Centro de Informática
Universidade Federal da Paraíba

Relatório Projeto Final

Mole Attack

Wesley Ribeiro Farias de Araújo

Renan Carlos Gomes de Farias

João Pessoa, 2018

Centro de Informática
Universidade Federal da Paraíba

Mole Attack

Relatório elaborado para o projeto final da disciplina Circuito Lógicos II, ministrada pelo Professor Eudisley Gomes dos Anjos do Centro de Informática da Universidade Federal da Paraíba.

João Pessoa, 2018

Resumo

Mole Attack é um jogo de video game baseado no popular jogo de fliperama *Whac-a-Mole.* O objetivo deste último é acertar a cabeça de toupeiras que saem de buracos no chão. A proposta desse projeto é de implementar esse jogo em um FPGA a fim de colocar em prática os conhecimentos adquiridos na disciplina.

Esse relatório mostra em detalhes o código desenvolvido em Verilog e a manipulação dos componentes presentes no FPGA modelo DE2-155 utilizados como meio de interação do usuário com o jogo.

Palavras-chave: Verilog, FPGA, *Mole Attack*.

Lista de siglas

VHDL - Hardware Description Language (Linguagem de descrição de hardware)

FPGA - Field Programmable Gate Array (Arranjo de portas programáveis em campo)

IDE - Integrated Development Environment (Ambiente de desenvolvimento integrado)

Sumário

1. Introdução

Mole Attack é um jogo de videogame baseado no popular jogo de fliperama *Whac-a-Mole.* O objetivo deste último é acertar a cabeça de toupeiras que saem de buracos no chão. Se o jogador não atingir uma toupeira dentro de um certo tempo ou com força suficiente, ela acabará por afundar de volta em seu buraco sem pontuação. Embora a jogabilidade comece devagar o suficiente para a maioria das pessoas atingirem todas as toupeiras que sobem, ela aumenta gradualmente em velocidade, com cada toupeira gastando menos tempo acima do buraco e com mais toupeiras fora de seus buracos ao mesmo tempo. Após um limite de tempo designado, o jogo termina, independentemente da habilidade do jogador. A pontuação final é baseada no número de moles que o jogador atingiu.

A proposta desse projeto é de implementar esse jogo em um FPGA, usando os dispositivos integrados à própria placa como interface com o usuário, a fim de colocar em prática os conhecimentos adquiridos na disciplina. Foram feitas algumas modificações ao jogo original, como o tempo em que a toupeira fica para fora de buraco, a quantidade de níveis e a forma de contar a pontuação. Esse alterações foram necessárias para adaptar o jogo as limitações do hardware utilizado.

2. Metodologia

Para a elaboração do projeto, foi utilizado um FPGA modelo DE2-155 mostrado na Figura 1, onde foram utilizadas os 8 LEDs verdes, os 4 botões, os 3 displays de 7 segmentos e o display LCD integrados a placa. O código do projeto foi feito na linguagem HDL Verilog usando o ambiente de desenvolvimento integrado (IDE) Quartus II 13.0sp1 Web Edition.

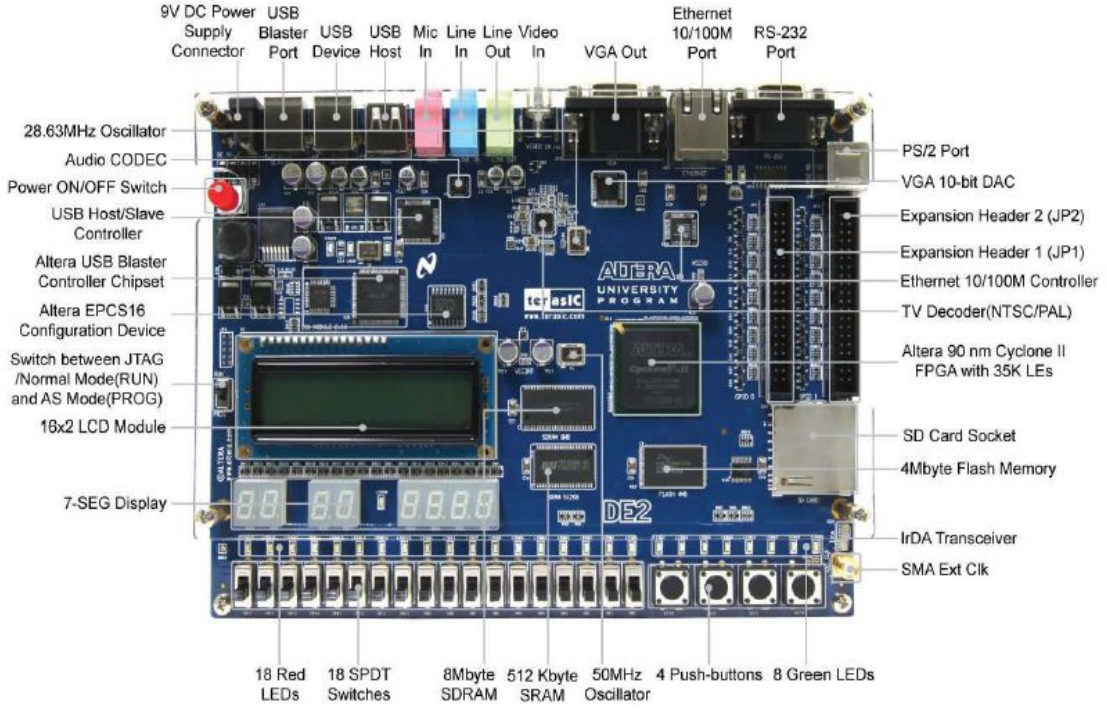

Figura 1 - FPGA modelo DE2-155

3. Descrição do Projeto

O jogo funciona da seguinte forma:

- Os LEDs verdes acima dos botões representam a toupeira;
- Para capturar uma toupeira é necessário apertar o botão correspondente ao LED acesso;
- São 4 níveis que vão aumentando a cada 5 acertos:
    - Cada toupeira do nível 1 vale 30 pontos;
    - Cada toupeira do nível 2 vale 50 pontos;
    - Cada toupeira do nível 3 vale 70 pontos;
    - Cada toupeira do nível 4 vale 100 pontos;
- O objetivo é capturar 20 toupeiras;
- Se errar 3 vezes o jogo termina (game over);
- Os displays de 7 segmentos indicam o nível, os erros e a pontuação;
- O display LCD mostra frases como:
    - X toupeiras capturadas
    - Game Over
    - Parabéns você venceu!

obs.: X é o número de toupeiras capturadas

4. Execução do Projeto, Testes e Resultados

A execução do projeto foi feito no Quartus II utilizando a linguagem Verilog. O módulo é declarado conforme mostra a Figura 2. São 4 botões denominados de KEY, 7 Leds verdes denominados de LEDG, os displays de 7 segmentos são denominados de HEX, as demais entradas e saídas são referentes ao display LCD.

```
module moles_attack (
    input CLOCK_50,
    input [3:0] KEY, //Botoes
    output [8:0] LEDG,     //    LED Green
    output reg [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7, //Display de 7 segmentos
    inout [35:0] GPIO_0,GPIO_1,     //    GPIO Connections
   //---> LCD Module 16X2
    output LCD_ON,         // LCD Power ON/OFF
   output LCD_BLON,       // LCD Back Light ON/OFF
   output LCD_RW,         // LCD Read/Write Select, 0 = Write, 1 = Read
   output LCD_EN,         // LCD Enable
   output LCD_RS,         // LCD Command/Data Select, 0 = Command, 1 = Data
   inout [7:0] LCD_DATA);  // LCD Data bus 8 bits
```

Figura 2 - Módulo moles_attack

As variáveis que fazem a comunicação entre esse módulo e o módulo do LCD são denominadas de lcd e são atribuída a elas as variáveis denominadas de auxiliar conforme mostra a Figura 3. A funcionalidade de cada uma é a seguinte:

- auxiliar0: identifica a frase que deve ser mostrada no display LCD
    - Se for igual a 0 a frase no LCD é “Capture a toupeira”
    - Se for igual a 1 a frase no LCD é “X toupeiras capturadas”
    - Se for igual a 2 a frase no LCD é “Toupeira escapou”
    - Se for igual a 3 a frase no LCD é “Game Over”
    - Se for igual a 4 a frase no LCD é “Parabéns você venceu!”
- auxiliar1: influencia na contagem do auxiliar7
- auxiliar2: influencia na contagem do auxiliar6
- auxiliar5: variável que vai determinar se o jogador ganhou
- auxiliar6: contador do bit da esquerda que é mostrado no LCD
- auxiliar7: contador do bit da direita que é mostrado no LCD

```
wire [6:0] lcd0, lcd1, lcd2, lcd5, lcd6, lcd7;
assign lcd0 = auxiliar0; //estado do lcd
assign lcd1 = auxiliar1; // influencia auxilia7
assign lcd2 = auxiliar2; // influencia auxilia6
assign lcd5 = auxiliar5; //saber se ganhou
assign lcd6 = auxiliar6; //contagem das topeiras 1 casa do display
assign lcd7 = auxiliar7; //contagem das topeiras 2 casa do display
```

Figura 3 - Variáveis de comunicação com LCD

As variáveis são inicializadas conforme mostra a Figura 4. O contador de acertos é denominado **cont_a** e o de erros **cont_e**.

```
initial
   begin
   cont_a = 'd0;
   cont_e = 'd0;
   auxiliar0 = 'd0;
   auxiliar1 = 'd0;
   auxiliar2 = 'd0;
   auxiliar5 = 8'h0;
   auxiliar6 = 8'h0;
   auxiliar7 = 8'h0;
   end
```

Figura 4 - Inicialização das variáveis

O trecho do código que faz a comunicação com o módulo do LCD é mostrado na figura 5. São passados como parâmetros as variáveis **lcd**, **cont_a**, **cont_e** e os botões.

Esse módulo LCD utilizado foi disponibilizado em um dos exemplos do professor Eudisley através do Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA). Ele foi alterado de forma a mostrar as frases necessárias.

```
    LCD_Display u1(
// Host Side
    .iCLK_50MHZ(CLOCK_50),
    .iRST_N(DLY_RST),
    .lcd0(lcd0),
    .lcd1(lcd1),
    .lcd2 (lcd2),
    .lcd3 (lcd3),
    .lcd6 (lcd6),
    .lcd7 (lcd7),
    .cont_a(cont_a),
    .cont_e (cont_e),
    .botoes (KEY[3:0]),
// LCD Side
    .DATA_BUS(LCD_DATA),
    .LCD_RW(LCD_RW),
    .LCD_E(LCD_EN),
    .LCD_RS(LCD_RS)
);
```

Figura 5 - Comunicação com o módulo LCD_Display u1

O principal Always do projeto, que controla toda a lógica do jogo, é mostrado na Figura 6. Toda vez que um botão é pressionado, ele é executado. A função dele é verificar se o botão foi pressionado junto com o LED correspondente, se sim, o contador de acertos é incrementado, bem como os auxiliares correspondentes a cada bit do LCD. Caso o botão foi pressionado sem o LED correspondente estar aceso, então o contador de erros é incrementado. Posteriormente é verificado o game over ou se o jogador venceu, através da comparação do cont_e e cont_a.

```
always @(negedge KEY[rand])begin
      if(MOLE[rand] & ~KEY[rand]) begin
         cont_a <= cont_a + 1;
         auxiliar0 <= 'd1;
         auxiliar1 <= auxiliar1 + 1;
         auxiliar2 <= auxiliar2 + 1;
      end
      else begin
         cont_e <= cont_e +1;
         auxiliar0 <= 'd2;
         if(cont_e == 'd3) begin
            auxiliar0 <= 'd3;
            auxiliar2 <= 'd0;
            cont_a <= 'd0;
            auxiliar1 <= 8'h0;
         end
         else begin
            if(cont_a == 'd20)begin
               auxiliar0 <= 'd4;
            end
         end
      end
   end
```

Figura 6 - Principal Always

O local onde a toupeira aparece foi determinado aleatoriamente. Para isso, foi implementado um *Linear Feedback Shift Register (LFSR)*, que gera uma sequência de números pseudo-aleatórios. Um LFSR é simplesmente um registrador de deslocamento com alguns de seus bits ligados a um porta XOR para criar um *feedback*. Neste projeto, foi implementado um LFSR de 5 bits, capaz de gerar 31 estados aleatórios antes de começar a se repetir, o que é mais do que suficiente para as 20 toupeiras do jogo. A Figura 7 mostra o código.

```
module LFSR_counter
#(parameter BITS = 5)
(
  input              clk,
  //input              rst_n,

  output reg [4:0] data
);
reg [4:0] data_next;

initial
begin
   data <= 5'h1f;
end

always @* begin
  data_next[4] = data[4]^data[1];
  data_next[3] = data[3]^data[0];
  data_next[2] = data[2]^data_next[4];
  data_next[1] = data[1]^data_next[3];
  data_next[0] = data[0]^data_next[2];
end

always @(posedge clk /*or negedge rst_n*/) begin
 // if(!rst_n)
  //  data <= 5'h1f;
 // else
    data <= data_next;
end
endmodule
```

Figura 7 - Módulo LFSR.

4.1 Testes e Resultados

Durante a fase de testes, foi verificado que o número de acertos estava sendo contado corretamente, porém o número de erros não estava. Toda vez que o jogador passava de nível, ou após algum tempo, o contador de erros era incrementado automaticamente, além de algumas vezes o jogo ignorar quando o jogador cometia erro. Essas falhas decorrem, provavelmente, da lógica utilizada para para a verificação de quando houve acerto e erro.

Para contornar esse problema, a parte lógica do jogo foi refeita algumas vezes, porém a cada nova iteração de testes com as novas partes lógicas, o jogo apresentava novos problemas que o tornavam impossível de ser jogado. Diante dessa situação, o grupo decidiu que a versão final seria aquela com o menor número de problemas, e que tivesse um nível de jogabilidade aceitável.

5. Conclusões

Embora a implementação do jogo tenha tido alguns erros, ainda assim o desenvolvimento do projeto foi de suma importância para colocar em prática o que vimos na teoria, servindo também para nos mostrar em detalhes as etapas da criação de uma aplicação bem como as habilidades necessárias para o desenvolvimento de projetos em Verilog.

Dessa forma, aprendemos a utilizar o FPGA e o Quartus II, ferramentas essas que serão de grande utilidade no decorrer da vida acadêmica e profissional.

6. Referências

The Whac-A-Mole Story. 2010. (4m53s). Disponível em: <https://www.youtube.com/watch?v=Agjaa1DyKyA>. Acesso em: 07 jun. 2018.

http://simplefpga.blogspot.com/2013/02/random-number-generator-in-verilog-fpga.html <Acesso em: 07 jun. 2018>.

http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf

<Acesso em: 07 jun. 2018>.

http://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf <Acesso em: 07 jun. 2018>