Departamento de Engenharia Eletrônica - Universidade Federal de Minas Gerais

Trabalho Final-Entrega 1 : Laboratório de Sistemas Digitais

Eduardo Carvalho Biagini de Mello (2022055572), Gabriel Maia Pereira(2023422625)

Glaucus Miranda de Almeida (2021032986)

Máquina de Vendas

Belo Horizonte, Minas Gerais, Brasil; Agosto de 2024

1. INTRODUÇÃO

1.1 Descrição do problema e da solução implementada

O projeto consiste em uma máquina de vendas, máquina comum em locais mais bem frequentados, principalmente fora do país, sendo uma máquina que se baseia no autoatendimento, onde o cliente escolhe um produto, paga o preço do produto e o recebe. Infelizmente, hoje em dia ainda existem problemas com o êxito dessas máquinas de venda, um desses problemas que é o que procuramos resolver é a máquina ter problemas para dar troco e o produto desejado.

A maneira que solucionamos é limitando a quantidade de produtos a 7(um produto para cada cédula existente do real), cada produto valeria o valor exato de uma das cédulas, o local por onde se coloca as cédulas se fecharia após receber a primeira cédula. Vale ressaltar que o preço de cada produto estaria disponível para consulta do cliente.

Após o cliente inserir a nota, a máquina pediria confirmação de que não houve engano por parte do cliente. No caso de o cliente recusar a compra, a nota será devolvida e a máquina volta ao seu estado inicial. Em caso de aprovação do cliente, um comparador irá comparar o valor da nota inserida com os valores dos produtos e liberará o produto correspondente. Feito isso, o produto é retirado e a máquina volta ao seu estado inicial.

1.2 Diagrama de blocos

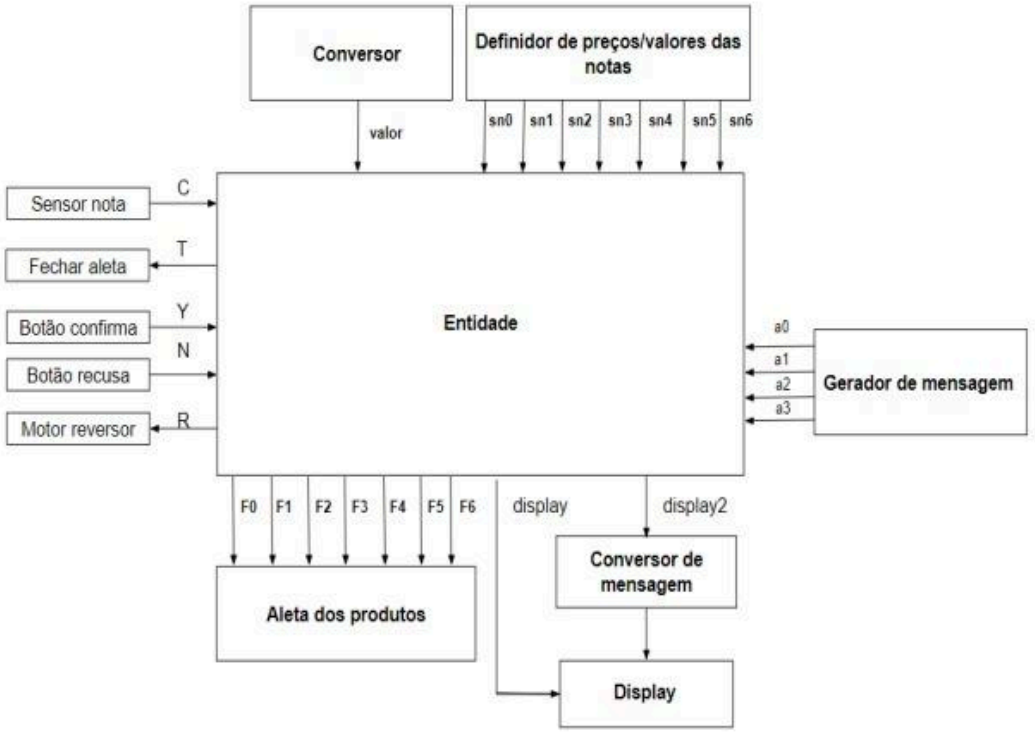
A partir do diagrama de blocos abaixo o caminho percorrido pode ser representado de forma mais elucidativa da seguinte maneira:

1. A cédula é inserida na máquina onde fará o “sensor nota” enviar o sinal C, o qual indica que uma nota foi inserida. Além disso, ao mesmo tempo um “conversor” analisa a nota depositada e converte seu valor para uma palavra de 16 bits, a qual é enviada como “valor” para FSM.

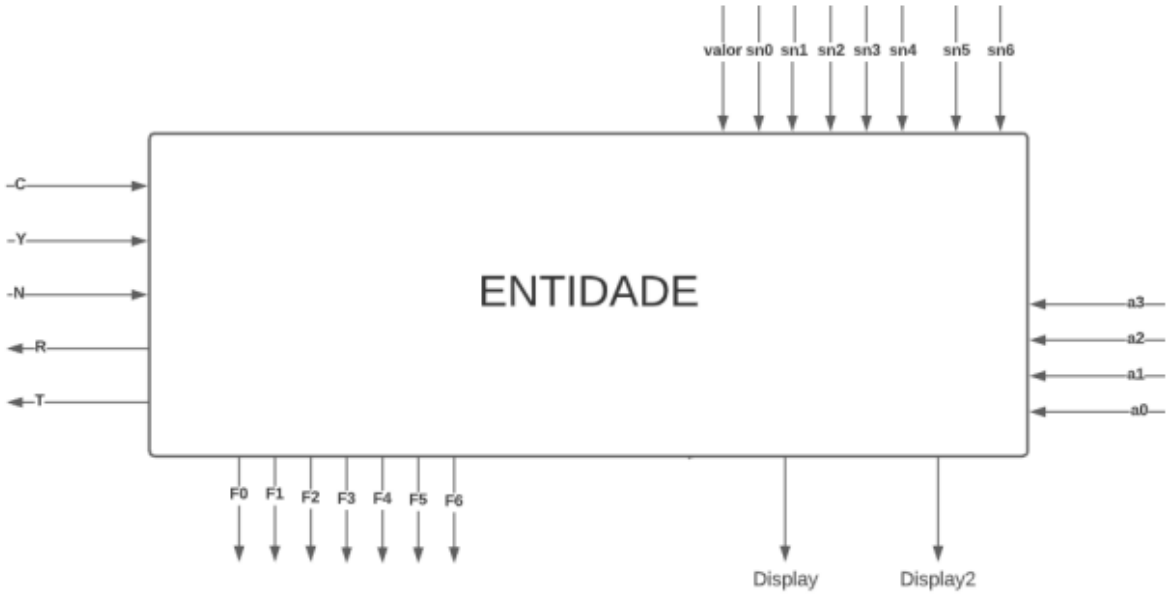
2. Após isso uma mensagem será apresentada para o cliente, perguntando se ele deseja confirmar a operação, caso ele não confirme, a FSM acionará o sinal R, onde fará com que o “motor reversor” devolva a cédula ao cliente. Além disso, nesse momento a FSM aciona o sinal T, que tranca a aleta para que novas cédulas não sejam inseridas.

3. Caso ele confirme, a FSM passa para o estado seguinte, onde jogará o valor da cédula depositada em um display. Após isso, será realizada uma comparação com os valores das cédulas oriundas do “Definidor de preços/valores das nota”, onde apenas uma comparação será verdadeira

4. A comparação verdadeira acionará a saída F (F0 a F6) correspondente,
onde possui o objetivo de acionar a “aleta de produtos” e entregar o produto
ao cliente. Após isso,a FSM volta para o estado “inicio” e fica a espera de novos sinais do
“sensor nota”

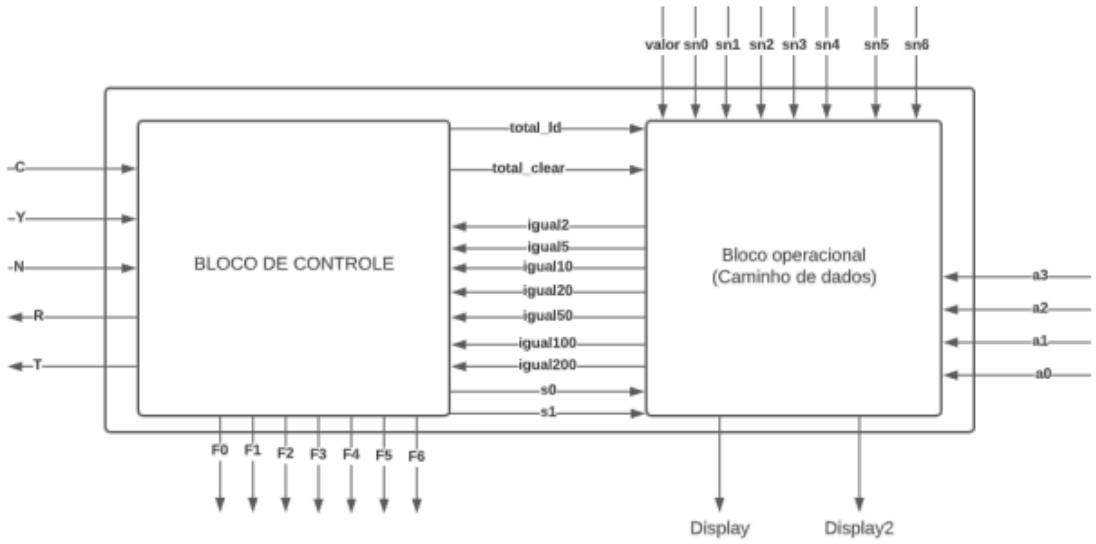


Representações da entidade em diferentes níveis de abstração e suas respectivas tabelas de entradas e saídas



Entradas	Bits	Funcionalidades
C	1	Indica que uma cédula foi inserida
Y	1	(Yes) botão de confirmar foi acionado
N	1	(No) botão de recusar foi acionado
sn0 a sn6	8	Valores das cédulas de R$2 a R$200
a0	8	Comando para gerar mensagem para inserir nota
a1	8	Comando para gerar mensagem para confirmar
a2	8	Comando para gerar mensagem para retirar dinheiro
a3	8	Comando para gerar mensagem para retirar produto
valor	8	Valor da cédula depositada

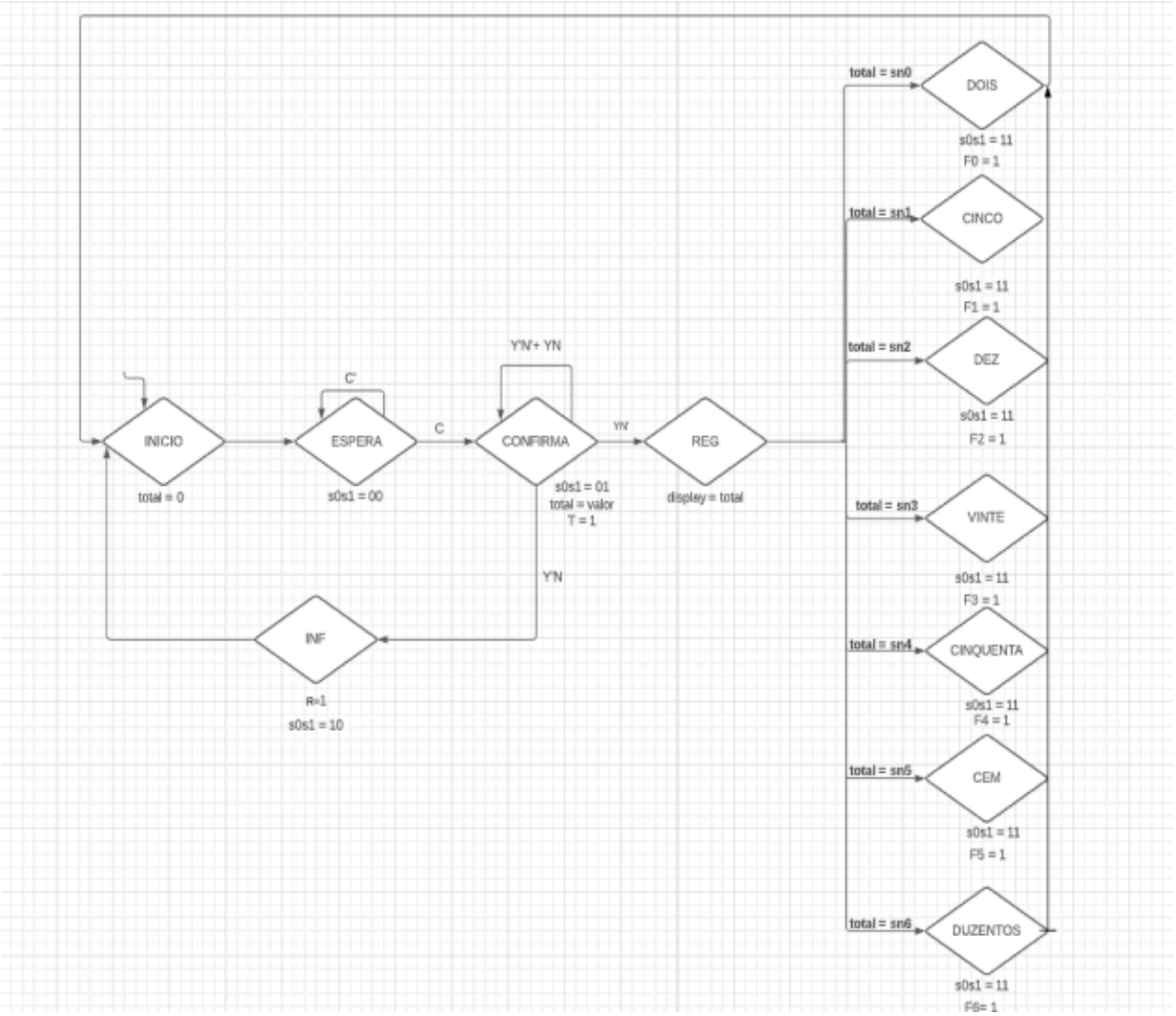
Saídas	Bits	Funcionalidades
Display	8	Valor da cédula inserida pelo usuário
Display2	8	Comando da mensagem escolhida para aparecer no display
R	1	Sinal que permite a devolução da cédula
T	1	Ativa o travamento da aleta
F0	1	Sinal que aciona a liberação do produto 0
F1	1	Sinal que aciona a liberação do produto 1
F2	1	Sinal que aciona a liberação do produto 2
F3	1	Sinal que aciona a liberação do produto 3
F4	1	Sinal que aciona a liberação do produto 4
F5	1	Sinal que aciona a liberação do produto 5
F6	1	Sinal que aciona a liberação do produto 6



Entradas(caminho de dados←bloco de controle)	**Bits**	**Funcionalidades**
total_ld	1	Indica o carregamento do registrador
total_clr	1	Indica o reset do registrador
s0	1	Sinal de seleção do MUX
s1	1	Sinal de seleção do MUX

Saídas(caminho de dados→bloco de controle)	**Bits**	**Funcionalidades**
igual2	1	Indica que o valor da nota inserida é de R$2
igual5	1	Indica que o valor da nota inserida é de R$5
igual10	1	Indica que o valor da nota inserida é de R$10
igual20	1	Indica que o valor da nota inserida é de R$20
igual50	1	Indica que o valor da nota inserida é de R$50
igual100	1	Indica que o valor da nota inserida é de R$100
igual200	1	Indica que o valor da nota inserida é de R$200

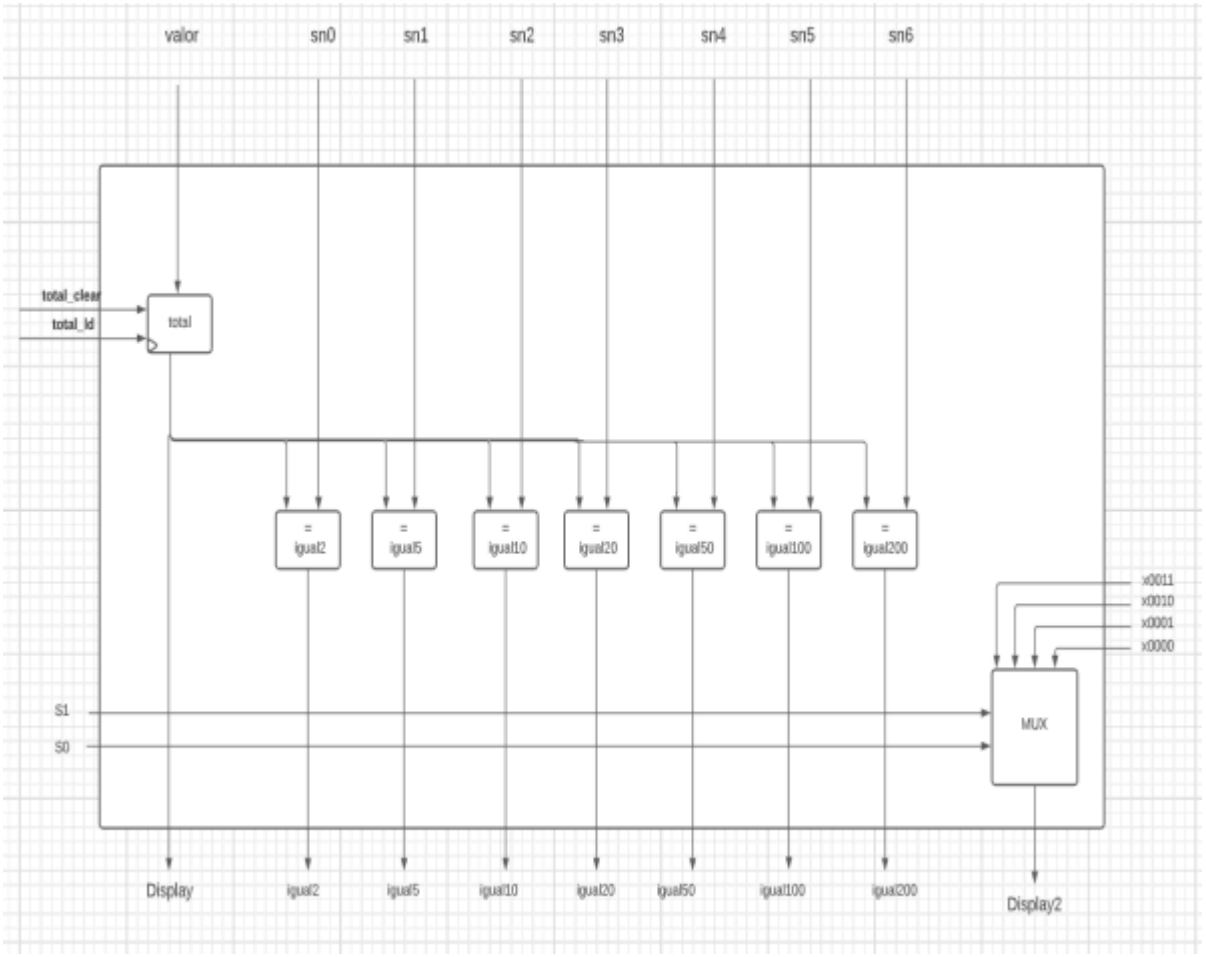
● Diagrama de Máquina de Estados de Alto Nível (diagrama conceitual)



● Projeto dos caminhos de dados (bloco operacional)

Assim que uma cédula de dinheiro de valor 2, 5, 10, 20, 50, 100 ou 200 reais é depositado na máquina de vendas e a compra seja confirmada pelo usuário, a entrada de controle (total_ld) torna-se 1 permitindo que o registrador, chamado de “total”, receba o valor da nota inserida que também corresponde ao valor do produto escolhido. Após isso, o valor de “total” é enviado aos comparadores igual2, igual5, igual10, igual20, igual50, igual100 e igual200, onde apenas uma deles estará com sua saída em nível lógico alto, pois apenas uma comparação será verdadeira.

O comparador no qual a igualdade seja verdadeira irá gerar o sinal que permitirá o sistema seguir para o estado da FSM onde está o produto referente a nota inserida. Chegando nesse estado o sistema liberará um sinal (F0 a F6), o qual permitirá a saída do produto ao usuário. Após isso, o sistema volta para o estado inicial, reseta o registrador “total” e fica à espera de um novo cliente.


valor
sn0
sn1
sn2
sn3
sn4
sn5
sn6
total_clear
total_ld
total
=
igual2
=
igual5
=
igual10
=
igual20
=
igual50
=
igual100
=
igual200
x0011
x0010
x0001
x0000
S1
S0
MUX
Display
igual2
igual5
igual10
igual20
igual50
igual100
igual200
Display2

- **SIMULAÇÃO DE CADA COMPONENTE INDIVIDUAL**

2.1 COMPARADOR

Código do comparador:

```
1   -- comparador de 8 bits
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   entity comparador is
6       generic(w : natural := 8);
7       port(
8           data1, data2 : in std_logic_vector(w-1 downto 0);
9           output       : out std_logic
10      );
11  end comparador;
12
13  architecture arch of comparador is
14  begin
15      output <= '1' when (data1 = data2) else '0';
16  end arch;
17
```

O componente possui uma porta de saída do tipo std_logic chamada output, que resulta em 1 se todos os bits dos vetores data1 e data2 forem iguais, e 0 caso contrário. A comparação é realizada na arquitetura, onde a igualdade entre os dois vetores é avaliada e atribuída ao sinal de saída. Isso é útil em circuitos digitais para verificar se dois conjuntos de dados binários são idênticos.

Testbench do comparador:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3
4   entity tb_comparador is
5   generic(
6       W: natural :=8
7   );
8   end tb_comparador;
9
10  architecture teste of tb_comparador is
11
12  signal data1, data2 :std_logic_vector(W-1 downto 0);
13  signal output : std_logic;
14
15  begin
16  instancia_comparador: entity work.comparador(arch) port map(data1=>data1,data2=>data2, output=>output);
17  data1<=x"02",x"05" after 5 ns, x"0A" after 10 ns, x"14" after 20 ns;
18  data2<=x"05",x"02" after 5 ns, x"0A" after 10 ns, x"14" after 20 ns;
19  end teste;
```

Este testbench tem o objetivo de verificar o funcionamento do comparador ao aplicar diferentes valores aos vetores de entrada data1 e data2 e observar o resultado no sinal output. A sequência de teste está configurada para verificar situações em que os vetores são iguais e diferentes, assegurando que o comparador funcione conforme esperado.

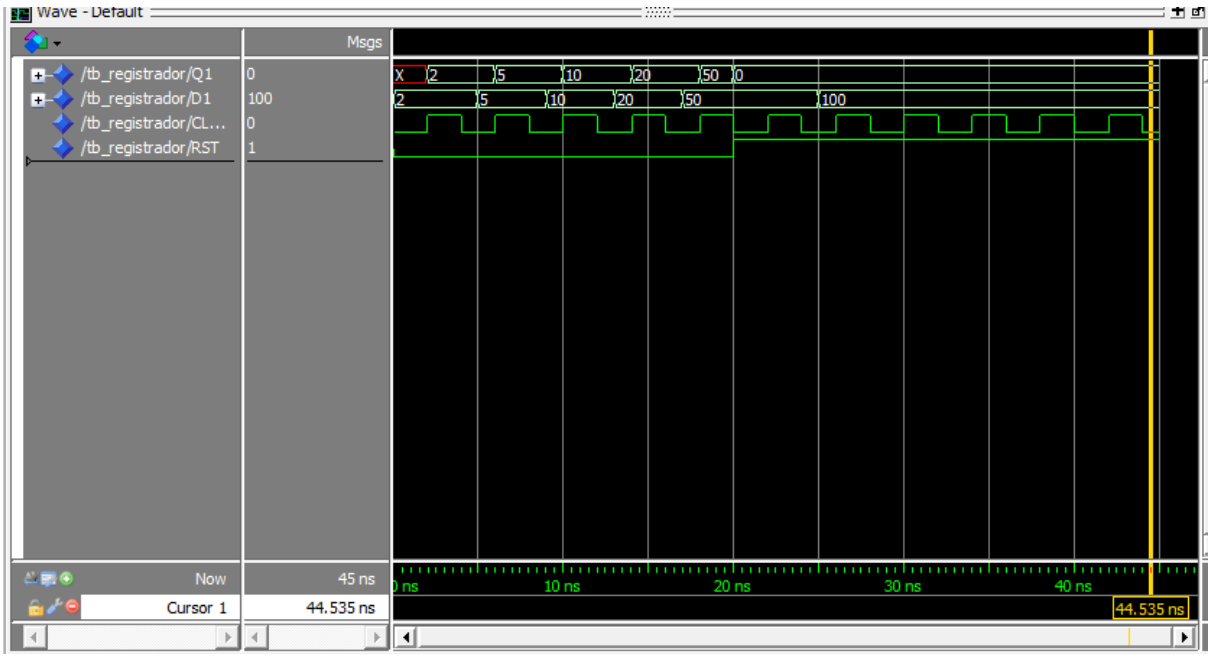
2.2 REGISTRADOR

Código do registrador:

```
1   -- registrador de 8 bits
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5
6   entity registrador is
7   generic(w: natural :=8);
8      port   (
9               d : in  std_logic_vector(w-1 downto 0);   -- entrada dados
10              q : out std_logic_vector(w-1 downto 0);   -- saida de dados
11              reset: in std_logic;                      -- reset assincrono
12              clk  : in std_logic                       -- signal clock
13          );
14  end registrador;
15
16  architecture arch of registrador is
17
18  begin
19     process(clk, reset) is
20        begin
21        if(reset = '1') then
22           q <= x"00";
23        elsif(rising_edge(clk)) then
24           q <= d;
25        end if;
26     end process;
27  end arch;
```

Testbench do registrador:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

entity tb_registrador is
    generic(
        W : natural := 8
    );
end tb_registrador;

architecture teste of tb_registrador is

signal Q1,D1 : std_logic_vector(W-1 downto 0);
signal CLOCK : std_logic := '0';
signal RST   : std_logic;

begin

instancia_registrador: entity work.registrador(arch) port map(q=>Q1,d=>D1,clk=>CLOCK,reset=>RST);

CLOCK <= not(CLOCK) after 2 ns;
D1 <= x"02", x"05" after 5 ns, x"0A" after 9 ns, x"14" after 13 ns,
      x"32" after 17 ns, x"64" after 25 ns;
RST <= '0', '1' after 20 ns;
end teste;

```



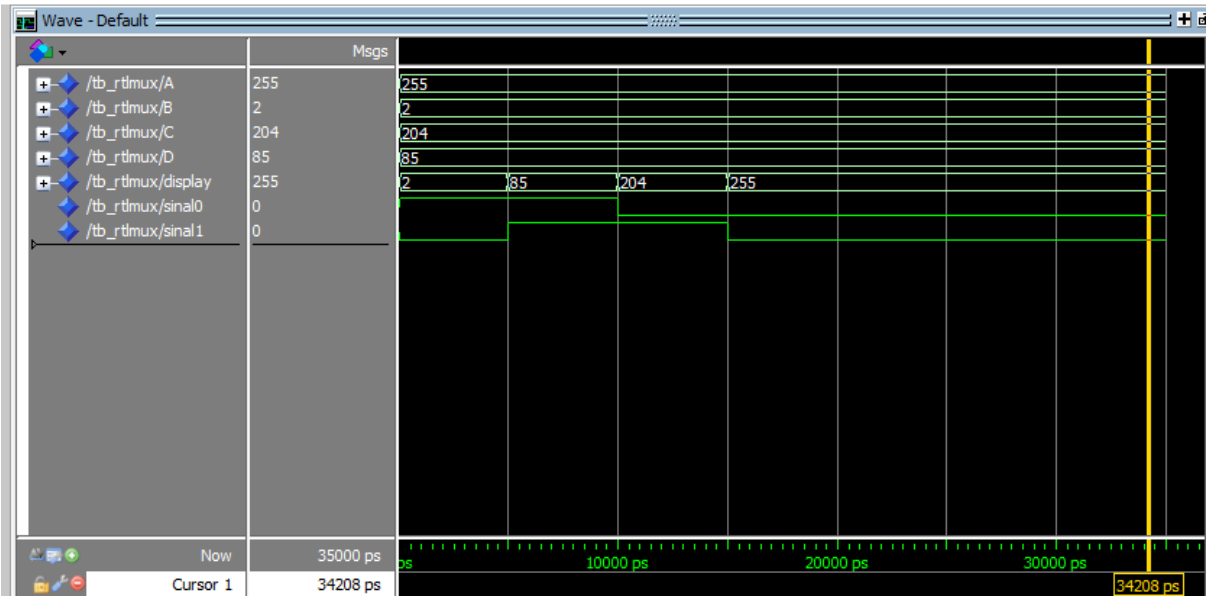
2.3 MUX

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity RTLMux is
5   generic(w: natural :=8);
6   port(
7     a1      : in  std_logic_vector(w-1 downto 0);
8     a2      : in  std_logic_vector(w-1 downto 0);
9     a3      : in  std_logic_vector(w-1 downto 0);
10    a4      : in  std_logic_vector(w-1 downto 0);
11    s0     : in  std_logic;
12    s1     : in  std_logic;
13    Display2       : out std_logic_vector(w-1 downto 0));
14  end RTLMux;
15
16  architecture rtl of RTLMux is
17
18  signal sel : std_logic_vector(1 downto 0);
19
20  begin
21  sel <= (s1 & s0);
22    with sel select
23      Display2 <= a1 when "00",
24         a2 when "01",
25         a3 when "10",
26         a4 when others;
27  end rtl;
```

O código VHDL define um componente chamado RTLMux, que implementa um multiplexador (mux) de 4 para 1 parametrizado, utilizando sinais de controle s0 e s1 para selecionar qual dos quatro vetores de entrada (a1, a2, a3, a4) será enviado para a saída Display2. A largura dos vetores de entrada e saída é definida pelo parâmetro genérico w, que por padrão é 8 bits. Dentro da arquitetura rtl, os sinais de controle s0 e s1 são combinados em um vetor de seleção sel. Através de uma estrutura with select, a saída Display2 é atribuída ao vetor de entrada correspondente: a1 se sel for "00", a2 se for "01", a3 se for "10", e a4 para qualquer outra combinação de sel, permitindo a escolha entre múltiplas entradas baseadas nas entradas de seleção.

Testbench do MUX

```
3
4   entity tb_RTLMux is
5   end tb_RTLMux;
6
7   architecture teste of tb_RTLMux is
8
9   component RTLMux is
10  generic(w: natural :=8);
11      port (
12          a1 : in std_logic_vector(w-1 downto 0);
13          a2 : in std_logic_vector(w-1 downto 0);
14          a3 : in std_logic_vector(w-1 downto 0);
15          a4 : in std_logic_vector(w-1 downto 0);
16          s0 : in std_logic;
17          s1 : in std_logic;
18          Display2 : out std_logic_vector(w-1 downto 0)
19      );
20  end component;
21
22  signal A, B, C, D, display: std_logic_vector(7 downto 0);
23  signal sinal0, sinal1: std_logic;
24  begin
25      instancia_RTLMux: RTLMux port map(a1=>A, a2=>B, a3=>C, a4=>D, Display2=>display, s0=>sinal0, s1=>sinal1);
26      A <= x"FF";
27      B <= x"02";
28      C <= x"CC";
29      D <= x"55";
30      sinal0 <= '1', '1' after 5 ns, '0' after 10 ns, '0' after 15 ns;
31      sinal1 <= '0', '1' after 5 ns, '1' after 10 ns, '0' after 15 ns;
32  end teste;
33
```



Este testbench verifica se o componente RTLMux está funcionando corretamente ao testar todas as combinações possíveis de seleção para as entradas do multiplexador. Ele aplica valores específicos aos sinais de entrada e modifica os sinais de seleção ao longo do tempo para observar se o comportamento do multiplexador está de acordo com as expectativas. A execução do testbench em um simulador VHDL, como ModelSim ou Vivado Simulator, permitirá observar a saída display e confirmar que a lógica do multiplexador é correta.

- **Controladora, Caminho de dados e Interligação**

3.1 Caminho de Dados

Código do caminho de dados:

```
--caminho
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity caminho is
    port (
        ctotal_clr: in std_logic;
        ctotal_ld: in std_logic;
        cvalor: in std_logic_vector(7 downto 0);
        csn0, csn1, csn2, csn3, csn4, csn5, csn6: in std_logic_vector(7 downto 0);
        ca0, ca1, ca2, ca3: in std_logic_vector(7 downto 0);
        cs0, cs1: in std_logic;
        cdisplay, cdisplay2: out std_logic_vector(7 downto 0);
        cigual2, cigual5, cigual10, cigual20, cigual50, cigual100, cigual200: out std_logic
    );
end caminho;

architecture cam of caminho is
    component registrador is
        port (
            d: in std_logic_vector(7 downto 0);
            q: out std_logic_vector(7 downto 0);
            reset: in std_logic;
            clk: in std_logic
        );
    end component;

    component comparador is
        port (
            data1, data2: in std_logic_vector(7 downto 0);
            output: out std_logic
        );
    end component;

    component RTLMux is
        port (
            a1: in std_logic_vector(7 downto 0);
            a2: in std_logic_vector(7 downto 0);
            a3: in std_logic_vector(7 downto 0);
            a4: in std_logic_vector(7 downto 0);
            s0: in std_logic;
            s1: in std_logic;
            Display2: out std_logic_vector(7 downto 0)
        );
    end component;

    signal sdisplay, sdisplay2: std_logic_vector(7 downto 0);

begin
    registra: registrador port map(
        d => cvalor,
        q => sdisplay,
        clk => ctotal_ld,  -- Conectado ao sinal de carregamento
        reset => ctotal_clr  -- Conectado ao sinal de reset
    );
    cdisplay <= sdisplay;

    combinacao2: comparador port map(data1 => sdisplay, data2 => csn0, output => cigual2);
    combinacao5: comparador port map(data1 => sdisplay, data2 => csn1, output => cigual5);
    combinacao10: comparador port map(data1 => sdisplay, data2 => csn2, output => cigual10);
    combinacao20: comparador port map(data1 => sdisplay, data2 => csn3, output => cigual20);
    combinacao50: comparador port map(data1 => sdisplay, data2 => csn4, output => cigual50);
    combinacao100: comparador port map(data1 => sdisplay, data2 => csn5, output => cigual100);
    combinacao200: comparador port map(data1 => sdisplay, data2 => csn6, output => cigual200);

    mux: RTLMux port map(
        a1 => ca0,
        a2 => ca1,
        a3 => ca2,
        a4 => ca3,
        s0 => cs0,
        s1 => cs1,
        Display2 => sdisplay2
    );
    cdisplay2 <= sdisplay2;
end cam;
```

O módulo “caminho” é responsável por processar e comparar valores binários de entrada e gerar os sinais correspondentes de igualdade para os valores das cédulas (2, 5, 10, 20, 50, 100, 200). Este módulo também controla a atualização de dois displays de 8 bits (display e display2), dependendo dos sinais de controle recebidos, sendo o display no final utilizado para exibir o valor salvo no registrador da cédula inserida e o display2 o ‘estado’ em que a máquina está(a0 a a4) selecionado pelo RTLMux.

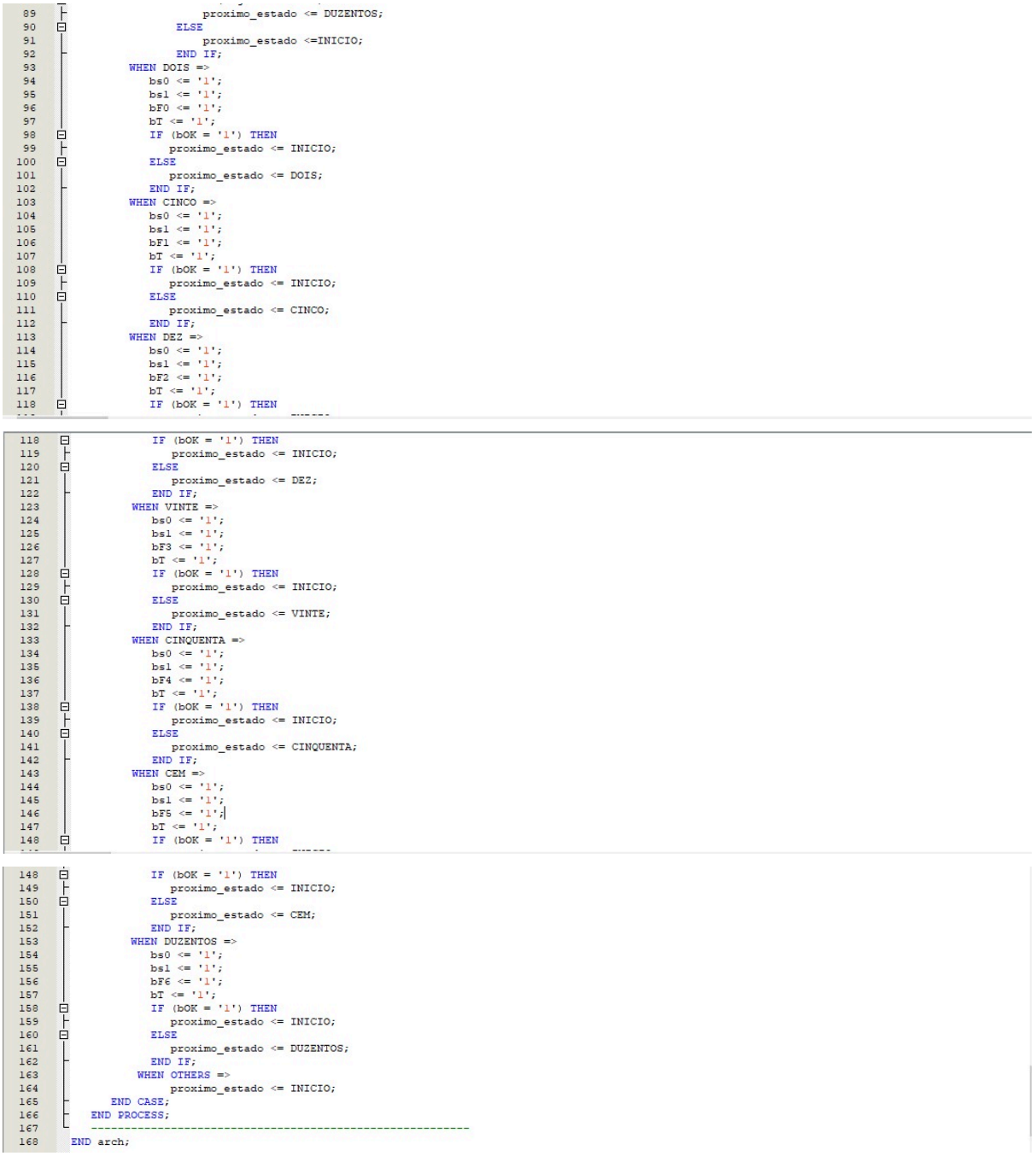
3.2 Controladora

Código do Bloco de Controle:

```
--BlocoControle
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY blococontrole IS
   PORT (
      bC, bY, bN, bCLOCK, bRESET, bOK, bD : IN STD_LOGIC;
      bigual2, bigual5, bigual10, bigual20, bigual50, bigual100, bigual200 : IN STD_LOGIC;
      btotal_ld, btotal_clr, bs0, bs1 : BUFFER STD_LOGIC;
      bT, bR, bF0, bF1, bF2, bF3, bF4, bF5, bF6 : OUT STD_LOGIC);
END blococontrole;

ARCHITECTURE arch OF blococontrole IS
    TYPE estado IS (INICIO, ESPERA, INF, CONFIRMA, REG, DOIS, CINCO, DEZ, VINTE, CINQUENTA, CEM, DUZENTOS);
    SIGNAL estado_atual : estado := INICIO;
    SIGNAL proximo_estado : estado;
    ATTRIBUTE syn_encoding : STRING;
    ATTRIBUTE syn_encoding OF estado : TYPE IS "one_hot";
BEGIN
    sequencial : PROCESS (bRESET, bCLOCK) IS
    BEGIN
        IF (bRESET = '1') THEN
            estado_atual <= INICIO;
        ELSIF (rising_edge(bCLOCK)) THEN
            estado_atual <= proximo_estado;
        END IF;
    END PROCESS;

    combinacional_moore : PROCESS (estado_atual, bC, bY, bN, bOK, bD) IS
    BEGIN
        btotal_clr <= '0';
```

```
        btotal_clr <= '0';
        btotal_ld <= '0';

        CASE estado_atual IS
            WHEN INICIO =>
                btotal_clr <= '1';
                bF0 <= '0';
                bF1 <= '0';
                bF2 <= '0';
                bF3 <= '0';
                bF4 <= '0';
                bF5 <= '0';
                bF6 <= '0';
                bT <= '0';
                bR <= '0';
                btotal_clr <= '1';
                proximo_estado <= ESPERA;

            WHEN ESPERA =>
                bs0 <= '0';
                bs1 <= '0';
                IF (bC = '1') THEN
                    proximo_estado <= CONFIRMA;
                ELSE
                    proximo_estado <= ESPERA;
                END IF;
```

```
             WHEN CONFIRMA =>
                 bs0 <= '0';
                 bs1 <= '1';
                 bT <= '1';
                 btotal_ld <= '1';
                 IF (bY = '1' AND bN = '0') THEN
                     proximo_estado <= REG;
                 ELSIF (bY = '0' AND bN = '1') THEN
                     proximo_estado <= INF;
                 ELSE
                     proximo_estado <= CONFIRMA;
                 END IF;

             WHEN REG =>
                 btotal_ld <= '1';  -- Load the total value during registration
                 bs0 <= '0';
                 bs1 <= '1';
                 bT <= '1';
                 IF (bigual2 = '1') THEN
                     proximo_estado <= DOIS;
                 ELSIF (bigual5 = '1') THEN
                     proximo_estado <= CINCO;
                 ELSIF (bigual10 = '1') THEN
                     proximo_estado <= DEZ;
                 ELSIF (bigual20 = '1') THEN
                     proximo_estado <= VINTE;
                 ELSIF (bigual50 = '1') THEN
                     proximo_estado <= CINQUENTA;
                 ELSIF (bigual100 = '1') THEN
                     proximo_estado <= CEM;
                 ELSIF (bigual200 ='1') THEN
```



```
                proximo_estado <= DUZENTOS;
            ELSE
                proximo_estado <=INICIO;
            END IF;
        WHEN DOIS =>
            bs0 <= '1';
            bs1 <= '1';
            bF0 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= DOIS;
            END IF;
        WHEN CINCO =>
            bs0 <= '1';
            bs1 <= '1';
            bF1 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= CINCO;
            END IF;
        WHEN DEZ =>
            bs0 <= '1';
            bs1 <= '1';
            bF2 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
```



```
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= DEZ;
            END IF;
        WHEN VINTE =>
            bs0 <= '1';
            bs1 <= '1';
            bF3 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= VINTE;
            END IF;
        WHEN CINQUENTA =>
            bs0 <= '1';
            bs1 <= '1';
            bF4 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= CINQUENTA;
            END IF;
        WHEN CEM =>
            bs0 <= '1';
            bs1 <= '1';
            bF5 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
```



```
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= CEM;
            END IF;
        WHEN DUZENTOS =>
            bs0 <= '1';
            bs1 <= '1';
            bF6 <= '1';
            bT <= '1';
            IF (bOK = '1') THEN
                proximo_estado <= INICIO;
            ELSE
                proximo_estado <= DUZENTOS;
            END IF;
         WHEN OTHERS =>
                proximo_estado <= INICIO;
    END CASE;
  END PROCESS;
  ------------------------------------------------------------
END arch;
```

O módulo “blococontrole” implementa uma máquina de estados finitos (FSM) que gerencia o fluxo de controle do sistema. Ele monitora os sinais de entrada, determina o próximo estado com base nas condições atuais, e gera os sinais de controle apropriados para os outros módulos do sistema.
O módulo “blococontrole” começa em um estado inicial (INICIO) e, com base nos sinais de entrada e na lógica interna, transita entre diferentes estados de espera, confirmação ou um dos estados correspondentes a uma das notas(2,5,10,20,50,100,200). Em cada estado, diferentes sinais de controle são ativados para dirigir o comportamento do sistema, como carregar novos valores, limpar registros ou sinalizar condições específicas.

3.3 Interligação

Código da interligação

```
--interligacao
library IEEE;
use IEEE.std_logic_1164.all;

entity interligacao is
  port(
    C, Y, N, CLOCK, RESET, OK, D : in std_logic;
    R, T, F0, F1, F2, F3, F4, F5, F6 : out std_logic;

    valor, sn0, sn1, sn2, sn3, sn4, sn5, sn6, a0, a1, a2, a3 : in std_logic_vector(7 downto 0);
    display, display2 : out std_logic_vector(7 downto 0)
  );
end interligacao;

architecture inter of interligacao is

  component caminho is
    port(
      cvalor, csn0, csn1, csn2, csn3, csn4, csn5, csn6, ca0, ca1, ca2, ca3 : in std_logic_vector(7 downto 0);
      cdisplay, cdisplay2 : out std_logic_vector(7 downto 0);
      ctotal_ld, ctotal_clr, cs0, cs1 : in std_logic;
      cigual2, cigual5, cigual10, cigual20, cigual50, cigual100, cigual200 : out std_logic
    );
  end component;

  component blococontrole is
    port(
      bC, bY, bN, bCLOCK, bRESET, bOK, bD : in std_logic;
      bigual2, bigual5, bigual10, bigual20, bigual50, bigual100, bigual200 : in std_logic;
      btotal_ld, btotal_clr, bs0, bs1 : out std_logic;
      bT, bR, bF0, bF1, bF2, bF3, bF4, bF5, bF6 : out std_logic
    );
  end component;

  signal total_ld, total_clr, s0, s1 : std_logic;
  signal igual2, igual5, igual10, igual20, igual50, igual100, igual200 : std_logic;

begin
  caminho1 : caminho port map(
    cvalor => valor,
    csn0 => sn0,
    csn1 => sn1,
    csn2 => sn2,
    csn3 => sn3,
    csn4 => sn4,
    csn5 => sn5,
    csn6 => sn6,
    cdisplay => display,
    cdisplay2 => display2,
    ctotal_ld => total_ld,
    ctotal_clr => total_clr,
    cs0 => s0,
    cs1 => s1,
    ca0 => a0,
    ca1 => a1,
    ca2 => a2,
    ca3 => a3,
    cigual2 => igual2,
    cigual5 => igual5,
    cigual10 => igual10,
    cigual20 => igual20,
    cigual50 => igual50,
    cigual100 => igual100,
    cigual200 => igual200
  );

  controle : blococontrole port map(
    bC => C, bY => Y, bN => N, bCLOCK => CLOCK, bRESET => RESET, bOK => OK, bD => D,
    bigual2 => igual2, bigual5 => igual5, bigual10 => igual10,
    bigual20 => igual20, bigual50 => igual50, bigual100 => igual100,
    bigual200 => igual200,
    btotal_ld => total_ld, btotal_clr => total_clr, bs0 => s0, bs1 => s1,
    bT => T, bR => R, bF0 => F0, bF1 => F1, bF2 => F2, bF3 => F3,
    bF4 => F4, bF5 => F5, bF6 => F6
  );
end inter;
```

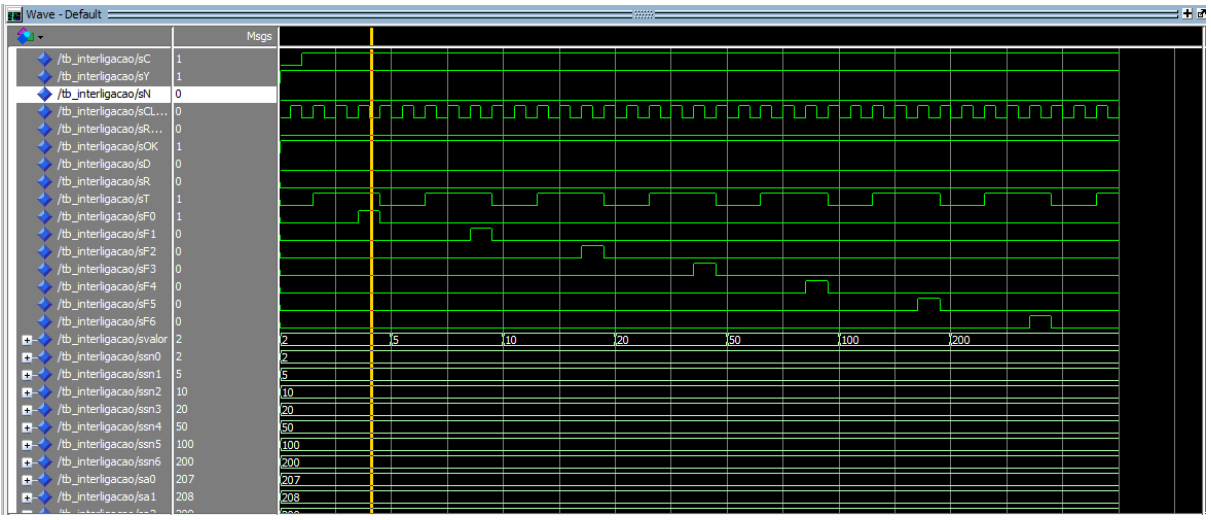
O módulo “interligacao” atua como um módulo de integração, conectando o módulo de controle (“blococontrole”) com o módulo de processamento de dados (“caminho”). Ele garante que os sinais de controle e dados sejam corretamente roteados entre os módulos, permitindo que o sistema funcione de forma coordenada. O módulo “interligacao” conecta os sinais de controle e dados entre o “blococontrole” e o “caminho”. Ele mapeia as saídas de um

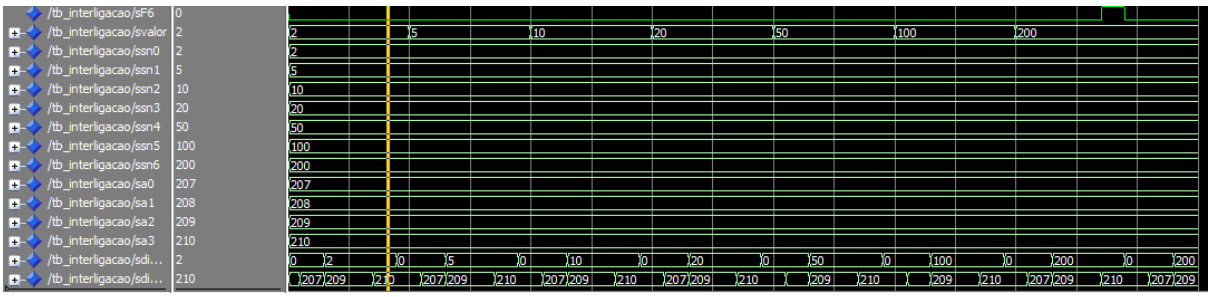
módulo para as entradas de outro, garantindo que a FSM possa controlar corretamente o processamento dos dados e a exibição dos resultados.

Testbench da Interligação:

```
--tb_interligacao
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY tb_interligacao IS
  GENERIC (
    w : NATURAL := 4
  );
END tb_interligacao;

ARCHITECTURE teste OF tb_interligacao IS

  COMPONENT interligacao IS
    PORT (
      C, Y, N, CLOCK, RESET, OK, D : IN STD_LOGIC;
      R, T, F0, F1, F2, F3, F4, F5, F6 : OUT STD_LOGIC;

      valor, sn0, sn1, sn2, sn3, sn4, sn5, sn6, a0, a1, a2, a3 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      display, display2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
  END COMPONENT;

  SIGNAL sC, sY, sN, sCLOCK, sRESET, sOK, sD : STD_LOGIC := '0';
  SIGNAL sR, sT, sF0, sF1, sF2, sF3, sF4, sF5, sF6 : STD_LOGIC;

  SIGNAL svalor, ssn0, ssn1, ssn2, ssn3, ssn4, ssn5, ssn6, sa0, sa1, sa2, sa3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL sdisplay, sdisplay2 : STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN
    instancia_interligacao : interligacao
    PORT MAP(
        C => sC, Y => sY, N => sN, CLOCK => sCLOCK, RESET => sRESET, OK => sOK, D => sD,
        R => sR, T => sT, F0 => sF0, F1 => sF1, F2 => sF2, F3 => sF3, F4 => sF4,
        F5 => sF5, F6 => sF6, valor => svalor, sn0 => ssn0, sn1 => ssn1, sn2 => ssn2,
        sn3 => ssn3, sn4 => ssn4, sn5 => ssn5, sn6 => ssn6, a0 => sa0, a1 => sa1,
        a2 => sa2, a3 => sa3, display => sdisplay, display2 => sdisplay2
    );

    sCLOCK <= NOT sCLOCK AFTER 2 ns;
    sRESET <= '0';

    sC <= '1' AFTER 4 ns;
    sY <= '1';
    sN <= '0';
    sOK <= '1';
    sD <= '0';

   svalor <= x"02", -- Ativa sf0
             x"05" after 20 ns,  -- Ativa sf1
             x"0A" after 40 ns,  -- Ativa sf2
             x"14" after 60 ns,  -- Ativa sf3
             x"32" after 80 ns,  -- Ativa sf4
             x"64" after 100 ns,  -- Ativa sf5
             x"C8" after 120 ns;  -- Ativa sf6
    ssn0 <= x"02";
    ssn1 <= x"05";
    ssn2 <= x"0A";
    ssn3 <= x"14";
    ssn4 <= x"32";
    ssn5 <= x"64";
    ssn6 <= x"C8";
    sa0 <= x"CF";
    sa1 <= x"D1";
    sa2 <= x"D2";
    sa3 <= x"D2";

END teste;
```

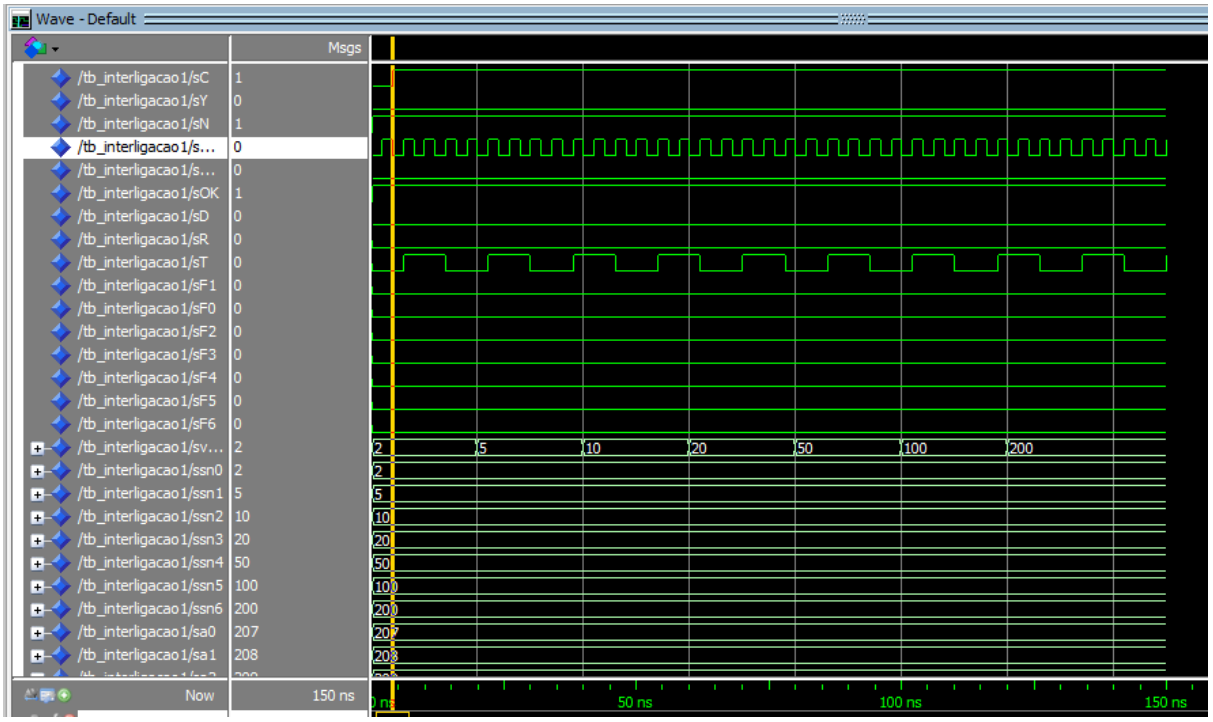
Simulação da interligação:

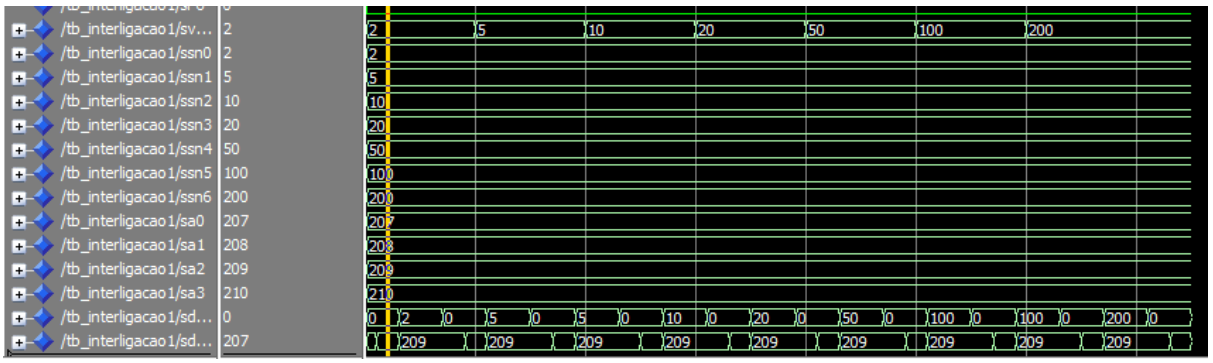




A simulação do testbench nos mostra a máquina funcionando como o esperado, “liberando” os produtos de acordo com o valor, já que como no trecho abaixo fica explicito, deixamos na simulação sC, sY e sOK em 1 para simular que o sinal de recebimento da nota sendo sempre enviado, o usuário sempre aceitando o pedido feito e ele sempre retira seu produto após a compra. sN, sD, sR foram definidos sempre em 0, simulando uma situação em que o cliente não recusa a compra, tampouco o reset é acionado.

```
sCLOCK <= NOT sCLOCK AFTER 2 ns;
sRESET <= '0';

sC <= '1' AFTER 4 ns;
sY <= '1';
sN <= '0';
sOK <= '1';
sD <= '0';
```





Nesse segundo testbench fazemos o caso contrário ao primeiro, sendo um cliente que sempre recusando o produto, sC é sempre 1 para ele sempre conseguir processar a cédula como recebida, mas sN está em 1 para colocar a recusa do cliente como uma constante. Como é visível na simulação, os produtos não são liberados, e a mensagem em vez de ser a de retirar o produto(210) é a de retirar o dinheiro(209)

```
sCLOCK <= NOT sCLOCK AFTER 2 ns;
sRESET <= '0';

sC <= '1' AFTER 4 ns;
sY <= '0';
sN <= '1';
sOK <= '1';
sD <= '0';
```

https://github.com/Glaucus-M-Alm/TPFINALLSD