Computação Reconfigurável

Aula 2

Valeri Skliarov, Prof. Catedrático

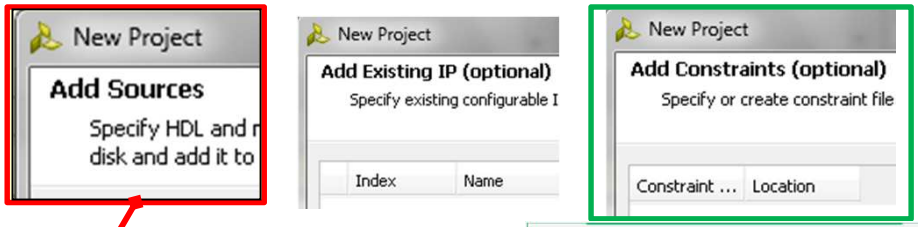
Email: skl@ua.pt
URL: http://sweet.ua.pt/skl/

Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

http://elearning.ua.pt/

Revisão da aula anterior

Primeiros passos:




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

entity TopTrivial is
    port (      sw      : in  std_logic_vector (15 downto 0);
                led     : out  std_logic_vector (15 downto 0));
end TopTrivial;

architecture Behavioral of TopTrivial is
begin
                led <= sw;
end Behavioral;
```

```
# Switches
#Bank = 34, Pin name = IO_L21P_T3_DQS_34, Sch name = SW0
set_property PACKAGE_PIN U9 [get_ports {sw[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
#Bank = 34, Pin name = IO_25_34,Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports {sw[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
#Bank = 34, Pin name = IO_L23P_T3_34,Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {sw[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#Bank = 34, Pin name = IO_L19P_T3_34,Sch name = SW3
set_property PACKAGE_PIN R6 [get_ports {sw[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#Bank = 34, Pin name = IO_L19N_T3_VREF_34,Sch name = SW4
set_property PACKAGE_PIN R5 [get_ports {sw[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
#Bank = 34, Pin name = IO_L20P_T3_34,Sch name = SW5
set_property PACKAGE_PIN V7 [get_ports {sw[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
#Bank = 34, Pin name = IO_L20N_T3_34,Sch name = SW6
set_property PACKAGE_PIN V6 [get_ports {sw[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#Bank = 34, Pin name = IO_L10P_T1_34,Sch name = SW7
set_property PACKAGE_PIN V5 [get_ports {sw[7]}]
```

Aula 2

- **Síntese e simulação**
- **Tipos de dados, objetos e operadores**
- **Instruções de controlo – decisão: *if*, *when*, *with*, *case***
- **Processos combinatórios e sequenciais**
- **VHDL comportamental e estrutural**
- **Exemplos:**
  - **Operações aritméticas**
  - **Processos**
  - **Conversão de tipos**

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

VHDL (três partes principais)

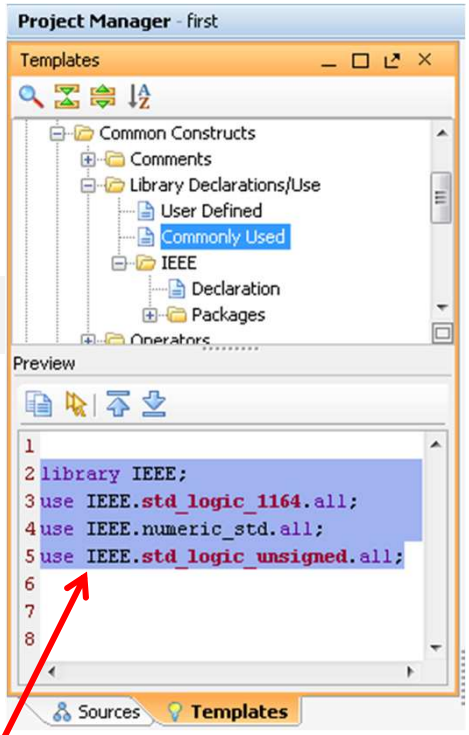
Três partes principais:

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

entity TopTrivial is
   port (      sw     : in  std_logic_vector (15 downto 0);
               led    : out  std_logic_vector (15 downto 0));
end TopTrivial;

architecture Behavioral of TopTrivial is
begin
               led <= sw;
end Behavioral;
```

Incluir bibliotecas necessárias

Declarar entidade

Descrever arquitetura



Bibliotecas mais comuns:

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

ou

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

VHDL (três tipos de descrição principais)

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

entity TopTrivialNew is
        port (     clk      : in std_logic;
                   sw       : in  std_logic_vector (15 downto 0);
                   led      : out  std_logic_vector (15 downto 0));
end TopTrivialNew;

architecture my of TopTrivialNew is

            signal    divided_clk          : std_logic;

begin

led <= sw when divided_clk = '1' else (others => '0');

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

VHDL:

1. **Comportamental.**
2. **Estrutural.**
3. **Misto**

Declaração de sinais

Frequência do relógio da placa *100 MHz*

VHDL (sumário)

Declaração de bibliotecas
library . . .
use . . .

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

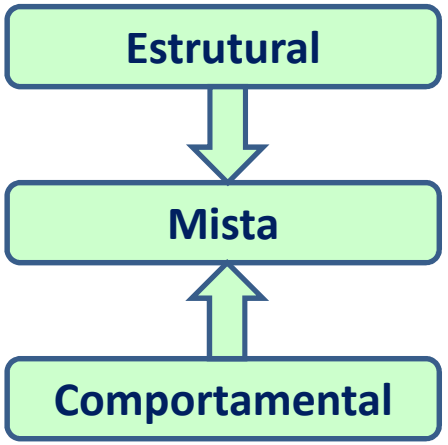
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

Descrição de entidade (interface)
entity <nome de entidade> **is**
Declaração de parâmetros genéricos – **generic**
Declaração de portas – **port**
end <nome de entidade>

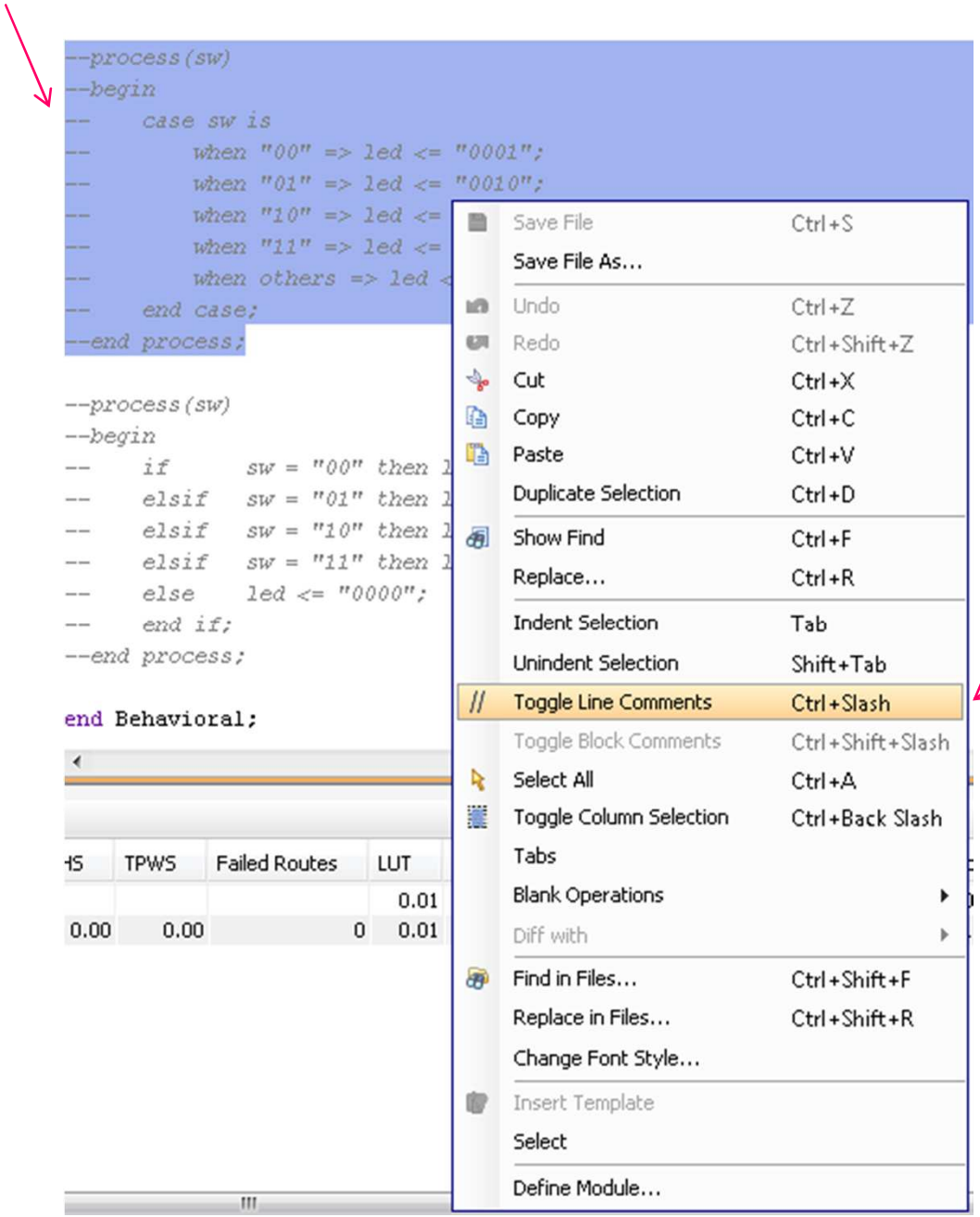
parte declarativa de entidade

```
entity Counter is
    generic (N : integer := 3);
    port (
          reset : in std_logic;
          clk   : in std_logic;
          My_count: out std_logic_vector(N downto 0)
          );
end Counter;
```

Descrição de arquitetura (funcionalidade)
arquitecture <nome de arquitetura > **of** <nome de entidade> **is**
Declaração de sinais– **signal**
Declaração de constantes – **constant**
Declaração de tipos– **type**
Declaração de componentes– **component**
Declaração de funções– **function**
Declaração de procedimentos– **procedure**
Declaração de variáveis partilhadas– **shared variable**
begin
Corpo de arquitetura
end <nome de arquitetura >



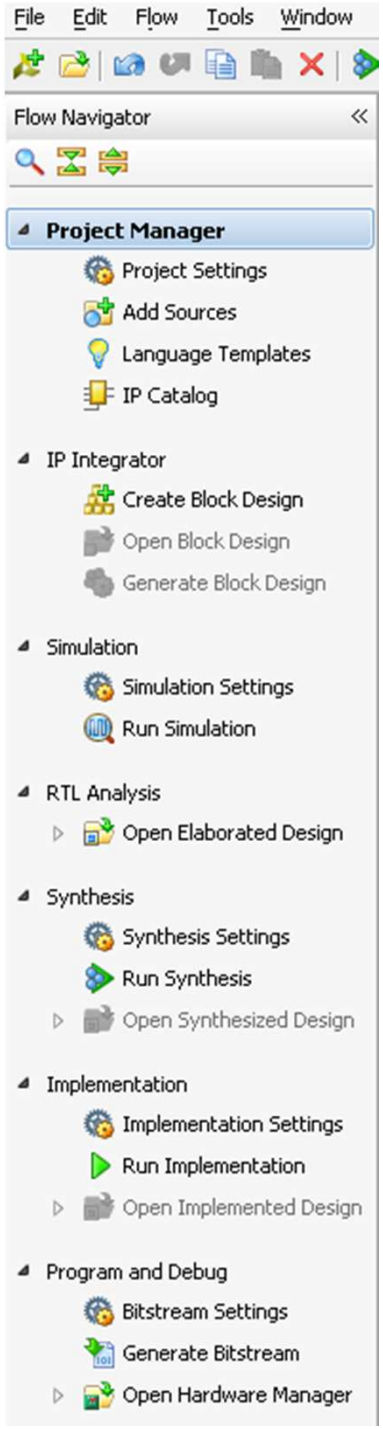
parte declarativa de arquitetura



-- comentários até final da linha

Inserir/remover comentários para um conjunto de linhas

Síntese e simulação



Simulação comportamental

Síntese, implementação e teste

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

entity TopTrivial is
   port (       sw     : in  std_logic_vector (15 downto 0);
                led    : out  std_logic_vector (15 downto 0));
end TopTrivial;

architecture Behavioral of TopTrivial is
begin
                led <= sw;
end Behavioral;
```

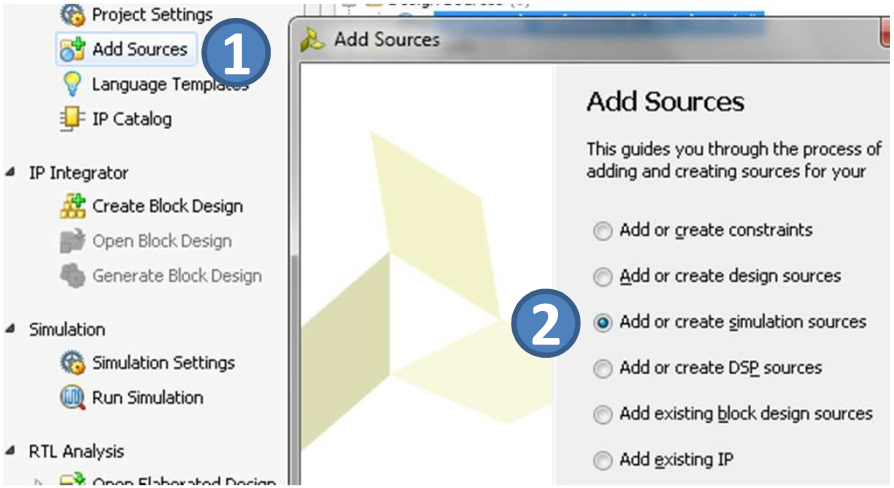
por exemplo:

Geralmente vai precisar de duas partes:

1) **Código VHDL de que queremos simular;**
2) **Código VHDL adicional que se chama *testbench*.**

Simulação. Exemplo 1.

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

entity TopTrivial is
   port (      sw     : in  std_logic_vector (15 downto 0);
               led    : out  std_logic_vector (15 downto 0));
end TopTrivial;

architecture Behavioral of TopTrivial is
begin
                led <= sw;
end Behavioral;
```



Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
entity for_example is
end for_example;
architecture behavior of for_example is
    signal sw_in      : std_logic_vector (15 downto 0);
    signal led_out    : std_logic_vector (15 downto 0);
    component TopTrivial
    port (      sw      : in  STD_LOGIC_VECTOR (15
downto 0);
                led     : out  STD_LOGIC_VECTOR (15
downto 0));
    end component;
 begin
uut: TopTrivial
port map (sw => sw_in, led => led_out);
stim_proc: process
begin
   sw_in <= (others => '1');
             wait for 50 ns;
   sw_in <= (others => '1');
             wait for 100 ns;
   sw_in <= (15 downto 10 => '1', others => '0');
             wait for 50 ns;
   sw_in <= "1010101010101010";
             wait for 70 ns;
   sw_in <= (15 downto 13 => '1',10 => '1', 6 downto 5
=> '1', others => '0');
end process;
end behavior;
```

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

entity TopTrivial is
```

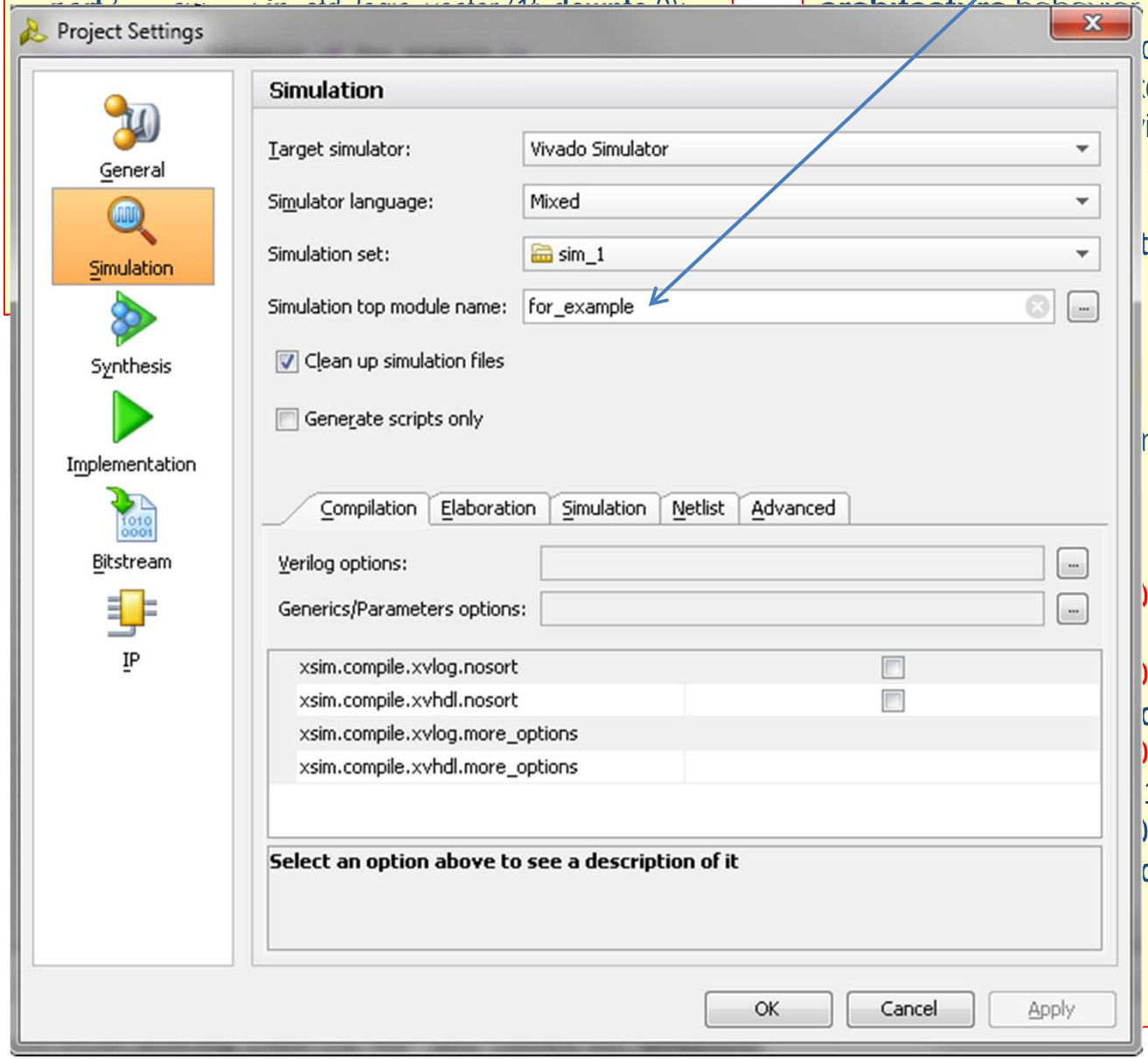
Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
entity for_example is
end for_example;
```

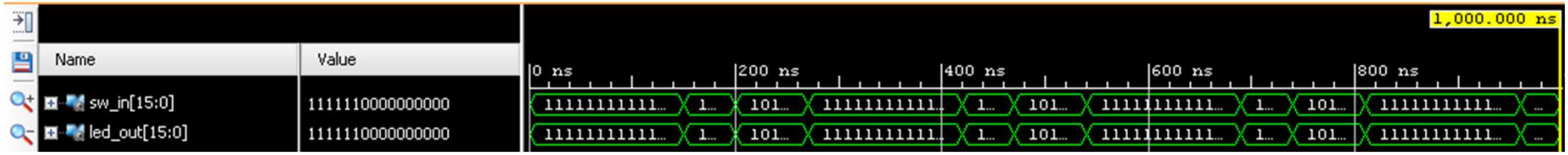
```
of for_example is
d_logic_vector (15 downto 0);
d_logic_vector (15 downto 0);
ial
 STD_LOGIC_VECTOR (15

t  STD_LOGIC_VECTOR (15
```

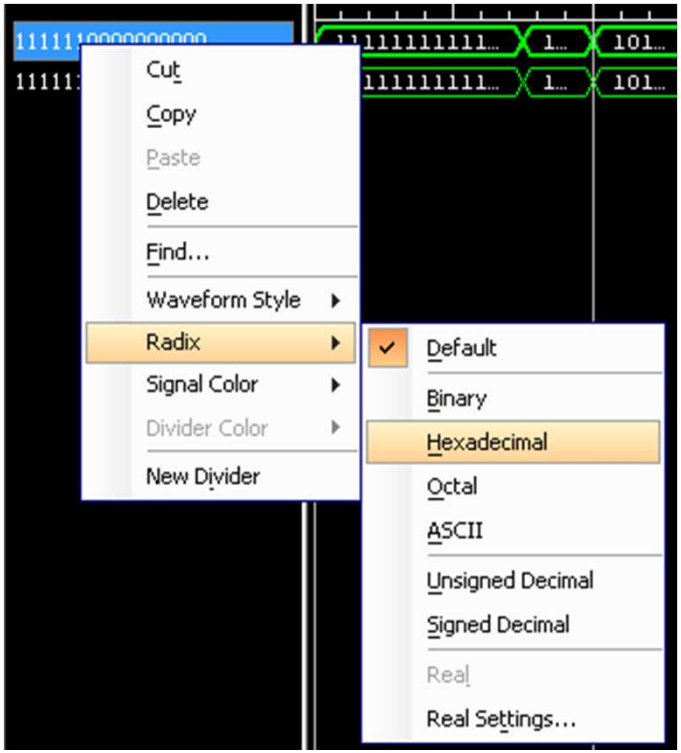
```
n, led => led_out);
```

```
 '1');
) ns;
 '1');
)0 ns;
o 10 => '1', others => '0');
) ns;
 10101010";
) ns;
o 13 => '1',10 => '1', 6 downto 5
```

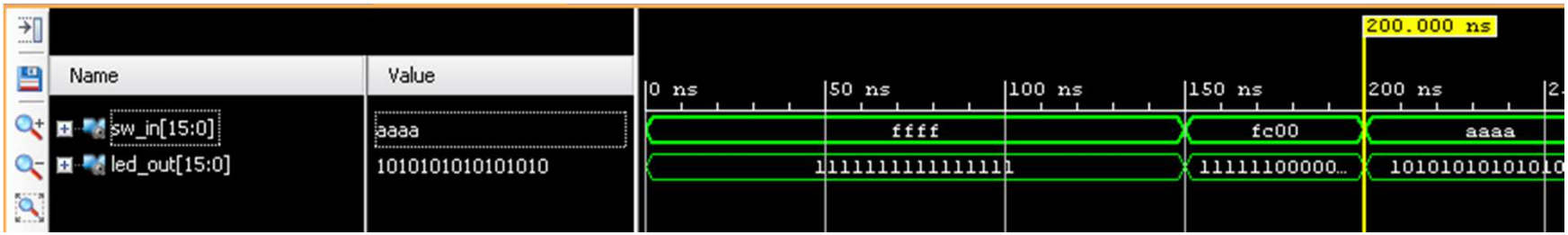



Zoom Fit


Name
Value
sw_in[15:0]
led_out[15:0]
1111110000000000
1111110000000000
1,000.000 ns
0 ns
200 ns
400 ns
600 ns
800 ns


Cut
Copy
Paste
Delete
Find...
Waveform Style
Radix
Signal Color
Divider Color
New Divider
Default
Binary
Hexadecimal
Octal
ASCII
Unsigned Decimal
Signed Decimal
Real
Real Settings...


Name
Value
sw_in[15:0]
led_out[15:0]
fc00
1111110000000000
0 ns
200 ns
400 ns
600 ns
800 ns




```
library IEEE;
use IEEE.std_logic_1164.all;
entity for_example is
end for_example;
architecture behavior of for_example is
    signal sw_in      : std_logic_vector (15 downto 0);
    signal led_out    : std_logic_vector (15 downto 0);
    component TopTrivial
    port (     sw      : in  STD_LOGIC_VECTOR (15 downto 0);
               led     : out  STD_LOGIC_VECTOR (15 downto 0));
    end component;
 begin
uut: TopTrivial
port map (sw => sw_in, led => led_out);
stim_proc: process
begin
   sw_in <= (others => '1');                                  wait for 50 ns;
   sw_in <= (others => '1');                                  wait for 100 ns;
   sw_in <= (15 downto 10 => '1', others => '0');             wait for 50 ns;
   sw_in <= "1010101010101010";                               wait for 70 ns;
   sw_in <= (15 downto 13 => '1',10 => '1', 6 downto 5 => '1', others => '0');
end process;
end behavior;
```

Pode alterar o código da seguinte forma:

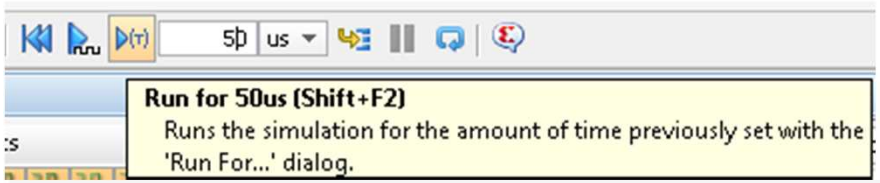
remover

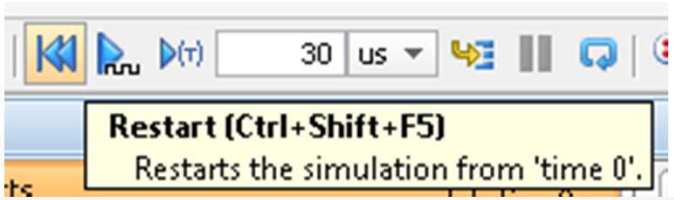
```
library IEEE;
use IEEE.std_logic_1164.all;
library xil_defaultlib;
entity for_example is
end for_example;
architecture behavior of for_example is
    signal sw_in      : std_logic_vector (15 downto 0);
    signal led_out    : std_logic_vector (15 downto 0);
    component TopTrivial
    port (     sw      : in  STD_LOGIC_VECTOR (15 downto 0);
               led     : out  STD_LOGIC_VECTOR (15 downto 0));
    end component;
 begin
uut: TopTrivial
uut: entity xil_defaultlib.TopTrivial
port map (sw => sw_in, led => led_out);
stim_proc: process
begin
   sw_in <= (others => '1');                                  wait for 50 ns;
   sw_in <= (others => '1');                                  wait for 100 ns;
   sw_in <= (15 downto 10 => '1', others => '0');             wait for 50 ns;
   sw_in <= "1010101010101010";                               wait for 70 ns;
   sw_in <= (15 downto 13 => '1',10 => '1', 6 downto 5 => '1', others => '0');
end process;
end behavior;
```

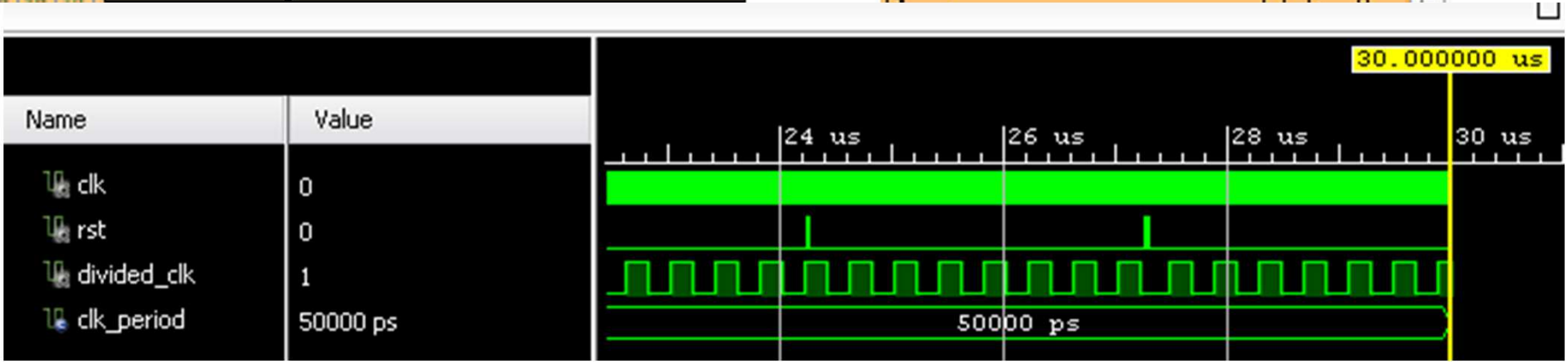
Simulação. Exemplo 2 (divisor da frequência).

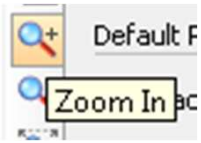
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;
entity clock_divider is
port          ( clk, reset      : in std_logic;
                divided_clk  : out std_logic          );
end clock_divider;
architecture Behavioral of clock_divider is
  signal internal_clock : std_logic_vector (2 downto 0);
begin
process(clk)
begin
if rising_edge(clk) then
  if reset = '1' then       -- reset sincrono
               internal_clock <= (others=>'0');
  else         internal_clock <= internal_clock+1;
  end if;
end if;
end process;
divided_clk <= internal_clock(internal_clock'left);
end Behavioral;
```

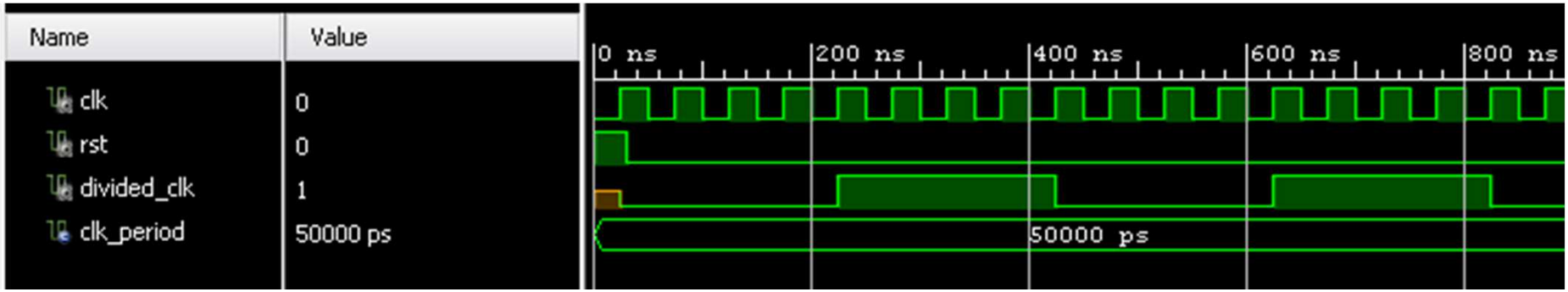
```
library IEEE;
use IEEE.std_logic_1164.all;
library xil_defaultlib;
entity testbench is
end    testbench;
architecture behavior of for_example is
    signal clk              : std_logic;
    signal rst              : std_logic;
    signal divided_clk      : std_logic;
constant clk_period     : time := 50 ns;
begin
div: entity xil_defaultlib.clock_divider
port map (clk, rst, divided_clk);
clock_gen process
begin
clk <= ‘0’;                 wait for clk_period/2;
clk <= ‘1’;                 wait for clk_period/2;
end process clock_gen;
stim_proc: process
begin
   rst <= ‘1’;              wait for 30 ns;
   rst <= ‘0’;              wait for 1000 ns;
end process;
end behavior;
```


5p
us
Run for 50us (Shift+F2)
Runs the simulation for the amount of time previously set with the 'Run For...' dialog.


30
us
Restart (Ctrl+Shift+F5)
Restarts the simulation from 'time 0'.


Name
Value
clk
0
rst
0
divided_clk
1
clk_period
50000 ps
30.000000 us
24 us
26 us
28 us
30 us
50000 ps

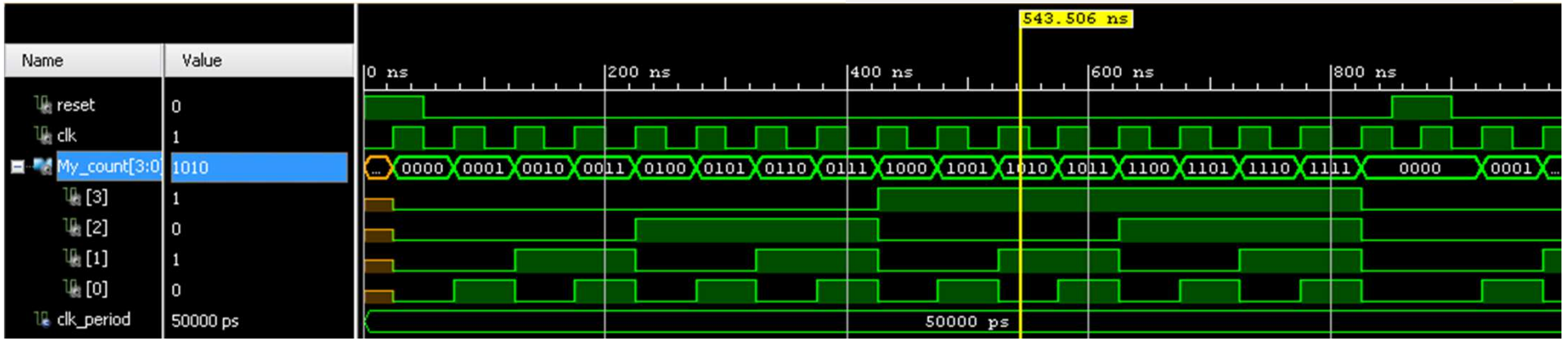

Default
Zoom In


Name
Value
clk
0
rst
0
divided_clk
1
clk_period
50000 ps
0 ns
200 ns
400 ns
600 ns
800 ns
50000 ps

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
entity Counter is
    port (reset             : in std_logic;
          clk               : in std_logic;
          My_count          : out std_logic_vector(3 downto 0));
end Counter;
architecture Behavioral of Counter is
signal count : std_logic_vector(3 downto 0);
begin
counter: process (clk)
begin
   if rising_edge(clk) then
       if reset = '1' then
           count <= (others => '0');
       else    count <= count + 1;
       end if;
   end if;
end process counter;
My_count <= count;
end Behavioral;
```

Simulação. Exemplo 3 (contador).

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
library xil_defaultlib;
entity for_example is
end for_example;
architecture behavior of for_example is
   signal reset , clk   : std_logic := '0';
   signal My_count   : std_logic_vector(3 downto 0);
   constant clk_period       : time := 50 ns;
 begin
 uut: entity xil_defaultlib.Counter
port map (clk, reset, My_count);
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
   reset <= '1';                wait for 50 ns;
   reset <= '0';                wait for 100 ns;
   reset <= '0';                wait for 100 ns;
   reset <= '0';                wait for 100 ns;
   reset <= '0';                wait for 500 ns;
end process;
end behavior;
```



Tipos de dados, objetos e operadores

Tipos

Vamos utilizar estes tipos:

Type	Where declared	Possible values
bit	standard in VHDL	'0', '1'
bit_vector	standard in VHDL	array of bits
boolean	standard in VHDL	false, true
character	standard in VHDL	7-bit ASCII codes in ISE
integer	standard in VHDL	at least 32 bits (-2^{31} to $2^{31}-1$)
natural	standard in VHDL	subtype of integer: at least from 0 to $2^{31}-1$
positive	standard in VHDL	subtype of integer: at least from 1 to $2^{31}-1$
real	There are many restrictions for synthesis	floating-point values
signed	packages: ieee.std_logic_arith, ieee.numeric_std	array of std_logic
std_logic	package: ieee.std_logic_1164	resolved std_ulogic
std_logic_vector	package: ieee.std_logic_1164	array of std_logic
std_ulogic	package: ieee.std_logic_1164	'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'
std_ulogic_vector	package: ieee.std_logic_1164	array of std_ulogic
string	standard in VHDL	array of characters
time	standard in VHDL	time units: hr, min, sec, ms, us, ns, ps, fs
unsigned	packages: ieee.numeric_std, ieee.std_logic_arith	array of std_logic

Só para simulação

O tipo **record** é definido por utilizador e permite criar um conjunto de dados dentro de uma estrutura

Tipos de dados, objetos e operadores

Vamos considerar três tipos de objetos que são:

1. Sinais (**signal**) que são declarados na parte declarativa de arquitetura.

```
arquitecture <nome de arquitetura > of <nome de entidade> is
Declaração de sinais– signal
begin
                Corpo de arquitetura
end <nome de arquitetura >
```

2. Variáveis (**variable**) que são declaradas na parte declarativa de processo, função e procedimento.

```
process (sw)
variable hwc   : integer range 0 to 8;
begin
  -- . . . . . .
end process;
```

3. Constantes (**constant**) que são declaradas na parte declarativa de arquitetura, processo, função e procedimento.

```
architecture Behavioral of CombCircuit is
        type for_test is array (0 to 3) of std_logic_vector(3 downto 0);
        constant for_ex : for_test:=("0001","0010","0100","1000");
begin
-- . . . . . .
```

Tipos de dados, objetos e operadores

aritméticos (arithmetic)	(+, -, *, /, **abs**, **mod**, **rem**, sign + and -, **),
concatenação (concatenation)	(&),
lógicos (logic)	(**and**, **nand**, **nor**, **not**, **or**, **xnor**, **xor**),
relacionais (relational)	(=, /=, <, <=, >, >=),
atribuição (assignment)	(:=, <=)
deslocamento (shift)	(**rol**, **ror**, **sla**, **sll**, **sra**, **srl**).

Prioridade (precedence) :

(**),
(**abs**),
(**not**),
(*),
(/),
(**mod**, **rem**),
(+ identity, - negation),
(+,-),
(&),
(**rol**, **ror**, **sll**, **srl**),
(**sla**, **sra**),
(=,/=),
(<, <=, >, >=),
(**and**, **nand**, **nor**, **or**, **xnor**, **xor**)

Instruções de controlo – decisão: *if*, *when*, *with*, *case* .
Processos combinatórios

Funcionamento

sw_0 sw_1	led_3 led_2 led_1 led_0
00	0001
01	0010
10	0100
11	1000

Instruções de controlo – decisão: *if*, *when*, *with*, *case* .
Processos combinatórios

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

entity CombCircuit is
    port (   sw          : in   std_logic_vector (1 downto 0);
             led         : out  std_logic_vector (3 downto 0));
end CombCircuit;

architecture Behavioral of CombCircuit is
begin

led <=       "0001"      when sw = "00" else
             "0010"      when sw = "01" else
             "0100"      when sw = "10" else
             "1000"      when sw = "11" else "0000";

end Behavioral;
```

00	0001
01	0010
10	0100
11	1000

Utilizar dentro de arquitetura

Instruções de controlo – decisão: *if*, *when*, *with*, *case* .
Processos combinatórios

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

entity CombCircuit is
    port (   sw          : in   std_logic_vector (1 downto 0);
             led         : out  std_logic_vector (3 downto 0));
end CombCircuit;

architecture Behavioral of CombCircuit is
begin

with sw select led <=          "0001" when "00",
                               "0010" when "01",
                               "0100" when "10",
                               "1000" when "11",
                               "0000" when others;

end Behavioral;
```

00	0001
01	0010
10	0100
11	1000

Utilizar dentro de arquitetura

Instruções de controlo – decisão: *if*, *when*, *with*, *case* .
Processos combinatórios

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

entity CombCircuit is
    port (   sw          : in   std_logic_vector (1 downto 0);
             led         : out  std_logic_vector (3 downto 0));
end CombCircuit;

architecture Behavioral of CombCircuit is
begin

process(sw)
begin
   case sw is
      when "00" => led <= "0001";
      when "01" => led <= "0010";
      when "10" => led <= "0100";
      when "11" => led <= "1000";
      when others => led <= "0000";
   end case;
end process;

end Behavioral;
```

00	0001
01	0010
10	0100
11	1000

Não pode (!!!)
utilizar dentro de arquitetura

Instruções de controlo – decisão: *if*, *when*, *with*, *case* .
Processos combinatórios

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

entity CombCircuit is
    port (   sw          : in   std_logic_vector (1 downto 0);
             led         : out  std_logic_vector (3 downto 0));
end CombCircuit;

architecture Behavioral of CombCircuit is
begin

process(sw)
begin
   if      sw = "00" then led <= "0001";
   elsif   sw = "01" then led <= "0010";
   elsif   sw = "10" then led <= "0100";
   elsif   sw = "11" then led <= "1000";
   else     led <= "0000";
   end if;
end process;

end Behavioral;
```

00	0001
01	0010
10	0100
11	1000

Não pode (!!!)
utilizar dentro de arquitetura

Utilização de constantes.

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

entity CombCircuit is
    port (   sw          : in   std_logic_vector (1 downto 0);
             led         : out  std_logic_vector (3 downto 0));
end CombCircuit;

architecture Behavioral of CombCircuit is
             type for_test is array (0 to 3) of std_logic_vector(3 downto 0);
             constant for_ex : for_test:=("0001","0010","0100","1000");
begin

led <= for_ex(conv_integer(sw));

end Behavioral;
```

00	**0001**
01	**0010**
10	**0100**
11	**1000**

Processos combinatórios

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

entity TestCombProc is
port (      sw          : in   std_logic_vector(7 downto 0);
            led         : out std_logic_vector(7 downto 0));
end TestCombProc;

architecture Behavioral of TestCombProc is
   constant low      : integer := 5;
   constant high     : integer := 10;
begin

process(sw)
begin
   if (sw > low) and (sw < high) then led <= sw;
   elsif sw < low then led <= not sw;
   else led <= (others => '0');
   end if;
end process;

end Behavioral;
```

led <= sw; para valores de interruptores 6,7,8,9 (“0110”, “0111”, “1000”, “1001”).

led <= **not** sw; para valores de interruptores 0, 1, 2, 3, 4.

Para outros valores de interruptores todos os leds são iguais a 0

Todos os sinais que podem alterar valores dentro do processo devem aparecer na lista de sensibilidade

Processos sequenciais. Contador

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

entity TestSeqProc is
generic (  how_fast : integer := 30);
port (        clk       : in std_logic;
              sw        : in   std_logic_vector(15 downto 0);

              led       : out std_logic_vector(15 downto 0);
              btnL      : in std_logic;
              btnC      : in std_logic;
              btnR      : in std_logic);
end TestSeqProc;

architecture Behavioral of TestSeqProc is
   signal internal_clock          : unsigned(how_fast downto 0);
   signal divided_clk             : std_logic;
   signal increment               : std_logic := btnL;
   signal positive_reset          : std_logic := btnC;
   signal count_enable            : std_logic := btnR;
   signal count                   : std_logic_vector(14 downto 0);
begin

led(14 downto 0) <= count;
led(15) <= divided_clk;
```

Processos sequenciais. Contador

```
sp1: process(clk)
begin
   if rising_edge(clk) then        internal_clock <= internal_clock+1;
   end if;
   if falling_edge(clk) then
       divided_clk <= internal_clock(internal_clock'left - conv_integer(sw));
   end if;
end process sp1;

sp2: process (divided_clk)
begin
   if rising_edge(divided_clk) then
              if  positive_reset = '1' then count <= (others=>'0');
              else
                 if count_enable = '1' then
                              if increment='0' then    count <= count + 1;
                              else                     count <= count - 1;
                              end if;
                 end if;
              end if;
   end if;
end process sp2;

end Behavioral;
```

Processos sequenciais.
Registo de deslocamento

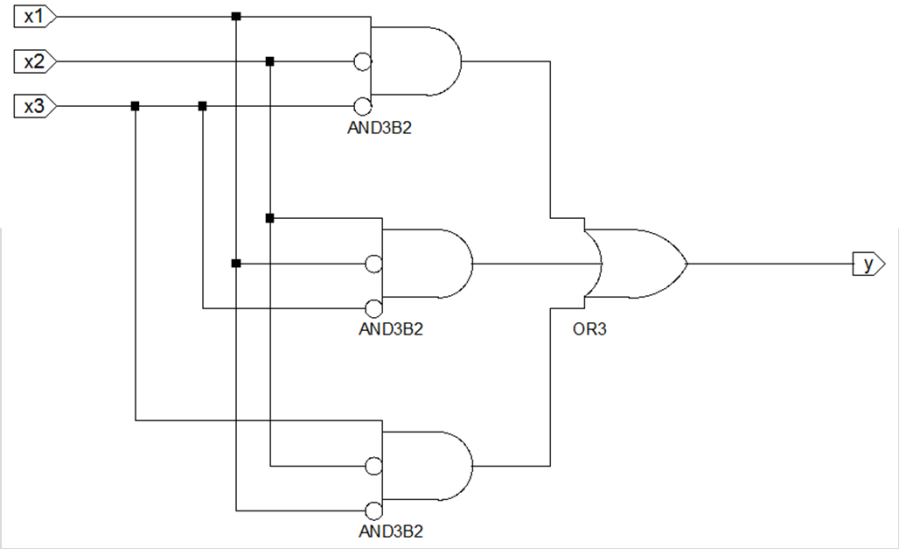
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.numeric_std.all;

entity TestSeqProc is
generic (  how_fast : integer := 30);
port (        clk       : in std_logic;
              sw        : in   std_logic_vector(15 downto 0);
              led       : out std_logic_vector(15 downto 0);
              btnL      : in std_logic;
              btnC      : in std_logic;
              btnR      : in std_logic);
end TestSeqProc;

architecture Behavioral of TestSeqProc is
   signal internal_clock          : unsigned(how_fast downto 0);
   signal divided_clk             : std_logic;
   signal right                   : std_logic := btnL;
   signal positive_reset          : std_logic := btnC;
   signal load_enable             : std_logic := btnR;
   signal shift                   : std_logic_vector(14 downto 0);
begin

led(14 downto 0) <= shift;
led(15) <= divided_clk;
```

Processos sequenciais.
Registo de deslocamento

```
sp1: process(clk)
begin
  if rising_edge(clk) then internal_clock <= internal_clock+1;
  end if;
  if falling_edge(clk) then
     divided_clk <= internal_clock(internal_clock'left);
  end if;
end process sp1;

sp3: process (divided_clk)
begin
  if rising_edge(divided_clk) then
            if  positive_reset = '1' then shift <= (others=>'0');
            else
              if load_enable = '1' then   shift <= sw;
              elsif right = '1' then
                        shift <= shift(0) & shift(14 downto 1);
              else
                        shift <= shift(13 downto 0) & shift(14);
              end if;
            end if;
  end if;
end process sp3;

end Behavioral;
```

VHDL comportamental e estrutural

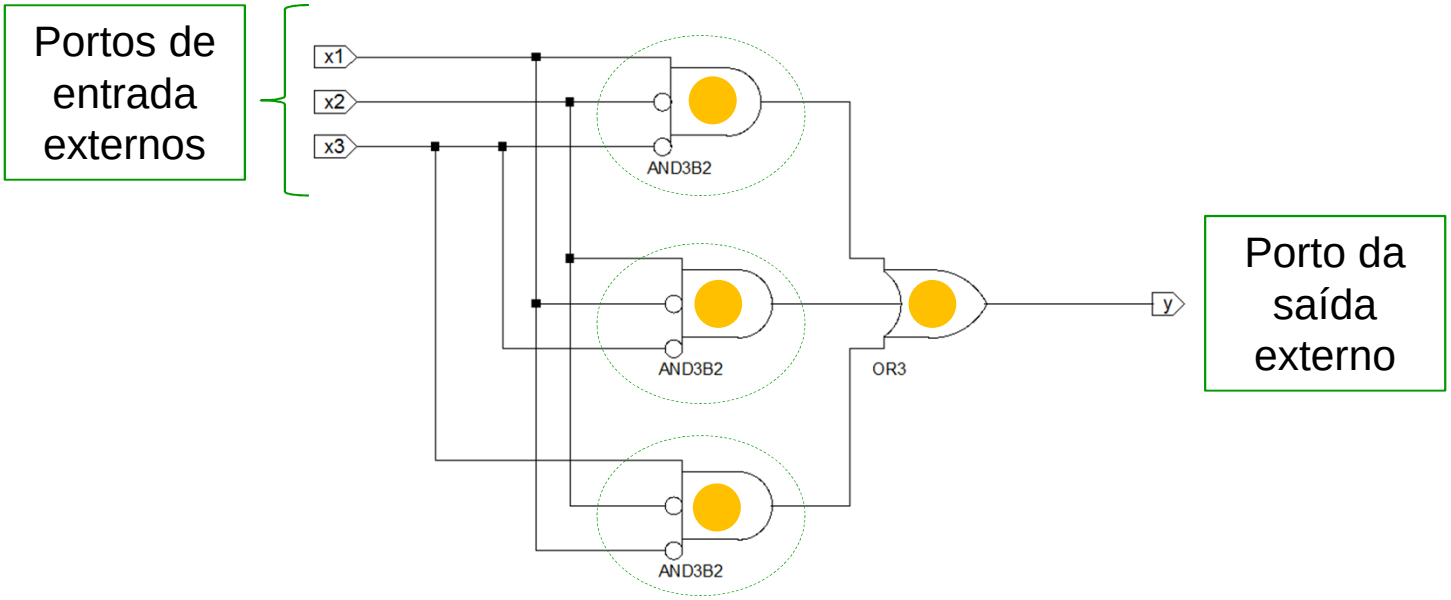
Descrição do comportamento semelhante à descrição na programação




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

entity behavioralVHDL is
   port (x1, x2, x3     : in    std_logic;
                 y      : out   std_logic);
end BehavioralVHDL;

architecture behavioral of BehavioralVHDL is
begin

    y <= (x1 and not x2 and not x3) or (not x1 and x2 and not x3) or (not x1 and not x2 and x3);

end behavioral;
```

VHDL comportamental e estrutural

Permite descrever um circuito com base em componentes existentes que são incluídos na descrição. Ligações entre componentes são escritas utilizando sinais

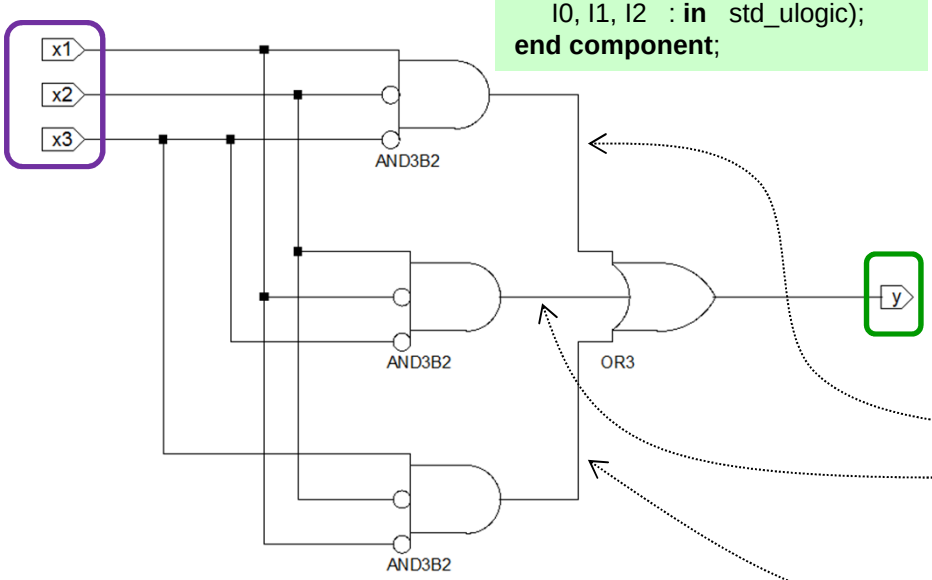


Biblioteca Xilinx UNISIM (ficheiro *unisim_VCOMP.vhd*). Exemplo de descrição:

```
component AND3B3
  port (    O          : out      std_ulogic;
            I0, I1, I2 : in       std_ulogic);
end component;
```

VHDL comportamental e estrutural

```
component AND3B3
  port (O      : out  std_ulogic;
        I0, I1, I2 : in   std_ulogic);
end component;
```




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

library UNISIM;
use UNISIM.Vcomponents.all;
```

Biblioteca Xilinx

```
entity StructuralVHDL is
  port ( x1,x2,x3        : in   std_logic;
         y               : out  std_logic  );
end StructuralVHDL;

architecture BEHAVIORAL of StructuralVHDL is
  signal out_and1 : std_logic;
  signal out_and2 : std_logic;
  signal out_and3 : std_logic;
begin
  or_circuit : OR3
    port map (I0=>out_and1, I1=>out_and2,
              I2=>out_and3, O=>y);

  and1_circuit : AND3B2
    port map (I0=>x3, I1=>x2, I2=>x1, O=>out_and1);

  and2_circuit : AND3B2
    port map (I0=>x3, I1=>x1, I2=>x2, O=>out_and2);

  and3_circuit : AND3B2
    port map (I0=>x1, I1=>x2, I2=>x3, O=>out_and3);

end BEHAVIORAL;
```

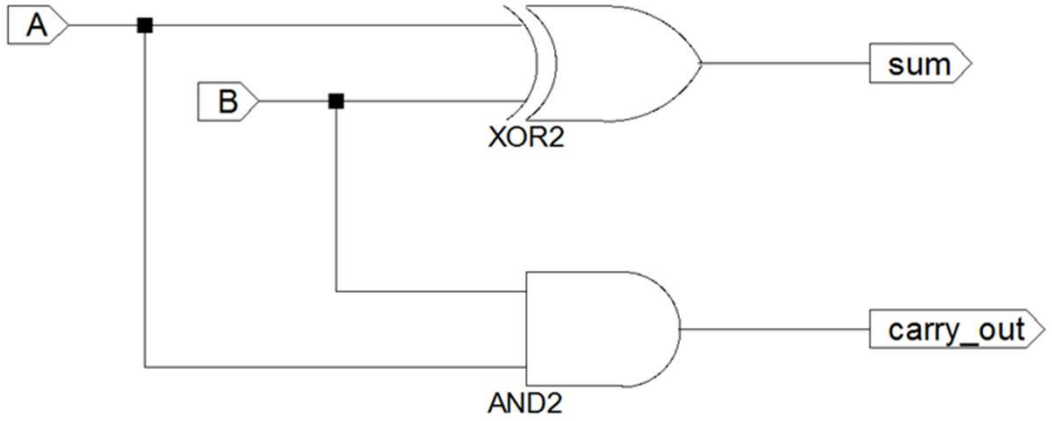
Mapeamento nomeado

Mapeamento posicional

```
begin
  or_circuit : OR3
    port map (out_and1, out_and2, out_and3,y);

  and1_circuit : AND3B2
    port map (x3, x2, x1, out_and1);

  and2_circuit : AND3B2
    port map (x3, x1, x2, out_and2);

  and3_circuit : AND3B2
    port map (x1, x2, x3, out_and3);
end BEHAVIORAL;
```

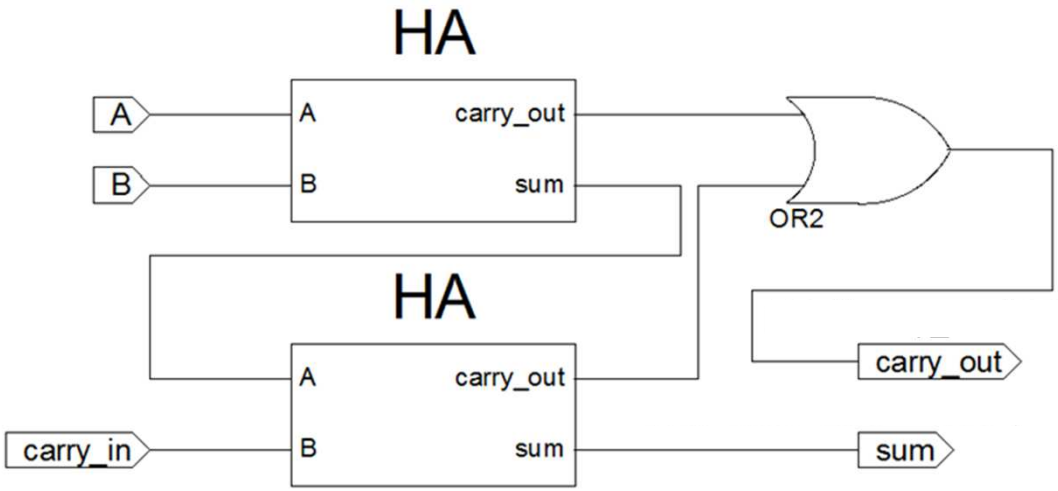
VHDL comportamental e estrutural

Half-adder




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

entity half_adder is
   port (  A          : in std_logic;
           B          : in std_logic;
           carry_out  : out std_logic;
           sum        : out std_logic);
end half_adder;

architecture half_adder_behavior of half_adder is
begin

           sum       <= A xor B;
           carry_out <= A and B;

end half_adder_behavior;
```

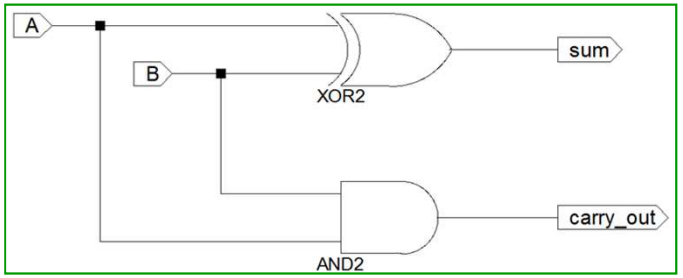
VHDL comportamental e estrutural




```
library IEEE;
use IEEE.std_logic_1164.all;
entity FULLADD is
port (      A, B, carry_in          : in   std_logic;
            sum, carry_out          : out std_logic );
end FULLADD;
architecture STRUCT of FULLADD is
            signal s1, s2, s3 : std_logic;
component half_adder
  port(     A,B                     : in   std_logic;
            carry_out, sum          : out std_logic);
end component;
begin
            u1:   half_adder        port map(A, B, s2, s1);
            u2:   half_adder        port map(s1, carry_in, s3, sum);
            carry_out <= s2 or s3;
end STRUCT;
```

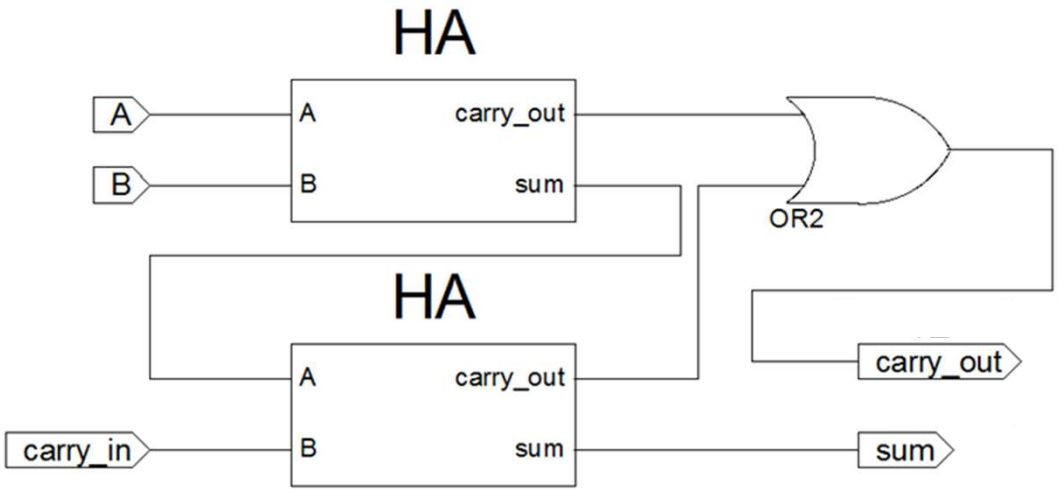
Descrição mista

Half-adder (HA)

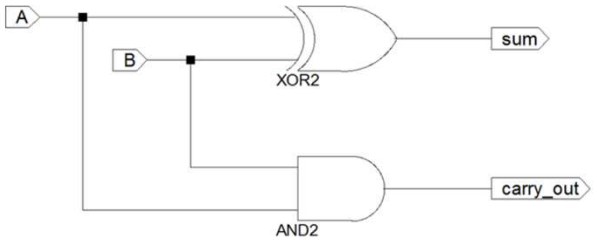



```
library IEEE;
use IEEE.std_logic_1164.all;

entity half_adder is
   port (      A: in std_logic;
               B: in std_logic;
               carry_out : out std_logic;
               sum: out std_logic);
end half_adder;

architecture half_adder_behavior of
half_adder is
begin

   sum <= A xor B;
   carry_out <= A and B;

end half_adder_behavior;
```

VHDL comportamental e estrutural



Half-adder (HA)



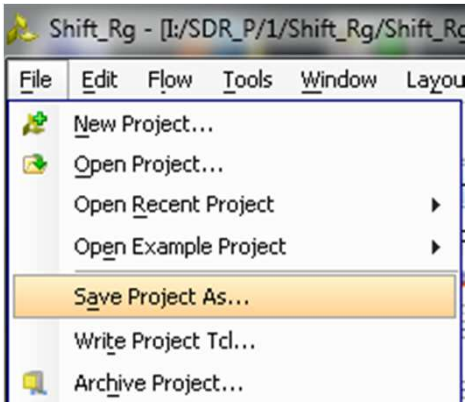

```
library IEEE;
use IEEE.std_logic_1164.all;

entity half_adder is
   port (          A              : in std_logic;
                   B              : in std_logic;
                   carry_out      : out std_logic;
                   sum            : out std_logic);

end half_adder;

architecture half_adder_behavior of half_adder is
begin

                   sum <= A xor B;
                   carry_out <= A and B;

end half_adder_behavior;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
entity FULLADD is
port (      A, B, carry_in          : in   std_logic;
            sum, carry_out          : out std_logic );
end FULLADD;
architecture STRUCT of FULLADD is
            signal s1, s2, s3 : std_logic;
component half_adder
  port(     A,B                     : in   std_logic;
            carry_out, sum          : out std_logic);
end component;
begin
            u1:   half_adder        port map(A, B, s2, s1);
            u2:   half_adder        port map(s1, carry_in, s3, sum);
            carry_out <= s2 or s3;
end STRUCT;
```

Descrição mista

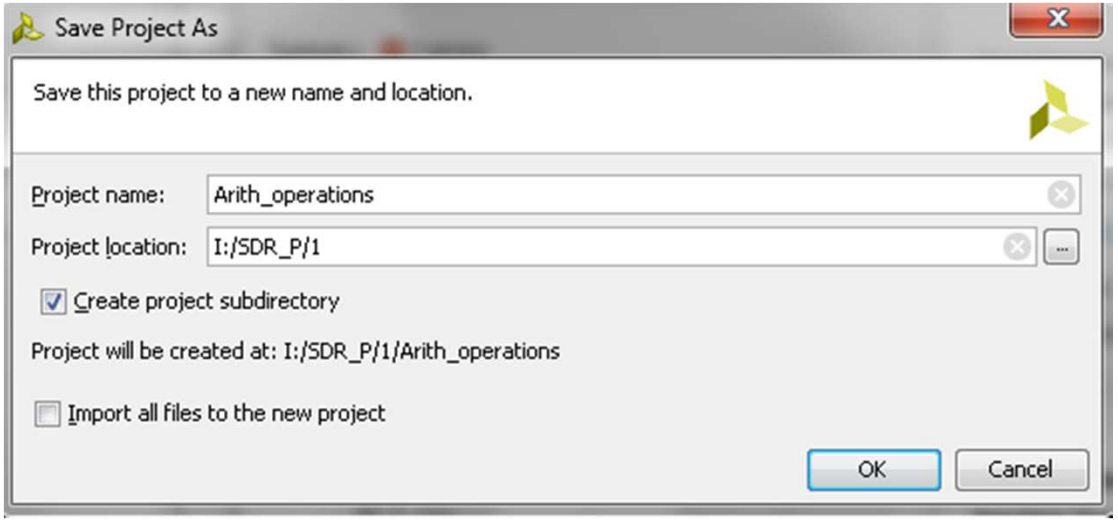
```
library xil_defaultlib;

entity xil_defaultlib.half_adder
```

Exemplos: operações aritméticas



Pode criar em Vivado um projeto com base em qualquer projeto existente

Depois pode só alterar código VHDL e modificar o ficheiro XDC



Exemplos: operações aritméticas

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;
entity arith_op is
port (      sw                              : in std_logic_vector(7 downto 0);
            led                             : out std_logic_vector(8 downto 0);
            btnU, btnC, btnD, btnL, btnR    : in std_logic);
end arith_op;
-- btnR  division; btnL  multiplication; btnD  addition; btnC  subtraction; btnU  rest of division
architecture Behavioral of arith_op is
  signal result         : integer range 0 to 256;
  signal but            : std_logic_vector(4 downto 0);
begin
  but <= btnU & btnC & btnD & btnL & btnR;
  result <=  256 when conv_integer(sw(3 downto 0)) = 0 else
                  conv_integer(sw(7 downto 4)) / conv_integer(sw(3 downto 0))
                   when but = "00001" else
                  conv_integer(sw(7 downto 4)) * conv_integer(sw(3 downto 0))
                   when but = "00010" else
                  conv_integer(sw(7 downto 4)) + conv_integer(sw(3 downto 0))
                   when but = "00100" else
                  conv_integer(sw(7 downto 4)) - conv_integer(sw(3 downto 0))
                   when but = "01000" else
                 conv_integer(sw(7 downto 4)) rem conv_integer(sw(3 downto 0))
                   when but = "10000" else 0;
  led(7 downto 0) <= conv_std_logic_vector(result, 8);
end Behavioral;
```

Exemplos: operações aritméticas

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.numeric_std.all;        -- use IEEE.STD_LOGIC_ARITH.all;
entity arith_op is
port (      sw                              : in std_logic_vector(7 downto 0);
            led                             : out std_logic_vector(8 downto 0);
            btnU, btnC, btnD, btnL, btnR    : in std_logic);
end arith_op;
-- btnR  division; btnL  multiplication; btnD  addition; btnC  subtraction; btnU  rest of division
architecture Behavioral of arith_op is
  signal result         : integer range 0 to 256;
  signal but            : std_logic_vector(4 downto 0);
begin
  but <= btnU & btnC & btnD & btnL & btnR;
  result <=  256 when conv_integer(sw(3 downto 0)) = 0 else
                  conv_integer(sw(7 downto 4)) / conv_integer(sw(3 downto 0))
                   when but = "00001" else
                  conv_integer(sw(7 downto 4)) * conv_integer(sw(3 downto 0))
                   when but = "00010" else
                  conv_integer(sw(7 downto 4)) + conv_integer(sw(3 downto 0))
                   when but = "00100" else
                  conv_integer(sw(7 downto 4)) - conv_integer(sw(3 downto 0))
                   when but = "01000" else
                 conv_integer(sw(7 downto 4)) rem conv_integer(sw(3 downto 0))
                   when but = "10000" else 0;
 led(7 downto 0) <= std_logic_vector(to_unsigned(result,8));  --led(7 downto 0) <= conv_std_logic_vector(result, 8);
end Behavioral;
```

Exemplos: processos

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.numeric_std.all;
entity arith_op is
port (          sw                                  : in std_logic_vector(7 downto 0);
                led                                 : out std_logic_vector(8 downto 0);
                btnU, btnC, btnD, btnL, btnR        : in std_logic);
end arith_op;
-- btnR  division; btnL  multiplication; btnD  addition; btnC  subtraction; btnU  rest of division
architecture Behavioral of arith_op is
  signal result             : integer range 0 to 256;
  signal but                : std_logic_vector(4 downto 0);
begin
  but <= btnU & btnC & btnD & btnL & btnR;
  process(but,sw)
  begin
   if conv_integer(sw(3 downto 0)) /= 0 then
      case but is
         when "00001" => result <= conv_integer(sw(7 downto 4)) / conv_integer(sw(3 downto 0));
         when "00010" => result <= conv_integer(sw(7 downto 4)) * conv_integer(sw(3 downto 0));
         when "00100" => result <= conv_integer(sw(7 downto 4)) + conv_integer(sw(3 downto 0));
         when "01000" => result <= conv_integer(sw(7 downto 4)) - conv_integer(sw(3 downto 0));
         when "10000" => result <= conv_integer(sw(7 downto 4)) rem conv_integer(sw(3 downto 0));
         when others => result <= 0;
      end case;
  else result <= 256;
  end if;
  end process;

  led(7 downto 0) <= std_logic_vector(to_unsigned(result,8));

end Behavioral;
```

Processo combinatório

Exemplos: conversão de tipos

```
signed_vector              <= signed(std_logic_vector_signal);
unsigned_vector            <= unsigned(std_logic_vector_signal);
std_logic_vector_signal    <= std_logic_vector(signed_vector);
std_logic_vector_signal    <= std_logic_vector(unsigned_vector);
```

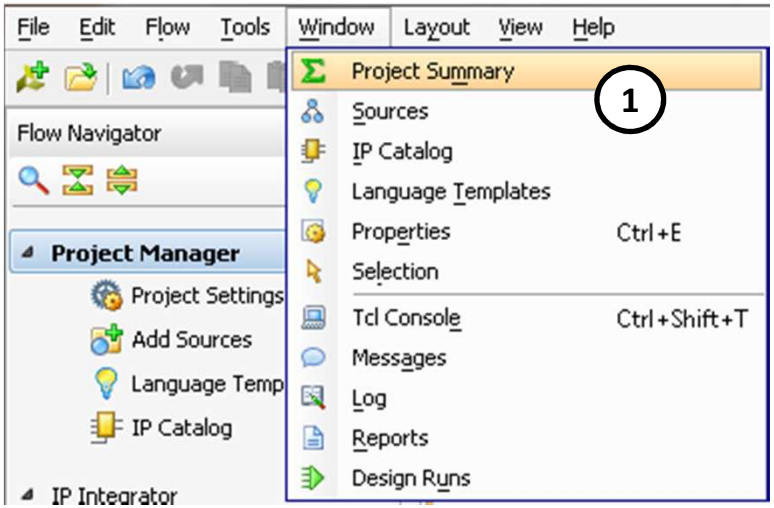
Seguintes expressões devem utilizar funções de conversão:

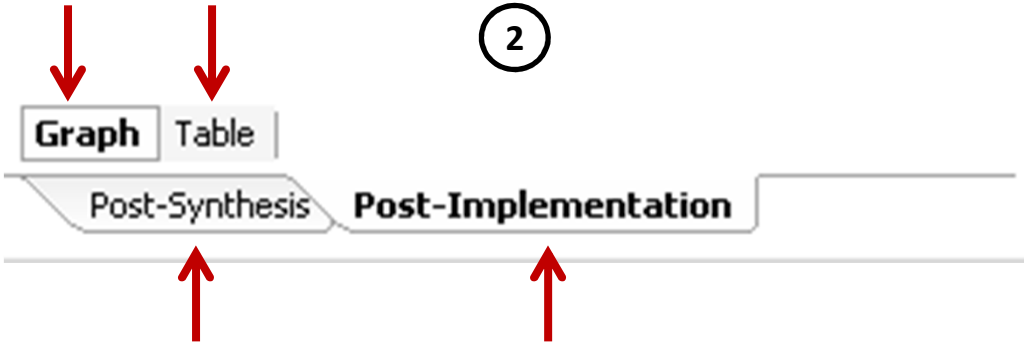
```
integer_signal      <= conv_integer (unsigned_vector);
integer_signal      <= conv_integer (signed_vector);
integer_signal      <= conv_integer (std_logic_vector_signal);
unsigned_vector     <= conv_unsigned (integer_signal, size_of_unsigned_vector);
signed_vector       <= conv_signed (integer_signal, size_of_signed_vector);
std_logic_vector_signal     <= conv_std_logic_vector (integer_signal, size);
```

As funções de conversão são diferentes para *pacotes diferentes*

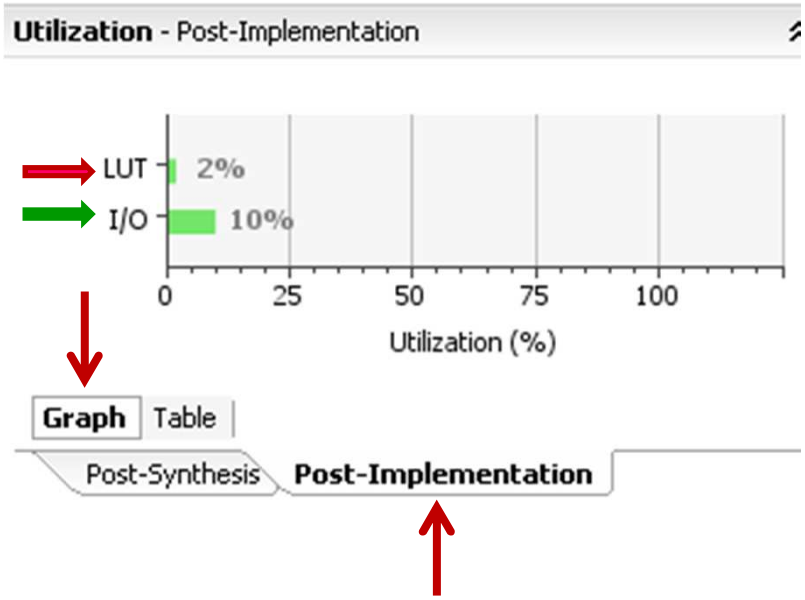
```
use ieee.std_logic_arith.all; → led(7 downto 0) <= conv_std_logic_vector(result, 8);

use ieee.numeric_std.all;    → led(7 downto 0) <= std_logic_vector(to_unsigned(result,8));
```

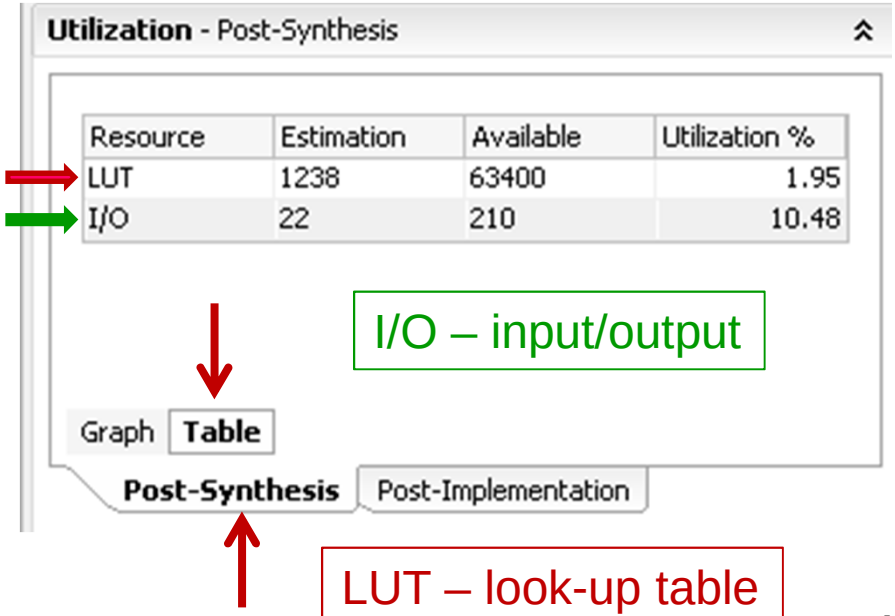
Como encontrar recursos da FPGA utilizados para o seu projeto





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

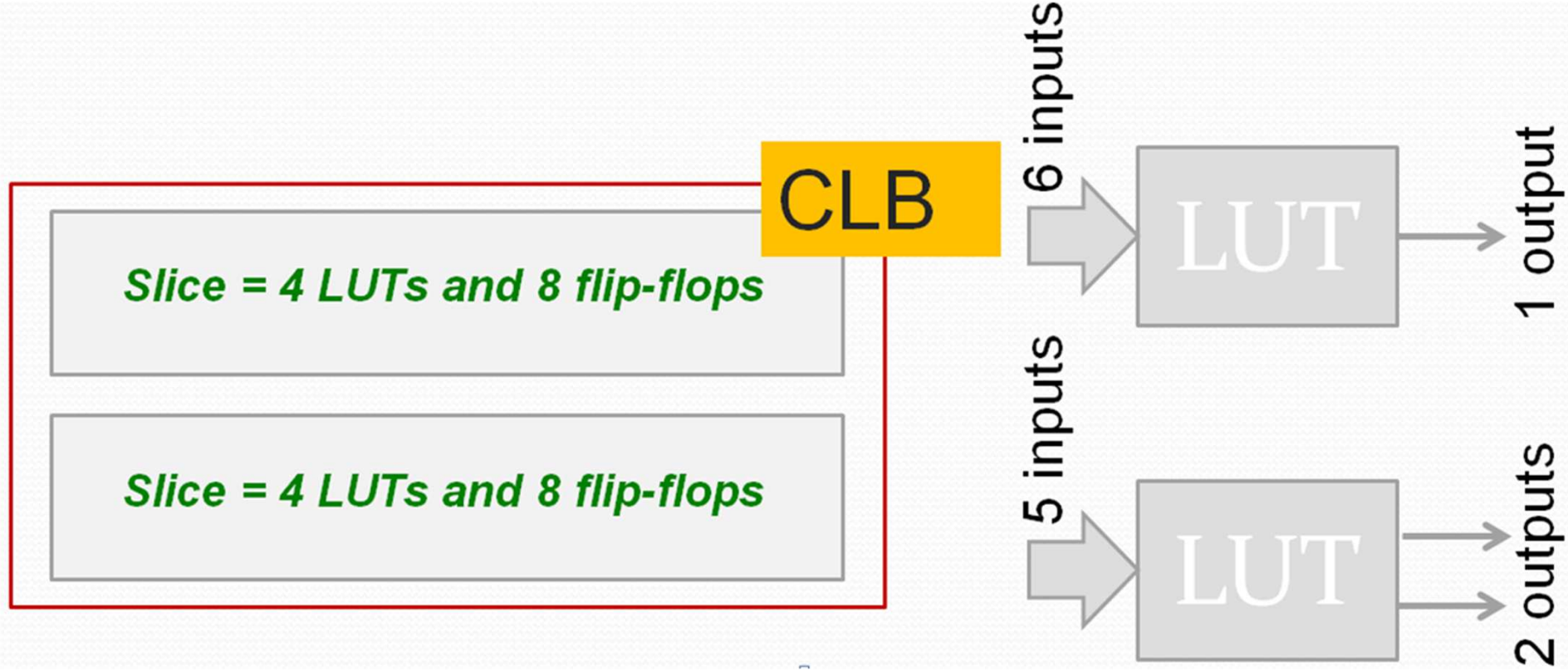


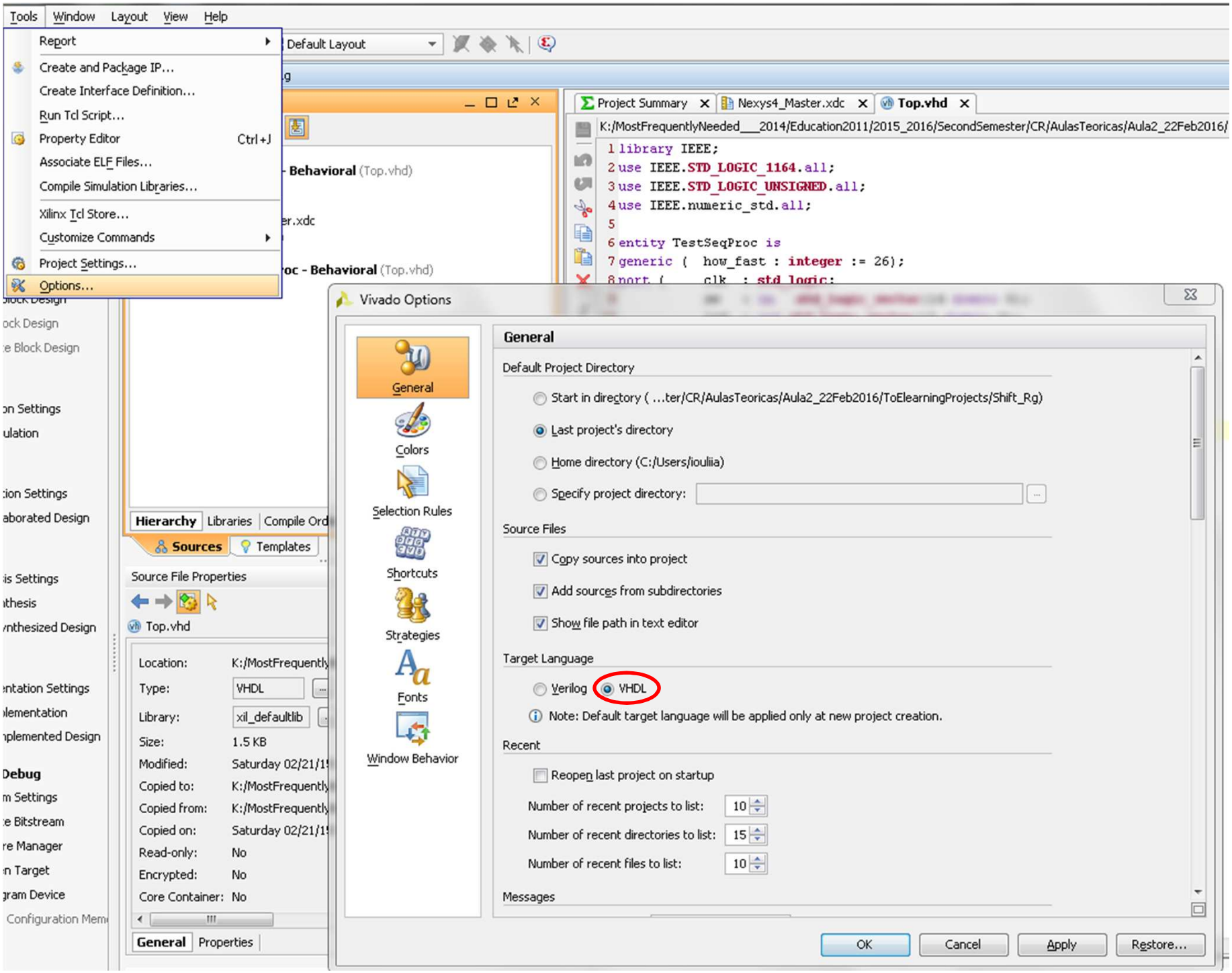


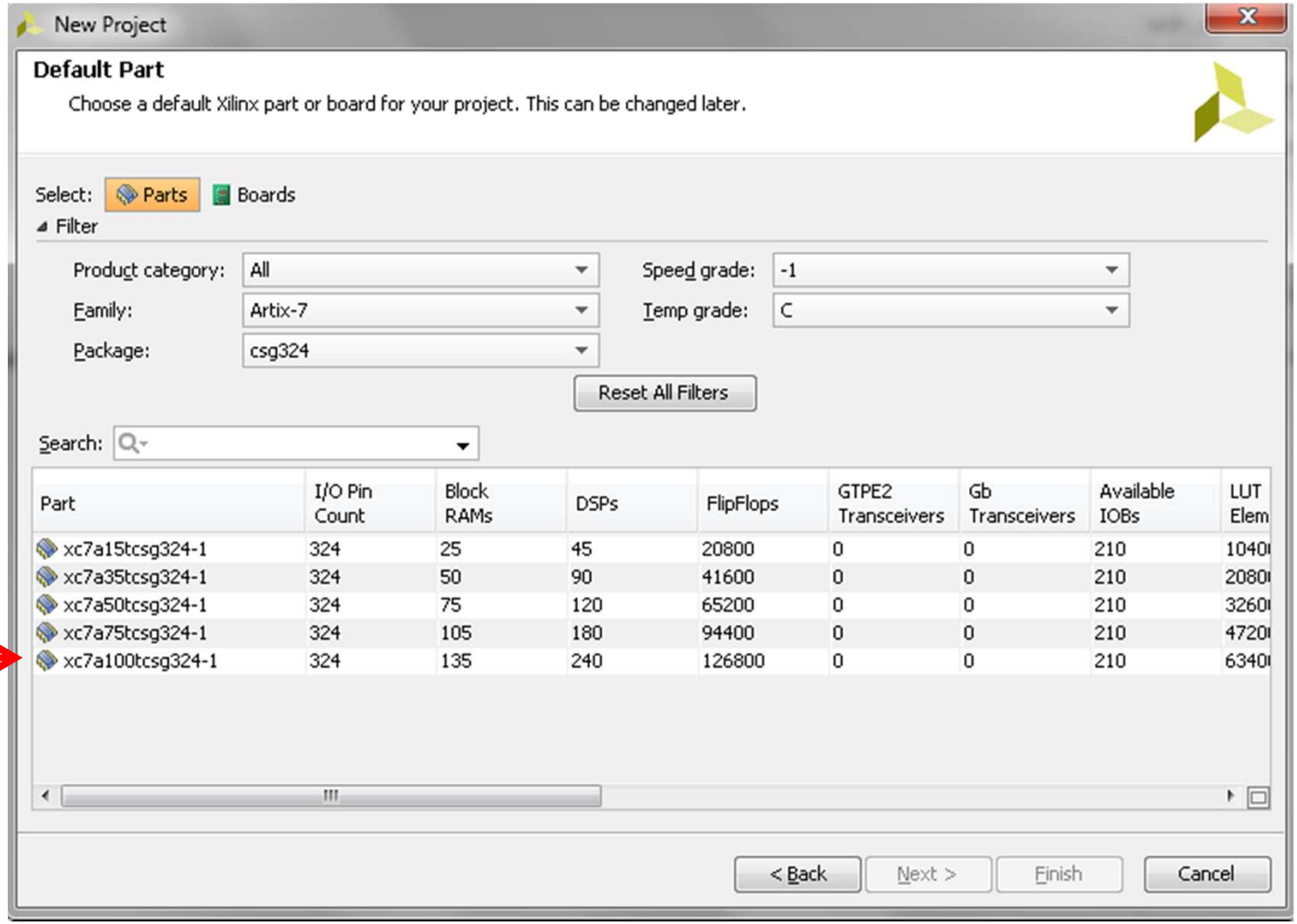
I/O – input/output

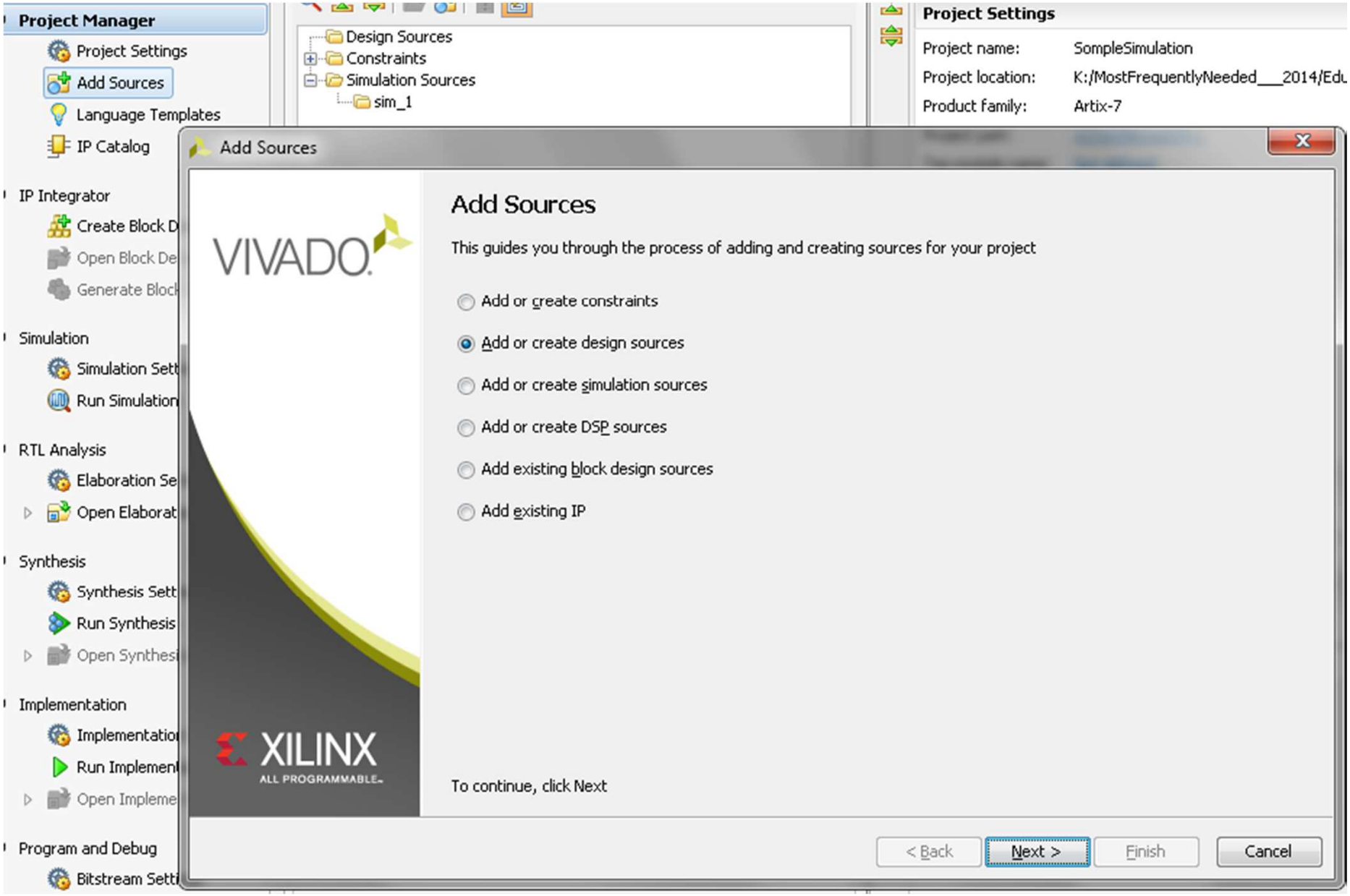
LUT – look-up table

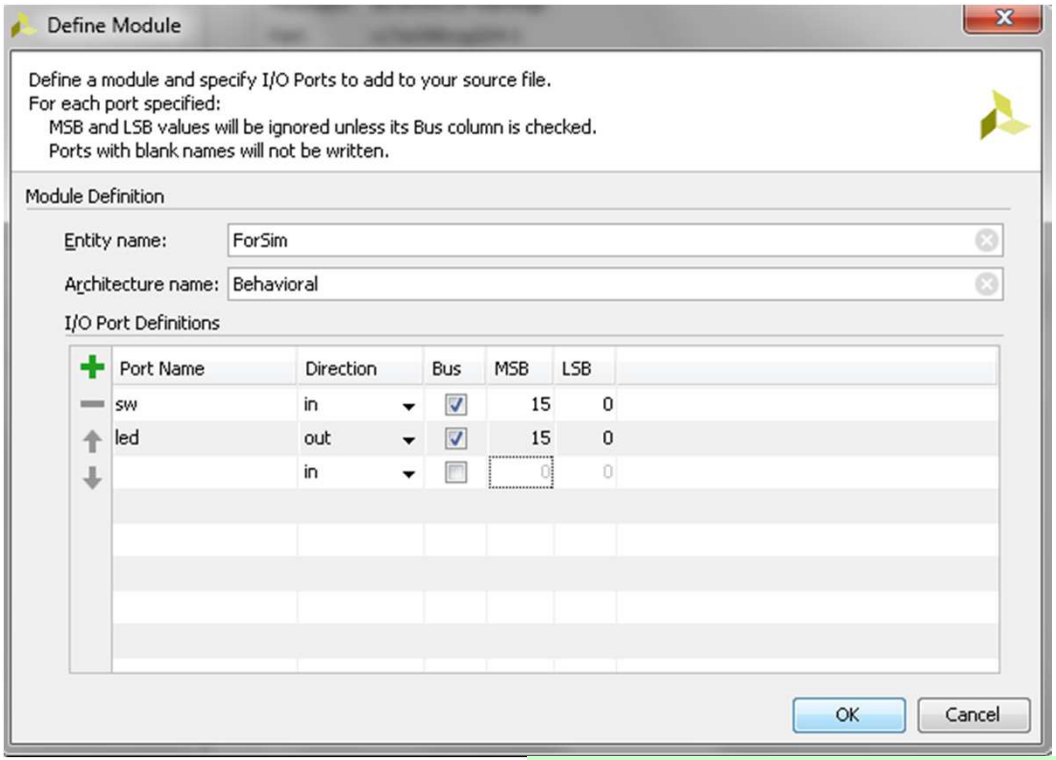
FPGA Artix-7 (família 7 de FPGAs de Xilinx)




Tools
Window
Layout
View
Help
Report
Create and Package IP...
Create Interface Definition...
Run Tcl Script...
Property Editor
Ctrl+J
Associate ELF Files...
Compile Simulation Libraries...
Xilinx Tcl Store...
Customize Commands
Project Settings...
Options...
Default Layout
Project Summary
Nexys4_Master.xdc
Top.vhd
K:/MostFrequentlyNeeded___2014/Education2011/2015_2016/SecondSemester/CR/AulasTeoricas/Aula2_22Feb2016/
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.STD_LOGIC_UNSIGNED.all;
4 use IEEE.numeric_std.all;
5
6 entity TestSeqProc is
7 generic (how_fast : integer := 26);
8 port (clk : std_logic;
Vivado Options
General
Colors
Selection Rules
Shortcuts
Strategies
Fonts
Window Behavior
Default Project Directory
Start in directory (...ter/CR/AulasTeoricas/Aula2_22Feb2016/ToElearningProjects/Shift_Rg)
Last project's directory
Home directory (C:/Users/ioulia)
Specify project directory:
Source Files
Copy sources into project
Add sources from subdirectories
Show file path in text editor
Target Language
Verilog
VHDL
Note: Default target language will be applied only at new project creation.
Recent
Reopen last project on startup
Number of recent projects to list: 10
Number of recent directories to list: 15
Number of recent files to list: 10
Messages
OK
Cancel
Apply
Restore...
Hierarchy
Libraries
Sources
Templates
Source File Properties
Top.vhd
Location: K:/MostFrequently
Type: VHDL
Library: xil_defaultlib
Size: 1.5 KB
Modified: Saturday 02/21/15
Copied to: K:/MostFrequently
Copied from: K:/MostFrequently
Copied on: Saturday 02/21/15
Read-only: No
Encrypted: No
Core Container: No
General
Properties
Debug

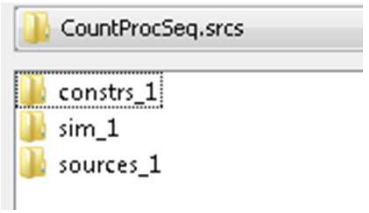

New Project
Default Part
Choose a default Xilinx part or board for your project. This can be changed later.
Select: Parts Boards
Filter
Product category: All
Family: Artix-7
Package: csg324
Speed grade: -1
Temp grade: C
Reset All Filters
Search:
Part | I/O Pin Count | Block RAMs | DSPs | FlipFlops | GTPE2 Transceivers | Gb Transceivers | Available IOBs | LUT Elem
xc7a15tcsg324-1 | 324 | 25 | 45 | 20800 | 0 | 0 | 210 | 1040
xc7a35tcsg324-1 | 324 | 50 | 90 | 41600 | 0 | 0 | 210 | 2080
xc7a50tcsg324-1 | 324 | 75 | 120 | 65200 | 0 | 0 | 210 | 3260
xc7a75tcsg324-1 | 324 | 105 | 180 | 94400 | 0 | 0 | 210 | 4720
xc7a100tcsg324-1 | 324 | 135 | 240 | 126800 | 0 | 0 | 210 | 6340
< Back
Next >
Finish
Cancel


Project Manager
Project Settings
Add Sources
Language Templates
IP Catalog
IP Integrator
Create Block D
Open Block De
Generate Block
Simulation
Simulation Sett
Run Simulation
RTL Analysis
Elaboration Se
Open Elaborat
Synthesis
Synthesis Sett
Run Synthesis
Open Synthesi
Implementation
Implementatio
Run Implement
Open Impleme
Program and Debug
Bitstream Setti
Design Sources
Constraints
Simulation Sources
sim_1
Project Settings
Project name: SompleSimulation
Project location: K:/MostFrequentlyNeeded___2014/Edu
Product family: Artix-7
Add Sources
VIVADO
Add Sources
This guides you through the process of adding and creating sources for your project
Add or create constraints
Add or create design sources
Add or create simulation sources
Add or create DSP sources
Add existing block design sources
Add existing IP
XILINX
ALL PROGRAMMABLE
To continue, click Next
< Back
Next >
Finish
Cancel



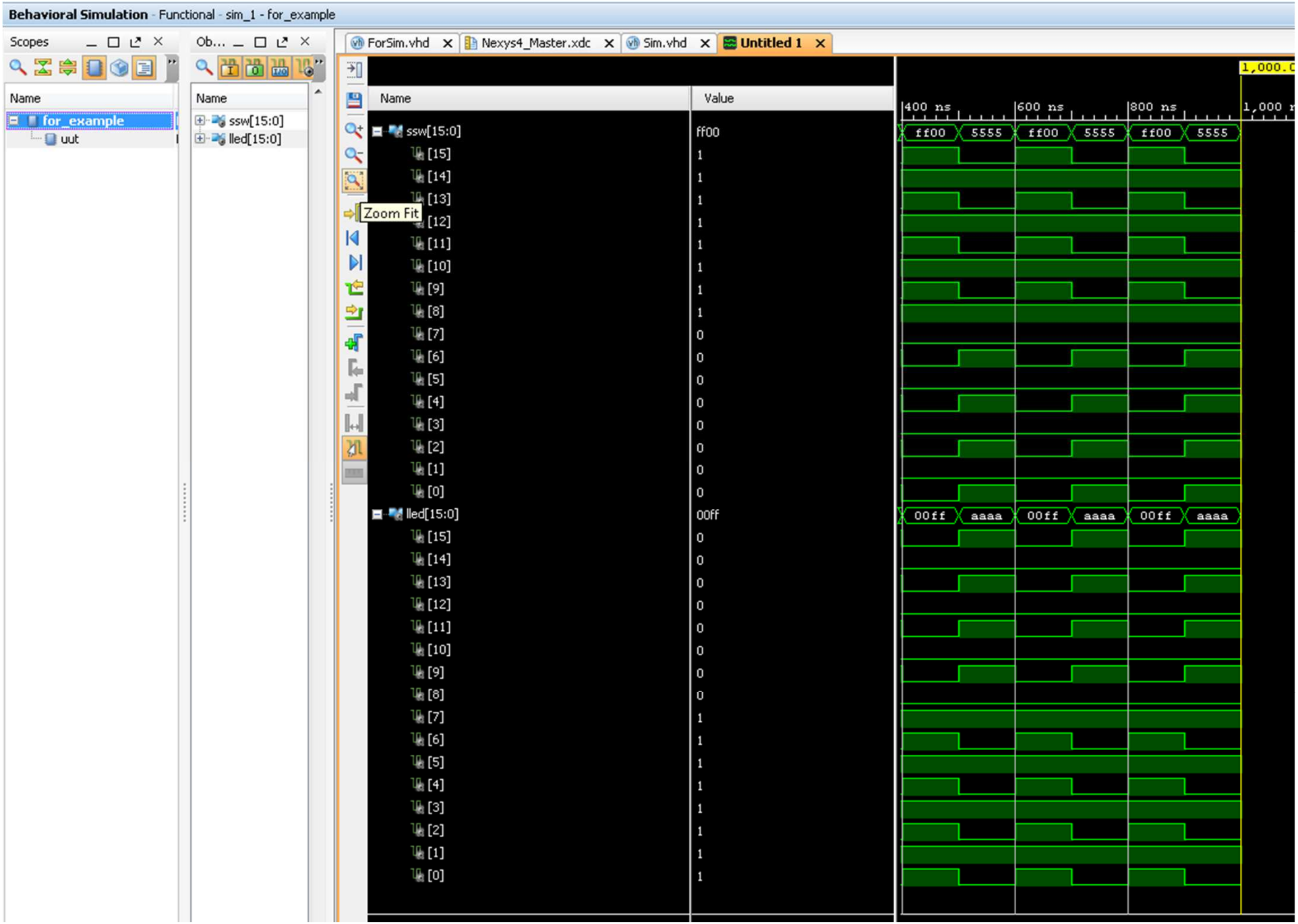

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ForSim is
    Port ( sw  :  in STD_LOGIC_VECTOR (15 downto 0);
           led : out STD_LOGIC_VECTOR (15 downto 0));
end ForSim;

architecture Behavioral of ForSim is

begin

end Behavioral;
```

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

entity ForSim is
    Port ( sw  :  in STD_LOGIC_VECTOR (15 downto 0);
           led : out STD_LOGIC_VECTOR (15 downto 0));
end ForSim;

architecture Behavioral of ForSim is

begin
            led <= not sw;
end Behavioral;
```



```
library IEEE;
use IEEE.std_logic_1164.all;
library xil_defaultlib;
                entity for_example is
                end for_example;
architecture behavior of for_example is
    signal ssw   : STD_LOGIC_VECTOR (15 downto 0);
    signal lled  : STD_LOGIC_VECTOR (15 downto 0);
 --   constant clk_period          : time := 50 ns;
 begin
 uut: entity xil_defaultlib.ForSim
                port map (sw=>ssw, led=>lled);
--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
   ssw <= "1111111100000000";            wait for 100 ns;
   ssw <= "0101010101010101";            wait for 100 ns;
end process;

end behavior;
```

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

entity ForSim is
    Port ( sw  :  in STD_LOGIC_VECTOR (15 downto 0);
           led : out STD_LOGIC_VECTOR (15 downto 0));
end ForSim;

architecture Behavioral of ForSim is

begin
                led <= not sw;
end Behavioral;
```


Behavioral Simulation - Functional - sim_1 - for_example
Scopes
Name
for_example
uut
Ob...
Name
ssw[15:0]
lled[15:0]
ForSim.vhd
Nexys4_Master.xdc
Sim.vhd
Untitled 1
Zoom Fit
Name
Value
ssw[15:0]
ff00
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
lled[15:0]
00ff
400 ns
600 ns
800 ns
ff00
5555
00ff
aaaa