Centro de Informática
Universidade Federal da Paraíba

Relatório Final

Jogo do Pega-Ladrão em Verilog no Altera FPGA DE2

Bruno Sidnei Guerra Passeti

Claudio Souza Brito

João Gabriel Soares de Melo

João Pessoa, 2018

Centro de Informática
Universidade Federal da Paraíba

Jogo do Pega-Ladrão em Verilog no Altera FPGA DE2

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

O presente projeto foi desenvolvido como Projeto Final da disciplina de Circuitos Lógicos 2, ministrada pelo Professor Eudisley Anjos. Utilizando do FPGA Altera DE2, com a linguagem Verilog, foi reproduzido uma réplica do jogo de Pega-Ladrão, semelhante ao renomado jogo de fliperama “Whac-A-Mole”, onde o objetivo é bater nos objetos que surgem de buracos na plataforma. O projeto foi dividido em 3 partes: Controle do tempo, controle dos displays de sete segmentos e a parte lógica do jogo. A primeira parte usa do clock interno do FPGA de 27Mhz para criar os segundos e minutos utilizados no projeto. Isto criava um cronômetro que delimitava o tempo de cada partida. O controle dos displays era feito a partir de um decodificador que recebia o sinal de entrada no formato BCD (binary coded decimal) para o formato dos display de sete segmentos. Já a parte da lógica do jogo começa com a geração de um conjunto de bits, que forma um número pseudo-aleatório através do embaralhamento dos mesmos por XORs e NOTs entre os próprio elementos deste conjunto. O número pseudo-aleatório irá dizer em qual posição o ladrão irá aparecer, e esta será comparada com a posição das chaves do FPGA, se ambas forem iguais, 1 ponto é adicionado no placar. Caso não forem iguais, após 3 segundos os LEDs são apagados. Por fim, se o tempo chegar até zero ou se os pontos adiquiridos forem iguais à meta do nivel, a partida terminará e o jogo irá para o estado de espera.

Palavras-chave: HDL, Verilog, FPGA.

Lista de figuras

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)

GPIOS – General Purpose Input/Output (Portas de entrada e saída de dados de propósito geral)

RTC – Real Time Clock (Relógio de Tempo Real)

Sumário

1. Introdução

Alguns tipos de aplicações computacionais exigem uma capacidade de flexibilidade e velocidade de adaptação que nem sempre é possível realizar em um computador ou em um chip projetado especificamente para apenas uma determinada funcionalidade. Para estes casos, uma excelente opção para resolver os problemas é a utilização de um FPGA (Field Programmable Gate Array).

Os FPGAs tem diversas possibilidades de utilização, pois os mesmos se propõem a resolver problemas em hardware sem a necessidade de construir um hardware específico para cada situação. O projetista poderia alterar o sistema de acordo com as suas necessidades, sem precisar reconstruir outra placa de circuito impresso todas as vezes que necessitar de alguma mudança física. Assim, os FPGAs podem ser utilizados como protótipos de sistemas eletrônicos, fornecendo, assim, custos reduzidos. Neste trabalho será detalhada a configuração e desenvolvimento do Game Pega-Ladrão apresentado como Projeto Final da disciplina. Desenvolvido na Altera Cyclone II, e codificado em Linguagem Verilog.

2. Metodologia

Neste projeto foi utilizada a linguagem de descrição de hardware (HDL) Verilog, o circuito integrado FPGA (Field Programmable Gate Array, em português "Arranjo de Portas Programáveis em Campo") Altera DE2 e como Ambiente de Desenvolvimento Integrado (IDE) foi utilizado o software Quartus II para o desenvolvimento do código utilizado neste trabalho e para carregar o código na FPGA Altera DE2.

2.1 Quartus II

O software Quartus II, da Altera, foi utilizado em sua versão 9.1sp1, para a criação e compilação do código de exemplo em Verilog e para o carregamento do mesmo na FPGA Altera DE2.

2.2 Componentes da FPGA usados

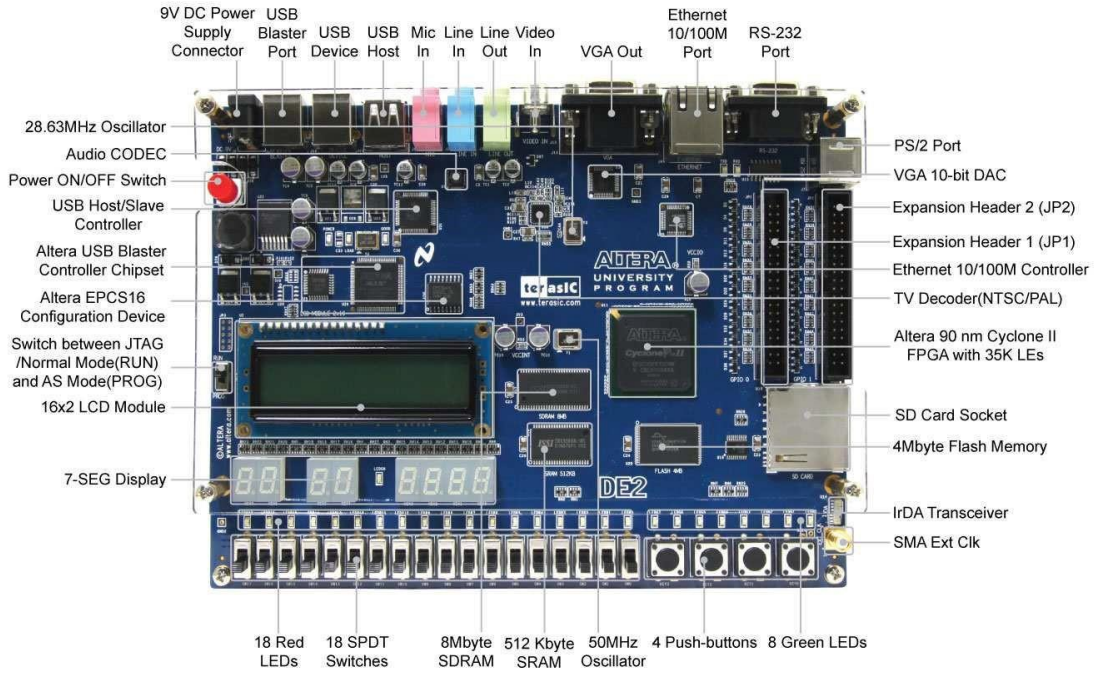

Figura 1 - FPGA Altera DE2

A FPGA possui uma imensa gama de componentes, fazendo com que ele se torne uma ferramenta muito útil para programadores. Neste projeto em particular, usamos apenas alguns para obter nossa meta, são eles: Os quatro botões (push-buttons), um do LEDs verdes, as dezoito chaves (SPDT switches), os dezoito LEDs vermelhos, e ainda sete displays de sete segmentos (7-SEG Display).

3. Descrição do Projeto

Neste tópico falaremos sobre a função de cada parte da FPGA em especifico, e explicando como elas funcionam, além de falar também sobre a lógica trabalhada na máquina.

3.1 Descrição detalhada dos componentes da FPGA

3.1.1 Push-buttons

Os botões tinham como função selecionar os níveis do jogo, do nível um a quatro (da esquerda para a direita).

3.1.2 LED verde

Apenas um LED verde foi utilizado, e ele tinha função de piscar quando jogador adquirisse ponto.

3.1.3 LEDs vermelhos

Onde é indicado a(s) posição(ões) do ladrão(ões). Estes ligam de maneira aleatória, isso será mostrado mais adiante.

3.1.4 SPDT switches

São usadas pelo jogador para “capturar” o ladrão, caso o LED vermelho correspondente estiver ligado.

3.1.5 7-SEG Display

Sendo a função dos dois primeiros (esquerda para a direita) mostrar a quantidade de pontos necessária para passar na fase, os próximos dois mostrarão o placar atual do jogador, o próximo não foi usado, e os três últimos servem para mostrar o tempo restante, de maneira semelhante a um relógio digital.

3.2 Descrição detalhada do código em verilog

Nesta sessão trataremos do código comentado, mostrando como ele funciona, e como ele se relaciona com as partes da FPGA.

3.2.1 LED verde e a passagem de tempo decrescente

A função “always” sempre irá acontecer na velocidade do clock_27, que possui uma frequência de 27Hz, sendo assim, a variável “acender” controla o momento em que a pontuação é captada, quando isso acontece, o LED verde acende (aceita o valor 1), quando isso não acontece, ele se mantem apagado (valor 0).

Ao longo do jogo, o tempo passará , e para isso temos operadores ternários para as variáveis que controlam o tempo (UNI, DEZ, CEN), se UNI chega a 0, ele volta a ser 9, se não, terá seu valor diminuído em 1. No DEZ acontecerá a mesma coisa, mas perceba que só acontecerá se UNI for 0, e de maneira semelhante, mas não igual, a contraparte dos minutos terá seu valor diminuído em 1, acho UNI e DEZ forem iguais a 0.

```
always @(posedge clk) begin
    //acendendo o LED verde quando ganha ponto
    if(acende == 1)begin
        LEDVenceu = 1'b1;
    end
    //mantendo o LED apagado quando nao ganha ponto
    else LEDVenceu = 1'b0;

    //tempo do jogo
    if(cntovf) begin
        UNI <= (UNI==4'h0 ? 4'h9 : UNI-4'h1);
    end
    if(UNI == 0) //o digito das dezenas so descresce quando o das unidade chega a 0
        if(cntovf1) begin
            DEZ <= (DEZ==4'h0 ? 4'h5 : DEZ-4'h1);
        end
    if(DEZ == 0 && UNI == 0)// "CEN" remete a casa dos minutos
        if(cntovf2) begin
            CEN <= CEN - 1;
        end
```

Figura 2 – Definição do Tempo do Jogo

Nesse mesmo Always, temos a parte relacionada ao fim do jogo, que só ocorre quando o tempo se esgotar, ou o jogador vencer, ou o estado anterior já seja de espera. E por último, a parte referente ao início de um novo jogo, onde o tempo é restaurado para três minutos.

```
//condicao de parada do jogo, voltando ao estado de espera
if((UNI == 0 && DEZ == 0 && CEN == 0) || level == 4'b0000 || venceu == 1'b1)begin
    UNI <= 1;
    DEZ <= 0;
    CEN <= 0;
    level = 4'b0000;
    dezPoint = 4'h0;
    uniPoint = 4'h0;
end
if(level == 4'b0000 && ~BT) begin // RESET DO JOGO
    CEN <= 3;
    UNI <= 0;
    DEZ <= 0;
    acabouOTempo = 1'b0;
end
```

Figura 3 - Reset do jogo

3.2.2 Botões, e a escolha de nível

Para as escolhas de níveis usamos os quatro push buttons, o estado de jogo muda quando esses botões são apertados, mas apenas se o estado atual seja o "estado de espera". As variáveis dezpoint e unipoint são referentes aos pontos necessários para passar de fase, e eles variam dependendo do nível.

```
//level 1
if(BT == 4'b1110 && level == 4'b0000) begin
    level = 4'b0001;
    dezPoint = 4'h1;
    uniPoint = 4'h0;
end
//level 2
if(BT == 4'b1101 && level == 4'b0000) begin
    level = 4'b0010;
    dezPoint = 4'h1;
    uniPoint = 4'h5;
end
//level 3
if(BT == 4'b1011 && level == 4'b0000) begin
    level = 4'b0011;
    dezPoint = 4'h2;
    uniPoint = 4'h0;
end
//level 4
if(BT == 4'b0111 && level == 4'b0000) begin
    level = 4'b0100;
    dezPoint = 4'h3;
    uniPoint = 4'h0;
end
```

Figura 4 – Escolha do Nível

3.2.3 Os displays de 7 segmentos

Lembrando que usamos sete displays, dentro do always, cada display varia de acordo com sua variável, dependendo do valor recebido, os display respondem de maneira correspondente, mas no sistema binário BCD. A estrutura na figura a seguir se repete para todos os sete displays.

```
//display de 7 segmentos
always @(*)
case(UNI)
    4'h0: SevenSeg = 8'b11111100;
    4'h1: SevenSeg = 8'b01100000;
    4'h2: SevenSeg = 8'b11011010;
    4'h3: SevenSeg = 8'b11110010;
    4'h4: SevenSeg = 8'b01100110;
    4'h5: SevenSeg = 8'b10110110;
    4'h6: SevenSeg = 8'b10111110;
    4'h7: SevenSeg = 8'b11100000;
    4'h8: SevenSeg = 8'b11111110;
    4'h9: SevenSeg = 8'b11110110;
    default: SevenSeg = 8'b00000000;
endcase

assign {segA, segB, segC, segD, segE, segF, segG,segDP} = ~SevenSeg;

reg [7:0] SevenSeg1;
//display de 7 segmentos de novo
always @(*)
case(DEZ)
    4'h0: SevenSeg1 = 8'b11111100;
    4'h1: SevenSeg1 = 8'b01100000;
    4'h2: SevenSeg1 = 8'b11011010;
    4'h3: SevenSeg1 = 8'b11110010;
    4'h4: SevenSeg1 = 8'b01100110;
    4'h5: SevenSeg1 = 8'b10110110;
    4'h6: SevenSeg1 = 8'b10111110;
    4'h7: SevenSeg1 = 8'b11100000;
    4'h8: SevenSeg1 = 8'b11111110;
    4'h9: SevenSeg1 = 8'b11110110;
    default: SevenSeg1 = 8'b00000000;
```

Figura 5 – Display de 7 Segmentos

3.2.4 Gerador de número aleatório

Dessa vez temos um always que varia de acordo com a variável UNI, que seria um tempo de aproximadamente um segundo. O que acontece basicamente é que criamos um array de 18 valores, e fazemos com que cada valor seja a operação de uma XNOR entre outros valores, ou apenas uma NOT, essas operações feitas varias vezes repetidas dessa forma faz com que seja praticamente

impossível prever os movimentos da maquina, dessa forma, ele vira praticamente um gerador de números aleatória. Após isso, dividimos o número obtido pelo número de elementos e pegamos o resto dessa divisão, dessa forma evitando que ele ultrapasse o valor de dezoito, esse número vai para o vetor de ladrões, que é logo depois passado para o vetor de LEDs vermelhos

```
//gerador de numeros aleatorios
integer number = 0;
integer countTime = 0;
always@(posedge UNI) begin
venceu = 1'b0;
    if(level) begin
        if(countTime == 0)begin
            for(i = 0; i < level; i = i + 1) begin

                array[0] = !array[1];
                array[1] = !array[2] ^ array[1];
                array[2] = !array[3];
                array[3] = !array[4] ^ array[1];
                array[4] = !array[5] ^ array[1];
                array[5] = !array[6] ^ array[1];
                array[6] = !array[7] ^ array[1];
                array[7] = !array[8];
                array[8] = !array[9] ^ array[1];
                array[9] = !array[10] ^ array[1];
                array[10] = !array[11];
                array[11] = !array[12];
                array[12] = !array[13] ^ array[1];
                array[13] = !array[14];
                array[14] = !array[15] ^ array[1];
                array[15] = !array[16] ^ array[1];
                array[16] = !array[17] ^ array[1];
                array[17] = !array[10];
                number = array%18;
                ladrao[number] = 1;
                mudou = 1'b1;
            end
        end
        LEDR = ladrao;
```

Figura 6 – Gerador de Numeros Aleatorios

3.2.5 Marcando pontos

Agora falaremos sobre como ganhar o jogo, a contagem de pontos que são ministradas pelas variáveis unihit, dezhit, hitU, e hitD só serão alteradas se o vetor de switchs for igual ao de ladrões, dessa forma o jogador capturou todos os ladrões, após isso, o ponto é contado ,mas apenas se a variável “jacontou” estiver em 0, ou seja, não tiver contado ainda.

```
//marcando pontos
if(ladrao != 18'b000000000000000000 && ladrao == SW) begin
    acende = 0;
    LEDG = 8'b00000001;
    //para nao contar mais de uma vez na mesma rodada
    if(jaContou == 1'b0)begin
        hitU = hitU +  1;
        jaContou = 1'b1;
    end
    if(hitU > 9)begin
        hitU = 0;
        hitD = hitD + 1;
    end

    if(hitD>10) hitD = 0;
    uniHit = hitU;
    dezHit = hitD;

end else LEDG = 8'b00000000;
//para nao contar mais de uma vez na mesma rodada
if(mudou == 1'b1)begin
        jaContou = 1'b0;
end
```

Figura 7 – Marcando pontos

3.2.6 Zerando o jogo

Em algum momento da jogatina, o tempo acabara, ou o jogador vence o jogo, quando isso acontece, chegamos a condição final do jogo, caso o jogo tenha acabado por motivo do jogador zerar, as variáveis vencer, e acender irão para 1, e os placares serão zerados, além do sistema entrar no “estado de espera” mais uma vez.

```
//encerrando o jogo, e zerando os placares
if(zerar == 1'b1 || acabouOTempo == 1'b1)begin
    dezHit = 1'h0;
    uniHit = 1'h0;
    if(zerar == 1'b1)begin
        venceu = 1'b1;
        acende <= 1;
    end
    hitU = 0;
    hitD = 0;
end
```

Figura 8 – Zerando o jogo

4. Execução do Projeto, Testes e Resultados

4.1. Testes Realizados

Nos primeiros experimentos da fase final do jogo, foi testada a eficácia da detecção contra múltiplos pontos em uma jogada. Por exemplo: ao aparecer um ladrão na posição 3, haveria a possibilidade de o jogador alternar o switch na posição 3 para marcar múltiplos pontos em uma mesma jogada. Isso seria uma infração às regras do jogo. Devido a esta possibilidade, foi implementada uma espécie de robustez no código para que este tipo de prática se tornasse ineficaz. Assim, fazendo com que jamais houvesse a possibilidade desta acontecer.

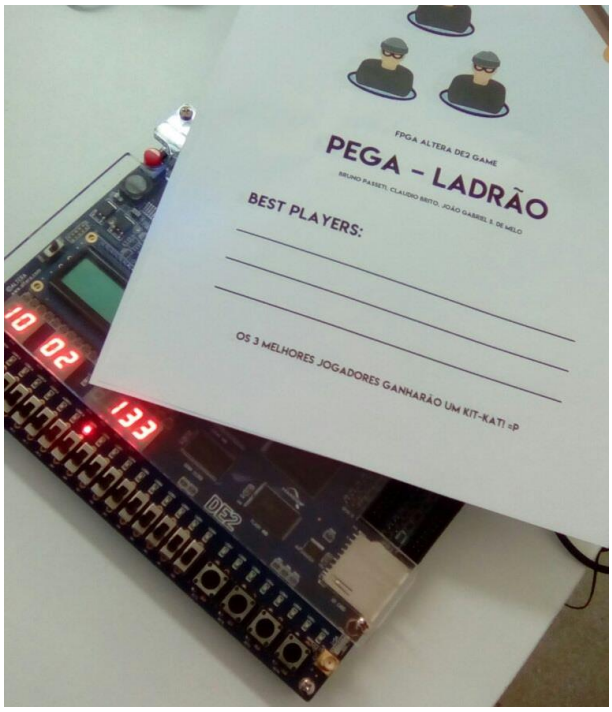

Figura 9 – Kit Altera e Pega-Ladrão em funcionamento

Além disso, foi experimentada precisão do tempo de jogo. Foram utilizados LEDs no próprio FPGA e um cronômetro externo para fazer as devidas comparações de períodos de tempo. Assim, foi observado que, na verdade, o tempo do jogo atinge um valor próximo a um segundo. Com isso, foi demonstrado que o jogador não haveria realmente três minutos reais para terminar cada rodada. Todavia, como a diferença de tempo era minúscula, outras especificações foram abordadas.

4.2. Dificuldades Encontradas

Uma das principais dificuldades encontradas foram o escopo de variáveis em Verilog. Quando uma variável se encontra sendo modificada em um bloco always, e outro bloco always tenta modificá-la, obtém-se erros de compilação. Inicialmente, o grupo enfrentou muitas dificuldades devido a estas limitações. Todavia, percebeu-se que atribuições não eram permitidas, porém, comparações sim. Por fim, o problema acabou sendo contornado com a utilização de flags (variáveis sinalizadoras), que enviavam informações de um bloco always para outro.

4.3. Sugestões de Melhorias Futuras

As principais sugestões de melhoria seriam a modificação de switches por botões. A justificativa desta mudança baseia-se na dificuldade de *gameplay* que a mecânica dos switches oferece. Para fazer estas mudanças, poderiam ser utilizados os *General Purpose Input/Output* (GPIOS), que são encontrados na parte lateral do FPGA. Com estes GPIOS poderia também ser implementado um sistema de som utilizando-se de buzzers e alto-falantes.

Um aprimoramento interessante seria a utilização do LCD da placa FPGA Altera Cyclone II, para que fossem implementados imagens e menu de jogo. Desta forma, o jogo seria mais atrativo, intuitivo e acessível ao público.

Outra possiblidade seria a utilização de módulos *Real Time Clock* (RTC) para a modificação do sistema de tempo utilizado no jogo. Desta forma, ao invés de depender do clock interno da placa FPGA, este módulo seria utilizado para medir minutos e segundos. Assim, o jogo obteria uma maior precisão no tempo e na contagem de pontos.

5. Conclusões

O trabalho realizado foi de essencial importância para a assimilação e expansão de conhecimento dos conteúdos adquiridos ao longo do curso de Circuitos Lógicos II, tendo como base todos os tópicos abordados durante as aulas teóricas e práticas.

Através dos conceitos adquiridos em sala e no desenvolvimento do projeto, entendemos como a utilização de pesquisas de documentação de Linguagens de Programação na internet são poderosas, gerando sempre uma enorme fonte de conhecimento. Assim, compreendemos melhor a utilização do FPGA em conjunto com a Linguagem Verilog e as mais variadas aplicações que poderiam ser desenvolvidas com estes artifícios tecnológicos. Além disso, o presente trabalho servirá de exemplo e consulta para quaisquer alunos que futuramente cursarem a disciplina.

6. Referências

[1] Portal de documentação e ensino de FPGA e Verilog - FPGA 4 Fun. < https://www.fpga4fun.com>. Acesso em 05/06/2018.

[2] Documentação do Verilog.
<http://www.asic-world.com/examples/verilog/index.html>. Acesso em: 07/06/2018.