UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIAS
CURSO DE ENGENHARIA ELÉTRICA

LUCAS FILUS RAMOS - GRR20202598
VITORIA MARIANA DA ROCHA GASINO - GRR20202564

TE351 DA – LABORATÓRIO 1

CURITIBA
2024

UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIA
CURSO DE ENGENHARIA ELÉTRICA

Lucas Filus Ramos
Vitoria Mariana da Rocha Gasino

TE 351 DA – LABORATÓRIO 1

Relatório apresentado à disciplina Microeletrônica do Curso de Engenharia Elétrica do Setor de Tecnologia da Universidade Federal do Paraná.

Orientador: Sibilla Batista da Luz Franca.

CURITIBA

2024

Sumário

1. INTRODUÇÃO

O relatório a seguir tem como objetivo detalhar um projeto de um multiplexador em cascata em conjunto com um flip-flop descrito em VHDL. O multiplexador em uso apresenta 4 portas de entrada, uma porta de saída e uma entrada de seleção, enquanto o flip-flop empregado é do tipo D.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. VHDL

Uma linguagem de descrição de hardware (HDL - *Hardware Description Language*) é uma forma especializada para descrever a funcionalidade e o comportamento de circuitos eletrônicos e sistemas digitais. Ao contrário das linguagens de programação convencionais, que se concentram em instruções sequenciais para executar tarefas em um processador, as linguagens de descrição de hardware permitem especificar o design lógico e a interconexão de componentes eletrônicos, como portas lógicas, flip-flops e multiplexadores. Essas linguagens, como o VHDL e o Verilog, permitem aos engenheiros projetar, simular e verificar circuitos digitais antes da implementação física em dispositivos como as FPGAs (*Field-Programmable Gate Arrays*).

As FPGAs são dispositivos programáveis capazes de implementar lógica digital e foram desenvolvidas para descrever circuitos combinacionais e sequenciais [7]. Devido à sua natureza reconfigurável, as FPGAs podem ser adaptadas para realizar operações específicas de forma mais eficiente em comparação com as CPUs clássicas e microprocessadores convencionais. Esse desempenho superior em operações específicas as torna valiosas para aplicações que requerem processamento intensivo em tempo real, processamento paralelo, aceleração de algoritmos complexos e execução eficiente de tarefas personalizadas. Além disso, a versatilidade das FPGAs permite a criação de circuitos customizados para tarefas específicas, reduzindo a dependência de hardwares genéricos e proporcionando maior flexibilidade na implementação de sistemas digitais.

Podemos medir a eficiência e otimização de recursos de uma FPGA a partir de várias métricas e análises. Uma das métricas-chave é a utilização de LUTs (*Look-Up Tables*) e flip-flops, onde uma utilização baixa em relação à capacidade total da FPGA indica um design eficiente. Além disso, a taxa de clock máxima que a FPGA pode atingir é um indicador importante de rapidez. A análise de temporização também é fundamental para garantir que as operações lógicas sejam concluídas dentro dos requisitos de tempo. O consumo de energia é outra métrica crítica, pois uma FPGA eficiente deve executar seu design com um consumo mínimo de energia. Em última análise, a combinação dessas métricas e análises ajuda a

avaliar a eficiência global e a otimização de recursos de uma FPGA em um determinado contexto de aplicação.

2.1.1. STD_LOGIC e STD_LOGIC_VECTOR

Em sistemas digitais descritos em VHDL, o tipo de dados BIT é utilizado para representar um único bit, permitindo apenas dois valores distintos: '0' para nível lógico baixo e '1' para nível lógico alto.

Em contrapartida, o tipo de dados STD_LOGIC expande essa representação, adicionando estados como 'Z' para alta impedância, 'U' para não inicializado(Uninitialized), 'X' para irrelevante, e os valores padrão '0' e '1'.

2.1.2. Simulação

A simulação comportamental permite verificar se o design atende aos requisitos funcionais, fazendo o teste com o circuito se comportando exatamente conforme o esperado em termos de entradas e saídas. Por outro lado, a simulação *post-route*, considera os detalhes de implementação, como atrasos de sinal e layout físico. Essa etapa é fundamental para validar a integridade da implementação física, verificando se o design funciona dentro das restrições de tempo e desempenho estabelecidas.

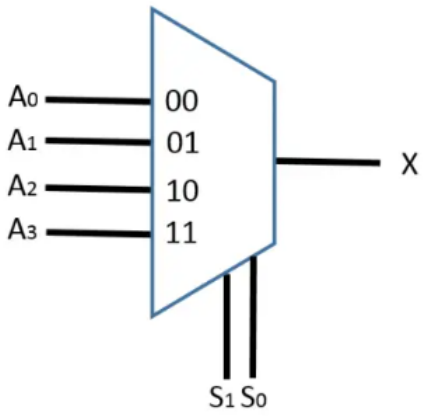
Neste relatório , foi implementado uma simulação Behaviour utilizando o software da Xilinx ISE 14.7. A FPGA utilizada para o projeto foi uma Spartan 3E 500.

2.2. Multiplexador e Filp-Flop

O multiplexador, com suas 4 portas de entrada, uma saída e uma entrada de seleção, tem a função de selecionar uma dentre as várias entradas para ser direcionada à saída, de acordo com o sinal presente na entrada de seleção. A figura a seguir ilustra a tabela verdade de um multiplexador de 4 entradas.

FIGURA 1 – Circuito e tabela verdade de um multiplexador

S_1	S_0	X
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3



FONTE: Embarcados.com

Por outro lado, o flip-flop do tipo D é um componente capaz de armazenar um único bit de informação, que é atualizado na entrada do pulso de clock. A seguir está a tabela verdade deste componente.

FIGURA 2 – Tabela verdade de um Flip-Flop tipo D.

D	CLK	SAÍDA
0	↑	Q = 0
1	↑	Q = 1

FONTE: AJPEletroinfo.com

O objetivo desta simulação é observar e verificar o comportamento da saída do multiplexador quando conectada ao flip-flop, em resposta aos pulsos de clock.

3. PROJETO 1 (Entradas de vetores de 4 bits)

3.1. Desenvolvimento do código

Após análise das tabelas verdades é possível transformar este circuito composto por um multiplexador em cascata com um flip-flop com o seguinte código em VHDL.

FIGURA 3 – Tabela verdade de um Flip-Flop tipo D.

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


entity lab1_proj1 is
    Port ( a, b, c, d : in  STD_LOGIC_VECTOR (3 downto 0);
           sel : in  STD_LOGIC_VECTOR (1 downto 0);
           clk : in  STD_LOGIC;
           x , y : out  STD_LOGIC_VECTOR (3 downto 0));
end lab1_proj1;

architecture Behavioral of lab1_proj1 is
signal mux: STD_LOGIC_VECTOR(3 downto 0);

begin
--Parte Combinacional
   mux <= a WHEN sel="00" ELSE
          b WHEN sel="01" ELSE
          c WHEN sel="10" ELSE
          d;

--Registrador
   process(clk)
   begin
      if(clk'event and clk='1') then
         y<=mux;
      end if;
   end process;

   x<=mux;
end Behavioral;
```

FONTE: Os autores (2024)

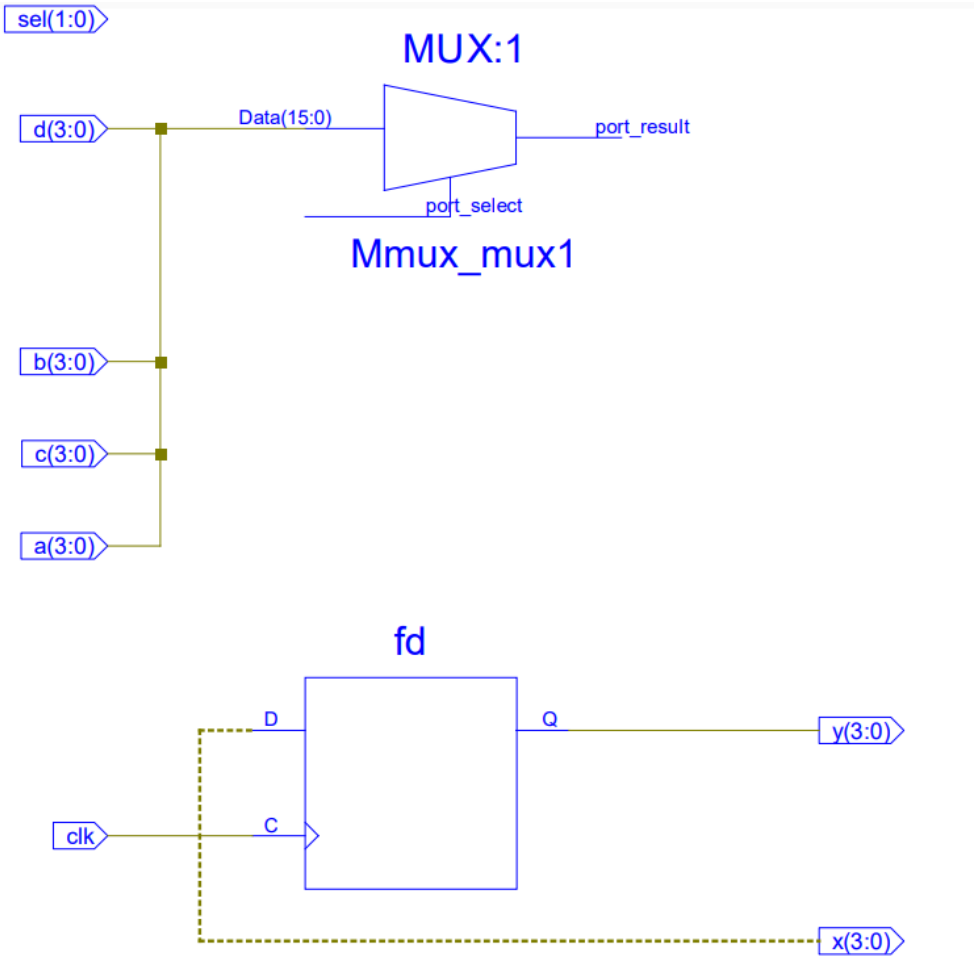
As variáveis de entrada 'a', 'b', 'c' e 'd', e também as variáveis de saída 'x' e 'y' são todas com 4 bits. Também são estabelecidas as variáveis de seleção 'sel', com 2 bits, e o sinal de clock de entrada 'clk' com 1 bit.

Na implementação, o sinal 'mux', de 4 bits, recebe o sinal de uma das entradas, e a saída do multiplexador 'x' é determinada pelo valor de 'mux'. Foi necessário definir um sinal interno à arquitetura pois o valor de saída do multiplexador é utilizado tanto como entrada para o flip-flop quanto para saída, em x. A saída do flip-flop 'y' é atualizada somente quando o sinal de clock está em nível lógico alto, ou seja, quando seu valor é igual a 1.

Com o código pronto, na área “Desing Summary” é possível avaliar o número de LUT’s e flip-flops utilizados, assim como outros parâmetros importantes. O número total de LUTs utilizado ficou em 8, indicando um design relativamente simples. Já o número de flip-flops foi de 4. Como um flip-flop é necessário para cada bit do registro, registrando-se um vetor de 4 bits, resulta em um design com o total de 4 flip-flops.

Além disso, também é possível verificar o esquemático do circuito após fazer a síntese, como ilustrado na figura 4.

FIGURA 4 – Circuito gerado em VHDL - Projeto 1



FONTE: Os autores (2024)

Como é possível avaliar na imagem 4, o circuito é composto por uma parte combinacional(MUX_1), e outra parte sequencial(fd), que depende das batidas do clock para executar as operações

3.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, o Test Bench descrito na imagem 5 foi preparado.

FIGURA 5 – Testbench gerado para o Projeto 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY testebench IS
END testebench;

ARCHITECTURE behavior OF testebench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT lab1_proj1
    PORT(
         a : IN  std_logic_vector(3 downto 0);
         b : IN  std_logic_vector(3 downto 0);
         c : IN  std_logic_vector(3 downto 0);
         d : IN  std_logic_vector(3 downto 0);
         sel : IN  std_logic_vector(1 downto 0);
         clk : IN  std_logic;
         x : OUT  std_logic_vector(3 downto 0);
         y : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal a : std_logic_vector(3 downto 0) := "0000";
   signal b : std_logic_vector(3 downto 0) := "0001";
   signal c : std_logic_vector(3 downto 0) := "0010";
   signal d : std_logic_vector(3 downto 0) := "0100";
   signal sel : std_logic_vector(1 downto 0) := (others => '0');
   signal clk : std_logic := '0';

  --Outputs
   signal x : std_logic_vector(3 downto 0);
   signal y : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;


    BEGIN

    -- Instantiate the Unit Under Test (UUT)
       uut: lab1_proj1 PORT MAP (
              a => a,
              b => b,
              c => c,
              d => d,
              sel => sel,
              clk => clk,
              x => x,
              y => y
            );

       -- Clock process definitions
       clk_process :process
       begin
          clk <= '0';
          wait for clk_period/2;
          clk <= '1';
          wait for clk_period/2;
     end process;


       stim_proc: process
       begin
          wait for 30 ns;
          sel <= "01";
          wait for clk_period;
          sel <= "10";
          wait for clk_period;
          sel <= "11";
          wait;
       end process;

    END;
```

FONTE: Os autores (2024)

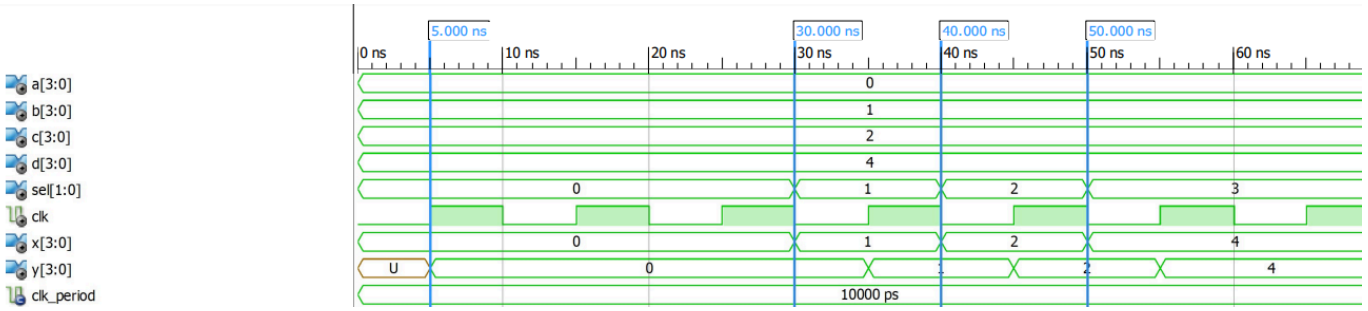
A declaração das portas segue o mesmo padrão do primeiro arquivo. Em seguida, são declarados os sinais que serão utilizados na simulação, juntamente com seus valores iniciais. Posteriormente, é feito o mapeamento entre esses sinais e as portas do componente.

Além disso, foi ajustado o valor do clock para 10 ns por ciclo, assim como os tempos nos quais as variáveis de entrada e saída seriam modificadas, desconsiderando a primeira mudança, pois ocorre após 30 ns. A cada período do clock, o valor de sel muda. Em valores decimais, o valor de sel aumenta uma unidade.

Para conseguir avaliar melhor os resultados variando apenas o sel, foi colocado valores iniciais de a,b,c e d diferentes, onde, em valores decimais , a=0 , b=1, c=2 e d=4.

Após a conclusão da codificação do Test Bench foi possível realizar a simulação do tipo Behaviour em que se verificar se o design atende aos requisitos funcionais, fazendo o teste com o circuito se comportando exatamente conforme o esperado em termos de entradas e saídas. Os resultados são exibidos na figura 6.

FIGURA 6 – Simulação Behaviour do Projeto 1.



FONTE: Os autores (2024)

Na simulação é possível avaliar que o processo de seleção de saída ocorreu como o esperado considerando a simulação de Test Bench codificada, sendo observado os valores dos sinais “a”, “b”, “c”, “d”, “sel”, “x” e “y” em valores do tipo signed, o que torna mais fácil a visualização.

Na imagem, os marcadores horizontais em azul indicam os momentos em que o valor de 'sel' é alterado, exceto o primeiro marcador à esquerda, que indica a primeira batida do clock. O sinal y anterior a 5s está marcado com "u", que representa undefined. Como o sinal y é dependente do registro do flip-flop, o primeiro valor de y é observado após a primeira borda positiva do clock, em 5 ns. Como x recebe o valor diretamente da saída do multiplexador, sem registro, quando o valor do sel muda, automaticamente, o valor x fica disponível. No marcado em 30 ns, é possível observar que quando o valor do sel passa de '0' para '1', no mesmo instante o valor de x atualiza, porém o valor de y demora até a próxima batida de clock.

4. PROJETO 2 (Entradas de bits únicos)

4.1. Desenvolvimento do código

Para o Projeto 2, a lógica dos códigos foi igual à do projeto 1. A principal alteração foi feita nas declarações das portas 'a', 'b', 'c', 'd', 'x' e 'y', onde agora são declaradas como bits únicos e não vetores, ou seja, declaradas apenas como std_logic.

FIGURA 7 – Código .vhd Projeto 2

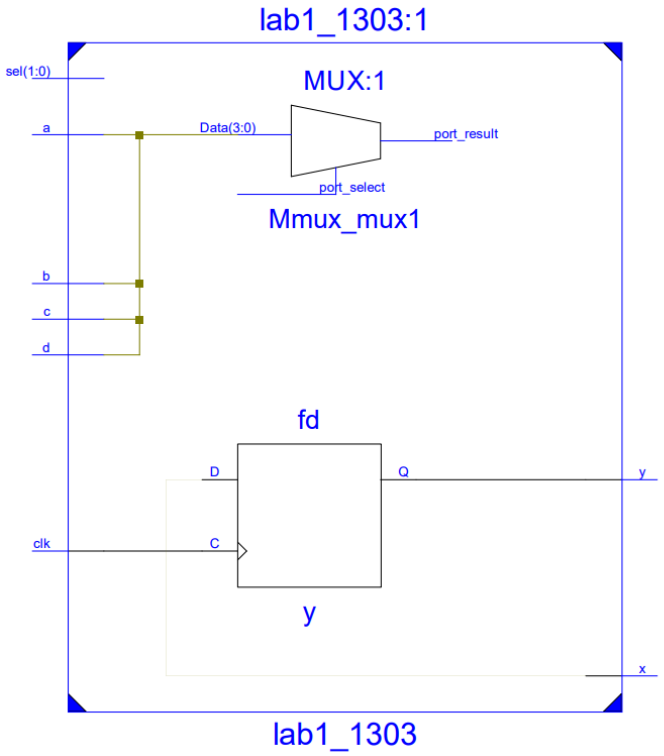
```
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;


   entity lab1_proj1 is
       Port ( a,b,c,d,clk :in STD_LOGIC;
       sel : in STD_LOGIC_VECTOR(1 downto 0);
       x,y : out STD_LOGIC);
   end lab1_proj1;

   architecture Behavioral of lab1_proj1 is
   signal mux: STD_LOGIC;
begin
--Parte Combinacional
   mux <= a WHEN sel="00" ELSE
          b WHEN sel="01" ELSE
          c WHEN sel="10" ELSE
          d;
--Registrador: Determinado pelo evento de clk
-- proces(lista de sensibilidade)
   process(clk)
   begin
      if(clk'event and  clk='1') then
         y <= mux;
      end if;
   end process;
   x <= mux;
end Behavioral;
```

FONTE: Os autores (2024)

Ao reduzirmos o número de posições de a, b, c e d de 4 para 1, era esperado que o número de recursos da FPGA utilizados diminuísse. O número de LUTs observado no resumo do design foi de 2, indicando uma otimização significativa no uso dos recursos da FPGA.

A imagem 8 apresenta o circuito gerado.

FIGURA 8 – Circuito gerado em VHDL - Projeto 2



FONTE: Os autores (2024)

Assim como no projeto 1, o circuito é composto por uma parte combinacional(MUX_1), e outra parte sequencial(fd), que depende das batidas do clock para executar as operações.

4.2. Implementação do Test Bench e resultados

Para avaliar o funcionamento do circuito implementado acima, foi realizado o Teste descrito na imagem 9.

FIGURA 9 – Testbench gerado para o Projeto 2

```
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY testebench IS
END testebench;

ARCHITECTURE behavior OF testebench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT lab1_1303
    PORT(
         a : IN  std_logic;
         b : IN  std_logic;
         c : IN  std_logic;
         d : IN  std_logic;
         clk : IN  std_logic;
         sel : IN  std_logic_vector(1 downto 0);
         x : OUT  std_logic;
         y : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal a : std_logic := '1';
   signal b : std_logic := '0';
   signal c : std_logic := '0';
   signal d : std_logic := '0';
   signal clk : std_logic := '0';
   signal sel : std_logic_vector(1 downto 0) := (others => '0');

   --Outputs
   signal x : std_logic;
   signal y : std_logic;

   -- Clock period definitions
   constant clk_period : time := 10 ns;


   BEGIN

      -- Instantiate the Unit Under Test (UUT)
      uut: lab1_1303 PORT MAP (
             a => a,
             b => b,
             c => c,
             d => d,
             clk => clk,
             sel => sel,
             x => x,
             y => y
           );

      -- Clock process definitions
      clk_process :process
      begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
      end process;


      -- Stimulus process
      stim_proc: process
      begin
         wait for 30 ns;
         sel <= "01";
         wait for clk_period;
         c <= '1';
         sel <= "10";
         wait for clk_period;
         sel <= "11";
         wait;
      end process;
   END;
```

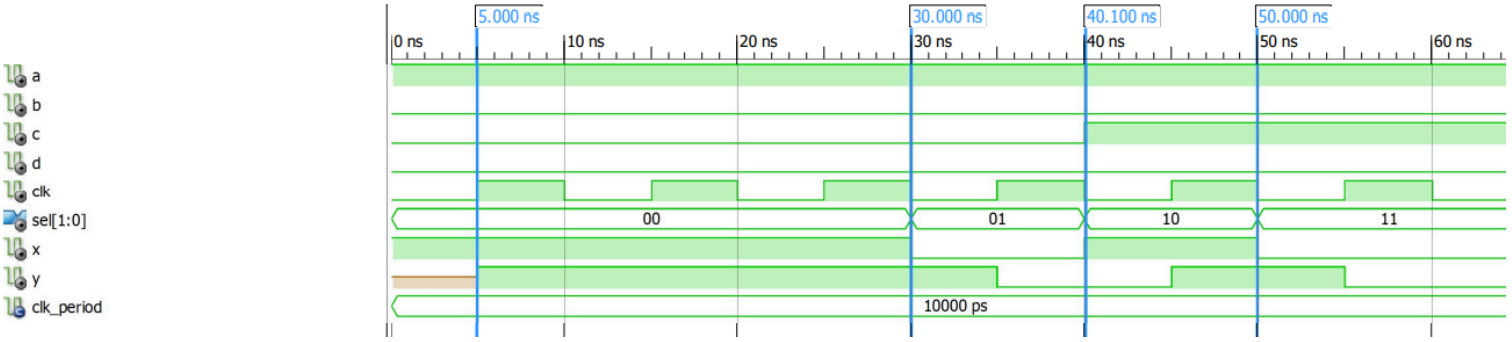
FONTE: Os autores (2024)

Assim como no Projeto 1, a declaração das portas segue o mesmo padrão. Em seguida, são declarados os sinais que serão utilizados na simulação, juntamente com seus valores iniciais. Posteriormente, é feito o mapeamento entre esses sinais e as portas do componente.

Foi utilizado o mesmo valor para o período do clock, de 10 ns . Desconsiderando a primeira mudança, pois ocorre após 30 ns, a cada período do clock, o valor de 'sel' muda. Em valores decimais, o valor de 'sel' aumenta uma unidade. Na segunda batida, foi alterado o valor de 'c' para conseguir avaliar melhor os resultados.

Após a conclusão da codificação do Test Bench foi possível realizar a simulação mostrada na imagem 10.

FIGURA 10 – Simulação Behaviour do Projeto 2.



FONTE: Os autores (2024)

Os resultados e pontos importantes são os mesmos do Projeto 1, assim como a indefinição da saída y antes pro primeiro impulso de clock, porém a única diferença é que neste test bench foi preferido a visualização por meio de valores binários e não decimais.

Com o projeto 2, foi possível carregar o código na FPGA e observar os resultados obtidos na placa física. Foi utilizado o próprio clock da placa de 500MHz. Para a variável 'sel', que possui 2 bits, foi necessário mapear dois pinos, um para cada bit. Os resultados de 'x' e 'y' foram observados nas saídas dos leds.

5. CONCLUSÃO

Este relatório apresentou a operação do multiplexador com entradas de diferentes bits em VHDL. Com o código gerado no projeto 2, foi realizado o carregamento para a placa física, permitindo a observação de uma correspondência direta entre os resultados obtidos na placa física, os valores simulados e as expectativas teóricas de funcionamento de um multiplexador. Um detalhe importante é que, devido ao uso do clock interno da placa de 500MHz, não foi possível observar o delay entre x e y.