ЦИФРОВА СХЕМОТЕХНИКА

ЛЕКЦИЯ #13

Конструкция process – синтаксис, разновидности (обобщение)

- Пример: Описание на архитектура с помощта на **process** на ЛЕ ИЛИ с два типа *sensitivity list* – ***implicit*** или ***explicit***:

```
signal X,Y,Z: bit;
OR: process (X,Y)      -- чрез sensitivity list тип implicit /OR-етикет/
 begin
        if X='1' then Z <='1';
           elsif Y='1' then Z<='1';
        else Z <='0';
        end if;
 end process;
```

```
signal X,Y,Z: bit;       -- чрез sensitivity list тип explicit
OR: process
 begin
        if  X='1'  then  Z <='1';
          elsif  Y='1'  then  Z<='1';
        else  Z <='0';
        end if;
             wait on X,Y;   -- оператор wait on списък от променливи
 end process;            -- въвежда "explicit" sensitivity list
```

Алтернативно въвеждане на архитектурно описание чрез конструкция block: синтаксис, особености

- Общ синтаксис:

```
Block_label: block <guard израз>   -- дефинираните в блока променливи са само
<generic>                          -- локално "видими"
            <port>
            <декларативна част на блока>
            begin
                <операторна част на блока>
            end block  Block_label;
```

- Пример: D-тригер с асинхронно нулиране:

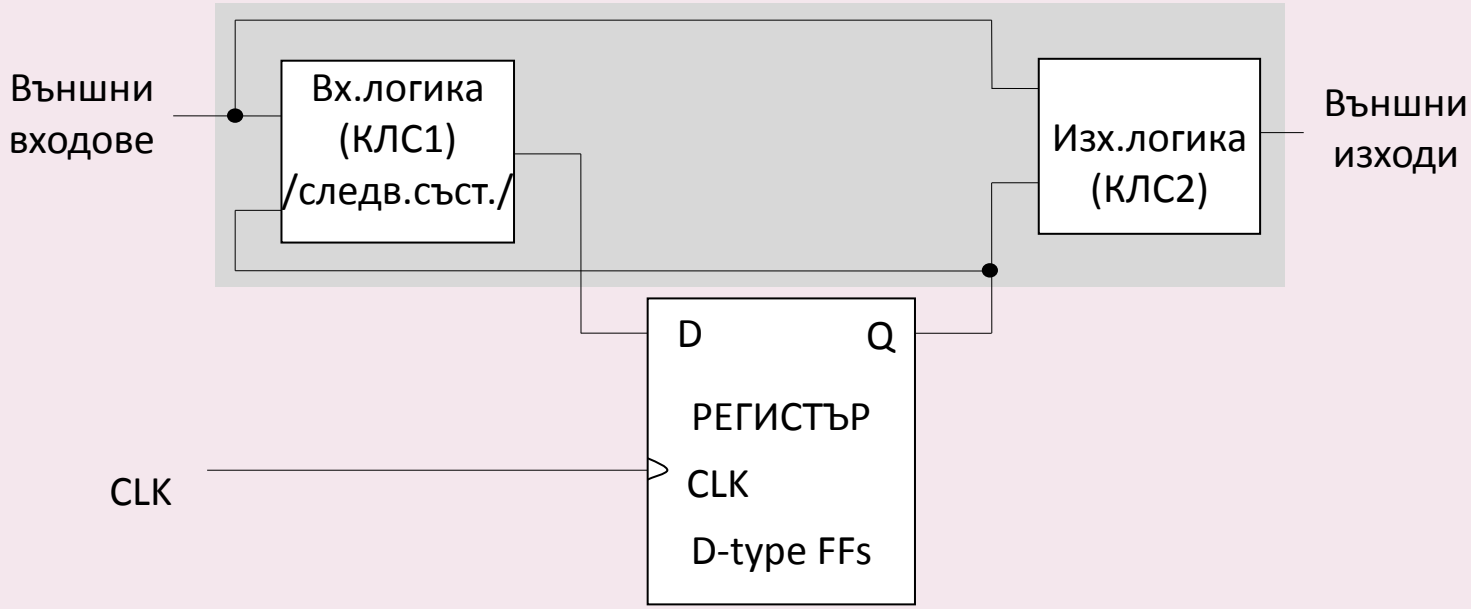
```
entity D_Latch  is
port (clk, clr, d: in bit; Q, Q_not: out bit);

architecture GUARDED_BlOCK  of  D_Latch  is   -- архитектура с изпозване на block
begin
      d_block: block (clk='1' or clr='1')
          signal S: bit;                          -- guard: допълнителен сигнал от тип Boolean
              begin
                  S <= guarded '0' when clr='1' else D;      -- оператора(ите) след guarded се
                  Q <= S after 10ns;                         -- изпълняват САМО ако стойността
                  Q_not <= not(S) after 10ns;                -- на guard е true
       end block d_block;
end GUARDED_BlOCK;
```

Процес на проектиране на последователностни схеми /синхронни структури/ с FPGA. Описание на базови синхронни структури с VHDL. Особености, примери

- Структура на Последователностна Схема, ПС (*Sequential Circuit*): изходно състояние – функция на вътрешното състояние (*internal state* → Елементи Памет, ЕП, тригери – D FF) и стойностите на входните сигнали;
- Синхронни (тактувани) схеми → методология за проектиране на синхронни структури (synchronous design methodology);
- Общ тактов сигнал към всички ЕП (тригери) – цялата система се контролира/тактува от общ тактов (CLK) сигнал .

➢ Максимална тактова честота (мин.период) : $T_{CLK} = 1/F_{MAX} = T_{ЗАК\_КС1} + T_{ЗАК\_CLK\text{-}Q} + T_{SETUP}$



Процес на проектиране на последователностни структури с FPGA
Тригерни структури и регистри – VHDL описание

▪ **D FF (D-type Flip-Flop)**: VHDL код на синхронен по фронт D-тригер, въвеждане на асинхронен Reset, синхронен Enable

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is
    port (
        rst: in std_logic;
        clk: in std_logic;
         d: in std_logic;
         q: out std_logic
          );
end D_FF;
```

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

```
architecture behav_rst of D_FF is
begin
   process (clk, rst)
   begin
      if rst = '1' then q <= '0';
      elsif (clk'event and clk='1') then
          q <= d;
      end if;
   end process;
end behav_ rst;
```

Процес на проектиране на последователностни структури с FPGA
Тригерни структури и регистри – VHDL описание

▪ **D FF (D-type Flip-Flop)**: VHDL код на синхронен по фронт D-тригер, въвеждане на асинхронен Reset, синхронен Enable (по ниво)

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is
    port (
        rst: in std_logic;
        en: in std_logic;
        clk: in std_logic;
         d: in std_logic;
         q: out std_logic
          );
end D_FF;
```

```
architecture behav_rst of D_FF is
begin
   process (clk, rst)
   begin
      if rst = '1' then q <= '0';
      elsif (clk'event and clk='1') then
          q <= d;
      end if;
   end process;
end behav_ rst;
```

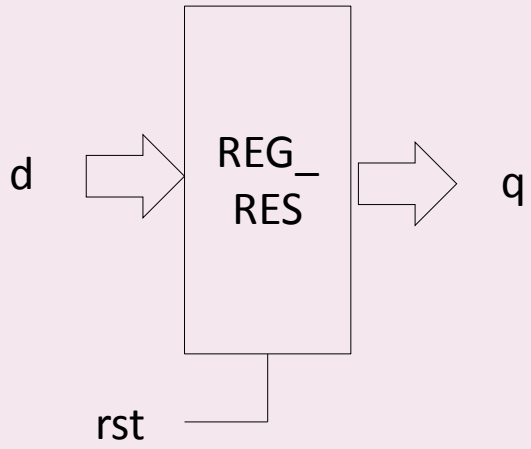
```
architecture behav_rst _en of D_FF is
begin
   process (clk, rst, en)
   begin
      if rst = '1' then q <= '0';
      elsif (clk'event and clk='1') then
          if en = '1' then q <= d;
          end if;
      end if;
   end process;
end behav_ rst_en;
```

Enable - за съгласуване и запазване на общата синхронизация между относително по-бърза и по-бавна подсистеми!

Процес на проектиране на последователностни структури с FPGA
Тригерни структури, регистри, броячи – VHDL описание

▪ **Регистър** – аналогично на описанието на тригер, но входни/изходни променливи са от ВЕКТОРЕН тип.

```
library ieee;
use ieee.std_logic_1164.all;
entity  REG_RES  is
     port (
          clk, rst:  in  std_logic;
          d:  in  std_logic_vector (15 downto 0);
          q:  out  std_logic_vector (15 downto 0);
           );
end REG_RES;
```

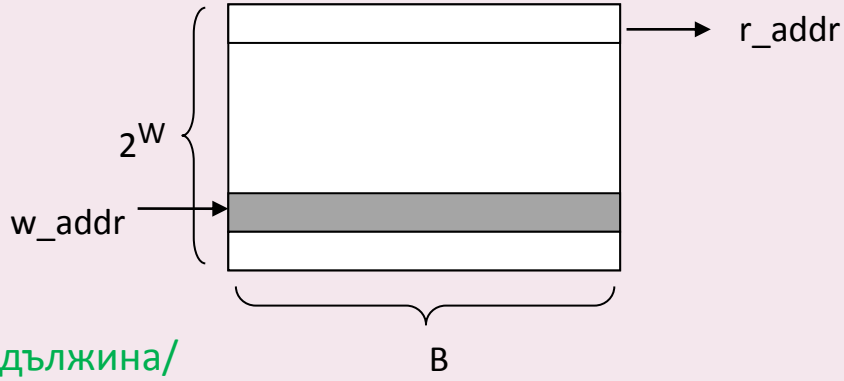



```
architecture  behav  of  REG_RES  is
begin
   process (clk, rst)
   begin
      if rst = '1' then
           q <= (others =>'0');  -- вместо "00..0"
      elsif  (clk'event  and  clk='1')  then
           q <= d;
      end if;
   end process;
end behav;
```

Процес на проектиране на последователностни структури с FPGA
Тригерни структури, регистри, броячи – VHDL описание

- **Съвкупност от регистри (Register file)** – масив от регистри с размерност 2^W (думи, регистри) x B (брой битове, дължина на регистъра)

! За реализация на бързо временно съхранение на голям обем данни.




```
library ieee;
use ieee.std_logic_1164.all;
use  ieee.numeric_std.all;
entity  REG_FILE  is
    generic (
            W: integer:=4;   -- брой адресни битове
            B:  integer:=15   -- брой битове в регистъра /дължина/
    port (
            clk, rst:  in  std_logic;
            wr_en:  in  std_logic;                          -- разрешение за запис
            w_addr, r_addr:  in  std_logic_vector (W-1 downto 0);   -- адреси за запис и за четене
            w_data: in  std_logic_vector (B-1 downto 0);    -- за запис
            r_data:  out  std_logic_vector (B-1 downto 0)    -- за четене
          );
end REG_FILE;
```

Процес на проектиране на последователностни структури с FPGA
Тригерни структури и регистри – VHDL описание

- **Съвкупност от регистри (Register file)** – масив от регистри с размерност 2^W (думи, регистри) x B (брой битове, дължина на регистъра)

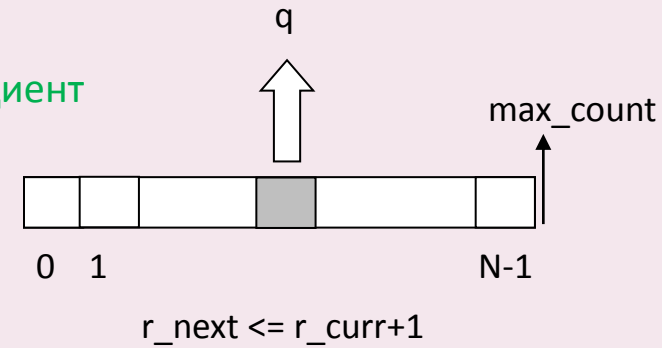
```
architecture  behav  of  REG_FILE  is
    type  reg_type_file  is  array (2**W-1 downto 0)  of  std_logic_vector(B-1 downto 0); -- спец. тип
    signal  array_reg:  reg_type_file     -- дефинира сигнал от тип масив от регистри

begin
      process (clk, rst)
      begin
            if  rst='0'  then
               array_reg <=  (others => (others => '0'))        -- нулиране
            elsif  (clk'event  and clk='1')  then
               if  wr_en='1'  then                  -- разрешение за запис
                  array_reg (to_integer(unsigned(w_addr)))  <= w_data;       -- запис на данните
               end if;
            end if;
       end process;
       r_data  <= array_reg (to_integer (unsigned(r_addr)))    -- четене на вече записаните данни
end  behav;
```

Процес на проектиране на последователностни структури с FPGA
Тригерни структури, регистри, броячи – VHDL описание

- **Описание на БРОЯЧ в непрекъснат режим на броене (т.нар. “free-running counter”)**

```
library ieee;
use ieee.std_logic_1164.all;
use  ieee.numeric_std.all;
entity  COUNTER_FREE  is
    generic  (N:integer: = 10);   -- дефиниране разрядност на брояча
    port (
        clk, rst: in std_logic;
        max_count: out  std_logic;  -- маркер за дост.макс.коефициент
        q: out  std_logic_vector (N-1 downto 0)
         );
end COUNTER_FREE;
```

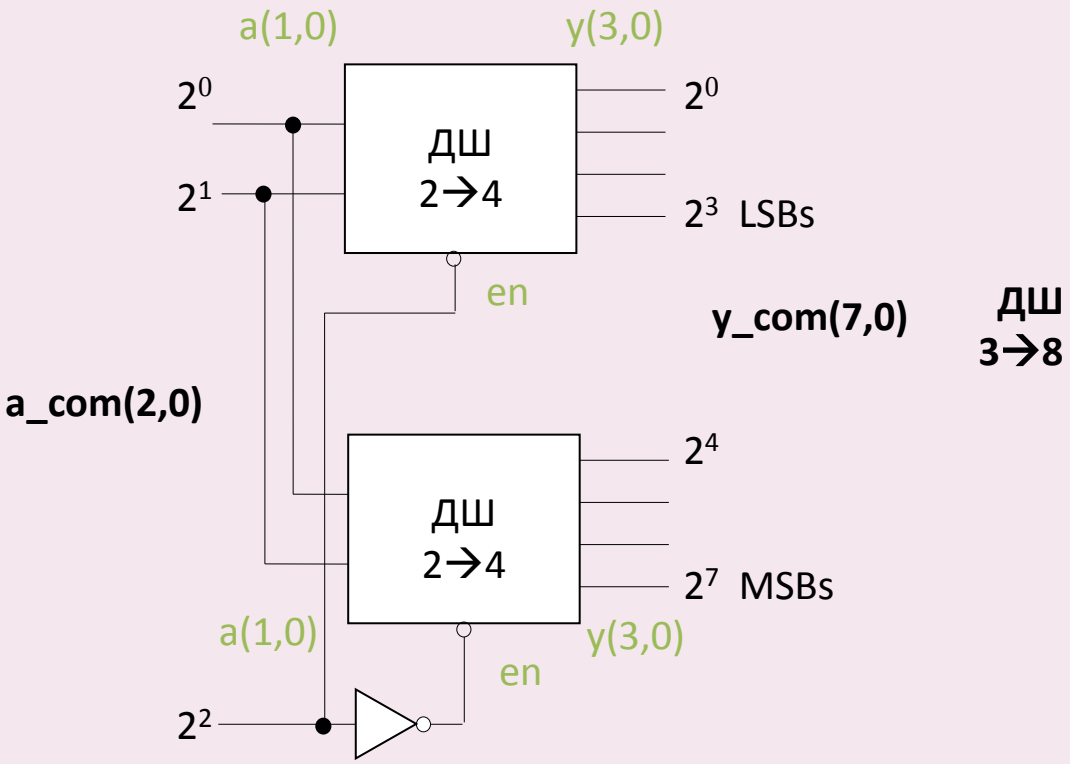



```
architecture  behav  of  COUNTER_FREE  is
     signal  reg_curr: unsigned(N-1 downto 0);
     signal  reg_next:unsigned (N-1 downto 0);
begin
  process (clk,rst)
      if rst='0'  then  r_curr <= (others =>'0');    -- нулиране
       elsif (clk'event  and  clk='1')  then  r_next <= r_curr;  -- следващо състояние равно на текущото
      endif;
  end process;
  r_next  <=  r_curr+1;                          -- увеличаване с единица следващо състояние
  q  <=  std_logic_vector(r_curr);               -- стойност на изходния вектор
  max_count  <= '1'  when  r_curr=(2**N-1)  else  '0';   -- достигнат максимум
end behav;
```

Процес на проектиране на последователностни структури с FPGA
Йерархично описание на проекти в среда Xilinx ISE WebPack

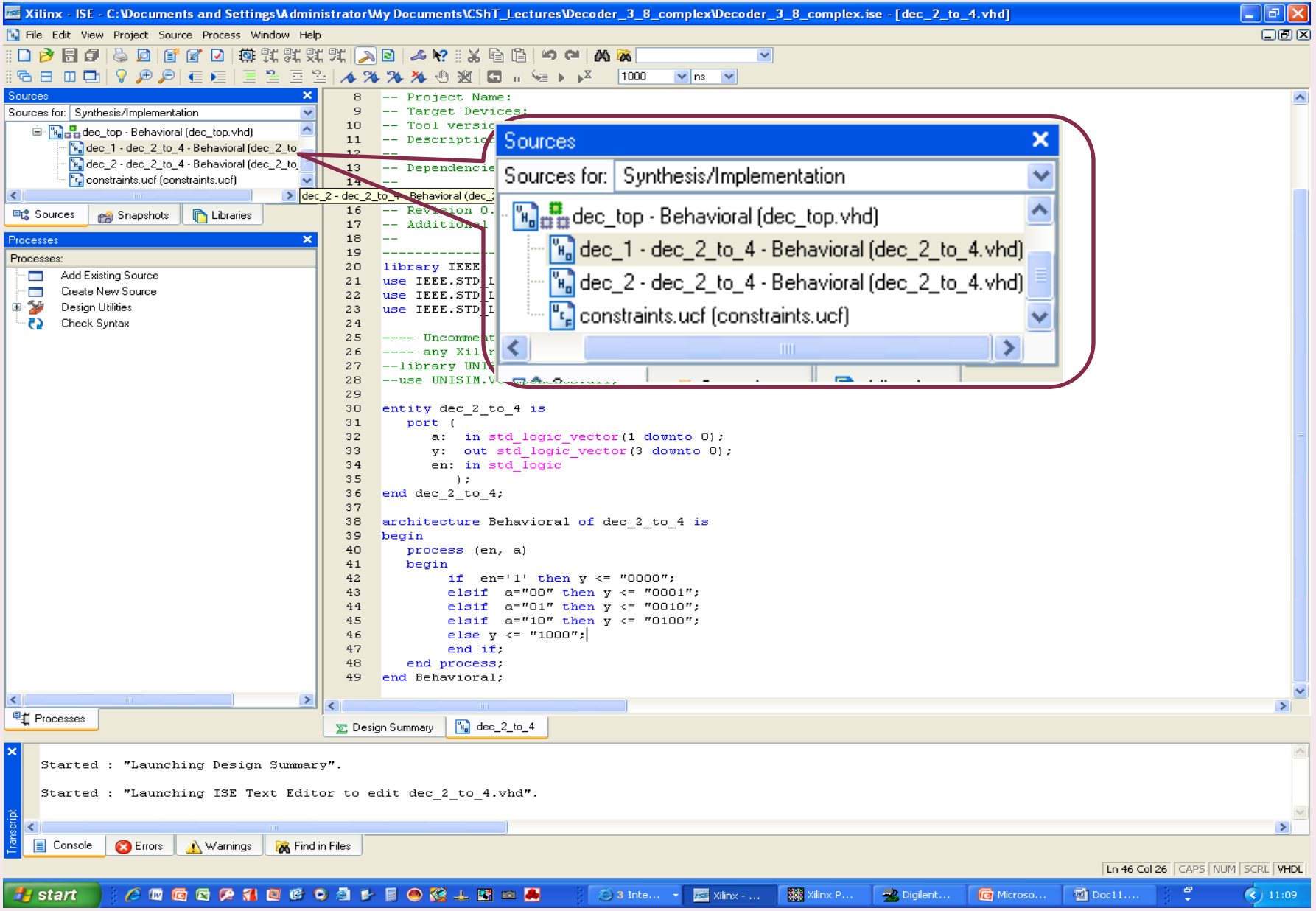
○ Пример: ДШ 3→8 изграден с блокове ДШ 2→4.

- изграждане на проекта (йерархия, особености на средата);
- изходен код на модулите на ниско ниво (ДШ 2→4);
- port map съответствие на входове/изходи.



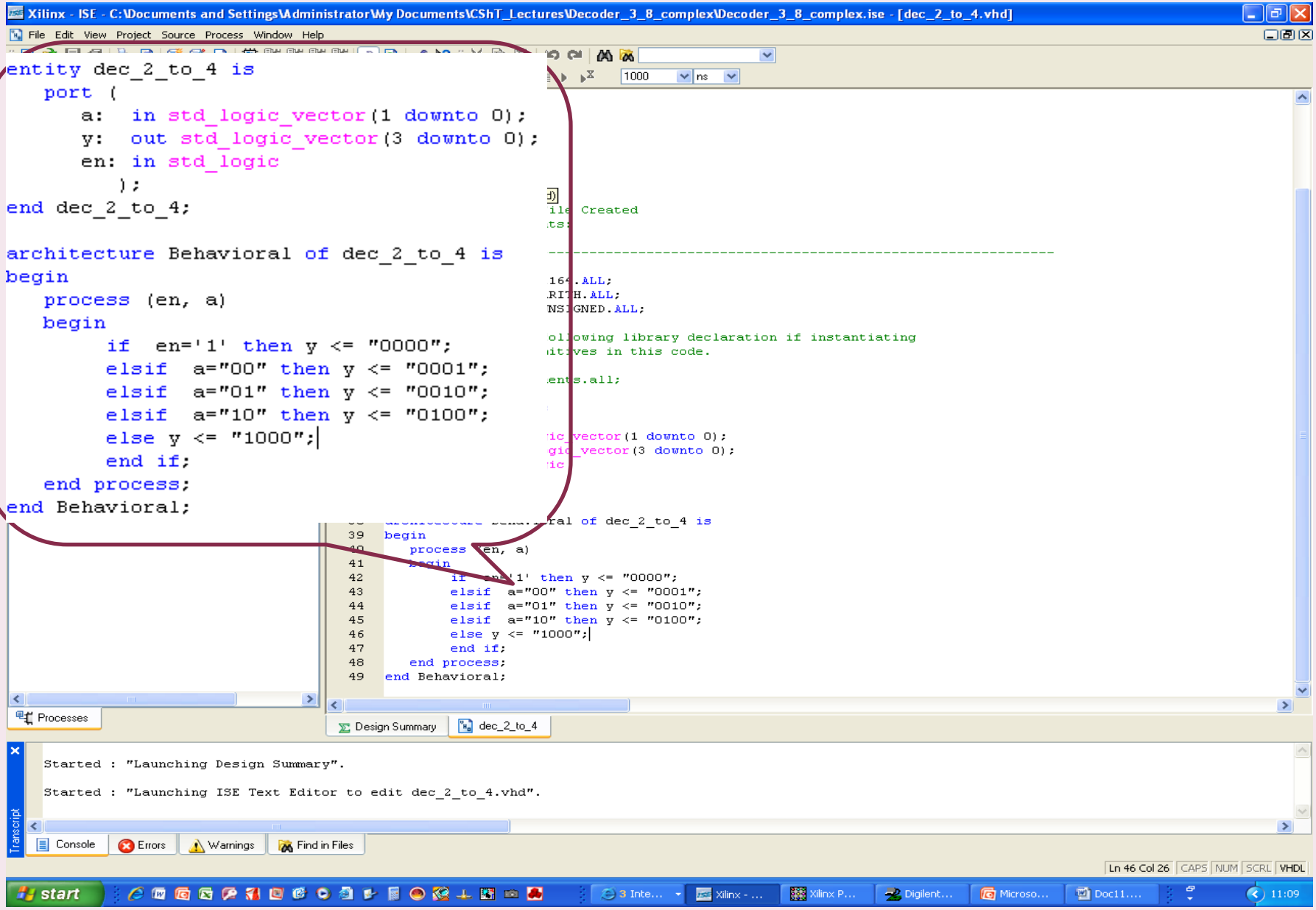
Процес на проектиране на последователностни структури с FPGA

Йерархично описание на проекти в среда Xilinx ISE WebPack (ДШ 3→8 от блокове ДШ 2→4)



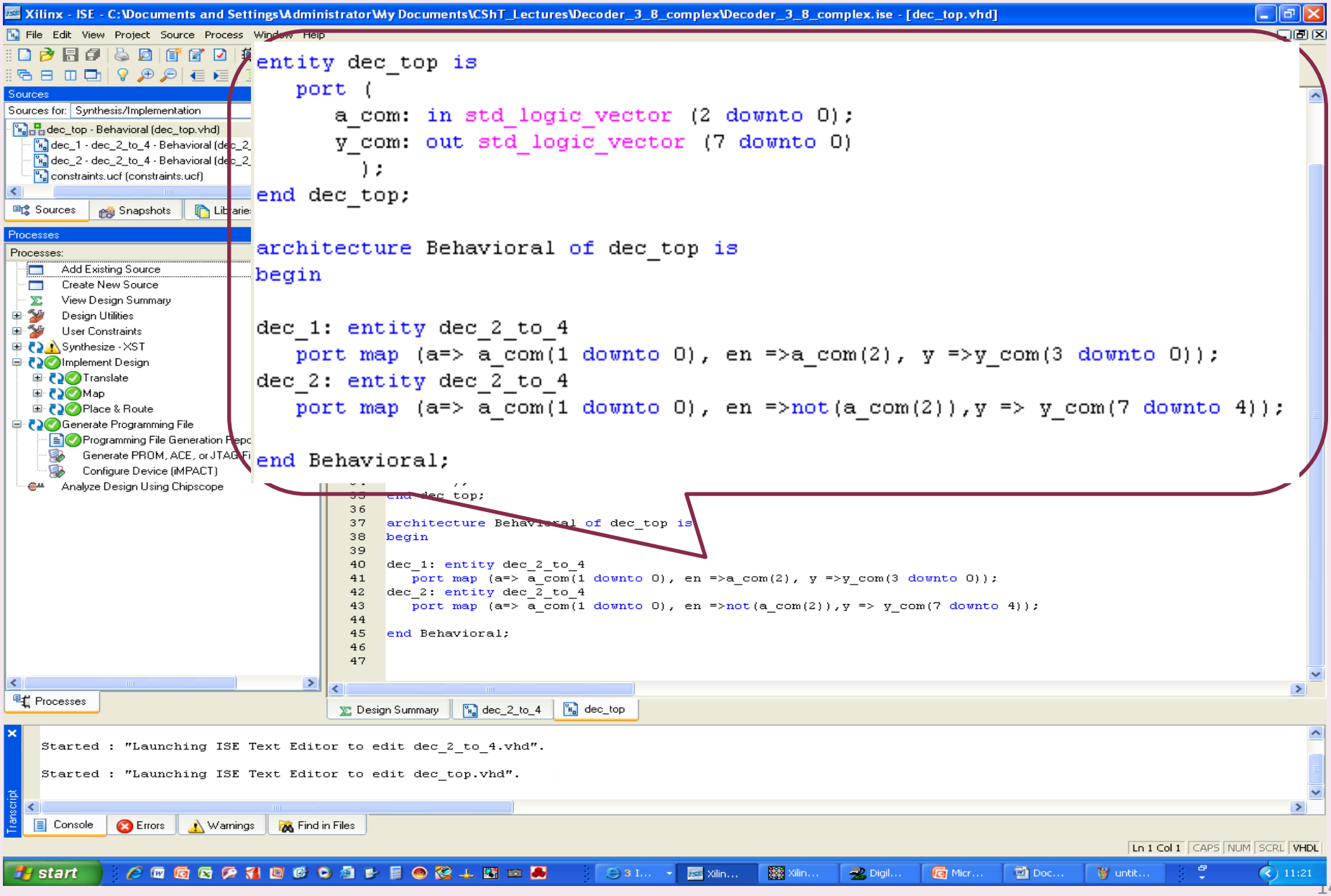
Процес на проектиране на последователностни структури с FPGA

Йерархично описание на проекти в среда Xilinx ISE WebPack (ДШ 3→8 от блокове ДШ 2→4)



Процес на проектиране на последователностни структури с FPGA

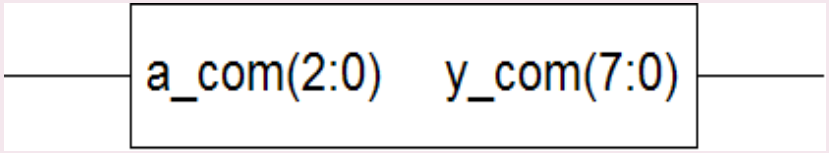
Йерархично описание на проекти в среда Xilinx ISE WebPack (ДШ 3→8 от блокове ДШ 2→4)

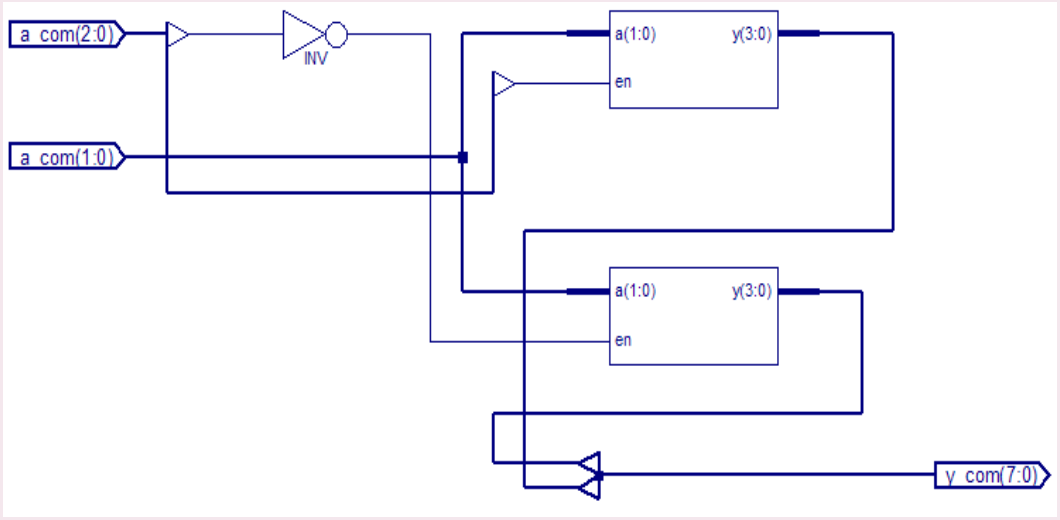


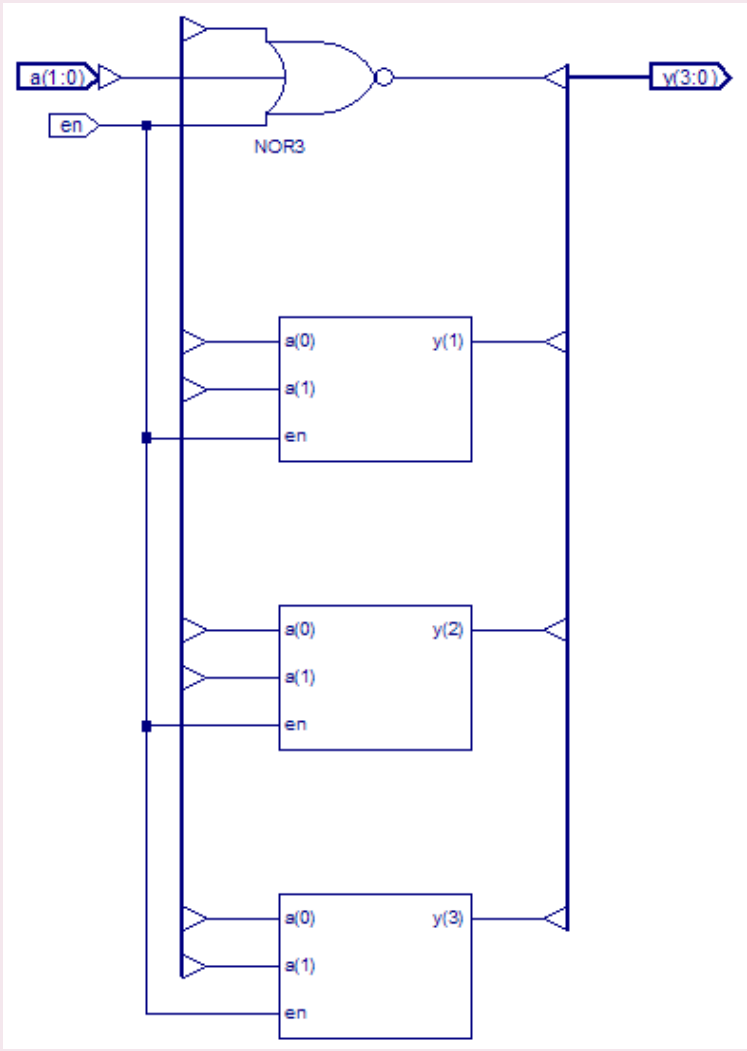
Процес на проектиране на последователностни структури с FPGA

Йерархично описание на проекти в среда Xilinx ISE WebPack (ДШ 3→8 от блокове ДШ 2→4)

RTL представяне на проекта, използвани ресурси







Процес на проектиране на последователностни структури с FPGA
Тригерни структури, регистри, броячи – VHDL описание

- Практически аспекти за използване ресурсите на развойни платки Digilent BASYS/BASYS2 Board:

  - Управление на седемсегментите индикатори - изходен код (принцип, необходимост) ;

  - Мултиплексиране (циркулиране) на изхода към отделните панели - реализация;

  - Използване на вградения тактов генератор в развойната платка (25/50/100 MHz):

    - избор на тактова честота;

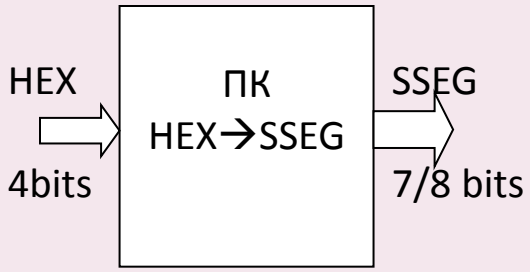
    - възможност за използване на допълнителен външен тактов генератор, ограничение.

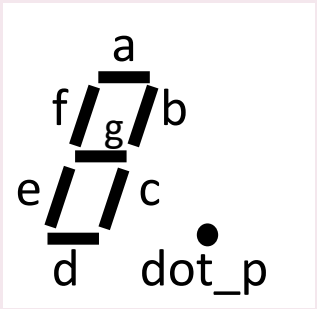
VHDL описание на отделни функционални блокове

- Седемсегментна дешифрация (преобразувател на код ПК HEX→SSEG LED) за управление на 7-сегментни индикатори

```
library ieee;
use ieee.std_logic_1164.all;
entity HEX_to_SSEG is
    port (     hex: in std_logic_vector(3 downto 0);
               sseg:  out std_logic_vector(7 downto 0);
               dot_p: in std_logic  );
end HEX_to_SSEG;

architecture behav of HEX_to_SSEG is
begin
      with hex select
         sseg (6 downto 0) <=
              "0000001" when "0000",   -- изобразява 0
              "1001111" when "0001",   -- изобразява 1
              "0010010" when "0010",   -- изобразява 2
              "0000110" when "0011",   -- изобразява 3
              "1001100" when "0100",   -- изобразява 4
              "0100100" when "0101",   -- изобразява 5
              "0100000" when "0110",   -- изобразява 6
              "0001111" when "0111",   -- изобразява 7
              "0000000" when "1000",   -- изобразява 8
              "0000100" when "1001",   -- изобразява 9
              "0001000" when "1010",   -- изобразява a
              "1100000" when "1011",   -- изобразява b
              "0110001" when "1100",   -- изобразява c
              "1000010" when "1101",   -- изобразява d
              "0110000" when "1110",   -- изобразява e
              "0111000" when others;    -- изобразява f
        sseg(7) <= dot_p;     --  десетична точка  /отделно като 8-ми бит/
end behav;
```



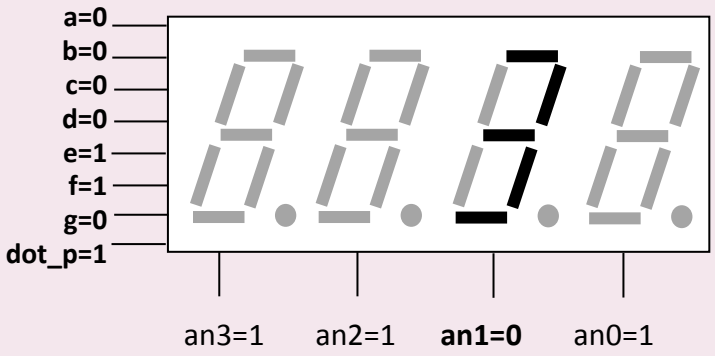


VHDL описание на отделни функционални блокове

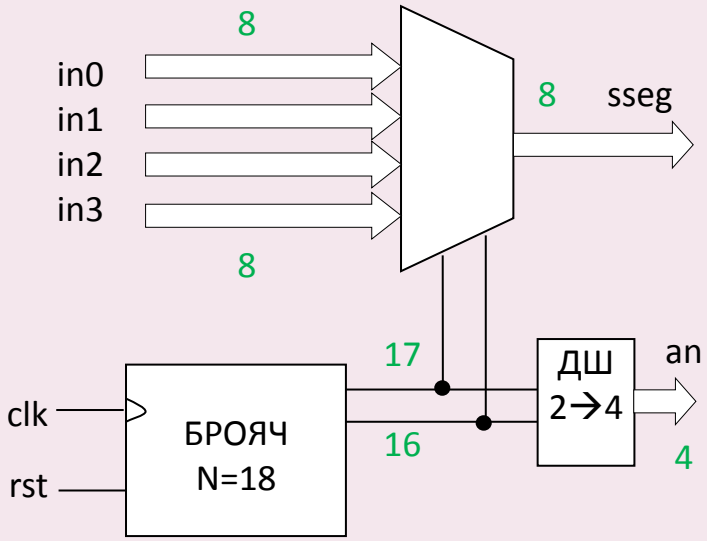
- **Управление на повече от един 7-сегментни индикаторни панела чрез мултиплексиране**

○ *необходимост – поради наличие на общи катоди за всички индикатори (пестят се шини: напр при 4 SSEGs – общо 8(12) вместо 32(36);*

○ *в развойни платки на Digilent – реализация с индивидуални общи катоди и отделни аноди за всеки SSEG /с активно ниско ниво за аноди и катоди/.*




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DISP_MULT_SEGS  is
    port (   clk, rst:  in  std_logic;
             in3,in2,in1,in0:   in  std_logic_vector(7 downto 0);
             sseg:  out  std_logic_vector(7 downto 0);
             an:  out  std_logic_vector(3 downto 0);
             );
end  DISP_MULT_SEGS ;
```

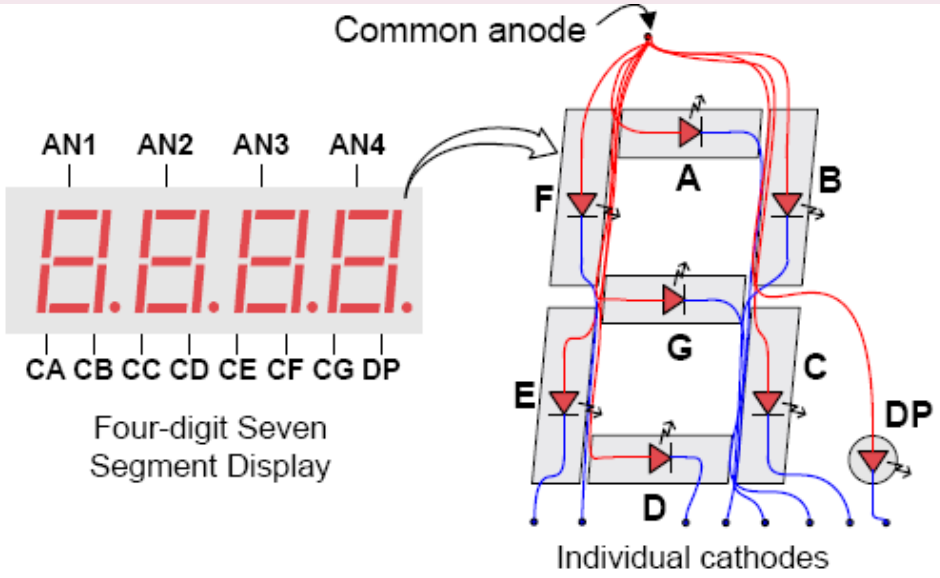


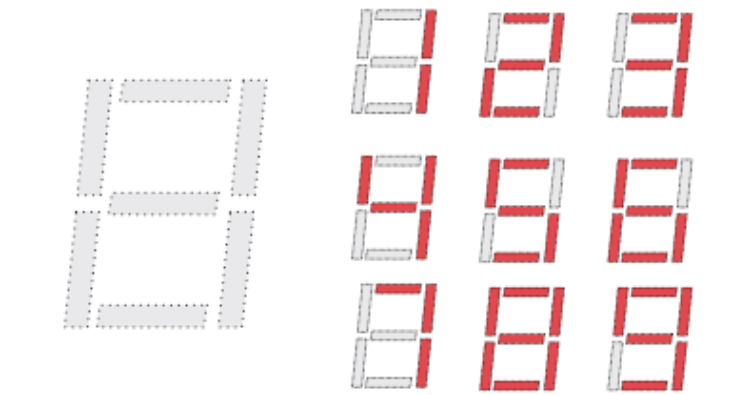
VHDL описание на отделни функционални блокове

- **Управление на повече от един 7-сегментни индикаторни панела чрез мултиплексиране**

○ *необходимост – поради наличие на общи катоди за всички индикатори (пестят се шини: напр при 4 SSEGs – общо 12 вместо 36;*

○ *в развойни платки на Digilent – реализация с индивидуални общи катоди и отделни аноди за всеки SSEG*





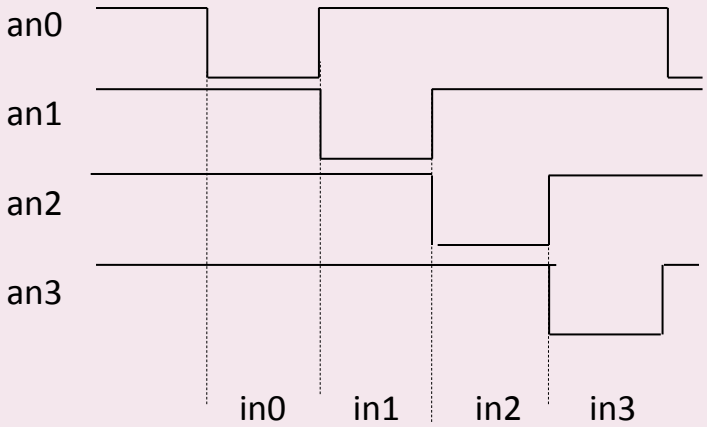
An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

VHDL описание на отделни функционални блокове

- **Управление на един от четири 7-сегментни индикаторни панела чрез мултиплексиране**

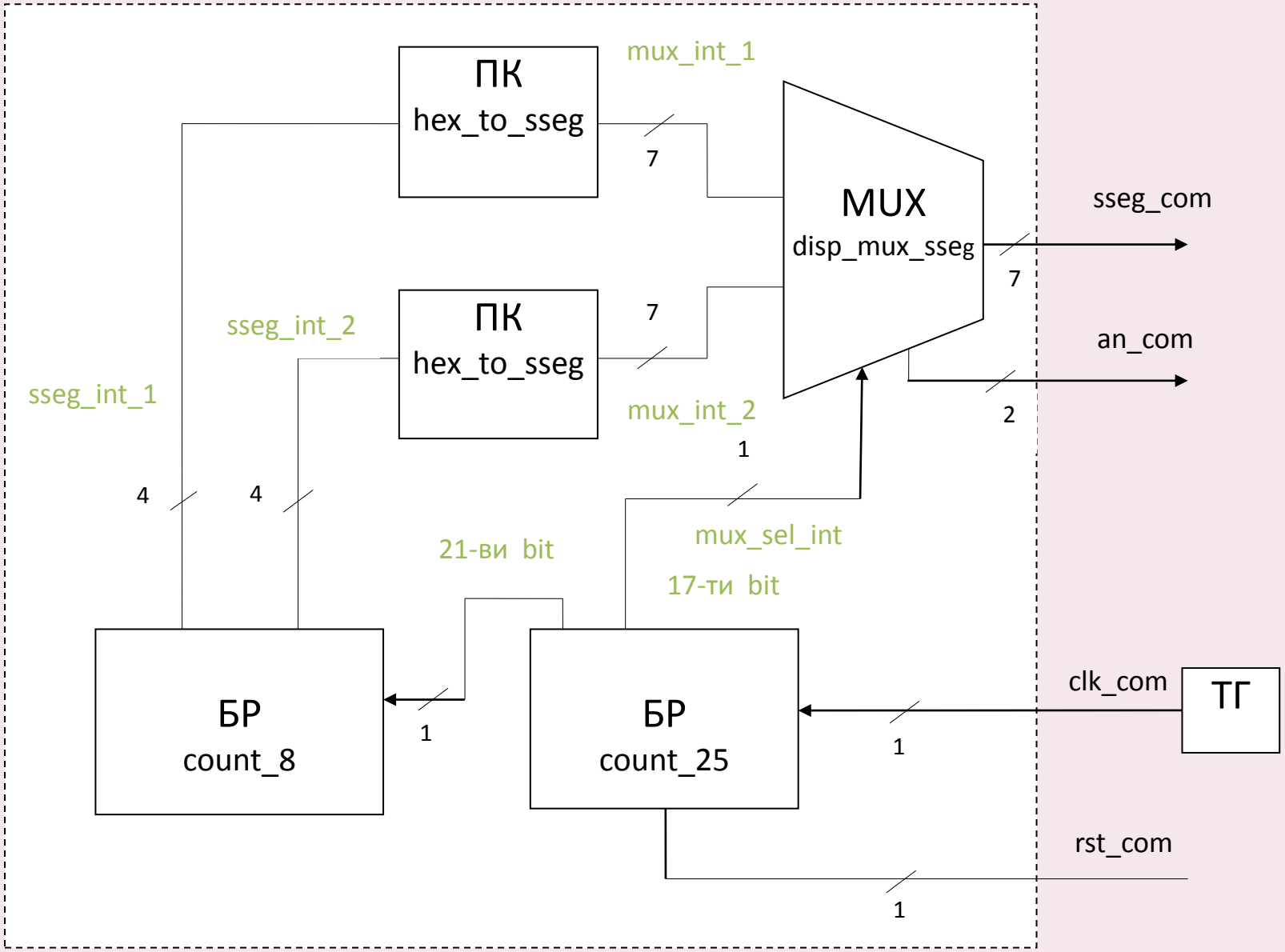
```
architecture behav  of  DISP_MULT_SEGS  is
  constant  N:  integer:=18;                    -- опресняване честота прибл.800Hz (762Hz)=50MHz/2**16/
  signal q_curr, q_next:  unsigned(N-1 downto 0);
  signal sel: std_logic_vector(1 downto 0);     -- управляващи /селекторни/ сигнали към MUX

begin
  process (clk,rst)
      begin
        if rst='1' then
              q_curr <= (others =>'0');
        elsif  (clk'event  and  clk='1')  then
              q_curr <= q_next;
        end if;
   end process;
   q_next  <=  q_curr+1;
   sel  <=  std_logic_vector(q_curr (N-1 downto N-2));
   process (sel, in0, in1, in2, in3)
   begin
       case  sel  is          -- мултиплексиране на векторите за катодите за всеки панел
              when "00" =>  an <="1110";  sseg <= in0;
              when "01" =>  an <="1101";  sseg <= in1;
              when "10" =>  an <="1011";  sseg <= in2;
              when "11" =>  an <="0111";  sseg <= in3;
      end case;
   end process;
end behav;
```



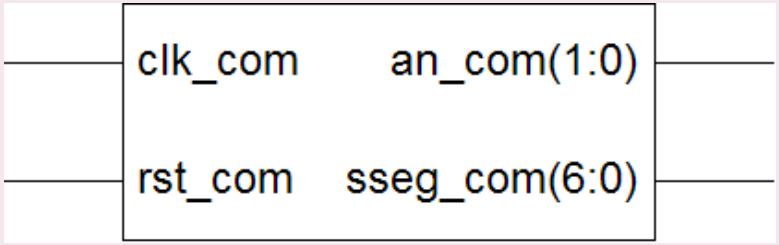
катоди на SSEG (конкретно число/знак)

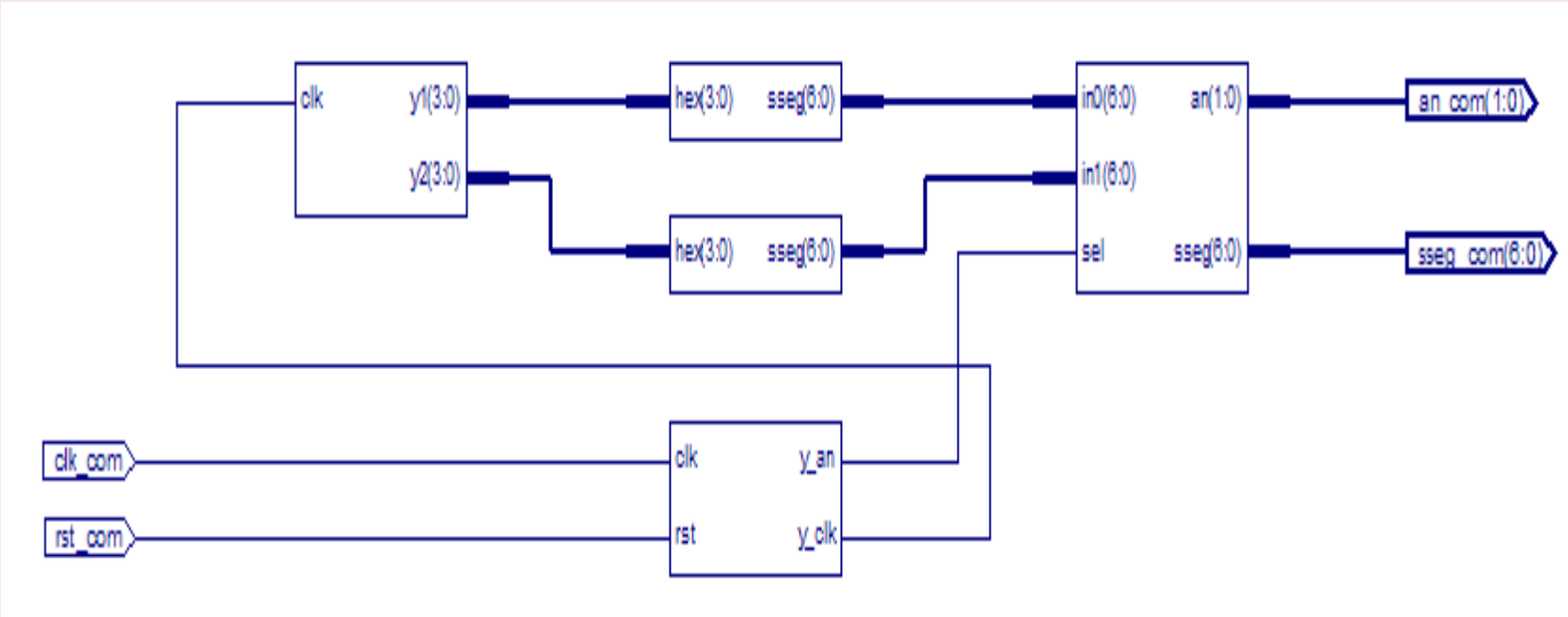
VHDL описание на отделни функционални блокове /упраление на 2 сегмента/



VHDL описание на отделни функционални блокове

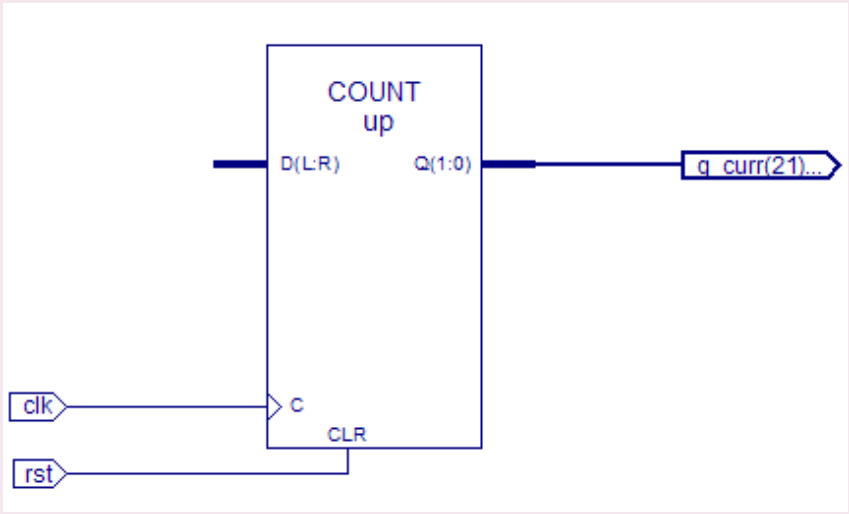
Хардуерна интерпретация на проекта и на отделните блокове на архитектурно и RTL-ниво

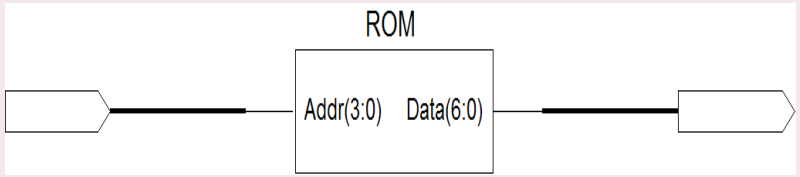


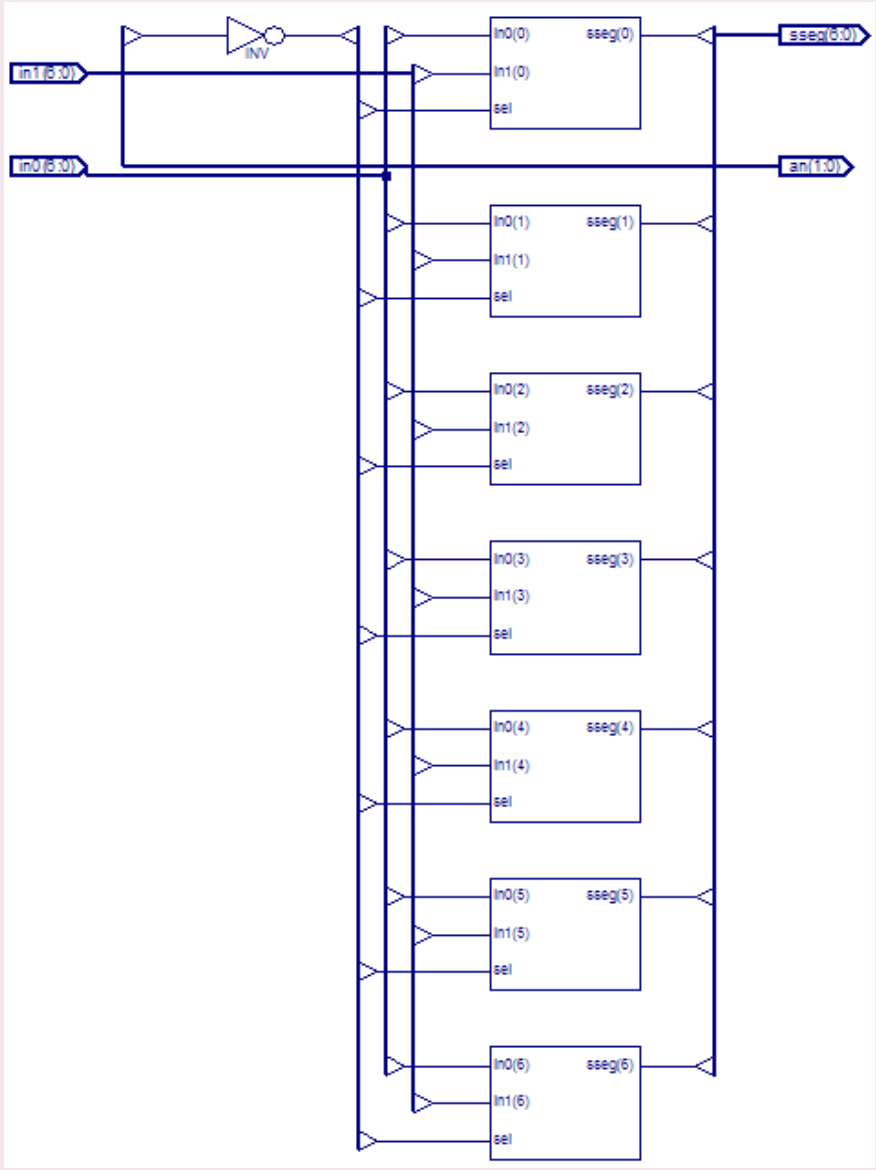


VHDL описание на отделни функционални блокове

Представяне на отделните блокове на RTL-ниво







VHDL описание на отделни функционални блокове

Представяне на отделните блокове на RTL-ниво

```
=========================================================================
Device utilization summary:
---------------------------

Selected Device : 3s100etq144-5

 Number of Slices:                       24  out of    960     2%
 Number of Slice Flip Flops:             30  out of   1920     1%
 Number of 4 input LUTs:                 45  out of   1920     2%
 Number of bonded IOBs:                  11  out of    108    10%
 Number of GCLKs:                         1  out of     24     4%
=========================================================================

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 4.677ns (Maximum Frequency: 213.819MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 6.400ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
=========================================================================
```

VHDL описание на отделни функционални блокове

Използване на вградения тактов генератор в развойната платка (25/50/100 MHz):

✓ избор на тактова честота (в примера – от модул **count_25**;

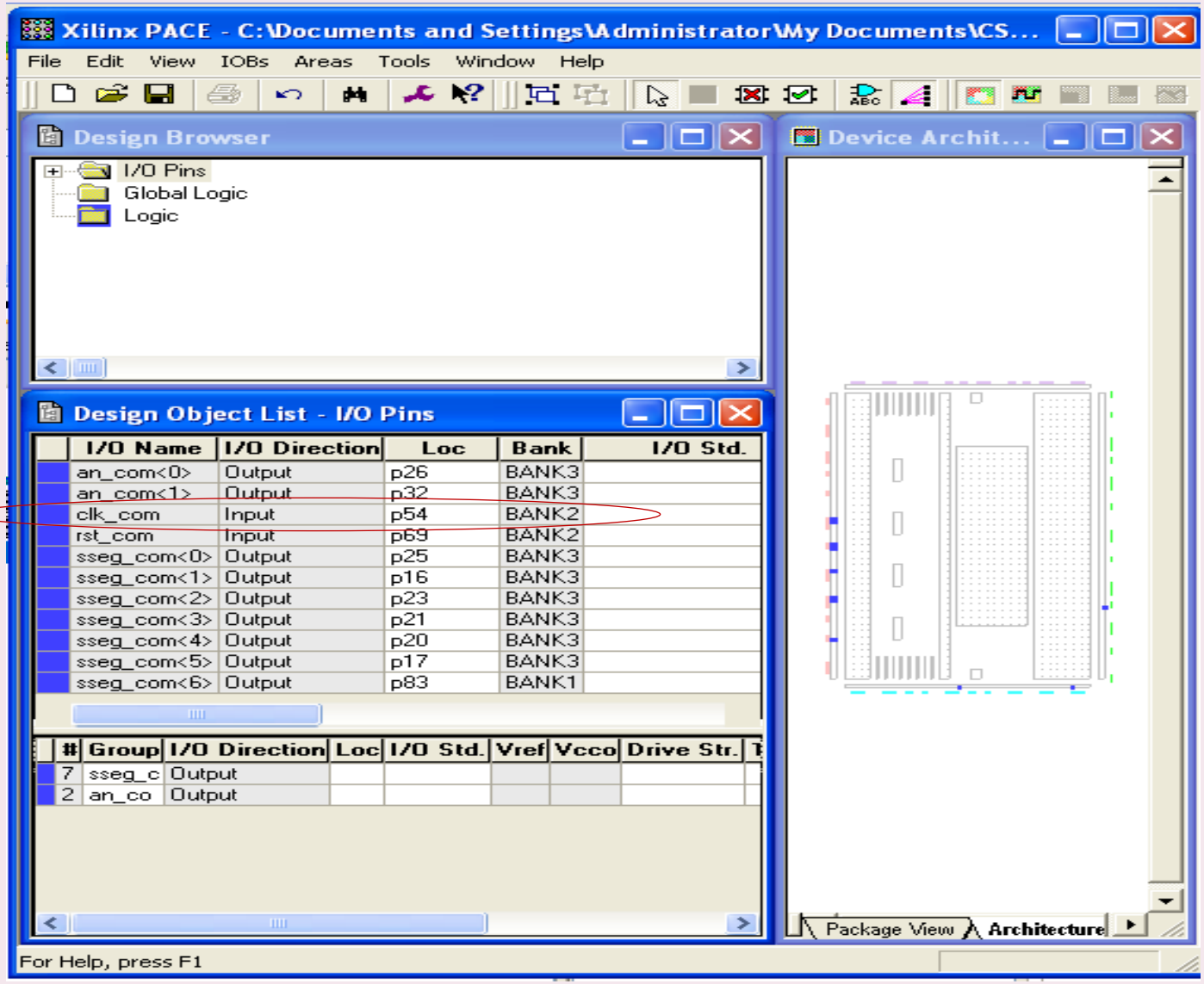
Код за брояча (избор на битове за управление на процеси)

```
entity count_25 is
port ( clk: in std_logic; rst: in std_logic;
        y_clk: out std_logic; y_an: out std_logic );
end count_21;

architecture Behavioral of count_21 is
constant  N:  integer:=25;              -- MSB за следващия брояч (прибл.1.4Hz)
constant  K:  integer:=17;              -- за управление на MUX и 7-сегментните индикатори
signal  q_curr, q_next:  unsigned (N-1 downto 0);
signal  max_count: std_logic;
begin
   process (clk,rst)
   begin
      if  rst='1'   then  q_curr <=(others =>'0');
        elsif (clk'event  and  clk='1')  then  q_curr <= q_next;
      end if;
   end process;
q_next  <=  q_curr+1;
y_an  <= std_logic(q_curr(K-1)); y_clk <= std_logic(q_curr(N-1));
max_count  <= '1'  when  q_curr=(2**N-1)  else  '0';
end Behavioral;
```

VHDL описание на отделни функционални блокове

✓ **UCF файл за примера, използване на вградения ТГ, ограничения**



Подпрограми (ПП) във VHDL – видове, особености, приложение

- Съдържат съвкупност от оператори, които могат да бъдат извиквани от произволни места в кода на програмата;
- Биват два вида: функции, процедури.

Особености: в езиците с общо предназначение – ПП са един вид естествена форма на йерархия (задачата се разделя на подзадачи и всяка може да се извиква съответно).

Във VHDL йерархията се дефинира от двойката entity/architecture, описваща компонент (entity) – т.е. всеки дизайн (проект) следва да бъде декомпозиран (partitioned) в отделни компоненти. Всеки от компонентите може в последствие да се използва в проекти от по-високо ниво. ДИРЕКТНАТА ДЕКОМПОЗИЦИЯ НА ПРОЕКТИ ВЪВ VHDL КАТО ПП Е ГРЕШКА!

ПП не могат да съдържат оператори от цикличен тип – т.е. процеси (process). Могат да служат само за описание на комбинационни логически структури. Приложение – само за малки по обем (т.нар. atomic) операции!

Подпрограми (ПП) във VHDL – видове, особености, приложение

❑ **Функции във VHDL** – извикване, деклариране, примери:

Извикване:

- в изрази при присвояване;
- или от конструкции от типа if или case

Пример:
```
library ieee;
use std_logic_1164.all
entity carry_f is
port (a1,b1,c1: in std_logic; c_out: out std_logic);
end;

architecture behaviour of carry_f is
begin
  c_out <= carry (a1,b1,c1);      -- извиква функция с име carry()
end;
```

Пример *(използване в process)*:
```
process (a1, b1, c1)   -- всеки път при промяна
begin                  -- на a1, b1 или c1
  c_out <= carry (a1,b1,c1); -- т.е. вместо signal
end process;
```

Подпрограми (ПП) във VHDL – видове, особености, приложение

❑ **Функции във VHDL** - извикване, деклариране, примери:

Деклариране:

- *в декларационната част на process (преди begin);*
- *в декларационната част на architecture;*
- *на произволно място в рамките на пакети;*
- *в рамките на други ПП.*

Пример:

```
function CARRY (bit1, bit2, bit3: in std_logic) return std_logic is
  variable result: std_logic;
begin
  result: (bit1 and bit2) or (bit1 and bit3) or (bit2 and bit3);
  return result;
end;
```

Вариант: **function** carry (bit1, bit2, bit3: std_logic) **return** std_logic **is** – подразбира се
-- режим **in** за входните параметри

Подпрограми (ПП) във VHDL – видове, особености, приложение

❑ **Функции във VHDL** - синтаксис:

- входни параметри / тип;

- тип на връщаната стойност – САМО ЕДНА, от произволен тип;

- изчислителен алгоритъм за получаване на връщаната стойност.

Пример: Функция, връщаща максималното от две цели числа:

```
function F_MAX(x,y:integer) return integer is
begin
        if  x>y  then  return  x;
              else  return  y;
        end if;
end;
```

Обръщане/използване на външни за функцията променливи или сигнали не се допуска!

Подпрограми (ПП) във VHDL – видове, особености, приложение

- **Процедури -** синтаксис:

  - *име*
  - *входни и изходни параметри / тип* – *от всеки режим* (*mode*) *in, out, inout*
  - *декларации в тялото на процедурата*
  - *изчислителен алгоритъм*
  - НЯМАТ ВРЪЩАНА СТОЙНОСТ!

  Параметрите от режим in – входни за процедурата, от режим out – изходни резултати, от режим inout – и като входни и като изходни ;

Пример: Процедура, връщаща целочисления резултат от делене на едно число на 16

```
procedure DIV_16(a: inout integer) is     -- параметърът a е и
begin                  -- входна и изходна променлива - mode inout
      a:= a/16;
end;
```