



**Universidade de Brasília**  
Departamento da Engenharia Elétrica  
Laboratório de Sistemas Digitais

**Relatório 07**  
Máquinas de estado Moore

José Antônio Alcântara da Silva de Andrade      Mat: 232013031

Professor:  
Eduardo B. R. F. Paiva  
Turma 08

Brasília, DF  
**3 de fevereiro de 2025**

# 1 Objetivos

1. Implementar uma máquina de estados síncrona do tipo Moore em VHDL e a simular no ModelSim.

## 2 Atividade

Essa sessão do laboratório tinha apenas uma tarefa: a implementação de uma moedeira como uma máquina de estado Moore. Ela deve receber moedas de 25 e 50 centavos, devolver troco, cancelar transações e fornecer um produto quando o total for de 1 real. Por requisito do exercício, a máquina deve possuir apenas uma entrada além do clock,  $A$ , a qual indica qual operação está sendo requisitada, e três saídas: uma que indica que o produto saiu, e duas outras que indicam qual moeda saiu pelo troco (25 ou 50 centavos).

### 2.1 Modelagem

Para essa atividade, deve-se primeiro construir o diagrama de estados que reina sob a máquina. Já que a máquina deve ser feita no formato Moore, deveram existir, no mínimo, nove estados, como será visto em seguida.

Necessita-se de um estado estático inicial em que a máquina permanece enquanto não há entradas: chama-se este de **ESPR**. Desse estado, há duas possibilidades, ou se insere uma moeda de 25 centavos, ou uma moeda de 50 centavos. Dois novos estados: **ST25** e **ST50**. A partir desses, a única possibilidade que não resulta em alguma saída é a inserção de 25 centavos no estado **ST50** ou 50 no estado **ST25**, chegando ao estado **ST75**.

Com esses três estados, cria-se três outros em que ocorre o cancelamento da transação e retorno das moedas: **TC25**, **TC50** e **TC75**. Todos decaem sem entrada para o estado **ESPR**.

Do estado **ST50** e **ST75**, pode-se atingir 1 real ao adicionar 50 e 25 centavos, respectivamente, chegando-se ao estado que dispensa o produto, **ST10**. Do **ST75** pode-se adicionar 50 centavos e chegar em 1,25 reais, o estado **ST12**. O estado **ST10** decai para **ESPR**, enquanto **ST12** primeiro decai a **TC25** antes de atingir **ESPR**.

Na máquina, os quatro valores de entrada  $A$  estão bem definidos:  $A = 00$  indica que não ocorreu entrada,  $A = 01$  indica que uma moeda de 25 foi inserida,  $A = 10$  indica que uma moeda de 50 foi inserida e  $A = 11$  indica que a operação foi cancelada.

O modelo que esse relatório implementa está desenhado na Figura 1.

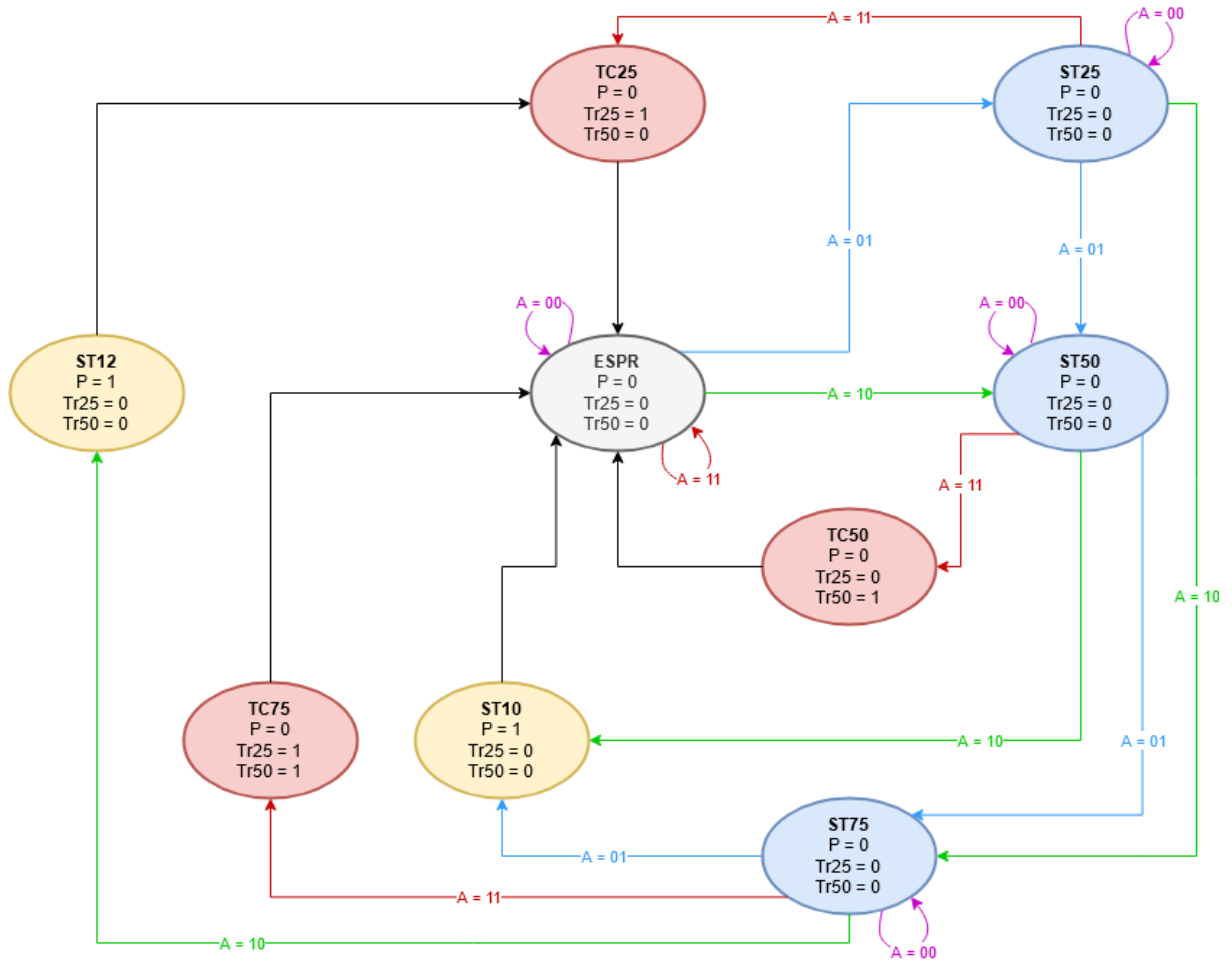
O diagrama da Figura 1 está dividido em três grupos de estado principais: coleta de moeda, entrega de produto e troco. Além destes, o estado central, **ESPR**, indica o estado inicial da máquina. Em azul, estão indicados os estados em que alguma moeda está contabilizada no sistema, mas ainda não se atingiu o total de 1 real. Em amarelo, os estados em que o valor contabilizado ultrapassa 1 real, e o produto é dispensado. Finalmente, em vermelho, os estados em que troco é dispensado.

Na Figura 1, as entradas são indicadas pelas coloração das setas. Roxo, azul, verde e vermelho, respectivamente. Setas pretas indicam que o valor de  $A$  é irrelevante, ou seja, essas transições ocorreram independente do valor da entrada.

### 2.2 Implementação

A moedeira a ser construída possui duas entradas e três saídas. Isto é: um clock, uma entrada de vetor indicando a operação atual, uma saída indicando a dispensa de produto e outras duas para a dispensa de troco. Define-se, então, a sua entidade, na Listagem 1.

**Figura 1:** Diagrama de estados da moedeira.



**Listagem 1:** Requisitos para a moedeira.

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity MOEDEIRA is
5     port (
6         A: in std_logic_vector(1 downto 0);
7         clock: in std_logic;
8         P, Tr25, Tr50: out std_logic
9     );
10 end MOEDEIRA;

```

Após isso, inicia-se a arquitetura do circuito, na Listagem 2. Como esse circuito é uma máquina de estados, usa-se de uma nova palavra chave do VHDL para facilitar sua programação: “*type is*”.

**Listagem 2:** Definição da arquitetura.

```

1 architecture MOEDEIRA_ARCH of MOEDEIRA is
2
3     type estado is (ESPR, ST25, ST50, ST75, ST10, ST12, TC75, TC50, TC25);
4
5     signal currState, nextState : estado;

```

Na linha 3 da Listagem 2 temos a criação de um novo tipo, “estado”, o qual tem como valores todos os nove possíveis estados que a máquina possa se encontrar. E, então, na linha 5, usamos desse estado para definir os dois principais sinais do sistema.

### Listagem 3: Processo síncrono.

```
1 begin
2
3     sync_proc: process(clock)
4     begin
5         if rising_edge(clock) then
6             currState <= nextState;
7         end if;
8     end process sync_proc;
```

Em seguida, dentro do processamento da arquitetura, cria-se dois processos. Primeiramente, o processo síncrono, definido na Listagem 3, e, em seguida, o processo combinatório, na Listagem 4.

Na Listagem 3, o processo é bem simples. Se o valor do clock se alterar (linha 3), checka-se se está em subida (linha 5), e então, se estiver, insere-se o valor do próximo estado no estado atual (linha 6).

### Listagem 4: Processo combinatório.

```
1     comb_proc: process(currState, A)
2     begin
3         case currState is
4             when ST25 =>
5                 P <= '0';
6                 Tr25 <= '0';
7                 Tr50 <= '0';
8                 case A is
9                     when "01" => nextState <= ST50;
10                    when "10" => nextState <= ST75;
11                    when "11" => nextState <= TC25;
12                    when others => nextState <= ST25;
13                end case;
14
15            when ST10 =>
16                P <= '1';
17                Tr25 <= '0';
18                Tr50 <= '0';
19                nextState <= ESPR;
20
21            -- ...
22        end case;
23    end process;
24 end MOEDEIRA_ARCH;
```

Na Listagem 4, o processo combinatório está descrito; contudo não em completude, apenas dois exemplos. Para ver todos os casos definidos, verifique a Listagem 6, no final deste documento. O processo combinatório é acionado diante da mudança de estado ou alteração da entrada *A* (linha 1 da Listagem 4).

Vemos aqui, dois exemplos, o primeiro para os estados em que a entrada *A* é relevante, e o segundo para os casos em que tal é irrelevante. Em ambos casos, checka-se qual é o valor atual no sinal do estado atual (linhas 3, 4 e 15), e, então, define-se as saídas do estado (linhas 5-7 e 16-18) de acordo com a Figura 1. Para o caso que a entrada *A* é relevante, realiza-se um *case statment* para verificar qual entrada foi realizada, e insere-se tal estado no sinal de novo estado (linhas 8-13). No caso em que a entrada é irrelevante, apenas insere-se o novo valor no sinal (linha 19).

Com isso, o circuito já está implementado, e seu código completo pode ser visualizado na Listagem 6 no final do documento. Contudo, para facilitar as simulações, constrói-se um Top Module e um Testbench para automaticamente realizar as alterações de estado.

O Top Module pode ser encontrado na Listagem 7, e o Testbench na Listagem 8, ambas no final do documento (omitidas devido sua extensão).

**Listagem 5:** Exemplo do testbench.

```
1 -- ST50 => ST10
2 A <= "10";
3 clock <= '0';
4 wait for 250 ns;
5 clock <= '1';
6 report "A = 10 | ST50 => ST10" severity NOTE;
7 wait for 250 ns;
8 assert (P = '1')    report "Falha." severity ERROR;
9 assert (Tr25 = '0') report "Falha." severity ERROR;
10 assert (Tr50 = '0') report "Falha." severity ERROR;
```

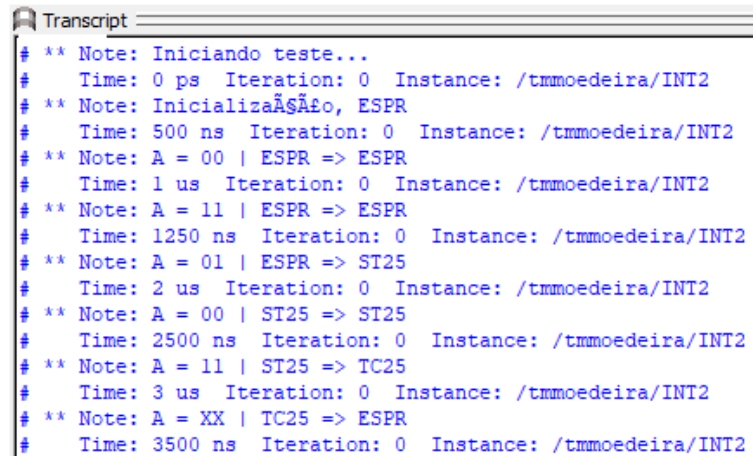
Os testes seguem o exemplo disponível na Listagem 5. Primeiramente, cita-se qual estado realiza-se a transição para (linha 1) e insere-se a entrada devida para tal (linha 2). Em seguida, realiza-se uma atualização do clock (linhas 3, 4, 5 e 7) com delays para que todas as variáveis e sinais sejam corretamente atribuídos. Por último, envia-se uma mensagem notificando da alteração de estado (linha 6), e checa-se se as saídas atuais são as devidamente estabelecidas na Figura 1 (linhas 8-10).

Uma observação a se fazer é que os testes são realizados de forma sequencial, ou seja, o próximo teste depende do sucesso do anterior. Dessa forma, é possível verificar se a máquina apresenta o comportamento requisitado pelo roteiro.

## 2.3 Simulação

A simulação se torna simples com a implementação do Top Module (Listagem 7) e do Testbench (Listagem 8). Basta apenas realizar uma simulação completa no ModelSim, e obtém-se o resultado, como visto da Figura 2 até a Figura 5. A figura de onda gerada pela simulação está disponível na Figura 6.

**Figura 2:** Parte 1 do console durante a simulação.



```
Transcript
# ** Note: Iniciando teste...
#   Time: 0 ps Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: Inicialização, ESPR
#   Time: 500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 00 | ESPR => ESPR
#   Time: 1 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 11 | ESPR => ESPR
#   Time: 1250 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ESPR => ST25
#   Time: 2 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 00 | ST25 => ST25
#   Time: 2500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 11 | ST25 => TC25
#   Time: 3 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | TC25 => ESPR
#   Time: 3500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
```

Pode-se observar que todos os caminhos descritos na Figura 1 são devidamente testados.

Na Figura 2, começa-se pelo estado ESPR, testando ambas situações em que permanece neste, em seguida adiciona-se 25 centavos. Permanece-se no estado ST25 e, então, requisita-se o cancelamento (troco) por TC25, finalmente retornando a ESPR.

Na Figura 3, adiciona-se duas moedas de 25 para verificar a funcionalidade do caminho ST25 para ST50. Checa-se a permanência no estado ST50, e então pede-se o troco por TC50, como anteriormente. Então, salta-se do estado ESPR para o estado ST50 colocando-se uma moeda de 50 centavos. Adiciona-se mais uma moeda, de 25 centavos, para atingir o estado ST75 por ST50. Checa-se a permanência em ST75.

**Figura 3:** Parte 2 do console durante a simulação.

```
Transcript
# ** Note: A = 01 | ESPR => ST25
#   Time: 4 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ST25 => ST50
#   Time: 4500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 00 | ST50 => ST50
#   Time: 5 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 11 | ST50 => TC50
#   Time: 5500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | TC50 => ESPR
#   Time: 6 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 10 | ESPR => ST50
#   Time: 6500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ST50 => ST75
#   Time: 7 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 00 | ST75 => ST75
#   Time: 7500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
```

**Figura 4:** Parte 3 do console durante a simulação.

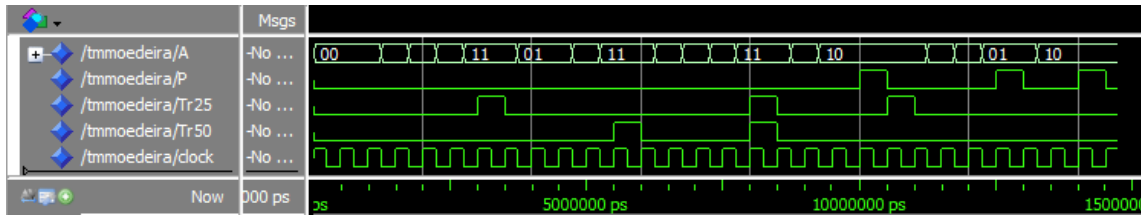
```
Transcript
# ** Note: A = 11 | ST75 => TC75
#   Time: 8 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | TC75 => ESPR
#   Time: 8500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ESPR => ST25
#   Time: 9 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 10 | ST25 => ST75
#   Time: 9500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 10 | ST75 => ST12
#   Time: 10 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | ST12 => TC25
#   Time: 10500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | TC25 => ESPR
#   Time: 11 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ESPR => ST25
#   Time: 11500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
```

**Figura 5:** Parte 4 do console durante a simulação.

```
Transcript
# ** Note: A = 10 | ST25 => ST75
#   Time: 12 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 01 | ST75 => ST10
#   Time: 12500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | ST10 => ESPR
#   Time: 13 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 10 | ESPR => ST50
#   Time: 13500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = 10 | ST50 => ST10
#   Time: 14 us Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: A = XX | ST10 => ESPR
#   Time: 14500 ns Iteration: 0 Instance: /tmmoeadeira/INT2
# ** Note: Teste finalizado.
#   Time: 14750 ns Iteration: 0 Instance: /tmmoeadeira/INT2
```

Na Figura 4, realiza-se o cancelamento da transação, passando-se pelo estado de troco TC75. Novamente em ESPR, direciona-se a ST75 ao colocar uma moeda de 25, ST25, e depois uma de 50. Adiciona-se mais uma de 50 para chegar a ST12, que naturalmente decai para TC25 e este para ESPR. Adiciona-se mais uma moeda de 25 centavos.

**Figura 6:** Figura de onda gerada após a simulação.



Na Figura 5, adiciona-se uma moeda de 50 centavos para atingir ST75, e deste uma de 25 para atingir ST10. ST10 naturalmente decai para ESPR, e, agora, apenas um trajeto resta: ST50 para ST10, o qual é realizado sem algum problema. A nota final confirma o término com sucesso da simulação.

## 2.4 Extra

### 2.4.1 Compilação

Como segue na Figura 7, nenhum dos códigos obteve erros de compilação.

**Figura 7:** Compilação dos códigos.

```
# Compile of main.vhd was successful.
# Compile of testbench.vhd was successful.
# Compile of topmodule.vhd was successful.
VSIM 24> piles, 0 failed with no errors.
```

### 2.4.2 Códigos Diversos

**Listagem 6:** Código final da moedeira.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity MOEDEIRA is
5     port (
6         A: in std_logic_vector(1 downto 0);
7         clock: in std_logic;
8         P, Tr25, Tr50: out std_logic
9     );
10 end MOEDEIRA;
11
12 architecture MOEDEIRA_ARCH of MOEDEIRA is
13
14     type estado is (ESPR, ST25, ST50, ST75, ST10, ST12, TC75, TC50, TC25);
15
16     signal currState, nextState : estado;
17
18 begin
19
20     sync_proc: process(clock)
21     begin
22         if rising_edge(clock) then
23             currState <= nextState;
24         end if;
25     end process sync_proc;
26
27     comb_proc: process(currState, A)
28     begin
29         case currState is
30             when ESPR =>
```

```

31         P <= '0';
32         Tr25 <= '0';
33         Tr50 <= '0';
34         case A is
35             when "01" => nextState <= ST25;
36             when "10" => nextState <= ST50;
37             when others => nextState <= ESPR;
38         end case;
39
40     when ST25 =>
41         P <= '0';
42         Tr25 <= '0';
43         Tr50 <= '0';
44         case A is
45             when "01" => nextState <= ST50;
46             when "10" => nextState <= ST75;
47             when "11" => nextState <= TC25;
48             when others => nextState <= ST25;
49         end case;
50
51     when ST50 =>
52         P <= '0';
53         Tr25 <= '0';
54         Tr50 <= '0';
55         case A is
56             when "01" => nextState <= ST75;
57             when "10" => nextState <= ST10;
58             when "11" => nextState <= TC50;
59             when others => nextState <= ST50;
60         end case;
61
62     when ST75 =>
63         P <= '0';
64         Tr25 <= '0';
65         Tr50 <= '0';
66         case A is
67             when "01" => nextState <= ST10;
68             when "10" => nextState <= ST12;
69             when "11" => nextState <= TC75;
70             when others => nextState <= ST75;
71         end case;
72
73     when ST10 =>
74         P <= '1';
75         Tr25 <= '0';
76         Tr50 <= '0';
77         nextState <= ESPR;
78
79     when ST12 =>
80         P <= '1';
81         Tr25 <= '0';
82         Tr50 <= '0';
83         nextState <= TC25;
84
85     when TC75 =>
86         P <= '0';
87         Tr25 <= '1';
88         Tr50 <= '1';
89         nextState <= ESPR;
90
91     when TC50 =>
92         P <= '0';
93         Tr25 <= '0';
94         Tr50 <= '1';
95         nextState <= ESPR;
96
97     when TC25 =>
98         P <= '0';
99         Tr25 <= '1';
100        Tr50 <= '0';
101        nextState <= ESPR;
102
103     when others =>

```



```

104         P <= '0';
105         Tr25 <= '0';
106         Tr50 <= '0';
107         nextState <= ESPR;
108     end case;
109 end process;
110 end MOEDEIRA_ARCH;

```

**Listagem 7:** Top Module da moedeira.

```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity TMMOEDEIRA is
5  end TMMOEDEIRA;
6
7  architecture TMMOEDEIRA_ARCH of TMMOEDEIRA is
8      component MOEDEIRA is
9          port (
10             A: in std_logic_vector(1 downto 0);
11             clock: in std_logic;
12             P, Tr25, Tr50: out std_logic
13          );
14      end component;
15
16      component TBMOEDEIRA is
17          port (
18             P, Tr25, Tr50: in std_logic;
19             A: out std_logic_vector(1 downto 0);
20             clock: out std_logic
21          );
22      end component;
23
24      signal A : std_logic_vector(1 downto 0);
25      signal P, Tr25, Tr50, clock : std_logic;
26
27  begin
28      INT1 : MOEDEIRA port map(A, clock, P, Tr25, Tr50);
29      INT2 : TBMOEDEIRA port map(P, Tr25, Tr50, A, clock);
30  end TMMOEDEIRA_ARCH;

```

**Listagem 8:** Testbench da moedeira.

```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity TBMOEDEIRA is
5      port (
6          P, Tr25, Tr50: in std_logic;
7          A: out std_logic_vector(1 downto 0);
8          clock: out std_logic
9      );
10 end TBMOEDEIRA;
11
12 architecture TBMOEDEIRA_ARCH of TBMOEDEIRA is
13 begin
14     process
15     begin
16         report "Iniciando teste..." severity NOTE;
17
18         -- SETUP
19         A <= "00";
20         clock <= '1';
21         wait for 250 ns;
22         clock <= '0';
23         wait for 250 ns;
24         clock <= '1';
25         report "Inicializacao, ESPR" severity NOTE;
26         wait for 250 ns;
27         assert (P = '0')      report "Falha." severity ERROR;
28         assert (Tr25 = '0')  report "Falha." severity ERROR;
29         assert (Tr50 = '0')  report "Falha." severity ERROR;
30

```

```

31  -- ESPR => ESPR
32  A <= "00";
33  clock <= '0';
34  wait for 250 ns;
35  clock <= '1';
36  report "A = 00 | ESPR => ESPR" severity NOTE;
37  wait for 250 ns;
38  assert (P = '0')      report "Falha." severity ERROR;
39  assert (Tr25 = '0')   report "Falha." severity ERROR;
40  assert (Tr50 = '0')   report "Falha." severity ERROR;
41  A <= "11";
42  clock <= '0';
43  report "A = 11 | ESPR => ESPR" severity NOTE;
44  wait for 250 ns;
45  clock <= '1';
46  wait for 250 ns;
47  assert (P = '0')      report "Falha." severity ERROR;
48  assert (Tr25 = '0')   report "Falha." severity ERROR;
49  assert (Tr50 = '0')   report "Falha." severity ERROR;
50
51  -- ESPR => ST25
52  A <= "01";
53  clock <= '0';
54  wait for 250 ns;
55  clock <= '1';
56  report "A = 01 | ESPR => ST25" severity NOTE;
57  wait for 250 ns;
58  assert (P = '0')      report "Falha." severity ERROR;
59  assert (Tr25 = '0')   report "Falha." severity ERROR;
60  assert (Tr50 = '0')   report "Falha." severity ERROR;
61
62  -- ST25 => ST25
63  A <= "00";
64  clock <= '0';
65  wait for 250 ns;
66  clock <= '1';
67  report "A = 00 | ST25 => ST25" severity NOTE;
68  wait for 250 ns;
69  assert (P = '0')      report "Falha." severity ERROR;
70  assert (Tr25 = '0')   report "Falha." severity ERROR;
71  assert (Tr50 = '0')   report "Falha." severity ERROR;
72
73  -- ST25 => TC25
74  A <= "11";
75  clock <= '0';
76  wait for 250 ns;
77  clock <= '1';
78  report "A = 11 | ST25 => TC25" severity NOTE;
79  wait for 250 ns;
80  assert (P = '0')      report "Falha." severity ERROR;
81  assert (Tr25 = '1')   report "Falha." severity ERROR;
82  assert (Tr50 = '0')   report "Falha." severity ERROR;
83
84  -- TC25 => ESPR
85  -- INDEPENDENTE DE A
86  clock <= '0';
87  wait for 250 ns;
88  clock <= '1';
89  report "A = XX | TC25 => ESPR" severity NOTE;
90  wait for 250 ns;
91  assert (P = '0')      report "Falha." severity ERROR;
92  assert (Tr25 = '0')   report "Falha." severity ERROR;
93  assert (Tr50 = '0')   report "Falha." severity ERROR;
94
95  -- ESPR => ST25
96  A <= "01";
97  clock <= '0';
98  wait for 250 ns;
99  clock <= '1';
100 report "A = 01 | ESPR => ST25" severity NOTE;
101 wait for 250 ns;
102 assert (P = '0')      report "Falha." severity ERROR;
103 assert (Tr25 = '0')   report "Falha." severity ERROR;

```

```

104     assert (Tr50 = '0') report "Falha." severity ERROR;
105
106     -- ST25 => ST50
107     A <= "01";
108     clock <= '0';
109     wait for 250 ns;
110     clock <= '1';
111     report "A = 01 | ST25 => ST50" severity NOTE;
112     wait for 250 ns;
113     assert (P = '0')      report "Falha." severity ERROR;
114     assert (Tr25 = '0') report "Falha." severity ERROR;
115     assert (Tr50 = '0') report "Falha." severity ERROR;
116
117     -- ST50 => ST50
118     A <= "00";
119     clock <= '0';
120     wait for 250 ns;
121     clock <= '1';
122     report "A = 00 | ST50 => ST50" severity NOTE;
123     wait for 250 ns;
124     assert (P = '0')      report "Falha." severity ERROR;
125     assert (Tr25 = '0') report "Falha." severity ERROR;
126     assert (Tr50 = '0') report "Falha." severity ERROR;
127
128     -- ST50 => TC50
129     A <= "11";
130     clock <= '0';
131     wait for 250 ns;
132     clock <= '1';
133     report "A = 11 | ST50 => TC50" severity NOTE;
134     wait for 250 ns;
135     assert (P = '0')      report "Falha." severity ERROR;
136     assert (Tr25 = '0') report "Falha." severity ERROR;
137     assert (Tr50 = '1') report "Falha." severity ERROR;
138
139     -- TC50 => ESPR
140     -- INDEPENDENTE DE A
141     clock <= '0';
142     wait for 250 ns;
143     clock <= '1';
144     report "A = XX | TC50 => ESPR" severity NOTE;
145     wait for 250 ns;
146     assert (P = '0')      report "Falha." severity ERROR;
147     assert (Tr25 = '0') report "Falha." severity ERROR;
148     assert (Tr50 = '0') report "Falha." severity ERROR;
149
150     -- ESPR => ST50
151     A <= "10";
152     clock <= '0';
153     wait for 250 ns;
154     clock <= '1';
155     report "A = 10 | ESPR => ST50" severity NOTE;
156     wait for 250 ns;
157     assert (P = '0')      report "Falha." severity ERROR;
158     assert (Tr25 = '0') report "Falha." severity ERROR;
159     assert (Tr50 = '0') report "Falha." severity ERROR;
160
161     -- ST50 => ST75
162     A <= "01";
163     clock <= '0';
164     wait for 250 ns;
165     clock <= '1';
166     report "A = 01 | ST50 => ST75" severity NOTE;
167     wait for 250 ns;
168     assert (P = '0')      report "Falha." severity ERROR;
169     assert (Tr25 = '0') report "Falha." severity ERROR;
170     assert (Tr50 = '0') report "Falha." severity ERROR;
171
172     -- ST75 => ST75
173     A <= "00";
174     clock <= '0';
175     wait for 250 ns;
176     clock <= '1';

```

```

177     report "A = 00 | ST75 => ST75" severity NOTE;
178     wait for 250 ns;
179     assert (P = '0')      report "Falha." severity ERROR;
180     assert (Tr25 = '0')   report "Falha." severity ERROR;
181     assert (Tr50 = '0')   report "Falha." severity ERROR;
182
183     -- ST75 => TC75
184     A <= "11";
185     clock <= '0';
186     wait for 250 ns;
187     clock <= '1';
188     report "A = 11 | ST75 => TC75" severity NOTE;
189     wait for 250 ns;
190     assert (P = '0')      report "Falha." severity ERROR;
191     assert (Tr25 = '1')   report "Falha." severity ERROR;
192     assert (Tr50 = '1')   report "Falha." severity ERROR;
193
194     -- TC75 => ESPR
195     -- INDEPENDENTE DE A
196     clock <= '0';
197     wait for 250 ns;
198     clock <= '1';
199     report "A = XX | TC75 => ESPR" severity NOTE;
200     wait for 250 ns;
201     assert (P = '0')      report "Falha." severity ERROR;
202     assert (Tr25 = '0')   report "Falha." severity ERROR;
203     assert (Tr50 = '0')   report "Falha." severity ERROR;
204
205     -- ESPR => ST25
206     A <= "01";
207     clock <= '0';
208     wait for 250 ns;
209     clock <= '1';
210     report "A = 01 | ESPR => ST25" severity NOTE;
211     wait for 250 ns;
212     assert (P = '0')      report "Falha." severity ERROR;
213     assert (Tr25 = '0')   report "Falha." severity ERROR;
214     assert (Tr50 = '0')   report "Falha." severity ERROR;
215
216     -- ST25 => ST75
217     A <= "10";
218     clock <= '0';
219     wait for 250 ns;
220     clock <= '1';
221     report "A = 10 | ST25 => ST75" severity NOTE;
222     wait for 250 ns;
223     assert (P = '0')      report "Falha." severity ERROR;
224     assert (Tr25 = '0')   report "Falha." severity ERROR;
225     assert (Tr50 = '0')   report "Falha." severity ERROR;
226
227     -- ST75 => ST12
228     A <= "10";
229     clock <= '0';
230     wait for 250 ns;
231     clock <= '1';
232     report "A = 10 | ST75 => ST12" severity NOTE;
233     wait for 250 ns;
234     assert (P = '1')      report "Falha." severity ERROR;
235     assert (Tr25 = '0')   report "Falha." severity ERROR;
236     assert (Tr50 = '0')   report "Falha." severity ERROR;
237
238     -- ST12 => TC25
239     -- INDEPENDENTE DE A
240     clock <= '0';
241     wait for 250 ns;
242     clock <= '1';
243     report "A = XX | ST12 => TC25" severity NOTE;
244     wait for 250 ns;
245     assert (P = '0')      report "Falha." severity ERROR;
246     assert (Tr25 = '1')   report "Falha." severity ERROR;
247     assert (Tr50 = '0')   report "Falha." severity ERROR;
248
249     -- TC25 => ESPR

```

```

250 -- INDEPENDENTE DE A
251 clock <= '0';
252 wait for 250 ns;
253 clock <= '1';
254 report "A = XX | TC25 => ESPR" severity NOTE;
255 wait for 250 ns;
256 assert (P = '0') report "Falha." severity ERROR;
257 assert (Tr25 = '0') report "Falha." severity ERROR;
258 assert (Tr50 = '0') report "Falha." severity ERROR;
259
260 -- ESPR => ST25
261 A <= "01";
262 clock <= '0';
263 wait for 250 ns;
264 clock <= '1';
265 report "A = 01 | ESPR => ST25" severity NOTE;
266 wait for 250 ns;
267 assert (P = '0') report "Falha." severity ERROR;
268 assert (Tr25 = '0') report "Falha." severity ERROR;
269 assert (Tr50 = '0') report "Falha." severity ERROR;
270
271 -- ST25 => ST75
272 A <= "10";
273 clock <= '0';
274 wait for 250 ns;
275 clock <= '1';
276 report "A = 10 | ST25 => ST75" severity NOTE;
277 wait for 250 ns;
278 assert (P = '0') report "Falha." severity ERROR;
279 assert (Tr25 = '0') report "Falha." severity ERROR;
280 assert (Tr50 = '0') report "Falha." severity ERROR;
281
282 -- ST75 => ST10
283 A <= "01";
284 clock <= '0';
285 wait for 250 ns;
286 clock <= '1';
287 report "A = 01 | ST75 => ST10" severity NOTE;
288 wait for 250 ns;
289 assert (P = '1') report "Falha." severity ERROR;
290 assert (Tr25 = '0') report "Falha." severity ERROR;
291 assert (Tr50 = '0') report "Falha." severity ERROR;
292
293 -- ST10 => ESPR
294 -- INDEPENDENTE DE A
295 clock <= '0';
296 wait for 250 ns;
297 clock <= '1';
298 report "A = XX | ST10 => ESPR" severity NOTE;
299 wait for 250 ns;
300 assert (P = '0') report "Falha." severity ERROR;
301 assert (Tr25 = '0') report "Falha." severity ERROR;
302 assert (Tr50 = '0') report "Falha." severity ERROR;
303
304 -- ESPR => ST50
305 A <= "10";
306 clock <= '0';
307 wait for 250 ns;
308 clock <= '1';
309 report "A = 10 | ESPR => ST50" severity NOTE;
310 wait for 250 ns;
311 assert (P = '0') report "Falha." severity ERROR;
312 assert (Tr25 = '0') report "Falha." severity ERROR;
313 assert (Tr50 = '0') report "Falha." severity ERROR;
314
315 -- ST50 => ST10
316 A <= "10";
317 clock <= '0';
318 wait for 250 ns;
319 clock <= '1';
320 report "A = 10 | ST50 => ST10" severity NOTE;
321 wait for 250 ns;
322 assert (P = '1') report "Falha." severity ERROR;

```

```

323     assert (Tr25 = '0') report "Falha." severity ERROR;
324     assert (Tr50 = '0') report "Falha." severity ERROR;
325
326     -- ST10 => ESPR
327     -- INDEPENDENTE DE A
328     clock <= '0';
329     wait for 250 ns;
330     clock <= '1';
331     report "A = XX | ST10 => ESPR" severity NOTE;
332     wait for 250 ns;
333     assert (P = '0')      report "Falha." severity ERROR;
334     assert (Tr25 = '0') report "Falha." severity ERROR;
335     assert (Tr50 = '0') report "Falha." severity ERROR;
336
337     report "Teste finalizado." severity NOTE;
338     wait;
339 end process;
340 end TBMOEDEIRA_ARCH;

```