UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CURSO DE CIÊNCIA DA COMPUTAÇÃO
GEX606 - SISTEMAS DIGITAIS

Trabalho Final
Blackjack

Gabriel Santos Costa
20240019345
gabrielsantoscosta005@gmail.com

Pedro Augusto Sciesleski
2311100057
psciesleski@gmail.com

Chapecó
05/12/2024

SUMÁRIO

1. INTRODUÇÃO

No contexto da disciplina de Sistemas Digitais, oferecida no curso de Ciência da Computação da Universidade Federal da Fronteira Sul, realizamos um projeto que alia teoria e prática para implementar um clássico jogo de cassino: o Blackjack (ou "21"). Nosso objetivo foi desenvolver uma máquina de estados finitos (FSM - Finite State Machine) programada em VHDL e integrada a uma FPGA DE1-Cyclone II, capaz de simular as mecânicas do jogo.

O Blackjack, em sua essência, é um jogo de cartas no qual um jogador enfrenta um carteador, buscando obter a soma de cartas mais próxima de 21 sem ultrapassá-lo. Este projeto desafiou nossa equipe a traduzir essas dinâmicas para o ambiente digital, contemplando desde a distribuição inicial de cartas até as decisões de “HIT” (pedir uma nova carta) e “STAY” (manter a mão atual). Além disso, o sistema deveria avaliar automaticamente os resultados da partida, determinando vitória, derrota ou empate.

A implementação foi conduzida em etapas bem estruturadas. Iniciamos identificando os estados necessários para o funcionamento do jogo e elaboramos um diagrama de transições que respeitasse as condições descritas no enunciado. A partir disso, projetamos e codificamos as lógicas de próximo estado e de saída em VHDL, garantindo a integração com entradas como botões de controle e displays de 7 segmentos para exibir cartas e somas.

O resultado final é um sistema funcional em grande parte, combinando lógica digital e interatividade, e atendendo à maioria das especificações propostas. Apesar de nossos esforços, algumas funcionalidades não operaram como esperado, devido a desafios encontrados durante a implementação e integração. No entanto, o projeto representou um aprendizado valioso, reforçando nosso conhecimento em VHDL e FSM e demonstrando como aplicar princípios teóricos para criar soluções práticas, mesmo diante de dificuldades.

2. DESENVOLVIMENTO

Diagrama de Estados e Descrição

O jogo de Blackjack foi estruturado em uma máquina de estados para organizar o fluxo de ações de maneira clara e intuitiva. A seguir, são descritos os estados principais:

1. Estado Inicial

Este é o ponto de partida do jogo, onde todas as variáveis necessárias são inicializadas:

- As somas das cartas do jogador e do carteador são definidas como zero.
- Sinais de controle como **`distribui`** e **`possui_as`** também são zerados.
- Em seguida, as primeiras cartas são distribuídas para ambos os jogadores.

Transição:

- Após a distribuição das cartas, o jogo segue para o estado Jogador, onde o jogador começa a tomar decisões.

2. Estado Jogador

Neste estado, o jogador tem a liberdade de escolher suas ações:

- HIT: Solicitar uma carta adicional (controlado por `sw(6)`).
  - Caso a soma das cartas ultrapasse 21, o jogo avança para o estado Resultado.
  - Se a soma for válida, o jogador permanece neste estado para novas decisões.
- STAY: Manter a soma atual (controlado por `sw(7)`) e passar a vez para o carteador.
  - Se o jogador possuir um Ás utilizável como 11, o estado muda para Jogador_AS.
  - Caso contrário, o jogo segue para o estado Carteador.

3. Estado Jogador_as

Este estado é ativado quando o jogador possui um Ás e decide utilizá-lo como 11, desde que isso não o faça ultrapassar 21.

- As opções de HIT e STAY permanecem disponíveis.
- Transições:
  - O jogador pode avançar para o estado Carteador se optar por parar (STAY).
  - Se ultrapassar 21, o jogo avança diretamente para o estado Resultado.

4. Estado Carteador

Neste estado, o carteador joga automaticamente de acordo com as regras do Blackjack:

- Solicita cartas adicionais enquanto a soma for menor que 17.
- Caso ultrapasse 21, o jogo avança para o estado Resultado.
- Se o carteador possuir um Ás e puder usá-lo como 11, o estado muda para Carteador_AS.
- Quando atingir ou ultrapassar 17 sem estourar, o jogo avança para o estado Resultado.

5. Estado Carteador_as

Quando o carteador possui um Ás, ele pode utilizá-lo como 11, desde que isso não faça sua soma ultrapassar 21.

- O carteador continua jogando até alcançar ou ultrapassar 17.

Transição:

- Se a soma ultrapassar 21 ou atingir o limite de 17, o jogo avança para o estado Resultado.

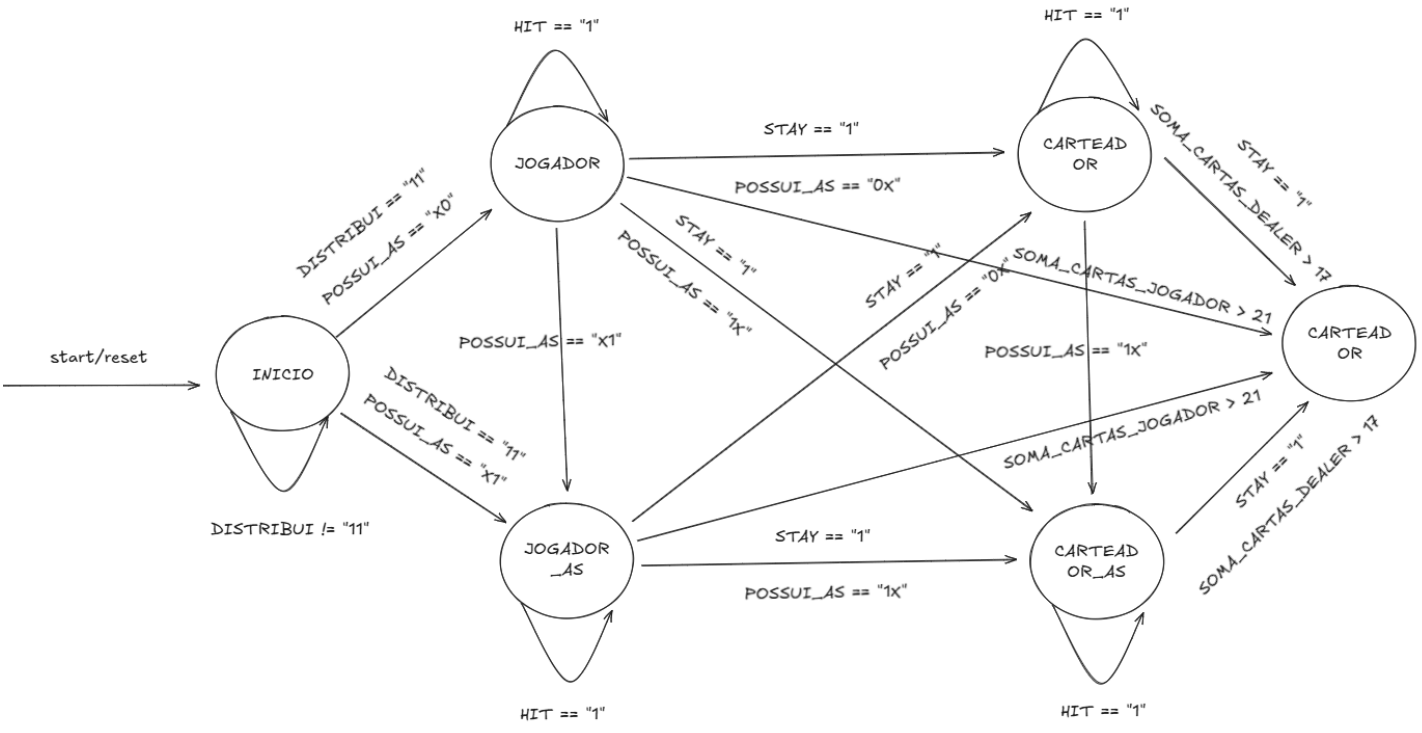
6. Estado Resultado

O jogo termina neste estado, onde o resultado é calculado:

- As somas das cartas do jogador e do carteador são comparadas para determinar o vencedor.
- LEDs indicam o resultado:
    - Vitória do jogador: LEDs correspondentes acendem.
    - Empate ou derrota: LEDs refletem a situação.

Transição:

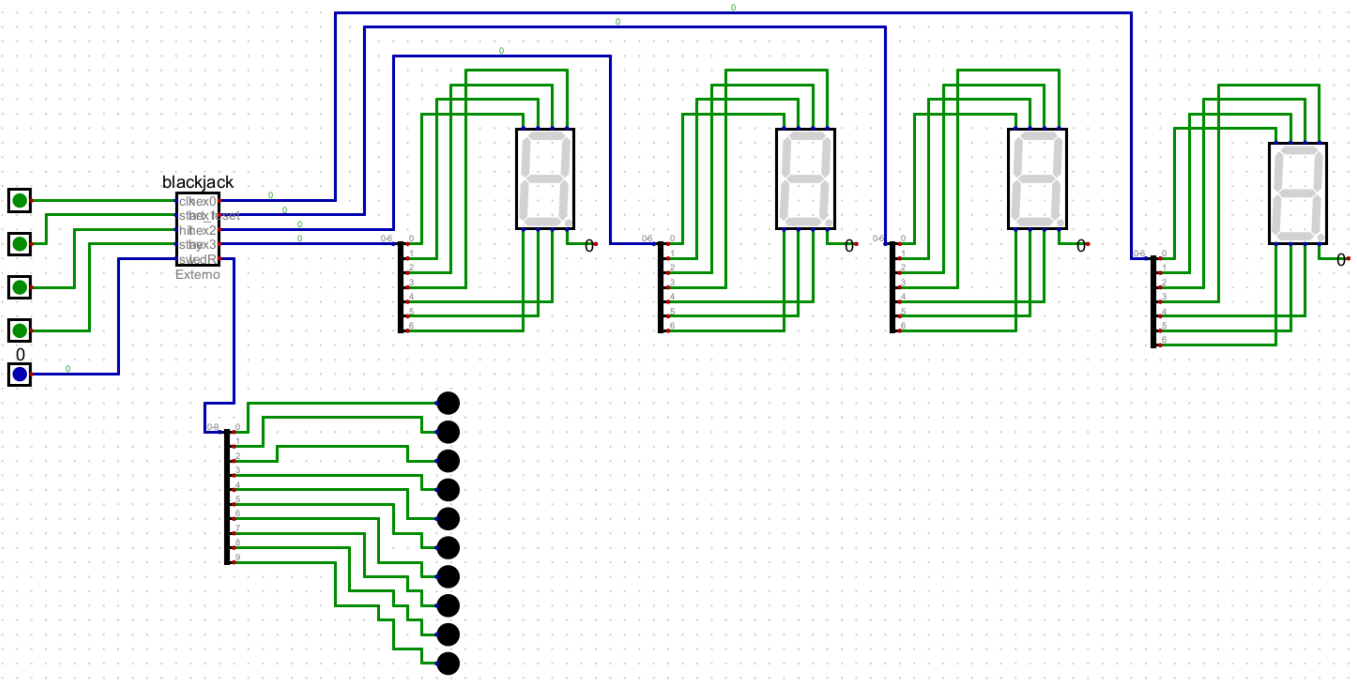
- O jogo pode ser reiniciado pressionando o botão de START/RESET



2.1 Tabela de Transições de Estado

Estado Atual	Condição	Próximo estado	Descrição
início	Cartas iniciais distribuídas	**jogador**	O jogador está pronto para tomar decisões (HIT ou STAY)
jogador	`sw(6) = '1'` (HIT)	**jogador**	O jogador solicita mais uma carta e continua no mesmo estado.
jogador	`sw(7) = '1'` (STAY)	**carteador**	O jogador decide não pegar mais cartas e passa a vez.
jogador	`soma_cartas_jogador>21`	**resultado**	O jogador ultrapassa 21 pontos e perde o jogo.
jogador	Possui As e decide usar como 11	**jogador_as**	O jogador utiliza o Ás como 11 para melhorar a soma.
jogador_as	`sw(6) = '1'` (HIT)	**jogador_as**	O jogador pega mais uma carta com o Ás considerado.
jogador_as	`sw(7) = '1'` (STAY)	**carteador**	O jogador decide parar e passa a vez para o carteador.
jogador_as	`soma_cartas_jogador>21`	**resultado**	O jogador ultrapassa 21 pontos, mesmo com o Ás como 11
carteador	`soma_cartas_carteador <17`	**carteador**	O carteador pega mais cartas automaticamente.
carteador	`soma_cartas_carteador >= 17`	**resultado**	O carteador atinge ou ultrapassa 17 pontos e para de jogar.
carteador	`soma_cartas_carteador > 21`	**resultado**	O carteador ultrapassa 21 pontos e perde o jogo.
carteador	`possui_as(1) = '1')` e decide usar como 11	**carteador_as**	O carteador utiliza o Ás como 11 para melhorar a soma.
carteador_as	`soma_cartas_carteador + 10 < 17`	**carteador_as**	O carteador decide parar, considerando o Ás como 11
resultado	Resultado calculado (vitória, empate ou derrota)	**inicio**	O jogo reinicia se o botão de reset for pressionado.

2.3 Testes e Aplicação no Digital



2.3 Código VHDL Mapeado para FPGA

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity blackjack is
    port (

        key : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- key(2) => START/RESET; key(3) => CLOCK;
        sw : IN STD_LOGIC_VECTOR(9 DOWNTO 0); -- -- sw(6) => HIT; sw(7) => STAY; sw(3), sw(2), sw(1), sw(0) =>
CARTA_CIRCUITO_EXTERNO;

        hex0 : out std_logic_vector(6 downto 0);   -- unidade soma cartas decimal
        hex1 : out std_logic_vector(6 downto 0);   -- dezena soma cartas decimal
        hex2 : out std_logic_vector(6 downto 0);   -- usado para escrever em alguns estados
        hex3 : out std_logic_vector(6 downto 0);   -- hexadecimal carta mão

        --ledR : out std_logic_vector(9 downto 0)   -- WIN (7,8,9); TIE (4,5); LOSE (0,1,2);
        ledg : out std_logic_vector(7 downto 0)
```

```
    );
end blackjack;

architecture gurizes of blackjack is



  type tipo_estado is (
        inicio,
        jogador,
        jogador_as,
        carteador,
        carteador_as,
        resultado
    );

    signal estado_atual : tipo_estado := inicio;

    signal carta_atual : std_logic_vector(3 downto 0) := "0000";
    signal carta_circuito_externo : std_logic_vector(3 downto 0) := "0000";

    signal soma_cartas_jogador : integer := 0;
    signal soma_cartas_carteador : integer := 0;

    signal possui_as : std_logic_vector(1 downto 0) := "00";

    signal distribui : std_logic_vector(1 downto 0) := "00";

    signal temp_hex0 : std_logic_vector(6 downto 0);
    signal temp_hex1 : std_logic_vector(6 downto 0);

    function conversao_hexadecimal(carta : std_logic_vector(3 downto 0)) return std_logic_vector is
    begin
        case carta is
            when "0000" => return "1000000"; -- 0
            when "0001" => return "1111001"; -- 1
            when "0010" => return "0100100"; -- 2
            when "0011" => return "0110000"; -- 3
            when "0100" => return "0011001"; -- 4
            when "0101" => return "0010010"; -- 5
            when "0110" => return "0000010"; -- 6
            when "0111" => return "1111000"; -- 7
            when "1000" => return "0000000"; -- 8
            when "1001" => return "0010000"; -- 9
            when "1010" => return "0001000"; -- A (10)
            when "1011" => return "0000011"; -- b (11)
            when "1100" => return "1000110"; -- c (12)
            when "1101" => return "0100001"; -- d (13)
            when others => return "1000000";
```

```
    end case;
end conversao_hexadecimal;



function conversao_unidade(valor : integer) return std_logic_vector is
begin
    case valor is
        when 0 => return "1000000";
        when 1 => return "1111001";
        when 2 => return "0100100";
        when 3 => return "0110000";
        when 4 => return "0011001";

        when 5 => return "0010010";
        when 6 => return "0000010";
        when 7 => return "1111000";
        when 8 => return "0000000";
        when 9 => return "0010000";
        when 10 => return "1000000";
        when 11 => return "1111001";
        when 12 => return "0100100";
        when 13 => return "0110000";
        when 14 => return "0011001";
        when 15 => return "0010010";
        when 16 => return "0000010";
        when 17 => return "1111000";
        when 18 => return "0000000";
        when 19 => return "0010000";
        when 20 => return "1000000";
        when 21 => return "1111001";
        when others => return "1000000";
    end case;
end function conversao_unidade;

function conversao_dezena(numero : integer) return std_logic_vector is
begin
    if numero >= 0 and numero < 10 then
        return "1000000"; -- 0
    elsif numero >= 10 and numero <= 19 then
        return "1111001"; -- 1
    elsif numero >=20 and numero <= 21 then
        return "0100100"; -- 2
    else
        return "1000000"; -- Default (0)
    end if;
end function conversao_dezena;
```

```
begin

    process(key(2), key(3))   -- key(2) => start/reset; key(3) => clock;
    begin
        if (key(2) = '0') then  -- start/reset
            soma_cartas_jogador <= 0;
            soma_cartas_carteador <= 0;
            distribui <= "00";
            possui_as <= "00";
            estado_atual <= inicio;
            hex0 <= "1101101";
            hex1 <= "0000111";
            hex2 <= "0110011";
            hex3 <= "0000111";

        elsif falling_edge(key(3)) then  -- lógica de transição de estados embutida no process do clock

            if ((distribui /= "01" and estado_atual = inicio) or ((estado_atual = jogador or estado_atual =
jogador_as) and sw(7) /= '1')) then
                hex3 <= conversao_hexadecimal(carta_circuito_externo);

            end if;

            if (estado_atual = inicio) then
                hex2 <= "1101111";
                if (distribui(0) = '0') then
                    if (carta_atual = "0001") then
                        possui_as(0) <= '1';
                    end if;
                    soma_cartas_jogador <= soma_cartas_jogador + to_integer(unsigned(carta_atual));
                    hex1 <= conversao_dezena(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
                    hex0 <= conversao_unidade(soma_cartas_jogador + to_integer(unsigned(carta_atual)));

                    distribui(0) <= '1';
                else
                    if (carta_atual = "0001") then
                        possui_as(1) <= '1';
                    end if;
                    soma_cartas_carteador <= soma_cartas_carteador + to_integer(unsigned(carta_atual));
                    distribui <= "10";
                end if;
                    if (distribui(0) = '1' AND distribui(1) = '1') then
                    estado_atual <= jogador;
                    end if;
            end if;

            if (estado_atual = jogador) then
                hex2 <= "1001111";
```

```
    if (sw(7) = '1') then  -- STAY
        if (possui_as(1) = '1') then
            estado_atual <= carteador_as;
        else
            estado_atual <= carteador;
        end if;
    elsif (sw(6) = '1') then  -- sw(6)
        soma_cartas_jogador <= soma_cartas_jogador + to_integer(unsigned(carta_atual));
        if (soma_cartas_jogador > 21) then
            estado_atual <= resultado;
        else
            hex1 <= conversao_dezena(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
            hex0 <= conversao_unidade(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
        end if;
    end if;
end if;

if (estado_atual = jogador_as) then
    if (sw(7) = '1') then  -- STAY
        if (possui_as(0) = '1' and soma_cartas_jogador + 10 < 22) then
            soma_cartas_jogador <= soma_cartas_jogador + 10;

            hex1 <= conversao_dezena(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
            hex0 <= conversao_unidade(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
        elsif (possui_as(1) = '1') then
            estado_atual <= carteador_as;

        else
            estado_atual <= carteador;
        end if;
    elsif (sw(6) = '1') then  -- HIT
        soma_cartas_jogador <= soma_cartas_jogador + to_integer(unsigned(carta_atual));
        if (soma_cartas_jogador > 21) then
            estado_atual <= resultado;
        else
            hex1 <= conversao_dezena(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
            hex0 <= conversao_unidade(soma_cartas_jogador + to_integer(unsigned(carta_atual)));
        end if;
    end if;
end if;

if (estado_atual = carteador) then
    hex2 <= "1110011";
    if (soma_cartas_carteador < 17) then
     0  soma_cartas_carteador <= soma_cartas_carteador + to_integer(unsigned(carta_atual));
        if (soma_cartas_carteador + to_integer(unsigned(carta_atual)) > 21) then
            estado_atual <= resultado;
        end if;
```

```
                if (carta_atual = "0001") then
                    estado_atual <= carteador_as;
                end if;
            else
                estado_atual <= resultado;
                hex2 <= "1001111";
            end if;
        end if;

        if (estado_atual = carteador_as) then
            if (soma_cartas_carteador + 10 < 17) then
                soma_cartas_carteador <= soma_cartas_carteador + to_integer(unsigned(carta_atual));
                if (soma_cartas_carteador + to_integer(unsigned(carta_atual)) > 21) then
                    estado_atual <= resultado;
                end if;
            else
                if (soma_cartas_carteador + 10 < 22) then
                    soma_cartas_carteador <= soma_cartas_carteador + 10;
                end if;
                estado_atual <= resultado;
                hex2 <= "1001111";
            end if;
        end if;

    end if;
end process;

process (estado_atual)
begin
    if (estado_atual = resultado) then
        -- Se o jogador perdeu (soma > 21 ou carta do jogador é menor que a do carteador, e carteador não
passou de 21)


        if (soma_cartas_jogador > 21 or (soma_cartas_jogador < soma_cartas_carteador and
soma_cartas_carteador < 22)) then
          --  ledR(0) <= '1';  -- Lose
          --  ledR(1) <= '1';
          --  ledR(2) <= '1';
            ledg(7) <= '1';
        -- Se o jogador ganhou (soma do carteador > 21 ou carta do jogador é maior que a do carteador)
        elsif (soma_cartas_carteador > 21 or soma_cartas_carteador < soma_cartas_jogador) then
          --  ledR(7) <= '1';  -- Win
          --  ledR(8) <= '1';
          --  ledR(9) <= '1';
              ledg(6) <= '1';
        -- Se for empate
        else
```

```
                ledg(7) <= '1';
                ledg(6) <= '1';
                -- ledR(4)<= '0';
               -- ledR(5) <= '1';
            end if;
        end if;
    end process;

    process (sw(3), sw(2), sw(1), sw(0))
    variable valor_carta : std_logic_vector(3 downto 0) := "0000";
    begin

        valor_carta(0) := sw(0);
        valor_carta(1) := sw(1);
        valor_carta(2) := sw(2);
        valor_carta(3) := sw(3);

        if (to_integer(unsigned(valor_carta)) > 10) then
            carta_atual <= "1010";
        else
            carta_atual <= valor_carta;
        end if;

        carta_circuito_externo <= sw(3 downto 0);
    end process;

end gurizes;
```

3. CONCLUSÃO

Neste projeto, desenvolvemos uma Máquina de Estados Finitos (FSM) para implementar o jogo de Blackjack em um ambiente digital, utilizando VHDL e a plataforma FPGA DE1-Cyclone II. Partimos do diagrama de estados, no qual definimos cuidadosamente as transições entre os estados principais: START, JOGADOR, JOGADOR_AS, CARTEADOR, CARTEADOR_AS e RESULTADO.

A elaboração da tabela de transição foi essencial para detalhar as condições que direcionaram cada mudança de estado, assegurando que o sistema simulasse as regras do Blackjack de forma precisa. Por meio das simulações e testes práticos, verificamos que o funcionamento geral do sistema atendeu às expectativas, permitindo uma jogabilidade consistente e alinhada às regras oficiais do jogo.

Este projeto representou uma oportunidade valiosa de aplicar conceitos teóricos de máquinas de estados e lógica digital a um problema real. Além disso, superamos desafios relacionados à implementação e integração do sistema, consolidando habilidades técnicas e reforçando nossa capacidade de solucionar problemas. O aprendizado adquirido estabelece uma base sólida para enfrentar projetos futuros, nos preparando para desenvolver sistemas ainda mais complexos e interativos.