Snaklnator

Mini-projeto Final LSDig. - José Duarte (64194) e Mariana Mendes (45422) - PR23

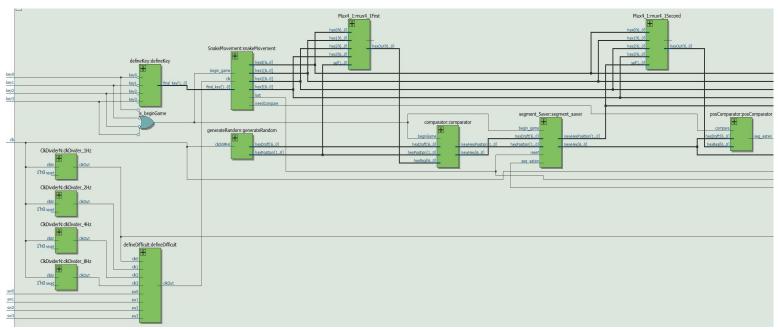
1. Introdução

O projeto que propusemos desenvolver trata-se do clássico jogo "Snake". Neste jogo, o jogador manipula uma linha, usualmente chamada de cobra, de modo a que ela apanhe o segmento que vai aparecendo no display do dispositivo em posições aleatórias. O jogador perde quando a cobra colidir com uma das paredes limite do jogo. Tínhamos como objetivo inicial desenvolver o jogo nos displays de 7 segmentos e fazer com que a FPGA emitisse um som quando o segmento é apanhado.

A interação com o jogador é feita através das KEYs (movimento) e dos SWITCHES (seletor de dificuldade) da placa. Para iniciar o jogo, basta que o jogador carregue numa qualquer das KEYs sabendo que o movimento da cobra será influenciado pela KEY que pressionar. Quando o jogador perde, é apresentado nos displays o score (displays mais à esquerda) e o tempo no formato mm:ss (displays mais à direita). Nos displays do meio podem observar-se as iniciais do nome do projeto, S I (SnakeInator).

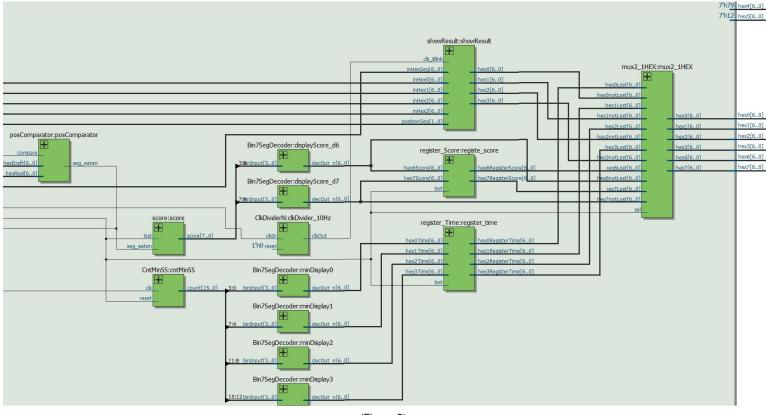
2. Arquitetura

Na figura 1 estão presentes a primeira e segunda fases do projeto. A primeira fase consiste na parte responsável por definir qual a atual e próxima posição em função das instruções dadas pelo jogador. A esta fase acresce ainda a particularidade de definir a dificuldade efetivada através de diferentes sinais de relógio. A segunda fase consiste na geração do segmento aleatório, na comparação de forma a que esse segmento não coincida com a cobra, registado novo segmento e mantido até que novo comparador verifique a interseção com a cobra.



(Figura 1)

A terceira fase, ilustrada pela figura 2, consiste em registar o score e o tempo de cada tentativa.



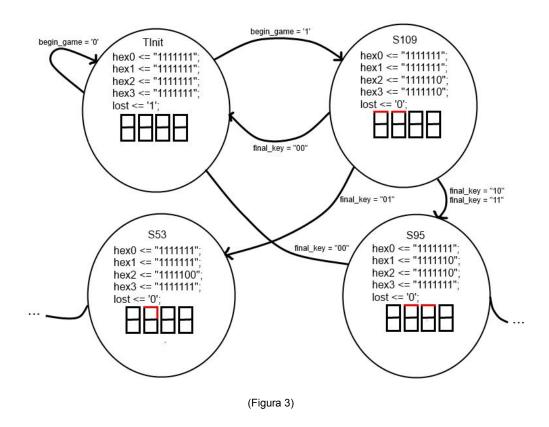
(Figura 2)

2.1 Breve descrição de cada módulo:

- **defineDifficult:** Gera 4 sinais de relógio e, através da interação do utilizador com os *SWITCHES*, seleciona um único de forma a alterar a dificuldade do jogo, ou seja, diminuindo/aumentando o tempo de mudança de estado para estado.
- *defineKey*: Gera a utilização das KEYS por parte do utilizador selecionando sempre a última que pressionou no intervalo de tempo entre cada estado.
- snakeMovement: Gera os estados da cobra: a posição atual e a posição para a qual deve seguir.
- *generateRandom*: Gera um número aleatório de 0 a 6 (correspondente aos segmentos de um *display*) e outro de 0 a 3 (correspondente aos 4 *displays*).
- *comparator*: Compara o segmento gerado no módulo anterior com a posição atual da cobra e se igual, altera.
- **segment_Saver**: Recebe o segmento do módulo anterior e mantêm-no até que seja encontrada uma colisão de cobra-segmento.
- posComparator: Encarrega-se de fazer a comparação do segmento registado no módulo anterior com a posição atual da cobra (cada vez que existe uma alteração de posição) e comunica-o ao mesmo módulo. Na sequência disto segment_Saver recebe novo segmento do comparator.
- Mux4_1: Multiplexer que seleciona qual o display a comparar com o segmento no comparator ou no posComparator.

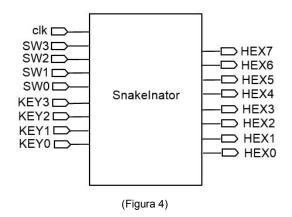
- **Score:** Contador que incrementa sempre que a cobra intersecta o segmento gerado.
- *CntMinSS*: Contador (com algumas particularidades) que incrementa a um sinal de relógio de 1 Hz e começa a contar quando o jogador começa cada jogo.
- register_Score: Guarda a pontuação no momento em que o jogador perde e mostra-a enquanto não é iniciado novo jogo.
- register_Time: Guarda o tempo no momento em que o jogador perde e mostra-o enquanto não é iniciado novo jogo.
- **showResult**: Faz a combinação da informação dos *displays* do **snakeMovement** com a do segmento do **segment_Saver** colocando o segmento a piscar a uma certa frequência (10 Hz) enquanto a cobra continua contínua.
- *Mux2_1HEX*: *Multiplexer* que permite que, no caso de o jogador ter perdido e ainda não ter começado novo jogo, seja mostrado a pontuação e tempo do jogo anterior e, caso ainda não tenha perdido, seja mostrado "o jogo em si" e o score a incrementar.

3. Implementação



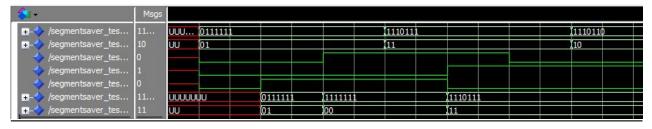
A figura 3 descreve um exemplo dos estados implementados. Para uma melhor perceção da imagem, é de salientar que os valores de *final_key* a "00", "01", "10" e "11" representam, respetivamente, a direção cima, baixo, esquerda e direita. A entrada *begin_game* serve para que, quando o jogo se encontra no estado *Tlnit*, ao ser pressionada qualquer KEY, o jogo começará.

A figura 4 ilustra a ligação a periféricos do kit DE2-115.

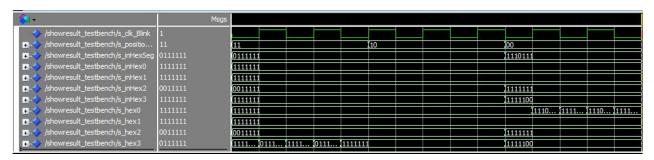


4. Validação

Nas figuras seguintes podemos ver dois exemplos de simulação feitos. O primeiro (figura 5) representa o módulo **segment_Saver** o segundo (figura 6) o módulo **showResult**.



(Figura 5)



(Figure 6)

5. Conclusão

Com este trabalho sentimos que aprendemos bastante sobre o comportamento do *hardware*. Usando a linguagem VHDL como ferramenta foi possível aplicar os nossos conhecimentos e vê-los em funcionamento na prática o que é muito gratificante. Quando nos propusemos a este projeto não pensámos encontrar tantas dificuldades na sua resolução e, por isso, não conseguimos cumprir a agenda faseada que tínhamos planeado. Decorrente disto, o fator tempo não nos deixou implementar a funcionalidade de emissão de um som quando o "segmento é apanhado", tal como referimos na introdução. Ainda assim, achamos que todas estas dificuldades nos ajudaram muito a perceber o que está a acontecer, a procurar soluções para o resolver e, efetivamente, a resolvê-lo.