Universidade de Brasília
Faculdade de Tecnologia
Laboratório de Sistemas Digitais

Relatório 03

Carla de Araujo Clementino Ribeiro Mat:180030736

Professor:
Guilherme de Sousa Torres

1 Objetivos

Implementar um multiplexar 8 para 1 e um decodificar de 4 para 16 bits, utilizando atribuições condicionais e a atribuições seletivas na linguagem de descrição de hardware VHDL.

2 Questões Propostas

1. Utilizando atribuições condicionais (when-else), escrever em VHDL e implementar em FPGA uma entidade que descreva um multiplexador 8 para 1. Essa entidade deve ter dois vetores de entrada (S com 3 bits e D com 8 bits) e um bit de saída (Y). A tabela verdade do multiplexador é apresentada abaixo. Respeitando a ordem de significância, associe cada um dos bits de D a diferentes chaves (SW0 a SW7), os bits de S a diferentes botões (BTN0 a BTN2) e o bit de saída a um dos LEDs (LD0 a LD7).

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Tabela 1: Tabela verdade do multiplexador de 8 para 1.

Considerando a questão proposta, primeiramente, foi implementado um componente com 2 entradas de 3 e 8 bits e uma saída de 1 bits, que realiza a operação proposta de acordo com a tabela verdade fornecida.

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

entity mux_8 is
    Port (
        S : in STD_LOGIC_VECTOR (2 downto 0);
        D : in STD_LOGIC_VECTOR (7 downto 0);
        Y : out STD_LOGIC
    );
end mux_8;

architecture Behavioral of mux_8 is

signal entrada: STD_LOGIC_VECTOR(2 downto 0);

begin

    entrada <= S(2) & S(1) & S(0);

        Y <= D(0) when (entrada = "000") else
             D(1) when (entrada = "001") else
             D(2) when (entrada = "010") else
             D(3) when (entrada = "011") else
             D(4) when (entrada = "100") else
             D(5) when (entrada = "101") else
             D(6) when (entrada = "110") else
             D(7);

end Behavioral;

```

Figura 1: Multiplexador 8 para 1. Fonte: Autor.

Em seguida, criamos uma entidade para que fosse possível realizar simulações do nosso circuito por meio de sinais controlados:

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

entity Master is
    Port (
        SW: in STD_LOGIC_VECTOR (7 downto 0);
        BTN: in STD_LOGIC_VECTOR (2 downto 0);
        LD: out STD_LOGIC
      );
end Master;

architecture Behavioral of Master is

signal u0: STD_LOGIC_VECTOR(2 downto 0);
signal u1: STD_LOGIC_VECTOR(7 downto 0);
signal u2: STD_LOGIC;

component mux_8
    Port (
        S : in STD_LOGIC_VECTOR (2 downto 0);
        D : in STD_LOGIC_VECTOR (7 downto 0);
        Y : out STD_LOGIC
      );
end component;

begin

u0 <= BTN;
u1 <= SW;

mux8: mux_8 port map(S=> u0, D => u1, Y => u2);

LD <= u2;

end Behavioral;
```

Figura 2: Entidade - Multiplexador 8 para 1. Fonte: Autor.

Cada um dos bits de entrada são associados a uma chave (sw0 a sw7) e o bit de saída a um LED, como especificado na questão.

Por fim, foi criado um aquivo para simular todas as possíveis entradas do circuito:

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

entity test_bench is
end test_bench;

architecture Behavioral of test_bench is
    signal U0 : STD_LOGIC_VECTOR(7 downto 0) := "00000000";
    signal U1 : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal Y : STD_LOGIC;

    component master is
        Port (
          SW: in STD_LOGIC_VECTOR (7 downto 0);
          BTN: in STD_LOGIC_VECTOR (2 downto 0);
          LD: out STD_LOGIC
        );
    end component;
begin

UUT: master port map (
    SW => U0,
    BTN => U1,
    LD => Y
);

U0(0) <= not U0(0) after 2ns;
U0(1) <= not U0(1) after 4ns;
U0(2) <= not U0(2) after 8ns;
U0(3) <= not U0(3) after 16ns;
U0(4) <= not U0(4) after 32ns;
U0(5) <= not U0(5) after 64ns;
U0(6) <= not U0(6) after 128ns;
U0(7) <= not U0(7) after 256ns;

U1(0) <= not U1(0) after 512ns;
U1(1) <= not U1(1) after 1024ns;
U1(2) <= not U1(2) after 2048ns;

end Behavioral;
```

Figura 3: Casos de Teste - Multiplexador 8 para 1. Fonte: Autor.

Realizando a simulação e comparando com a tabela verdade acima obtemos os seguintes resultados:

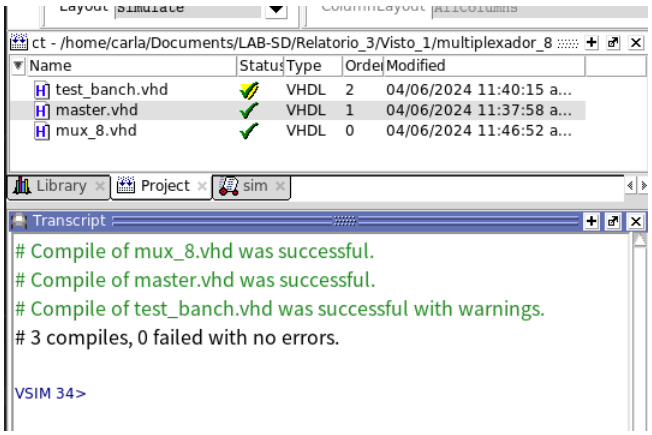

Figura 4: Compilação dos arquivos - Multiplexador 8 para 1. Fonte: Autor.

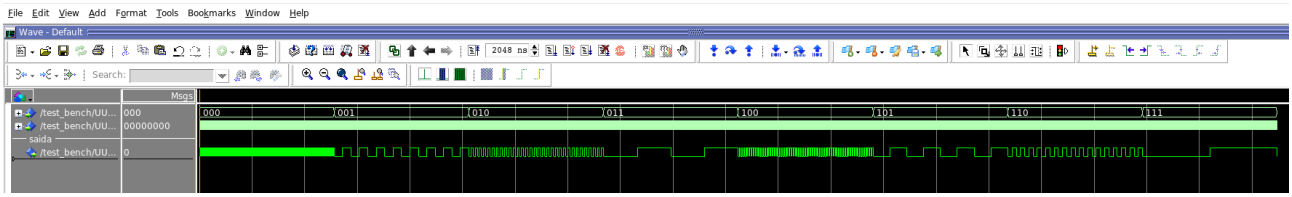

Figura 5: Simulação do Multiplexador 8 para 1. Fonte: Autor.

2. Utilizando atribuições seletivas (with-select), escrever em VHDL e implementar em FPGA uma entidade que descreva um decodificador de 4 para 16. Essa entidade deve ter como entrada um vetor A de 4 bits e, como saída, um vetor Y de 16 bits. A tabela verdade do decodificador é apresentada abaixo. Este visto será tomado em duas etapas, em que será usado o mesmo código VHDL, mas diferentes arquivos UCF:

A (4 bits)	Y (16 bits)
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000
0100	0000000000010000
0101	0000000000100000
0110	0000000001000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

Tabela 2: Tabela verdade do decodificador 4 para 16

a. Respeitando a ordem de significância, associe cada um dos bits de A a diferentes chaves (SW0 a SW3) e cada um dos 8 bits menos significativos de Y a um dos LEDs (LD0 a LD7).

   Assim como na questão anterior, começamos fazendo um componente mas dessa vez com uma entrada de 4 bits e uma saída de 16, obedecendo a tabela verdade acima:

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity dec_4_to_16 is
6     Port (
7       A : in  STD_LOGIC_VECTOR (3 downto 0);
8       Y : out STD_LOGIC_VECTOR (15 downto 0)
9     );
10  end dec_4_to_16;
11
12  architecture Behavioral of dec_4_to_16 is
13  begin
14    with A select
15      Y <= "0000000000000001" when "0000",
16           "0000000000000010" when "0001",
17           "0000000000000100" when "0010",
18           "0000000000001000" when "0011",
19           "0000000000010000" when "0100",
20           "0000000000100000" when "0101",
21           "0000000001000000" when "0110",
22           "0000000010000000" when "0111",
23           "0000000100000000" when "1000",
24           "0000001000000000" when "1001",
25           "0000010000000000" when "1010",
26           "0000100000000000" when "1011",
27           "0001000000000000" when "1100",
28           "0010000000000000" when "1101",
29           "0100000000000000" when "1110",
30           "1000000000000000" when others;
31  end Behavioral;
32
```

Figura 6: Decodificador. Fonte: Autor.

Implementando uma entidade para manipular que possamos realizar simulações temos:

```vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Master is
    Port (
        SW : in STD_LOGIC_VECTOR (3 downto 0);
        LED: out STD_LOGIC_VECTOR(7 downto 0)
      );
end Master;

architecture Behavioral of Master is

signal u1: STD_LOGIC_VECTOR(3 downto 0);
signal u0: STD_LOGIC_VECTOR(15 downto 0);

component dec_4_to_16
    Port (
       A : in  STD_LOGIC_VECTOR (3 downto 0);
       Y : out STD_LOGIC_VECTOR (15 downto 0)
       );
end component;

begin

u1 <= SW;

decodificador: dec_4_to_16 port map(A => u1, Y => u0);

LED <= u0(7 downto 0);

end Behavioral;
```

Figura 7: Entidade - Decodificador - Bits Menos Significativos.. Fonte: Autor.

Dessa forma, podemos ver que os 4 bits de entrada são associados as chaves SW (SW0 a SW3) e os bits menos significativos a LEDs (LD0 a LD7).

E por fim, foi implementado o arquivo de teste:

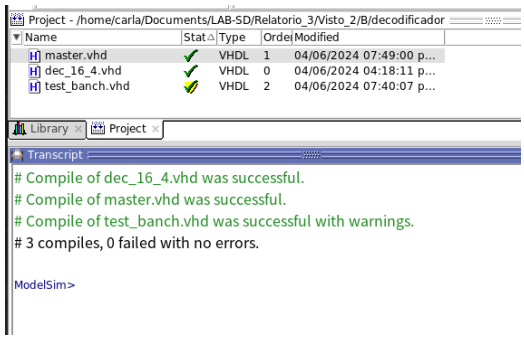

Figura 8: Compilação dos Arquivos - Decodificador. Fonte: Autor.

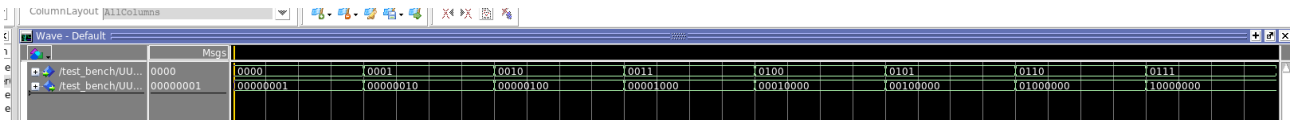

Figura 9: Simulação do Decodificador - Bits menos significativos. Fonte: Autor.

b. Respeitando a ordem de significância, associe cada um dos bits de A a diferentes chaves (SW0 a SW3) e cada um dos 8 bits mais significativos de Y a um dos LEDs (LD0 a LD7).

Aproveitando o componente anterior porém analisando os bits mais significativos, temos a seguinte entidade:

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity Master is
6     Port (
7         SW : in STD_LOGIC_VECTOR (3 downto 0);
8         LED: out STD_LOGIC_VECTOR(7 downto 0)
9       );
10  end Master;
11
12  architecture Behavioral of Master is
13
14  signal u1: STD_LOGIC_VECTOR(3 downto 0);
15  signal u0: STD_LOGIC_VECTOR(15 downto 0);
16
17  component dec_4_to_16
18    Port (
19       A : in  STD_LOGIC_VECTOR (3 downto 0);
20       Y : out STD_LOGIC_VECTOR (15 downto 0)
21       );
22  end component;
23
24  begin
25
26  u1 <= SW;
27
28  decodificador: dec_4_to_16 port map(A => u1, Y => u0);
29
30  LED <= u0(15 downto 8);
31
32  end Behavioral;
```

Figura 10: Entidade - Decodificador - Bits Mais Significativos. Fonte: Autor.

E realizando as simulações obtemos:

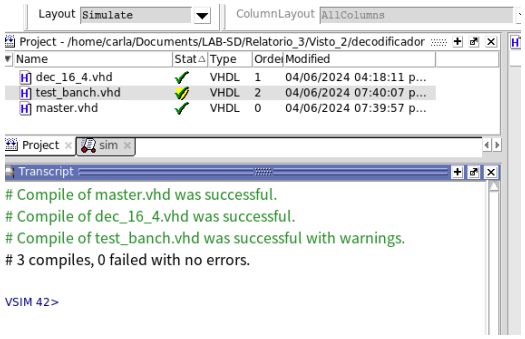

Figura 11: Compilação dos Arquivos - Decodificador. Fonte: Autor.

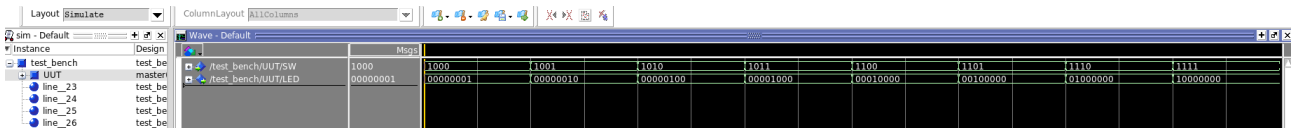

Figura 12: Simulação do Decodificador - Bits Mais Significativos. Fonte: Autor.