Sequential VHDL

- Ακολουθιακός κώδικας
- Signals και variables
- Process statement
- Παράδειγμα process
  - Περιγραφή VHDL
  - Προσομοίωση
- Combinational Process
- Clocked Process
- Παράδειγμα clocked process
  - Περιγραφή VHDL
  - Προσομοίωση

Sequential VHDL

- IF statement
- Case statement
- Null statement
- Wait statement
- For loop statement
- Παράδειγμα loop statement
  - Περιγραφή VHDL
  - Προσομοίωση
- While loop statement
- Flip-flop asynchronous reset
- Flip-flop synchronous reset
- Latches
- Ασκήσεις-Προβλήματα

Ακολουθιακός κώδικας

- κώδικας VHDL
  - εκτελείται μέσα στην architecture με συντρέχοντα τρόπο (concurrently)
  - **εκτός άν** συναντούμε processes, functions και procedures

```
Architecture rtl of ex is
concurrent declaration part
begin
 concurrent VHDL

 process(...)
 sequential declaration part
 begin
      sequential VHDL
 end process;

 concurrent VHDL
end;
```

Process, one concurrent statement

Signals και variables

- **signals**
  - ορίζονται στο συντρέχον μέρος του κώδικα
  - λαμβάνουν τιμές με το σύμβολο <=
  - χρησιμοποιούνται τόσο στο συντρέχον όσο και στο ακολουθιακό μέρος

- **variables**
  - ορίζονται και χρησιμοποιούνται **μόνο στο ακολουθιακό μέρος** του κώδικα
  - λαμβάνουν τιμές με το σύμβολο :=
  - έχουν την ίδια συμπεριφορά με τις **variables** στις κοινές γλώσσες προγραμματισμού (C++, C, Fortran κτλ)
  - μπορούν να ορισθούν για τους ίδιους data types όπως και τα **signals**

- επιτρέπεται ο ορισμός της τιμής μίας variable με την τιμή ενός signal και αντιστρόφως *(εφόσον βέβαια είναι του ίδιου data type)*

Signals και variables

Sum1 and sum2 are signals:

```
p0:process
begin
    wait for 10 ns;
    sum1<=sum1+1;
    sum2<=sum1+1;
end process;
```

Sum1, Sum2 = Signals

Time	Sum 1	Sum 2
0	0	0
10	0	0
10 + 1 delta	1	1
20	1	1
20 + 1 delta	2	2
30	2	2
30 + 1 delta	3	3

Sum1 and sum2 are variables:

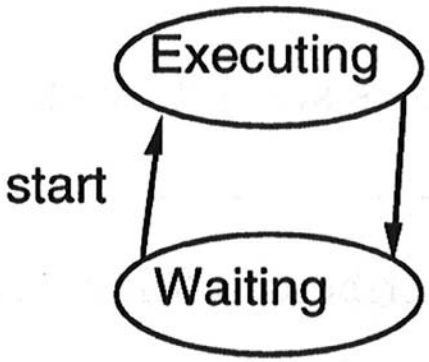
```
p1:process
variable sum1, sum2: integer;
begin
    wait for 10 ns;
    sum1:=sum1+1;
    sum2:=sum1+1;
end process;
```

Sum1, Sum2 = Variables

Time	Sum 1	Sum 2
0	0	0
10	1	2
10 + 1 delta	1	2
20	2	3
20 + 1 delta	2	3
30	3	4
30 + 1 delta	3	4

Process statement

- η ιδέα του process έρχεται από το κοινό software το οποίο είναι ακολουθιακό (sequential)
- μία process μπορεί να βρίσκεται σε κατάσταση αναμονής (waiting) ή εκτέλεσης (executing)
- άν η process βρίσκεται σε κατάσταση αναμονής
  μία ειδική συνθήκη πρέπει να πληρείται ώστε να αρχίσει να εκτελείται, π.χ. **wait until clk='1';**



Process statement

The syntax for the process is:

```
[<process_name> :] process [ (<sensitivity_list >) ]
      [<process_declarative_part>]
begin
      <process_statement_part>
end process [<process_name>];
```

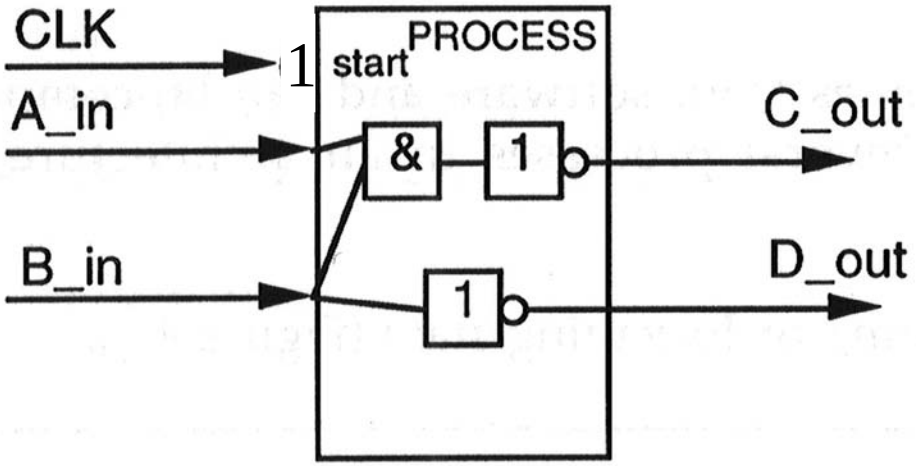
Syntax (VHDL-93):

```
[<process_name> :] process [ (<sensitivity_list >) ] [is]
      ...
begin
      ...
end process [<process_name>];
```

- μία process εκτελείται **με κάθε αλλαγή** ενός από τα εισερχόμενα signals, τα οποία περιέχονται στην sensitivity list

Παράδειγμα process

- Σε αυτό το παράδειγμα δεν δηλώνουμε τα σήματα στη sensitivity list αλλά χρησιμοποιούμε την εντολή **wait until**.



Περιγραφή VHDL

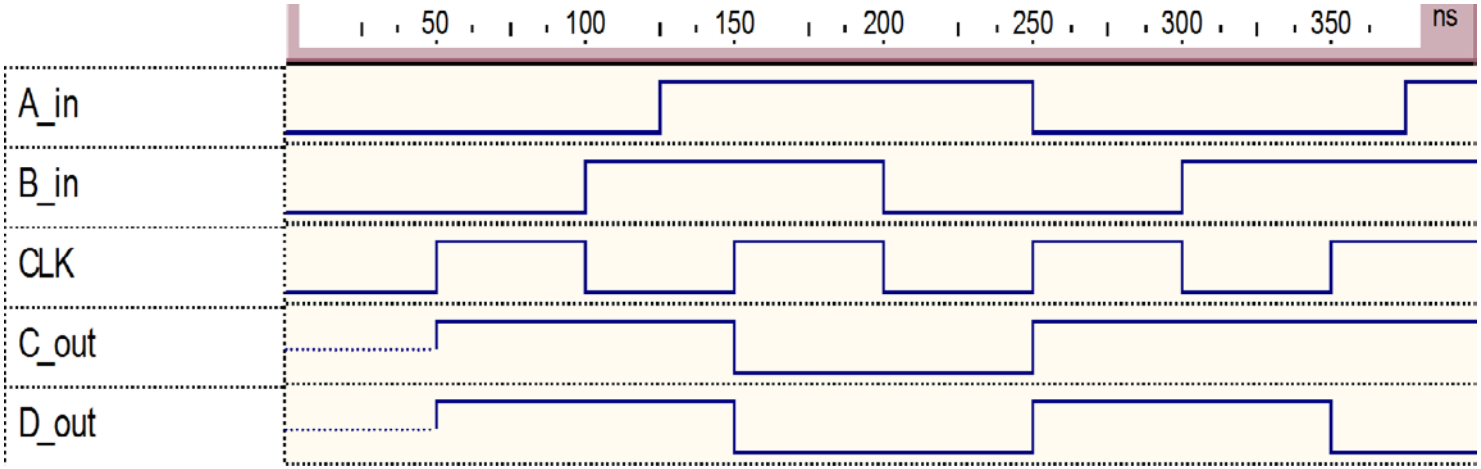
```
--  File: c:\my designs\ex_1_proc\SRC\ex_1_proc.VHD
--  Example process Created by Design Wizard: 01/30/01 13:36:15
library IEEE;
use IEEE.std_logic_1164.all;

entity ex_1_proc is
   port (CLK, A_in, B_in : in STD_LOGIC;
         C_out, D_out      : out STD_LOGIC);
end ex_1_proc;

architecture ex_1_proc_behav of ex_1_proc is
begin
  sync_process: process
  begin
     wait until CLK='1';
      C_out <= not (A_in and B_in);
      D_out <= not B_in;
  end process sync_process;
end ex_1_proc_behav;
```

Προσομοίωση

- CLK : 10 MHz , A_in : 4 MHz, B_in : 5MHz



Combinational Process

- Οι combinational process χρησιμοποιούνται στον σχεδιασμό συνδυαστικής λογικής (combinational logic).
- Σε μία combinational process όλα τα εισερχόμενα signals πρέπει να δηλώνονται στην sensitivity list (λίστα ευαισθησίας).
- Η process εκτελείται με κάθε αλλαγή ενός από τα εισερχόμενα signals τα οποία περιέχονται στην sensitivity list.
- Υπάρχει η δυνατότητα παράληψης ενός signal στην sensitivity list. Αυτό όμως πρέπει να γίνεται μόνο σε περιπτώσεις modeling.
- Εάν ένα signal παραλειφθεί από την sensitivity list τότε το αποτέλεσμα της προσομοίωσης και της σύνθεσης (hardware) θα είναι διαφορετικά. Αυτό είναι ένα σοβαρό λάθος.

- **Κανόνας** : Πρέπει να δηλώνονται όλα τα signals στην sensitivity list στην περίπτωση της combinational process.

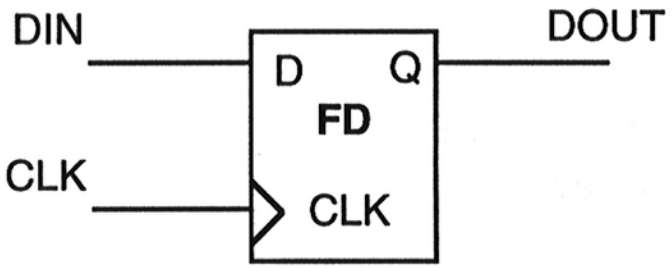
Combinational Process

- Σωστό παράδειγμα
  - comb_process : **process** (A_in, B_in)
    begin
      C_out <= **not** (A_in **and** B_in);
      D_out <= **not** B_in;
    end process comb_process;
- Παράδειγμα προς αποφυγήν
  - comb_process : **process** (B_in)
    begin
      C_out <= **not** (A_in **and** B_in);
      D_out <= **not** B_in;
    end process comb_process;

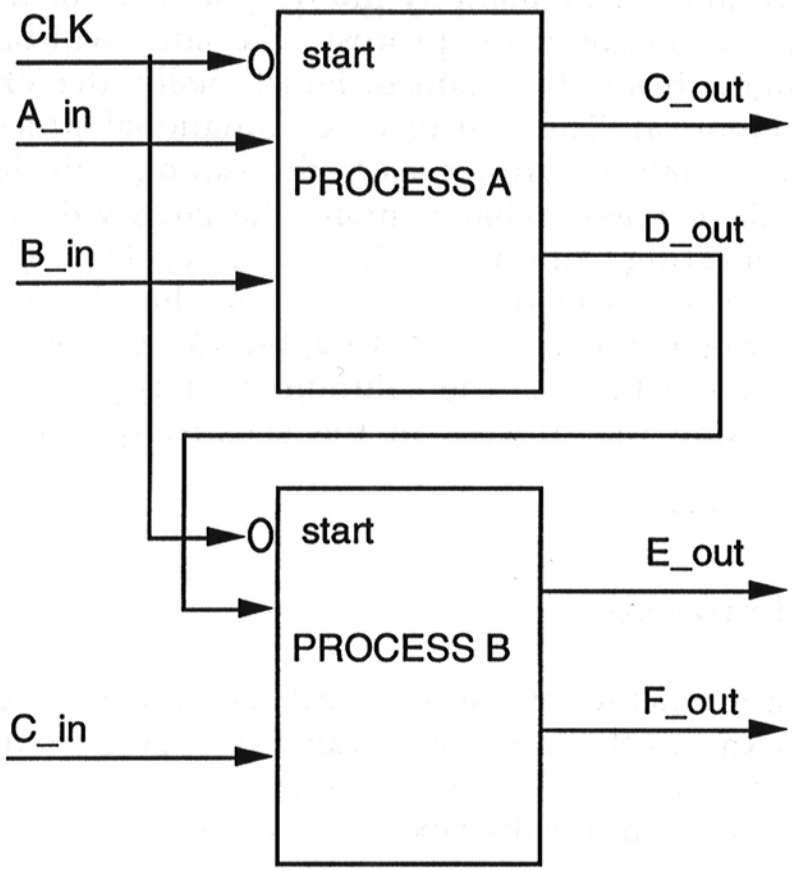
Clocked Process

- Οι clocked process χρησιμοποιούνται στον σχεδιασμό flip-flops και πιθανόν και σε συνδυαστική λογική (combinational logic).
- Οι clocked process είναι σύγχρονες και πολλές από αυτές μπορούν να χρησιμοποιούν το ίδιο ρολόϊ (clock).
- Εκτελούνται με την αλλαγή (πτώση ή άνοδο) του ρολογιού.
- Οι clocked process δίνουν flip-flops

```
example : process
begin
   wait until CLK='1';
   DOUT <= DIN;
end process example;
```



Παράδειγμα clocked Process



Περιγραφή VHDL

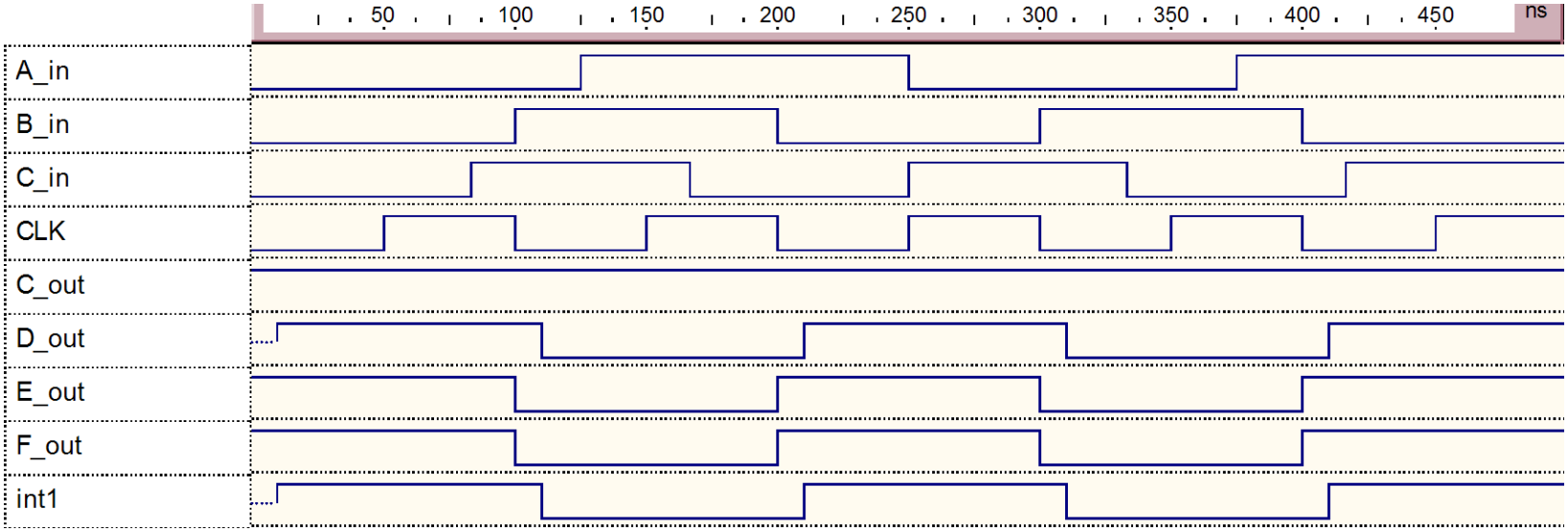
```
--  File: c:\my designs\ex_2_proc\SRC\ex_2_proc.VHD
--  Example clocked process. Created by Design Wizard: 01/30/01 18:07:40
library IEEE;
use IEEE.std_logic_1164.all;

entity ex_2_proc is
   port (   CLK: in STD_LOGIC;
            A_in: in STD_LOGIC;
            B_in: in STD_LOGIC;
            C_in: in STD_LOGIC;
            C_out: out STD_LOGIC;
            D_out: out STD_LOGIC;
            E_out: out STD_LOGIC;
            F_out: out STD_LOGIC);
end ex_2_proc;
```

Περιγραφή VHDL

```
architecture ex_2_proc_beh of ex_2_proc is
signal int1 : STD_LOGIC;
begin
    A_process : process
    begin
       wait until CLK='0';
            C_out <= not (A_in and B_in);
            D_out <= not B_in after 10ns;
            int1    <= not B_in after 10ns;
    end process A_process;
    B_process : process
    begin
       wait until CLK='0';
            E_out <= not (int1 and C_in);
            F_out <= not C_in;
    end process B_process;
end ex_2_proc_beh;
```

Προσομοίωση



IF statement

- Η εντολή **if** αντιστοιχεί στη συντρέχουσα εντολή **when-else**

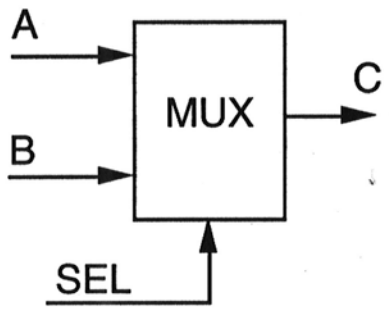
Syntax:

if <condition> **then** <sequence_of_statements>
[elsif <condition> **then** <sequence_of_statements>] ....
[else <sequence_of_statements>]
end if;

<condition>=
<expression> [<relational_operator> <expression>]..

<relational_operator>=
= /= < <= > >=

- **If** SEL = '1' **then**
  C <= B;
  else
  C <= A;
  end if;



IF statement

- Παραδείγματα

```
if sel = '1' then
   c<='1'; ...
elsif John = "1001" then
   c <= '0'; ...
elsif David > John then
   c <= '1'; ...
else
   c <= 'X'; ...
end if;
```

```
Entity ex_if is
  port( ssel:     in   std_logic;
        a,b,syn:  in   std_logic;
        sout:     out  std_logic);
end;

 Architecture rtl of ex_if is
 begin
  process(ssel,syn,a,b)
  begin
    if ssel='0' and syn='1' then
      sout<=a;
    else
      if ssel='1' and syn='0' then
        sout<=b;
      else
        sout<='0';
      end if;
    end if;
  end process;
end;
```

Case statement

- Η εντολή **case** αντιστοιχεί στη συντρέχουσα εντολή **with-select**

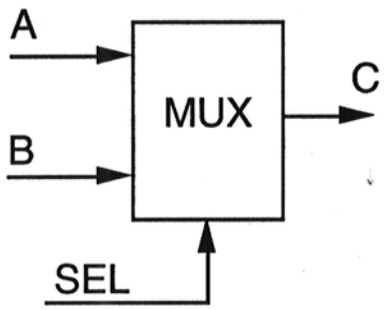
Syntax:

case <expression> **is**
    when <choice> => <sequence_of_statements> ;
    when <choice> => <sequence_of_statements> ;
    ...
    [**when others =>** [<sequence_of_statements>] ;]
end case ;

<choice>= <choice> [| <choice> [...]]

| = or

- **case** SEL **is**
    when '0' => C <= A;
    when '1' => C <= B;
  end case;



Case statement

- Παραδείγματα

```
Entity case_ex2 is
   port( a:   in    integer range 0 to 30;
         b:   out   integer range 0 to 6);
end;
Architecture rtl of case_ex2 is
begin
   p1:process(a)
   begin
      case a is
        when 0 =>          b<=3;
        when 1 | 2=>       b<=2;
        when others =>  b<=0;
      end case;
   end process;
end;
```

```
Architecture rtl of ex1 is
begin
 p1:process(a)
 begin
   case a is
      when "00" =>      q1<='1';
                        q2<='0';
                        q3<='0';

      when "10" =>      q1<='0';
                        q2<='1';
                        q3<='1';

      when others =>    q1<='0';
                        q2<='0';
                        q3<='1';
   end case;
 end process;
end;
```

Null statement

- Η εντολή null σημαίνει “δεν κάνω τίποτε”.
- Μπορεί για παράδειγμα να χρησιμοποιηθεί στην εντολή case όταν για κάποια επιλογή της δεν πρέπει να γίνει τίποτε.
- Η εντολή null μπορεί να παραληφθεί και χρησιμοποιείται επειδή αυξάνει την κατανόηση του κώδικα.
- Παράδειγμα
  - **case** a **is**

    when “00” => q1 <= ‘1’;

    when “01” => q2 <= ‘1’;

    q3 <= ‘1’;

    when others => null;

    end case;

Wait statement

- Υπάρχουν τέσσερις τρόποι για να περιγράψουμε την εντολή wait σε μία process

  process(a,b)

  wait until a=‘1’;

  wait on a,b;

  wait for 10ns;

- **Κανόνας** :
  Όλες οι processes εκτελούνται στο ξεκίνημα **αυτόματα**
  μέχρι να φτάσουν στο πρώτο wait.

Wait statement

- Παραδείγματα

Example 1

```
process(a)
begin
   c1<=not a;
end process;
```

Example 2

```
process
begin
  c2<=not a;
  wait on a;
end process;
```

Example 3

```
process
begin
  wait on a;
  c3<=not a;
end process;
```

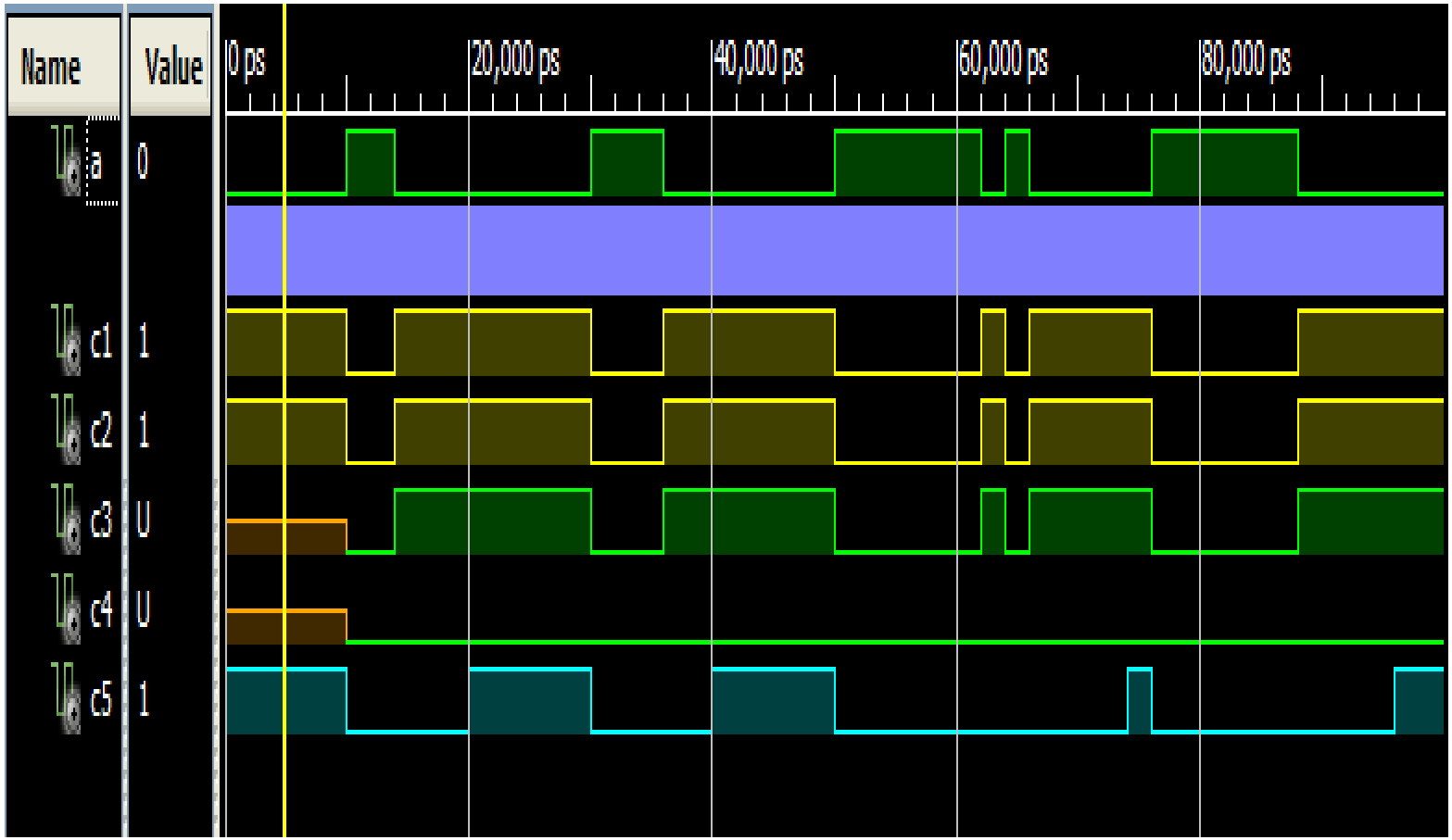
Example 4

```
process
begin
   wait until  a='1';
   c4<=not a;
end process;
```

Example 5

```
process
begin
  c5<=not a;
  wait until a='1' for 10 ns;
end process;
```

Wait statement



For loop statement

- Η εντολή for-loop είναι χρήσιμη όταν σχεδιάζουμε και χρησιμοποιούμε vectors.

Syntax:

[loop_label:] **for** <identifier> **in** <discrete_range> **loop**

  sequence_of_statement

end loop [loop_label];

- Μπορούμε να παραλείψουμε το range και να δημιουργήσουμε ένα άπειρο loop. Μπορούμε να βγούμε από ένα loop με την εντολή **exit-when**.

Παράδειγμα for loop - VHDL

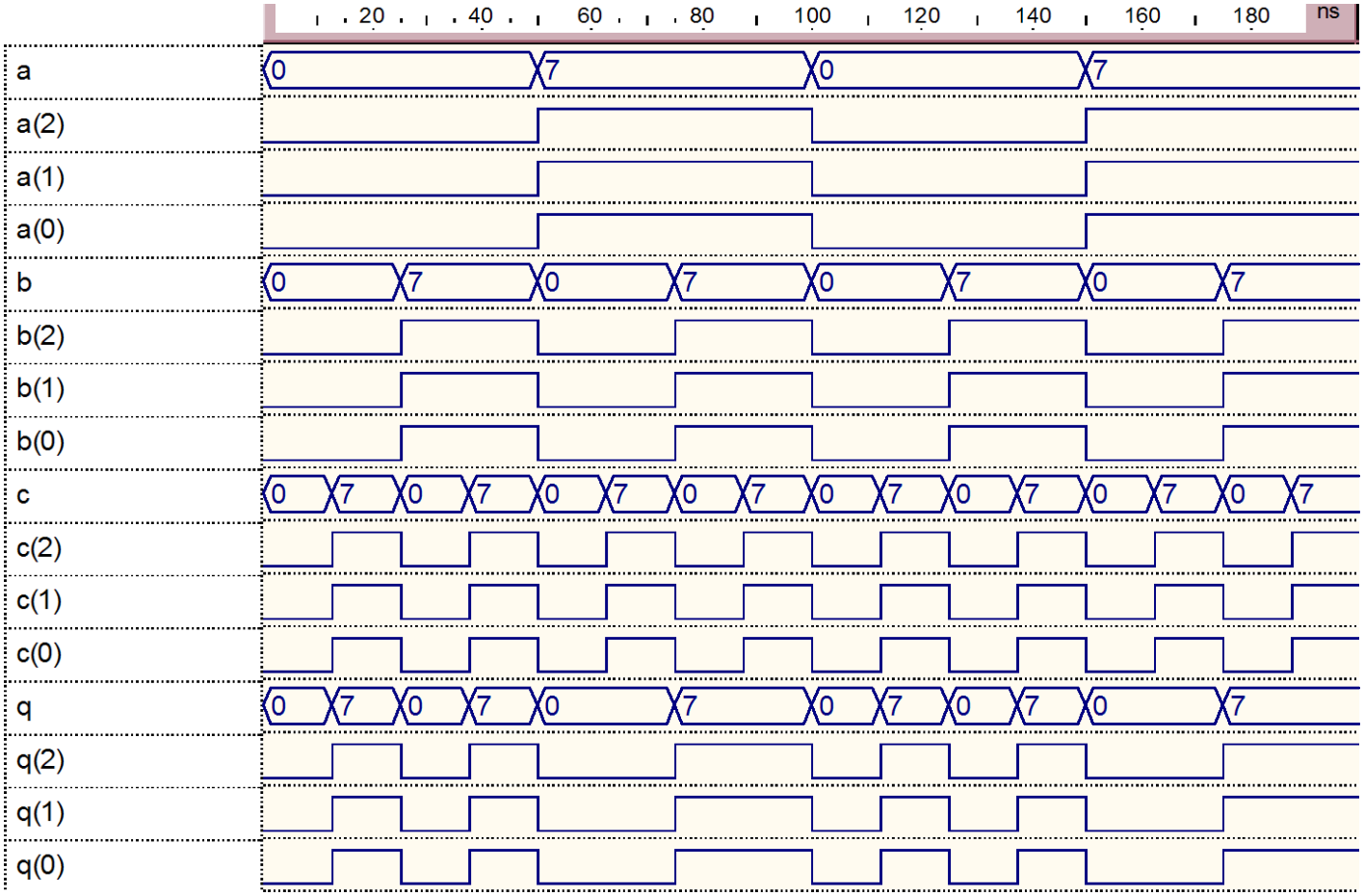
```
--  File: c:\my designs\ex_for_loop\SRC\ex_for_loop.VHD
-- Example for-loop. Created by Design Wizard: 01/31/01 11:44:35
--
library IEEE;
use IEEE.std_logic_1164.all;

entity ex_for_loop is
   port ( a, b, c : in   STD_LOGIC_VECTOR (63 downto 0);
          q       : out STD_LOGIC_VECTOR (63 downto 0));
end ex_for_loop;
```

Παράδειγμα for loop – VHDL

```
architecture ex_for_loop_beh of ex_for_loop is
begin
    process(a,b,c)
    begin
        for i in 0 to 63 loop
            if a(i)='1' then
                q(i) <= b(i);
            else
                q(i) <= c(i);
            end if;
        end loop;
    end process;
end ex_for_loop_beh;
```

Προσομοίωση



While loop statement

- Η εντολή while-loop υποστηρίζεται από πολύ προχωρημένα εργαλεία σύνθεσης.

Syntax:

[loop_label:] **while** <condition> **loop**

  sequence_of_statement

end loop [loop_label];

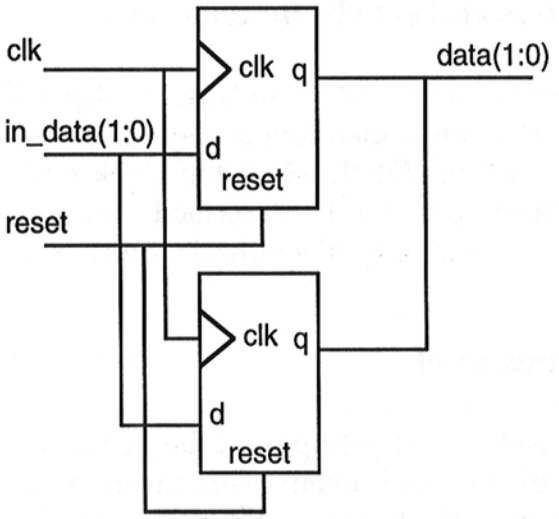
While loop statement

- Παράδειγμα

```
process(a,b,resetn)
variable i: integer range 0 to 31;
begin
  if resetn='0' then
    i:=0;
    q<=(others=>'0');
  else
    while q<12 loop
      if i=31 then
        exit;
      else
        i<=i+1;
        q<=a(i) + b(i) + q;
      end if;
    end loop;
  end if;
end process;
```

Flip-flop asynchronous reset

- Ενα flip-flop με asynchronous reset γίνεται reset μόλις το σήμα reset ενεργοποιηθεί και ανεξάρτητα από το ρολόϊ. Αυτό σημαίνει πως στην sensitivity list μιάς clocked process πρέπει να συμπεριληφθεί τόσο το reset όσο και το ρολόϊ.
- Παράδειγμα
  - **process**(clk, reset)
    begin
      if reset='1' **then**
        data <= "00";
      elseif clk'event **and** clk='1' **then**
        data <= **not** in_data;
      end if;
    end process;



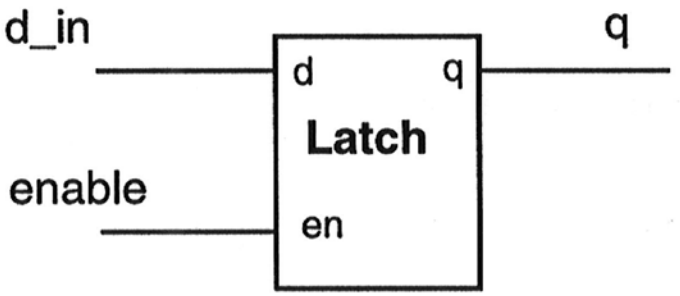
Flip-flop synchronous reset

- Ενα flip-flop με synchronous reset γίνεται reset στην αλλαγή του ρολογιού. Αυτό σημαίνει πως στην sensitivity list μιάς clocked process πρέπει να συμπεριληφθεί μόνο το ρολόϊ.
- Παράδειγμα
  - **process**(clk)
    begin
      if clk'event **and** clk='1' **then**
        if reset='1' **then**
          data <= "00";
        else
          data <= **not** in_data;
        end if;
      end if;
    end process;

Latches

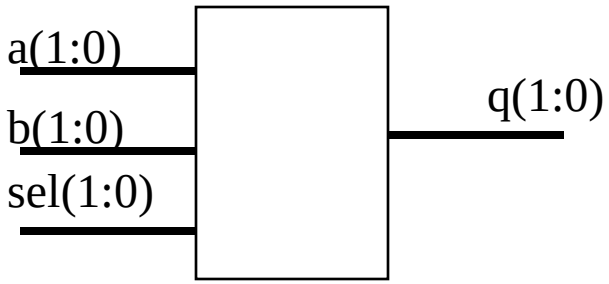
- Οι latches περιγράφονται στην VHDL με την εντολή process και πιο συγκεκριμένα με τις combinational processes.

- Παράδειγμα
  - **process** (enable, d_in)
    begin
      if enable=‘1’ **then**
        q <= d_in;
      end if;
    end process;



Ασκήσεις-Προβλήματα

1. Περιγράψτε με VHDL ένα positive edge-triggered D flip-flop με reset. Ελέγξτε την λειτουργία του με προσομοίωση.
2. Περιγράψτε με VHDL το παρακάτω εξάρτημα

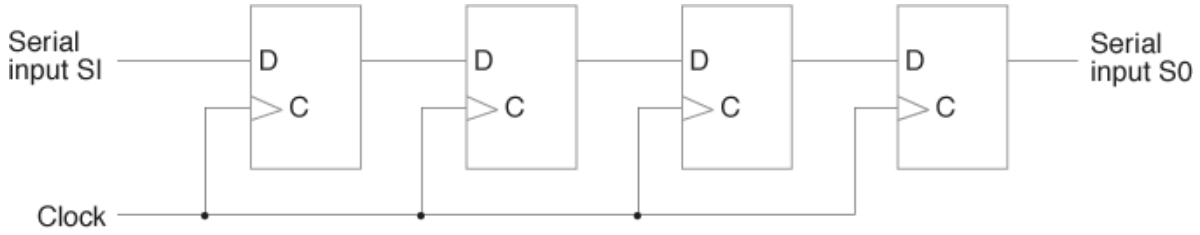


sel	q
00	a **xor** b
01	a **or** b
10	a **nor** b
11	a **and** b
others	“XX”

Χρησιμοποιώντας α) την εντολή **if**, β) την εντολή **case** και γ) την εντολή **when-else**. Ελέγξτε σε κάθε περίπτωση την λειτουργία του με προσομοίωση.

Ασκήσεις-Προβλήματα

3. Περιγράψτε με VHDL έναν 4-bit Shift Register ο οποίος εικονίζεται στο παρακάτω διάγραμμα. Ελέγξτε την λειτουργία του με προσομοίωση

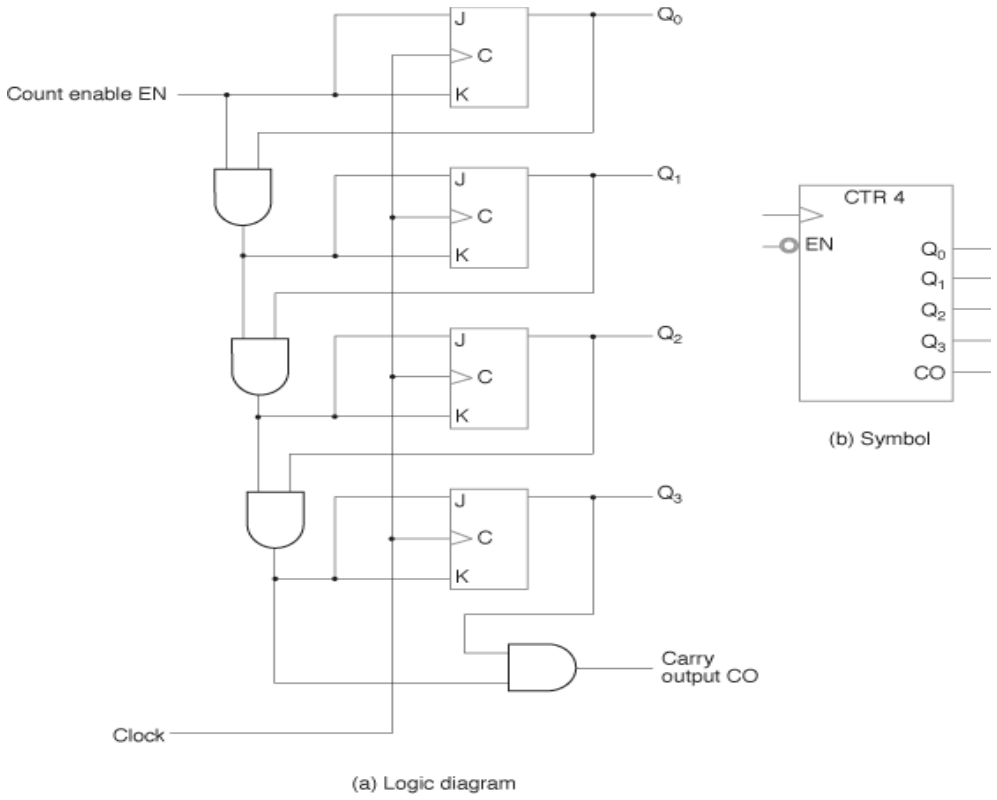


(a) Logic diagram

(b) Symbol

Ασκήσεις-Προβλήματα

4. Περιγράψτε με VHDL έναν 4-bit Binary Counter with reset ο οποίος εικονίζεται στο παρακάτω διάγραμμα. Ελέγξτε την λειτουργία του με προσομοίωση.

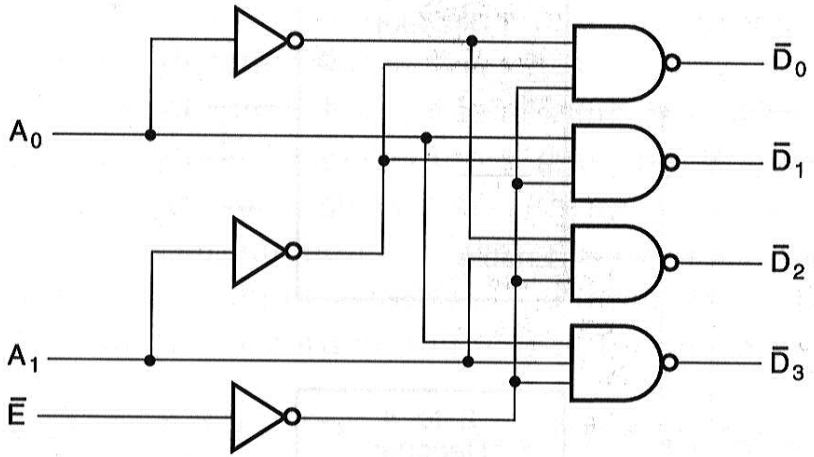


(a) Logic diagram

(b) Symbol

Ασκήσεις-Προβλήματα

5. Περιγράψτε με VHDL έναν 2-to-4 line decoder ο οποίος εικονίζεται στο παρακάτω διάγραμμα. Ελέγξτε την λειτουργία του με προσομοίωση.



(a) Logic diagram

$\bar{E}$	A_1	A_0	$\bar{D}_0$	$\bar{D}_1$	$\bar{D}_2$	$\bar{D}_3$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

(b) Truth table

$$\bar{D}_0 = \overline{E\ \bar{A}_1\ \bar{A}_0}$$
$$\bar{D}_1 = \overline{E\ \bar{A}_1\ A_0}$$
$$\bar{D}_2 = \overline{E\ A_1\ \bar{A}_0}$$
$$\bar{D}_3 = \overline{E\ A_1\ A_0}$$

(c) Logic Equations