Lógica digital

1.

a. 16 linhas de entrada.

b. 5 linhas de entrada.

c. 8 entradas.

2.

a. y <= (d0 AND NOT c) OR (d1 AND c);

b. y <= (d0 AND NOT c1 AND NOT c2) OR (d1 AND NOT c1 AND c2) OR (d2 AND c1 AND NOT c2) OR (d3 AND c1 AND c2);

3.
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity interface is
   port (
      WXYZ : in  std_logic_vector(3 downto 0);
      letra : out std_logic_vector(6 downto 0)
   );
end entity interface;

architecture comportamental of interface is
begin
   process(WXYZ)
   begin
      case WXYZ is
         when "0001" => letra <= "0000001";
         when "0010" => letra <= "1001111";
         when "0011" => letra <= "0011001";
         when "0100" => letra <= "0110010";
```

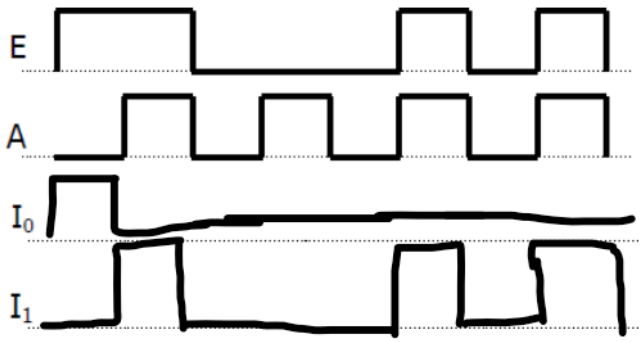
```
            when "0101" => letra <= "1001010";
            when "0110" => letra <= "0110001";
            when "0111" => letra <= "0100001";
            when "1000" => letra <= "1001110";
            when "1001" => letra <= "1111110";
            when "1010" => letra <= "1111100";
            when others => letra <= "1111111";
        end case;
    end process;
end architecture comportamental;
```

4.

f <= (b and not a and not (a or b)) or (c and not a and (a or b)) or (not c and a and not (a or b)) or (d and a and (a or b));

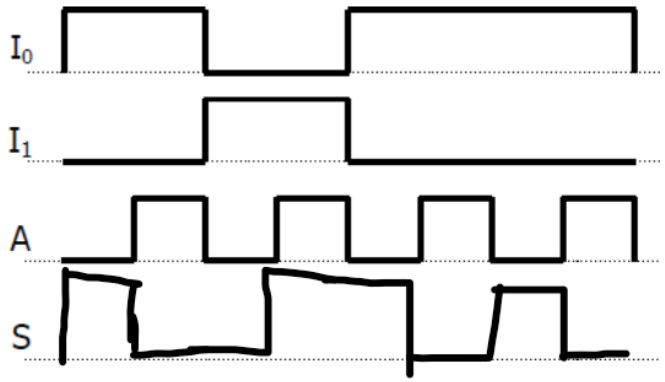
5.

5. Esboce os sinais de informação (I_0 e I_1) de um sistema demultiplex para os sinais de entrada (E) e de seleção (A) abaixo.



6.

6. A figura abaixo representa os sinais de informação de entrada e de seleção de um multiplex de 2 canais. Esboce o sinal de saída (S) multiplexado.



7.

Um codificador converte dados de um formato para outro, enquanto um decodificador faz o oposto, restaurando os dados ao seu formato original. Eles trabalham juntos em sistemas de comunicação, compactação de dados e outras aplicações para codificar e decodificar informações de forma eficiente e precisa. Já Um codificador binário converte dados em uma sequência de bits, enquanto um decodificador binário faz o inverso, convertendo a sequência de bits de volta aos dados originais. Juntos, eles são fundamentais para sistemas digitais de comunicação e armazenamento de informações.

8.

a. codificador

b, codificador.

c. decodificador.

d. decodificador.

9.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity Codificador_4_para_2 is

    port (
```

```
      A : in  std_logic_vector(3 downto 0);
      Y : out std_logic_vector(1 downto 0)
   );
end entity Codificador_4_para_2;

architecture estrutural of Codificador_4_para_2 is
begin

   Y(0) <= A(3) and (not (A(2) or A(1) or A(0)));

   Y(1) <= (A(2) and A(1)) or (A(2) and A(0)) or (A(1) and A(0));

end architecture estrutural;
```

10. 6 entradas e 64 saídas.

11.

ESTRUTURAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Demux_1_para_8_Estrutural is
   port (
      D : in  std_logic;
      S : in  std_logic_vector(2 downto 0);
      Y : out std_logic_vector(7 downto 0)
   );
end entity Demux_1_para_8_Estrutural;

architecture estrutural of Demux_1_para_8_Estrutural is
```

```
    component AND_GATE is
        port (
            A, B : in  std_logic;
            Y    : out std_logic
        );
    end component;

begin
    process(S)
    begin
        case S is
            when "000" =>
                Y <= (D, '0', '0', '0', '0', '0', '0', '0');
            when "001" =>
                Y <= ('0', D, '0', '0', '0', '0', '0', '0');
            when "010" =>
                Y <= ('0', '0', D, '0', '0', '0', '0', '0');
            when "011" =>
                Y <= ('0', '0', '0', D, '0', '0', '0', '0');
            when "100" =>
                Y <= ('0', '0', '0', '0', D, '0', '0', '0');
            when "101" =>
                Y <= ('0', '0', '0', '0', '0', D, '0', '0');
            when "110" =>
                Y <= ('0', '0', '0', '0', '0', '0', D, '0');
            when "111" =>
                Y <= ('0', '0', '0', '0', '0', '0', '0', D');
            when others =>
                Y <= (others => '0');
        end case;
    end process;
```

```
end architecture estrutural;
```

COMPORTAMENTAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Demux_1_para_8_Comportamental is
   port (
      D : in  std_logic;
      S : in  std_logic_vector(2 downto 0);
      Y : out std_logic_vector(7 downto 0)
   );
end entity Demux_1_para_8_Comportamental;

architecture comportamental of Demux_1_para_8_Comportamental is
begin
   process(D, S)
   begin
      case S is
         when "000" =>
            Y <= (D, '0', '0', '0', '0', '0', '0', '0');
         when "001" =>
            Y <= ('0', D, '0', '0', '0', '0', '0', '0');
         when "010" =>
            Y <= ('0', '0', D, '0', '0', '0', '0', '0');
         when "011" =>
            Y <= ('0', '0', '0', D, '0', '0', '0', '0');
         when "100" =>
            Y <= ('0', '0', '0', '0', D, '0', '0', '0');
         when "101" =>
            Y <= ('0', '0', '0', '0', '0', D, '0', '0');
```

```
        when "110" =>
            Y <= ('0', '0', '0', '0', '0', '0', D, '0');
        when "111" =>
            Y <= ('0', '0', '0', '0', '0', '0', '0', D');
        when others =>
            Y <= (others => '0');
    end case;
  end process;
end architecture comportamental;
```

12.

ESTRUTURAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexador_8_para_1 is
  port (
    Entradas : in  std_logic_vector(7 downto 0);  -- 8 entradas
    Selecao  : in  std_logic_vector(2 downto 0);  -- 3 bits de seleção
    Saida    : out std_logic                      -- Saída única
  );
end entity Multiplexador_8_para_1;

architecture estrutural of Multiplexador_8_para_1 is
begin
  with Selecao select
    Saida <= Entradas(0) when "000",
             Entradas(1) when "001",
             Entradas(2) when "010",
             Entradas(3) when "011",
             Entradas(4) when "100",
```

```
            Entradas(5) when "101",
            Entradas(6) when "110",
            Entradas(7) when others;
end architecture estrutural;
```

COMPORTAMENTAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexador_8_para_1 is
   port (
      Entradas : in  std_logic_vector(7 downto 0);  -- 8 entradas
      Selecao  : in  std_logic_vector(2 downto 0);  -- 3 bits de seleção
      Saida    : out std_logic                  -- Saída única
   );
end entity Multiplexador_8_para_1;

architecture comportamental of Multiplexador_8_para_1 is
begin
   process(Selecao)
   begin
      case Selecao is
         when "000" => Saida <= Entradas(0);
         when "001" => Saida <= Entradas(1);
         when "010" => Saida <= Entradas(2);
         when "011" => Saida <= Entradas(3);
         when "100" => Saida <= Entradas(4);
         when "101" => Saida <= Entradas(5);
         when "110" => Saida <= Entradas(6);
         when others => Saida <= Entradas(7);
      end case;
```

```
    end process;
end architecture comportamental;
```

13.

Nesse código temos 2 vetores como entradas, A e B, e 3 saídas que dependem do comportamento de B, caso a entrada A seja igual à B, AeqB será 1 e as outras 0, caso seja maior, AgtB será 1 e as outras 0, e se for menor AltB será 1 e as outras 0.

a. AeqB = 0
   AgtB = 0
   AltB = 1
b. AeqB = 1
   AgtB = 0
   AltB = 0
c. AeqB = 0
   AgtB = 1
   AltB = 0
d. AeqB = 0
   AgtB = 1
   AltB = 0

14.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Separador_Porcas_Parafusos is
    port (
        Esteira : in  std_logic_vector(1 downto 0); -- Entradas da esteira (Porcas, Parafusos)
        Caixa_A : out std_logic;
        Caixa_B : out std_logic;
        Caixa_C : out std_logic
    );
end entity Separador_Porcas_Parafusos;

architecture comportamental of Separador_Porcas_Parafusos is
begin
    process(Esteira)
```

```
    begin
        case Esteira is
            when "00" => -- Somente porcas
                Caixa_A <= '1';
                Caixa_B <= '0';
                Caixa_C <= '0';
            when "01" => -- Somente parafusos
                Caixa_A <= '0';
                Caixa_B <= '1';
                Caixa_C <= '0';
            when "10" => -- Porcas e parafusos
                Caixa_A <= '0';
                Caixa_B <= '0';
                Caixa_C <= '1';
            when others => -- Nenhuma peça
                Caixa_A <= '0';
                Caixa_B <= '0';
                Caixa_C <= '0';
        end case;
    end process;
end architecture comportamental;
```