dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Εντολές επανάληψης – Generic – Constant
Κωδικοποιητές – Αποκωδικοποιητές - Ακολουθιακά Κυκλώματα

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Επαναλήψεις

Εντολή FOR

optional_label: for variable in range loop

sequential statements;

end loop;

θα το χρησιμοποιείτε για simulation

Δεν χρειάζεται δήλωση. Ο τύπος υπονοείται ως integer

Μόνο μέσα σε Process

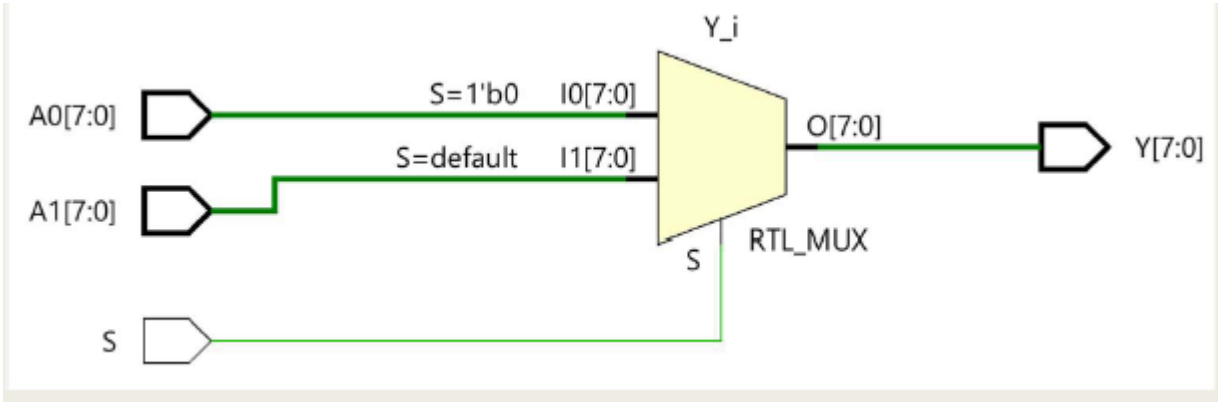
```
for i in 0  to 2 loop
   a_tb<=std_logic_vector(to_signed(i,a_tb'length));
   for j in 0 to 2 loop
      b_tb<=std_logic_vector(to_signed(j,a_tb'length));wait for 10ns;
   end loop;
end loop;
```

wait μόνο σε simulation

Το for εκτελείται ακολουθιακά. Πρέπει να εκτελεστεί σε ένα κύκλο ρολογιού

VHDL – Generic

```
entity MUX2in1_n is
generic (WIDTH : positive := 8); -- προεπιλεγμένη τιμή
port (      S: in STD_LOGIC;
            A0: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
            A1: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
            Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end MUX2in1_n;
architecture BEHAVIORAL of MUX2in1_n is
begin
process (A0, A1, S)
begin
            if (S = '0') then
                        Y <= A0;
            else
                        Y <= A1;
            end if;
end process;
end BEHAVIORAL;
```



VHDL – Generic

Παραμετροποίηση του μεγέθους μίας αρτηρίας (bus) σε μία οντότητα με τη δήλωση της εντολής generic που ορίζει την σταθερά WIDTH

- Η δήλωση της εντολής generic γίνεται πριν από τη δήλωση των ports στην αρχή της οντότητας
- Η σταθερά WIDTH είναι θετικός ακέραιος (positive) και μπορεί να έχει προεπιλεγμένη τιμή
  - generic (WIDTH: positive := 8);
- Η σταθερά WIDTH χρησιμοποιείται κατά τη δήλωση των ports
  - STD_LOGIC_VECTOR (WIDTH-1 downto 0);
- Η τιμή της σταθεράς WIDTH μπορεί να παρακάμψει την προεπιλεγμένη τιμή με τη φράση generic map (συνδυάζεται με το port map).
  - generic map (WIDTH => 8)
- Πιθανότατα θα έχει κάποια προβλήματα στην προσομοίωση (Σύνθεση/Υλοποίηση)

https://electronics.stackexchange.com/questions/571759/when-to-set-defaults-for-vhdl-generics
https://fpgatutorial.com/vhdl-generic-generate/

VHDL – Constant

Μπορούν να δηλωθεί στην οντότητα, αρχιτεκτονική ή και σε process. Ανάλογα με το που δηλώνεται έχει και την ανάλογη εμβέλεια/ορατότητα (visibilty).

Δήλωση

constant constant_name **:** type **:=** value;

constant Size: Positive := 8;

Η τιμή της ΔΕΝ αλλάζει ποτέ

https://www.hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/Constant.htm
https://peterfab.com/ref/vhdl/vhdl_renerta/source/vhd00022.htm

VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit

Πρόσθεση

4 bit

```
      1001
     +1101
     --------
Carry=1 0110
```

8 bit

1=Carry in στη 2η ALU

```
0010 1001
+0101 1101
---------------
1000 0110   Carry_out=1 από 1η ALU
```

Carry_in='0' 1η ALU

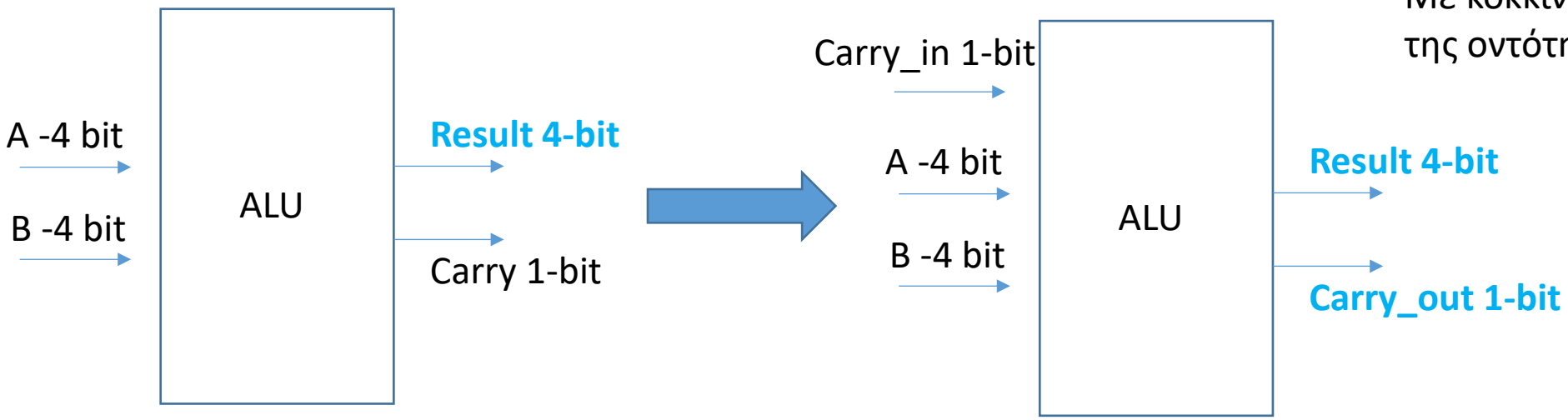
Carry_out=0 2η ALU

VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit

Για να μπορέσουμε να συνδυάσουμε 2 alu 4-bit θα πρέπει να μπορέσουμε να χειριστούμε το carry

Με κόκκινο οι έξοδοι της οντότητας

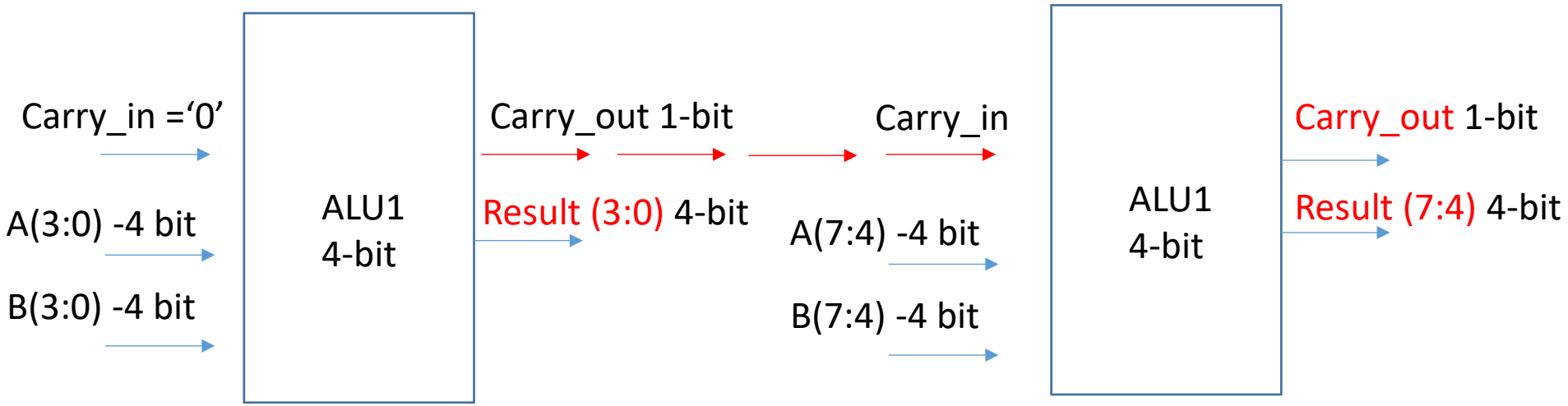


Πρώτα 4 bit (3 downto 0)

Επόμενα 4 bit (7 downto 4)

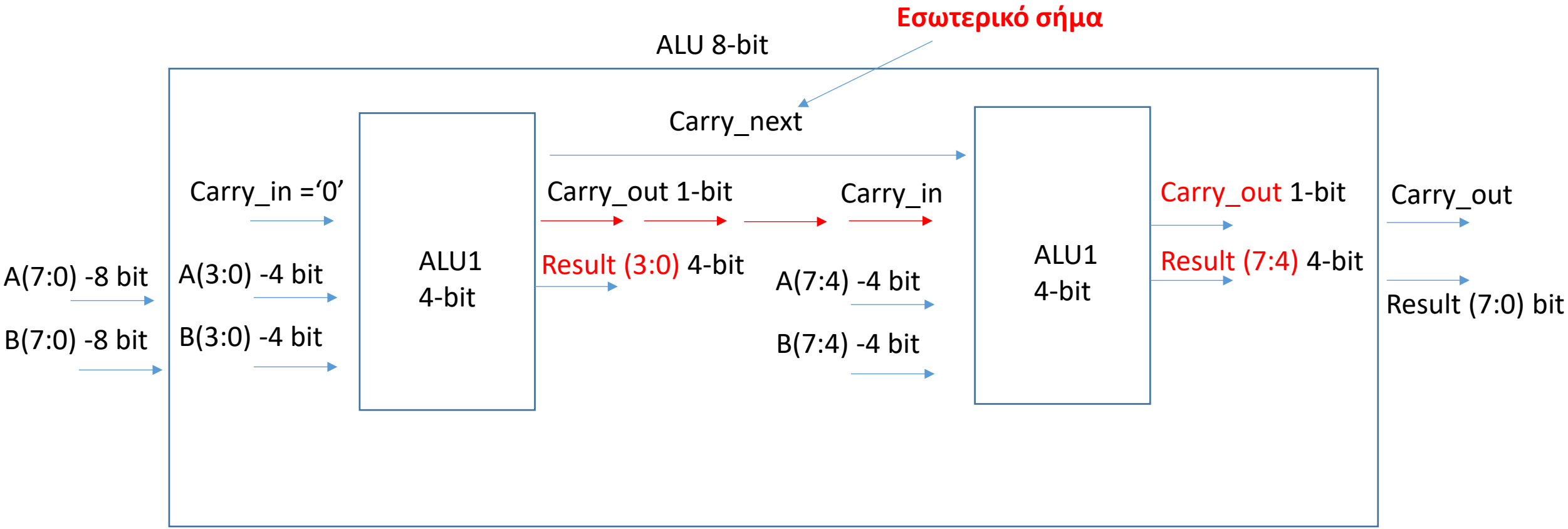
VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit



VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit



VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit

first_half: Adder_4bit port map(A_8bit(3 downto 0), B_8bit(3 downto 0), '0', Sum_8bit(3 downto 0), Carry_next);

second_half: Adder_4bit port map(A_8bit(7 downto 4), B_8bit(7 downto 4), Carry_next, Sum_8bit(7 downto 4), Cout_8bit);

VHDL – Κωδικοποίηση

- Πώς αναπαριστούμε πληροφορία με περισσότερες από δύο πιθανές τιμές;
  - Πολλαπλά δυαδικά σήματα (πολλαπλά bit)
- (a_1, a_0): (0, 0), (0, 1), (1, 0), (1, 1)
  - Αυτός είναι ένας *δυαδικός κώδικας*
  - Κάθε ζεύγος τιμών είναι μια *κωδική λέξη*

VHDL – Κωδικοποίηση

Code Length

- Ένας κώδικας των *n* bit έχει 2^n κωδικές λέξεις
- Για να αναπαραστήσουμε *N* πιθανές τιμές
  - χρειαζόμαστε τουλάχιστον $\lceil \log_2 N \rceil$ bit για τις λέξεις
  - περισσότερα bit μπορούν να είναι χρήσιμα σε κάποιες περιπτώσεις
- Παράδειγμα: κώδικας εκτυπωτή ψεκασμού
  - Black, cyan, magenta, yellow, light cyan, light magenta
  - έξι τιμές, $\lceil \log_2 6 \rceil = 3$
  - Black : (0, 0, 1), cyan : (0, 1, 0), magenta : (0, 1, 1),
    yellow : (1, 0, 0), light cyan : (1, 0, 1), light magenta : (1, 1, 0)

VHDL – Κωδικοποίηση

One Hot

- Κάθε κωδική λέξη έχει ακριβώς ένα bit με την τιμή ‘1'
- Φωτεινός σηματοδότης:
  - κόκκινο: (1,0,0), πορτοκαλί: (0,1,0), πράσινο: (0,0,1)
  - τρεις αγωγοί σημάτων: κόκκινο, πορτοκαλί, πράσινο
- Κάθε bit ενός κώδικα one-hot αντιστοιχεί σε μια κωδικοποιημένη τιμή
  - Μήκος κώδικα ίσο με πλήθος των προς κωδικοποίηση τιμών.
  - Όχι ελάχιστο μήκος

VHDL – Κωδικοποίηση

Παράδειγμα One Hot (1/2)

- Ελεγκτής φωτεινού σηματοδότη με κώδικα 1-hot
  - enable = 1: lights_out = lights_in
  - enable = 0: lights_out = (0, 0, 0)

```
library ieee; use       ieee.std_logic_1164.all;
entity   light_controller is
   port   ( lights_in  :      in   std_logic_vector(1 to 3);
            enable     :      in   std_logic;
            lights_out :      out  std_logic_vector(1 to 3) );
end entity light_controller;
```

VHDL – Κωδικοποίηση

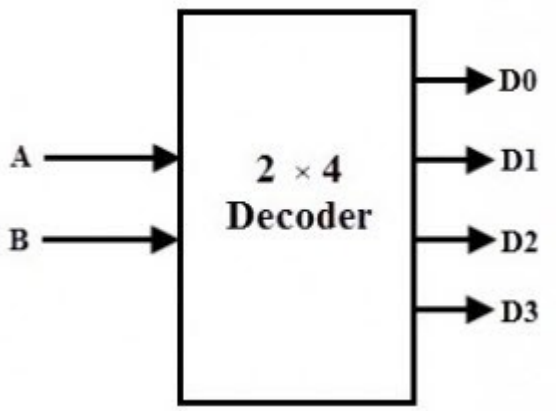
Παράδειγμα One Hot (2/2)

```
architecture and_enable of light_controller is
begin
   lights_out(1)  <=  lights_in(1)  and  enable;
   lights_out(2)  <=  lights_in(2)  and  enable;
   lights_out(3)  <=  lights_in(3)  and  enable;
end architecture and_enable;
```

```
architecture conditional_enable of light_controller is
begin
  lights_out  <= lights_in  when enable = '1' else
                 "000";
end architecture conditional_enable;
```

or just **when enable**

VHDL – Αποκωδικοποίηση



- Ο αποκωδικοποιητής εξάγει σήματα ελέγχου από ένα δυαδικά κωδικοποιημένο σήμα
  - Ένα σήμα ανά κωδική λέξη
  - Το σήμα ελέγχου είναι 1 όταν η είσοδος έχει την αντίστοιχη κωδική λέξη, διαφορετικά 0

VHDL – Αποκωδικοποίηση

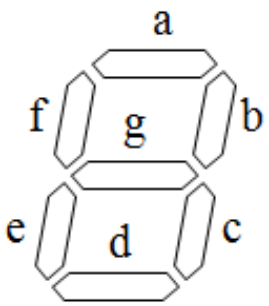
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0);
        output : out STD_LOGIC_VECTOR(3 downto 0));
end decoder_2to4;

architecture Behavioral of decoder_2to4 is
begin
process(input)    is
begin
        case input is
        when "00" => output <= "0001";
        when "01" => output <= "0010";
        when "10" => output <= "0100";
        when "11" => output <= "1000";
        when others => output <= "0000"; -- Default case
        end case;
end process;
end Behavioral;
```

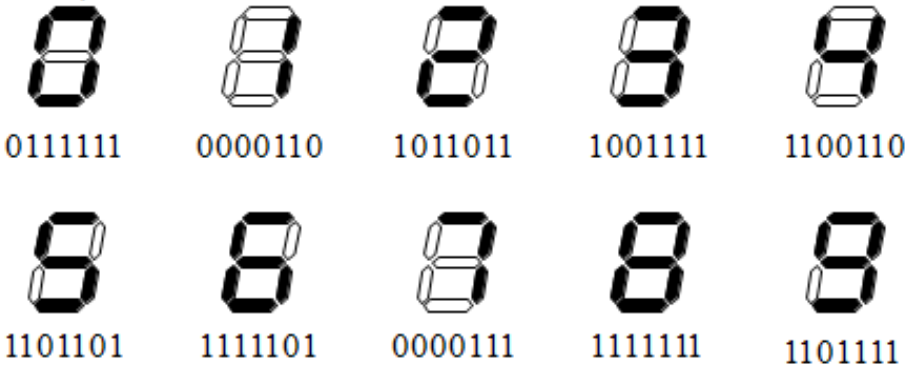
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0); -- 2-bit input
        output : out STD_LOGIC_VECTOR(3 downto 0) -- 4-bit
output);
end decoder_2to4;

architecture Dataflow of decoder_2to4 is
begin
    output <=  "0001" when input = "00" else
               "0010" when input = "01" else
               "0100" when input = "10" else
               "1000" when input = "11" else
               "0000"; -- Default case
end Dataflow;
```

VHDL – Αποκωδικοποίηση

Παράδειγμα BCD to 7segment (1/2)

- Αποκωδικοποιεί τον κώδικα BCD (binary coded decimal) για να οδηγήσει μια οθόνη (LED ή LCD) 7 τμημάτων
  - Τμήματα: (g, f, e, d, c, b, a)





- Κώδικας BCD: Δυαδικά κωδικοποιημένοι δεκαδικοί
  - Κώδικας 4 bit για τα δεκαδικά ψηφία

0: 0000	1: 0001	2: 0010	3: 0011	4: 0100
5: 0101	6: 0110	7: 0111	8: 1000	9: 1001

VHDL – Αποκωδικοποίηση

Παράδειγμα BCD to 7segment (2/2)

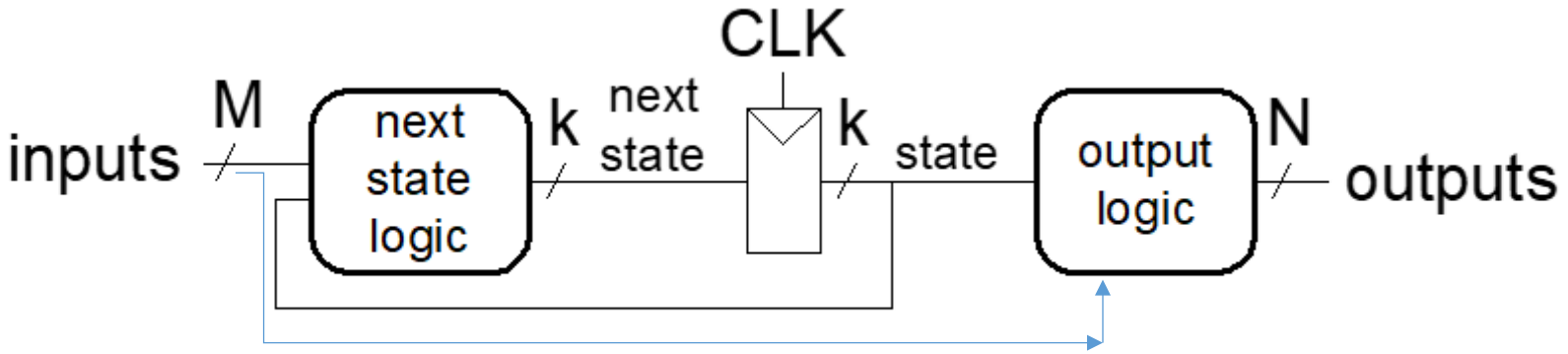
```
library ieee; use ieee.std_logic_1164.all;
entity seven_seg_decoder is
  port ( bcd   : in std_logic_vector (3 downto 0);
         blank : in std_logic;
         seg   : out std_logic_vector (7 downto 1) );
end entity seven_seg_decoder;
```

```
architecture behavior of seven_seg_decoder is
  signal seg_tmp : std_logic_vector (7 downto 1);
begin
  with bcd select
    seg_tmp <= "0111111" when "0000", -- 0
               "0000110" when "0001", -- 1
               "1011011" when "0010", -- 2
               "1001111" when "0011", -- 3
               ...
               "1101111" when "1001", -- 9
               "1000000" when others; -- "-"
seg <= "0000000" when blank = '1' else seg_tmp;
end architecture behavior;
```

Selected signal assignment

VHDL – Ακολουθιακά Κυκλώματα

- Οι έξοδοι Q_t (τιμή εξόδου τη χρονική στιγμή t) των ακολουθιακών κυκλωμάτων εξαρτώνται όχι μόνο από τις τρέχουσες τιμές των εισόδων, αλλά και από τις προηγούμενες τιμές των εξόδων $Q_{t-1.}$ Στο κύκλωμα αυτό εμφανίζετε ως ανάδραση
- Έχουν μνήμη (κατάσταση-state). Για N αποθηκευμένες καταστάσεις το κύκλωμα χρειάζεται από $\log_2 N$ έως και N bit.
- Οι έξοδοι είναι συνάρτηση των εισόδων και της αποθηκευμένης κατάστασης.
- Συνήθως υπάρχει ρολόϊ CLK



3 Block
- Λογική επόμενης κατάσταης (συνδυαστικό)
- Καταχωρητής κατάστασης
- Λογική εξόδου (συνδυαστικό)

VHDL – Ακολουθιακά Κυκλώματα

S-R Latch (1η έκδοση)

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

entity sr_latch is
    Port (
        S : in STD_LOGIC;    -- Set input
        R : in STD_LOGIC;    -- Reset input
        Q : out STD_LOGIC;   -- Q output
        Qn : out STD_LOGIC   -- Q' (complement of Q) output
    );
end sr_latch;
```

```
architecture Behavioral of sr_latch is
begin
    process(S, R)
    begin
        if S = '1' and R = '0' then
            Q <= '1';
            Qn <= '0';
        elsif S = '0' and R = '1' then
            Q <= '0';
            Qn <= '1';
        elsif S = '0' and R = '0' then  -- No action, keep values
        else            -- Invalid state (S = '1' and R = '1')
            Q <= 'X';  -- Indeterminate state
            Qn <= 'X';
        end if;
    end process;
end Behavioral;
```

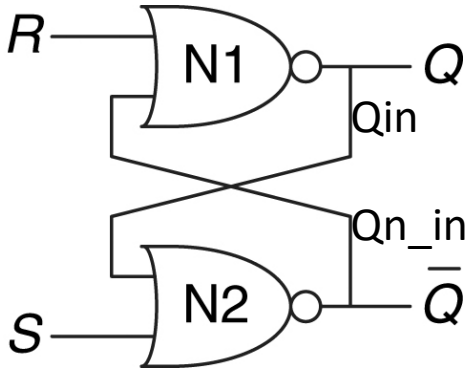
Πίνακας Αλήθειας

S	R	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q_{prev}}$
0	1	0	1
1	0	1	0
1	1	0	0

VHDL – Ακολουθιακά Κυκλώματα

S-R Latch (2η έκδοση)

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

entity sr_latch is
    Port (
        S : in STD_LOGIC;    -- Set input
        R : in STD_LOGIC;    -- Reset input
        Q : out STD_LOGIC;   -- Q output
        Qn : out STD_LOGIC   -- Q' (complement of Q) output
    );
end sr_latch;
```




```
architecture Dataflow of sr_latch is

    signal Q_in   : STD_LOGIC := '0';  -- Internal feedback for Q
    signal Qn_in : STD_LOGIC := '1'; -- Internal feedback for Q'

begin
    -- Concurrent feedback logic
    Q_in   <= not (R or Q_in);   -- Q feedback
    Qn_in <= not (S or Qn_in); -- Q' feedback

    -- Map internal signals to outputs
    Q <= Q_in;
    Qn <= Qn_in;

end Dataflow;
```

VHDL – Ακολουθιακά Κυκλώματα

D Latch (1η έκδοση)

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

entity d_latch is
    Port (
        D : in STD_LOGIC;    -- Data input
        CLK : in STD_LOGIC;  -- Clock input
        Q : out STD_LOGIC;   -- Output
        Qn : out STD_LOGIC   -- Complementary output
    );
end d_latch;
```

CLK	D	D'	S	R	Q	$\overline{Q}$
0	X	X'	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

```
architecture Behavioral of d_latch is

signal Q_in : STD_LOGIC := '0';  -- Internal signal for Q

begin

    -- Latch behavior with concurrent assignments
    Q_in <= D when CLK = '1' else
            Q_in;

    Q <= Q_in;              -- Assign internal signal to Q
    Qn <= not Q_in;         -- Complementary output

end Behavioral;
```

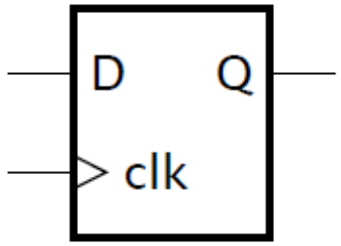
VHDL – Ακολουθιακά Κυκλώματα

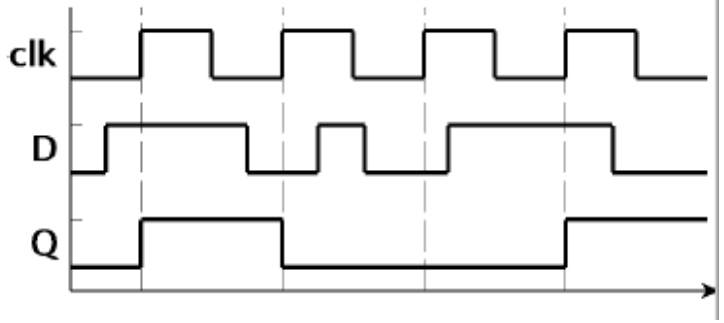
D Flip Flop

- Στοιχείο αποθήκευσης του 1 bit
- Άλλοι τύποι flip-flop
  - JK, T (toggle)

```
entity dff is
   port ( clk,d : in std_logic;
          q     : out std_logic);
end entity;
architecture beh of dff is
begin
   process (clk)
   begin
    if clk'event and clk = '1' then
       q <= d;
    end if;
   end process;
end beh;
```

Μόνο το clk στο sensitivity list





VHDL – Ακολουθιακά Κυκλώματα

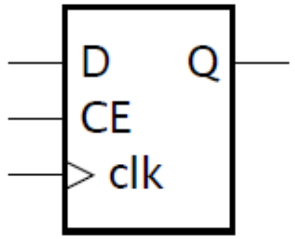
D Flip Flop with Enable

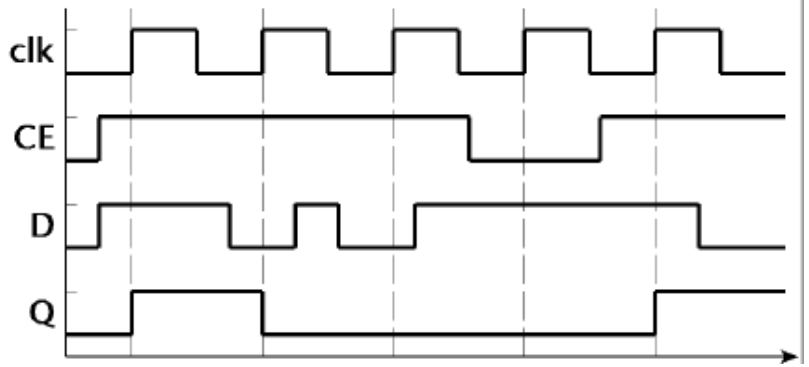
Η αποθήκευση της τιμής D εξαρτάται από το σήμα έγκρισης CE (en)

- Αποθηκεύει μόνο όταν CE = 1 σε μια ανοδική ακμή ρολογιού
- Το CE είναι μια σύγχρονη είσοδος ελέγχου

```
entity dff_en is
   port ( clk   : in std_logic;
          ce    : in std_logic;
          d     : in std_logic;
          q     : out std_logic);
end dff_en ;
architecture beh of dff_en is
begin
   process (clk)
   begin
      if clk'event and clk='1' then
         if ce='1' then
            q <= d;
         end if;
      end if;
   end process;
end beh;
```

Σύγχρονο: Πρώτα έλεγχος για το clock



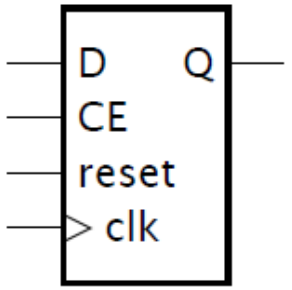


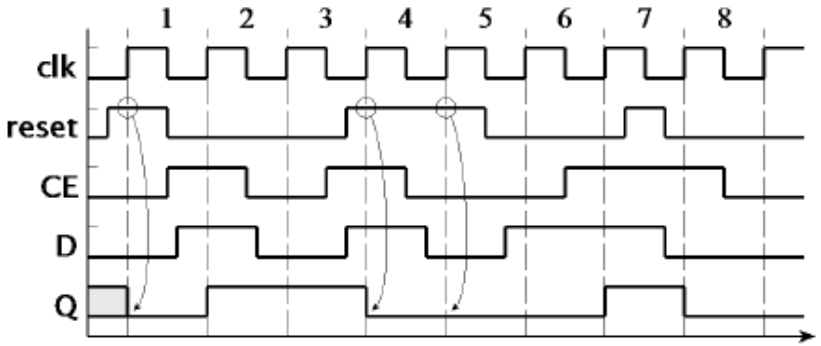
VHDL – Ακολουθιακά Κυκλώματα

D Flip Flop with Reset (σύγχρονο) και Enable

- Η είσοδος μηδενισμού (reset) θέτει την αποθηκευμένη τιμή στο 0
  - η είσοδος reset πρέπει να είναι σταθερή γύρω από την ανοδική ακμή του clk

```
entity dff_en_reset is
   port ( clk      : in std_logic;
          ce,reset : in std_logic;
          d        : in std_logic;
          q        : out std_logic);
end dff_en ;
architecture beh of dff_en_reset is
begin
   process (clk)
   begin
      if clk'event and clk='1' then
         if reset = '1' then
           q <= '0';
         elsif ce = '1' then
           q <= d;
         end if;
      end if;
   end process;
end beh;
```



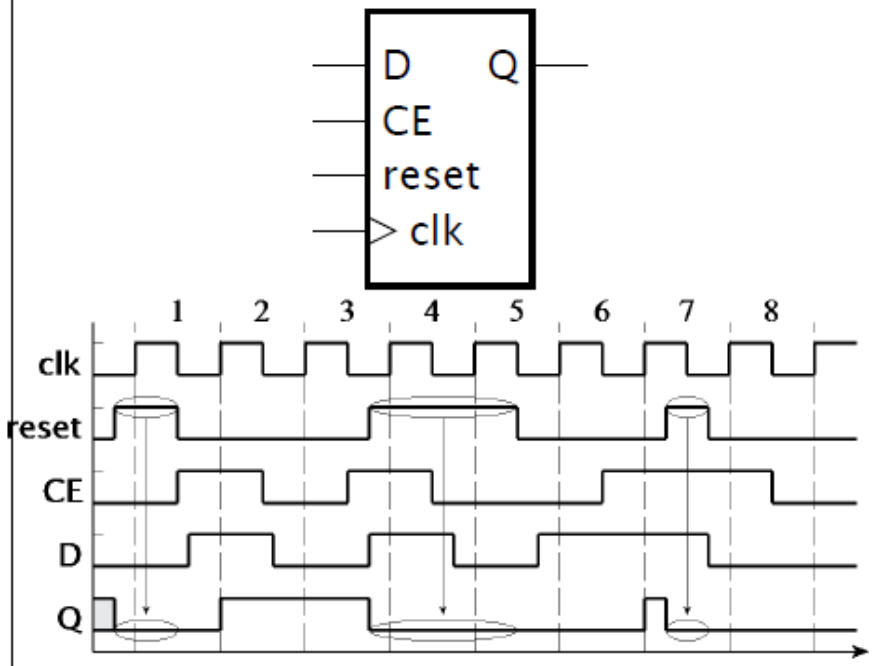


VHDL – Ακολουθιακά Κυκλώματα

D Flip Flop with Reset (ασύγχρονο)

- Η είσοδος μηδενισμού (reset) θέτει την αποθηκευμένη τιμή στο 0
  - το reset μπορεί να γίνει 1 οποιαδήποτε στιγμή, και το αποτέλεσμα είναι άμεσο
  - το συμπεριλαμβάνουμε στη λίστα ευαισθησίας (η διεργασία ανταποκρίνεται άμεσα σε αλλαγή)

```
entity dff_en_areset is
   port ( clk      : in std_logic;
          ce,reset : in std_logic;
          d        : in std_logic;
          q        : out std_logic);
end dff_en ;
architecture beh of dff_en_areset is
begin
   process (clk, reset)
   begin
     if reset = '1' then
        q <= '0';
     elsif clk'event and clk='1' then
       if ce = '1' then
         q <= d;
       end if;
     end if;
   end process;
end beh;
```

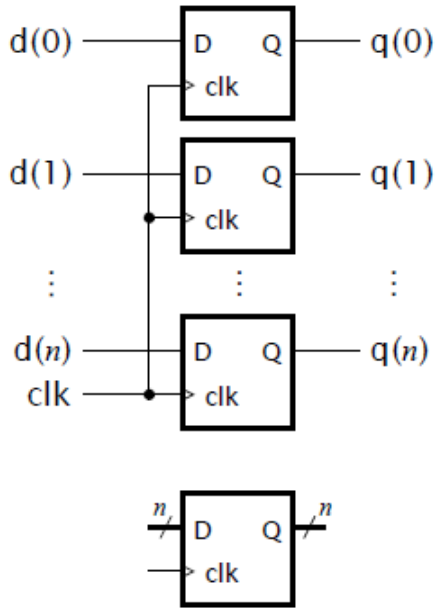


VHDL – Ακολουθιακά Κυκλώματα

Καταχωρητές (D Flip Flop με ασύγχρονο reset)

Αποθηκεύουν μια τιμή πολλαπλών bit

- Ένα flip-flop D ανά bit
- Απαιτεί αλλαγή στο array data type
  - std_logic_vector




```
entity reg_reset is
   port (clk, reset: in std_logic;
      d: in std_logic_vector(7 downto 0);
      q: out std_logic_vector(7 downto 0)
);
end reg_reset;

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

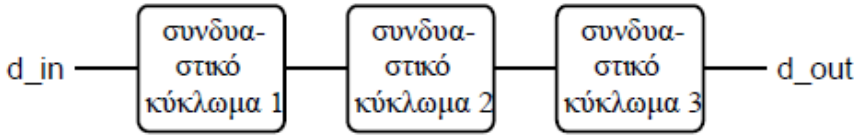
Συντομογραφία του όλα στο 0

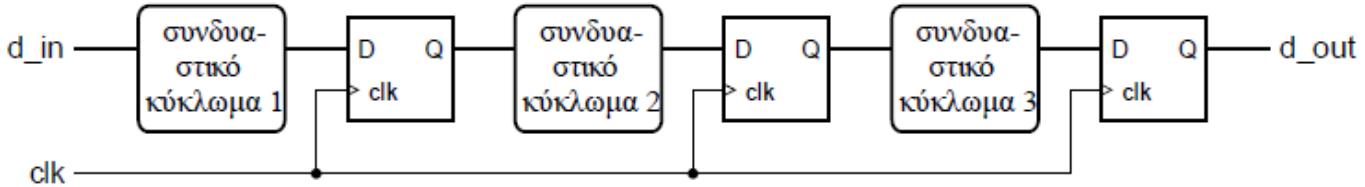
VHDL – Ακολουθιακά Κυκλώματα

Pipelines (διοχέτευση)

Συνολική καθυστέρηση = $Delay_1 + Delay_2 + Delay_3$

Διάστημα μεταξύ των εξόδων > Συνολική καθυστέρηση





Περίοδος ρολογιού = $\max(Delay_1, Delay_2, Delay_3)$

Συνολική καθυστέρηση = 3 × περίοδος ρολογιού

Διάστημα μεταξύ των εξόδων = 1 περίοδος ρολογιού

VHDL – Ακολουθιακά Κυκλώματα

Συσσωρευτής (accumulator 1/2)

Αθροίστε μια ακολουθία.
Ένας νέος αριθμός (**data_in**) φτάνει σε κάθε ανοδική ακμή του clock.
Το άθροισμα (**data_out**) μηδενίζει με σύγχρονο reset

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

entity accumulator is

port (
        clk :in std_logic;
        reset : in std_logic;
        data_in : in std_logic_vector(15 downto 0);
        data_out : out std_logic_vector(19 downto 0));

end entity accumulator;
```

VHDL – Ακολουθιακά Κυκλώματα

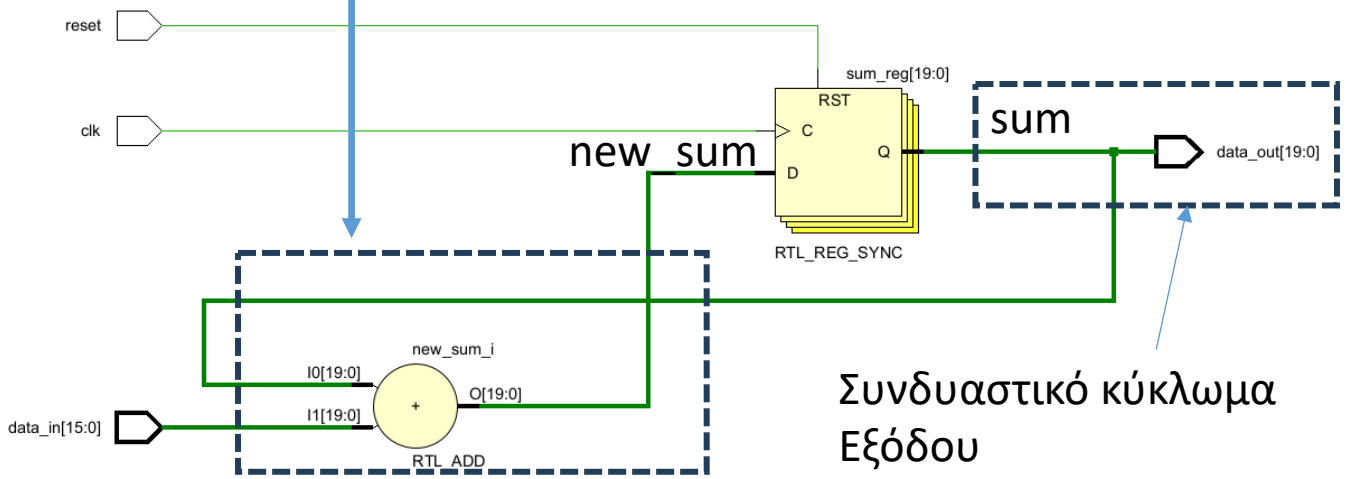
Συσσωρευτής (accumulator 2/2)

```
architecture rtl of accumulator is

signal sum, new_sum : signed(19 downto 0);
begin
        new_sum <= sum+resize(data_in, sum'length);
        reg:  process (clk) is
        begin
                if rising_edge(clk) then
                        if reset = '1' then
                                sum <= (others => '0');
                        else
                                sum <= new_sum;
                        end if;
                end if;
        end process reg;

        data_out <= std_logic_vector(sum);
end architecture rtl;
```

Συνδυαστικό κύκλωμα επόμενης κατάστασης

Εναλλακτικός τρόπος για την ανοδική ακμή του clk. Το rising_edge είναι συνάρτηση για οποιοδήποτε σήμα.

Συνδυαστικό κύκλωμα Εξόδου



Συνδυαστικό κύκλωμα Εξόδου

VHDL – Ακολουθιακά Κυκλώματα

Μετρητές (counters)

- Αποθηκεύουν την τιμή ενός απρόσημου ακεραίου
  - αυξάνουν ή μειώνουν την τιμή
- Χρησιμοποιούνται για να μετράνε πόσες φορές:
  - έχουν συμβεί κάποια γεγονότα
  - έχει επαναληφθεί ένα βήμα επεξεργασίας
- Χρησιμοποιούνται ως χρονομετρητές (timers)
  - μετράνε πόσα χρονικά διαστήματα έχουν περάσει καθώς αυξάνονται περιοδικά

VHDL – Ακολουθιακά Κυκλώματα

Μετρητές ασύγχρονοι

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity counter4 is
       port (clk, reset : in std_logic;
       count            : out std_logic(3 downto 0);
end entity;

architecture beh of counter4 is
    signal counter : unsigned(3 downto 0);
begin
count <= std_logic_vector(counter);
   process (clk, reset)
   begin
     if reset = '1' then
        counter <= (others => '0');
      elsif clk'event and clk = '1' then
        if counter = 15 then
          counter <= (others => '0');
        else
          counter <= counter + 1;
        end if;
     end if;
   end process;
end architecture;
```

VHDL – Ακολουθιακά Κυκλώματα

Δεκαδικός Μετρητής

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity decade_counter is
  port ( clk : in std_logic;  q : out std_logic_vector(3 downto 0) );
end entity decade_counter;

architecture rtl of decade_counter is
  signal count_value : unsigned(3 downto 0);
begin
  count : process (clk) is
  begin
    if rising_edge(clk) then
      count_value <= (count_value + 1) mod 10;
    end if;
  end process count;
  q <= std_logic_vector(count_value);
end architecture rtl;
```

Περίληψη

- For
- Generic
- Κωδικοποίηση/Αποκωδικοποίηση
- Ακολουθιακά κυκλώματα
- Accumulator, counter, shifter,
- Conditional Statements
- Διαβάζετε τις παραγράφους 2.2, 2.3.1, 2.4, 4.1, 4.2, 4.3 (θεωρία και VHDL) από Ashenden και 2.8.2, 4.4, 4.7.2, 4.8, 4.9 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.