dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

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

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

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

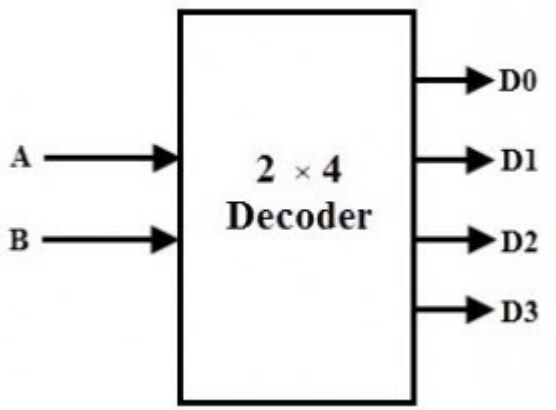
VHDL – Κωδικοποίηση

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

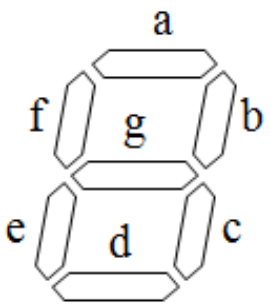
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0);
        output : out STD_LOGIC_VECTOR(3 downto 0));
end decoder_2to4;

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;
```

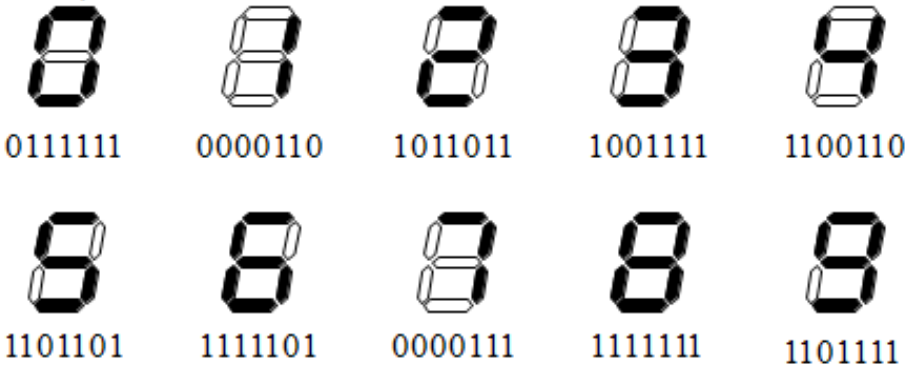
```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decoder_2to4 is
Port (
        input : in STD_LOGIC_VECTOR(1 downto 0); -- 2-bit input
        output : out STD_LOGIC_VECTOR(3 downto 0) -- 4-bit
output);
end decoder_2to4;

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)

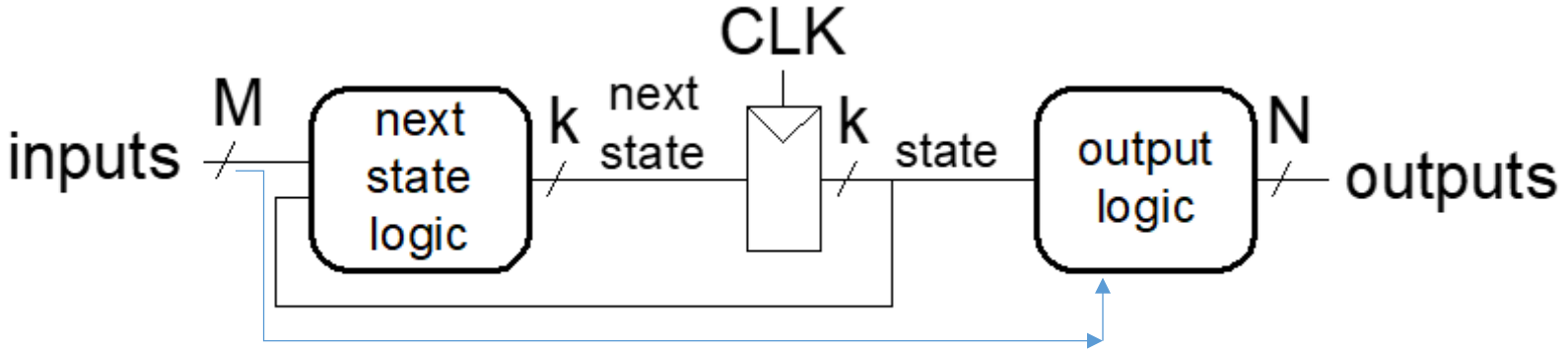
```
library ieee; use ieee.std_logic_1164.all;
entity seven_seg_decoder is
  port ( bcd   : in std_logic_vector (3 downto 0);
         blank : in std_logic;
         seg   : out std_logic_vector (7 downto 1) );
end entity seven_seg_decoder;
```

```
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 – 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)

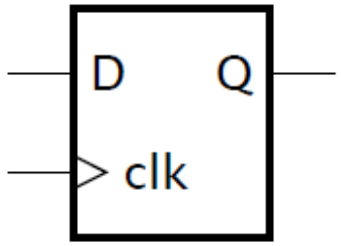
VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

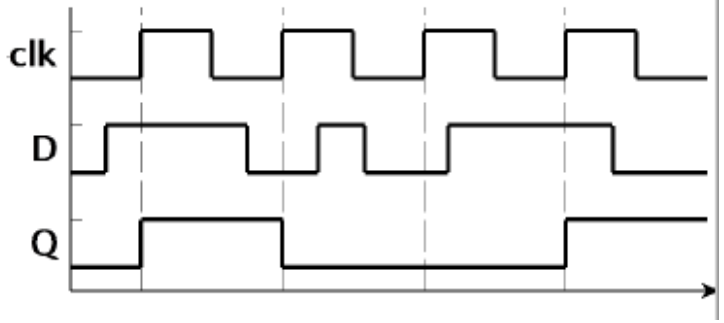
D Flip Flop

- 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





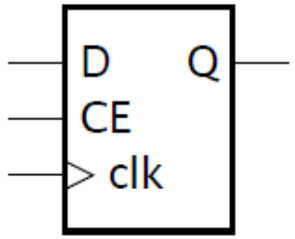
VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

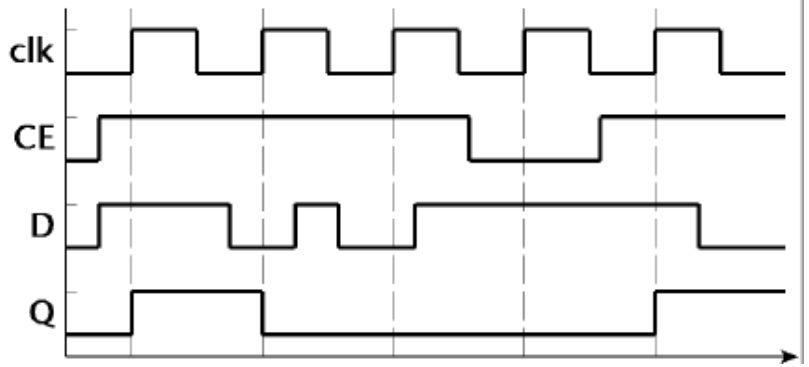
D Flip Flop with Enable

- 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





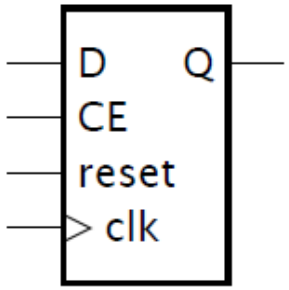
VHDL – Σύγχρονα Ακολουθιακά Κυκλώματα

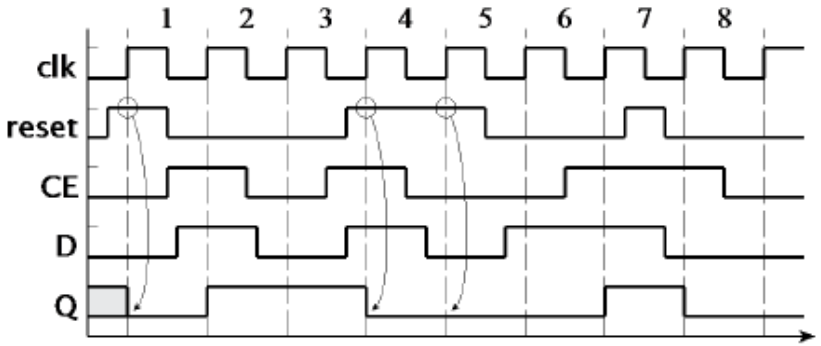
D Flip Flop with Reset (σύγχρονο)

- 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



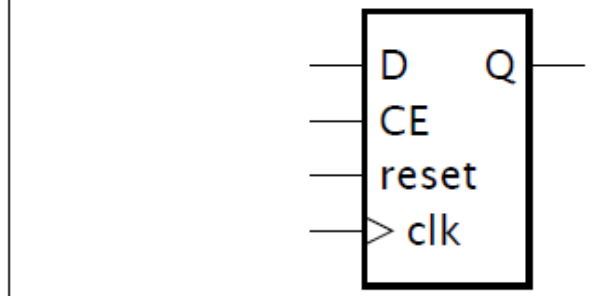


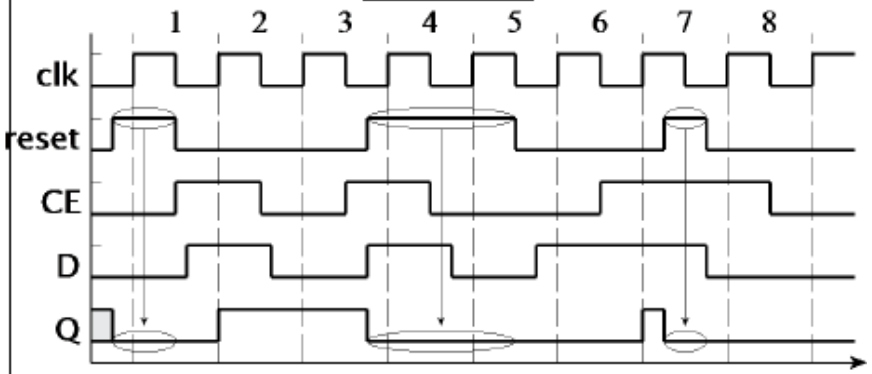
VHDL – Ασύγχρονα Ακολουθιακά Κυκλώματα

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;
```



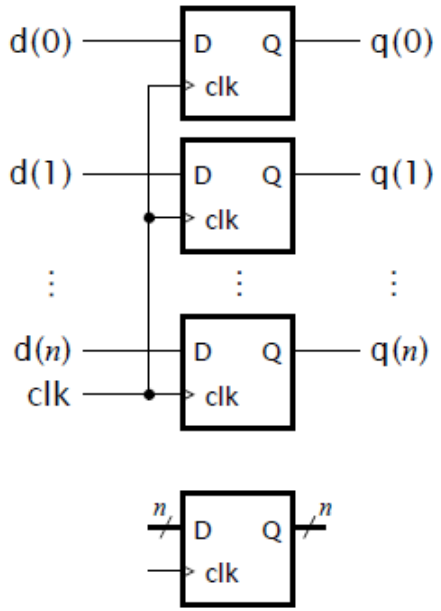


VHDL – Ασύγχρονα Ακολουθιακά Κυκλώματα

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;
```

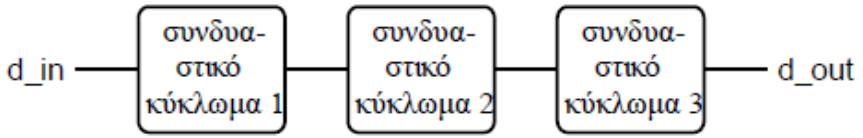
Συντομογραφία του όλα στο 0

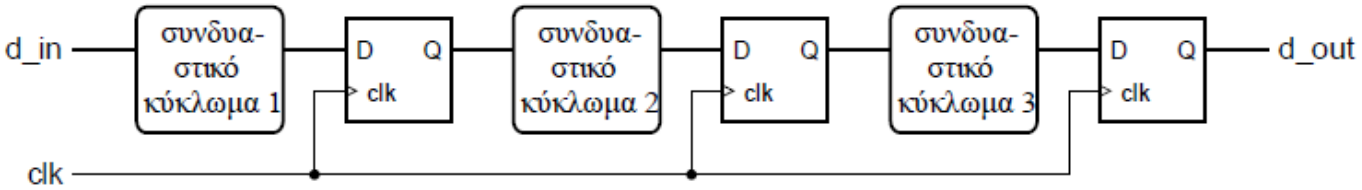
VHDL – Ακολουθιακά Κυκλώματα

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

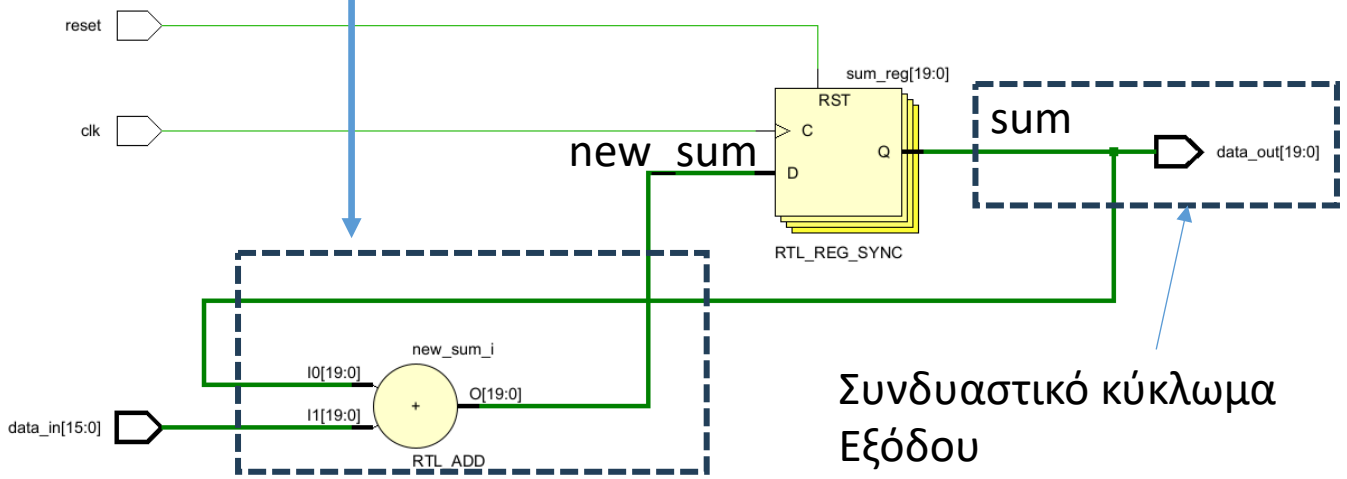
Συσσωρευτής (accumulator 2/2)

```
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 1/3)

- 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;
```

Χρήση bit/bit_vector
Προφανώς το std_logic/std_logic_vector είναι προτιμότερο

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 με ασύγχρονο reset

```
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

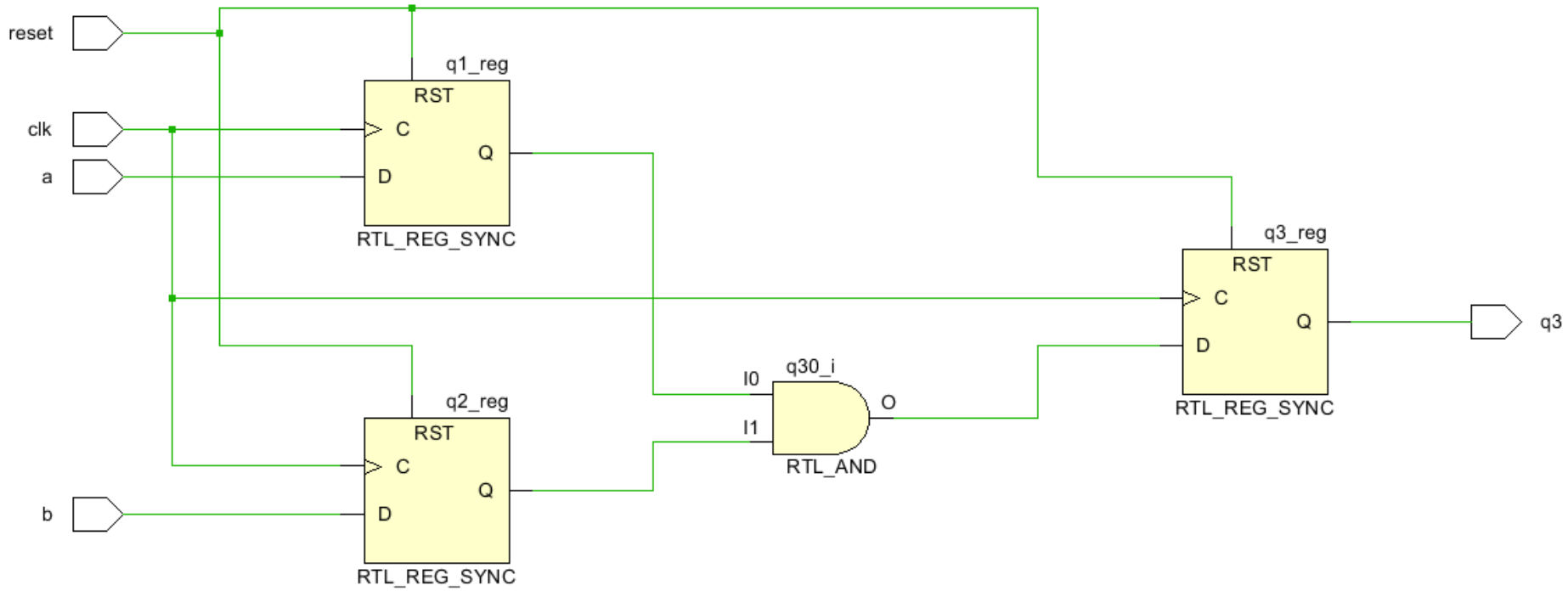
Τι κύκλωμα μοντελοποιεί ο κώδικας;

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

VHDL – Ακολουθιακά Κυκλώματα

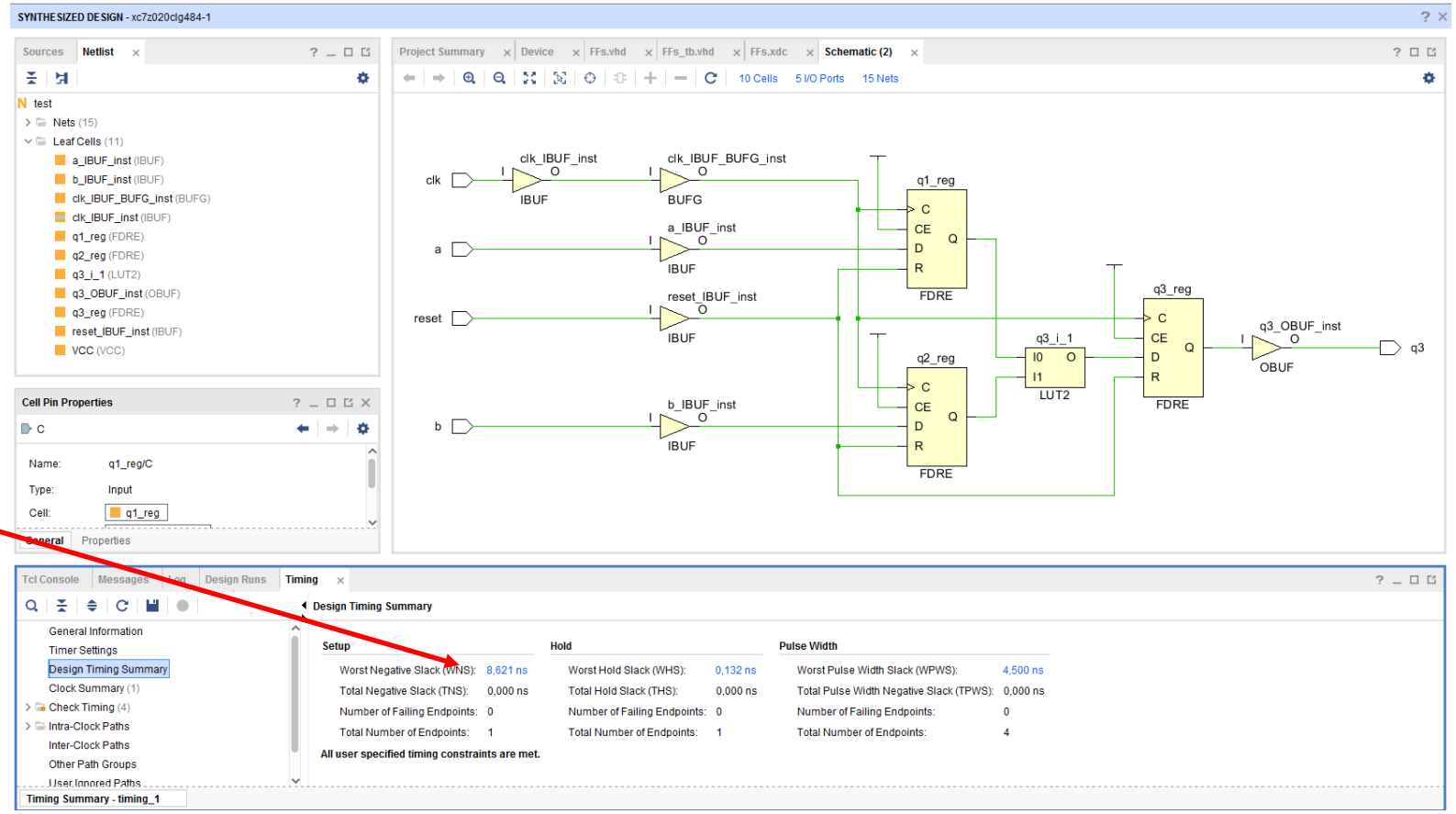
Παράδειγμα



VHDL – Ακολουθιακά Κυκλώματα

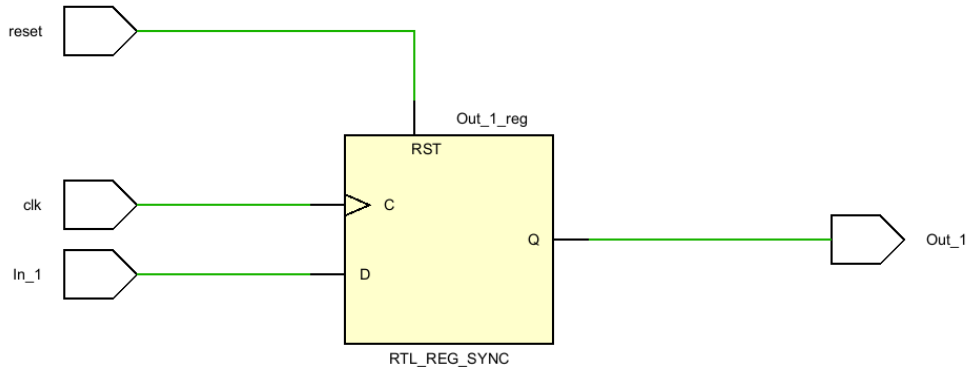
Παράδειγμα

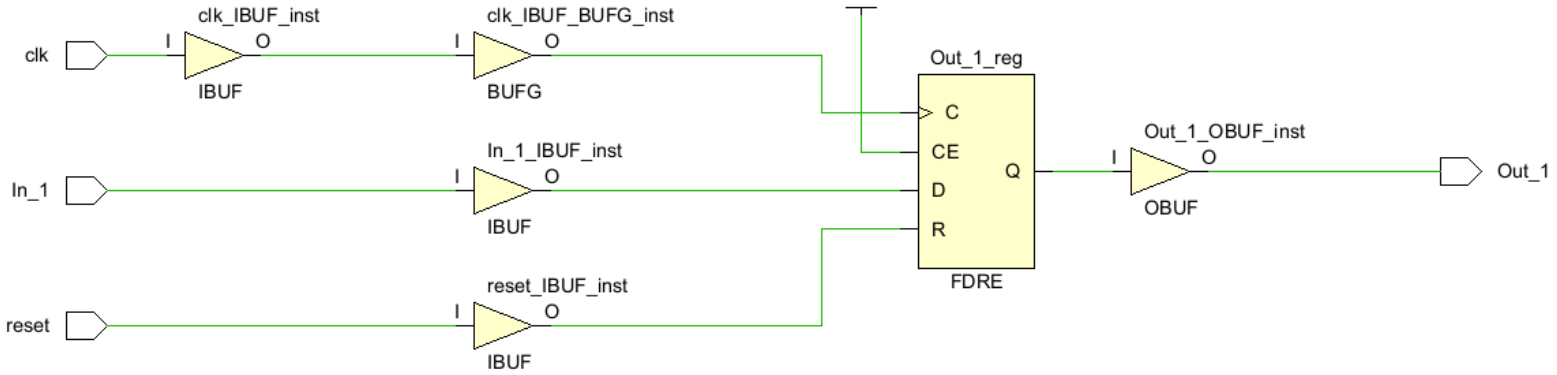
Tc=10ns-Slack



VHDL – Variables

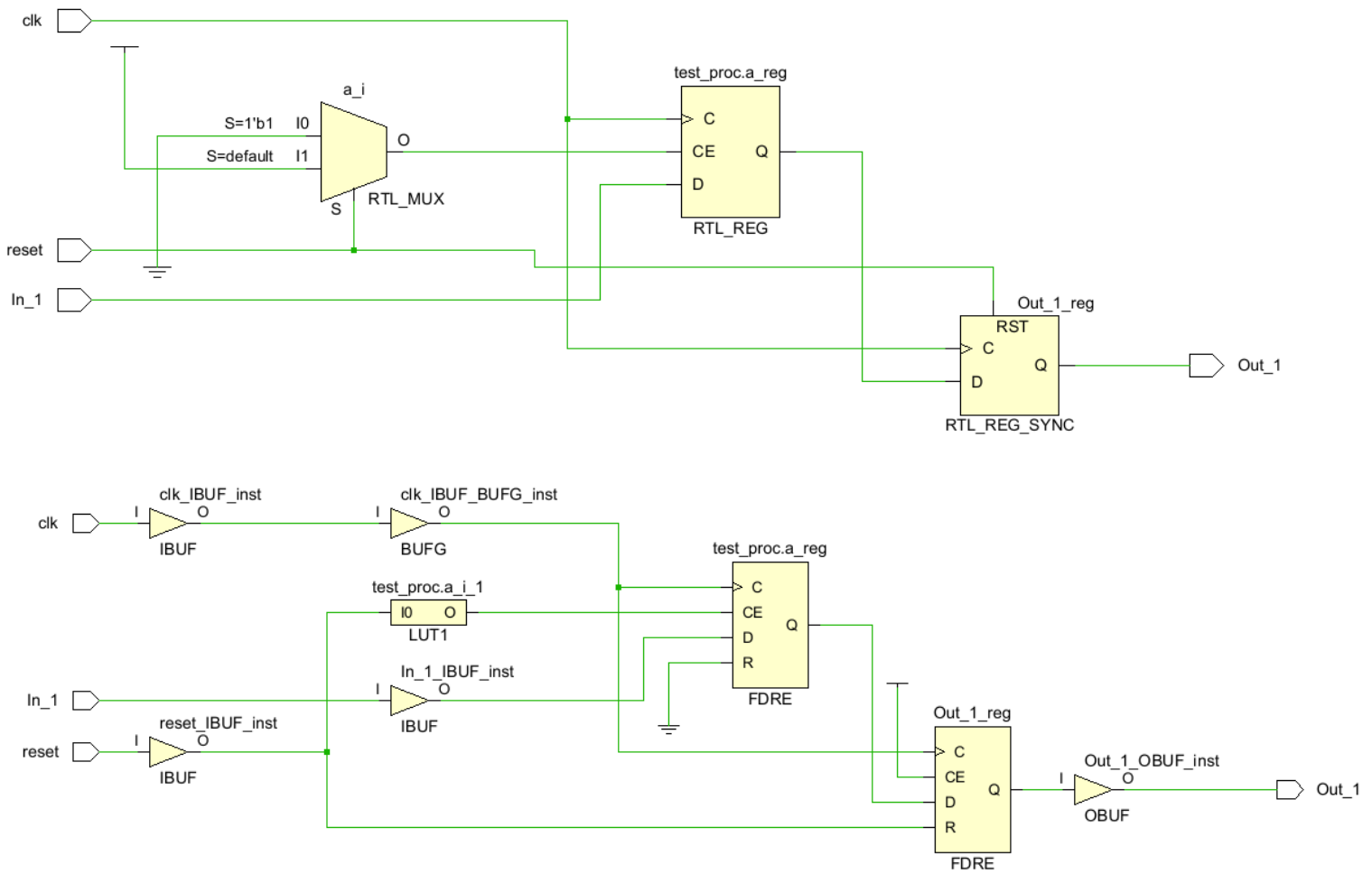
```
test_proc: process (clk) is

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
- 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.