dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

# Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων

**VHDL**

**-**

**Εντολές επανάληψης, Procedure, Functions, Generic**
**Κωδικοποίηση – 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;

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 – Επαναλήψεις

## Εντολή FOR Generate

Εκτός Process

```
entity INV4 is
  port (
    X: in STD_LOGIC_VECTOR (0 to 3);
    Y: out STD_LOGIC_VECTOR (0 to 3));
end INV4;
architecture INV4_STR1 of INV4 is
  component INV
    port (O : out STD_LOGIC; I : in STD_LOGIC);
  end component;
begin
  U0: INV port map (Y(0), X(0));
  U1: INV port map (Y(1), X(1));
  U2: INV port map (Y(2), X(2));
  U3: INV port map (Y(3), X(3));
end INV4_STR1;
```

```
G1: for I in 0 to 3 generate
    U1: INV port map (Y(I), X(I));
  end generate G1;
```

optional_label: for variable in range generate
**concurrent** statements;
end loop;

# VHDL – Επαναλήψεις

## Εντολή While

```
while condition loop
        sequential statements;
end loop;
```

```
process (A)
variable I : integer range 0 to 4;
begin
Z <= "0000"; I := 0;
while (I <= 3) loop
if (A = I) then
        Z(I) <= '1';
end if;
I := I + 1;
end loop;
end process;
```

# VHDL – Procedure

**procedure** procedure_name(input and output parameters) **is**

declarations

**begin**

sequential statement1;
sequential statement2;
.....

**end procedure**;

https://www.ics.uci.edu/~jmoorkan/vhdlref/procedur.html

# VHDL – Procedure

```
Ctr_tb<='0';
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 j;--end loop i;

Ctr_tb<='1';
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;
```

```
procedure sim_test is
begin

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;

end procedure;

Ctr_tb<='0';sim_test;
Ctr_tb<='1';sim_test;
```

# VHDL – Procedure

```
procedure sim_test (min, max: in integer; step: in integer) is
variable a2: integer;
begin
for i in min  to max loop
        a<=std_logic_vector(to_signed(i*step,a'length));
        for j in min to max loop
                b<=std_logic_vector(to_signed(j*step,a'length));wait for 10 ns;
        end loop;
end loop;

end procedure;

Ctr<='0';sim_test(0,2,5);
Ctr<='1';sim_test(0,2,5);
```

Όταν δεν αναφέρεται τύπος εννοείται variable. Σε αυτή την περίπτωση τα ορίσματα πρέπει να είναι σταθερές ή variables

a, b έχουν οριστεί εκτός procedure και
εντός του process.

**Όταν ο τύπος στα ορίσματα είναι signal τότε και στα ορίσματα στην κλήση της procedure θα πρέπει να υπάρχουν signal**

# VHDL – Function

**Function** function_name(input parameters) return return_type **is**

declarations

**begin**

sequential statement1;
sequential statement2;
.....

**end function_name**;

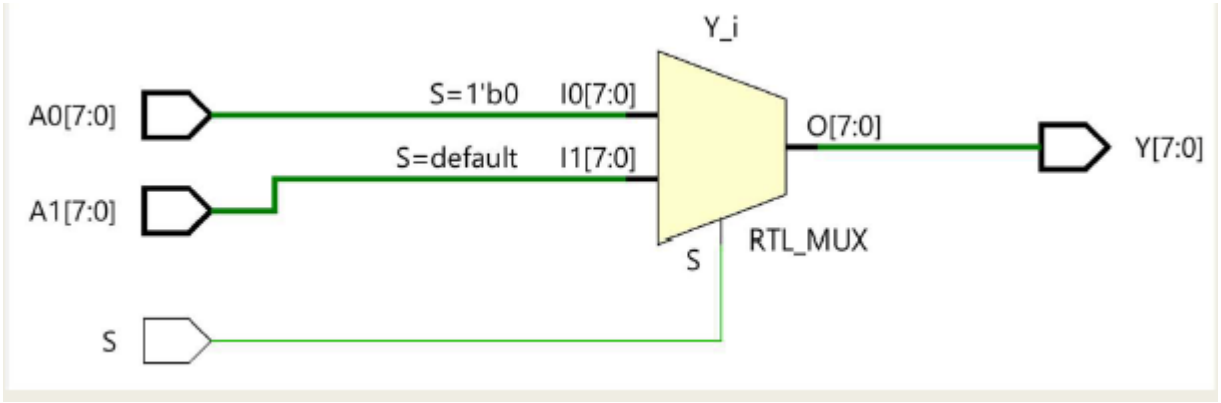
https://www.ics.uci.edu/~jmoorkan/vhdlref/function.html

# VHDL – Function

```
function BOOL_TO_SL(X : boolean) return std_logic is

begin
  if X then
    return '1';
  else
    return '0';
  end if;

end function BOOL_TO_SL;

std_logic_signal<=BOOL_TO_SL(X);
```

# 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 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)

# VHDL – Structural architecture

## ALU 8bit από 2 ALU των 4bit

Πρόσθεση

4 bit

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

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

Διπλασιασμός

4 bit

```
 1001 & '0'
 --------
  0010   Carry='1'
```

8 bit

1=Carry in στη 2η ALU

```
0010 1001 & '0'
---------------
0101 0010     Carry_out='1' από 1η ALU
```

Carry_in='0' 1η ALU

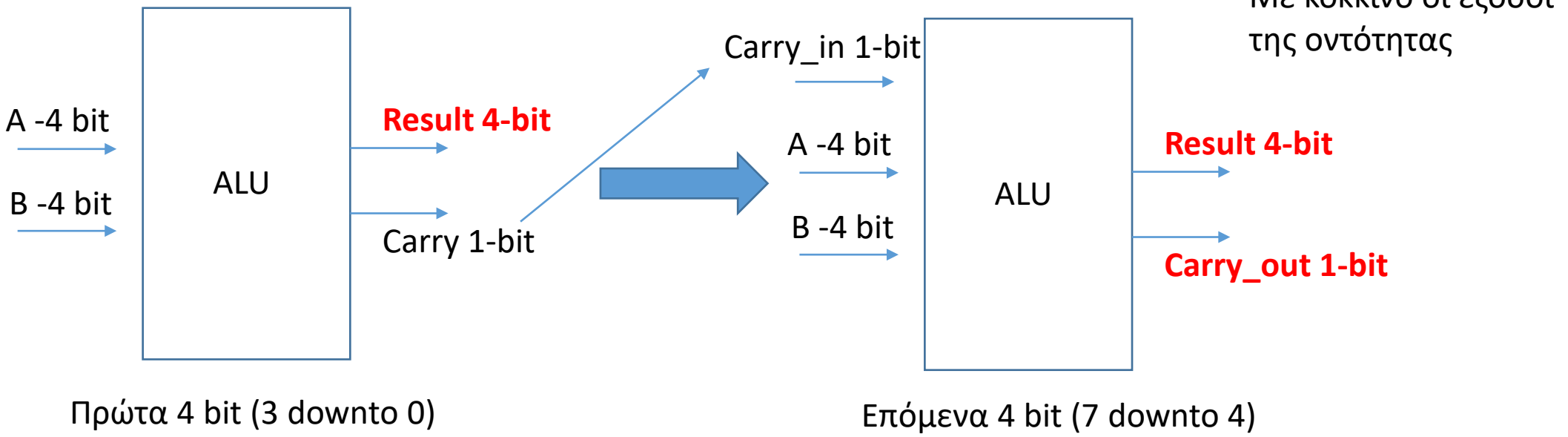
Carry_out='0' 2η ALU

# VHDL – Structural architecture

## ALU 8bit από 2 ALU 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

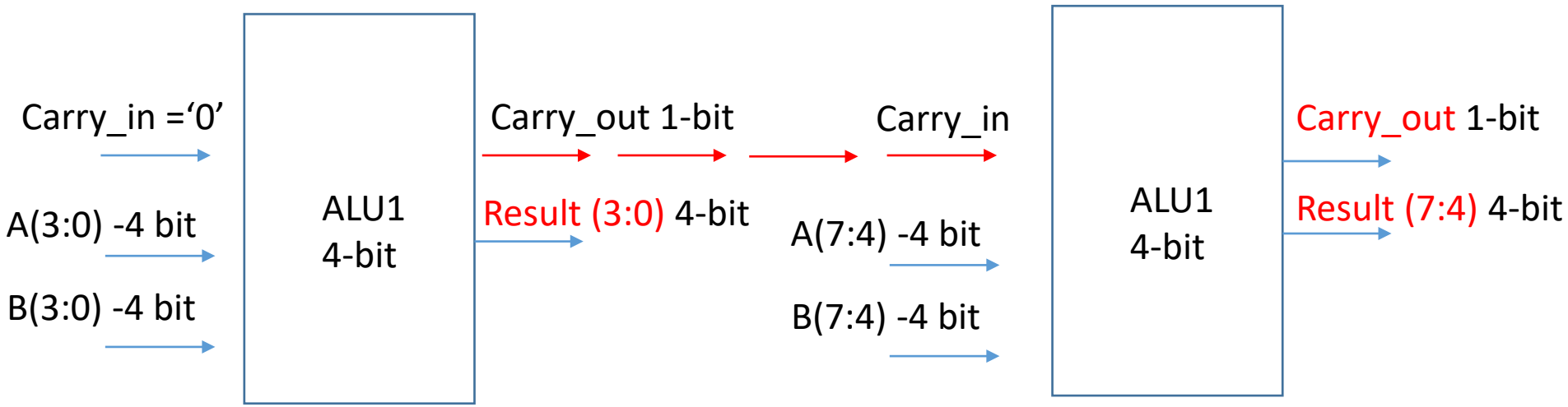



Πρώτα 4 bit (3 downto 0)

Επόμενα 4 bit (7 downto 4)

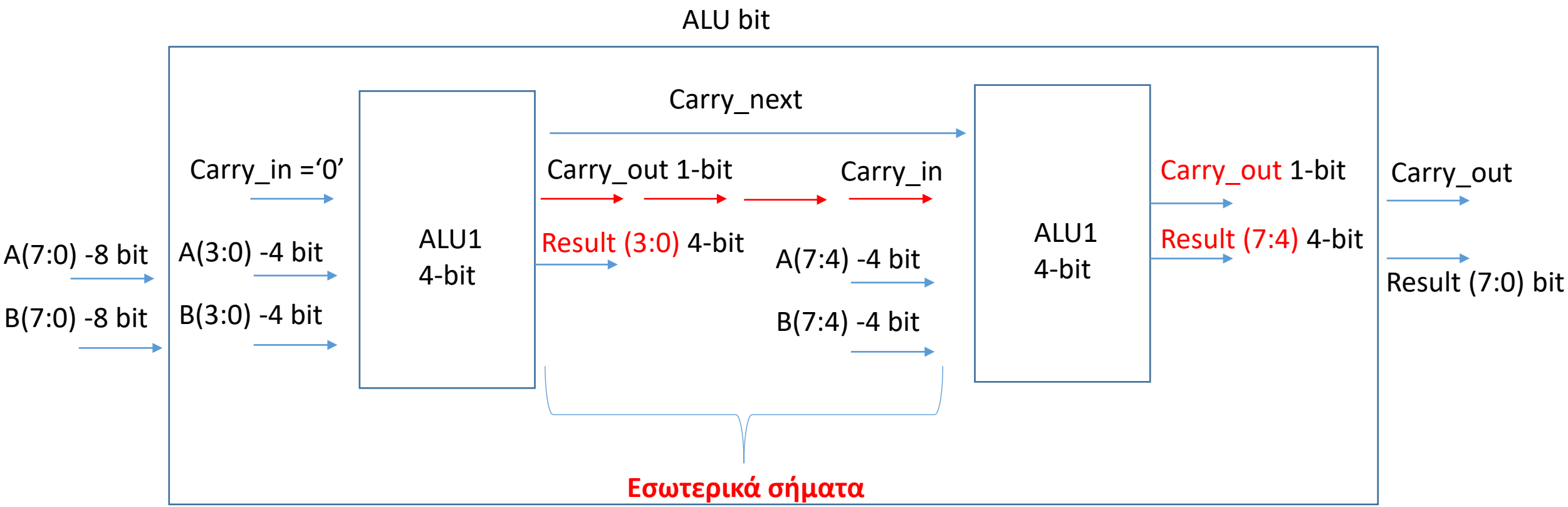
# VHDL – Structural architecture

## ALU 8bit από 2 ALU των 4bit



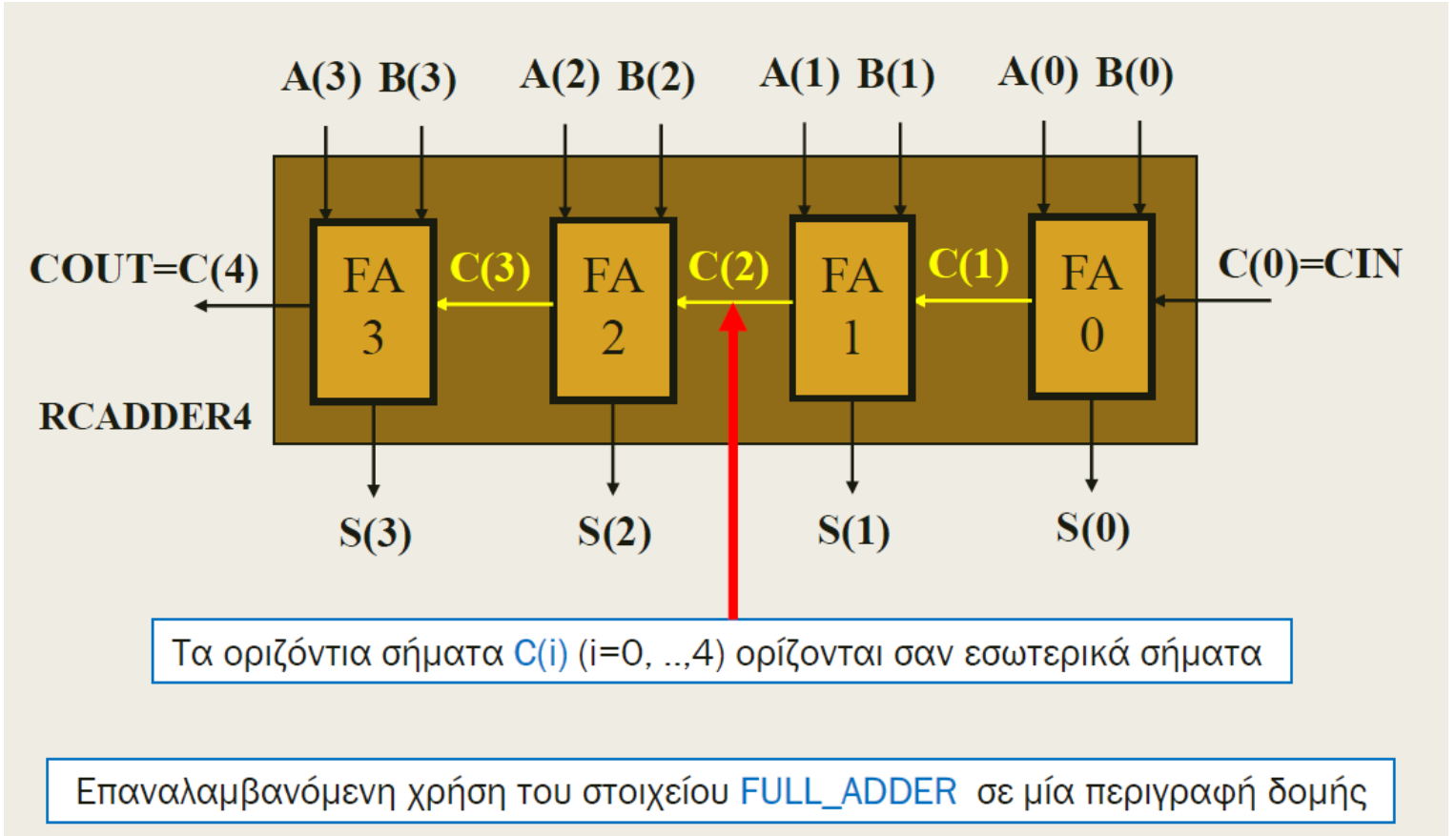
# VHDL – Structural architecture

## Αθροιστής 8bit από 2 αθροιστές των 4bit



# VHDL – Structural architecture

## Αθροιστής Ριπής Κρατουμένου των 4 bit (for generate)



# VHDL – Structural architecture

## Αθροιστής Ριπής Κρατουμένου των 4 bit (for generate)

1st Entity - Component

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

entity FULL_ADDER is
    port (  A, B, CIN: in STD_LOGIC;
            SUM, CARRY: out STD_LOGIC);
end FULL_ADDER;

architecture FA_DATAFLOW2 of FULL_ADDER is
begin
    SUM <= A xor B xor CIN;
    CARRY <= (A and B) or (A and CIN) or (B and CIN);
end FA_DATAFLOW2;
```

2nd Entity

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RCADDER4 is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0);
           Cout : out  STD_LOGIC);
end RCADDER4;

architecture Behavioral of RCADDER4 is

  signal C: STD_LOGIC_VECTOR (4 downto 0); -- οριζόντια σήματα
  component FULL_ADDER
    port (
    A, B, CIN: in STD_LOGIC;
    SUM, CARRY: out STD_LOGIC);
  end component;
begin
    C(0) <= CIN;     -- αρχικοποίηση
    G1: for I in 0 to 3 generate
        U1: FULL_ADDER port map (A(I),B(I),C(I),S(I),C(I+1));
        end generate G1;
    COUT <= C(4);    -- ανάθεση σε σήμα εξόδου

end Behavioral;
```

Στο ίδιο αρχείο δημιουργούνται 2 οντότητες
Οι επικεφαλίδες επαναλαμβάνονται.

# 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 (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 (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 – Κωδικοποίηση

## Παράδειγμα (1/3) – Κωδικοποιητής προτεραιότητας

- Εάν περισσότερες από μία είσοδοι μπορεί να είναι 1
  - Κωδικοποιούμε την είσοδο που είναι 1 με την υψηλότερη προτεραιότητα

| zone | | | | | | | | intruder_zone | | | valid |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (2) | (1) | (0) | |
| 1 | – | – | – | – | – | – | – | 0 | 0 | 0 | 1 |
| 0 | 1 | – | – | – | – | – | – | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | – | – | – | – | – | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | – | – | – | – | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | – | – | – | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | – | – | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | – | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | – | – | – | 0 |

Συναγερμός: 8 είσοδοι ανίχνευσης (8 αισθητήρες, ένας ανά είσοδο). Υπάρχουν προτεραιότητες στις εισόδους (π.χ. πόρτα οικίας VS εξώπορτα κήπου).

- 8 bit εισόδου: ένας αισθητήρας για κάθε ζώνη
- 3 bit εξόδου για την κωδικοποίηση των ζωνών

# VHDL – Κωδικοποίηση

## Παράδειγμα (2/3) – Κωδικοποιητής προτεραιότητας (1η υλοποίηση)

- Παράδειγμα: Αντικλεπτικό σύστημα συναγερμού - κωδικοποιεί ποια ζώνη έχει ενεργοποιηθεί
  - 8 bit εισόδου: ένας αισθητήρας για κάθε ζώνη
  - 3 bit εξόδου για την κωδικοποίηση των ζωνών

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

entity alarm is
  port ( zone           : in std_logic_vector (1 to 8);
         intruder_zone  : out std_logic_vector(2 downto 0);
         valid   : out std_logic );
end entity alarm;

architecture eqn of alarm is
begin
  intruder_zone(2) <= zone(5) or zone(6) or zone(7) or zone(8);
  intruder_zone(1) <= zone(3) or zone(4) or zone(7) or zone(8);
  intruder_zone(0) <= zone(2) or zone(4) or zone(6) or zone(8);
  valid <= zone(1) or zone(2) or zone(3) or zone(4) or zone(5)
           or zone(6) or zone(7) or zone(8);
end architecture eqn;
```

| Ζώνη | Κωδική λέξη |
|---|---|
| Zone 1 | 0, 0, 0 |
| Zone 2 | 0, 0, 1 |
| Zone 3 | 0, 1, 0 |
| Zone 4 | 0, 1, 1 |
| Zone 5 | 1, 0, 0 |
| Zone 6 | 1, 0, 1 |
| Zone 7 | 1, 1, 0 |
| Zone 8 | 1, 1, 1 |

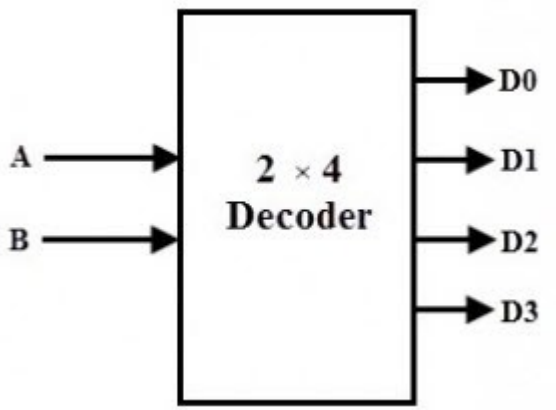
# VHDL – Κωδικοποίηση

## Παράδειγμα (3/3) – Κωδικοποιητής προτεραιότητας (2η υλοποίηση)

Conditional signal assignment

```
architecture priority_1 of alarm is
begin
  intruder_zone <= "000" when zone(1) = '1' else
                   "001" when zone(2) = '1' else
                   "010" when zone(3) = '1' else
                   "011" when zone(4) = '1' else
                   "100" when zone(5) = '1' else
                   "101" when zone(6) = '1' else
                   "110" when zone(7) = '1' else
                   "111" when zone(8) = '1' else
                   "000";
  valid <= zone(1) or zone(2) or zone(3) or zone(4)
           or zone(5) or zone(6) or zone(7) or zone(8);
end architecture priority_1;
```

# 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 decoder4 is
port (a: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(3 downto 0));
end entity decoder4 ;
architecture sel_arch of decoder4 is
begin
   with a select y <=
    "0001" when "00",
    "0010" when "01",
    "0100" when "10",
    "1000" when "11",
    "0000" when others;
end sel_arch ;
```

For simulation, you don't want to enumerate all 77 (81-4) meta-values of std_logic..
For synthesis, it is ignored...

**Selected signal assignment**

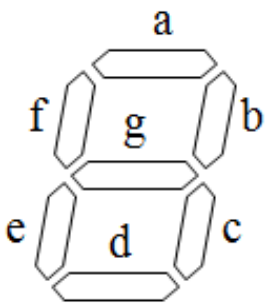
```
library ieee; use ieee.std_logic_1164.all;
entity decoder4 is
port (a: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(3 downto 0));
end entity decoder4 ;
architecture case_arch of decoder4 is
begin
    process (a)
    begin
      case a is
         when "00" =>   y<= "0001";
         when "01" =>   y<= "0010";
         when "10" =>   y<= "0100";
         when "11" =>   y<= "1000";
         when others => y<= "0000";
      end case;
    end process;
end case_arch;
```

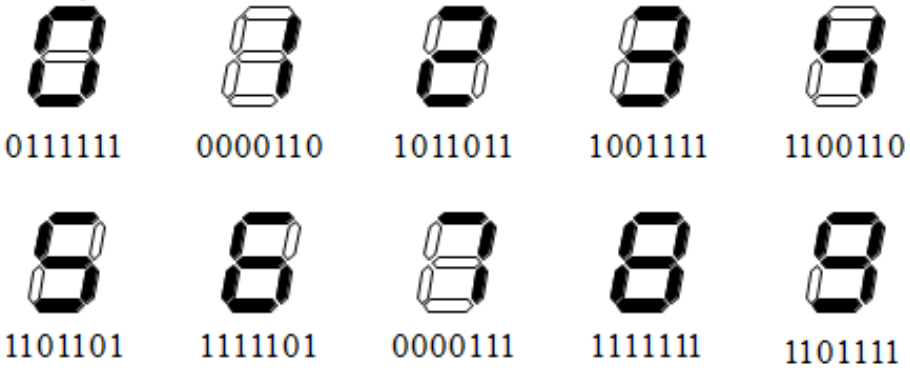
**Combinational Process with Case Statement**

# VHDL – Αποκωδικοποίηση

## Παράδειγμα (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 (2/2)

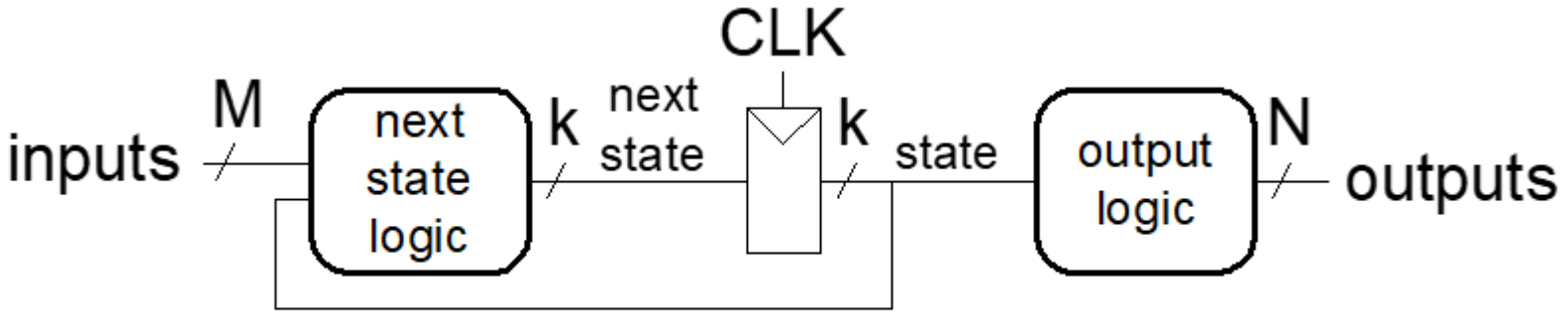
```
library ieee; use ieee.std_logic_1164.all;
entity seven_seg_decoder is
  port ( bcd   : in std_logic_vector (3 downto 0);
         blank : in std_logic;
         seg   : out std_logic_vector (7 downto 1) );
end entity seven_seg_decoder;
```

```
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

- Ακολουθιακά κυκλώματα
  - Οι έξοδοι εξαρτώνται από τις τρέχουσες και από τις προηγούμενες εισόδους
  - Αποθήκευση *κατάστασης (state)*: μια αφαίρεση του ιστορικού των εισόδων
- Συνήθως, ελέγχεται από σήμα ρολογιού (clock)




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

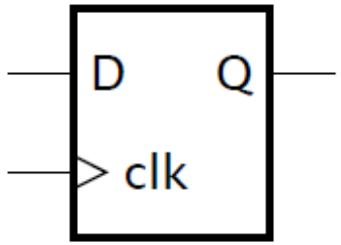
# VHDL – Ακολουθιακά Κυκλώματα

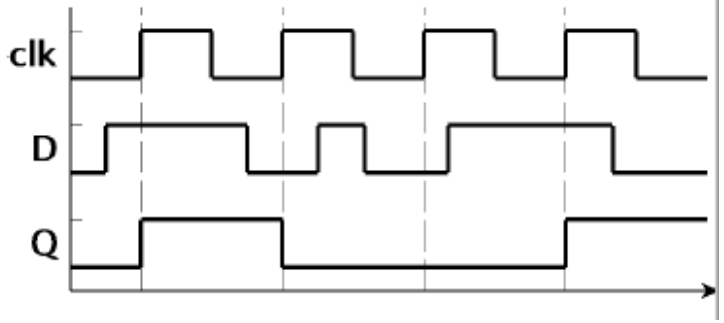
## D Flip Flop

- 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



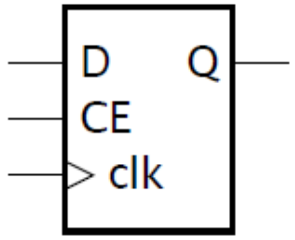



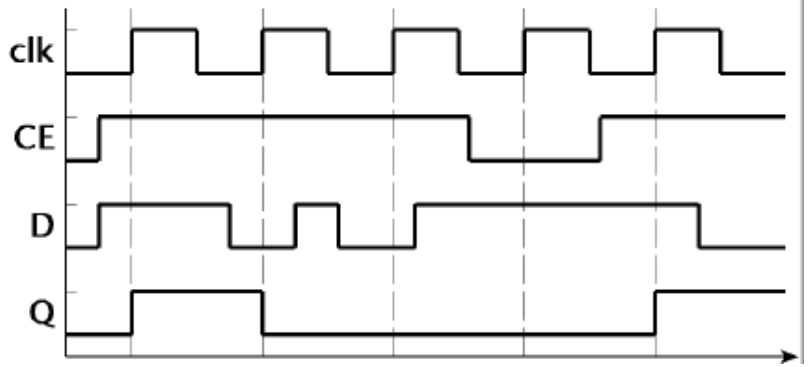
# VHDL – Ακολουθιακά Κυκλώματα

## D Flip Flop with Enable

- Η αποθήκευση ελέγχεται από την επίτρεψη (clock-enable)
  - 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;
```



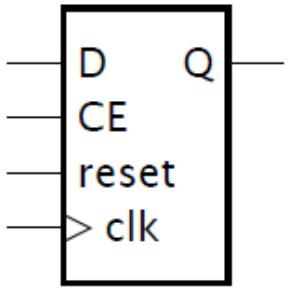



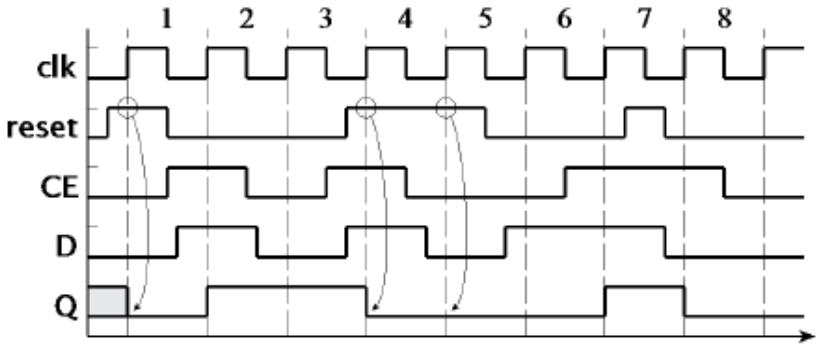
# VHDL – Ακολουθιακά Κυκλώματα

## D Flip Flop with Reset (σύγχρονο)

- 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;
```



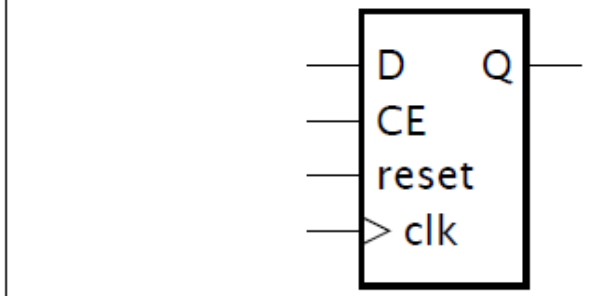



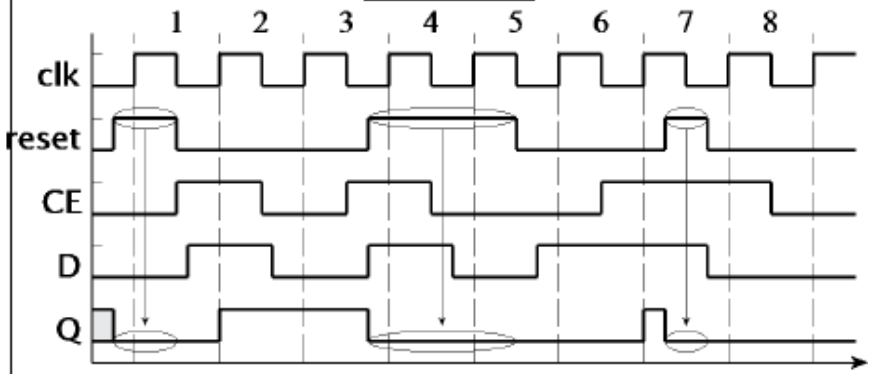
# VHDL – Ακολουθιακά Κυκλώματα

## 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;
```



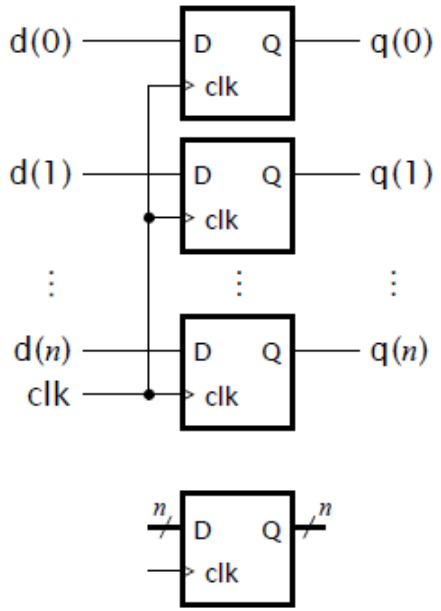



# VHDL – Ακολουθιακά Κυκλώματα

## 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;
```

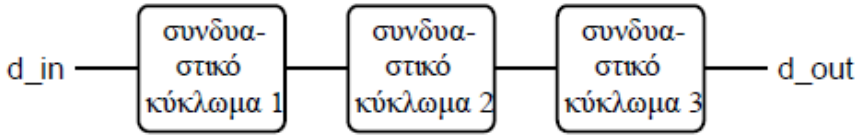
Συντομογραφία του όλα στο 0

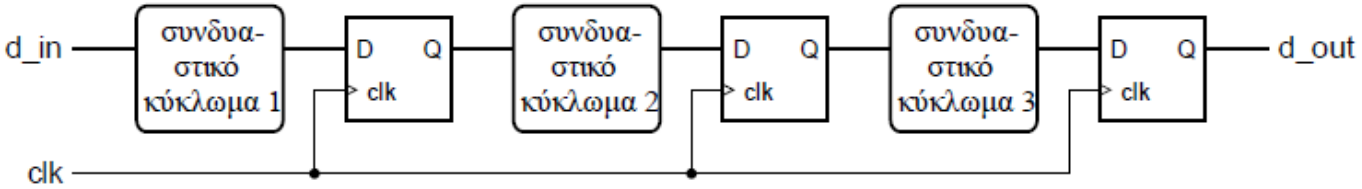
# VHDL – Ακολουθιακά Κυκλώματα

## 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 προσημασμένων αριθμών
  - Ένας νέος αριθμός φθάνει όταν data_en = 1
  - Μηδενίστε το άθροισμα με σύγχρονο reset

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

entity accumulator is
  port ( clk, reset, data_en      : in std_logic;
         data_in                  : in signed(15 downto 0);
         data_out                 : out signed(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');
      elsif data_en = '1' then
        sum <= new_sum;
      end if;
    end if;
  end process reg;
  data_out <= sum;
end architecture rtl;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 1/3)

- Εκτελούν ολίσθηση (shift) στα αποθηκευμένα δεδομένα
  - Σειριακή μεταφορά δεδομένων
- Καταχωρητής ολίσθησης των 8 bit με ασύγχρονη είσοδο reset, σειριακή είσοδο και παράλληλη έξοδο

```
entity sreg8 is
  port (clk, reset: in bit;
        sin      : in bit;
        dout     : out bit_vector(7 downto 0));
end entity;
architecture bef of sreg8 is
 signal shift_reg : bit_vector(7 downto 0);
begin
        dout <= shift_reg;
        process (clk, reset)
        ...
        end process;
end architecture;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 2/3)

- Υλοποίηση με array slices

```
process (clk, reset)
begin
   if reset = '1' then
      shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
      shift_reg(7 downto 1) <= shift_reg(6 downto 0);
      shift_reg(0) <= sin;
   end if;
end process;
```

- Υλοποίηση με array concatenation

```
process (clk, reset)
begin
   if reset = '1' then
      shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
      shift_reg <= shift_reg(6 downto 0) & sin;
   end if;
end process;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 3/3)

- Υλοποίηση με for-loop

```
process (clk, reset)
begin
   if reset = '1' then
     shift_reg <= (others => '0');
   elsif clk'event and clk = '1' then
     for i in 7 downto 1 loop
     shift_reg(i) <= shift_reg(i-1);
   end loop;
   shift_reg(0) <= sin;
  end if;
end process;
```

# 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;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Περιοδικό σήμα ελέγχου

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity decoded_counter is
  port   ( clk : in std_logic; ctrl : out std_logic );
end entity decoded_counter;

architecture rtl of decoded_counter is
  signal count_value : unsigned(3 downto 0);
begin
  counter : process (clk) is
  begin
    if rising_edge(clk) then
      count_value <= count_value + 1;
    end if;
  end process counter;
  ctrl <= '1' when count_value = "0111" or count_value = "1011" else '0';
end architecture rtl;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Παράδειγμα

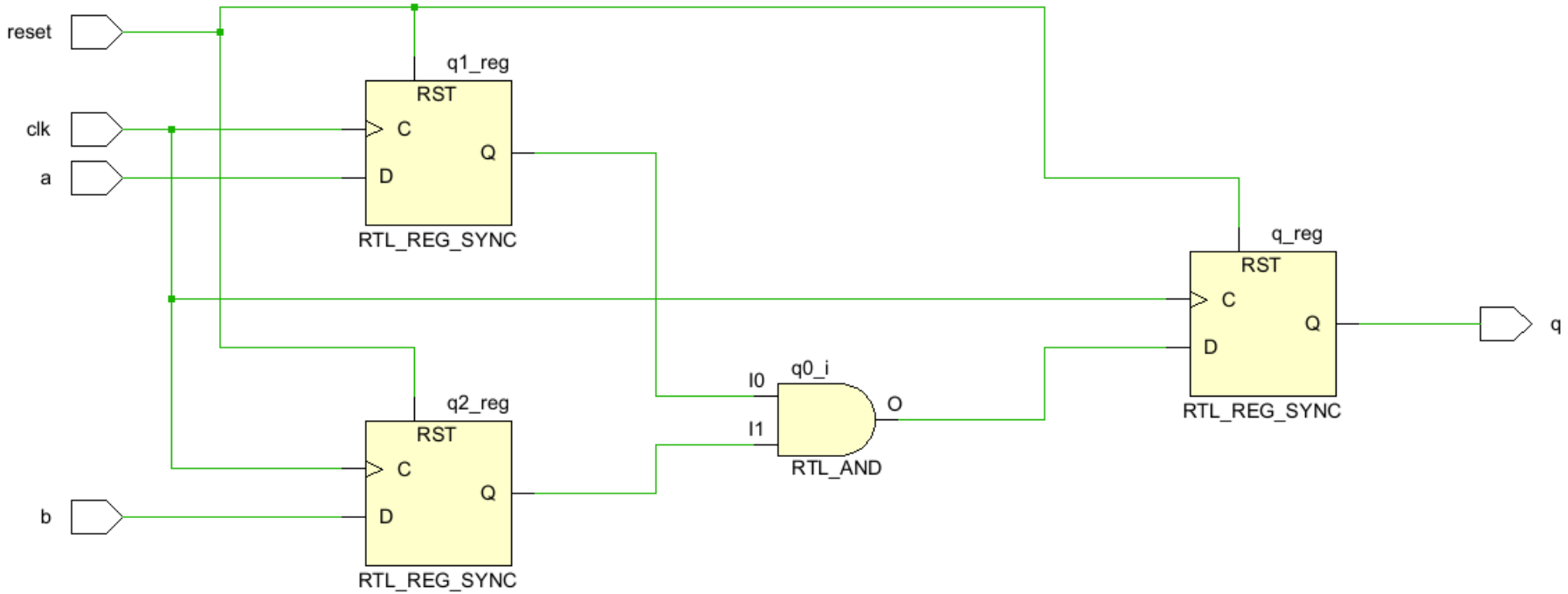
Τι κύκλωμα μοντελοποιεί ο κώδικας;

Πως υπολογίζεται η συχνότητα λειτουργίας του κυκλώματος μετά τη σύνθεση;

```
process (clk) is
begin
   if clk'event and clk='1' then
      if reset = '1' then
         q<='0';q1<='0';q2<='0';
      else
         q1<=a;
         q2<=b;
         q<=q1 and q2;
      end if;
   end if;
end process;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Παράδειγμα



# Περίληψη

- For/For Generate
- Procedure/Function
- 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.