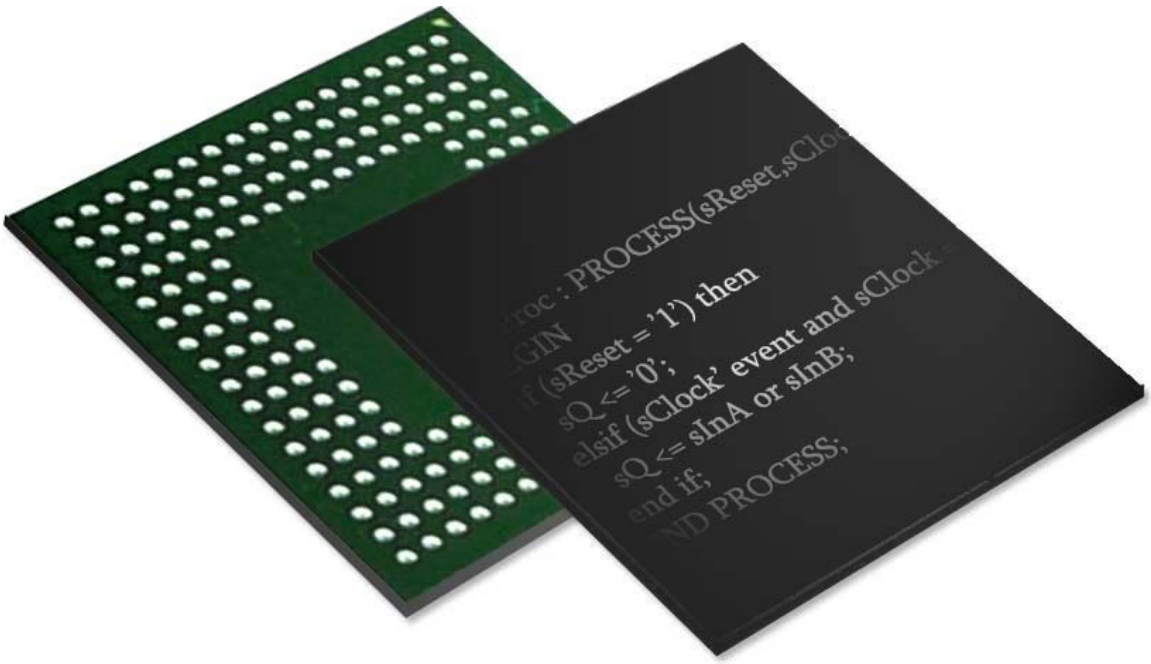


VHDL alapismeretek

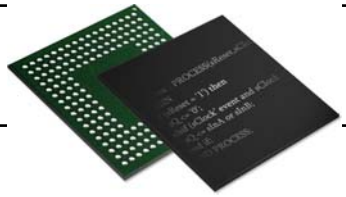
(Xilinx FPGA-k programozása VHDL nyelven)

Oktatási jegyzet

Összeállította: dr. Zigó Tamás

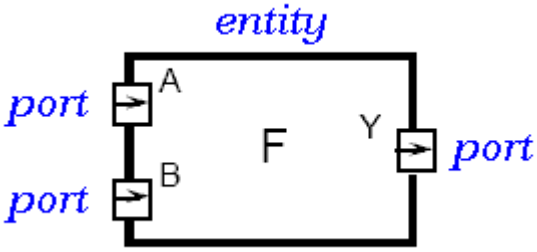
E-mail: zigotamas@bytestudio.hu
www.bytestudio.hu

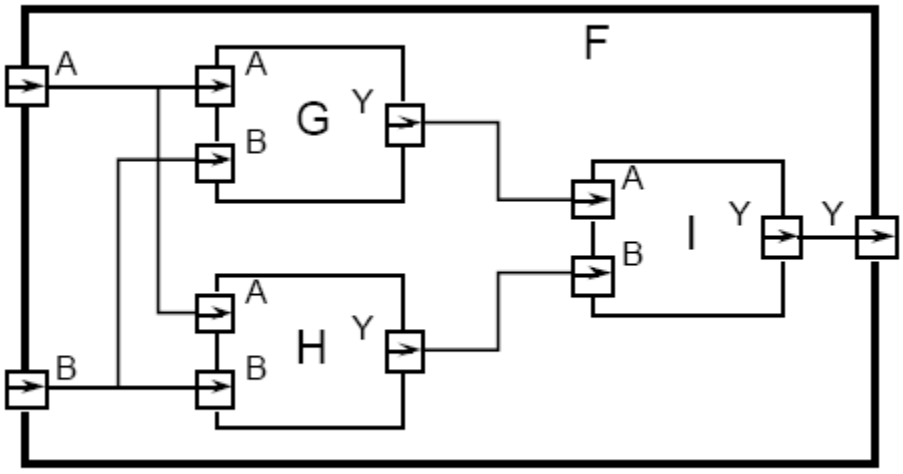
ByteStudio Bt. 2010. 10. 07.



VHSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage

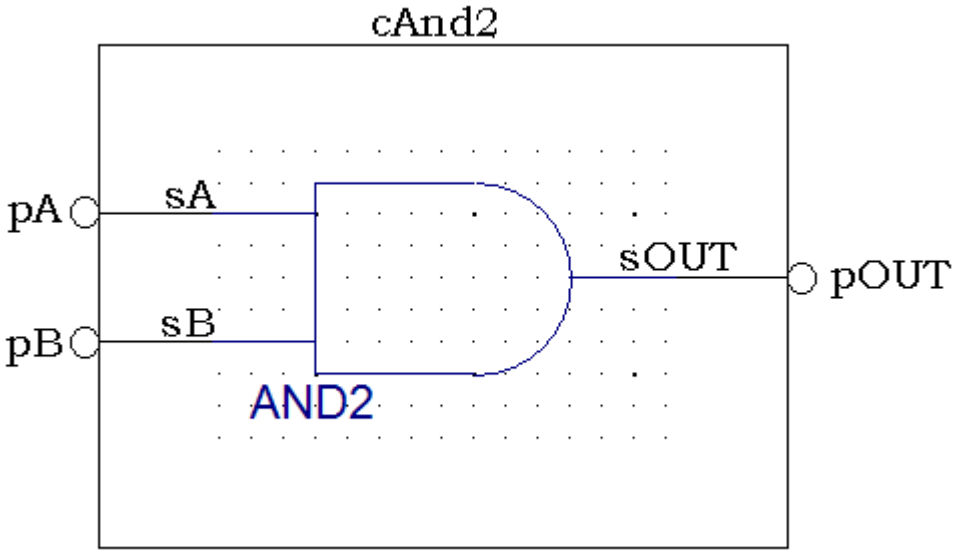
A digitális rendszerek modelljei





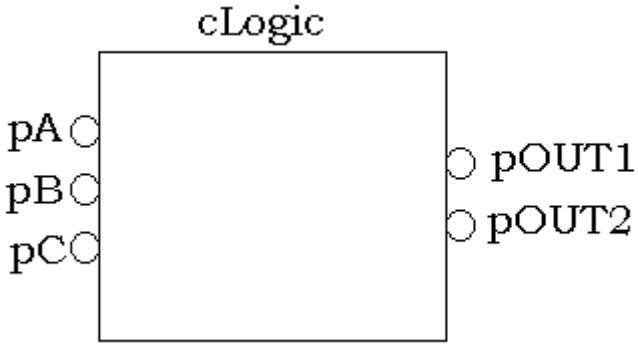
cAnd2.vhd

Egyetlen ÉS kaput tartalmazó modul (*entity*) viselkedésének leírása VHDL nyelven




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

entity cAnd2 is
  Port ( pA   : in  STD_LOGIC;
         pB   : in  STD_LOGIC;
         pOUT : out STD_LOGIC);
end cAnd2;

architecture Behavioral of cAnd2 is

signal sA,sB,sOUT : STD_LOGIC;

begin

 sA <= pA;
 sB <= pB;
 sOUT <= sB and sA;
 pOUT <= sOUT;

end Behavioral;
```

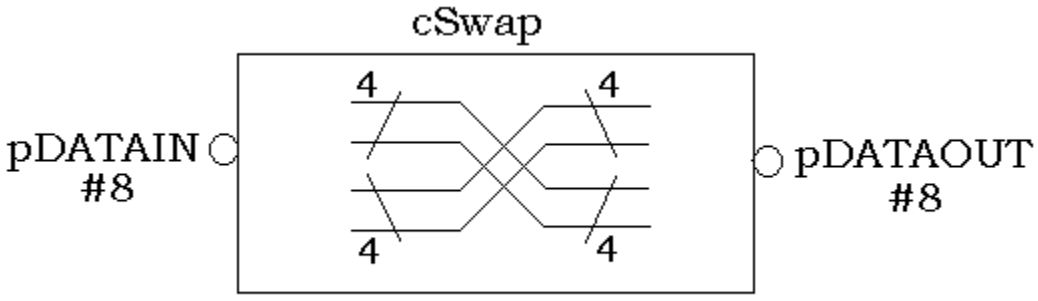
Összetettebb kombinációs hálózat




```
entity cLogic is
    Port ( pA    : in  STD_LOGIC;
           pB    : in  STD_LOGIC;
           pC    : in  STD_LOGIC;
           pOUT1 : out STD_LOGIC;
           pOUT2 : out STD_LOGIC);
end cLogic;

architecture Behavioral of cLogic is

signal sA,sB,sC    : STD_LOGIC;
signal sOUT1,sOUT2 : STD_LOGIC;

begin

 -- Inputs
 sA <= pA;
 sB <= pB;
 sC <= pC;

 -- Bitwise operators
 sOUT1 <= (sA or sB) xor sC;
 sOUT2 <= (not sOUT1) and sC;

 -- Outputs
 pOUT1 <= sOUT1;
 pOUT2 <= sOUT2;

end Behavioral;
```

cSwap.vhd

Több bites portok (buszok)




```
entity cSwap is
    Port ( pDATAIN  : in  STD_LOGIC_VECTOR(7 downto 0);
           pDATAOUT : out STD_LOGIC_VECTOR(7 downto 0));
end cSwap;

architecture Behavioral of cSwap is

signal sDATAIN,sDATAOUT : STD_LOGIC_VECTOR (7 downto 0);

begin

 sDATAIN <= pDATAIN;

 sDATAOUT(3 downto 0) <= sDATAIN(7 downto 4);
 sDATAOUT(7 downto 4) <= sDATAIN(3 downto 0);

 pDATAOUT <= sDATAOUT;

end Behavioral;
```

Előre definiált típusok (IEEE 1164)

```
STD_LOGIC              Értékek ('0','1','Z')
STD_LOGIC_VECTOR
```

Port deklaráció

Bemenet:
```
<port_name> : in STD_LOGIC;
<port_name> : in STD_LOGIC_VECTOR(7 downto 0);
```

Kimenet:
```
<port_name> : out STD_LOGIC;
<port_name> : out STD_LOGIC_VECTOR(4 downto 0);
```

Signal (vezeték) deklaráció

Általánosan: **signal** <name>: <type>

Pl. :
```
signal sLOAD : STD_LOGIC;
signal sA1, sA2, sA3 : STD_LOGIC;
signal sDATAIN : STD_LOGIC_VECTOR(7 downto 0);
signal sBUS1, sBUS2 : STD_LOGIC_VECTOR(1 downto 0);
```

Értékadás

Általánosan: <name> <= <value>

Pl:
```
sLOAD <= '0';
sA1 <= '1';
sA3 <= sA1;
sDATAIN <= "01011011";
sDATAIN <= X"5B";
sDATAIN(3 downto 0) <= "1100";
sDATAIN(7 downto 4) <= X"5";
sBUS2 <= sDATAIN(7 downto 6);
sDATAIN(6 downto 5) <= sBUS2;
sDATAIN(2) <= sLOAD;
```

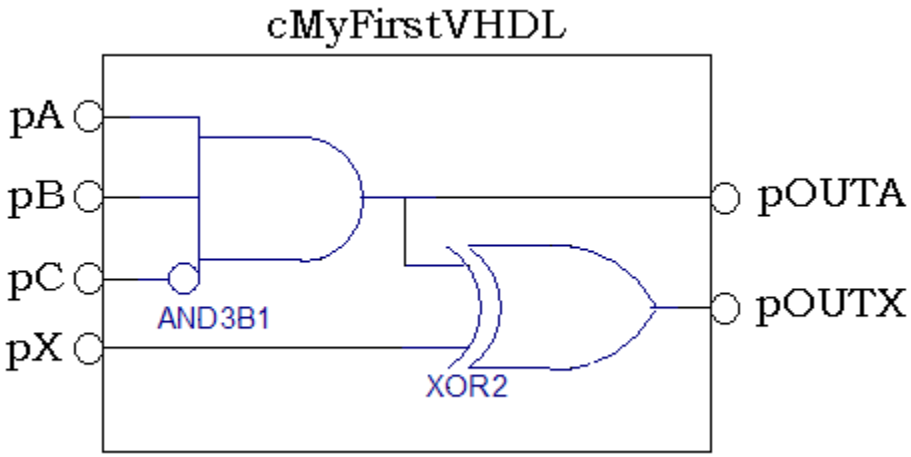
Bitenkénti operátorok

not, and, or, xor, nor, nand

Pl. `sA1 <= (sA2 and sA3) or (not sLOAD)`

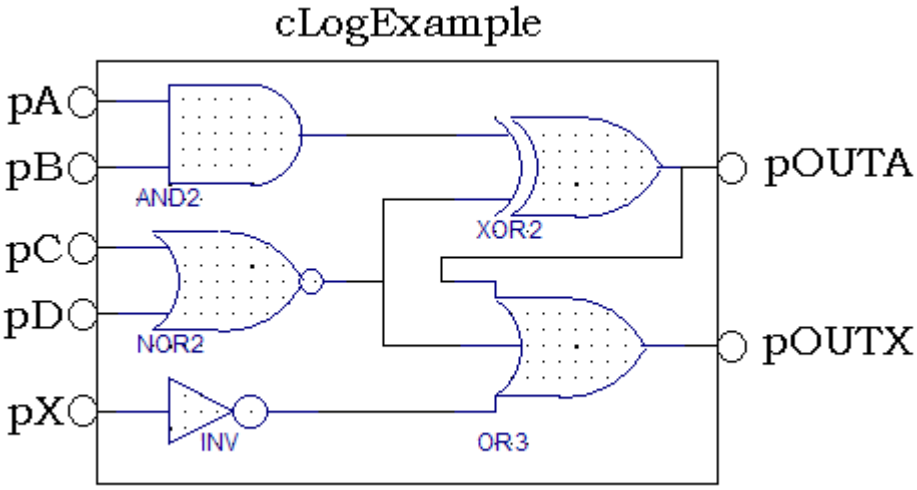
Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven!

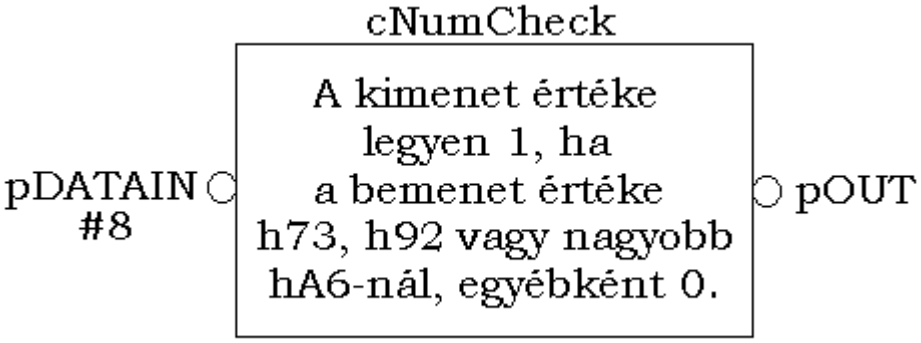


Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven!



Az *if* utasítás



Egyszerű értékadással:

```
 sOUT <= (not sDATAIN(7) and sDATAIN(6)…) or (sDATAIN(7)
          and …) or (…)
```

```
entity cNumCheck is
    Port ( pDATAIN : in  STD_LOGIC_VECTOR (7 downto 0);
           pOUT    : out STD_LOGIC);
end cNumCheck;

architecture Behavioral of cNumCheck is

signal sDATAIN : STD_LOGIC_VECTOR (7 downto 0);
signal sOUT    : STD_LOGIC;

begin

 sDATAIN <= pDATAIN;
 pOUT    <= sOUT;

 process (sDATAIN)
 begin
   if (sDATAIN = X"73") or (sDATAIN = X"92") or (sDATAIN > X"A6") then
    sOUT <= '1';
   else
    sOUT <= '0';
   end if;
 end process;

end Behavioral;
```

```
if <condition> then
  <statement>
else
  <statement>
end if;

if <condition> then
  <statement>
else
 if <condition> then
    <statement>
 else
    <statement>
 end if;
end if;

if <condition> then
  <statement>
elsif <condition> then
  <statement>
else
  <statement>
end if;
```

A feltétel (*condition*) egy logikai állítás/kifejezés, értéke lehet igaz vagy hamis.

Példák:
```
if (sLOAD = '1') then …
if sDATAIN /= X"73" then …
if ((sDATAIN >= X"71") and (sDATAIN(2) = '0')) then …
if (sDATAIN(2 downto 0) = "101") then …
```

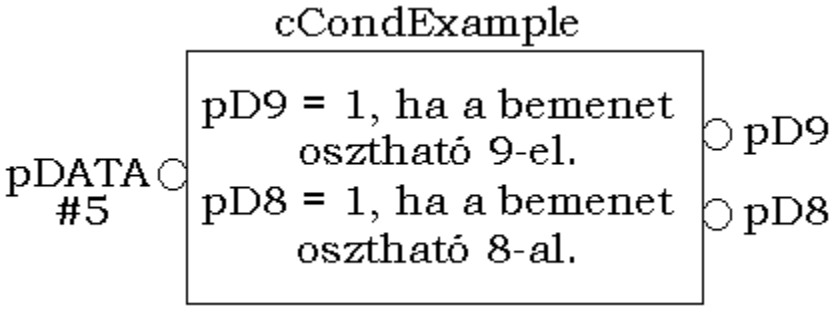
A VHDL nyelv logikai és relációs operátorai

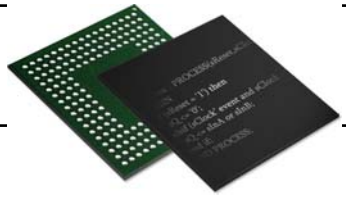
```
not -- Negálás
and -- Logikai ÉS
or  -- Logikai VAGY
=   -- Egyenlő
/=  -- Nem egyenlő
<   -- Kisebb
<=  -- Kisebb vagy egyenlő
>   -- Nagyobb
>=  -- Nagyobb vagy egyenlő
```

cCondExample.vhd

Feladat

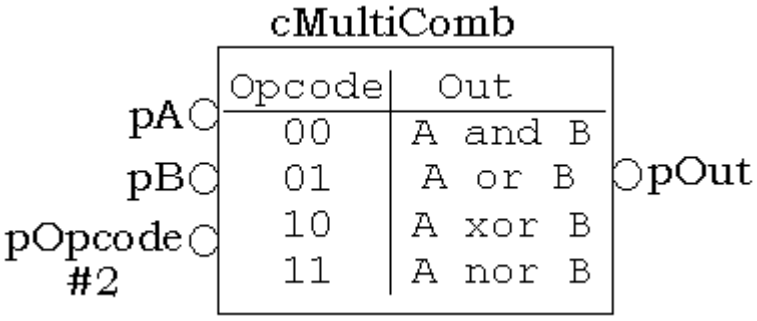
Írja meg az alábbi ábrán látható modult VHDL nyelven!





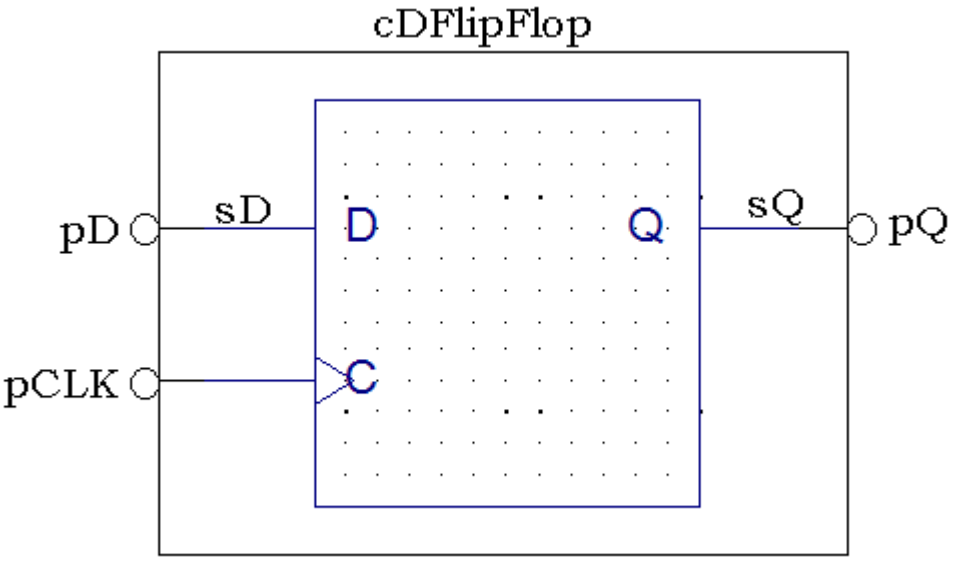
Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven!

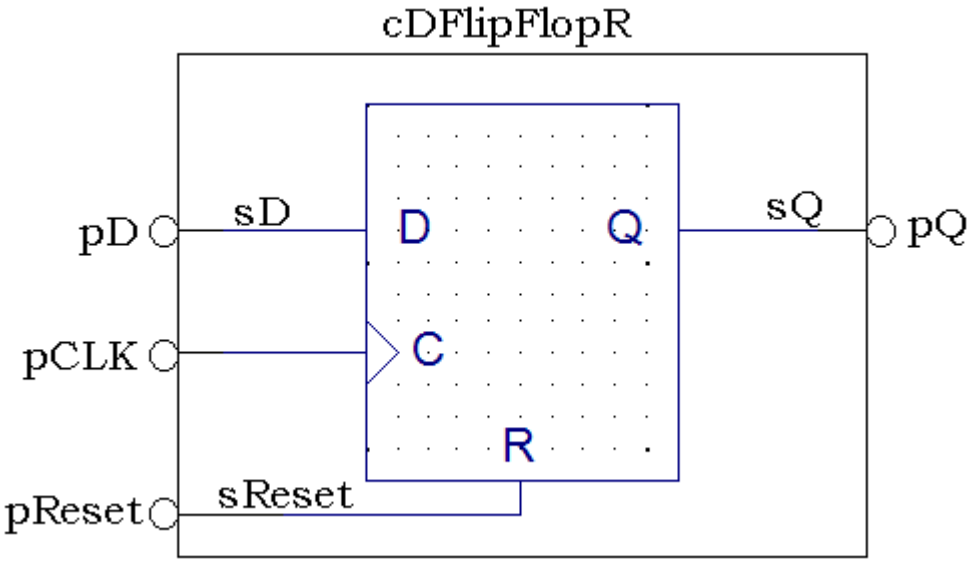


cDFlipFlop.vhd

Szinkron működésű modulok




```
entity cDFlipFlop is
    Port ( pCLK : in  STD_LOGIC;
           pD   : in  STD_LOGIC;
           pQ   : out STD_LOGIC);
end cDFlipFlop;

architecture Behavioral of cDFlipFlop is

signal sD,sQ : STD_LOGIC;

begin

 sD <= pD;
 pQ <= sQ;

 process (pCLK)
 begin
   if (pCLK' event and pCLK = '1') then
    sQ <= sD;
   end if;
 end process;

end Behavioral;
```



Aszinkron Reset:

```
process (pCLK,sReset)
 begin
  if sReset = '1' then
   sQ <= '0';
  elsif (pCLK' event and pCLK = '1') then
   sQ <= sD;
  end if;
 end process;
```

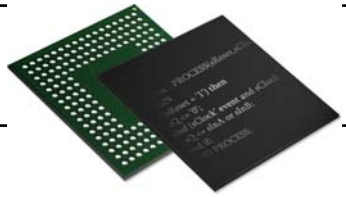
Szinkron Reset:

```
MyProc : process (pCLK)
 begin
  if (pCLK' event and pCLK = '1') then
   if sReset = '1' then
    sQ <= '0';
   else
    sQ <= sD;
   end if;
  end if;
 end process;
```

A *Process*

```
process
begin
   <statements>;
   wait on sA, sB;
end process;
```

```
process (sA,sB)
begin
   <statements>;
end process;
```



A kombinációs *process* általános formája:

```
process (<all_input_signals_separated_by_commas>)
begin
   <statements>;
end process;
```

Órejel felfutó illetve lefutó élére működő *process*-ek:

```
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      <statements>;
   end if;
end process;
```

```
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '0') then
      <statements>;
   end if;
end process;
```

A konkurrens értékadás

Gyakran előfordul, hogy egy folyamat kizárólag egy jelhez rendel értékeket különböző feltételek mellett. Ilyen esetekben lehetőség van arra, hogy egy külön folyamat definiálása nélkül, egyszerűsített formában rendeljünk értékeket az adott jelhez. Az értékadásnak ezt a formáját a VHDL-ben konkurrens értékadásnak nevezzük. A konkurrens értékadás szintaktikája a következő:

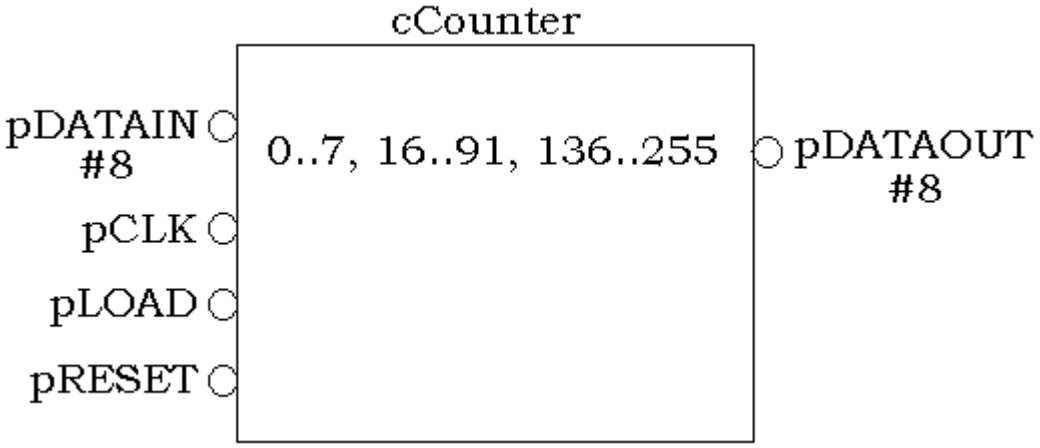
Értékadás process-ben

```
process (sA,sB,sSelect)
begin
 sY <= sA and sB;

 if (sA = '1' and sB = '1') then
  sY <= '1';
 else
  sY <= '0';
 end if;

 case sSelect is
  when "00" => sOut <= "0001";
  when "01" => sOut <= "0010";
  when "10" => sOut <= "0100";
  when "11" => sOut <= "1000";
  when others => sOut <= "0000";
 end case;


end process;
```

Konkurrens értékadás

```
......
sY <= sA and sB;

sY <= '1' when (sA = '1' and
           sB = '1') else
      '0';


with sSelect select
 sOut <= "0001" when "00",
         "0010" when "01",
         "0100" when "10",
         "1000" when "11",
         "0000" when others;
```

cCounter.vhd




```
entity cCounter is
    Port ( pCLK     : in  STD_LOGIC;
           pRESET   : in  STD_LOGIC;
           pLOAD    : in  STD_LOGIC;
           pDATAIN  : in  STD_LOGIC_VECTOR (7 downto 0);
           pDATAOUT : out STD_LOGIC_VECTOR (7 downto 0));
end cCounter;

architecture Behavioral of cCounter is

signal sLOAD : STD_LOGIC;
signal sDATAIN,sDATAOUT : STD_LOGIC_VECTOR (7 downto 0);

begin

 sLOAD    <= pLOAD;
 sDATAIN  <= pDATAIN;
 pDATAOUT <= sDATAOUT;

 Count : process (pCLK,pRESET)
 begin
   if (pRESET = '1') then
    sDATAOUT <= "00000000";
   elsif (pCLK' event and pCLK = '1') then
    if (sLOAD = '1') then
     sDATAOUT <= sDATAIN;
    elsif (sDATAOUT = "00000111") then -- 7
     sDATAOUT <= "00010000";           -- 16
    elsif (sDATAOUT = "01011011") then -- 91
     sDATAOUT <= "10001000";           -- 136
    else
     sDATAOUT <= sDATAOUT + 1;
    end if;
   end if;
 end process;

end Behavioral;
```

Operátorok

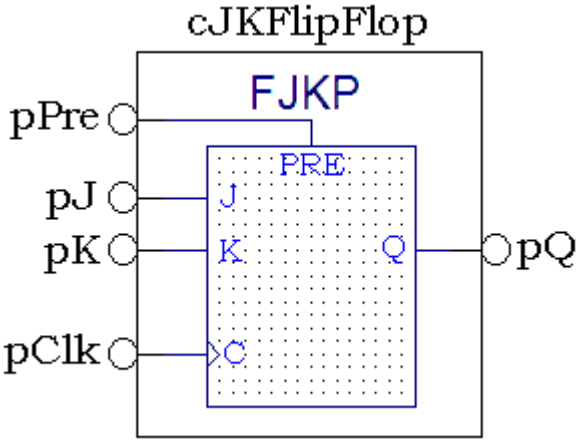
Csoport	Szimbólum	Funkció
aritmetikai	+ - * / **mod** **	összeadás kivonás szorzás osztás modulus hatványozás
relációs	= /= < > <= >=	egyenlő nem egyenlő kisebb nagyobb kisebb vagy egyenlő nagyobb vagy egyenlő
logikai	**and** **or** **not**	logikai és logikai vagy logikai negálás
bitenkénti	**and** **or** **xor** **not** **(nand** **nor** **xnor)**	bitenkénti és bitenkénti vagy bitenkénti kizáró vagy negálás
összefűző	&	összefűzés

Példa az összefűző operátor használatára:

```
signal sA,sB : STD_LOGIC_VECTOR (3 downto 0);
signal sOut  : STD_LOGIC_VECTOR (7 downto 0);
signal sL1,sL2,sL3,sL4 : STD_LOGIC;

sOut <= sA & sB;
sA   <= sL1 & sL2 & sL3 & sL4;
sB   <= "101" & sL2;
```

Feladat

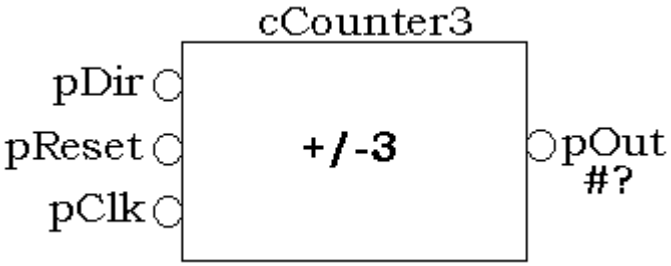
Írja meg az alábbi ábrán látható modult VHDL nyelven!



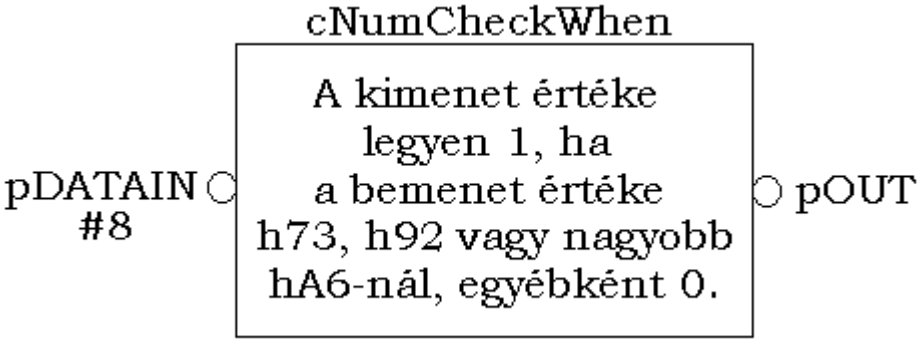
cCounter3.vhd

Feladat

Írjon olyan VHDL modult, amely a pDir bemenet állapotától függően felfelé (pDir = ’1’) vagy lefelé (pDir = ’0’) számol hármasával nulla és 30 között! Aszinkron reset hatására (pReset = ’1’) a kimenet 0 lesz.



A *when* utasítás

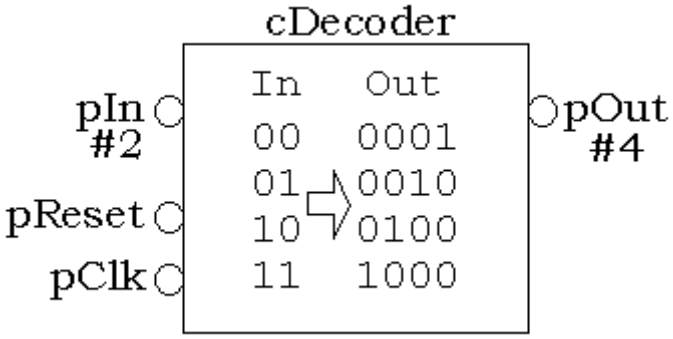



```
entity cNumCheck is
    Port ( pDATAIN : in  STD_LOGIC_VECTOR (7 downto 0);
           pOUT    : out STD_LOGIC);
end cNumCheck;

architecture Behavioral of cNumCheck is

signal sDATAIN : STD_LOGIC_VECTOR (7 downto 0);
signal sOUT    : STD_LOGIC;

begin

 sDATAIN <= pDATAIN;
 pOUT    <= sOUT;

 sOUT <= '1' when (sDATAIN = X"73") or
                  (sDATAIN = X"92") or
                  (sDATAIN > X"A6") else
         '0';

end Behavioral;
```

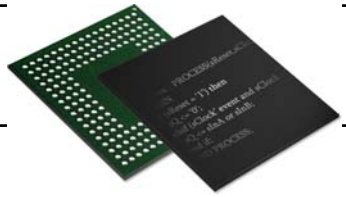
A *when* utasítás általános formája:

```
<name> <= <expression> when <condition> else
          <expression> when <condition> else
          <expression>;
```

A *case* utasítás




```
entity cDecoder is
    Port ( pClk   : in  STD_LOGIC;
           pReset : in  STD_LOGIC;
           pIn    : in  STD_LOGIC_VECTOR (1 downto 0);
           pOut   : out STD_LOGIC_VECTOR (3 downto 0));
end cDecoder;

architecture Behavioral of cDecoder is

signal sIn    : STD_LOGIC_VECTOR (1 downto 0);
signal sOut   : STD_LOGIC_VECTOR (3 downto 0);
signal sReset : STD_LOGIC;

begin

 sReset <= pReset;
 sIn    <= pIn;
 pOut   <= sOut;

 DecoderProc : process(pClk)
 begin
  if ( pClk' event and pClk = '1') then
   if ( sReset = '1') then
     sOut <= "0000";
   else
     case sIn is
       when "00" => sOut <= "0001";
       when "01" => sOut <= "0010";
       when "10" => sOut <= "0100";
       when "11" => sOut <= "1000";
       when others => sOut <= "0000";
     end case;
   end if;
  end if;
 end process;

end Behavioral;
```



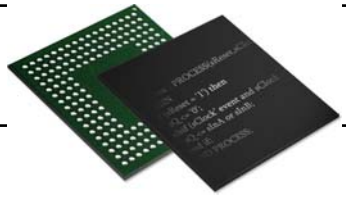
Általános forma:

```
case (<n-bit select>) is
   when "0…00" =>
      <statement>;
   when "0…01" =>
      <statement>;
   when "0…10" =>
      <statement>;
   .......
   when others =>
      <statement>;
end case;
```

Példák:

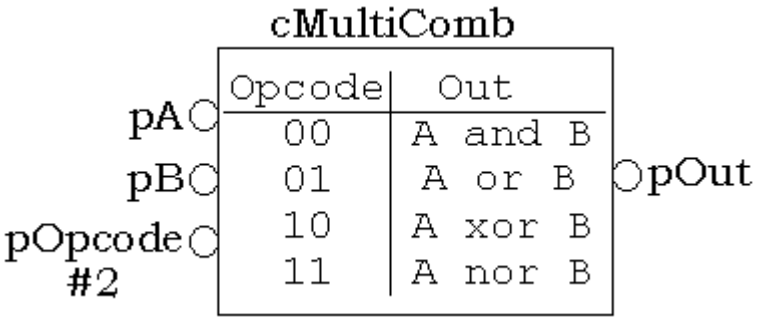
```
case sDataIn is
   when X"00" => sDataOut <= X"03";
                 sPage    <= '1';
   when X"0A" => sDataOut <= X"A5";
                 if (sLoad = '1') then
                  sPage <= '1';
                 else
                  sPage <= '0';
                 end if;
   when others => sDataOut <= "01011100";
                  sPage    <= '0';
end case;
```

```
case sSelector is
   when "001"|"011" => sStore <= not sStore;
   when "010"|"101"|"111" => sStore <= '1';
   when "110" => sStore <= sA;
   when others => sStore <= '0';
end case;
```

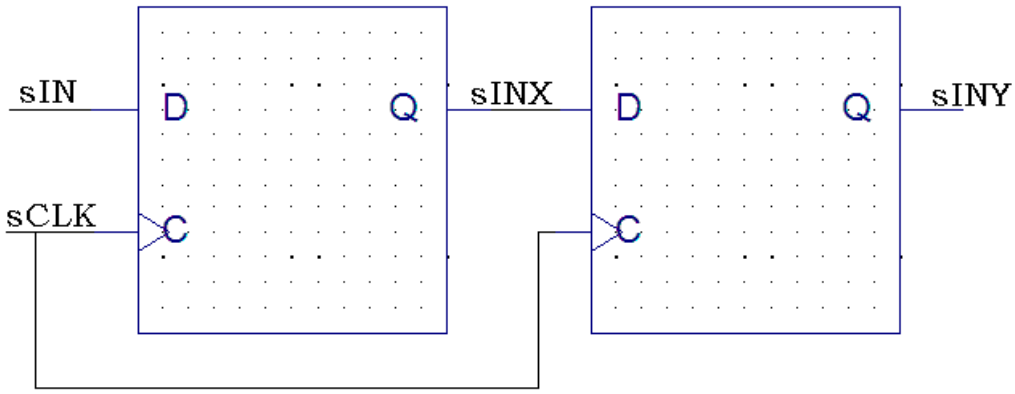


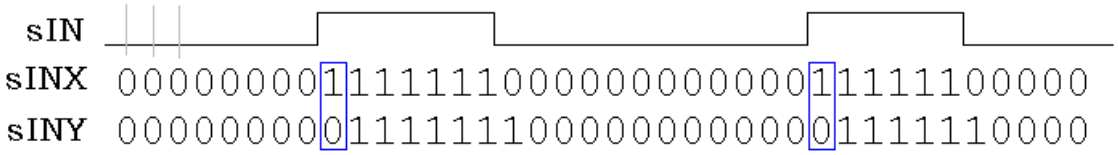
Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven *case* utasítást használva!

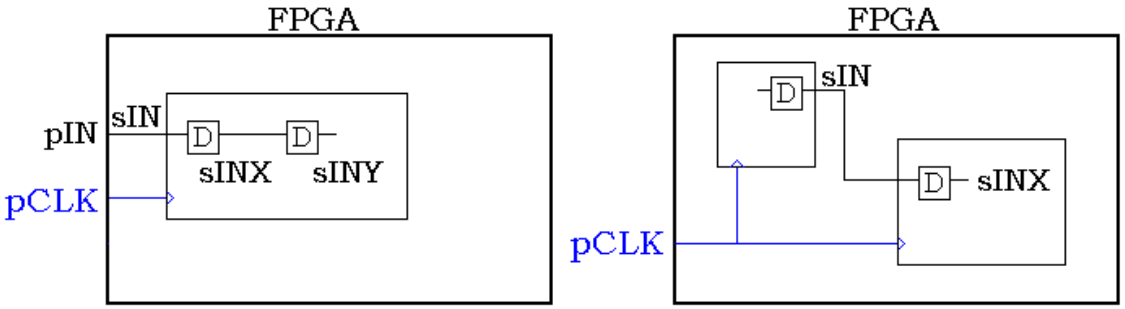


Jelek mintavételezése



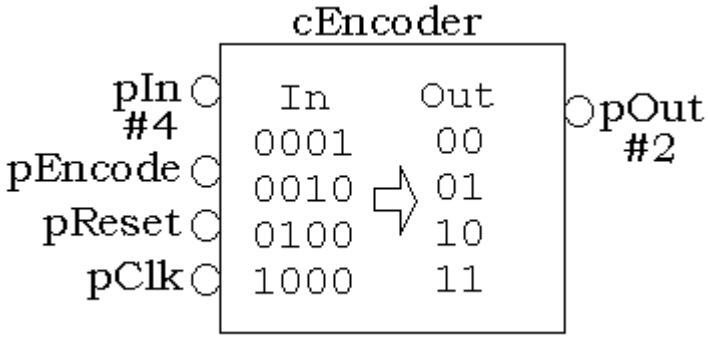



```
Sampling : process(pClk)
 begin
  if ( pClk' event and pClk = '1') then
   sINX <= sIN;
   sINY <= sINX;
   if ( sINX = '1' and sINY = '0') then
     ....
   end if;
  end if;
 end process;
```

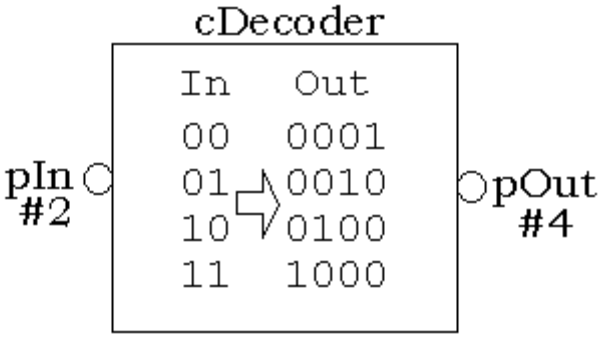


Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven! A modul pClk órajellel mintavételezze a pEncode jelet, és annak lefutó élére működjön, valamint pClk-hoz képest szinkron reset-tel rendelkezzen.



A *with* utasítás

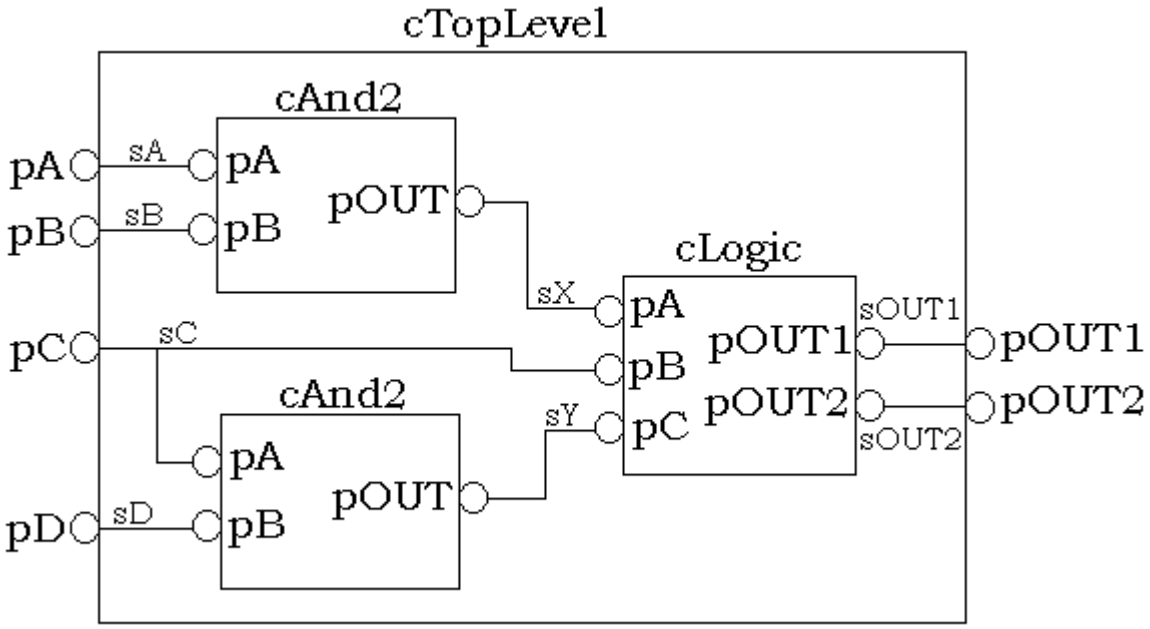



```
architecture Behavioral of cDecoder is

signal sIn    : STD_LOGIC_VECTOR (1 downto 0);
signal sOut   : STD_LOGIC_VECTOR (3 downto 0);

begin

 sIn    <= pIn;
 pOut   <= sOut;

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

end;
```

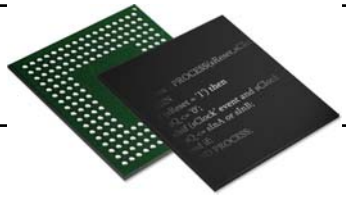
A *with* utasítás általános formája:

```
with <choice_expression> select
  <name> <= <expression> when <choices>,
            <expression> when <choices>,
            <expression> when others;
```

Strukturális modell, almodulok




```
entity cTopLevel is
    Port ( pA    : in  STD_LOGIC;
           pB    : in  STD_LOGIC;
           pC    : in  STD_LOGIC;
           pD    : in  STD_LOGIC;
           pOUT1 : out STD_LOGIC;
           pOUT2 : out STD_LOGIC);
end cTopLevel;

architecture Structure of cTopLevel is

component cAnd2 is
    Port ( pA   : in   STD_LOGIC;
           pB   : in   STD_LOGIC;
           pOUT : out  STD_LOGIC);
end component;

component cLogic is
    Port ( pA    : in  STD_LOGIC;
           pB    : in  STD_LOGIC;
           pC    : in  STD_LOGIC;
           pOUT1 : out STD_LOGIC;
           pOUT2 : out STD_LOGIC);
end component;
```




```
signal sA,sB,sC,sD : STD_LOGIC;
signal sX,sY       : STD_LOGIC;
signal sOUT1,sOUT2 : STD_LOGIC;

begin

 sA <= pA;
 sB <= pB;
 sC <= pC;
 sD <= pD;

 pOUT1 <= sOUT1;
 pOUT2 <= sOUT2;

 And2_Inst1 : cAnd2
  Port Map( pA   => sA,
            pB   => sB,
            pOUT => sX );

 And2_Inst2 : cAnd2
  Port Map( pA   => sC,
            pB   => sD,
            pOUT => sY );

 Logic_Inst1 : cLogic
  Port Map( pA    => sX,
            pB    => sC,
            pC    => sY,
            pOUT1 => sOUT1,
            pOUT2 => sOUT2 );

end Structure;
```

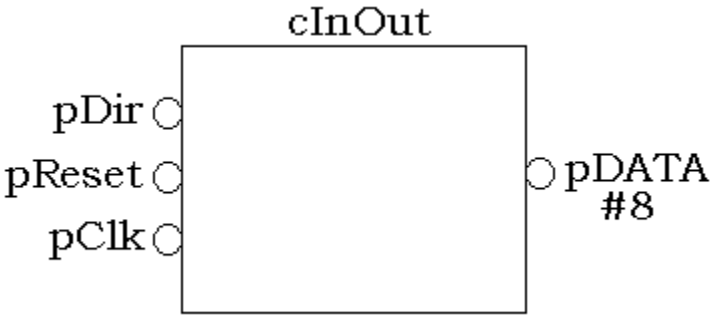
Komponens deklarálása:

```
component <component_name> is
port (
   <port_name> : <mode> <type>;
   <other ports>...
);
end component;
```

Példány létrehozása:

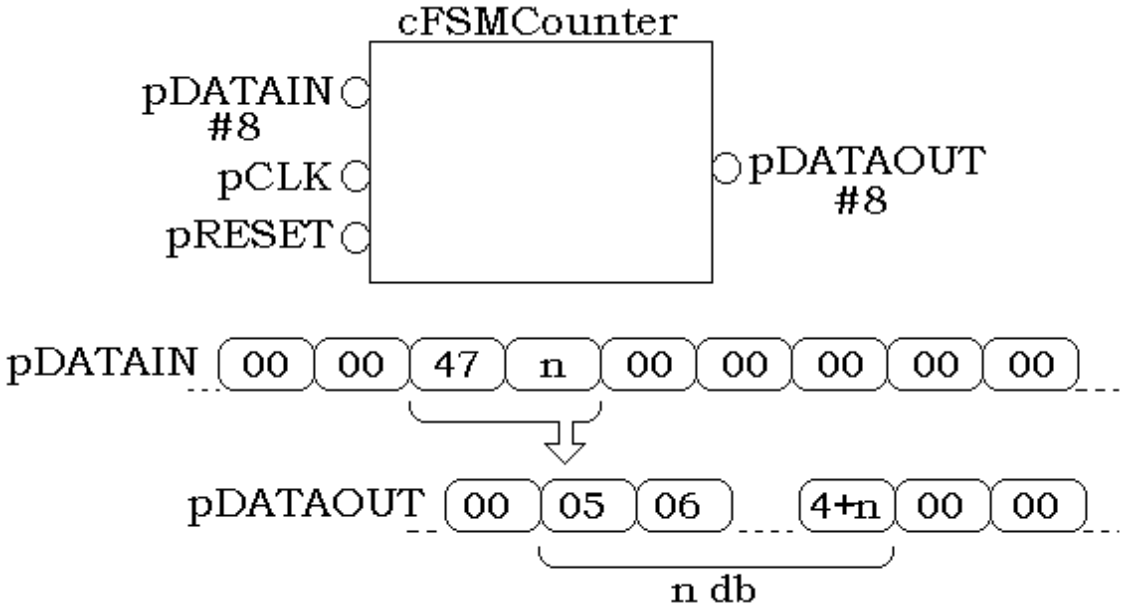
```
<instance_name> : <component_name>
port map (
   <port_name> => <signal_name>,
   <other ports>...
);
```

Kétirányú portok kezelése




```
entity cInOut is
    Port ( pDATA  : inout  STD_LOGIC_VECTOR (7 downto 0);

           pRESET : in  STD_LOGIC;
           pCLK   : in  STD_LOGIC;
           pDIR   : in  STD_LOGIC);
end cInOut;

architecture Behavioral of cInOut is

signal sRESET,sDIR : STD_LOGIC;
signal sDATAIN,sDATAOUT : STD_LOGIC_VECTOR (7 downto 0);
signal sCOUNTER : STD_LOGIC_VECTOR (7 downto 0);

begin

 sRESET   <= pRESET;
 sDIR     <= pDIR;

 sDATAIN <= pDATA;
 pDATA   <= sDATAOUT when sDIR = '1' else "ZZZZZZZZ";

 Count : process (pCLK,sRESET)
 begin
   if (sRESET = '1') then
    sDATAOUT <= X"00";
   elsif (pCLK' event and pCLK = '1') then
    if (sDIR = '1') then
     sDATAOUT <= sDATAOUT + 1;
    else
     sDATAOUT <= sDATAIN;
    end if;
   end if;
 end process;

end Behavioral;
```

Állapotgépek (FSM)



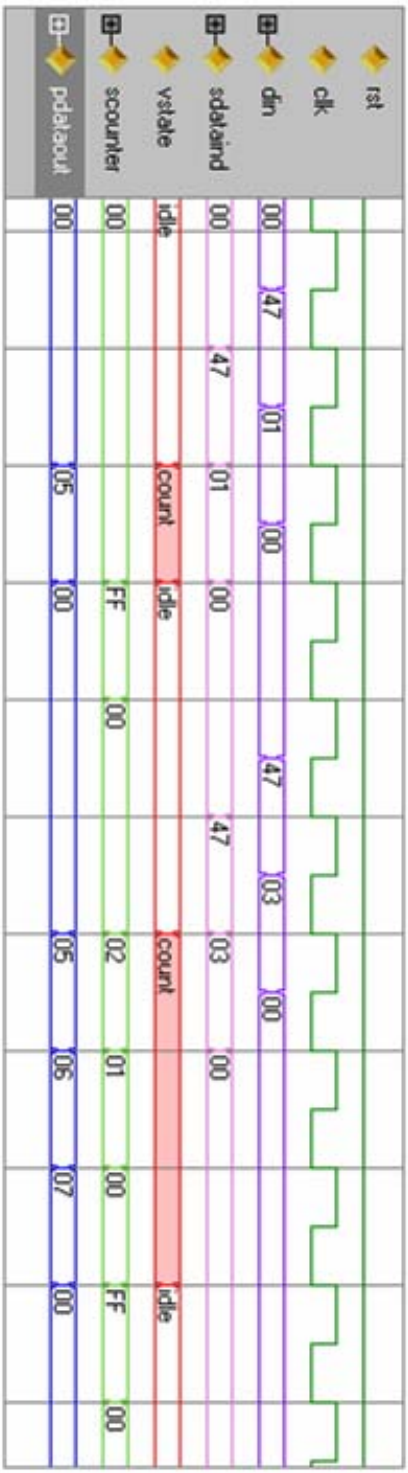
A fenti rendszer két állapot között ugrál:

- Várakozik egy bemeneti h47-n kombinációra (Idle)
- Számlál a kimeneten h05-től (4+n)-ig (Count)

```
entity cFSMCounter is
    Port ( pRESET   : in  STD_LOGIC;
           pCLK     : in  STD_LOGIC;
           pDATAIN  : in  STD_LOGIC_VECTOR(7 downto 0);
           pDATAOUT : out STD_LOGIC_VECTOR(7 downto 0)
          );
end cFSMCounter;

architecture Behavioral of cFSMCounter is

type tState is (Idle,Count);

signal sRESET   : STD_LOGIC;
signal sDATAIN  : STD_LOGIC_VECTOR(7 downto 0);
signal sDATAIND : STD_LOGIC_VECTOR(7 downto 0);

signal sDATAOUT : STD_LOGIC_VECTOR(7 downto 0);
signal sCOUNTER : STD_LOGIC_VECTOR(7 downto 0);
```

```
begin

  sRESET   <= pRESET;
  sDATAIN  <= pDATAIN;
  pDATAOUT <= sDATAOUT;

FSMCount : process(sRESET,pCLK)
variable vState : tState;
begin

 if (sRESET = '1') then
  vState   := Idle;
  sDATAIND <= X"00";
  sDATAOUT <= X"00";
  sCOUNTER <= X"00";
 elsif (pCLK' event and pCLK = '1') then

  sDATAIND <= sDATAIN;

  case vState is
   when Idle =>
     if (sDATAIND = X"47") and (sDATAIN /= X"00") then
      vState   := Count;
      sCOUNTER <= sDATAIN-1;
      sDATAOUT <= X"05";
     else
      vState   := vState;
      sCOUNTER <= X"00";
      sDATAOUT <= X"00";
     end if;

   when Count  =>
     sCOUNTER <= sCOUNTER-1;
     if sCounter = X"00" then
      vState   := Idle;
      sDATAOUT <= X"00";
     else
      vState   := vState;
      sDATAOUT <= sDATAOUT+1;
     end if;

   when others  =>
     vState   := Idle;
     sCOUNTER <= X"00";
     sDATAOUT <= X"00";
  end case;

 end if;
end process;

end Behavioral;
```

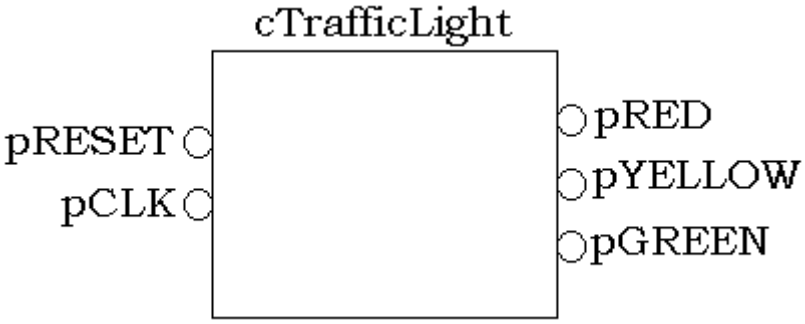
ModelSim SE 6.0 programmal lefuttatott szimuláció eredménye:



Feladat

Írja meg az alábbi ábrán látható modult VHDL nyelven, kétféle változatban!

1) A jelzőlámpa minden órajel élre állapotot vált.
2) Az egyes állapotokban töltött idő a valóságos működésnek megfelelően különböző.

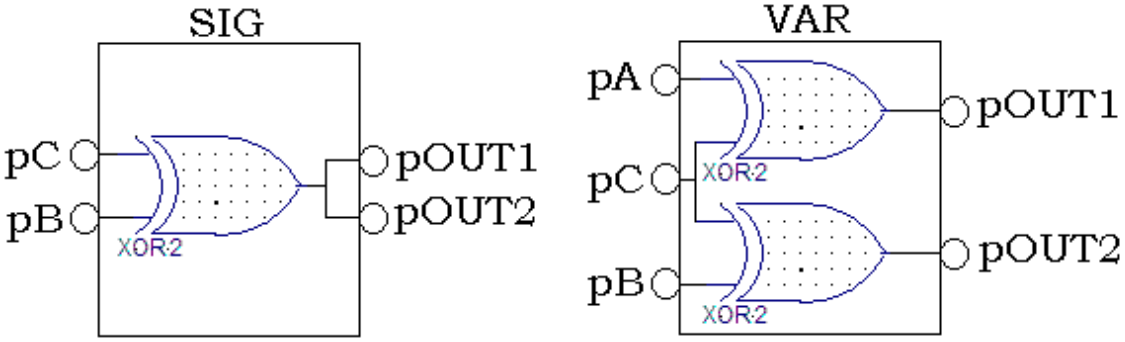


A *Signal* és a *Variable* közötti különbségek

Moduljainkban egyaránt használhatunk signal-okat és változókat (*variable*):

- A signal-ok a hardverhez (D-tároló, vezeték) hasonlítanak, értékük nem frissül a *process* végéig. Ha egy *process*-ben egy *signal* többször is értéket kap, csak a legutolsó értékadást kell figyelembe venni.
- A változók ezzel szemben rögtön megváltoztatják az értéküket, és újabb értékadásig tarják azt.

```
signal sA,sB,sC    : STD_LOGIC;
signal sOUT1,sOUT2 : STD_LOGIC;

signal   sD : STD_LOGIC;
variable vD : STD_LOGIC;

SIG : process (sA,sB,sC)
 begin
  sD <= sA;              -- ignored !!
  sOUT1 <= sC xor sD;
  sD <= sB;              -- overrides !!
  sOUT2 <= sC xor sD;
 end;

VAR : process (sA,sB,sC)
 begin
  vD := sA;
  sOUT1 <= sC xor vD;
  vD := sB;
  sOUT2 <= sC xor vD;
 end;
```



Előre definiált típusok

```
STD_LOGIC            --'U','X','0','1','Z','W','L','H','-'
STD_LOGIC_VECTOR     --Natural Range of STD_LOGIC
BOOLEAN              --True or False
INTEGER              --32 or 64 bits
NATURAL              --Integers >= 0
POSITIVE             --Integers > 0
REAL                 --Floating-point
BIT                  --'0','1'
BIT_VECTOR(Natural)  --Array of bits
CHARACTER            --7-bit ASCII
STRING(POSITIVE)     --Array of charachters
```

Signal és változó deklaráció

Általános forma:
```
signal <name> : <type> := <value>;
variable <name> : <type> := <value>;
```

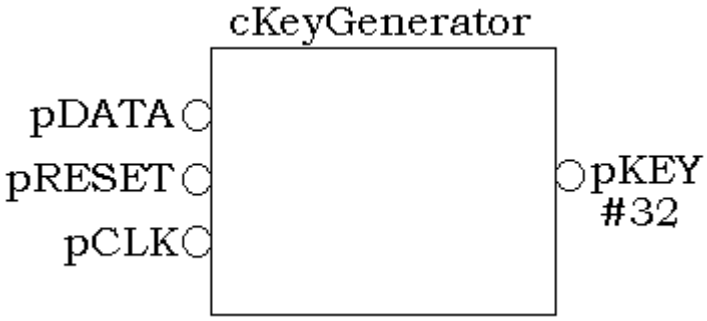
Példák:
```
variable D : STD_LOGIC_VECTOR(3 downto 0);
variable IntR : INTEGER := 55;
```

Konstansok

Általános forma: `constant <name>: <type> := <value>;`

```
constant cID : std_logic_vector(23 downto 0) := X"000309";
constant cSI_Key : std_logic_vector(7 downto 0) := X"56";
constant INPUTS : natural := 8;

sID <= cID;
sENABLE34U(0) <= '1' when INPUTS > 2 else '0';
sENABLE34U(1) <= '1' when INPUTS > 4 else '0';
```

Függvények




```
entity cKeyGenerator is
    Port ( pRESET : in  STD_LOGIC;
           pCLK   : in  STD_LOGIC;
           pDATA  : in  STD_LOGIC;
           pKEY   : out STD_LOGIC_VECTOR (31 downto 0));
end cKeyGenerator;

architecture Behavioral of cKeyGenerator is

--- Next Key function ------------------------------
function NextKey
  ( Data :  std_logic;
    Key  :  std_logic_vector(31 downto 0))
  return std_logic_vector is

  variable NewKey : std_logic_vector(31 downto 0);

  begin

    NewKey(0)  := Data xor Key(31);
    NewKey(1)  := Data xor Key(0) xor Key(31);
    NewKey(2)  := Data xor Key(1) xor Key(31);
    -- ....
    NewKey(30) := Key(29);
    NewKey(31) := Key(30);

  return NewKey;

end NextKey;
----------------------------------------------------

signal sRESET,sDATA : STD_LOGIC;
signal sKEY : STD_LOGIC_VECTOR(31 downto 0);
```

```
begin

 sRESET <= pRESET;
 sDATA  <= pDATA;
 pKEY   <= sKEY;

 process (pCLK)
 begin
   if sRESET = '1' then
    sKEY <= X"FFFFFFFF";
   elsif (pCLK' event and pCLK = '1') then
    sKEY <= NextKey(sDATA,sKEY);
   end if;
 end process;

end Behavioral;
```

Általános forma:

```
function <FUNC_NAME> (<comma_separated_inputs> : <type>;
                      <comma_separated_inputs> : <type>)
return <type> is
 -- subprogram_declarative_items (constant declarations,
                                  variable declarations, etc.)
begin

 -- function body

  return <ret_val>;
end <FUNC_NAME>;
```

Paraméterezhető modulok, Generic

Állítható bitszélességű összeadó-kivonó modul:

```
entity cNAdder is
  generic( width : natural );
  port(
        pA      : in  std_logic_vector(width-1 downto 0);
        pB      : in  std_logic_vector(width-1 downto 0);
        pSUBSEL : in std_logic;
        pSUM    : out std_logic_vector(width-1 downto 0));
end cNAdder;

architecture rtl of cNAdder is

signal sA      : std_logic_vector(width-1 downto 0);
signal sB      : std_logic_vector(width-1 downto 0);
signal sSUM    : std_logic_vector(width-1 downto 0);
signal sSUBSEL : std_logic;

begin

sA      <= pA;
sB      <= pB;
sSUBSEL <= pSUBSEL;
pSUM    <= sSUM;

ADDERPROC : process(sA,sB,sSUBSEL)
begin

  if sSUBSEL = '1' then
    sSUM <= sA - sB;
  else
    sSUM <= sA + sB;
  end if;

end process ADDERPROC;
end rtl;
```

Példányosítás:

```
NAdder : cNAdder
 Generic Map( width => 8 )
 Port Map(
        pA      => sA,
        pB      => sB,
        pSUBSEL => sSUBSEL,
        pSUM    => sSUM);
```

Általános forma:

```
entity <entity_name> is
generic ( <generic_name> : <type> := <value>;
          <other generics>...);
port ( <port_name> : <mode> <type>;
       <other ports>...);
end <entity_name>;
```

Példa (Spartan 3 DCM modulja):

```
   component DCM
      generic( CLK_FEEDBACK : string :=  "1X";
               CLKDV_DIVIDE : real :=  2.000000;
               CLKFX_DIVIDE : integer :=  1;
               CLKFX_MULTIPLY : integer :=  4;
               CLKIN_DIVIDE_BY_2 : boolean :=  FALSE;
               CLKIN_PERIOD : real :=  10.000000;
               CLKOUT_PHASE_SHIFT : string :=  "NONE";
               DESKEW_ADJUST : string :=  "SYSTEM_SYNCHRONOUS";
               DFS_FREQUENCY_MODE : string :=  "LOW";
               DLL_FREQUENCY_MODE : string :=  "LOW";
               DUTY_CYCLE_CORRECTION : boolean :=  TRUE;
               FACTORY_JF : bit_vector :=  x"C080";
               PHASE_SHIFT : integer :=  0;
               STARTUP_WAIT : boolean :=  FALSE;
               DSS_MODE : string :=  "NONE");
      port ( CLKIN    : in    std_logic;
             CLKFB    : in    std_logic;
             RST      : in    std_logic;
             PSEN     : in    std_logic;
             PSINCDEC : in    std_logic;
             PSCLK    : in    std_logic;
             DSSEN    : in    std_logic;
             CLK0     : out   std_logic;
             CLK90    : out   std_logic;
             CLK180   : out   std_logic;
             CLK270   : out   std_logic;
             CLKDV    : out   std_logic;
             CLK2X    : out   std_logic;
             CLK2X180 : out   std_logic;
             CLKFX    : out   std_logic;
             CLKFX180 : out   std_logic;
             STATUS   : out   std_logic_vector (7 downto 0);
             LOCKED   : out   std_logic;
             PSDONE   : out   std_logic);
   end component;
```

DCM modul példányosítása:

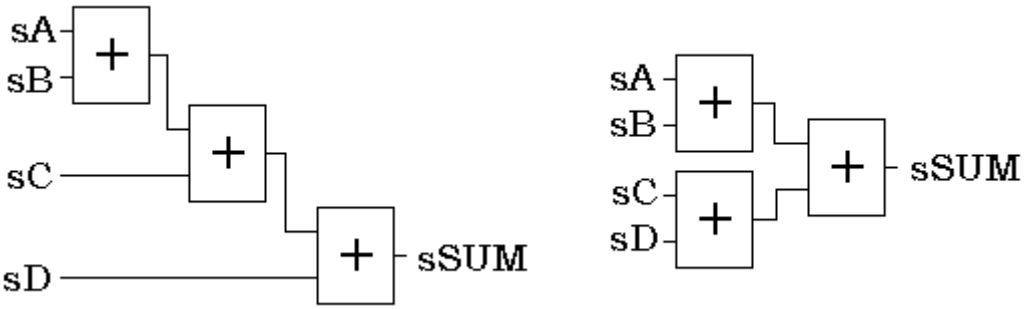
```
DCM_INST : DCM
generic map( CLK_FEEDBACK => "1X",
         CLKDV_DIVIDE => 2.000000,
         CLKFX_DIVIDE => 1,
         CLKFX_MULTIPLY => 4,
         CLKIN_DIVIDE_BY_2 => FALSE,
         CLKIN_PERIOD => 40.000000,
         CLKOUT_PHASE_SHIFT => "NONE",
         DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
         DFS_FREQUENCY_MODE => "LOW",
         DLL_FREQUENCY_MODE => "LOW",
         DUTY_CYCLE_CORRECTION => TRUE,
         FACTORY_JF => x"C080",
         PHASE_SHIFT => 0,
         STARTUP_WAIT => TRUE)
   port map (CLKFB    => CLKFB_IN,
             CLKIN    => CLKIN_IBUFG,
             DSSEN    => GND1,
             PSCLK    => GND1,
             PSEN     => GND1,
             PSINCDEC => GND1,
             RST      => GND1,
             CLKDV    => open,
             CLKFX    => open,
             CLKFX180 => open,
             CLK0     => CLK0_BUF,
             CLK2X    => open,
             CLK2X180 => open,
             CLK90    => open,
             CLK180   => open,
             CLK270   => open,
             LOCKED   => open,
             PSDONE   => open,
             STATUS   => open);
```

Kezdeti értékadás

```
signal sA      : std_logic_vector(7 downto 0) := X"0F";
signal sSUBSEL : std_logic := '0';
```

Zárójelezés

```
sSUM <= sA + sB + sC + sD;

sSUM <= (sA + sB) + (sC + sD);
```



Latch-ek és tárolók

```
LATCH : process (sDATA,sGATE)
 begin
  if (sGATE = '1') then
    sQ <= sDATA;
  end if;
 end process;

D_REG : process (pCLK)
 begin
   if (pCLK' event and pCLK = '1') then
    sQ <= sD;
   end if;
 end process;
```

Hogyan konvertálható egy latch D-tárolóvá? *Else* ággal vagy órajellel.

If és *Case* szerkezetek

If utasítás használatánál figyeljünk az alábbiakra:

- Lehetőleg legyen *else* ág (lásd latch-ek).
- Minden kimenetről minden ágban rendelkezzünk. Célszerű minden *if* előtt egy alapértéket adni a kimeneteknek.

Példa:

```
sDATA <= X"F7";
if (sA = '1') then
   sDATA <= X"88";
elsif (sB = '1') then
   sDATA <= X"89";
end if;
```

- Kevesebb bemeneti jel csökkentheti a logikai szintek számát.
- Lehetőleg csak a vezérlőjeleket állítsuk elő *if* szerkezetben, ne a teljes jelfolyamot.

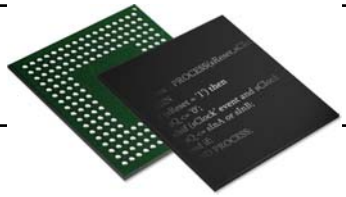
A *Case* utasítás szabályai:

- Mindig legyen *others* ág.
- Minden bemeneti értéket soroljunk fel.
- Minden ágban rendelkezzünk minden kimenetről.

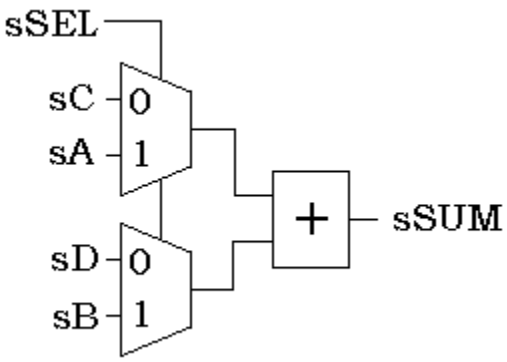
Erőforrás megosztás (resource sharing)

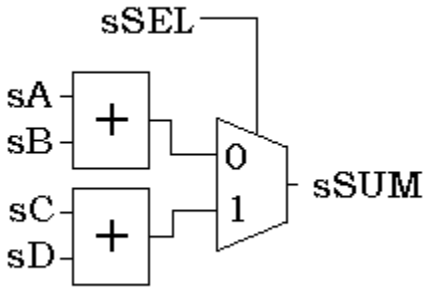
Az erőforrás megosztás egy optimalizációs eljárás, melynek lényege, hogy egyetlen funkcionális blokkot (pl. összeadó, komparátor) többször használunk fel. Tipikusan az alábbi operátoroknál van erre lehetőség:

```
*
+ -
> >= < <=
```



```
RESOURCE_SHARING : process (sA,sB,sC,sD,sSEL)
 begin
  if (sSEL = '1') then
    sSUM <= sA + sB;
  else
    sSUM <= sC + sD;
  end if;
 end process;
```





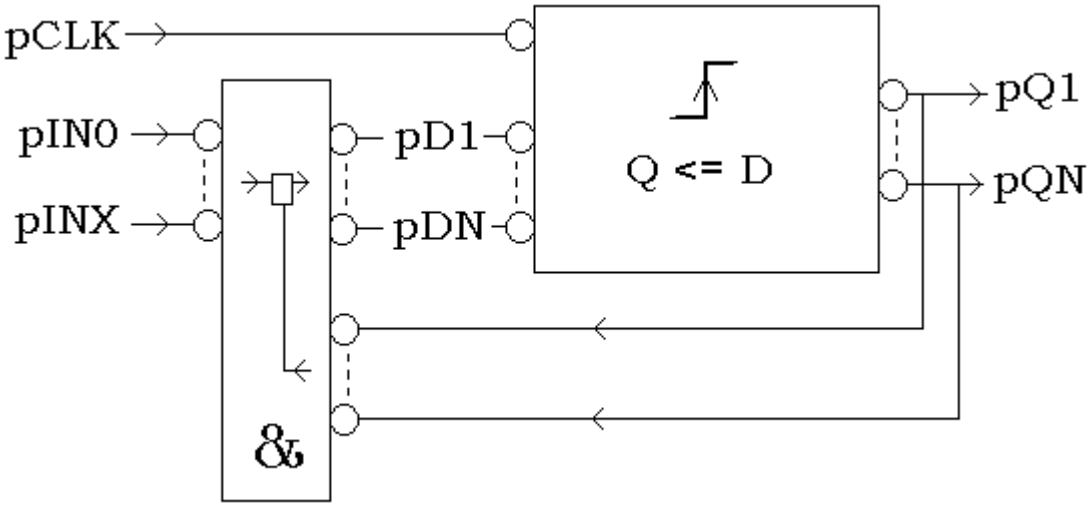
Órajel engedélyezés, órajel kapuzás

Kapuzott órajelek helyett használjunk Clock Enable bemenetet.

```
sGATECLOCK <= sCLK and sA and sB;

D_REG : process (sGATECLOCK)
 begin
   if (sGATECLOCK' event and sGATECLOCK = '1') then
    sQ <= sD;
   end if;
 end process;
```

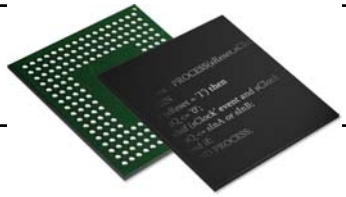
```
sCLOCKENABLE <= sA and sB;

D_REG : process (sCLK)
 begin
   if (sCLK' event and sCLK = '1') then
     if (sCLOCKENABLE = '1') then
       sQ <= sD;
     end if;
   end if;
 end process;
```

RTL (Register Transfer Level)




```
entity cCounterRTL is
    Port ( pCLK     : in  STD_LOGIC;
           pRESET   : in  STD_LOGIC;
           pLOAD    : in  STD_LOGIC;
           pDATAIN  : in  STD_LOGIC_VECTOR (7 downto 0);
           pDATAOUT : out STD_LOGIC_VECTOR (7 downto 0));
end cCounterRTL;
```


```
architecture RTL of cCounterRTL is

signal sLOAD,sRESET : STD_LOGIC;
signal sDATAIN : STD_LOGIC_VECTOR (7 downto 0);

signal sDATAD : STD_LOGIC_VECTOR (7 downto 0);
signal sDATAQ : STD_LOGIC_VECTOR (7 downto 0);

begin

 sLOAD    <= pLOAD;
 sRESET   <= pRESET;
 sDATAIN  <= pDATAIN;
 pDATAOUT <= sDATAQ;

 DQPROC : process (pCLK,sRESET)
 begin
   if (pRESET = '1') then
    sDATAQ <= "00000000";
   elsif (pCLK' event and pCLK = '1') then
    sDATAQ <= sDATAD;
   end if;
 end process;

 LOGICPROC : process (sRESET,sDATAIN,sLOAD,sDATAQ)
 begin
   if (sRESET = '1') then
    sDATAD <= "00000000";
   elsif (sLOAD = '1') then
    sDATAD <= sDATAIN;
   elsif (sDATAQ = "00000111") then -- 7
    sDATAD <= "00010000";           -- 16
   elsif (sDATAQ = "01011011") then -- 91
    sDATAD <= "10001000";           -- 136
   else
    sDATAD <= sDATAQ + 1;
   end if;
 end process;

end RTL;
```



Xilinx FPGA eszközök hardver elemei

1., Xilinx ISE környezetben válasszuk az *Edit* menü *Language Templates* parancsát. Az FPGA speciális hardver elemeit a *VHDL | Device Primitive Instantiation | FPGA* alpontban érhetjük el. Az elemre kattintva az ablakban megjelenik a példányosításhoz kimásolható VHDL kód. A fordító program csak akkor ismeri fel az elemet, ha a modul elején szerepel az alábbi könyvtármegadás:

```
library UNISIM;
use UNISIM.vcomponents.all;
```

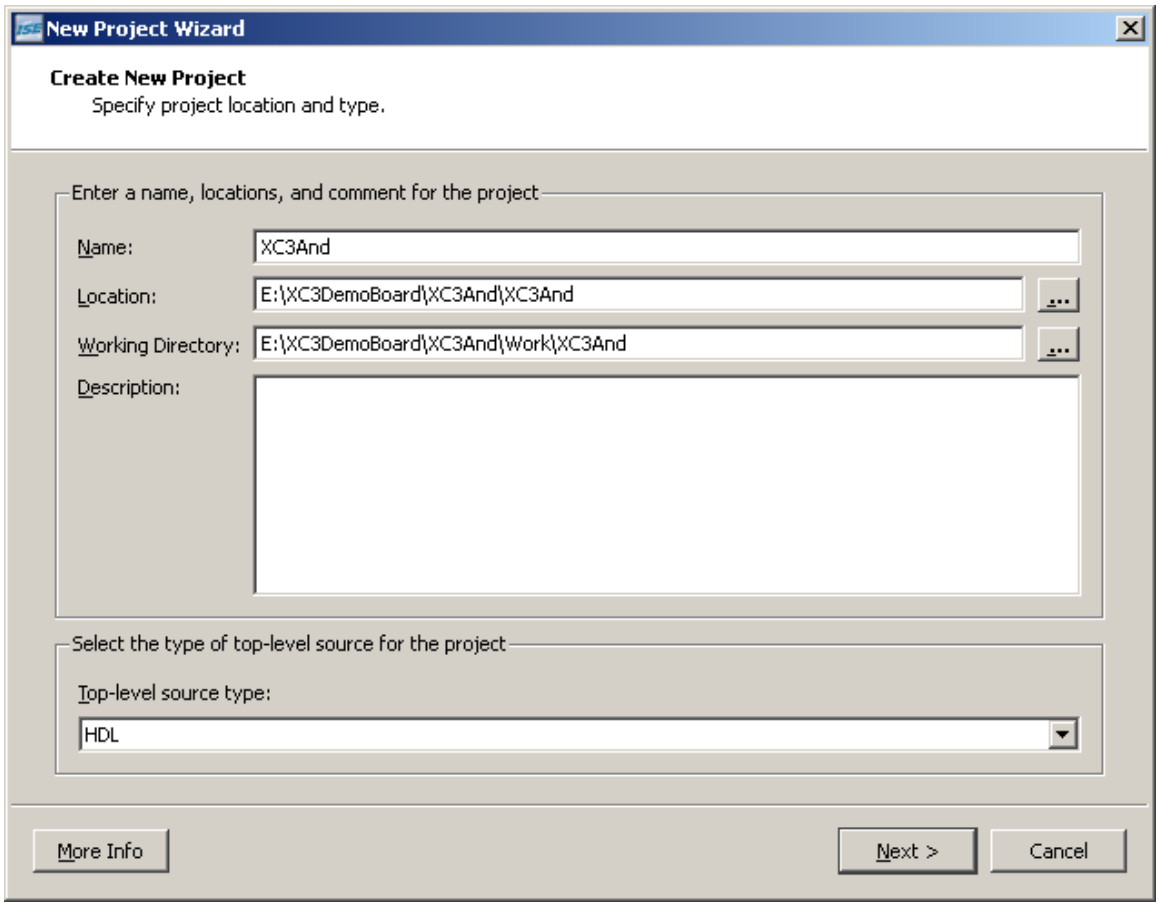
Példa:

```
BUFGMUX_inst : BUFGMUX
port map (
   O => O,     -- Clock MUX output
   I0 => I0,   -- Clock0 input
   I1 => I1,   -- Clock1 input
   S => S      -- Clock select input
);
```

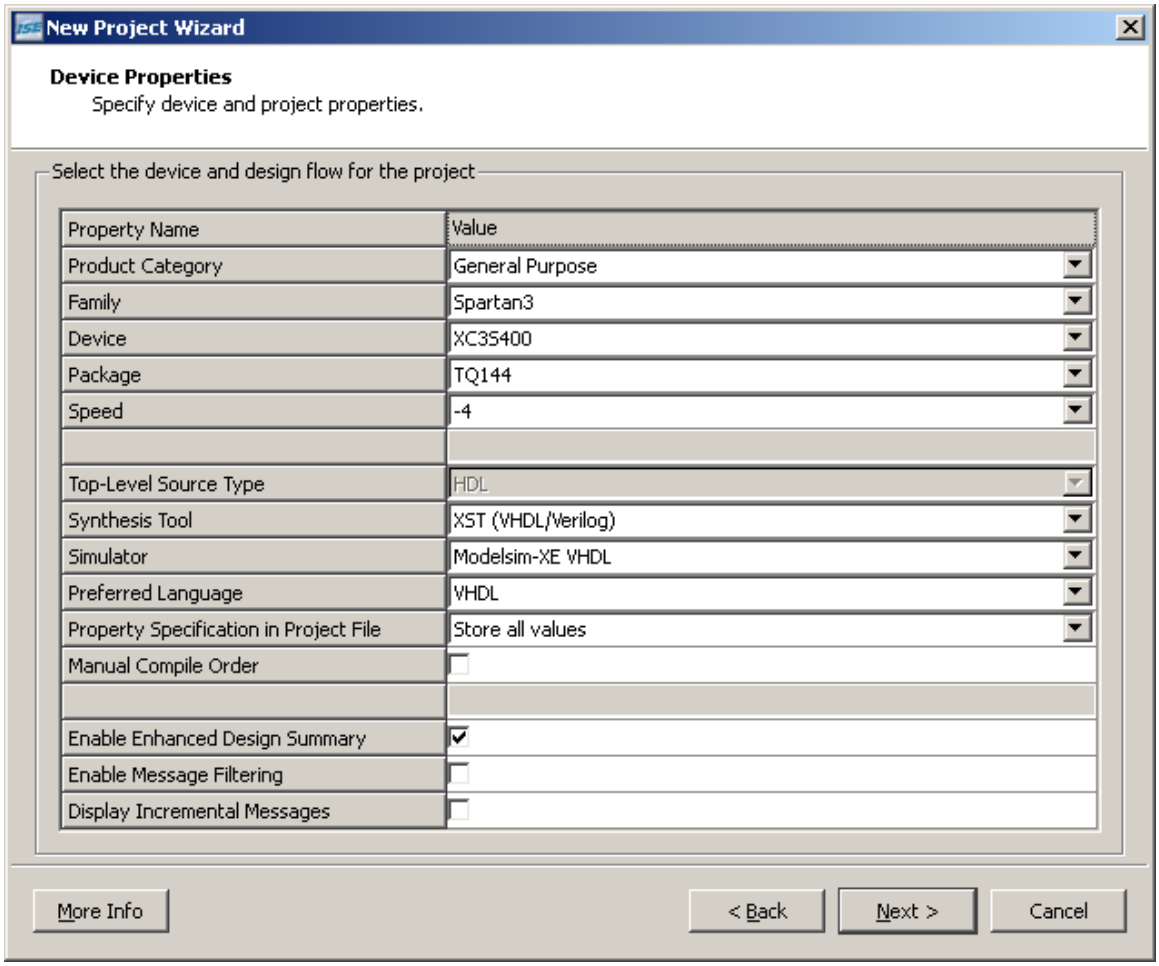
2., Egyes elemek létrehozásához és konfigurálásához a Xilinx ISE program grafikus felületet is biztosít. Adjunk hozzá új modult a project-hez a szokott módon, de a modul típusának válasszuk az *IP (Coregen and Architecture Wizard)*-ot. A következő ablakban válasszuk ki, milyen elemet szeretnénk létrehozni (pl. *FPGA Features and Design | Clocking | Single DCM*). Ezek után grafikusan állíthatjuk be a DCM modul paramétereit (portok, frekvencia, stb.). A konfigurált DCM elem .xaw kiterjesztésű fájlként adódik a project-hez. A *Sources* ablakban a fájlra kattintva megjelennek a modulhoz tartozó feladatok. Példányosításhoz futtasuk a *View HDL Instantiation Template* parancsot, és a megjelenő ablakból másoljuk ki a megfelelő kódrészletet. A *View HDL Source* paranccsal a modul teljes forráskódját elérhetjük. Mindkét parancsnál, a parancsra jobb egérgombbal kattintva beállíthatjuk, hogy VHDL vagy Verilog nyelven kérjük-e az eredményt.

Project létrehozása Xilinx Project Navigator 11.3-al

Indítsuk el a *Project Navigator*-t, majd válasszuk a *File* menü *New Project* parancsát. Adjuk meg a project nevét (pl. XC3And) és elérési útját (munkakönyvtárnak célszerű a project könyvtáron belüli alkönyvtárat megadni), majd állítsuk be a legfelső szint (Top-Level) típusát *HDL*-re.

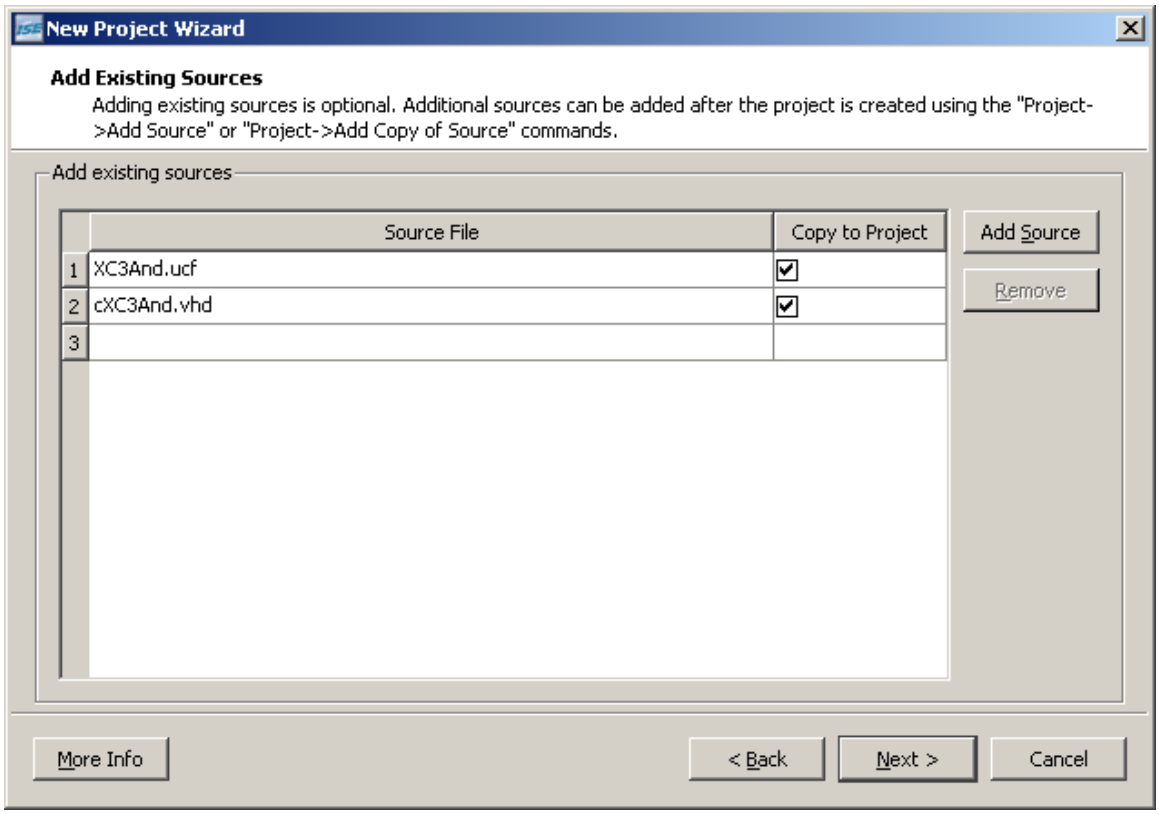


Állítsuk be az CPLD vagy FPGA eszköz paramétereit. XC3 Demo Board esetén az FPGA típusa (Xilinx Spartan 3) XC3S400, TQ144-es tokban. *Synthesis Tool*-nak válasszuk az *XST*-t, *Preferred Language*-nak a VHDL-t.

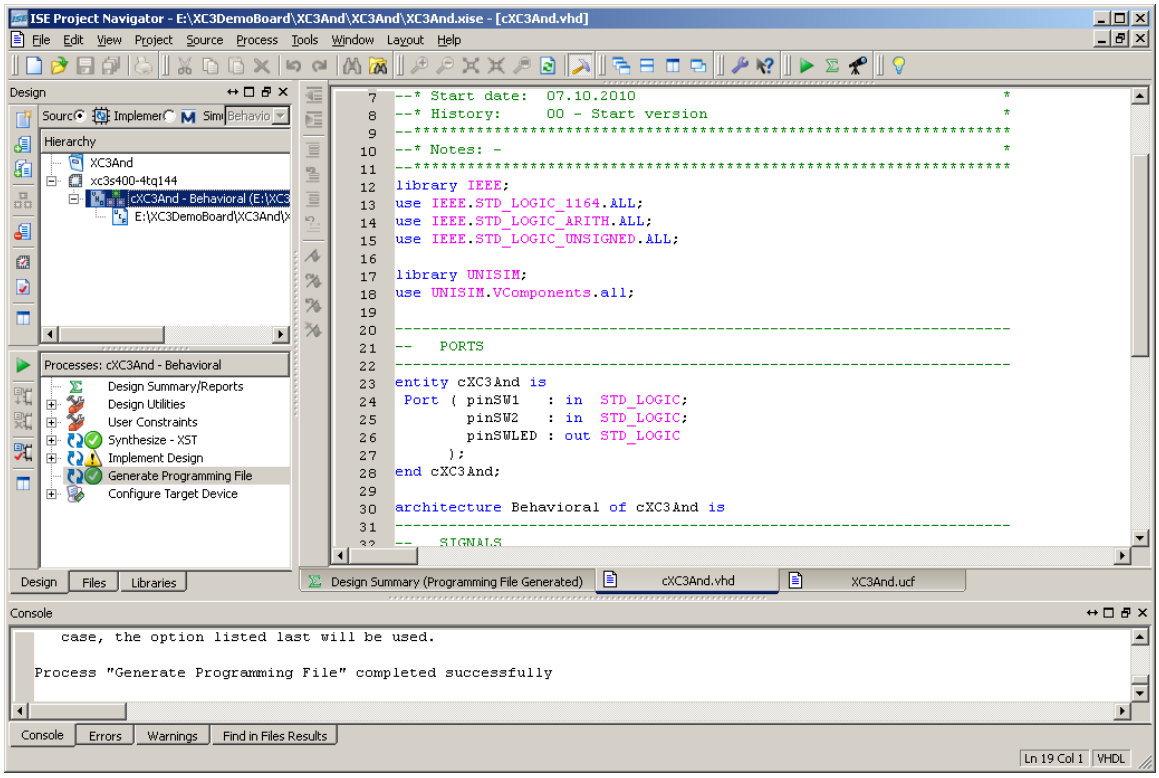


Ezt követően a *Create New Source* ablakot ugorjuk át (a jövőben itt definiálhatunk új modulokat).

Az *Add Existing Sources* ablakban adjuk hozzá a projecthez az cXC3And.vhd és az XC3And.ucf fájlokat.



A *Project Summary* ablakban kattintsunk a *Finish* gombra, majd az *Adding Source Files* ablakban az *OK* gombra.



A programozó fájlok előállítása

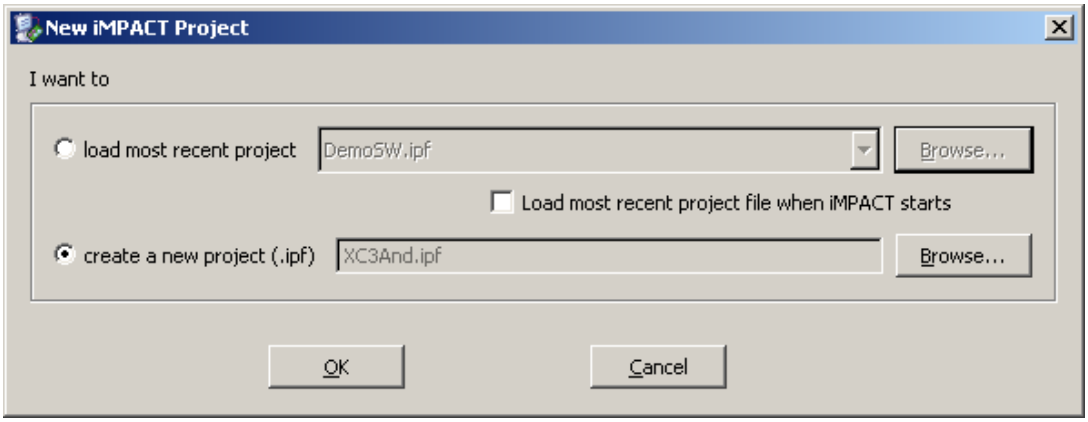
A design lefordításával létre kell hozni egy .bit kiterjesztésű fájlt:

- A *Hierarchy* ablakban legyen kijelölve a cXC3And – Behavioral… sor.
- A *Processes* ablakban kattintsunk kétszer a *Generating Programming File* sorra.
- Sikeres fordítás esetén a *Console* ablakban a „*Process Generate Programming File completed successfully*” kiírás jelenik meg. A project könyvtárban létrejön a .bit fájl.

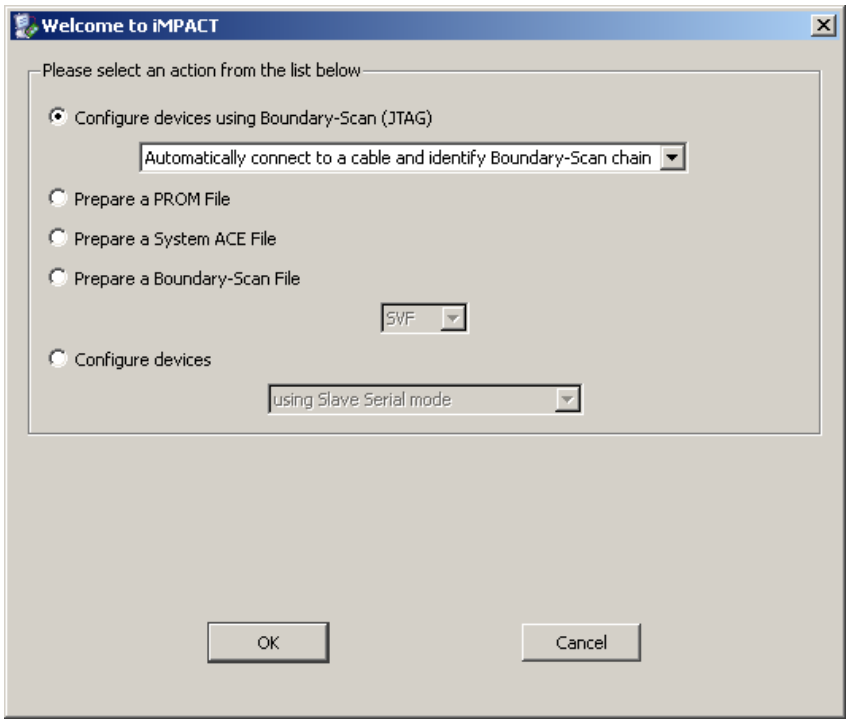
Az így előállított .bit fájl közvetlenül letölthető az FPGA-ba, de a platform flash-be nem. Ehhez az iMPACT programmal generálni kell egy .mcs kiterjesztésű fájlt a .bit fájlból. Indítsuk el az Impact programot, vagy kattintsunk kétszer *Processes* ablak *Configure Target Device* sorának *Manage Configuration Project* alpontjára.

.mcs fájl előállítása iMPACT-el

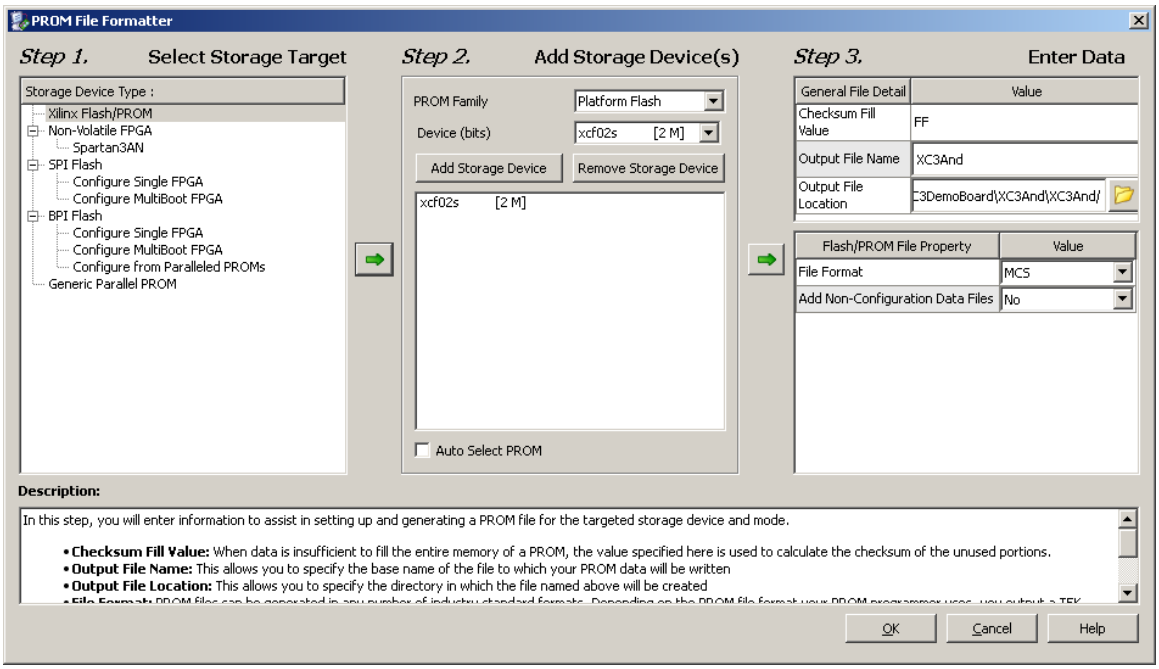
Indítsuk el az Impact programot, majd válasszuk a *File* menü *New Project* parancsát. A felbukkanó ablakban válasszuk a *Create a new project* opciót és adjuk meg az Impact project fájl nevét és elérési útját, majd *OK*.



A következő ablakban válasszuk a *Configure devices using Boundary-Scan (JTAG)* opciót, majd *OK*.



Kattintsunk kétszer az *iMPACT Flows* ablak *Create PROM File* sorára.



Eszköznek adjuk meg az xcf02s platform flash-t, és állítsuk be a kimeneti (programozó) fájl nevét. *OK*.

A program kérésére adjuk meg annak a .bit fájlnak a nevét, amelyből az .mcs fájlt elő kívánjuk állítani. Az előállításhoz az *iMPACT Processes* ablakban kattintsunk kétszer a *Generate File…* sorra. A művelet eredményeként létrejön a platform flash-be letölthető .mcs kiterjesztésű állomány.

Létező modul hozzáadása a project-hez

Válasszuk a *Project* menü *Add Source* parancsát, majd adjuk meg a csatolni kívánt modul (VHDL fájl, Schematic szimbólum…) elérési útját.

Új modul hozzáadása a project-hez

Válasszuk a *Project* menü *New Source* parancsát. A bal oldali listából jelöljük ki a *VHDL Module*-t. Adjuk meg a modul nevét (Pl. cMultiComb2), elérési útját, majd definiáljuk a modul portjait.

UCF fájl módosítása

A *Hierarchy* ablakban kattintsunk az .ucf fájlra. A *Processes* ablakban megjelennek a fájlhoz tartozó műveletek. Szöveges szerkesztéshez kattintsunk kétszer az *Edit Contstrains (Text)* sorra.

A Language Templates

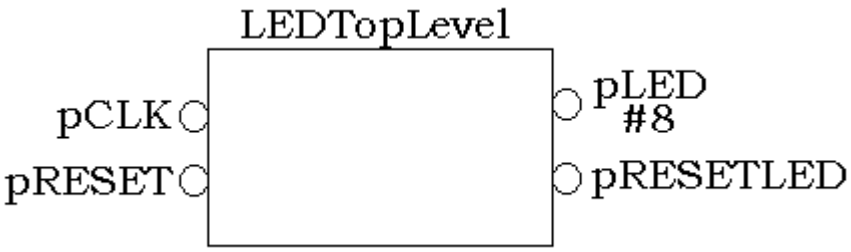
Válasszuk az *Edit* menü *Language Templates* parancsát. Az FPGA speciális hardver elemeit elérhetjük a *VHDL* | *Device Primitive Instantiation* | *FPGA* alpontban.

Az XST (Xilinx Synthesis Tool) lefoglalt szavai

abs	configuration	impure	null	rem	type
access	constant	in	of	report	unaffected
after	disconnect	inertial	on	return	units
alias	downto	inout	open	rol	until
all	else	is	or	ror	use
and	elsif	label	others	select	variable
architecture	end	library	out	severity	wait
array	entity	linkage	package	signal	when
assert	exit	literal	port	shared	while
attribute	file	loop	postponed	sla	with
begin	for	map	procedure	sll	xnor
block	function	mod	process	sra	xor
body	generate	nand	pure	srl	
buffer	generic	new	range	subtype	
bus	group	next	record	then	
case	guarded	nor	register	to	
component	if	not	reject	transport	

Feladat

Hozzunk létre új projectet, amely egyetlen top-level szintű VHDL modulból és a hozzá tartozó .ucf fájlból áll. A project és a modul neve legyen LEDTopLevel.



Működés:

- Active low reset (pRESET = ’0’) hatására a pLED kimeneti port nullázódjon, reset alatt a Reset LED világítson.
- A bemeneti órajel 8 MHz. A modulban hozzunk létre egy 24 bites számlálót, amely 0-tól számol felfelé, és pontosan fél másodpercenként nullázódik.
- A LED kimenetek 8 bites számlálóként viselkedjenek. A számláló értéke fél másodpercenként inkrementálódjon.
- A pRESET bemeneti porton legyen aktív felhúzó ellenállás (ucf-ben).

Külalak:

- Signal, port stb. elnevezések (s,p + NAGYBETŰK)
- Kommentek (fejléc, tagolás, egyéni kommentek)
- Bekezdések (process – begin - end process, if – else – end if)

```
----------------------------------------------------------------------------
-- Company:  ByteStudio Bt.
-- Engineer: dr. Tamás Zigó
--
-- Module Name:  LEDTopLevel - Behavioral
-- Project Name: LEDTopLevel
-- Target Devices: XC3S400
-- Additional Comments: -
----------------------------------------------------------------------------

entity cNumCheck is
    Port ( pDATAIN : in  STD_LOGIC_VECTOR (7 downto 0);
           pOUT    : out STD_LOGIC);
end cNumCheck;


architecture Behavioral of cNumCheck is

----------------------------------------------------------------------------
--    SIGNALS
----------------------------------------------------------------------------
signal sDATAIN : STD_LOGIC_VECTOR (7 downto 0);
signal sOUT    : STD_LOGIC;

begin
----------------------------------------------------------------------------
--    WIRES
----------------------------------------------------------------------------
 sDATAIN <= pDATAIN;
 pOUT    <= sOUT;

----------------------------------------------------------------------------
--    PROCESSES
----------------------------------------------------------------------------
 process (sDATAIN)
 begin
   if (sDATAIN = X"73") or (sDATAIN = X"92") or (sDATAIN > X"A6") then
    sOUT <= '1';
   else
    sOUT <= '0';
   end if;
 end process;

----------------------------------------------------------------------------
--    COMPONENTS
----------------------------------------------------------------------------

----------------------------------------------------------------------------
--    END
----------------------------------------------------------------------------
end Behavioral;
```