UNIVERSIDADE FEDERAL DO PARANÁ

ELOIZA ARANDA BRUSCHI GRR20212721
MARIA PAULA BASTOS GRR20212762

RELATÓRIO LABORATÓRIO 1: PROJETO D UM MUX DE 4X1

CURITIBA
2024

ELOIZA ARANDA BRUSCHI GRR20212721
MARIA PAULA BASTOS GRR20212762

RELATÓRIO LABORATÓRIO 1: PROJETO DE UM MUX DE 4X1

Relatório apresentada à disciplina de Microeletrônica, Setor de Tecnologia da Universidade Federal do Paraná, como atividade avaliativa.
Professora: Dra. Sibilla Batista da Luz França.

CURITIBA
2024

SUMÁRIO

1 INTRODUÇÃO 16
1.1 VHDL 16
2 PROJETO 1 17
2.1 ESQUEMÁTICOS RTL 17
2.2 SIMULAÇÕES 18
2.3 CÓDIGO COMENTADO 19
2.3.1 VHDL Module 19
2.3.2 VHDL Test Bench 19
3 PROJETO 2 21
3.1 MAPEAMENTO DOS PINOS 21
3.2 ESQUEMÁTICOS RTL 21
3.3 SIMULAÇÕES 22
3.4 CÓDIGO 23
3.4.1 VHDL Module 23
3.4.2 VHDL Test Bench 24
ANEXO 1 – VHDL MODULE – PROJETO 1 16
ANEXO 2 – VHDL TEST BENCH – PROJETO 1 17
ANEXO 3 – VHDL MODULE – PROJETO 2 20
ANEXO 4 – VHDL TEST BENCH – PROJETO 2 21

1 INTRODUÇÃO

Este trabalho apresenta a implementação de dois projetos utilizando a ferramenta ISE. O primeiro projeto é realizado apenas por simulação, enquanto o segundo simplifica o primeiro, adaptando-se à quantidade de elementos de entrada e saída disponíveis na placa do kit Nexys 2.

O primeiro projeto consiste na criação de um circuito que integra um multiplexador e um flip-flop. O segundo projeto modifica a estrutura do primeiro, alterando a dimensão dos sinais de entrada e saída de vetores de 4 bits para valores escalares de 1 bit. Ao final do trabalho, foi elaborado um relatório técnico que documenta todo o processo, incluindo a descrição e o desenvolvimento dos projetos, as simulações realizadas e a análise dos resultados obtidos.

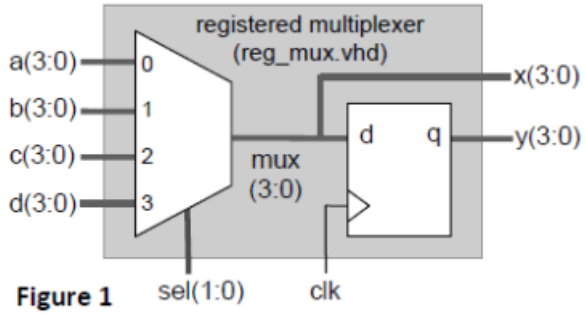
1.1 VHDL

Neste trabalho, utilizamos a plataforma ISE 14.7 da Xilinx para implementar o VHDL (VHSIC Hardware Description Language), uma linguagem de descrição de hardware (HDL) que possibilita a representação de sistemas eletrônicos digitais em diversos níveis de abstração, como RTL e gate-level. Essa flexibilidade torna a modelagem e simulação de circuitos complexos mais acessível. Além disso, o VHDL suporta processos concorrentes e sequenciais, essenciais para a representação de sistemas digitais com múltiplos sinais e eventos simultâneos.

2 PROJETO 1

Utilizou-se a ferramenta ISE para seguir o tutorial apresentado na aula, que implementava um circuito composto por um multiplexador e um flip-flop, conforme mostrado na figura abaixo.

FIGURA 1 – ESQUEMÁTICO CONCEITUAL DO PROJETO 1



FONTE: ‘LAB 1’, Sibilla França (20??).

O tutorial tinha como objetivo ensinar a desenvolver um multiplexador lógico de 4 entradas, cada uma com 4 bits. O ‘mux’ permite selecionar uma das entradas de dados e encaminhá-la para a saída, denotada como ‘x’. A seleção é feita por meio de duas variáveis de controle, SEL0 e SEL1, que determinam qual entrada (a, b, c ou d) será transmitida. A saída também está conectada a um flip-flop D com entrada de clock, o que resulta em uma saída ‘y’ que ocorre apenas no evento da borda de subida do clock.

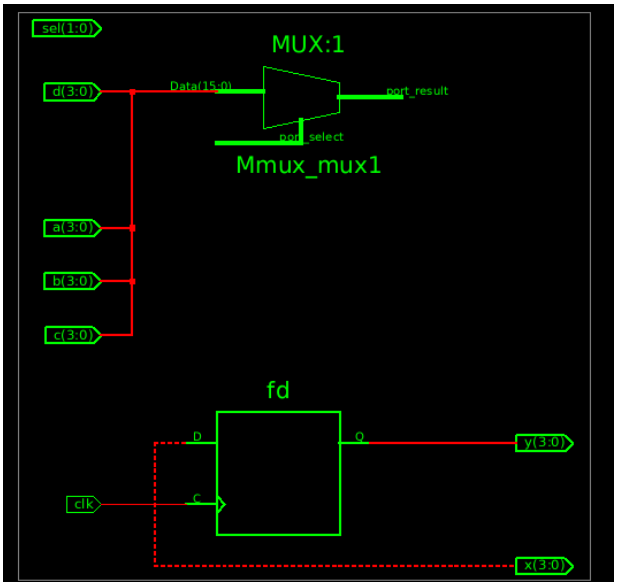
Como todas as entradas ‘a’, ‘b’, ‘c’ e ‘d’ e as saídas ‘x’ e ‘y’ possuem 4 bits, sua implementação real nas configurações da placa Nexys 2 não é viável. Portanto, a solução será realizada de forma simulada.

2.1 ESQUEMÁTICOS RTL

RTL, ou Register Transfer Level, é um nível de abstração em que os registradores e a lógica que manipula os dados entre eles são explicitamente definidos, permitindo a inferência de circuitos sequenciais e combinacionais sem a necessidade de descrever detalhes de hardware, como transistores ou portas lógicas. Assim, esquemáticos RTL são maneiras de visualizar a entrada e saída do projeto de forma ilustrada.

O esquemático RTL do projeto 1 está ilustrado na figura a abaixo, e mostra que o projeto tem entradas e saídas.

FIGURA 2 – ESQUEMÁTICO RTL DO PROJETO 1

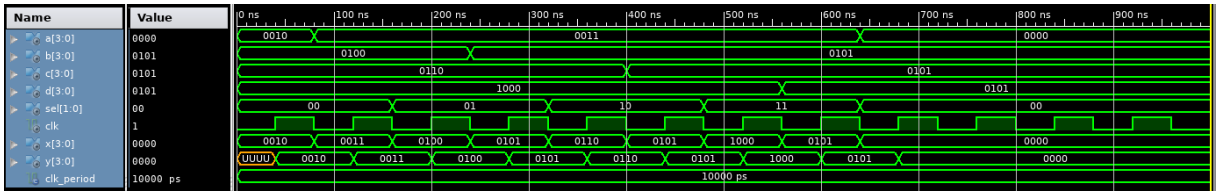


FONTE: As autoras (2024).

2.2 SIMULAÇÕES

Para garantir o funcionamento adequado do projeto, é fundamental realizar simulações. A seguir, apresentamos os resultados do Projeto 1, utilizando um sinal de clock com um período de 80 ns e portas seletoras que variam a cada dois ciclos de clock. Observamos que, quando a porta seletora está em "00", a saída da porta ‘x’ corresponde ao valor da entrada ‘a’, enquanto a saída ‘y’ é igual ao valor de ‘x’ no momento da borda de subida do clock. Da mesma forma, quando a porta seletora é "01", a saída ‘x’ se iguala a ‘b’; com "10", a saída ‘x’ é ‘c’; e, finalmente, quando a porta seletora é "11", a saída ‘x’ é ‘d’. Esses resultados confirmam que o projeto atende às especificações, uma vez que o multiplexador opera conforme o esperado.

FIGURA 3 – SIMULAÇÃO DO PROJETO 1



FONTE: As autoras (2024).

A partir da figura, podemos observar que, durante os primeiros 160 ns, a saída ‘x’ é igual à entrada ‘a’, que inicialmente é "0010" e muda para "0011" quando ‘a’ sofre uma alteração. É importante notar que a saída ‘y’ assume o valor de ‘a’ apenas aos 40 ns de simulação, devido à borda de subida do clock. Em seguida, ‘y’ só se atualiza para "0011" após a próxima borda de subida, ocorrida aos 80 ns. Isso se repete ao longo da simulação conforme a porta seletora muda.

2.3 CÓDIGO COMENTADO

O código comentado é dividido em duas partes, no código de fato da placa física e no código para simular, com as mudanças que seriam feitas se fosse possível testar na placa.

2.3.1 VHDL Module

O código VHDL Module, do projeto 1, está apresentado sem alterações no Anexo 1 deste presente trabalho.

O código é estruturado com quatro entradas em STD_LOGIC_VECTOR, cada uma com 4 bits, e duas saídas também em STD_LOGIC_VECTOR de 4 bits. Além disso, há um seletor chamado ‘sel ’, representado como um vetor de 2 bits, que permite escolher entre as quatro entradas, e um sinal de clock.

A saída ‘x’ é atribuída diretamente à entrada correspondente, conforme definido pelo vetor de seleção ‘sel’. Por outro lado, a saída ‘y’ na placa é atualizada com o valor de ‘x’ somente a cada borda de subida do clock, o que é realizado através de um bloco PROCESS em VHDL.

2.3.2 VHDL Test Bench

O código VHDL Test Bench, do projeto 1, está apresentado sem alterações no Anexo 2 deste presente trabalho. A fim de realizar a simulação, as entradas ‘a’, ‘b’, ‘c’ e ‘d’ foram variadas algumas vezes, com valores distintos entre si, mas o importante

para entender o funcionamento do projeto é ter o período do clock 4 vezes menor que o período da porta seletora ‘sel’,

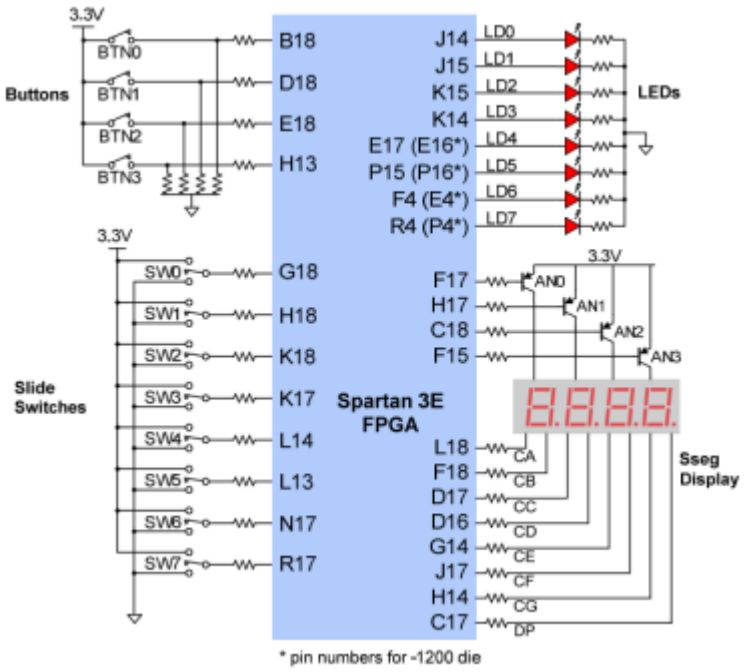
3 PROJETO 2

Neste projeto, o foco é alterar o Projeto 1, transformando os sinais de entrada (a, b, c, d), as saídas (x, y) e o sinal do multiplexador de vetores de 4 bits para sinais escalares de 1 bit. Essa abordagem permitirá uma análise mais focada na implementação e funcionalidade do multiplexador em um formato simplificado que é possível passar para a placa física XC3S500E.

3.1 MAPEAMENTO DOS PINOS

Para a implementação no kit Nexys 2, as entradas e saídas serão conectadas aos seguintes pinos: a será ligada a G18, b a H18, c a K18, d a K17, sel(1) a L14, sel(0) a L13. As saídas x e y estarão, respectivamente, conectadas a J14 e J15. É possível entender o que isso significa na figura abaixo, de um esquema de funcionamento da placa com os nomes das entradas e saídas, que demonstra que as saídas são LEDs e as entradas são *'switches'*.

FIGURA 4 – ESQUEMÁTICO DOS PINOS DA PLACA

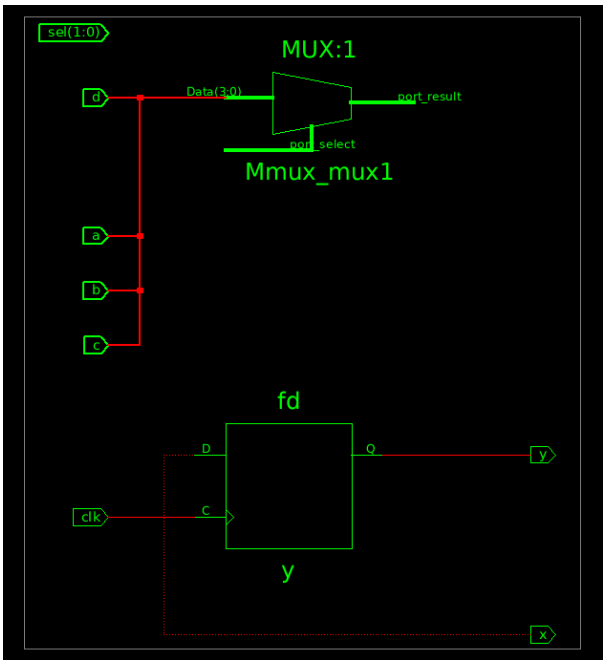


FONTE: Digilent Nexys2 Board Reference Manual (2011).

3.2 ESQUEMÁTICOS RTL

O esquemático RTL do projeto 2 está apresentado na figura abaixo, e mostra que as entradas e saídas do projeto.

FIGURA 5 – ESQUEMÁTICO RTL DO PROJETO 2

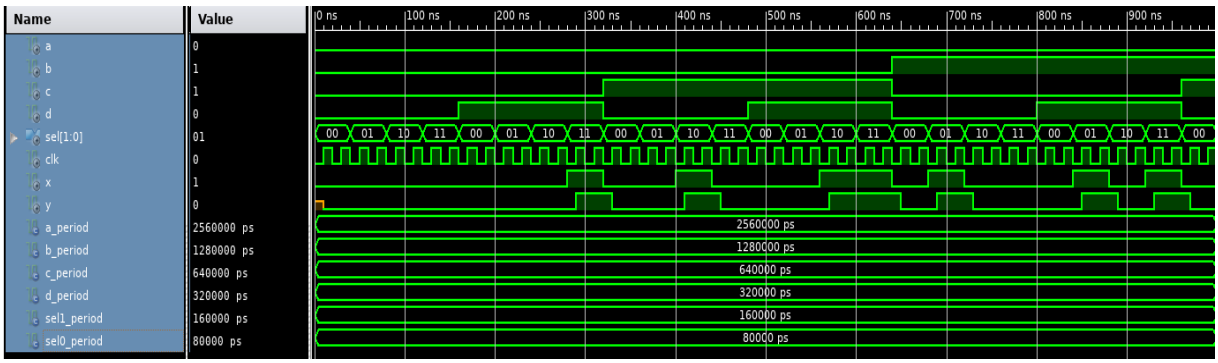


FONTE: As autoras (2024).

3.3 SIMULAÇÕES

As simulações foram realizadas de modo a testar todas as combinações possíveis do projeto. Assim, as entradas variam de maneira que o período de 'a' é o dobro do de 'b', que, por sua vez, é o dobro do de 'c', e assim por diante, com o clock sendo o de menor período. A figura abaixo apresenta uma parte da simulação, permitindo observar que os resultados de 'x' e 'y' funcionam de forma análoga aos resultados do Projeto 1, embora agora sejam representados como um escalar em vez de um valor de múltiplos bits.

FIGURA 6 – SIMULAÇÃO DO PROJETO 2



FONTE: As autoras (2024).

Ao observar a figura, podemos notar que 'x' e 'y' só assumem um valor diferente de 0 quando as portas seletoras correspondem a uma das entradas (a, b, c ou d) que não estão zeradas. Isso ocorre, por exemplo, quando a seletora se torna "11" pela segunda vez, momento em que 'x' se transforma imediatamente em 1. No entanto, 'y' só é ativado após a borda de subida do clock, ou seja, com um leve atraso em relação a 'x'.

3.4 CÓDIGO

Nessa seção será apresentado não apenas o código do VHDL Module, mas também o código realizado para que a simulação seja possível.

3.4.1 VHDL Module

O código VHDL Module, do projeto 2, está apresentado sem alterações no Anexo 3 deste presente trabalho. Todas as portas utilizadas foram definidas como abaixo.

```
Port ( a : in  STD_LOGIC;
       b : in  STD_LOGIC;
       c : in  STD_LOGIC;
       d : in  STD_LOGIC;
       sel : in  STD_LOGIC_VECTOR (1 downto 0);
       clk : in  STD_LOGIC;
       x : out  STD_LOGIC;
       y : out  STD_LOGIC);
```

Para realizar a lógica do multiplexador, foi realizado o seguinte código utilizando ‘if’ como base, já que se a porta seletora for de um jeito, a saída é a porta correspondente aquele valor, por isso, foi feito como abaixo.

```
mux <= a WHEN sel="00" ELSE
          b WHEN sel="01" ELSE
          c WHEN sel="10" ELSE
          d;
x<= mux;
```

Como, a porta ‘y’ só se torna o que ‘x’ é no evento de borda de subida do clock, foi necessário adicionar o seguinte código.

```
 PROCESS                                                    (clk)
begin
     if(clk'EVENT          and          clk='1')          then
      y<=mux;
     end                                                    if;
end process;
```

Ou seja, no evento de mudança do ‘clock’, sendo essa mudança, para quando o clock se torne ‘1’, ‘y’ se torna o que ‘x’ é, no caso desse exemplo, é também a variável “`signal mux: STD_LOGIC;`”

3.4.2 VHDL Test Bench

O código VHDL do Test Bench, referente ao projeto 2, é apresentado sem modificações no Anexo 4 deste trabalho. Observa-se que foi realizado um teste de simulação para verificar todas as combinações possíveis de entrada. Isso foi alcançado utilizando processos, como no exemplo abaixo, onde o período de cada entrada é o dobro do anterior, começando pelos menores períodos, que são da porta seletora e do clock, o menor de todos.

```
A_process                                                   :process
     begin
A                          <=                          '0';
wait                     for                      A_period/2;
A                          <=                          '1';
wait                     for                      A_period/2;
     end process;
```

ANEXO 1 – VHDL MODULE – PROJETO 1

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

entity reg_mux is
    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 reg_mux;

architecture Behavioral of reg_mux is

       signal mux: std_logic_vector (3 downto 0);

begin

       mux <= a when sel="00" else
                      b when sel="01" else
                      c when sel="10" else
                      d;
process(clk)
begin
       if(clk'event and clk='1') then
              y<= mux;
              end if;
end process;

       x<= mux;
end Behavioral;
```

ANEXO 2 – VHDL TEST BENCH – PROJETO 1

```
LIBRARY ieee;
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 reg_mux_tb IS
END reg_mux_tb;

ARCHITECTURE behavior OF reg_mux_tb IS

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

    COMPONENT reg_mux
    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) := "0010";
   signal b : std_logic_vector(3 downto 0) := "0100";
   signal c : std_logic_vector(3 downto 0) := "0110";
```

```
   signal d : std_logic_vector(3 downto 0) :=  "1000";
   signal sel : std_logic_vector(1 downto 0) := "00";
   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: reg_mux PORT MAP (
          a => a,
          b => b,
          c => c,
          d => d,
          sel => sel,
          clk => clk,
          x => x,
          y => y
        );

  ----------------------------------------------------------
  clk <= NOT clk AFTER  40ns;
  a <= "0011" AFTER 80ns,"0000" AFTER 640ns;
  b <= "0101" AFTER 240ns;
  c <= "0101" AFTER 400ns;
  d <= "0101" AFTER 560ns;
  sel <= "01" AFTER 160ns,
                "10" AFTER 320ns,
```

```
                "11" AFTER 480ns,
                "00" AFTER 640ns;

END;
```

ANEXO 3 – VHDL MODULE – PROJETO 2

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

entity mux_bit_unico is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
           sel : in  STD_LOGIC_VECTOR (1 downto 0);
           clk : in  STD_LOGIC;
           x : out  STD_LOGIC;
           y : out  STD_LOGIC);
end mux_bit_unico;
architecture Behavioral of mux_bit_unico is
signal mux: STD_LOGIC;
begin
       mux <= a WHEN sel="00" ELSE
                     b WHEN sel="01" ELSE
                     c WHEN sel="10" ELSE
                     d;
       x<= mux;
PROCESS (clk)
       begin
              if(clk'EVENT and clk='1') then
               y<=mux;
              end if;
       end process;
end Behavioral;
```

ANEXO 4 – VHDL TEST BENCH – PROJETO 2

```
        LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY mux_unico_tb IS
END mux_unico_tb;
ARCHITECTURE behavior OF mux_unico_tb IS
COMPONENT mux_bit_unico

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

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

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

   -- Clock period definitions
constant A_period : time :=  2560 ns;
constant B_period : time :=  1280 ns;
constant C_period : time :=  640 ns;
constant D_period : time :=  320 ns;
constant SEL1_period : time := 160 ns;
constant SEL0_period : time := 80 ns;

BEGIN
```

```
    -- Instantiate the Unit Under Test (UUT)
uut: mux_bit_unico PORT MAP (
       a => a,
       b => b,
       c => c,
       d => d,
       sel => sel,
       clk => clk,
       x => x,
       y => y
     );
     -- Clock process definitions
           A_process :process
           begin
    A <= '0';
    wait for A_period/2;
    A <= '1';
    wait for A_period/2;
           end process;

    B_process :process
           begin
    B <= '0';
    wait for B_period/2;
    B <= '1';
    wait for B_period/2;
           end process;

    C_process :process
           begin
    C <= '0';
    wait for C_period/2;
    C <= '1';
    wait for C_period/2;
           end process;

    D_process :process
           begin
    D <= '0';
    wait for D_period/2;
    D <= '1';
    wait for D_period/2;
           end process;
```

```
        SEL1_process :process
                begin
        sel(1) <= '0';
        wait for SEL1_period/2;
        sel(1) <= '1';
        wait for SEL1_period/2;
                end process;

        SEL0_process :process
                begin
        sel(0) <= '0';
        wait for SEL0_period/2;
        sel(0) <= '1';
        wait for SEL0_period/2;
                end process;

  clk <= NOT clk AFTER  10ns;


END;
```