Desenvolvimento da Primeira Lista de VHDL

Aluno: Gabriel Cavalcanti Coelho

Este documento tem como objetivo apresentar as resoluções das questões da primeira lista de VHDL fornecida pelos mentores. Serão incluídas capturas de tela dos RTLs elaborados e das simulações realizadas no Vivado, acompanhadas de breves descrições sobre o funcionamento de cada um. Além disso, o código-fonte será transcrito neste documento e um link para o repositório no GitHub será disponibilizado, onde estarão todos os códigos e prints vistos aqui.

Link do Repositório no GitHub

https://github.com/GabrielC248/Atv_Lista_VHDL_01_Gabriel_Cavalcanti_Coelho.git

Primeira Questão

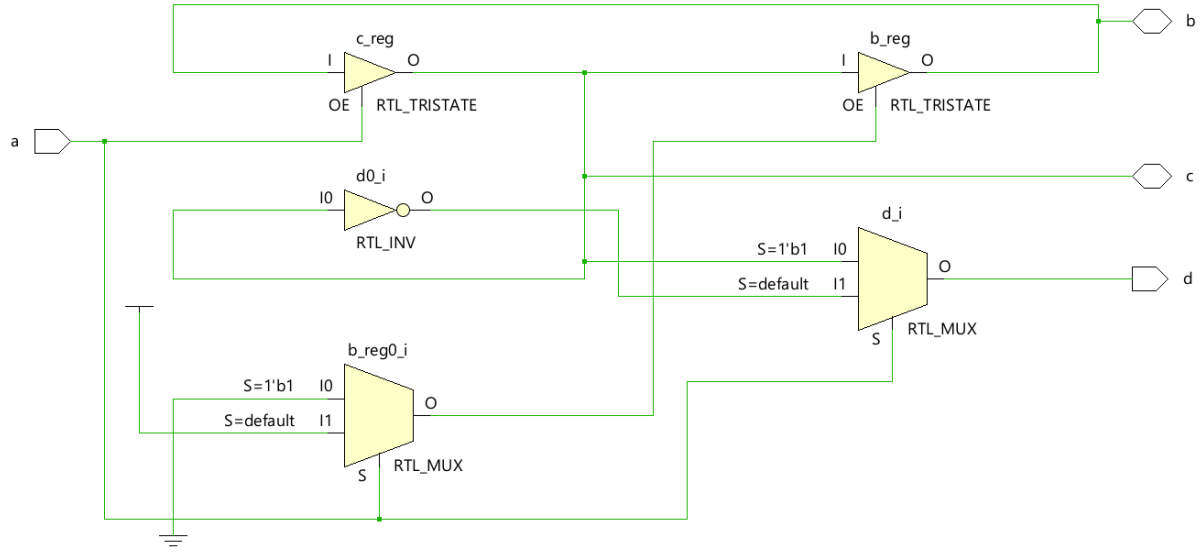
Código Transcrito do Circuito

```
library IEEE;
use IEEE.std_logic_1164.all;

entity circuit01 is
    port (
        a : in std_logic;
        b : inout std_logic;
        c : inout std_logic;
        d : out std_logic
    );
end entity circuit01;

architecture arch of circuit01 is

    signal b_reg0 : std_logic; -- "Fio" do inverso de a
    signal d0 : std_logic;     -- "Fio" do inverso de c

begin

    -- Buffer tri-state que passa o valor de b para c quando a = 1
    c <= b when a = '1' else 'Z';

    -- Buffer tri-state que passa o valor de c para b quando a = 0
    b <= c when b_reg0 = '1' else 'Z';
```

```
    -- Inversor do valor de c
    d0 <= not c;

    -- REMOVER COMENTARIOS CASO QUEIRA O RTL IGUAL AO VISTO NA QUESTAO,
MAS QUE NAO FUNCIONA CONFORME O ENUNCIADO
    -- Mux que define que quando a = 1 o valor de d eh c, quando a = 0
o valor de d eh o not c
    -- mux_01: process(a)
    -- begin
    --     case a is
    --         when '1' =>
    --             d <= c;
    --         when others =>
    --             d <= d0;
    --     end case;
    -- end process mux_01;

    -- Atribui a d o valor invertido de c (COMENTAR CASO QUEIRA O RTL
IGUAL AO VISTO NA QUESTAO)
    d <= d0;

    -- Mux que inverte a saida de a para o controle do buffer tristate
de b
    mux_02: process(a)
    begin
        case a is
            when '1' =>
                b_reg0 <= '0';
            when others =>
                b_reg0 <= '1';
        end case;
    end process mux_02;

end architecture arch;
```

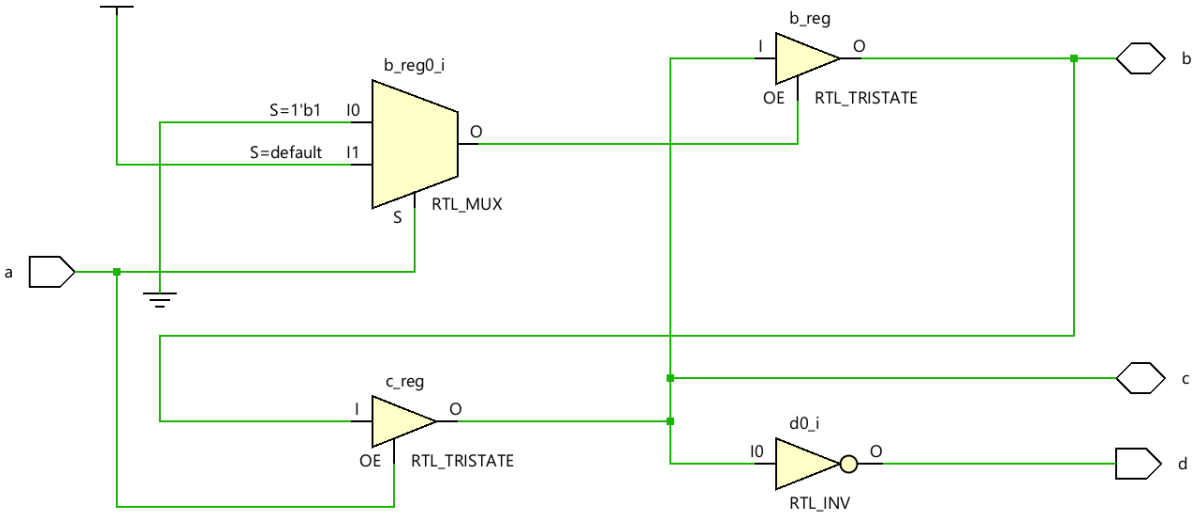
RTLs Elaborados

RTL Elaborado com o Mux da Saída d:



Fonte: Elaboração própria.

RTL da Versão Funcional:



Fonte: Elaboração própria.

A arquitetura da entidade desenvolvida inicia com a declaração dos sinais internos. Um sinal é utilizado para armazenar o **inverso da entrada a**, que controla o buffer responsável por direcionar o **valor de c para b**. O outro sinal armazena o **inverso de c**, que vai para a **saída d**. Em seguida, são implementados os **buffers tri-state** que constituem o barramento bidirecional entre b e c, assim como o circuito inversor de c.

O código do multiplexador (mux) feito com base no **RTL do enunciado foi mantido entre comentários**, pois sua implementação não resultou no funcionamento solicitado. Portanto, para garantir o comportamento esperado pela descrição da questão, o valor invertido de c foi atribuído diretamente à saída d. Por fim, implementou-se o **mux** que **inverte** a **entrada a** para controlar o **buffer tri-state de b**.

Código Transcrito do Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_circuit01 is
end entity tb_circuit01;

architecture testbench of tb_circuit01 is

    -- Sinais do testbench
    signal a_tb, b_tb, c_tb, d_tb : std_logic;

    -- Declara o circuito da questao
    component circuit01 is
        port (
            a : in std_logic;
            b : inout std_logic;
            c : inout std_logic;
            d : out std_logic
        );
    end component circuit01;

begin

    -- Instancia o circuito da questao
    uut : circuit01
        port map (
            a => a_tb,
            b => b_tb,
            c => c_tb,
            d => d_tb
        );

    -- Sequencia de estimulos para testar o circuito da questao
    stimulus: process
    begin
        -- Inicializa o teste com todos os sinais como 0
        a_tb <= '0';
        b_tb <= '0';
        c_tb <= '0';
        wait for 10 ps;
```

```
        -- Teste para verificar se o valor de b esta sendo passado para
c quando a = 1
        a_tb <= '1'; -- Sinal de controle
        b_tb <= '1'; -- Sinal "ativo"
        c_tb <= 'Z'; -- Alta impedancia em c para receber o valor de b
        wait for 10 ps;
        b_tb <= '0';
        wait for 10 ps;
        b_tb <= '1';
        wait for 10 ps;
        b_tb <= '0';
        wait for 10 ps;

        -- Pausa para facilitar a visualizacao no waveform
        wait for 20 ps;

        -- Teste para verificar se o valor de c esta sendo passado para
b quando a = 0
        a_tb <= '0'; -- Sinal de controle
        b_tb <= 'Z'; -- Alta impedancia em b para receber o valor de c
        c_tb <= '1'; -- Sinal "ativo"
        wait for 10 ps;
        c_tb <= '0';
        wait for 10 ps;
        c_tb <= '1';
        wait for 10 ps;
        c_tb <= '0';
        wait for 10 ps;

        -- Fim do teste
        a_tb <= '0';
        b_tb <= '0';
        c_tb <= '0';
        wait;
    end process stimulus;

end architecture testbench;
```

Para a construção do testbench, primeiro declarei e instanciei o módulo, conectando-o aos sinais internos do testbench. Em seguida, iniciei a sequência de testes.

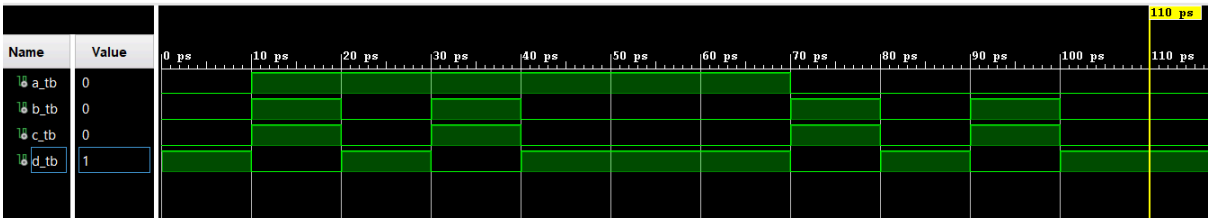
Primeiro, todos os **sinais** foram **zerados** durante **10 ps**. Depois, para testar o **fluxo de b para c**, o sinal **a** foi para **'1'** e **c** foi colocado em **alta impedância ('Z')** para poder receber

os dados. O valor de **b** foi então **alterado por 40 ps** para garantir que c o acompanhava corretamente.

Após uma pequena **pausa** de **20 ps**, o mesmo foi feito para **testar o fluxo de c para b**. O sinal **a** foi para **'0'**, **b** foi colocado em **alta impedância** e os **valores de c** foram **variados** para verificação. Ao final, os sinais foram novamente zerados para concluir o teste.

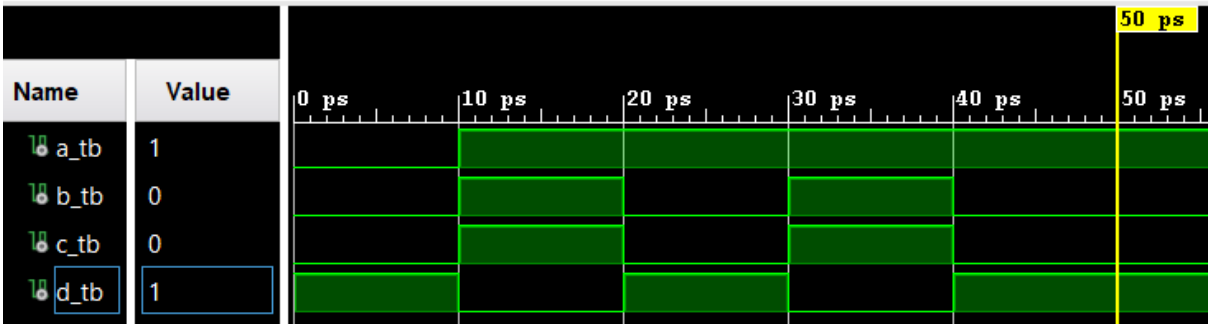
Prints do Testbench

Simulação Completa:



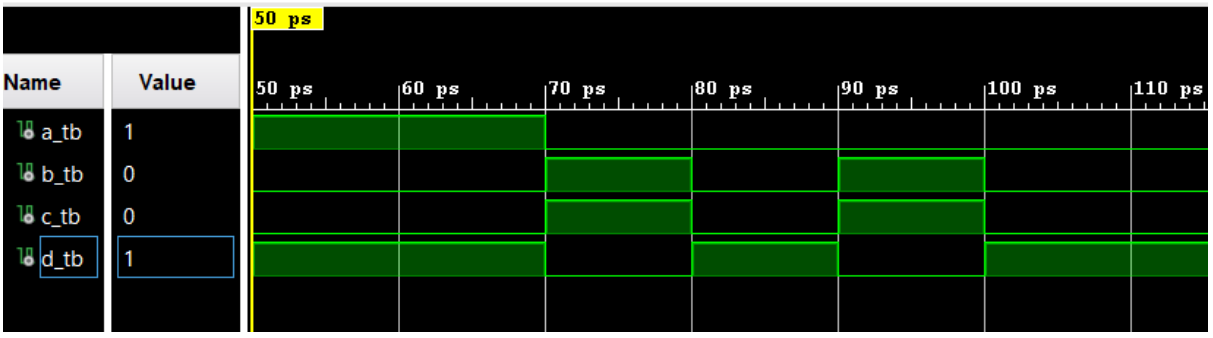
Fonte: Elaboração própria.

Verificação de a = ‘1’ (b para c):



Fonte: Elaboração própria.

Verificação de a = ‘0’ (c para b):



Fonte: Elaboração própria.

Segunda Questão

Código Transcrito do Contador Integer

```
library IEEE;
```

```vhdl
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity integer_counter is
    port (
        clk : in std_logic;
        rst : in std_logic;
         output : out integer range -128 to 127 -- Output definido como
inteiro de -128 ate 127
    );
end entity integer_counter;

architecture behavior of integer_counter is

    -- Sinal de contagem do contador que vai para a saida
    signal cnt : integer range -128 to 127;

begin

    -- Definicao do contador
    counter: process(clk, rst) -- Lista sensitiva com o clk e o rst
    begin
         if rst = '1' then -- Se reset entao cnt(saida) recebe o menor
valor (-128)
            cnt <= -128;
          elsif rising_edge(clk) then -- Se borda de subida do clock
incrementa a contagem ou volta para o menor valor caso chegue no maior
valor
            if cnt >= 127 then
                cnt <= -128;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process counter;

    -- Atribui a contagem a saida
    output <= cnt;

end architecture behavior;
```

Código Transcrito do Contador Signed

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity signed_counter is
    port (
        clk : in std_logic;
        rst : in std_logic;
         output : out signed(7 downto 0) -- Output definido como inteiro
com sinal
    );
end entity signed_counter;

architecture behavior of signed_counter is

    -- Sinal de contagem do contador que vai para a saida
    signal cnt : signed(7 downto 0);

begin

    -- Definicao do contador
    counter: process(clk, rst) -- Lista sensitiva com o clk e o rst
    begin
         if rst = '1' then -- Se reset entao cnt(saida) recebe o menor
valor (10000000)
            cnt <= (7 => '1', others => '0');
          elsif rising_edge(clk) then -- Se borda de subida do clock
incrementa a contagem
            cnt <= cnt + 1;
        end if;
    end process counter;

    -- Atribui a contagem a saida
    output <= cnt;

end architecture behavior;
```

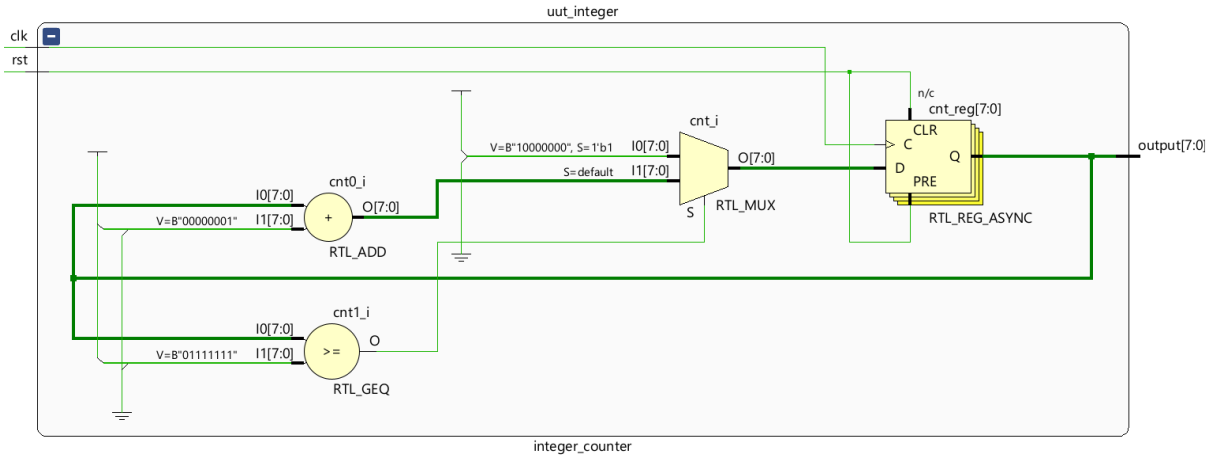
Código Transcrito do Contador Unsigned

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```vhdl
entity unsigned_counter is
    port (
        clk : in std_logic;
        rst : in std_logic;
           output : out unsigned(7 downto 0) -- Output definido como
inteiro sem sinal
    );
end entity unsigned_counter;

architecture behavior of unsigned_counter is

    -- Sinal de contagem do contador que vai para a saida
    signal cnt : unsigned(7 downto 0);

begin

    -- Definicao do contador
    counter: process(clk, rst) -- Lista sensitiva com o clk e o rst
    begin
         if rst = '1' then -- Se reset entao cnt(saida) recebe o menor
valor (00000000)
            cnt <= (others => '0');
          elsif rising_edge(clk) then -- Se borda de subida do clock
incrementa a contagem
            cnt <= cnt + 1;
        end if;
    end process counter;

    -- Atribui a contagem a saida
    output <= cnt;

end architecture behavior;
```

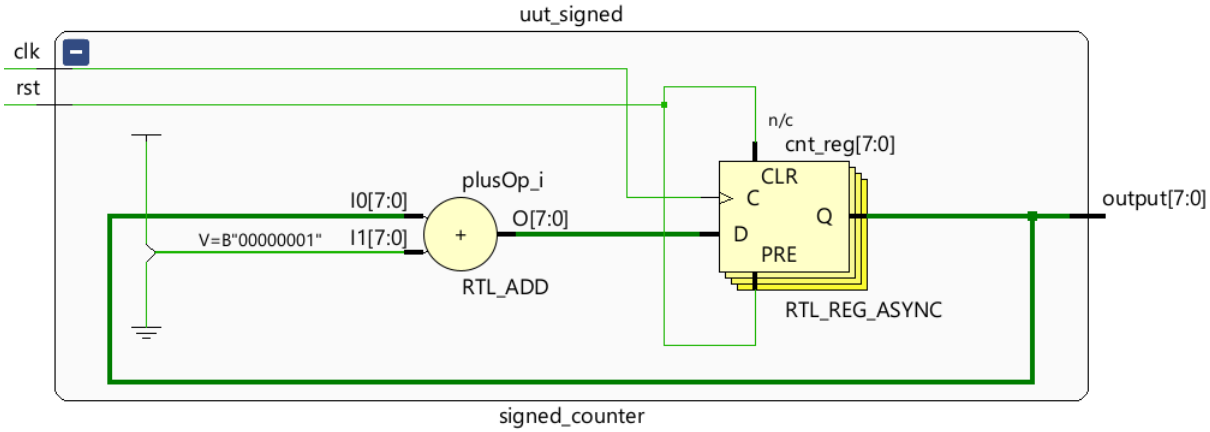
RTLs Elaborados

RTL do Contador Integer:



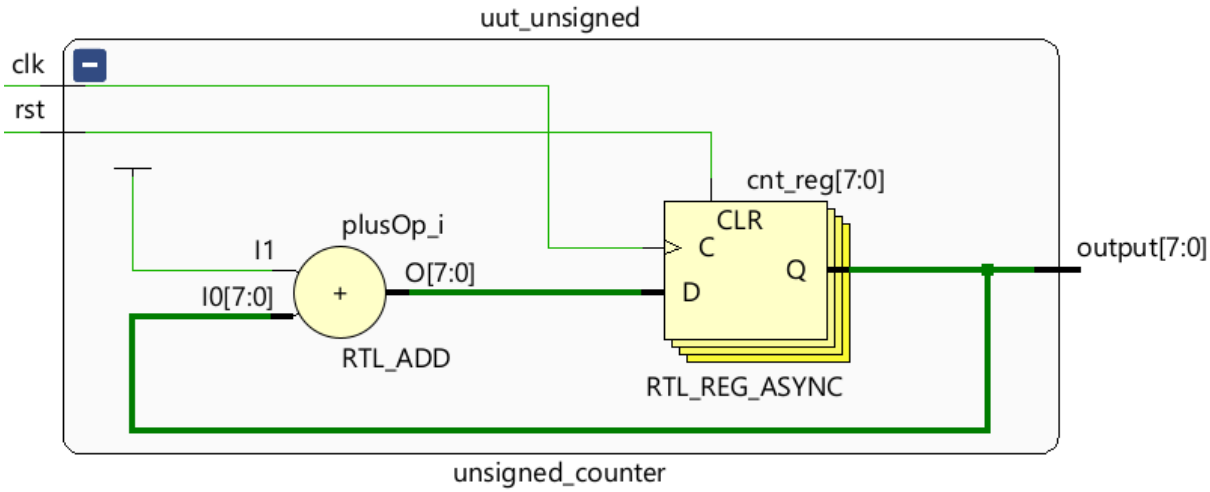
Fonte: Elaboração própria.

RTL do Contador Signed:



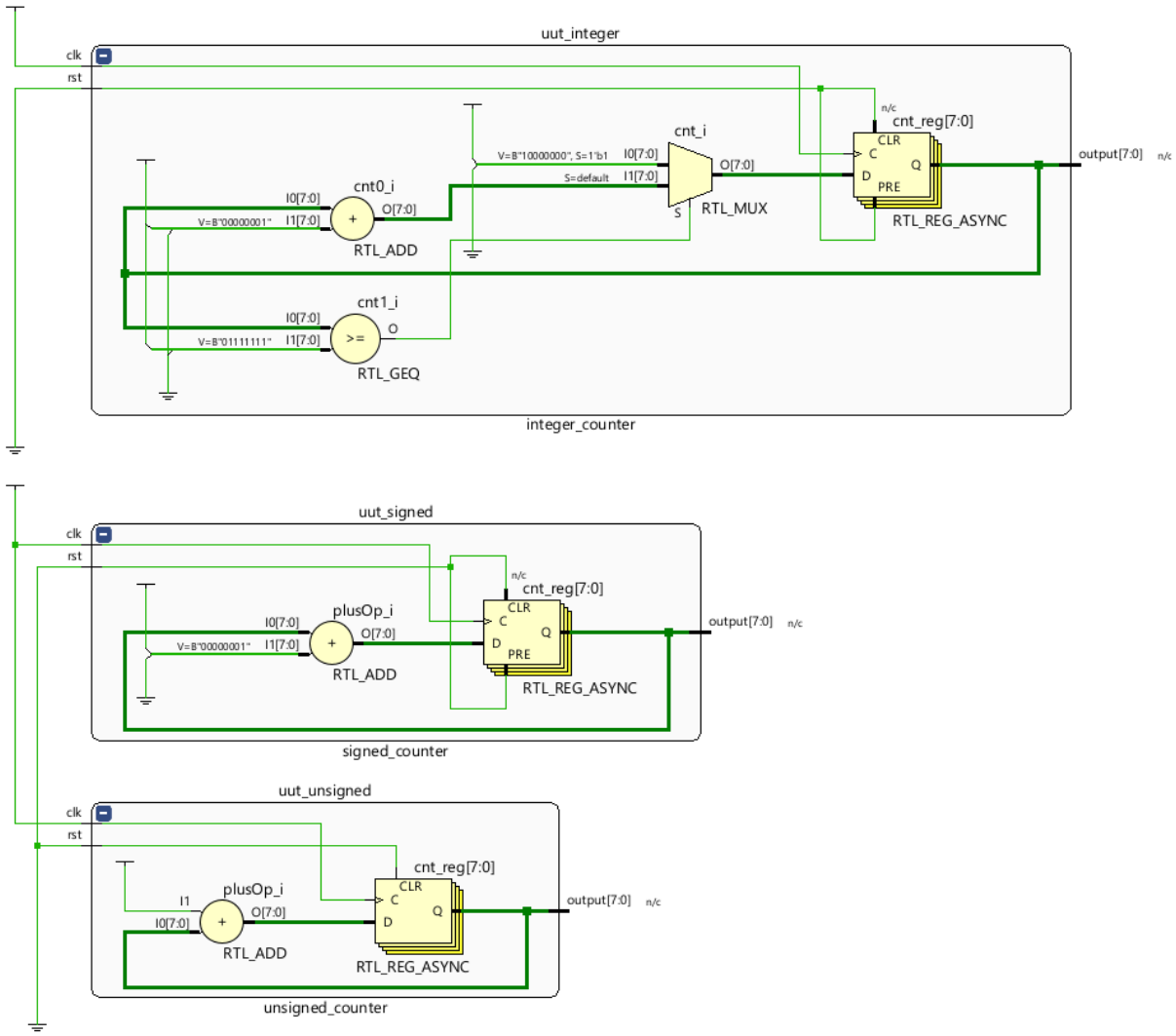
Fonte: Elaboração própria.

RTL do Contador Unsigned:



Fonte: Elaboração própria.

RTL do Testbench:



Fonte: Elaboração própria.

Todos os contadores desenvolvidos compartilham o **mesmo princípio**. Um sinal interno, do mesmo tipo da porta de saída, é utilizado para armazenar o valor da contagem. A lógica está contida em um processo com um reset assíncrono no qual, se a entrada de **reset** estiver em nível lógico **alto**, a contagem é **reiniciada** para seu **valor mínimo**, caso contrário, a cada **borda de subida do clock**, o contador é **incrementado** em uma unidade.

A única diferença notável foi observada na implementação com o tipo integer. Neste, ao declarar um integer sem range especificado no testbench, o contador não realizava o retorno de 127 para -128 ao chegar no maior valor (01111111 + 1 = 10000000). Em vez disso, ele continuava a incrementar para valores acima de 127. Para forçar o comportamento correto, foi adicionada uma verificação para que quando a contagem atinja o valor máximo, no próximo pulso de clock, o contador seja redefinido para seu valor inicial (-128).

Código Transcrito do Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```vhdl
use IEEE.numeric_std.all;

entity tb_counters is
end entity tb_counters;

architecture testbench of tb_counters is

    signal clk_tb, rst_tb : std_logic; -- Sinais comuns dos contadores

     signal output_integer : integer; -- Output do contador de inteiro
de -128 ate 127
     signal output_signed : signed(7 downto 0); -- Output do contador de
inteiro com sinal
       signal  output_unsigned  :  unsigned(7  downto  0);  --  Output  do
contador de inteiro sem sinal

    -- Declaracao do contador de inteiro de -128 ate 127
    component integer_counter
        port (
            clk : in std_logic;
            rst : in std_logic;
            output : out integer range -128 to 127
        );
    end component integer_counter;

    -- Declaracao do contador de inteiro com sinal
    component signed_counter
        port (
            clk : in std_logic;
            rst : in std_logic;
            output : out signed(7 downto 0)
        );
    end component signed_counter;

    -- Declaracao do contador de inteiro sem sinal
    component unsigned_counter
        port (
            clk : in std_logic;
            rst : in std_logic;
            output : out unsigned(7 downto 0)
        );
    end component unsigned_counter;
```

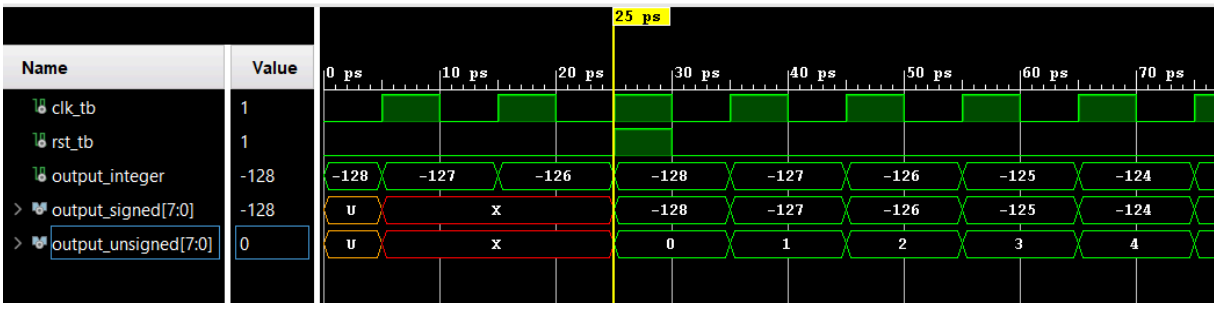
```
begin

    -- Instanciacao do contador de inteiro de -128 ate 127
    uut_integer : integer_counter
        port map (
            clk => clk_tb,
            rst => rst_tb,
            output => output_integer
        );

    -- Instanciacao do contador de inteiro com sinal
    uut_signed : signed_counter
        port map (
            clk => clk_tb,
            rst => rst_tb,
            output => output_signed
        );

    -- Instanciacao do contador de inteiro sem sinal
    uut_unsigned : unsigned_counter
        port map (
            clk => clk_tb,
            rst => rst_tb,
            output => output_unsigned
        );

    -- Gera o clock do testbench
    clock_gen: process
    begin
        clk_tb <= '0';
        wait for 5 ps;
        clk_tb <= '1';
        wait for 5 ps;
    end process clock_gen;

    -- Aciona o reset em tempos definidos para teste
    reset_gen: process
    begin
        rst_tb <= '0';

        -- Ativa o reset no inicio e na borda de subida do clock
        wait for 25 ps;
        rst_tb <= '1';
```

```
        wait for 5 ps;
        rst_tb <= '0';

         -- Ativa o reset um pouco depois da primeira contagem completa
na borda de descida do clock
        wait for 2670 ps;
        rst_tb <= '1';
        wait for 5 ps;
        rst_tb <= '0';

        wait;
    end process reset_gen;

end architecture testbench;
```

Para o testbench, fiz a declaração e a instanciação dos três módulos realizados dos contadores, conectando um **sinal comum** de **clock** e **reset** do testbench neles. O processo de teste consistiu na geração de um sinal de **clock** contínuo com período de **10 ps** e no controle do **reset assíncrono**, que foi ativado uma vez no início da simulação, após **25 ps**, e outra vez em **2700 ps**, um pouco depois da primeira contagem completa, para verificar a funcionalidade de reinicialização dos contadores.

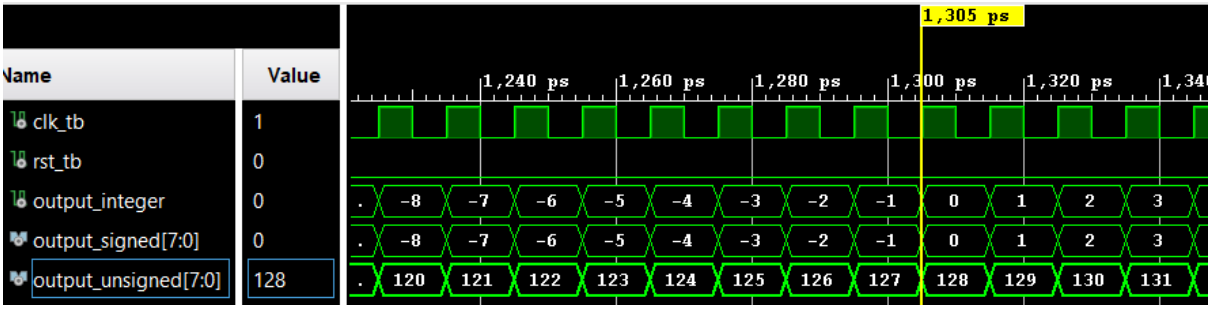
Prints do Testbench

Início da Simulação:



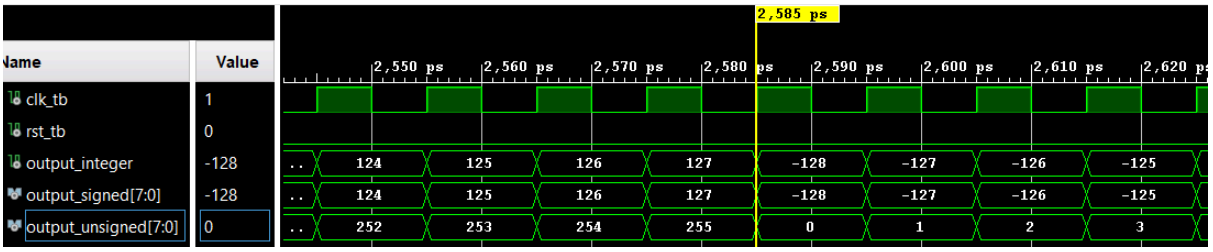
Fonte: Elaboração própria.

Meio da Contagem:



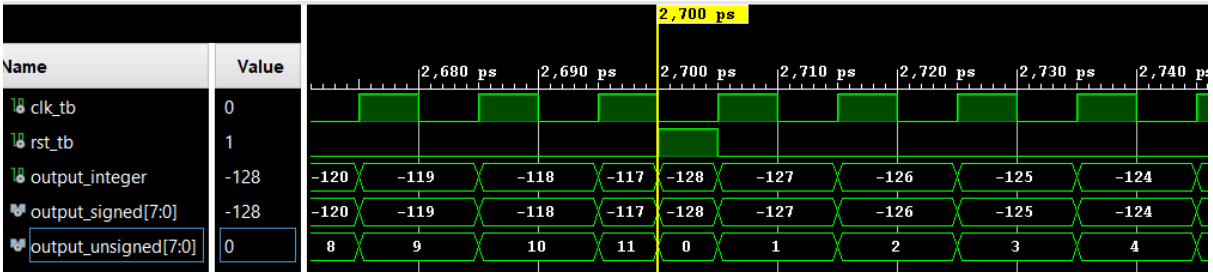
Fonte: Elaboração própria.

Fim da Primeira Contagem:



Fonte: Elaboração própria.

Segundo Reset:



Fonte: Elaboração própria.

Terceira Questão

Código Transcrito das Arquiteturas dos Muxes

```
library IEEE;
use IEEE.std_logic_1164.all;

entity custom_mux is
    port (
        sel : in std_logic_vector(1 downto 0);
        input1 : in std_logic_vector(3 downto 0);
        input2 : in std_logic_vector(3 downto 0);
        input3 : in std_logic_vector(3 downto 0);
        input4 : in std_logic_vector(3 downto 0);
        output : out std_logic_vector(3 downto 0)
    );
end entity custom_mux;

architecture arch_case of custom_mux is -- Arquitetura do mux com o
case

begin

    mux: process(sel, input1, input2, input3, input4)
```

```
    begin
        case sel is
            when "00" =>
                output <= input1;
            when "01" =>
                output <= input2;
            when "10" =>
                output <= input3;
            when others =>
                output <= input4;
        end case;
    end process mux;

end architecture arch_case;

architecture arch_if of custom_mux is -- Arquitetura do mux com o
if-elsif

begin

    mux: process(sel, input1, input2, input3, input4)
    begin
        if (sel = "00") then
            output <= input1;
        elsif (sel = "01") then
            output <= input2;
        elsif (sel = "10") then
            output <= input3;
        else
            output <= input4;
        end if ;
    end process mux;

end architecture arch_if;

architecture arch_when of custom_mux is -- Arquitetura do mux com o
when-else

begin

    output <= input1 when sel = "00" else
              input2 when sel = "01" else
              input3 when sel = "10" else
```

```
              input4;

end architecture arch_when;

architecture arch_with of custom_mux is -- Arquitetura do mux com o
with-select

begin

    with sel select output <=
        input1 when "00",
        input2 when "01",
        input3 when "10",
        input4 when others;

end architecture arch_with;
```

Código Transcrito do Tipo bus_type

```
library IEEE;
use IEEE.std_logic_1164.all;

package custom_types is

     type bus_type is array (natural range <>) of std_logic_vector(3
downto 0);

end package custom_types;
```

Código Transcrito do Circuito do RTL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library work;
use  work.custom_types.all; -- Biblioteca para o tipo bus_type "array
(natural range <>) of std_logic_vector(3 downto 0)"

entity circuit01 is
    port (
        en : in std_logic;
        clk : in std_logic;
        rst : in std_logic;
```

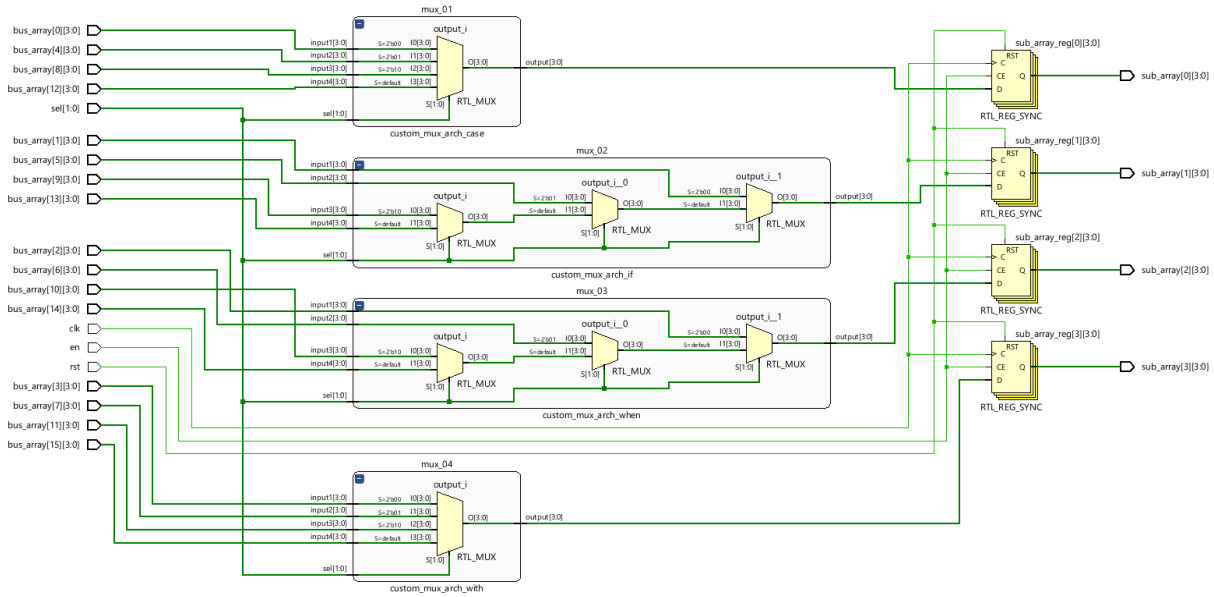
```
        sel : in std_logic_vector(1 downto 0);
        bus_array : in bus_type(15 downto 0);
        sub_array : out bus_type(3 downto 0)
    );
end entity circuit01;

architecture arch of circuit01 is

    signal array_reg_wire : bus_type(3 downto 0); -- "Fio" para a saida
dos muxes

    -- Declaracao do componente mux
    component custom_mux
        port (
            sel : in std_logic_vector(1 downto 0);
            input1 : in std_logic_vector(3 downto 0);
            input2 : in std_logic_vector(3 downto 0);
            input3 : in std_logic_vector(3 downto 0);
            input4 : in std_logic_vector(3 downto 0);
            output : out std_logic_vector(3 downto 0)
        );
    end component custom_mux;

begin

    mux_01 : entity work.custom_mux(arch_case) -- Instancia do primeiro
mux com a versao case
        port map(
            sel => sel,
            input1 => bus_array(0),
            input2 => bus_array(4),
            input3 => bus_array(8),
            input4 => bus_array(12),
            output => array_reg_wire(0)
        );

     mux_02 : entity work.custom_mux(arch_if) -- Instancia do segundo
mux com a versao if-elsif
        port map(
            sel => sel,
            input1 => bus_array(1),
            input2 => bus_array(5),
            input3 => bus_array(9),
```

```
            input4 => bus_array(13),
            output => array_reg_wire(1)
        );

    mux_03 : entity work.custom_mux(arch_when) -- Instancia do terceiro
mux com a versao when-else
        port map(
            sel => sel,
            input1 => bus_array(2),
            input2 => bus_array(6),
            input3 => bus_array(10),
            input4 => bus_array(14),
            output => array_reg_wire(2)
        );

     mux_04 : entity work.custom_mux(arch_with) -- Instancia do quarto
mux com a versao with-select
        port map(
            sel => sel,
            input1 => bus_array(3),
            input2 => bus_array(7),
            input3 => bus_array(11),
            input4 => bus_array(15),
            output => array_reg_wire(3)
        );

     array_reg: process(clk) -- Registrador sincrono para atualizar o
array de saida com base no clock e em um sinal de habilitacao
    begin
        if rising_edge(clk) then
             if (rst = '1') then -- Definicao do reset (todas as saidas
recebem 0)
                sub_array(0) <= (others => '0');
                sub_array(1) <= (others => '0');
                sub_array(2) <= (others => '0');
                sub_array(3) <= (others => '0');
             else -- Atualizacao da saida caso o enable esteja em sinal
alto
                if (en = '1') then
                    sub_array(0) <= array_reg_wire(0);
                    sub_array(1) <= array_reg_wire(1);
                    sub_array(2) <= array_reg_wire(2);
                    sub_array(3) <= array_reg_wire(3);
```

```
                end if;
            end if;
        end if;
    end process array_reg;

end architecture arch;
```

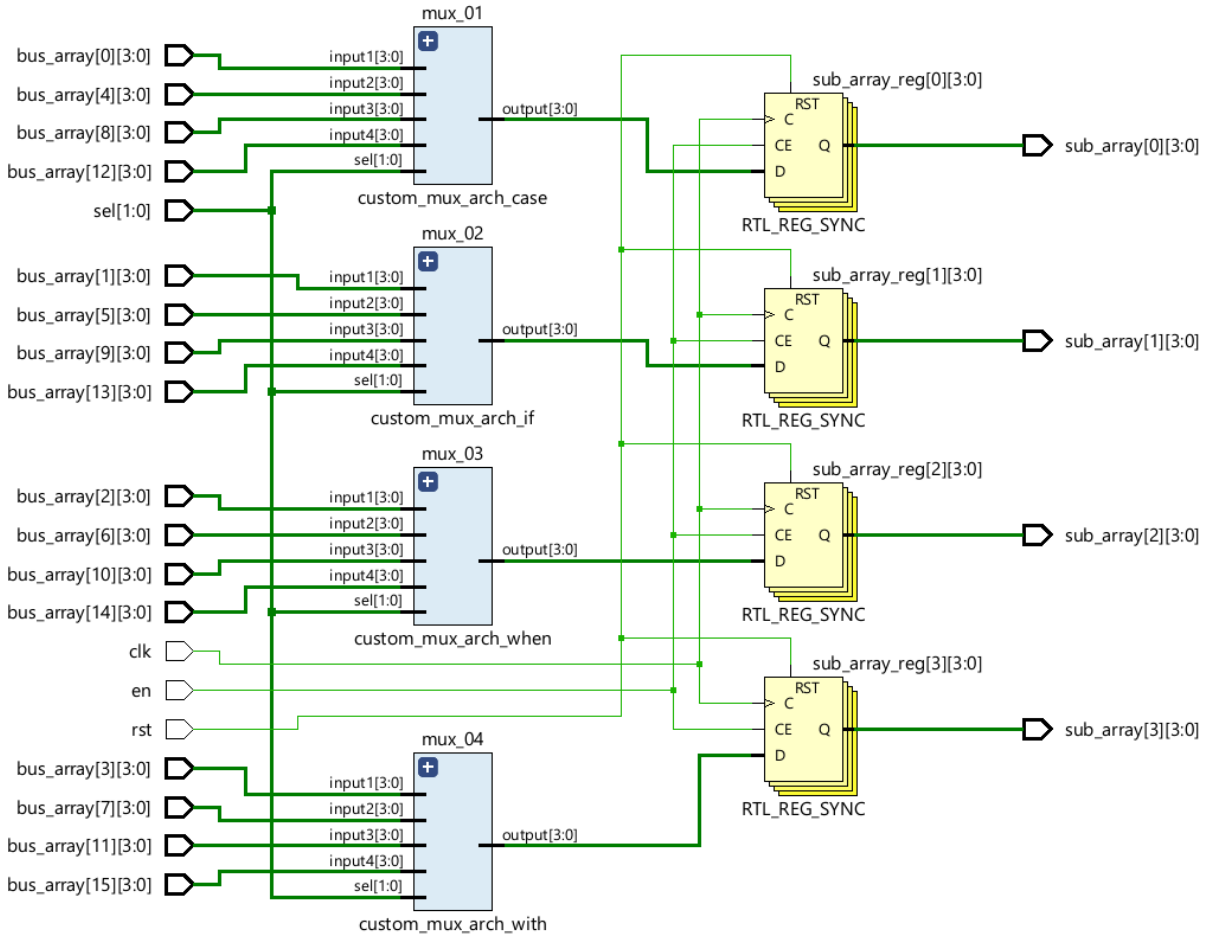
RTLs Elaborados

RTL Completo:



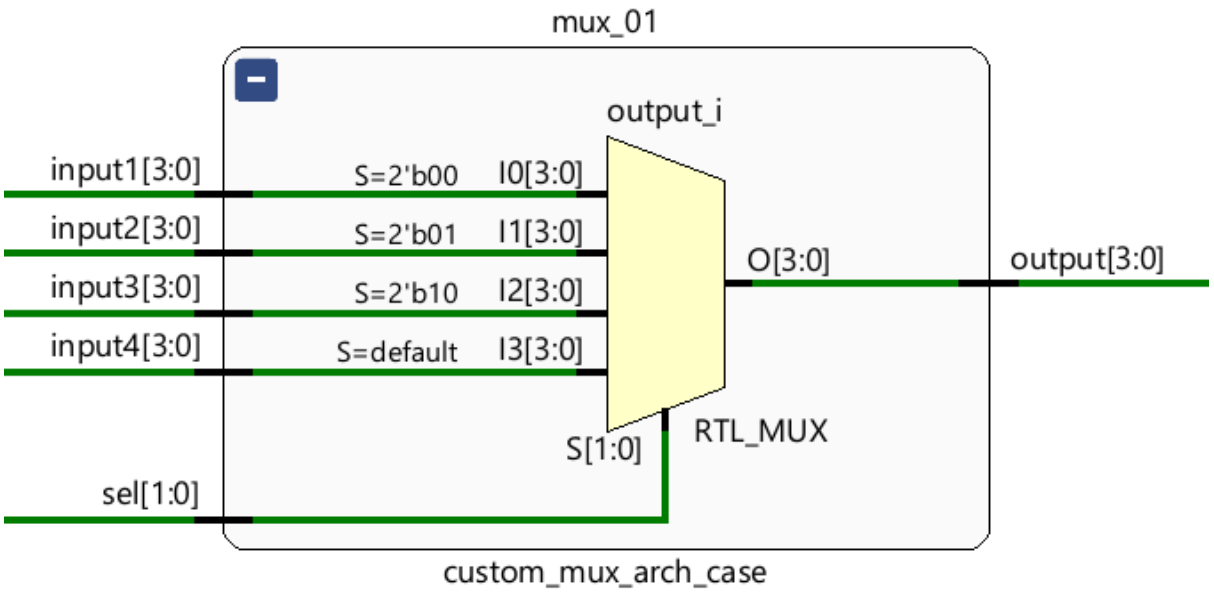
Fonte: Elaboração própria.

RTL Completo com os Blocos Fechados:



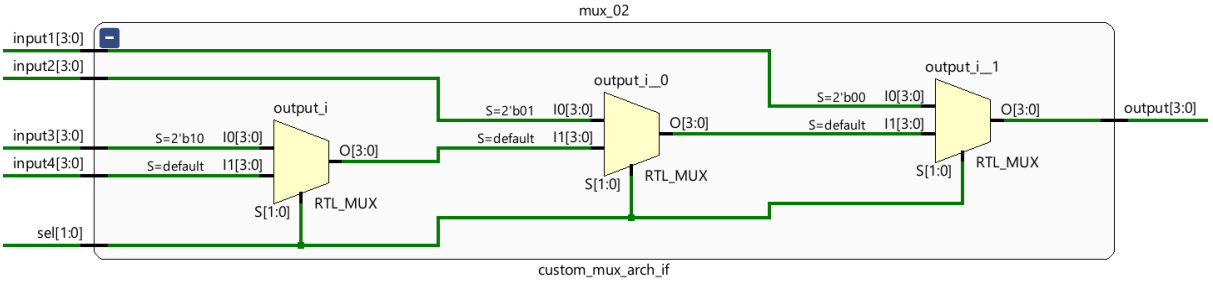
Fonte: Elaboração própria.

RTL do Mux com o case:



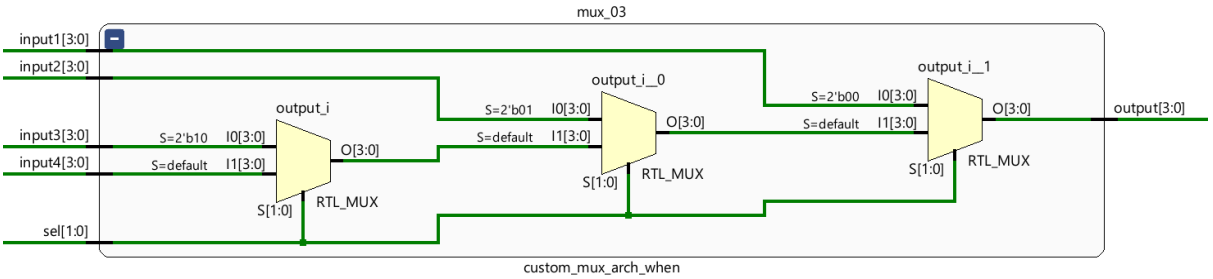
Fonte: Elaboração própria.

RTL do Mux com o if-elsif:



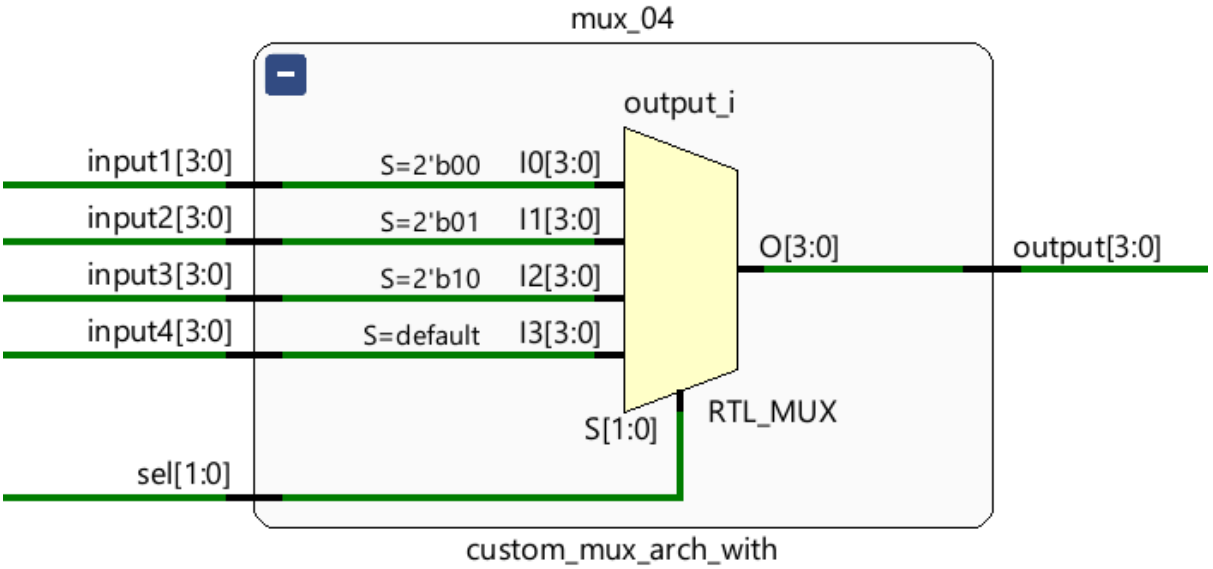
Fonte: Elaboração própria.

RTL do Mux com o when-else:



Fonte: Elaboração própria.

RTL do Mux com o with-select:



Fonte: Elaboração própria.

Para a implementação, foi criada a entidade **custom_mux** com **quatro arquiteturas distintas**, aplicando as estruturas case, if-elsif, when-else e with-select, conforme solicitado na questão. Em seguida, um arquivo **custom_types** foi desenvolvido para definir o tipo de **barramento** (bus_type) da questão. No módulo principal, o **circuit01**, foram instanciados quatro desses multiplexadores, cada um utilizando uma das arquiteturas (nas mentorias, ficou entendido que a entrega poderia ser feita dessa forma). Esses componentes foram conectados para, com base na entrada sel, selecionar um subconjunto de 4 vetores do barramento de entrada de 16 vetores, conforme visto no RTL da questão.

Por fim, foi implementado um processo síncrono que atua como um registrador de saída: na borda de subida do clock, se o **reset** estiver **ativo**, a **saída** é **zerada**; caso contrário, e se o sinal **en** (enable) estiver **habilitado**, os dados selecionados pelos multiplexadores são transferidos para a saída.

Código Transcrito do Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library work;
use  work.custom_types.all;  -- Biblioteca  para  o  tipo  bus_type  "array
(natural range <>) of std_logic_vector(3 downto 0)"

entity tb_circuit01 is
end entity tb_circuit01;

architecture testbench of tb_circuit01 is

    -- Definicao do periodo do clock
    constant T : time := 10 ps;

    -- Sinais do tb para o circuito feito
    signal tb_en, tb_clk, tb_rst : std_logic;
    signal tb_sel : std_logic_vector(1 downto 0);
    signal tb_bus_array : bus_type(15 downto 0);
    signal tb_sub_array : bus_type(3 downto 0);

    -- Declaracao do circuito feito
    component circuit01
        port (
            en : in std_logic;
            clk : in std_logic;
            rst : in std_logic;
            sel : in std_logic_vector(1 downto 0);
            bus_array : in bus_type(15 downto 0);
            sub_array : out bus_type(3 downto 0)
        );
    end component circuit01;

begin

    -- Instanciacao do circuito feito
```

```
    uut :  circuit01
        port map (
            en => tb_en,
            clk => tb_clk,
            rst => tb_rst,
            sel => tb_sel,
            bus_array => tb_bus_array,
            sub_array => tb_sub_array
        );

    -- Gera o clock do testbench
    clock_gen: process
    begin
        tb_clk <= '0';
        wait for (T/2);
        tb_clk <= '1';
        wait for (T/2);
    end process clock_gen;

    -- Acionamento do reset
    reset_gen: process
    begin
        -- Aciona o reset logo no comeco
        tb_rst <= '1';
        wait for (T);
        tb_rst <= '0';

        -- Aciona o reset um pouco depois do primeiro teste
        wait for (10*T);
        tb_rst <= '1';
        wait for (T);
        tb_rst <= '0';

        wait;
    end process reset_gen;

    -- Estimulos fornecidos para realizar o testbench
    stimulus: process
    begin
        -- Inicia com tudo zerado
        tb_sel <= "00";
        tb_bus_array <= (others => (others => '0'));
```

```
        -- Habilita a saida
        tb_en <= '1';

            -- Atribui a cada posicao do vetor um valor de 4 bits
"correspondente" a posicao para facilitar a visualizacao na forma de
onda
        wait for (2*T);
        tb_bus_array <= (
                0  => "0000",
                1  => "0001",
                2  => "0010",
                3  => "0011",
                4  => "0100",
                5  => "0101",
                6  => "0110",
                7  => "0111",
                8  => "1000",
                9  => "1001",
                10 => "1010",
                11 => "1011",
                12 => "1100",
                13 => "1101",
                14 => "1110",
                15 => "1111"
            );
        wait for (2*T);
        tb_sel <= "01"; -- Testa o sel quando em 01
        wait for (2*T);
        tb_sel <= "10"; -- Testa o sel quando em 10
        wait for (2*T);
        tb_sel <= "11"; -- Testa o sel quando em 11

        -- Segundo teste
        wait for (5*T);
        tb_sel <= "00"; -- Testa o sel quando em 00
          tb_bus_array <= ( -- Atribui os mesmos valores anteriores mas
agora invertido
            0  => "1111",
            1  => "1110",
            2  => "1101",
            3  => "1100",
            4  => "1011",
            5  => "1010",
```

```
            6  => "1001",
            7  => "1000",
            8  => "0111",
            9  => "0110",
            10 => "0101",
            11 => "0100",
            12 => "0011",
            13 => "0010",
            14 => "0001",
            15 => "0000"
        );
        wait for (2*T);
        tb_sel <= "01"; -- Testa o sel quando em 01
        wait for (2*T);
        tb_sel <= "10"; -- Testa o sel quando em 10
        wait for (2*T);
        tb_sel <= "11"; -- Testa o sel quando em 11
        wait for (2*T);

            -- Repete o primeiro teste novamente, mas com o enable
desabilitado (todas as saidas devem ficar congeladas)
        tb_en <= '0';

        -- Zera tudo depois de 2 periodos de clock
        wait for (2*T);
        tb_sel <= "00";
        tb_bus_array <= (others => (others => '0'));

            -- Atribui a cada posicao do vetor um valor de 4 bits
"correspondente" a posicao para facilitar a visualizacao na forma de
onda
        wait for (2*T);
        tb_bus_array <= (
                0  => "0000",
                1  => "0001",
                2  => "0010",
                3  => "0011",
                4  => "0100",
                5  => "0101",
                6  => "0110",
                7  => "0111",
                8  => "1000",
                9  => "1001",
```

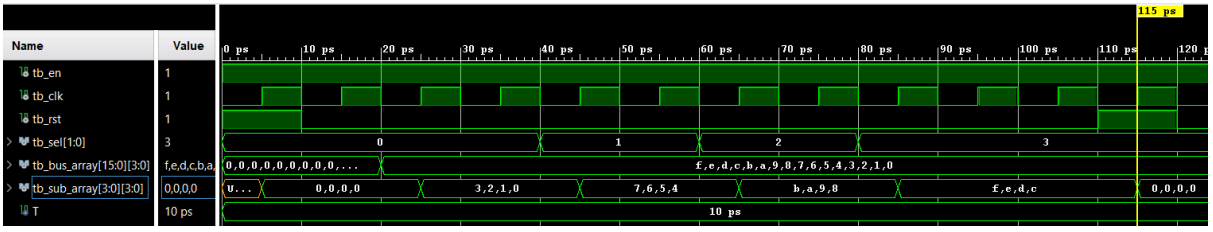
```
                10 => "1010",
                11 => "1011",
                12 => "1100",
                13 => "1101",
                14 => "1110",
                15 => "1111"
            );
        wait for (2*T);
        tb_sel <= "01"; -- Testa o sel quando em 01
        wait for (2*T);
        tb_sel <= "10"; -- Testa o sel quando em 10
        wait for (2*T);
        tb_sel <= "11"; -- Testa o sel quando em 11

        wait;
    end process stimulus;

end architecture testbench;
```

Para testar o circuito, criei um testbench com três processos, um processo gerou um **clock** contínuo com período de **10 ps**, enquanto outro controlou o **reset síncrono**, ativando-o no **início da simulação** e novamente **durante a execuçã**o para testar sua funcionalidade. O terceiro processo, responsável pelos estímulos, executou três cenários de teste principais: primeiramente, com o circuito habilitado (**en = '1'**), o barramento de entrada foi carregado com conjuntos de 4 bits sequenciais, para facilitar a visualização no waveform, e o seletor sel percorreu todas as combinações para validar a lógica de multiplexação; em seguida, um segundo teste foi realizado com um padrão de dados invertido para verificar o funcionamento.

Por fim, o primeiro cenário foi repetido, mas com o sinal de **enable desativado**, para confirmar que o registrador de saída mantinha seu valor anterior e não era influenciado pelas novas entradas.

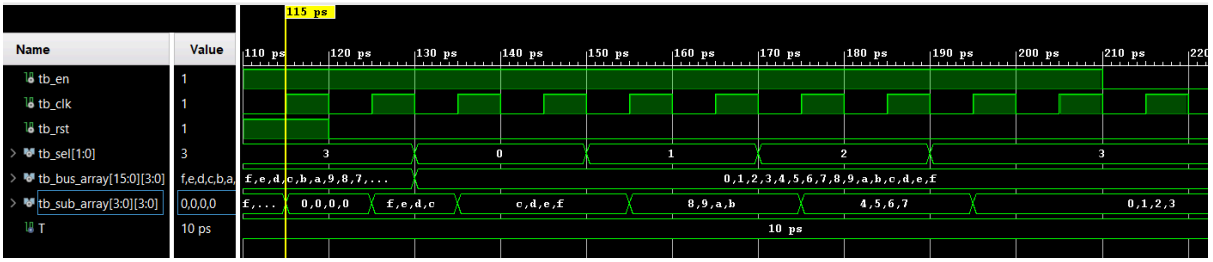
Prints do Testbench

Início da Simulação:



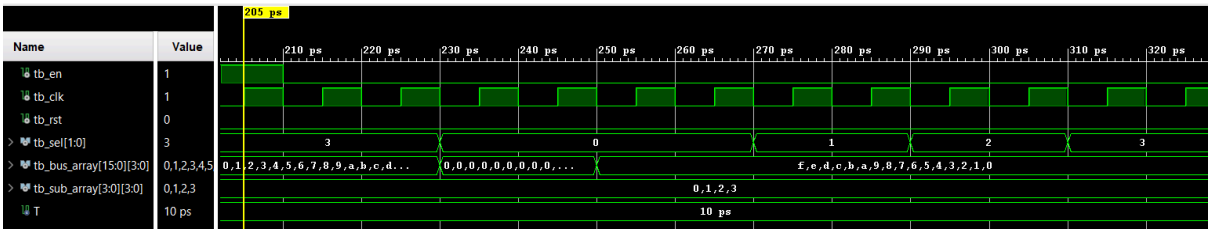
Fonte: Elaboração própria.

Segundo Teste:



Fonte: Elaboração própria.

Enable Desabilitado:



Fonte: Elaboração própria.

Quarta Questão

Código Transcrito do Tipo Utilizado

```
library IEEE;
use IEEE.numeric_std.all;

package custom_types is

    type integer_array is array (natural range <>) of integer;

end package custom_types;
```

Código Transcrito do Circuito

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library work;
use  work.custom_types.all;  --  Biblioteca  para  o  tipo  integer_array
"type integer_array is array (natural range <>) of integer;"

entity circuit01 is
    port (
        clk : in std_logic;
        rst : in std_logic;
```

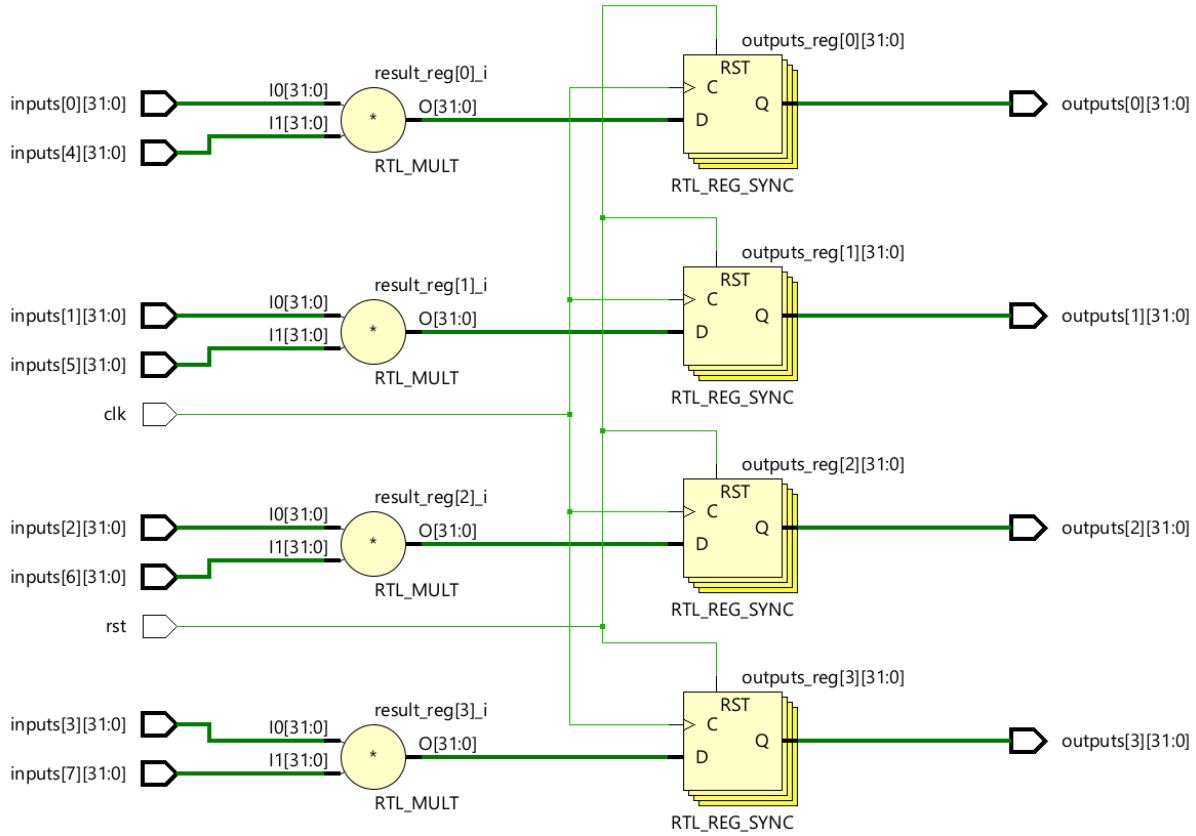
```
        inputs : in integer_array (7 downto 0);
        outputs : out integer_array (3 downto 0)
    );
end entity circuit01;

architecture version_A of circuit01 is -- Versao paralela

      signal result_reg : integer_array (3 downto 0); -- Array de
inteiros para o "fio" que vai para o registrador

begin

    -- Realiza as multiplicacoes
    result_reg(0) <= inputs(0) * inputs(4);
    result_reg(1) <= inputs(1) * inputs(5);
    result_reg(2) <= inputs(2) * inputs(6);
    result_reg(3) <= inputs(3) * inputs(7);

    -- Registrador para atualizar a saida com base no clock
    sync_reg: process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then -- Definicao do reset
                outputs(0) <= 0;
                outputs(1) <= 0;
                outputs(2) <= 0;
                outputs(3) <= 0;
            else -- Joga os valores nas saidas
                outputs(0) <= result_reg(0);
                outputs(1) <= result_reg(1);
                outputs(2) <= result_reg(2);
                outputs(3) <= result_reg(3);
            end if;
        end if;
    end process sync_reg;

end architecture version_A;

architecture version_B of circuit01 is -- Versao sequencial (Fiz de
forma a ter apenas um multiplicador fisico)

      type states is (first,second,third,fourth); -- Definicao dos
estados
```

```
      signal current_state : states; -- estado atual da maquina de
estados
    signal first_value, second_value, mult_result : integer; -- sinais
intermediarios para fazer a multiplicacao em amenas um multiplicador
fisico
      signal result_reg : integer_array (3 downto 0); -- Array de
inteiros como registradores de saida

begin

      -- Processo combinacional para selecionar quais valores serao
multiplicados de acordo com o estado atual
    sel_mult: process(current_state, inputs)
    begin
        case (current_state) is
                  when first => -- Seleciona os valores da primeira
multiplicacao (0 e 4)
                first_value <= inputs(0);
                second_value <= inputs(4);
                  when second => -- Seleciona os valores da segunda
multiplicacao (1 e 5)
                first_value <= inputs(1);
                second_value <= inputs(5);
                  when third => -- Seleciona os valores da terceira
multiplicacao (2 e 6)
                first_value <= inputs(2);
                second_value <= inputs(6);
                   when others => -- Seleciona os valores da quarta
multiplicacao (3 e 7)
                first_value <= inputs(3);
                second_value <= inputs(7);
        end case;
    end process sel_mult;

     -- Multiplica os valores escolhidos (Dessa forma, tera apenas um
multiplicador fisico)
    mult_result <= first_value * second_value;

     -- Processo sincrono que registra os resultados das multiplicacoes
e atualiza o estado atual
    sync_proc: process(clk)
    begin
        if rising_edge(clk) then
```

```
            if rst = '1' then -- Reset sincrono
                    current_state <= first; -- Reseta o estado para o
inicial
                       result_reg <= (others => 0); -- Atribui 0 aos
registradores de saida
            else
                case (current_state) is
                      when first => -- Registra o primeiro resultado e
passa para o proximo estado
                        result_reg(0) <= mult_result;
                        current_state <= second;
                      when second => -- Registra o segundo resultado e
passa para o proximo estado
                        result_reg(1) <= mult_result;
                        current_state <= third;
                      when third => -- Registra o terceiro resultado e
passa para o proximo estado
                        result_reg(2) <= mult_result;
                        current_state <= fourth;
                       when others => -- Registra o quarto resultado e
passa para o proximo estado
                        result_reg(3) <= mult_result;
                        current_state <= first;
                end case;
            end if;
        end if;
    end process sync_proc;

    -- Atribui o resultado das multiplicacoes na saida
    outputs(0) <= result_reg(0);
    outputs(1) <= result_reg(1);
    outputs(2) <= result_reg(2);
    outputs(3) <= result_reg(3);

end architecture version_B;
```

RTLs Elaborados

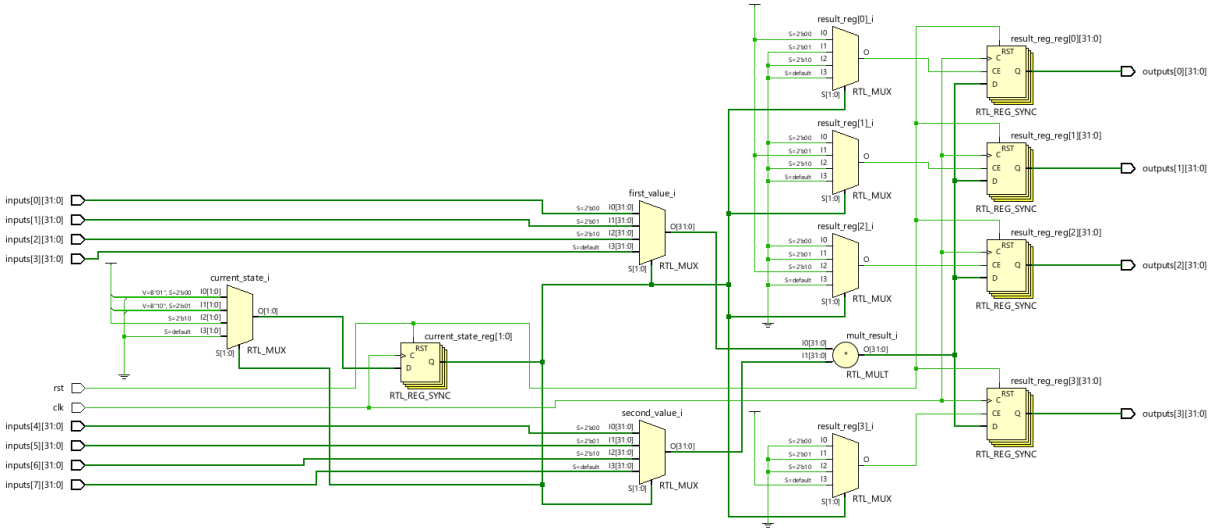
RTL da versão A (Paralela):



Fonte: Elaboração própria.

RTL da versão B (Sequencial):

Fonte: Elaboração própria.



Para a implementação da **versão paralela**, a arquitetura **version_A** realiza as **quatro multiplicações** de forma **simultânea**. Nesta versão, a lógica combinacional executa os cálculos inputs(0)*inputs(4), inputs(1)*inputs(5), inputs(2)*inputs(6) e inputs(3)*inputs(7) concorrentemente e um processo síncrono (sync_reg) atua como um banco de registradores de saída. Na borda de subida do clock, se o sinal de **reset** (rst) estiver **ativo**, todas as **saídas**

são **zerada**s. Caso contrário, os resultados das multiplicações são transferidos para as portas de output.

Já para a **versão sequencial**, a arquitetura **version_B** implementa uma máquina de estados com quatro estados (first, second, third, fourth), onde cada estado corresponde a uma das multiplicações a serem realizadas. A lógica foi dividida em dois processos principais.

Primeiro, um processo combinacional (sel_mult) funciona como um multiplexador, selecionando o par correto para fazer a multiplicação do barramento de entrada (inputs) com base no estado atual da máquina. Esses operandos são multiplicados de forma a **exigir apenas um circuito de multiplicação**. Em seguida, um processo síncrono (sync_proc) gerencia as transições de estado e, a cada borda de subida do clock, armazena o resultado da multiplicação no registrador de saída correspondente, avançando para o próximo estado.

Código Transcrito do Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library work;
use  work.custom_types.all;  -- Biblioteca para o tipo integer_array
"type integer_array is array (natural range <>) of integer;"

entity tb_circuit01 is
end entity tb_circuit01;

architecture testbench of tb_circuit01 is

    -- Definicao do periodo do clock
    constant T : time := 10 ps;

    -- Sinais do tb para o circuito feito
    signal tb_clk, tb_rst : std_logic;
       signal  tb_inputs  :  integer_array  (7  downto  0);  --  inputs  do
testbench ligada as estradas de ambas as versoes
     signal  tb_A_outputs  :  integer_array  (3  downto  0);  --  outputs  da
versao A (paralela)
     signal  tb_B_outputs  :  integer_array  (3  downto  0);  --  outputs  da
versao B (sequencial)

    -- Declaracao do circuito feito
    component circuit01
        port (
            clk : in std_logic;
            rst : in std_logic;
```

```
            inputs : in integer_array (7 downto 0);
            outputs : out integer_array (3 downto 0)
        );
    end component circuit01;

begin

    -- Instanciacao da versao A do circuito feito
    uut_A : entity work.circuit01(version_A)
        port map(
            clk => tb_clk,
            rst => tb_rst,
            inputs => tb_inputs,
            outputs => tb_A_outputs
        );

    -- Instanciacao da versao B do circuito feito
    uut_B : entity work.circuit01(version_B)
        port map(
            clk => tb_clk,
            rst => tb_rst,
            inputs => tb_inputs,
            outputs => tb_B_outputs
        );

    -- Gera o clock do testbench
    clock_gen: process
    begin
        tb_clk <= '0';
        wait for (T/2);
        tb_clk <= '1';
        wait for (T/2);
    end process clock_gen;

    -- Acionamento do reset
    reset_gen: process
    begin
        -- Aciona o reset logo no comeco
        tb_rst <= '1';
        wait for (T);
        tb_rst <= '0';
        -- Aciona o reset em 200 ps
        wait for (T*19);
```

```
        tb_rst <= '1';
        wait for (T);
        tb_rst <= '0';
        wait;
    end process reset_gen;

    -- Estimulos fornecidos para realizar o testbench
    stimulus: process
    begin
        -- Inicia com tudo zerado
        tb_inputs(0) <= 0;
        tb_inputs(1) <= 0;
        tb_inputs(2) <= 0;
        tb_inputs(3) <= 0;
        tb_inputs(4) <= 0;
        tb_inputs(5) <= 0;
        tb_inputs(6) <= 0;
        tb_inputs(7) <= 0;

        -- Primeiro Teste
        wait for (T);
        tb_inputs(0) <= 1;
        tb_inputs(1) <= 2;
        tb_inputs(2) <= 3;
        tb_inputs(3) <= 4;
        tb_inputs(4) <= 5;
        tb_inputs(5) <= 6;
        tb_inputs(6) <= 7;
        tb_inputs(7) <= 8;

        -- Segundo Teste
        wait for (4*T);
        tb_inputs(0) <= 8;
        tb_inputs(1) <= 7;
        tb_inputs(2) <= 6;
        tb_inputs(3) <= 5;
        tb_inputs(4) <= 4;
        tb_inputs(5) <= 3;
        tb_inputs(6) <= 2;
        tb_inputs(7) <= 1;

        -- Terceiro Teste
        wait for (8*T);
```

```
        tb_inputs(0) <= 1;
        tb_inputs(1) <= 53;
        tb_inputs(2) <= 3;
        tb_inputs(3) <= 157;
        tb_inputs(4) <= 359;
        tb_inputs(5) <= 5;
        tb_inputs(6) <= 53;
        tb_inputs(7) <= 2;

        wait;
    end process stimulus;

end architecture testbench;
```

Para validar as duas versões do circuito, criei um testbench que **instancia ambas** as arquiteturas: a paralela (version_A) e a sequencial (version_B). As duas foram conectadas aos mesmos sinais de clock, reset e entrada, permitindo uma comparação mais direta de seus resultados.

O teste foi organizado em três processos principais. Um processo gerou um **clock** contínuo com período de **10 ps**, enquanto outro controlou o sinal de **reset**, aplicando um pulso no **início** e **outro durante** a simulação para verificar a resposta dos circuitos. O terceiro processo foi responsável por aplicar os estímulos em três cenários de teste distintos:

1. Primeiro, após um reset inicial, valores sequenciais de **1, 2, 3…** foram colocados nas entradas.
2. Em seguida, após aguardar tempo suficiente para a conclusão do circuito sequencial, um segundo conjunto em ordem invertida (**8, 7, 6…**) foi enviado para testar.
3. Por fim, um terceiro cenário com valores um pouco maiores foi aplicado para validar o comportamento dos multiplicadores. O resultado esperado são os **12 primeiros dígitos de pi** da **saída 3 para a saída 1**.

Resultados esperados:

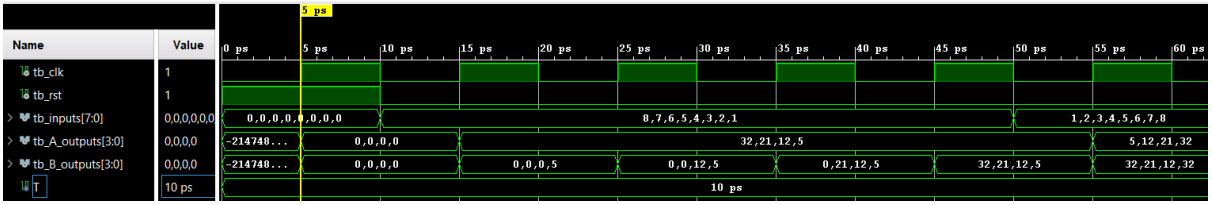
1. Primeiro Teste:
    - outputs(0) = inputs(0) * inputs(4) = 1 × 5 = 5
    - outputs(1) = inputs(1) * inputs(5) = 2 × 6 = 12
    - outputs(2) = inputs(2) * inputs(6) = 3 × 7 = 21
    - outputs(3) = inputs(3) * inputs(7) = 4 × 8 = 32

2. Segundo Teste:
    - outputs(0) = inputs(0) * inputs(4) = 8 × 4 = 32
    - outputs(1) = inputs(1) * inputs(5) = 7 × 3 = 21
    - outputs(2) = inputs(2) * inputs(6) = 6 × 2 = 12
    - outputs(3) = inputs(3) * inputs(7) = 5 × 1 = 5

3. Terceiro Teste:
    - outputs(0) = inputs(0) * inputs(4) = 1 × 359 = 359
    - outputs(1) = inputs(1) * inputs(5) = 53 × 5 = 265
    - outputs(2) = inputs(2) * inputs(6) = 3 × 53 = 159
    - outputs(3) = inputs(3) * inputs(7) = 157 × 2 = 314

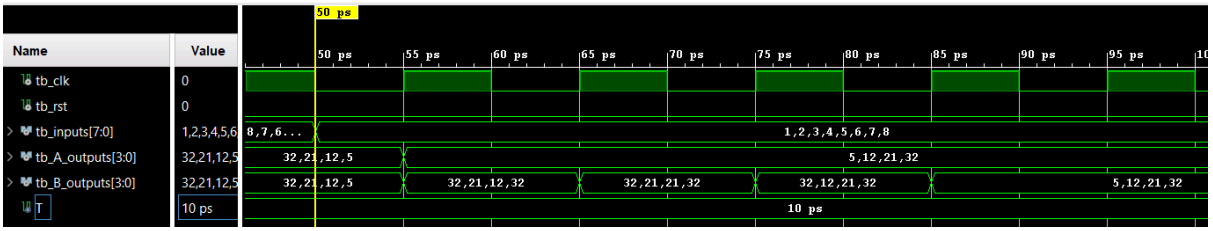
Prints do Testbench

Início da Simulação (Primeiro Teste):



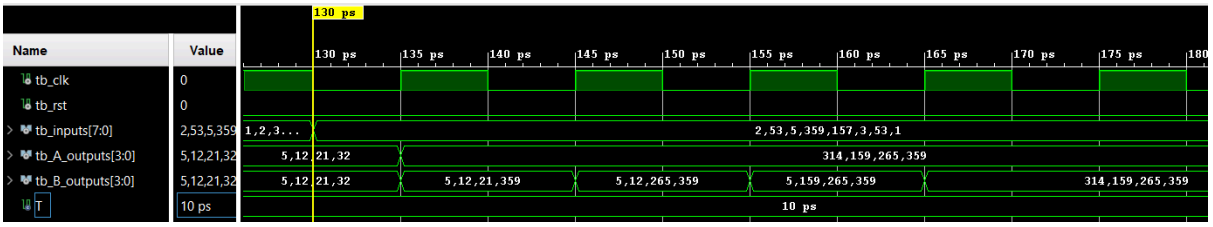
Fonte: Elaboração própria.

Segundo Teste:



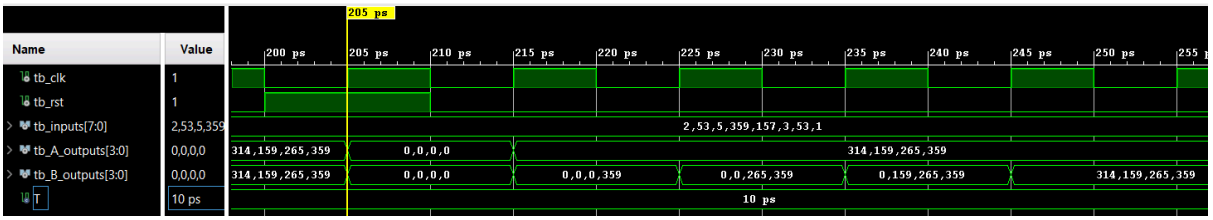
Fonte: Elaboração própria.

Terceiro Teste:



Fonte: Elaboração própria.

Teste do Reset:



Fonte: Elaboração própria.