Universidade do Vale do Itajaí - UNIVALI
Engenharia de Computação - 2º Período
Disciplina: Projetos de Sistemas Digitais
Alunos: Taryck, Pedro Kons, Cauã Domingos

Avaliação 2 – Projeto de Circuitos Combinacionais - TMR

Circuito 1: Demultiplexador ligado a um multiplexador

Enunciado: Implemente um circuito de compartilhamento de canal composto de um multiplexador com duas entradas atribuídas a chaves (SWs) ligado a um demultiplexador com saída em dois leds (LEDs), ambos controlados pela mesma chave de seleção (SEL), conforme ilustrado pelo diagrama que segue.

Circuito 1 - Testbench:

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_compart_canal is
end tb_compart_canal;

architecture arch_1 of tb_compart_canal is

component compart_canal is
port ( i_SEL : in std_logic; -- Seletor de entradas a ser
usada no mux
       i_SW0 : in std_logic; -- Entrada 1
       i_SW1 : in std_logic; -- Entrada 2
       o_LED0 : out std_logic; -- Saída led 1
       o_LED1 : out std_logic); -- Saída led 2
end component;
-- Fios internos que serão usados no mux
signal w_SEL, w_SW0, w_SW1, w_LED0, w_LED1 : std_logic;
begin

        -- Conexão dos fios
        u_DUT: compart_canal port map(i_SEL => w_SEL,
                                      i_SW0 => w_SW0,
                                      i_SW1 => w_SW1,
```

```
                                        o_LED0 => w_LED0,
                                        o_LED1 => w_LED1);
        process
        begin

-- Casos para que a entrada selecionada seja a entrada A
SELECTOR = 0
        w_SEL <= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;

        w_SEL <= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;

        -- Casos para que a entrada selecionada seja a
entrada B SELECTOR = 1
        w_SEL <= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;
```

```
        w_SEL <= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        w_SEL <= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        -- Para limpar as entradas
        w_SEL <= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        assert false report "Test done." severity note;
        wait;
    end process;
end arch_1;
```

Circuito 1 - Design

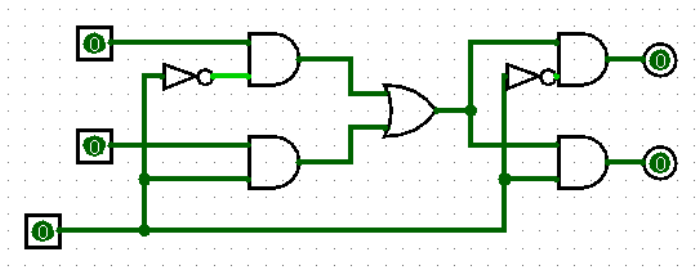
```
library ieee;
use ieee.std_logic_1164.all;

entity compart_canal is
port ( i_SEL : in std_logic; -- Seletor de entradas
       i_SW0 : in std_logic; -- Entrada 1
        i_SW1 : in std_logic; -- Entrada 2
       o_LED0 : out std_logic; -- Saída led 1
       o_LED1 : out std_logic); -- Saída led 2
end compart_canal ;
```

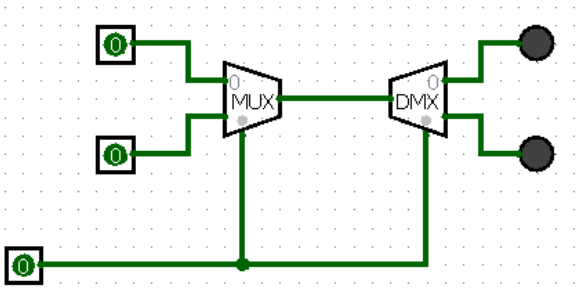
```
architecture arch_1 of compart_canal is
begin
                process(i_SEL, i_SW0, i_SW1)
          begin
                        o_LED0 <= (((i_SW0 and not i_SEL)or(i_SW1
and i_SEL)) and not i_SEL);
                        o_LED1 <= (((i_SW0 and not i_SEL)or(i_SW1
and i_SEL)) and i_SEL);
          end process;
end arch_1;
```

Circuito 1: Tabela verdade

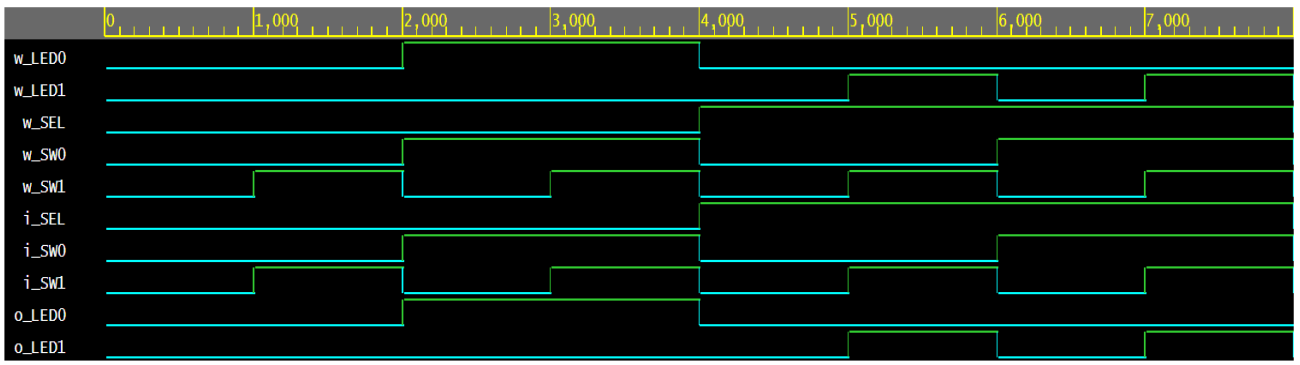
i_SEL	i_SW0	i_SW1	o_LED0	o_LED1
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	0	1

Projetos logisim: Mux-demux

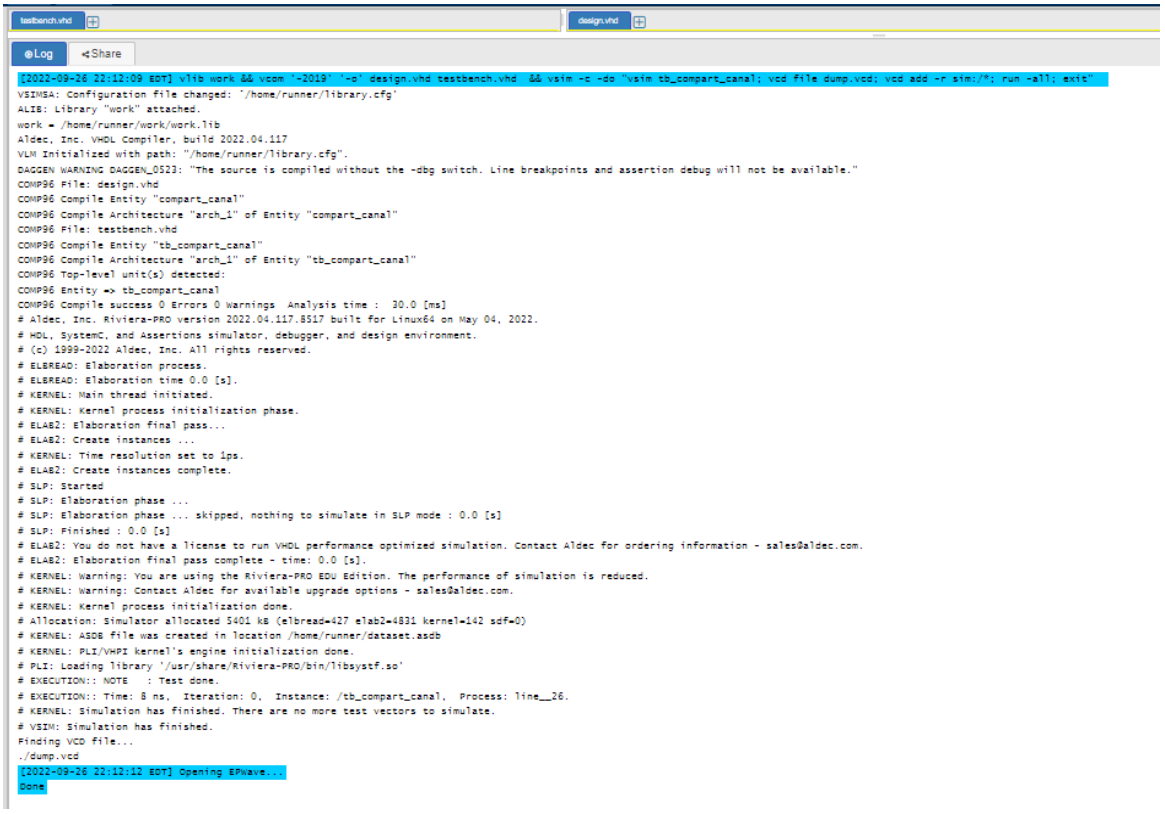




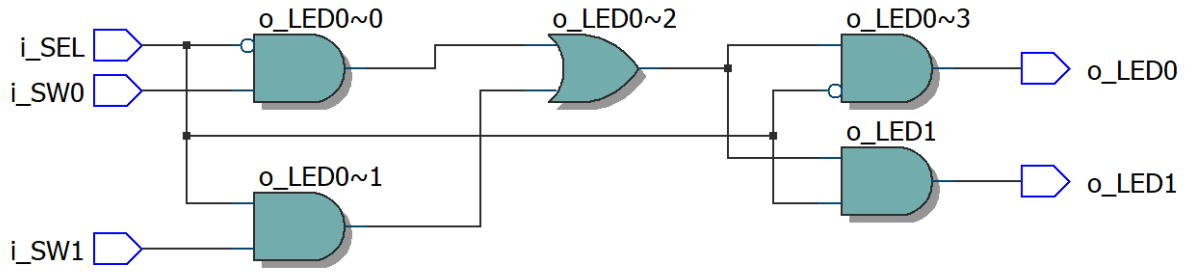
Resultado: diagrama de ondas



Console resultante: Mux-demux



Resultado: QUARTUS



Discussão dos resultados:

A função e funcionamento de um multiplexador é o conjunto de ações de selecionar apenas **uma** saída para **"N"** entradas, que no caso deste exercício veio a ser apenas duas (mux2_1bit), além de uma chave seletora que define qual das entradas vai realizar a operação. Já função e funcionamento de um demultiplexador é o ato de distribuir informações de **uma única entrada** para um número **N** de saídas, ou seja, a função contrária do multiplexador apresentado.

Ambos os plexers, estão ligados entre si, utilizando o canal de saída do multiplexador como a entrada do demultiplexador. As entradas são nomeadas de **i_SW0** e **i_SW1**, enquanto as saídas, que simbolizam leds, são nomeadas de

o_LED0 e **o_LED1**. Todos os plexers e todas as portas lógicas utilizam do único e mesmo bit usado como seletor: o **i_SEL**.

Após os devidos testes de mesa realizados, foi observado que as saídas de led correspondem respectivamente a entrada que tem o seu mesmo número estabelecido (0 ou 1), sendo ativadas de acordo com o estado encontrado na chave seletora **i_SEL**. Caso a chave seletora esteja em estado baixo, as mudanças ocorreram somente no **o_LED0,** que passará a receber a mesma informação da entrada **i_SW0**, ignorando totalmente a entrada **o_LED1** e seus estados**.** Caso a chave seletora **i_SEL** esteja em nível alto, o led ativado será apenas o **o_LED1**, que passará a receber a mesma informação da entrada **i_SW1,** ignorando totalmente os bits estabelecidos em **i_SW0.**

Circuito 2: Multiplexador ligado a um demultiplexador com 1 bit de *push button*

Enunciado: A redução do comprimento do canal dos transistores e o aumento da frequência de operação fazem com que os componentes de um sistema integrado estejam cada vez mais suscetíveis a fontes de ruído internas (fonte de alimentação, diafonia, etc.) e externas (interferência eletromagnética, partículas alfa, etc.). Essas fontes podem resultar em faltas, sendo que uma falta pode vir a produzir um erro. Um exemplo de erro que pode ocorrer é a inversão de um bit de um dado em uma linha de comunicação. Modifique o circuito do Projeto 1 incluindo uma porta XOR para emular a injeção de erros de mudança de bit, a qual deve ser acionada por meio de um push-button (BT0), conforme ilustrado no diagrama de blocos a seguir

Circuito 2 - Testbench:

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_compart_canal is
end tb_compart_canal;

architecture arch_1 of tb_compart_canal is

component compart_canal is
port ( i_SEL : in std_logic; -- Seletor de entradas a ser
usada no mux
       i_SW0 : in std_logic; -- Entrada 1
       i_SW1 : in std_logic; -- Entrada 2
       i_BT0 : in std_logic; -- Entrada de ruído
```

```
       o_LED0 : out std_logic; -- Saída led 1
       o_LED1 : out std_logic); -- Saída led 2
end component;
-- Fios internos que serão usados no mux
signal w_SEL, w_SW0, w_SW1, w_BT0, w_LED0, w_LED1 :
std_logic;
begin

        -- Conexão dos fios
        u_DUT: compart_canal port map(i_SEL => w_SEL,
                                      i_SW0 => w_SW0,
                                      i_SW1 => w_SW1,
                                      i_BT0 => w_BT0,
                                      o_LED0 => w_LED0,
                                      o_LED1 => w_LED1);
        process
        begin

-- Casos para que a entrada selecionada seja a entrada A
SELECTOR = 0
        w_SEL <= '0';
        w_BT0 <= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '0';
        w_BT0 <= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '0';
        w_BT0 <= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;
```

```
        w_SEL <= '0';
        w_BT0 <= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;

        w_SEL <= '0';
        w_BT0 <= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;

        w_SEL <= '0';
        w_BT0 <= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
10" severity error;

        w_SEL <= '0';
        w_BT0 <= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '0';
        w_BT0 <= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        -- Casos para que a entrada selecionada seja a
entrada B SELECTOR = 1
        w_SEL <= '1';
```

```
        w_BT0 <= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '1';
        w_BT0 <= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        w_SEL <= '1';
        w_BT0 <= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '1';
        w_BT0 <= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        w_SEL <= '1';
        w_BT0 <= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        w_SEL <= '1';
        w_BT0 <= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
```

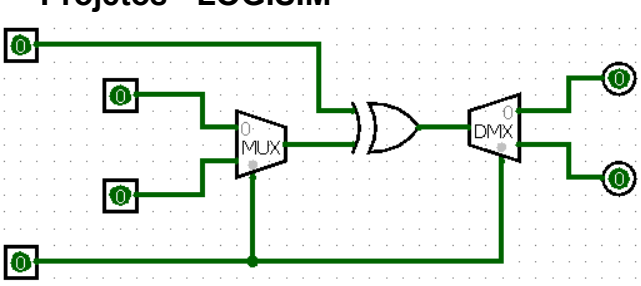
```
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;

        w_SEL <= '1';
        w_BT0 <= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
01" severity error;

        w_SEL <= '1';
        w_BT0 <= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
00" severity error;
        -- Para limpar as entradas
        w_SEL <= '0';
        w_BT0 <= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        assert false report "Test done." severity note;
        wait;
    end process;
end arch_1;
```

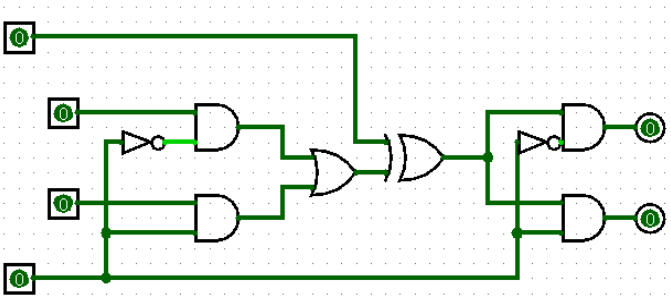
Circuito 2 - Design

```
library ieee;
use ieee.std_logic_1164.all;

entity compart_canal is
port ( i_SEL : in std_logic; -- Seletor de entradas
       i_SW0 : in std_logic; -- Entrada 1
       i_SW1 : in std_logic; -- Entrada 2
       i_BT0 : in std_logic; -- Entrada de ruído
       o_LED0 : out std_logic; -- Saída led 1
       o_LED1 : out std_logic); -- Saída led 2
end compart_canal ;
```

```
architecture arch_1 of compart_canal is
begin
                process(i_SEL, i_SW0, i_SW1,i_BT0)
          begin
                        o_LED0 <= (((((i_SW0 and not
i_SEL)or(i_SW1 and i_SEL))) xor i_BT0) and not i_SEL);
                        o_LED1 <= (((((i_SW0 and not
i_SEL)or(i_SW1 and i_SEL))) xor i_BT0) and i_SEL);
          end process;
end arch_1;
```

Projetos - LOGISIM

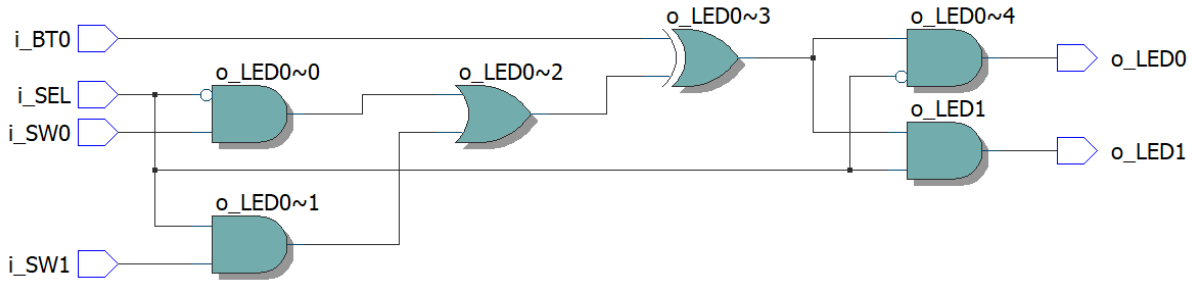




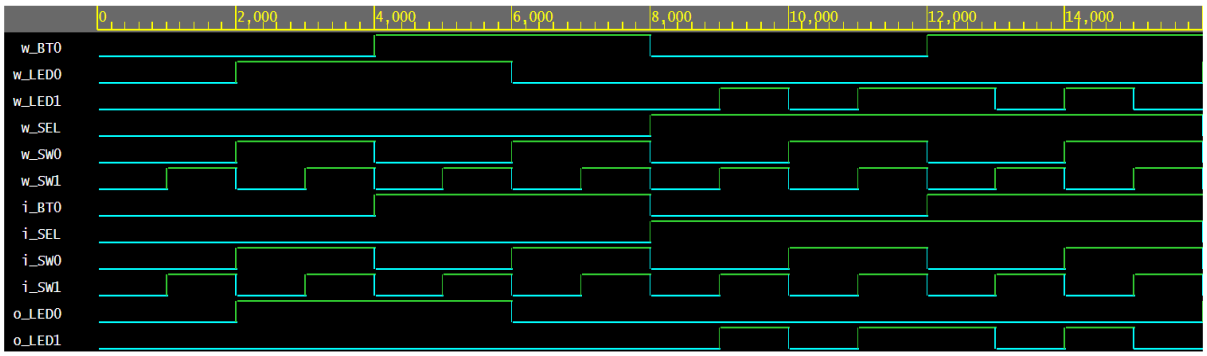
Circuito 2: Tabela verdade

i_SEL	i_BT0	i_SW0	i_SW1	o_LED0	o_LED1
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	0

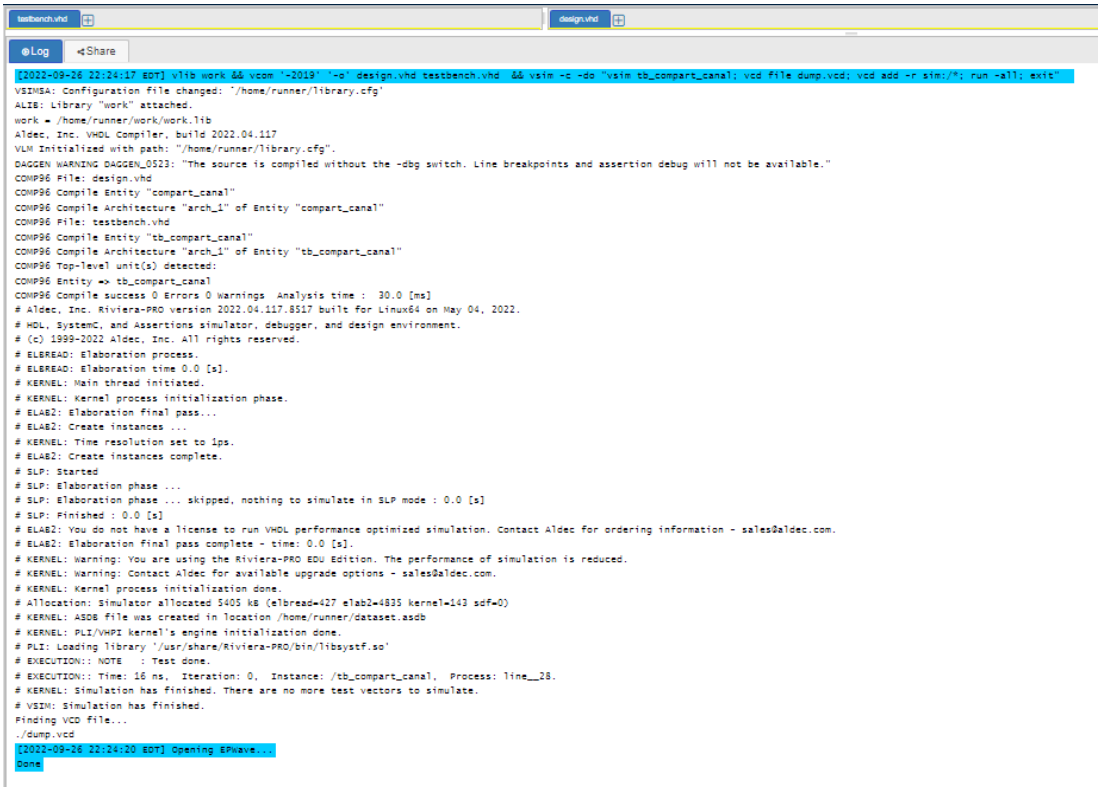
Resultado: QUARTUS



Resultado: diagrama de ondas



Console resultante - Demux



Discussão dos resultados:

O design do segundo circuito faz juz ao primeiro. Este, contém também as mesmas entradas apresentadas no circuito 1, seus mesmos plexers e conexões. Há duas entradas: **i_SW0** e **i_SW1**, duas saídas: **o_LED0** e **o_LED1**, e também uma chave seletora: **i_SEL**. O diferencial deste circuito, é a adição de 1 bit que tem como objetivo simular um erro que pode ser causado durante a implementação de um circuito: o **i_BT0**.

A implementação deste circuito é semelhante a do primeiro, mudando apenas o fato de que entre a conexão direta entre o multiplexador e o demultiplexador,

haverá uma entrada **XOR** que liga a saída do mux com a entrada de ruído **i_BT0**, sujeitando o circuito a algum erro que pode vir a acontecer. A porta **XOR** tem como característica: passar informação em estado alto apenas quando suas duas entradas forem diferentes e caso sejam iguais, passará informação em estado baixo.

Como todos os outros passos anteriores, o circuito 2 também é parelho ao primeiro circuito, onde as saídas **o_LED0** e **o_LED1** recebem as informações das entradas **i_SW0** e **i_SW1** respectivamente, dependendo da seleção da entrada **i_SEL**: caso esteja em 0, a saída usada será a **o_LED0** e ela recebe os valores de **i_SW0** e, caso esteja em 1, **o_LED1** receberá as informações de **i_SW1**. Mas este funcionamento só acontecerá desta maneira se o bit de ruído **i_BT0** estiver em 0.

Caso haja ruído por algum acontecimento, a entrada **i_BT0** se transformará em 1 automaticamente e mudará todo o processo do circuito. As saídas resultantes quando o bit de ruído está ativado funcionam da seguinte maneira: caso a entrada **i_SW0** estiver desligada e a chave seletora **i_SEL** estiver em 0, a saída **o_LED0** ganhará as mesmas informações de **i_BT0**, não dependendo da estrada **i_SW1** e a saída **o_LED1** não será acionada de forma alguma. Caso a entrada **i_SW0** seja ativada durante o alto nível de **i_BT0** e **i_SEL** estiver em 0, todas as saídas receberam 0.

Caso a chave seletora **i_SEL** estiver em 1, a entrada **i_SW0** e a saída **o_LED0** serão totalmente desprezados. Sendo assim, se **i_SW1** estiver em 0 e **i_SEL** em 1, a saída **o_LED1** receberá todo o valor que **i_BT0** estabelecer. Caso os dois estabeleçam 1 simultaneamente, todas as saídas receberam valores de nível baixo.

Circuito 3: Multiplexador ligado a 3 bits *push button* com um votador que tem a sua saída ligada a um demultiplexador. (Triple Modular Redundancy)

Enunciado: Uma técnica que busca prover confiabilidade em sistemas suscetíveis a ruídos é a tolerância modular tripla ou TMR (Triple Modular Redundancy). Essa técnica consiste na replicação de recursos e uso de um votador com três entradas que seleciona o valor produzido pela maioria das entradas e o apresenta na saída. Esse circuito permite detectar e corrigir a inversão de um bit. Modifique o circuito do Projeto 2 replicando a linha de comunicação com a técnica de TMR, incluindo a injeção de faltas em cada uma das três linhas por meio de 3 push-buttons (BTs) e um circuito votador que deve ser desenvolvido seguindo a metodologia de projetos de circuitos combinacionais apresentadas em sala de aula.

Circuito 3 - Testbench:

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

entity tb_mux_demux_TMR is
end tb_mux_demux_TMR;

architecture arch_1 of tb_mux_demux_TMR is
component mux_demux_TMR is
port (i_SEL  :  in std_logic; -- Seletor de entradas
      i_SW0  :  in std_logic; -- Entrada 1
      i_SW1  :  in std_logic; -- Entrada 1
      i_BUT1 :  in std_logic; -- push-button 1
      i_BUT2 :  in std_logic; -- push-button 2
      i_BUT3 :  in std_logic; -- push-button 3
      o_LED0 :  out std_logic; -- Saída led 1
      o_LED1 :  out std_logic); -- Saída led 2
end component;

signal w_SEL, w_SW0, w_SW1,w_BUT1,w_BUT2,w_BUT3, w_LED0,
w_LED1: std_logic;
begin

        u_DUT : mux_demux_TMR port map(i_SEL  => w_SEL,
                                       i_SW0   => w_SW0,
                                       i_SW1   => w_SW1,
                                       i_BUT1  => w_BUT1,
                                       i_BUT2  => w_BUT2,
                                       i_BUT3  => w_BUT3,
                                       o_LED0  => w_LED0,
                                       o_LED1  => w_LED1);

process
begin

-- Todos os casos quando o select está em 0
        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
```

```
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
000000" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
000001" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
000010" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
000011" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
000100" severity error;
```

```
        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
000101" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
000110" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
000111" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
001000" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
```

```
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
001001" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
001010" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
001011" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
001100" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
```

```
        wait for 1ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
001101" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
001110" severity error;

        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
001111" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
010000" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
010001" severity error;
```

```
        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
010010" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
010011" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
010100" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
010101" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
```

```
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
010110" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
010111" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
011000" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
011001" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
```

```
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
011010" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
011011" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
011100" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '1' and w_LED1 = '0') report "Fail @
011101" severity error;

        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
011110" severity error;
```

```
        w_SEL <= '0';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
011111" severity error;

-- Todos os casos quando o select está em 1
        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
100000" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
100001" severity error;

-- Todos os casos quando o select está em 0
        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
100010" severity error;
```

```
        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
100011" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
100100" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
100101" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
100110" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '0';
```

```
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
100111" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
101000" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
101001" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
101010" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
```

```
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
101011" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
101100" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
101101" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
101110" severity error;

        w_SEL <= '1';
        w_BUT1<= '0';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
101111" severity error;
```

```
        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
110000" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
110001" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
110010" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
110011" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
```

```
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
110100" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
110101" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
110110" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '0';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
110111" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
```

```
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
111000" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
111001" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1 ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
111010" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '0';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
111011" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
111100" severity error;
```

```
        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '0';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
111101" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '0';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '1') report "Fail @
111110" severity error;

        w_SEL <= '1';
        w_BUT1<= '1';
        w_BUT2<= '1';
        w_BUT3<= '1';
        w_SW0 <= '1';
        w_SW1 <= '1';
        wait for 1ns;
        assert(w_LED0 = '0' and w_LED1 = '0') report "Fail @
111111" severity error;

-- Para limpar as entradas
        w_SEL <= '0';
        w_BUT1<= '0';
        w_BUT2<= '0';
        w_BUT3<= '0';
        w_SW0 <= '0';
        w_SW1 <= '0';
        assert false report "Test done." severity note;
  wait;
  end process;
end arch_1;
```

Circuito 3 - Design

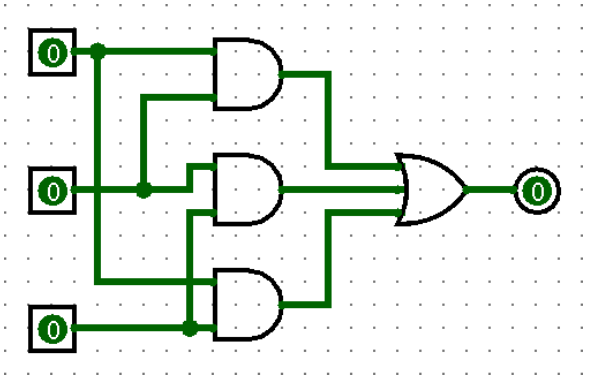
```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_demux_TMR is
port (i_SEL  :  in std_logic; -- Seletor de entradas
      i_SW0  :  in std_logic; -- Entrada 1
      i_SW1  :  in std_logic; -- Entrada 2
      i_BUT1 :  in std_logic; -- push-button 1
      i_BUT2 :  in std_logic; -- push-button 2
      i_BUT3 :  in std_logic; -- push-button 3
      o_LED0 :  out std_logic; -- Saída led 1
      o_LED1 :  out std_logic); -- Saída led 2
end mux_demux_TMR;

architecture arch_1 of mux_demux_TMR is
begin

        process(i_SEL,i_SW0,i_SW1,i_BUT0,i_BUT1,i_BUT2)
       begin

        o_LED0 <= (((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT0)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT1)) or ((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT0)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT2)) or ((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT1)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT2)))and not i_SEL);

        o_LED1 <= (((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT0)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT1)) or ((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT0)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT2)) or ((((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT1)and(((not i_SEL and i_SW0) or (i_SW1 and
i_SEL))xor i_BUT2)))and i_SEL);

    end process;
end arch_1;
```

Circuito 3: Tabela verdade

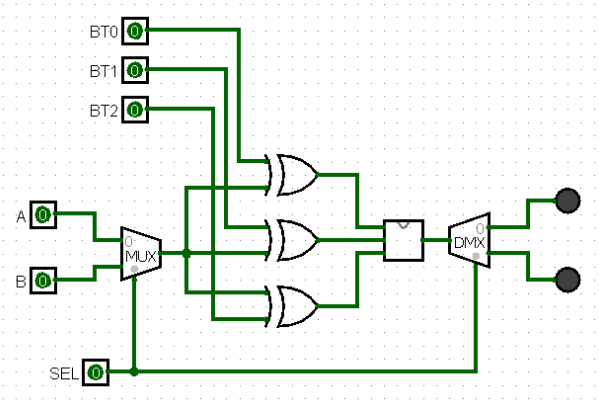
i_SEL	i_SW1	i_SW0	i_BT2	i_BT1	i_BT0	o_LED1	o_LED0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	1
0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	1
0	0	0	1	1	0	0	1
0	0	0	1	1	1	0	1
0	0	1	0	0	0	0	1
0	0	1	0	0	1	0	1
0	0	1	0	1	0	0	1
0	0	1	0	1	1	0	0
0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	0
0	1	0	1	0	1	0	1
0	1	0	1	1	0	0	1
0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	1
0	1	1	0	0	1	0	1
0	1	1	0	1	0	0	1
0	1	1	0	1	1	0	0
0	1	1	1	0	0	0	1
0	1	1	1	0	1	0	0
0	1	1	1	1	0	0	0
0	1	1	1	1	1	0	0

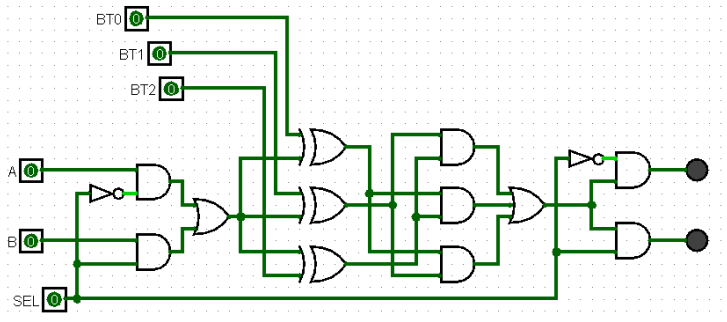
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	0	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	1	0	1	1	0
1	0	0	1	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	0	0	0	0
1	0	1	0	0	1	0	0
1	0	1	0	1	0	0	0
1	0	1	0	1	1	1	0
1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0
1	0	1	1	1	0	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	0
1	1	0	0	0	1	1	0
1	1	0	0	1	0	1	0
1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	0
1	1	0	1	0	1	0	0
1	1	0	1	1	0	0	0
1	1	0	1	1	1	0	0
1	1	1	0	0	0	1	0
1	1	1	0	0	1	1	0
1	1	1	0	1	0	1	0
1	1	1	0	1	1	0	0
1	1	1	1	0	0	1	0
1	1	1	1	0	1	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0

Projeto: Lógica “voter” no LOGISIM



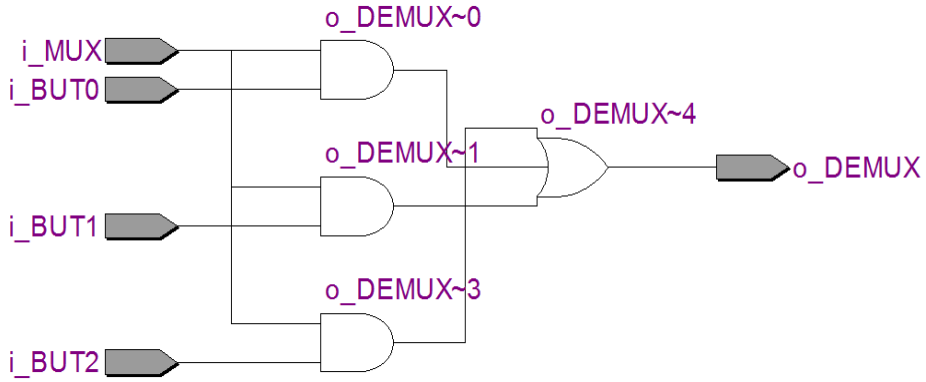
Projeto: Circuito no LOGISIM



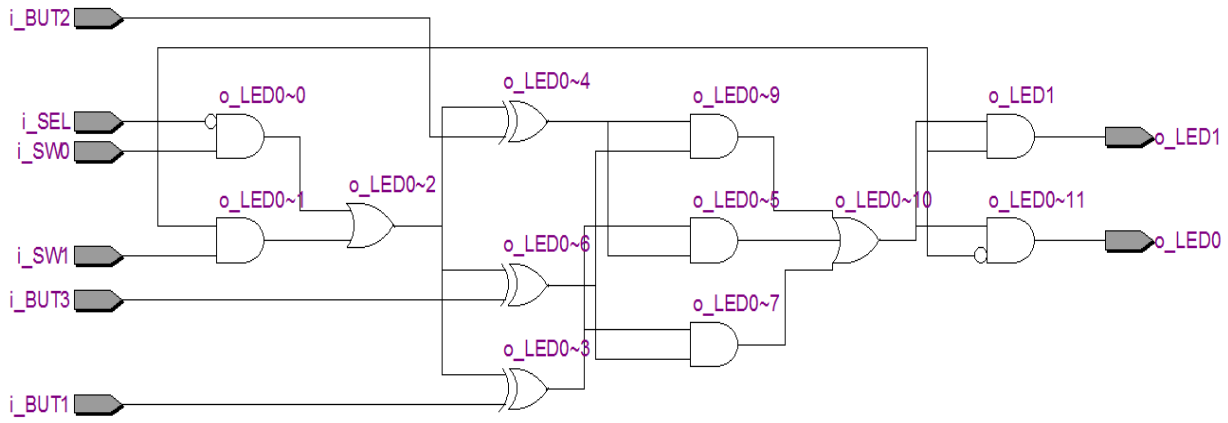


Resultado: QUARTUS

Votador:



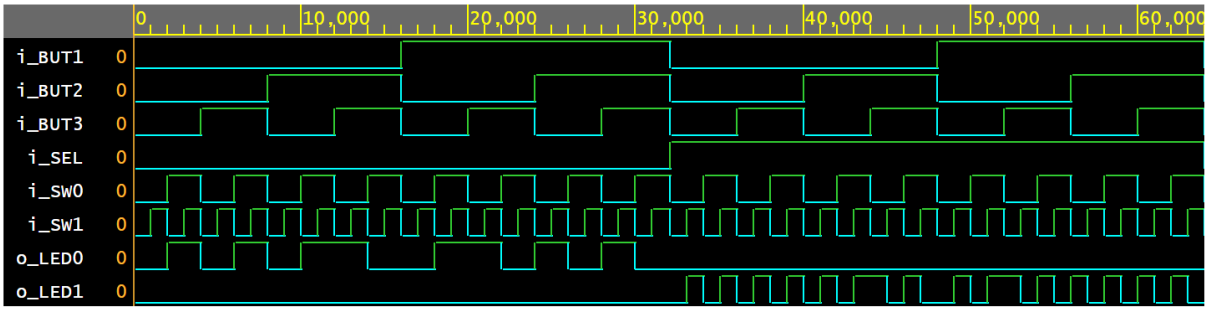
Circuito completo



Console resultante

```
[2022-09-27 21:55:49 EDT] vlib work && vcom '-2019' '-o' design.vhd testbench.vhd  && vsim -c -do "vsim tb_mux_demux_TMR; vcd file dump.vcd; vcd add -r sim:/*; run -all; exit"
VSIMSA: Configuration file changed: `/home/runner/library.cfg'
ALIB: Library "work" attached.
work = /home/runner/work/work.lib
Aldec, Inc. VHDL Compiler, build 2022.04.117
VLM Initialized with path: "/home/runner/library.cfg".
DAGGEN WARNING DAGGEN_0523: "The source is compiled without the -dbg switch. Line breakpoints and assertion debug will not be available."
COMP96 File: design.vhd
COMP96 Compile Entity "mux_demux_TMR"
COMP96 Compile Architecture "arch_1" of Entity "mux_demux_TMR"
COMP96 File: testbench.vhd
COMP96 Compile Entity "tb_mux_demux_TMR"
COMP96 Compile Architecture "arch_1" of Entity "tb_mux_demux_TMR"
COMP96 Top-level unit(s) detected:
COMP96 Entity => tb_mux_demux_TMR
COMP96 Compile success 0 Errors 0 Warnings  Analysis time :  40.0 [ms]
# Aldec, Inc. Riviera-PRO version 2022.04.117.8517 built for Linux64 on May 04, 2022.
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2022 Aldec, Inc. All rights reserved.
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.0 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initialization phase.
# ELAB2: Elaboration final pass...
# ELAB2: Create instances ...
# KERNEL: Time resolution set to 1ps.
# ELAB2: Create instances complete.
# SLP: Started
# SLP: Elaboration phase ...
# SLP: Elaboration phase ... skipped, nothing to simulate in SLP mode : 0.0 [s]
# SLP: Finished : 0.0 [s]
# ELAB2: You do not have a license to run VHDL performance optimized simulation. Contact Aldec for ordering information - sales@aldec.com.
# ELAB2: Elaboration final pass complete - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is reduced.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 5431 kB (elbread=427 elab2=4861 kernel=143 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO/bin/libsystf.so'
# EXECUTION:: NOTE    : Test done.
# EXECUTION:: Time: 64 ns,  Iteration: 0,  Instance: /tb_mux_demux_TMR,  Process: line__31.
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2022-09-27 21:55:52 EDT] Opening EPWave...
Done
```

Resultado: diagrama de ondas



Discussão dos resultados:

Igualmente o circuito anterior, o circuito 3 tem a mesma base do circuito 1, onde há um multiplexador como entrada e um demultiplexador como saída. Suas entradas principais são as mesmas que os circuitos anteriores: **i_SW0**, **i_SW1** e o seletor **i_SEL**. Suas saídas também são iguais: **o_LED0** e **o_LED1**. Junto a isso, foram adicionadas mais três entradas: **i_BUT1**, **i_BUT2** e **i_BUT3,** que fazem o papel das entradas do Triple Modular Redundancy.

Foi criado um votador à parte, usando lógicas já vistas em matérias anteriores e após o esboço utilizando o software *logisim*, que foi transformado de imagem para código em vhdl. O voter está localizado entre o mux e o demux e suas três entradas

são combinações **XOR** do resultado do multiplexador e dos três *push-buttons*. Para criar o votador, foram adicionadas três entradas (simulando as portas **XOR**) e cada entrada ligava-se nas outras duas com uma porta **AND** até que todas estivessem interligadas entre si. A saída é uma ligação **OR** entre as três entradas **AND**, gerando assim um circuito baseado em **maioria**.

Foi observado que, quando todos os *push-buttons* estão desligados, o TMR funciona igualmente como o circuito 1, onde cada saída recebe sua respectiva entrada dependendo do estado em que o seletor está atribuído (**o_LED0** = **i_SW0** com **i_SEL** em 0**,** e **o_LED1** = **i_SW1** com **i_SEL** em 1). Com apenas 1 *push-button* ligado, independentemente de qual seja ele, o funcionamento do circuito continuará com a mesma lógica apresentada anteriormente.

A partir do momento em que dois ou mais *push-buttons* são acionados ao mesmo tempo, as saídas passam a receber o valor contrário de suas entradas, dependendo de qual estado o seletor estará. Se o **i_SEL** estiver em 0, o **o_LED0** receberá o valor contrário de **i_SW0**. Se o **i_SEL** estiver em 1, o **o_LED1** receberá o valor contrário de **i_SW1**. Em ambos os casos a saída contrária será totalmente ignorada, em estado baixo, quando o seletor estiver em seu estado contrário. As entradas contrárias (**o_LED0** e **o_LED1**) também serão totalmente ignoradas quando o seletor estiver em estados diferentes (**i_SEL** = 1 e **i_SEL** = 0, respectivamente).

Links para os códigos no EDA Playground

Circuito 1: https://www.edaplayground.com/x/7wsb

Circuito 2: https://www.edaplayground.com/x/Qg_Z

Circuito 3: https://www.edaplayground.com/x/C6Yq