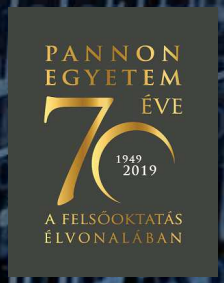
Pannon Egyetem, MIK-VIRT, Veszprém



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

Tervezési módszerek programozható logikai eszközökkel

7. VHDL FELADATOK:
Speciális nyelvi szerkezetek. Sorrendi hálózatok megvalósítása.

Frissítve: 2020. 12. 03.

VHDL - Speciális nyelvi szerkezetek és Sorrendi hálózatok megvalósításai

TRAFFIC LIGHT (JELZŐLÁMPA)

Feladat 1/a.)

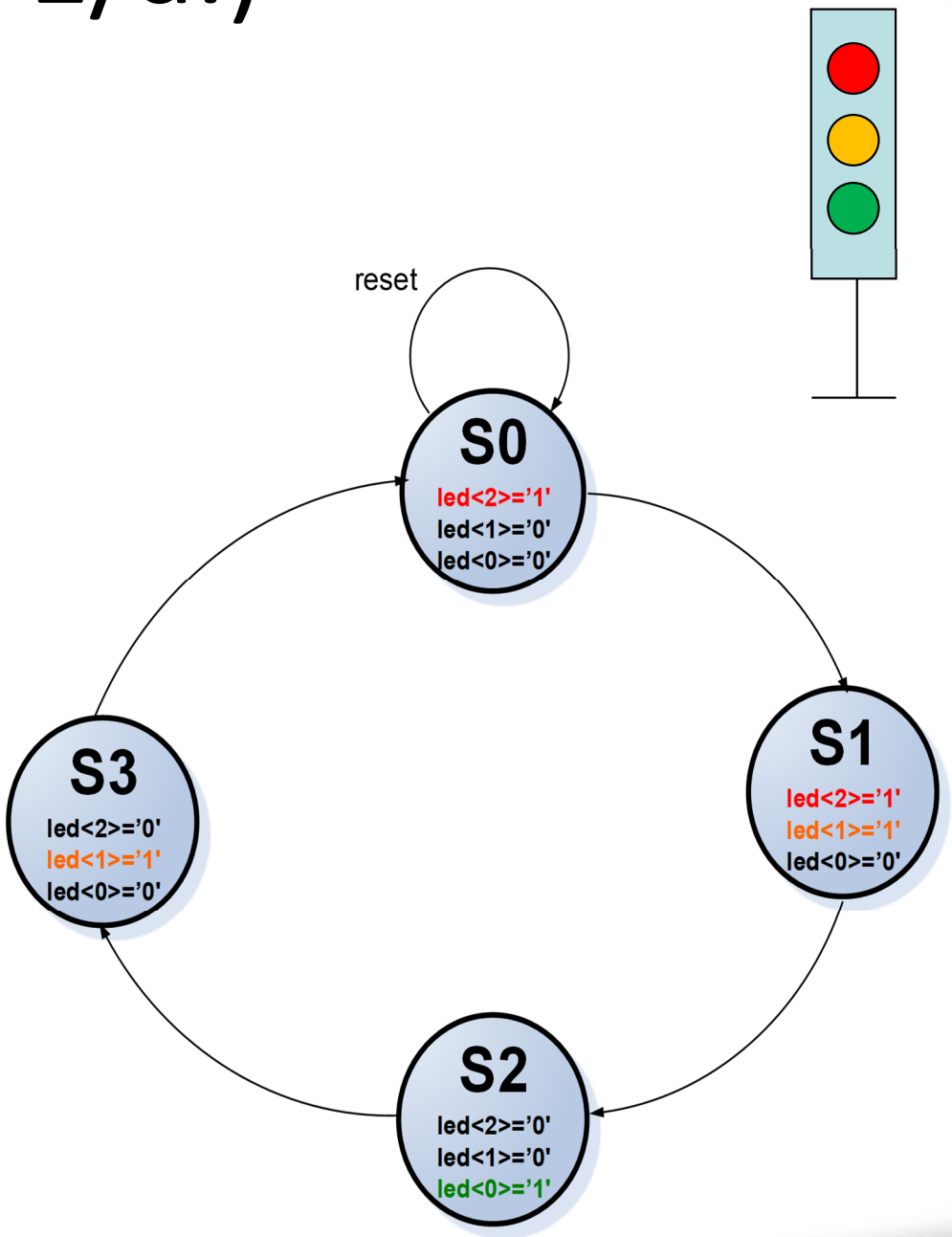
Tervezzen egy közlekedési lámpa (traffic light) vezérlőt a mellékelt állapotdiagram (FSM) alapján.

- Megj: megvalósításhoz használja a szinkron *Moore* S.H. modelljét („traffic_moore.vhd”)

- Készítsen egy test-bench-et hozzá: („traffic_moore_tb.vhd”)
- Szimulálja le a viselkedését (Vivado Sim)
- Példányosítsa a korábban megtervezett „*clkdiv.vhd*” (125 MHz -> ~1 Hz) modult és ezt a traffic_moore.vhd Moore modell alapú állapotgépet , majd pedig kösse össze őket egy új „top-level” modulban. („traffic_moore_top.vhd”)

- Szerkessze meg az .**XDC** filet /használja a 3 legbaloldalibb LED-et (*led* <2:0>), *mclk* külső 125 MHz órajelet, és a legjobboldalibb nyomógombot (btn(0)) a *reset*-hez. /

- Végül implementálja, majd tesztelje FPGA-n a top-modult. MIT A TAPASZTALAT ??



Feladat 1/a.) Megoldás: Moore modell

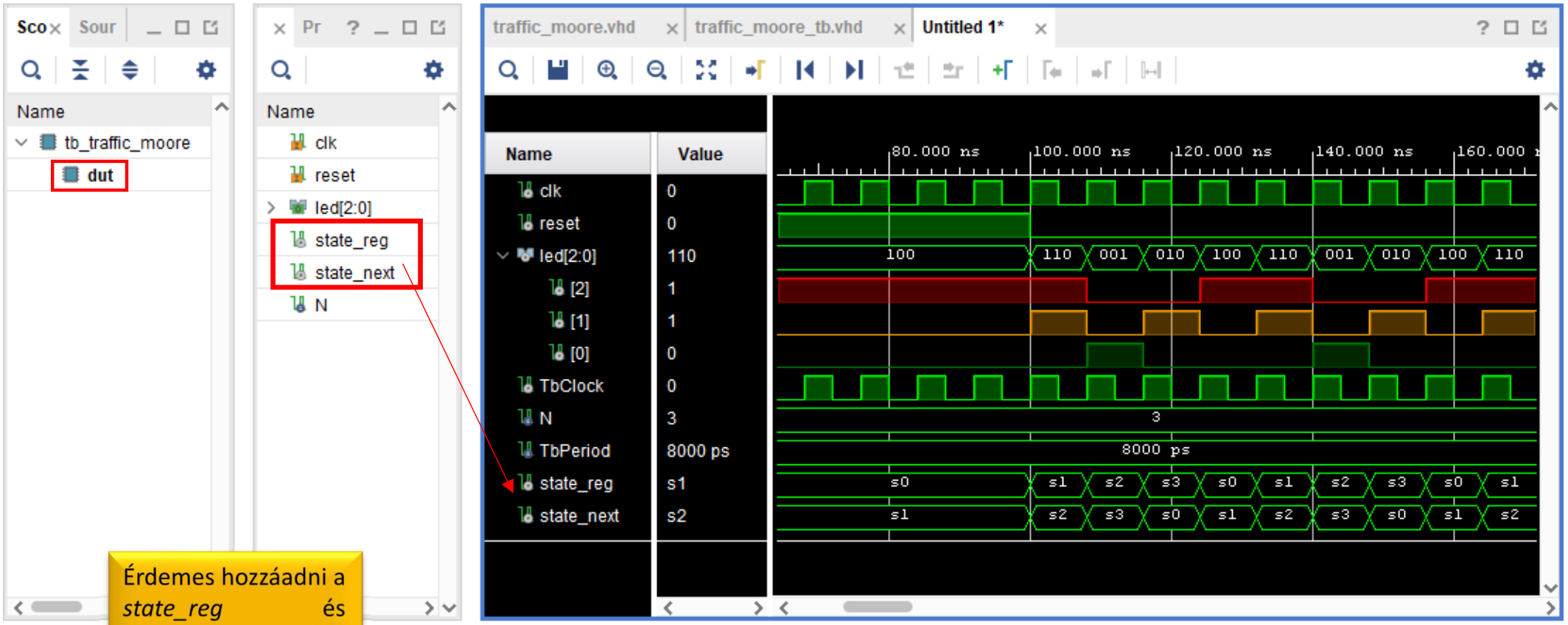
(traffic_moore.vhd)

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

entity traffic_moore is
generic (N: natural := 3);
port ( clk : in  STD_LOGIC;
       reset : in  STD_LOGIC;
       led : out  STD_LOGIC_VECTOR(N-1 downto 0));
end traffic_moore;

architecture Behavioral of traffic_moore is
    type traff_state_type is (s0, s1, s2, s3);
    -- P, PS, Z ,S -> P
    signal state_reg, state_next:
    traff_state_type;

begin
    -- state register
   process(clk, reset)
   begin
      if (reset='1') then
         state_reg <= s0;
      elsif rising_edge(clk) then
         state_reg <= state_next;
      end if;
   end process;
```

```
   -- next-state logic
   process(state_reg)
   begin
      case state_reg is
         when s0 =>
            state_next <= s1;
         when s1 =>
            state_next <= s2;
         when s2 =>
            state_next <= s3;
         when s3 =>
            state_next <= s0;
         end case;
   end process;

   -- Moore output logic (led2, led1, led0)   !
   process(state_reg)
   begin
      case state_reg is
         when s0 =>
            led(N-1 downto 0) <= "100"; --P
         when s1 =>
            led(N-1 downto 0) <= "110"; --PS
          when s2  =>
            led(N-1 downto 0) <= "001"; --Z
          when s3 =>
            led(N-1 downto 0) <= "010"; --S
      end case;
   end process;
end Behavioral;
```

Ebben a példában nincs lényeges különbség a Moore / Mealy megvalósítási modellek között. Ezért akár a kimeneti logika összevonható a next state logikával egy közös process()-be.

Feladat 1/a.) TestBench eredménye

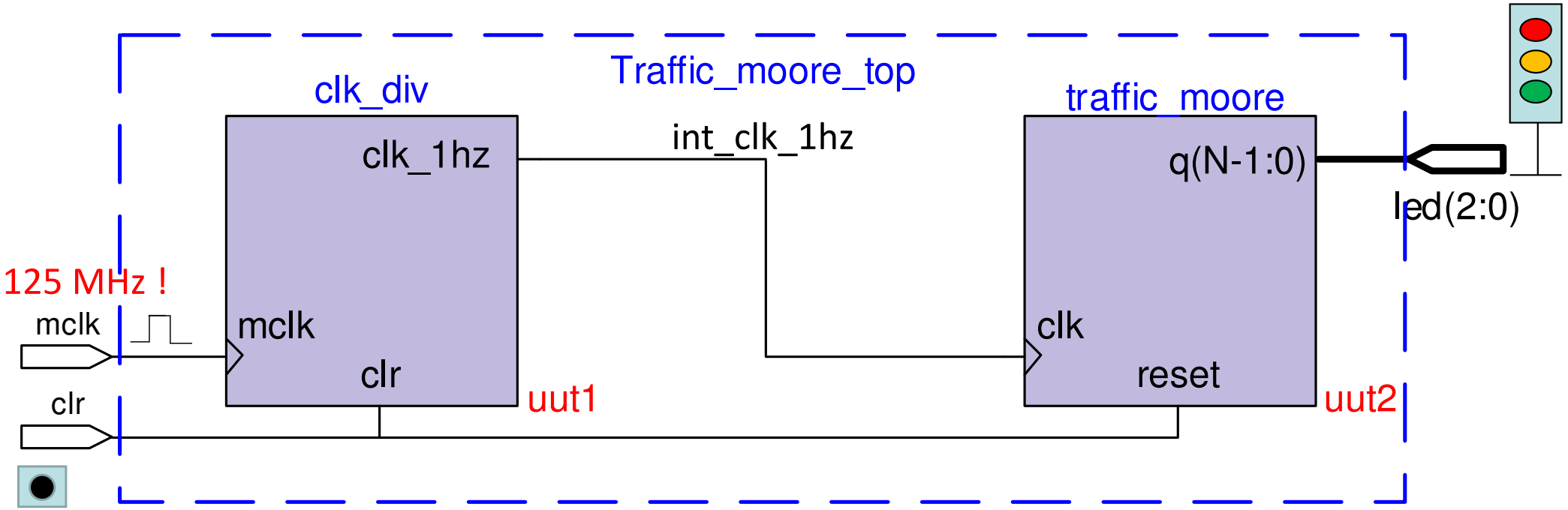


Érdemes hozzáadni a *state_reg* és *state_next* belső jeleket is, majd XSim Relaunch.

Feladat 1/a.) megoldása:

(„traffic_moore_top.vhd”)

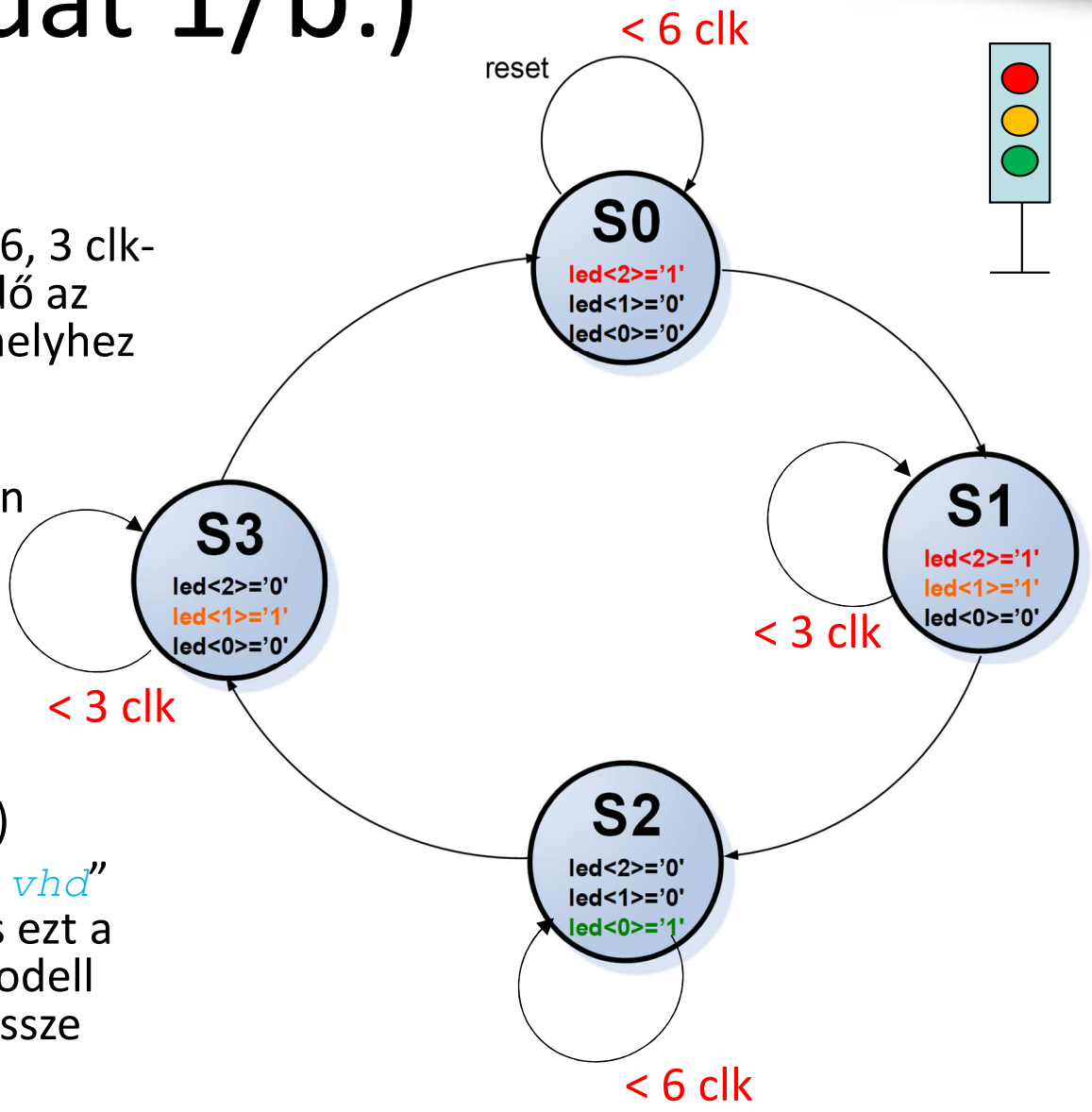
- Adjuk hozzá a clkdiv.vhd forrását a projekthez.
- Példányosítsuk a clkdiv és traffic_moore entitásokat a „top” szinten. Lásd alábbi ábra.



Feladat 1/b.)

Készítsen időzített közlekedési lámpa (**traffic_ light_delayed**) vezérlőt az állapotdiagram alapján, hogy az egyes állapotok (P→ PS → Z → S) rendre 6, 3, 6, 3 clk-val legyenek késleltetve. A várakozási idő az egyes állapotokban megadott legyen, melyhez használjon egy belső számlálót (`cnt`) és konstanst Pl. `C_SEC6=6, C_SEC3=3`.

- Megj: megvalósításhoz használja a szinkron *Moore* S.H. modellét („`traffic_moore_delayed.vhd`”)



- Készítsen egy test-bench-et hozzá: („`traffic_moore_delayed_tb.vhd`”)
- Szimulálja le a viselkedését (Vivado Sim)
- Példányosítsa a megtervezett „*`clkdiv.vhd`*” (125 MHz -> ~1 Hz, ha D:=27) modult és ezt a `traffic_moore_delayed.vhd` Moore modell alapú állapotgépet , majd pedig kösse össze őket a „top-level” modulban. („`traffic_moore_delayed_top.vhd`”)
- Implementálja FPGA-n. XDC: 3 *LED*, *mclk* - külső 125 MHz órajel, és *reset* nyomógomb.

125 MHz -> ~1Hz = váltás / sec

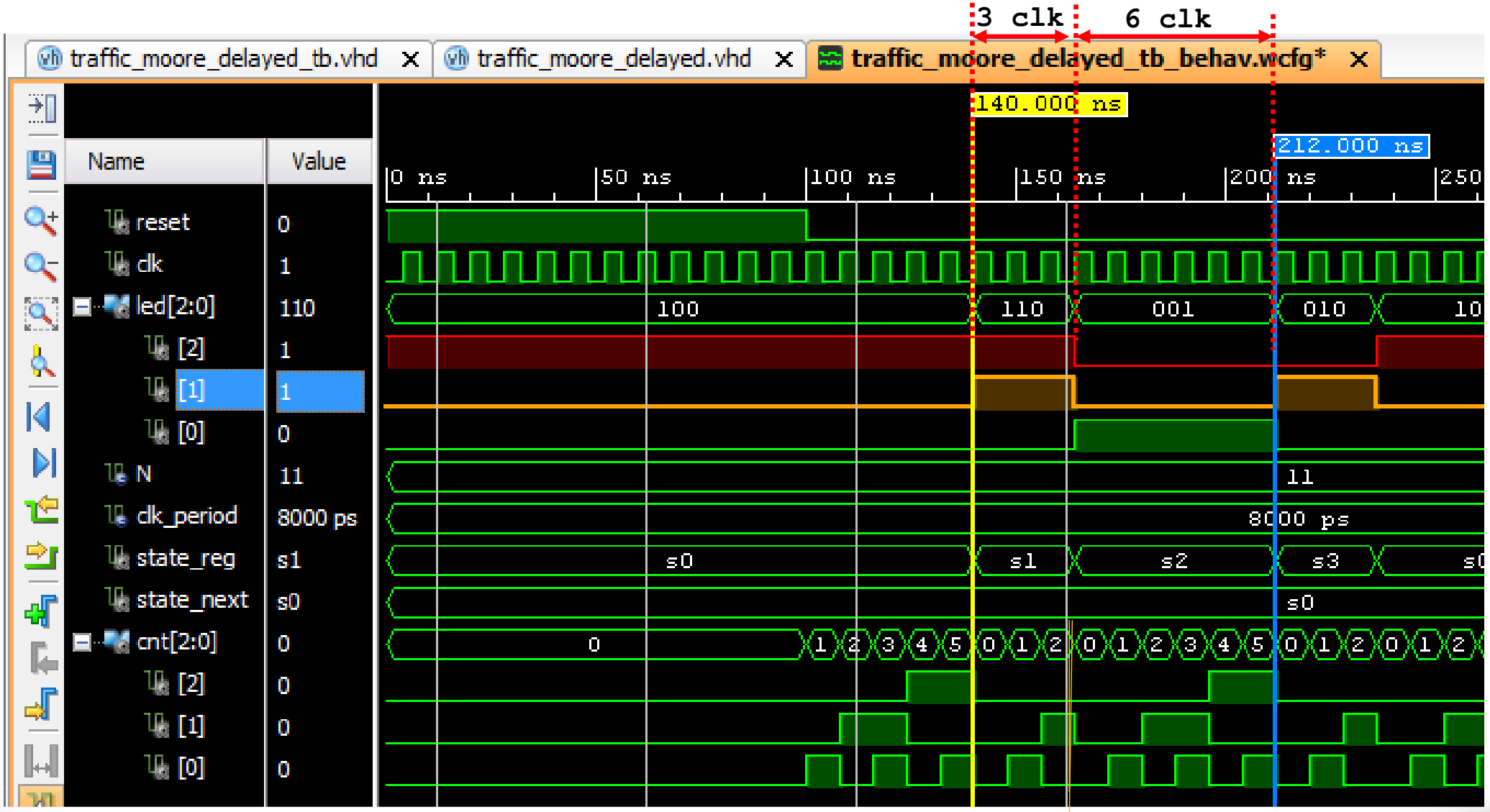
Feladat 1./b.) Megoldás I.: Moore modell

(traffic_moore_delayed.vhd)

```
use IEEE.STD_LOGIC_unsigned.all; -- „+”
. . .

architecture Behavioral of traffic_moore_delayed is
   type traff_state_type is (s0, s1, s2, s3); -- P, PS,
    Z ,S -> P
    signal state_reg : traff_state_type;
    -- delay
    constant C_SEC6: natural := 6;
    constant C_SEC3: natural := 3;
 !  constant C_BIT_WIDTH : integer :=
          integer(ceil(log2(real(C_SEC6))));
    signal cnt: STD_LOGIC_VECTOR(C_BIT_WIDTH downto
    0):= (others => '0');

begin
    -- state register
   process(clk, reset)
   begin
      if (reset='1') then
         state_reg <= s0;
     elsif rising_edge(clk) then
      case state_reg is
        when s0 =>
          if cnt < C_SEC6-1 then
            state_reg <= s0;
            cnt <= cnt + 1;
          else
            state_reg <= s1;
            cnt <= (others => '0');
          end if;
        when s1 =>
           if cnt < C_SEC3-1 then
            state_reg <= s1;
            cnt <= cnt + 1;
           else
            state_reg <= s2;
            cnt <= (others => '0');
           end if;
        . . .
        when s3 =>
           if cnt < C_SEC3-1 then
            state_reg <= s3;
            cnt <= cnt + 1;
           else
            state_reg <= s0;
            cnt <= (others => '0');
           end if;
      end case;
   end if;
end process;
-- Moore output logic (led2, led1, led0)  !
   változatlan!
.. .
```

PI: State register és a Next state logika össze lett vonva!

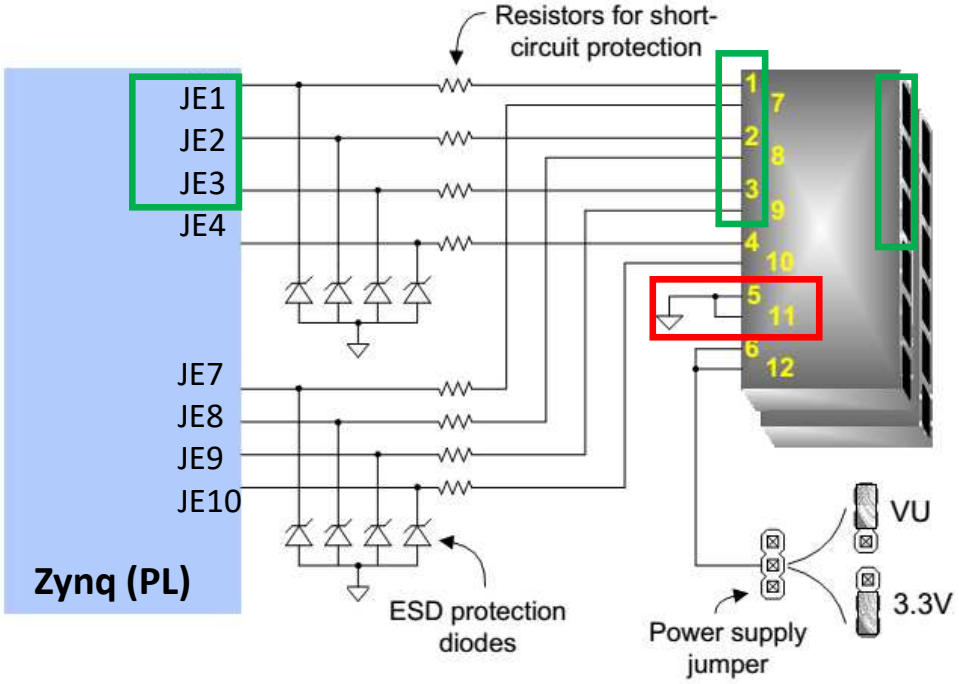
Szimulációs eredmény

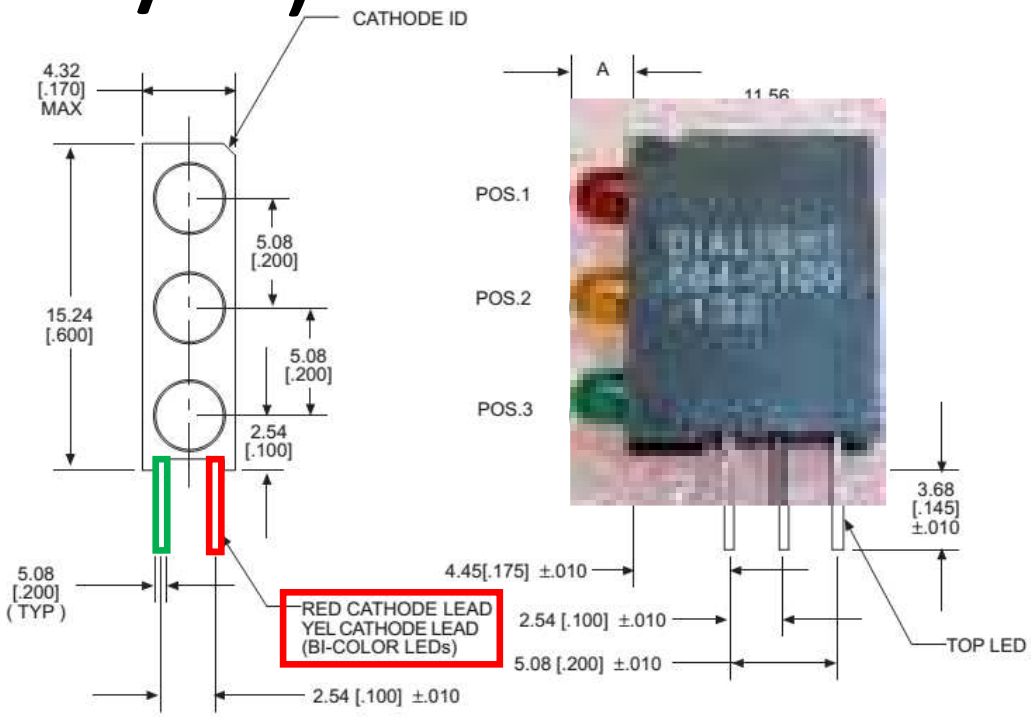


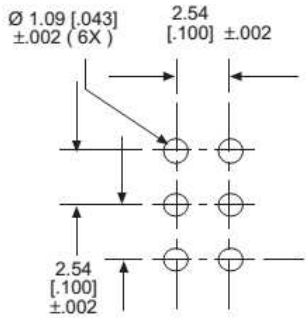
Feladat 1/c.)

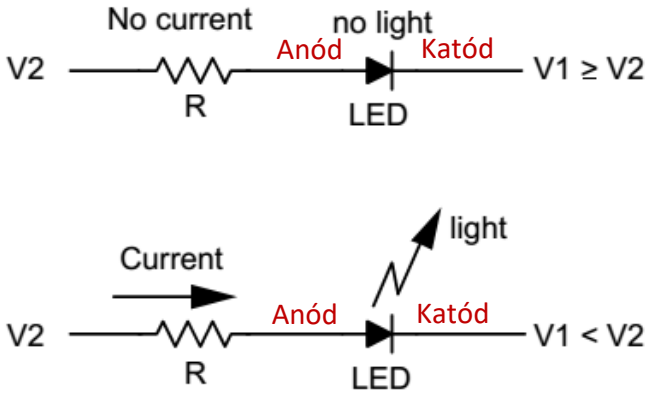
Tervezzen egy közlekedési lámpa (**traffic_light_delayed**) vezérlőt a Feladat 1/b.) alapján.

- Megj: megvalósításhoz használja a DiaLight *564-0100-132* Piros-Sárga-Zöld LED-eket tartalmazó jelző áramkört.
- Implementálja FPGA-n úgy, hogy mind a 3 LED-et, mind pedig a DiaLight áramkört használja. Ez utóbbit a PMOD **JE<2:0>** jelű lábakra kösse be. Az *mclk* - külső 125 MHz órajel, és a nyomógomb (*reset*) is legyen bekötve.









KÉRDÉS:

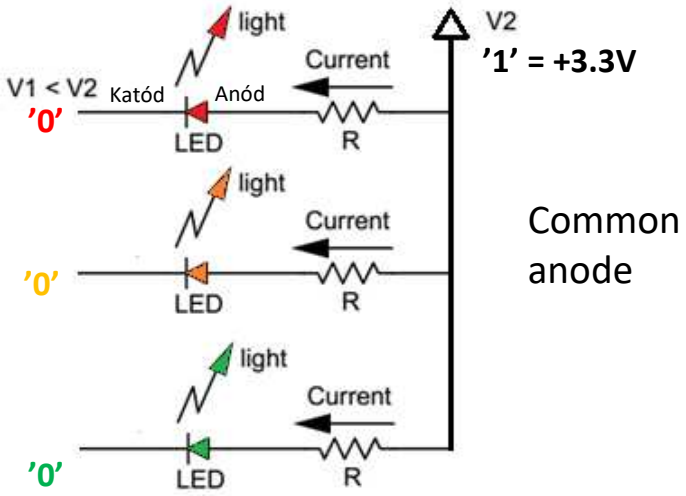
- Hogyan kell vezérelni a Dialight LED-jeit?

- Közös anódos, vagy közös katódos módszerrel?

DiaLight kijelzők

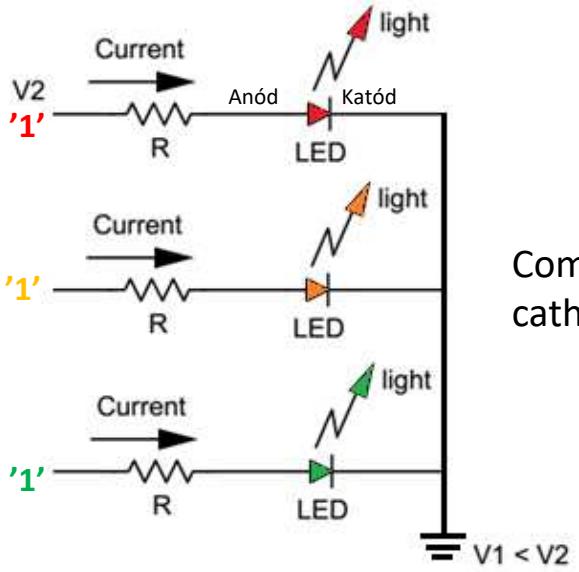
Válasz: mindkét megoldás lehetséges

- **Közös anódos**: ha az **anódok** kapnak **közös tápot** (+3.3V), míg a **katódok** ’0’-val vannak vezérelve (ez a negatív logika – alacsony aktív).



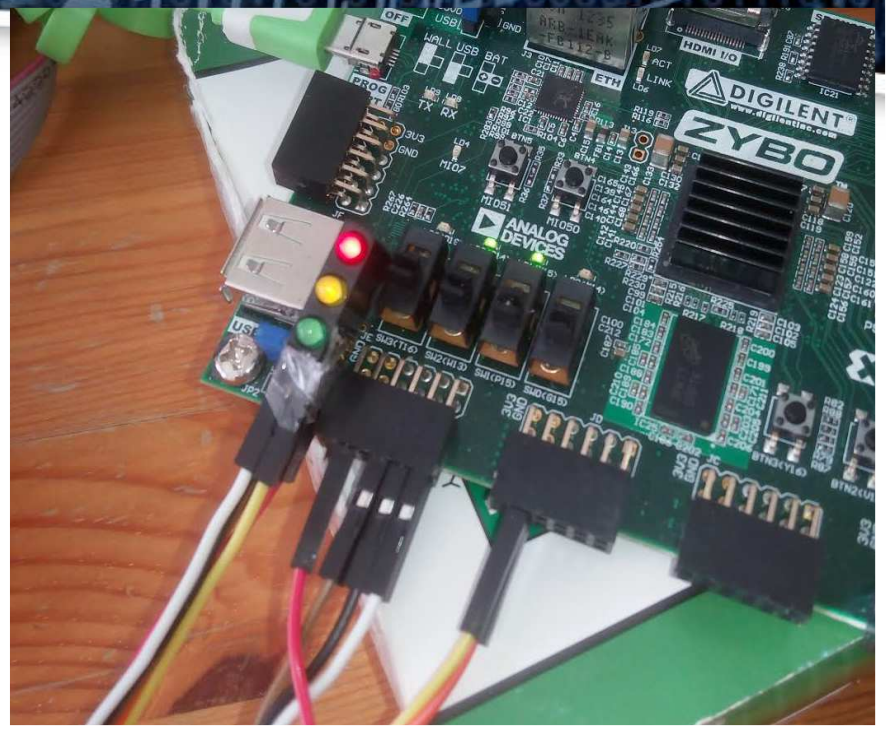
Common anode

- **Közös katódos**: ha az **anódok** ’1’-el vezérelve, míg a **katódok** kapnak **közös földet** (hagyományos pozitív logika – magas aktív).

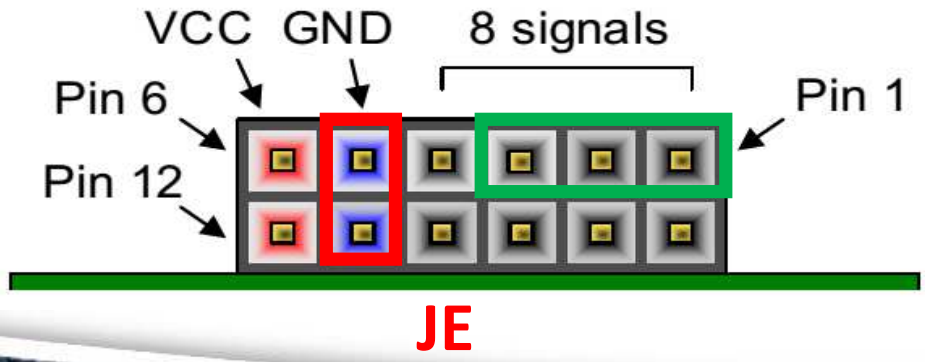


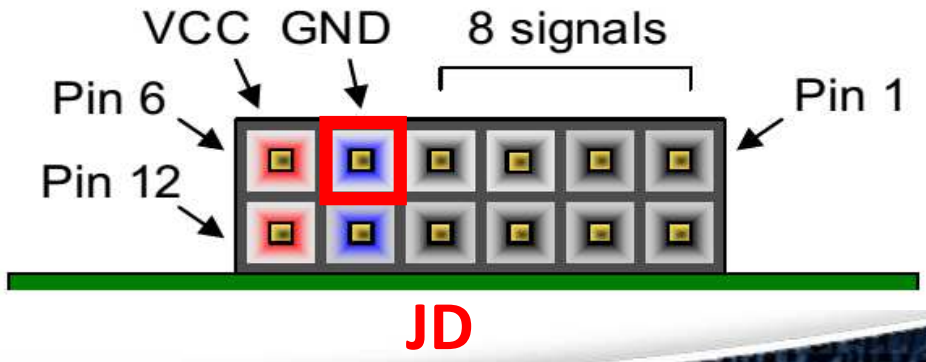
Common cathode

PMOD bekötés



- ZyBo: összesen 6 konnektor van (**5**+1)
- 1 konnektoron:
  - 8 pin + 2-2 GND, VDD pin
- Használjuk a **JE**-t, **JD**-t (ez a közös katódos eset!)
  - 3 GND + 3 pin kell!





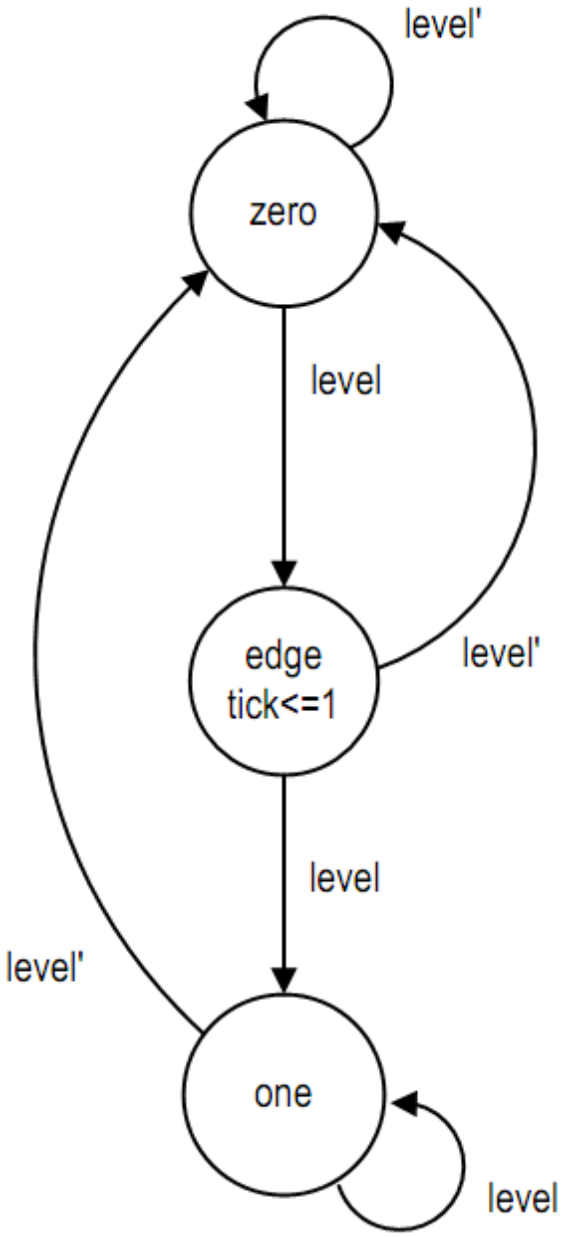
VHDL - Speciális nyelvi szerkezetek és Sorrendi hálózatok megvalósításai

MEALY / MOORE MODELLEK

Feladat 2.)

Tervezzen egy szinkron **Moore modell** alapú állapotgépet, amely a *felfutó éleket detektálja*, a mellékelt FSM diagram alapján. Ha felfutó élt detektált akkor tick <= ’1’.

- Megj: Használja a szinkron S.H. Moore modelljét. („`rising_edge_moore.vhd`”)
- level’ = $\overline{\text{level}}$
- Készítse el a tesztbench-et. („ `rising_edge_moore _tb.vhd`”)
- Szimulálja le a viselkedését XSim segítségével.

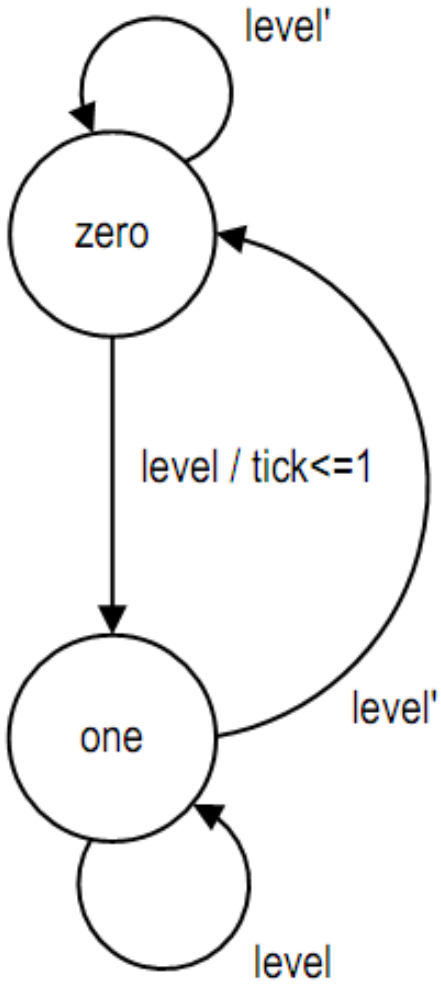


Feladat 3.)

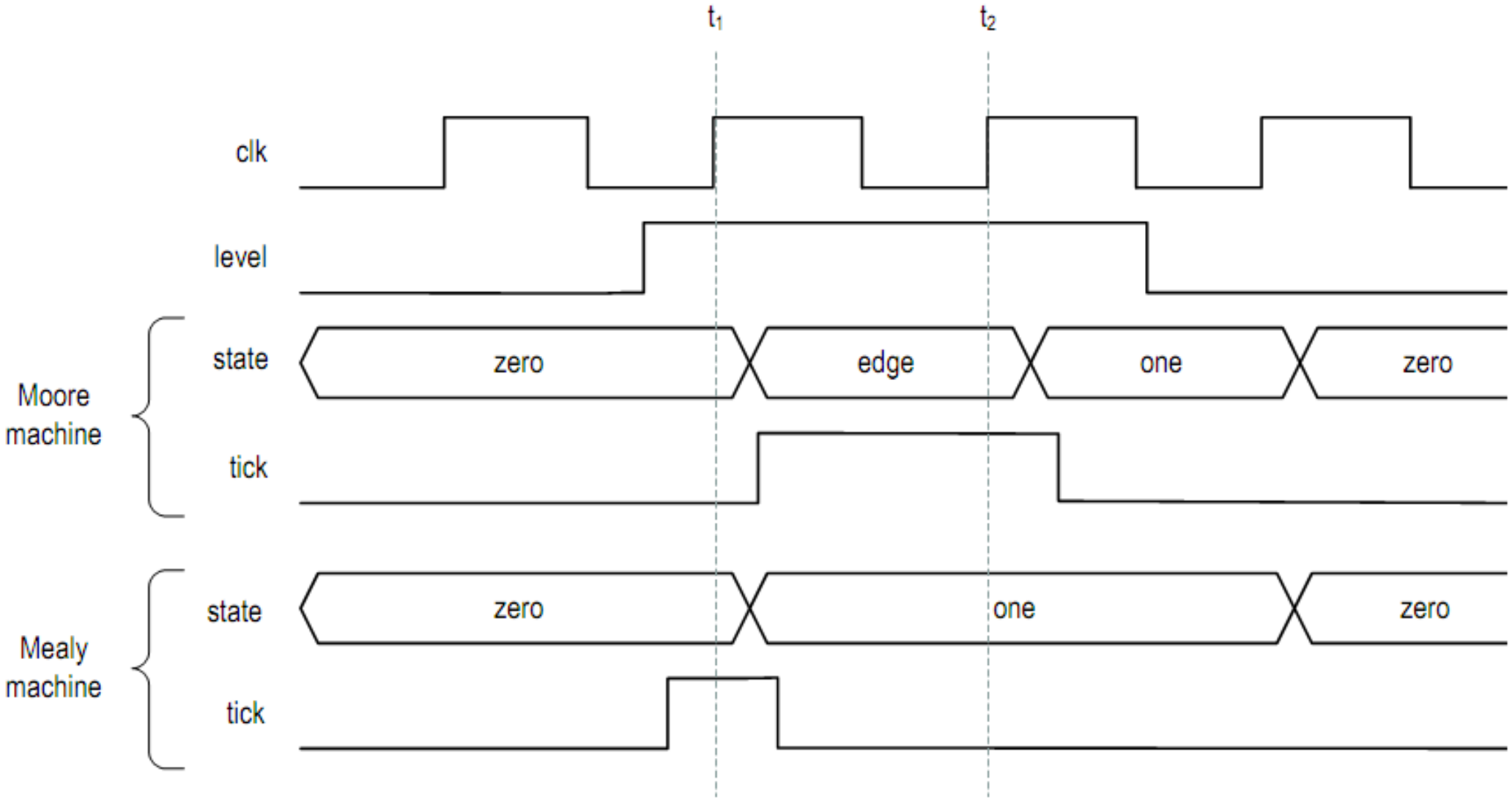
-Tervezzen egy szinkron **Mealy modell** alapú állapotgépet, amely a *felfutó éleket detektálja*, a mellékelt FSM diagram alapján. Ha felfutó élt detektált akkor tick <= ’1’.

- Megj: Használja a szinkron S.H. Mealy modelljét. („`rising_edge_mealy.vhd`”)
- level’ = $\overline{\text{level}}$

- Készítse el a tesztbench-et. („ `rising_edge_mealy_tb.vhd`”)
- Szimulálja le a viselkedését XSim segítségével.



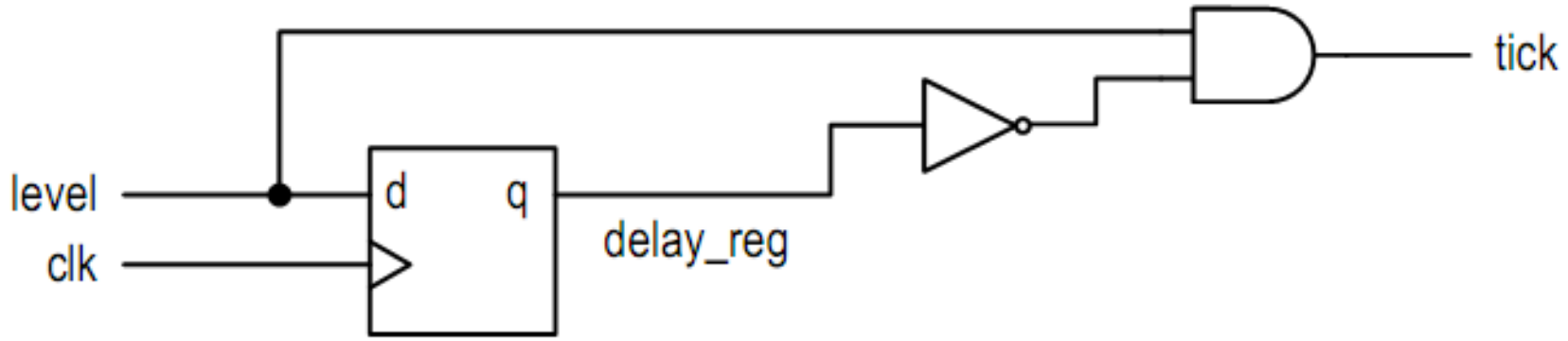
Moore / Mealy modellek időzítési diagramjai



Feladat 4.)

Tervezze meg a **felfutó-él detektor áramkör kapu-szintű VHDL (strukturális)** leírását, az alábbi ábra alapján.

- Megj: Tervezzen egy kombinációs hálózatot („`rising_edge_gate.vhd`”). tick <= ’1’, amikor felfutó élt detektált.
- Készítse el a top-modul testbench-ét. („`rising_edge_gate_tb.vhd`”)
- Szimulálja le a viselkedését Vivado Simulator-ban és hasonlítsa össze a S.H Mealy/Moore modelljének viselkedésével (Feladat 2-3.).



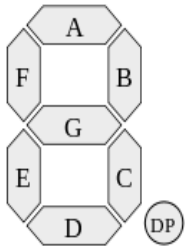
VHDL – Hierarchikus tervezés komplex áramkör esetén

7-SZEGMENSES KIJELZŐK

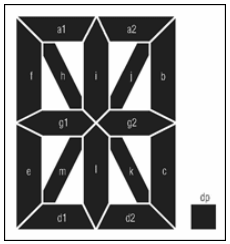
Kijelzők / Displays

- LED "0 / 1"

- 7+1 szegmenses kijelzők "0..9 / A ... F„



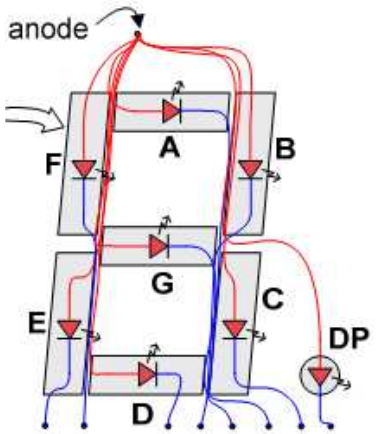
  - (dec/hexa) + diag. hibakódok jelzése.



- 15+1 szegmenses kijelzők
- Karakter LCD kijelzők
  (1 v. több soros pl. 16x2 character LCD) - pixel

- OLED karakter/grafikus kijelzők (1 v. több sornyi pixel)

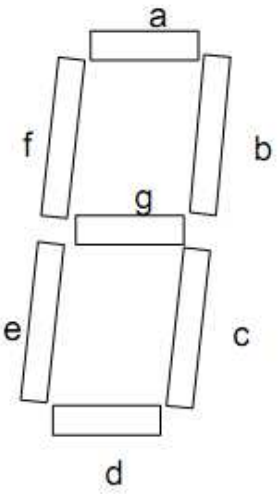
7-szegmenses kijelzők



Szegmensek 7+1!:
(a…g, dp)

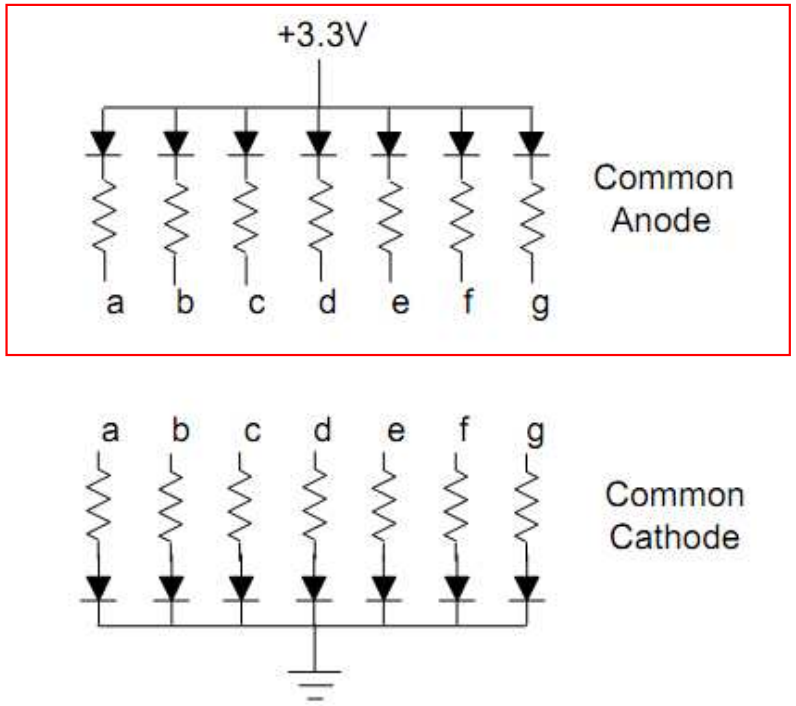
Közös anódos mód:

- ‘**0**’ kimenet esetén adott szegmens bekapcsolása - **ON**
  (a, … , g, dp),
- ’**1**’ kimenet esetén adott szegmens kikapcsolása - **OFF**.



1 = off

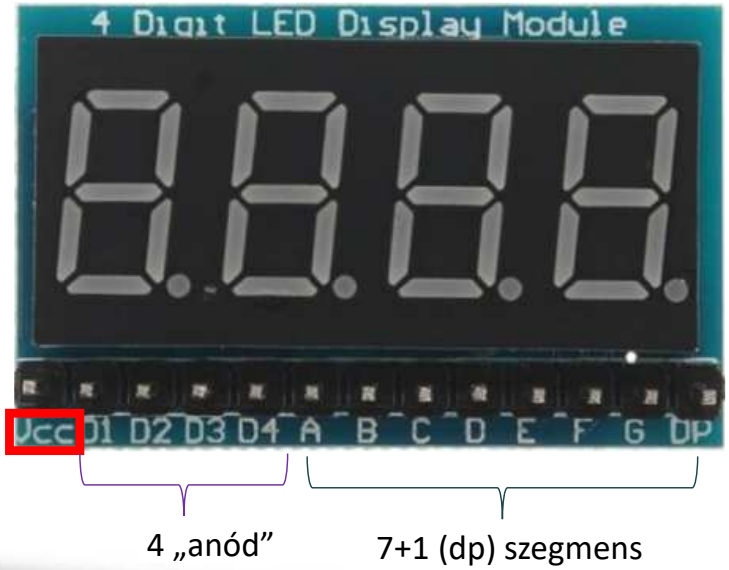
0 = on

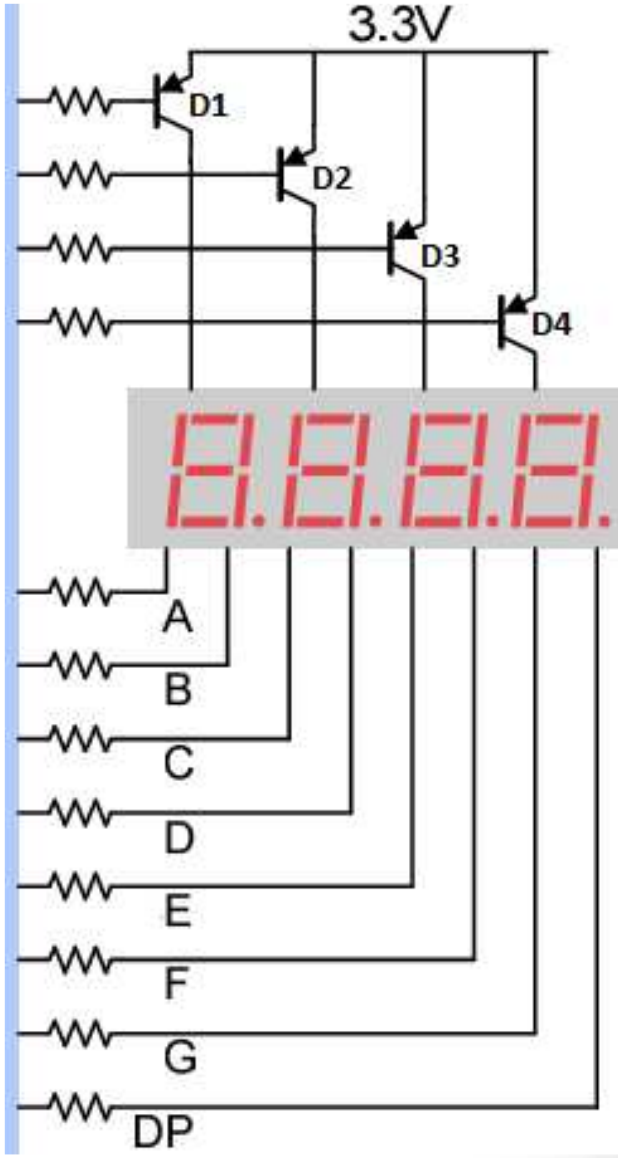


100Ω áramkorlátozó ellenállások

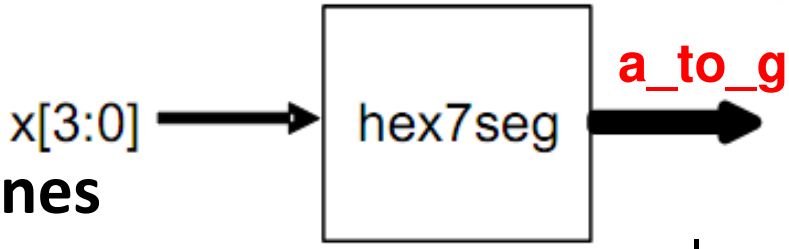
7-szemgenses kijelző

- 4 számjegy (digit D1…D4), azaz 4 „anód” vezérlése
  - 7+1 (dp) szegmens / számjegy
  - Közös táp (VCC 3V3)





Feladat 5/a.)

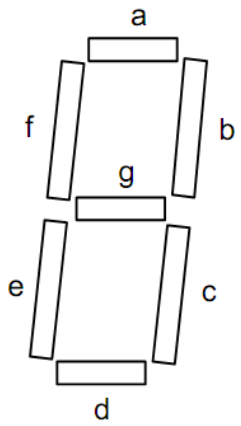


a.) Tervezzen meg egy **7-szegmenes kijelző dekódoló** áramkörét a mellékelt igazság táblázatnak megfelelően!

- Megj: ZyBo PMOD JE és JD-n keresztül a **közös-anódos** módszert alkalmazzuk az egyes szegmensek vezérelésére (’1’ = OFF, ’0’ = ON).
- Implementálja VHDL-ben („`hex7seg.vhd`”) vagy:
  - ***konkurens (pl. with…select)***, vagy
  - ***szekvenciális (pl. if…else, v. case ..)***
    VHDL szerkezeteket használva.
- Készítsen egy testbench-et („`hex7seg_tb.vhd`”) ehhez a modulhoz.

Szimulálja le Vivado Simulatorban

Digit:



1 = off

0 = on

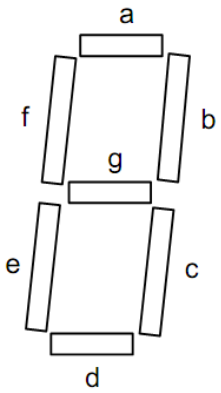
Igazság tábla:

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Megoldás Feladat 5/a.) (CASE-WHEN hex7seg.vhd)

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

entity hex7seg is
  port(
   x : in STD_LOGIC_VECTOR(3 downto 0);
   a_to_g : out STD_LOGIC_VECTOR(0 to 6)
     );
end hex7seg;
```

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0



1 = off

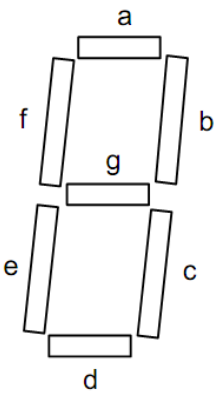
0 = on

```
architecture behav of hex7seg is
begin
    hex7seg_proc : process(x) is
          begin
   case x is
          --                              abcdefg
          when X"0" =>  a_to_g <= "0000001";  --0
          when X"1" =>  a_to_g <= "1001111";  --1
          when X"2" =>  a_to_g <= "0010010";  --2
          when X"3" =>  a_to_g <= "0000110";  --3
          when X"4" =>  a_to_g <= "1001100";  --4
          when X"5" =>  a_to_g <= "0100100";  --5
          when X"6" =>  a_to_g <= "0100000";  --6
          when X"7" =>  a_to_g <= "0001101";  --7
          when X"8" =>  a_to_g <= "0000000";  --8
          when X"9" =>  a_to_g <= "0000100";  --9
          when X"A" =>  a_to_g <= "0001000";  --A
          when X"B" =>  a_to_g <= "1100000";  --b
          when X"C" =>  a_to_g <= "0110001";  --C
          when X"D" =>  a_to_g <= "1000010";  --D
          when X"E" =>  a_to_g <= "0110000";  --E
          when others => a_to_g <= "0111000";  --F
       end case;
   end process;
end behav;
```

Megoldás Feladat 5/a.) (TEST_BENCH hex7seg_tb.vhd)

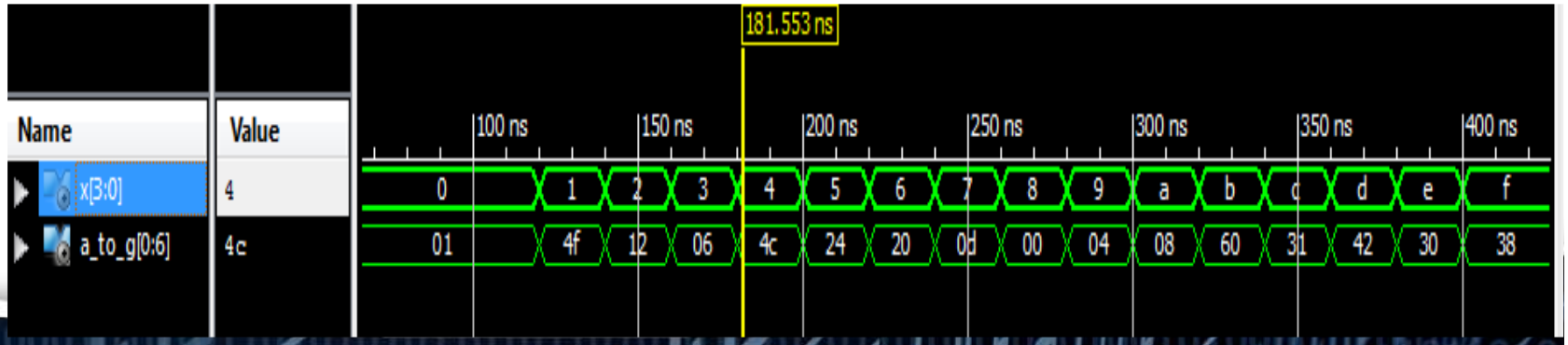
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all; --'+'

entity hex7seg_tb is
  port(
   x : in STD_LOGIC_VECTOR(3 downto 0);
   a_to_g : out STD_LOGIC_VECTOR(0 to 6)
      );
end hex7seg_tb;

architecture behav of hex7seg_tb is
. . .
stim_proc: process
   begin
      x <= (others => '0');
          wait for 100 ns;
      for i in 0 to (2**4-1) loop
          x <= x + 1; --std_logic_unsigned package!
          wait for 20 ns;
      end loop;

      wait;
   end process;
. . .
```

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0



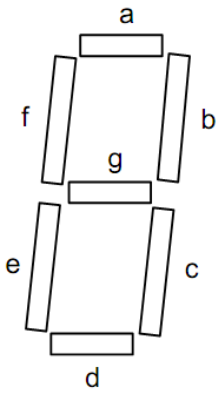
1 = off

0 = on



Megjegyzés Feladat 5/a.) (*Xilinx sablon alapján!)

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

entity hex7seg is
  port(
   x : in STD_LOGIC_VECTOR(3 downto 0);
   g_to_a : out STD_LOGIC_VECTOR(6 downto 0)
      );
end hex7seg;
```

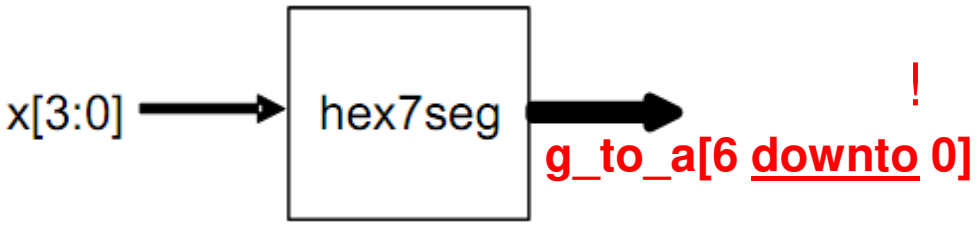


1 = off

0 = on

```
architecture behav of hex7seg is
begin
 with x select
  g_to_a <= "1111001" when "0001",    --1
            "0100100" when "0010",    --2
            "0110000" when "0011",    --3
            "0011001" when "0100",    --4
            "0010010" when "0101",    --5
            "0000010" when "0110",    --6
            "1111000" when "0111",    --7
            "0000000" when "1000",    --8
            "0010000" when "1001",    --9
            "0001000" when "1010",    --A
            "0000011" when "1011",    --b
            "1000110" when "1100",    --C
            "0100001" when "1101",    --d
            "0000110" when "1110",    --E
            "0001110" when "1111",    --F
            "1000000" when others;    --0
end behav;
```

!



!

Xilinx beépített nyelvi template is használható (de ekkor Little Endian!): Language Templates

VHDL → Synthesis Constructs → Coding Examples → Misc → „7-segment display HEX Conversion”

Feladat 5/b.)

folytatás.

- b.) Példányosítsa az előző „hex7seg.vhd” modult és kösse össze a megfelelő bemeneteket és kimeneteket a top-modullal („hex7seg_top.vhd”)

.xdc file megadása:

- Implementálja a tervet FPGA-n, ha a 4 kapcsoló = x(3:0) a bemenet, míg a kiement (a_to_g(0:6)) = A, B, C, … G szegmensek.
- Anódokat kösse egy-egy nyomógombra, és a lenyomottat NE jelenítse csak meg:
  - Btn(0) = D1, btn(1) = D2 ….. btn(3) = D4.
- Ne jelenjen meg a dp-decimális pont (OFF) (azaz dp = ‘1’). (AN(0) azaz D4 = ’0’) .

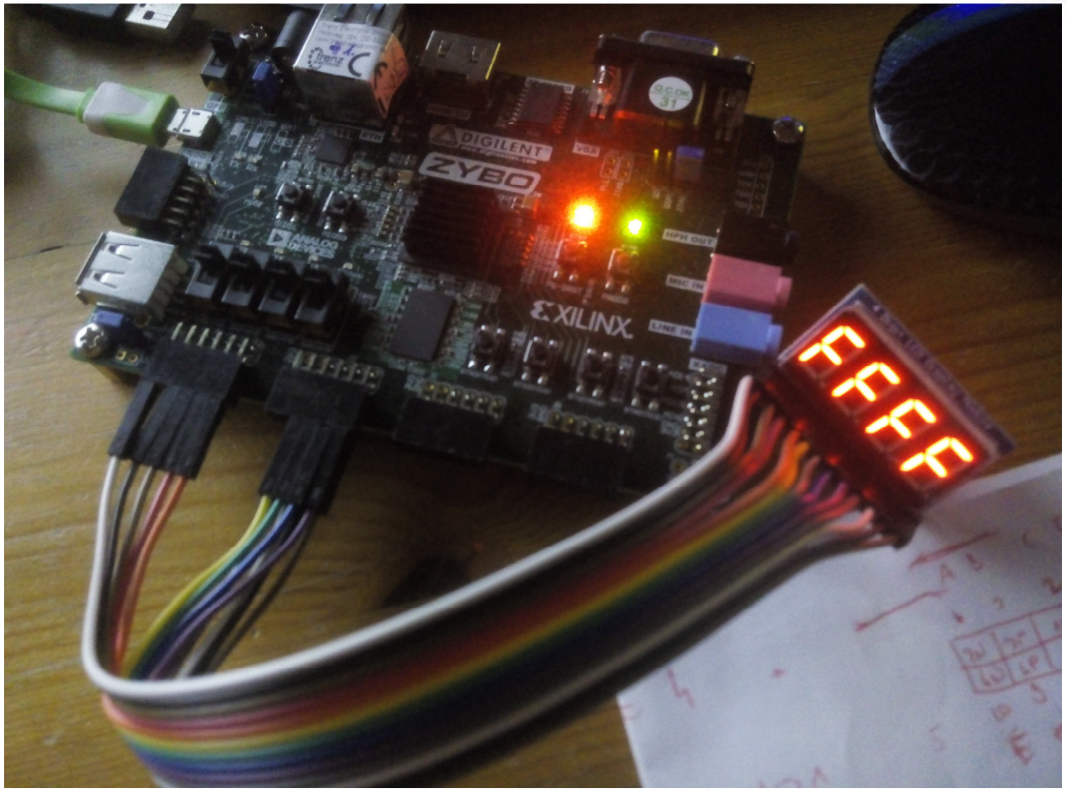
```
entity hex7seg_top is
port(
   sw : in STD_LOGIC_VECTOR(3 downto 0);
   a_to_g : out STD_LOGIC_VECTOR(0 to 6);
   an : out STD_LOGIC_VECTOR(3 downto 0);
   dp : out STD_LOGIC);
end seg7test;

-- dp = ’1’, azaz dp legyen kikapcsolva
-- an = ”0000” minden anód bekapcsolása a
4 számjegy (digit) megjelenítéséhez
--   D4   =   an(0)   =   ’0’,   pl.   a
legjobboldalibb anód bekapcsolása, D1 =
an(3) a legbaloldalibb
```

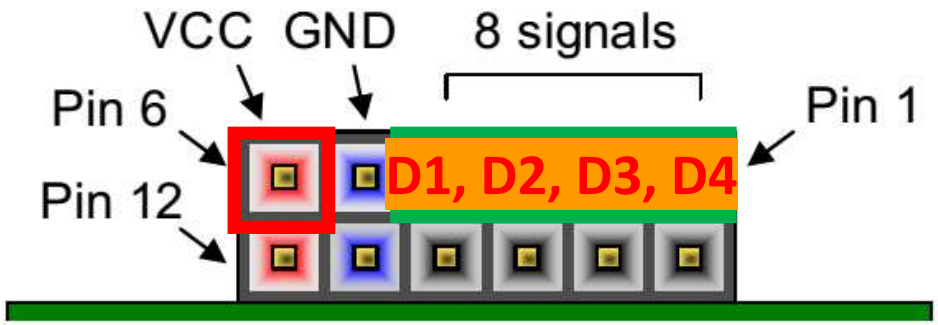
Megoldás Feladat 5/b.) (hex7seg_top.vhd)

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

entity hex7seg_top is
  port(
   sw : in STD_LOGIC_VECTOR(3 downto 0);
   btn : in STD_LOGIC_VECTOR(3 downto 0);
   --a_to_g : out STD_LOGIC_VECTOR(0 to 6);
   A, B, C, D, E, F, G : out STD_LOGIC;
   an : out STD_LOGIC_VECTOR(3 downto 0); -- D1 D2 D3 D4
   dp : out STD_LOGIC
      );
end seg7test;

architecture hex7seg_top of hex7seg_top is
component hex7seg is
  port(
   x : in STD_LOGIC_VECTOR(3 downto 0);
   a_to_g : out STD_LOGIC_VECTOR(0 to 6)
      );
end component;
   signal a_to_g_sig : STD_LOGIC_VECTOR(0 to 6);
begin
an <= btn;        -- számjegy (digit) ki/bekapcsolva (OFF='1')
dp <= '1';        -- decimális pont kikapcsolva (OFF='1')

hex7seg_uut: entity work.hex7seg(Behavioral)
 port map (x => sw,
           a_to_g => a_to_g_sig
           );

   A <= a_to_g_sig(0);
   B <= a_to_g_sig(1);
   C <= a_to_g_sig(2);
   D <= a_to_g_sig(3);
   E <= a_to_g_sig(4);
   F <= a_to_g_sig(5);
   G <= a_to_g_sig(6);

end hex7seg_top;
```

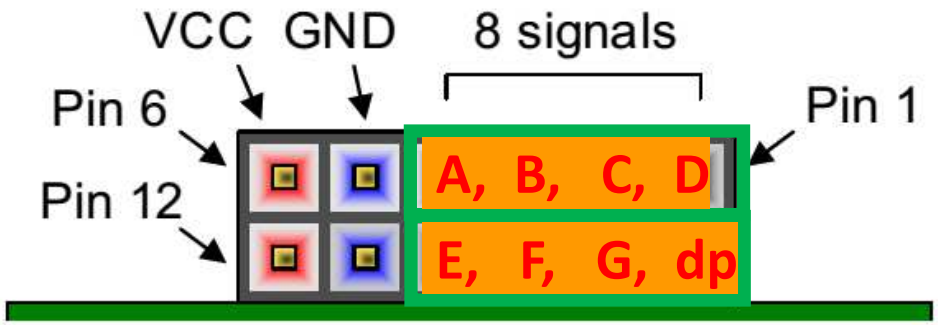
PMOD bekötés

- ZyBo: 2 konnektor kell
  - JE (VCC + anódok)
  - JD (8 szegmens A,B…dp)
- közös anódos eset!





PMOD_JE

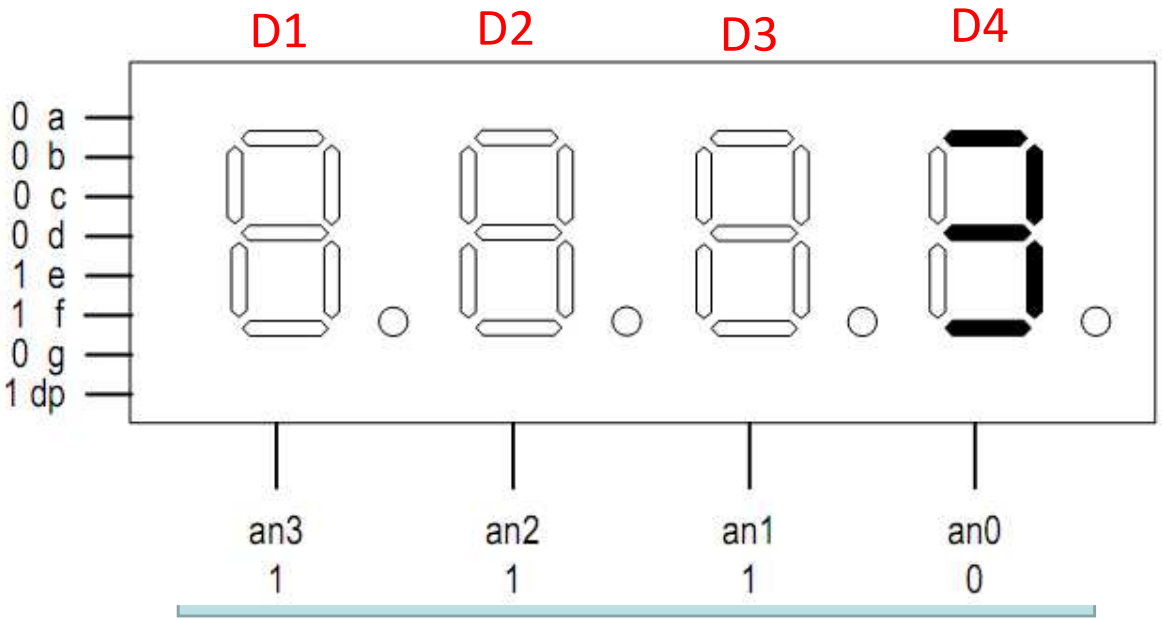


PMOD_JD

Idő-multiplexált 7-szegmenses kijelző

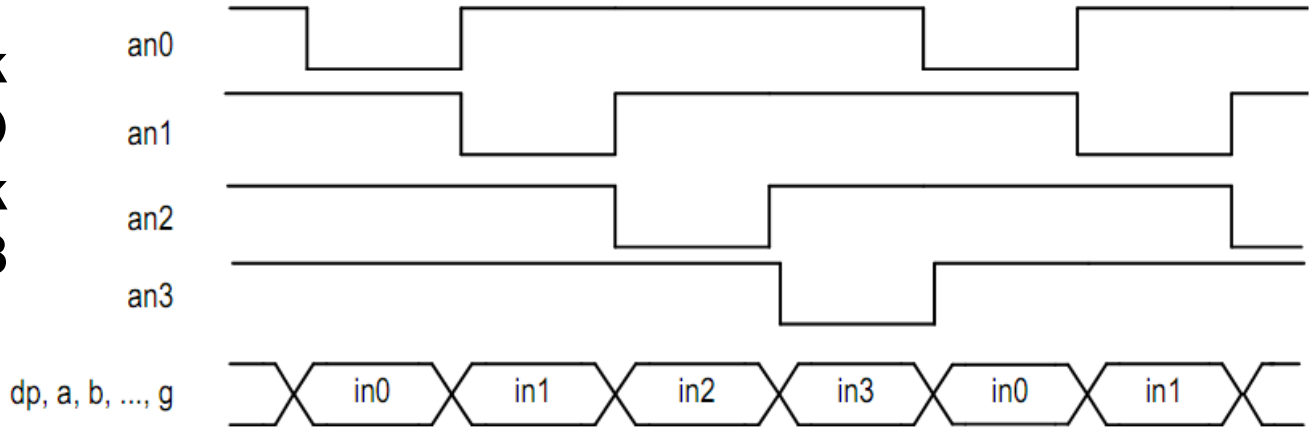
Ha a 4 független anód (an), mint engedélyező jel periodikus frissítési rátája (frekvenciája) „elegendően gyors” (f > ~**100-150 Hz**), az emberi szem már nem tud különbséget tenni az *ON* és *OFF* állapotok között (egyidejű, folyamatos megjelenítéssel érzékelve a kijelzőket).

Láb redukció (anód-ok multiplex-álásának oka): I/O lábak számának csökkentése 8+4 (a 32 = 4x8 helyett)!



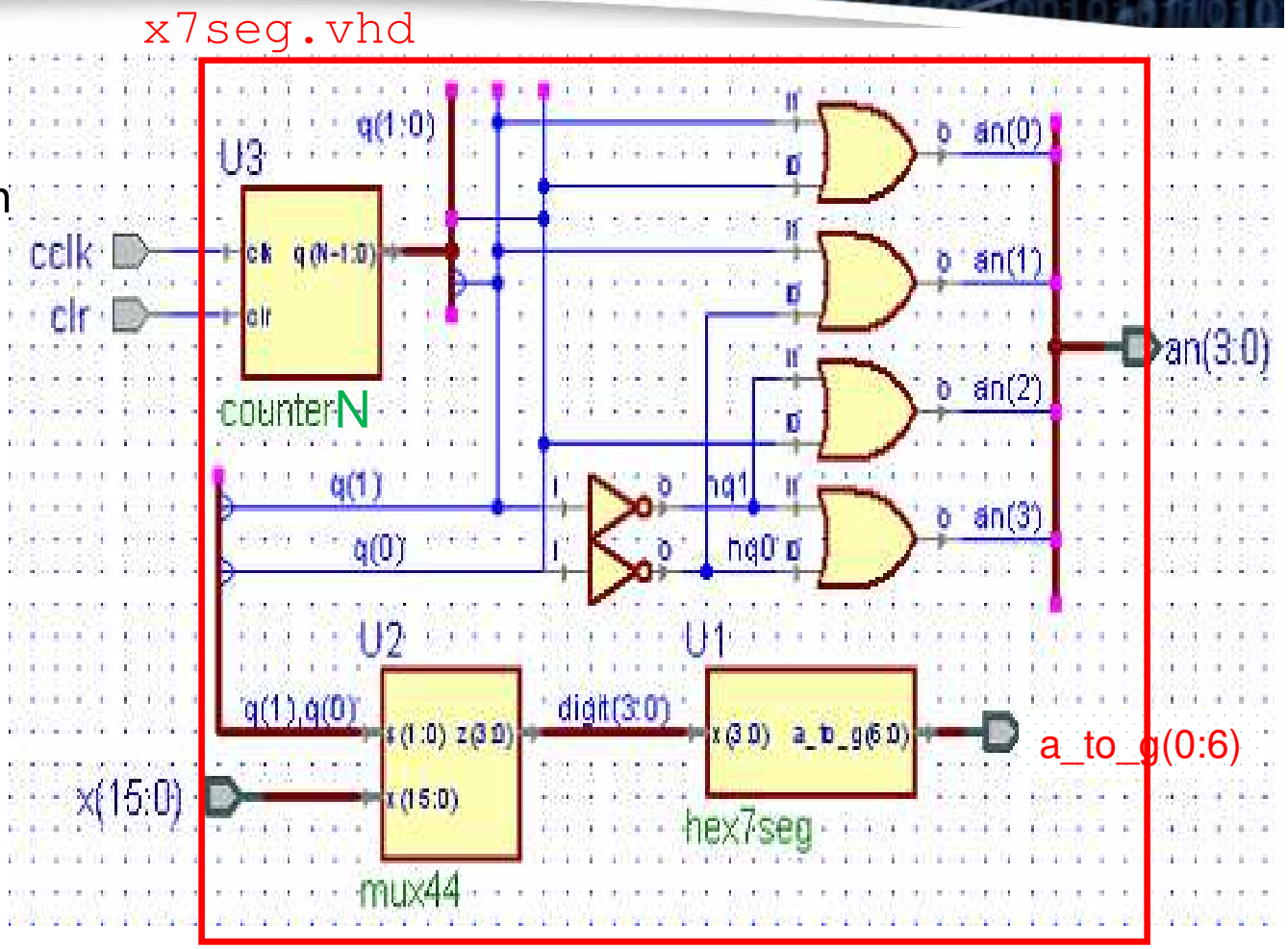

```
when "0011" => a_to_g <= "0000110";    --3
an <= "1110";
```

!



Feladat 6.)

- A mellékelt ábrának megfelelően tervezzen egy olyan VHDL top-modult (`x7seg.vhd`), amely a beérkező különböző hexadecimális értékeket (pl. „`1E2F`” megjeleníti a 4 db 7-szegmenses kijelzőkön.
- Megj: a Feladat 4.)-ben szereplő (`hex7seg.vhd`) modult példányosítsa.
  - Először tervezzen majd példányosítson egy *ún. quad* (azaz egyenként 4-bites bemenetekkel rendelkező) *4:1 MUX-ot*. (`mux44.vhd`). Lehet a korábbi 4:1 alapján is!
  - Példányosítson egy N=2-bites számlálót (`counterN.vhd`).
  - Használjon generic-et.
- Ezek alapján állítsa össze VHDL-ben a top-modult. (`x7seg.vhd`).
- Készítsen testcbench-et és szimulálja a működését Vivado-ban. (`x7seg_tb.vhd`).



Megj: az x(15:0) 16-bites bemenetet fel kell osztani 4db egyenként 4-bites bemeneti értékre, amely értékek a 7-szegmenses kijelző független számjegyein külön-külön kerülnek megjelenítésre!

Feladat 6.) folyt.

q(1)	q(0)	an(3)	an(2)	an(1)	an(0)
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Megjegyzés: a mellékelt táblázat alapján a q(1:0) kimenetek jelentik a 2-bites számláló kimeneti értékeit, melyeket a *4:1-es quad MUX* s(1:0) szelektor jeleihez kell kapcsolni, azért hogy azok folyamatosan, de periodikusan válasszanak a négy (0 … 3) egyenként 4-bites bemenetek közül (125 MHz frekvencián).

- Ugyanekkor *an(3:0)* anód (D1…D4) kimeneteit szinkronizálni kell az *s(1:0)* szelektor jelekkel, azért hogy a megfelelő hexadecimális számjegy (digit) a megfelelő időpillanatban jelenjen meg.

- Korábbi .xcd filet remove-olja a projektből!

- Implementálja a VHDL terveket FPGA-n, a 4db 7-szegmenses kijelző használatával, *125 MHz*-es órajelen (**cclk**) és használja a legjobboldalibb nyomógombot (**clr**) a reset-eléshez, míg **dp**, **an**(3:0) (D1, … D4) és **a_to_g**(0:6) is kerüljön az *x7seg.xdc* fileba./

Mit tapasztalunk ?!

Megoldás Feladat 6.)

(x7seg.vhd)

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

entity x7seg is
  port(
       cclk : in STD_LOGIC;
       clr : in STD_LOGIC;
       --x : in STD_LOGIC_VECTOR(15 downto 0);
       --a_to_g : out STD_LOGIC_VECTOR(0 to 6);
       A, B, C, D, E, F, G, dp : out STD_LOGIC;
       an : out STD_LOGIC_VECTOR(3 downto 0);
       dp : out  STD_LOGIC
  );
end x7seg;

architecture struct of x7seg is
component counterN
  generic(
       N : natural:= 2
  );
  port (
       clk : in STD_LOGIC;
       clr : in STD_LOGIC;
       q : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

component hex7seg
  port (
       x : in STD_LOGIC_VECTOR(3 downto 0);
       a_to_g : out STD_LOGIC_VECTOR(0 to 6)
  );
end component;
```

```
component mux44
  port (
       s : in STD_LOGIC_VECTOR(1 downto 0);
       in1,in2,in3,in4 : in STD_LOGIC_VECTOR(15 downto 0);
       y : out STD_LOGIC_VECTOR(3 downto 0)
  );
end component;
signal x :  STD_LOGIC_VECTOR (15 downto 0) := X"1E2F";
signal nq0, nq1 : STD_LOGIC;
signal digit : STD_LOGIC_VECTOR(3 downto 0);
signal q : STD_LOGIC_VECTOR(1 downto 0);
signal a_to_g_sig : STD_LOGIC_VECTOR(0 to 6);

begin
U1 : entity work.hex7seg(Behavioral)
  port map( a_to_g => a_to_g, x => digit  );
U2 : entity work.mux44(Behavioral)
  port map(
       s(0) => q(0),  s(1) => q(1),
       in1 => x(3 downto 0), in2 => x(7 downto 4),
       in3 => x(11 downto 8),in4 => x(15 downto 12),
       y => digit );
U3 : entity work.counterN(Behavioral)
  generic map(N => 2)
  port map(
       clk => cclk,
       clr => clr,
       q => q( N-1 downto 0 )
  );
      nq1 <= not(q(1));
      nq0 <= not(q(0));

      an(0) <= q(0) or q(1);
      an(1) <= nq0 or q(1);
      an(2) <= q(0) or nq1;
      an(3) <= nq0 or nq1;

      A <= a_to_g_sig(0);
      B <= a_to_g_sig(1);
      .. .
      dp <= '1'; --not displayed
end struct;
```

Órajel frekvencia leosztása

- f: bemeneti frekvencia (ZyBo @ **125** MHz – **L6** láb)
- q(i): kimeneti számított frekvencia képlet alapján:

$$f_i = \frac{f}{2^{i+1}}$$

Mekkora legyen a D értéke?

q(i)	Frekvencia (Hz)	Periódus (ms)
i	125000000.00	0.000008
0	62500000.00	0.000016
1	31250000.00	0.000032
2	15625000.00	0.000064
3	7812500.00	0.000128
4	3906250.00	0.000256
5	1953125.00	0.000512
6	976562.50	0.001024
7	488281.25	0.002048
8	244140.63	0.004096
9	122070.31	0.008192
10	61035.16	0.016384
11	30517.58	0.032768
12	15258.79	0.065536
13	7629.39	0.131072

D=1

14	3814.70	0.262144
15	1907.35	0.524288
16	953.67	1.048576
17	476.84	2.097152
18	238.42	4.194304
19	119.21	8.388608
20	59.60	16.777216
21	29.80	33.554432
22	14.90	67.108864
23	7.45	134.217728
24	3.73	268.435456
25	1.86	536.870912
26	0.93	1073.741824

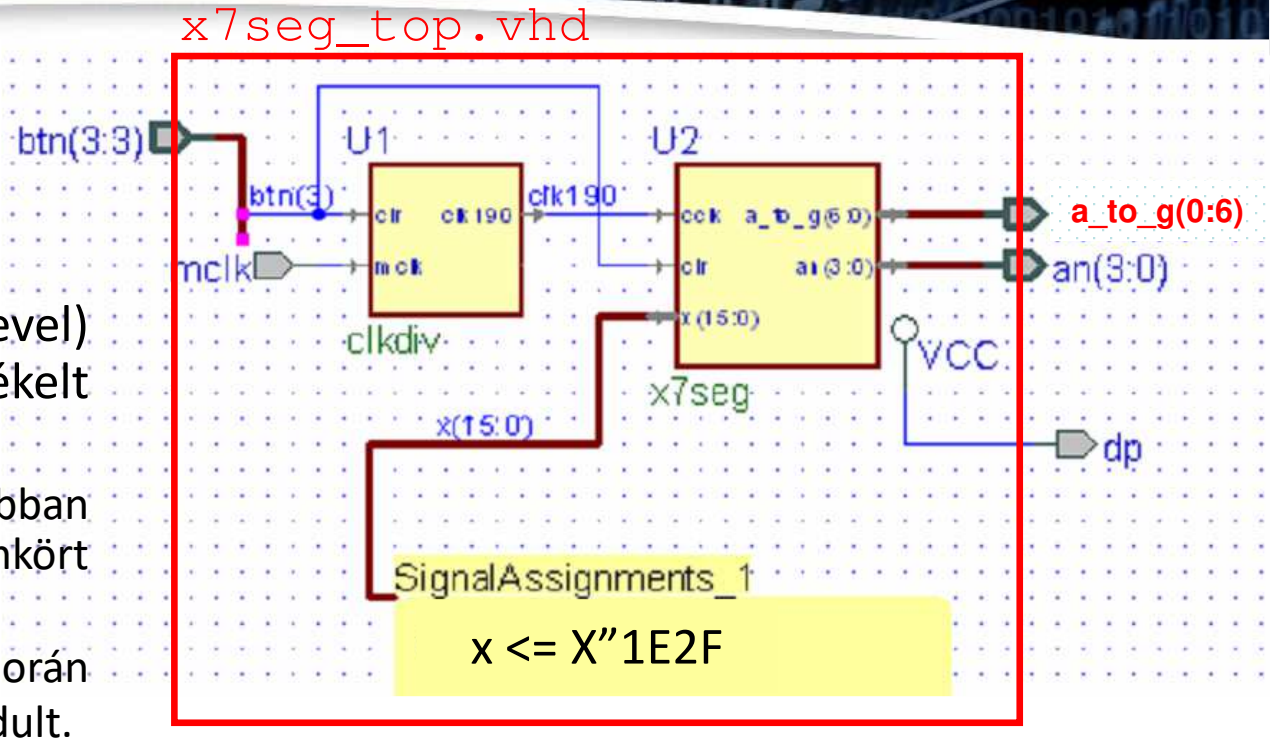
ok

D=27

Feladat 7.)

x7seg_top.vhd

- Végül a legfelsőbb szintű (top-level) VHDL modult tervezze meg a mellékelt ábra alapján.
  - Példányosítsa a korábban megtervezett órajel osztó áramkört (`clkdiv.vhd`)
  - Majd példányosítsa a Feladat 6.) során megtervezett (`x7seg.vhd`) modult.
- Kapcsolja őket össze a mellékelt ábrának megfelelően.



- Gerjesztésként a belső jelre (X(15:0)) adjunk x <= *X"1E2F"* hexadecimális értéket, míg dp = '1'.
- Implementálja a terveket FPGA-n, a 4 db 7-szegmenses kijelzőn, 125 MHz-es órajelen (**mclk**), és használja a legjobboldalibb nyomógombot (**btn**) a resethez, míg **dp**, **an**(3:0) (D1, … D4) és **a_to_g**(0:6) is kerüljön az *x7seg_top.xdc* fileba./
- Generálja le a bitstreamet, úgy, hogy a D értékét először állítsa 20 -> majd 18 –ra.
MIT TAPASZTAL?

Megoldás Feladat 7.)

(x7seg_top.vhd)

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

entity x7seg_top is
  port(
       mclk : in STD_LOGIC;
       btn : in STD_LOGIC;
       --x : in STD_LOGIC_VECTOR(15 downto 0);
       --a_to_g : out STD_LOGIC_VECTOR(0 to 6);
       A, B, C, D, E, F, G, dp : out STD_LOGIC;
       an : out STD_LOGIC_VECTOR(3 downto 0)
  );
end x7seg_top;
 T
architecture Behavioral of x7seg_top is

component clkdiv
  generic( D : natural:= 24 );
  port (
     clr : in STD_LOGIC;
     mclk : in STD_LOGIC;
     clk_1hz : out STD_LOGIC_VECTOR(D-1 downto 0));
end component;

component x7seg
  port (
    cclk : in  STD_LOGIC;
    clr : in  STD_LOGIC;
    x : in  STD_LOGIC_VECTOR (15 downto 0);
    --a_to_g : out  STD_LOGIC_VECTOR (0 to 6);
    A, B, C, D, E, F, G, dp : out STD_LOGIC;
    an : out  STD_LOGIC_VECTOR (3 downto 0 ) );
end component;
```

```
  signal x :  STD_LOGIC_VECTOR (15 downto 0) := X"1E2F";
  signal int_clk_hz : STD_LOGIC := '0';
  signal a_to_g_sig : STD_LOGIC_VECTOR(0 to 6);

begin

U1 : entity work.clkdiv(Behavioral)
generic (D : natural := 20);   --20 vs 24
port map(
       clr => btn,
       mclk => mclk,
       clk_1hz => int_clk_hz
  );

U2 : entity work.x7seg(struct)
  port map(
            cclk => int_clk_hz,
            clr => btn,
            x => x,
            A => A,
            B => B,
            C => C,
            D => D,
            E => E,
            F => F,
            G => G,
            an => an,
            dp => dp);
end Behavioral;
```

VHDL – Hierarchikus tervezés komplex áramkör esetén

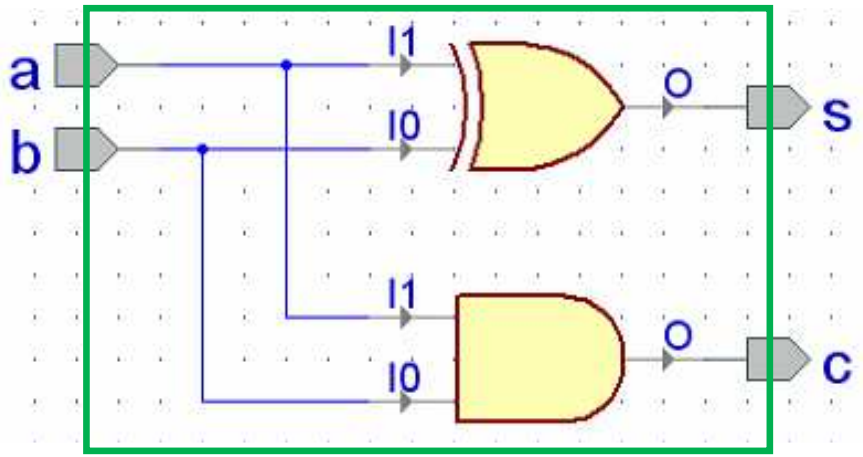
N-BITES ÖSSZEADÓ ÁRAMKÖR

Feladat 8.)

1-bites Half Adder

- „**Half**”, azaz **fél-összeadó**, mivel
  - nem kezeli a carry_in bemenetet , csak az ‘a’ és ‘b’ bemeneteket. Kimenetén ‘s’ (sum-összeg) és ‘cout’ (carry out) generál.
- Tervezzen egy 1-bites Half Adder („`half_adder.vhd`”), a mellékelt ábrának, vagy igazság táblázatnak megfelelően.
- Készítse el a testbench-ét Half Adder-nek („`half_adder_tb.vhd`”)
- Szimulálja le a viselkedését az összes lehetséges bemeneti kombinációra.

`half_adder.vhd`



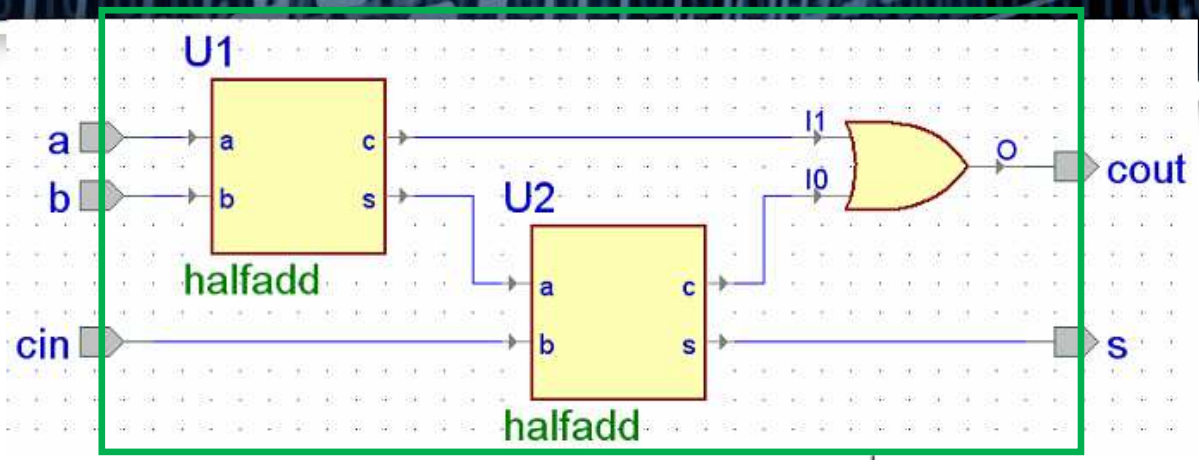
Igazság tábla:

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Feladat 9.)

1-bites Full Adder

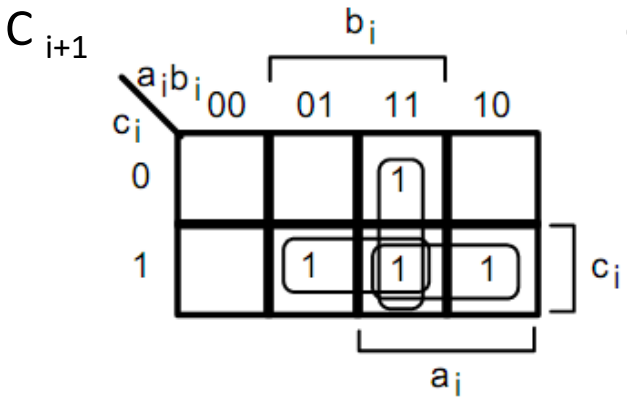
- „**Full**”, azaz **teljes összeadó**, mivel ez már képes kezelni a carry input bemeneteket is.
- Tervezzen egy 1-bites Full Adder („`full_adder.vhd`”), a mellékelt ábra alapján, és példányosítsa a megtervezett („`half_adder.vhd`”) áramköröket. Kösse össze őket megfelelően egy OR kapu segítségével.
- Készítsen egy testbenchet a Full Adder-hez („`full_adder_tb.vhd`”)
- Szimulálja a viselkedését az összes lehetséges bemeneti kombináció alapján.

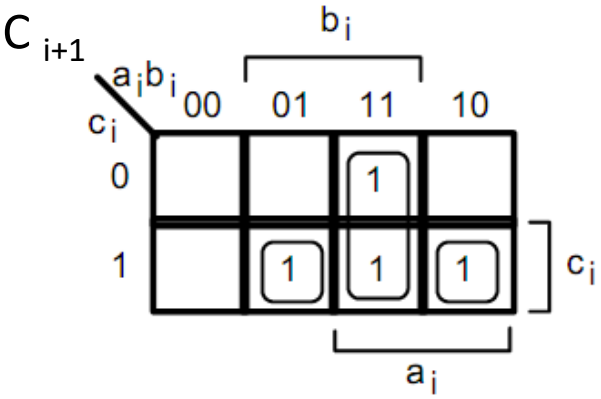


`full_adder.vhd`

Igazság tábla:

c_i	a_i	b_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





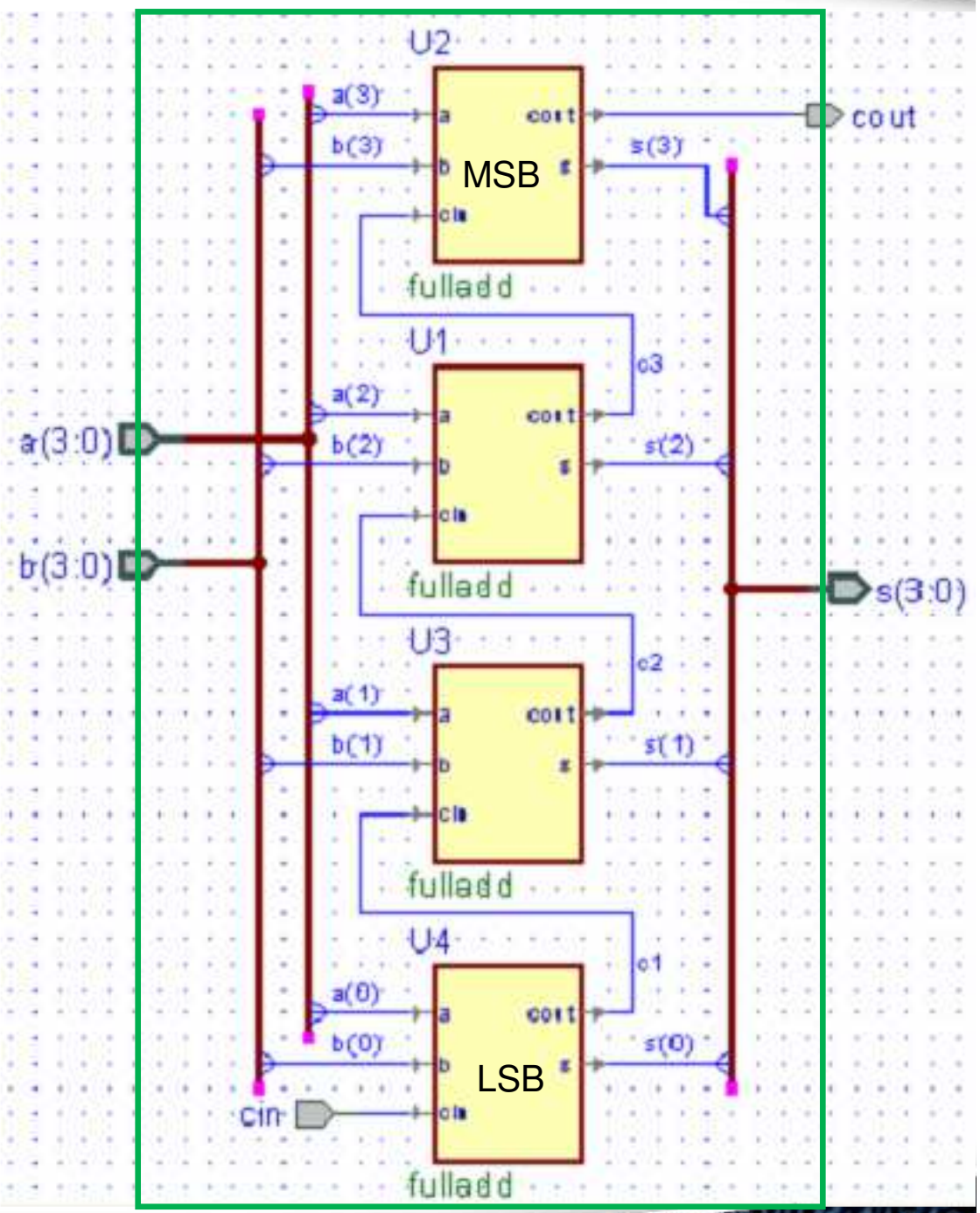
Feladat 10.)

RCA: N-bit Full Adder

(N:=4)

- Tervezzen egy N=4-bites Full Addert („full_adder_N.vhd”), a mellékelt ábrának megfelelően, és példányosítsa a korábbi 1-bites full addert („full_adder.vhd”) 4-szer. Kösse össze őket megfelelően.
- Készítsen egy testbench-et („full_adder_N_tb.vhd”)
- Szimulálja le a viselkedését XSim-ben.
- Implementálja FPGA-ra (.xdc).

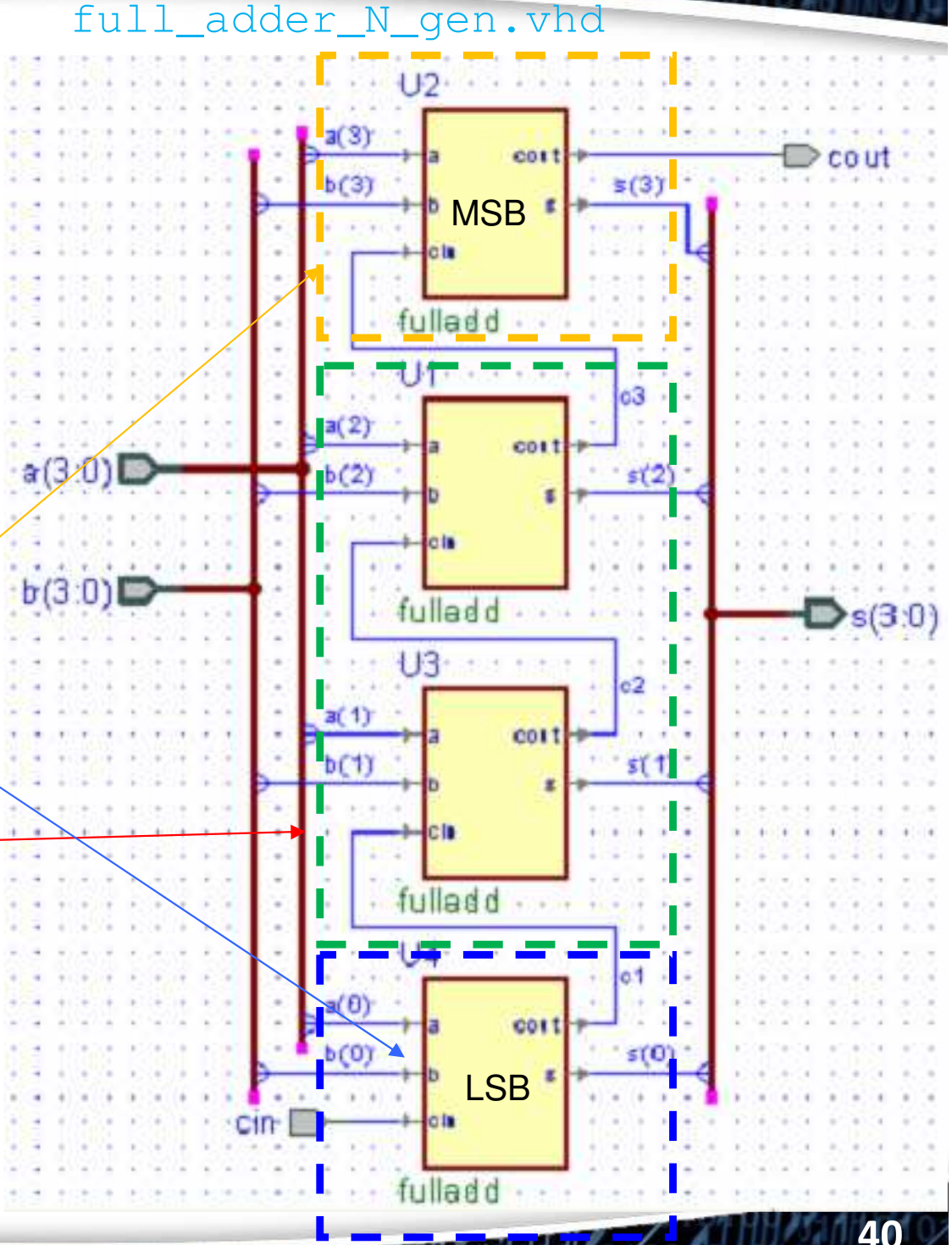
full_adder_N.vhd



Feladat 11.)

RCA: N-bit Full Adder (N:=4) + generate struktúra

- Tervezzen egy N=4-bites összeadót („full_adder_N_gen.vhd”), a mellékelt ábrának alapján, és példányosítsa a korábbi 1-bites full addert („full_adder.vhd”) 4-szer. Használjon generáló struktúrákat:
  - a.) **if generate-et** az LSB FA(0), illetve MSB FA(N-1) esetén, valamint
  - b.) **for generate-et** az FA(1)…FA(N-2) identikus blokkokra nézve.
- Készítsen egy testbench-et („full_adder_N_tb.vhd”)
- Szimulálja le a viselkedését XSim-ben. Implementálja FPGA-ra (.xdc).



Megoldás: Feladat 11.) generate

```
architecture Behavioral of full_adder_N_for is
    signal cout_sig : STD_LOGIC_VECTOR(N-1 downto 0);
begin
    RCA_N : for index in 0 to N-1 generate begin
        position_LSB: if index = 0 generate begin
            U0: entity work.full_adder(Behavioral) --LSB
            port map (a=>a(index), b=>b(index), cin=>cin, cout=>cout_sig(index),
            s=>s(index));
        end generate position_LSB;

        position_internal: if index >= 1 and index <= N-2 generate begin
            U_internal: entity work.full_adder(Behavioral)  --internal
             port map (a=>a(index), b=>b(index), cin=>cout_sig(index-1),
             cout=>cout_sig(index), s=>s(index));
        end generate position_internal;

        position_MSB: if index = N-1 generate begin
             U3: entity work.full_adder(Behavioral) --MSB
             port map (a=>a(index), b=>b(index), cin=>cout_sig(index-1),
             cout=>cout, s=>s(index));
        end generate position_MSB;

    end generate RCA_N; --end of for generate
end Behavioral;
```

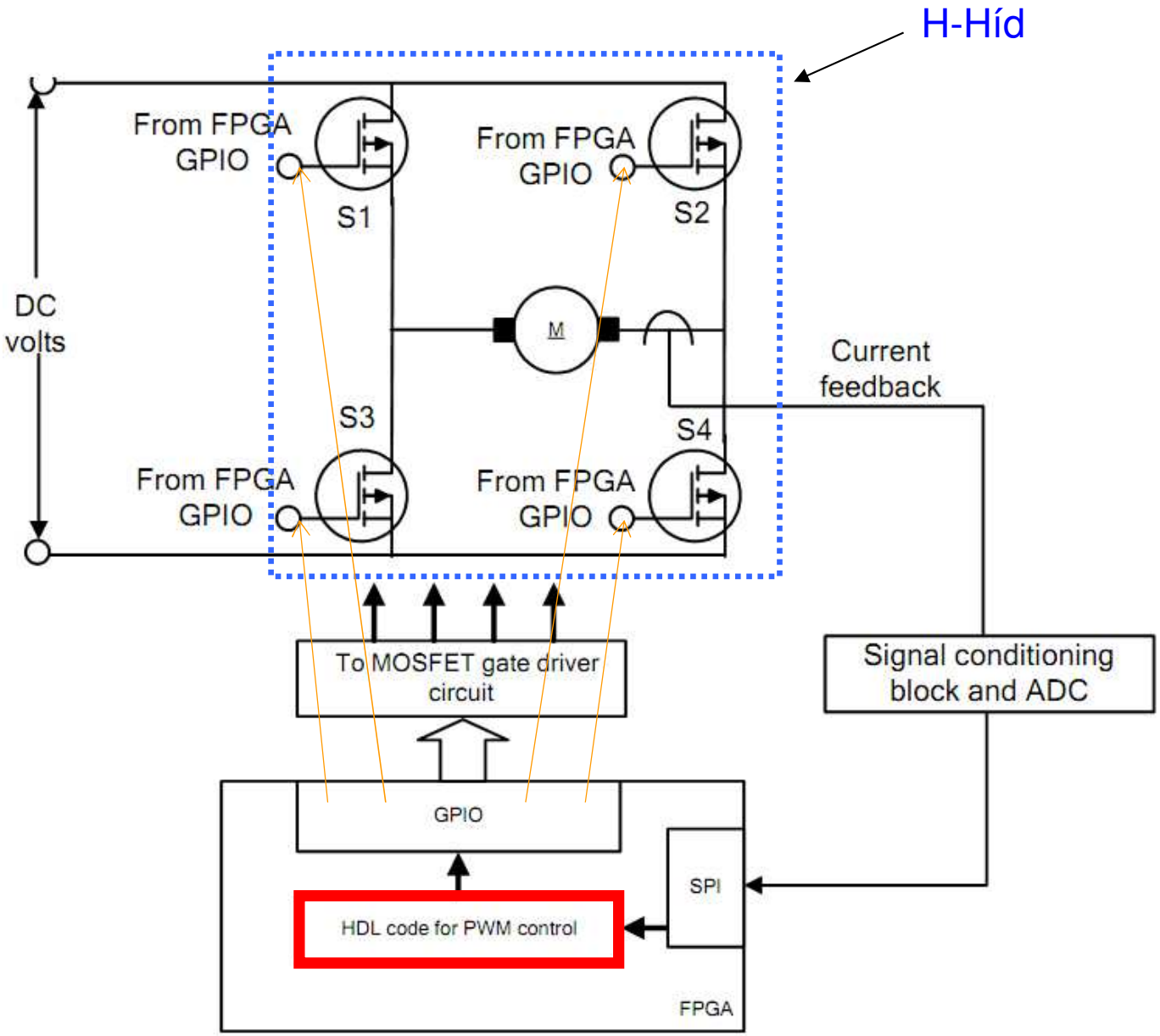
VHDL – gyakorlat

PWM SZABÁLYOZÓ

PWM: háttér

- **Pulse Width Modulation**: impulzus-szélesség moduláció
- Alkalmazási példa:
  - Az állandó mágneses egyenáramú motor (PMDC) az egyik leggyakrabban használt motor napjainkban.
    - A sebessége egyenesen arányos a rákapcsolt feszültséggel → egyszerű vezérelhetőség.
    - **„H-bridge**” 4 MOSFET-jének meghajtásával ¼ -es sebesség szabályozás biztosítható.
    - Ez a vezérlési séma egy szabadon futó körkörös **számláló** (counterN) megvalósításon alapul, amely **fűrészjelet** (**ramp**) generál. A fűrészjel arra használható, hogy a PWM kitöltési tényezőjét (**duty cycle**) változtatni lehessen.
    - *Módszer*: a counterN értékét a (Vc) szabályozó feszültséggel kell mindig összehasonlítani. Definíciótól függően, ha pl. a Vc értéke nagyobb, akkor a PWM feszültsége is nagyobb lesz. A motor kivezetésein mérhető feszültség a PWM kitöltési tényezőjének átlagos értéke.

Permanens mágneses DC motor vezérlése



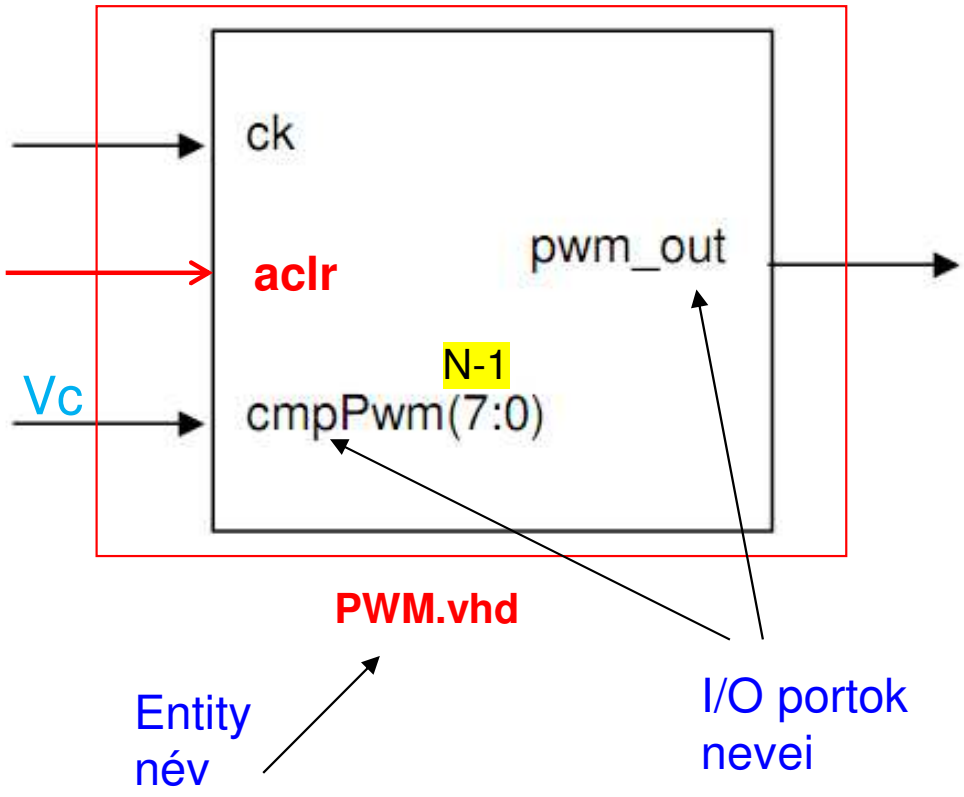
Feladat 12.)

Tervezzen egy N=4 bites PWM szabályozót az ábrának megfelelően (`PWM.vhd`)

Töltse le hozzá .zip csomagot *.

Megjegyzések:

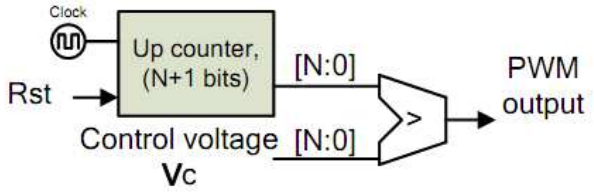
- Tanulmányozza a *PWMRefComp.pdf* dokumentációt.
- Használjon szekvenciális VHDL szerkezetet. Adjon egy aszinkron clear (*aclr*) jelet a reset-eléshez.
- Használjon generic konstanst N = 4;
- Készítse el a testbench-et („`pwm_tb.vhd`”) és szimulálja le a terveket XSim-ben.
- Módosítsa a pwmCmp = Vc értékét 0x0 – 0xF között és vizsgálja meg az eredményt. Futtassa le a szimulációt legalább 20 ms ideig!
- Implementálja a terveket FPGA-n
  - /használjon külső CLK 125MHz-et (‘ck’), legbaloldalibb nyomógombot a törléshez (‘aclr’ – btn(3)), dip kapcsolókat (N-1:0) Vc állításához, és LED(0) = pwm_out -nak!



Megj: használja fel a counterN.vhd forrását is!

*Referencia terv (Digilent): https://virt.uni-pannon.hu/index.php/hu/component/phocadownload/category/39-voroshazi-zsolt?download=340:tm-lab06-pwmrefcomp1

Megoldás 13.): PWM




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PWM is
    Generic (N : natural := 4);
    Port ( ck, aclr : in  STD_LOGIC;
           Vc :       in  STD_LOGIC_VECTOR(N-1 downto 0);      --switch bemenet(=Control Voltage Vc)
           pwm_out :  out  STD_LOGIC);                         --pwm_out kimenet ('1' or '0')
end PWM;
architecture Behavioral of PWM is
    constant ckPwmRange: integer:= 5; -- LSB in the cntPwm alias of cntDiv
    signal cntPwm: std_logic_vector(N+ckPwmRange downto 0):= (others => '0');
   -- inicializáció => aritmetikai művelet során az N=4 felső bitet használjuk PW Modulációhoz: cntPwm
    számol 125MHz/2^ckPwmRange
begin
    PwmCounter: process(ck, aclr) is
    begin
        if aclr = '1' then
                cntPwm <= (others => '0');      -- hozzáadni PWM.vhd
        elsif ck'event and ck='1' then
                cntPwm <= cntPwm + '1';
        end if;
    end process;
    PWM: process(cntPwm, Vc, aclr) is
    begin                   --felső N=4-bit PWM modulációra (9 downto 5)!!
        if aclr = '0' and cntPwm(ckPwmRange+N-1 downto ckPwmRange) <= Vc then
                -- számláló kisebb (egyenlő), mint a referencia feszültség (Vc)
                pwm_out <= '1';          -- kimenet HIGH
        else
                -- számláló nagyobb, mint a referencia feszültség (Vc)
                pwm_out <= '0';          -- Kimenet LOW
        end if;
    end process;
end Behavioral
```

PWM (N=4-bit) testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;  -- to_unsigned() cast
use IEEE.STD_LOGIC_unsigned.all;  -- inkrementálás + std_logic_vector típuson
ENTITY pwm_tb IS
GENERIC ( N: natural := 4);
END pwm_tb;

ARCHITECTURE behavior OF pwm_tb IS
   -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT PWM
  GENERIC ( N: natural := 4);
  PORT(  ck, aclr : IN  std_logic;
         Vc: IN  std_logic_vector(N-1 downto 0);
         pwm_out : OUT  std_logic );
   END COMPONENT;
    --Inputs
    signal ck : std_logic := '0';
    signal aclr : std_logic := '0';
    signal Vc : std_logic_vector(N-1 downto 0) := (others => '0');
    --Outputs
    signal pwm_out : std_logic;
    constant TbPeriod : time := 8 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';
BEGIN
 -- Instantiate the Unit Under Test (UUT)
  uut: PWM
      Generic map (N => N)
      PORT MAP (
         ck => ck,
         aclr => aclr,
         Vc => Vc,
         pwm_out => pwm_out
        );
--generate clock
TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else
'0';
   -- EDIT: Check that ck is really your main clock signal
   ck <= TbClock;
stimuli : process
   begin
      -- EDIT Adapt initialization as needed
      Vc <= (others => '0');
      -- Reset generation
      aclr <= '1';   wait for 100 ns;
      aclr <= '0';   wait for 100 ns;
--      for i in 0 to (2**N - 1) loop
--         Vc <= Vc + 1;
--         wait for 100_000 * TbPeriod;
--      end loop;

--       for i in 0 to N loop
--         Vc <= std_logic_vector ( to_unsigned(2**i-1,N) );
--          wait for 100_000 * TbPeriod;
--       end loop;

       for i in 0 to (2**(Vc'length) -1) loop
            Vc <= Vc + 1;
            wait for 100_000 * TbPeriod;
         end loop;
      wait;
   end process;
end pwm_tb;
```

Három lehetséges megoldás.

Szimulációs eredmény: PWM