UNIVERSIDADE FEDERAL DO PARANÁ DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Rian Marcos Sepulveda
GRR20196632

Microeletrônica

Lab 1

CURITIBA

2024

PROJETO 1

O Projeto 1 consiste na criação de um Multiplexador (Mux) de 4 entradas, cada uma com 4 bits de largura, acompanhado por uma chave seletora de 2 bits para determinar a entrada ativa do circuito. Além disso, inclui um elemento de memória com uma entrada X, que é controlada por um sinal de clock, produzindo uma saída Y correspondente. A implementação desse projeto foi realizada utilizando o software ISE Design Suite na versão 14.7 utilizando da programação VHDL. A seguir, será apresentado o código fonte e o seu funcionamento detalhado.

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity reg_mux is
7       Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
8              b : in  STD_LOGIC_VECTOR (3 downto 0);
9              c : in  STD_LOGIC_VECTOR (3 downto 0);
10             d : in  STD_LOGIC_VECTOR (3 downto 0);
11             sel : in  STD_LOGIC_VECTOR (1 downto 0);
12             clk : in  STD_LOGIC;
13             x : out  STD_LOGIC_VECTOR (3 downto 0);
14             y : out  STD_LOGIC_VECTOR (3 downto 0));
15  end reg_mux;
16
17  architecture Behavioral of reg_mux is
18     signal mux: std_logic_vector (3 downto 0);
19  begin
20
21  --multiplexador
22  mux <= a when sel="00" else
23     b when sel="01" else
24     c when sel="10" else
25     d;
26
27  x <= mux;
28  --process para o codigo rodar de forma sequencial
29  --o clock faz com que y mude o valor
30  --flipflop:
31  process(clk)
32  begin
33     if(clk'event and clk='1') then
34        y <= mux;
35     end if;
36  end process;
37
38  end Behavioral;
39
40
41
```

Figura 1 - Código de implementação VHDL

Este código VHDL define um registro multiplexador (reg_mux) que aceita quatro entradas (a, b, c e d), cada uma com um vetor lógico padrão de 4 bits, e uma entrada de seleção (sel) de 2 bits. Além disso, há uma entrada de clock (clk) e duas saídas (x e y), ambas com vetores lógicos padrão de 4 bits.

Na linha 17 (arquitetura Behavioral), é definido um sinal interno chamado "mux", que é um vetor lógico padrão de 4 bits. Este sinal "mux" atua como um multiplexador que seleciona uma das quatro entradas (a, b, c ou d) com base no valor da entrada "sel". A seguir, a saída "x" recebe o valor do multiplexador "mux".

Após isto, há um processo sensível ao clock (clk). Este processo atualiza a saída "y" com o valor do multiplexador "mux" sempre que ocorre uma borda de subida do sinal de clock. Isso significa que a saída "y" muda seu valor sincronizado com o clock *(flip-flop)*
Após a síntese do código VHDL foram obtidos os seguintes esquemáticos RTL:

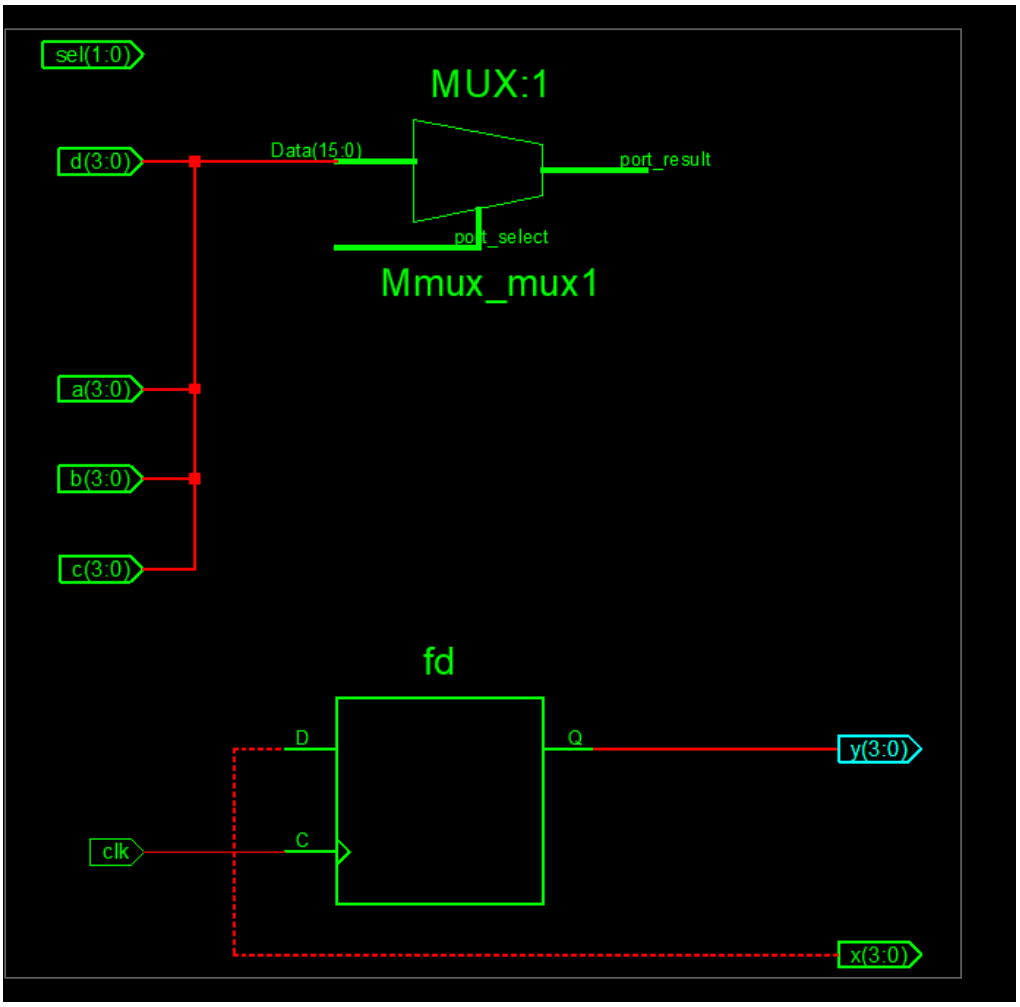

Figura 2: Esquemático RTL

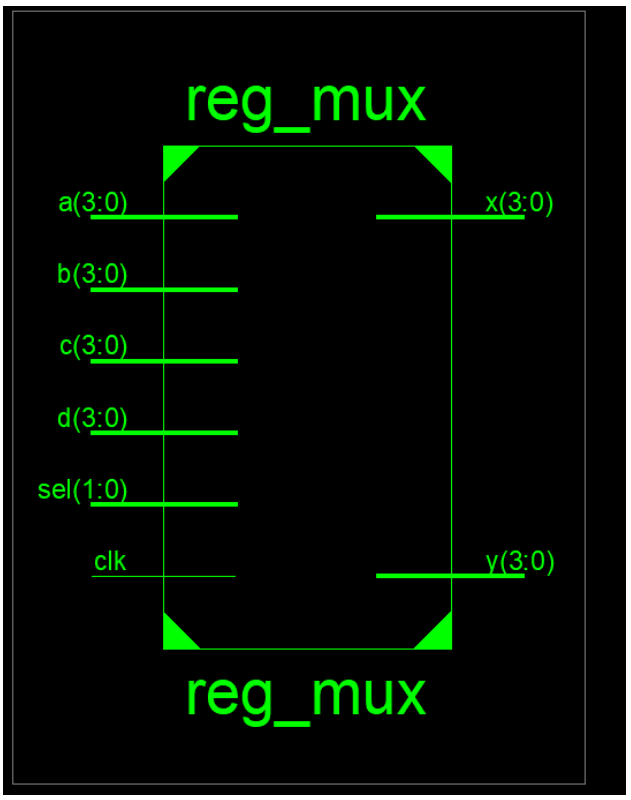

Figura 3: Esquemático RTL

Com o esquemático é possível observar as conexões entre os componentes e como os mesmos interagem para implementar a lógica programada.

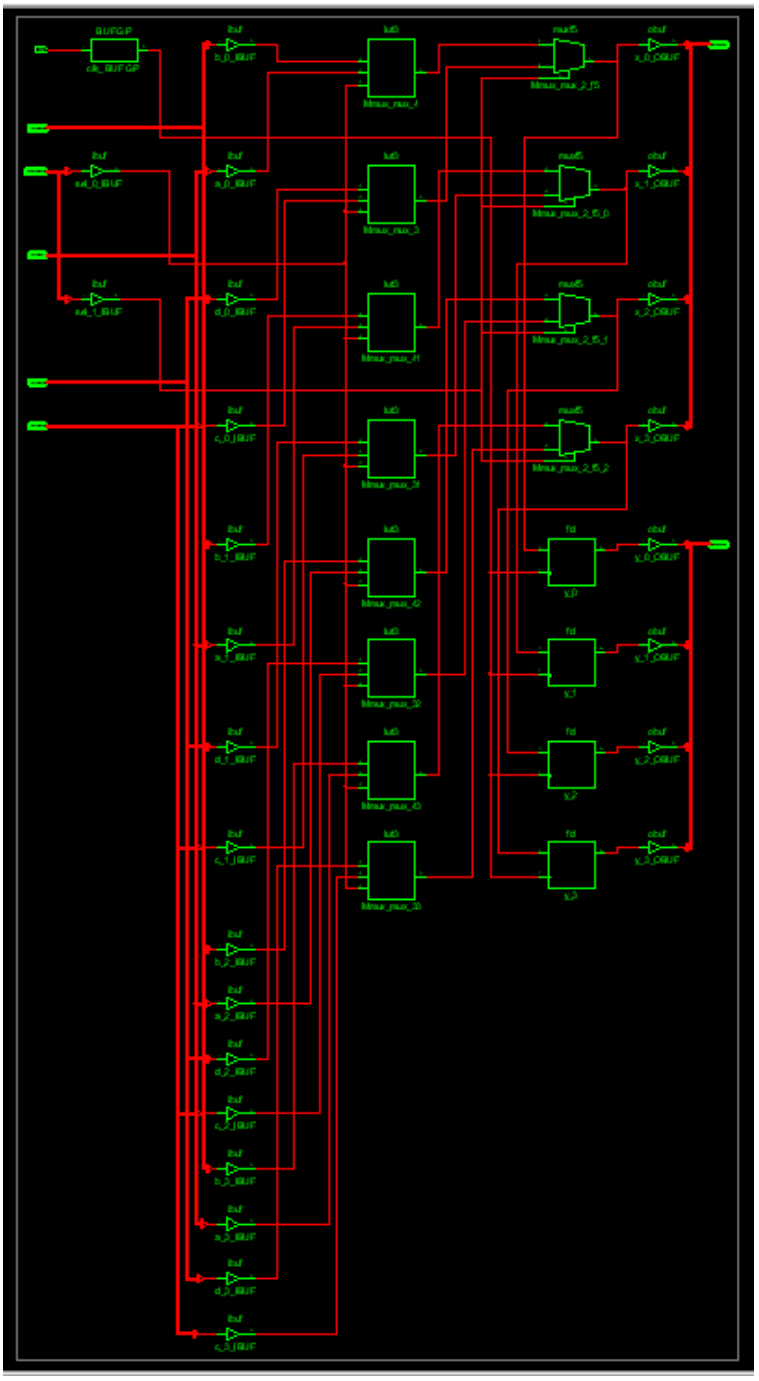

Figura 4: Esquemático da tecnologia

Após esta fase, foi elaborado um código de *testbench* para verificar a adequação da implementação, garantindo a correção lógica do projeto. Este testbench será implementado para validar o funcionamento do circuito, fornecendo diversos padrões de entrada e comparando as saídas geradas com o comportamento esperado conforme a lógica previamente definida. Abaixo segue imagem do código aplicado:

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5
6  ENTITY reg_mux_tb IS
7  END reg_mux_tb;
8
9  ARCHITECTURE behavior OF reg_mux_tb IS
10
11     -- Component Declaration for the Unit Under Test (UUT)
12
13     COMPONENT reg_mux
14     PORT(
15          a : IN  std_logic_vector(3 downto 0);
16          b : IN  std_logic_vector(3 downto 0);
17          c : IN  std_logic_vector(3 downto 0);
18          d : IN  std_logic_vector(3 downto 0);
19          sel : IN  std_logic_vector(1 downto 0);
20          clk : IN  std_logic;
21          x : OUT  std_logic_vector(3 downto 0);
22          y : OUT  std_logic_vector(3 downto 0)
23         );
24     END COMPONENT;
25
26
27    --Inputs
28    signal a : std_logic_vector(3 downto 0) := "0010";
29    signal b : std_logic_vector(3 downto 0) := "0100";
30    signal c : std_logic_vector(3 downto 0) := "0110";
31    signal d : std_logic_vector(3 downto 0) := "1000";
32    signal sel : std_logic_vector(1 downto 0) := (others => '0');
33    signal clk : std_logic := '0';
34
35    --Outputs
36    signal x : std_logic_vector(3 downto 0);
37    signal y : std_logic_vector(3 downto 0);
38
39    -- Clock period definitions
40    constant clk_period : time := 10 ns;
41
```

Figura 5 : Código do Testbench

```
42 BEGIN
43
44     -- Instantiate the Unit Under Test (UUT)
45     uut: reg_mux PORT MAP (
46             a => a,
47             b => b,
48             c => c,
49             d => d,
50             sel => sel,
51             clk => clk,
52             x => x,
53             y => y
54           );
55
56
57
58     -- Stimulus process
59    clk <= not clk after clk_period/2;
60 a <= "0011" after 80 ns, "0000" after 640 ns;
61 b <= "0101" after 240 ns;
62 c <= "0111" after 400 ns;
63 d <= "1001" after 560 ns;
64 sel <= "01" after 160 ns,
65        "10" after 320 ns,
66        "11" after 480 ns,
67        "00" after 640 ns;
68
69 END;
70
```

Figura 6 : Continuação código do Testbench

Este código pode ser dividido em duas partes principais. Primeiramente, ocorre uma definição de componente, que faz referência à entidade "*reg_mux*" previamente criada. Essencialmente, esse bloco se assemelha à definição de entidade, exceto pela substituição da palavra-chave "*entity*" por "*component*". Adicionalmente, são estabelecidos os sinais destinados à simulação de teste, junto com o mapeamento desses sinais para as portas correspondentes do componente.
Na segunda parte do *testbench*, definimos a simulação propriamente dita. Isso é feito através de um **processo** no qual as ações são executadas sequencialmente no tempo. A simulação envolve, essencialmente, a modificação da entrada selecionada, aguardando um período de clock, alterando então o valor do sinal de entrada, esperando novamente por um período de clock e repetindo esse processo até passar por todas as entradas disponíveis.

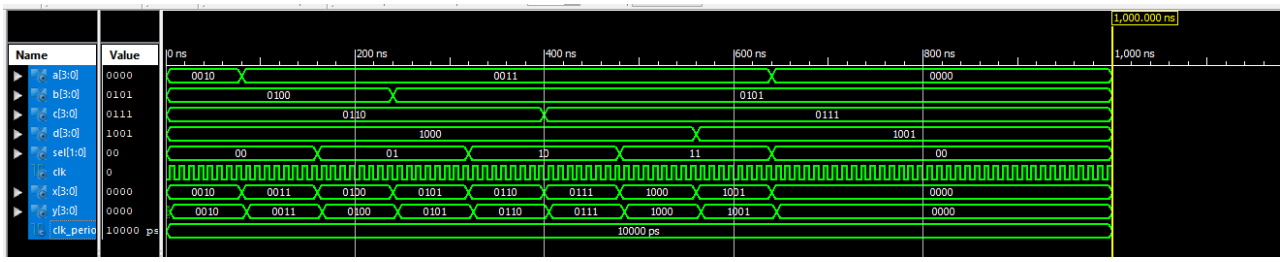

Figura 7: Simulação reg_mux projeto 1

Percebemos que a saída "x" recebeu os valores de entrada de acordo com a seleção do vetor "sel", enquanto a saída recebeu o valor de entrada quando o sinal de clock estava em um nível lógico alto. Ou seja, obtemos êxito no gráfico.

Abaixo segue o resumo dos recursos lógicos utilizados. Tal recurso é fundamental pois auxilia os desenvolvedores a identificar os requisitos de desempenho e de funcionalidade antes da implementação física.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	8	9,312	1%	
Number of occupied Slices	4	4,656	1%	
Number of Slices containing only related logic	4	4	100%	
Number of Slices containing unrelated logic	0	4	0%	
Total Number of 4 input LUTs	8	9,312	1%	
Number of bonded IOBs	27	232	11%	
IOB Flip Flops	4			
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	1.64			

Figura 8: Resumo de utilização dos recursos do dispositivo

PROJETO 2

O segundo projeto é bem similar ao primeiro, porém desta vez alteramos o código para empregar escalares de apenas 1 bit, substituindo o uso do STD_LOGIC_VECTOR por STD _LOGIC, pois o multiplexador será configurado na placa Nexys2, com os switches sendo empregados para as comutações do mux. Para essa configuração os botões da placa atuam como chaves seletoras de 1 bit cada, serão utilizados dois botões, formando um vetor de 2 bits, para controlar a seleção do mux. Dessa forma, torna-se viável escolher qual dos quatro inputs será direcionado para a saída X, a qual está conectada a um elemento de memória com saída Y. No contexto da placa de desenvolvimento, essa saída é visualizada através de um LED, que possui dois estados distintos, e é atualizada com o valor de X a cada sinal de clock.

Então temos o código de implementação da seguinte forma:

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity reg_mux is
7       Port ( a : in  STD_LOGIC;
8              b : in  STD_LOGIC;
9              c : in  STD_LOGIC;
10             d : in  STD_LOGIC;
11             sel : in  STD_LOGIC_VECTOR (1 downto 0);
12             clk : in  STD_LOGIC;
13             x : out  STD_LOGIC;
14             y : out  STD_LOGIC );
15  end reg_mux;
16
17  architecture Behavioral of reg_mux is
18     signal mux: std_logic;
19  begin
20  mux <= a when sel="00" else
21     b when sel="01" else
22     c when sel="10" else
23     d;
24  x <= mux;
25  process(clk)
26  begin
27     if(clk'event and clk='1') then
28        y <= mux;
29     end if;
30  end process;
31
32  end Behavioral;
```

Figura 9 - Código de implementação VHDL - Projeto 2

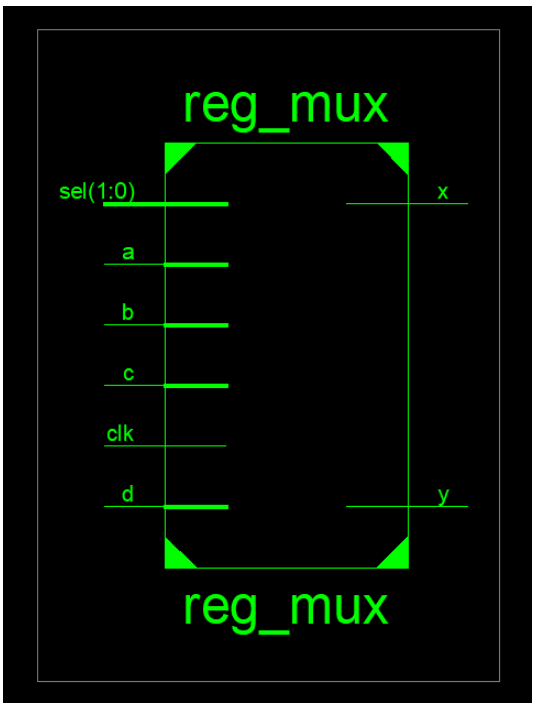

Figura 10: Esquemático RTL gerado pelo ISE

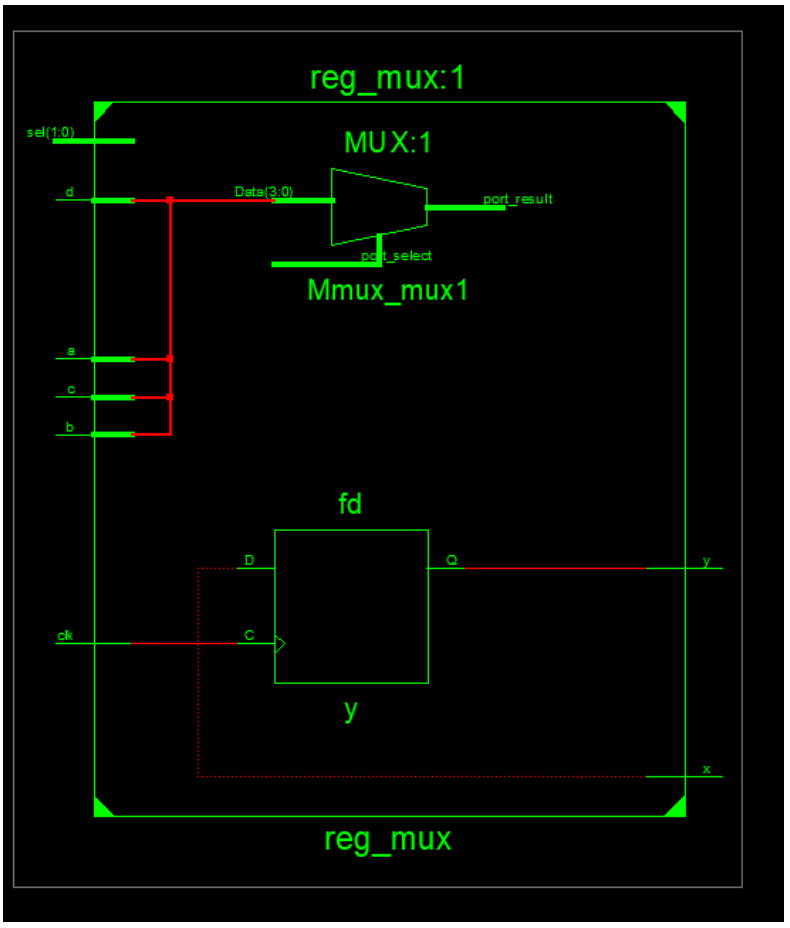

Figura 11: Segunda camada do Esquemático RTL mostrando o mux e o flip flop y

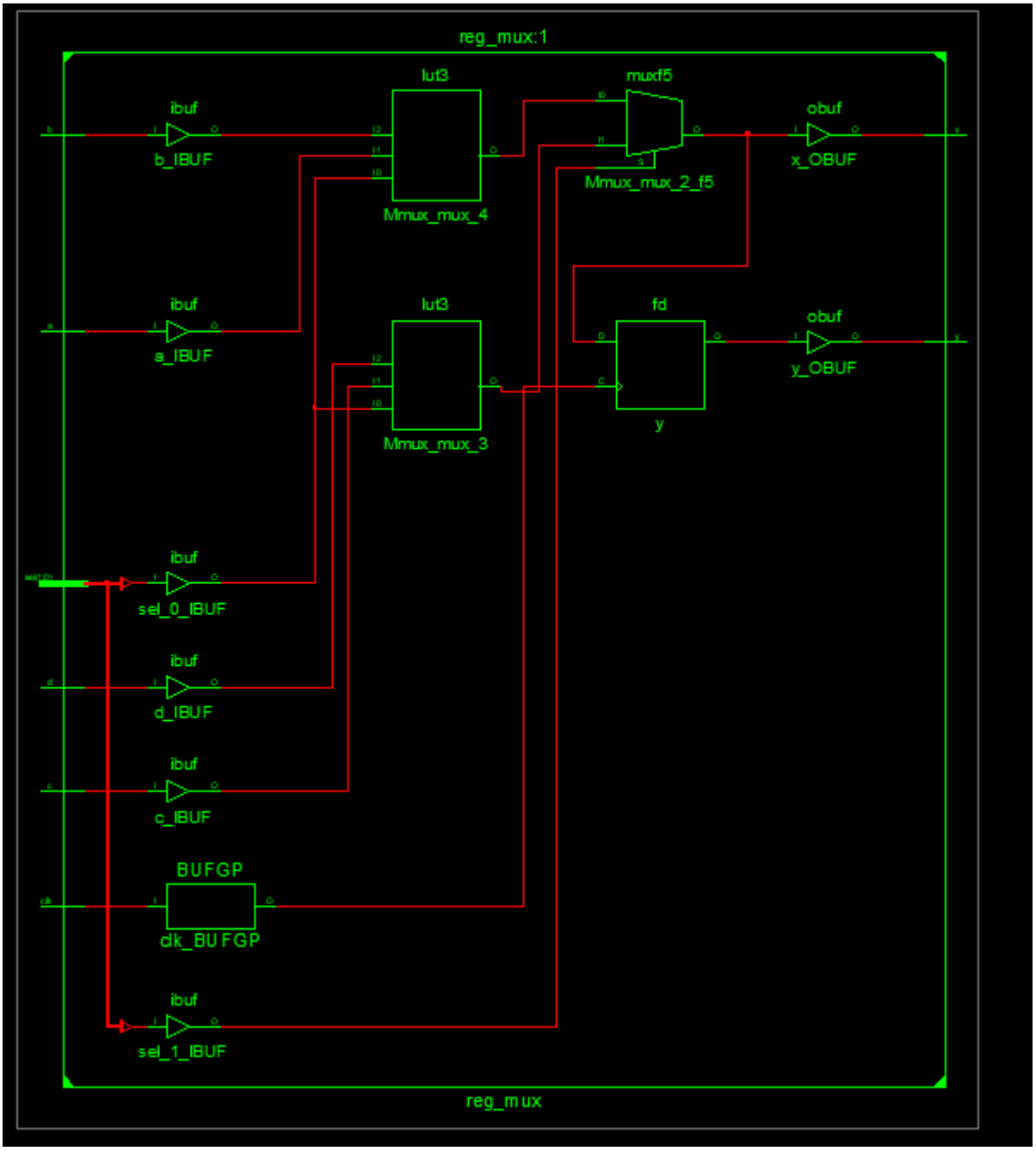

Figura 12: Esquemático de tecnologia

Nas próximas figuras observamos as adaptações feitas no testbench:

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4
5
6   ENTITY reg_mux_tb IS
7   END reg_mux_tb;
8
9   ARCHITECTURE behavior OF reg_mux_tb IS
10
11      -- Component Declaration for the Unit Under Test (UUT)
12
13      COMPONENT reg_mux
14      PORT(
15           a : IN  std_logic;
16           b : IN  std_logic;
17           c : IN  std_logic;
18           d : IN  std_logic;
19           sel : IN  std_logic_vector(1 downto 0);
20           clk : IN  std_logic;
21           x : OUT  std_logic;
22           y : OUT  std_logic
23          );
24      END COMPONENT;
25
26
27     --Inputs
28
29     signal a : std_logic := '0';
30     signal b : std_logic := '0';
31     signal c : std_logic := '0';
32     signal d : std_logic := '0';
33     signal sel : std_logic_vector(1 downto 0) := (others => '0');
34     signal clk : std_logic := '0';
35
36     --Outputs
37     signal x : std_logic;
38     signal y : std_logic;
39
40     -- Clock period definitions
41     constant clk_period : time := 10 ns;
42
43  BEGIN
44
```

```
43  BEGIN
44
45     -- Instantiate the Unit Under Test (UUT)
46     uut: reg_mux PORT MAP (
47             a => a,
48             b => b,
49             c => c,
50             d => d,
51             sel => sel,
52             clk => clk,
53             x => x,
54             y => y
55           );
56
57
58
59      -- Stimulus process
60    clk <= not clk after clk_period/2;
61    a <= not a after 320 ns;
62    b <= not b after 160 ns;
63    c <= not c after 80 ns;
64    d <= not d after 40 ns;
65    sel <= "01" after 200 ns,
66           "10" after 400 ns,
67           "11" after 600 ns,
68           "00" after 800 ns;
69
70  END;
71
```

Figura 13: Código do Testbench - Projeto 2

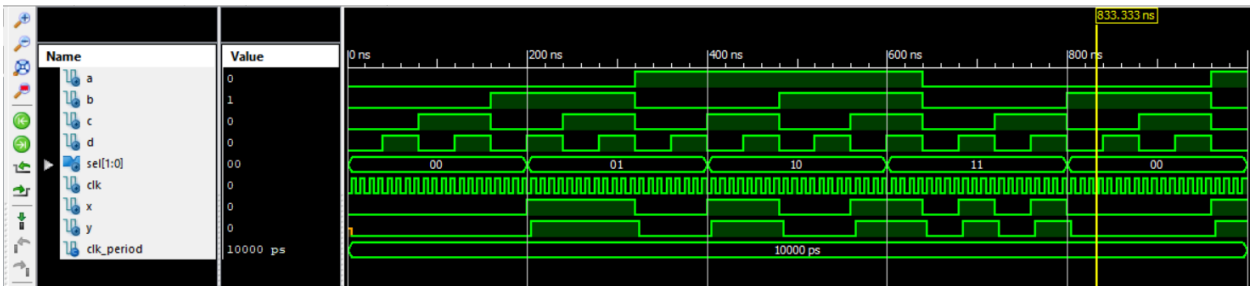

Figura 14: Simulação reg_mux - Projeto 2

Ao realizar a simulação, os sinais de entrada e saída são definidos com valores individuais de 1 ou 0, representando os bits individualmente, em vez de vetores de 4 bits. Essa abordagem reflete a alteração no código para empregar escalares de 1 bit em vez de vetores de 4 bits, conforme mencionado anteriormente.

Como próximo passo será necessário mapear as portas de I/O utilizando pinos disponíveis na placa Nexys2. Para isto, será adicionado um novo arquivo ao projeto, do tipo *Implementation Constraints File:*

```
1  NET "a" LOC = "G18";
2  NET "b" LOC = "H18";
3  NET "c" LOC = "k18";
4  NET "d" LOC = "k17";
5  NET "sel(0)" LOC = "L14";
6  NET "sel(1)" LOC = "L13";
7  NET "clk" LOC = "B8";
8  NET "x" LOC = "J14";
9  NET "y" LOC = "J15";
```

Figura 15: Mapeamento das portas de I/O

Para a implementação na placa Nexys2 foi utilizado o programa Adept, então foi possível verificar o resultado esperado, assim obtendo êxito na atividade.