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

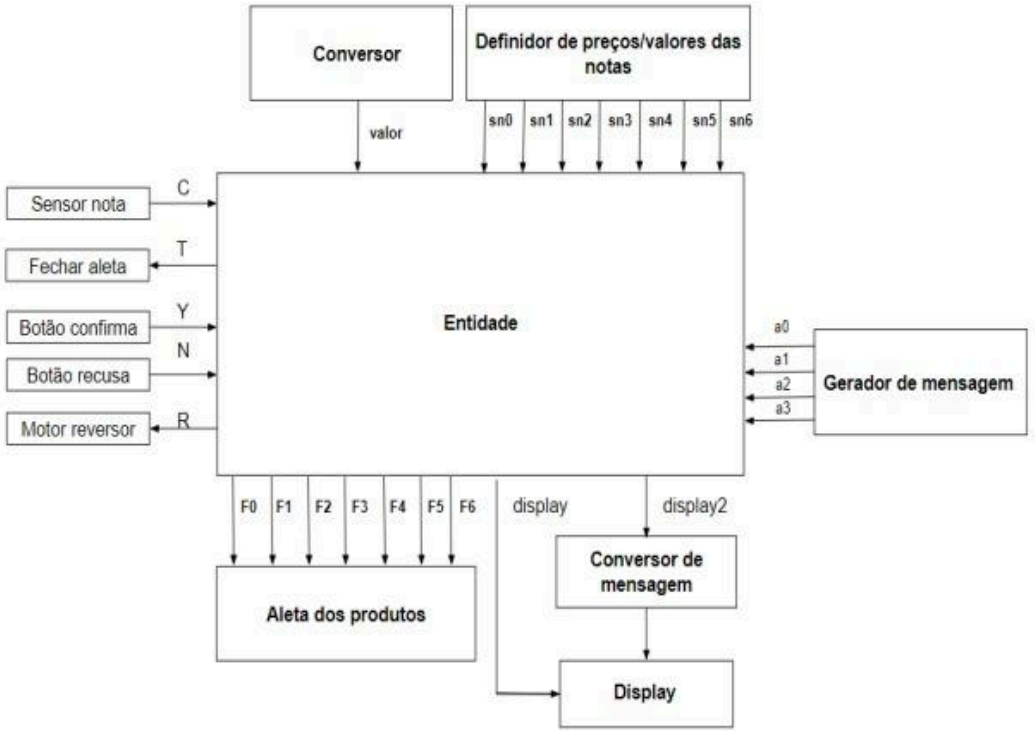
A partir do diagrama de blocos abaixo o caminho percorrido pode ser representado de forma mais elucidativa da seguinte maneira:

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.

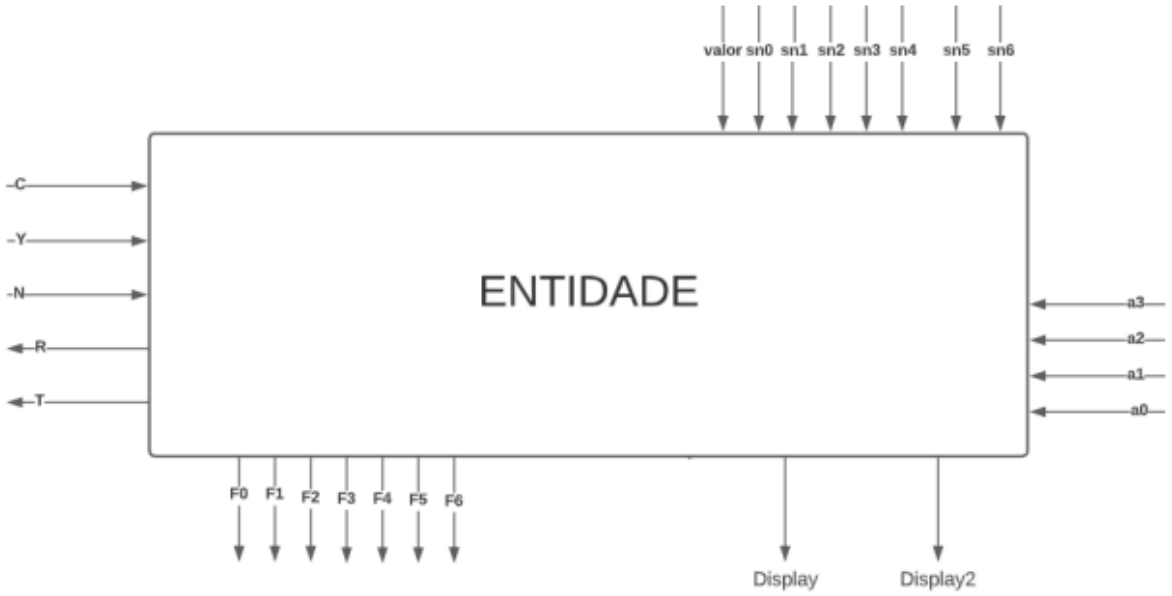
2. Após isso uma mensagem será apresentada para o cliente, perguntando se ele deseja confirmar a operação, caso ele não confirme, a FSM acionará o sinal R, onde fará com que o “motor reversor” devolva a cédula ao cliente. Além disso, nesse momento a FSM aciona o sinal T, que tranca a aleta para que novas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

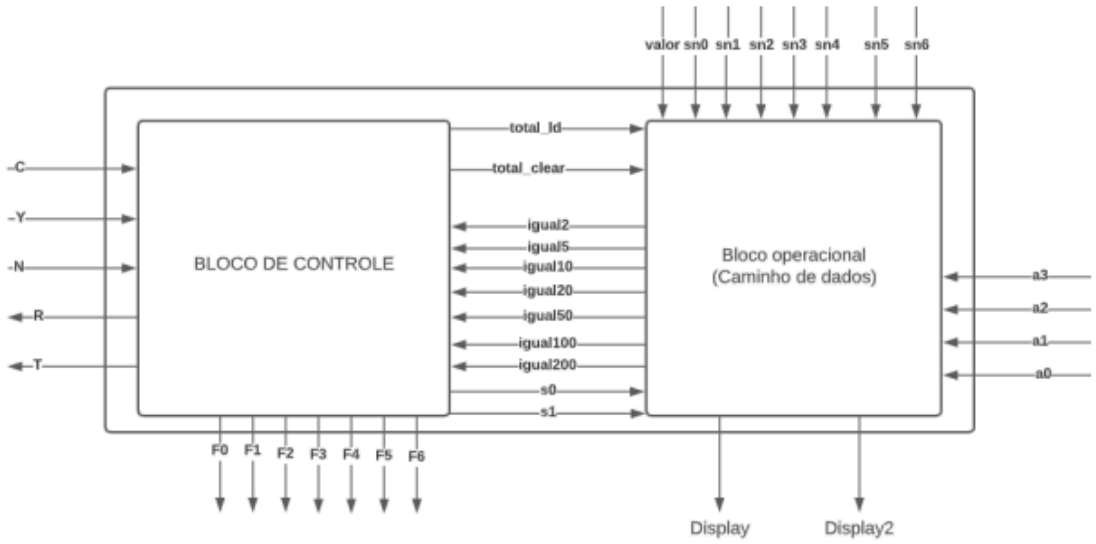


Representações da entidade em diferentes níveis de abstração e suas respectivas tabelas de entradas e saídas



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

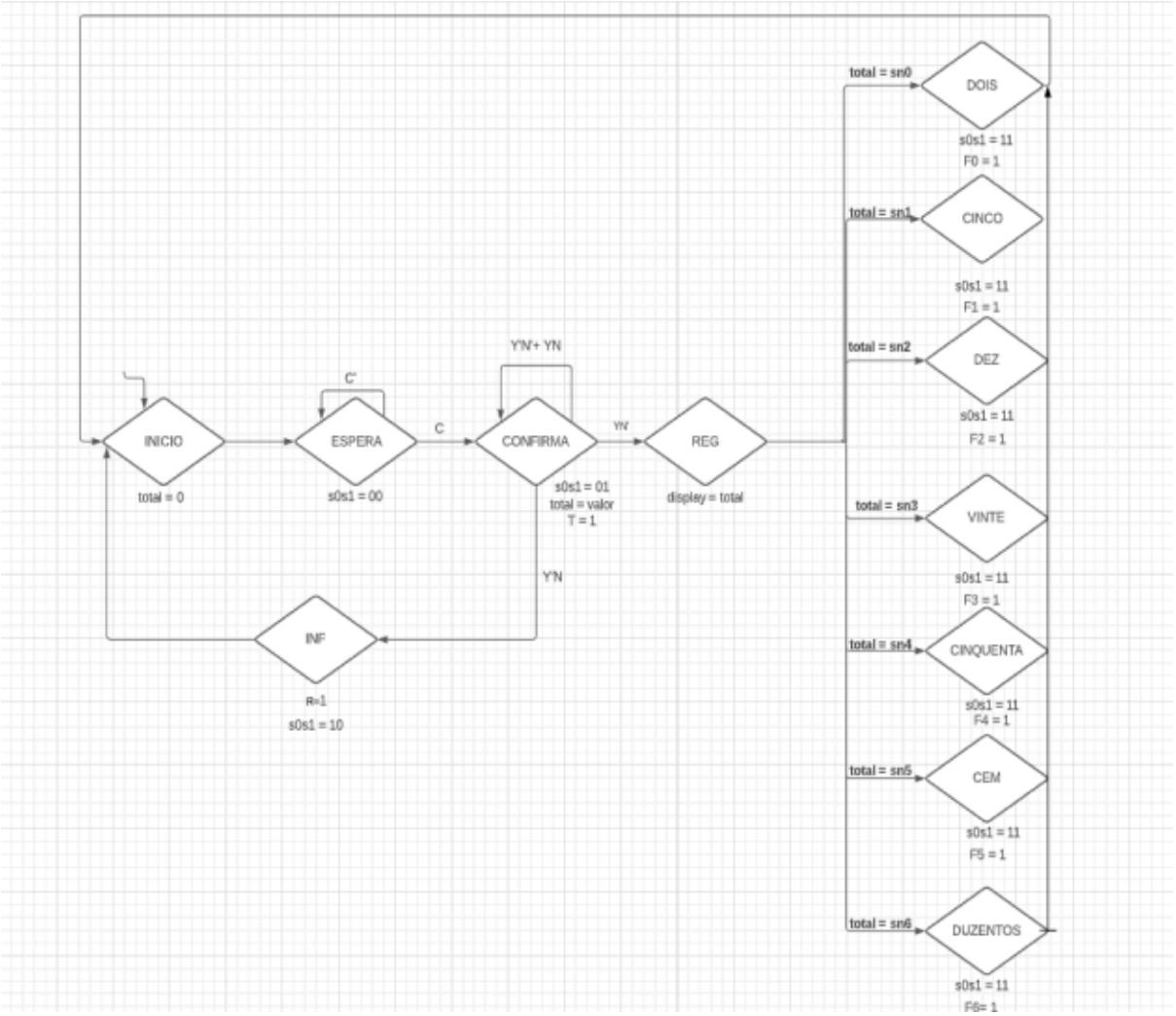
Saídas	Bits	Funcionalidades
Display	8	Valor da cédula inserida pelo usuário
Display2	8	Comando da mensagem escolhida para aparecer no display
R	1	Sinal que permite a devolução da cédula
T	1	Ativa o travamento da aleta
F0	1	Sinal que aciona a liberação do produto 0
F1	1	Sinal que aciona a liberação do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

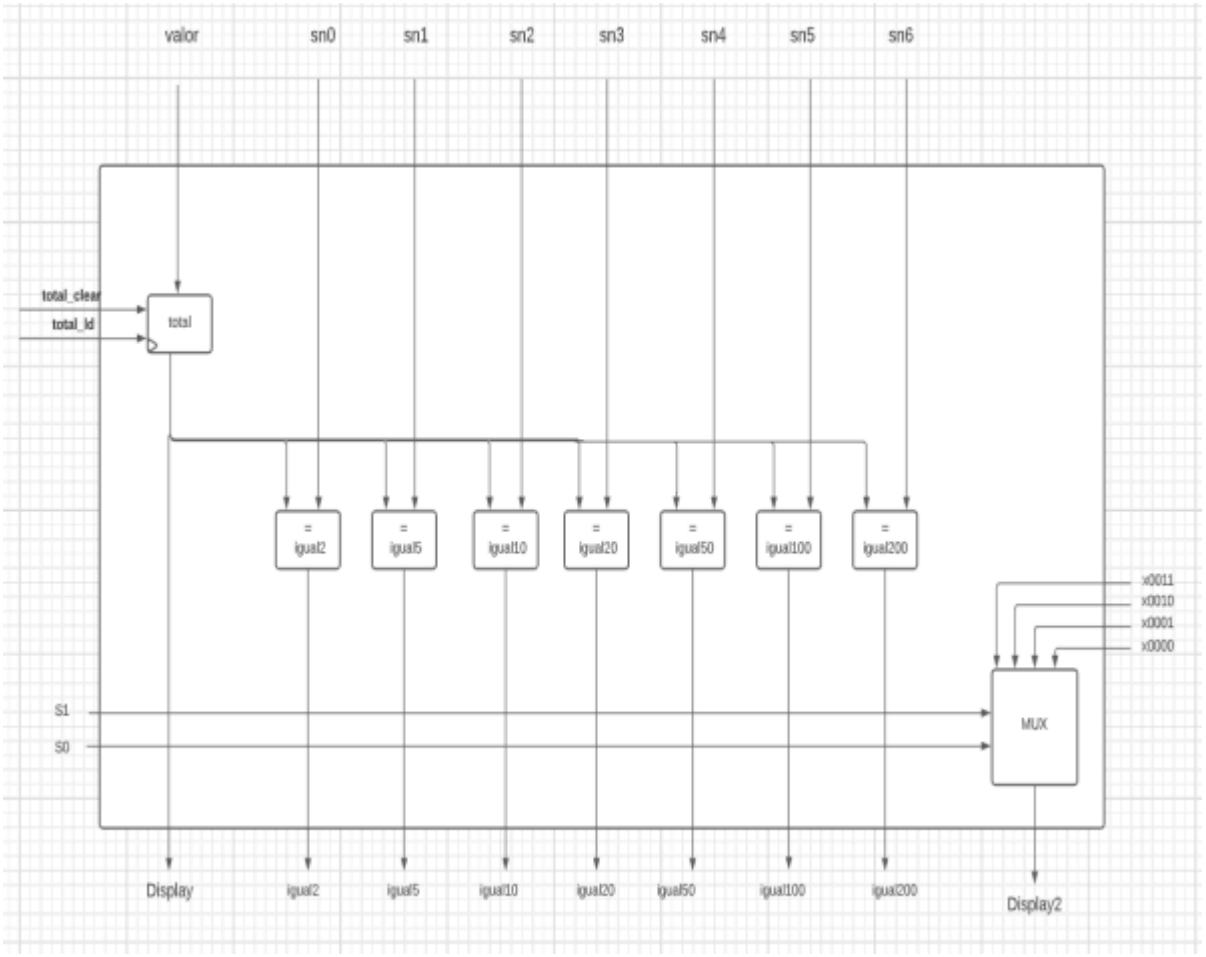
● Diagrama de Máquina de Estados de Alto Nível (diagrama conceitual)



● 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

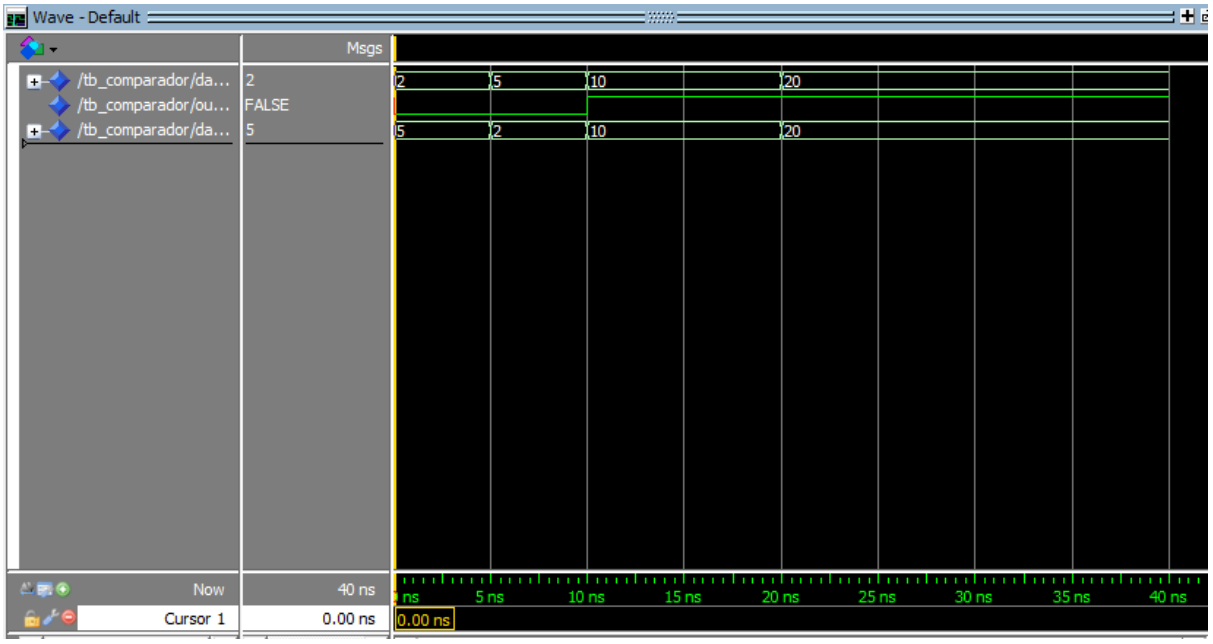
Código do comparador:

```
--comparador
library ieee;
use ieee.std_logic_1164.all;

entity comparador is
generic(w: natural :=8);
port(
      data1, data2 : in std_logic_vector(W-1 downto 0);
      output       : out boolean
      );
end comparador;

architecture arch of comparador is
begin
   output <= (data1 = data2);
end arch;
```

O componente possui uma porta de saída do tipo boolean chamada output, que resulta em true se todos os bits dos vetores data1 e data2 forem iguais, e false 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 : boolean;
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.

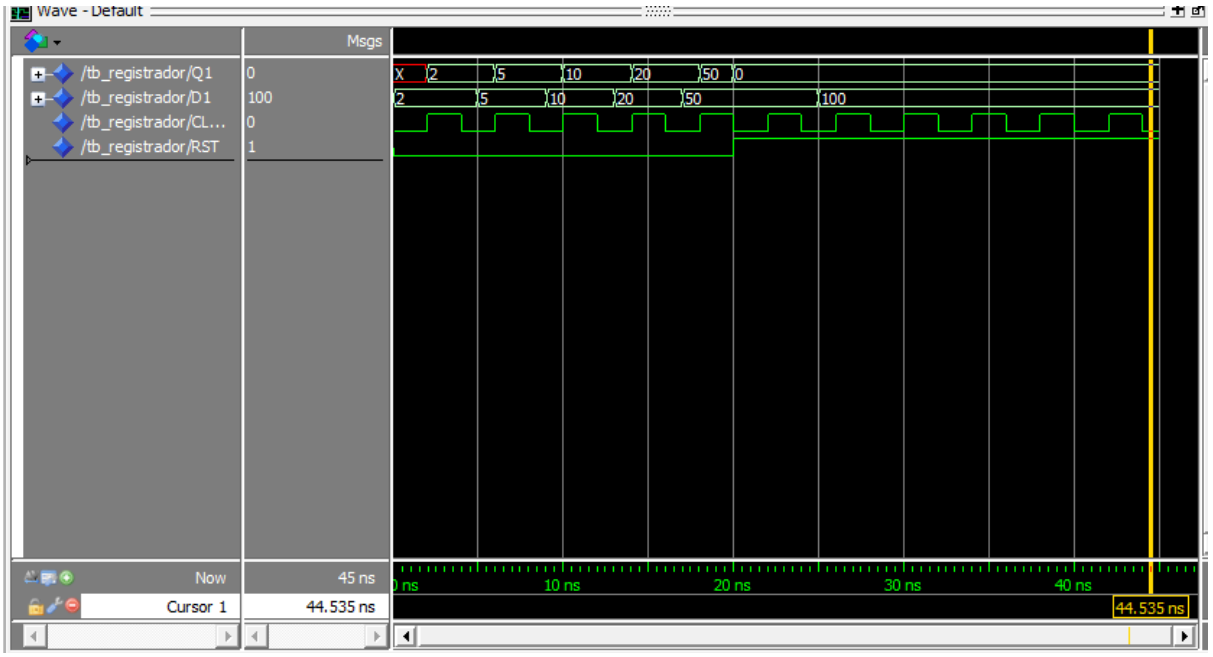
2.2 REGISTRADOR

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;

```



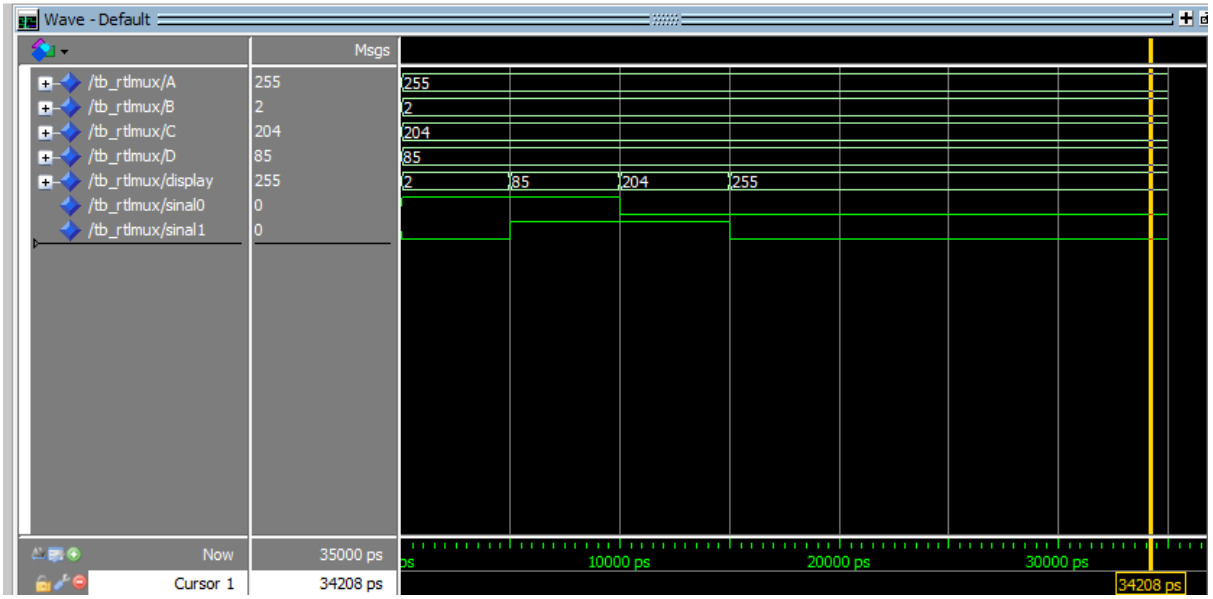
2.3 MUX

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity RTLMux is
generic(w: natural :=8);
port(
  a1      : in  std_logic_vector(w-1 downto 0);
  a2      : in  std_logic_vector(w-1 downto 0);
  a3      : in  std_logic_vector(w-1 downto 0);
  a4      : in  std_logic_vector(w-1 downto 0);
  s0     : in  std_logic;
  s1     : in  std_logic;
  Display2       : out std_logic_vector(w-1 downto 0));
end RTLMux;

architecture rtl of RTLMux is

signal sel : std_logic_vector(1 downto 0);

begin
sel <= (s1 & s0);
  with sel select
    Display2 <= a1 when "00",
       a2 when "01",
       a3 when "10",
       a4 when others;
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

```vhdl
entity tb_RTLMux is
end tb_RTLMux;

architecture teste of tb_RTLMux is

component RTLMux is
generic(w: natural :=8);
    port (
        a1 : in std_logic_vector(w-1 downto 0);
        a2 : in std_logic_vector(w-1 downto 0);
        a3 : in std_logic_vector(w-1 downto 0);
        a4 : in std_logic_vector(w-1 downto 0);
        s0 : in std_logic;
        s1 : in std_logic;
        Display2 : out std_logic_vector(w-1 downto 0)
    );
end component;

signal A, B, C, D, display: std_logic_vector(7 downto 0);
signal sinal0, sinal1: std_logic;
begin
    instancia_RTLMux: RTLMux port map(a1=>A, a2=>B, a3=>C, a4=>D, Display2=>display, s0=>sinal0, s1=>sinal1);
    A <= x"FF";
    B <= x"02";
    C <= x"CC";
    D <= x"55";
    sinal0 <= '1', '1' after 5 ns, '0' after 10 ns, '0' after 15 ns;
    sinal1 <= '0', '1' after 5 ns, '1' after 10 ns, '0' after 15 ns;
end teste;
```



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.

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