Γλώσσες Περιγραφής Υλικού

Μηχανές πεπερασμένων καταστάσεων

Νικόλαος Καββαδίας
nkavv@physics.auth.gr, nkavv@uop.gr

12 Μαΐου 2009

Σκιαγράφηση της διάλεξης

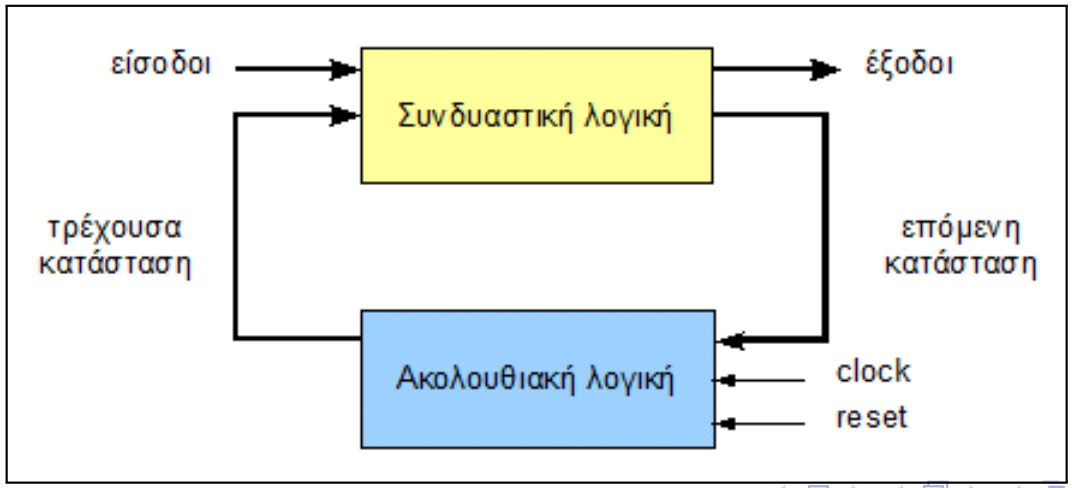
- Μηχανές πεπερασμένων καταστάσεων (FSM: Finite-State Machine)
  - Ορισμός του FSM
  - FSM κατά Mealy και κατά Moore - Μικτοί τύποι
  - Τρόποι καταγραφής της λειτουργίας ενός FSM
  - Δομή του FSM
  - Επανατοποθέτηση (αρχικοποίηση) ενός FSM
  - Κωδικοποίηση της τρέχουσας και επόμενης κατάστασης
  - Τεχνικές και διαφορετικά στυλ για την περιγραφή ενός FSM
  - Καταχωρημένες έξοδοι στα FSM (registered outputs)
  - Παραδείγματα: απαριθμητής BCD, ελεγκτής φωτεινού σηματοδότη (traffic light controller)

Μηχανές Πεπερασμένων Καταστάσεων: Εισαγωγή και ορισμός

- Οι μηχανές πεπερασμένων καταστάσεων (FSMs: Finite State Machines) αποτελούν ένα τύπο περιγραφής ακολουθιακών λογικών κυκλωμάτων ο οποίος είναι κατάλληλος για τη μοντελοποίηση κυκλωμάτων που πραγματοποιούν μια σειρά από λειτουργίες
- **Ορισμός:** FSM αποτελεί οποιοδήποτε κύκλωμα ειδικά σχεδιασμένο να διέρχεται με ακολουθιακό τρόπο από ένα σύνολο καταστάσεων

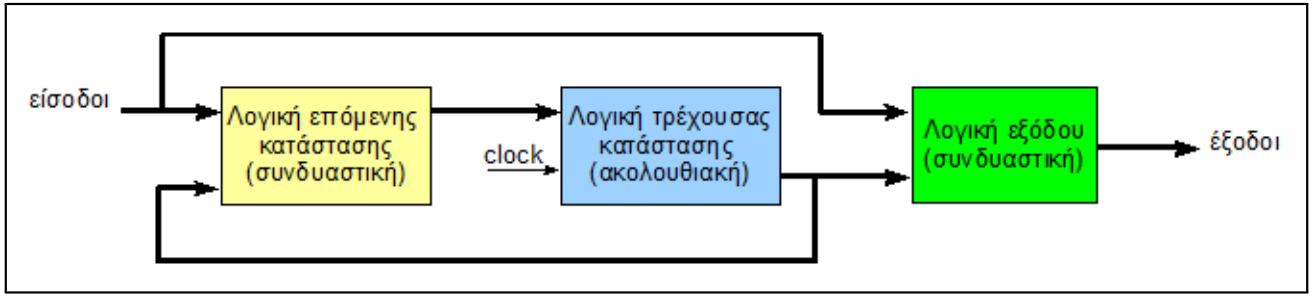
ℹ Ο λόγος που σχεδιάζονται FSM στη VHDL είναι ότι μια υλοποίηση σε υλικό είναι κατά κανόνα πολύ ταχύτερη σε χρόνο επεξεργασίας από την αντίστοιχη υλοποίηση σε λογισμικό ενός μικροεπεξεργαστή

- Γενικό διάγραμμα ενός FSM



Δομή ενός FSM

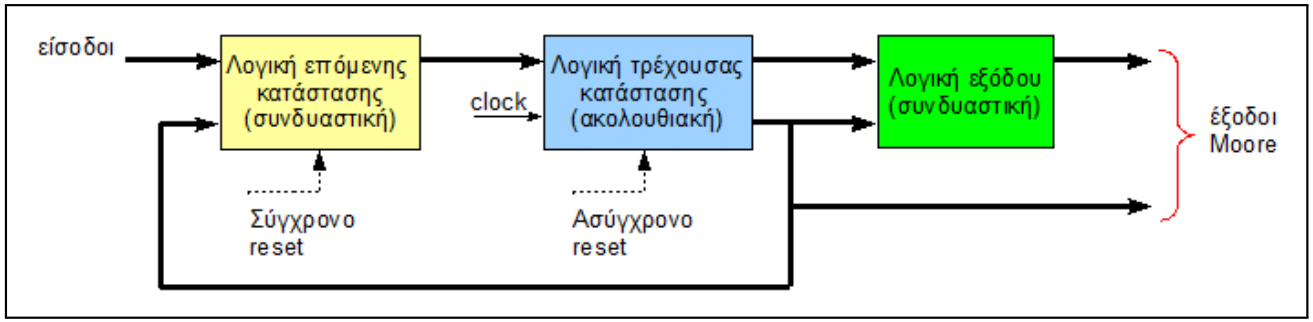
Τυπική οργάνωση ενός FSM



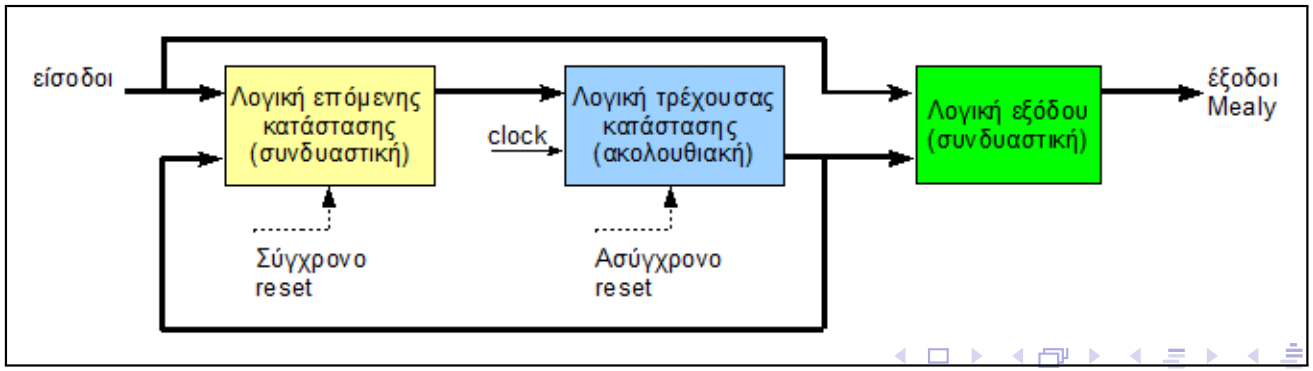
1. Λογική τρέχουσας κατάστασης: Υλοποιείται από καταχωρητή για την αποθήκευση της τρέχουσας κατάστασης του FSM. Η τιμή του αντιπροσωπεύει το συγκεκριμένο στάδιο στο οποίο βρίσκεται η λειτουργία του FSM
2. Λογική επόμενης κατάστασης: Συνδυαστική λογική η οποία παράγει την επόμενη κατάσταση της ακολουθίας. Η επόμενη κατάσταση αποτελεί συνάρτηση των εισόδων του FSM και της τρέχουσας κατάστασης
3. Λογική εξόδου: Συνδυαστική λογική που χρησιμοποιείται για την παραγωγή των σημάτων εξόδου του κυκλώματος. Οι έξοδοι αποτελούν συνάρτηση της εξόδου του καταχωρητή (τρέχουσας) κατάστασης και ΠΙΘΑΝΩΣ των εισόδων του FSM

Κατηγορίες FSM: τύπου Moore και τύπου Mealy (1)

- Στα FSM τύπου Moore οι έξοδοι είναι συνάρτηση μόνο της τρέχουσας κατάστασης
- Οργάνωση ενός FSM τύπου Moore

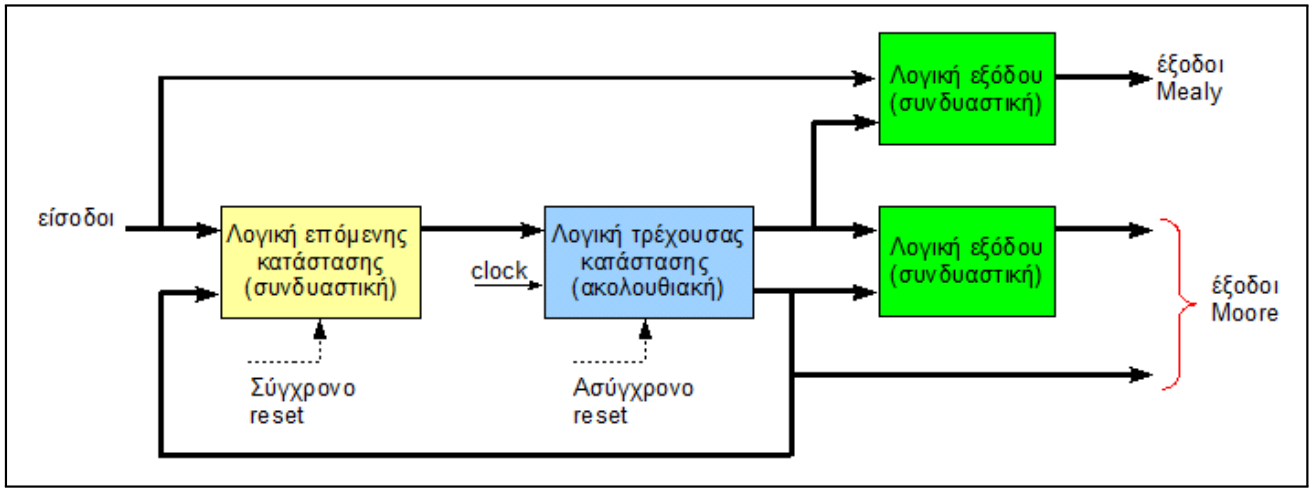


- Στα FSM τύπου Mealy οι έξοδοι είναι συνάρτηση των εισόδων και της τρέχουσας κατάστασης
- Οργάνωση ενός FSM τύπου Mealy



Κατηγορίες FSM: τύπου Moore και τύπου Mealy (2)

- Τα FSM τύπου Moore απαιτούν ένα κύκλο ρολογιού παραπάνω από τα τύπου Mealy για τον υπολογισμό των ίδιων εξόδων για τις ίδιες προδιαγραφές ενός FSM
- Σε ορισμένες περιπτώσεις FSM, οι έξοδοι παράγονται απευθείας από τον καταχωρητή τρέχουσας κατάστασης και έτσι δεν απαιτείται η ύπαρξη λογικής εξόδου
- Είναι εφικτός ο συνδυασμός των τύπων Mealy και Moore στο ίδιο FSM
- Οργάνωση ενός FSM μικτού τύπου



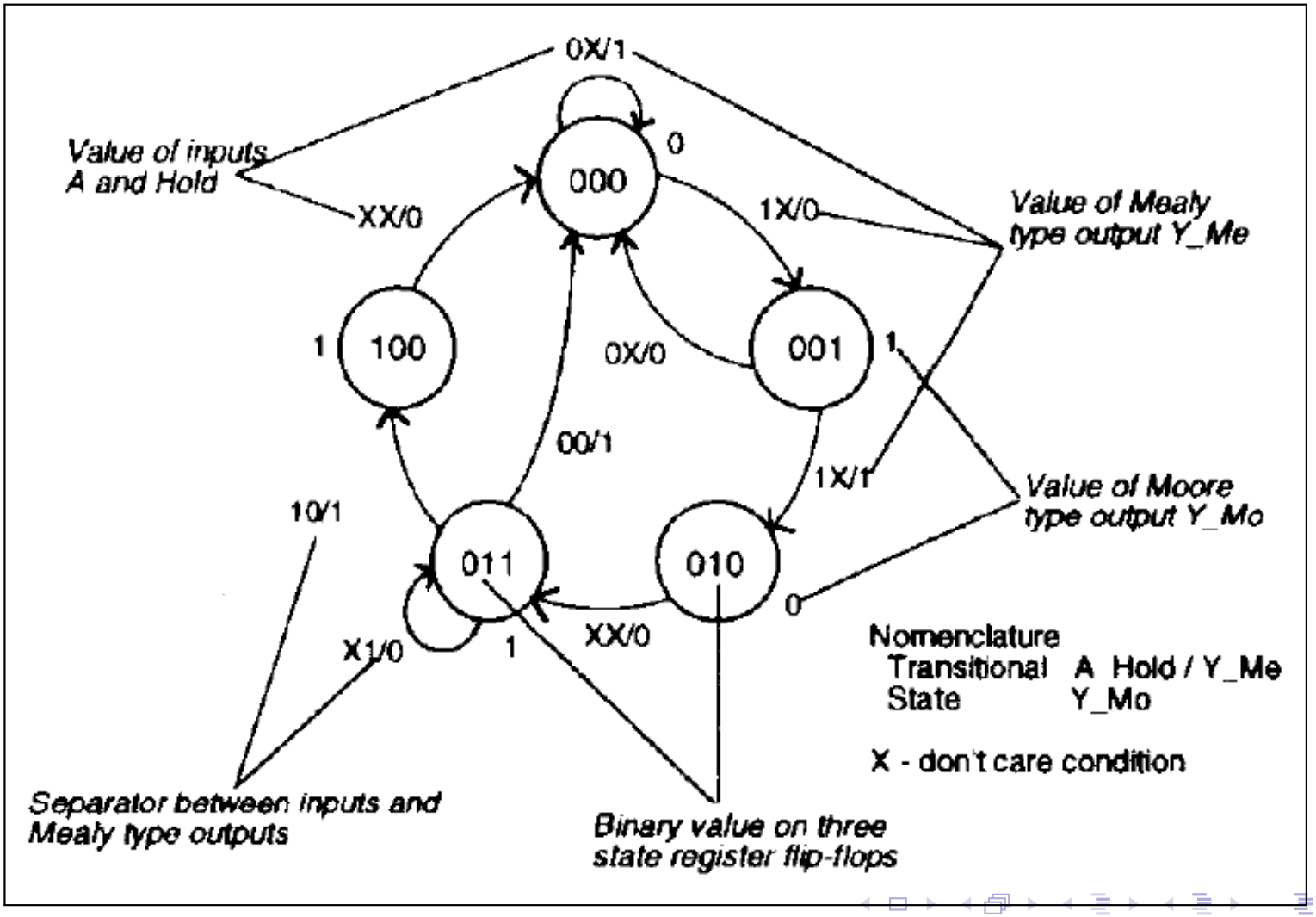
Τρόποι αναπαράστασης ενός FSM (1)

- Ένα FSM μπορεί να περιγραφεί είτε με χρήση πίνακα καταστάσεων (state table) είτε με γραφικό τρόπο χρησιμοποιώντας διάγραμμα μεταγωγής καταστάσεων (state transition graph)
- Πίνακας καταστάσεων

Inputs		Current state	Next state	Outputs	
A	Hold			Y_Me	Y_Mo
0	X	000	000	1	0
1	X	000	001	0	0
0	X	001	000	0	1
1	X	001	010	1	1
X	X	010	011	0	0
X	1	011	011	1	1
0	0	011	000	1	1
1	0	011	100	0	1
X	X	100	000	0	1

Τρόποι αναπαράστασης ενός FSM (2)

- Διάγραμμα μεταγωγής καταστάσεων



Κωδικοποίηση της κατάστασης στα FSM (1)

- Για την κωδικοποίηση της τρέχουσας κατάστασης ενός FSM χρησιμοποιείται κάποιου είδους δυαδική αριθμητική αναπαράσταση:
  - Ακολουθιακή (sequential): σε κάθε κατάσταση ανατίθενται δυαδικοί αριθμοί κατά αύξουσα σειρά. Ένας καταχωρητής κατάστασης των n bit μπορεί να χρησιμοποιηθεί για 2^n διακριτές καταστάσεις.
  - Κωδικοποίηση Gray: δύο διαδοχικοί αριθμοί διαφέρουν κατά ένα bit. Το i-οστό bit μιας λέξης Gray δίνεται από τη σχέση: $G_i = B_{i+1} \oplus B_i$, όπου B είναι ο αντίστοιχος αριθμός σε κωδικοποίηση binary

Κωδικοποίηση της κατάστασης στα FSM (2)

- (συνέχεια)
  - Κωδικοποίηση Johnson: κωδικοποίηση στην οποία επίσης διαδοχικοί αριθμοί διαφέρουν κατά ένα μόνο bit. Εκφράζεται ως αριστερή ή δεξιά περιστροφή (rotation) του αριθμού $2^n - 1$ κατά συγκεκριμένο αριθμό θέσεων
  - Κωδικοποίηση one-hot: σε κάθε κατάσταση αντιστοιχίζεται ξεχωριστό flip-flop. Σε κάθε κατάσταση ένα μόνο flip-flop έχει την τιμή ΄1΄. Για την περιγραφή n καταστάσεων απαιτούνται n flip-flop
  - Κωδικοποίηση one-cold: όπως n one-hot αλλά για κάθε κατάσταση ένα μόνο flip-flop έχει την τιμή ΄0΄

Κωδικοποίηση της κατάστασης στα FSM (3)

- Άλλες κωδικοποιήσεις
  - Κωδικοποίηση καθοριζόμενη από το χρήστη: οποιαδήποτε ανάθεση αριθμητικών αντιστοιχίσεων σε καταστάσεις
  - Παράδειγμα με χρήση CONSTANT

```
constant S1: std_logic_vector(3 downto 0) := "0110";
constant S2: std_logic_vector(3 downto 0) := "0111";
constant S3: std_logic_vector(3 downto 0) := "0000";
constant S4: std_logic_vector(3 downto 0) := "1010";
```

  - Παράδειγμα σύμφωνα με το πρότυπο IEEE 1076.6-2004 για το υποσύνολο της VHDL που υποστηρίζεται για λογική σύνθεση

```
type STATES is (S1, S2, S3, S4);
signal STATE : STATES;
attribute FSM_STATE of STATE: signal is "0110 0111 0000 1010";
```

  - Κωδικοποίηση προσδιοριζόμενη κατά τη σύνθεση: το εργαλείο σύνθεσης επιλέγει την κωδικοποίηση με βάση κάποιο μετρικό, π.χ. μικρότερη επιφάνεια υλικού ή κατανάλωση ενέργειας

Κωδικοποίηση της κατάστασης στα FSM (4)

- Διαμορφώσεις για την κωδικοποίηση των καταστάσεων σε ένα FSM

State	Sequential	Gray	Johnson	One-hot
0	0000	0000	00000000	0000000000000001
1	0001	0001	00000001	0000000000000010
2	0010	0011	00000011	0000000000000100
3	0011	0010	00000111	0000000000001000
4	0100	0110	00001111	0000000000010000
5	0101	0111	00011111	0000000000100000
6	0110	0101	00111111	0000000001000000
7	0111	0100	01111111	0000000010000000
8	1000	1100	11111111	0000000100000000
9	1001	1101	11111110	0000001000000000
10	1010	1111	11111100	0000010000000000
11	1011	1110	11111000	0000100000000000
12	1100	1010	11110000	0001000000000000
13	1101	1011	11100000	0010000000000000
14	1110	1001	11000000	0100000000000000
15	1111	1000	10000000	1000000000000000

Αρχικοποίηση και ασφαλής λειτουργία του FSM

- Για την αρχικοποίηση του FSM σε μία γνωστή αρχική κατάσταση επιβάλλεται η χρήση ασύγχρονης επανατοποθέτησης (asynchronous reset)
- Με τον τρόπο αυτό εξασφαλίζεται ότι με την πρώτη ακμή του ρολογιού το FSM μπορεί να μεταβεί σε μία νέα κατάσταση
- Σε περίπτωση που είτε δεν χρησιμοποιείται reset είτε χρησιμοποιείται μόνο σύγχρονη επανατοποθέτηση (synchronous reset)
  - Δεν υπάρχει τρόπος να προβλεφθεί η αρχική κατάσταση του FSM
  - Υπάρχει περίπτωση εγκλωβισμού του FSM σε κάποια αχρησιμοποίητη κατάσταση (unused state)
  - Προκειμένου να υπάρχει η δυνατότητα επιστροφής σε έγκυρη κατάσταση επιβάλλεται να ληφθούν υπόψη στο σχεδιασμό και οι τυχόν αχρησιμοποίητες καταστάσεις

Τεχνικές περιγραφής ενός FSM (1)

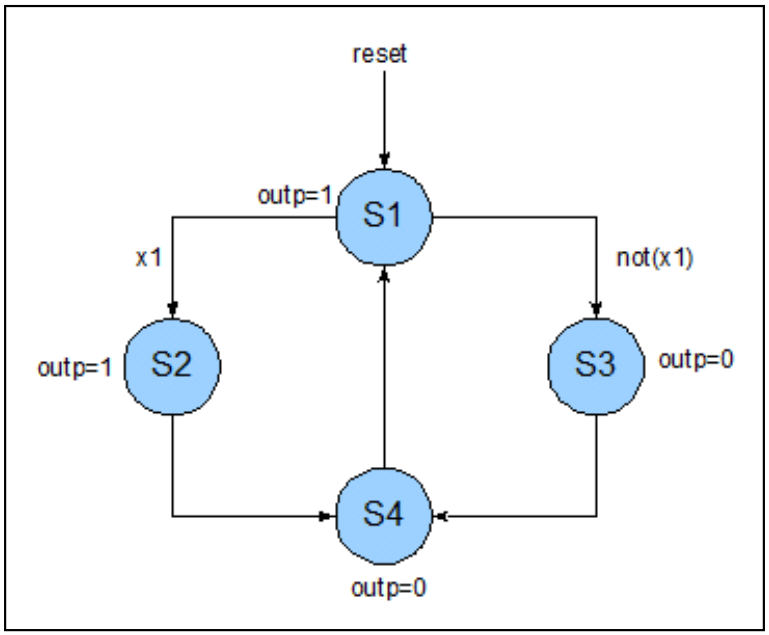
- Γενικά υφίστανται αρκετές τεχνικές για την σύνταξη της περιγραφής ενός FSM
- Ορισμένες από τις περισσότερο διαδεδομένες είναι οι εξής:
  1. FSM με μία διεργασία (process): Η λογική επόμενης κατάστασης, τρέχουσας κατάστασης και εξόδου περιγράφονται σε μία PROCESS
  2. FSM με δύο διεργασίες: Η λογική επόμενης κατάστασης και τρέχουσας κατάστασης υλοποιούνται σε μία PROCESS και η λογική εξόδου σε μία δεύτερη
  3. FSM με τρεις διεργασίες: Η λογική επόμενης κατάστασης, τρέχουσας κατάστασης και εξόδου περιγράφονται σε ξεχωριστές PROCESS

Τεχνικές περιγραφής ενός FSM (2)

- Εναλλακτικές, εξίσου διαδεδομένες, τεχνικές για την σύνταξη της περιγραφής ενός FSM
  1. FSM με δύο διεργασίες με τη λογική τρέχουσας κατάστασης να υλοποιείται σε μία PROCESS και τη λογική επόμενης κατάστασης και εξόδου να υλοποιούνται σε μία δεύτερη PROCESS
  2. Σε πολλές εφαρμογές τα σήματα εξόδου πρέπει να είναι σύγχρονα έτσι ώστε η έξοδος να ενημερώνεται μόνο με την ακμή του ρολογιού. Ένα τέτοιο FSM μπορεί να υλοποιηθεί με χρήση σημάτων για την αποθήκευση των εξόδων και περιγράφεται με μικρή τροποποίηση των παραπάνω τεχνικών

Διάγραμμα μεταγωγής καταστάσεων για ένα απλό FSM 4 καταστάσεων

- Το FSM του παραδείγματος καθορίζεται από:
  - Τέσσερις καταστάσεις: S1, S2, S3, S4
  - Μία είσοδο: x1
  - Μία έξοδο: outp
  - Πέντε περιπτώσεις μετάβασης από κατάσταση σε κατάσταση σύμφωνα με το παρακάτω διάγραμμα μεταγωγής καταστάσεων



Παράδειγμα FSM με μία διεργασία

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_1 is
port (
  clk, reset, x1 : IN std_logic;
  outp : OUT std_logic
);
end fsm_1;

architecture beh1 of fsm_1 is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type;
begin
  process (clk, reset)
  begin
  if (reset ='1') then
    state <= s1;
    outp <= '1';
  elsif (clk='1' and clk'event) then
    case state is
      when s1 =>
        if (x1 = '1') then
          state <= s2;
          outp <= '1';
        else
          state <= s3;
          outp <= '0';
        end if;
      when s2 =>
        state <= s4;
        outp <= '0';
      when s3 =>
        state <= s4;
        outp <= '0';
      when s4 =>
        state <= s1;
        outp <= '1';
    end case;
  end if;
  end process;
end beh1;
```

Παράδειγμα FSM με δύο διεργασίες

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_2 is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic
  );
end fsm_2;

architecture beh1 of fsm_2 is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type;
begin
  process1: process (clk, reset)
  begin
  if (reset ='1') then
    state <= s1;
  elsif (clk='1' and clk'EVENT) then
    case state is
      when s1 =>
        if (x1 = '1') then
          state <= s2;
        else
          state <= s3;
        end if;
      when s2 =>
        state <= s4;
      when s3 =>
        state <= s4;
      when s4 =>
        state <= s1;
    end case;
  end if;
  end process process1;

  process2 : process (state)
  begin
  case state is
    when s1 => outp <= '1';
    when s2 => outp <= '1';
    when s3 => outp <= '0';
    when s4 => outp <= '0';
  end case;
  end process process2;
end beh1;
```

Παράδειγμα FSM με τρεις διεργασίες

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_3 is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic
  );
end fsm_3;

architecture beh1 of fsm_3 is
  type state_type is (s1,s2,s3,s4);
  signal current_state, next_state:
    state_type;
begin
  process1: process (clk, reset)
  begin
  if (reset ='1') then
    state <= s1;
  elsif (clk='1' and clk'EVENT) then
    current_state <= next_state;
  end if;
  end process process1;
```

```
  process2 : process (current_state, x1)
  begin
  case current_state is
    when s1 =>
      if (x1 = '1') then
        next_state <= s2;
      else
        next_state <= s3;
      end if;
    when s2 =>
      next_state <= s4;
    when s3 =>
      next_state <= s4;
    when s4 =>
      next_state <= s1;
  end case;
  end process process2;

  process3 : process (current_state)
  begin
  case current_state is
    when s1 => outp <= '1';
    when s2 => outp <= '1';
    when s3 => outp <= '0';
    when s4 => outp <= '0';
  end case;
  end process process3;
end beh1;
```

Το ίδιο παράδειγμα με δύο διεργασίες και απομόνωση της λογικής τρέχουσας κατάστασης

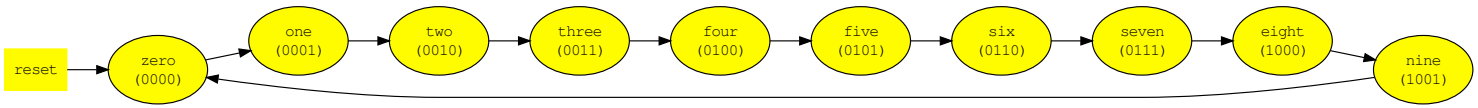
```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_2b is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic
  );
end fsm_2b;

architecture beh1 of fsm_2b is
  type state_type is (s1,s2,s3,s4);
  signal current_state, next_state:
    state_type;
begin
  process1: process (clk, reset)
  begin
  if (reset ='1') then
    current_state <= s1;
  elsif (clk='1' and clk'EVENT) then
    current_state <= next_state;
  end if;
  end process process1;

  process2: process (current_state, x1)
  begin
  case current_state is
    when s1 =>
      outp <= '1';
      if (x1 = '1') then
        state <= s2;
      else
        state <= s3;
      end if;
    when s2 =>
      outp <= '1';
      state <= s4;
    when s3 =>
      outp <= '0';
      state <= s4;
    when s4 =>
      outp <= '0';
      state <= s1;
  end case;
  end process process2;
end beh1;
```

Το ίδιο παράδ. με δύο διεργασίες, απομόνωση λογικής τρέχουσας κατάστασης και αποθηκευμένη έξοδο

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_2c is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic
  );
end fsm_2c;

architecture beh1 of fsm_2c is
  type state_type is (s1,s2,s3,s4);
  signal current_state, next_state:
    state_type;
  signal temp: std_logic;
begin
  process1: process (clk, reset)
  begin
  if (reset ='1') then
    state <= s1;
  elsif (clk='1' and clk'EVENT) then
    outp <= temp;
    current_state <= next_state;
  end if;
  end process process1;

  process2: process (current_state, x1)
  begin
  case current_state is
    when s1 =>
      temp <= '1';
      if (x1 = '1') then
        state <= s2;
      else
        state <= s3;
      end if;
    when s2 =>
      temp <= '1';
      state <= s4;
    when s3 =>
      temp <= '0';
      state <= s4;
    when s4 =>
      temp <= '0';
      state <= s1;
  end case;
  end process process2;
end beh1;
```

Σχεδιασμός FSM για τον απαριθμητή BCD

- Ο απαριθμητής BCD αποτελεί παράδειγμα υλοποίησης μιας μηχανής Moore επειδή η έξοδός του εξαρτάται μόνο από την αποθηκευμένη (τρέχουσα) κατάσταση
- Για την κωδικοποίηση της κατάστασης του FSM ως ακολουθιακή χρειάζεται ένας καταχωρητής εύρους $\lceil log_2(10) \rceil = 4$ bit.
- Διάγραμμα μεταγωγής καταστάσεων για τον απαριθμητή BCD



Περιγραφή του απαριθμητή BCD σε VHDL (1)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity bcd is
  port (
    clk, reset : std_logic;
    count : out std_logic_vector(3 downto 0)
  );
end bcd;

architecture fsm of bcd is
  TYPE state is (zero, one, two, three, four, five, six,
  seven, eight, nine);
  signal current_state, next_state: state;
begin

  process (clk, reset)
  begin
    if (reset = '1') then
      current_state <= zero;
    elsif (clk='1' and clk'EVENT) then
      current_state <= next_state;
    end if;
  end process;
```

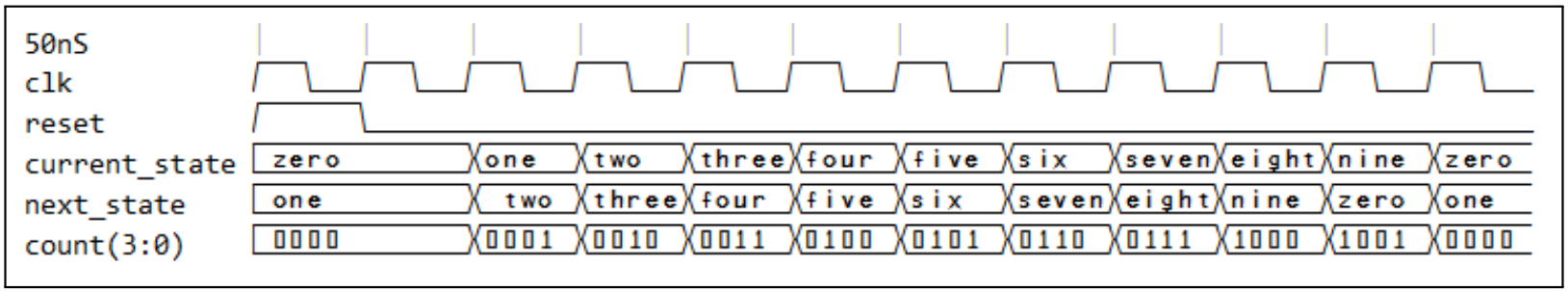
Περιγραφή του απαριθμητή BCD σε VHDL (2)

```
process (current_state)
begin
  case current_state is
    when zero =>
      count      <= "0000";
      next_state <= one;
    when one =>
      count      <= "0001";
      next_state <= two;
    when two =>
      count      <= "0010";
      next_state <= three;
    when three =>
      count      <= "0011";
      next_state <= four;
    when four =>
      count      <= "0100";
      next_state <= five;
```

```
    when five =>
      count      <= "0101";
      next_state <= six;
    when six =>
      count      <= "0110";
      next_state <= seven;
    when seven =>
      count      <= "0111";
      next_state <= eight;
    when eight =>
      count      <= "1000";
      next_state <= nine;
    when nine =>
      count      <= "1001";
      next_state <= zero;
  end case;
end process;

end fsm;
```

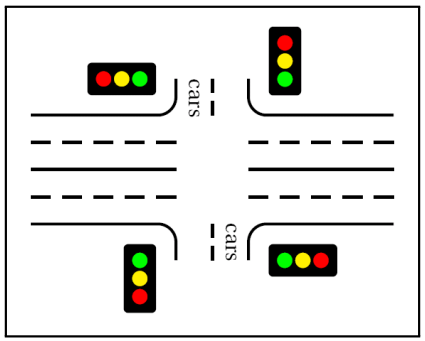
Προσομοίωση του απαριθμητή BCD

Χρονικό διάγραμμα του κυκλώματος



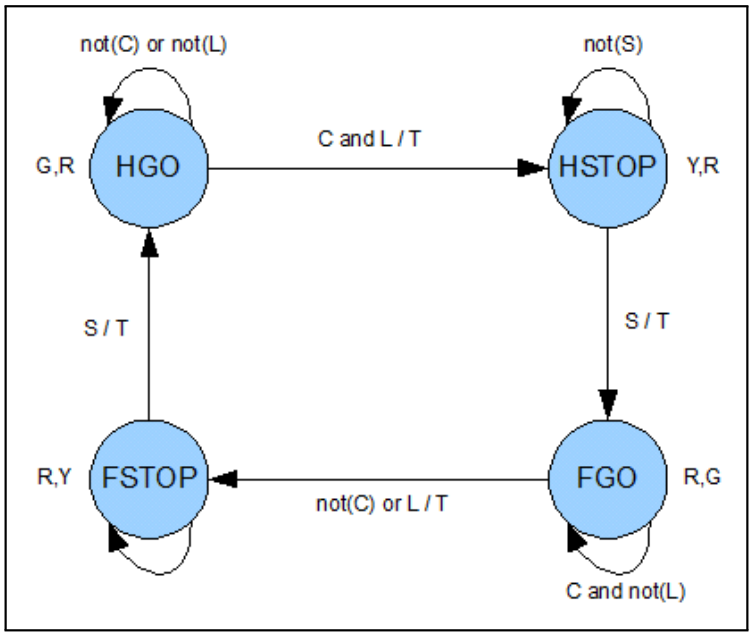
Το πρόβλημα του ελεγκτή φωτεινού σηματοδότη (TLC: Traffic Light Controller

Το FSM ελέγχει τους σηματοδότες σε μια διασταύρωση της εθνικής οδού Αθηνών-Λαμίας (highway) με έναν επαρχιακό δρόμο (farm road). Ο σηματοδότης στην εθνική οδό παραμένει στο πράσινο μέχρις ότου κατάλληλος αισθητήρας ο οποίος δημιουργεί μαγνητικό βρόχο ανιχνεύσει ένα αυτοκίνητο στον επαρχιακό δρόμο, οπότε αυτός αλλάζει σε κίτρινο και έπειτα σε κόκκινο. Ο σηματοδότης στον επαρχιακό γίνεται πράσινος και παραμένει έτσι μέχρι να μην εμφανιστεί άλλο αυτοκίνητο. Σε άλλη περίπτωση γίνεται πράσινοας με την πάροδο μεγάλου χρονικού διαστήματος (long timeout). Έπειτα, ο σηματοδότης γίνεται πορτοκαλί και κατόπιν κόκκινος και ο σηματοδότης στην εθνική αλλάζει σε πράσινο. Το χρονικό διάστημα παραμονής του σηματοδότη στον επαρχιακό δρόμο σε πράσινο θεωρείται σύντομο (short timeout).



Διάγραμμα μεταγωγής καταστάσεων για τον TLC

- Οι είσοδοι του FSM είναι η έξοδος του αισθητήρα (C:cars), το σήμα σύντομης διακοπής (S:short), και το σήμα μακράς διακοπής (L:long). Έξοδοι του FSM είναι το σήμα ενεργοποίησης του συστήματος μέτρησης χρόνου (T:start_timer) και οι ενδείξεις των δύο σηματοδοτών



Περιγραφή του TLC σε VHDL (1)

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

entity tlc is
  port (
    clk, reset : in std_logic;
    cars, short, long : in std_logic;
    highway_green, highway_yellow, highway_red : out std_logic;
    farm_green, farm_yellow, farm_red : out std_logic;
    start_timer : out std_logic
  );
end tlc;

architecture imp of tlc is
  type tlc_states is (HGO, HSTOP, FGO, FSTOP);
  signal current_state, next_state : tlc_states;
begin

  -- current state logic
  process (clk)
  begin
    if rising_edge(clk) then
      current_state <= next_state;
    end if;
  end process;
```

Περιγραφή του TLC σε VHDL (2)

```
-- next state and output logic combined
process (current_state, reset, cars, short, long)
begin
if (reset = '1') then
  start_timer <= '1';
  next_state <= HGO;
else
  case current_state is
    when HGO =>
      highway_green <= '1'; highway_yellow <= '0'; highway_red <= '0';
      farm_green <= '0'; farm_yellow <= '0'; farm_red <= '1';
      if (cars = '1' and long = '1') then
        start_timer <= '1';
        next_state <= HSTOP;
      else
        start_timer <= '0';
        next_state <= HGO;
      end if;
    when HSTOP =>
      highway_green <= '0'; highway_yellow <= '1'; highway_red <= '0';
      farm_green <= '0'; farm_yellow <= '0'; farm_red <= '1';
      if (short = '1') then
        start_timer <= '1';
        next_state <= FGO;
      else
        start_timer <= '0';
        next_state <= HSTOP;
      end if;
```

Περιγραφή του TLC σε VHDL (3)

```
      when FGO =>
        highway_green <= '0'; highway_yellow <= '0'; highway_red <= '1';
        farm_green <= '1'; farm_yellow <= '0'; farm_red <= '0';
        if (cars = '0' or long = '1') then
          start_timer <= '1';
          next_state <= FSTOP;
        else
          start_timer <= '0';
          next_state <= FGO;
        end if;
      when FSTOP =>
        highway_green <= '0'; highway_yellow <= '0'; highway_red <= '1';
        farm_green <= '0'; farm_yellow <= '1'; farm_red <= '0';
        if (short = '1') then
          start_timer <= '1';
          next_state <= HGO;
        else
          start_timer <= '0';
          next_state <= FSTOP;
        end if;
      end case;
    end if;
  end process;
end imp;
```

Προσομοίωση του ελεγκτή φωτεινού σηματοδότη

Χρονικό διάγραμμα του κυκλώματος