```
--refinar algoritmo para entrega parcial -15/10

-- Projeto de Sistemas Digitais:
-- Grupo: Arthur Pellenz Winck (19201620), Gabriel Fornaza da Silveira(19201618)

--DESCRIÇÃO:
--O jogo simula uma partida de blackjack ou vinte-e-um. Nele, se o jogador fica com mais de 21 pontos ele perde e se
--somar 17 pontos ele nao pode pegar mais cartas. O jogador deve tentar alcancar o maior numero de pontos sem ultrapassar o limite. Os pontos vem dos numeros das cartas podendo variar entre 1 e 13, sendo que
--o baralho é composto por 52 cartas (1,1,1,1,2,2,2,2...) e esta armazenado na memoria ram. A carta que sera retirada a cada rodada e aleatoria e sera recebida do testbench. Quando a carta for somada aos pontos do jogador, ela sera retirada
--do baralho. No fim do algoritmo (quando o jogador perde ou desiste de pegar cartas) a pontuacao e retornada. Se a pontuacao e zerada quando passa-se de 20 pontos
--
--
-- ALGORITMO:
-- inicio do algoritmo (Baixamos o nível do algoritmo para se basear mais em C para simplificar a criação da FSMD)
        --baralho = [1,2,3,4,5,6,7,8,9,10,11,12,13]*4
        --int tam = 52
        --int pontos = 0
        --int pos = 0
        --addCarta = 0
        --
        -- while True:
            --      addCarta = input ();
            --      if (addCarta == 1){
            --      //Recebe a carta aleatória do Testbench
            --      uint carta = input ();
```

```
        --
        --
        --      //Remove a carta correspondente do baralho; se a carta existe,
executa a retirada da carta do baralho, a soma de pontos e a checagem da
condição de vitória
        --      for (i=0; i<tam-1; i++){
        --          if  (baralho[i] == carta){
        --              pos = i
        --
        --              pontos = pontos + carta
        --              for(i=pos; i<tam-1; i++){
        --                  baralho[i] = baralho[i + 1];
        --              };
        --              tam --;
        --
        --              if pontos > 20{
        --                  pontos = 0;
        --                  addCarta = 0;
        --                  break;
        --              };
        --
        --              if pontos > 16{
        --                  addCarta = 0
        --                  break;
        --              };
        --              };
        --      else{
        --          break
        --              }
        --
        --output(pontos)

-- fim do algoritmo

-- VARIAVEIS:
```

```
        -- bool addCarta
        -- uint tam
        -- uint i
        -- uint carta
        -- uint pontos
        -- uint pos
        -- array baralho

-- OPERACOES E COMANDOS/STATUS EQUIVALENTES (==>) , ordenado por variavel tipo do sinal (comando ou status) e depois por variavel
--      COMANDOS:
--      addCarta = input();          ==>cmdAddCarta
--      carta = input();             ==> cmdSetCarta
--      tam = tam --;                ==> cmdSetTam, cmdResetTam
--      i = i ++;                    ==> cmdSetI, cmdResetI
--      pos = carta - 1              ==> cmdSetPos, cmdResetPos
--      baralho[i] = baralho[i+1] ==> cmdUpdateBaralho, cmdResetBaralho

--      STATUS:
--      addCarta == 1 ==> sttAddCartaIgualUm
--      i < tam - 1       ==> sttIMenorTam
--      pontos > 20     ==> sttPontosMaiorVinte
--      pontos > 16     ==> sttPontosMaiorDezesseis

--FSMD DO BLOCO DE CONTROLE (a ser refinado apos o projeto do Bloco Operativo)
-- inicio do algoritmo (Baixamos o nível do algoritmo para se basear mais em C para simplificar a criação da FSMD)
        --SL00: inicio do algoritmo
        --SL01: baralho = [1,2,3,4,5,6,7,8,9,10,11,12,13]*4
        --SL02: int tam = 52
        --SL03: int pontos = 0
        --SL04: int pos = 0
        --SL05: addCarta = 0
        --
```

```
--SL06: while True:
    --SL07:    addCarta = input ();
    --SL08:    if (addCarta == 1){
    --SL09     uint carta = input ();
    --      //Remove a carta correspondente do baralho; se a carta existe, executa a retirada da carta do baralho, a soma de pontos e a checagem da condição de vitória
    --SL10     for (i=0; i<tam-1; i++){
    --SL11        if  (baralho[i] == carta){
    --SL12            pos = i
    --SL13            }
    --SL14            pontos = pontos + carta
    --SL15            for(i=pos; i<tam-1; i++){
    --SL16               baralho[i] = baralho[i + 1];
    --SL17            };
    --SL18            tam --;
    --
    --SL19            if pontos > 20{
    --SL20               pontos = 0;
    --SL21               addCarta = 0;
    --SL22               break;
    --SL23            };
    --
    --SL24            if pontos > 16{
    --SL25               addCarta = 0
    --SL26               break;
    --SL27            };
    --SL28            };
    --SL29     else{
    --SL30        break
    --SL31            }
    --
    --SL32 output(pontos)
    --SL33 fim do algoritmo
```

```
-- BLOCO OPERATIVO - Circuitos COMBINACIONAIS para implementaçao
das operaçoes
--   Comparador Menor: i < tam -1                          ==> sttIMenorTam
--   Incrementador:         i++                          ==> cmdSetI
--   Somador:               pontos + carta               ==> cmdSomaPontos
--   Decrementador:         tam--                        ==> cmdSubTam
--   Comparador Maior: pontos > 16                       ==>
sttPontosMaiorDezesseis
--   Comparador Maior: pontos > 20                       ==>
sttPontosMaiorVinte
--   Decrementador:     carta - 1                        ==> cmdSubCarta
--   Comparador:  baralho[i] == carta                    ==> sttCompBaralho
--   Multiplexador: Reset para o baralho inicial;  baralho =
[1,2,3,4,5,6,7,8,9,10,11,12,13]*4  => cmdResetBaralho
--   Multiplexador: Reset parta o tam inicial; tam = 52        ==> cmdResetTam
--   Multiplexador: i = 0 ou i = pos                     ==> cmdMultI
--   Comparador: addCarta == 1                           ==> sttCompAddCarta

--  sinal para ir para o ultimo estado: break ==> cmdAcabaJogo

-- BLOCO OPERATIVO - Circuitos SEQUENCIAIS para implementaçao das
operaçoes
-- Memoria Ram ;  baralho[i] = baralho[i+1]  ==> cmdSetBaralho
-- Registrador (carga):     pos = carta - 1             ==> cmdSetPos
-- Registrador (reset):     pos = 0             == cmdResetPos
-- Registrador (carga):  carta = input()    ==> cmdSetCarta
-- Registrador (carga):  tam = input()      ==> cmdSetTam
-- Registrador (reset) pontos = 0; - cmdResetPontos
--Registrador (carga) pontos = pontos + carta - cmdSetPontos
-- Registrador (reset): i = 0   ==> cmdResetI
--Registrador (carga): i = 0    ==> cmdSetI
--Registrador (carga): addCarta = input() ==> cmdSetAddCarta
--Registrador(reset): addCarta = 0;         ==> cmdResetAddCarta
```

```
-- DIAGRAMA DE TRANSICAO DE ESTADOS DO BLOCO DE CONTROLE
        --SL00: inicio do algoritmo
        --SL01: baralho = [1,2,3,4,5,6,7,8,9,10,11,12,13]*4
        --SL02: int tam = 52
        --SL03: int pontos = 0
        --SL04: int pos = 0
        --SL05: addCarta = 0
        --
        --SL06: while True:     --- SL07
            -- //Recebe 1 ou 0 para continuar o jogo ou parar
            --SL07:     addCarta = input ();                    ----    input_read/SL07a
            --SL07a:    //Recebe addCarta                       ---- SL08
            --SL08:     if (addCarta == 1){
----cmdCompAddCarta/SL09    !cmdCompAddCarta/SL30
            --SL09      uint carta = input ();                  ---- input_read/SL09a
            --SL09a     //Recebe carta                          ---- SL10
            --      //Remove a carta correspondente do baralho; se a carta existe,
executa a retirada da carta do baralho, a soma de pontos e a checagem da
condição de vitória
            --SL10      for (i=0; i<tam-1; i++){              ----sttIMenorTam/SL11
!sttIMenorTam/SL14
            --SL11          if  (baralho[i] == carta){        ----- cmdCompBaralho/SL12
!cmdCompBaralho/SL10
            --SL12              pos = i                       -----SL13
            --SL13              };                            -----SL10
            --SL14              pontos = pontos + carta     ----SL15
            --SL15              for(i=pos; i<tam-1; i++){   ----- sttIMenorTam/SL16
!sttIMenorTam/SL18
            --SL16                  baralho[i] = baralho[i + 1];    --SL17
            --SL17              };                            ----SL15
            --SL18              tam --;                       ----SL19
            --
            --SL19              if pontos > 20{
----sttPontosMaiorVinte/SL20 !sttPontosMaiorVinte/SL23
```

```
          --SL20                pontos = 0;                  ----SL21
          --SL21                addCarta = 0;                 -----SL22
          --SL22                break;                       ----- cmdAcabaJogo =1
==> SL32
          --SL23            };                               ----SL24
          --
          --SL24            if pontos > 16{
----sttPontosMaiorDezesseis/SL25 !sttPontosMaiorDezesseis/SL27
          --SL25                addCarta = 0                 ----SL26
          --SL26                break;                       ----- cmdAcabaJogo =1
==> SL32
          --SL27            };                               ----SL28
          --SL28            };                               ----SL31
          --SL29      else {                                 ----SL30
          --SL30          break                ----- cmdAcabaJogo =1  ==> SL32
          --SL31            }                  ----- SL06
          --
          --SL32 output(pontos)                    ----SL32a
          --SL32a //Receber saida
          --SL33 fim do algoritmo                  ----SL00




-- DIAGRAMA DE SAIDAS DO BLOCO DE CONTROLE

        --SL00: inicio do algoritmo
        --SL01: baralho = [1,2,3,4,5,6,7,8,9,10,11,12,13]*4 ----cmdResetBaralho
        --SL02: int tam = 52        ----cmdResetTam
        --SL03: int pontos = 0      ----cmdResetPontos
        --SL04: int pos = 0          ----cmdResetPos
        --SL05: addCarta = 0        ----cmdResetAddCarta
        --
        --SL06: while True:     --- SL07
            -- //Recebe 1 ou 0 para continuar o jogo ou parar
```

```
        --SL07:     addCarta = input ();           ----interrupt
        --SL07a:                                   ----cmdSetAddCarta
        --SL08:     if (addCarta == 1){
        --SL09      uint carta = input ();      ----interrupt
        --SL09a     //Recebe carta              ----cmdSetCarta
        --      //Remove a carta correspondente do baralho; se a carta existe,
executa a retirada da carta do baralho, a soma de pontos e a checagem da
condição de vitória
        --SL10      for (i=0; i<tam-1; i++){
        --SL11          if  (baralho[i] == carta){
        --SL12              pos = i                 ----cmdSetPos
        --SL13              };
        --SL14              pontos = pontos + carta         ----cmdSetPontos
        --SL15              for(i=pos; i<tam-1; i++){
        --SL16                  baralho[i] = baralho[i + 1];    ----cmdSetBaralho
        --SL17              };
        --SL18              tam --;                         ----cmdSetTam
        --
        --SL19              if pontos > 20{               -------sttPontosMaiorVinte
        --SL20                  pontos = 0;               ----cmdResetPontos
        --SL21                  addCarta = 0;           ----cmdResetAddCarta
        --SL22                  break;
        --SL23              };
        --
        --SL24              if pontos > 16{           ----sttPontosMaiorDezesseis
        --SL25                  addCarta = 0           ----cmdResetAddCarta
        --SL26                  break;
        --SL27              };
        --SL28              };
        --SL29      if (addCarta == 0){
        --SL30          break
        --SL31              }
        --
        --SL32 output(pontos)              ----interrupt
        --SL32a: //Receber saida
```

```
        --SL33 fim do algoritmo

entity Blackjack is
    generic(
        dataWidth: positive := 8;
        addressWidth: positive := 6
    );
    port(
        -- control inputs
        clk: in std_logic;
        reset_req: in std_logic;
        chipselect: in std_logic;
        readd: in std_logic;
        writee: in std_logic;
        -- data inputs
        address: in std_logic_vector(addressWidth-1 downto 0);
        writedata: in std_logic_vector(dataWidth-1 downto 0);
        -- control outputs
        interrupt: out std_logic;
        -- data outputs
        readdata: out std_logic_vector(dataWidth-1 downto 0);
        -- tests
        --testRegI, testVetorI, testMaxdado: out std_logic_vector(dataWidth-1
downto 0);
        testEstadoAtual: out Estado
    );
end entity;

architecture structural of Blackjack is

    component bloco_controle_blackjack is
        port(
            clk: in std_logic;
            reset_req: in std_logic;
            chipselect: in std_logic;
```

```
        readd: in std_logic;
        writee: in std_logic;
        -- status from OperativeBlock
        sttPontosMaiorDezesseis, sttPontosMaiorVinte, sttCompAddCarta,
sttMenorTam, sttCompBaralho: in std_logic;
        -- control outputs
        interrupt: out std_logic;
        --commands to OperativeBlock
        cmdSetI, cmdResetI, cmdSomaPontos, cmdSubTam, cmdSubCarta,
cmdResetBaralho, cmdSetBaralho cmdResetTam,
            cmdSetTam, cmdMultIBaralho, cmdMultTam,
cmdSetPontos,cmdResetPontos, cmdSetAddCarta, cmdResetAddCarta,
            cmdSetCarta,cmdResetCarta,cmdResetPos, cmdSetPos: out
std_logic;
        --tests
        testEstadoAtual out Estado
    );
  end component;

  component bloco_operativo_blackjack is
    generic(
        dataWidth: positive := 8;
        addressWidth: positive := 8
    );
    port(
        -- control inputs
        clk     : in std_logic;
        reset_req: in std_logic;
        -- data inputs
        address : IN STD_LOGIC_VECTOR (addressWidth-1 DOWNTO 0);
        writedata: IN STD_LOGIC_VECTOR (dataWidth-1 DOWNTO 0);
        -- data outputs
        readdata    : OUT STD_LOGIC_VECTOR (dataWidth-1 DOWNTO 0);
        -- commands from OperativeBlock
```

```
        signal cmdSetI, cmdResetI, cmdSomaPontos, cmdSubTam,
cmdSubCarta, cmdResetBaralho, cmdSetBaralho cmdResetTam,
            cmdSetTam, cmdMultIBaralho, cmdMultTam,
cmdSetPontos,cmdResetPontos, cmdSetAddCarta, cmdResetAddCarta,
            cmdSetCarta,cmdResetCarta,cmdResetPos, cmdSetPos: in
std_logic;

        --status to OperativeBlock
        sttPontosMaiorDezesseis, sttPontosMaiorVinte, sttCompAddCarta,
sttMenorTam, sttCompBaralho: out std_logic
    );
  end component;

  --comandos
  signal cmdSetI, cmdResetI, cmdSomaPontos, cmdSubTam, cmdSubCarta,
cmdResetBaralho, cmdSetBaralho cmdResetTam,
  cmdSetTam, cmdMultIBaralho, cmdMultTam,
cmdSetPontos,cmdResetPontos, cmdSetAddCarta, cmdResetAddCarta,
  cmdSetCarta,cmdResetCarta,cmdResetPos, cmdSetPos: in std_logic;

  --status
  signal sttPontosMaiorDezesseis, sttPontosMaiorVinte, sttCompAddCarta,
sttMenorTam, sttCompBaralho: out std_logic;

begin
  blocoControle: bloco_controle_blackjack
    port map(clk, reset_req, chipselect, readd, writee,
    sttPontosMaiorDezesseis, sttPontosMaiorVinte, sttCompAddCarta,
sttMenorTam, sttCompBaralho,
    interrupt,
    cmdSetI, cmdResetI, cmdSomaPontos, cmdSubTam, cmdSubCarta,
cmdResetBaralho, cmdSetBaralho cmdResetTam,
    cmdSetTam, cmdMultIBaralho, cmdMultTam,
cmdSetPontos,cmdResetPontos, cmdSetAddCarta, cmdResetAddCarta,
    cmdSetCarta,cmdResetCarta,cmdResetPos, cmdSetPos,
```

```
        testEstadoAtual);

    blocoOpertativo: bloco_operativo_blackjack
        generic map (dataWidth, addressWidth);
        port map (clk, reset_req,
        address, writedata,
        readdata,
        cmdSetI, cmdResetI, cmdSomaPontos, cmdSubTam, cmdSubCarta,
cmdResetBaralho, cmdSetBaralho cmdResetTam,
        cmdSetTam, cmdMultIBaralho, cmdMultTam,
cmdSetPontos,cmdResetPontos, cmdSetAddCarta, cmdResetAddCarta,
        cmdSetCarta,cmdResetCarta,cmdResetPos, cmdSetPos,
        sttPontosMaiorDezesseis, sttPontosMaiorVinte, sttCompAddCarta,
sttMenorTam, sttCompBaralho,
        );

end architecture;
```