



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

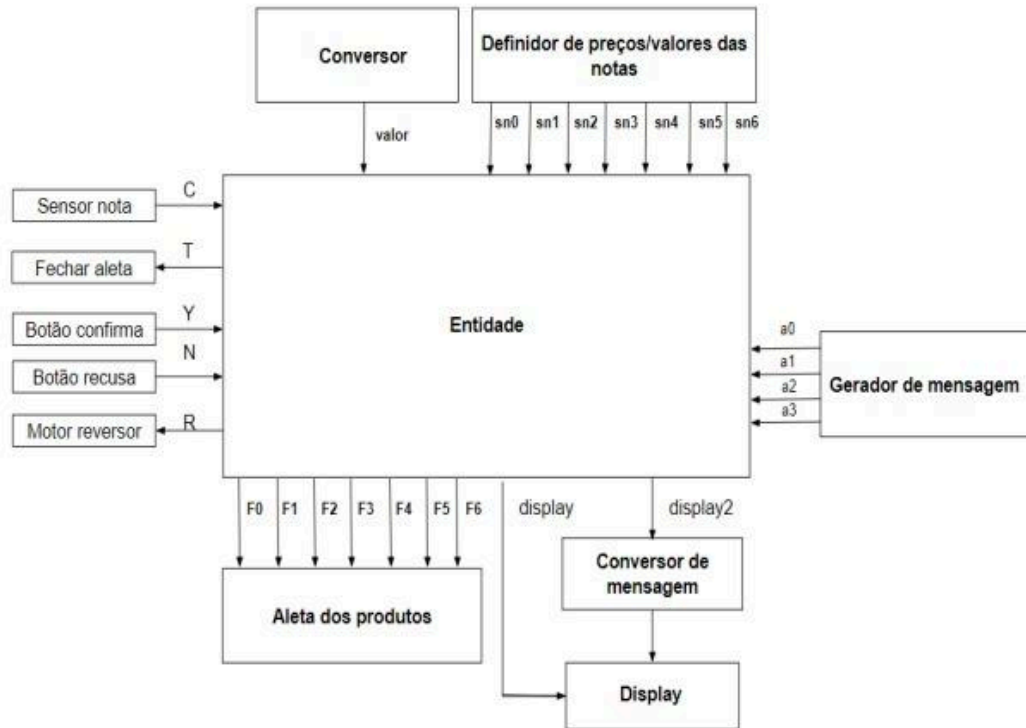
A partir do diagrama de blocos abaixo o caminho percorrido pode ser representado de forma mais elucidativa da seguinte maneira:

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.

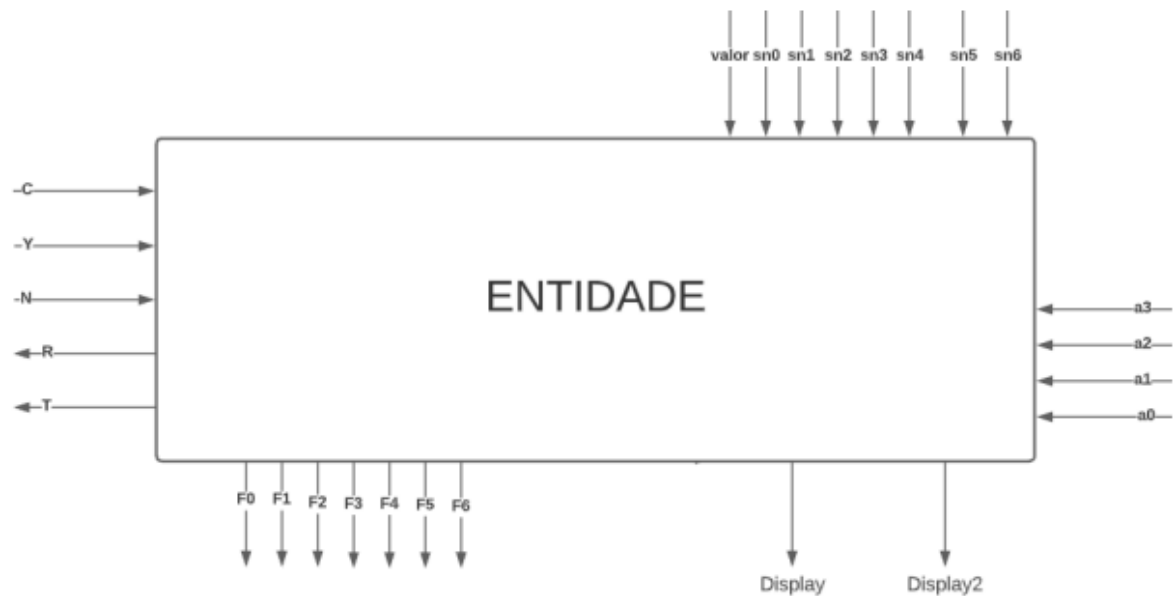
2. Após isso uma mensagem será apresentada para o cliente, perguntando se ele deseja confirmar a operação, caso ele não confirme, a FSM acionará o sinal R, onde fará com que o “motor reversor” devolva a cédula ao cliente. Além disso, nesse momento a FSM aciona o sinal T, que tranca a aleta para que novas 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 “início” e fica a espera de novos sinais do “sensor nota”

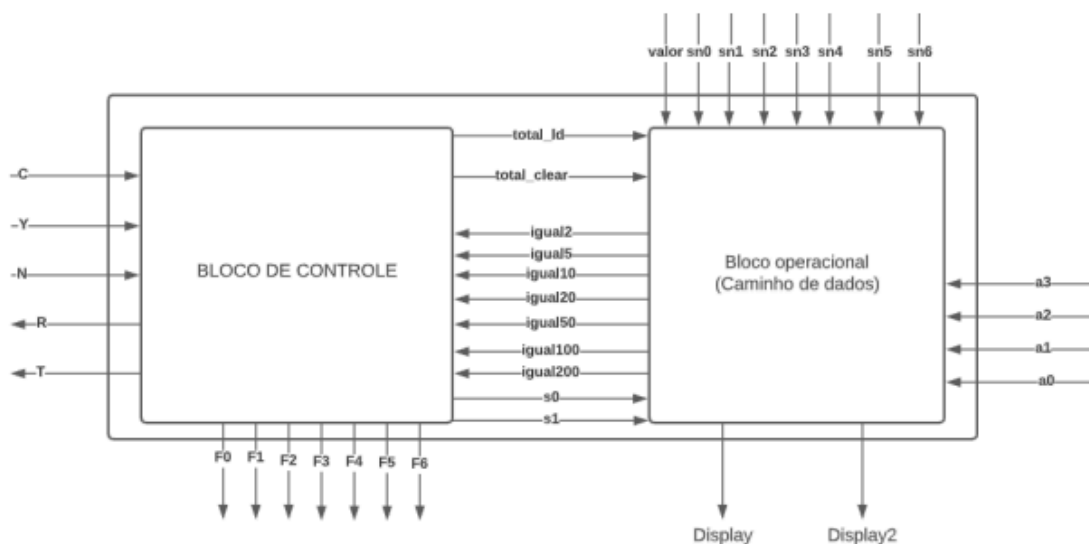


Representações da entidade em diferentes níveis de abstração e suas respectivas tabelas de entradas e saídas



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

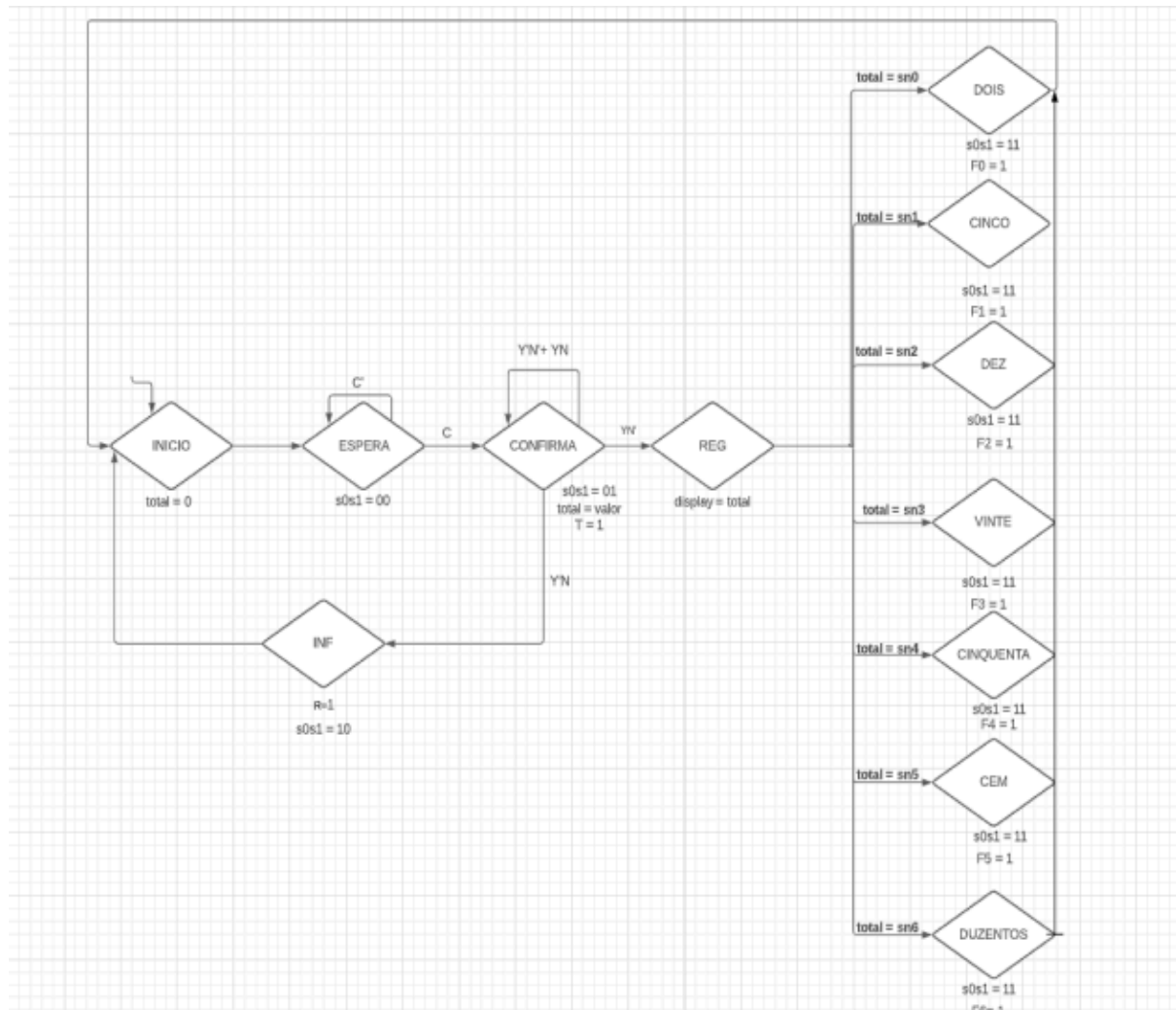
Saídas	Bits	Funcionalidades
Display	8	Valor da cédula inserida pelo usuário
Display2	8	Comando da mensagem escolhida para aparecer no display
R	1	Sinal que permite a devolução da cédula
T	1	Ativa o travamento da aleta
F0	1	Sinal que aciona a liberação do produto 0
F1	1	Sinal que aciona a liberação do 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

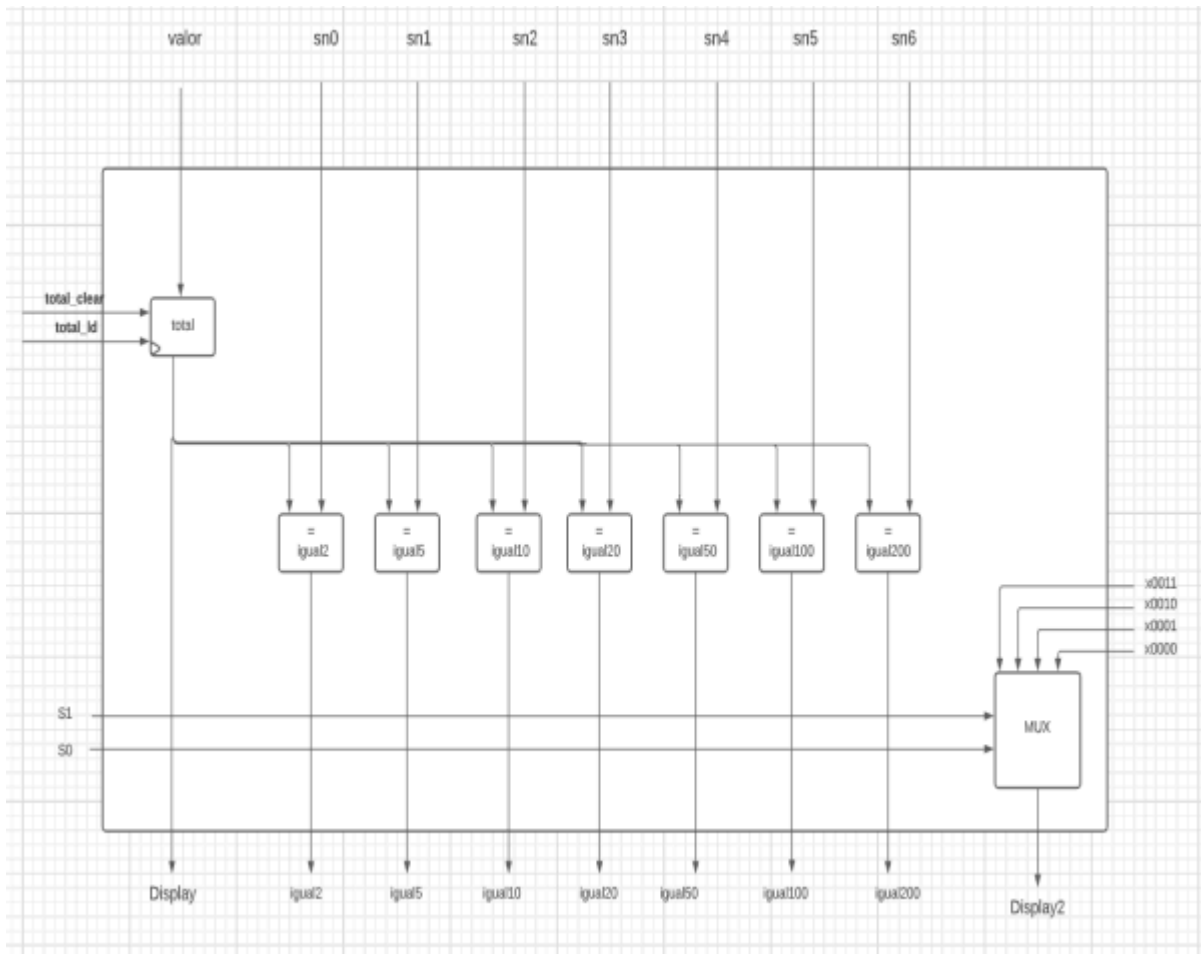
- Diagrama de Máquina de Estados de Alto Nível (diagrama conceitual)



- 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.



- **SIMULAÇÃO DE CADA COMPONENTE INDIVIDUAL**

2.1 COMPARADOR

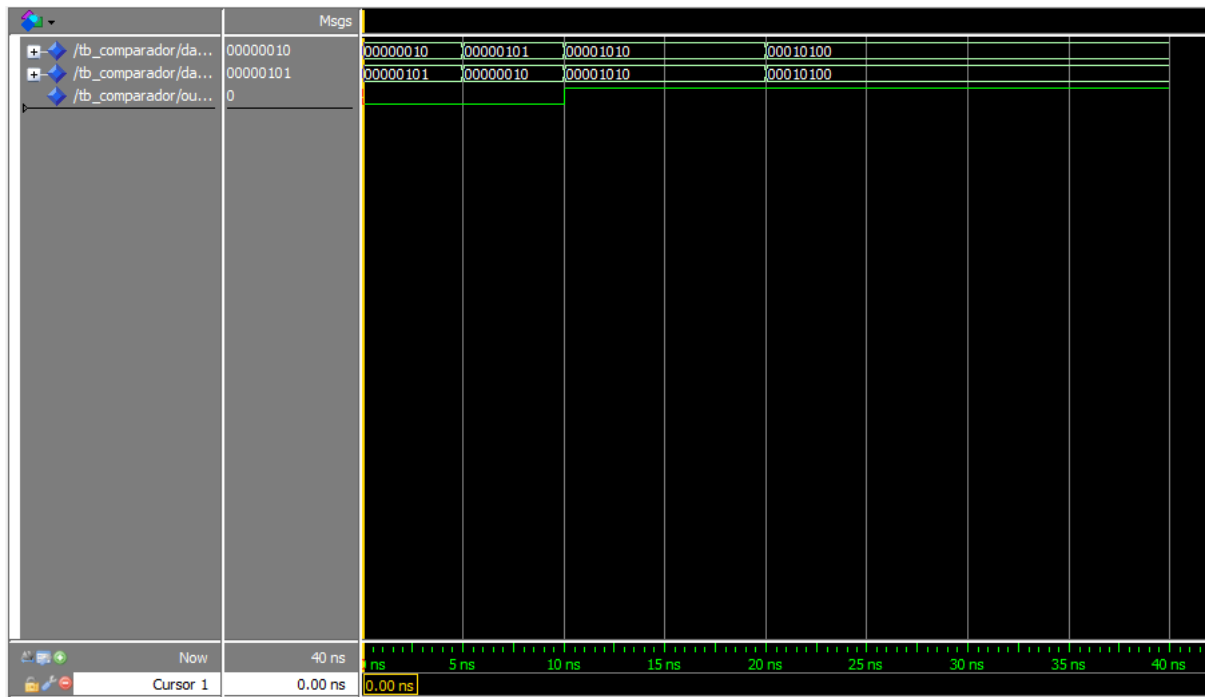
Código do comparador:

```
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); -- saída de dados
11     reset: in std_logic; -- reset assíncrono
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;

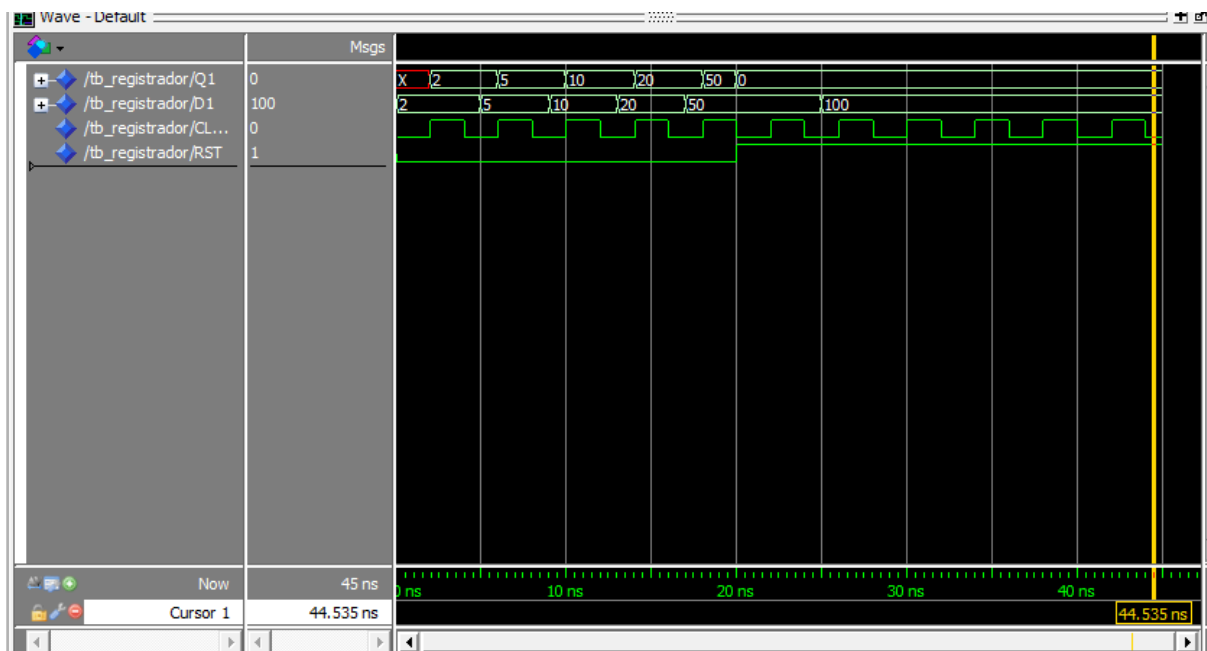
```

Testbench do registrador:

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity tb_registrador is
6  generic(
7      W : natural := 8
8  );
9  end tb_registrador;
10
11 architecture teste of tb_registrador is
12
13     signal Q1,D1 : std_logic_vector(W-1 downto 0);
14     signal CLOCK : std_logic := '0';
15     signal RST   : std_logic;
16
17 begin
18
19     instancia_registrador: entity work.registrador (arch) port map (q=>Q1,d=>D1,clk=>CLOCK,reset=>RST);
20
21     CLOCK <= not(CLOCK) after 2 ns;
22     D1 <= x"02", x"05" after 5 ns, x"0A" after 9 ns, x"14" after 13 ns,
23         x"32" after 17 ns, x"64" after 25 ns;
24     RST <= '0', '1' after 20 ns;
25 end teste;
26

```



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;
```

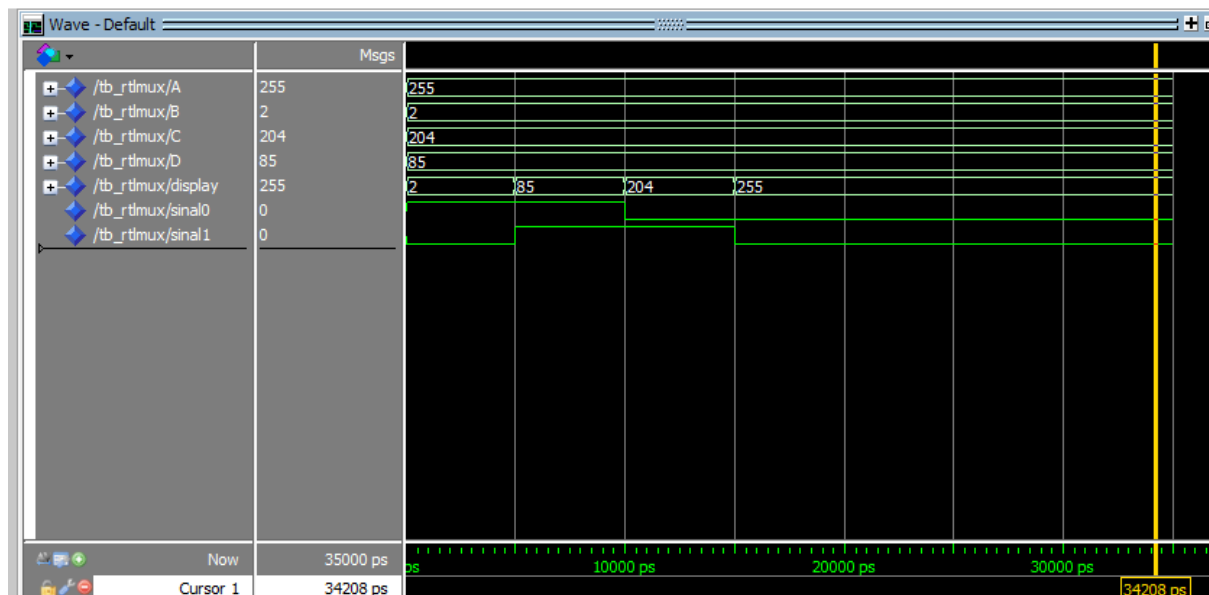
O código VHDL define um componente chamado RTLMux, que implementa um multiplexador (mux) de 4 para 1 parametrizado, utilizando sinais de controle s0 e s1 para selecionar qual dos quatro vetores de entrada (a1, a2, a3, a4) será enviado para a saída Display2. A largura dos vetores de entrada e saída é 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:

```
1  --caminho
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_arith.all;
5  use ieee.std_logic_unsigned.all;
6
7  entity caminho is
8  port (
9      cttotal_clr: in std_logic;
10     cttotal_ld: in std_logic;
11     cvalor: in std_logic_vector(7 downto 0);
12     cs0, csn1, csn2, csn3, csn4, csn5, csn6: in std_logic_vector(7 downto 0);
13     ca0, ca1, ca2, ca3: in std_logic_vector(7 downto 0);
14     cs0, cs1: in std_logic;
15     cdisplay, cdisplay2: out std_logic_vector(7 downto 0);
16     cigual2, cigual5, cigual10, cigual20, cigual50, cigual100, cigual200: out std_logic
17 );
18 end caminho;
19
20 architecture cam of caminho is
21     component registrador is
22     port (
23         d: in std_logic_vector(7 downto 0);
24         q: out std_logic_vector(7 downto 0);
25         reset: in std_logic;
26         clk: in std_logic
27     );
28     end component;
29
30     component comparador is
31     port (
32         data1, data2: in std_logic_vector(7 downto 0);
33         output: out std_logic
34     );
35     end component;
36
37     component RTLMux is
38     port (
39         a1: in std_logic_vector(7 downto 0);
40         a2: in std_logic_vector(7 downto 0);
41         a3: in std_logic_vector(7 downto 0);
42         a4: in std_logic_vector(7 downto 0);
43         s0: in std_logic;
44         s1: in std_logic;
45         Display2: out std_logic_vector(7 downto 0)
46     );
47     end component;
48
49     signal sdisplay, sdisplay2: std_logic_vector(7 downto 0);
50
51 begin
52     registra: registrador port map(
53         d => cvalor,
54         q => sdisplay,
55         clk => cttotal_ld, -- Conectado ao sinal de carregamento
56         reset => cttotal_clr -- Conectado ao sinal de reset
57     );
58     cdisplay <= sdisplay;
59
60     combinacao2: comparador port map(data1 => sdisplay, data2 => csn0, output => cigual2);
61     combinacao5: comparador port map(data1 => sdisplay, data2 => csn1, output => cigual5);
62     combinacao10: comparador port map(data1 => sdisplay, data2 => csn2, output => cigual10);
63     combinacao20: comparador port map(data1 => sdisplay, data2 => csn3, output => cigual20);
64     combinacao50: comparador port map(data1 => sdisplay, data2 => csn4, output => cigual50);
65     combinacao100: comparador port map(data1 => sdisplay, data2 => csn5, output => cigual100);
66     combinacao200: comparador port map(data1 => sdisplay, data2 => csn6, output => cigual200);
67
68     mux: RTLMux port map(
69         a1 => ca0,
70         a2 => ca1,
71         a3 => ca2,
72         a4 => ca3,
73         s0 => cs0,
74         s1 => cs1,
75         Display2 => sdisplay2
76     );
77     cdisplay2 <= sdisplay2;
78 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:

```
1  --BlocoControle
2  LIBRARY IEEE;
3  USE IEEE.std_logic_1164.ALL;
4
5  ENTITY blococontrole IS
6  PORT (
7      bC, bY, bN, bCLOCK, bRESET, bOK, bD : IN STD_LOGIC;
8      bigual2, bigual5, bigual10, bigual20, bigual50, bigual100, bigual200 : IN STD_LOGIC;
9      bttotal_ld, bttotal_clr, bs0, bs1 : BUFFER STD_LOGIC;
10     bT, bR, bF0, bF1, bF2, bF3, bF4, bF5, bF6 : OUT STD_LOGIC);
11 END blococontrole;
12
13 ARCHITECTURE arch OF blococontrole IS
14     TYPE estado IS (INICIO, ESPERA, INF, CONFIRMA, REG, DOIS, CINCO, DEZ, VINTE, CINQUENTA, CEM, DUZENTOS);
15     SIGNAL estado_atual : estado := INICIO;
16     SIGNAL proximo_estado : estado;
17     ATTRIBUTE syn_encoding : STRING;
18     ATTRIBUTE syn_encoding OF estado : TYPE IS "one_hot";
19 BEGIN
20     sequencial : PROCESS (bRESET, bCLOCK) IS
21     BEGIN
22         IF (bRESET = '1') THEN
23             estado_atual <= INICIO;
24         ELSIF (rising_edge(bCLOCK)) THEN
25             estado_atual <= proximo_estado;
26         END IF;
27     END PROCESS;
28
29     combinacional_moore : PROCESS (estado_atual, bC, bY, bN, bOK, bD) IS
30     BEGIN
31         bttotal_clr <= '0';
32         bttotal_ld <= '0';
33
34         CASE estado_atual IS
35             WHEN INICIO =>
36                 bttotal_clr <= '1';
37                 bF0 <= '0';
38                 bF1 <= '0';
39                 bF2 <= '0';
40                 bF3 <= '0';
41                 bF4 <= '0';
42                 bF5 <= '0';
43                 bF6 <= '0';
44                 bT <= '0';
45                 bR <= '0';
46                 bttotal_clr <= '1';
47                 proximo_estado <= ESPERA;
48
49             WHEN ESPERA =>
50                 bs0 <= '0';
51                 bs1 <= '0';
52                 IF (bC = '1') THEN
53                     proximo_estado <= CONFIRMA;
54                 ELSE
55                     proximo_estado <= ESPERA;
56                 END IF;
57
58             WHEN CONFIRMA =>
59                 bs0 <= '0';
60                 bs1 <= '1';
61                 bT <= '1';
62                 bttotal_ld <= '1';
63                 IF (bY = '1' AND bN = '0') THEN
64                     proximo_estado <= REG;
65                 ELSIF (bY = '0' AND bN = '1') THEN
66                     proximo_estado <= INF;
67                 ELSE
68                     proximo_estado <= CONFIRMA;
69                 END IF;
70
71             WHEN REG =>
72                 bttotal_ld <= '1'; -- Load the total value during registration
73                 bs0 <= '0';
74                 bs1 <= '1';
75                 bT <= '1';
76                 IF (bigual2 = '1') THEN
77                     proximo_estado <= DOIS;
78                 ELSIF (bigual5 = '1') THEN
79                     proximo_estado <= CINCO;
80                 ELSIF (bigual10 = '1') THEN
81                     proximo_estado <= DEZ;
82                 ELSIF (bigual20 = '1') THEN
83                     proximo_estado <= VINTE;
84                 ELSIF (bigual50 = '1') THEN
85                     proximo_estado <= CINQUENTA;
86                 ELSIF (bigual100 = '1') THEN
87                     proximo_estado <= CEM;
88                 ELSIF (bigual200 = '1') THEN
89                     proximo_estado <= DUZENTOS;
```

```

89         proximo_estado <= DUZENTOS;
90     ELSE
91         proximo_estado <= INICIO;
92     END IF;
93 WHEN DOIS =>
94     bs0 <= '1';
95     bs1 <= '1';
96     bF0 <= '1';
97     bT <= '1';
98     IF (bOK = '1') THEN
99         proximo_estado <= INICIO;
100    ELSE
101        proximo_estado <= DOIS;
102    END IF;
103 WHEN CINCO =>
104     bs0 <= '1';
105     bs1 <= '1';
106     bF1 <= '1';
107     bT <= '1';
108     IF (bOK = '1') THEN
109         proximo_estado <= INICIO;
110    ELSE
111        proximo_estado <= CINCO;
112    END IF;
113 WHEN DEZ =>
114     bs0 <= '1';
115     bs1 <= '1';
116     bF2 <= '1';
117     bT <= '1';
118     IF (bOK = '1') THEN

```

```

118     IF (bOK = '1') THEN
119         proximo_estado <= INICIO;
120     ELSE
121         proximo_estado <= DEZ;
122     END IF;
123 WHEN VINTE =>
124     bs0 <= '1';
125     bs1 <= '1';
126     bF3 <= '1';
127     bT <= '1';
128     IF (bOK = '1') THEN
129         proximo_estado <= INICIO;
130     ELSE
131         proximo_estado <= VINTE;
132     END IF;
133 WHEN CINQUENTA =>
134     bs0 <= '1';
135     bs1 <= '1';
136     bF4 <= '1';
137     bT <= '1';
138     IF (bOK = '1') THEN
139         proximo_estado <= INICIO;
140     ELSE
141         proximo_estado <= CINQUENTA;
142     END IF;
143 WHEN CEM =>
144     bs0 <= '1';
145     bs1 <= '1';
146     bF5 <= '1';
147     bT <= '1';
148     IF (bOK = '1') THEN

```

```

148     IF (bOK = '1') THEN
149         proximo_estado <= INICIO;
150     ELSE
151         proximo_estado <= CEM;
152     END IF;
153 WHEN DUZENTOS =>
154     bs0 <= '1';
155     bs1 <= '1';
156     bF6 <= '1';
157     bT <= '1';
158     IF (bOK = '1') THEN
159         proximo_estado <= INICIO;
160     ELSE
161         proximo_estado <= DUZENTOS;
162     END IF;
163 WHEN OTHERS =>
164     proximo_estado <= INICIO;
165 END CASE;
166 END PROCESS;
167 -----
168 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

```
1  --interligacao
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity interligacao is
6  port(
7      C, Y, N, CLOCK, RESET, OK, D : in std_logic;
8      R, T, F0, F1, F2, F3, F4, F5, F6 : out std_logic;
9
10     valor, sn0, sn1, sn2, sn3, sn4, sn5, sn6, a0, a1, a2, a3 : in std_logic_vector(7 downto 0);
11     display, display2 : out std_logic_vector(7 downto 0)
12 );
13 end interligacao;
14
15 architecture inter of interligacao is
16
17     component caminho is
18     port(
19         cvalor, csn0, csn1, csn2, csn3, csn4, csn5, csn6, ca0, ca1, ca2, ca3 : in std_logic_vector(7 downto 0);
20         cdisplay, cdisplay2 : out std_logic_vector(7 downto 0);
21         cttotal_id, cttotal_clr, cs0, cs1 : in std_logic;
22         cigual2, cigual5, cigual10, cigual20, cigual50, cigual100, cigual200 : out std_logic
23     );
24     end component;
25
26     component blococontrolre is
27     port(
28         bC, bY, bN, bCLOCK, bRESET, bOK, bD : in std_logic;
29         bigual2, bigual5, bigual10, bigual20, bigual50, bigual100, bigual200 : in std_logic;
30         bttotal_id, bttotal_clr, bs0, bs1 : out std_logic;
31         bT, bR, bF0, bF1, bF2, bF3, bF4, bF5, bF6 : out std_logic
32     );
33     end component;
34
35     signal total_id, total_clr, s0, s1 : std_logic;
36     signal igual2, igual5, igual10, igual20, igual50, igual100, igual200 : std_logic;
37
38 begin
39     caminho1 : caminho port map(
40         cvalor => valor,
41         csn0 => sn0,
42         csn1 => sn1,
43         csn2 => sn2,
44         csn3 => sn3,
45         csn4 => sn4,
46         csn5 => sn5,
47         csn6 => sn6,
48         cdisplay => display,
49         cdisplay2 => display2,
50         cttotal_id => total_id,
51         cttotal_clr => total_clr,
52         cs0 => s0,
53         cs1 => s1,
54         ca0 => a0,
55         ca1 => a1,
56         ca2 => a2,
57         ca3 => a3,
58         cigual2 => igual2,
59         cigual5 => igual5,
60         cigual10 => igual10,
61         cigual20 => igual20,
62
63         cigual50 => igual50,
64         cigual100 => igual100,
65         cigual200 => igual200
66     );
67
68     controle : blococontrolre port map(
69         bC => C, bY => Y, bN => N, bCLOCK => CLOCK, bRESET => RESET, bOK => OK, bD => D,
70         bigual2 => igual2, bigual5 => igual5, bigual10 => igual10,
71         bigual20 => igual20, bigual50 => igual50, bigual100 => igual100,
72         bigual200 => igual200,
73         bttotal_id => total_id, bttotal_clr => total_clr, bs0 => s0, bs1 => s1,
74         bT => T, bR => R, bF0 => F0, bF1 => F1, bF2 => F2, bF3 => F3,
75         bF4 => F4, bF5 => F5, bF6 => F6
76     );
77 end inter;
```

O módulo “interligacao” atua como um módulo de integração, conectando o módulo de controle (“blococontrolre”) 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 “blococontrolre” 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.

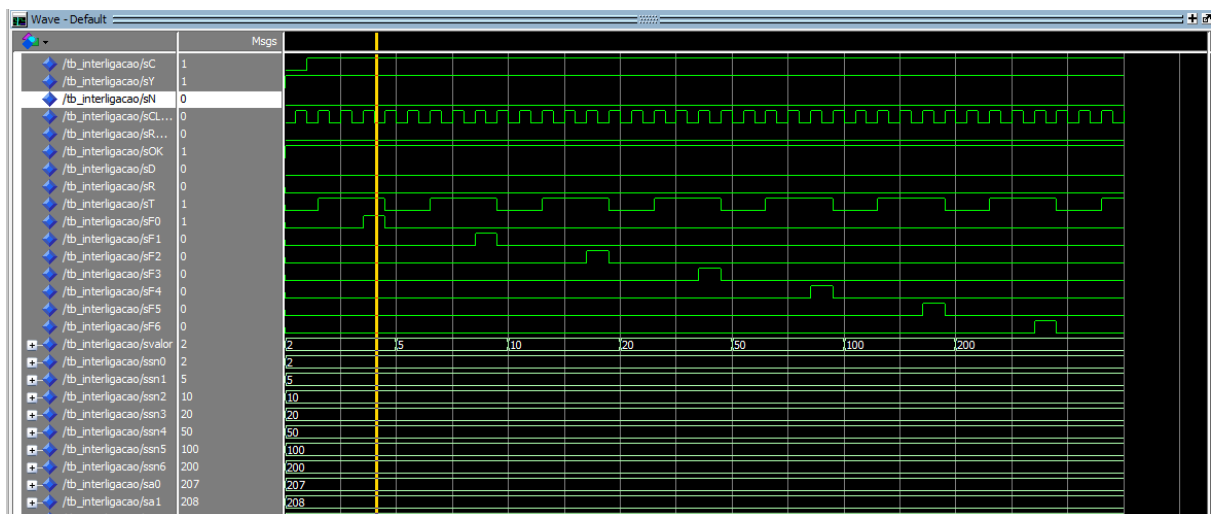
Testbench da Interligação:

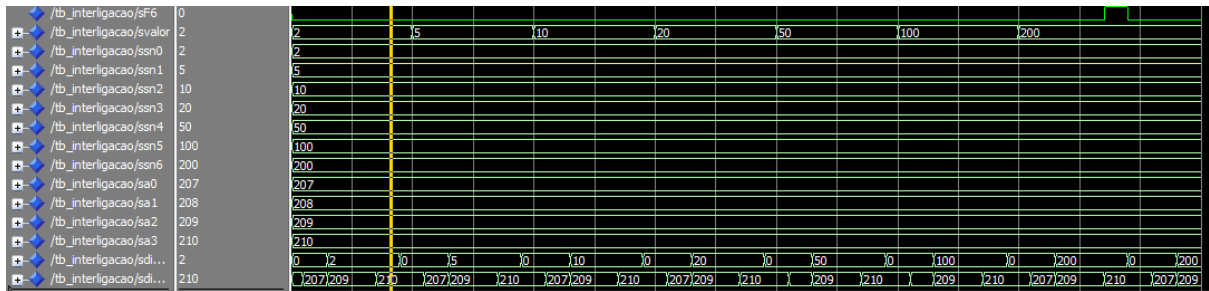
```

1  --tb_interligacao
2  LIBRARY IEEE;
3  USE IEEE.STD_LOGIC_1164.ALL;
4  USE IEEE.numeric_std.ALL;
5
6  ENTITY tb_interligacao IS
7  GENERIC (
8      w : NATURAL := 4
9  );
10 END tb_interligacao;
11
12 ARCHITECTURE teste OF tb_interligacao IS
13
14     COMPONENT interligacao IS
15     PORT (
16         C, Y, N, CLOCK, RESET, OK, D : IN STD_LOGIC;
17         R, T, F0, F1, F2, F3, F4, F5, F6 : OUT STD_LOGIC;
18
19         valor, sn0, sn1, sn2, sn3, sn4, sn5, sn6, a0, a1, a2, a3 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
20         display, display2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
21     END COMPONENT;
22
23     SIGNAL sC, sY, sN, sCLOCK, sRESET, sOK, sD : STD_LOGIC := '0';
24     SIGNAL sR, sT, sF0, sF1, sF2, sF3, sF4, sF5, sF6 : STD_LOGIC;
25
26     SIGNAL svalor, ssn0, ssn1, ssn2, ssn3, ssn4, ssn5, ssn6, sa0, sa1, sa2, sa3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
27     SIGNAL sdisplay, sdisplay2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
28
29 BEGIN
30     instancia_interligacao : interligacao
31     PORT MAP(
32
33         C => sC, Y => sY, N => sN, CLOCK => sCLOCK, RESET => sRESET, OK => sOK, D => sD,
34         R => sR, T => sT, F0 => sF0, F1 => sF1, F2 => sF2, F3 => sF3, F4 => sF4,
35         F5 => sF5, F6 => sF6, valor => svalor, sn0 => ssn0, sn1 => ssn1, sn2 => ssn2,
36         sn3 => ssn3, sn4 => ssn4, sn5 => ssn5, sn6 => ssn6, a0 => sa0, a1 => sa1,
37         a2 => sa2, a3 => sa3, display => sdisplay, display2 => sdisplay2
38     );
39
40     sCLOCK <= NOT sCLOCK AFTER 2 ns;
41     sRESET <= '0';
42
43     sC <= '1' AFTER 4 ns;
44     sY <= '1';
45     sN <= '0';
46     sOK <= '1';
47     sD <= '0';
48
49     svalor <= x"02", -- Ativa sf0
50     x"05" after 20 ns, -- Ativa sf1
51     x"0A" after 40 ns, -- Ativa sf2
52     x"14" after 60 ns, -- Ativa sf3
53     x"32" after 80 ns, -- Ativa sf4
54     x"64" after 100 ns, -- Ativa sf5
55     x"C8" after 120 ns; -- Ativa sf6
56
57     ssn0 <= x"02";
58     ssn1 <= x"05";
59     ssn2 <= x"0A";
60     ssn3 <= x"14";
61     ssn4 <= x"32";
62     ssn5 <= x"64";
63     ssn6 <= x"C8";
64
65     sa0 <= x"CF";
66     sa1 <= x"D1";
67     sa2 <= x"D2";
68     sa3 <= x"D2";
69
70 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 explícito, 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';
```

