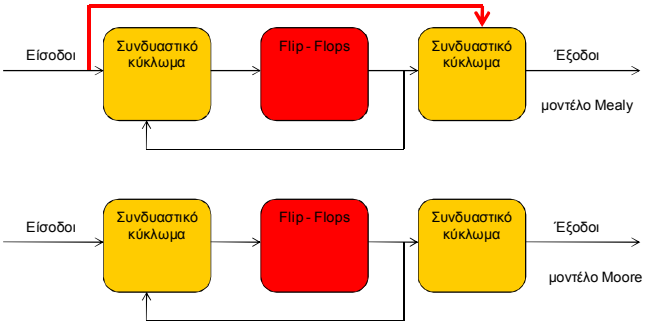
# 16. Μηχανές καταστάσεων (State Machines)

## 1. Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με τη σχεδίαση ενός ακολουθιακού κυκλώματος με σκοπό την αναγνώριση μιας συγκεκριμένης ακολουθίας bits εισόδου και πειραματική επαλήθευση της λειτουργίας του.

Γνωρίζουμε ότι η έξοδος ενός ακολουθιακού κυκλώματος εξαρτάται από τις εισόδους αλλά και από την κατάσταση που αυτό βρίσκεται πριν την εφαρμογή νέων τιμών στην είσοδο. Αυτό είναι αποτέλεσμα του κυκλώματος μνήμης που περιέχεται στο κύκλωμα ενός ακολουθιακού κυκλώματος μαζί με τις λογικές πύλες.

Τα ακολουθιακά κυκλώματα διακρίνονται σε σύγχρονα και ασύγχρονα. Στα σύγχρονα ακολουθιακά κυκλώματα με τα οποία ασχολούμαστε εδώ, οι λειτουργίες συντονίζονται από τους παλμούς ενός ρολογιού.




**Εικόνα 1.** Στο μοντέλο Mealy, οι έξοδοι είναι συναρτήσεις τόσο της παρούσας κατάστασης όσο και των εισόδων. Στο μοντέλο Moore, οι έξοδοι είναι συνάρτηση της παρούσας κατάστασης μόνο. Ωστόσο, τα δύο αυτά μοντέλα είναι ισοδύναμα.

Το πιο γνωστό μοντέλο ενός ακολουθιακού κυκλώματος έχει εισόδους, εξόδους και εσωτερικές καταστάσεις. Υπάρχουν δύο βασικά μοντέλα ακολουθιακών κυκλωμάτων:

- το μοντέλο Mealy και
- το μοντέλο Moore.

Στο **μοντέλο Mealy**, οι έξοδοι είναι συναρτήσεις τόσο της παρούσας κατάστασης όσο και των εισόδων.

Στο **μοντέλο Moore**, οι έξοδοι είναι συνάρτηση της παρούσας κατάστασης μόνο. Ωστόσο, τα δύο αυτά μοντέλα είναι ισοδύναμα.

Στην **Εικόνα 1** φαίνεται ποιοτικά η τοπολογία αυτών των δύο μοντέλων ακολουθιακών κυκλωμάτων.

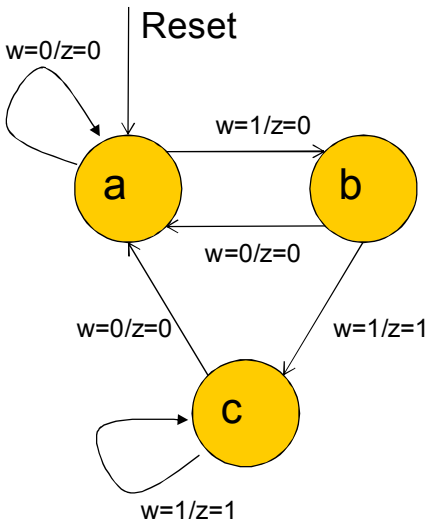
## 2. Υλοποίηση ακολουθιακού κυκλώματος τύπου Moore

Ένα κύκλωμα αναγνώρισης μιας συμβολοσειράς είναι ένα σύγχρονο ακολουθιακό κύκλωμα. Η υλοποίηση ενός κυκλώματος ανίχνευσης ακολουθίας bits ξεκινά με μια φραστική περιγραφή του προβλήματος. Π.χ. έστω ότι θέλουμε να σχεδιάσουμε ένα ακολουθιακό κύκλωμα με μια είσοδο w και μια έξοδο z, έτσι ώστε η έξοδος z να είναι 1 αν κατά δύο διαδοχικούς κύκλους ρολογιού η είσοδος w ήταν 1. Διαφορετικά η έξοδος να είναι μηδέν. Όλες οι αλλαγές στο κύκλωμα έστω ότι συμβαίνουν σε θετική ακμή ρολογιού. Συνεπώς το κύκλωμα αυτό ανιχνεύει αν δύο διαδοχικά 1 έχουν σταλεί στην είσοδό του. Μια τυπική καταγραφή εισόδου εξόδου φαίνεται στον **Πίνακα 1**, όπου με T είναι οι διάφορες περίοδοι ρολογιού (10 περίοδοι στη συγκεκριμένη περίπτωση):

| **Κύκλος ρολογιού** | **T0** | **T1** | **T2** | **T3** | **T4** | **T5** |
|---|---|---|---|---|---|---|
| **w** | 0 | 1 | 0 | **1** | **1** | 0 |
| **z** | 0 | 0 | 0 | 0 | 0 | **1** |
| **Κύκλος ρολογιού** | **T6** | **T7** | **T8** | **T9** | **T10** | |
| **w** | **1** | **1** | **1** | 0 | 1 | |
| **z** | 0 | 0 | **1** | **1** | 0 | |

**Πίνακας 1.**

Η ανάλυση του προβλήματος ξεκινά με τη σχεδίαση του διαγράμματος καταστάσεων (**Εικόνα 2**).



**Εικόνα 2.** Αρχικό διάγραμμα καταστάσεων κυκλώματος ανίχνευσης της ακολουθίας 11.

Παρατηρούμε ότι η ανάλυση του προβλήματος φανερώνει ότι η διαδικασία χρειάζεται 3 καταστάσεις a, b, c. Αυτές τοποθετούνται σε πίνακα κατάστασης (**Πίνακας 2**).

| **Παρούσα Κατάσταση** | **Είσοδος (w)** | **Επόμενη Κατάσταση** | **Έξοδος (z)** |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 1 |
| c | 0 | a | 0 |
| c | 1 | c | 1 |

**Πίνακας 2.**

Οι καταστάσεις a, b, μπορούν να κωδικοποιηθούν με τη χρήση δύο bits, άρα δύο Flip-Flop π.χ. τύπου D. Αυτό φαίνεται στον επόμενο τελικό πίνακα κατάστασης (**Πίνακας 3**), όπου έχουμε προσθέσει και την αχρησιμοποίητη κατάσταση.

Υλοποιώντας τους χάρτες Karnaugh για τα z, $D_A$, $D_B$ έχουμε τελικά τις συναρτήσεις που βλέπουμε στη συνέχεια.

| **Παρούσα Κατάσταση** | | | **Είσοδος** | **Επόμενη Κατάσταση** | | | **Έξοδος** | **Είσοδοι FF** | |
|---|---|---|---|---|---|---|---|---|---|
| | **A** | **B** | **w** | | **$A^+$** | **$B^+$** | **z** | **$D_A=A^+$** | **$D_B=B^+$** |
| a | 0 | 0 | 0 | a | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 1 | b | 0 | 1 | 0 | 0 | 1 |
| b | 0 | 1 | 0 | a | 0 | 0 | 0 | 0 | 0 |
| b | 0 | 1 | 1 | c | 1 | 0 | 1 | 1 | 0 |
| c | 1 | 0 | 0 | a | 0 | 0 | 0 | 0 | 0 |
| c | 1 | 0 | 1 | c | 1 | 0 | 1 | 1 | 0 |
| | 1 | 1 | 0 | | d | d | d | d | d |
| | 1 | 1 | 1 | | d | d | d | d | d |

**Πίνακας 3.**

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| w=0 | | | d | |
| w=1 | 1 | | d | |

$D_A$ = w A’ B’

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| w=0 | | | d | |
| w=1 | | 1 | d | 1 |

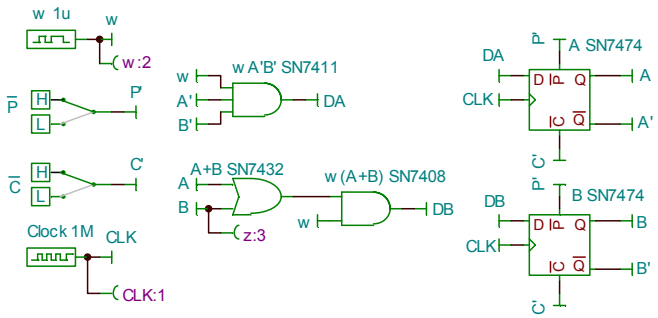
$D_B$= w A + w B = w (A + B)

| | B=0 | B=1 |
|---|---|---|
| A=0 | | 1 |
| A=1 | | d |

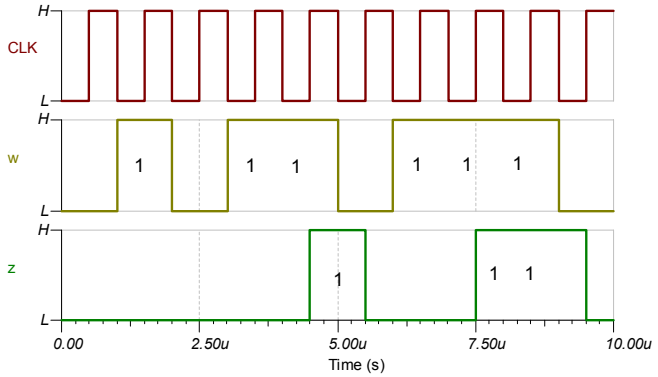
z = B

Παρατηρούμε ότι η z εκφράζεται μόνο ως συνάρτηση της κατάστασης AB και όχι της εισόδου σε συμφωνία με τις προδιαγραφές του μοντέλου Moore, στο οποίο η έξοδος είναι μόνο συνάρτηση της κατάστασης και όχι των εισόδων.

Η υλοποίηση του κυκλώματος στο TINA φαίνεται στην **Εικόνα 3** και η χρονική ανάλυση στην **Εικόνα 4**.




**Εικόνα 3.** Υλοποίηση του κυκλώματος στο TINA.

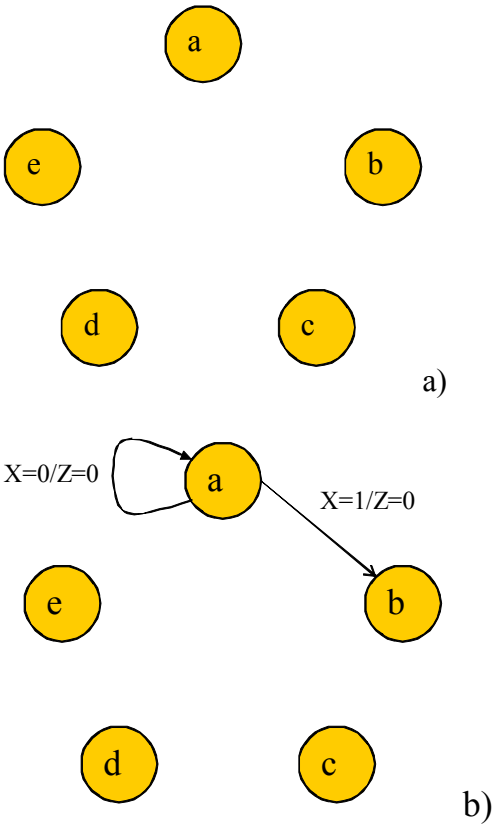



**Εικόνα 4.** Χρονική ανάλυση για επαλήθευση.

## 3. Υλοποίηση ακολουθιακού κυκλώματος τύπου Mealy

Θέλουμε να σχεδιάσουμε ένα σύγχρονο ακολουθιακό κύκλωμα τύπου Mealy, με μια είσοδο X και μια έξοδο Z η οποία να είναι 1 όταν σε μια σειρά δυαδικών αριθμών που εφαρμόζονται σειριακά στην είσοδό του, αναγνωρίζεται η διαδοχή **1101**. Πολλές φορές, το πρόβλημα δίνεται κατευθείαν με τη μορφή του

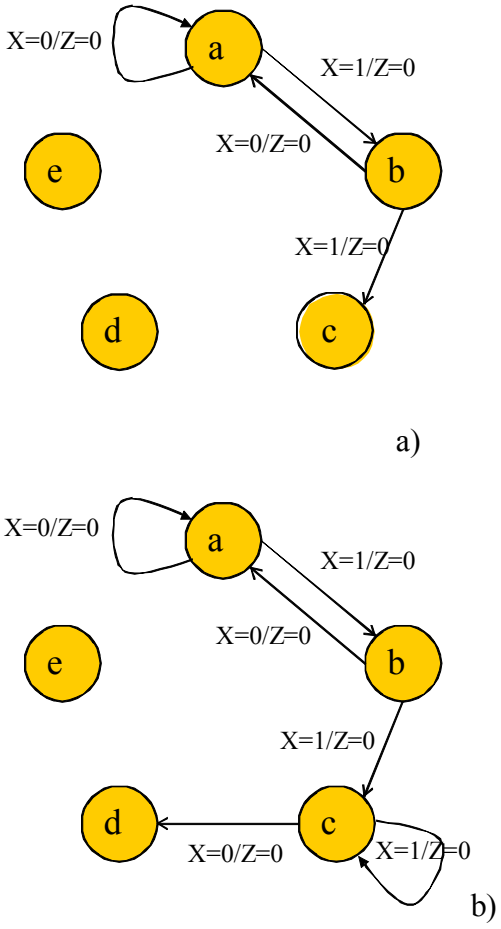
διαγράμματος καταστάσεων. Εδώ θα ξεκινήσουμε κατασκευάζοντας το διάγραμμα καταστάσεων. Αυτό θα έχει μια **αρχική κατάσταση**, που την ονομάζουμε **a** και τέσσερεις ακόμα τις **b, c, d, e** οι **οποίες αντιστοιχούν στα τέσσερα ψηφία της ακολουθίας 1101**. (**Εικόνα 5a**). Από την κατάσταση **a** αν η είσοδος είναι **X=0** το σύστημα μένει στην κατάσταση αυτή, ενώ αν **X=1** θα πάει στην κατάσταση **b** αναγνωρίζοντας το 1ο ψηφίο της ακολουθίας. (**Εικόνα 5b**).



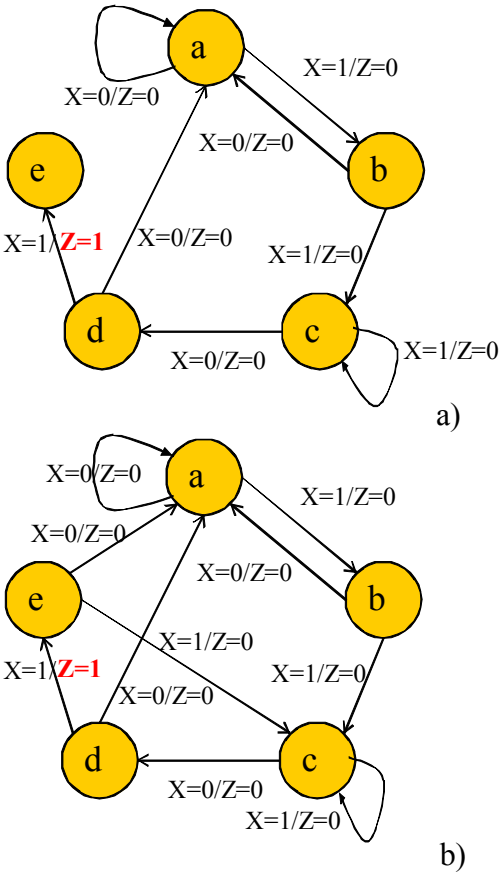

**Εικόνα 5.**

Από την κατάσταση **b** αν η είσοδος είναι **X=1** το σύστημα θα πάει στην **c** αναγνωρίζοντας το 2ο ψηφίο της δεδομένης σειράς, ενώ αν είναι **X=0** το σύστημα θα επιστρέψει στην κατάσταση **a**. (**Εικόνα 6a**). Από την κατάσταση **c** αν η είσοδος είναι **X=0** το σύστημα θα πάει στην d κατάσταση αναγνωρίζοντας το 3ο ψηφίο της δεδομένης σειράς, ενώ αν **X=1** μένει στην κατάσταση αυτή ως θέσης αναγνώρισης δύο διαδοχικών 1. (**Εικόνα 6b**).

Από την κατάσταση **d** αν η είσοδος είναι **X=0** το σύστημα θα πάει πίσω στην a αφού στη δεδομένη σειρά δεν υπάρχει διπλό μηδέν, ενώ αν είναι **X=1** το σύστημα θα πάει στην κατάσταση e και η έξοδος θα γίνει **Z=1** επειδή θα έχει αναγνωρίσει επιτυχώς τη σειρά 1101. (**Εικόνα 7a**). Από την κατάσταση **e** αν η είσοδος **X=0** το σύστημα πάει πίσω στην κατάσταση **a**, ενώ αν **X=1** το σύστημα πάει στην κατάσταση c (θέση αναγνώρισης δύο συνεχόμενων 1). (**Εικόνα 7b**).




**Εικόνα 6.**



**Εικόνα 7.**

Μετασχηματίζοντας άμεσα το διάγραμμα καταστάσεων σε πίνακα καταστάσεων, έχουμε τον **Πίνακα 4**.

Αν δύο ή περισσότερες καταστάσεις για την ίδια είσοδο έχουν τις ίδιες επόμενες καταστάσεις και τις ίδιες εξόδους, τότε είναι ισοδύναμες και μπορούμε να κρατήσουμε μία και να απαλείψουμε τις επόμενες, αντικαθιστώντας το όνομα των ισοδύναμων με το όνομα αυτής που κρατήσαμε. Στη συγκεκριμένη περίπτωση παρατηρούμε ότι οι καταστάσεις **b** και **e** είναι ισοδύναμες γιατί για τις ίδιες εισόδους πηγαίνουν στην ίδια επόμενη κατάσταση και με την ίδια έξοδο, όπως φαίνεται στον **Πίνακας 5**.

| Παρούσα Κατάσταση | Είσοδος (X) | Επόμενη Κατάσταση | Έξοδος (Z) |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 0 |
| c | 0 | d | 0 |
| c | 1 | c | 0 |
| d | 0 | a | 0 |
| d | 1 | e | 1 |
| e | 0 | a | 0 |
| e | 1 | c | 0 |

**Πίνακας 4.**

| Παρούσα Κατάσταση | Είσοδος (X) | Επόμενη Κατάσταση | Έξοδος (Z) |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 0 |
| c | 0 | d | 0 |
| c | 1 | c | 0 |
| d | 0 | a | 0 |
| d | 1 | e | 1 |
| e | 0 | a | 0 |
| e | 1 | c | 0 |

**Πίνακας 5.**

Συνεπώς μπορούμε να διαγράψουμε την **e** και κρατάμε τη **b**. Όπου το **e** εμφανίζεται ως επόμενη κατάσταση αντικαθίσταται με το **b**. (**Πίνακας 6**)

Στο νέο πίνακα που προέκυψε, δεν παρατηρούμε άλλη ισοδυναμία (αν προέκυπτε, θα επαναλαμβάναμε την προηγούμενη διαδικασία απλοποίησης). Οι τέσσερεις καταστάσεις που μένουν χρειάζονται για την κωδικοποίησή τους 2 FF. Στη συγκεκριμένη εφαρμογή θα επιλέξουμε τα JK-FF/NETr (αρνητικής ακμής).

| Παρούσα Κατάσταση | Είσοδος (X) | Επόμενη Κατάσταση | Έξοδος (Z) |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 0 |
| c | 0 | d | 0 |
| c | 1 | c | 0 |
| d | 0 | a | 0 |
| d | 1 | e ➔ b | 1 |

**Πίνακας 6.**

Η κωδικοποίηση των καταστάσεων, για να έχουμε το απλούστερο συνδυαστικό κύκλωμα, γίνεται με τον **κανόνα του Hamphrey**. Σύμφωνα με τον κανόνα αυτό: **δύο ή περισσότερες καταστάσεις κωδικοποιούνται γειτονικά αν έχουν τις ίδιες επόμενες καταστάσεις ανεξάρτητα από την είσοδο και την έξοδο**.

Στην περίπτωσή μας έχουμε τον **Πίνακα 7**:

| Παρούσα Κατάσταση | Είσοδος (X) | Επόμενη Κατάσταση | Έξοδος (Z) |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 0 |
| c | 0 | d | 0 |
| c | 1 | c | 0 |
| d | 0 | a | 0 |
| d | 1 | b | 1 |

**Πίνακας 7.**

Δηλαδή η **a** και **d** πρέπει να κωδικοποιηθούν γειτονικά (**Πίνακας 8**).

Δηλαδή η **b** συνδυάζεται μερικώς με τις **a** και **c**.

Επιλέγουμε την κωδικοποίηση: **a ➔ 00, b ➔10, c ➔ 11, d➔01**.

| Παρούσα Κατάσταση | Είσοδος (X) | Επόμενη Κατάσταση | Έξοδος (Z) |
|---|---|---|---|
| a | 0 | a | 0 |
| a | 1 | b | 0 |
| b | 0 | a | 0 |
| b | 1 | c | 0 |
| c | 0 | d | 0 |
| c | 1 | c | 0 |
| d | 0 | a | 0 |
| d | 1 | b | 1 |

**Πίνακας 8.**

Ο τελικός κωδικοποιημένος πίνακας καταστάσεων έχει ως εξής (**Πίνακας 9**):

| Παρούσα Κατάσταση | | | Είσοδος | Επόμενη Κατάσταση | | | Έξοδος | Είσοδοι FF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | X | | $A^+$ | $B^+$ | Z | JA | KA | JB | KB |
| a | 0 | 0 | 0 | a | 0 | 0 | 0 | 0 | X | 0 | X |
| a | 0 | 0 | 1 | b | 1 | 0 | 0 | 1 | X | 0 | X |
| b | 1 | 0 | 0 | a | 0 | 0 | 0 | X | 1 | 0 | X |
| b | 1 | 0 | 1 | c | 1 | 1 | 0 | X | 0 | 1 | X |
| c | 1 | 1 | 0 | d | 0 | 1 | 0 | X | 1 | X | 0 |
| c | 1 | 1 | 1 | c | 1 | 1 | 0 | X | 0 | X | 0 |
| d | 0 | 1 | 0 | a | 0 | 0 | 0 | 0 | X | X | 1 |
| d | 0 | 1 | 1 | b | 1 | 0 | 1 | 1 | X | X | 1 |

**Πίνακας 9.**

Οι πίνακες Karnaugh έχουν ως εξής:

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| X=0 | | | X | X |
| X=1 | 1 | 1 | X | X |

JA = X

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| X=0 | X | X | 1 | 1 |
| X=1 | X | X | | |

KA = X’

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| X=0 | | X | X | |
| X=1 | | X | X | 1 |

JB = A X

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| X=0 | X | 1 | | X |
| X=1 | X | 1 | | X |

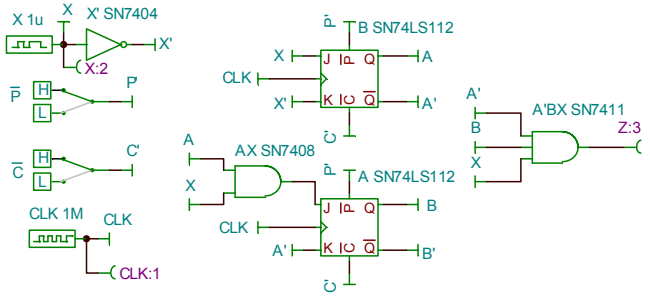
KB = A’

| | AB=00 | AB=01 | AB=11 | AB=10 |
|---|---|---|---|---|
| X=0 | | | | |
| X=1 | | 1 | | |

Z = A’ B X

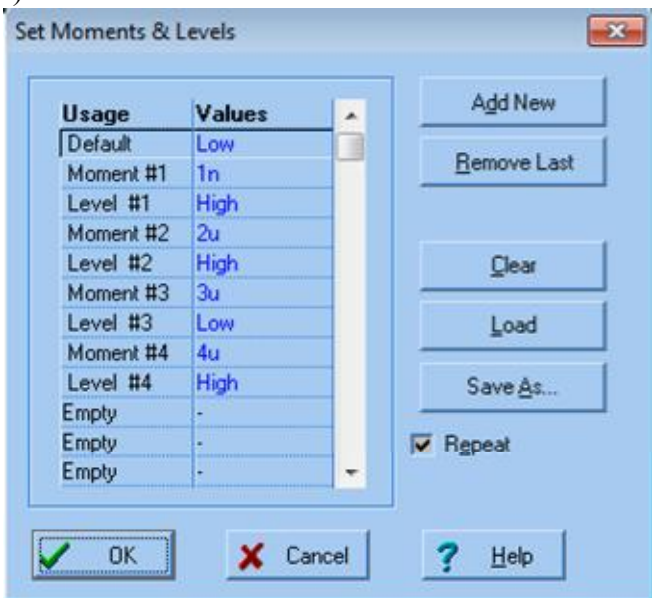
Παρατηρούμε ότι η Z εκφράζεται ως συνάρτηση της κατάστασης και της εισόδου σε συμφωνία με τις προδιαγραφές του μοντέλου Mealy, στο οποίο η έξοδος είναι συνάρτηση της κατάστασης και των εισόδων.

Η υλοποίηση του κυκλώματος στο TINA φαίνεται στη **Εικόνα 8** και η χρονική ανάλυση στην **Εικόνα 9.**

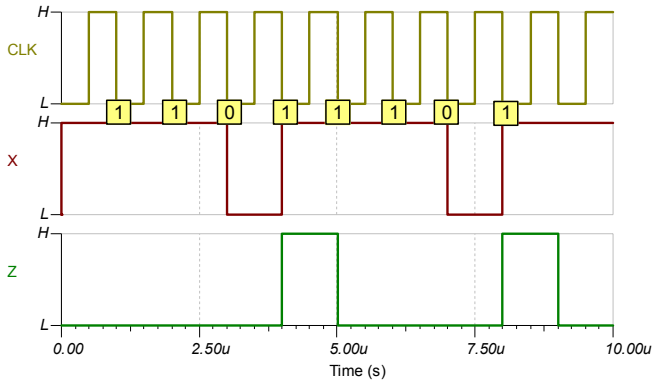



a)

b)




**Εικόνα 8.** a) Υλοποίηση του κυκλώματος στο TINA. b) Η λίστα τιμών στη γεννήτρια X.



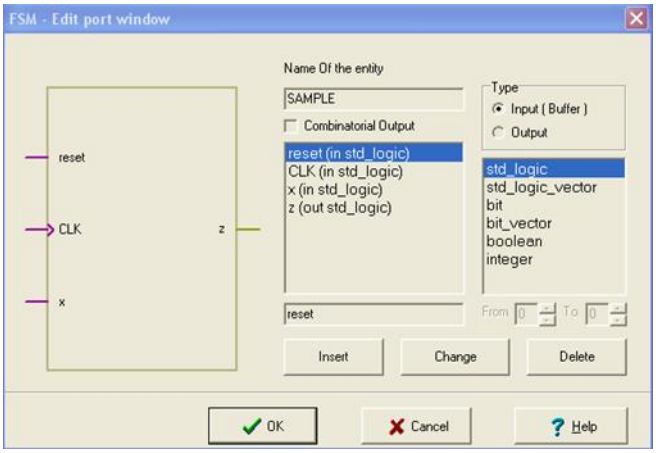

**Εικόνα 9.** Χρονική ανάλυση για επαλήθευση.

## 4. Ασκήσεις

1. **Σχεδιάστε ακολουθιακό κύκλωμα αναγνώρισης της ακολουθίας 1010.**
2. **Υλοποιήστε το κύκλωμα ανίχνευσης της ακολουθίας 1001, ως μηχανή πεπερασμένων καταστάσεων σε VHDL με τον FSM Editor του TINA και κάντε προσομοίωση της λειτουργίας του.**

Λύση.

Εκκινούμε τον FSM Editor και ξεκινάμε καινούργια εργασία. Ορίζουμε τις εισόδους του μπλοκ διαγράμματος του κυκλώματος όπως στην **Εικόνα 1**.



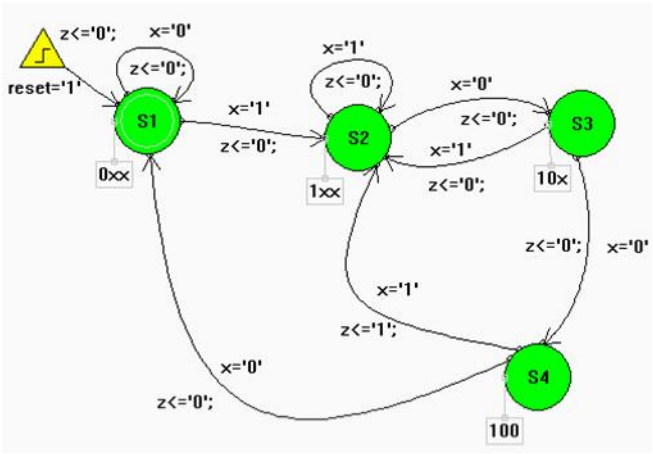

**Εικόνα 1.** Ορισμός και τύπος θυρών εισόδου εξόδου και ρολογιού.

Συνεχίζουμε σχεδιάζοντας την αλληλουχία των καταστάσεων και των μεταβάσεων μεταξύ τους όπως στην **Εικόνα 2.**

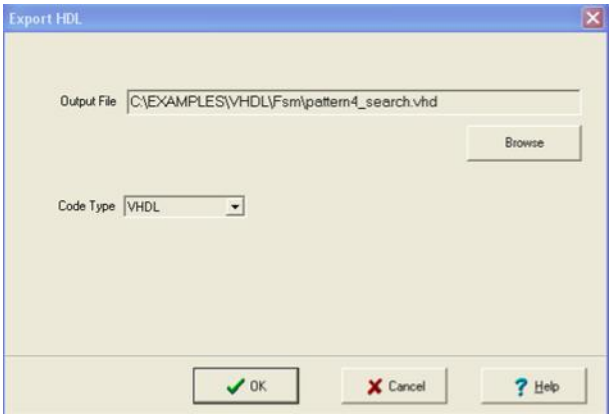
Στη συνέχεια εξάγουμε τον VHDL κώδικα που παρήγαγε ο FSM Editor σε αρχείο, όπως φαίνεται στην **Εικόνα 3**.

Μεταβαίνουμε στο TINA και από τα Tools επιλέγουμε New Macro Wizard. Τσεκάρουμε την επιλογή From File και προσδιορίζουμε το προηγούμενο αρχείο VHDL που εξαγάγαμε από τον FSM Editor (**Εικόνα 4**). Πατάμε Next και στη συνέχεια OK, για να τοποθετήσουμε το νέο μπλοκ διάγραμμα του κυκλώματος.

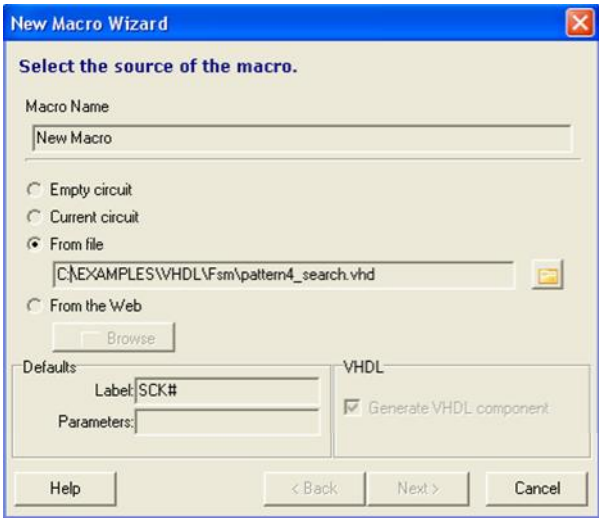
.




**Εικόνα 2.** Μεταβάσεις καταστάσεων.



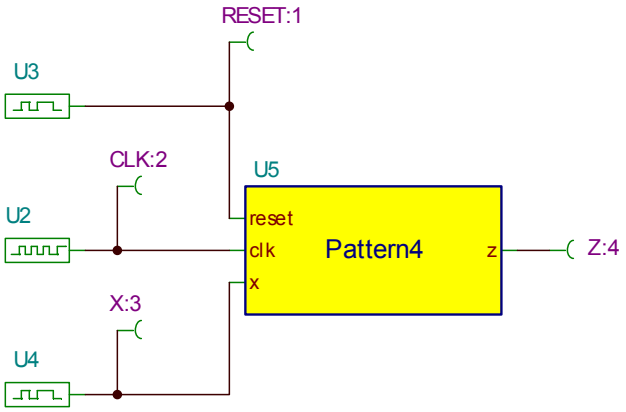

**Εικόνα 3.** Εξαγωγή VHDL κώδικα σε αρχείο.



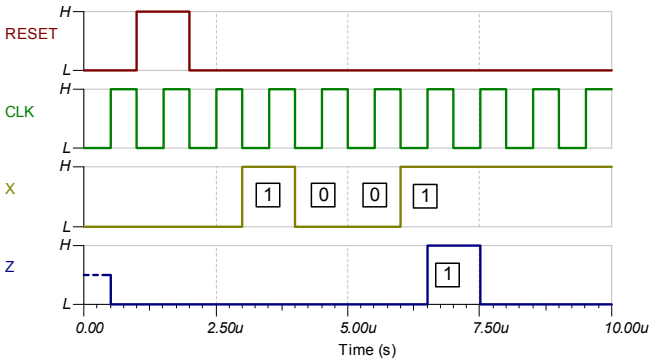

**Εικόνα 4.** Εισαγωγή του VHDL κώδικα ως Macro στο TINA.

Εισάγουμε τις απαραίτητες πηγές παλμών και ρολογιού και απαραίτητα voltage pins για την

προσομοίωση, όπως φαίνεται στην **Εικόνα 5**. Τα αποτελέσματα της χρονικής ανάλυσης φαίνονται στην **Εικόνα 6**, όπου έχουμε σημειώσει το σημείο που γίνεται η ανίχνευση με την άνοδο του z στο 1.




**Εικόνα 5.** Προετοιμασία του κυκλώματος για προσομοίωση.



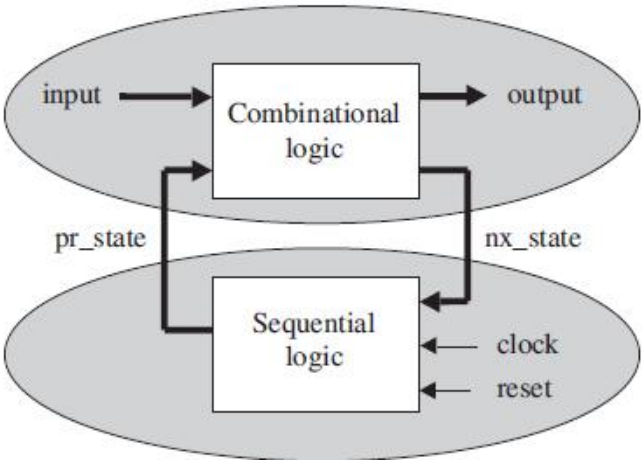

**Εικόνα 6.** Αποτελέσματα χρονικής προσομοίωσης κυκλώματος ανίχνευσης της ακολουθίας 1001.

## Υλοποιήσεις μηχανών πεπερασμένων καταστάσεων (FSMs) με τη VHDL

Το επόμενο σχήμα δείχνει το μπλοκ διάγραμμα μια μηχανής πεπερασμένων καταστάσεων μονής φάσης. Το κάτω τμήμα περιέχει την ακολουθιακή λογική (flip-flops), ενώ το πάνω τμήμα περιέχει τη συνδυαστική λογική.

Το συνδυαστικό κύκλωμα έχει δύο εισόδους. Η μία είναι η pr_state (η παρούσα κατάσταση) και η άλλη η κανονική είσοδος (input). Έχει δύο εξόδους. Η μία είναι η nx_state (επόμενη κατάσταση) και η άλλη, η κανονική έξοδος.

Το ακολουθιακό κύκλωμα έχει τρεις εισόδους: το ρολόι (clock), την επανάθεση (reset) και την επόμενη κατάσταση (nx_state). Επειδή όλα τα flip-flops βρίσκονται σε αυτό το τμήμα, το clock και το reset πρέπει να συνδεθεί με αυτά.




Αν η έξοδο της μηχανής εξαρτάται όχι μόνο από την παρούσα κατάσταση αλλά και από την τρέχουσα είσοδο, τότε η μηχανή ονομάζεται Mealy. Διαφορετικά, αν εξαρτάται μόνο από την τρέχουσα κατάσταση, ονομάζεται Moore.

Ο διαχωρισμός του συστήματος σε δύο μέρη μας επιτρέπει να το σχεδιάσουμε σε δύο τμήματα. Από τη σκοπιά της VHDL είναι προφανές ότι το κάτω τμήμα επειδή είναι ακολουθιακό, θα απαιτεί χρήση PROCESS, ενώ το πάνω τμήμα, επειδή είναι συνδυαστικό όχι. Ωστόσο, ακολουθιακός κώδικας μπορεί να χρησιμοποιηθεί τόσο για ακολουθιακά όσο και για συνδυαστικά κυκλώματα. Συνεπώς και το πάνω τμήμα (το συνδυαστικό) μπορεί να υλοποιηθεί με χρήση PROCESS.

Τα σήματα ρολογιού και επανάθεσης (clock και reset), συνήθως εμφανίζονται στη λίστα παραμέτρων της PROCESS του κάτω τμήματος (εκτός και αν το reset είναι σύγχρονο, ή δε χρησιμοποιείται, ή αν το WAIT χρησιμοποιείται αντί το IF). Αν υποτεθεί το reset, η pr_state θα τεθεί στην αρχική κατάσταση του συστήματος. Διαφορετικά, στην κατάλληλη ακμή ρολογιού, τα flip-flops θα αποθηκεύσουν την nx_state, οπότε τη μεταφέρουν στο κάτω τμήμα του κυκλώματος (pr_state).

Πρέπει να σημειώσουμε ότι οποιοδήποτε ακολουθιακό κύκλωμα, μπορεί να μοντελοποιηθεί ως FSM. Ωστόσο κάτι τέτοιο δεν είναι πάντα πλεονεκτικό. Ο λόγος είναι ότι ο κώδικας μπορεί να γίνει μεγάλος, πολύπλοκος και επιρρεπής σε λάθη, όπως και συμβαίνει συνήθως σε κυκλώματα με καταχωρητές π.χ. κυκλώματα μετρητών. Γενικά, η χρήση FSMs είναι κατάλληλη σε περιπτώσεις που το σύστημα είναι καλά δομημένο και οι καταστάσεις του μπορούν εύκολα να απαριθμηθούν. Συνεπώς, στις περιπτώσεις αυτές θα συναντάμε στην αρχή της ARCHITECTURE, τύπους ορισμένους από το χρήστης για

απαριθμητά δεδομένα, που θα περιέχουν όλη τη λίστα με τις καταστάσεις του συστήματος.

Όλα τα κυκλώματα που έχουν στοιχεία μνήμης δεν σημαίνει ότι είναι ακολουθιακά. Ένα παράδειγμα είναι η μνήμη τυχαίας προσπέλασης (RAM). Σε αυτή, η διαδικασία-ανάγνωσης εξαρτάται μόνο από τα bits διεύθυνσης που εφαρμόζονται κάποια στιγμή στη RAM και με την τιμή που ανακτάται τότε να μην έχει καμία σχέση με τις προηγούμενες τιμές που προσπελάστηκαν από τη μνήμη. Σε τέτοιες περιπτώσεις συνεπώς δεν είναι χρήσιμη η μοντελοποίηση με FSM.

## Παράδειγμα υλοποίησης FSM

Θα περιγράψουμε τον τρόπο σχεδίασης ενός FSM στον οποίο το ακολουθιακό τμήμα είναι πλήρως διακριτό από το συνδυαστικό και συνεπώς σχεδιάζονται ξεχωριστά. Όλες οι καταστάσεις του FSM θα πρέπει να δηλωθούν με χρήση απαριθμητού τύπου δεδομένων.

Μια τυπική δομή του ακολουθιακού κυκλώματος του FSM είναι η επόμενη:

```
PROCESS (reset, clock)
BEGIN
IF (reset='1') THEN
pr_state <= state0;
ELSIF (clock'EVENT AND clock='1')
THEN
pr_state <= nx_state;
END IF;
END PROCESS;
```

Ο κώδικας αυτός είναι πολύ απλός. Αποτελείται από μια ασύγχρονη reset που προσδιορίζει την αρχική κατάσταση του συστήματος (state0) και ακολουθεί μια σύγχρονη αποθήκευση της nx_state (στη θετική ακμή του ρολογιού) που θα παράγει την pr_state στην έξοδο output του ακολουθιακού κυκλώματος. Το καλό με αυτή την προσέγγιση είναι ότι η σχεδίαση του ακολουθιακού κυκλώματος είναι βασικά η ίδια κάθε φορά.

Ένα ακόμα πλεονέκτημα με αυτή την τεχνική σχεδίασης είναι ότι το πλήθος των καταχωρητών που απαιτούνται θα είναι το ελάχιστο. Γνωρίζουμε ότι το πλήθος των flip-flops που θα θεωρήσει ο μεταγλωττιστής ότι χρειάζονται είναι ίσο με το πλήθος των απαιτούμενων bits για την κωδικοποίηση όλων των καταστάσεων του FMS (γιατί το μόνο σήμα στο οποίο αποδίδεται τιμή στην μετάβαση ενός άλλου σήματος είναι το pr_state). Συνεπώς, αν χρησιμοποιηθεί η εξορισμού (δυαδική) κωδικοποίηση, τότε θα απαιτηθούν $[\log_2 n]$ flip-flops, όπου n το πλήθος των καταστάσεων.

Το συνδυαστικό μέρος του κυκλώματος δε χρειάζεται να γίνει με ακολουθιακό κώδικα. Μπορεί να γίνει και με παράλληλο κώδικα. Στην επόμενη υλοποίηση ωστόσο, έχουμε χρησιμοποιήσει ακολουθιακό κώδικα, με τη δήλωση CASE να παίζει τον κύριο ρόλο.

```
PROCESS (input, pr_state)
BEGIN
CASE pr_state IS
WHEN state0 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state1;
ELSE ...
END IF;
WHEN state1 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state2;
ELSE ...
END IF;
WHEN state2 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state2;
ELSE ...
END IF;
...
END CASE;
END PROCESS;
```

Ο κώδικας αυτός είναι επίσης απλός και κάνει δύο πράγματα: Πρώτον, αποδίδει στην output τιμή και δεύτερον ορίζει την επόμενη κατάσταση. Όλα τα σήματα εισόδου βρίσκονται στη λίστα παραμέτρων και όλοι οι συνδυασμοί εισόδου/εξόδου είναι προσδιορισμένοι. Επιπλέον παρατηρούμε ότι δε γίνεται απόδοση τιμής σε σήμα κατά τη μετάβαση άλλου σήματος, οπότε δεν υποτίθενται flip-flops.

Στη συνέχεια παρουσιάζουμε το πρότυπο του κώδικα στην πλήρη του μορφή:

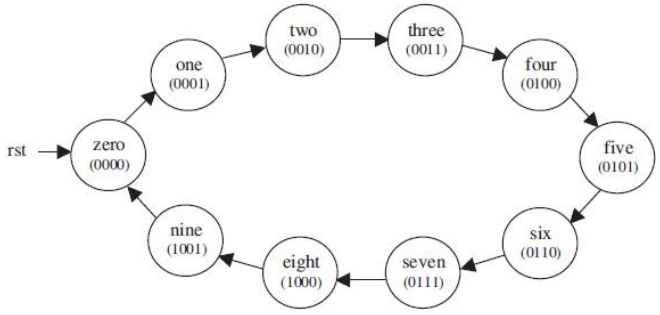
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY <entity_name> IS
PORT ( input: IN <data_type>;
reset, clock: IN STD_LOGIC;
output: OUT <data_type>);
END <entity_name>;

ARCHITECTURE <arch_name> OF
<entity_name> IS
TYPE state IS (state0, state1,
state2, state3, ...);
SIGNAL pr_state, nx_state: state;
BEGIN
---------- Lower section: ----------
PROCESS (reset, clock)
BEGIN
```

```
IF (reset='1') THEN
pr_state <= state0;
ELSIF (clock'EVENT AND clock='1')
THEN
pr_state <= nx_state;
END IF;
END PROCESS;
---------- Upper section: ----------
PROCESS (input, pr_state)
BEGIN
CASE pr_state IS
WHEN state0 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state1;
ELSE ...
END IF;
WHEN state1 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state2;
ELSE ...
END IF;
WHEN state2 =>
IF (input = ...) THEN
output <= <value>;
nx_state <= state3;
ELSE ...
END IF;
...
END CASE;
END PROCESS;
END <arch_name>;
```

## Υλοποίηση μετρητή BCD ως μηχανή πεπερασμένων καταστάσεων

Ένας μετρητής είναι παράδειγμα μηχανής Moore γιατί η έξοδός του εξαρτάται μόνο από την παρούσα κατάσταση.

Το διάγραμμα καταστάσεων φαίνεται στη συνέχεια:

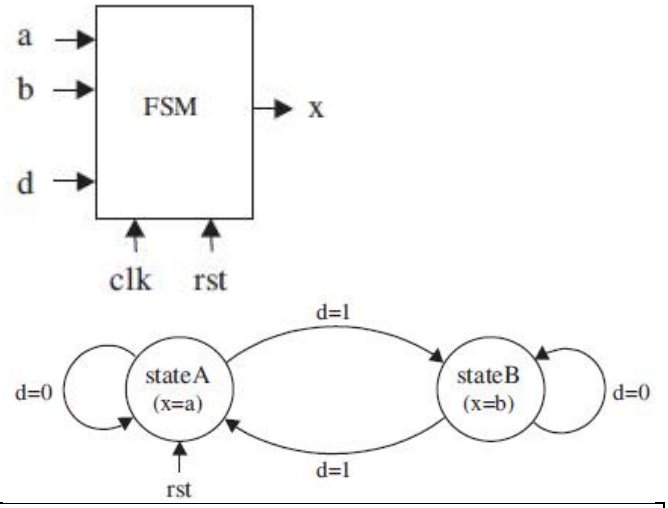



Ο κώδικας VHDL φαίνεται στη συνέχεια:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY counter IS
PORT ( clk, rst: IN STD_LOGIC;
count: OUT
STD_LOGIC_VECTOR (3 DOWNTO 0));
END counter;

ARCHITECTURE state_machine OF
counter IS
TYPE state IS (zero, one, two,
three, four,five, six, seven, eight,
nine);
SIGNAL pr_state, nx_state: state;
BEGIN

PROCESS (rst, clk)
BEGIN
IF (rst='1') THEN
pr_state <= zero;
ELSIF (clk'EVENT AND clk='1') THEN
pr_state <= nx_state;
END IF;
END PROCESS;

PROCESS (pr_state)
BEGIN
CASE pr_state IS
WHEN zero =>
count <= "0000";
nx_state <= one;
WHEN one =>
count <= "0001";
nx_state <= two;
WHEN two =>
count <= "0010";
nx_state <= three;
WHEN three =>
count <= "0011";
nx_state <= four;
WHEN four =>
count <= "0100";
nx_state <= five;
WHEN five =>
count <= "0101";
nx_state <= six;
WHEN six =>
count <= "0110";
nx_state <= seven;
WHEN seven =>
count <= "0111";
nx_state <= eight;
WHEN eight =>
count <= "1000";
nx_state <= nine;
WHEN nine =>
count <= "1001";
nx_state <= zero;
END CASE;
END PROCESS;
END state_machine;
```

## Υλοποίηση απλού FSM

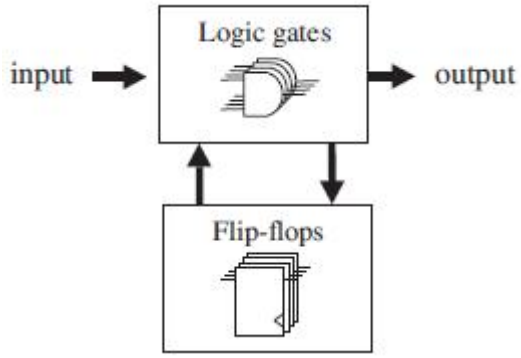
Θα υλοποιήσουμε στη συνέχεια ένα FSM με τα επόμενα χαρακτηριστικά:




```
PORT ( a, b, d, clk, rst: IN BIT;
x: OUT BIT);
END simple_fsm;

ARCHITECTURE simple_fsm OF
simple_fsm IS
TYPE state IS (stateA, stateB);
SIGNAL pr_state, nx_state: state;
BEGIN

PROCESS (rst, clk)
BEGIN
IF (rst='1') THEN
pr_state <= stateA;
ELSIF (clk'EVENT AND clk='1') THEN
pr_state <= nx_state;
END IF;
END PROCESS;

PROCESS (a, b, d, pr_state)
BEGIN
CASE pr_state IS
WHEN stateA => x <= a;
IF (d='1') THEN nx_state <= stateB;
ELSE nx_state <= stateA;
END IF;
WHEN stateB => x <= b;
IF (d='1') THEN nx_state <= stateA;
ELSE nx_state <= stateB;
END IF;
END CASE;
END PROCESS;
END simple_fsm;
```

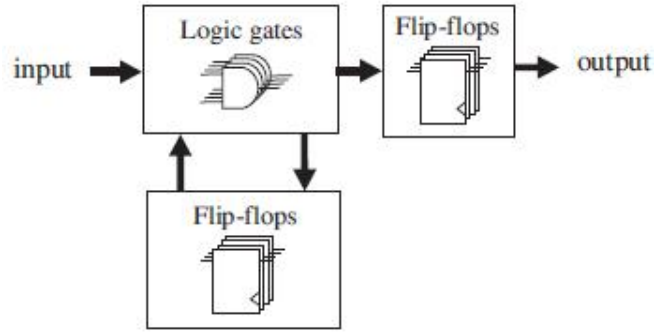
## Υλοποίηση FSM με αποθήκευση εξόδου

Στα προηγούμενα παραδείγματα είδαμε πως μόνο η pr_state χρειαζόταν να αποθηκευτεί. Δηλαδή το συνολικό κύκλωμα είχε την επόμενη δομή:




Δηλαδή ήταν μηχανή Mealy (της οποίας δηλαδή η έξοδος εξαρτάται από την τρέχουσα τιμή εισόδου), οπότε η έξοδος μπορεί να αλλάξει όταν αλλάζει η είσοδος (σύγχρονη έξοδος).

Σε πολλές εφαρμογές, τα σήματα χρειάζεται να είναι σύγχρονα, οπότε η έξοδος θα πρέπει να ανανεώνεται στην κατάλληλη ακμή του ρολογιού. Για να γίνουν οι μηχανές Mealy σύγχρονες, θα πρέπει η έξοδος να αποθηκευτεί όπως φαίνεται στην επόμενη εικόνα:




Οι απαραίτητες τροποποιήσεις φαίνονται στον επόμενο κώδικα:

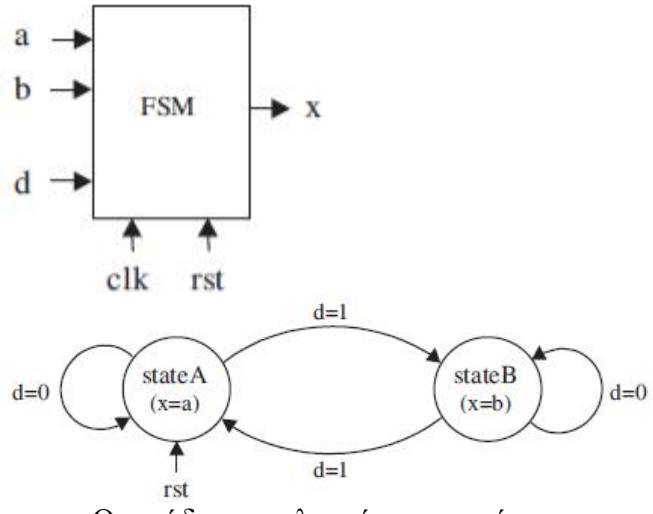
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY <ent_name> IS
PORT (input: IN <data_type>;
reset, clock: IN STD_LOGIC;
output: OUT <data_type>);
END <ent_name>;

ARCHITECTURE <arch_name> OF
<ent_name> IS
TYPE states IS (state0, state1,
state2, state3, ...);
SIGNAL pr_state, nx_state: states;
SIGNAL temp: <data_type>;
BEGIN
---------- Lower section: ----------
PROCESS (reset, clock)
BEGIN
IF (reset='1') THEN
pr_state <= state0;
ELSIF (clock'EVENT AND clock='1')
THEN
output <= temp;
pr_state <= nx_state;
END IF;
```

```
END PROCESS;
---------- Upper section: ----------
PROCESS (pr_state)
BEGIN
CASE pr_state IS
WHEN state0 =>
temp <= <value>;
IF (condition) THEN nx_state <=
state1;
...
END IF;
WHEN state1 =>
temp <= <value>;
IF (condition) THEN nx_state <=
state2;
...
END IF;
WHEN state2 =>
temp <= <value>;
IF (condition) THEN nx_state <=
state3;
...
END IF;
...
END CASE;
END PROCESS;
END <arch_name>;
```

Η διαφορά αυτού του κώδικα είναι μόνο στην εισαγωγή του εσωτερικού σήματος temp. Αυτό το σήμα χρησιμοποιείται ως αποθήκη της εξόδου της μηχανής καταστάσεων και η τιμή του περνάει στην τελική έξοδο όταν έχουμε clk/EVENT.

## Υλοποίηση απλού FSM με σύγχρονη έξοδο

Θα υλοποιήσουμε στη συνέχεια ένα FSM με τα επόμενα χαρακτηριστικά, του οποίου η έξοδος θέλουμε να είναι σύγχρονη, δηλαδή να αλλάζει όταν ανεβαίνει η ακμή του ρολογιού.




Ο κώδικας υλοποίησης φαίνεται στη συνέχεια:

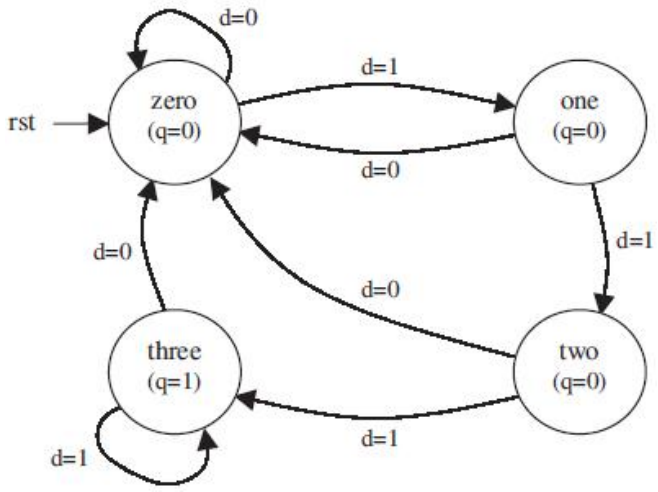
```
ENTITY simple_fsm IS
PORT ( a, b, d, clk, rst: IN BIT;
x: OUT BIT);
END simple_fsm;

ARCHITECTURE simple_fsm OF
simple_fsm IS
TYPE state IS (stateA, stateB);
SIGNAL pr_state, nx_state: state;
SIGNAL temp: BIT;
BEGIN
----- Lower section: ---------------
PROCESS (rst, clk)
BEGIN
IF (rst='1') THEN
pr_state <= stateA;
ELSIF (clk'EVENT AND clk='1') THEN
x <= temp;
pr_state <= nx_state;
END IF;
END PROCESS;
---------- Upper section: ----------
PROCESS (a, b, d, pr_state)
BEGIN
CASE pr_state IS
WHEN stateA =>
temp <= a;
IF (d='1') THEN nx_state <= stateB;
ELSE nx_state <= stateA;
END IF;
WHEN stateB =>
temp <= b;
IF (d='1') THEN nx_state <= stateA;
ELSE nx_state <= stateB;
END IF;
END CASE;
END PROCESS;
END simple_fsm;
```

Το αρχείο αναφοράς του μεταγλωττιστή θα δείξει ότι στην περίπτωση αυτή χρειάζονται δύο flip-flops, ένα για την κωδικοποίηση των καταστάσεων της μηχανής και το άλλο για την αποθήκευση της εξόδου.

## Υλοποίηση ανιχνευτή ακολουθίας δυαδικών ψηφίων (string detector)

Θα περιγράψουμε με VHDL ένα κύκλωμα μηχανή πεπερασμένων καταστάσεων που θα ανιχνεύει την ακολουθία 111. Δηλαδή η έξοδός του θα κάθε φορά 1 όταν στην είσοδο ανιχνεύεται η σειρά 111, ενώ διαφορετικά θα είναι 0.

Το διάγραμμα καταστάσεων της μηχανής φαίνεται στην επόμενη εικόνα:




Ο κώδικας υλοποίησης έχει ως εξής:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY string_detector IS
PORT ( d, clk, rst: IN BIT;
q: OUT BIT);
END string_detector;

ARCHITECTURE my_arch OF
string_detector IS
TYPE state IS (zero, one, two,
three);
SIGNAL pr_state, nx_state: state;
BEGIN
----- Lower section: ------------
PROCESS (rst, clk)
BEGIN
IF (rst='1') THEN
pr_state <= zero;
ELSIF (clk'EVENT AND clk='1') THEN
pr_state <= nx_state;
END IF;
END PROCESS;
---------- Upper section: -------
PROCESS (d, pr_state)
BEGIN
CASE pr_state IS
WHEN zero =>
q <= '0';
IF (d='1') THEN nx_state <= one;
ELSE nx_state <= zero;
END IF;
WHEN one =>
q <= '0';
IF (d='1') THEN nx_state <= two;
ELSE nx_state <= zero;
END IF;
WHEN two =>
q <= '0';
39 IF (d='1') THEN nx_state <=
three;
40 ELSE nx_state <= zero;
41 END IF;
42 WHEN three =>
43 q <= '1';
44 IF (d='0') THEN nx_state <= zero;
45 ELSE nx_state <= three;
46 END IF;
47 END CASE;
48 END PROCESS;
49 END my_arch;
```

Στο συγκεκριμένο παράδειγμα, η έξοδος δεν εξαρτάται από την τρέχουσα τιμή της εισόδου. Αυτό φαίνεται από το γεγονός ότι όλες οι αποδόσεις τιμής στην q είναι χωρίς συνθήκη (δηλαδή δεν εξαρτάται από το d). Δηλαδή η έξοδος είναι αυτόματα σύγχρονη (συμπεριφορά μηχανής Moore), οπότε δεν είναι η χρήση βοηθητικού σήματος και επιπλέον flip-flop αποθήκευσης εξόδου. Στην πραγματική του υλοποίηση το κύκλωμα αυτό χρειάζεται δύο flip-flops για να κωδικοποιηθούν οι τέσσερις καταστάσεις της μηχανής.