dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

# 7ο Εργαστήριο Space Data Systems

## VHDL

## -

## Κωδικοποίηση – Ακολουθιακά κυκλώματα

**Βασιλόπουλος Διονύσης**

**ΕΔΙΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ**

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

# 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

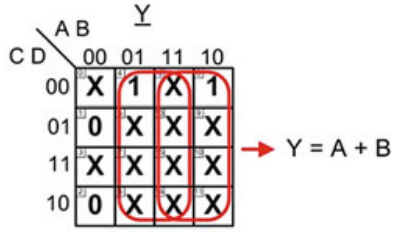
Example: 4-to-2 Binary Encoder – Logic Synthesis by Hand

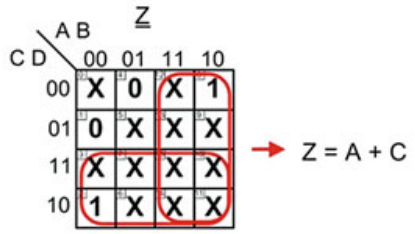
The block diagram and truth table for this system are as follows:

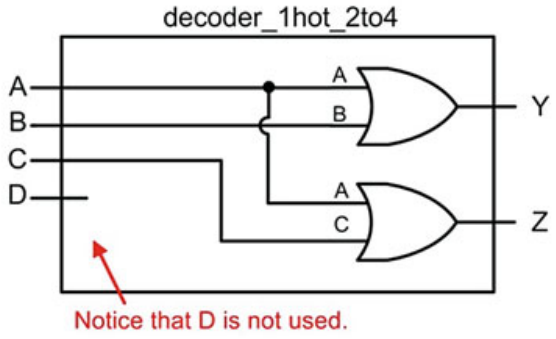

| A | B | C | D | Y | Z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

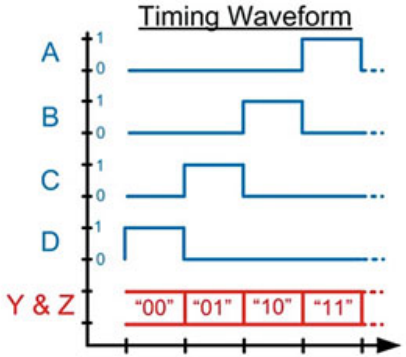
D => "00"
C => "01"
B => "10"
A => "11"

When designing this circuit, each output needs to have its own separate combinational logic circuit. When constructing the K-maps for Y and Z, each will have 4-inputs (A, B, C, D). The output values for many of the input codes are not specified in the above truth table. As such, we can use Don't Cares (X) to simplify the logic.



# 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.
  - Δεν έχει 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 Κωδικοποιητής προτεραιότητας (1/3)

- 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 Κωδικοποιητής προτεραιότητας (1η υλοποίηση) (2/3)

- 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 |

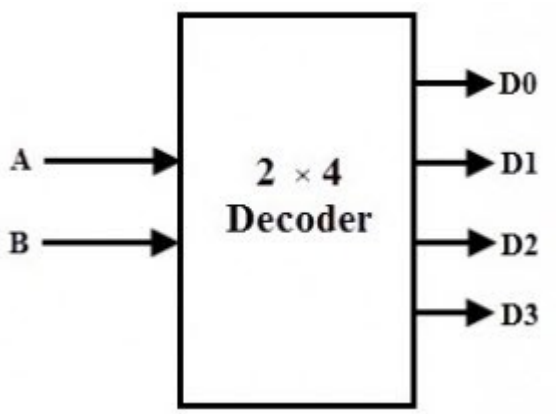
# VHDL – Κωδικοποίηση

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

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

## One Hot Decoder (2 είσοδοι – 4 έξοδοι)

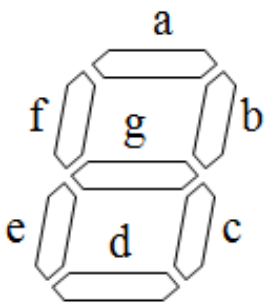
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0);
        output : out STD_LOGIC_VECTOR(3 downto 0));
end decoder_2to4;

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;
```

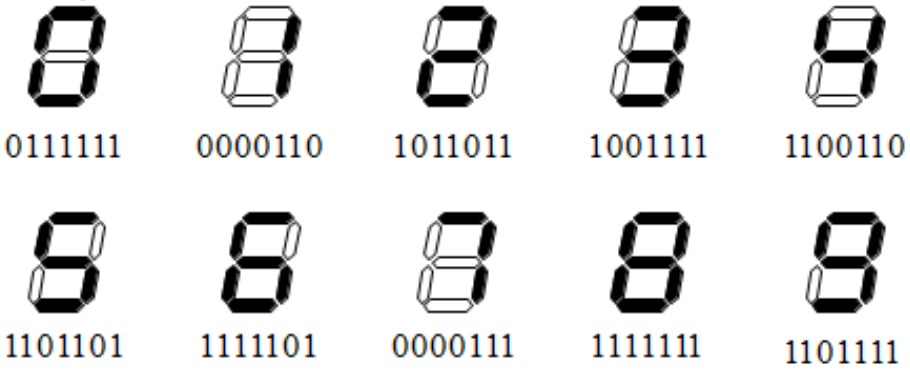
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0); -- 2-bit input
        output : out STD_LOGIC_VECTOR(3 downto 0) -- 4-bit
output);
end decoder_2to4;

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 (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 – Extra Packages

## Αρχεία

**library STD;**
**use STD.textio.all**

Reading and writing types: bit, bit_vector, integer, character, string

**library IEEE;**
**use IEEE.std_logic_textio.all**

Reading and writing types: **std_logic**, **std_logic_vector**

Ορίζονται δύο νέοι τύποι για την αλληλεπίδραση με αρχεία: **files**, **lines**

**Χρειάζονται και τα 2 πακέτα.**
Δεν είναι synthesizable
Μόνο σε προσομοίωση

# VHDL – Package textio

## Type : file

**file file_handle : <file_type> open <file_mode> is <"filename">;**

file Fout: TEXT open WRITE_MODE is "output_file.txt";
file Fin:  TEXT open READ_MODE  is "input_file.txt";

<file_type>: Περιγράφει την πληροφορία που περιέχει το αρχείο.
TEXT: Περιέχει ακολουθία χαρακτήρων
INTF: Περιέχει προσημασμένους αριθμούς των 32bit

Η δήλωση γίνεται σε process ανάμεσα στο is και το begin.
STD.textio.ALL

# VHDL – Package textio

## Type : line

**variable <line_variable_name> : line;**

Αποθηκεύει μία γραμμή που **διαβάζουμε** από ένα αρχείο ή μια γραμμή που θέλουμε να **γράφουμε** σε αρχείο. Είναι variable και όχι signal γιατί θέλουμε η εντολή να εκτελεστεί άμεσα.

Δύο συναρτήσεις για μεταφορά δεδομένων
**readline**(<file_handle>, <line_variable_name>);
**writeline**(<file_handle>, <line_variable_name>);

Διάβασμα/εγγραφή από/σε αρχείο

**writeline**(**OUTPUT**, <line_variable_name>);

**OUTPUT**: Δεσμευμένο <file handler) για το STD_OUT του υπολογιστή (στην περίπτωσή μας **Tcl Console**)

Οι γραμμές διαβάζονται (**readline**) η μία μετά την άλλη σειριακά. Πρώτα η 1η γραμμή, ύστερα η 2η , κλπ, μέχρι να φτάσουμε στο τέλος του αρχείου. Αυτό το βρίσκουμε με τη συνάρτηση **endfile()** που επιστρέφει **true** αν είμαστε στο τέλος του αρχείου. Η **readline** διαβάζει την **επόμενη** γραμμή.
Με τη **writeline** προσθέτουμε γραμμές σειριακά στο αρχείο.

# VHDL – Package textio

## Type : line

Δύο συναρτήσεις για να διαβάσουμε (**read**) τα περιεχόμενα μιας γραμμής μεταφορά δεδομένων ή βάλουμε δεδομένα (**write**) σε μια γραμμή.
**read**(<line_variable_name>, <destination_variable>);
**write**(<line_variable_name>, <source_variable>);

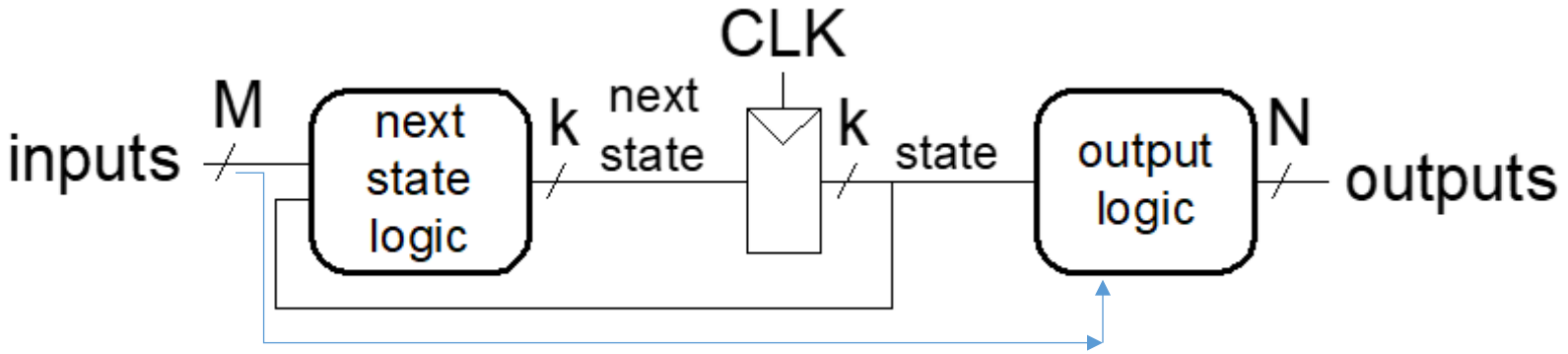
**Read**
Οι γραμμές θεωρούνται ως space-delimeted, δηλαδή μέχρι να βρεθεί κενό θεωρείται μία τιμή. Αν π.χ. μία γραμμή έχει τρεις τιμές std_logic_vector, θα πρέπει να χρησιμοποιήσουμε τρεις φορές τη read. Η <destination_variable> θα πρέπει να είναι ίδιου τύπου και μεγέθους με την πληροφορία που υπάρχει στη γραμμή του αρχείου.

**Write**
Η <source_variable> θεωρείται ότι είναι τύπου bit, bit_vector, **integer**, **std_logic**, ή **std_logic_vector**. Αν θέλουμε να γράψουμε string θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση string: string' <"characters. . .">.

Σε μια γραμμή μπορεί να υπάρχουν δεδομένα διαφορετικών τύπων

# 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 εμφανίζεται 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 ρολόι CLK




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

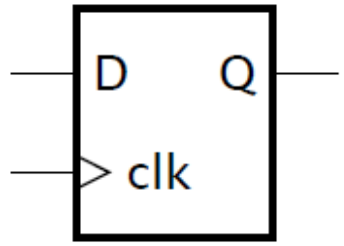
# VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

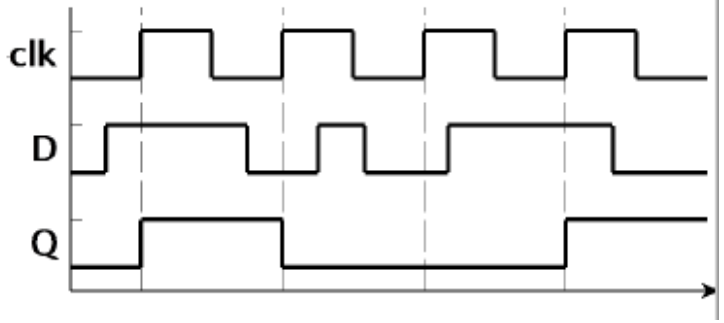
## D Flip Flop

- 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

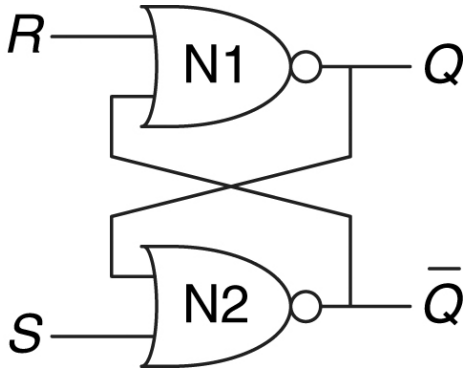


# VHDL – Ακολουθιακά Κυκλώματα

## 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 <= '0';  -- Intermediate state
            Qn <= '0';
        end if;
    end process;
end Behavioral;
```

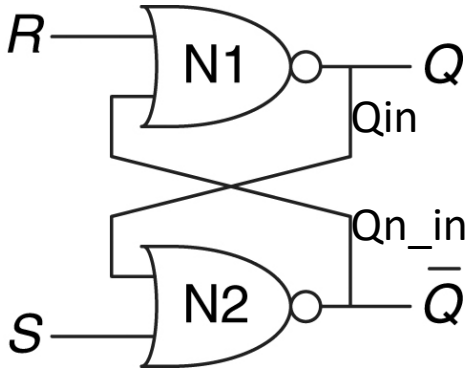
**Πίνακας Αλήθειας**

| $S$ | R | $Q$ | $\bar{Q}$ |
|---|---|---|---|
| 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;
```

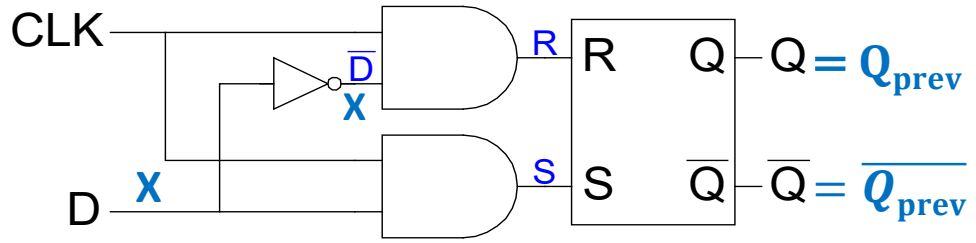
# VHDL – Ακολουθιακά Κυκλώματα

## D Latch

```
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;
```

```
architecture Behavioral of d_latch is
signal Q_next : STD_LOGIC := '0';  -- Internal signal for Q
begin
    -- Latch behavior with concurrent assignments
    Q_next <= D when CLK = '1' else
              Q_next;
    Q <= Q_next;              -- Assign internal signal to Q
    Qn <= not Q_next;         -- Complementary output
end Behavioral;
```

| 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 |



# VHDL – Ακολουθιακά Κυκλώματα

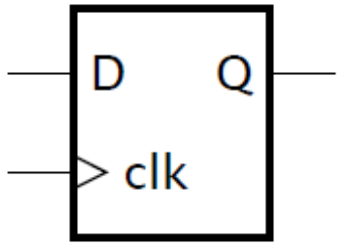
## D Flip Flop

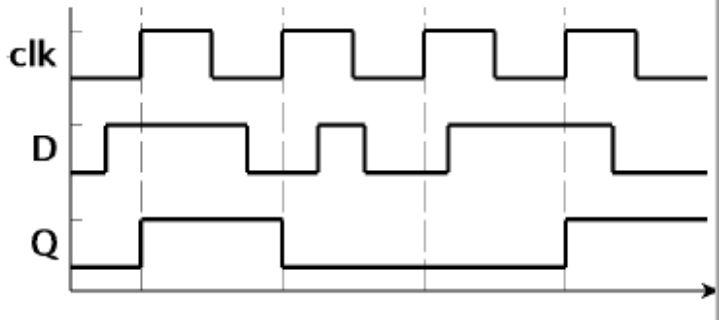
- Στοιχείο αποθήκευσης του 1 bit
- Άλλοι τύποι flip-flop
  - JK, T (toggle)

Σε όλους τους VHDL compiler δημιουργεί F/F.
Δεν χρειάζεται else, θεωρεί ότι αν δεν έχουμε ανοδική ακμή του clk κρατά την προηγούμενη τιμή

```
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



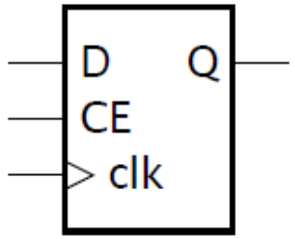
# VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

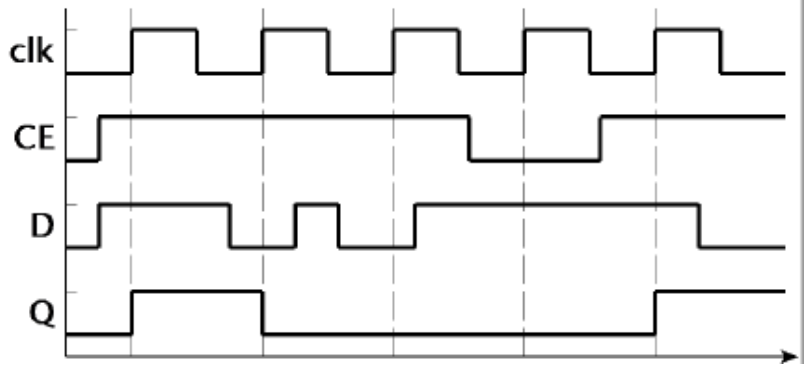
## D Flip Flop with Enable

- 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;
```

Σύγχρονο: Πρώτα έλεγχος για το clock



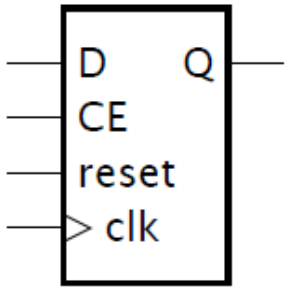
# VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

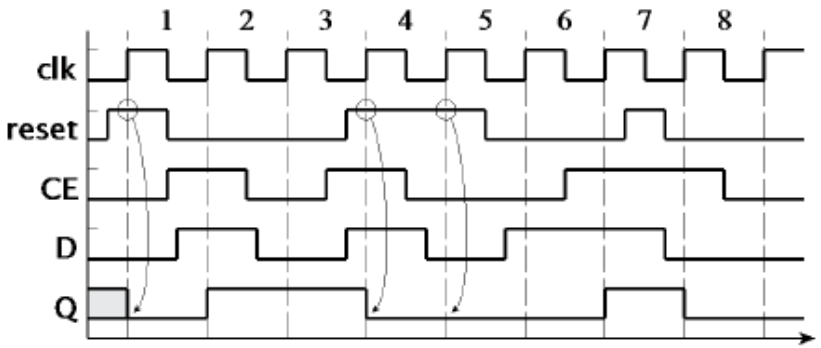
## D Flip Flop with Reset (σύγχρονο)

- 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;
```

Σύγχρονο reset: Πρώτα έλεγχος για το clock



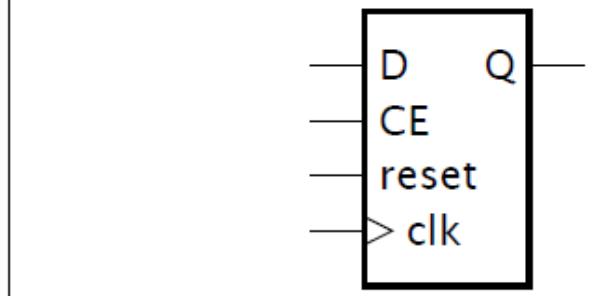



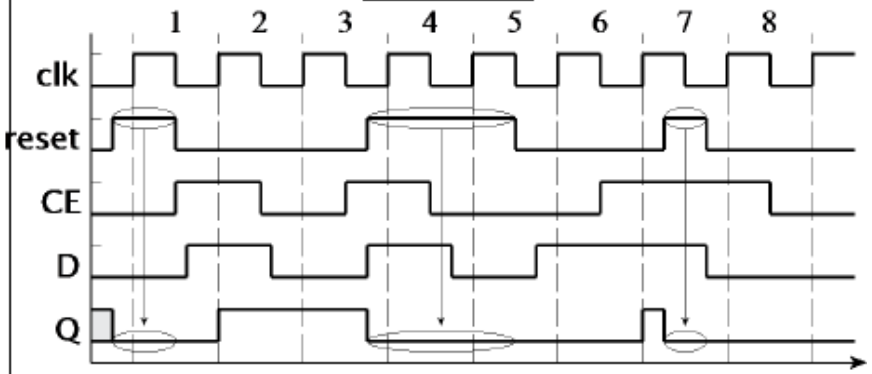
# VHDL – Ασύγχρονα Ακολουθιακά Κυκλώματα

## D Flip Flop with Reset (ασύγχρονο) - Clear

- 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;
```



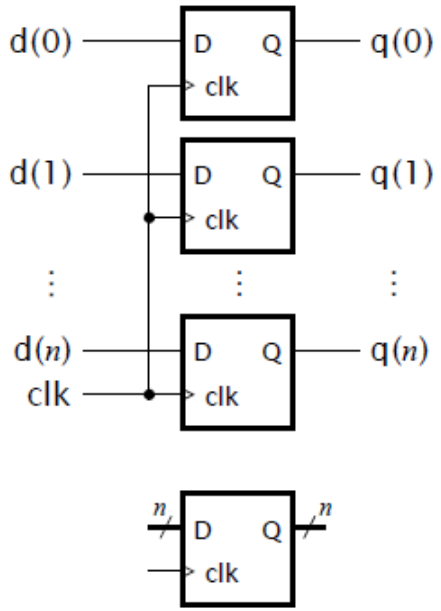



# VHDL – Ασύγχρονα Ακολουθιακά Κυκλώματα

## 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;
```

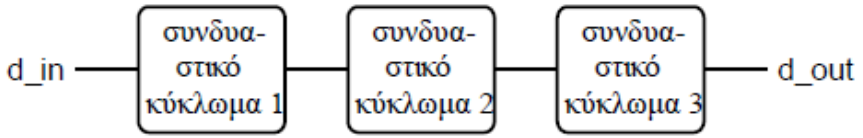
Συντομογραφία του όλα στο 0

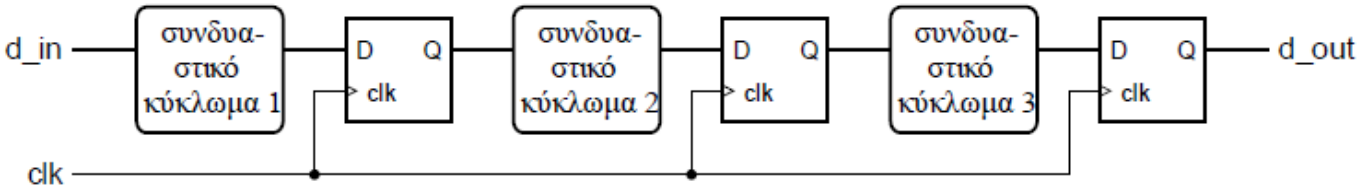
# VHDL – Ακολουθιακά Κυκλώματα

## 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

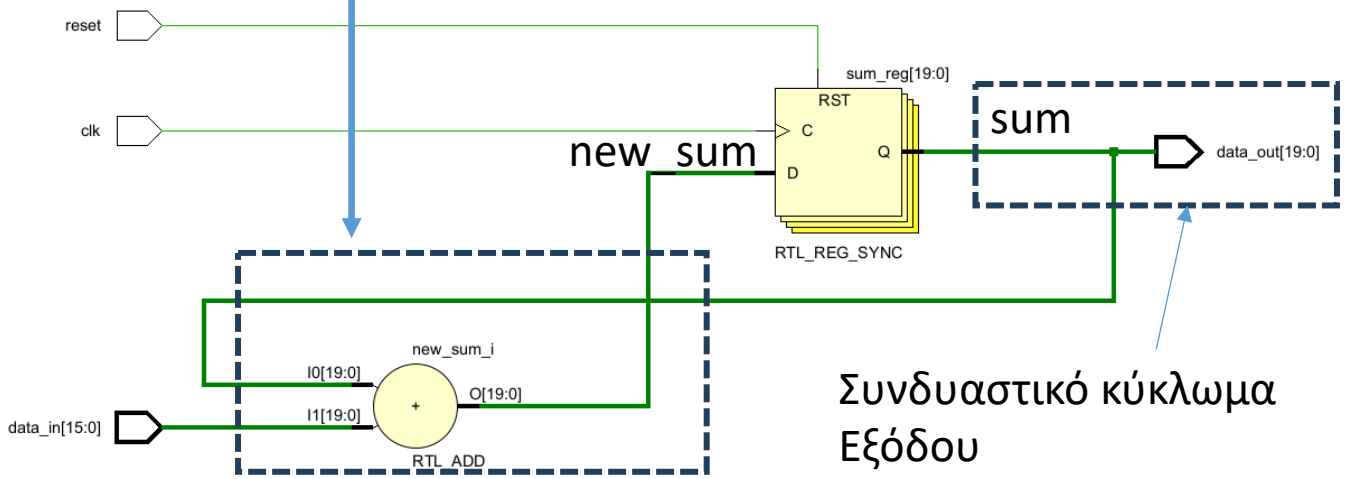
## Συσσωρευτής (accumulator 2/2)

```
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

Συνδυαστικό κύκλωμα Εξόδου

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




Συνδυαστικό κύκλωμα Εξόδου

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 1/5)

Εκτελούν ολίσθηση (shift) στα δεδομένα του καταχωρητή. Παράδειγμα **Καταχωρητή ολίσθησης** με ασύγχρονη είσοδο reset, σειριακή είσοδο και παράλληλη έξοδο.

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

entity ShiftRegister is
   Port (
      clk, reset   : in  std_logic;
      D     : in  std_logic;
      Q     : out std_logic_vector(7 downto 0) );
end ShiftRegister;

architecture Behavioral of ShiftRegister is
   signal reg : std_logic_vector(7 downto 0);
begin
   process(clk, reset)
…..
   end process;
   Q <= reg;
end Behavioral;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 2/5)

### Υλοποίηση με array concatenation

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

entity ShiftRegister is
   Port (
      clk, reset   : in  std_logic;
      D     : in  std_logic;
      Q     : out std_logic_vector(7
downto 0) );
end ShiftRegister;
```

```
architecture Behavioral of ShiftRegister is
   signal reg : std_logic_vector(7 downto 0);
begin
   process(clk, reset)
   begin
      if reset = '1' then
         reg <= (others => '0');  -- Asynchronous reset
      elsif rising_edge(clk) then
         reg <= reg(6 downto 0) & D;  -- Shift left and insert D at LSB
      end if;
   end process;

   Q <= reg;
end Behavioral;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 3/5)

### Υλοποίηση με array slices

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

entity ShiftRegister is
    Port (
        clk, reset   : in  std_logic;
        D     : in  std_logic;
        Q     : out std_logic_vector(7
downto 0) );
end ShiftRegister;
```

```
architecture Behavioral of ShiftRegister is
    signal reg : std_logic_vector(7 downto 0);
begin
    process(clk, reset)
    begin
        if reset = '1' then
            reg <= (others => '0');  -- Asynchronous reset
        elsif rising_edge(clk) then
            reg(7 downto 1) <= reg(6 downto 0);  -- Shift left using slice
            reg(0) <= D;                         -- Insert new bit at LSB
        end if;
    end process;

    Q <= reg;
end Behavioral;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 4/5)

### Υλοποίηση με for … loop

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

entity ShiftRegister is
   Port (
      clk, reset   : in  std_logic;
      D     : in  std_logic;
      Q     : out std_logic_vector(7 downto
0) );
end ShiftRegister;
```

```
architecture Behavioral of ShiftRegister is
   signal reg : std_logic_vector(7 downto 0);
begin
   process(clk, reset)
   begin
      if reset = '1' then
         reg <= (others => '0');  -- Asynchronous reset
      elsif rising_edge(clk) then
         -- Shift using a for loop
         for i in 7 downto 1 loop
            reg(i) <= reg(i - 1);
         end loop;
         reg(0) <= D;                                  -- Insert new bit at LSB
      end if;
   end process;
   Q <= reg;
end Behavioral;
```

# VHDL – Ακολουθιακά Κυκλώματα

## Ολισθητές (shift registers 5/5)

### 4-stage – 8bit Shift Register

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

entity FourStageEightBitShifter is
    Port (
        clk   : in  std_logic;
        reset : in  std_logic;
        D     : in  std_logic_vector(7 downto 0);
        Q0    : out std_logic_vector(7 downto 0);  -- Stage 0
        Q1    : out std_logic_vector(7 downto 0);  -- Stage 1
        Q2    : out std_logic_vector(7 downto 0);  -- Stage 2
        Q3    : out std_logic_vector(7 downto 0)   -- Stage 3
    );
end FourStageEightBitShifter;

architecture Behavioral of FourStageEightBitShifter is
    signal stage0 : std_logic_vector(7 downto 0);
    signal stage1 : std_logic_vector(7 downto 0);
    signal stage2 : std_logic_vector(7 downto 0);
    signal stage3 : std_logic_vector(7 downto 0);
begin
    process(clk, reset) is
    begin
        if reset = '1' then
            stage0 <= (others => '0'); stage1 <= (others => '0');
            stage2 <= (others => '0'); stage3 <= (others => '0');
        elsif rising_edge(clk) then
            stage0 <= D;
            stage1 <= stage0;
            stage2 <= stage1;
            stage3 <= stage2;
        end if;
    end process;

    -- Outputs
    Q0 <= stage0;
    Q1 <= stage1;
    Q2 <= stage2;
    Q3 <= stage3;
end Behavioral;
```

# 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;
use IEEE.NUMERIC_STD.ALL;

entity async_counter is
   Port (
      clk   : in  std_logic;
      reset : in  std_logic;
      count : out std_logic_vector(4 downto 0)
   );
end entity async_counter;

architecture rtl of async_counter is
   signal count_reg : unsigned(4 downto 0) := (others => '0');
begin
   process (clk, reset)
   begin
      if reset = '1' then
         count_reg <= (others => '0');
      elsif rising_edge(clk) then
         if count_reg = 20 then
            count_reg <= (others => '0');
         else
            count_reg <= count_reg + 1;
         end if;
      end if;
   end process;

   count <= std_logic_vector(count_reg);

end architecture rtl;
```

Μετρητής από το 0 έως το 20. Επόμενη τιμή από το 20 είναι το 0.

# VHDL – Ακολουθιακά Κυκλώματα

## Δεκαδικός Μετρητής (σύγχρονος)

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

entity decade_counter is
   Port (
      clk    : in  std_logic;
      reset  : in  std_logic;
      counts : out std_logic_vector(3 downto 0)
   );
end entity decade_counter;

architecture rtl of decade_counter is
   signal count_reg : unsigned(3 downto 0) := (others => '0');
begin
   process (clk)
   begin
      if rising_edge(clk) then
         if reset = '1' then
            count_reg <= (others => '0');
         elsif count_reg = 9 then
            count_reg <= (others => '0');
         else
            count_reg <= count_reg + 1;
         end if;
      end if;
   end process;
   counts <= std_logic_vector(count_reg);
end architecture rtl;
```

Μετρητής από 0 έως το 9. Επόμενη τιμή από το 9 είναι το 0.

# VHDL – Ακολουθιακά Κυκλώματα

## Μετρητής με Περιοδικό Σήμα Ελέγχου

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

entity periodic_control_counter is
   Port (
      clk   : in  std_logic;
      reset : in  std_logic;
      ctrl  : out std_logic
   );
end entity periodic_control_counter;

architecture rtl of periodic_control_counter is
   signal count_reg : unsigned(4 downto 0) := (others => '0');
begin
  process (clk)
    begin
      if rising_edge(clk) then
        if reset = '1' then
          count_reg <= (others => '0');
        else
          count_reg <= count_reg + 1;
        end if;
      end if;
    end process;
    ctrl <=    '1' when (count_reg = 4) or (count_reg = 12)
          else '0';
end architecture rtl;
```

Μετρητής από 0 έως το 31. Επόμενη τιμή από το 31 είναι το 0.

# VHDL – Ακολουθιακά Κυκλώματα

## Ασύγχρονος Μετρητής με φόρτωση

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

entity sync_load_counter is
   Port (
      clk       : in  std_logic;
      reset     : in  std_logic;
      load      : in  std_logic;
      load_value : in  std_logic_vector(4 downto 0);
      counts    : out std_logic_vector(4 downto 0)
   );
end entity sync_load_counter;
architecture rtl of sync_load_counter is
   signal count_reg : unsigned(4 downto 0) := (others => '0');
begin
   process(clk, reset)
    begin
      if reset = '1' then
         count_reg <= (others => '0');
      elsif rising_edge(clk) then
         if load = '1' then
            count_reg <= unsigned(load_value);
         else
            count_reg <= count_reg + 1;
         end if;
      end if;
   end process;
   counts <= std_logic_vector(count_reg);
end architecture rtl;
```

Μετρητής από 0 έως το 31. Επόμενη τιμή από το 31 είναι το 0.

# VHDL – Ακολουθιακά Κυκλώματα

## Παράδειγμα

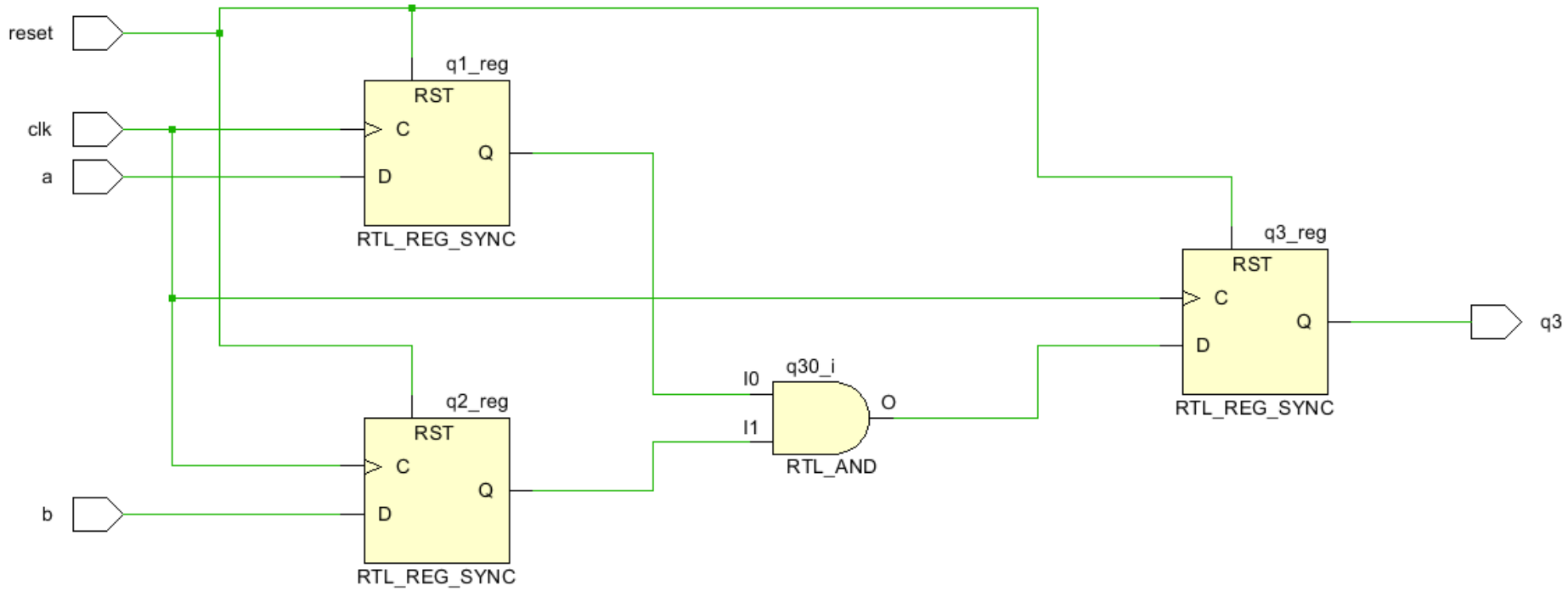
Τι κύκλωμα μοντελοποιεί ο κώδικας;

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
         q3<='0';q1<='0';q2<='0';
      else
         q1<=a;
         q2<=b;
         q3<=q1 and q2;
      end if;
   end if;
end process;
```

# VHDL – Ακολουθιακά Κυκλώματα

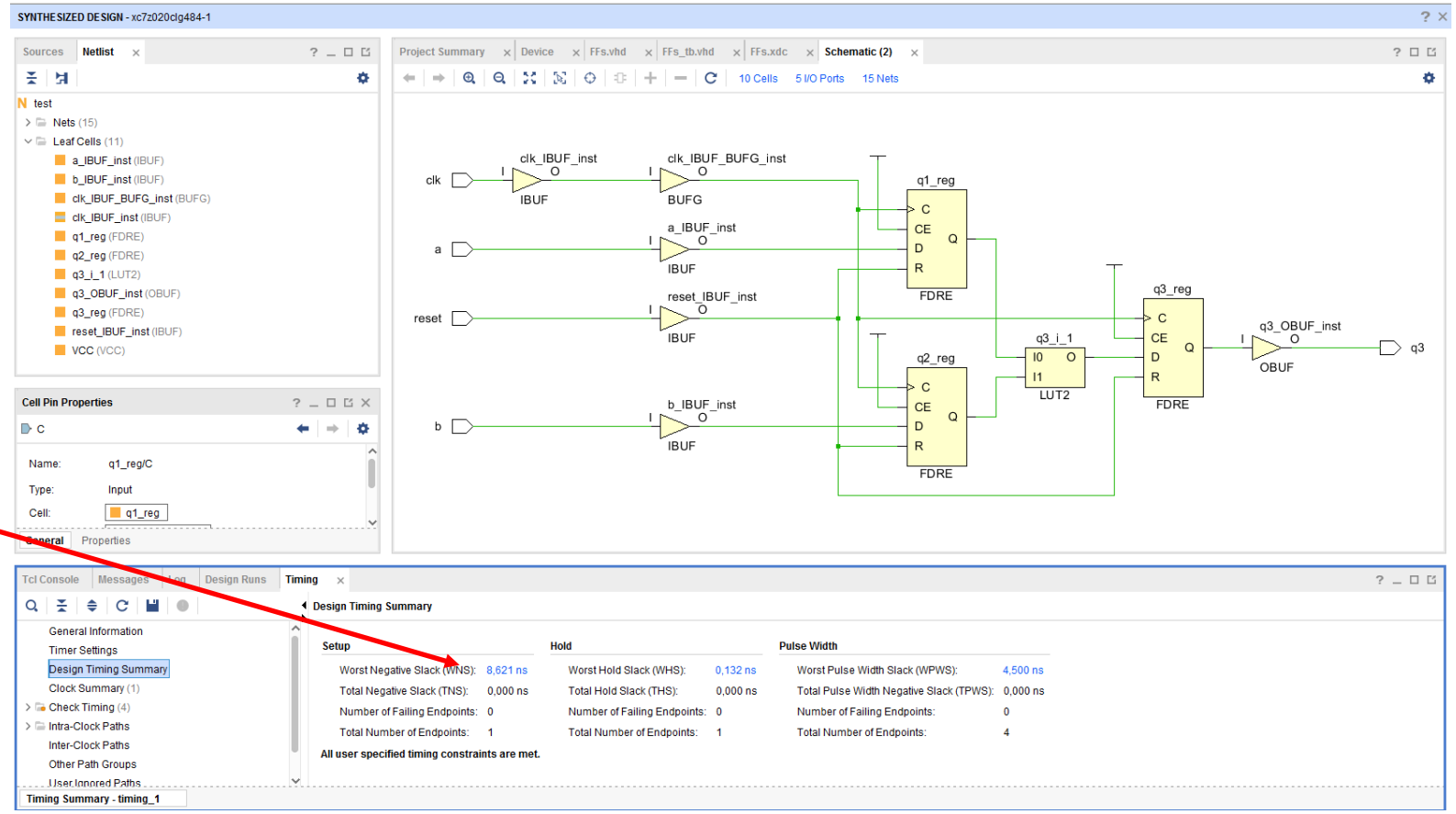
## Παράδειγμα



# VHDL – Ακολουθιακά Κυκλώματα

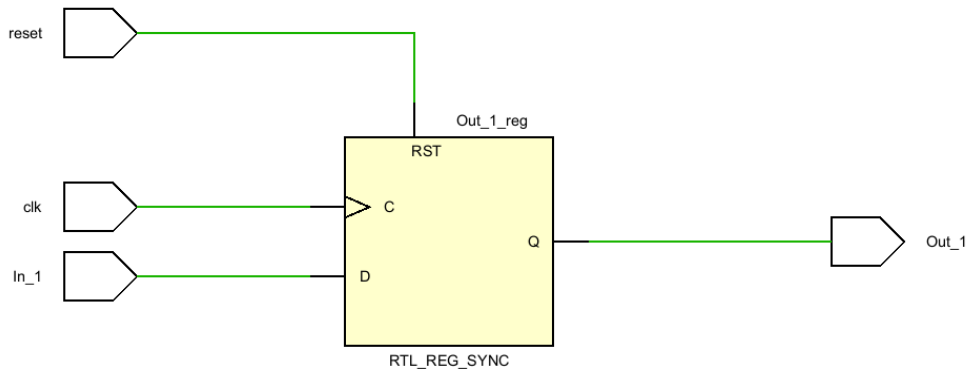
## Παράδειγμα

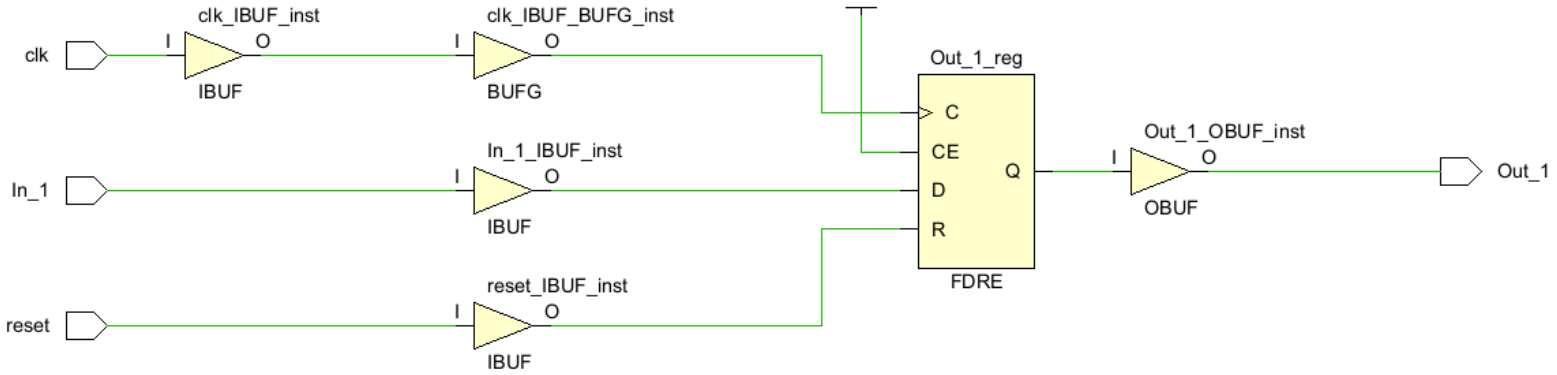
Tc=10ns-Slack



# VHDL – Variables

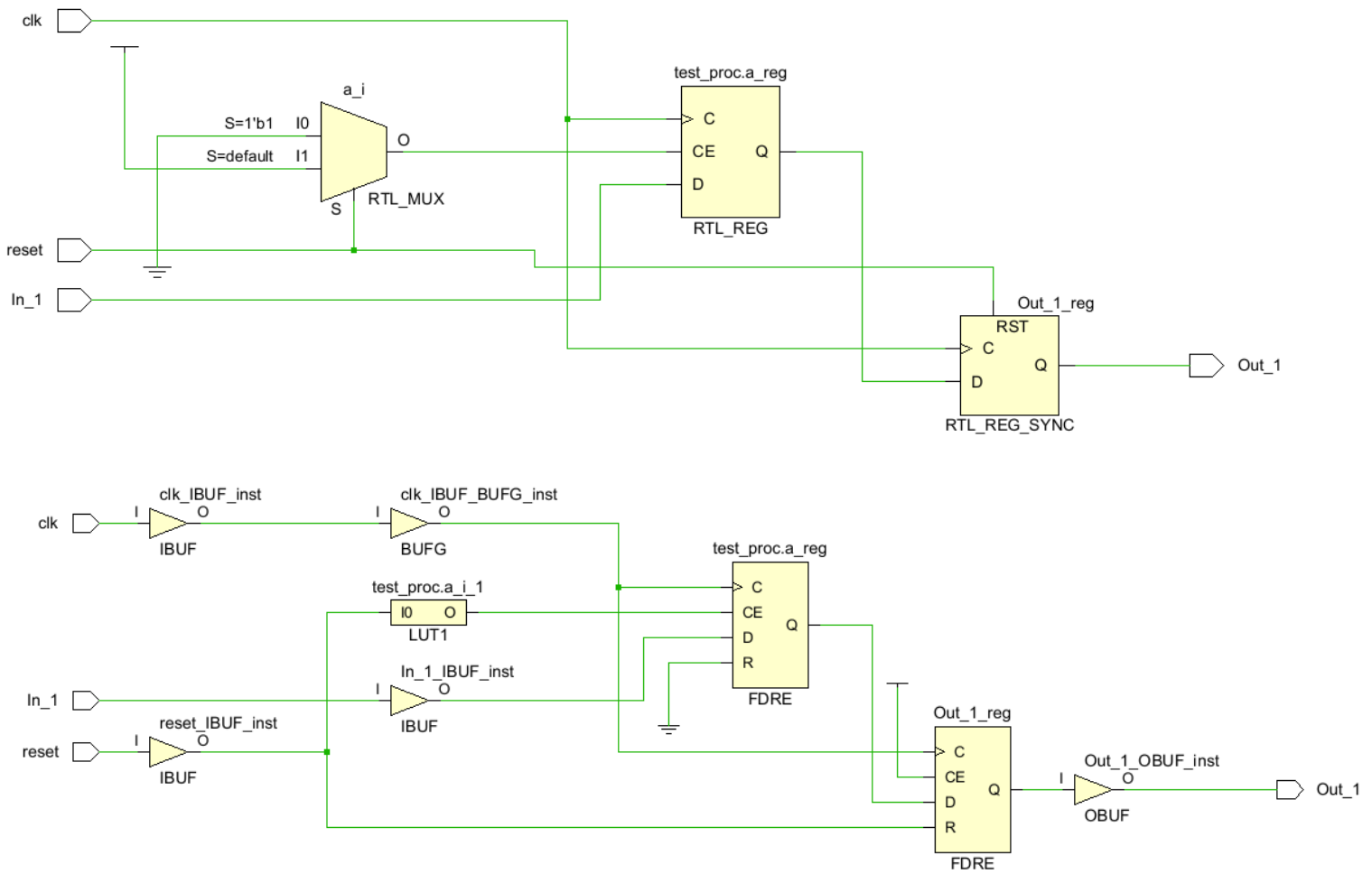
```
test_proc: process (clk) is

variable a: std_logic;

begin

if rising_edge(clk) then
   if reset='1' then
      Out_1<='0';
   else
      a:=In_1;
      Out_1<=a;
   end if;
end if;

end process test_proc;
```



# VHDL – Variables

```
test_proc: process (clk) is

variable a: std_logic;

begin


if rising_edge(clk) then
   if reset='1' then
      Out_1<='0';
   else
      Out_1<=a;
      a:=In_1;
   end if;
end if;

end process test_proc;
```



# Περίληψη

- Κωδικοποίηση/Αποκωδικοποίηση
- Ακολουθιακά κυκλώματα
- Latches/Flip Flop/Registers
- Accumulator, counter, shifter,
- Διαβάζετε τις παραγράφους  2.3.1, 4.1, 4.1.1, 4.1.2, 4.2 (θεωρία και VHDL) από Ashenden
- Διαβάζετε τις παραγράφους 2.8.2, 3.1, 3.2, 3.3, 3.6, 4.4, 4.8, 5.2.5, 5.4.1, 5.4.2, 7.5 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Διαβάζετε τις παραγράφους 4.3, 4.4, 5.1-5.6, 5.9, 5.13, 12.8, 12.9.4, 12.10.1, 12.10.2, 12.10.5, 12.10.6, 12.10.7, 12.10.8, 12.10.9 από το βιβλίο των Brown-Vranesic.