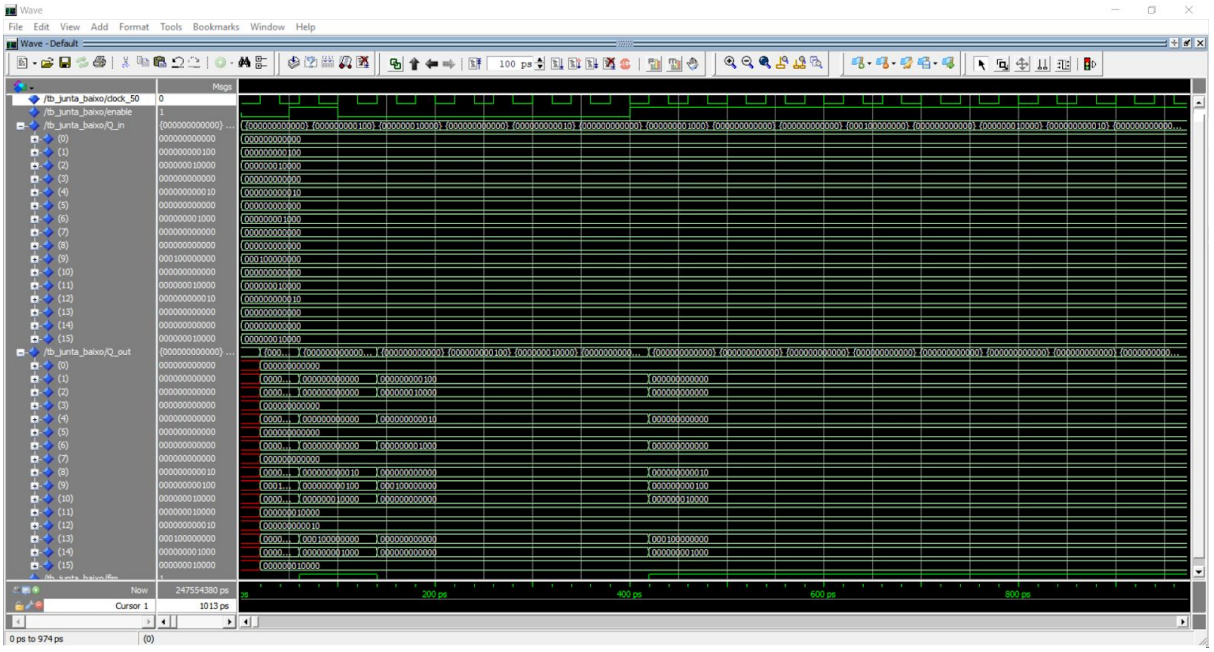

**Fig.6:** Simulação da forma de onda da entidade junta_baixo.vhd.

**combina_baixo:** A entidade baixo recebe o clock_50 para a sincronização e o enable para habilitar ou desabilitar as suas alterações. Seu enable estará ativo quando a máquina de estados da entidade Controle estiver no estado referente à combina_baixo. Recebe também Q_in do tipo Q_Array que fornecerá à entidade os valores em cada posição do tabuleiro. Essa entidade irá fazer a verificação de se posições adjacentes verticalmente que possuem valores iguais e são diferentes de zero para realizar a combinação dessas, via shift_register, sendo as verificações e atribuições realizadas dentro de um process de forma similar a entidade “junta_baixo”, garantindo a sequência das ações e que a mudança no valor das variáveis ocorre imediatamente, havendo a atribuição dele ao sinal ao fim do process. É utilizado também um vetor de variable comb para determinar se alguma das peças já foi combinada anteriormente, considerando sempre a verificação de baixo para cima de 4 vetores de 4 posições cada. E, como as posições são verificadas sempre 2 a 2, é necessário que se verifique para não combinar a mesma peça duas vezes. Por exemplo, considerando o vetor ao lado a verificação ocorre na ordem Q(0) e Q(1), Q(1) e Q(2), e assim sucessivamente, verificando sempre se as duas anteriores foram combinadas para que não ocorra combinação duas vezes de peças intermediárias. Nesta entidade é instanciada a entidade shift_register para todos as posições de memória e ela retorna o dobro dos valores dessas posições de memória. Então, se duas posições vizinhas verticalmente possuírem o mesmo valor (diferente de 0) e nenhuma das peças já tiver sido combinada nessa rodada,

| |
|---|
| Q(3) |
| Q(2) |
| Q(1) |
| Q(0) |

uma das peças tem seu valor alterado para o dobro e a outra se torna 0. Os novos valores do mapa de memória  e um signal fim indicando o fim das alterações são retornados.

Exemplo de uma instância de combina_baixo:
Antes:

| | | | |
|---|---|---|---|
| 0 | 0 | 16 | 32 |
| 0 | 2 | 16 | 32 |
| 0 | 2 | 0 | 256 |
| 0 | 0 | 0 | 256 |

Depois:

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 32 | 64 |
| 0 | 4 | 0 | 0 |
| 0 | 0 | 0 | 512 |

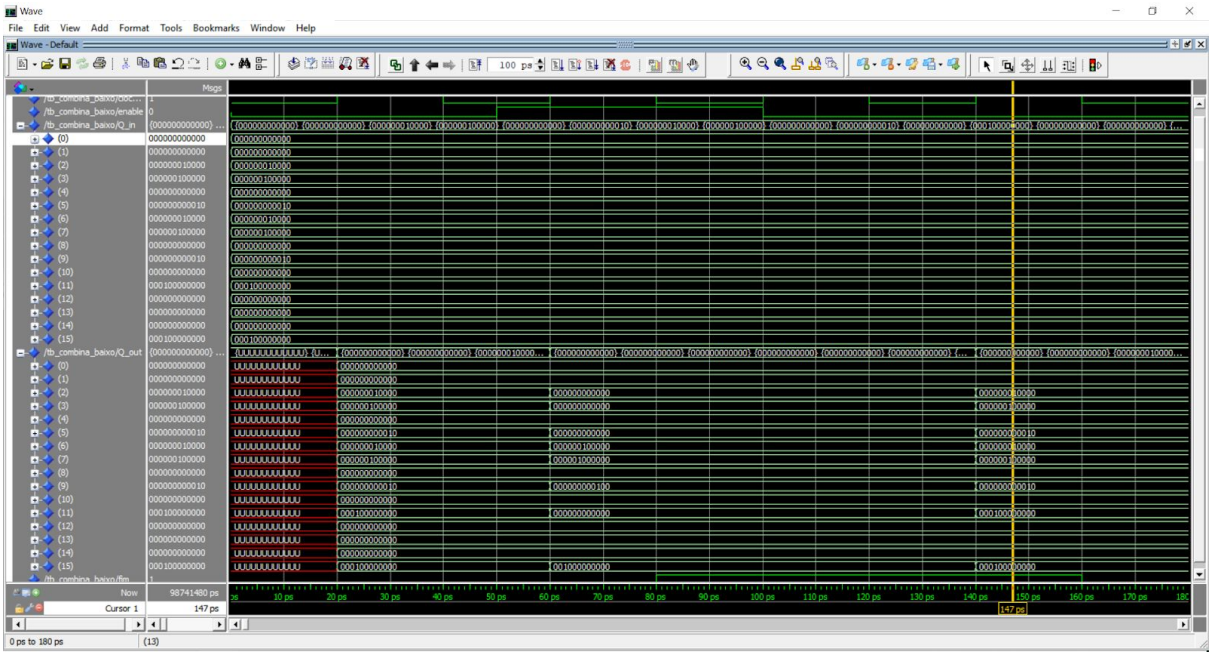

**Fig.7:** Simulação da forma de onda da entidade combina_baixo.vhd.

**junta_cima:** É uma entidade equivalente à junta_baixo, modificando apenas por sua verificação por espaços vazios se iniciar de cima para baixo, pois visa aglomerar as peças na região superior do tabuleiro.

**combina_cima:** É uma entidade equivalente à baixo, modificando apenas por sua verificação por valores iguais ser feita de cima para baixo. Irá também combinar as peças de valores iguais utilizando a mesma lógica.

**junta_direita:** É uma entidade equivalente à junta_baixo, modificando apenas por sua verificação por espaços vazios se iniciar da direita para esquerda (de maneira horizontal), pois visa aglomerar as peças no lado direito do tabuleiro.

**combina_direita:** É uma entidade equivalente à baixo, modificando apenas por sua verificação por valores iguais ser feita da direita para esquerda, de maneira horizontal sempre. Irá também combinar as peças de valores iguais utilizando a mesma lógica.

**junta_esquerda:** É uma entidade equivalente à junta_baixo, modificando apenas por sua verificação por espaços vazios se iniciar da esquerda para direita (de maneira horizontal), pois visa aglomerar as peças no lado esquerdo do tabuleiro.

**combina_esquerda:** É uma entidade equivalente à baixo, modificando apenas por sua verificação por valores iguais ser feita da esquerda para direita, de maneira horizontal sempre. Irá também combinar as peças de valores iguais utilizando a mesma lógica.

**shift_register:** A entidade shift_register recebe o valor de uma posição, o clock_50 e o enable para sincronizar e habilitar o shift left do valor recebido. Ela é usada para retornar o dobro do valor recebido, realizando os cálculos durante o jogo para as combinações das peças. Essa entidade é instanciada dentro das entidades Esquerda, Cima, Direita e Baixo, pois nelas que são verificadas possíveis combinações e feitas as devidas alterações para promover a atualização dos valores do tabuleiro.

**multiplexador:** Essa entidade escreve na memória do vgacon, na qual cada endereço é a posição do pixel no monitor e o valor da cor desejada. Essa entidade recebe o estado atual do tabuleiro (valor da memória) Q_in, com a posição do pixel que será escrito na tela (coluna e linha). Dado essas entradas, associa a posição do pixel e devolve a cor do pixel da posição de entrada. É fundamental para a escrita dos números e quadrados na tela.

**vgacon:** Sincroniza o monitor com a placa, controlando o VGA, com o intuito de imprimir pixels na tela. Destaca-se que essa componente possui uma memória que cada posição representa a cor do pixel.

**Output hardware (Tela VGA):** Estado atual do tabuleiro que será enviado via VGA para impressão na tela.

## VI. Conclusão

O jogo apresentou o funcionamento que gostaríamos, permitindo uma boa jogabilidade e apresentando as principais funcionalidades do jogo. O jogo iniciava com as duas peças em posições aleatórias e as peças eram adicionadas corretamente ao fim de cada jogada. A movimentação das peças também não apresentou problemas, sendo possível aglomera-las em qualquer uma das quatro laterais. A combinação das peças também ocorria quando era feita uma jogada que possibilitava essa junção, no entanto, apesar da tentativa de não realizar combinações duplicadas através da

variável comb nas entidades e da utilização dos enables nos estados e nas entidades, elas aconteciam. Isso acontecia, provavelmente, por algum problema de sincronização, que acabava permitindo a execução mais de uma vez das entidades de combinação das peças (cima, baixo, direita e esquerda). Um possível solução talvez fosse a criação de uma nova máquina de estados para as jogadas, porém, isso implicaria uma reorganização geral do projeto que acabou impossibilitando sua implementação.

Além disso, pode-se observar que o jogo segue bem os estados da máquina, aguardando sempre a jogada e completando os estados referentes à jogada escolhida até o fim, retornando ao estado de constroi_quadro após a inserção de uma nova peça. Ao fim do jogo, o tabuleiro se torna cheio e não é mais possível realizar movimentações, sendo necessário que o jogador resete o jogo através do SW(0). A funcionalidade de resetar o jogo também funciona adequadamente, reiniciando o jogo com um novo e vazio tabuleiro, com apenas duas peças em posições aleatórias.

## VII. Comentários e Possíveis Correções

No diagrama de blocos inicial o grupo havia se proposto a realizar o aprimoramento de realizar a soma de pontos e apresentá-la no display de 7 segmentos. A entidade responsável para realizar essa conversão do valor foi criada e a lógica foi projetada. No entanto, não foi possível realizá-la com sucesso pois a soma dos pontos seria feita na mesma entidade que faz as comparações para combinar as peças e estava sendo executada várias vezes como foi discutido anteriormente. Por esse motivo, o valor da pontuação também se alterava diversas vezes numa mesma jogada e infinitos pontos eram atribuídos.

Além disso, não foi tratado também o caso em que o jogador tenta realizar uma jogada inválida, isso é, movimentar as peças para uma direção que não há combinações ou junções disponíveis. Para essa correção, seria necessário que soubéssemos de antemão se a direção requisitada pelo jogador provocaria alguma mudança no tabuleiro para assim reconhecermos ou não essa jogada. Uma forma de implementá-la talvez fosse acrescentar sinais de saída das entidades que juntam e combinam para informar se alguma delas provocou mudança no tabuleiro para que, apenas se a resposta for afirmativa, inserir uma nova peça no tabuleiro indicando o fim da jogada. Pois dessa forma, não seria acrescentada uma nova peça no tabuleiro se não fosse reconhecida nenhuma movimentação válida e, sendo assim, o jogador saberia que sua jogada não foi realizada.

## VIII. Referências

**[1]** https://www.ic.unicamp.br/~cortes/mc613/tutoriais_curso/vga/vga.pdf

### Links dos códigos fontes

**A. Relatório Final**
https://docs.google.com/document/d/1-UTJ8V-azYuxTROelPPa1veNPpJKyQiDZQOYoWpQAXI/edit?usp=sharing

**B. Diagrama de Blocos:**
https://drive.google.com/file/d/19zLil1CRjloai7sercTky64gM5lDM9On/view?usp=sharing