Σχεδίαση Ψηφιακών Συστημάτων

Τμήμα Ψηφιακών Συστημάτων,
Πανεπιστήμιο Πελοποννήσου
Θάνος Παπαδημητρίου

Ακολουθιακά κυκλώματα

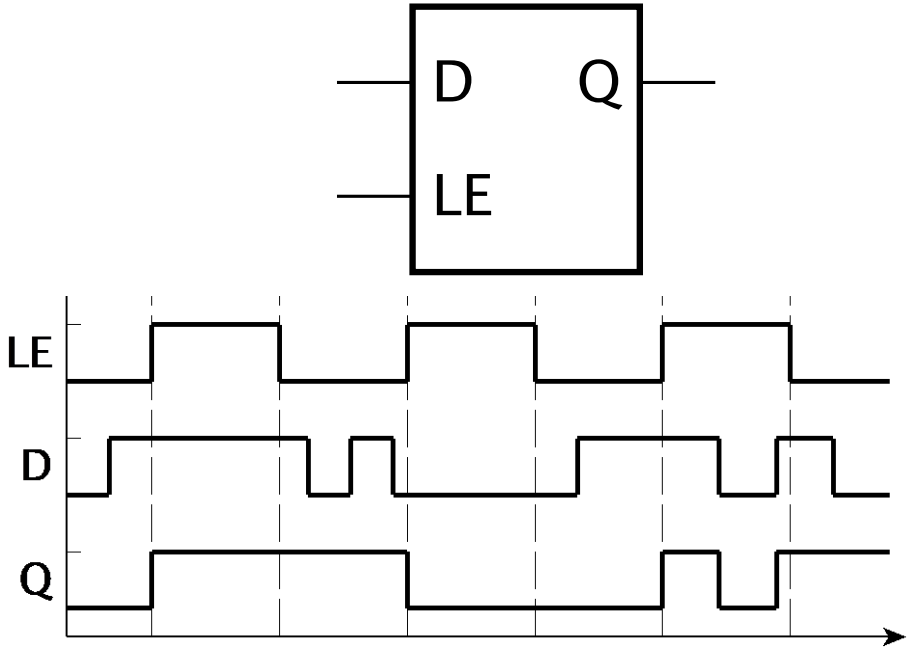
- Ακολουθιακά κυκλώματα
  - Οι έξοδοι εξαρτώνται από τις τρέχουσες και από τις προηγούμενες εισόδους
  - Αποθήκευση *κατάστασης (state)* : μια αφαίρεση του ιστορικού των εισόδων
- Συνήθως, ελέγχεται από ένα περιοδικό σήμα ρολογιού (clock)

Σχεδίαση ακολουθιακών κυκλωμάτων

- Μανδαλωτές
- Flip-flop, καταχωρητές
- Μετρητές
- Καταχωρητές ολίσθησης
- Μηχανές πεπερασμένων καταστάσεων

Μοντέλο μανταλωτή (latch)

```
entity latch is
  port (D, LE : in bit;
        Q     : out bit);
end entity latch;

architecture beh of latch is
begin

  p: process (D,LE)
  begin
    if LE = '1' then
      Q <= D;
    end if;
  end process;

end architecture beh;
```



Μανδαλωτές στην VHDL

- Η συμπεριφορά μανδάλωσης οφείλεται συνήθως σε σφάλμα!

```
mux_block : process (sel, a1, b1, a2, b2) is
begin
  if sel = '0' then
    z1 <= a1; z2 <= b1;
  else
    z1 <= a2; z3 <= b2;
  end if;
end process mux_block;
```

Οοοπ!
Θα έπρεπε να είναι
z2 <= ...

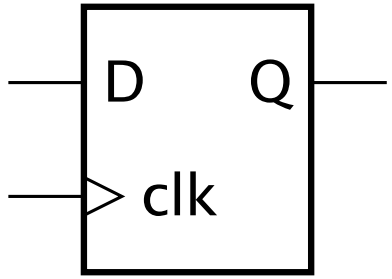
  - Οι τιμές πρέπει να αποθηκευτούν
    - για το z2 όταν το sel = '1'
    - για το z3 όταν το sel = '0'

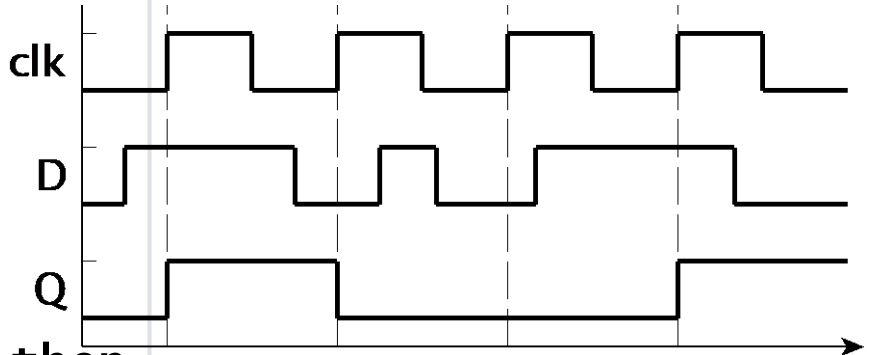
Μοντελοποίηση καταχωρητών στην VHDL

- Η περιγραφή ενός καταχωρητή στην VHDL γίνεται μόνο με τη χρήση διεργασίας (process)
- Δεν ορίζεται δομή της VHDL που να αντιστοιχίζεται στο υλικό σαν καταχωρητής
- Υπάρχουν πολλοί τρόποι για να περιγράψεις τη συμπεριφορά ενός καταχωρητή στη VHDL (simulation model)
- Δεν είναι όλες οι περιγραφές συνθέσιμες

Μοντέλο καταχωρητή (D flip-flop)

```
entity dff is
  port ( clk, d : in bit;
         q      : out bit);
end entity;

architecture bef of dff is
begin

  process (clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;

end architecture;
```





Ανοδική και καθοδική ακμή του ρολογιού

```
process (clk)
begin
  if clk'event and clk = '1' then
     q <= d;
  end if;
end process;
```

rising_edge(clk)

Ακμοπυροδότητα D flip-flop στην ανοδική μετάβαση

(rising edge-triggered)

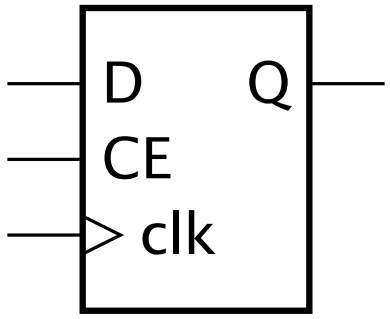
```
process (clk)
begin
  if clk'event and clk = '0' then
     q <= d;
  end if;
end process;
```

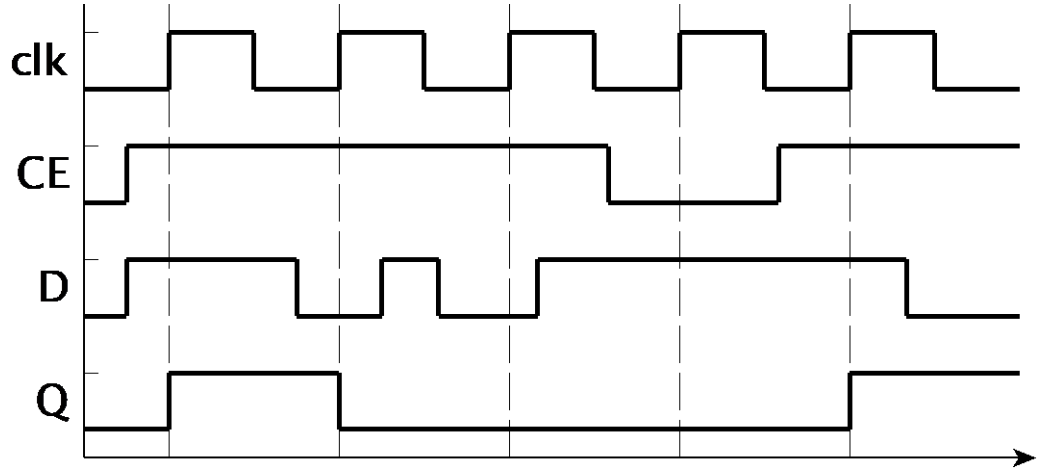
falling_edge(clk)

Ακμοπυροδότητα D flip-flop στην καθοδική μετάβαση

(falling edge-triggered)

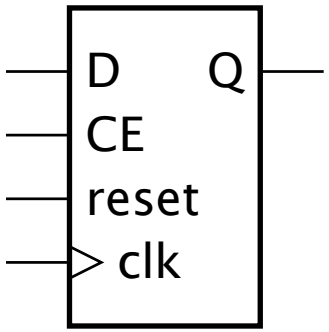
D-FF με επίτρεψη (clock enable)

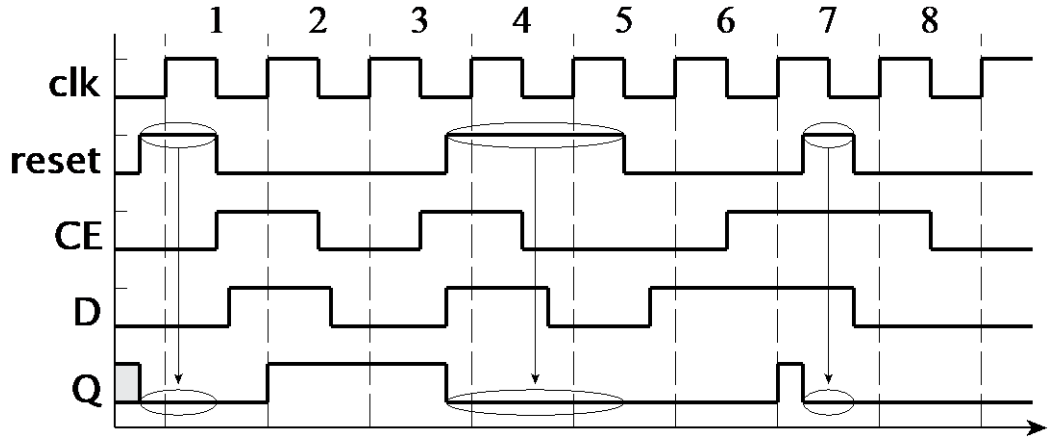





```
process (clk)
begin
  if clk'event and clk = '1' then
    if ce = '1' then
      q <= d;
    end if;
  end if;
end process;
```

D-FF με είσοδο μηδενισμού (reset)

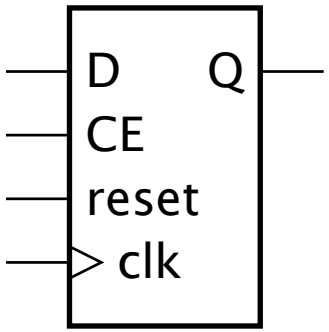


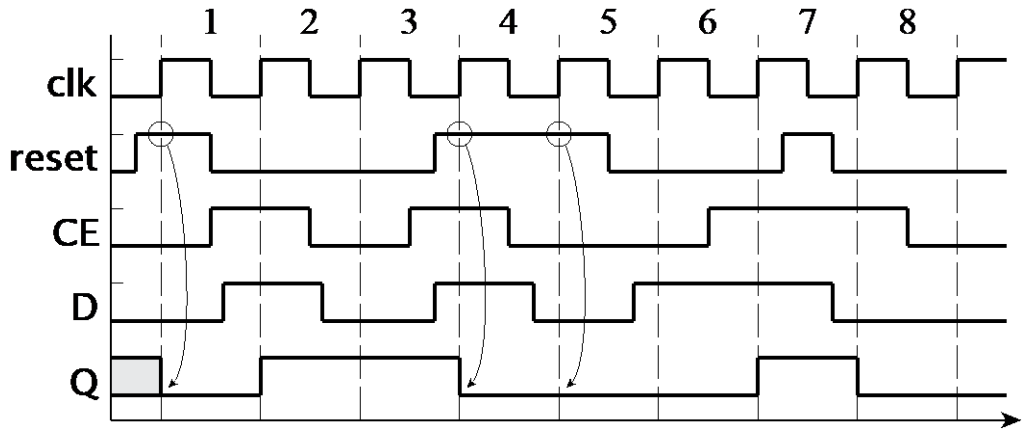



```
process (clk, reset) is
begin
  if reset = '1' then
     q <= '0';
  elsif clk'event and clk = '1' then
    if ce = '1' then
      q <= d;
    end if;
  end if;
end process;
```

D flip-flop με ασύγχρονη είσοδο μηδενισμού (asynchronous reset)

D-FF με είσοδο μηδενισμού (reset)






```
process (clk) is
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;
```

D flip-flop με σύγχρονη είσοδο μηδενισμού (synchronous reset)

Σχεδίαση ακολουθιακής λογικής

- Τι κύκλωμα μοντελοποιεί ο VHDL κώδικας;

```
process (clk)
begin
  if clk'event and clk = '1' then
    q1 <= a and b;
    q2 <= c or d;
  end if;
end process;
```

Σχεδίαση ακολουθιακής λογικής

- Τι κύκλωμα μοντελοποιεί ο VHDL κώδικας;
- Πώς υπολογίζεται η συχνότητα λειτουργίας του κυκλώματος μετά τη σύνθεση;

```
process (clk)
begin
  if clk'event and clk = '1' then
    q1 <= a;
    q2 <= b;
    q3 <= q1 and q2;
  end if;
end process;
```

Σχεδίαση ακολουθιακής λογικής

- Τι κύκλωμα μοντελοποιεί ο VHDL κώδικας;

```
entity dreg8 is
  port ( clk : in bit;
         d   : in bit_vector(7 downto 0);
         q   : out bit_vector(7 downto 0));
end entity;

architecture bef of dreg8 is
begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end architecture;
```

Σχεδίαση ακολουθιακής λογικής

- Τι διαφορά έχουν οι παρακάτω κώδικες;

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

```
process (clk, reset)
begin
  if reset = '1' then
    q <= "00001111";
  elsif clk'event and clk = '1' then
    q <= d;
  end if;
end process;
```

Παράδειγμα: Συσσωρευτής

- 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;
```

Παράδειγμα: Συσσωρευτής

```
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;
```

Μετρητές (counters)

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

entity counter4 is
   port (clk, reset : in std_logic;
         count : out unsigned(3 downto 0);
end entity;

architecture bef of counter4 is
   signal counter : unsigned(3 downto 0);
begin
   count <= 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;
```

4-bit σύγχρονος δυαδικός μετρητής με ασύγχρονη μηδένιση

(4-bit binary synchronous counter with asynchronous reset)

Παραδείγματα σχεδίασης μετρητών

- Υλοποιήστε τους ακόλουθους τύπους μετρητών:
  - BCD μετρητές
  - με σύγχρονη είσοδο μηδένισης
  - με είσοδο επίτρεψης
  - με παράλληλη φόρτωση
- Υλοποιήστε διαιρέτες συχνότητας (frequency divider) με τη χρήση μετρητών

Καταχωρητές ολίσθησης (shift registers)

- Καταχωρητής ολίσθησης των 8-bit με ασύγχρονη είσοδο μηδένισης (8-bit shift register with asynchronous reset input)

```
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;
```

Σχεδίαση καταχωρητών ολίσθησης (1)

- Υλοποίηση με τμήματα πινάκων (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;
```

Σχεδίαση καταχωρητών ολίσθησης (2)

- Υλοποίηση με συνένωση πινάκων (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;
```

Σχεδίαση καταχωρητών ολίσθησης (3)

- Υλοποίηση με βρόχο for (for loop)

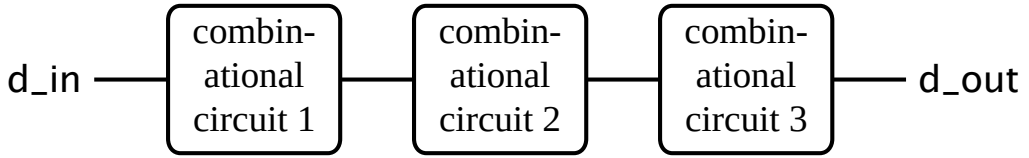
```
process (clk, reset)
begin

  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;
```

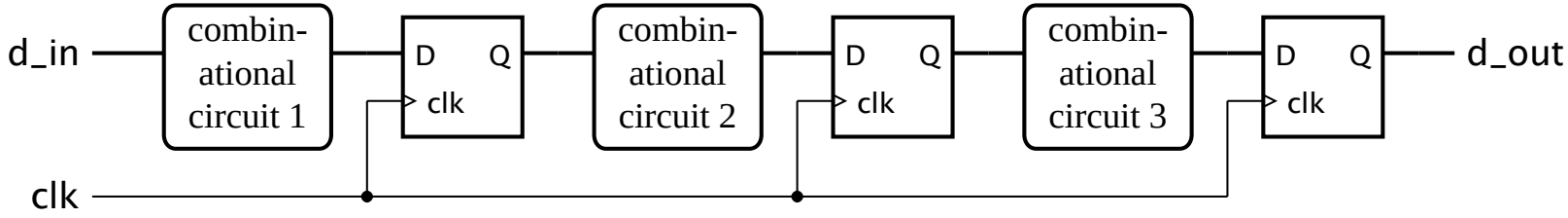
Παραδείγματα σχεδίασης καταχωρητών ολίσθησης

- Υλοποιήστε τους ακόλουθους τύπους καταχωρητών ολίσθησης:
  - με σύγχρονη είσοδο μηδένισης
  - με είσοδο επίτρεψης
  - με παράλληλη φόρτωση
  - με επιλογή αριστερής & δεξιάς ολίσθησης

Μηχανισμός διοχέτευσης (pipeline)

- Συνολική καθυστέρηση = Delay1 + Delay2 + Delay3
- Διάστημα μεταξύ δεδομένων > Συνολική καθυστέρηση





- Περίοδος ρολογιού = max(Delay1, Delay2, Delay3)
- Συνολική καθυστέρηση = 3 × περίοδος ρολογιού
- Διάστημα μεταξύ δεδομένων = 1 περίοδος ρολογιού

Παράδειγμα διοχέτευσης

- Υπολογισμός του μέσου όρου τριών ροών δεδομένων
- Νέα δεδομένα σε κάθε ακμή του ρολογιού

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

entity average_pipeline is
  port ( clk      : in std_logic;
         a, b, c : in integer;
         avg      : out integer);
end entity average_pipeline;
```

Παράδειγμα διοχέτευσης (συν.)

```
architecture rtl of average_pipeline is
  signal a_plus_b, sum, sum_div_3 : integer;
  signal saved_a_plus_b,
         saved_c, saved_sum : integer;
begin
  a_plus_b <= a + b;
  reg1 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_a_plus_b <= a_plus_b;
      saved_c <= c;
    end if;
  end process reg1;
...
```

Παράδειγμα διοχέτευσης (συν.)

```
  sum <= saved_a_plus_b + saved_c;

  reg2 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_sum <= sum;
    end if;
  end process reg2;

  sum_div_3 <= saved_sum/3;

  reg3 : process (clk) is
  begin
    if rising_edge(clk) then
      avg <= sum_div_3;
    end if;
  end process reg3;
end architecture average_pipeline;
```

Διαδρομές Δεδομένων και Έλεγχος

- Τα ψηφιακά συστήματα εκτελούν ακολουθίες λειτουργιών σε κωδικοποιημένα δεδομένα
- *Διαδρομή Δεδομένων (Datapath)*
  - Συνδυαστικά κυκλώματα για τις λειτουργίες
  - Καταχωρητές για την αποθήκευση των ενδιάμεσων αποτελεσμάτων
- *Τμήμα Ελέγχου (Control)*: ακολουθία ελέγχου
  - Παράγει *σήματα ελέγχου (control signals)*
    - Επιλέγει τις λειτουργίες που θα εκτελεστούν
    - Ενεργοποιεί τους καταχωρητές στις σωστές στιγμές
  - Χρησιμοποιεί *σήματα κατάστασης (status signals)* από τη διαδρομή δεδομένων

Παράδειγμα: Μιγαδικός Πολλαπλασιαστής

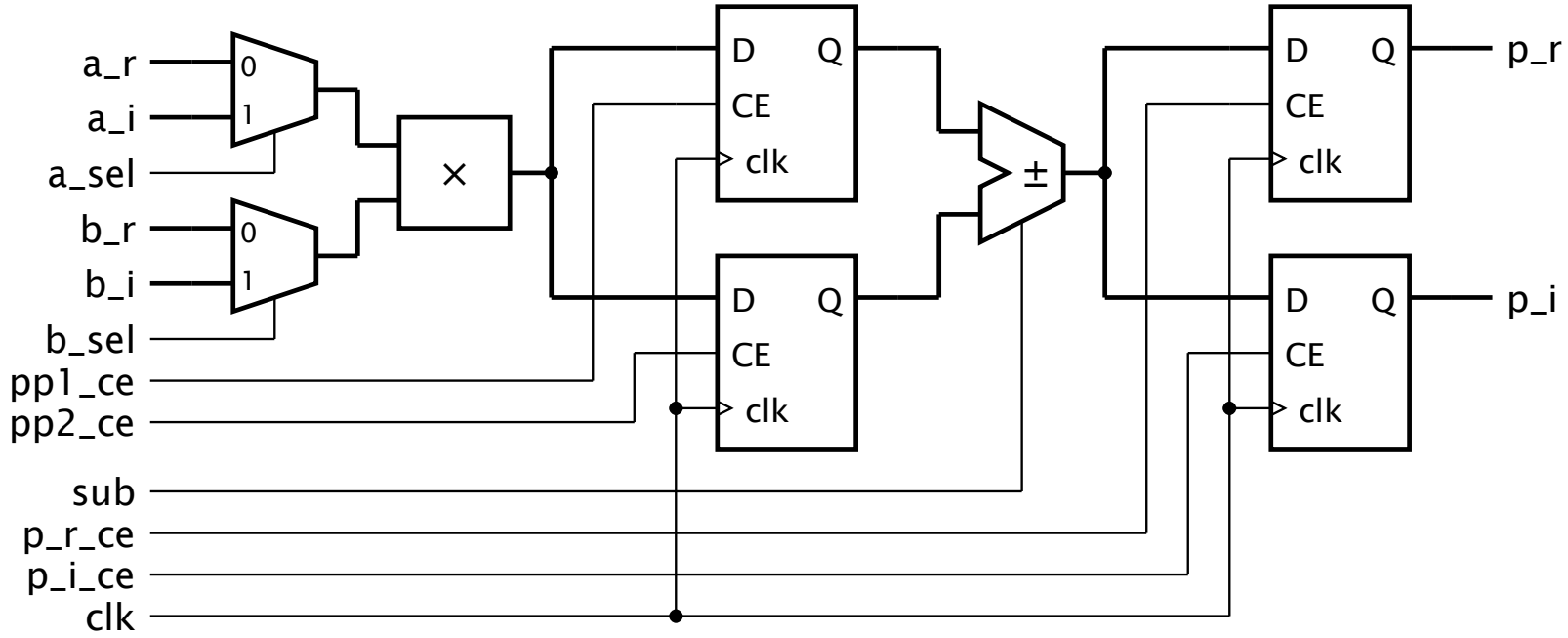
- Καρτεσιανή μορφή, σταθερή υποδιαστολή
  - τελεστέοι: 4 bit πριν, 12 μετά τη δυαδική υποδ.
  - αποτέλεσμα: 8 bit πριν, 24 μετά τη δυαδική υποδ.
- Με έμφαση στη μείωση της επιφάνειας (area)

$$a = a_r + ja_i \qquad b = b_r + jb_i$$

$$p = ab = p_r + jp_i = (a_r b_r - a_i b_i) + j(a_r b_i + a_i b_r)$$

  - 4 πολλαπλασιασμοί, 1 πρόσθεση, 1 αφαίρεση
    - Ακολουθιακή εκτέλεση χρησιμοποιώντας 1 πολλαπλασιαστή, 1 αθροιστή/αφαιρέτη

Διαδρομή Δεδομένων



Μιγαδικός Πολλαπλασιαστής στην VHDL

```
library ieee; use ieee.std_logic_1164.all, ieee.fixed_pkg.all;
entity multiplier is
  port ( clk, reset : in std_logic;
         input_rdy : in std_logic;
         a_r, a_i, b_r, b_i : in sfixed(3 downto -12);
         p_r, p_i : out sfixed(7 downto -24) );
end entity multiplier;
```

```
architecture rtl of multiplier is
  signal a_sel, b_sel, pp1_ce, pp2_ce,
         sub, p_r_ce, p_i_ce : std_logic; -- control signals
  signal a_operand, b_operand : sfixed(3 downto -12);
  signal pp, pp1, pp2, sum : sfixed(7 downto -24);
  ...
begin
```

Μιγαδικός Πολλαπλασιαστής στην VHDL

```
a_operand <= a_r when a_sel = '0' else a_i;  -- mux
b_operand <= b_r when b_sel = '0' else b_i;  -- mux

pp <= a_operand * b_operand;  -- multiplier

pp1_reg : process (clk) is  -- partial product register 1
begin
  if rising_edge(clk) then
    if pp1_ce = '1' then
      pp1 <= pp;
    end if;
  end if;
end process pp1_reg;

pp2_reg : process (clk) is -- partial product register 2
begin
  if rising_edge(clk) then
    if pp2_ce = '1' then
      pp2 <= pp;
    end if;
  end if;
end process pp2_reg;
```

Μιγαδικός Πολλαπλασιαστής στην VHDL

```
  sum <= pp1 + pp2 when sub = '0' else pp1 - pp2;  -- add/sub
  p_r_reg : process (clk) is  -- result real-part register
  begin
    if rising_edge(clk) then
      if p_r_ce = '1' then
        p_r <= sum;
      end if;
    end if;
  end process p_r_reg;
  p_i_reg : process (clk) is  -- result imag-part register
  begin
    if rising_edge(clk) then
      if p_i_ce = '1' then
        p_i <= sum;
      end if;
    end if;
  end process p_i_reg;
  ...  -- control circuit
end architecture rtl;
```

Ακολουθία Ελέγχου του Πολλαπλασιαστή

- Πρώτη υλοποίηση
  1. a_r * b_r → pp1_reg
  2. a_i * b_i → pp2_reg
  3. pp1 – pp2 → p_r_reg
  4. a_r * b_i → pp1_reg
  5. a_i * b_r → pp2_reg
  6. pp1 + pp2 → p_i_reg
- Αποφεύγει τη διένεξη των πόρων
- Διαρκεί 6 κύκλους ρολογιού

Ακολουθία Ελέγχου του Πολλαπλασιαστή

- Βελτιωμένη υλοποίηση
  1. a_r * b_r → pp1_reg
  2. a_i * b_i → pp2_reg
  3. pp1 – pp2 → p_r_reg
     a_r * b_i → pp1_reg
  4. a_i * b_r → pp2_reg
  5. pp1 + pp2 → p_i_reg
- Συγχωνεύει τα βήματα όπου δεν υπάρχει διένεξη των πόρων
- Διαρκεί 5 κύκλους ρολογιού

Σήματα Ελέγχου του Πολλαπλασιαστή

Βήμα	a_sel	b_sel	pp1_ce	pp2_ce	sub	p_r_ce	p_i_ce
1	0	0	1	0	–	0	0
2	1	1	0	1	–	0	0
3	0	1	1	0	1	1	0
4	1	0	0	1	–	0	0
5	–	–	0	0	0	0	1

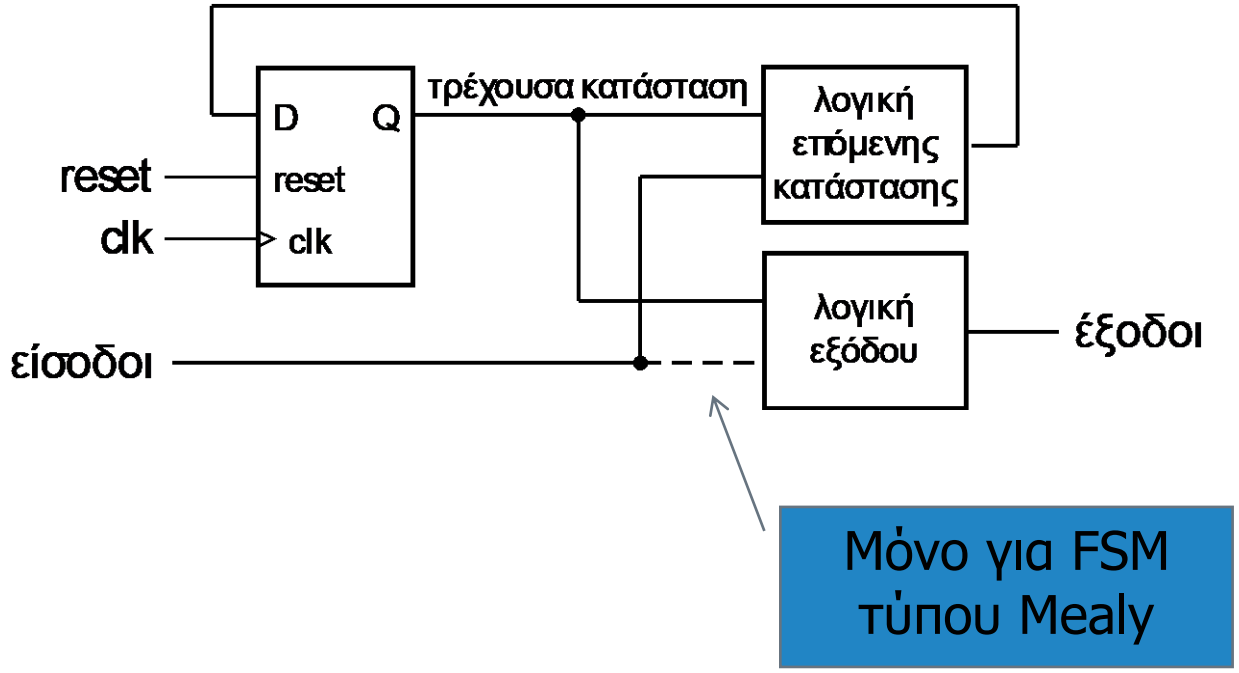
Μηχανές Πεπερασμένων Καταστάσεων

- Χρησιμοποιούνται για να υλοποιήσουν την ακολουθία ελέγχου

- Μία FSM (Finite-State Machine) ορίζεται από
  - σύνολο εισόδων: Σ
  - σύνολο εξόδων: Γ
  - σύνολο καταστάσεων: S
  - αρχική κατάσταση: $s_0 \in S$
  - συνάρτηση μετάβασης: $\delta: S \times \Sigma \rightarrow S$
  - συνάρτηση εξόδου: $\omega: S \times \Sigma \rightarrow \Gamma$ ή $\omega: S \rightarrow \Gamma$

Η FSM στο Υλικό

- FSM τύπου Mealy: ω: S × Σ → Γ
- FSM τύπου Moore: ω: S → Γ



Παράδειγμα FSM: Έλεγχος Πολλαπλασιαστή

- Μία κατάσταση ανά βήμα
- Ξεχωριστή κατάσταση αδράνειας (idle);
  - Αναμένει έως ότου input_rdy = '1'
  - Έπειτα, προχωράει στα βήματα 1, 2, ...
  - Αλλά αυτό σπαταλάει έναν κύκλο!
- Χρησιμοποιεί το βήμα 1 ως κατάσταση αδράνειας
  - Επαναλαμβάνει το βήμα 1 εάν input_rdy≠'1'
  - Διαφορετικά προχωράει στο βήμα 2
- Συνάρτηση εξόδου
  - Moore ή Mealy;

Συνάρτηση μετάβασης

current_state	input_rdy	next_state
step1	0	step1
step1	1	step2
step2	–	step3
step3	–	step4
step4	–	step5
step5	–	step1

Κωδικοποίηση Καταστάσεων

- Κωδικοποιούνται στο δυαδικό
  - N καταστάσεις: χρειάζονται τουλάχιστον $\lceil \log_2 N \rceil$ bit
- Η κωδικοποιημένη τιμή χρησιμοποιείται στα κυκλώματα για τις συναρτήσεις μετάβασης και εξόδου
  - η κωδικοποίηση επηρεάζει την πολυπλοκότητα του κυκλώματος
- Είναι δύσκολο να βρεθεί βέλτιστη κωδικοποίηση
  - τα εργαλεία CAD μπορούν να κάνουν αυτή τη δουλειά καλά
- Συχνά χρησιμοποιείται το 000…0 για την κατάσταση αδράνειας
  - μηδενίζει τον καταχωρητή στην κατάσταση αδράνειας

FSM στην VHDL

- Χρησιμοποιούμε έναν τύπο απαρίθμησης για τις τιμές των καταστάσεων
  - αφηρημένο, έτσι αποφεύγουμε να προδιαγράψουμε την κωδικοποίηση

```
type multiplier_state is (step1, step2, step3, step4, step5);
signal current_state, next_state : multiplier_state;
...
```

Έλεγχος Πολλαπλασιαστή στην VHDL

```
state_reg : process (clk, reset) is
begin
  if reset = '1' then
    current_state <= step1;
  elsif rising_edge(clk) then
    current_state <= next_state;
  end if;
end process state_reg;
```

```
next_state_logic : process
  (current_state, input_rdy) is
begin
  case current_state is
    when step1 =>
      if input_rdy = '0' then
        next_state <= step1;
      else
        next_state <= step2;
      end if;
    when step2 =>
      next_state <= step3;
    when step3 =>
      next_state <= step4;
    when step4 =>
      next_state <= step5;
    when step5 =>
      next_state <= step1;
  end case;
end process next_state_logic;
```

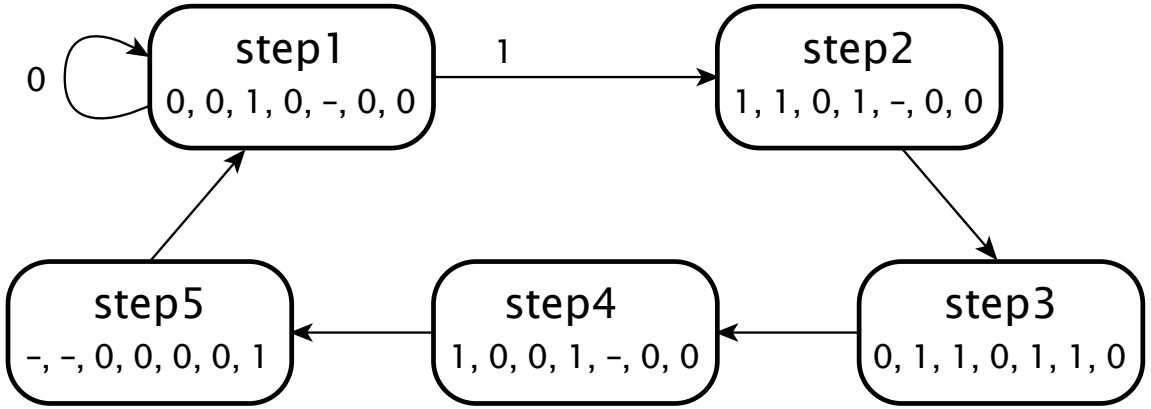
Έλεγχος Πολλαπλασιαστή στην VHDL

FSM τύπου Moore

```
output_logic : process (current_state) is
begin
  case current_state is
    when step1 =>
      a_sel <= '0'; b_sel <= '0'; pp1_ce <= '1'; pp2_ce <= '0';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step2 =>
      a_sel <= '1'; b_sel <= '1'; pp1_ce <= '0'; pp2_ce <= '1';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step3 =>
      a_sel <= '0'; b_sel <= '1'; pp1_ce <= '1'; pp2_ce <= '0';
      sub <= '1'; p_r_ce <= '1'; p_i_ce <= '0';
    when step4 =>
      a_sel <= '1'; b_sel <= '0'; pp1_ce <= '0'; pp2_ce <= '1';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step5 =>
      a_sel <= '0'; b_sel <= '0'; pp1_ce <= '0'; pp2_ce <= '0';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '1';
  end case;
end process output_logic;
```

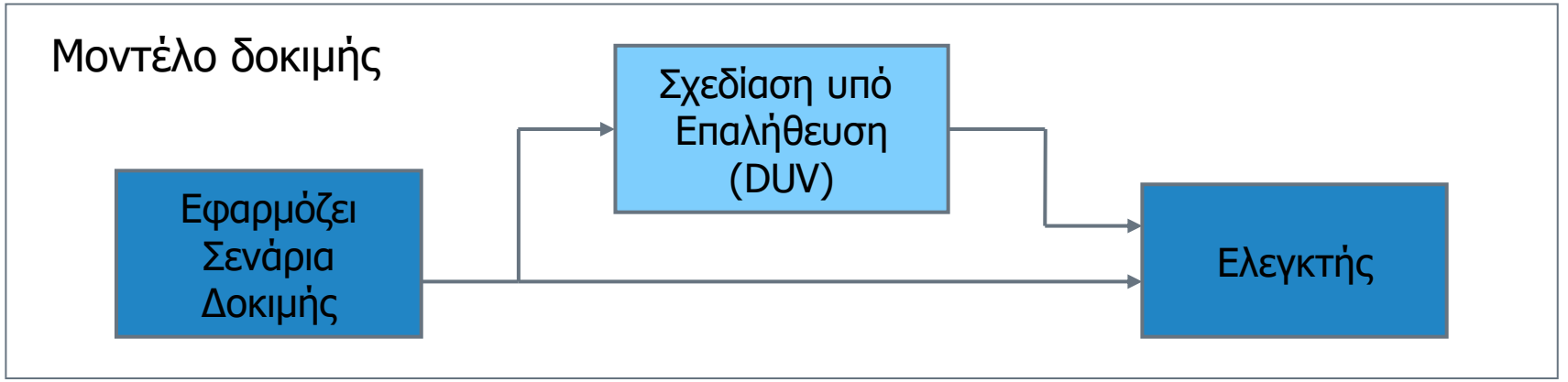
Διάγραμμα Ελέγχου του Πολλαπλασιαστή

- Είσοδος: input_rdy
- Έξοδοι:
  - a_sel, b_sel, pp1_ce, pp2_ce, sub, p_r_ce, p_i_ce



Επαλήθευση Ακολουθιακών Κυκλωμάτων

- Το DUV μπορεί να χρειαστεί πολλούς κύκλους για να παράγει την έξοδο
- Ο ελεγκτής χρειάζεται να
  - συγχρονιστεί με τη γεννήτρια δοκιμής
  - εξασφαλίσει ότι οι έξοδοι του DUV εμφανίζονται τη σωστή στιγμή
  - εξασφαλίσει ότι οι έξοδοι του DUV είναι σωστές



Μοντέλο Δοκιμής Πολλαπλασιαστή

```
entity multiplier_testbench is
end entity multiplier_testbench;
```

```
library ieee; use ieee.std_logic_1164.all,
                  ieee.fixed_pkg.all, ieee.math_complex.all;
architecture verify of multiplier_testbench is
  constant t_c : time := 50 ns;
  signal clk, reset : std_logic;
  signal input_rdy : std_logic;
  signal a_r, a_i, b_r, b_i : sfixed(3 downto -12);
  signal p_r, p_i : sfixed(7 downto -24);
  signal a, b : complex;
begin
  duv : entity work.multiplier(rtl)
    port map ( clk, reset, input_rdy,
               a_r, a_i, b_r, b_i,
               p_r, p_i );
```

```
  clk_gen : process is
  begin
    wait for t_c / 2; clk <= '1';
    wait for t_c / 2; clk <= '0';
  end process clk_gen;
  reset <= '1', '0' after 2 * t_c ns;
```

Μοντέλο Δοκιμής Πολλαπλασιαστή

```
apply_test_cases : process is
begin
  wait until falling_edge(clk) and reset = '0';
  a <= cmplx(0.0, 0.0); b <= cmplx(1.0, 2.0); input_rdy <= '1';
  wait until falling_edge(clk); input_rdy <= '0';
  for i in 1 to 5 loop
    wait until falling_edge(clk);
  end loop;
  a <= cmplx(1.0, 1.0); b <= cmplx(1.0, 1.0); input_rdy <= '1';
  wait until falling_edge(clk); input_rdy <= '0';
  for i in 1 to 6 loop
    wait until falling_edge(clk);
  end loop;
  -- further test cases ...
  wait;
end process apply_test_cases;

a_r <= to_sfixed(a.re, a_r'left, a_r'right);
a_i <= to_sfixed(a.im, a_i'left, a_i'right);
b_r <= to_sfixed(b.re, b_r'left, b_r'right);
b_i <= to_sfixed(b.im, b_i'left, b_i'right);
```

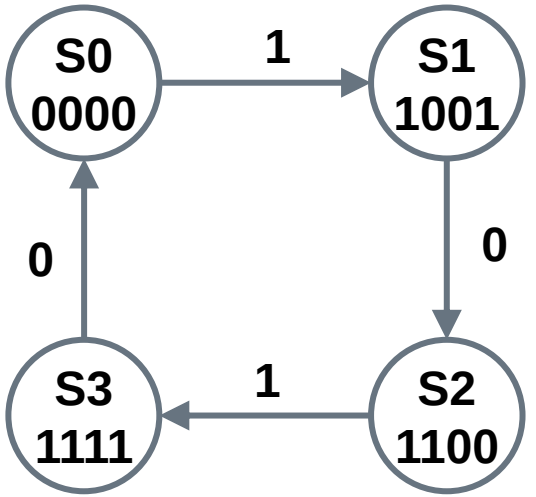
Μοντέλο Δοκιμής Πολλαπλασιαστή

```
  check_outputs : process is
    variable p : complex;
  begin
    wait until rising_edge(clk) and input_rdy = '1';
    p := a * b;
    for i in 1 to 5 loop
      wait until falling_edge(clk);
    end loop;
    assert abs (to_real(p_r) - p.re) < 2.0**(-12)
           and abs (to_real(p_i) - p.im) < 2.0**(-12);
  end process check_outputs;
end architecture verify;
```

Παράδειγμα: Μηχανή Moore

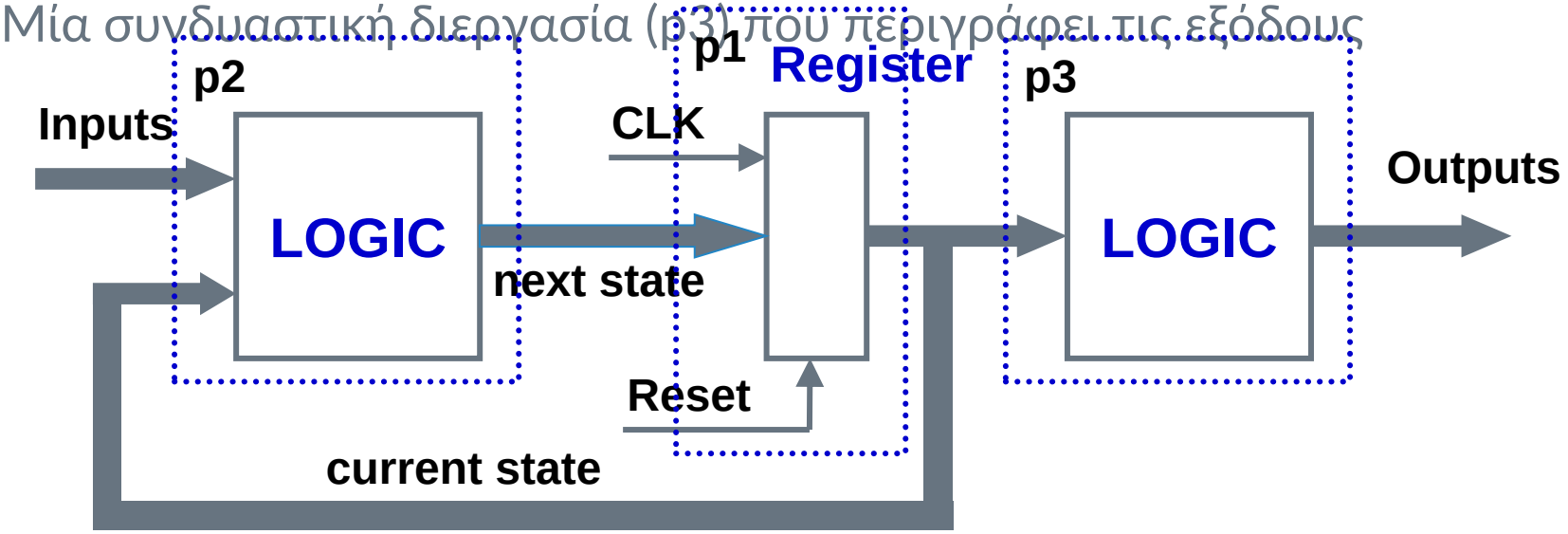
```
entity moore_machine is
      port (reset : in bit;
            clk : in bit;
            in1 : in bit;
            out1 : out bit_vector(3 downto 0));
end entity moore_machine;

architecture beh of moore_machine is

  type state_type is (s0,s1,s2,s3);
  signal current_state : state_type;
  signal next_state : state_type;

begin
  …
end architecture;
```

Κωδικοποίηση καταστάσεων
Δυο σήματα τύπου state_type



Υλοποίηση στην VHDL

- Υλοποίηση της μηχανής Moore με 3 διεργασίες:
  - Μία ακολουθιακή διεργασία (p1) που περιγράφει τους καταχωρητές,
  - Μία συνδυαστική διεργασία (p2) που περιγράφει τις μεταβάσεις καταστάσεων και
  - Μία συνδυαστική διεργασία (p3) που περιγράφει τις εξόδους



Υλοποίηση στην VHDL (συν.)

```
p1: process (reset,clk)
begin
   if (reset = '1') then
      current_state <= s0;
   elsif clk'event and clk = '1' then
      current_state <= next_state;
   end if;
end process;

p3: process (current_state)
begin
   case current_state is
      when s0 => out1 <= "0000";
      when s1 => out1 <= "1001";
      when s2 => out1 <= "1100";
      when s3 => out1 <= "1111";
   end case;
end process;
```

Αρχικοποίηση μηχανής

Αλλαγή κατάστασης

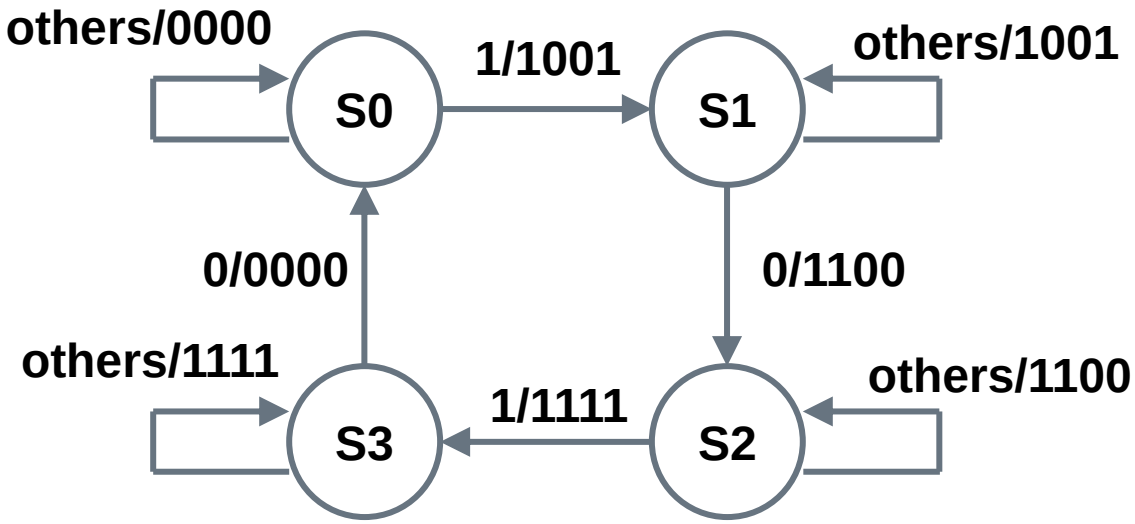
Η έξοδος εξαρτάται μόνο από την παρούσα κατάσταση

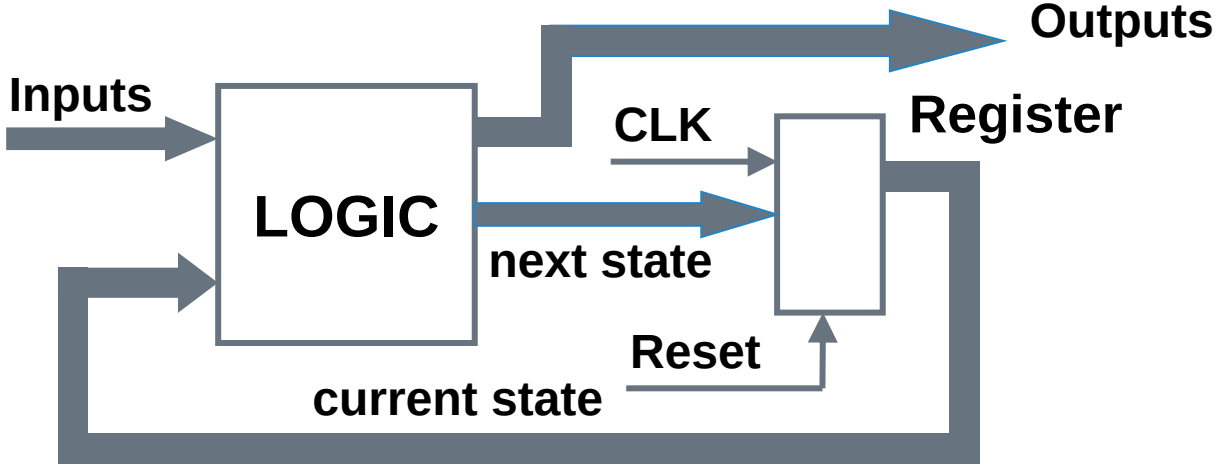
```
p2: process (current_state,in1)
begin
   next_state <= current_state;

   case current_state is
      when s0 =>
         if in1 = '1' then
               next_state <= s1;
         end if;
      ...
      when s3 =>
         if in1 = '0' then
               next_state <= s0;
         end if;
   end case;
end process;
```

Για την αποφυγή latches

Επιλογή επόμενης κατάστασης

Παράδειγμα: Μηχανή Mealy





Υλοποίηση στην VHDL

```
p1: process (reset,clk)
begin
    if (reset = '1') then
      current_state <= s0;
    elsif clk'event and clk = '1' then
      current_state <= next_state;
    end if;
end process;
p3: process (current_state,in1)
begin
    case current_state is
      when s0 =>
        if in1 = '1' then
              out1 <= "1001";
        else
              out1 <= "0000";
        end if;
      …
end process;
```

Η έξοδος εξαρτάται από την παρούσα κατάσταση και την είσοδο

```
p2: process (current_state,in1)
begin
  next_state <= current_state;

  case current_state is
    when s0 =>
      if in1 = '1' then
        next_state <= s1;
      end if;
…
     when s3 =>
      if in1 = '0' then
        next_state <= s0;
      end if;

   end case;
end process;
```

Κωδικοποίηση καταστάσεων

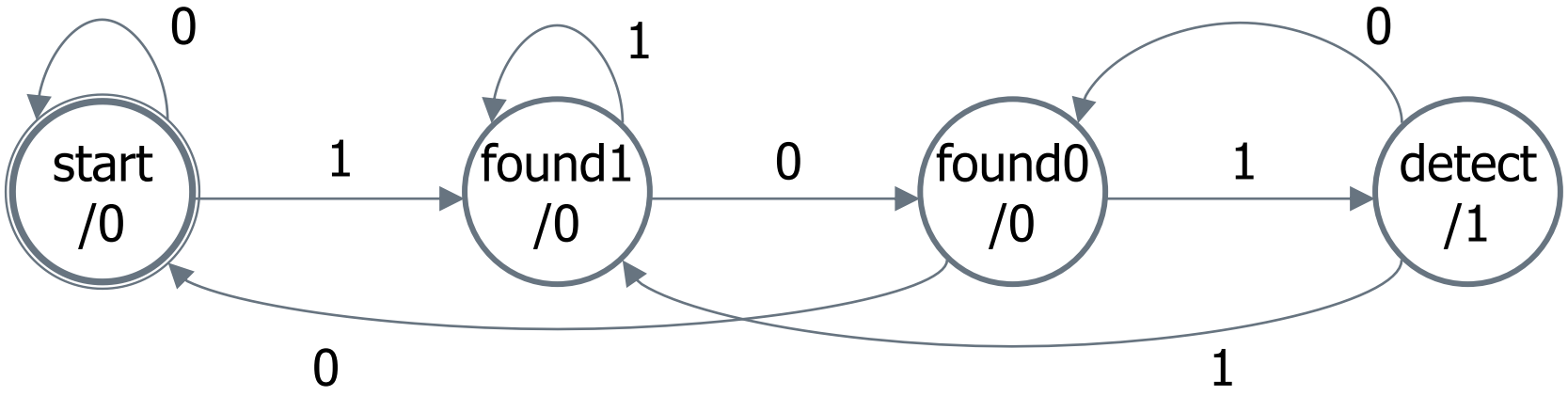
- Οι καταστάσεις αναπαριστώνται από έναν τύπο απαρίθμησης
  - τα εργαλεία σύνθεσης εκμεταλλεύονται τον τύπο απαρίθμησης για αποδοτικότερη βελτιστοποίηση
- Τα εργαλεία σύνθεσης υποστηρίζουν διαφορετικές μορφές κωδικοποίησης των καταστάσεων:
  - binary, onehot, gray
- Ο χρήστης μπορεί να καθορίσει την κωδικοποίηση των καταστάσεων. π.χ.:

```
constant s0: std_ulogic_vector(1 downto 0) :="00";
constant s1: std_ulogic_vector(1 downto 0) :="01";
constant s2: std_ulogic_vector(1 downto 0) :="10";
constant s3: std_ulogic_vector(1 downto 0) :="11";
signal cur_state, next_state : std_ulogic_vector(1 downto 0);
```

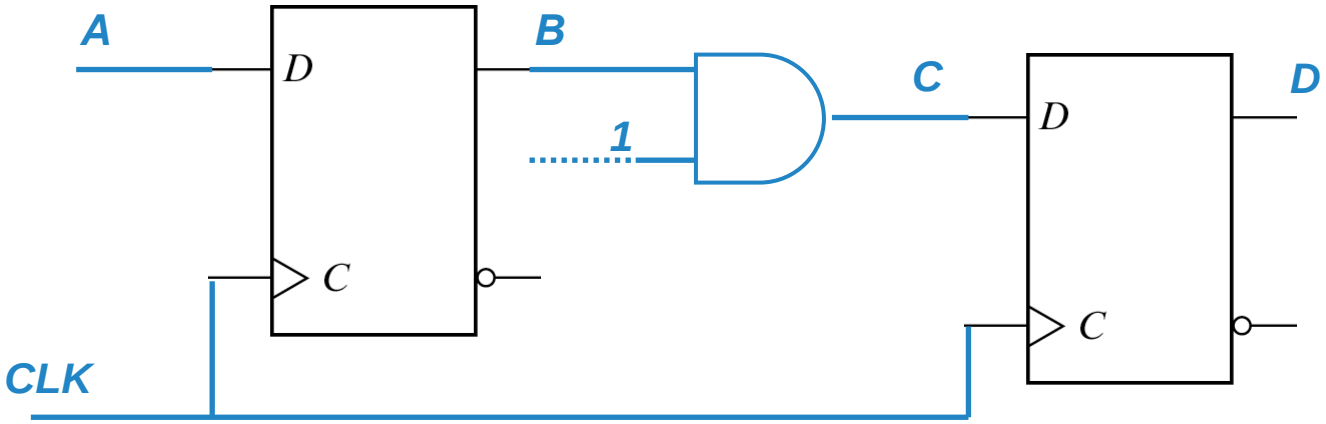
- Προτιμάτε τη χρήση των τύπων απαρίθμησης
  - παρέχει στο εργαλείο σύνθεσης μεγαλύτερη ευχέρεια στη βελτιστοποίηση της μηχανής
  - δηλώστε την αρχική κατάσταση της μηχανής στην αριστερή θέση του τύπου απαρίθμησης

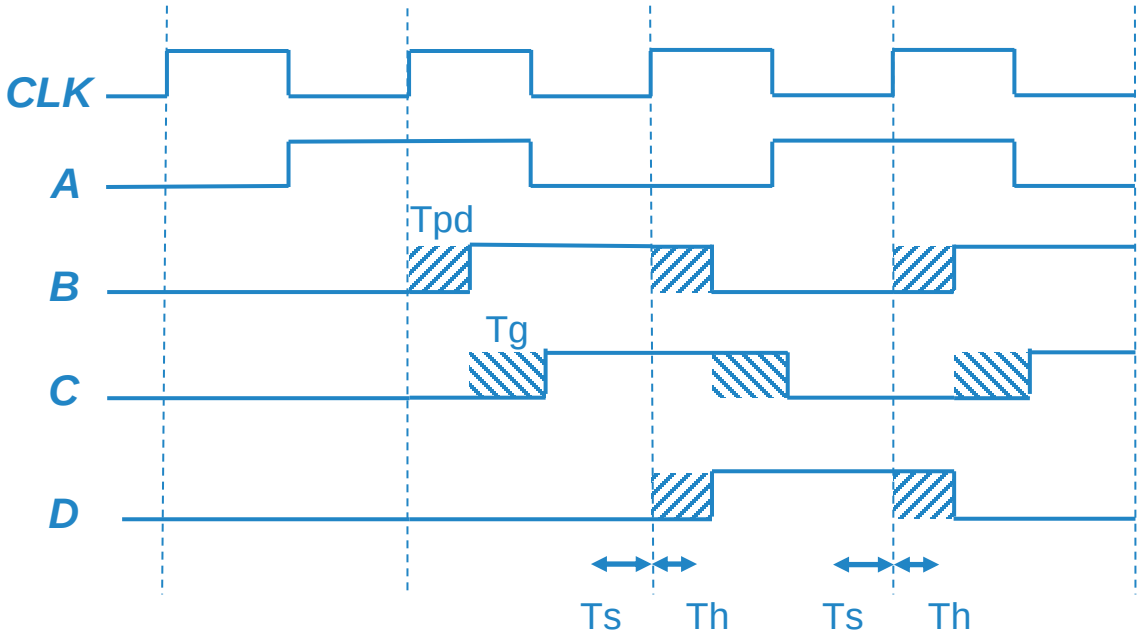
Παράδειγμα

- Σχεδιάστε ένα κύκλωμα που ανιχνεύει μία συγκεκριμένη ακολουθία ψηφίων (υπογραφή) σε μία σειριακή είσοδο δεδομένων
  - να υλοποιεί το διάγραμμα καταστάσεων



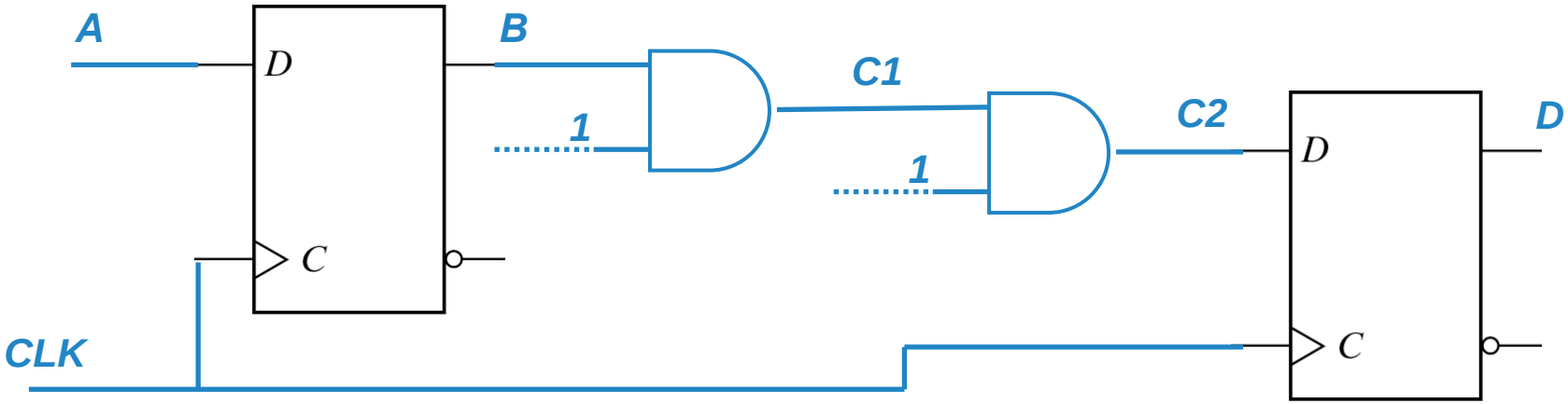
Παράδειγμα λειτουργίας (1)

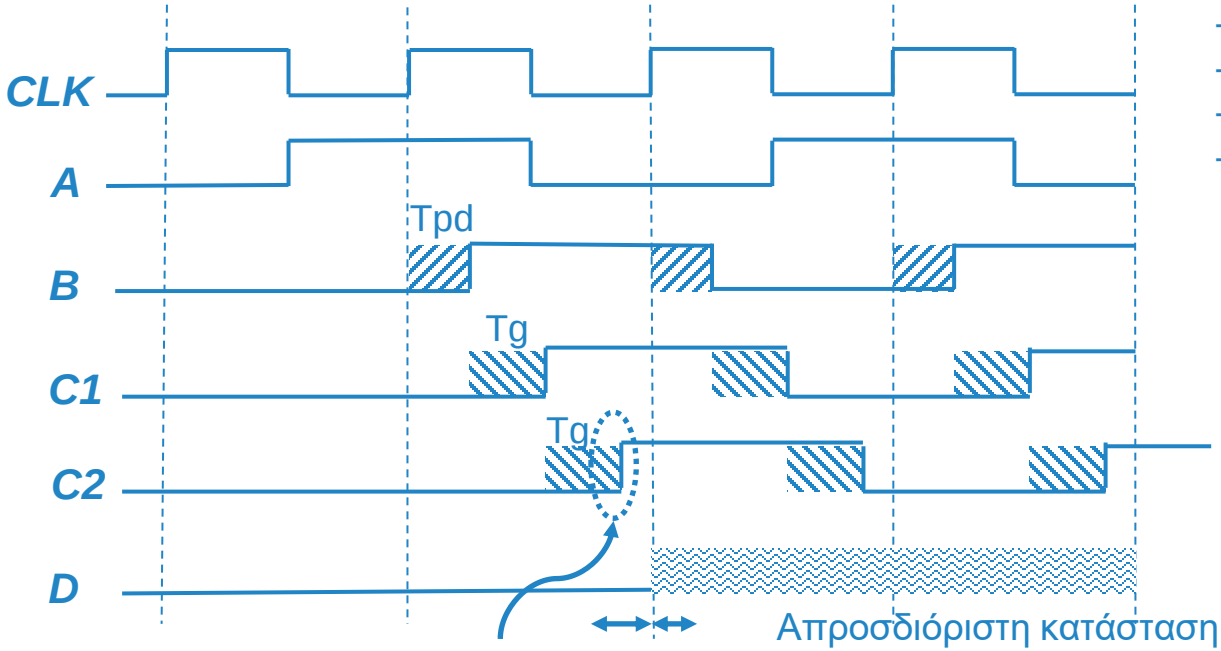




Tpd : propagation delay
Ts : flip-flop setup time
Th : flip-flop hold time
Tg : gate delay

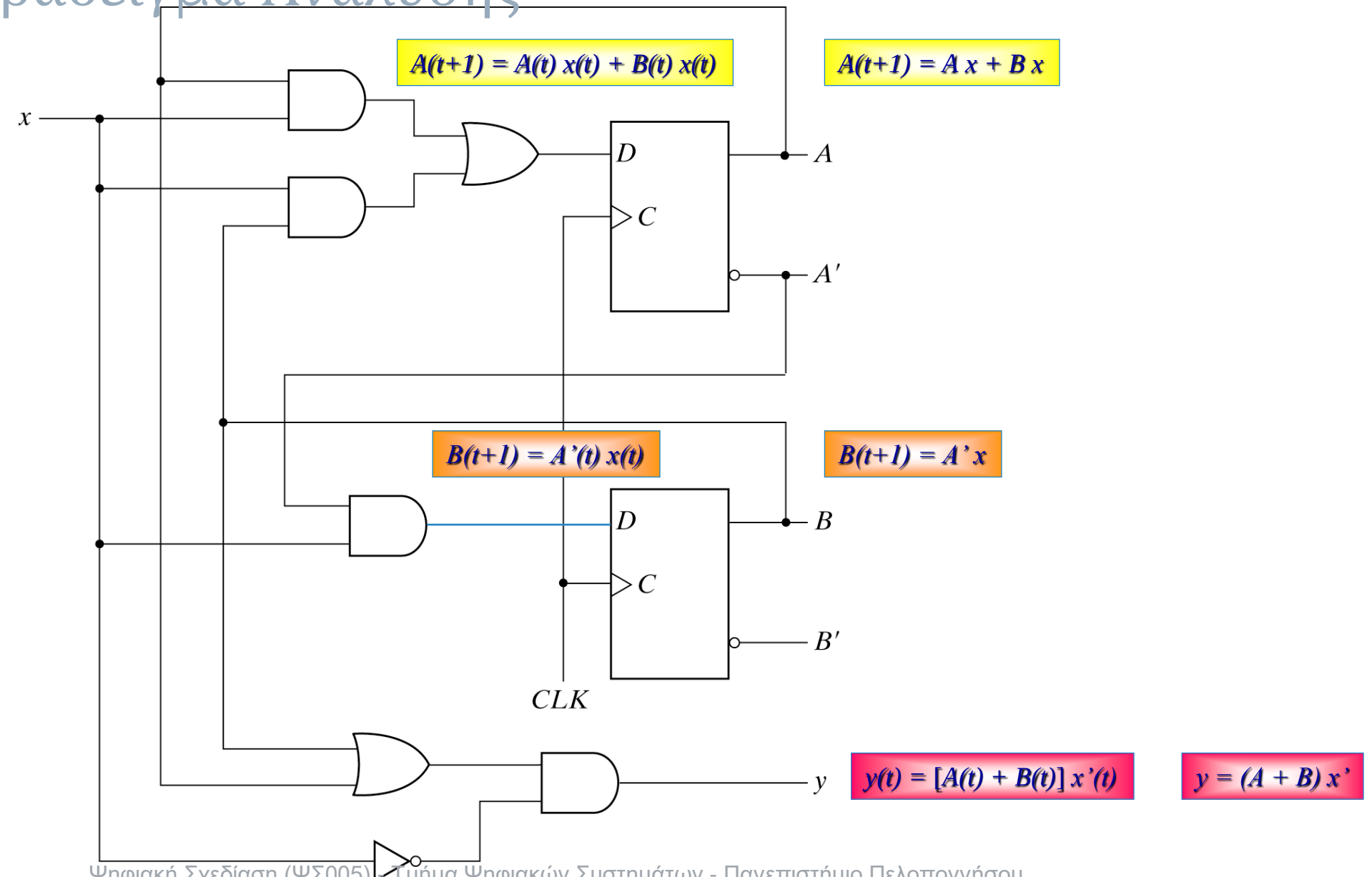
Παράδειγμα λειτουργίας (2)





Tpd : propagation delay
Ts : flip-flop setup time
Th : flip-flop hold time
Tg : gate delay

Παράδειγμα Ανάλυσης



Πίνακας καταστάσεων (state table)

$A(t+1) = A\,x + B\,x$

$B(t+1) = A'\,x$

$y = (A + B)\,x'$

Παρούσα Κατάσταση		Είσοδος	Επόμενη Κατάσταση		Έξοδος
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Πίνακας καταστάσεων (state table)

$A(t+1) = A\,x + B\,x$

$B(t+1) = A'\,x$

$y = (A + B)\,x'$

Παρούσα Κατάσταση	Επόμενη Κατάσταση		Έξοδος	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

Διαφορετική μορφή του πίνακα καταστάσεων

Διάγραμμα καταστάσεων (state diagram)

$A(t+1) = A\,x + B\,x$

$B(t+1) = A'\,x$

$y = (A + B)\,x'$

Παρούσα Κατάσταση	Επόμενη Κατάσταση		Έξοδος	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0