UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

ENGENHARIA DE COMPUTAÇÃO

PCS3115 - SISTEMAS DIGITAIS I

Grupo 8 - Atrasados

Carlos Maria Engler Pinto - 5104641

Gabriel Arthur Almeida de Carvalho - 12546360

João Pedro Bassetti Freitas - 13903411

Juliana Mitie Hosne Nakata - 13679544

Endereço do Github do projeto:

https://github.com/CarlosEngler/SnakeGame.git

Tema do Projeto - **Snake Game**

Tema do projeto:

O tema do projeto baseia-se no desenvolvimento, a partir dos conceitos abordados em aula, de um jogo análogo ao *Snake*, presente nos antigos celulares da *Nokia*. Com isso, pretende-se criar uma ferramenta simples de entretenimento, criando, a partir das ferramentas teóricas, algo lúdico voltado para a diversão.

Objetivos:

Pretende-se desenvolver um jogo funcional, que possa ser jogado por terceiros com a utilização de botões que afetam a movimentação da "cobrinha". O projeto consistirá numa interface de LED que representará o mapa do jogo e quatro botões (Up, Down, Left e Right) que controlarão a movimentação da cobrinha no mapa. O objetivo principal consiste na captura de "maçãs", que serão representadas por LEDs acesos de forma aleatória ao longo do mapa e renderão pontos ao jogador caso ele consiga alcançá-las. Caso a "cobrinha" atinja seu próprio corpo ou uma das extremidades do mapa, a partida é encerrada e a pontuação (número de maçãs adquiridas) é exibida na tela.

Problema:

O principal objetivo do projeto é desenvolver uma ferramenta de entretenimento simples para o dia a dia das pessoas, livrando-as do tédio e estresse rotineiro.

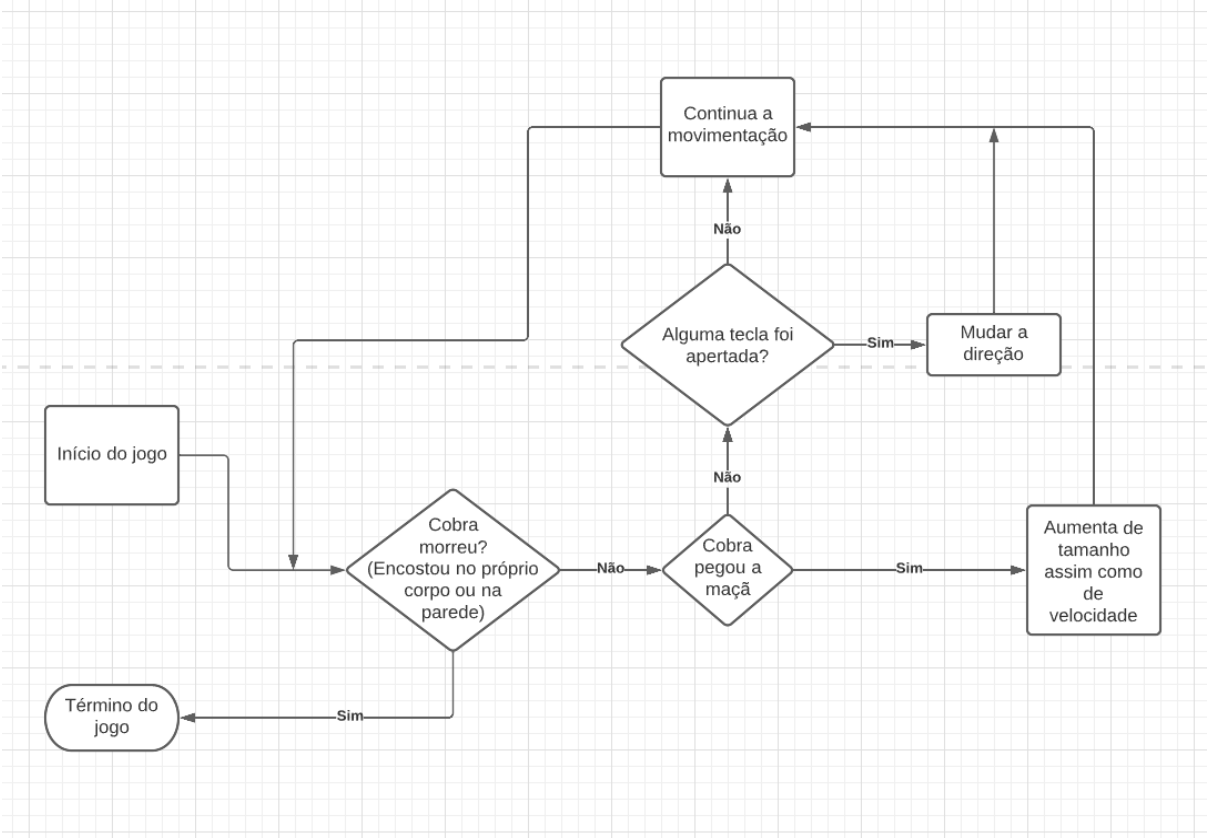
Tabela de inputs e outputs:

input	output
tecla cima	seta direção da cobra para cima
tecla baixo	seta direção da cobra para baixo
tecla direita	seta direção da cobra para direita
tecla esquerda	seta direção da cobra para esquerda

No nosso projeto a parte de inputs, ou seja, os sinais enviados externamente, serão apenas quatro teclas, como mencionado acima, que afetarão diretamente na

movimentação da cobrinha, e também poderão iniciar o jogo. Além dos outputs apresentados acima, vale ressaltar que dentro do programa principal do jogo, será enviado um sinal constante que afetará o estado da cobra. E por fim, ao morrer aparecerá mais um output, que será o placar.

Fluxograma:



A ideia principal do nosso projeto vai ser parecido com o fluxograma acima, que consiste basicamente da seguinte forma: ao apertar qualquer tecla o jogo responderá com um sinal que fará com que o jogo inicie, entrando em um loop infinito que só acaba quando a cobra morre, ou seja, ela bateu na parede ou nela mesma. A movimentação da cobra vai ser feita pelos inputs dos botões, ou seja, quando eles forem pressionados o sistema irá receber um sinal, e vai gerar um output que será a mudança de direção da cobra. E por fim, no loop principal do jogo há a presença da maçã que enviará um sinal constante para o jogo durante determinados intervalos de tempo, e a sua função será alterar o estado do tamanho e da velocidade da memória da cobra.

Funcionamento do projeto:

Para o seu funcionamento, o projeto dependerá das seguintes implementações:

-Detecção dos botões: Assim como descrito na tabela de inputs e outputs, haverão 4 botões (cima, baixo, esquerda, direita) que determinarão a respectiva mudança na direção do movimento da cobra em cada ciclo do jogo. De forma mais explicativa, isso significa que ao pressionar algum botão, esse determinará a casa em que a cabeça da cobra ocupará no respectivo ciclo (ou “frame”) do jogo.

Ainda nessa implementação, é necessário registrar o último estado da direção da movimentação da cobra, já que o intuito do jogo é que a cobra continue se movimentando mesmo sem que haja algum input (criando o risco da cobra bater em si mesma ou nas paredes). Para que isso seja feito, será necessário um circuito adicional que verifica os possíveis outputs dependo dos inputs realizados (por exemplo, para evitar que o jogo acabe quando é pressionado o botão esquerdo enquanto a cobra se move para a direita, como se a cobra tivesse batido em si mesma após um único ciclo do jogo). Além disso, será necessário implementar uma ordem de prioridade dos inputs, para determinar qual o único movimento que ocorrerá em um ciclo ao serem apertados múltiplos botões simultaneamente. Tudo isso é representado no diagrama como a parte de “verificação dos inputs”, cuja função é a descrita, de definir, seguindo a ordem de prioridade, o input de movimento que a cabeça realizará e também de armazenar o último input lido, para que haja o constante movimento da cobra.

-Movimentação do corpo da cobra: A movimentação do corpo da cobra é definida inicialmente pela posição e movimentação da sua cabeça. Cada parte da cobra, no próximo ciclo, ocupará a posição antiga da parte imediatamente à sua frente (com exceção à cabeça, cuja posição é determinada pelos inputs), o que cria o movimento icônico da cobra no jogo. Dessa maneira, em cada ciclo, a posição (x,y) de cada segmento da cobra será registrada na memória e no ciclo seguinte essa mesma posição será atribuída ao segmento sucessor (exemplo de uma cobra de 3 segmentos cuja cabeça é a primeira coordenada: (1,1)(1,2)(1,3)→(2,1)(1,1)(1,2) — nesse caso é como se a cobra tivesse feito um “L”).

Além disso, a cabeça vai ter sua movimentação a partir da soma e subtração dos valores x e y dependendo do input (por exemplo, ao apertar o botão direito, a variação da posição será (x+1,y). Ou seja, o registro da posição da cabeça estará conectado a um sistema de somadores (consequentemente, de subtratores), e a operação que vai ocorrer (x+ ou -, y+ ou -) é definida pelo último input registrado.

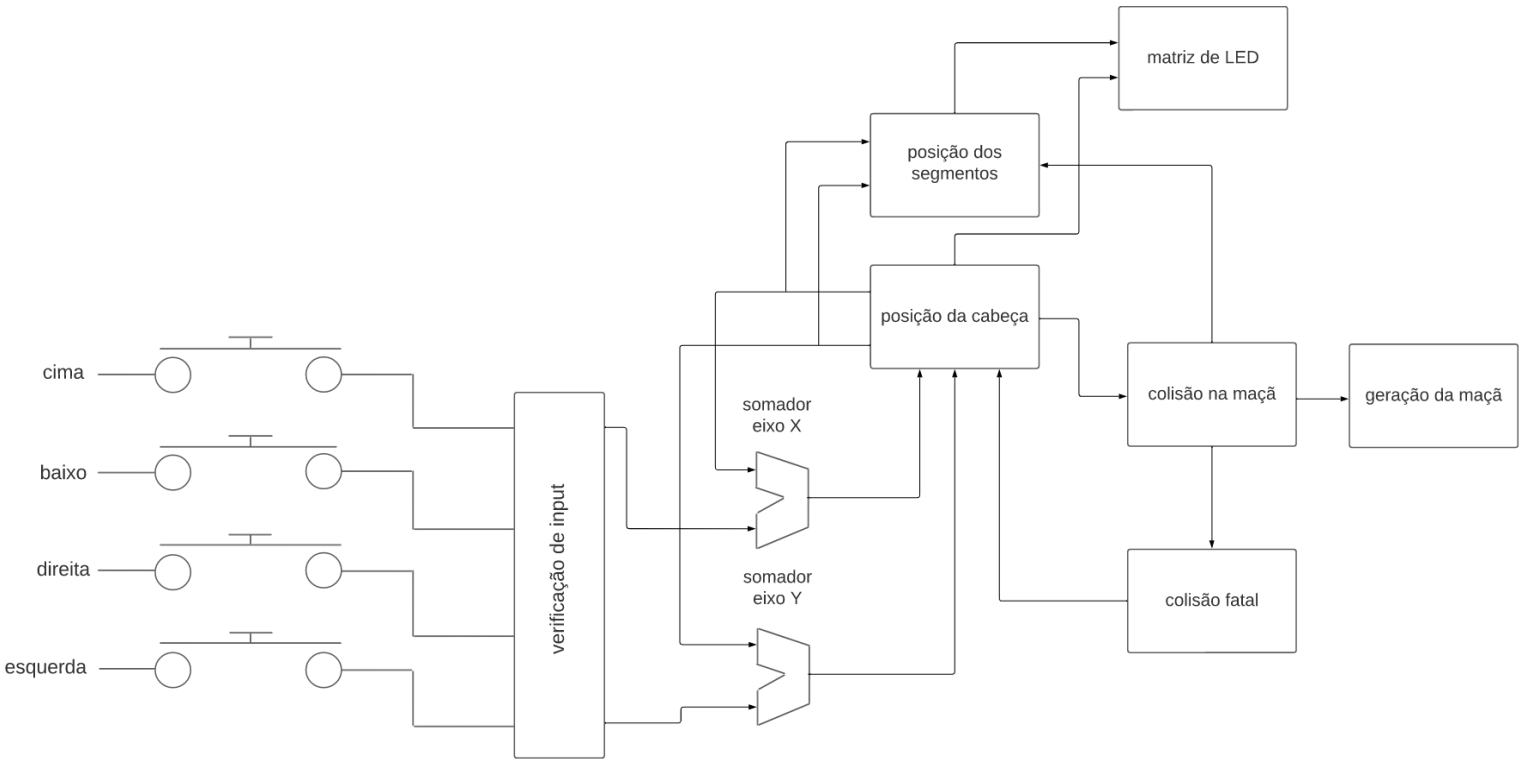
Geração da Maçã: Essa implementação é um sistema lógico que avalia, a partir do registro dos segmentos, as posições possíveis de se colocar a maçã (para evitar que ela surja dentro da cobra) e determinar, de uma maneira “aleatória”, a localização que ela de fato aparecerá no mapa.

Colisão com a maçã: Inicialmente será verificado se a posição atual da cabeça da cobra corresponde ou não com a da maçã gerada. Caso sim, é adicionado mais um segmento da cobra na memória e passa a ser representado no jogo a partir do próximo ciclo (além de ser gerada outra maçã). Caso não, verifica-se o próximo caso de colisão. O projeto do funcionamento dessa adição de um segmento será o seguinte: todos os possíveis segmentos já possuirão um espaço registrado no início do jogo, porém apenas os iniciais serão representados na matriz; conforme as maçãs são comidas, os respectivos segmentos sucessores passam a entrar no ciclo geral de movimentação do corpo.

Colisão Fatal: Nesse sistema de verificação há a avaliação das possíveis situações de término do jogo (caso a cobra colidiu com o próprio corpo ou a parede) e devolve nada para o registro da posição da cabeça (caso não tenha havido colisão), ou envia um sinal de término para esse mesmo registro (caso tenha havido colisão), o que impede a continuação normal do loop do jogo e o impede de atualizar o estado da matriz de led. A verificação dessas situações ocorreu por meio da comparação da posição da cabeça com todos os segmentos do corpo e com as paredes do mapa.

Matriz de LED: Por fim, a última implementação e que representa o último estágio de cada loop de funcionamento do projeto é o “print” de cada estado de jogo. A matriz de LED representará a cobra e seus segmentos como pontos acesos, assim como a maçã, e receberá as informações de quais LEDs acender a partir da memória dos segmentos e do registro da posição da cabeça. Assim como foi

mencionado no caso de colisão fatal, a matriz de LED vai printar cada loop de funcionamento do jogo enquanto a cobra não passar por uma das situações de término de jogo mencionadas, e dentro dessa circunstância a matriz de LED irá representar a finalização do jogo definida, retornando o funcionamento do projeto para o estado inicial (onde há a necessidade de apertar um botão para iniciar o jogo).



No diagrama de blocos as células “posição da cabeça” e “posição dos segmentos” representam os respectivos registros, e as setas que saem da posição da cabeça e vão até posição dos segmentos representam o processo de atualizar as coordenadas dos segmentos da cobra conforme os ciclos (assim como foi descrito em “Movimentação do corpo da cobra”).

Descrição em Verilog

Módulo InputsBotao

Esse módulo é responsável por interpretar as interações do usuário com os botões do jogo. Cada uma das direções que a cobrinha pode andar foi representada por um número em binário dada a seguinte correspondência:

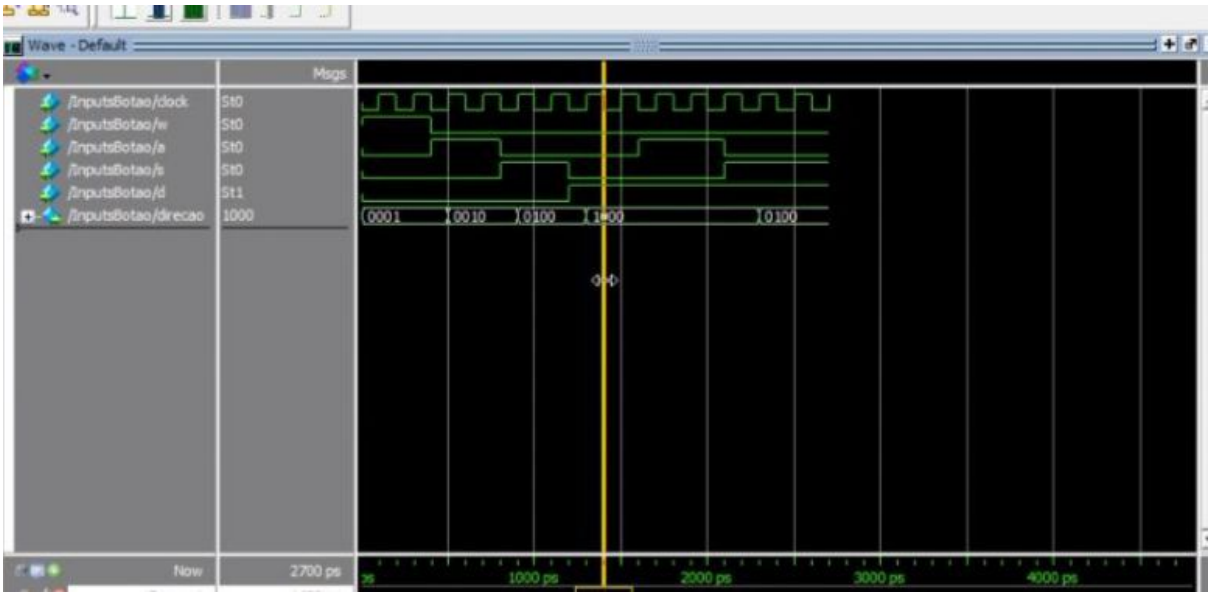
direção **cima**	**0001**
direção **esquerda**	**0010**
direção **baixo**	**0100**
direção **direita**	**1000**

Conforme uma das teclas é apertada, seu código em binário é salvo em um registrador de 4 bits, e esse valor pode ser atualizado a cada virada de clock. No código, as diferentes teclas são ilustradas pelos inputs ‘w’, ‘a’, ‘s’, ‘d’ e, caso uma delas esteja igual a 1, ou seja, esteja apertada, o registrador recebe o binário correspondente.

Além disso, dentro das condições, há uma restrição que garante que a cobra não baterá no seu próprio corpo imediatamente após a virada de clock, pois o registrador da direção só será alterado se a tecla apertada não corresponder à direção oposta da que estava guardada anteriormente (p. ex. esquerda e direita). Ainda, caso nenhuma tecla seja apertada durante o período de contabilização de novos inputs, a direção segue com o último valor atribuído.

Vale ressaltar que as condicionais são excludentes e estabelecem uma prioridade entre as teclas: no caso em que duas teclas são apertadas simultaneamente, apenas o valor da que aparece primeiro no código é contabilizada por conta da estrutura de else if’s feita. Assim, caso a tecla ‘a’ e a ‘d’ sejam apertadas ao mesmo tempo, o registrador contabiliza apenas 0010.

Abaixo, encontra-se um print da simulação de ondas feita no testbench do módulo, em que é possível observar a mudança de valores do registrador de acordo com os diferentes inputs a cada virada de clock, além da correção de entradas simultâneas de acordo com a prioridade dada.

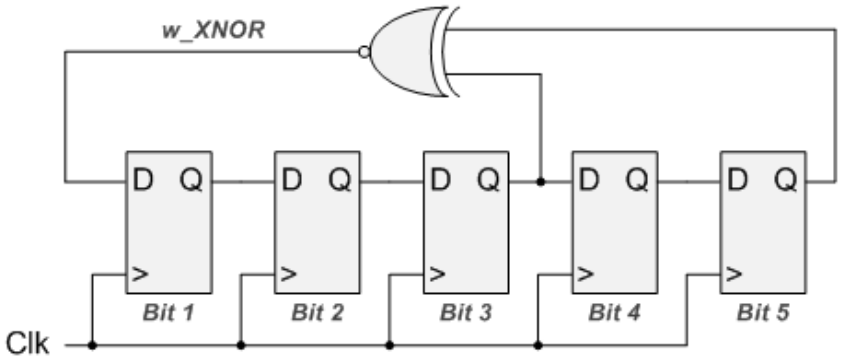


Módulo Geração da Maçã

Este módulo ficou responsável por estabelecer a posição da maçã no tabuleiro assim como por implementar a matriz de LED que foi usada no desenvolvimento do projeto.

A priori, ressalta - se que foi utilizado para geração da maçã no tabuleiro o mecanismo LFSR (Linear Feedback Shift Register) uma vez que são simples de sintetizar além de consumirem poucos recursos podendo ser executados a taxas consideradas elevadas de clock dentro de um FPGA (NAND_LAND, 2019) levando assim a uma maior economia de custo (como o projeto solicitava).

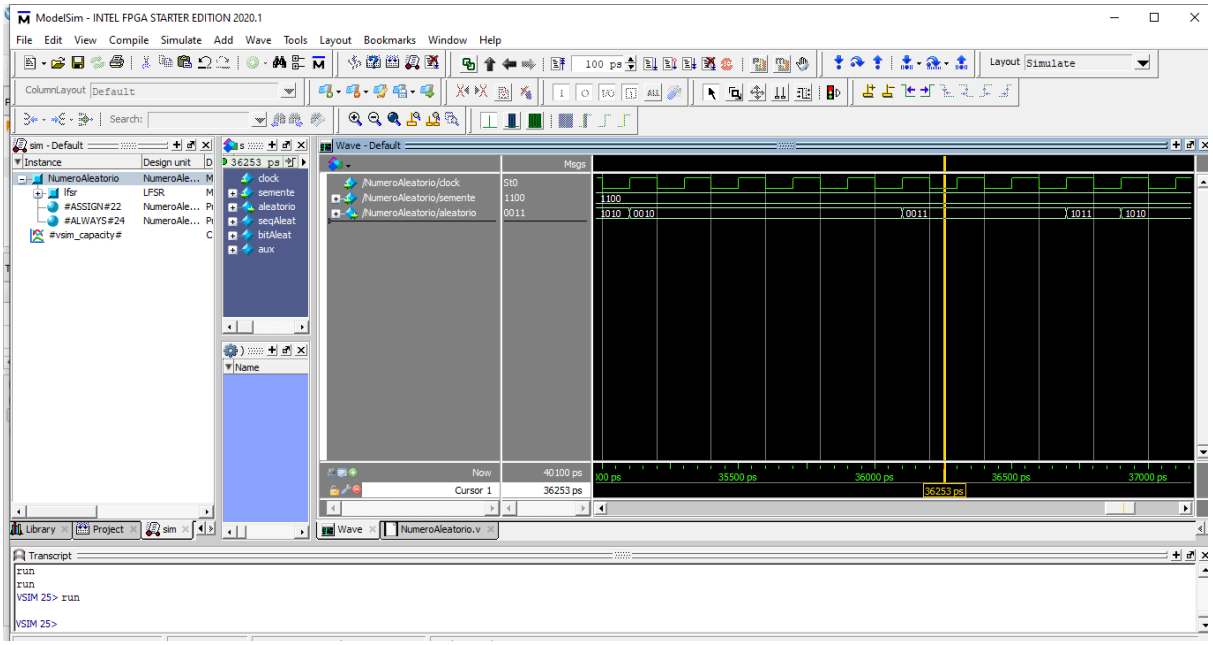
Ademais, é relevante esclarecer como tal mecanismo funciona para que assim terceiros possam também implementá - lo. Primeiramente, o LFSR é executado como uma série de Flip - flops conectados por um registrador de deslocamento possuindo diversos torque fora da cadeia que são usados como entrada para portões de tipo XOR ou XNOR. A saída dessa porta é utilizada então como feedback para o início da cadeia de registro de turno.

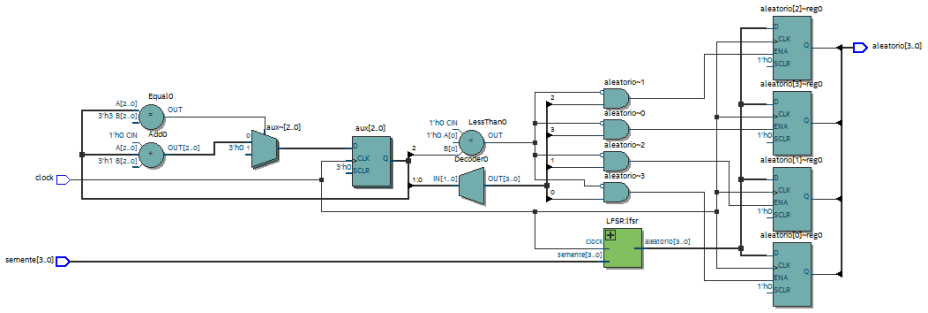


Além disso, é válido ressaltar que o padrão gerado pelos Flip - flops individuais não é totalmente aleatório visto que a partir de um estado inicial do LFSR consegue - se prever o próximo estado, contudo tal metodologia foi implementada no projeto da disciplina uma vez que o padrão gerado é próximo de uma aleatoriedade.

Sendo assim, pondera - se que o módulo 2 na qual ficou responsável pela geração da maçã utilizou de um mecanismo eficiente e coeso com a proposta da disciplina assim como com a finalidade do projeto, pois apesar de não ser completamente aleatório o padrão de saída é bem próxima do esperado além de seus custos serem bastante ínfimos contribuindo assim para uma melhor execução do projeto.

Abaixo, encontra-se a simulação de ondas e o módulo sintetizado:





Módulo Main:

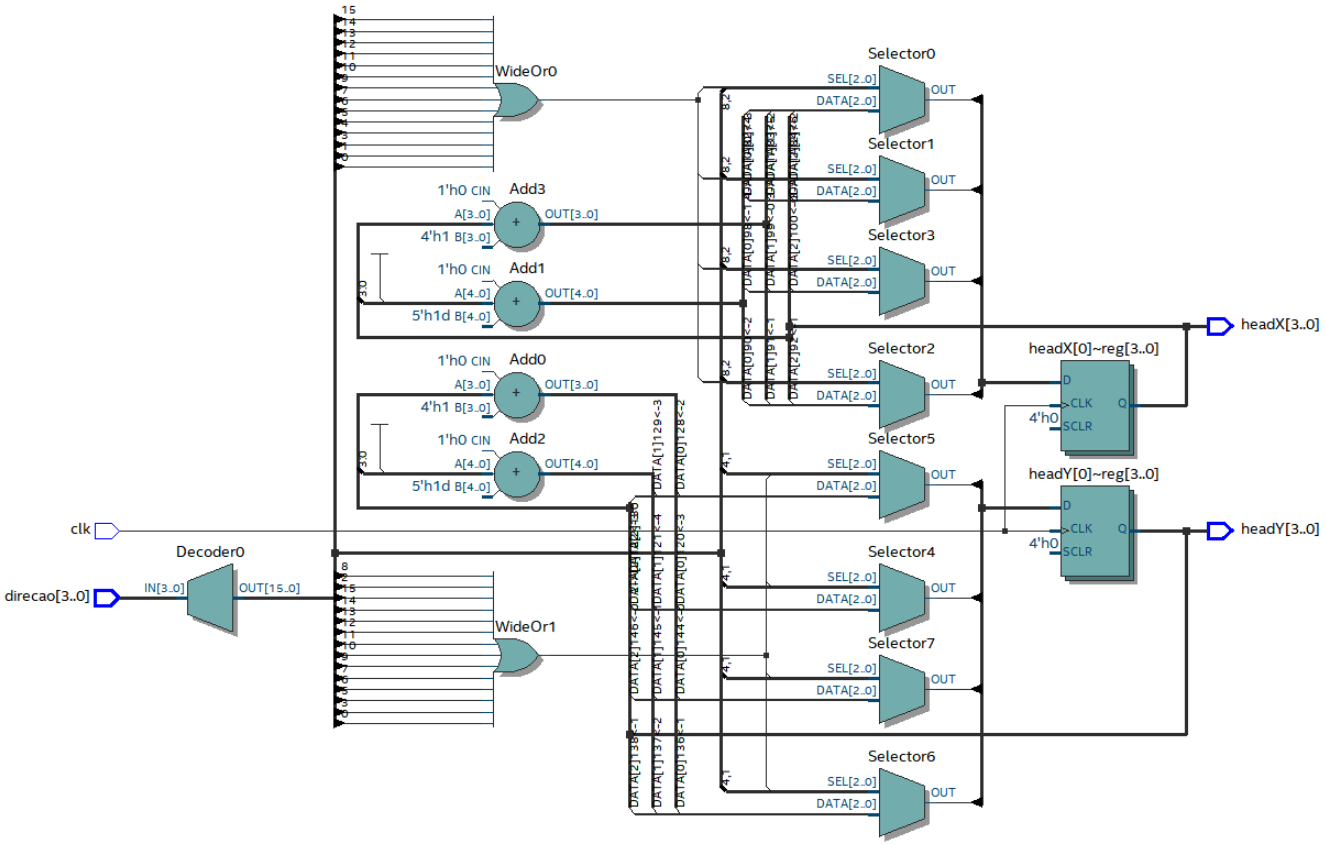
Parte: move cabeça

O terceiro módulo está associado a uma das funções essenciais da própria lógica do jogo, a da movimentação da Cabeça. Essa parte é uma das que mais determinam o comportamento e funcionamento do jogo, uma vez que ela é a conexão dos inputs com o corpo da prova, e também a responsável por possivelmente resultar situações de finalização do jogo (por meio das colisões)

A decisão em que optamos no projeto foi a de usar 4 bits para definir o tabuleiro, portanto, cada eixo tem 16 posições. Dessa maneira, para orientar

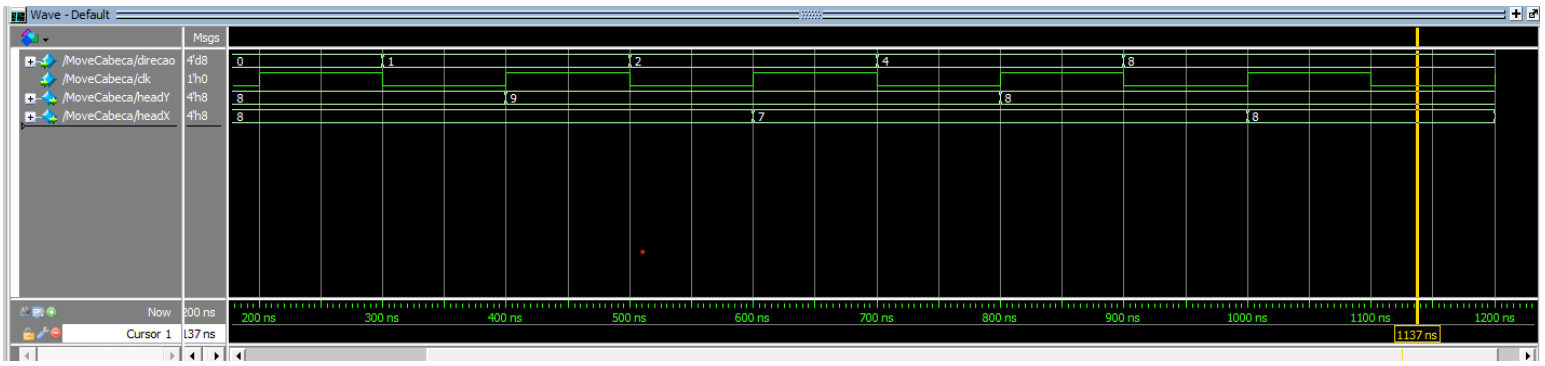
A sua lógica principal é vista em um case onde há a separação dos casos de movimentação da cobra dependendo do input gerado pelo registrador oriundo do módulo de leitura dos botões. Dessa forma, há a associação do resultado dos inputs com a posição da cabeça da cobra no tabuleiro por meio da soma e subtração de valores nos registradores headX e headY.

O módulo sintetizado (gerado dentro do Quartus):

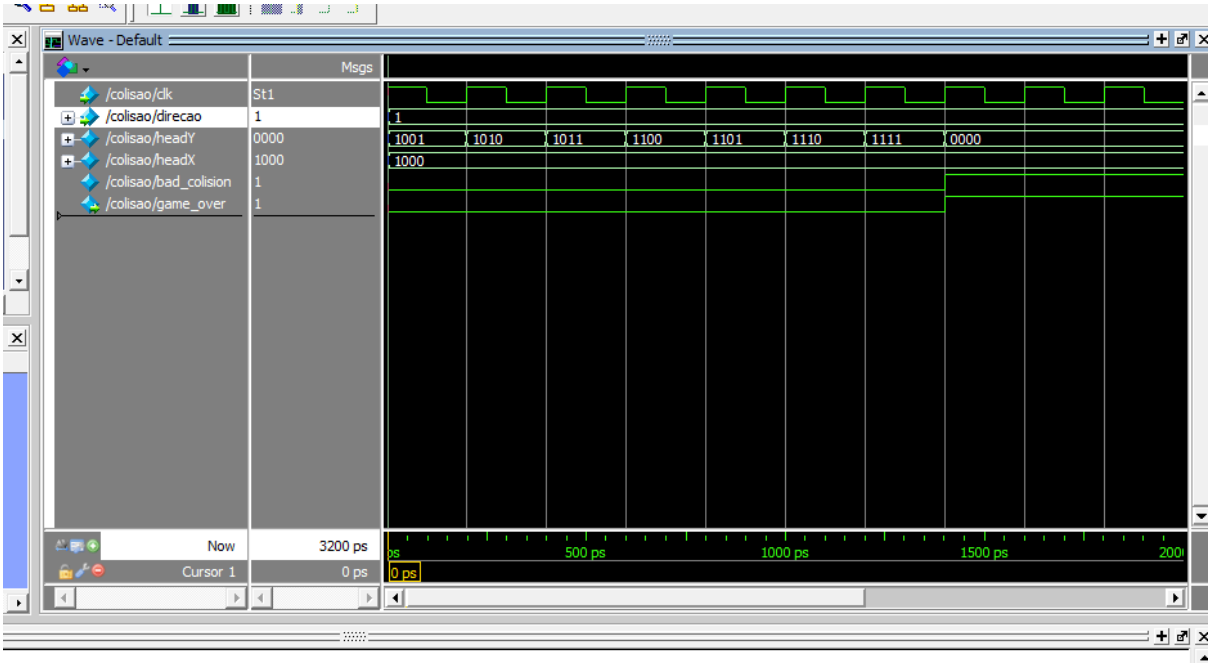


Nota-se o que o que define a movimentação da cabeça no sistema de coordenadas são os 4 somadores denominados Add(0-4). Ao código ser unido na main, foi-se adicionada uma condicional que avalia se o jogo não está em estado de game over, para que a cabeça não continue se movimentando (essa condicional não foi mostrada na foto

Abaixo segue a simulação individual da parte (clock de 200 ns):



Parte: colisão com a parede



Como foi definido pelo grupo que o jogo teria um tabuleiro 16x16 e usaria 2 coordenadas (x,y) com 4 bits, foi representado que cada coordenada na sua extremidade, ou seja, 0000 em binário/ 0 em decimal ou 1111 em binário/ 15 em decimal, seria uma borda do jogo, ou seja, se a posição da cabeça colidir com a posição de alguma borda, o output game over e ativo (vale 1) e o jogo acaba, encerrando a movimentação da cobrinha. Para essa parte, foi feita uma simulação separada, como mostra a imagem acima, em que pode-se perceber que quando o valor de uma coordenada (headY) começou a valer 1111/ borda, na próxima borda de clock o game over foi ativo, fazendo a cobrinha parar e o jogo ser finalizado. Logo, o código basicamente pega coordena da cabeça da cobra e compara com 1111, e 0000, para ver se ela é uma extremidade.

Parte: colisão com o corpo

Para fazer o sistema de detecção de colisão do corpo foi usado o fato de que o programa está salvando cada posição da cobrinha num array de 8 bits de dimensão

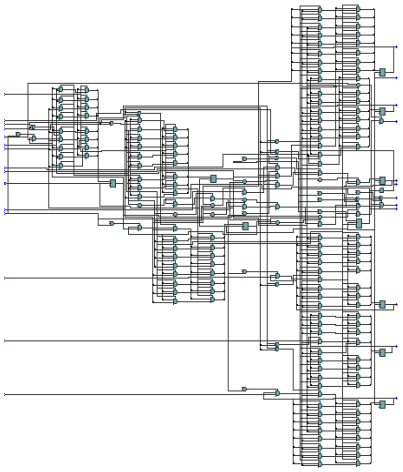
com 128 espaços. Nele, há as informações que guardam as coordenadas concatenadas, no sentido do eixo x depois do eixo y, de cada posição do corpo da cobrinha. Assim, para detectar se há alguma colisão entre o corpo e a cabeça, foi utilizado um loop finito, que tem alcance de 0 até o tamanho atual da cobra, que compara o valor de cada corpo da cobrinha com a concatenação do valor da cabeça no mesmo sentido salvo no array, ou seja, eixo x nos 4 primeiros bits e o eixo y nos 4 últimos bits. Logo, caso alguma das comparações seja verdadeira, o circuito atualiza o valor de game over para 1, encerrando o jogo.

Parte: colisão maçã e atualização do corpo da cobra:

Essa parte do projeto foi unida, com o fim de evitar uma possível colisão que não deveria ocorrer (ao atribuirmos o valor da cabeça à do corpo que imediatamente a precede). Essa lógica funciona da seguinte forma: A primeira comparação feita é para avaliar se houve ou não uma colisão com a maçã (que é feita de forma direta dentro do if), caso não haja, a cobra se move de forma padrão e passa as informações das seções posteriores do array corpo para as anteriores (o array é ordenado de forma que a posição mais próxima à cabeça seja a mais distante de corpo[0], ou seja, sendo corpo[tamanho-1]), que é feito atribuindo corpo[i+1] para corpo de [i]. Isso define o movimento característico da cobra no jogo, e é completamente definido apenas pelos inputs, já que cada segmento n da cobra representa a posição da cabeça em n “frames” de jogo atrás.

Caso haja a colisão com a maçã (denominada colisão não letal), o índice “tamanho” é acrescido de 1, e a parte mais próxima à cabeça recebe a própria posição da cabeça. Isso pode ser interpretado como a cobra “ficando parada” por um ciclo, enquanto apenas a cabeça se move e mais um segmento é adicionado na posição anterior da cabeça.

Circuito final sintetizado



(com muito zoom out para possivelmente caber dentro do pdf)

Considerações finais

Testbench

A testbench foi programada com o intuito de setar valores para os inputs dos botões, simulando a interação de um usuário. A cada borda de clock, um novo input é gerado, formando um caminho de 22 passos que visa alcançar a primeira maçã do jogo. Após obtê-la, os inputs direcionam a cobrinha para a direita, fazendo ela colidir com a borda do tabuleiro.

Tal teste não consegue cobrir o caso de vitória e todos os casos de colisão da cobrinha, como o de colisão com o corpo, visto que alguns deles são extremamente difíceis de simular, já que depois da primeira maçã ser obtida, ela vai “spawnar” em (idealmente) um lugar aleatório, tornando praticamente impossível prever qual será os movimentos que levaram a cobrinha para a nova posição da maçã.

Resultados finais:

Na hora de simular o código como um todo, houve êxito na parte de percorrer o tabuleiro com a cobrinha e sua colisão com a parede, mas houve um problema na geração aleatória da maçã, fazendo com que a cada borda de clock o valor gerado

fosse “degradado”. Porém, quando testado separadamente, a geração deu certo, assim como foi detalhado especificamente na sua parte (e como foi ilustrado pelo diagrama de onda mostrado)**.** Então, achamos que o erro consiste pelo fato de ser usado apenas um clock para todas as etapas do projeto, pois o problema está em algo coletivo, ou na hora de mandar os valores pelos módulos. Mas, infelizmente não conseguimos identificar o problema a tempo.

Assim como pode ser analisado pelo circuito final sintetizado, a proposta do projeto foi relativamente complexa e desafiadora em comparação às exigências (pelo menos um circuito combinatório, um sequencial, e elementos de memória - basicamente todas as nossas partes do projeto tiveram elementos desse tipo).

Algo que é válido de se mencionar é com relação à proposta da matriz de led. No início do semestre, quando decidimos o nosso projeto, não tínhamos noção de como funcionavam as ferramentas que usamos para desenvolver o projeto, e muito menos como possivelmente o iríamos simular. Portanto, pensamos que a melhor forma de representar o projeto fisicamente seria com uma matriz de led, entretanto, ao desenvolvermos o projeto e adquirirmos noção sobre o funcionamento dos simuladores (Questa e Modelsim), chegamos à conclusão que seria dificultada a realização da simulação direta apenas da matriz de led pensada na ideia do projeto. Sendo assim (após conversarmos sobre esse impasse com o professor), realizamos nossa simulação e análise do jogo diretamente a partir dos valores assumidos pelos índices de game-over, win, corpo e headY/headX, que podíamos ver e manipular por meio dos diagramas de onda.

Referências bibliográficas

- RUSSELL. Linear feedback shift register for FPGA. Disponível em: <https://nandland.com/lfsr-linear-feedback-shift-register/>. Acesso em: 30 nov. 2022.
- https://www.instructables.com/Snake-on-an-FPGA-Verilog/
- https://www.youtube.com/playlist?list=PLXyWBo_coJnN2rro0EMW5J9ruTDEgtdXg
- Digital Design: Principles and Practices; John f. Wakerly.

- https://fpgacademy.org/tutorials.html
- https://www.hdlworks.com/hdl_corner/verilog_ref/items/PortDeclaration.htm
- https://verilogguide.readthedocs.io/en/latest/verilog/testbench.html