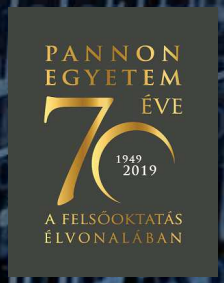
Pannon Egyetem, MIK-VIRT, Veszprém



Dr. Vörösházi Zsolt
voroshazi.zsolt@virt.uni-pannon.hu

Tervezési módszerek programozható logikai eszközökkel

5. A VHDL alapjai II.
Nyelvi szerkezetek. Konkurens és szekvenciális utasítások.

Frissítve: 2020. 10. 15.

Tárgyalt ismeretkörök
5. előadás

A VHDL alapjai II.:

- Nyelvi szerkezetek:
  - Konkurens,
  - Szekvenciális utasítások.

Általunk használt STD csomagok

Melyek szimulálhatók és szintetizálhatók is egyben:

- **LIBRARY IEEE;**
- **USE IEEE.STD_LOGIC_1164.ALL;**
  --std_logic, std_logic_vector támogatása
- **USE IEEE.NUMERIC_STD.ALL;**
  --aritmetikai operátorok támogatása unsigned, signed típusokon
- **USE IEEE.STD_LOGIC_UNSIGNED.ALL;**
  -- inkrementálás támogatása std_logic_vector típusokon

Felhasznált irodalom:

- 🕮 Pong P. Chu - FPGA Prototyping by VHDL Examples: Xilinx Spartan-3
  - http://academic.csuohio.edu/chu_p/rtl/fpga_vhdl.html
- 🕮 Hosszú Gábor - Keresztes Péter: VHDL ALAPÚ RENDSZERTERVEZÉS (2012 © Szak kiadó)
  - http://www.szak.hu/konyvek_htm/vhdl.html
- 🕮 Horváth – Harangozó - VHDL VHSIC HARDWARE DESCRIPTION LANGUAGE - BME SEGÉDLET (2006)
  - http://www.fsz.bme.hu/~tom/vhdl/vhdl_s.pdf

- 🕮 Richard E. Haskell & Darrin M. Hanna - Introduction to Digital Design VHDL (Digilent Inc.)
  - http://digilentinc.com/Data/Textbooks/Intro_Digital_Design-Digilent-VHDL_Online.pdf
- 🕮 *Real Digital - A hands-on approach to digital design* (Digilent Inc.)
  - http://www.digilentinc.com/classroom/realdigital/

IEEE 1076

VHDL Nyelvi szerkezetek

KONKURENS (EGYIDEJŰ) UTASÍTÁSOK

Konkurens (egyidejű) utasítások

- hozzárendelő (egyidejű) utasítás: „ **<=** ”
- **when-else: feltételes** jelértékadás
- **with-select (when): kiválasztó** jelértékadás

Konkurens (egyidejű) utasítások

- A VHDL architektúra **BEGIN… END** utasítások közötti részén, de egyben minden szekvenciális *process()* utasításon kívül van definiálva.
- Jelek (signal) hozzárendelése párhuzamosan, egyidejűleg történik:

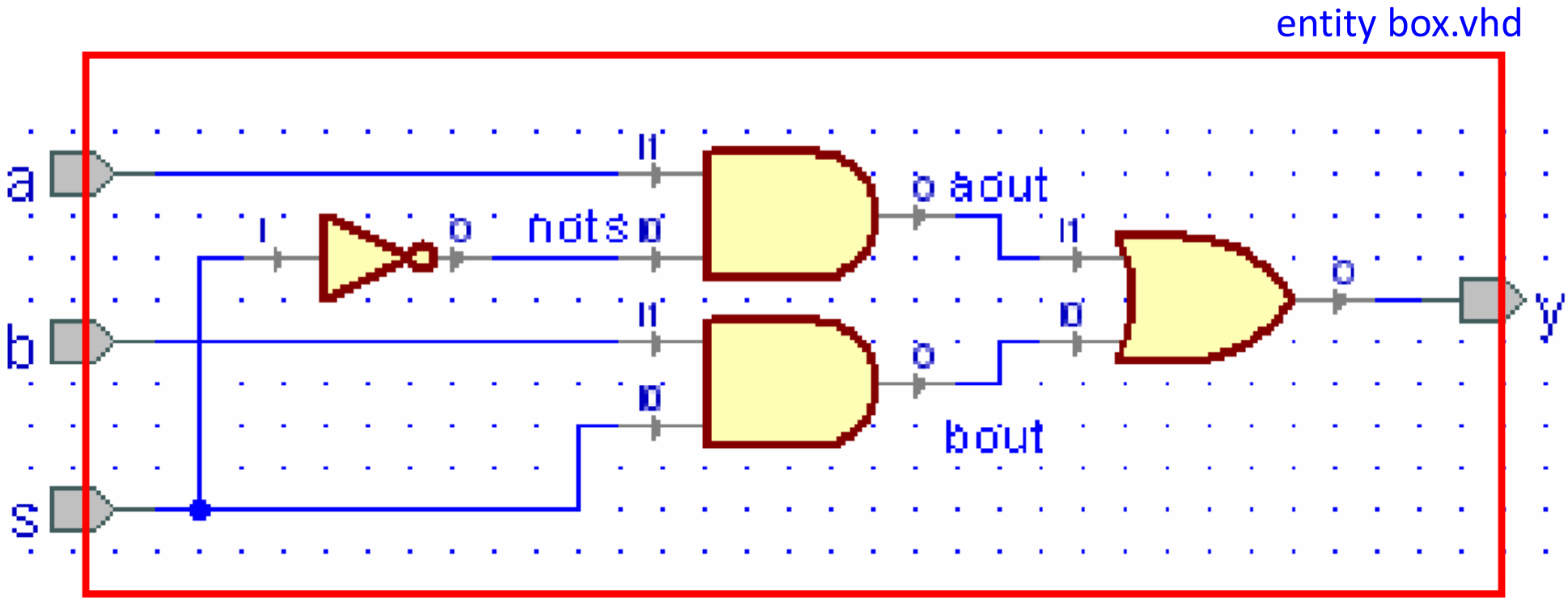
```
expl  <= "1000";
Frac1 <= '1' & sw(1) & sw(0) & "10101";
sign2 <= sw(7);
sign1 <= '0';
```

- Signal hozzárendelés: „<=”

konkurens = parallel utasítások (hagyományos nyelvektől ez különbözteti meg)

Feladat: VHDL modell készítése Xilinx Vivado segítségével

- VHDL neve: „box.vhd”
- Használjon strukturális VHDL modellt



Kérdés: Mit implementál a fenti box.vhd?

Válasz: A box.vhd egy…

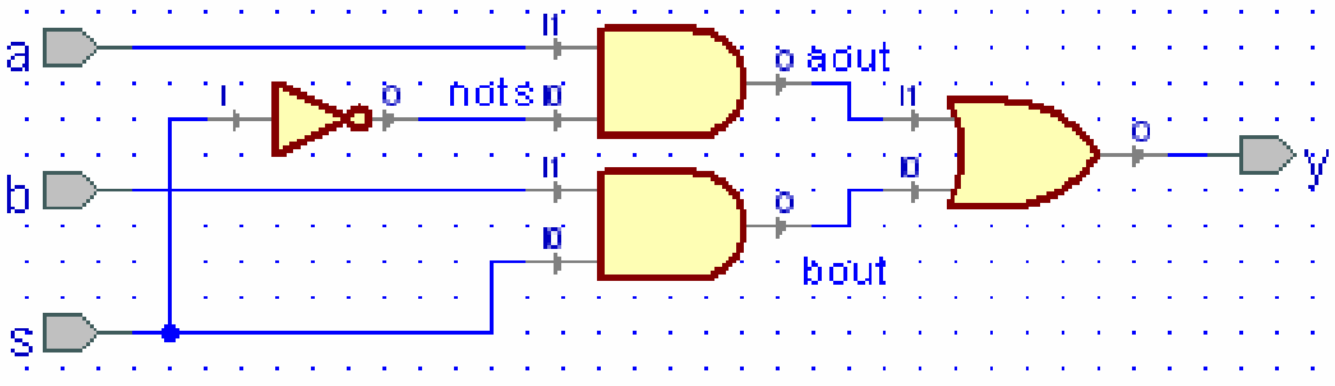
2-1 MUX: *strukturális modell*

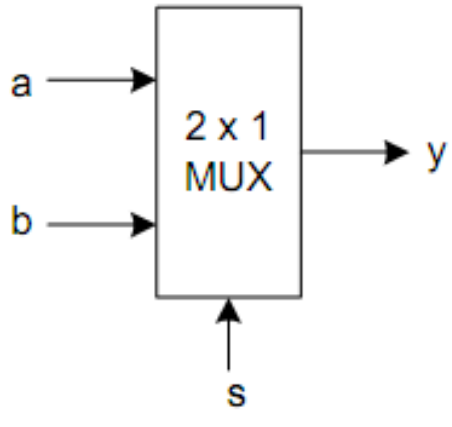
konkurens utasításokkal („ <= „)

```
-- 2-1 MUX logikai kapukkal
library IEEE;
use IEEE.std_logic_1164.all;
entity mux21 is
port(
    a : in STD_LOGIC;
    b : in STD_LOGIC;
    s : in STD_LOGIC;
    y : out STD_LOGIC
);
end mux21;
```

```
architecture arch of mux21 is
    signal aout : STD_LOGIC;
    signal bout : STD_LOGIC;
    signal nots : STD_LOGIC;
begin
    aout <= nots and a;
    bout <= s and b;
    nots <= not(s);
    y <= bout or aout;
end arch ;
```

Konkurens értékadások





4-1 MUX: *viselkedési modell*

„when…else” szerkezettel

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

entity mux4_1_when is
port (
        sel : in std_logic_vector(1 downto 0);
        i0, i1, i2, i3: in std_logic;
        y_out : out std_logic );
end entity mux4_1_when;
architecture behav of mux4_1_when is
begin
    y_out <=  i0 when sel = "00" else
              i1 when sel = "01" else
              i2 when sel = "10" else
              i3 when sel = "11" else
              'X'; --unknown value
end behav ;
```

Konkurens (egyidejű) when…else utasítás

4-1 MUX: *viselkedési modell*

„with…select” szerkezettel

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1_withsel is
port (
        sel : in std_logic_vector(1 downto 0);
        i0, i1, i2, i3: in std_logic;
        y_out : out std_logic );
end entity mux4_1_withsel;

architecture behav of mux4_withsel is
begin
        with sel select
                y_out <= i0 when "00"
                y_out <= i1 when "01"
                y_out <= i2 when "10"
                y_out <= i3 when "11" ;
end architecture behav ;
```

Konkurens (egyidejű)

With-select utasítások

4-1 MUX: *strukturális modell*

konkurens utasításokkal („ <= „)

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1_struct is
port (  sel : in std_logic_vector(1 downto 0);
        i0, i1, i2, i3: in std_logic;
        y_out : out std_logic );
end entity mux4_1_struct ;
architecture struct of mux4_1_struct is
        signal not_sel_0, not_sel_1             : std_logic;
        signal i0_int, i1_int, i2_int, i3_int : std_logic;
begin
    not_sel_0 <= not sel(0);
    not_sel_1 <= not sel(1);
    i0_int <= i0 and not_sel_1 and not_sel_0 ;    -- sel(1:0) "00"
    i1_int <= i1 and not_sel_1 and sel(0);        -- sel(1:0) "01"
    i2_int <= i2 and sel(1)    and not_sel_0 ;    -- sel(1:0) "10"
    i3_int <= i3 and sel(1)    and sel(0);        -- sel(1:0) "11"

    y_out <= i0_int or i1_int or i2_int or i3_int;
end struct;
```

Konkurens utasítások

4-1 MUX: *strukturális modell*

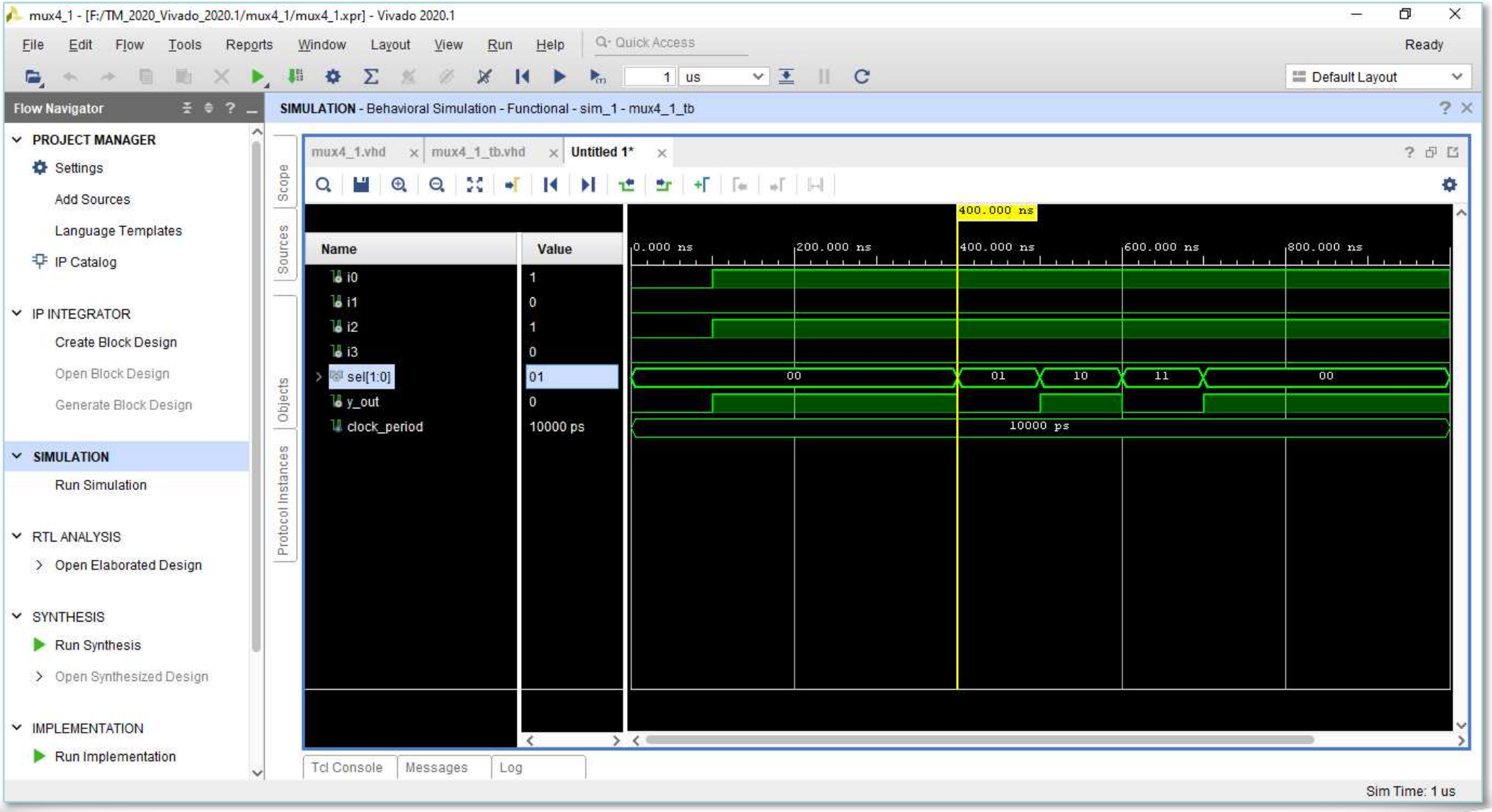
Testbench

(mux_4_1_stuct_tb.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1_struct_tb is
end entity mux4_1_struct_tb ;
...
uut: mux_4_1_struct PORT MAP (
          i0 => i0,
          i1 => i1,
          i2 => i2,
          i3 => i3,
          sel => sel,
          y_out => y_out
        );
...
begin
```

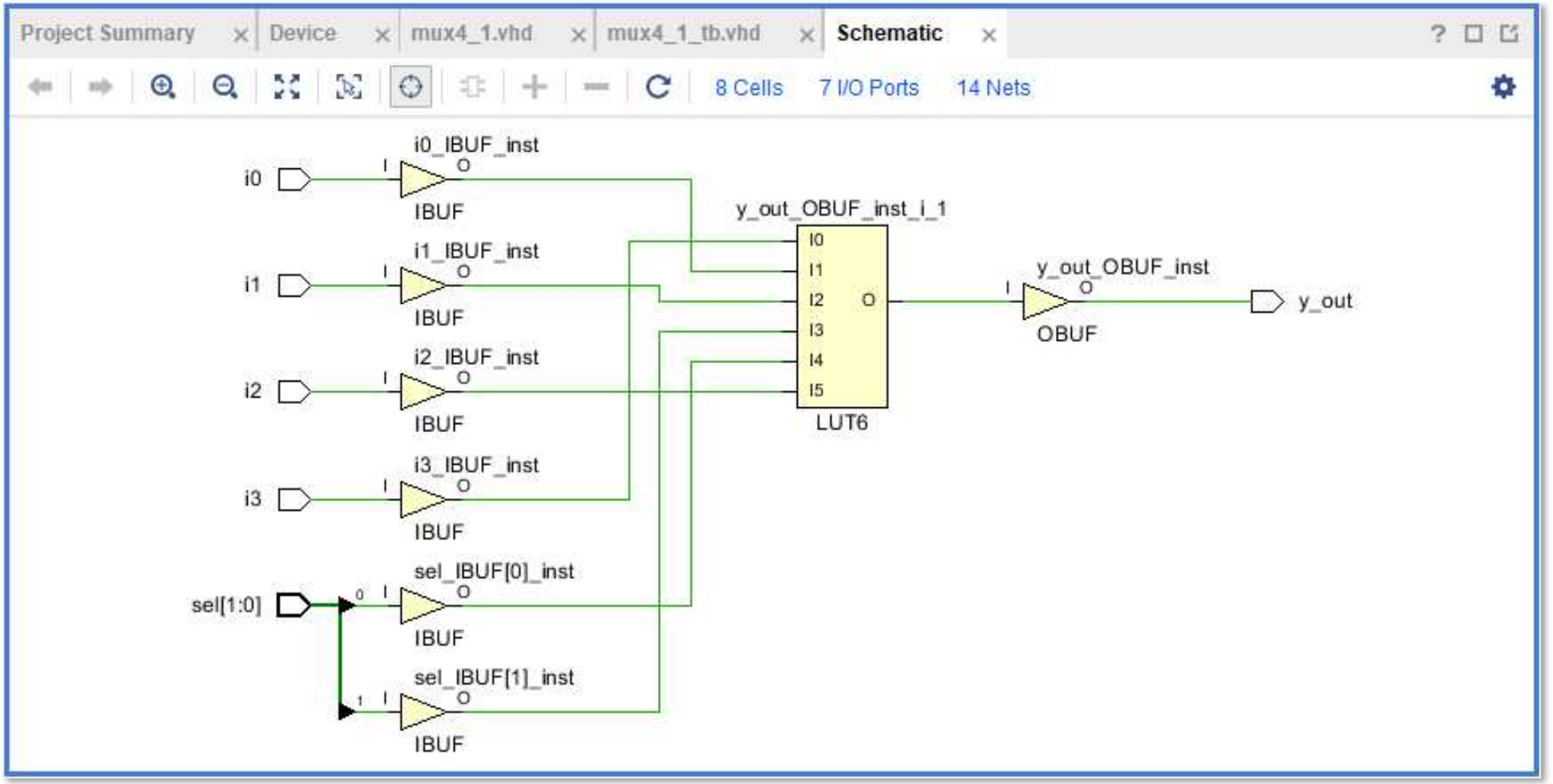
```
-- Stimulus process
stim_proc: process
begin   -- hold reset state for 100
ns.
    wait for 100 ns;
    i0 <= '1';  i2 <= '1';
    wait for clock_period*10;
    i1 <= '0'; i3 <= '0';
    wait for clock_period*10;
    sel <= "00";
    wait for clock_period*10;
    sel <= "01";
    wait for clock_period*10;
    sel <= "10";
    wait for clock_period*10;
    sel <= "11";
    wait for clock_period*10;
    sel <= "00";
    wait for clock_period*10;
    Wait;
end process;
end;
```

Vivado Simulator: szimulációs eredmény



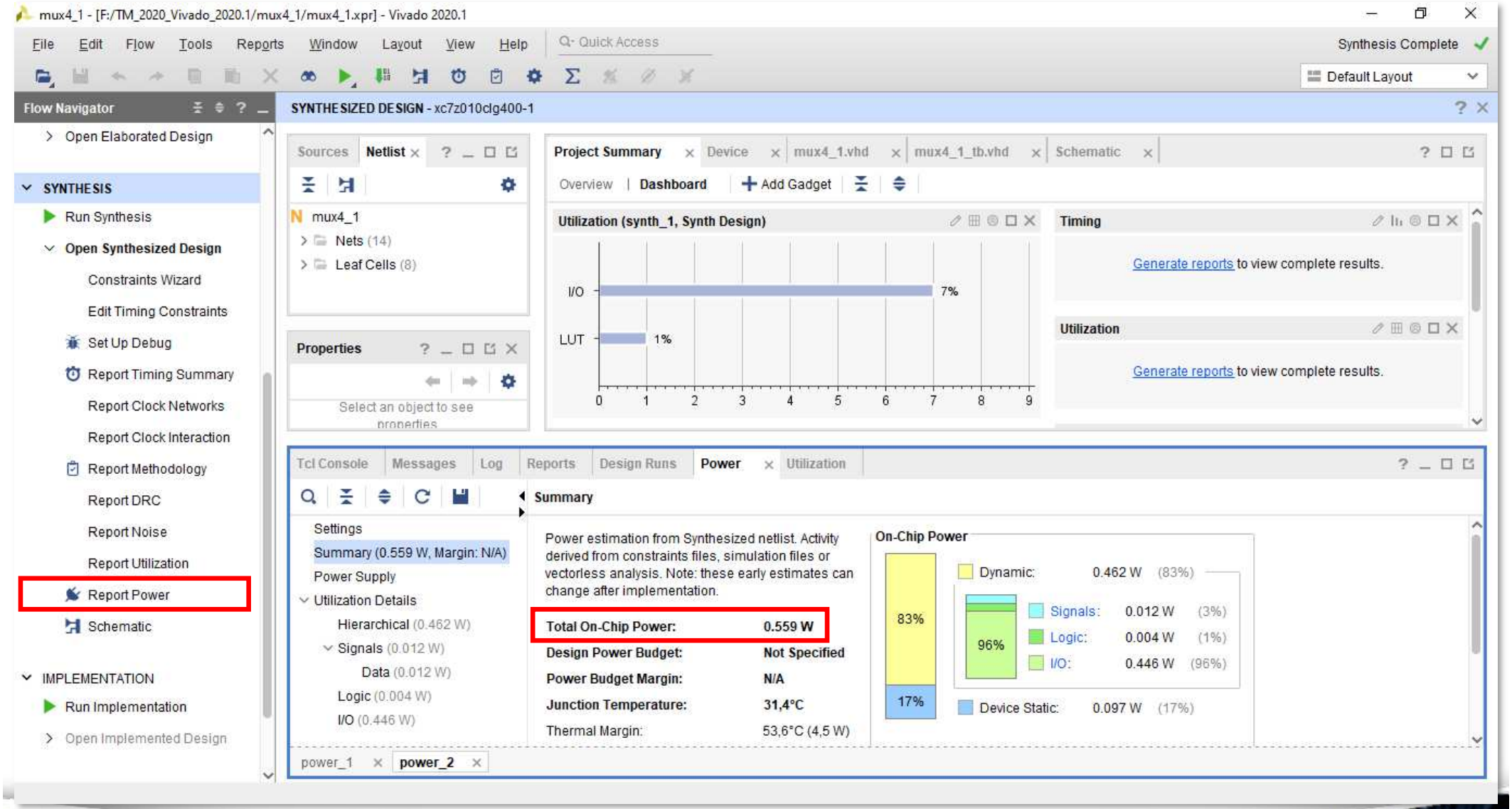
Technology schematic

Vivado → Synthesis → Open Synthesized Design → Schematic



Xilinx Vivado Power Analyzer

Vivado → Synthesis → Report Power



IEEE 1076

VHDL Nyelvi szerkezetek

SORRENDI (SZEKVENCIÁLIS) UTASÍTÁSOK

VHDL Szekvenciális utasítások

- **Process() – folyamat**
  - If … else
  - Case … when
  - **Ciklusok**
    - Loop
    - For
    - While
- **Speciális utasítások**
  - Next / Exit / Null
  - Assert / Wait

Tisztán szekvenciális utasítások (hasonlóan a legtöbb hagyományos programozási nyelvhez)

Process() - folyamat

- Önmagukban **szekvenciális** programrészek, amelyek egymással párhuzamosan futnak. Egy folyamat futását speciális utasításokkal fel is **függeszthetjük**, ilyenkor a folyamat valamilyen ***esemény*** bekövetkezéséig **várakozik**. [📖 KERESZTES_HOSSZÚ]

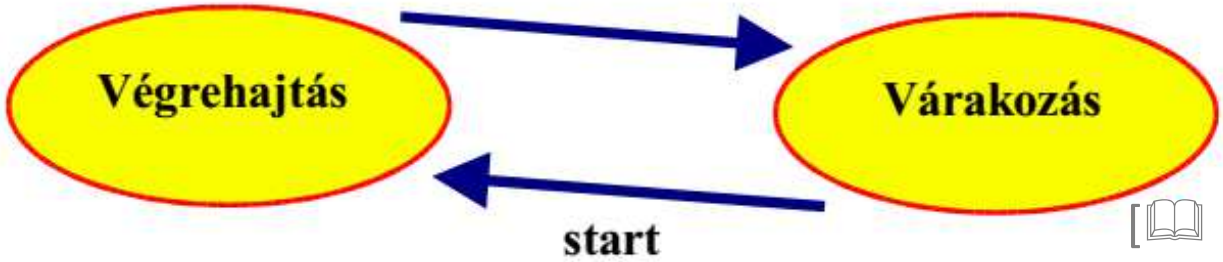
```
[process_label:]
process [(signal_name{,…}|all)] [is]
{process_declarative_item}
begin
{sequential_statements
…
…}
end process [process_label];
```

Tisztán sorrendi végrehajtás (nem egyszerre, hanem egymás után)

Process() - folyamat

- A *folyamatok* a törzsükben definiált utasítás sorozatokat *ciklikusan ismétlik*. [📖 KERESZTES_HOSSZÚ]
- A speciális **wait** várakoztató utasítás hatására azonban az utasítássorozat végrehajtása leáll, és a folyamat egészen addig *várakozik*, amíg valamilyen általunk megadott *esemény be nem következik*. Ilyen esemény lehet:
  - egy jel *értékének* megváltozása (sensitivity clause),
  - valamilyen *logikai feltétel* teljesülése (condition clause), vagy
  - megadott *időtartam* letelte (timeout clause).

Process() - folyamat



[📖 KERESZTES_HOSSZÚ]

Folyamat működése bármikor felfüggeszthető:

- Pl. feltétel bekövetkezéséig (akár végtelen hosszú ideig)

Folyamat a kezdetektől végig „él”, futása csak felfüggeszthető (lásd. wait utasítások)!

- *Aktív folyamat:* éppen végrehajtás alatt lévő, egyébként,
- felfüggesztett (várakoztatott – wait …).
- Passzív/tétlen folyamat: nem rendel értéket a jelekhez (ritkább eset).

Várakoztatások - WAIT utasítások

```
wait_utasítás ::=
Wait [ sensitivity_clause ] [ condition_clause ]
[ timeout_clause ];
```

- **[Sensitivity] wait on** – jel értékének változására várakozik
- **[Condition] wait until** – egy kifejezés értékének igazzá válására várakozik
- **[Timeout] wait for** – egy adott ideig várakozik

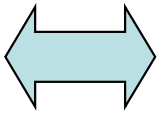
Példa:

```
wait on a, b; -- ez fejezi ki a process() érzékenységi
   listájában várakozó jeleket.

wait until clock = '1' and clock'event;

wait for 10 ns ;

wait for (a * (b+c));
```

Példa: Process() érzékenységi listák ekvivalens megadási formái

Érzékenységi lista jelekkel

```
latch_behavior :
process
begin
if clk = '1' then
   q <= d after 2 ns;
end if;
wait on clk, d;
end process
   latch_behavior;
```



```
latch_behavior : process
   (clk, d) is
begin
if clk = '1' then
   q <= d after 2 ns;
end if;
end process
   latch_behavior;
```

Példa: Process()

```
entity thermostat is
port (
   desired_temp, actual_temp : in integer;
   heater_on : out boolean );
end entity thermostat;
-----------------------------------------------
architecture example of thermostat is
begin
controller : process (desired_temp, actual_temp) is
begin
   if actual_temp < desired_temp - 2 then
       heater_on <= true;
   elsif actual_temp > desired_temp + 2 then
       heater_on <= false;
   end if;
end process controller;
end architecture example;
```

Feltételes utasítások a VHDL-ben

- A.) **If … Else szerkezet,**
- B.) **Case utasítás**

A.) If - feltételes utasítás

```
[if_label:] if boolean_expression
  then
      {sequential_statements}
{elsif boolean_expression then
      {sequential_statements}}
{else
      {sequential_statements}}
end if; [if_label];
```

Megjegyzés: elsif /= else if!!!

Példa: If utasítás

```
if en = '1' then
  stored_value := data_in; :
end if;
```

```
if sel = 0 then
  result <= input_0; -- executed if sel = 0
else
  result <= input_1; -- executed if sel /= 0
end if;
```

If utasítás vs. konkurens értékadás

```
process (en, sel)
  if en = '0' then
    s <= 'Z';
  elsif sel = '0' then
    s <= '0' after 10 ns;
  else
    s <= '1' after 10 ns;
  end if;
end process;
```

Vs.

```
s <= 'Z' when en = '0' else
     '0'  after 10 ns when sel = '0' else
     '1'  after 10 ns ;
```

Case utasítás vs. konkurens értékadás

```
process
  case alu_function is
    when alu_add | alu_incr =>
       alu_result <= op1 + op2;
    when alu_sub =>
       alu_result <= op1 - op2;
    when alu_and =>
       alu_result <= op1 and op2;
    when alu_or =>
       alu_result <= op1 or op2;
  end case;
  wait on alu_function;
end process;
```

Vs.

```
with alu_function select
   alu_result <= op1 + op2 when alu_add | alu_incr,
                 op1 - op2 when alu_sub,
                 op1 and op2 when alu_and,
                 op1 or op2 when alu_or;
```

Példa: If utasítás

```
type mode_type is (immediate, other_mode);
type opcode_type is (load, add, subtract,
  other_opcode);

if mode = immediate then
  operand := immed_operand;
elsif opcode = load or opcode = add or opcode
  = subtract then
  operand := memory_operand;
else
  operand := address_operand;
end if;
```

4:1 Multiplexer - If utasítással

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_if is
port (
    sel : in std_logic_vector(1 downto 0);
    i0, i1, i2, i3 : in std_logic;
    y_out : out std_logic );
end entity mux4_if;
-------------------------------------------------
architecture behaviour of mux4_if is
begin
    mux_select : process (sel, i0, i1, i2, i3) is
    begin
        if sel = "00" then
            y_out <= i0;
        elsif sel = "01" then
            y_out <= i1;
        elsif sel = "10" then
            y_out <= i2;
        else
            y_out <= i3;
        end if;
    end process mux_select;
end architecture behaviour;
```

B.) Case – feltételes utasítás

```
[case_label:] case expression is
(when choices =>
  {sequential_statements})
{...}
end case [case_label];
```

```
choices <= (simple_expression |
  discrete_range | others){ | ...}
```

Példa: Case feltételes utasítás

```
type alu_func is (pass1, pass2, add, subtract);

case func is
    when pass1 =>
        result := operand1;
    when pass2 =>
      result := operand2;
    when add =>
      result := operand1 + operand2;
    when subtract =>
      result := operand1 - operand2;
    [ when others =>
      result := null;]
end case;
```

Összes lehetséges esetet listázni kell = **all inclusive**, de **mutually exclusive**! (azaz minden esetet fel kell sorolni, de közöttük kölcsönös kizárás van!).

Példa: Case feltételes utasítás

```
type opcodes is (nop, add, subtract, load, store,
   jump, jumpsub, branch, halt);

subtype control_transfer_opcodes is opcodes range jump to
   branch;

variable opcode : opcodes;

case opcode is
when load | add | subtract =>
   operand := memory_operand;
when store | jump | jumpsub | branch =>
   operand := address_operand;
when others =>
   operand := 0;
end case;
```

Példa: 4:1 Multiplexer Case utasítással

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_case is
port (
    sel : in std_logic_vector(1 downto 0);
    i0, i1, i2, i3 : in std_logic;
    y_out : out std_logic );
end entity mux4_case;

architecture behav of mux4_case is
begin
     mux_select : process (sel, i0, i1, i2, i3) is
     begin
          case sel is
            when "00" =>
                   y_out <= i0;
            when "01" =>
                   y_out <= i1;
            when "10" =>
                   y_out <= i2;
            when others =>
                   y_out <= i3;
          end case;
     end process mux_select;
end architecture behav ;
```

Null utasítás

```
type opcode_type is (nop, add, subtract);

case opcode is
when add =>
  Acc := Acc + operand;
when subtract =>
  Acc := Acc - operand;
when nop =>
  null;   -- semmilyen hatása nincsen
end case;
```

Ciklusok a VHDL-ben

- A.) **Loop**: feltétel nélküli ciklus
- B.) **While**: addig hajtódik végre, amíg a feltétel IGAZ (hamissá nem válik)
- C.) **For**: adott számú iterációt hajt végre, amíg a feltétel IGAZ (hamissá nem válik)

A.) Loop (ciklus)

```
[loop_label:]
loop
  {sequential_statement(s)}
end loop [loop_label];
```

Példa: Loop utasítás

```
entity counter is
port
   ( clk : in std_logic; count : out natural );
end entity counter;
-------------------------------------------------
architecture behavior of counter is
begin
   incrementer : process is
        variable count_value : natural := 0;
   begin
        count <= count_value;
        loop
        wait until clk = '1';
                count_value := (count_value + 1) mod 16;
                count <= count_value;
        end loop;
   end process incrementer;
end architecture behavior;
```

B.) While ciklus

```
[loop_label:]
while boolean_expression loop
  {sequential_statement(s)}
end loop [loop_label:];

while index > 0 loop
  --... -- utasítás A: index értékét változtatjuk
end loop;
--... - utasítás B
```

C.) For ciklus

```
[loop_label:]
for identifier in discrete_range loop
  {sequential_statement(s)}
end loop [loop_label];
```

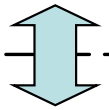
```
for count_value in 0 to 127 loop
  count_out <= count_value;    --after 5 ns
  wait for 5 ns;
end loop;
```

Példa: For ciklus

```
for i in 10 to 1 loop
  --...ez sosem fog végrehajtódni!!
end loop;

for i in 10 downto 1 loop
  --...
end loop; --OK: i--

for i in 1 to 10 loop
  --...
end loop; --OK: i++
```

EXIT utasítás: kilépés loop-ból

```
loop
   if condition then
      exit;
  end if;
end loop;
```



```
loop
   exit when condition;
end loop;
```

Exit: Process() folytatótdik az exit utáni utasításokkal (**kilép az aktuális ciklusból**)

Egymásba ágyazott loop-ok: Exit utasítással

```
outer : loop
   -- . . .
   inner : loop
        -- . . .
        exit outer when condition1; --kilépés outer loop-ból
        -- . . .
        exit when condition2; --kilépés inner loop-ból
        -- . . .
   end loop inner;
   -- . . . - utasítások A
   exit outer when condition3; --kilépés outer loop-ból
   -- . . .
end loop outer;
-- . . . - utasítások B
```

NEXT utasítás

```
[label:] next [loop_label][when
  boolean_expression];
```

```
next;   --feltétel és loop_címke nélkül

next when condition;

next loop_label;

next loop_label when condition;
```

Next: lehetővé teszi az aktuális iteráció végrehajtásának leállítását, és egyben a következő iterációra ugrik.

NEXT utasítás

```
loop
   -- statement_1;
next when condition;
   -- statement_2;
end loop;
```

```
loop
   -- statement_1;
   if not condition then
   -- statement_2;
    end if;
end loop;
```

Assertion és report utasítások

```
[label:] assert boolean_expression [report
  expression][severity expression];
```

Amikor *boolean_expression* **hamis** a *report_expression* (karakterfüzér) kerül megjelenítésre,

ha pedig **igaz** a riport kifejezés nem jelenik meg (a szimulátorban).
A riport a szimulátor üzenet ablakában látható!
Állhat szekvenciális és már konkurens* utasításrészen (is).

```
type severity_level is (note, warning, error,
  failure);
assert initial_value <= max_value;
```

*Assert a Vivado 2015.3 XSIM-ben: http://www.xilinx.com/support/answers/64138.html

Példa: Assertion és report utasítás

```
assert packet_length /= 0 -- =0
  report "empty network packet received"
  severity warning;

assert clock_pulse_width >= min_clock_width
  severity error; -- cpw < mcw

assert (last_position - first_position + 1)
  = number_of_entries
  report "inconsistency in buffer model"
  severity failure;
```

Példa: Assertion és report utasítás

```
assert initial_value <= max_value --init_v >
  max_v
report "initial value too large";

assert current_character >= '0' and
  current_character <= '9'     --
  current_char<>{0…9}
report "Input number " & input_string &
  "contains a non-digit";

assert free_memory >= low_water_limit
report "low on memory, about to start
  garbage collect"
severity note;
```

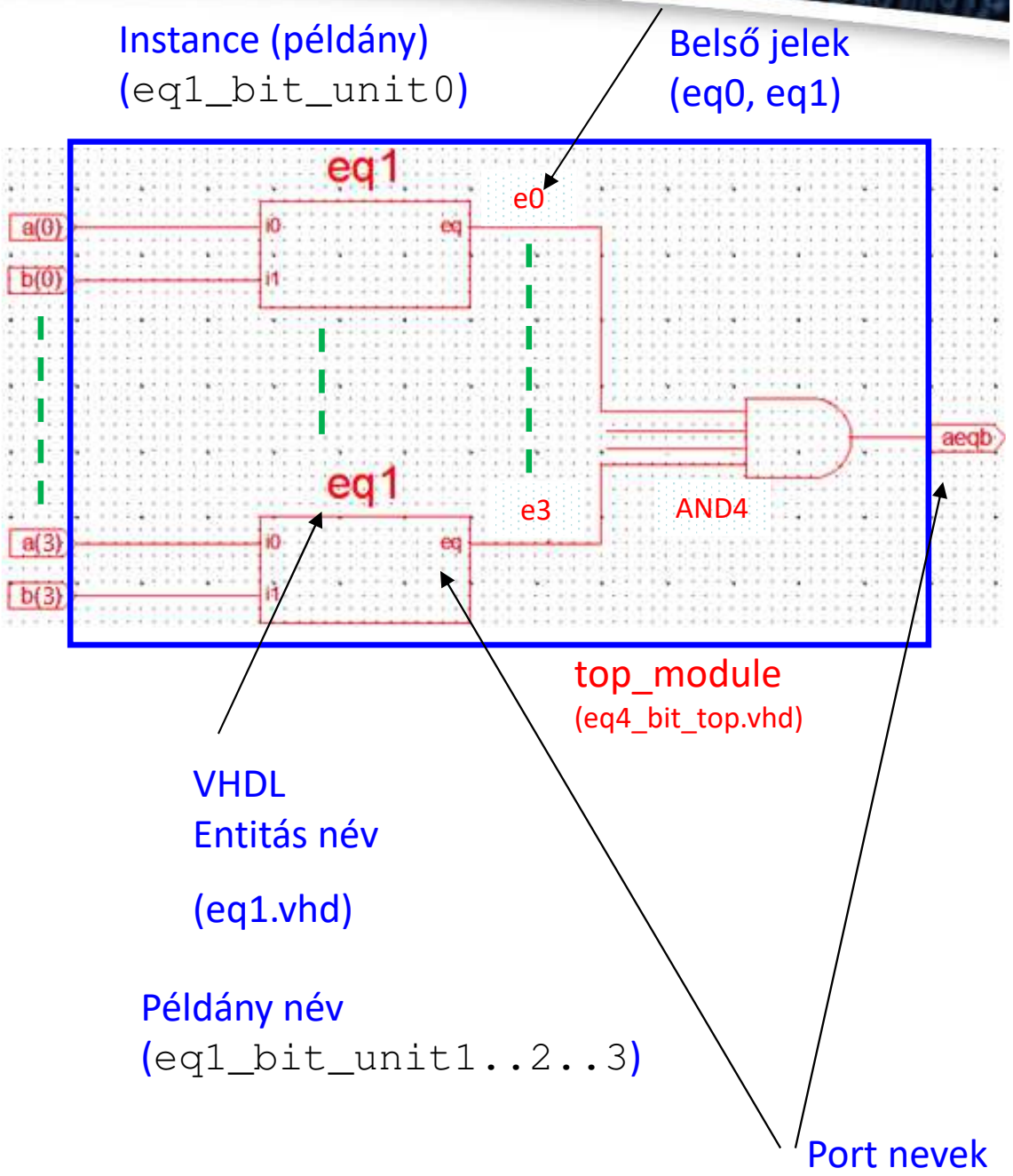
Példa: assert, report

```
entity edge_triggered_register is
port (   clock : in std_logic;
         d_in : in real;       d_out : out real );
end entity edge_triggered_register;
-------------------------------------------------
architecture check_timing of edge_triggered_register is
begin
    store_and_check : process (clock) is
      variable stored_value : real;
      variable pulse_start : time;
    begin
    case clock is
    when '1' =>
      pulse_start := now;
      stored_value := d_in;
      d_out <= stored_value;
    when '0' =>
      assert now = 0 ns or (now - pulse_start) >= 5 ns
      report "clock pulse too short„
      severity warning;
    end case;
    end process store_and_check;
end architecture check_timing;
```

Feladat 1.)

Tervezzen egy **4-bit egyenlőség komparátort (**eq4_bit_top.vhd) a mellékelt ábra alapján.

- Megj: „eq1.vhd” entitásnál használja az EQ (xnor) operátort
- Terv megvalósítása történhet
  a.) *konkurens* vagy
  b.) *szekvenciális* nyelvi szerkezetekkel
- Példányosítsa az „eq1” VHDL modult 4-szer, és kösse össze őket AND kapu segítségével a „top-level” modul szinten!
- Készítsen egy testbench-et!
- Szimulálja le a viselkedését (Xilinx XSim)!
- Implementálja FPGA-n a terveket (4-4 kapcsoló/nyomógomb= a(i), és b(i) bemenetre, ill. 1 LED-et a kimenet megjelenítésére!
- Használja a ZyBo master.xdc filet!



Megoldás 1./a.) „eq1” VHDL modul

konkurens utasításokkal

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

entity eq1 is
port ( i0, i1: in std_logic;
       eq: out std_logic);
end eq1;

architecture struct of eq1 is
begin
   eq <= i0 xnor i1;
end struct;
```

Konkurens utasítás(ok)

Megoldás 1./a) VHDL „eq4bit_top”

(top-level)

```
library ieee;
use ieee.std_logic_1164.all;
entity eq4bit_top is
port ( a, b: in std_logic_vector (3 downto 0);
       aeqb : out std_logic);
end eq4bit_top;

architecture structural of eq4bit_top is
   component eq1
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           eq : out STD_LOGIC);
   end component;
   signal e0, e1, e2, e3 : std_logic := '0';
begin
   eq1_bit_unit0 : entity work.eq1(struct)
   port map (i0=>a(0), i1=>b(0), eq=>e0);
   eq1_bit_unit1 : entity work.eq1(struct)
   port map (i0=>a(1), i1=>b(1), eq=>e1);
   …
   eq1_bit_unit3 : entity work.eq1(struct)
   port map (i0=>a(3), i1=>b(3), eq=>e3);
   aeqb <= e0 and e1 and e2 and e3;
end structural;
```

Entitások példányosítása port hozzárendeléssel (map)

Simulation result:

Megoldás 1.) VHDL „eq4bit_top_tb”

VHDL testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;use
IEEE.STD_LOGIC_unsigned.ALL; -- + incremention
entity eq4bit_top_tb is
--  Port ( );
end eq4bit_top_tb;

architecture Behavioral of eq4bit_top_tb is

   COMPONENT eq4bit_top
   port ( a, b: in std_logic_vector (3 downto 0);
         aeqb : out std_logic);
   END COMPONENT;

  signal a_in : std_logic_vector(3 downto 0) := (others => '0');
  signal b_in : std_logic_vector(3 downto 0) := (others => '0');
  signal aeqb_res : std_logic;

begin
-- Instantiate the Unit Under Test (UUT)
  uut: eq4bit_top PORT MAP (
        a => a_in,
        b => b_in,
        aeqb => aeqb_res
      );

-- Stimulus process
   stim_proc_for: process
      begin
         wait for 100 ns;
         a_in <= "0000";
         b_in <= "1111";
         wait for 20 ns;

         for index in 0 to (2**3 - 1) loop
         -- vagy (2**(a_in'lenght) -1)
            a_in <= a_in + 1;
            wait for 20 ns;
            b_in <= b_in - 1;
            wait for 20 ns;
         end loop;
         wait;
      end process stim_proc_for;

  end Behavioral;
```

Megj.: a gerjesztési teszt vektorok ***for*** ciklus segítségével lettek generálva.

Feladat 2.)

Tervezzen egy **4-bit ún. „nagyságrend” komparátort** (név: „`magncomp_4_bit.vhd`”) entitás

Lehet alkalmazni a

a.) *konkurrens* vagy,

b.) *szekvenciális* hozzárendelő utasításokat ,

c.) korábban még a Xilinx ISE Schematic Editor-t.

- Készítsen egy testbench-et VHDL-ben (`magncomp_4_bit_tb.vhd`)
- Szimulálja le a viselkedését Xilinx XSim-ben
- Implementálja a tervelek FPGA-n (a 4-4 kapcsolóra-nyomógombra rendre az a(i), és b(i), bemeneteket kötve). V
- Végül használjon 3 db LED-et ún. gt_out, eq_out, lt_out eredmények megjelenítésére, amennyiben:

- gt_out: `a_in` „greather than” `b_in`
- eq_out: `a_in` „equal with” `b_in`
- lt_out: `a_in` „less than” `b_in`

Feladat 2./a.) Megoldás
4-bit komparátor (pl. if..else)

```
entity magncomp_4_bit_seq is
    Port ( a_in : in  STD_LOGIC_VECTOR (3 downto 0);
           b_in : in  STD_LOGIC_VECTOR (3 downto 0);
           gt_out : out  STD_LOGIC;
           lt_out : out  STD_LOGIC;
           eq_out : out  STD_LOGIC);
end magncomp_4_bit_seq;

architecture Behavioral of magncomp_4_bit_seq is
begin
process (a_in, b_in) is        -- bemeneti jelekre érzékeny process()
begin
        if a_in = b_in then
                gt_out <= '0'; lt_out <= '0'; eq_out <= '1';
        elsif a_in > b_in then
                gt_out <= '1'; lt_out <= '0'; eq_out <= '0';
        else
                gt_out <= '0'; lt_out <= '1'; eq_out <= '0';
        end if;
end process;
end Behavioral;
```