UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CIÊNCIA DA COMPUTAÇÃO

Marco Aurélio Lunardi
Murilo Gemi de Carli
Quilian Augusto Moretti

Disciplina:Sistemas Digitais

Batalha Naval

Chapecó

12 de dezembro de 2023

1. APRESENTAÇÃO.

O projeto de batalha naval proposto na matéria de Sistemas Digitais do terceiro semestre de Ciência da Computação foi realizado usando os seguintes softwares: Visual Studio Code, para descrição do sistema em vhdl; Lucidspark, para representação do diagrama de estados da máquina de estados finita; e Digital, para a simulação do jogo.

2. DESCRIÇÃO DA SOLUÇÃO:

2.1 Codificações

Na codificação das posições da matriz 4x4, os valores originais das posições (Figura 1) foram embaralhados, e a partir disso, teve origem uma nova matriz 4x4(Figura1.1), selecionada no arquivo disponibilizado pelo professor(foi escolhido a codificação 13).

Linhas				
00	0000	0001	0010	0011
01	0100	0101	0110	0111
10	1000	1001	1010	1011
11	1100	1101	1110	1111
Colunas	00	01	10	11

Figura 1.

No final, cada posição faz “referência” a outra posição diferente da tabela original. Por exemplo: na Figura 1, a posição 0101, recebe na Figura 1.1 a “posição codificada” 0111, não representando mais o valor anterior.

Linhas				
00	0101	0000	1111	1110
01	1010	0111	0100	1000
10	1100	0110	0010	1011
11	0001	1101	1001	0011
Colunas	00	01	10	11

Figura 1.1.

2.2 Descrição VHDL

A estratégia adotada para codificação da matrix 4x4 foi a seguinte:
Primeiramente foi definido a entidade batalha com suas entradas e saídas:

```
entity batalha is
   port(
       key0, key1, key2 : in std_logic;
       sw0, sw1, sw2, sw3, sw4 : in std_logic;
       ledstart,ledacerto, ledwin, ledlose : out std_logic;
       hex0, hex1, hex2, hex3, hex4, hex5, hex6 : out std_logic
   );
   end batalha;
```

key0: É o botão de reset, responsável por redefinir o jogo ao seu início independentemente em qual fase esteja.

key1: É o botão de mudança de estado, responsável por migrar para as próximas etapas do jogo.

key2: Botão responsável por setar os navios em suas posições codificadas e também o botão de disparo quando no estado certo.

sw0, sw1, sw2, sw3, sw4: Botões de seleção das posições do navio e também da posição de disparo.

ledstart: Led que acende quando o jogo está no estado de disparo.

ledacerto: Led que acende caso o disparo do jogador acerte um navio.

ledwin: Led que acende caso o jogador acerte a posição dos 3 navios.

ledlose: Led que acende caso o jogador perca o jogo.

hex0 - hex6: Saídas do display de 7 segmentos que informam ao usuário os disparos restantes.

Depois, foi descrito o comportamento do sistema:

```
     architecture Behavior of batalha is
        type Tipo_estado is (inicio, navio1set, navio2set, start);
        signal estado_atual, prox_estado: Tipo_estado;
        signal navio1, navio2a, navio2b, posicao_disparo, novo_jogo
: std_logic_vector(0 to 3);
        begin
```

Tipo_estado: são os estados da FSM do jogo

estado_atual e **prox_estado**: Signals que fazem a transição dos estados da FSM.

navio1, **navio2a**, **navio2b**: signals que armazenam um vetor de std_logic com as posições codificadas dos navios;

posição_disparo: signal que armazena um vetor de std_logic com a posição de disparo informada pelo jogador.

Após isso, terá início o primeiro process:

```
process (key1, key2, estado_atual)
        variable disparos : integer;
        variable chance : integer;
        variable acertos : integer;
    begin
```

disparos: variável que armazena o número de disparos

chance: variável que armazena 1 caso o jogador acerte o seu próximo disparo(útil caso seja o último disparo e o jogador acerte, tendo assim mais chances).

acertos: variável que armazena o total de acertos do jogador.

Neste processo ocorre os estados da FSM, começando pelo estado “início”:

```
case estado_atual is
    when inicio =>
            navio1 <= novo_jogo;
            navio2a <= novo_jogo;
            navio2b <= novo_jogo;
            posicao_disparo <= novo_jogo;
            disparos := 6;
            chance := 0;
            acertos := 0;
            ledwin <= '0';
            ledlose <= '0';
            ledacerto <= '0';
            prox_estado <= navio1set;
```

Este estado basicamente redefine todos os leds para 0 e os navios para um novo jogo, para caso ocorra um reset.

O estado seguinte é o de codificação do navio1:

```
when navio1set =>
        if key2'event then
            navio1(0) <= sw0;
            navio1(1) <= sw1;
            navio1(2) <= sw2;
            navio1(3) <= sw3;
            if navio1 = "0000" then
                navio1 <= "0001";
            elsif navio1 = "0001" then
                navio1 <= "1100" e assim por diante…
```

Ao pressionar key2, o vetor navio1 armazena os valores de sw0, sw1, sw2 e sw3, e logo em seguida o atualiza de acordo com a codificação da matriz.

O Estado seguinte é o de codificação do navio2:

```
when navio2set =>
               if key2'event then
                   navio2a(0) <= sw0;
                   navio2a(1) <= sw1;
                   navio2a(2) <= sw2;
                   navio2a(3) <= sw3;
                   if navio2a(0) = '0' and navio2a(1) = '0' and
navio2a(2) = '0' and navio2a(3) = '0' then
                       navio2a <= "0001";
                        if sw4 = '0' then
                           navio2b <= "1111";
                       else
                           navio2b <= "0111";
                       end if;
                  elsif navio2a(0) = '0' and navio2a(1) = '0' and
navio2a(2) = '0' and navio2a(3) = '1' then
                       navio2a <= "1100";
                       if sw4 = '0' then
                           navio2b <= "1101";
                       else
                           navio2b <= "1100";
                       end if;           E assim por diante...
```

Ele segue o mesmo esquema de codificação do navio1, com a adição da entrada sw4, que define se o navio2 estará na posição vertical(sw4 = 1) ou horizontal (sw4 = 0). O sistema também checa se o navio está em alguma posição que não possa se adicionar a segunda parte à direita ou abaixo. Se esse for o caso, a devida alocação é feita à esquerda ou acima.

O próximo e último estado é o de disparo:

```
when start =>
           hex0 <= '1';
           hex2 <= '1';
           hex3 <= '1';
           hex4 <= '1';
```

```
            hex5 <= '1';
            hex6 <= '1';
             if key2'event and key2 = '1' then
                disparos := disparos -1;
                posicao_disparo(0) <= sw0;
                posicao_disparo(1) <= sw1;
                posicao_disparo(2) <= sw2;
                posicao_disparo(3) <= sw3;
```

Primeiramente, o estado define o display para mostrar o número 6, que é o total de disparos que o jogador tem à disposição. Após isso, ao clicar em key2, o signal posicao_disparo armazena a posição de tiro do jogador.

```
elsif key2'event and key2 = '0' then
              if (posicao_disparo = navio1) or
                  (posicao_disparo = navio2a) or
                  (posicao_disparo = navio2b) then
                  acertos := acertos + 1;
                  ledacerto <= '1';
                  chance := 1;
              else
                  chance := 0;
                  ledacerto <= '0';
              end if;
```

Quando key2 é liberado, ocorre a verificação do disparo do jogador. Caso a posição de disparo combine com alguma posição do navio, o contador de acertos é incrementado, o led de acerto liga e a chance é setada para 1. Caso contrário, a chance é setada para 0 e o led de acerto é desligado.

```
if disparos = 0 then
                  if chance = 1 then
                      disparos:= disparos +1;
                  else
                      ledlose <= '1';
                      hex0 <= '1';
                      hex1 <= '1';
                      hex2 <= '1';
                      hex4 <= '1';
                      hex3 <= '1';
                      hex5 <= '1';
```

```
                    hex6 <= '0';
                end if;
            end if;
            if acertos = 3 then
                ledwin <= '1';
            end if;
```

Nesta mesma borda de descida do key2, ocorre a verificação do número de disparos, e a verificação de vitória ou derrota. Caso a variável disparos chegue a 0, mas o jogador teve êxito em seu último disparo(chance =1), um novo disparo será concedido. Caso contrário, o led de derrota irá ligar indicando que o jogador perdeu.

```
        if disparos = 5 then
            hex0 <= '1';
            hex1 <= '0';
            hex2 <= '1';
            hex4 <= '0';
            hex3 <= '1';
            hex5 <= '1';
            hex6 <= '1';
        elsif disparos = 4 then    e segue para os demais...
```

Ocorre também a atualização do display de 7 segmentos, mostrando a quantidade de disparos restantes.

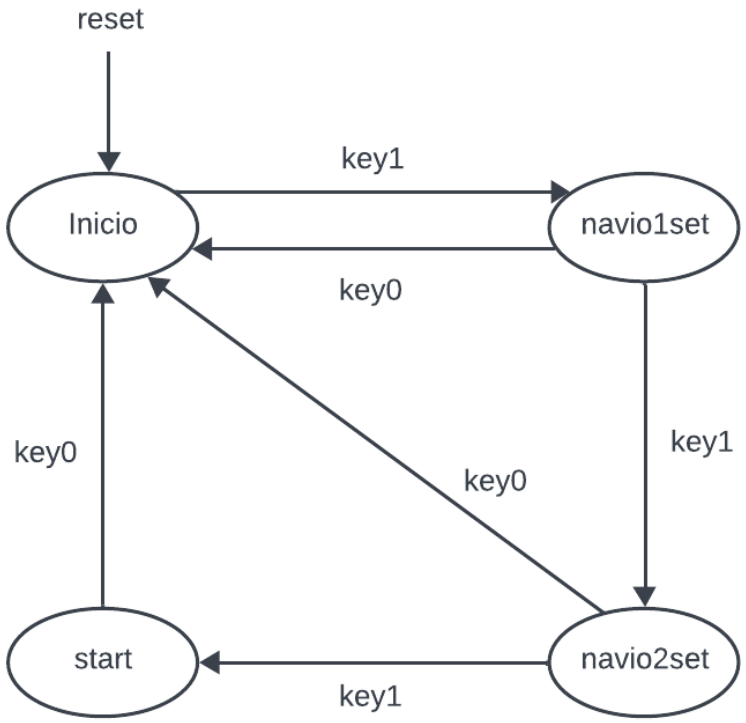
O outro processo do sistema basicamente controla a mudança de estados:

```
process (key1, key0)
   begin
       if key0 = '1' then
           estado_atual <= inicio;
       elsif (key1'event and key1 = '1') then
           estado_atual <= prox_estado;
       end if;
   end process;
```

Se key0 for pressionado, a sistema volta para o início, e se key1 for pressionado, ocorre a mudança de estados.

2.3 Diagrama de estados

key1 = proximo estado
key0 = reset



3. CONCLUSÃO

O trabalho de desenvolver o jogo batalha naval colocou em prática grande parte do conteúdo aprendido durante o semestre de Sistemas Digitais. O objetivo final do trabalho, entretanto, era passar o código VHDL para a FPGA para simulação física do jogo. Porém, pela falta de tempo hábil e também pela grande quantidade de erros nesta tentativa, esta parte do trabalho acabou por não ser feita. Este foi o erro encontrado na hora de passar o código para a FPGA:

couldnt implement register for assigments on this clock edge

Apesar disso, o uso de máquinas de estados e a codificação em VHDL foi bastante positivo para o entendimento do conteúdo, visto que na simulação digital o jogo funciona perfeitamente.