Computação Reconfigurável

Aula teórica 5

http://elearning.ua.pt/

Computação Reconfigurável

Aula 5

- **Projetos práticos para Nexys-4:**
  - **controlo de displays de segmentos;**
  - **gerador aleatório.**
- **Máquinas de estados finitos.**
- ***Look-up tables - LUTs*, slices, blocos lógicos programáveis.**
- **Utilização de blocos de memória embutidos.**
- **Vários projetos.**

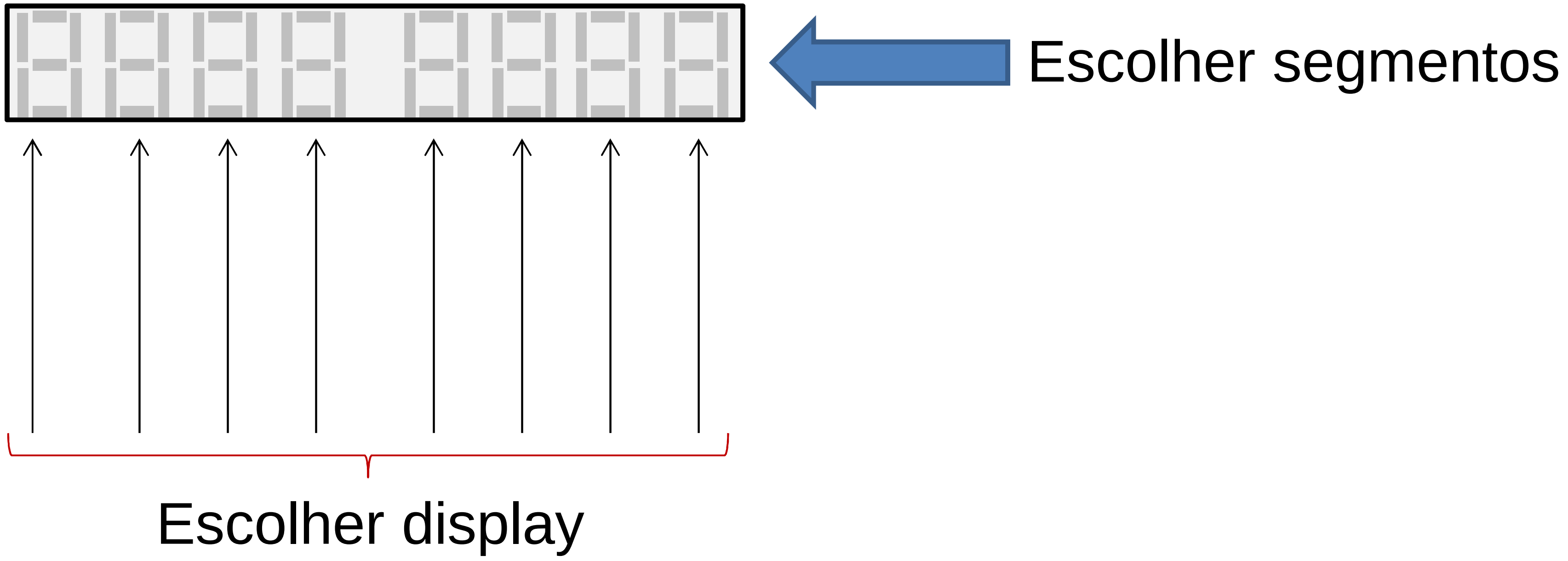
Bibliografia: V.Sklyarov, I.Skliarova, A.Barkalov, L.Titarenko. Synthesis and Optimization of FPGA-Based Systems. Springer, 2014.

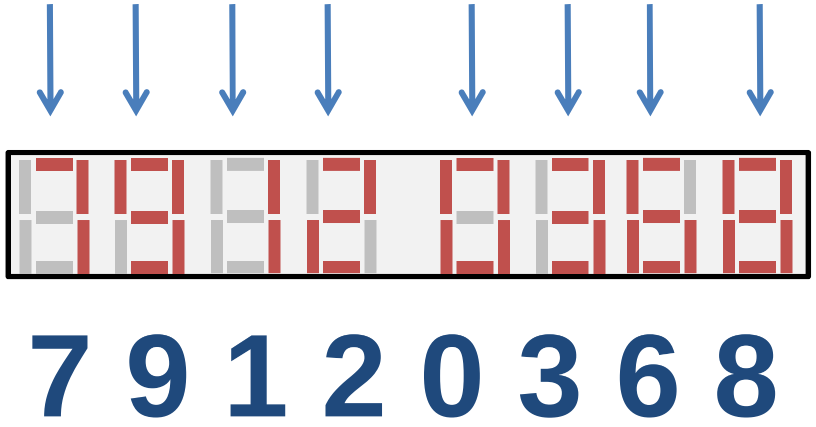
Aula 6

- **Paralelismo e concorrência.**
- **Implementação de redes de procura.**
- **Implementação de redes de ordenação.**
- **Computações *popcount* (peso e distância de *Hamming*).**

Bibliografia: V.Sklyarov, I.Skliarova, A.Barkalov, L.Titarenko. Synthesis and Optimization of FPGA-Based Systems. Springer, 2014.

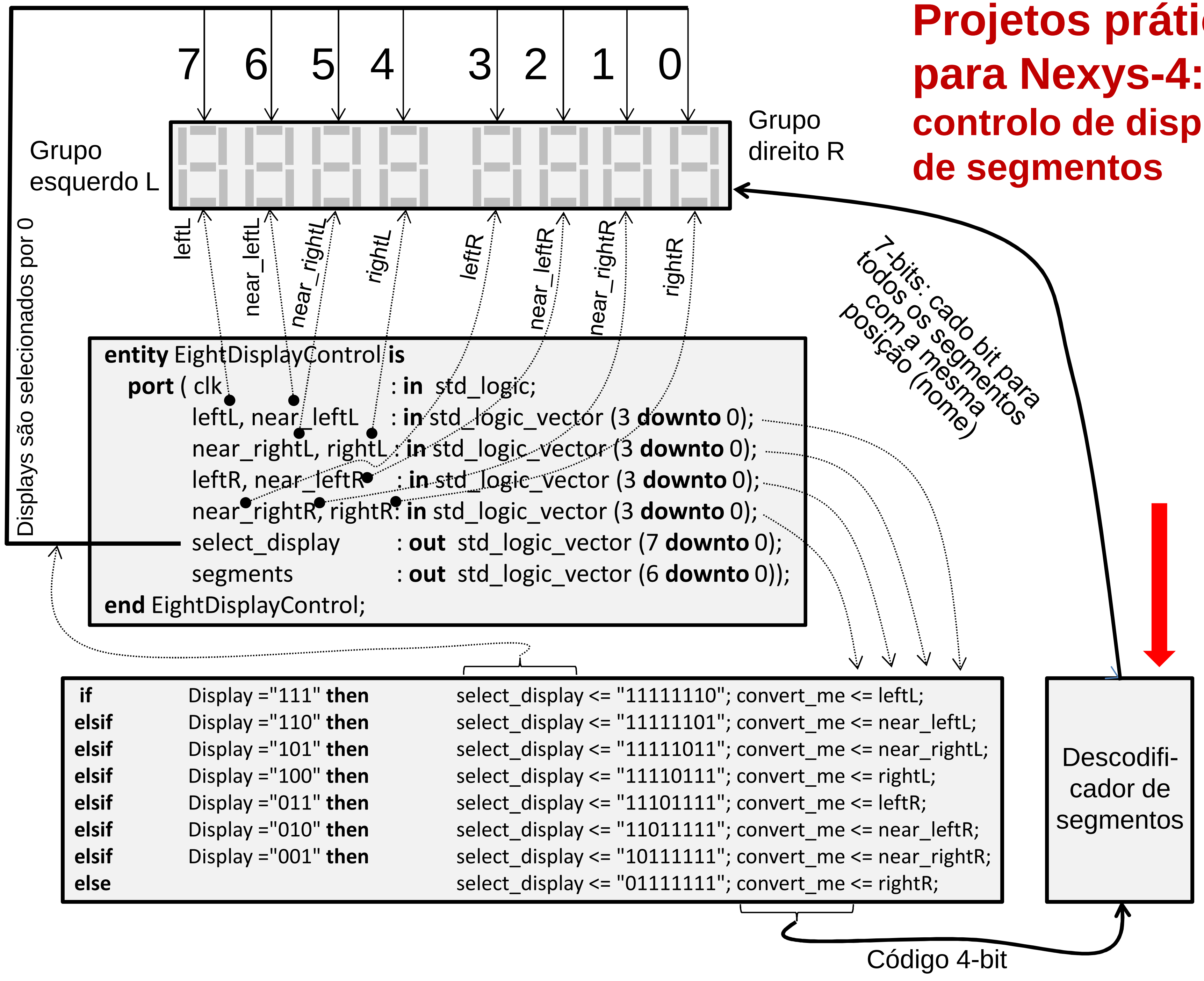
Projetos práticos para Nexys-4: controlo de displays de segmentos





Projetos práticos para Nexys-4:

controlo de displays de segmentos

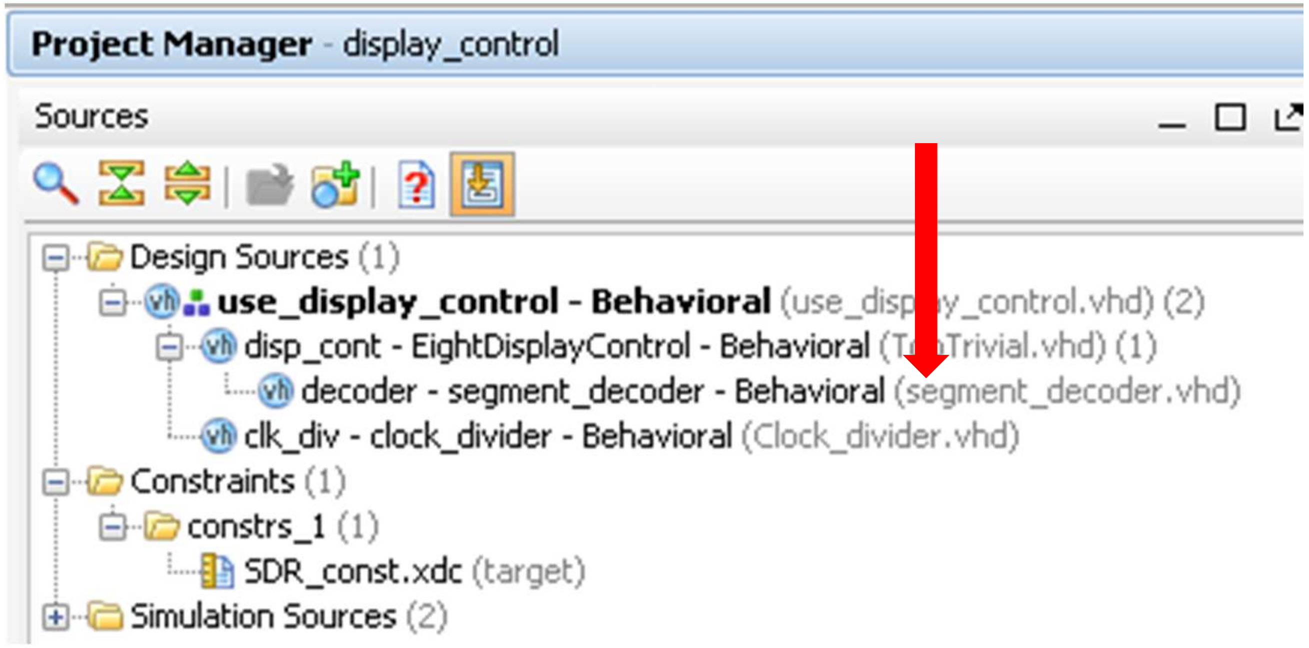

7
6
5
4
3
2
1
0
Grupo esquerdo L
Grupo direito R
leftL
near_leftL
near_rightL
rightL
leftR
near_leftR
near_rightR
rightR
Displays são selecionados por 0
7-bits: cado bit para todos os segmentos com a mesma posição (nome)
Descodifi-cador de segmentos
Código 4-bit

```vhdl
entity EightDisplayControl is
   port ( clk                    : in  std_logic;
          leftL, near_leftL      : in std_logic_vector (3 downto 0);
          near_rightL, rightL    : in std_logic_vector (3 downto 0);
          leftR, near_leftR      : in std_logic_vector (3 downto 0);
          near_rightR, rightR    : in std_logic_vector (3 downto 0);
          select_display         : out  std_logic_vector (7 downto 0);
          segments               : out  std_logic_vector (6 downto 0));
end EightDisplayControl;
```

```vhdl
 if      Display ="111" then     select_display <= "11111110"; convert_me <= leftL;
elsif    Display ="110" then     select_display <= "11111101"; convert_me <= near_leftL;
elsif    Display ="101" then     select_display <= "11111011"; convert_me <= near_rightL;
elsif    Display ="100" then     select_display <= "11110111"; convert_me <= rightL;
elsif    Display ="011" then     select_display <= "11101111"; convert_me <= leftR;
elsif    Display ="010" then     select_display <= "11011111"; convert_me <= near_leftR;
elsif    Display ="001" then     select_display <= "10111111"; convert_me <= near_rightR;
else                             select_display <= "01111111"; convert_me <= rightR;
```

Projetos práticos para Nexys-4: controlo de displays de segmentos

Estrutura do projeto


Project Manager - display_control

Sources

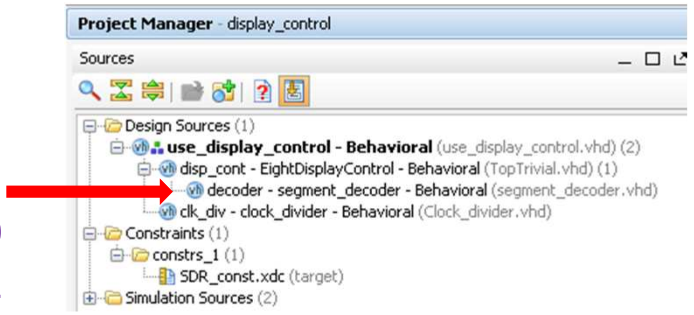
Design Sources (1)
- use_display_control - Behavioral (use_display_control.vhd) (2)
  - disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
    - decoder - segment_decoder - Behavioral (segment_decoder.vhd)
  - clk_div - clock_divider - Behavioral (Clock_divider.vhd)

Constraints (1)
- constrs_1 (1)
  - SDR_const.xdc (target)

Simulation Sources (2)

Descodificador de segmentos

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity segment_decoder is
port ( BCD                        :  in  std_logic_vector (3 downto 0);       -- entrada
        segments                   : out  std_logic_vector (7 downto 1));     -- saída
end segment_decoder;

architecture Behavioral of segment_decoder is
begin
  segments <=       "1000000" when BCD = "0000" else        -- 0
                    "1111001" when BCD = "0001" else        -- 1
                    "0100100" when BCD = "0010" else        -- 2
                    "0110000" when BCD = "0011" else        -- 3
                    "0011001" when BCD = "0100" else        -- 4
                    "0010010" when BCD = "0101" else        -- 5
                    "0000010" when BCD = "0110" else        -- 6
                    "1111000" when BCD = "0111" else        -- 7
                    "0000000" when BCD = "1000" else        -- 8
                    "0010000" when BCD = "1001" else        -- 9
                    "0001000" when BCD = "1010" else         -- a
                    "0000011" when BCD = "1011" else        -- b
                    "1000110" when BCD = "1100" else        -- c
                    "0100001" when BCD = "1101" else        -- d
                    "0000110" when BCD = "1110" else        -- e
                    "0001110" when BCD = "1111" else        -- f
                    "1111111";             -- todos os segmentos são passivos
end Behavioral;
```



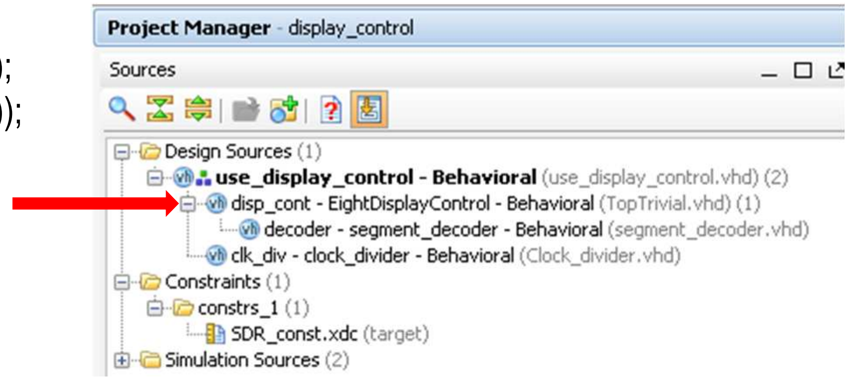
Project Manager - display_control

Sources

```
⊟ Design Sources (1)
   ⊟ use_display_control - Behavioral (use_display_control.vhd) (2)
      ⊟ disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
            decoder - segment_decoder - Behavioral (segment_decoder.vhd)
         clk_div - clock_divider - Behavioral (Clock_divider.vhd)
⊟ Constraints (1)
   ⊟ constrs_1 (1)
         SDR_const.xdc (target)
⊞ Simulation Sources (2)
```

Controlador de displays

```
library IEEE;  use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.all;
entity EightDisplayControl is
   port (           clk                   : in  std_logic;
                    leftL, near_leftL     : in std_logic_vector (3 downto 0);
                    near_rightL, rightL   : in std_logic_vector (3 downto 0);
                    leftR, near_leftR     : in std_logic_vector (3 downto 0);
                    near_rightR, rightR   : in std_logic_vector (3 downto 0);
                    select_display        : out std_logic_vector (7 downto 0);
                    segments              : out std_logic_vector (6 downto 0));
end EightDisplayControl;
architecture Behavioral of EightDisplayControl is
  signal Display             : std_logic_vector(2 downto 0);
  signal div                 : std_logic_vector(16 downto 0);
  signal convert_me          : std_logic_vector(3 downto 0);
begin
               div<= div + 1 when rising_edge(clk);
               Display <= div(16 downto 14);
process(Display,  leftL, near_leftL, near_rightL, rightL, leftR, near_leftR, near_rightR, rightR)
begin          -- ativação sequencial dos displays
  if           Display ="111" then      select_display <= "01111111"; convert_me <= leftL;
  elsif        Display ="110" then      select_display <= "10111111"; convert_me <= near_leftL;
  elsif        Display ="101" then      select_display <= "11011111"; convert_me <= near_rightL;
  elsif        Display ="100" then      select_display <= "11101111"; convert_me <= rightL;
  elsif        Display ="011" then      select_display <= "11110111"; convert_me <= leftR;
  elsif        Display ="010" then      select_display <= "11111011"; convert_me <= near_leftR;
  elsif        Display ="001" then      select_display <= "11111101"; convert_me <= near_rightR;
  else                                  select_display <= "11111110"; convert_me <= rightR;
  end if;
end process;
decoder :         entity work.segment_decoder                      -- descodificador de segmentos
                  port map (convert_me, segments);
end Behavioral;
```

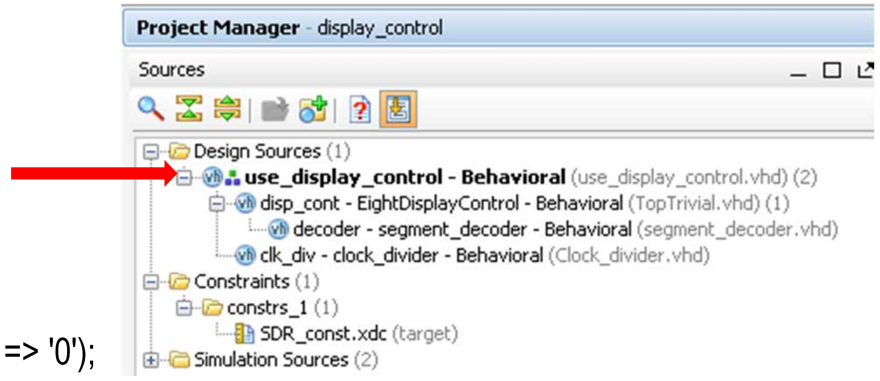

Project Manager - display_control

Sources

```
Design Sources (1)
  use_display_control - Behavioral (use_display_control.vhd) (2)
    disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
      decoder - segment_decoder - Behavioral (segment_decoder.vhd)
    clk_div - clock_divider - Behavioral (Clock_divider.vhd)
Constraints (1)
  constrs_1 (1)
    SDR_const.xdc (target)
Simulation Sources (2)
```

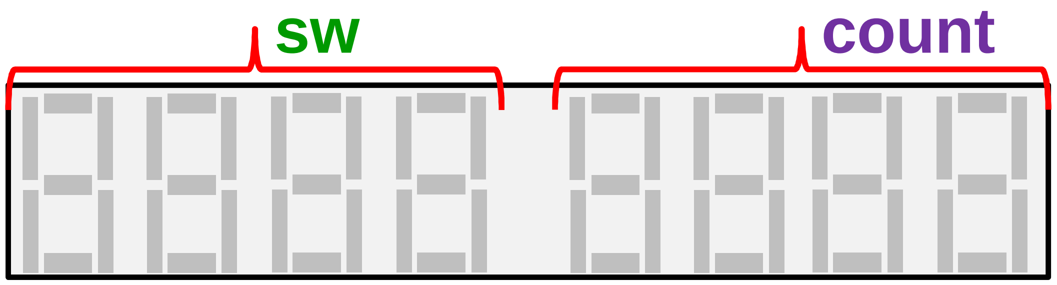
Utilização do controlador

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

entity use_display_control is
   Port ( clk                    : in std_logic;
          sw                     : in STD_LOGIC_VECTOR (15 downto 0);
          seg                    : out STD_LOGIC_VECTOR (6 downto 0);
          sel_disp               : out STD_LOGIC_VECTOR (7 downto 0));
end use_display_control;

architecture Behavioral of use_display_control is
                 signal count      : std_logic_vector(15 downto 0) := (others => '0');
                 signal divided_clk  : std_logic;
begin

count <= count+1 when divided_clk'event and divided_clk='1';  -- rising_edge(divided_clk);

disp_cont: entity work.EightDisplayControl
   port map ( clk=>clk,leftL=>sw(15 downto 12),near_leftL=>sw(11 downto 8),near_rightL=>sw(7 downto 4),rightL=>sw(3 downto 0),
              leftR=>count(15 downto 12),near_leftR=>count(11 downto 8),near_rightR=>count(7 downto 4),rightR=>count(3 downto 0),
              select_display=>sel_disp, segments=>seg);

clk_div: entity work.clock_divider
         generic map        (how_fast => 26  )
         port map           ( clk, divided_clk);

end Behavioral;
```


Project Manager - display_control

Sources

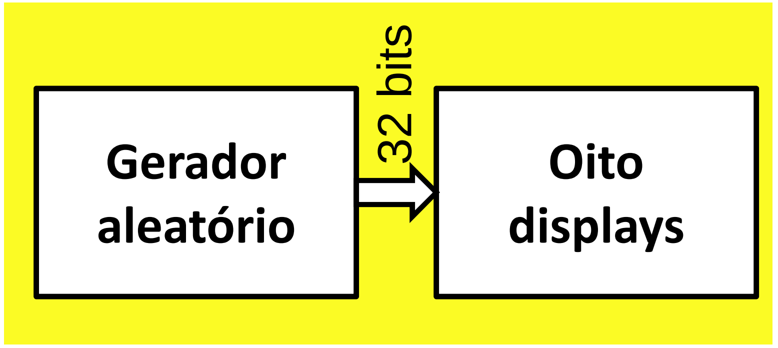
```
Design Sources (1)
  use_display_control - Behavioral (use_display_control.vhd) (2)
    disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
      decoder - segment_decoder - Behavioral (segment_decoder.vhd)
    clk_div - clock_divider - Behavioral (Clock_divider.vhd)
Constraints (1)
  constrs_1 (1)
    SDR_const.xdc (target)
Simulation Sources (2)
```



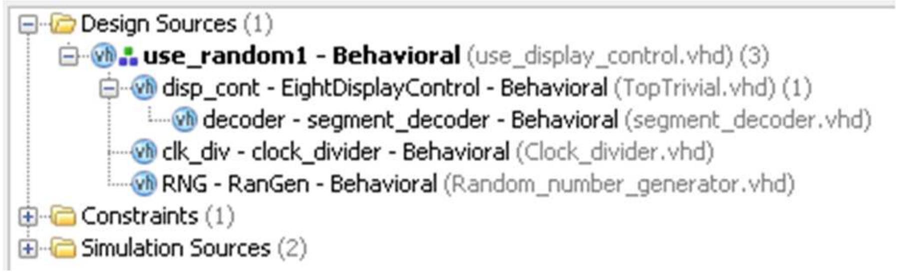
Projetos práticos para Nexys-4: gerador aleatório

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity RanGen is
   generic ( width                 : integer :=  32   );         -- tamanho de números aleatórios
   port (    clk                   : in std_logic;               -- relógio
             random_num            : out std_logic_vector (width-1 downto 0)      ); -- número gerado
end RanGen;

architecture Behavioral of RanGen is

begin

process(clk)
   variable rand_temp  : std_logic_vector(width-1 downto 0):=(width-1 => '1', others => '0');
   variable temp       : std_logic := '0';
begin
   if(rising_edge(clk)) then
      temp                         := rand_temp(width-1) xor rand_temp(width-2);
      rand_temp(width-1 downto 1)  := rand_temp(width-2 downto 0);
      rand_temp(0)                 := temp;
   end if;
   random_num <= rand_temp;
end process;

end Behavioral;
```

Projetos práticos para Nexys-4: utilização do gerador aleatório



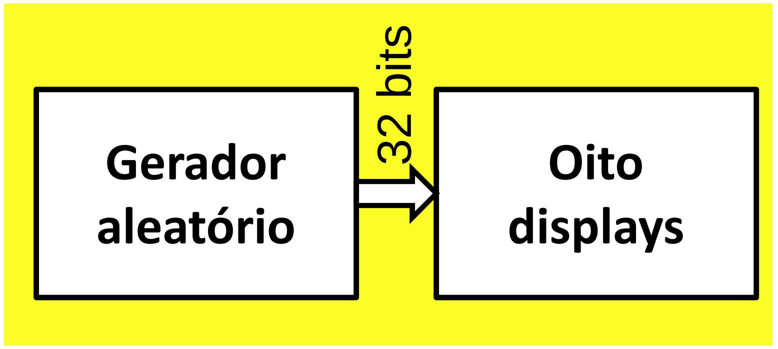
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity use_random1 is
   port (     clk          : in std_logic;
              sw           : in std_logic_vector (15 downto 0);
              seg          : out std_logic_vector (6 downto 0);
              sel_disp     : out std_logic_vector (7 downto 0));
end use_random1;
architecture Behavioral of use_random1 is
              signal count             : std_logic_vector(31 downto 0) := (others => '0');
              signal divided_clk       : std_logic;
begin
disp_cont: entity work.EightDisplayControl
   port map ( clk=>clk, leftL=>count(31 downto 28), near_leftL=>count(27 downto 24),
                near_rightL=>count(23 downto 20), rightL=>count(19 downto 16),
                leftR=>count(15 downto 12), near_leftR=>count(11 downto 8),
                near_rightR=>count(7 downto 4), rightR=>count(3 downto 0),
                 select_display=>sel_disp,segments=>seg);

clk_div: entity work.clock_divider
        generic map     (how_fast => 26  )
        port map        ( clk, divided_clk);

RNG :   entity work.RanGen
          generic map (32)                                            -- tamanho de números aleatórios
          port map   (clk=>divided_clk,random_num=>count);            -- número gerado

end Behavioral;
```



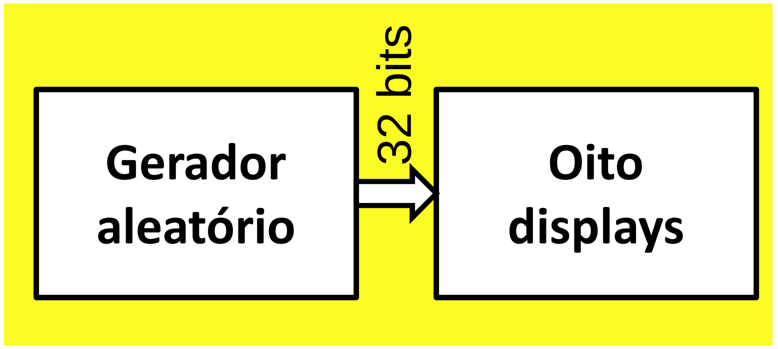
```
⊟ Design Sources (1)
   ⊟ use_random1 - Behavioral (use_display_control.vhd) (3)
      ⊟ disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
           decoder - segment_decoder - Behavioral (segment_decoder.vhd)
         clk_div - clock_divider - Behavioral (Clock_divider.vhd)
         RNG - RanGen - Behavioral (Random_number_generator.vhd)
⊞ Constraints (1)
⊞ Simulation Sources (2)
```

Projetos práticos para Nexys-4: utilização do gerador aleatório

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

entity use_random2 is
   port (     clk          : in std_logic;
              sw           : in std_logic_vector (15 downto 0);
              seg          : out std_logic_vector (6 downto 0);
              sel_disp     : out std_logic_vector (7 downto 0));
end use_random2;

architecture Behavioral of use_random2 is
              signal count         : std_logic_vector(31 downto 0) := (others => '0');
              signal random_num    : std_logic_vector(31 downto 0) := (others => '0');
              signal divided_clk   : std_logic;
begin

              count <= random_num when rising_edge(divided_clk);

disp_cont: entity work.EightDisplayControl
              -- ver slide anterior
clk_div: entity work.clock_divider
              -- ver slide anterior

RNG :   entity work.RanGen
          generic map (32)
          port map    (clk=>clk,random_num=>random_num);

end Behavioral;
```

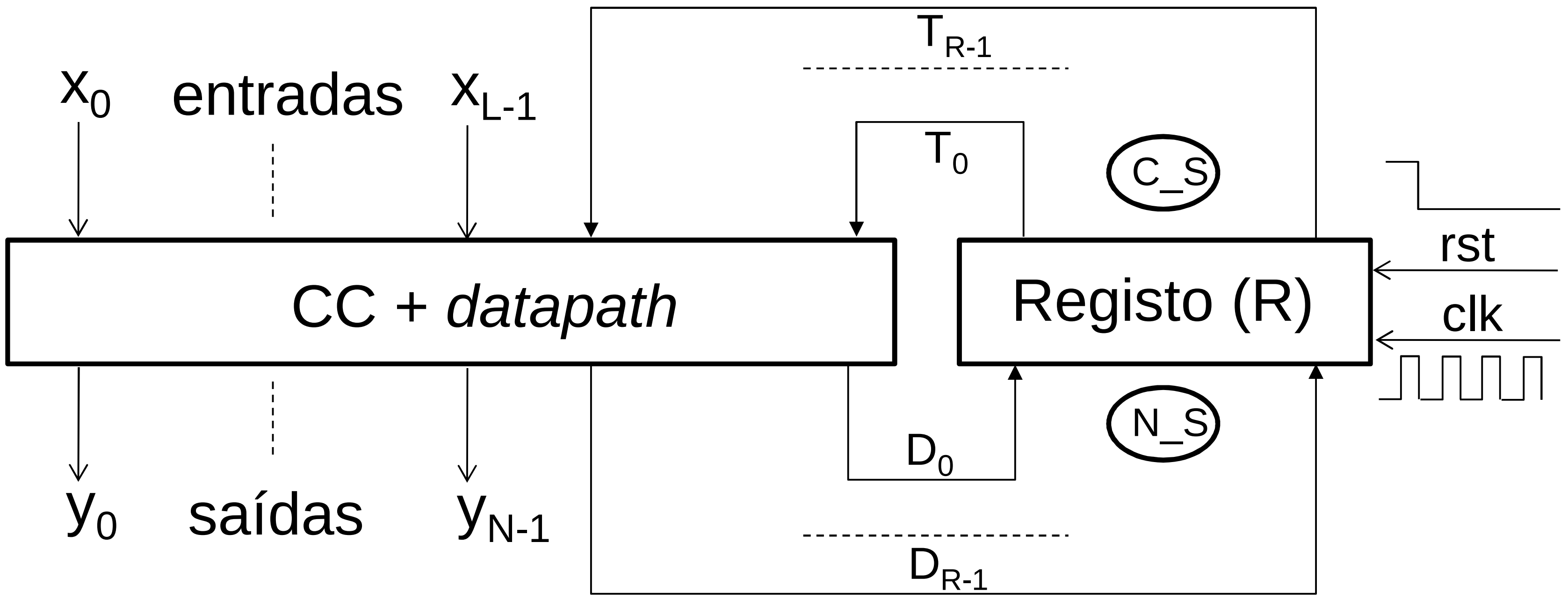


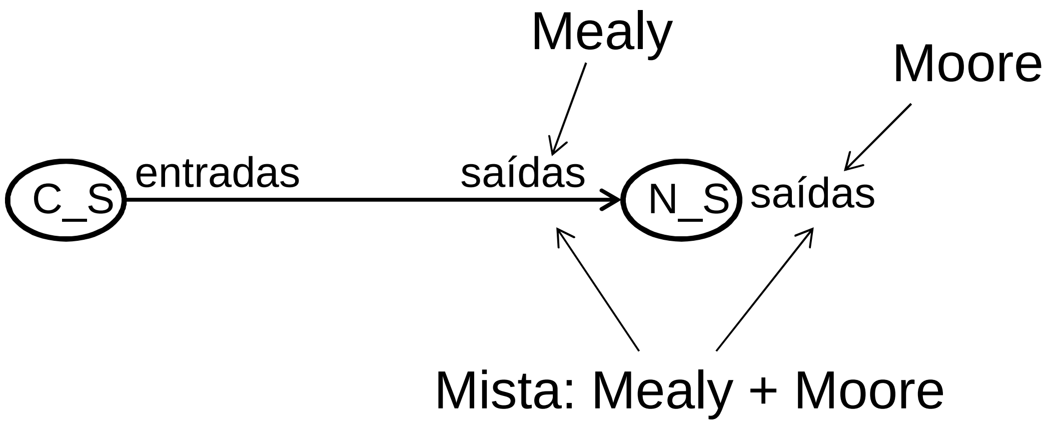
Projetos práticos para Nexys-4: utilização do gerador aleatório



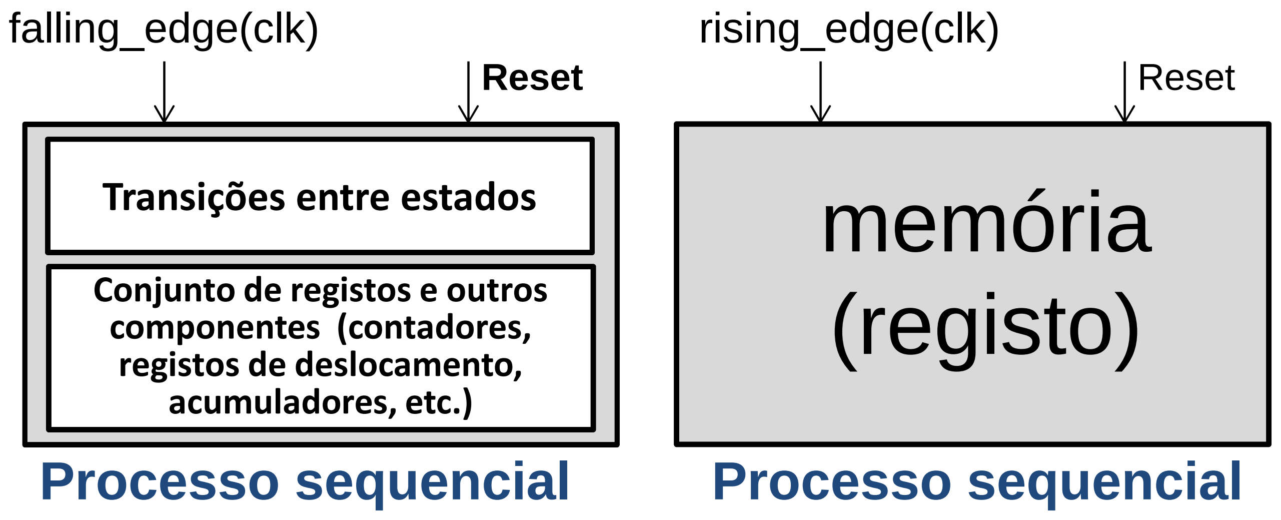
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity use_random2 is
   generic ( N              : integer := 32);
   port (      clk          : in std_logic;
               btnC         : in std_logic;
               sw           : in std_logic_vector (15 downto 0);
               seg          : out std_logic_vector (6 downto 0);
               sel_disp     : out std_logic_vector (7 downto 0));
end use_random2;

architecture Behavioral of use_random2 is
               signal count          : std_logic_vector(N-1 downto 0) := (others => '0');
               signal random_num     : std_logic_vector(N-1 downto 0) := (others => '0');
begin

count <= random_num when rising_edge(clk) and (btnC='1');
random_num <= random_num+1 when rising_edge(clk);

disp_cont: entity work.EightDisplayControl
    port map ( clk=>clk, leftL=>count(31 downto 28), near_leftL=>count(27 downto 24),
                    near_rightL=>count(23 downto 20), rightL=>count(19 downto 16),
                    leftR=>count(15 downto 12), near_leftR=>count(11 downto 8),
                    near_rightR=>count(7 downto 4), rightR=>count(3 downto 0),
                    select_display=>sel_disp,segments=>seg);

end Behavioral;
```

Máquinas de estados finitos com unidade de execução

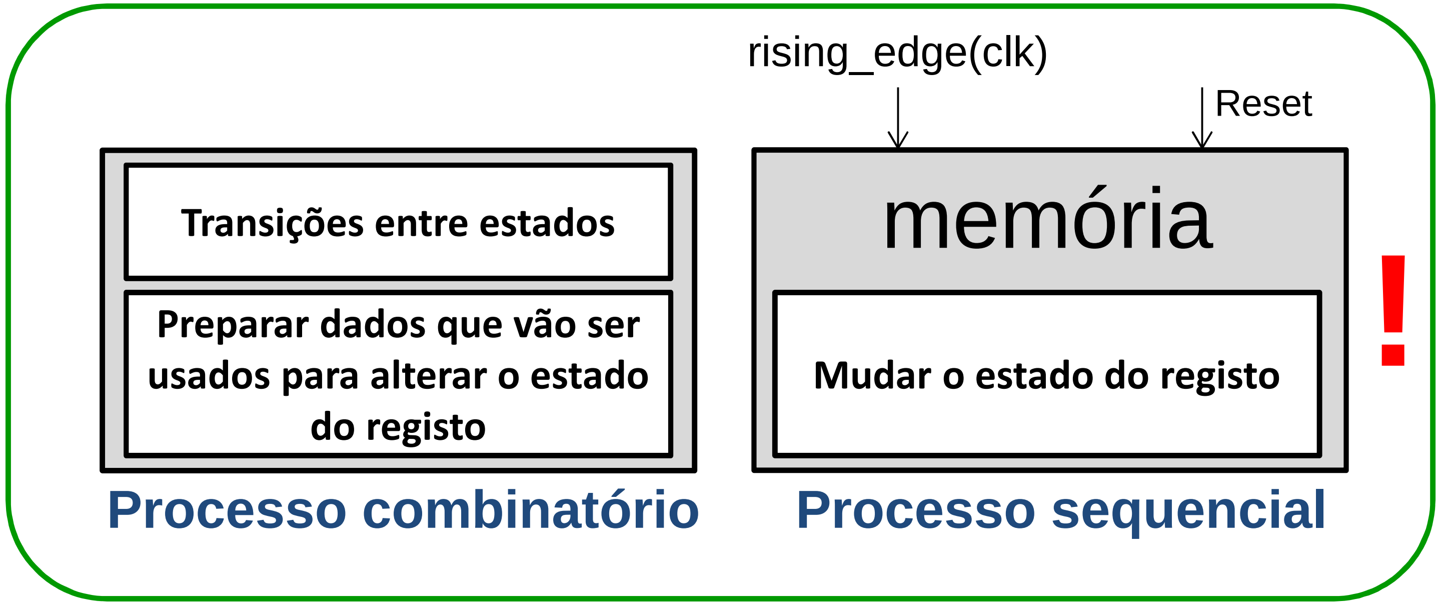




Máquinas de estados finitos. Sincronização



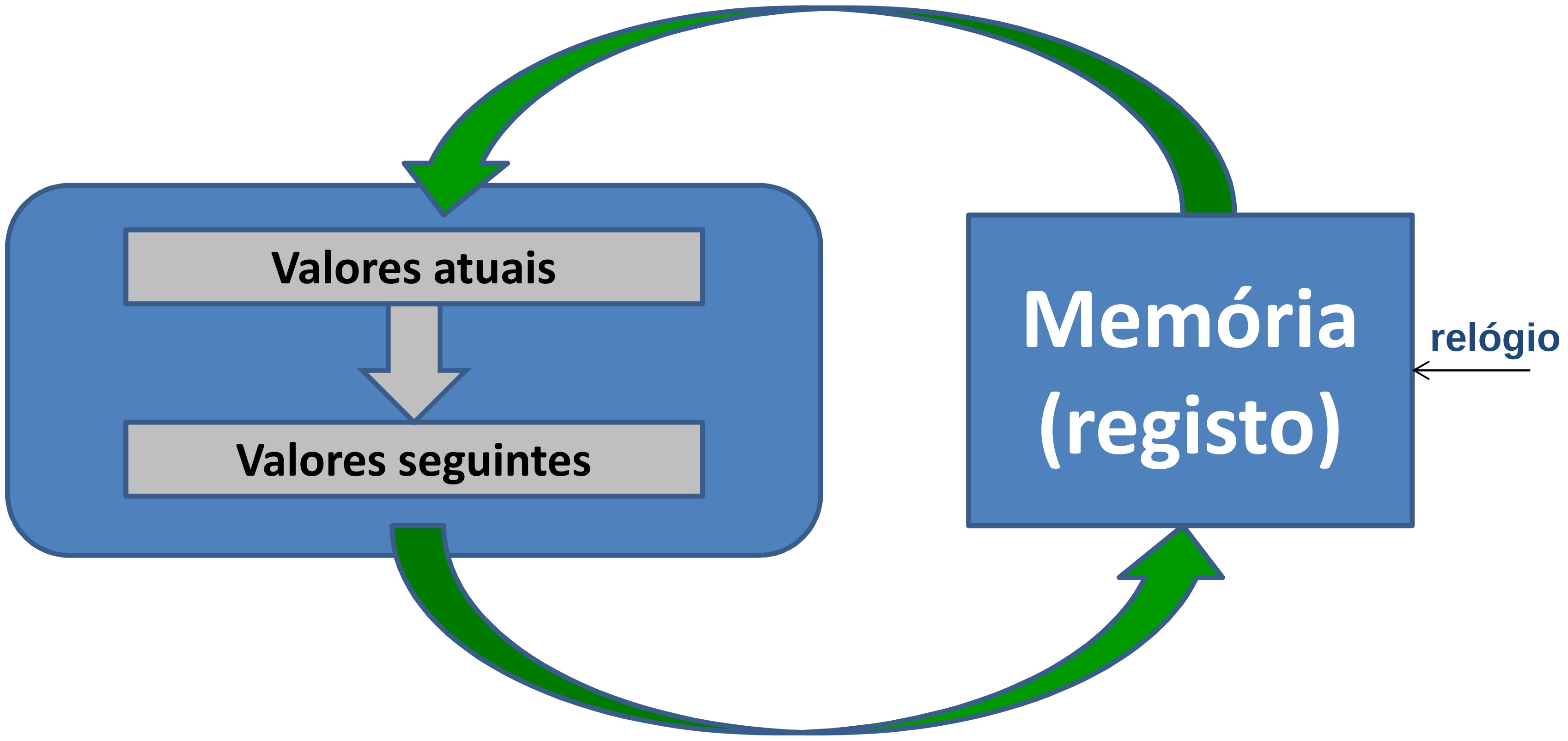
Mais simples



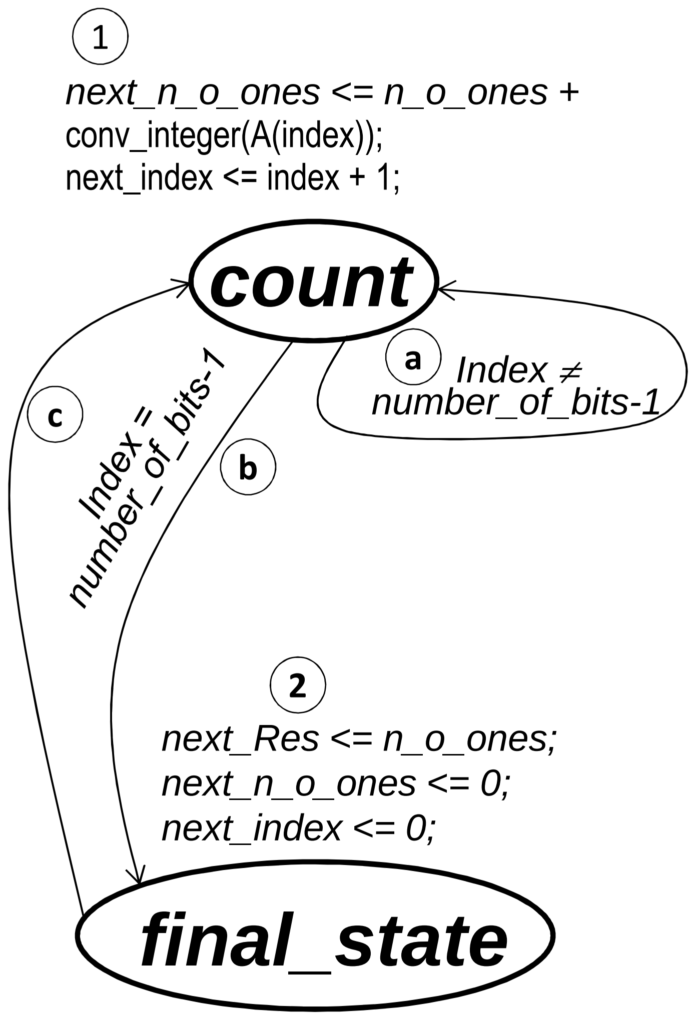
Mais rápido

Sincronização só para 0→1 ou 1→0

rising_edge(clk)
falling_edge(clk)



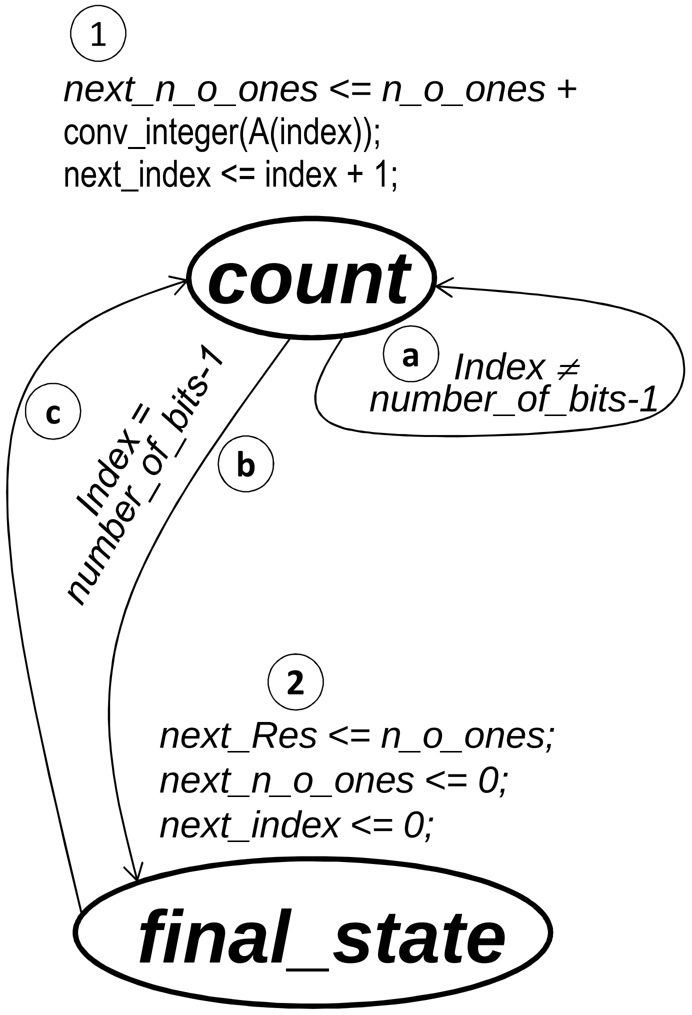
Máquinas de Estados Finitos



Exemplo

Contar o número de uns num vetor binário

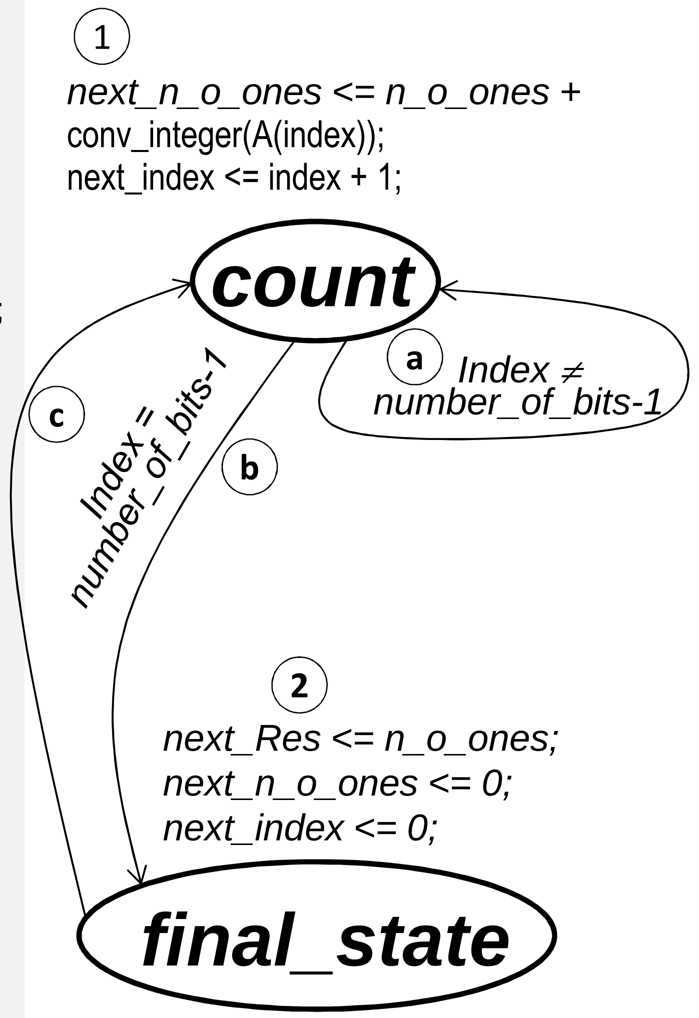
Máquinas de Estados Finitos



```
process (clk)                                  -- processo sequencial
begin
  if rising_edge(clk) then
    if (rst = '1') then C_S <= count; index <= 0; n_o_ones <= 0; Res <= 0;
    else       C_S <= N_S;
               index          <= next_index;      -- índice do vetor
               n_o_ones       <= next_n_o_ones;   -- número de uns
               Res            <= next_Res;        -- resultado
    end if;
  end if;
end process;
process (C_S, A, index, n_o_ones, Res)  -- processo combinatório
begin
  N_S                        <= C_S;
  next_index                 <= index;
  next_n_o_ones              <= n_o_ones;
  next_Res                   <= Res;
case C_S is
when count => next_index <= index + 1; N_S <= count;
    next_n_o_ones <= n_o_ones + conv_integer(A(index));
    if(index = number_of_bits-1) then N_S <= final_state;
    end if;
when final_state => N_S <= count;
    next_Res <= n_o_ones; next_n_o_ones <= 0; next_index <= 0;
when others => N_S <= count;
end case;
end process;
Result <= conv_std_logic_vector(Res, 8);   -- resultado
```

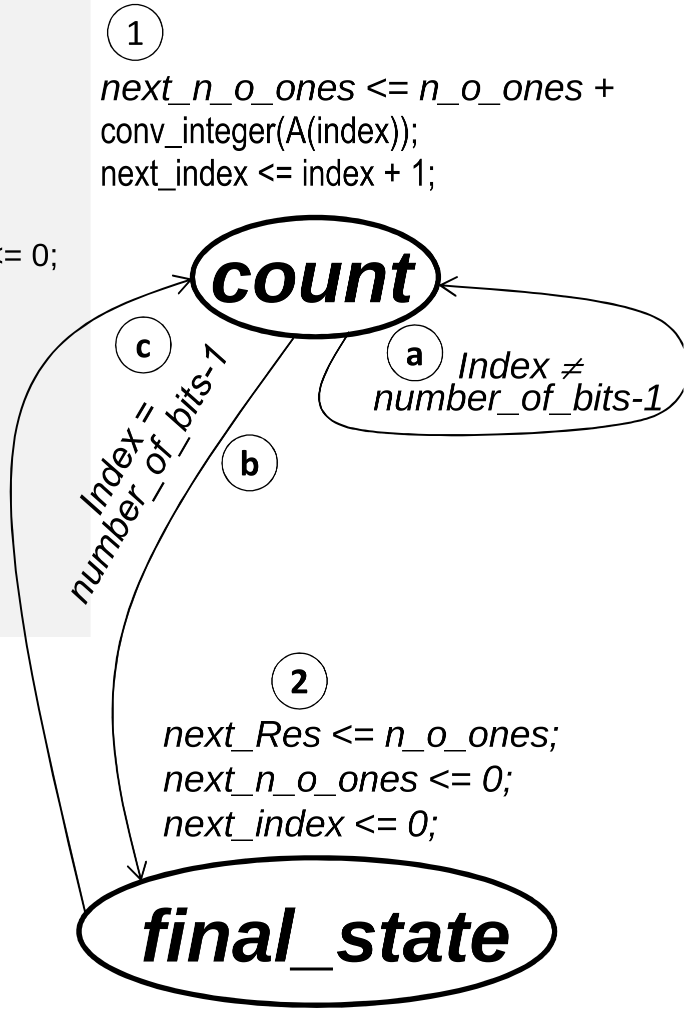
Projeto completo

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

entity FSM_count_ones is
              generic( number_of_bits   : integer := 16);
  port (      clk                       : in std_logic;
              btnC                      : in std_logic;
              sw                        : in  STD_LOGIC_VECTOR (15 downto 0);
              led                       : out  STD_LOGIC_VECTOR (4 downto 0));
end FSM_count_ones;

architecture Behavioral of FSM_count_ones is

type state_type is (initial_state, final_state);          -- enumeração de estados
signal C_S, N_S                        : state_type;
signal index, next_index               : integer range 0 to number_of_bits-1;
signal Res, next_Res                   : integer range 0 to number_of_bits;
signal n_o_ones, next_n_o_ones         : integer range 0 to number_of_bits;

begin

process (clk)                           -- processo sequencial
begin
  if rising_edge(clk) then
    if (btnC = '1') then C_S <= initial_state; index <= 0; n_o_ones <= 0; Res <= 0;
    else      C_S <= N_S;
              index        <= next_index;            -- índice do vetor
              n_o_ones     <= next_n_o_ones;         -- número de uns
              Res          <= next_Res;              -- resultado
    end if;
  end if;
end process;
```



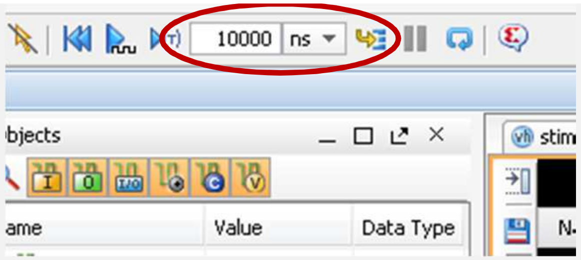
Projeto completo

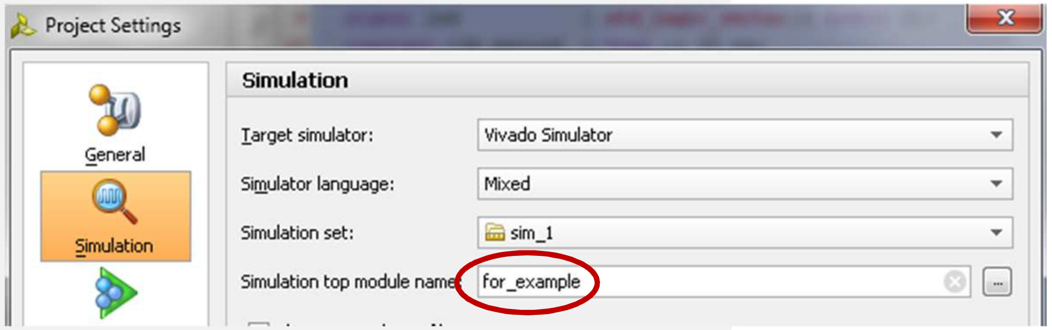
```
process (C_S, Sw, index, n_o_ones, Res)  -- processo combinatório
begin
  N_S                      <= C_S;
  next_index               <= index;
  next_n_o_ones            <= n_o_ones;        importante
  next_Res                 <= Res;
case C_S is
              when initial_state => next_index <= index + 1; N_S <= initial_state;
                            next_n_o_ones <= n_o_ones + conv_integer(sw(index));
                            if(index = number_of_bits-1) then N_S <= final_state;
                            end if;
              when final_state => N_S <= initial_state;
                            next_Res <= n_o_ones; next_n_o_ones <= 0; next_index <= 0;
              when others => N_S <= initial_state;
end case;
end process;

led <= conv_std_logic_vector(Res, 5);      -- resultado

end Behavioral;
```



Simulação

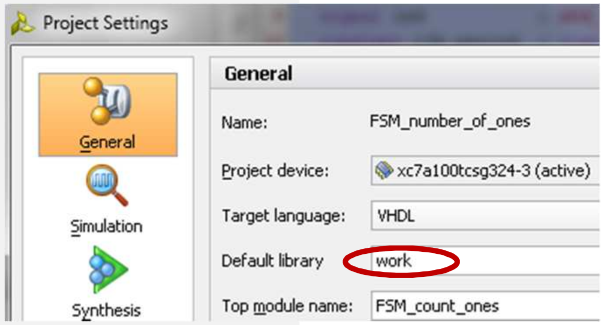
```
library IEEE;
use IEEE.std_logic_1164.all;

entity for_example is
end for_example;
architecture behavior of for_example is
   signal rst , clk             : std_logic := '0';
   signal sw                    : std_logic_vector(15 downto 0);
   signal led                   : std_logic_vector(4 downto 0);
   constant clk_period          : time := 50 ns;
 begin
 uut: entity work.FSM_count_ones
        port map (clk, rst, sw,led);

clk_generator: process
begin
   clk <= '0';                  wait for clk_period/2;
   clk <= '1';                  wait for clk_period/2;
end process clk_generator;
stim_proc: process
begin
   rst <= '1';                                                  wait for 130 ns;
   rst <= '0'; sw <= (15 downto 10 => '1', others => '0');      wait for 2000 ns;
   rst <= '1';                                                  wait for 130 ns;
   rst <= '0'; sw <= "1010101000101101";                        wait for 2000 ns;
   rst <= '1';                                                  wait for 130 ns;
   rst <= '0'; sw <= "0000000000000000";                        wait for 2000 ns;
   rst <= '1';                                                  wait for 130 ns;
   rst <= '0'; sw <= "1010100000000000";                        wait for 2000 ns;
end process;
end behavior;
```


10000 ns


Project Settings

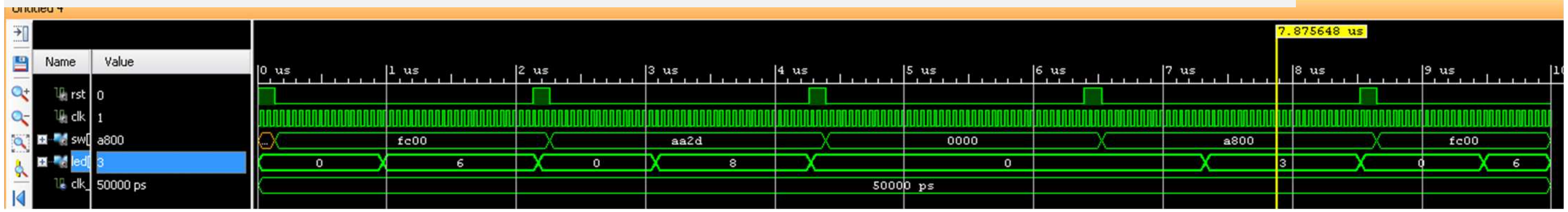
Simulation

Target simulator: Vivado Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: for_example


Project Settings

General

Name: FSM_number_of_ones
Project device: xc7a100tcsg324-3 (active)
Target language: VHDL
Default library: work
Top module name: FSM_count_ones

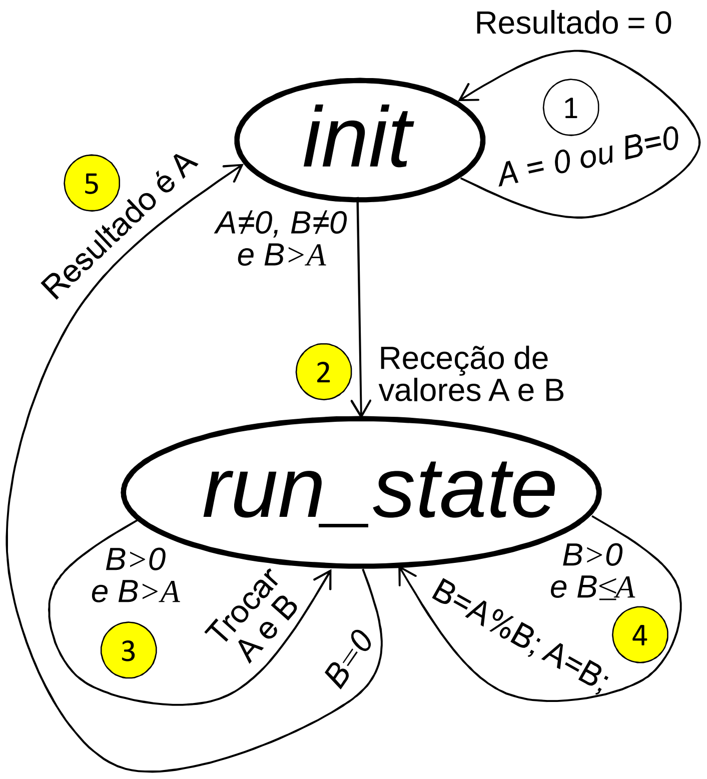

```
Name     Value
rst      0
clk      1
sw[      a800
led[     3
clk_     50000 ps
```

Exemplo

Código em C/Java

```
int IGCD(int A, int B)
{   int tmp;
    while (B > 0)
            {   if (B > A)       { tmp = A;          A = B;            B = tmp; }
                else             { tmp = B;          B = A%B;          A = tmp; }
            }
    return A;
 }
```

Encontrar o divisor máximo comum de dois inteiros positivos



```
int IGCD(int A, int B)                                   2
{   int tmp;
    while (B > 0)
    {   if (B > A) { tmp=A; A=B; B=tmp;    } 3
        else       { tmp=B; B= A%B; A=tmp; }  4
}
    return A; 5
 }
```

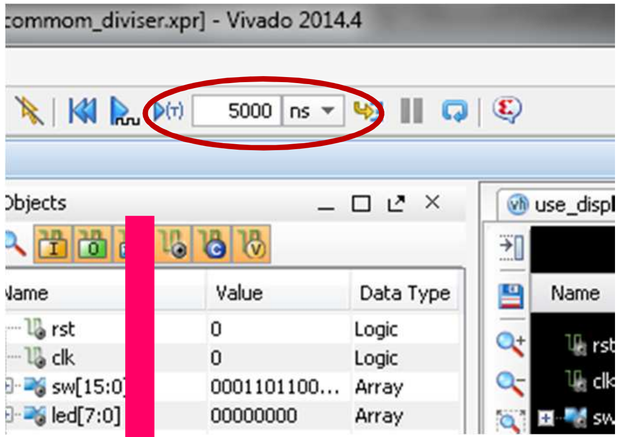
Código em C/Java

Projeto completo

```
library IEEE;              -- divisor máximo comum de dois inteiros positivos
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FSM_new is
generic(number_of_bits          : integer := 16);
port (       clk                : in  std_logic;
             btnC               : in  std_logic;
             sw                 : in std_logic_vector(number_of_bits-1 downto 0);
             led                : out std_logic_vector(7 downto 0)       );
end FSM_new;
architecture Behavioral of FSM_new is
signal A,B,FSM_A, FSM_B, FSM_A_next, FSM_B_next   : integer range 0 to 255;
type state_type is (init, run_state);
signal C_S, N_S                                   : state_type;
signal Res, Res_next                              : integer range 0 to 255;
begin
A <= conv_integer(sw(15 downto 8));   B <= conv_integer(sw(7 downto 0));
process (clk)                                  -- processo sequencial
begin
if rising_edge(clk) then
             if (btnC = '1') then       C_S <= init;
             else                       C_S <= N_S;
                                        FSM_A <= FSM_A_next;
                                        FSM_B <= FSM_B_next;
                                        Res <= Res_next;
             end if;
end if;
end process;
```

```
process (C_S, A, B, FSM_A, FSM_B, Res)  -- processo combinatório
begin
                N_S <= C_S;
                FSM_A_next <= FSM_A;
                FSM_B_next <= FSM_B;
                Res_next <= Res;
case C_S is
                when init =>
                                if ((A = 0) or (B = 0)) then              Res_next <= 0;
                                                                          N_S <= init;
                                else                                      FSM_A_next <= A;
                                                                          FSM_B_next <= B;
                                                                          N_S <= run_state;
                                end if;
                when run_state =>
                                if (FSM_B>0) then                         N_S <= run_state;
                                                if (FSM_B>FSM_A) then                     FSM_A_next <= FSM_B;
                                                                                          FSM_B_next <= FSM_A;
                                                else                                      FSM_A_next <= FSM_B;
                                                                                          FSM_B_next <= FSM_A rem FSM_B;
                                                end if;
                                else                                      Res_next <= FSM_A;
                                                                          N_S <= init;
                                end if;
                when others => N_S <= init;
end case;
end process;

led <= conv_std_logic_vector(Res,8);

end Behavioral;
```

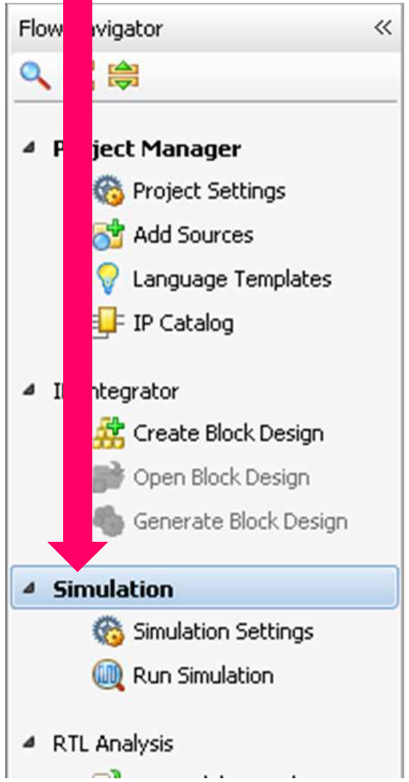
Simulação


commom_diviser.xpr] - Vivado 2014.4

5000 ns

Objects

Name	Value	Data Type
rst	0	Logic
clk	0	Logic
sw[15:0]	0001101100...	Array
led[7:0]	00000000	Array

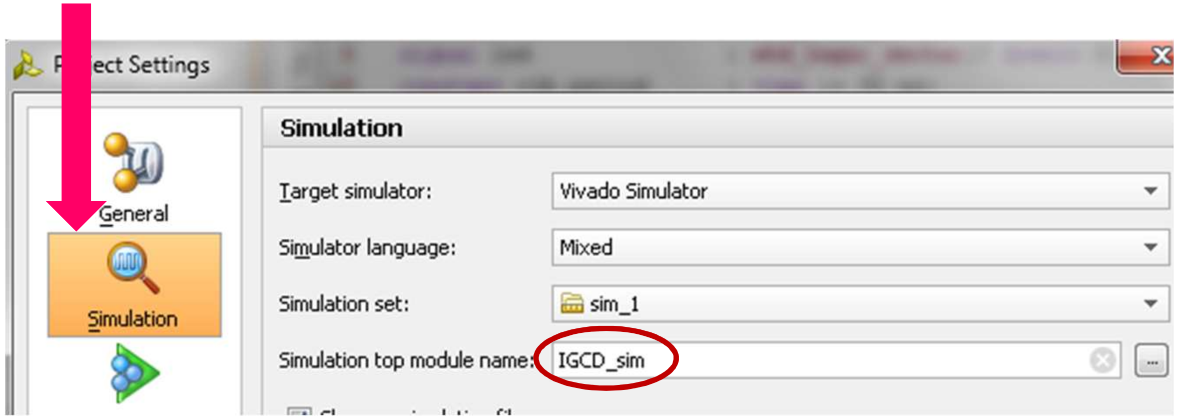

Flow Navigator

Project Manager
- Project Settings
- Add Sources
- Language Templates
- IP Catalog

IP Integrator
- Create Block Design
- Open Block Design
- Generate Block Design

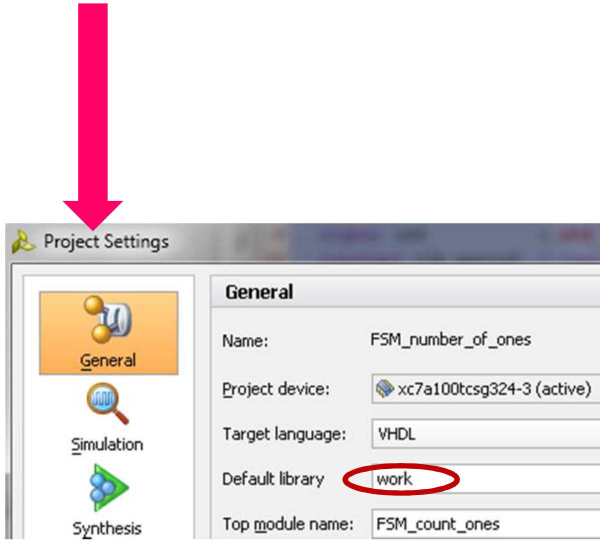
Simulation
- Simulation Settings
- Run Simulation

RTL Analysis


Project Settings

Simulation

Target simulator: Vivado Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: IGCD_sim

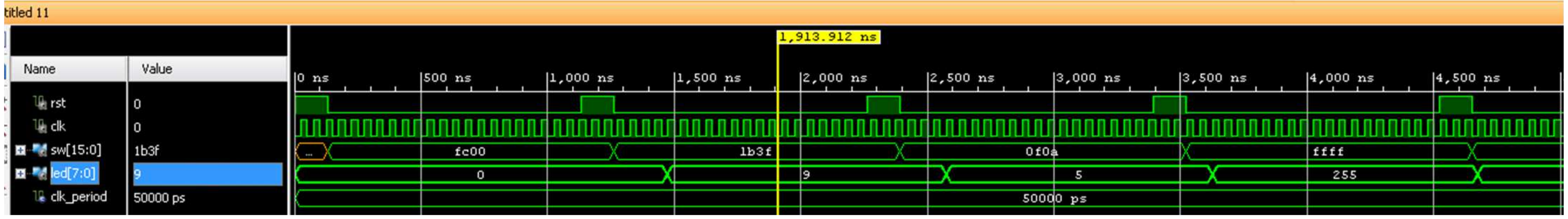

Project Settings

General

Name: FSM_number_of_ones
Project device: xc7a100tcsg324-3 (active)
Target language: VHDL
Default library: work
Top module name: FSM_count_ones

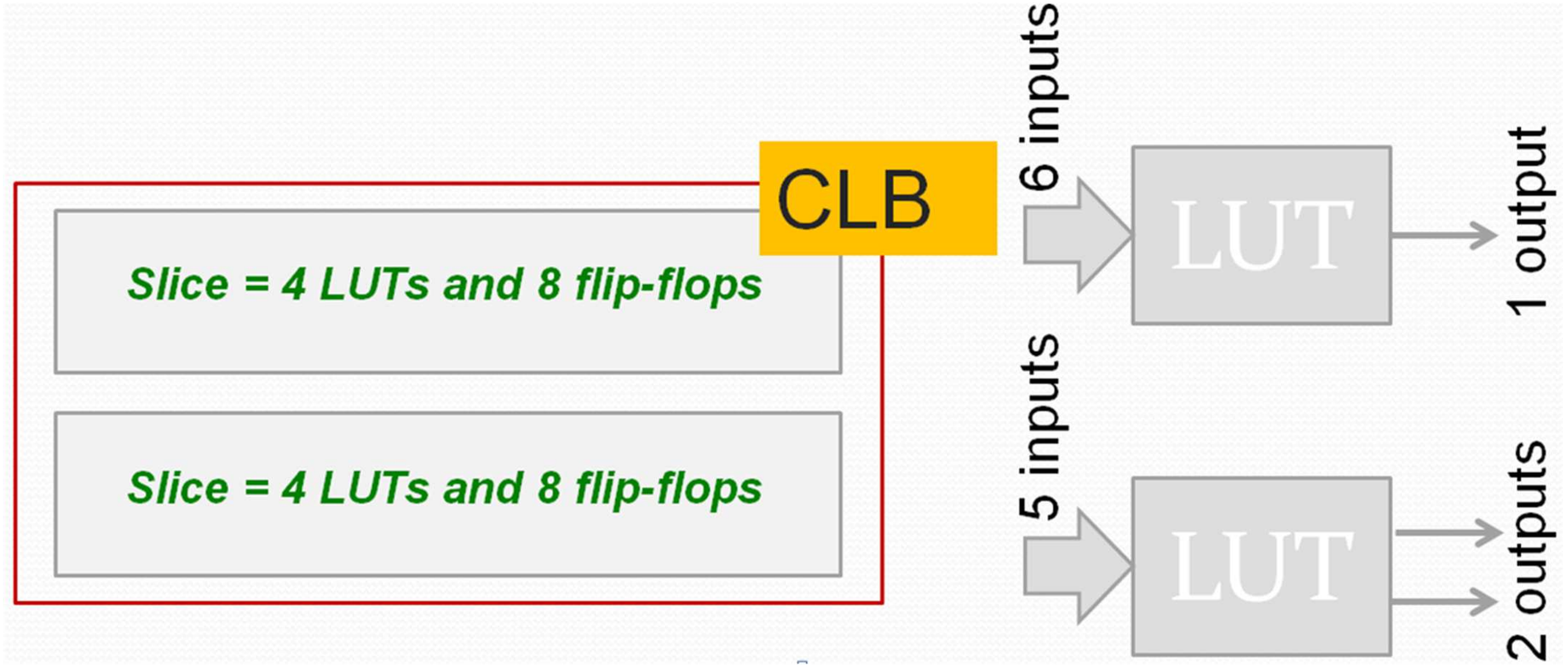
Simulação

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity IGCD_sim is
end IGCD_sim;
architecture behavior of IGCD_sim is
   signal rst , clk                          : std_logic := '0';
   signal sw                                 : std_logic_vector(15 downto 0);
   signal led                                : std_logic_vector(7 downto 0);
   constant clk_period                       : time := 50 ns;
 begin
 unit_to_test: entity work.FSM_new
                port map (clk, rst, sw, led);
clk_generator: process
begin
   clk <= '0';                               wait for clk_period/2;
   clk <= '1';                               wait for clk_period/2;
end process clk_generator;
stim_proc: process
begin
   rst <= '1';                                                         wait for 130 ns;
   rst <= '0'; sw <= (15 downto 10 => '1', others => '0');             wait for 1000 ns;
   rst <= '1';                                                         wait for 130 ns;
   rst <= '0'; sw <= "0001101100111111";                               wait for 1000 ns;
   rst <= '1';                                                         wait for 130 ns;
   rst <= '0'; sw <= "0000111100001010";                               wait for 1000 ns;
   rst <= '1';                                                         wait for 130 ns;
   rst <= '0'; sw <= "1111111111111111";                               wait for 1000 ns;
end process;
end behavior;
```

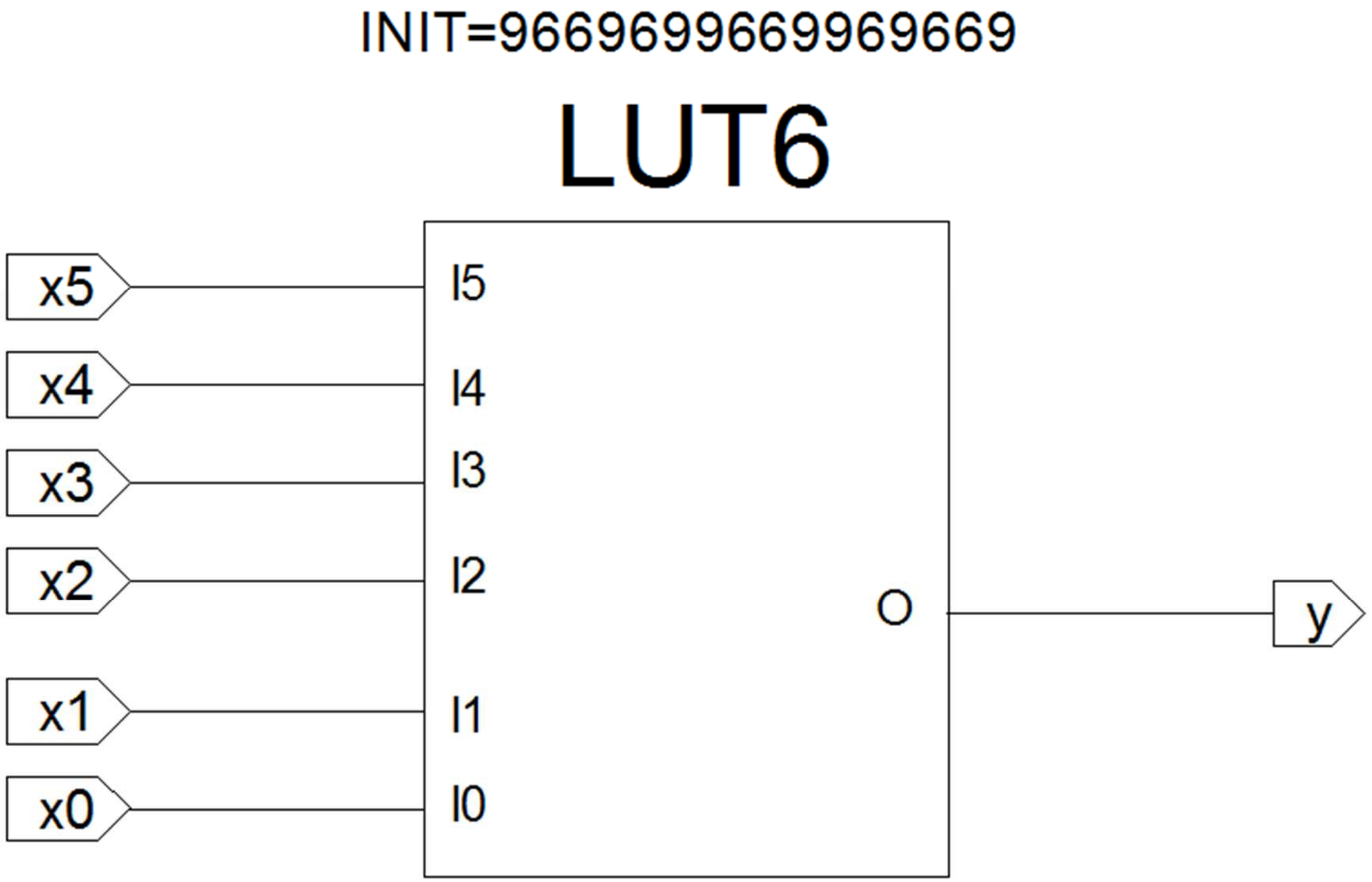


Name	Value
rst	0
clk	0
sw[15:0]	1b3f
led[7:0]	9
clk_period	50000 ps

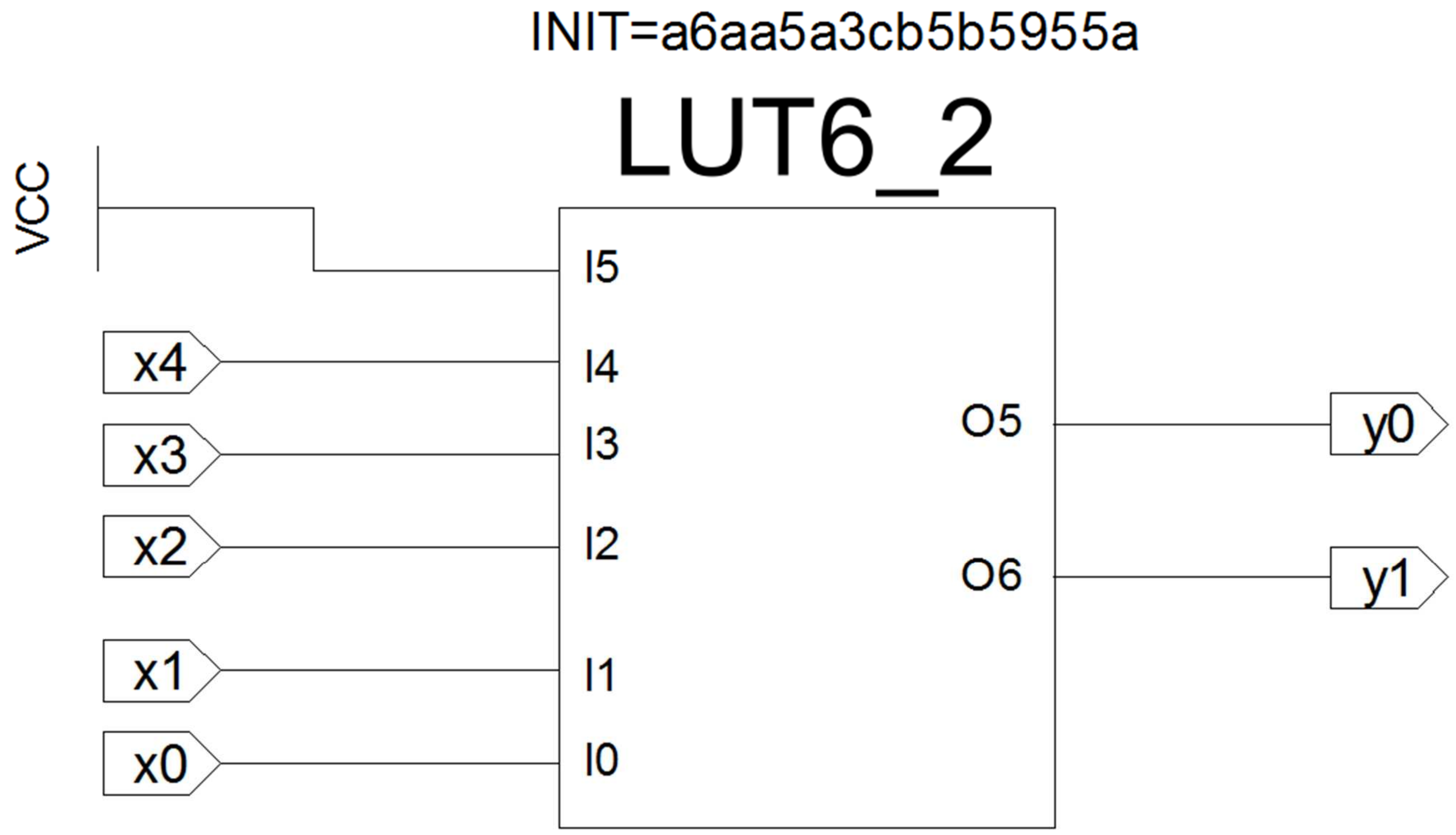
FPGA Artix-7 (família 7 de FPGAs de Xilinx)


CLB
Slice = 4 LUTs and 8 flip-flops
Slice = 4 LUTs and 8 flip-flops
6 inputs
LUT
1 output
5 inputs
LUT
2 outputs

Look-up tables - LUTs, *slices*, blocos lógicos programáveis



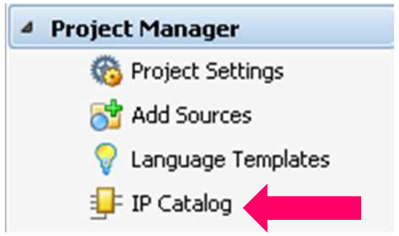
Look-up tables - LUTs, *slices*, blocos lógicos programáveis



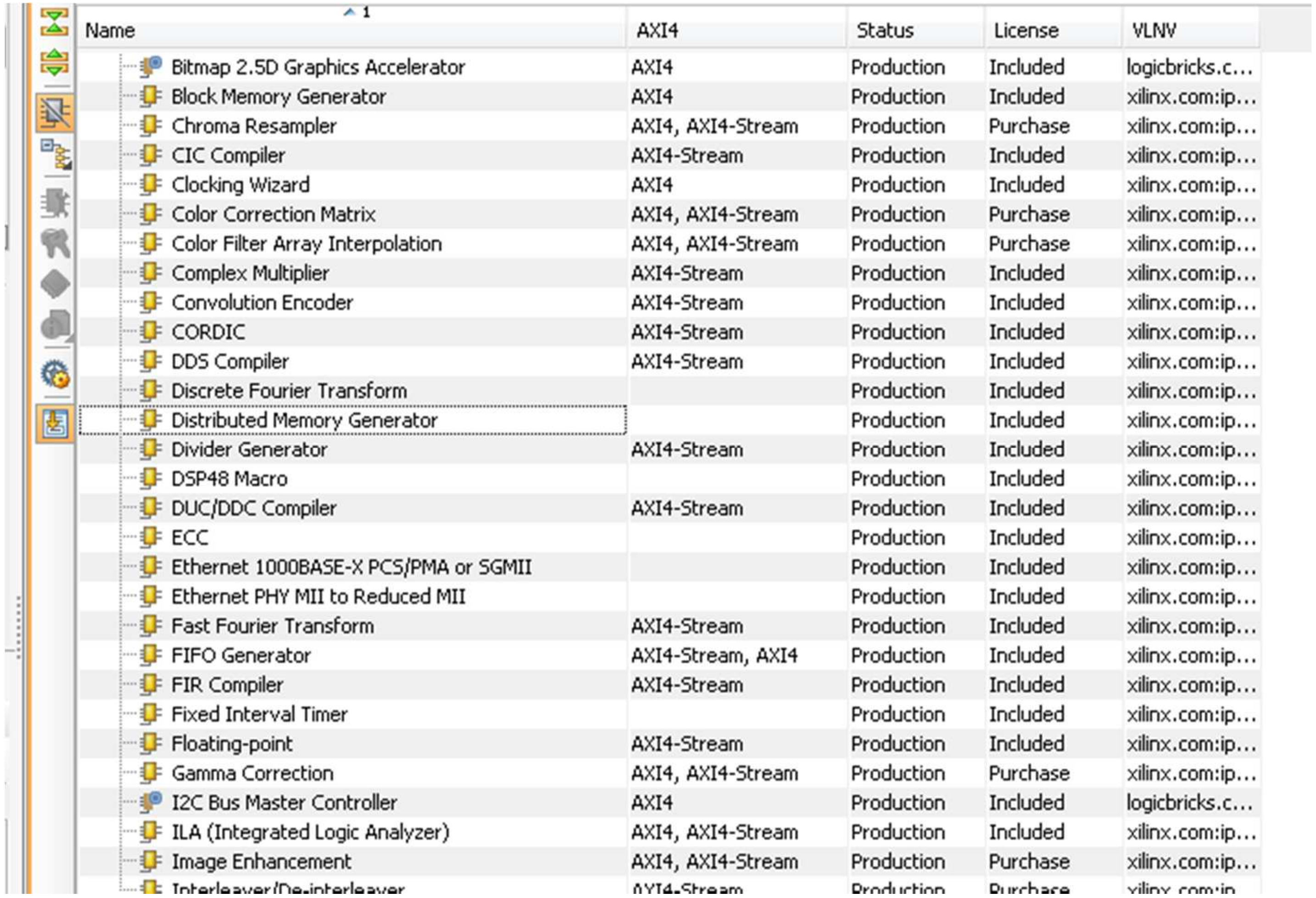
Podemos utilizar constantes para configurar (preencher) as LUTs

Look-up tables - LUTs, slices, blocos lógicos programáveis

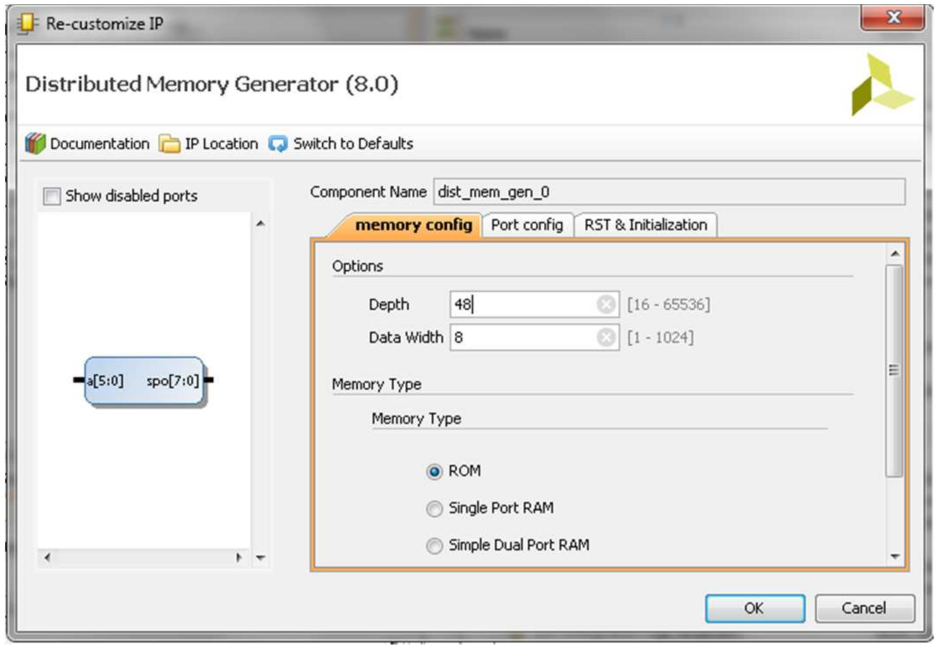
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LUT_memory is
port (      clk                         : in  std_logic;
            led                         : out std_logic_vector(7 downto 0)       );
end LUT_memory;

architecture Behavioral of LUT_memory is
   type for_LUT is array (0 to 8) of std_logic_vector(7 downto 0);
   constant write_LUT : for_LUT := (x"00", x"18", x"3c", x"7e", x"ff", x"7e", x"3c", x"18", x"00");
   signal divided_clk  : std_logic;                                  valores hexadecimais
   signal addr         : integer range 0 to 8;
begin
            addr <= addr+1 when rising_edge(divided_clk);
            led <= write_LUT(addr);

div:        entity work.clock_divider
            port  map (   clk, '0',divided_clk);

end Behavioral;
```



Project Manager
- Project Settings
- Add Sources
- Language Templates
- IP Catalog



Name	AXI4	Status	License	VLNV
Bitmap 2.5D Graphics Accelerator	AXI4	Production	Included	logicbricks.c...
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip...
Chroma Resampler	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip...
CIC Compiler	AXI4-Stream	Production	Included	xilinx.com:ip...
Clocking Wizard	AXI4	Production	Included	xilinx.com:ip...
Color Correction Matrix	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip...
Color Filter Array Interpolation	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip...
Complex Multiplier	AXI4-Stream	Production	Included	xilinx.com:ip...
Convolution Encoder	AXI4-Stream	Production	Included	xilinx.com:ip...
CORDIC	AXI4-Stream	Production	Included	xilinx.com:ip...
DDS Compiler	AXI4-Stream	Production	Included	xilinx.com:ip...
Discrete Fourier Transform		Production	Included	xilinx.com:ip...
Distributed Memory Generator		Production	Included	xilinx.com:ip...
Divider Generator	AXI4-Stream	Production	Included	xilinx.com:ip...
DSP48 Macro		Production	Included	xilinx.com:ip...
DUC/DDC Compiler	AXI4-Stream	Production	Included	xilinx.com:ip...
ECC		Production	Included	xilinx.com:ip...
Ethernet 1000BASE-X PCS/PMA or SGMII		Production	Included	xilinx.com:ip...
Ethernet PHY MII to Reduced MII		Production	Included	xilinx.com:ip...
Fast Fourier Transform	AXI4-Stream	Production	Included	xilinx.com:ip...
FIFO Generator	AXI4-Stream, AXI4	Production	Included	xilinx.com:ip...
FIR Compiler	AXI4-Stream	Production	Included	xilinx.com:ip...
Fixed Interval Timer		Production	Included	xilinx.com:ip...
Floating-point	AXI4-Stream	Production	Included	xilinx.com:ip...
Gamma Correction	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip...
I2C Bus Master Controller	AXI4	Production	Included	logicbricks.c...
ILA (Integrated Logic Analyzer)	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip...
Image Enhancement	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip...
Interleaver/De-interleaver	AXI4-Stream	Production	Purchase	xilinx.com:ip


Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name: dist_mem_gen_0

memory config | Port config | RST & Initialization

Options

Depth: 48 [16 - 65536]

Data Width: 8 [1 - 1024]

Memory Type

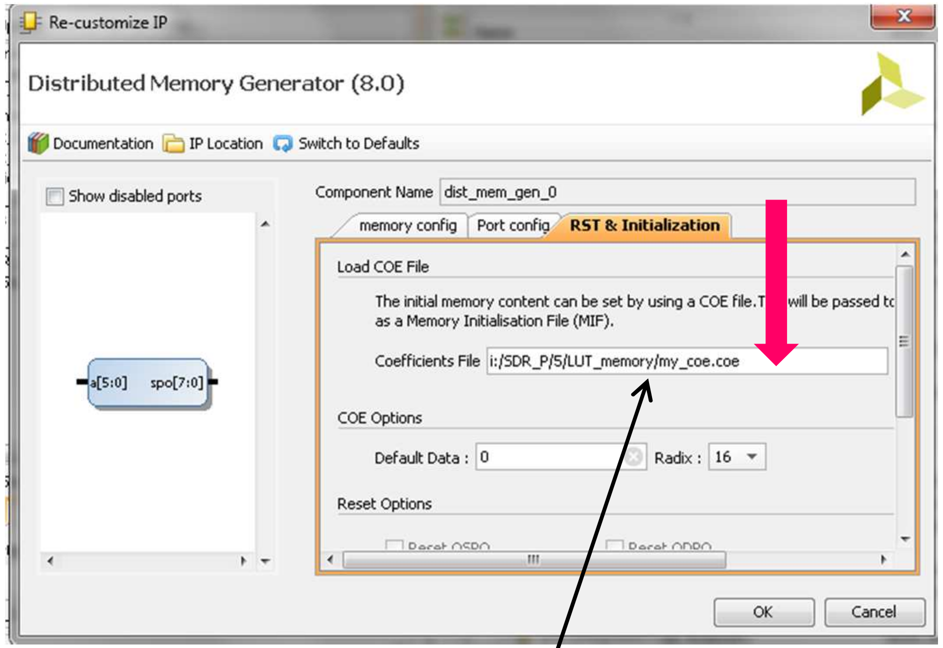
Memory Type

[✓ ROM]

Single Port RAM

Simple Dual Port RAM

OK Cancel


Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name: dist_mem_gen_0

memory config | Port config | RST & Initialization

Load COE File

The initial memory content can be set by using a COE file.T will be passed tc as a Memory Initialisation File (MIF).

Coefficients File: i:/SDR_P/5/LUT_memory/my_coe.coe

COE Options

Default Data : 0 Radix : 16

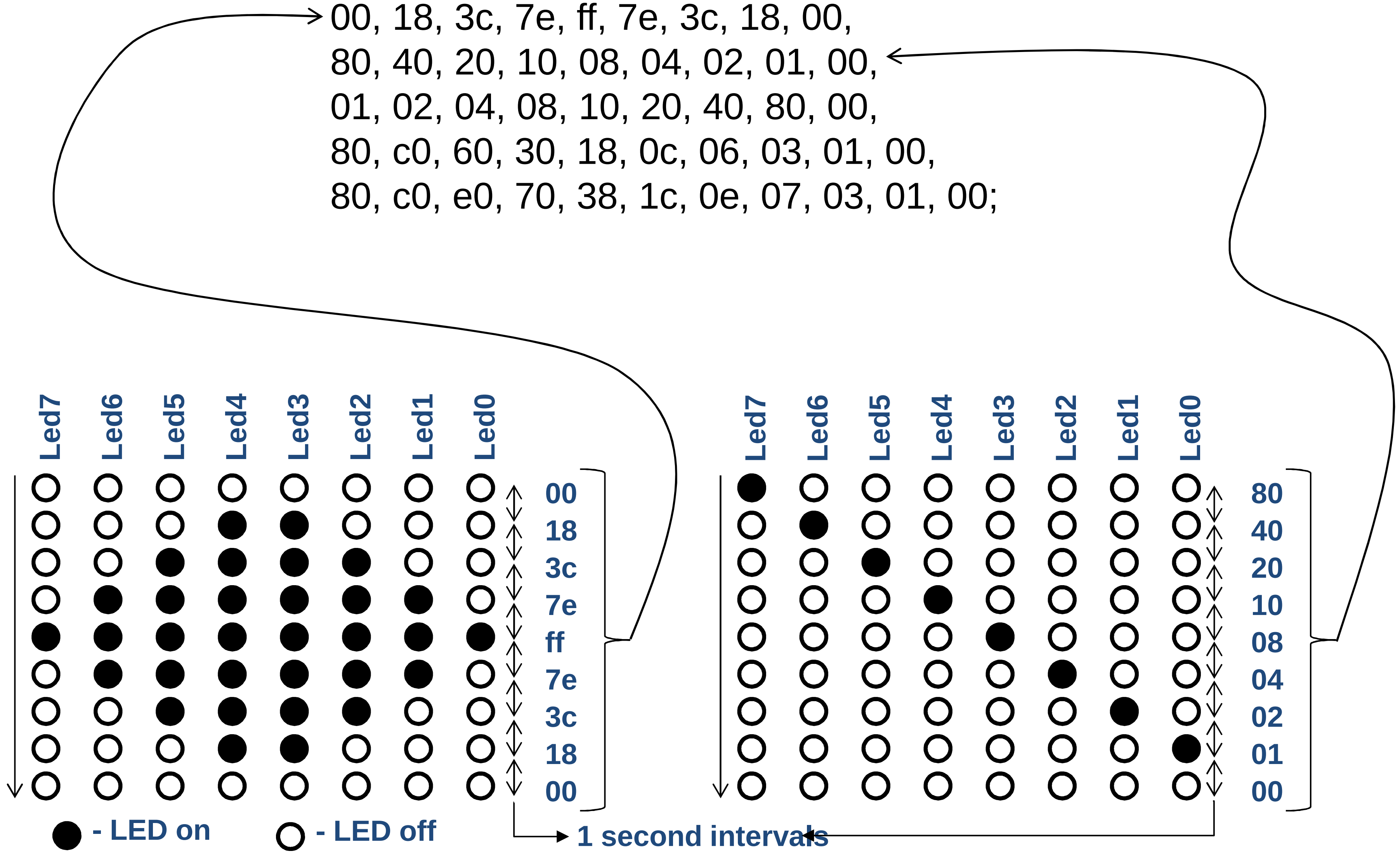
Reset Options

Reset OSPO Reset QDPO

OK Cancel

```
memory_initialization_radix = 16;
memory_initialization_vector =
00, 18, 3c, 7e, ff, 7e, 3c, 18, 00,
80, 40, 20, 10, 08, 04, 02, 01, 00,
01, 02, 04, 08, 10, 20, 40, 80, 00,
80, c0, 60, 30, 18, 0c, 06, 03, 01, 00,
80, c0, e0, 70, 38, 1c, 0e, 07, 03, 01, 00;
```

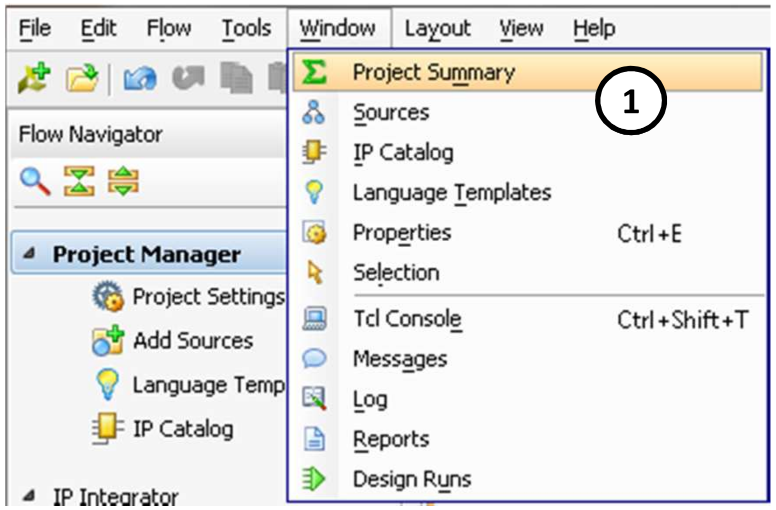
```
memory_initialization_radix = 16;
memory_initialization_vector =
00, 18, 3c, 7e, ff, 7e, 3c, 18, 00,
80, 40, 20, 10, 08, 04, 02, 01, 00,
01, 02, 04, 08, 10, 20, 40, 80, 00,
80, c0, 60, 30, 18, 0c, 06, 03, 01, 00,
80, c0, e0, 70, 38, 1c, 0e, 07, 03, 01, 00;
```

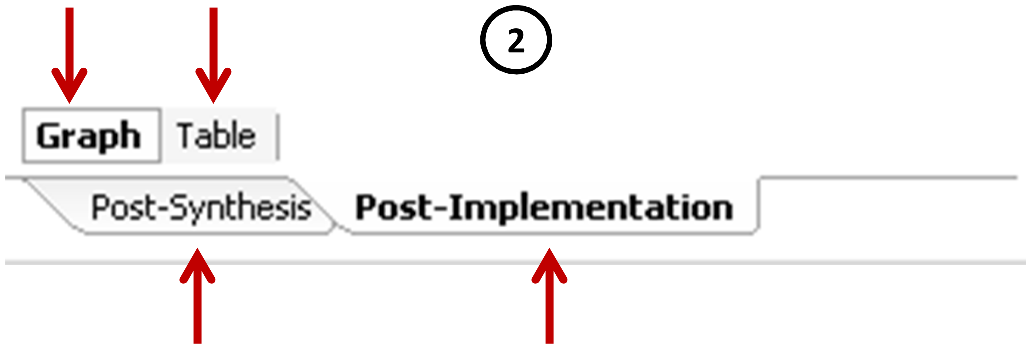


Look-up tables - LUTs, slices, blocos lógicos programáveis

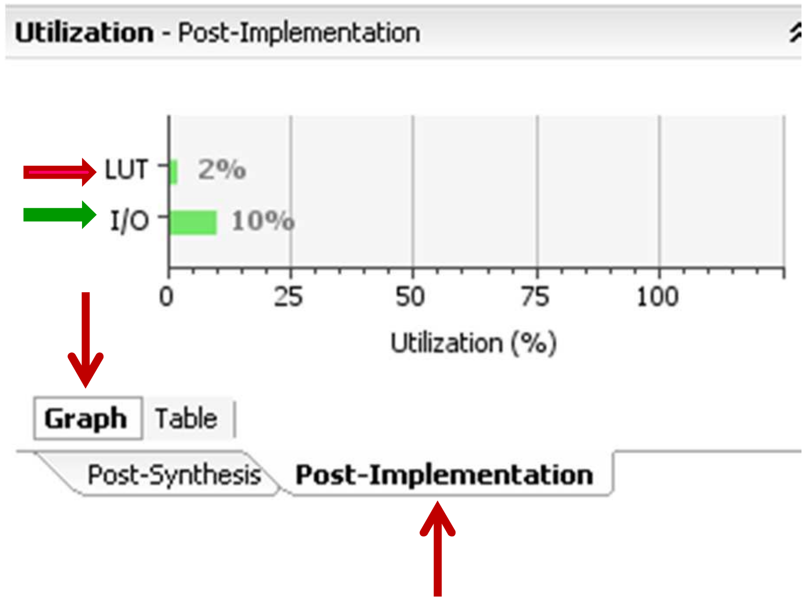
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity LUT_memory is
port (      clk                          : in  std_logic;
            led                          : out std_logic_vector(7 downto 0)       );
end LUT_memory;
architecture Behavioral of LUT_memory is
            signal divided_clk  : std_logic;
            signal address       : std_logic_vector(5 downto 0) := (others=>'0');
component dist_mem_gen_0
  port (
            a            : in std_logic_vector(5 downto 0);
            spo          : out std_logic_vector(7 downto 0)       );
end component;
begin

address <= address+1 when rising_edge(divided_clk);

dist_ROM :  dist_mem_gen_0
                        port map (address, led );

div: entity work.clock_divider
         port  map (   clk, '0',divided_clk);

end Behavioral;
```

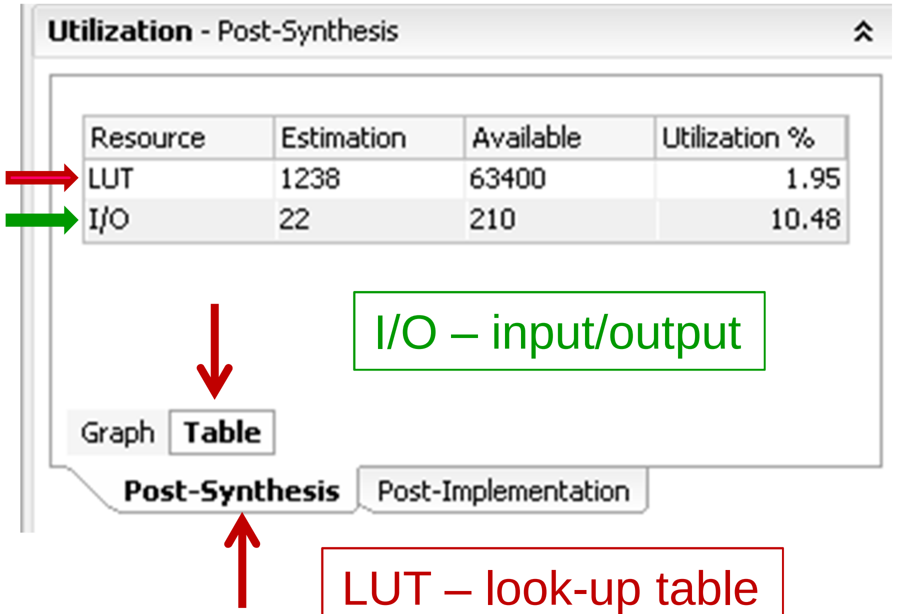
Como encontrar recursos da FPGA utilizados no seu projeto





1 CLB = 2 slices.
1 slice = 4 LUTs e 8 flip-flops



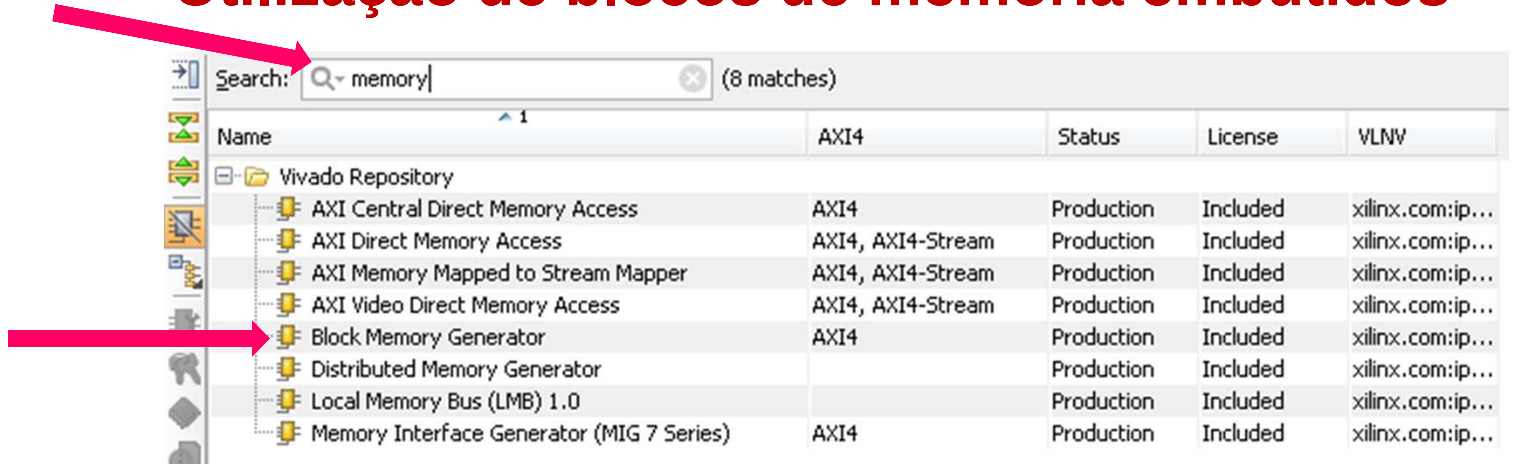

Utilization - Post-Synthesis

Resource	Estimation	Available	Utilization %
LUT	1238	63400	1.95
I/O	22	210	10.48

I/O – input/output

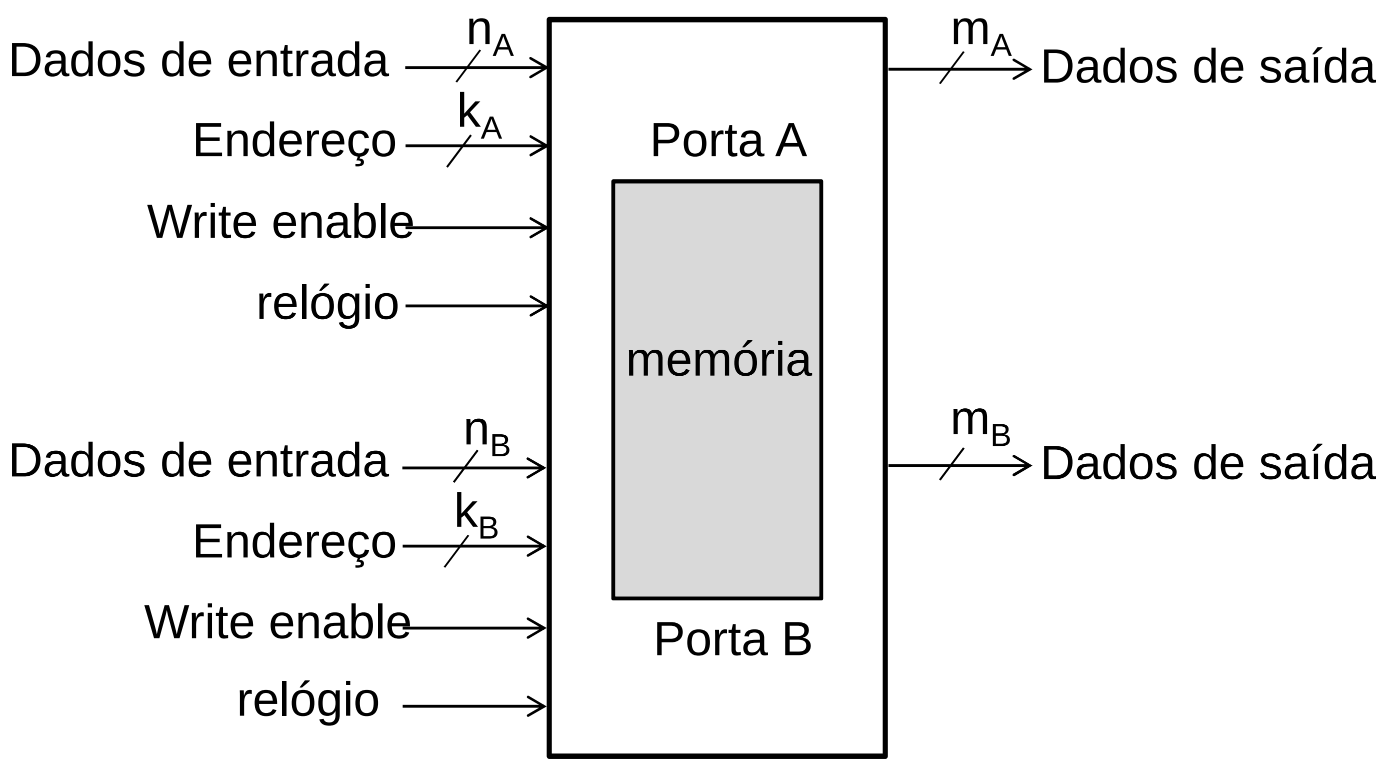
LUT – look-up table

Utilização de blocos de memória embutidos

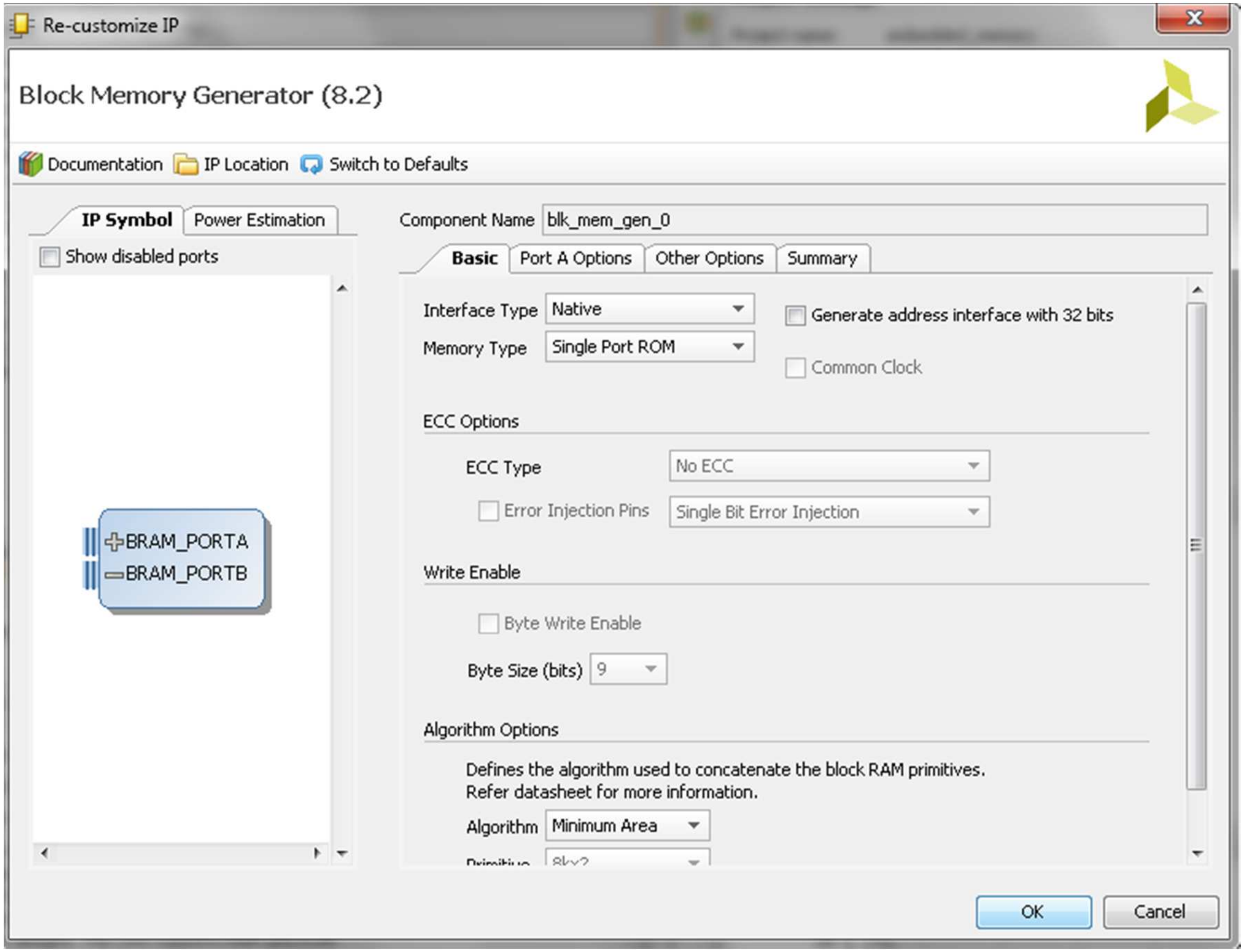


Search: memory (8 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
AXI Central Direct Memory Access	AXI4	Production	Included	xilinx.com:ip...
AXI Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip...
AXI Memory Mapped to Stream Mapper	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip...
AXI Video Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip...
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip...
Distributed Memory Generator		Production	Included	xilinx.com:ip...
Local Memory Bus (LMB) 1.0		Production	Included	xilinx.com:ip...
Memory Interface Generator (MIG 7 Series)	AXI4	Production	Included	xilinx.com:ip...



Utilização de blocos de memória embutidos



Re-customize IP

Block Memory Generator (8.2)

Documentation | IP Location | Switch to Defaults

IP Symbol | Power Estimation

Show disabled ports

Component Name: blk_mem_gen_0

Basic | Port A Options | Other Options | Summary

Interface Type: Native

Memory Type: Single Port ROM

Generate address interface with 32 bits

Common Clock

ECC Options

ECC Type: No ECC

Error Injection Pins: Single Bit Error Injection

Write Enable

Byte Write Enable

Byte Size (bits): 9

Algorithm Options

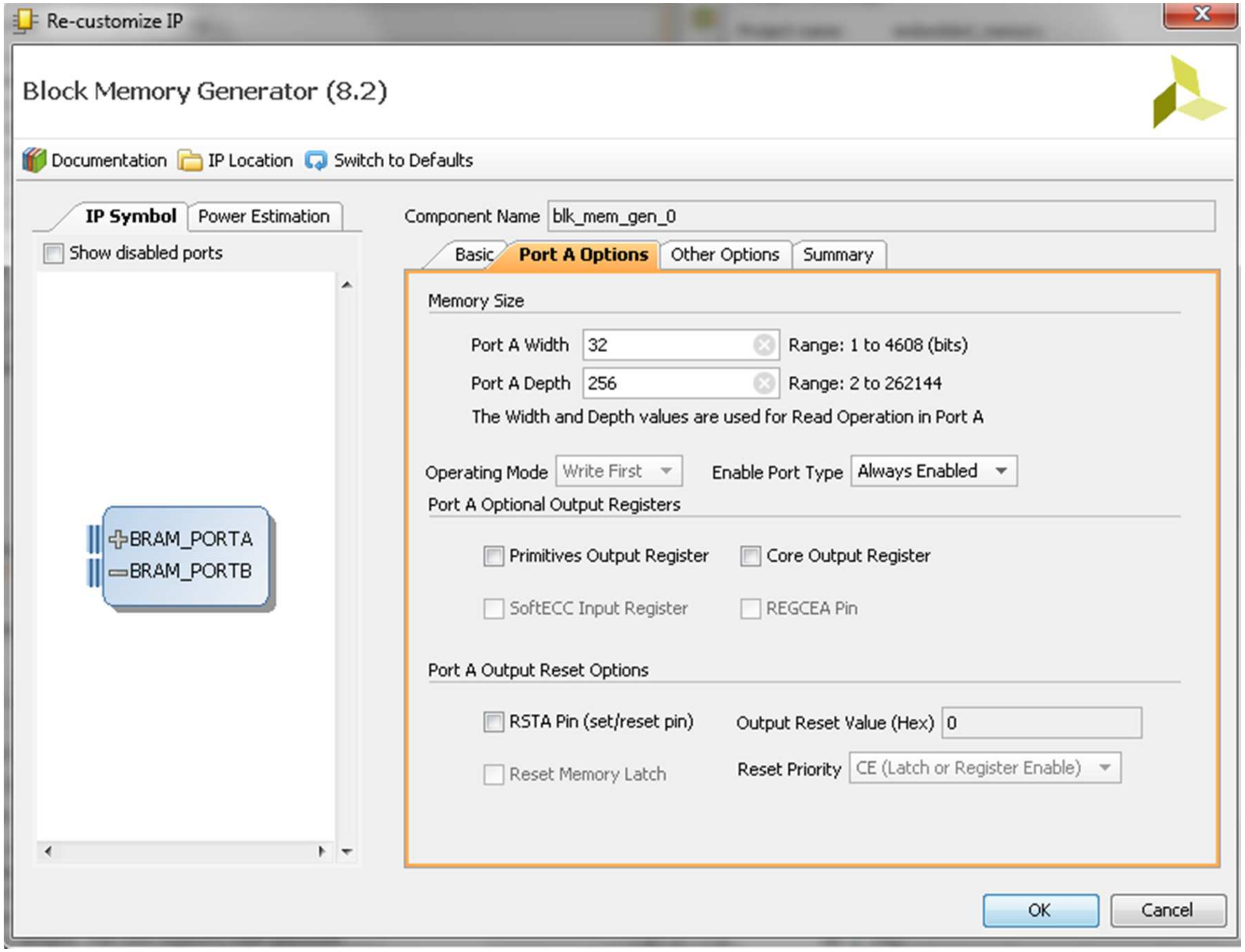
Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.

Algorithm: Minimum Area

Primitive: 8kx2

OK | Cancel

Utilização de blocos de memória embutidos



Re-customize IP

Block Memory Generator (8.2)

Documentation | IP Location | Switch to Defaults

IP Symbol | Power Estimation

Show disabled ports

Component Name: blk_mem_gen_0

Basic | **Port A Options** | Other Options | Summary

Memory Size

Port A Width: 32 — Range: 1 to 4608 (bits)

Port A Depth: 256 — Range: 2 to 262144

The Width and Depth values are used for Read Operation in Port A

Operating Mode: Write First — Enable Port Type: Always Enabled

Port A Optional Output Registers

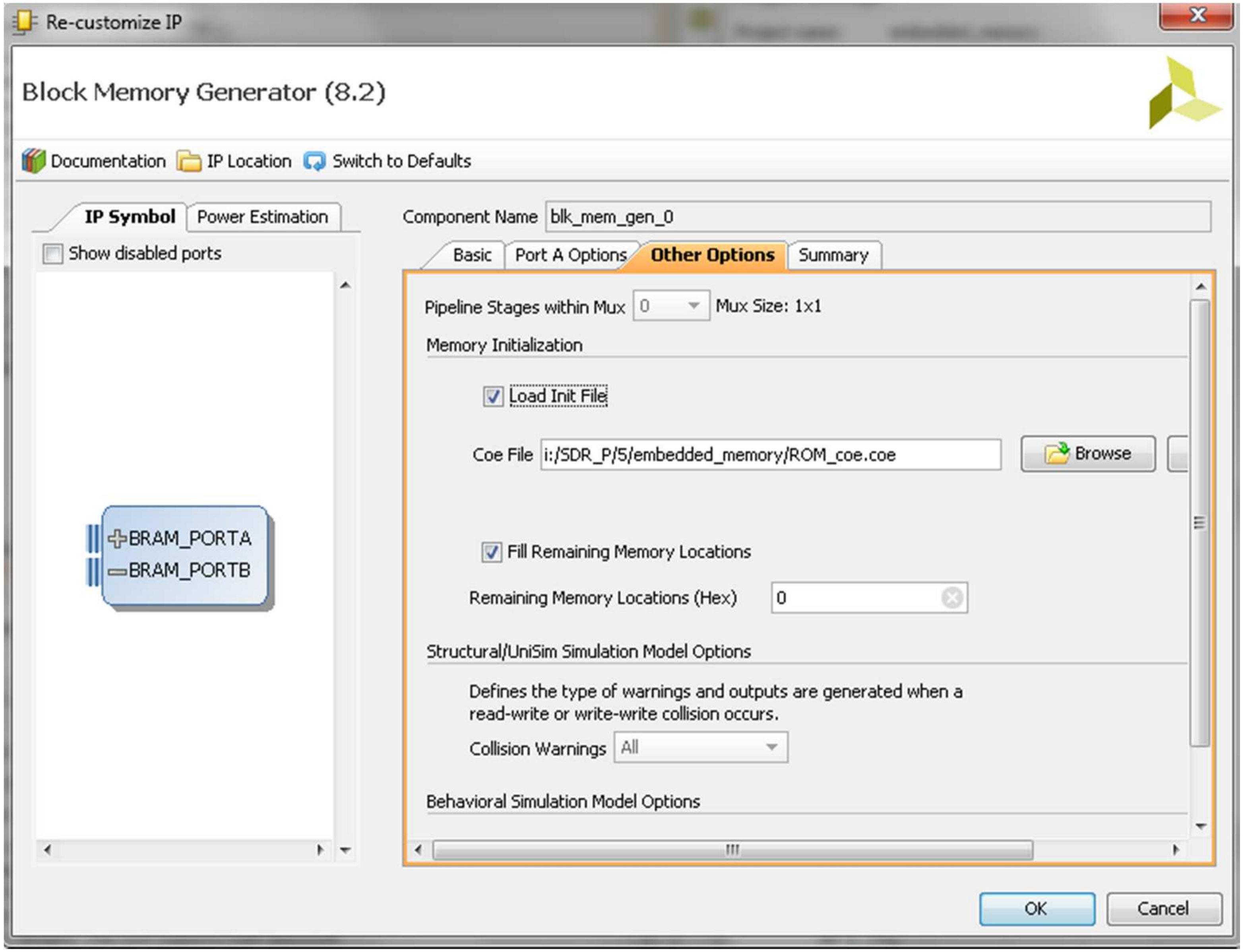
- [ ] Primitives Output Register
- [ ] Core Output Register
- [ ] SoftECC Input Register
- [ ] REGCEA Pin

Port A Output Reset Options

- [ ] RSTA Pin (set/reset pin) — Output Reset Value (Hex): 0
- [ ] Reset Memory Latch — Reset Priority: CE (Latch or Register Enable)

OK | Cancel

Utilização de blocos de memória embutidos



Re-customize IP

Block Memory Generator (8.2)

Documentation | IP Location | Switch to Defaults

Component Name: blk_mem_gen_0

Basic | Port A Options | **Other Options** | Summary

Pipeline Stages within Mux: 0 | Mux Size: 1x1

Memory Initialization

[✓] Load Init File

Coe File: i:/SDR_P/5/embedded_memory/ROM_coe.coe | Browse

[✓] Fill Remaining Memory Locations

Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

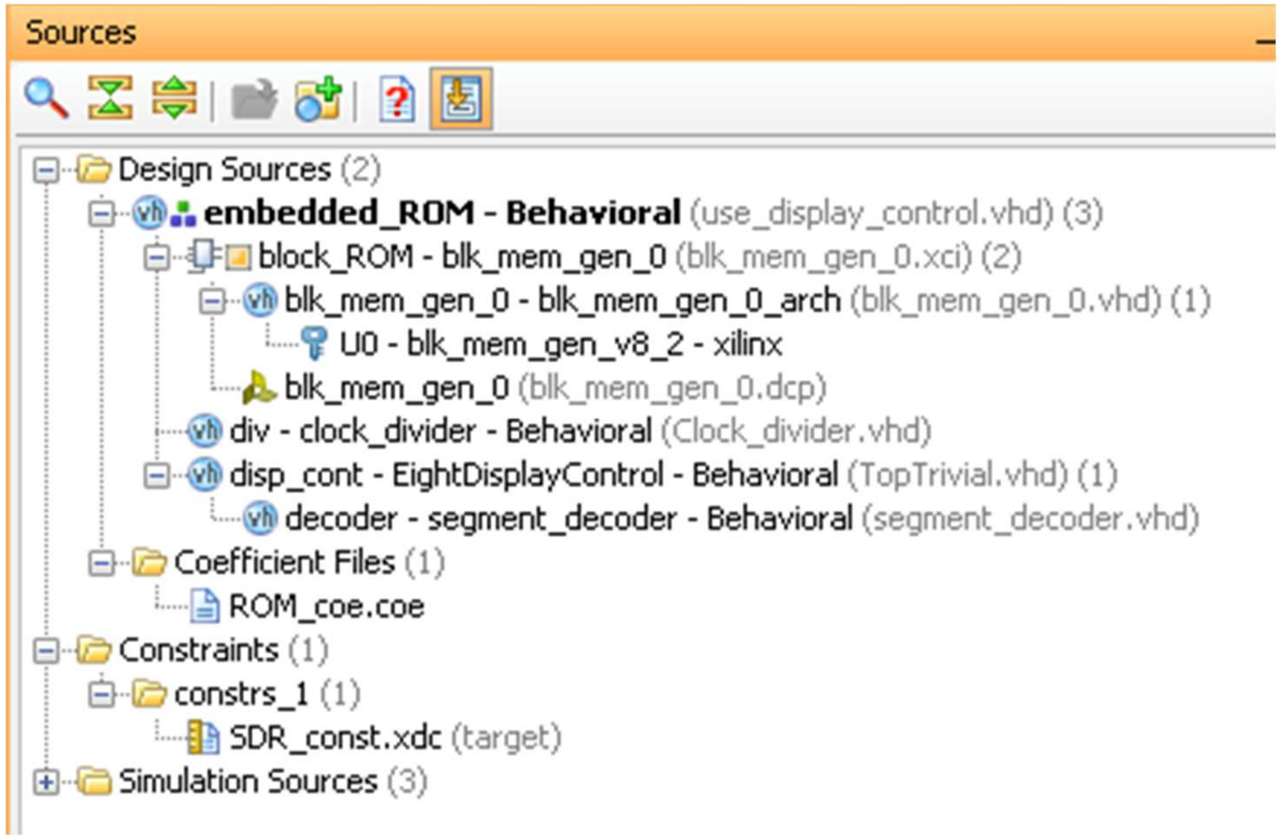
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings: All

Behavioral Simulation Model Options

OK | Cancel

Utilização de blocos de memória embutidos



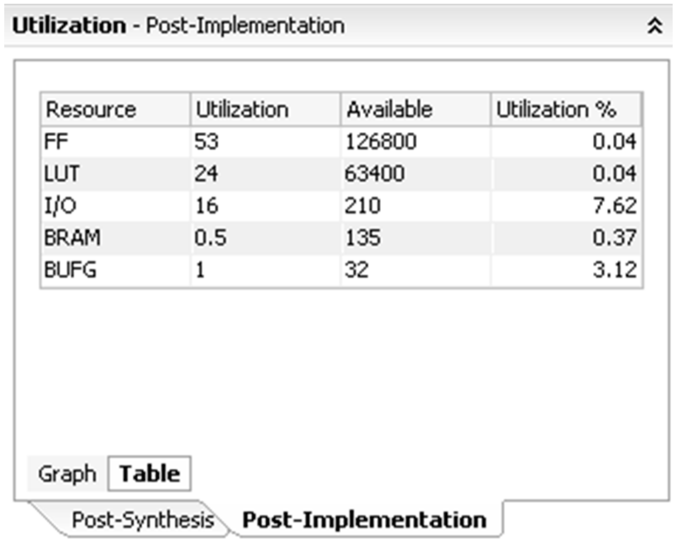
Sources

```
Design Sources (2)
  embedded_ROM - Behavioral (use_display_control.vhd) (3)
    block_ROM - blk_mem_gen_0 (blk_mem_gen_0.xci) (2)
      blk_mem_gen_0 - blk_mem_gen_0_arch (blk_mem_gen_0.vhd) (1)
        U0 - blk_mem_gen_v8_2 - xilinx
      blk_mem_gen_0 (blk_mem_gen_0.dcp)
    div - clock_divider - Behavioral (Clock_divider.vhd)
    disp_cont - EightDisplayControl - Behavioral (TopTrivial.vhd) (1)
      decoder - segment_decoder - Behavioral (segment_decoder.vhd)
  Coefficient Files (1)
    ROM_coe.coe
Constraints (1)
  constrs_1 (1)
    SDR_const.xdc (target)
Simulation Sources (3)
```

Utilização de blocos de memória embutidos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity embedded_ROM is
port (        clk          : in  std_logic;
              seg          : out STD_LOGIC_VECTOR (6 downto 0);
              sel_disp     : out STD_LOGIC_VECTOR (7 downto 0)     );
end embedded_ROM;
architecture Behavioral of embedded_ROM is
              signal divided_clk  : std_logic;
              signal address      : std_logic_vector(7 downto 0) := (others=>'0');
              signal data32bit    : std_logic_vector(31 downto 0);
component blk_mem_gen_0 is
  port (
    clka      : in std_logic;
    addra     : in std_logic_vector(7 downto 0);
    douta     : out std_logic_vector(31 downto 0)          );
end component;
begin
              address <= address+1 when rising_edge(divided_clk);
block_ROM :  blk_mem_gen_0
                port map (divided_clk, address, data32bit );
div: entity work.clock_divider
          port  map (   clk, '0',divided_clk);
disp_cont: entity work.EightDisplayControl
   port map ( clk=>clk, leftL=>data32bit(31 downto 28), near_leftL=>data32bit(27 downto 24),
                         near_rightL=>data32bit(23 downto 20), rightL=>data32bit(19 downto 16),
                         leftR=>data32bit(15 downto 12), near_leftR=>data32bit(11 downto 8),
                         near_rightR=>data32bit(7 downto 4), rightR=>data32bit(3 downto 0),
                select_display=>sel_disp,segments=>seg);

end Behavioral;
```

Memória embutida foi utilizada


Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
FF	53	126800	0.04
LUT	24	63400	0.04
I/O	16	210	7.62
BRAM	0.5	135	0.37
BUFG	1	32	3.12

Utilização de blocos de memória embutidos

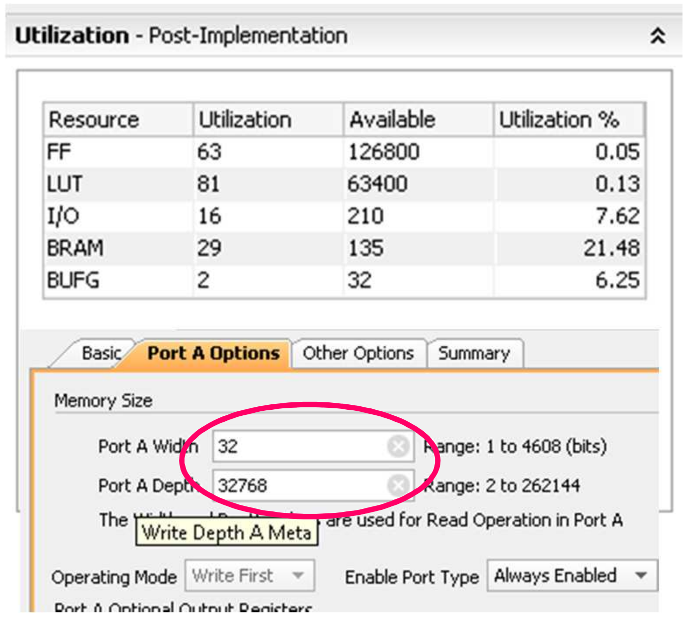
Geração de ficheiro *coe* a partir de JAVA

```
import java.util.*;   import java.io.*;
public class Random_to_file {
  static Random rand = new Random();
  static final int nBlocks = 256;
public static void main (String args[])   throws IOException     {
    int a[] = new int[nBlocks];
    for(int i = 0; i < a.length;  i++)
        a[i] = rand.nextInt(0x7FFFFFFF);
    File fout = new File("coe_from_java1.coe");
    PrintWriter pw = new PrintWriter(fout);
    pw.println("memory_initialization_radix = 16;");
    pw.println("memory_initialization_vector = ");
    for(int k=0; k<nBlocks; k++)
        pw.printf("%08x, ",a[k]);
    pw.println(";");
    pw.close();
                                                        }
                                  }
```

Utilização de blocos de memória embutidos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity embedded_ROM is
port (        clk          : in  std_logic;
              seg          : out STD_LOGIC_VECTOR (6 downto 0);
              sel_disp     : out STD_LOGIC_VECTOR (7 downto 0)     );
end embedded_ROM;
architecture Behavioral of embedded_ROM is
              signal divided_clk  : std_logic;
              signal address      : std_logic_vector(14 downto 0) := (others=>'0');
              signal data32bit    : std_logic_vector(31 downto 0);
component blk_mem_gen_0 is
  port (
    clka      : in std_logic;
    addra     : in std_logic_vector(14 downto 0);
    douta     : out std_logic_vector(31 downto 0)                );
end component;
begin
              address <= address+1 when rising_edge(divided_clk);
block_ROM :  blk_mem_gen_0
                port map (divided_clk, address, data32bit );
div: entity work.clock_divider
           port  map (   clk, '0',divided_clk);
disp_cont: entity work.EightDisplayControl
    port map ( clk=>clk, leftL=>data32bit(31 downto 28), near_leftL=>data32bit(27 downto 24),
                          near_rightL=>data32bit(23 downto 20), rightL=>data32bit(19 downto 16),
                          leftR=>data32bit(15 downto 12), near_leftR=>data32bit(11 downto 8),
                          near_rightR=>data32bit(7 downto 4), rightR=>data32bit(3 downto 0),
                select_display=>sel_disp,segments=>seg);

end Behavioral;
```

Memória embutida foi utilizada



Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
FF	63	126800	0.05
LUT	81	63400	0.13
I/O	16	210	7.62
BRAM	29	135	21.48
BUFG	2	32	6.25

Memory Size

Port A Width: 32 — Range: 1 to 4608 (bits)

Port A Depth: 32768 — Range: 2 to 262144

The Width and Depth values are used for Read Operation in Port A

Write Depth A Meta

Operating Mode: Write First — Enable Port Type: Always Enabled

Port A Optional Output Registers