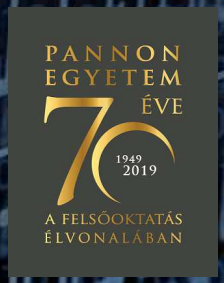
Pannon Egyetem, MIK-VIRT, Veszprém



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

Tervezési módszerek programozható logikai eszközökkel

6. VHDL: speciális nyelvi szerkezetek. Sorrendi hálózatok megvalósítása.

Frissítve: 2020. 11. 05.

Tárgyalt ismeretkörök
6. előadás

- VHDL **speciális** nyelvi szerkezetek:
  - A. generikus konstansok (**generics**),
  - B. generáló struktúrák (**generate**):
    - Ciklikus „`for … generate`” ,
    - Feltételes „`if … generate`” .
  - C. Függvények (**functions**),
  - D. Csomagok (**packages**),
- Sorrendi hálózatok megvalósítása
  - (Mealy, Moore)

Felhasznált irodalom:

- 🕮 Pong P. Chu - FPGA Prototyping by VHDL Examples: Xilinx Spartan-3
  - http://academic.csuohio.edu/chu_p/rtl/fpga_vhdl.html
- 🕮 Hosszú Gábor - Keresztes Péter: VHDL ALAPÚ RENDSZERTERVEZÉS (2012 © Szak kiadó)
  - http://www.szak.hu/konyvek_htm/vhdl.html
- 🕮 Horváth – Harangozó - VHDL VHSIC HARDWARE DESCRIPTION LANGUAGE - BME SEGÉDLET (2006)
  - http://www.fsz.bme.hu/~tom/vhdl/vhdl_s.pdf
- 🕮 Richard E. Haskell & Darrin M. Hanna - Introduction to Digital Design VHDL (Digilent Inc.)
  - https://reference.digilentinc.com/_media/textbooks:intro_digital_design-digilent-vhdl_online.pdf
- 🕮 *Real Digital - A hands-on approach to digital design* (Digilent Inc.)
  - https://learn.digilentinc.com/classroom/realdigital/

Általunk használt STD csomagok

Melyek szimulálhatók és szintetizálhatók is egyben:

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

IEEE 1076

VHDL Nyelvi szerkezetek

SPECIÁLIS NYELVI SZERKEZETEK

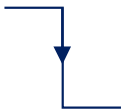
Megjegyzés

- Órajel vizsgálatának ekvivalens megadási módjai:

```
if clk'event and clk = '1' then
…
```

⇔

```
rising_edge(clk)
```



```
if clk'event and clk = '0' then
```

⇔

```
falling_edge(clk)
```

A. Generikus konstans (generic) deklarációja

```
entity identifier is
   [generic
  (generic_interface_list);]
   [port (port_interface_list);
end [entity] [identifier];
```

Példa: Generic konstans(Tpd)

```
entity myand2 is
  generic ( Tpd : time );
  port (
    a, b : in std_logic;
    y : out std_logic);
end entity myand2;

architecture simple of myand2 is
begin
      y <= a and b after Tpd;
end architecture simple;
…
```

generic alap érték: pl. fizikai idő típus adott.

Komponens példányosításakor adjuk meg: az alap generikus konstans érték **felülírható** egy megfelelő **generic map()** megadásával, és akár eltérhet az értéküktől is!!

```
gate1 : entity work.myand2(simple)
  generic map ( Tpd => 2 ns )
  port map ( a => sig1,  b => sig2,  y => sig_out );
gate2 : entity work.myand2(simple)
  generic map ( Tpd => 3 ns )
  port map ( a => a1,  b => b1,  y => sig1 );
```

N-bites Összeadó: generic (N) használatával (korábbi példa kiegészítése)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;  --összeadáshoz '+' ez a csomag kell

entity  gen_add_w_carry  is
   generic (N: integer  :=4);
   port ( a,  b: in std_logic_vector(N-1  downto  0);
          cout : out std_logic;
          sum : out std_logic_vector(N-1 downto  0)
end gen_add_w_carry ;

architecture  arch  of gen_add_w_carry is
   signal a_ext, b_ext, sum_ext : unsigned(N  downto  0) ; -- N+1 bit
begin
   a_ext <= unsigned('0'  &  a);
   b_ext  <= unsigned('0'  &  b);
   sum_ext <= a_ext  +  b_ext;
   sum <= std_logic_vector (sum_ext (N-1  downto  0))  ;
   cout  <=  sum_ext(N);       -- a sum MSB bitje lesz a carry kimenet
end  arch;
```

Példa: Generic konstans

```
entity D_FF is
generic ( Tpd_clk_q, Tsu_d_clk,
    Th_d_clk : time);
  port ( clk, d : in std_logic;
    q : out std_logic );
end entity D_FF;

architecture basic of D_FF is
begin
    q <= d after Tpd_clk_q when clk =
    '1' and clk'event;
check_setup : process is
begin
  wait until clk = '1';
  assert d'last_event >= Tsu_d_clk
    report "setup violation";
end process check_setup;
check_hold : process is
begin
  wait until clk'delayed(Th_d_clk) =
    '1';
  assert d'delayed'last_event >=
    Th_d_clk
    report "hold violation";
end process check_hold;
end architecture basic;
```

```
entity D_FF_testbench is
end entity D_FF_testbench ;

architecture test of D_FF_testbench is
  signal system_clock, request,
    request_pending : std_logic:= '0';
begin
request_flipflop : entity
    work.D_FF(basic)
  generic map ( Tpd_clk_q => 4 ns,
    Tsu_d_clk => 3 ns,
    Th_d_clk => 1 ns )
  port map ( clk => system_clock,
    d => request,
    q => request_pending);

clock_gen : system_clock <= '1' after
    10 ns,   '0' after 20 ns when
    system_clock = '0';

stimulus :  request <= '1' after 25 ns,
  '0' after  35 ns,
  '1' after  67 ns, '0' after  71 ns,
  '1' after 108 ns, '0' after 110.5 ns;
end architecture test;
```

Paraméterezhető struktúrák: generic használatával

```
entity reg is
  generic(width : positive);
  port(
  d:in std_logic_vector(0 to width-1);
  q:out std_logic_vector(0 to width-1);
  clk, reset : in std_logic);
end entity reg;

architecture behavioral of reg is
begin
  behavior : process (clk, reset) is
    constant ZERO : std_logic_vector(0
   to width-1) := (others => '0');
  begin
    if reset = '1' then
      q <= ZERO;
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process behavior;
end architecture behavioral;
```

```
subtype state_vector is
    std_logic_vector(1 to 5);
signal clk, reset : std_logic := '0';
signal word_in, word_out :
    std_logic_vector (0 to 31);
signal state_in, state_out :
    state_vector;
begin
word_reg : reg
  generic map ( width => 32 )
  port map (d => word_in,
    q => word_out,
    clk => clk,
    reset => reset);
state_reg : reg
  generic map ( width =>
   state_vector'length )
  port map (d => state_in,
    q => state_out,
    clk => clk,
    reset => reset );
```

B. Generáló struktúrák

For … generate struktúra

```
generate_label:
for identifier in discrete_range
  generate
[{block_declarative_item}
begin]
{concurrent_statement(s)}
end generate [generate_label]
```

For … generate struktúra

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
  generic ( width : positive );
  port ( clock : in std_logic;
    out_enable : in std_logic;
    data_in : in std_logic_vector(0 to
   width - 1);
    data_out : out std_logic_vector(0
   to width - 1) );
end entity reg;

architecture struct of reg is

component D_flipflop is
  port ( clk : in std_logic;
    d : in std_logic;
    q : out std_logic );
end component D_flipflop;
```

```
begin

cell_array : for bit_index in 0 to
   width - 1 generate
  signal data_unbuffered : std_logic;
  begin
    cell_storage : D_flipflop
      port map
        (clk => clock,
        d => data_in(bit_index),
       q => data_unbuffered);
   end generate cell_array;
end architecture struct;
```

```
entity D_flipflop is
  port ( d, clk : in std_logic; q : out
    std_logic );
end D_flipflop;
architecture basic of D_flipflop is
begin
  d_ff_behavior : process (clk, d) is
  begin
    if clk = '1' then
      q <= d after 2 ns;
    end if;
end process d_ff_behavior ;
end architecture basic;
```

Példa: For … generate struktúra

```
architecture behavioral of graphics_engine is
  type point is array (1 to 3) of real;
  type transformation_matrix is array (1 to 3, 1 to 3) of real;
  signal p, transformed_p : point;
  signal a : transformation_matrix;
  signal clock : std_logic;
  -- . . .
begin
  transform_stage : for i in 1 to 3 generate
  begin
    cross_product_transform : process is
      variable result1, result2, result3 : real := 0.0;
    begin
      wait until clock = '1';
      transformed_p(i) <= result3;
      result3 := result2;
      result2 := result1;
      result1 := a(i,1) * p(1) + a(i,2) * p(2) + a(i,3) * p(3);
    end process cross_product_transform;
  end generate transform_stage;
  -- . . .
```

Pipe-line késleltetés (3 órajelig)

For… Generate + VHDL-2008

redukciós operátorok „eq4bit_top” (korábbi pld)

```
library ieee;
use ieee.std_logic_1164.all;
entity eq4bit_top is
generic (N : integer := 4);
port ( a, b: in std_logic_vector (N-1 downto 0);
        aeqb : out std_logic);
end eq4bit_top;

architecture structural of eq4bit_top is
   component eq1
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           eq : out STD_LOGIC);
   end component;
   signal e: std_logic_vector(N-1 downto 0):=(others => '0');
begin
   eqS : for i in 0 to N-1 generate
        eq1_unit : entity work.eq1(struct)
        port map (i0=>a(i), i1=>b(i), eq=>e(i));
        aeqb <= and e;  --unary redukciós operátor csak VHDL-2008-ban!
end structural;
```

Sources ablak → vhd forrás → jobb gomb → listából Set File Type… → VHDL-2008 kiválasztása. OK.

Feltételes If … generate struktúra

```
generate_label:
if boolean_expression generate
[{block_declarative_item}
begin]
{concurrent_statement}
end generate [generate_label]
```

Példa: feltételes If…generate

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg is
  port ( phi1, phi2 : in
   std_logic;
         serial_data_in : in
   std_logic;
         parallel_data : inout
   std_logic_vector );
end entity shift_reg;

architecture cell_level of
   shift_reg is
  alias normalized_parallel_data :
          std_logic_vector(0 to
   parallel_data'length - 1)
        is parallel_data;
  component master_slave_flipflop
   is
    port ( phi1, phi2 : in
   std_logic;
           d : in std_logic;
           q : out std_logic );
  end component
   master_slave_flipflop;
```

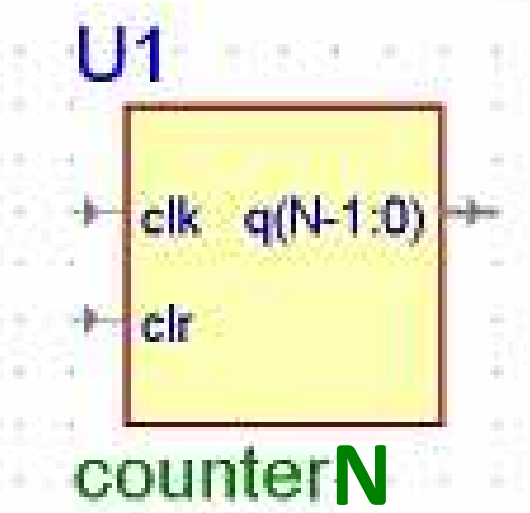
```
begin
  reg_array : for index in
normalized_parallel_data'range generate
  begin

    first_cell : if index = 0 generate
    begin
      cell : master_slave_flipflop
        port map ( phi1, phi2,
                   d => serial_data_in,
                   q =>
normalized_parallel_data(index) );
    end generate first_cell;

    other_cell : if index /= 0 generate
    begin
      cell : master_slave_flipflop
        port map ( phi1, phi2,
                   d =>
normalized_parallel_data(index - 1),
                   q =>
normalized_parallel_data(index) );
    end generate other_cell;
  end generate reg_array;
end architecture cell_level;
```

Példa: N:=4-bites RCA áramkör beágyazott „for…generate / if … generate” struktúrákkal

```
architecture GEN of RCA_4bit_gen is

  component FULLADD
    port (A,B,CIN : in std_logic;
          SUM, CARRY : out std_logic);
  end component;

  component HALFADD
  port(A,B : in std_logic;
       SUM, CARRY : out std_logic);
  end component;

  signal C : std_logic_vector(0 to N-
1);
```

```
begin
  GEN_ADD: for I in 0 to N-1 generate

    LOWER_BIT: if I=0 generate
      U0: HALFADD port map
          (A(I),B(I),S(I),C(I));
    end generate LOWER_BIT;

    UPPER_BITS: if I>0 generate
      UX: FULLADD port map
          (A(I),B(I),C(I-1),S(I),C(I));
    end generate UPPER_BITS;

  end generate GEN_ADD;
  COUT <= C(N-1);
end GEN;
```

Feladat 1.)



Tervezzen egy **N:=4-bites bináris számlálót (counterN)** a mellékelt ábra alapján (`counterN.vhd`).
- Megj: használjon *generikus konstanst* **N:=4**, és *szekvenciális* hozzárendelő utasításokat. Dolgozzon új projektben!
- Használjon aszinkron *clr-t* a számláló modul *reset*-elésére (reset to '0').
- Ha clr = '0', akkor inkrementálja 1-el a számláló *q* értékét a *clk* órajel minden egyes felfutó élére. Ehhez használja a **IEEE.STD_LOGIC_unsigned.all**; csomagot (inkrementálás *std_logic_vector* típusokon)!
- Készítsen egy testbench-et („`counterN_tb.vhd`").
- Szimulálja le a viselkedését Vivado Simulator segítségével
- Implementálja a terveket FPGA-n. Használja a CLK = 125 MHz bemeneti órajelet a *clk* jelhez, a legjobboldalibb nyomógombot az aszinkron *clr* (reset-hez), illetve az összes LED-et a számláló *q* aktuális értékének megjelenítéséhez!

Mit tapasztalunk ?!

Megoldás Feladat 1): N-bites számláló generic használatával (counterN.vhd)

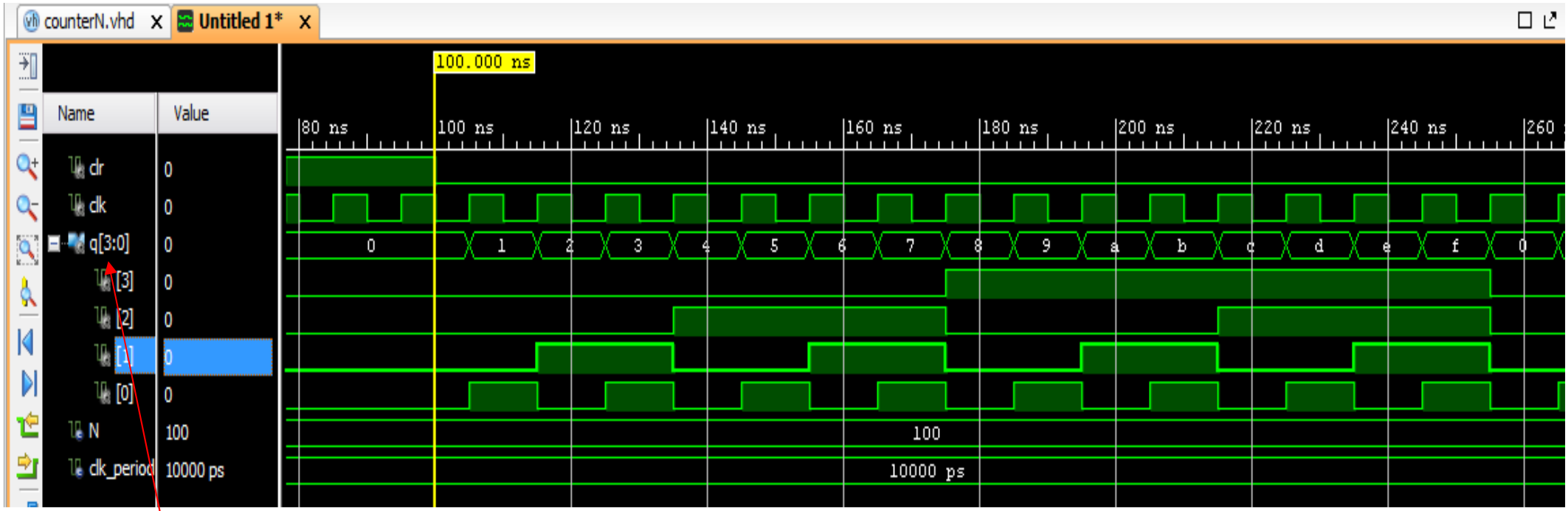
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_unsigned.all;  -- IEEE package increment (+) counter
entity counterN is
  generic(N : natural := 4);
  port(  clr : in STD_LOGIC;
         clk : in STD_LOGIC;
         q : out STD_LOGIC_VECTOR(N-1 downto 0) );
end counterN;

architecture Behavioral of counterN is
   signal count: STD_LOGIC_VECTOR(N-1 downto 0);
begin
      process (clk, clr)
       begin
            if clr = '1' then
                  count <= (others => '0');
             elsif clk'event and clk = '1' then  -- hasonlóan rising_edge(clk)
                  count <= count + 1;
             end if;
       end process;
      q <= count;
end Behavioral;
```

Megoldás Feladat 1): Szimuláció

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY counterN_tb IS
    generic (N: natural := 4);
END counterN_tb;

ARCHITECTURE behavior OF counterN_tb IS

    -- Component Declaration for (UUT)

    component counterN is
    generic (N: natural := 4);
    port(
         clr : IN  std_logic;
         clk : IN  std_logic;
         q   : OUT  std_logic_vector(N-1 downto 0)
        );
    end component;

   --Inputs
   signal clr : std_logic := '0';
   signal clk : std_logic := '0';

    --Outputs
   signal q : std_logic_vector(N-1 downto 0);

   -- Clock period definitions
   constant clk_period : time := 8 ns;
```

```
begin

   uut: entity work.counterN(Behavioral)
   generic map(N => 4)
   port map(
          clr => clr,
          clk => clk,
          q => q
        );

   -- Clock process definitions
   clk_process :process
   begin
          clk <= '0';
          wait for clk_period/2;
          clk <= '1';
          wait for clk_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      clr <= '1';
      wait for 100 ns;
      clr <= '0';
      wait for clk_period*10000;
      clr <= '1';
      wait for 1000 ns;
      clr <= '0';
      wait;
   end process;
END;
```

Megoldás Feladat 1): N-bites számláló generic használatával (counterN_tb.vhd)



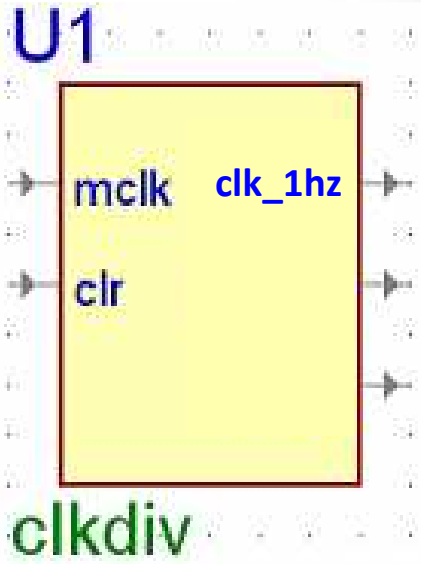
A számláló q értékének kiíratásánál érdemes hexadecimális alapot választani. q(i) kijelölése, majd Radix kiválasztása. Végül [+] busz jelek listázása.

Órajel frekvencia leosztása

- f: bemeneti frekvencia (ZyBo @ **125** MHz – „**L6” FPGA** láb)
- q(i): kimeneti számított frekvencia képlet alapján: $f_i = \frac{f}{2^{i+1}}$

	q(i)	Frekvencia (Hz)	Periódus (ms)
	i	125000000.00	0.000008
D=1	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
	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
D=27	**26**	0.93	1073.741824

Feladat 2.)



Tervezzen egy **Clock Divider, azaz órajel osztó** áramkört a mellékelt ábra alapján („`clkdiv.vhd`”).
Dolgozzon új projektben!
- Megj: használjon *generic* utasítást **D:=27**, és *szekvenciális* hozzárendelő utasításokat(s)
- Használjon aszinkron clr-t a *clkdiv* modul Reset-elésére (reset to ‘0’).

Készítsen egy testbench áramkört („`clkdiv_tb.vhd`”)
- Szimulálja le a viselkedését (Vivado Simulator)
- Implementálja a terveket FPGA-n
- Használja a CLK = 125 MHz bemeneti órajelet a *mclk* jelhez, a legjobboldalibb nyomógombot az aszinkron *clr* (reset-hez) és egy LED-et, amely *clk_1hz-enként* (~1 Hz) világít.
- Mi a korrekt értéke a kimenetnek? Az 1 Hz-es órajel számított értéke: $\longrightarrow$ $f_{i=26} = \frac{f}{2^{27}} = 0.93Hz = \sim 1Hz$

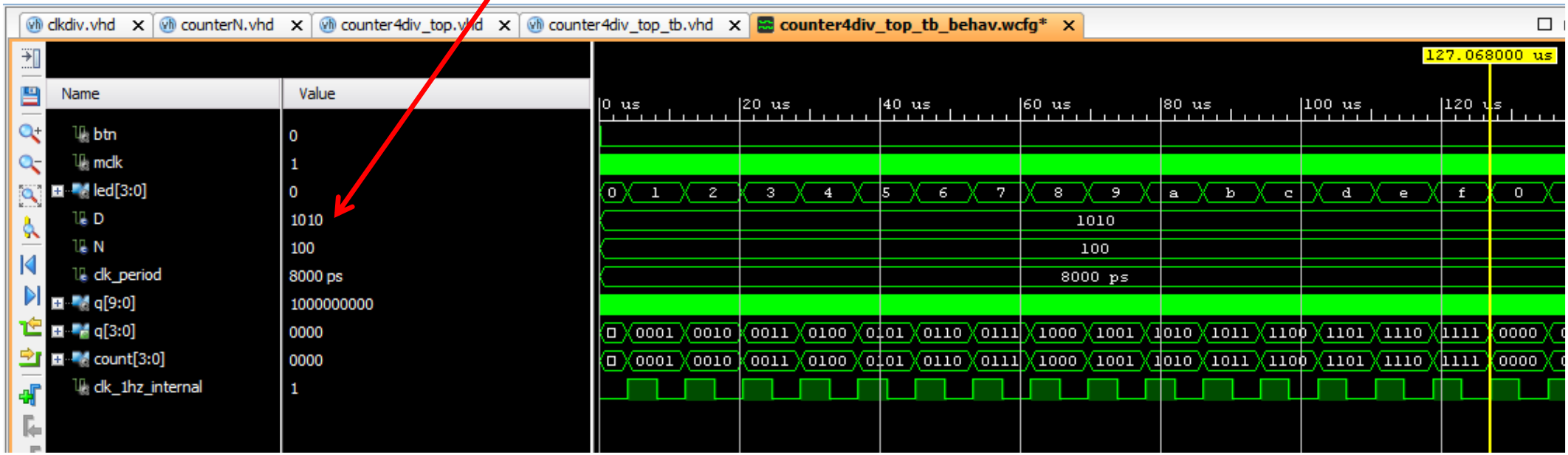
■ **Javaslat**: Szimuláció során érdemes az D paramétert kisebbre állítani (pl. D:= ~10) , hogy a szimulációs idő rövidebb legyen!

Megoldás Feladat 2.): órajel osztó generic használatával (clkdiv.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;  -- inkrementálást +1 -hez

entity clkdiv is
    generic (D : natural := 27);
    port( mclk : in STD_LOGIC;
          clr  : in STD_LOGIC;
          clk_1hz : out STD_LOGIC   );
end clkdiv;

architecture Behavioral of clkdiv is
signal q: STD_LOGIC_VECTOR(D-1 downto 0);
begin
  process (mclk, clr) is
  begin
    if clr = '1' then          -- aszinkron reset
        q <= (others => '0');
    elsif rising_edge(mclk) then  -- hasonlóan lehetne (mclk'event and mclk = '1')
        q <= q + 1;
    end if;
  end process;
  clk_1hz <= q(D-1);  -- órajel leosztása - clk ~ 1 Hz (0.93 Hz)
end Behavioral;
```

Megoldás 2.): Órajel osztó generic használatával (clkdiv.vhd) – szimulációs eredmény

- Javaslat: Szimulációs idő csökkentése végett érdemes a D értékét **példányosításkor** kisebbre választva futtatni a tesztet (pl. D:=10). Ekkor csak kb. 130 us-ig kell futtatni!



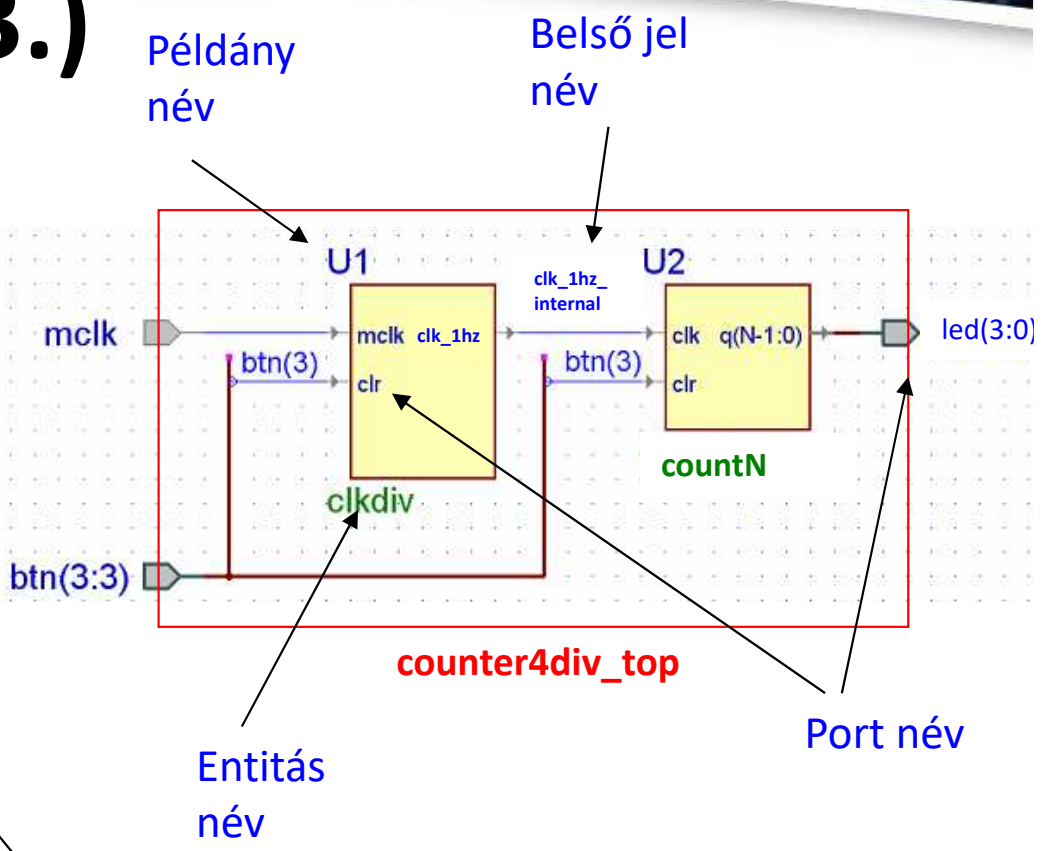
- f: input frekvencia (Digilent ZyBo @125 MHz)
- q(i)= ? Ha i := 26 (akkor D:= 27 bit)

$$f_{i=26} = \frac{f}{2^{i+1}} = \frac{125MHz}{2^{27}} = \frac{125000000Hz}{134217728} = 0.93Hz \sim (1Hz)$$

Feladat 3.)

Tervezzen egy **N:=4-bites számlálót órajel osztó áramkörrel kiegészítve** a mellékelt ábra alapján („counter4div_top.vhd”).
-Megj: használjon *generic* konstans órajel leosztásához **D:=27.** Dolgozzon új projektben!
-Használjon *szekvenciális* hozzárendelő utasításokat
- Használjon aszinkron reset-et (clr) az órajel osztó reset-eléséhez (clkdiv to ‘0’).

Készítsen egy testbench-et és („counter4div_top_tb”) szimulálja a viselkedést XSim-ben.
- Implementálja a terveket FPGA-n. Használja a CLK = 125MHz bemeneti órajelet a *mclk* jelhez, a legjobboldalibb nyomógombot az aszinkron *clr* (‘clr’ – btn), és LEDs(3:0)-t amely a számláló aktív értékét mutatja, amely ~1Hz-enként változik.



Megj.: Példányosísa a ‘countN’ és ‘clkdiv’ komponenseket, melyeket a korábbi Feladat 1.)-2.)-ben készítettünk el. Kösse össze őket a megfelelő belső jelekkel!

Megoldás 3.) N:=4-bites számláló órajel osztó áramkörrel kiegészítve (counter4div_top.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity counter4div_top is
    generic (D : natural := 10; N : natural := 4 );
    port ( mclk: in std_logic;
       btn : in std_logic;
       led : out std_logic_vector(N-1 downto 0));
end counter4div_top;

architecture Behavioral of counter4div_top is
    signal clk_1hz_internal : std_logic := '0';
    component clkdiv is
     port( mclk    : in STD_LOGIC;
           clr     : in STD_LOGIC;
           clk_1hz : out STD_LOGIC);
    end component clkdiv;
    component counterN is
    Port (  clk : in  STD_LOGIC;
            clr : in  STD_LOGIC;
            q   : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component counterN;
begin
    U1 : entity work.clkdiv(Behavioral)
    generic map (D => 27) --állítsa a szimulációhoz D => 10
    port map(mclk => mclk, clr => btn, clk_1hz => clk_1hz_internal);
    U2 : entity work.counterN(Behavioral)
    generic map (N => 4)
    port map(clk => clk_1hz_internal, clr => btn, q => led);
end Behavioral;
```

példányosítások

C.) Függvény (function)

Az alprogramok családjába tartozik. Mindig közvetlenül egyetlen értékkel tér vissza (**return**). Csak bemeneti (**in**) módú paramétereket fogad el a paraméter listájában. Egy függvényt VHDL csomagban (package), felépítményben (architecture), vagy folyamatban (process) lehet megadni. [KERESZTES_HOSSZÚ]

```
function designator [ ( formal_parameter_list ) ]
return type_mark
```

Példa: MaxVal Függvény

```
function MaxVal (a: in integer; b: in integer) return
integer is
begin
      if (a >= b) then
            return a;
      else
            return b;
      end if;
end function MaxVal;
```

Példa: log2() függvény ver.1

```
library IEEE;
use IEEE.math_real.all;
vagy
use IEEE.math_real."ceil";
use IEEE.math_real."log2";

bit_width:= integer(ceil(log2(real(a))));
```

‚a' értékétől függően számítjuk ki a bitszélességet

Példa: log2() függvény ver.2

```
function fct_log2_v2(n : natural) return integer is
        variable i : integer := 0;
        variable value: integer := 1;
begin
        if n = 0 then return 0;
        else
                for j in 0 to 29 loop
                        if value >= n then null;
                        else
                                i := i+1;
                                value := value*2;
                        end if;
                end loop;
                return i;
        end if;
end function fct_log2_v2;
```

Példa: log2() függvény ver.3

```
function fct_log2_v3(x : positive) return natural is
    variable i : natural;
begin
        i := 0;
        while (2**i < x) and i < 31 loop
            i := i + 1;
        end loop;
        return i;
end function fct_log2_v3;
```

Ekkor használható a következő a bitszélesség megadáshoz:

```
cnt : in std_logic_vector(fct_log2_v3(NUMBITS)-1 downto
0);
```

D.) Csomag (Package)

Csomag megosztott, közös építőelemek, VHDL objektumok tárolására szolgál, melyek újrafelhasználása különböző VHDL forrásokban újra megtörténhet. [*KERESZTES_HOSSZÚ*]

A csomag két részből áll:

- Deklarációs rész (globális nyilvános információk)
- Törzs (lokális információk)

Csomag deklarációja (Package)

```
package_declaration ::=
   package identifier is
      package_declarative_part
   end [package] [identifier];

package_declarative_item ::=
     subprogram_declaration
   | type_declaration
   | subtype_declaration
   | constant_declaration
   | alias_declaration
   | use_clause
```

Use Clauses (Package használata)

```
use_clause <= use selected_name
  { , selected_name } ;
selected_name <= name .
  (identifier | character_literal
  | operator_symbol | all)
```

Példa: package használata

```
use work.data_types_pck.address;
use work.data_types_pck.data;

variable data_word : data;
variable next_address : address;

Pl:
use ieee.std_logic_1164.all;
```

Példa: csomagok

```
package data_types_pck is          --deklaratív részek
  subtype address is std_logic_vector(24 downto 0);
  subtype data is std_logic_vector(15 downto 0);
  constant vector_table_loc : address;
  function data_to_int(value : data) return integer;
  function int_to_data(value : integer) return data;
end data_types_pck;

variable var : work.data_types_pck.address;
```

Hivatkozás, használat:

```
library work;
use work.data_types_pck.all;
```

Csomag törzs (body)

```
package_body ::=
package body package_simple_name is
  package_body_declarative_part
end [ package_simple_name ];

package_body_declarative_item ::=
   subprogram_declaration
 | subprogram_body
 | type_declaration
 | subtype_declaration
 | constant_declaration
 | alias_declaration
 | use_clause
```

Példa: csomag törzse

```
package body data_types_pck is
   constant vector_table_loc : address :=
   X"FFFF00";

  function data_to_int(value : data) return
   integer is
      …
  end data_to_int;

  function int_to_data(value : integer) return
   data is
      …
  end int_to_data;

end data_types_pck;
```

Csomagban függvény: fct_log2()

```
package sajat_pck is
        function fct_log2(n : natural) return integer;
end sajat_pck;
-- csomag törzse, pl. függvények megvalósítása
package body sajat_pck is
    function fct_log2(n : natural) return integer is
        variable i  : integer := 0;
        variable value: integer := 1;
    begin
        if n = 0 then return 0;
        else
                for j in 0 to 29 loop
                        if value >= n then null;
                        else
                                i    := i+1;
                                value := value*2;
                        end if;
                end loop;
                return i;
        end if;
    end function fct_log2;
end sajat_pck;
```

Csomagban függvény:

```
package sajat_pck2 is
        function fct_GT (op1, op2: in std_logic_vector) return std_logic;
        function fct_LT (op1, op2: in std_logic_vector) return std_logic;
        function fct_EQ (op1, op2: in std_logic_vector) return std_logic;
end sajat_pck2;
-- csomag törzse, pl. függvények megvalósítása
package body sajat_pck2 is
    function fct_GT (op1, op2: in std_logic_vector) return std_logic is
    begin
        if (op1 > op2) then return '1';
        else return '0';
        end if;
    end function fct_GT ;
    function fct_LT (op1, op2: in std_logic_vector) return std_logic is
    begin
        if (op1 < op2) then return '1';
        else return '0';
        end if;
    end function fct_LT ;

    function fct_EQ (op1, op2: in std_logic_vector) return std_logic is
    begin
        if (op1 = op2) then return '1';
        else return '0';
        end if;
    end function fct_EQ ;
end sajat_pck2;
```

Bitszélességet nem, de a típust meg kell adni!

Használat: függvény , csomag

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.sajat_pck2.all;

entity comp_funct is
         generic (N: positive := 4);
    port ( op_a : in  STD_LOGIC_VECTOR(N-1 downto 0);
           op_b : in  STD_LOGIC_VECTOR(N-1 downto 0);
           res_gt : out STD_LOGIC;
            res_lt : out STD_LOGIC;
            res_eq : out STD_LOGIC
          );
end comp_funct;

architecture Behavioral of comp_funct is
begin
        res_eq <= fct_EQ (op_a, op_b);
        res_gt <= fct_GT (op_a, op_b);
        res_lt <= fct_LT (op_a, op_b);
end Behavioral;
```

Csomag használata.

Függvény hívások (csomagból!)

IEEE 1076

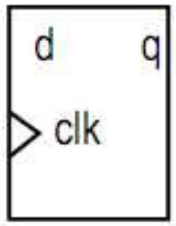
VHDL Nyelvi szerkezetek

SORRENDI HÁLÓZATOK MEGVALÓSÍTÁSA

Sorrendi hálózatok építőelemei
(D-tároló működési módjai)

- **Egyszerű D-FF / FlipFlop (enable és reset nélkül)**

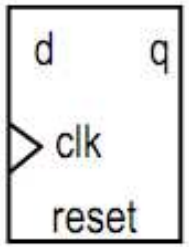
```
architecture  arch  of  d_ff  is
begin
   process (clk)
          begin
          if  (clk'event  and  clk = '1')  then
                    q  <=  d;
          end  if ;
   end  process;
end  arch;
```



clk	q*
0	q
1	q
↓	d

- **D-FF, aszinkron reset-tel (enable nélkül)**

```
architecture  arch  of  d_ff_reset  is
begin
   process (clk,  reset)  is
   begin
          if  (reset = '1')  then
                    q  <= '0';
          elsif  (clk'event and  clk = '1') then
                    q  <=  d;
          end  if ;
   end  process;
end  arch;
```

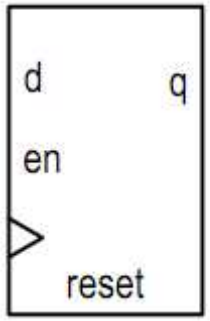


reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

Sorrendi hálózatok (tárolók)

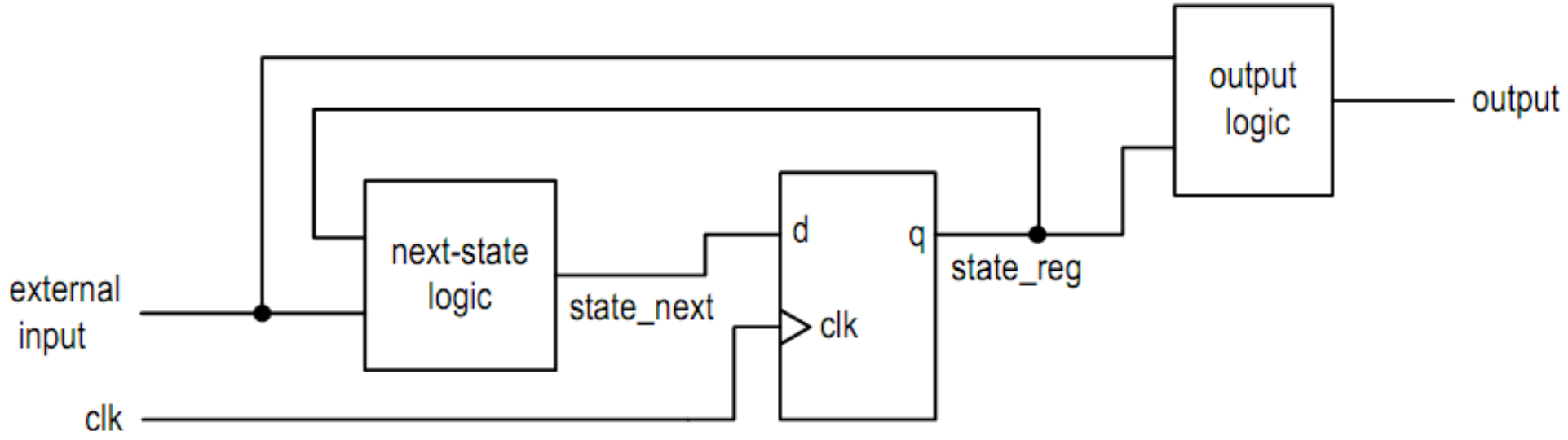
- **D-FF aszinkron reset-tel, és szinkon enable-vel**

```
architecture  arch  of  d_ff_en  is
begin
    process (clk,  reset, en) is
        begin
        if  (reset = '1') then
                q  <='0';
        elsif  (clk'event and clk = '1')
        then
                if  (en = '1') then
                        q  <= d;
                end  if ;

        end  if ;
    end  process;
end  arch;
```



reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	⌐↑	0	q
0	⌐↑	1	d

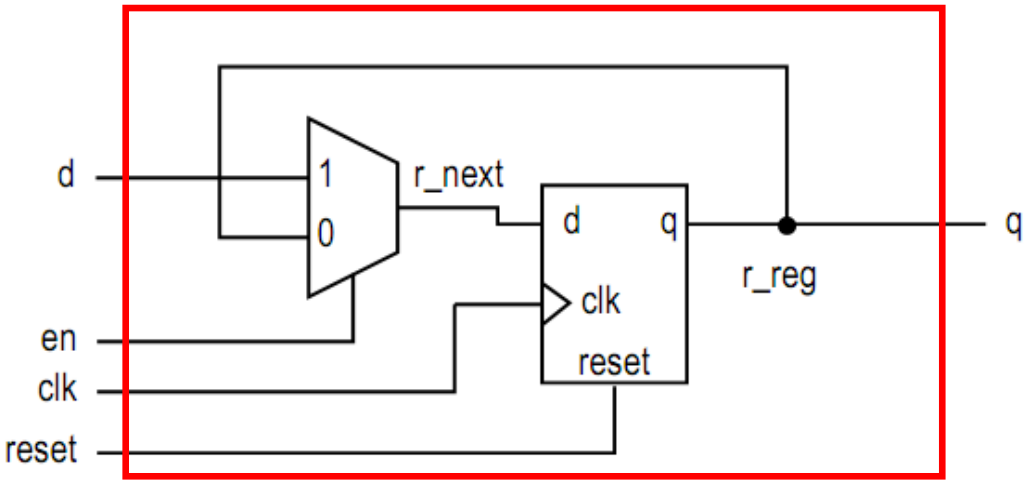
„Általános” sorrendi hálózat (S.H.) felépítése



Példa: D-FF szinkron engedélyező jellel

Tervezzen egy **1-bit D-FF a mellékelt ábrának megfelelően („d_ff_en.vhd”).**

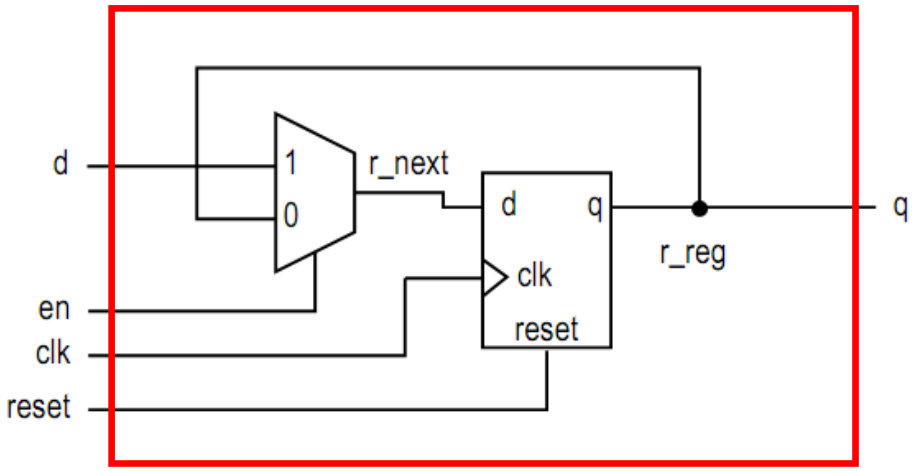
- Konkurens és szekvenciális VHDL utasítások is használhatóak
- Legyen *szinkron enable* (en) jele, amely az órajel felfutó élekor engedélyezi annak működését.
- Legyen *aszinkron reset* jele, mely reset-eli a D-FF tartalmát
- Ha(reset = ‘0’), és engedélyező (en = ‘1’) tárolja a d bemenet értékét a d_ff_en az órajel felfutó élére, mialatt a kimeneti érték kiolvasható.
- Készítsen testbench-et („d_ff_en_tb.vhd”)
- Szimulálja a viselkedését (Vivado Sim)



d_ff_en.vhd

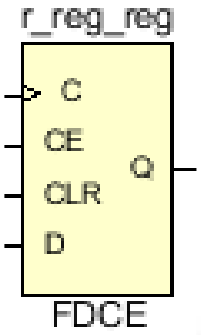
Megoldás: D-FF

```
architecture behav of d_ff_en is
    signal r_reg, r_next: std_logic;
begin
    -- D FF
    process(clk, reset)
    begin
        if (reset='1') then
            r_reg <='0';
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic
    r_next <= d when en ='1' else r_reg;
    -- output logic
    q <= r_reg;
end behav;
```



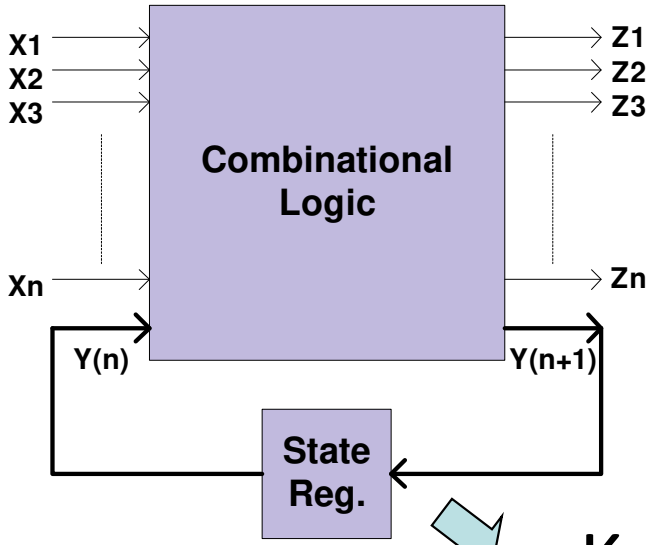
Milyen S.H. modell?

Resource	Utilization	Available
FF	1	35200
IO	5	100

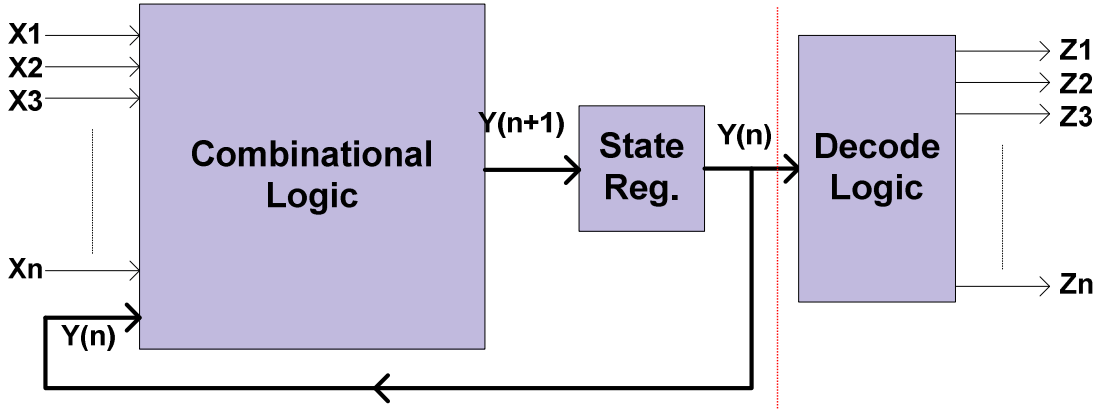


Véges állapotú automata (FSM)

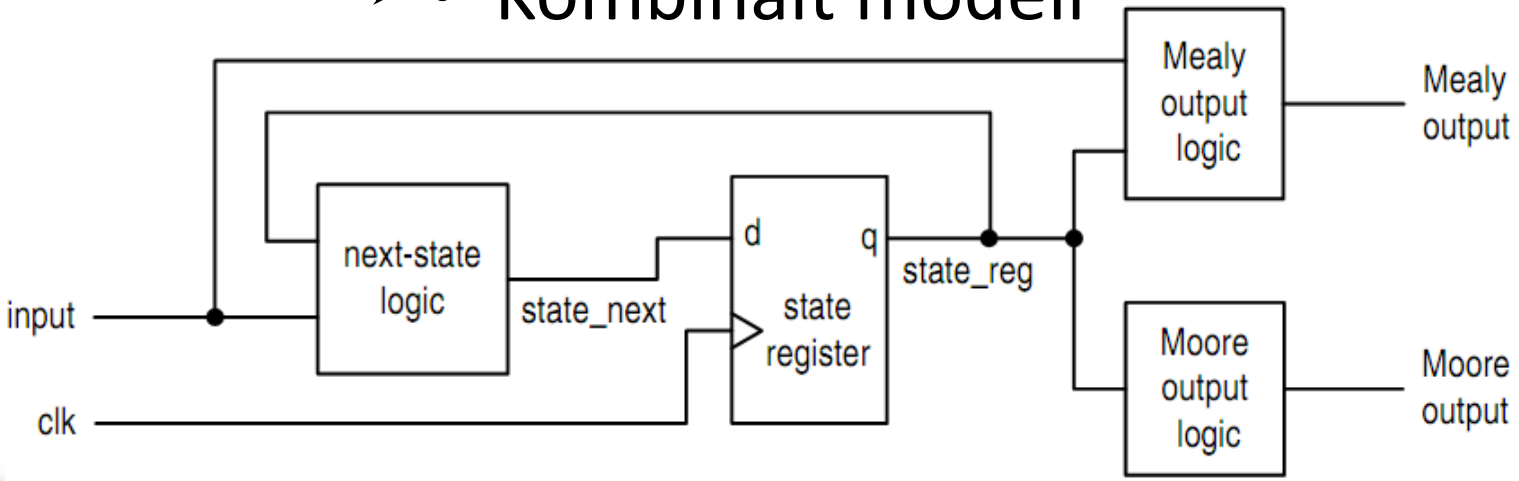
- Mealy State-Machine



- Moore State-Machine



- Kombinált modell



Alapvető állapotkódolási módszerek

- „One-hot” (1-es súlyú)
  - Mindenegyes állapotban 1-es bináris súly, de eltérő **bit-pozíción**

```
S1 <= ”0001”
S2 <= ”0010”
S3 <= ”0100”
S4 <= ”1000”
```

  - Szélesebb állapot regiszter kell
  - Könnyebb dekódolni.
- Bináris kódolás
  - Mindenegyes állapothoz egyedi állapot **érték tartozik**

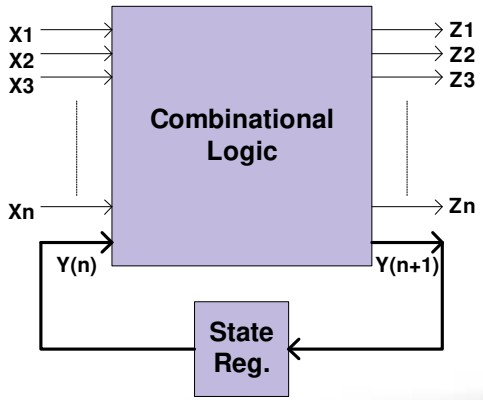
```
S1 <= ”00”
S2 <= ”01”
S3 <= ”10”
S4 <= ”11”
```

  - Kisebb méretű állapot regiszter kell
  - Nehezebb dekódolni.

a.) Mealy-féle állapotgép

```
type state_type is (st1_<name_state>,
    st2_<name_state>, ...);

signal state, next_state : state_type;

signal <output>_i : std_logic;

SYNC_PROC: process (<clock>)
   begin
      if (<clock>'event and <clock> =
    '1') then
         if (<reset> = '1') then
            state <= st1_<name_state>;
            <output> <= '0';
         else
            state <= next_state;
            <output> <= <output>_i;
         end if;
      end if;
   end process;

OUTPUT_DECODE: process (state, <input1>,
    <input2>, ...)
   begin
      if (state = st3_<name> and <input1>
    = '1') then
         <output>_i <= '1';
      else
         <output>_i <= '0';
      end if;
   end process;
```

```
NEXT_STATE_DECODE: process (state,
    <input1>, <input2>, ...)
   begin
      next_state <= state;
      case (state) is
         when st1_<name> =>
            if <input_1> = '1' then
               next_state <= st2_<name>;
            end if;
         when st2_<name> =>
            if <input_2> = '1' then
               next_state <= st3_<name>;
            end if;
         when st3_<name> =>
            next_state <= st1_<name>;
         when others =>
            next_state <= st1_<name>;
      end case;
   end process;
```



b.) Moore-féle állapotgép

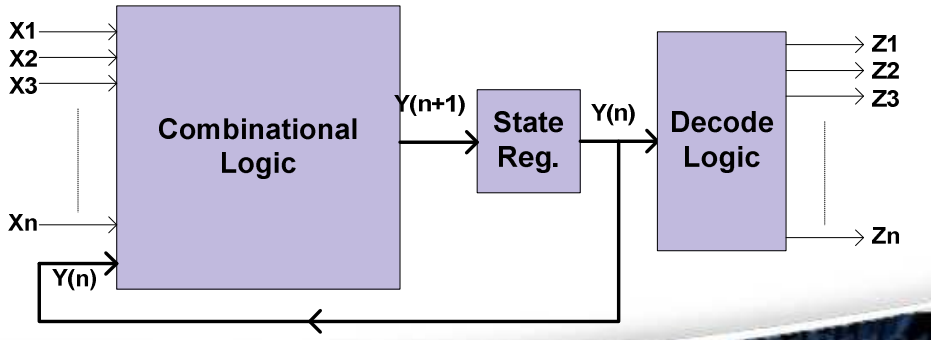
```
type state_type is (st1_<name_state>,
    st2_<name_state>, ...);

signal state, next_state : state_type;

signal <output>_i : std_logic;

SYNC_PROC: process (<clock>)
   begin
      if (<clock>'event and <clock> =
    '1') then
         if (<reset> = '1') then
            state <= st1_<name_state>;
            <output> <= '0';
         else
            state <= next_state;
            <output> <= <output>_i;
         end if;
      end if;
   end process;
```

```
NEXT_STATE_DECODE: process (state,
    <input1>, <input2>, ...)
begin
   next_state <= state;
   case (state) is
      when st1_<name> =>
         if <input_1> = '1' then
            next_state <= st2_<name>;
         end if;
      when st2_<name> =>
         if <input_2> = '1' then
            next_state <= st3_<name>;
         end if;
      when st3_<name> =>
         next_state <= st1_<name>;
      when others =>
         next_state <= st1_<name>;
   end case;
end process;
```

!

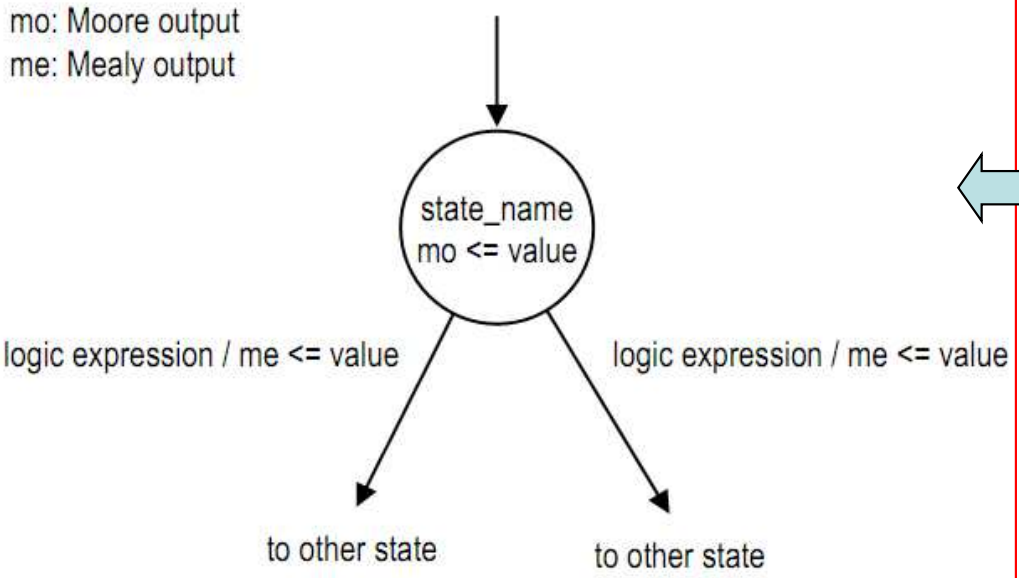
```
OUTPUT_DECODE: process (state)
   begin
      if state = st3_<name> then
         <output>_i <= '1';
      else
         <output>_i <= '0';
      end if;
   end process;
```

! Kimenetek direkt módon mindig az aktuális állapotoktól (state) függenek, de csak indirekt módon függenek a bemenetektől (inputs).



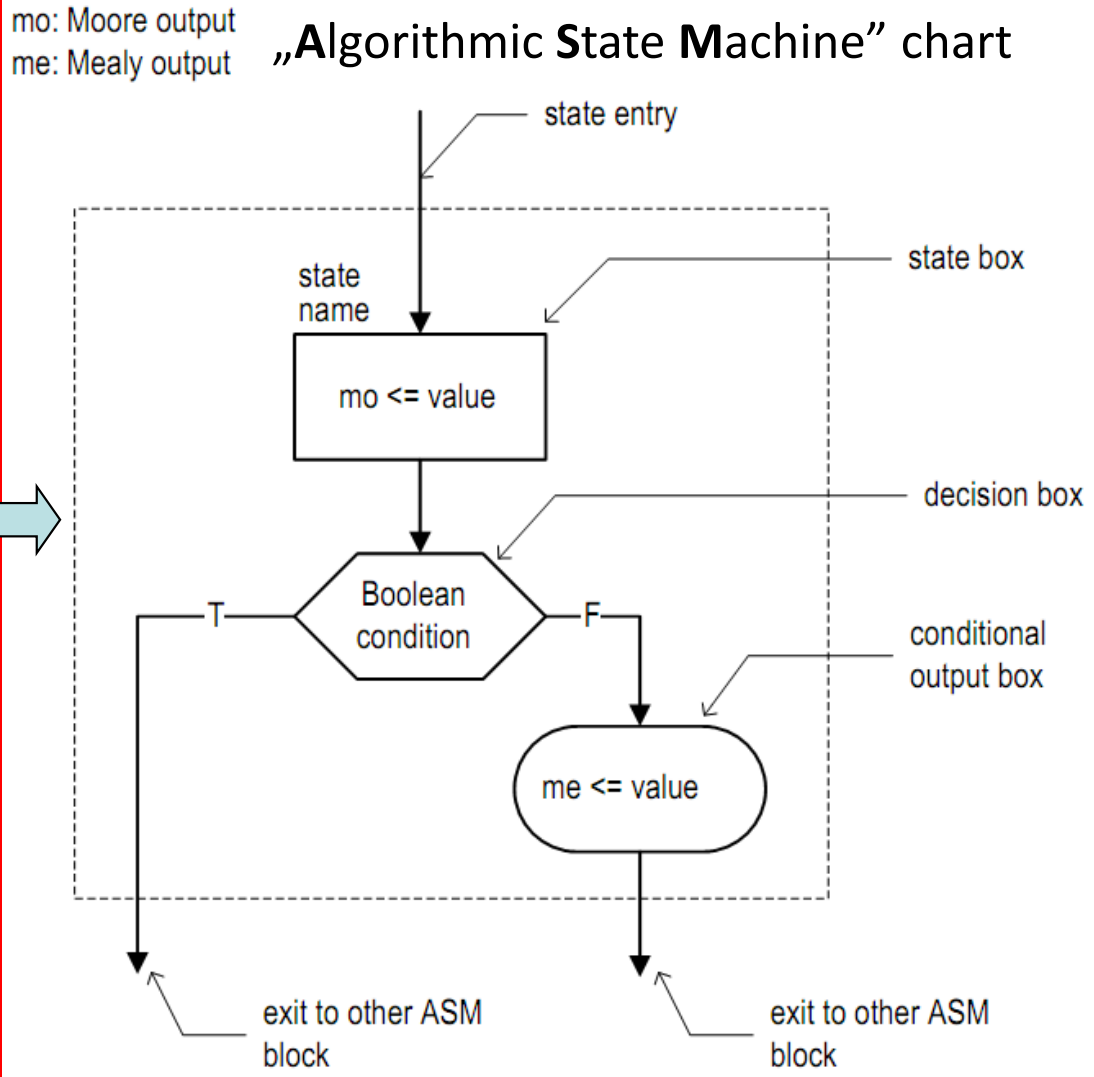
FSM ábrázolási módjai

- Állapot diagram



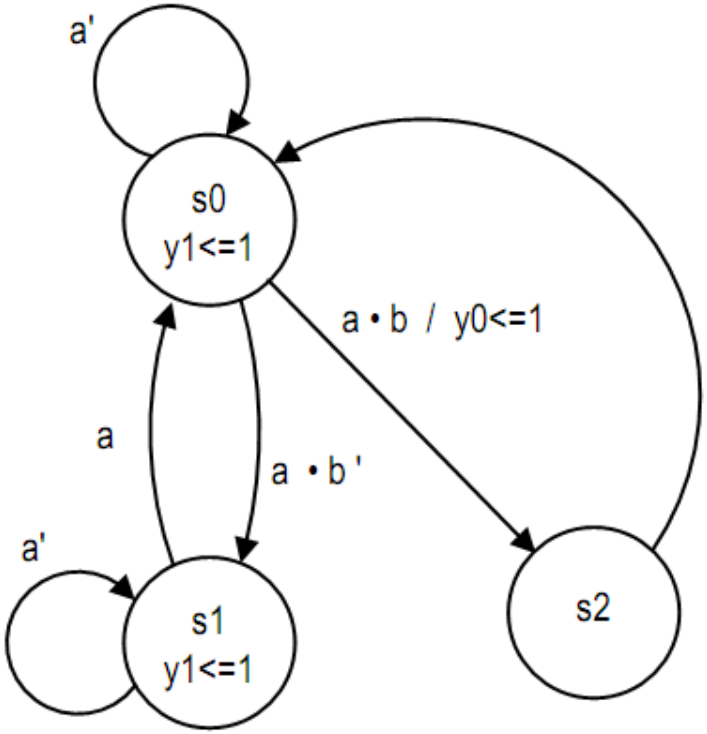
(me) Mealy kimenet : állapot átmeneten
(mo) Moore kimenet: állapotban (csúcs)

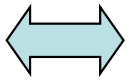
- ASM diagram

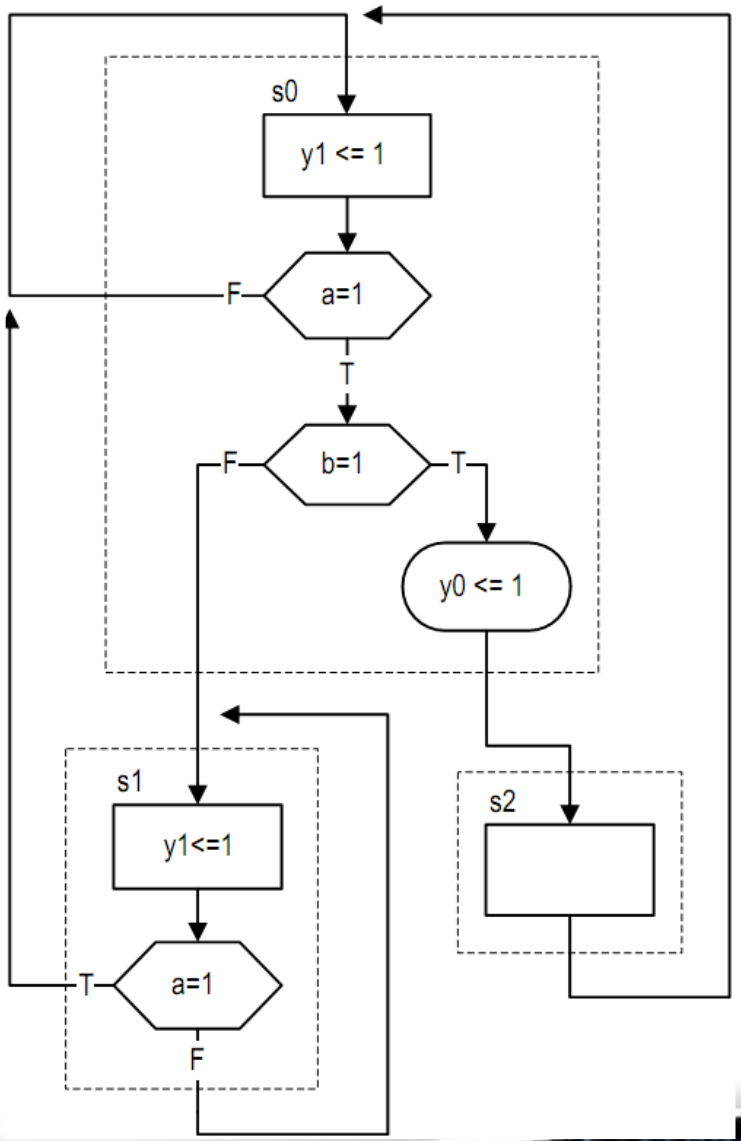


Példa: FSM

- Adott a következő állapotdiagram és / vagy ASM diagram





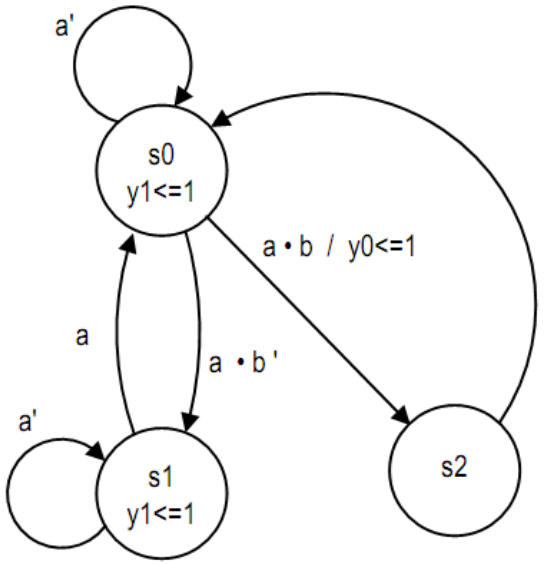


Symbol: a’ = not (a) --a inverze

Feladat: Valósítsa meg Mealy, ill. Moore automata modellként.

a.) Mealy modell

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm_eg is
   port(
      clk, reset: in std_logic;
      a, b: in std_logic;
      y0, y1: out std_logic
   );
end fsm_eg;

architecture mult_seg_arch of fsm_eg is
   type eg_state_type is (s0, s1, s2);
   signal state_reg, state_next:
          eg_state_type;
begin
   -- state register
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= s0;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
   -- next-state logic
   process(state_reg,a,b)
   begin
      case state_reg is
         when s0 =>
            if a='1' then
               if b='1' then
                  state_next <= s2;
               else
                  state_next <= s1;
               end if;
            else
               state_next <= s0;
            end if;
         when s1 =>
            if a='1' then
               state_next <= s0;
            else
               state_next <= s1;
            end if;
         when s2 =>
            state_next <= s0;
      end case;
   end process;
-- Mealy output logic (y0)
   process(state_reg, a, b)
   begin
      case state_reg is
         when s0 =>
            if (a='1') and (b='1') then
              y0 <= '1';
            else
              y0 <= '0';
            end if;
         when s1 | s2 =>
            y0 <= '0';
      end case;
   end process;
end mult_seg_arch;
```



b.) Moore modell

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm_eg is
   port(
      clk, reset: in std_logic;
      a, b: in std_logic;
      y0, y1: out std_logic
   );
end fsm_eg;

architecture mult_seg_arch of fsm_eg is
   type eg_state_type is (s0, s1, s2);
   signal state_reg, state_next:
eg_state_type;
begin
   -- state register
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= s0;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
```

```
-- next-state logic
   process(state_reg,a,b)
   begin
      case state_reg is
         when s0 =>
            if a='1' then
               if b='1' then
                  state_next <= s2;
               else
                  state_next <= s1;
               end if;
            else
               state_next <= s0;
            end if;
         when s1 =>
            if (a='1') then
               state_next <= s0;
            else
               state_next <= s1;
            end if;
         when s2 =>
            state_next <= s0;
      end case;
   end process;
-- Moore output logic (y1)
   process(state_reg)
   begin
      case state_reg is
         when s0|s1 =>
            y1 <= '1';
         when s2 =>
            y1 <= '0';
      end case;
   end process;
end mult_seg_arch;
```