11. ΑΚΟΛΟΥΘΙΑΚΑ ΚΥΚΛΩΜΑΤΑ - ΜΕΛΕΤΗ ΤΟΥ SR, D, JK ΚΑΙ T FLIP-FLOP

1. Τι είναι ένα συνδυαστικό λογικό κύκλωμα;

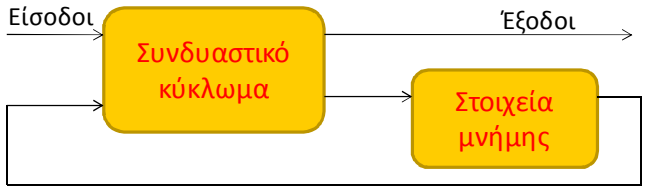
Συνδυαστικό κύκλωμα είναι εκείνο του οποίου οι έξοδοι εξαρτώνται μόνο από τις τρέχουσες εισόδους του.

2. Τί είναι ένα ακολουθιακό λογικό κύκλωμα;

Ακολουθιακό κύκλωμα είναι εκείνο του οποίου οι έξοδοι εξαρτώνται όχι μόνο από τις τρέχουσες εισόδους του, αλλά και από την προηγούμενη ακολουθία εισόδων, που ενδεχομένως εκτείνεται σε απροσδιόριστο βαθμό πίσω στο χρόνο.

Κάθε ψηφιακό σύστημα περιέχει συνήθως συνδυαστικά κυκλώματα, όμως τα περισσότερα περιέχουν επίσης και στοιχεία μνήμης, τα οποία κάνουν το όλο σύστημα να είναι ακολουθιακό (sequential). Τα στοιχεία μνήμης είναι συσκευές που αποθηκεύουν δυαδικές πληροφορίες μέσα τους.

Η σύνδεση του συνδυαστικού κυκλώματος με τα στοιχεία μνήμης γίνεται σε ένα σχηματισμό βρόγχου ανάδρασης (**Εικόνα 1**).



Εικόνα 1. Βασική δομή ακολουθιακού κυκλώματος.

3. Πώς ορίζεται η «κατάσταση» ενός ακολουθιακού κυκλώματος;

Κατάσταση (state) ενός ακολουθιακού κυκλώματος είναι μια συλλογή μεταβλητών κατάστασης (state variables) οι τιμές των οποίων σε οποιαδήποτε στιγμή περιέχουν όλες τις απαραίτητες πληροφορίες σχετικά με το παρελθόν, προκειμένου να υπολογιστεί η μελλοντική συμπεριφορά του κυκλώματος.

Στην κάθε χρονική στιγμή, οι δυαδικές πληροφορίες που είναι αποθηκευμένες στα στοιχεία μνήμης ενός ακολουθιακού κυκλώματος, αποτελούν την κατάσταση του (state).

Το ακολουθιακό κύκλωμα δέχεται πληροφορίες από τις εξωτερικές του εισόδους. Αυτές οι είσοδοι, μαζί με την παρούσα κατάσταση των στοιχείων μνήμης, καθορίζουν τις τιμές των εξόδων. Καθορίζουν επίσης το πώς θα αλλάξει η κατάσταση των στοιχείων μνήμης.

Δηλαδή, οι έξοδοι ενός ακολουθιακού κυκλώματος είναι συναρτήσεις όχι μόνο των εισόδων του, αλλά και της παρούσας κατάστασης των στοιχείων μνήμης του.

4. Τί είναι οι μηχανές πεπερασμένων καταστάσεων;

Σε ένα ψηφιακό λογικό κύκλωμα οι μεταβλητές κατάστασης είναι δυαδικές τιμές που αντιστοιχούν σε ορισμένα λογικά σήματα του κυκλώματος. Ένα κύκλωμα με n δυαδικές μεταβλητές κατάστασης έχει 2^n πιθανές καταστάσεις. Επειδή, όσο μεγάλο και αν είναι το 2^n, είναι πάντα πεπερασμένο και ποτέ άπειρο, τα ακολουθιακά κυκλώματα ονομάζονται μερικές φορές και μηχανές πεπερασμένων καταστάσεων (finite state machines).

5. Ασύγχρονο ακολουθιακό κύκλωμα

Ένα **ασύγχρονο ακολουθιακό κύκλωμα** μπορούμε να το δούμε σαν ένα **συνδυαστικό κύκλωμα με ανάδραση (feedback)**. Λόγω αυτής της ανάδρασης μεταξύ των πυλών, ένα τέτοιο κύκλωμα μπορεί μερικές φορές να γίνει ασταθές. Τέτοιες αστάθειες αποτελούν δύσκολα προβλήματα για τους σχεδιαστές. Στην πράξη, η εσωτερική καθυστέρηση των λογικών πυλών είναι αρκετή για τους σκοπούς μας και έτσι δε χρειάζεται να καταφεύγουμε σε άλλα πιο περίπλοκα στοιχεία. Σε τέτοια ασύγχρονα συστήματα, τα στοιχεία μνήμης είναι απλώς λογικές πύλες, με την καθυστέρηση που αυτές ούτως ή άλλως έχουν.

6. Σύγχρονο ακολουθιακό κύκλωμα

Ένα σύγχρονο ακολουθιακό σύστημα πρέπει, εξ' ορισμού να χρησιμοποιεί σήματα τα οποία επηρεάζουν τα στοιχεία μνήμης του, σε διακριτές μόνο στιγμές του χρόνου. Ένας τρόπος για να πετύχουμε αυτόν το σκοπό, είναι να χρησιμοποιήσουμε παντού στο σύστημα παλμούς μιας ορισμένης διάρκειας και να συμφωνήσουμε ότι οι παλμοί μιας ορισμένη πολικότητας θα παριστάνουν το λογικό 1 και της άλλης πολικότητας (ή η έλλειψη παλμού) το λογικό 0.

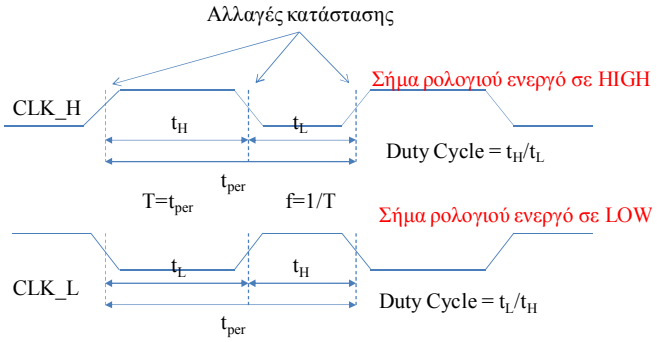
Αυτά τα κυκλώματα με τους παλμούς συγχρονισμού που εφαρμόζονται στα στοιχεία μνήμης τα λέμε **ακολουθιακά κυκλώματα με ρολόι** και είναι ο τύπος που συναντάμε συχνότερα στην πράξη. Δεν παρουσιάζουν προβλήματα

ευστάθειας και ο χρονισμός τους μπορεί να αναλυθεί σε ανεξάρτητα διακριτά βήματα, που καθένα τους μπορούμε να το εξετάσουμε ξεχωριστά.

Η δυσκολία με ένα τέτοιο σύστημα παλμών είναι ότι δύο παλμοί που φτάνουν στην ίδια πύλη προερχόμενοι από δύο διαφορετικές και ανεξάρτητες πηγές, θα έχουν υποστεί απόβλεπτές καθυστερήσεις καθ' οδόν και έτσι δε θα συμπίπτουν χρονικά με αποτέλεσμα η λειτουργία του συστήματος να μην είναι αξιόπιστη.

7. Ο ρόλος του ρολογιού στα ακολουθιακά κυκλώματα

Στην πράξη, τα σύγχρονα ακολουθιακά συστήματα χρησιμοποιούν επίπεδα τάσης (και όχι παλμούς) ως δυαδικά σήματα. Ο συγχρονισμός επιτυγχάνεται μέσω μιας γεννήτριας κύριου ρολογιού, η οποία τροφοδοτεί το σύστημα με μια περιοδική σειρά παλμών ρολογιού. Αυτοί οι παλμοί διανέμονται παντού στο σύστημα με τέτοιο τρόπο ώστε τα στοιχεία μνήμης να επηρεάζονται από τις εισόδους τους μόνο τις στιγμές που φτάνουν αυτοί οι παλμοί χρονισμού. Στην πράξη, αυτοί οι παλμοί εφαρμόζονται σε πύλες AND, μαζί με τα σήματα που καθορίζουν το πώς πρέπει να αλλαχθεί το περιεχόμενο των στοιχείων μνήμης. Έτσι, οι έξοδοι αυτών των πυλών AND μπορούν να στείλουν σήματα μόνο τις στιγμές που έρχονται οι παλμοί ρολογιού. Η **Εικόνα 2** παρουσιάζει διαγράμματα χρονισμού για τυπικά σήματα ρολογιού.

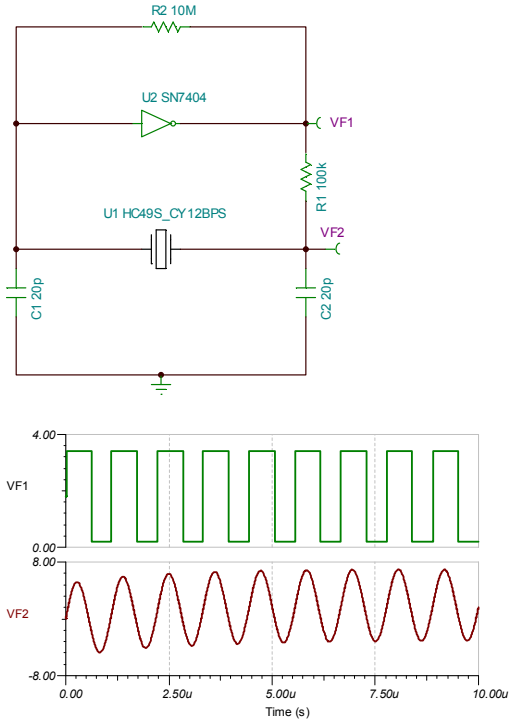


Εικόνα 2. Διαγράμματα χρονισμού για τυπικά σήματα ρολογιού.

Κατά σύμβαση, ένα σήμα ρολογιού είναι ενεργό σε HIGH αν οι μεταβολές κατάστασης σημειώνονται κατά την αύξηση της τιμής του ρολογιού ή όταν το ρολόι έχει τιμή HIGH, ενώ είναι ενεργό σε LOW στην αντίστροφη περίπτωση.

Τα τυπικά ψηφιακά κυκλώματα χρησιμοποιούν ταλαντωτές με κρυστάλλους quartz για τη δημιουργία ενός ελεύθερου σήματος ρολογιού.

Στην **Εικόνα 3** παρουσιάζεται ένα παράδειγμα προσομοίωσης παραγωγής σήματος ρολογιού με χρήση ταλαντωτή κρυστάλλου.



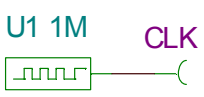
Εικόνα 3. Παράδειγμα προσομοίωσης παραγωγής σήματος ρολογιού με χρήση ταλαντωτή κρυστάλλου.

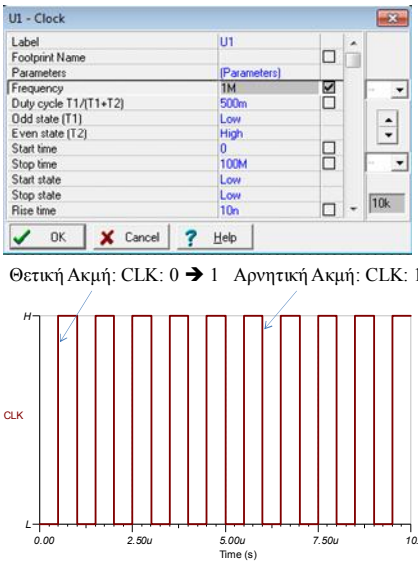
8. Ακολουθιακό κύκλωμα ανάδρασης

Το ακολουθιακό κύκλωμα ανάδρασης (feedback sequential circuit) χρησιμοποιεί συνηθισμένες πύλες και βρόχους ανάδρασης για να πετύχει τη δημιουργία μνήμης σε ένα λογικό κύκλωμα, σχηματίζοντας έτσι συνδυαστικά δομικά στοιχεία κυκλώματος, όπως κυκλώματα μανδάλωσης (latch) και δισταθή κυκλώματα (flip-flops), που χρησιμοποιούνται σε σχεδιάσεις ανώτερου επιπέδου.

9. Θετική – Αρνητική ακμή ρολογιού

Στην **Εικόνα 4** παρουσιάζεται παράδειγμα ρολογιού συχνότητας 1MHz και η θετική και η αρνητική ακμή των παλμών.





Εικόνα 4. Παράδειγμα ρολογιού συχνότητας 1MHz και η θετική και η αρνητική ακμή των παλμών.

10. Ακμοπυρόδοτο ακολουθιακό κύκλωμα

Ένα κύκλωμα λέγεται ακμοπυρόδοτο όταν ενεργοποιείται στη θετική ή στην αρνητική ακμή του ρολογιού.

11. Χρονισμένη σύγχρονη μηχανή καταστάσεων

Η χρονισμένη σύγχρονη μηχανή καταστάσεων (clocked synchronous state machine) χρησιμοποιεί ακολουθιακά κυκλώματα ανάδρασης και ιδιαίτερα ακμοπυρόδοτα Flip-Flops τύπου D, για τη δημιουργία κυκλωμάτων οι είσοδοι των οποίων εξετάζονται και οι έξοδοι των οποίων μεταβάλλονται σύμφωνα με ένα σήμα ρολογιού που ελέγχει την όλη διαδικασία.

12. Κύκλωμα μανδάλωσης

Χρησιμοποιούμε την ονομασία κύκλωμα μανδάλωσης (latch) για μια ακολουθιακή συσκευή η οποία παρακολουθεί όλες τις εισόδους της συνεχώς και μεταβάλλει τις εξόδους της οποιαδήποτε στιγμή, χωρίς να εξαρτάται από κάποιο σήμα ρολογιού.

13. Flip-Flops

Χρησιμοποιούμε την ονομασία flip-flop για μια ακολουθιακή συσκευή η οποία κατά κανόνα εκτελεί δειγματοληψία στις εισόδους της και μεταβάλλει τις εξόδους της μόνο κατά διαστήματα που καθορίζονται από ένα σήμα ρολογιού.

Τα στοιχεία μνήμης που χρησιμοποιούνται στα ακολουθιακά κυκλώματα με ρολόι, λέγονται flip-flops. Πρόκειται για δυαδικά κύτταρα που μπορούν να αποθηκεύσουν ένα bit πληροφορίας.

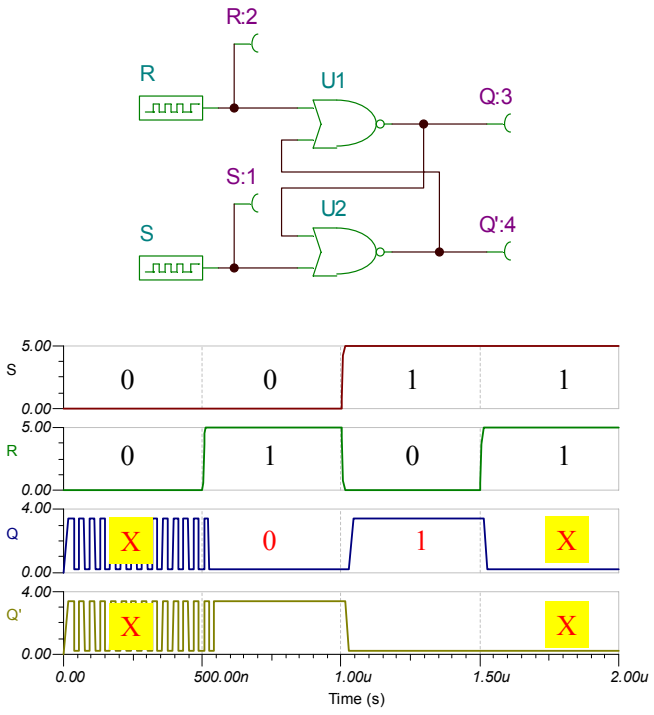
Τα flip-flops έχουν συνήθως δύο εξόδους – μια για την τιμή του bit που είναι αποθηκευμένο μέσα τους και μια για το συμπλήρωμά της.

Δυαδικές πληροφορίες μπορούν να μπουν στο flip-flop με διάφορούς τρόπους και έτσι έχουμε διάφορους τύπου flip-flop.

Ένα κύκλωμα flip-flop μπορεί να διατηρηθεί σε μια δυαδική κατάσταση επ' αόριστο (εφόσον το τροφοδοτούμε με ισχύ), έως ότου κάποιο σήμα εισόδου το κάνει να αλλάξει κατάσταση.

Οι σπουδαιότερες διαφορές ανάμεσα στους διάφορους τύπου flip-flop είναι ο αριθμός των εισόδων που έχουν και ο τρόπος με τον οποίο αυτές οι είσοδοι επηρεάζουν της δυαδική τους κατάσταση.

14. Μανδαλωτής SR



Εικόνα 5. Βασικό κύκλωμα μανδαλωτή SR με πύλες NOR.

Στην **Εικόνα 5** παρουσιάζεται η τοπολογία ενός μανδαλωτή SR με πύλες NOR. Το κύκλωμα αυτό ονομάζεται επίσης μανδαλωτής RS-άμεσης-σύζευξης. Οι συνδέσεις χιαστί από την έξοδο κάθε πύλης στην είσοδο της άλλης αποτελούν ένα βρόγχο ανάδρασης και για το λόγο αυτό τα κυκλώματα αυτά κατατάσσονται ως ασύγχρονα

ακολουθιακά κυκλώματα. Παρουσιάζεται επίσης και ένα παράδειγμα χρονισμού του κυκλώματος.

Ο πίνακας αληθείας/καταστάσεων του μανδαλωτή SR φαίνεται στη συνέχεια.

S	R	Q^+	$Q^{+\prime}$
0	0	Q	Q’
0	1	0	1
1	0	1	0
1	1	**0(X)**	**0(X)**

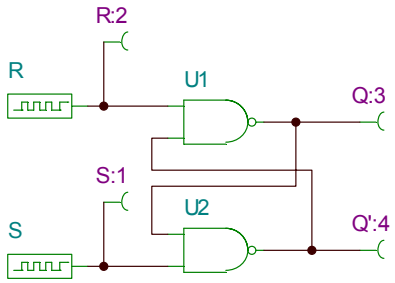
Πίνακας 1. Πίνακας αληθείας/καταστάσεων του μανδαλωτή SR.

Αν **S=1,R=1**, τότε **Q=0,Q’=0**. Αυτό αντιφάσκει με το γεγονός ότι η μια έξοδος είναι συμπληρωματική της άλλης. **Υπό κανονική λειτουργία αυτή η κατάσταση θα πρέπει να αποφεύγεται, δηλαδή θα πρέπει να φροντίζουμε να μη γίνονται ποτέ και οι δύο είσοδοι ταυτόχρονα 1**. Για το λόγο αυτό στο σχήμα σημειώνουμε X στις θέσεις αυτές.

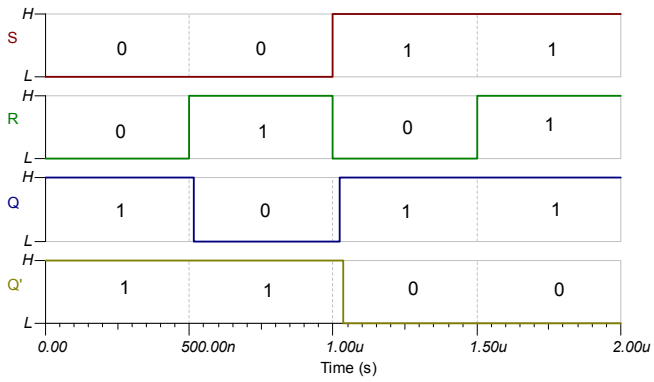
Αν **Q=1,Q’=0** αυτή είναι η κατάσταση θέσης ή **κατάσταση 1**.

Αν **Q=0,Q’=1** αυτή είναι η κατάσταση μηδένισης ή **κατάσταση 0**.

Αν δώσουμε την τιμή 0 και στις δύο εισόδους ταυτόχρονα, το κύκλωμα μανδάλωσης θα περιέλθει σε μια απρόβλεπτη επόμενη κατάσταση και μπορεί στην πραγματικότητα να αρχίσει να ταλαντώνεται ή να περιέλθει στη **μετασταθή κατάσταση**. Μεταστάθεια μπορεί να προκύψει επίσης αν εφαρμοστεί στο S ή στο R ένας πολύ βραχύς παλμός.



Εικόνα 6. Βασικό κύκλωμα μανδαλωτή SR με πύλες NAND.



Εικόνα 7. Διάγραμμα χρονισμού μανδαλωτή SR.

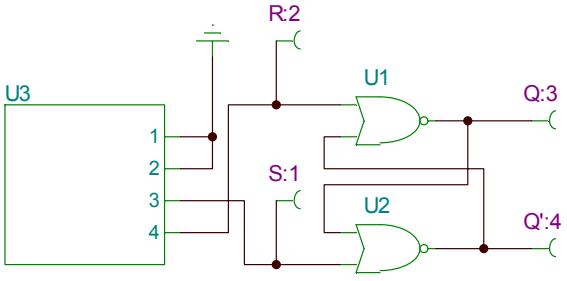
Η εφαρμογή ενός στιγμιαίου 1 στην είσοδο θέσης φέρνει το flip-flop στην κατάσταση θέσης. Μετά η είσοδος θέσης πρέπει να γυρίσει στο 0 πρωτού εφαρμόσουμε 1 στην είσοδο επαναφοράς. Ένα στιγμιαίο 1 στην είσοδο επαναφοράς φέρνει το SR στην κατάσταση μηδένισης. Στην **Εικόνα 6** παρουσιάζεται υλοποίηση του SR-μανδαλωτή με πύλες NAND καθώς και παράδειγμα χρονισμού στην **Εικόνα 7**.

Από το προηγούμενο διάγραμμα χρονισμού βλέπουμε ότι προκύπτει ο επόμενος πίνακας αληθείας/καταστάσεων (**Πίνακας 2**). Εδώ παρατηρούμε ότι το πρόβλημα εμφανίζεται στο συνδυασμό SR=00 με QQ’=11. Δηλαδή και πάλι ισχύει το ότι **θα πρέπει να αποφεύγουμε οι δύο είσοδοι S, R να είναι ταυτόχρονα ίδιες με 0**.

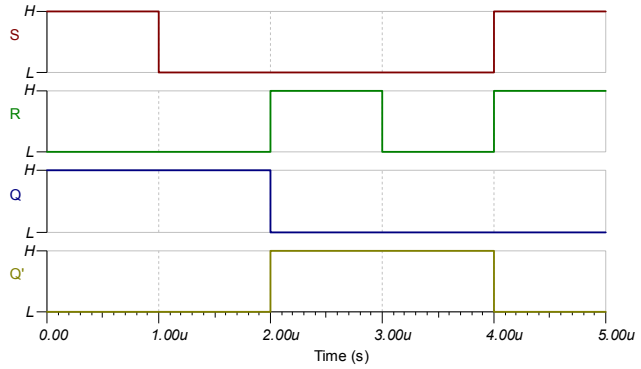
S	R	Q	Q’
0	0	1 (X)	1 (X)
0	1	0	1
1	0	1	0
1	1	1	0

Πίνακας 2. Πίνακας καταστάσεων μανδαλωτή SR με πύλες NAND.

Παράδειγμα 1. Σχεδιάστε το επόμενο κύκλωμα του SR-FF (Εικόνα 8) και προσομοιώστε τη λειτουργία του στο TINA χρησιμοποιώντας γεννήτρια παλμών 4bits (Εικόνα 9).



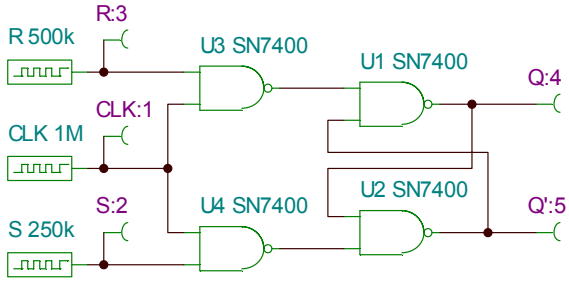
Εικόνα 8.



Εικόνα 9.

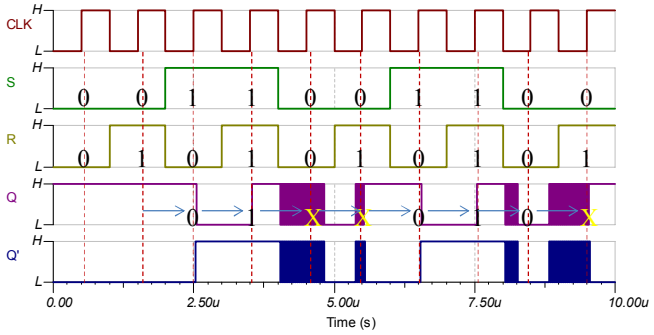
15. RS Μανδαλωτής με επίτρεψη – ρολόι = RS Flip-Flop

Η λειτουργία του βασικού μανδαλωτή RS ελέγχει σε όλες τις χρονικές στιγμές τις εισόδους S και R. Μπορεί όμως να τροποποιηθεί με την τοποθέτηση μιας πρόσθετης εισόδου ελέγχου που να καθορίζει το πότε πρόκειται να αλλαχθεί η κατάσταση του κυκλώματος. Το κύκλωμα που προκύπτει θα είναι το **RS flip-flop**. Ο παλμός εισόδου (του ρολογιού) παίζει το ρόλο σήματος επίτρεψης για τις άλλες δύο εισόδους. Η **Εικόνα 10** δείχνει την τοπολογία του κυκλώματος.



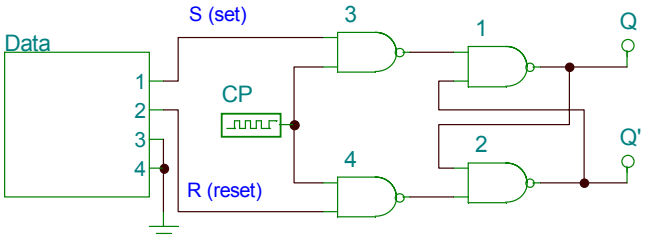
Εικόνα 10. Flip-Flop RS.

- CP=0 : κατάσταση ηρεμίας. Διατηρείται η προηγούμενη κατάσταση. Δηλαδή $Q^+=Q$.
- S=0, R=0, CP=1 : απροσδιόριστη κατάσταση. $Q^+=X$.
- S=1, R=0, CP=1 : κατάσταση θέσης. $Q^+=1$.
- S=0, R=1, CP=1 : κατάσταση επαναφοράς. $Q^+=0$.
- S=1, R=1, CP=1 : απροσδιόριστη κατάσταση. $Q^+=X$.
- Στην αλληλουχία διαγραμμάτων χρονισμού (**Εικόνα 11**) φαίνεται αναλυτικά η χρονική ανάλυση του flip-flop RS (με ενεργοποίηση του Flip-Flop στη θετική ακμή του ρολογιού).

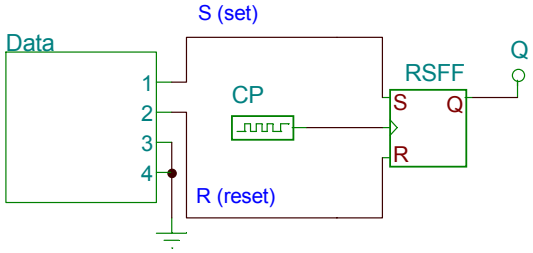


Εικόνα 6. Χρονική ανάλυση του κυκλώματος RS flip-flop.

Το SR-FF υλοποιημένο με πύλες και πηγή σημάτων 4bit του TINA φαίνεται στην **Εικόνα 12**. Στην **Εικόνα 13** φαίνεται το SR του TINA (από την παλέτα εργαλείων Flip-Flops).

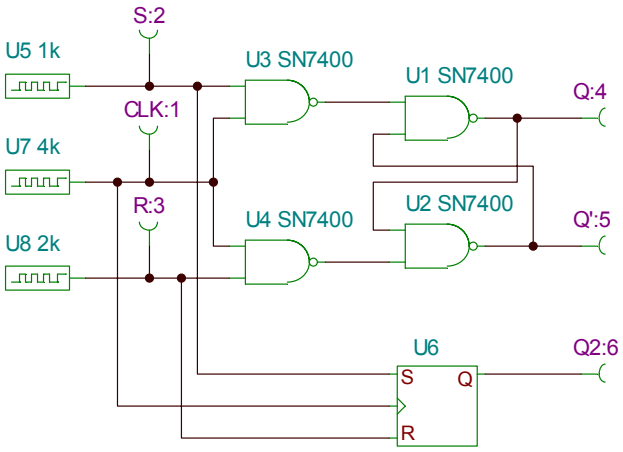


Εικόνα 7. SR-Flip-Flop με ρολόι υλοποιημένο με πύλες NAND.

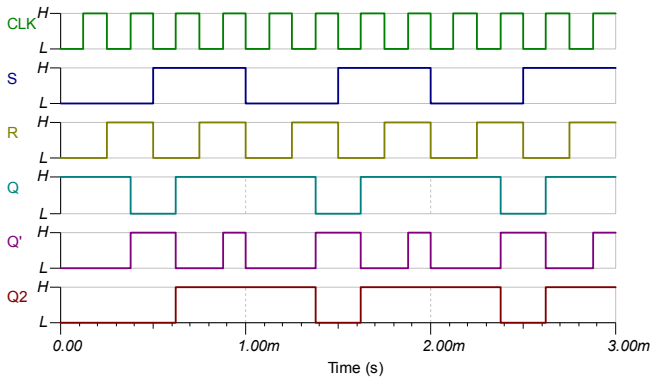


Εικόνα 8. Το SR-Flip-Flop με ρολόι του TINA.

Παράδειγμα 1. Προσομοιώστε στο TINA τη δομή SR-FF με ρολόι. Χρησιμοποιήστε το SR-FF που παρέχει το TINA αλλά και την αναλυτική δομή του με πύλες NAND (Εικόνα 14) και συγκρίνετε τα αποτελέσματα της προσομοίωσης (Εικόνα 15).



Εικόνα 9. Το κύκλωμα για την προσομοίωση.

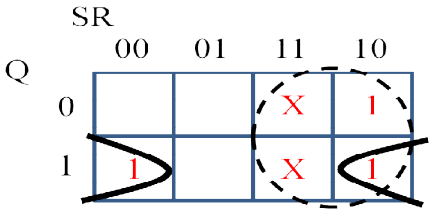


Εικόνα 10. Αποτελέσματα προσομοίωσης.

Παράδειγμα 2. Επαληθεύστε τον πίνακα καταστάσεων (Πίνακας 3) και τη συνάρτηση επόμενης κατάστασης (Εικόνα 16) του SR-FF.

Πίνακας 3. Πλήρης πίνακας καταστάσεων SR-FF.

S	**R**	**Q**	**Q^+**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

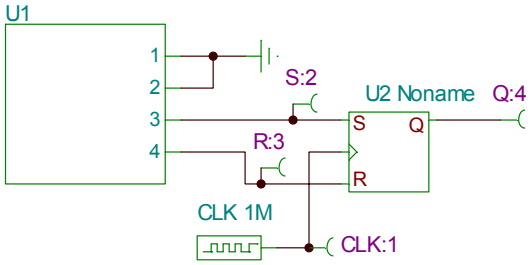


Εικόνα 11. Χάρτης Karnaugh πίνακα καταστάσεων SR-FF.

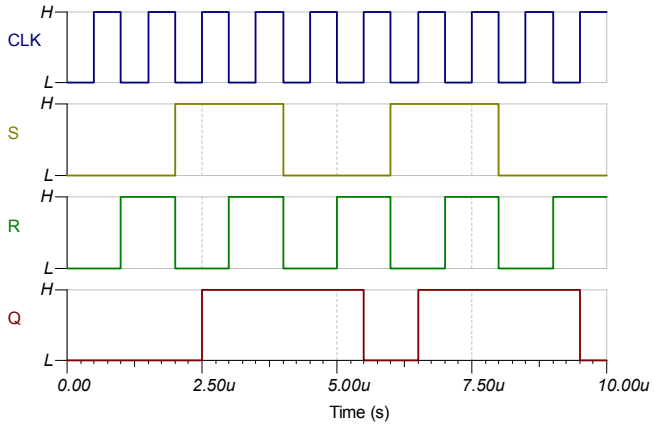
Όπως προκύπτει από τον πίνακα καταστάσεων και το χάρτη Karnaugh, η εξίσωση της επόμενης κατάστασης είναι:
$Q^+ = S + R'Q$ με $S R = 0$

(Ως τμήμα της χαρακτηριστική εξίσωσης του RS flip-flop πρέπει να συμπεριλάβουμε και τη συνθήκη SR=0, που υποδεικνύει ότι δε μπορεί να είναι ταυτόχρονα S=1,R=1.)

Παράδειγμα 3. Δίνεται το επόμενο κύκλωμα (Εικόνα 17) και η χρονική ακολουθία μεταβολών (Εικόνα 18). a) Σε ποια ακμή του ρολογιού ενεργοποιείται το κύκλωμα (δήλαδή είναι θετικής ή αρνητικής πυροδότησης). b) Πώς μπορώ να το κάνω να δουλεύει στην αντίθετη ακμή;



Εικόνα 12.



Εικόνα 13.

16. D-FLIP-FLOP

Ένας τρόπος εξάλειξης της ανεπιθύμητης συμπεριφοράς στην απροσδιόριστη κατάσταση ενός RS flip-flop είναι να εξασφαλιστεί ότι οι είσοδοι R και S δεν είναι ποτέ ταυτόχρονα 1.

Αυτό γίνεται με το D-flip-flop. Έχει δύο μόνο εισόδους : D, CP.

- Η είσοδος D περνάει όταν CP=1.
- Αν D=1, CP=1 τότε Q=1 : κατάσταση θέσης
- Αν D=0, CP=1 τότε Q=0 : κατάσταση επαναφοράς

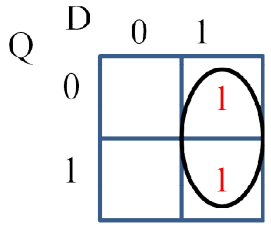
Το D-flip-flop παίρνει το όνομά του από την ιδιότητά του να κρατά δεδομένα (Data). Ονομάζεται και φυλασσόμενος μανδαλωτής τύπου D.

Ο πίνακας καταστάσεων και ο χάρτης Karnaugh για το D-FF φαίνονται στη συνέχεια.

D	**Q**	**Q^+**
0	0	0

0	1	0
1	0	1
1	1	1

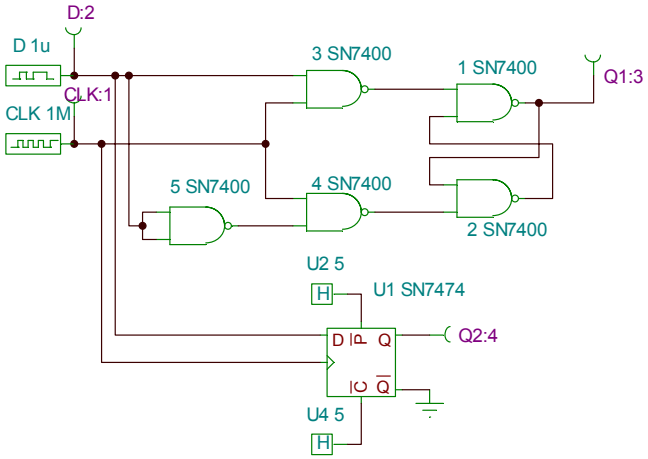
Πίνακας 4. Πίνακας καταστάσεων D-Flip-Flop.



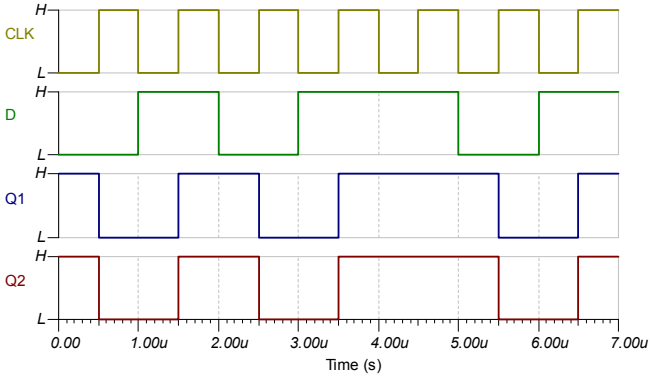
Εικόνα 19. Χάρτης καταστάσεων D-Flip-Flop.

Η συνάρτηση της επόμενης κατάστασης για το D-FF είναι συνεπώς:
$Q^+ = D$

Παράδειγμα 1. Προσομοιώστε στο TINA τη δομή DFF με ρολόι. Χρησιμοποιήστε το D-FF που παρέχει το TINA αλλά και την αναλυτική δομή του με πύλες NAND και συγκρίνετε τα αποτελέσματα της προσομοίωσης. (Εικόνα 20, 21).

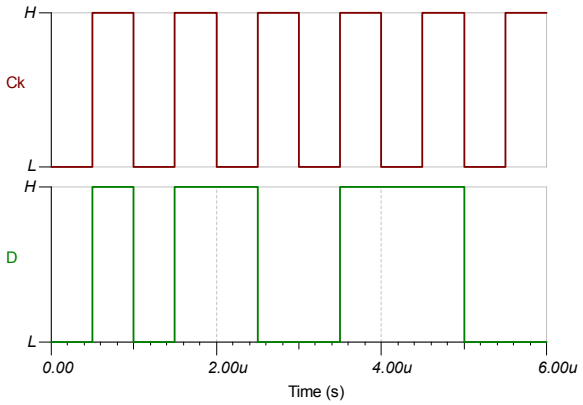


Εικόνα 20. Δομή του D-Flip-Flop με ρολόι, με πύλες NAND και υο D-Flip-Flop του TINA.

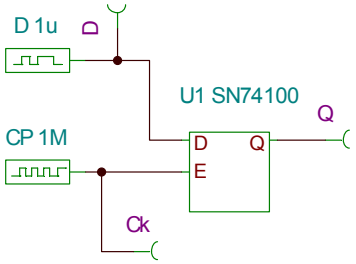


Εικόνα 21. Συγκριτικά αποτελέσματα προσομοίωσης.

Παράδειγμα 2. Θεωρήστε τις χρονικές ακολουθίες παλμών της Εικόνας 22, Αν το D-FF της Εικόνας 23 είναι θετικής πυροδότησης, να σχεδιάσετε την αντίστοιχη έξοδο Q.

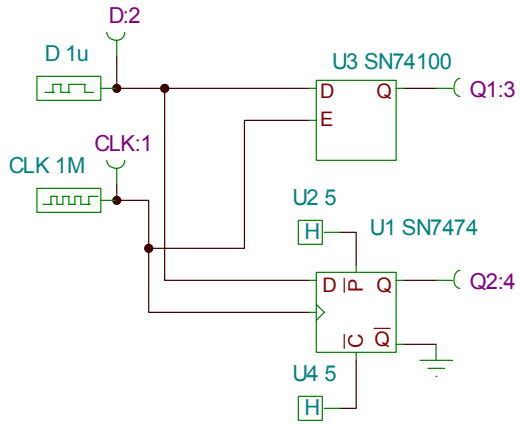


Εικόνα 22.

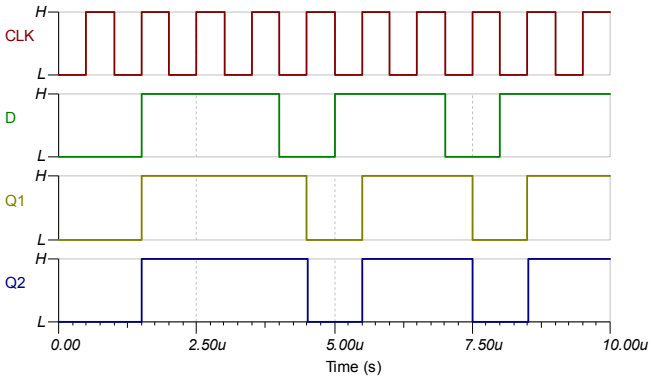


Εικόνα 23.

Παράδειγμα 3. Ελέξτε το D-FF και το D-Latch στο TINA. (Εικόνα 24, 25).



Εικόνα 24.



Εικόνα 25.

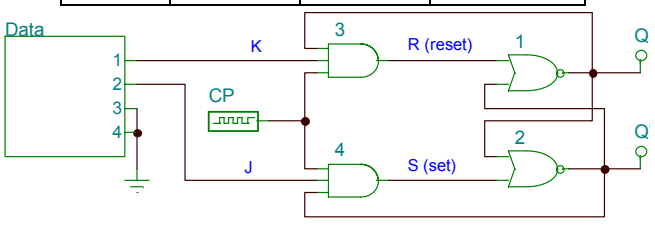
17. JK-FLIP-FLOP

Το JK-flip-flop είναι μια εξελιγμένη μορφή του RS-flip-flop με την έννοια ότι η απροσδιόριστη κατάσταση που έχει το τύπου RS, προσδιορίζεται στον τύπο JK. Οι είσοδος J, K συμπεριφέρονται σας τις R, S. Αν όμως διεγείρουμε ταυτόχρονα και τις δύο, τότε το JK αλλάζει κατάσταση και πάει στη συμπληρωματική αυτής στην οποία ήταν.

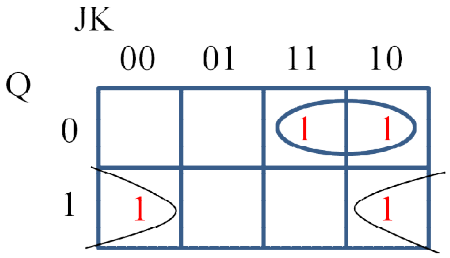
- J,K αν είναι 0,0 δεν αλλάζει κατάσταση δηλαδή Q(t+1)=Q
- J,K αν είναι 0,1 κάνει reset δηλαδή Q(t+1)=0
- J,K αν είναι 1,0 κάνει set δηλαδή Q(t+1)=1
- J,K αν είναι 1,1 κάνει toggle (τούμπα) δηλαδή Q(t+1)=Q'

Πίνακας 5. Πίνακας καταστάσεων JK-FF.

J	**K**	**Q**	**Q^+**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Εικόνα 26. Το βασικό κύκλωμα του JK-FF.



Εικόνα 27. Συνάρτηση επόμενης κατάστασης JK-FF.

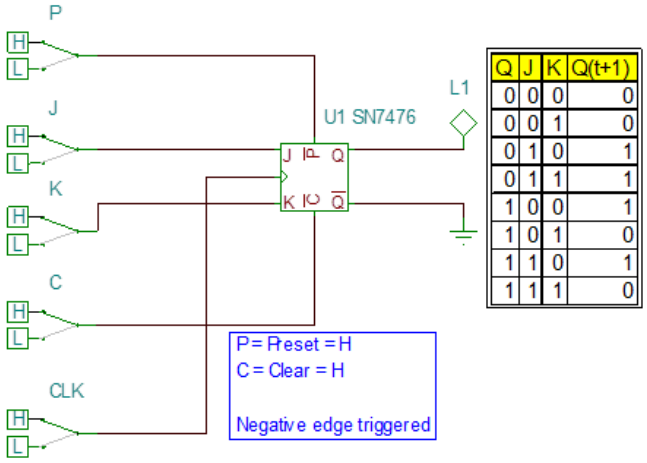
Η συνάρτηση επόμενης κατάστασης είναι:

Q(t+1)=JQ'+K'Q.

Λόγω της σύνδεσης ανάδρασης που υπάρχει στο JK-flip-flop, αν J=1 και το σήμα CP μείνει 1 για αρκετό χρόνο – αφού οι έξοδοι αντιστραφούν μια φορά, οι έξοδοι θα ξαναντιστραφούν και πάλι κ.ο.κ. μέχρις ότου ο παλμός CP γίνει 0. **Για να αποφύγουμε αυτόν τον ανεπιθύμητο τρόπο λειτουργίας, οι παλμοί του ρολογιού πρέπει να έχουν χρονική διάρκεια μικρότερη από την καθυστέρηση διάδοσης των σημάτων μέσω των flip-flop**. Αυτή η προδιαγραφή είναι υπερβολικά περιοριστική, αφού κάνει τα λειτουργία του κυκλώματος να εξαρτάται και μάλιστα κατά τρόπο κρίσιμο, από το πλάτος των παλμών. **Για το λόγο αυτό τα JK-flip-flop δεν κατασκευάζονται ποτέ όπως δείχνει το σχήμα, αλλά με τεχνικές αφέντη-σκλάβου ή ακμοπυροδότησης**.

Τα ίδια ισχύουν και για το T-flip-flop που θα δούμε στη συνέχεια.

Παράδειγμα 1. Εξακριβώστε τις συνθήκες λειτουργίας (θέσεις των P και C) καθώς και την ακμή πυροδότησης τους JK-FF SN7476. Πώς μπορούμε να αλλάξουμε εύκολα την ακμή πυροδότησης;



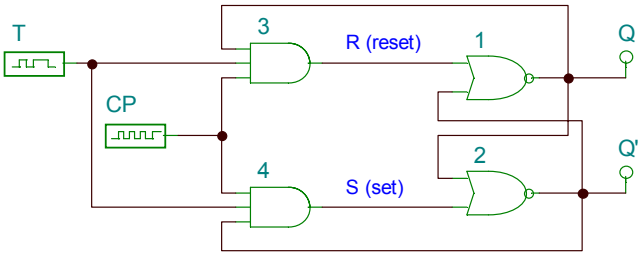
Εικόνα 28. Κύκλωμα διαδραστικής προσομοίωσης JK-FF.

18. T-FLIP-FLOP

Η ονομασία T-FF προέρχεται από τη δυνατότητα του flip-flop να αντιστρέφεται (toggle), δηλαδή να αλλάζει κατάσταση. Σε όποια κατάσταση και να βρίσκεται το flip-flop, όταν έλθει ο παλμός του ρολογιού ενώ T=1 πηγαίνει στη συμπληρωματική κατάσταση.Όταν T=0, Q(t+1)=Q, δηλαδή η επόμενη κατάσταση είναι ίδια με την παρούσα και καμιά αλλαγή δε συμβαίνει.

T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

Πίνακας 6. Πίνακας καταστάσεων T-FF.



Εικόνα 29. Το T-FF είναι ένα JK-FF με βραχυκυκλωμένες τις JK εισόδους του.

Q \ T	0	1
0		1
1	1	

Εικόνα 14. Συνάρτηση επόμενης κατάστασης T-FF.

Η συνάρτηση επόμενης κατάστασης για το T-FF είναι:
$Q^+ = T'Q + TQ'$

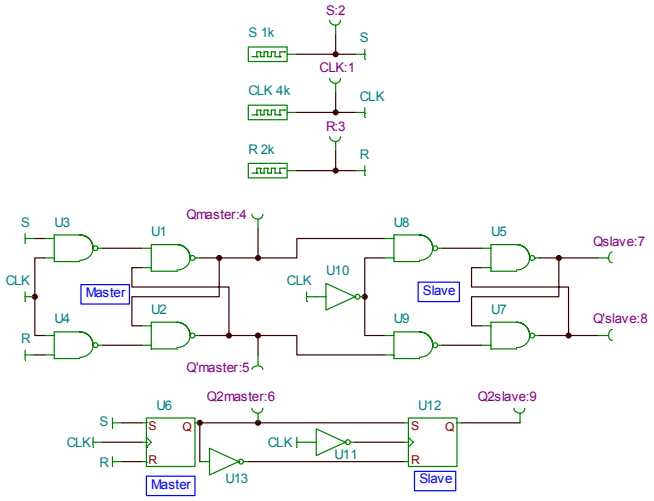
19. Πυροδότηση των FLIP-FLOPS – Τεχνική αφέντη - σκλάβου

Ένας τρόπος να κάνουμε το FF να αντιδρά μόνο στις ακμές των παλμών είναι να χρησιμοποιήσουμε χωρητική σύζευξη. Αυτό σημαίνει να βάλουμε ένα κύκλωμα RC (αντίσταση - πυκνωτής) στην είσοδο ρολογιού, το οποίο και θα δημιουργεί έναν σπινθήρα όποτε αλλάζει το σήμα εισόδου - με τη θετική ακμή δίνει ένα θετικό σπινθήρα και με την αρνητική ακμή δίνει αρνητικό σπινθήρα. Αν σχεδιάσουμε το FF έτσι ώστε να αγνοεί τον ένα σπινθήρα και να πυροδοτείται από τον άλλο, τότε θα έχουμε πετύχει την **ακμοπυροδότηση** (edge triggering).

Άλλοι τρόποι να πετύχουμε τον ίδιο σκοπό είναι οι τεχνικές των FF **αφέντη-σκλάβου (master-slave)** και των ακμοπυρόδοτων FF. Το FF αφέντη-σκλάβου περιέχει μέσα του **δύο από τα απλά FF**, το ένα από τα οποία εκτελεί χρέη αφέντη (master), ενώ το άλλο εκτελεί χρέη σκλάβου (slave).

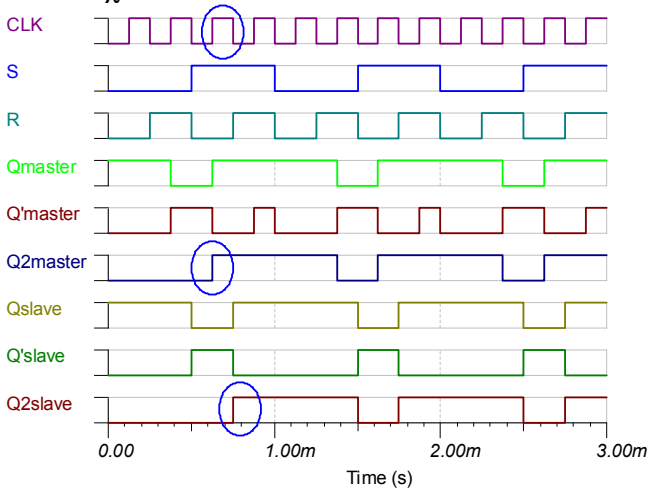
Στα FF αφέντη-σκλάβου μπορούμε να χρησιμοποιήσουμε για όλα το ίδιο ρολόι και να έχουμε και τις εισόδους και τις εξόδους να αλλάζουν την ίδια στιγμή. Για παράδειγμα η είσοδος S του FF μπορεί να έρχεται από την έξοδο Q ενός άλλου ίδιο FF που αλλάζει με το ίδιο ρολόι.

Η **Εικόνα 31** δείχνει παράδειγμα αφέντη – σκλάβου για την περίπτωση SR-FF.



Εικόνα 31. SR-FF τύπου αφέντη σκλάβου.

Η χρονική προσομοίωση του παραπάνω κυκλώματος φαίνεται στην **Εικόνα 32**. Οι βοηθητικοί κύκλοι δείχνουν ένα παράδειγμα παλμού και τις τιμές του Qmaster και Qslave να αλλάζουν στη θετική και στην αρνητική ακμή αντίστοιχα.

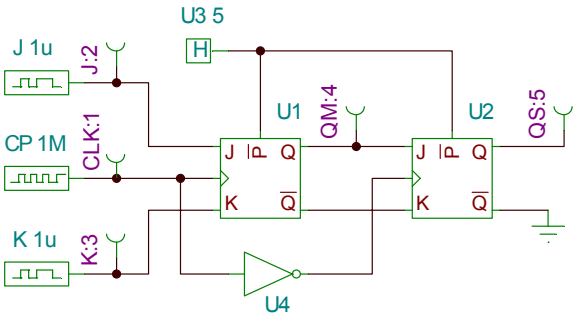


Εικόνα 32. Χρονική προσομοίωση του κυκλώματος SR-FF τύπου αφέντη σκλάβου.

Στα FF αφέντη-σκλάβου, οι αλλαγές της κατάστασής μπορεί να ρυθμιστούν να γίνονται με την αρνητική ή τη θετική ακμή του παλμού του ρολογιού, με κατάλληλη χρήση αντιστροφέα στην είσοδο ρολογιού.

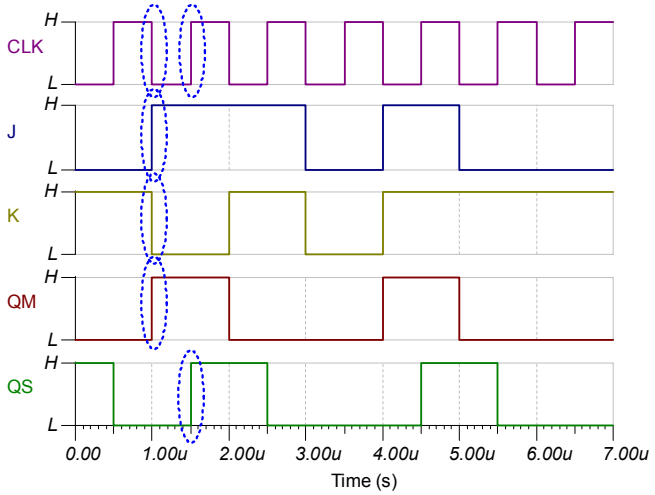
Ο συνδυασμός αφέντη-σκλάβου μπορεί να κατασκευαστεί για κάθε τύπο FF προσθέτοντας του ένα FF τύπου RS με αντιστραμένο ρολόι που αποτελεί το σκλάβο.

Στο παράδειγμα στην **Εικόνα 33** φαίνεται η σύνδεση δύο JK-FF σε ρόλο αφέντη – σκλάβου. Παρατηρούμε και πάλι τη χρήση του αντιστροφέα στην είσοδο του δεύτερου FF.



Εικόνα 33. JK-FF συνδεσμολογίας αφέντη-σκλάβου.

Η **Εικόνα 34** δείχνει μια χρονική ανάλυση του προηγούμενου κυκλώματος. Παρατηρούμε ότι ενώ το QM αλλάζει στην αρνητική ακμή του ρολογιού μαζί με τα J και K, το QS παρακολουθεί το QM αλλά με αλλαγή στην επόμενη θετική ακμή του ρολογιού.



Εικόνα 34. Παράδειγμα χρονικής ανάλυσης JK-FF συνδεσμολογίας αφέντη-σκλάβου.

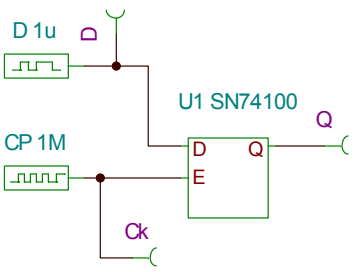
Ας θεωρήσουμε τώρα ένα ψηφιακό σύστημα που περιέχει πολλά FF αφέντη - σκλάβου και όπου οι έξοδοι μερικών από αυτά πηγαίνουν στις εισόδους άλλων. Υποθέστε ότι οι παλμοί του ρολογιού που φτάνουν σε όλα τα FF είναι συγχρονισμένοι (δηλαδή φτάνουν όλοι την ίδια στιγμή). Στην αρχή κάθε παλμου ρολογιού, μερικοί από τους αφέντες αλλάζουν κατάσταση, αλλά όλοι οι σκλάβου και άρα οι έξοδοι όλων των FF αλλάζουν μένουν στις προηγούμενες τιμές τους.

Μόλις ο παλμός του ρολογιού επιστρέψει στο 0, μερικές από τις εξόδους αλλάζουν κατάσταση, αλλά καμία από αυτές τις νέες καταστάσεις δεν επηρεάζει κανένα από τους αφέντες - εκτός μόνο με τον επόμενο παλμό ρολογιού. Έτσι, οι καταστάσεις των FF στο σύστημα μπορούν να αλλαχθούν ταυτόχρονα με τον ίδιο παλμό ρολογιού, έστω και αν οι έξοδοι κάποιων FF συνδέονται στις εισόδους άλλων. Αυτό είναι δυνατό διότι η νέα κατάσταση εμφανίζεται στις εξόδους των FF μόνο αφού ο παλμός του ρολογιού επιστρέψει στο 0.

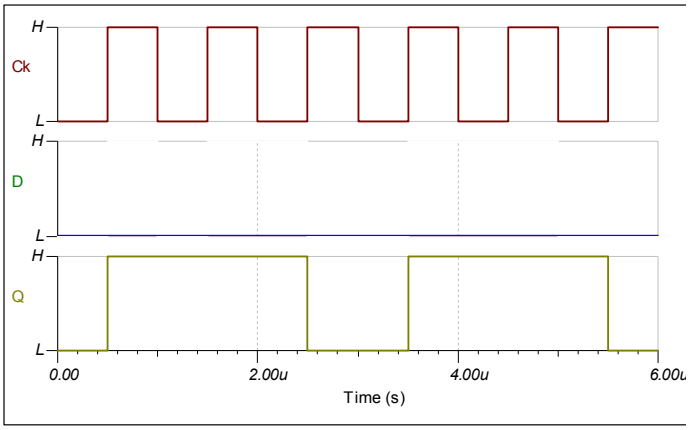
Έτσι το περιεχόμενο ενός FF μπορεί να μεταφερθεί σε ένα δεύτερο, και του δεύτερου στο πρώτο και μπορούν να γίνουν και οι δύο μεταφορές ταυτόχρονα με τον ίδιο παλμό ρολογιού.

20. Ασκήσεις

1. **Διατυπώστε τον πίνακα αληθείας του JK-FF. Ποια είναι η λογική συνάρτηση που υλοποιεί (αφού γίνει απλοποίηση με το χάρτη Karnaugh); Σχεδιάστε το κύκλωμα (με πύλες του βασικού JK-FF) και προσομοιώστε τη λειτουργία του. Ποια η χρήση του Preset και Clear του ολοκληρωμένου κυκλώματος 7476 που περιέχεται στο TINA; Ποιες τιμές πρέπει να έχουν αυτές οι είσοδοι για να λειτουργεί η διάταξη όπως προβλέπει ο πίνακας αληθείας της;**
2. **Να διατυπώσετε τον πίνακα αληθείας, τη λογική συνάρτηση και το σύμβολο για το D-FF και το T-FF.**
3. **Επαληθεύστε τη λειτουργία του D-FF με κατάλληλη τροποποίηση του JK-FF .**
4. **Να συμπληρώσετε στο διάγραμμα χρονισμού του D-FF (Εικόνα 35) τις εναλλαγές στην είσοδο D (Εικόνα 36).**

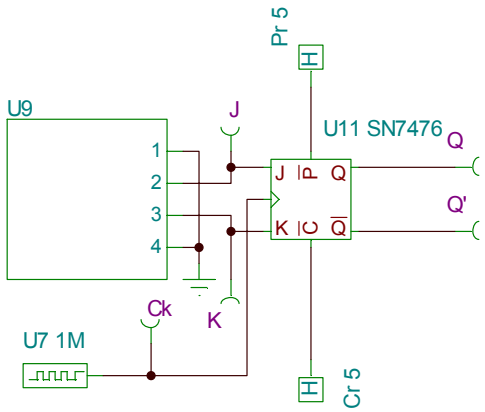


Εικόνα 35.

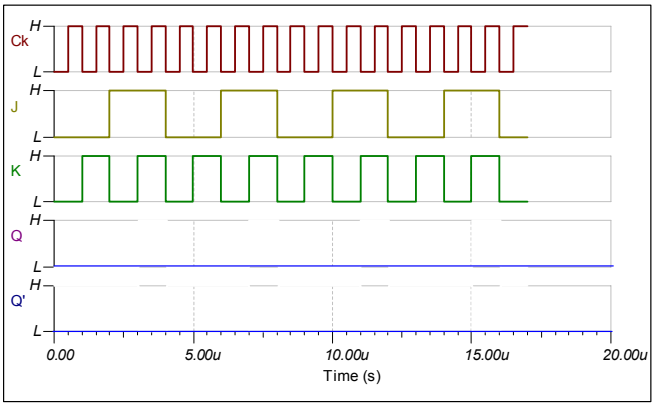


Εικόνα 36.

5. **Επαληθεύστε τη λειτουργία του T-FF με κατάλληλη τροποποίηση του JK-FF.**
6. **Να συμπληρώσετε στο διάγραμμα χρονισμού του JK-FF (Εικόνα 37) τις εναλλαγές στις εξόδους Q και Q' (Εικόνα 38).**

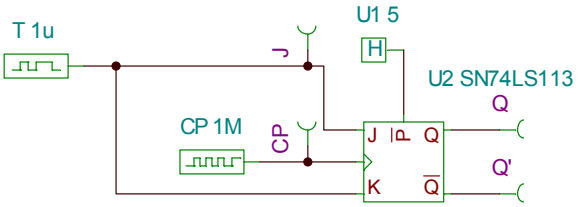


Εικόνα 37.

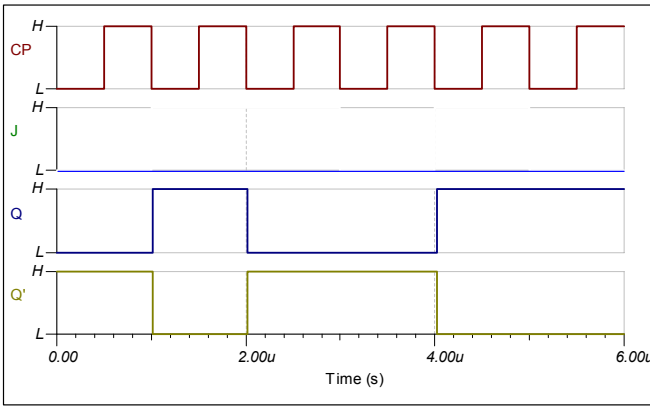


Εικόνα 38.

7. **Να συμπληρώσετε στο διάγραμμα χρονισμού του T-FF (Εικόνα 39) τις εναλλαγές στην είσοδο J (Εικόνα 40).**

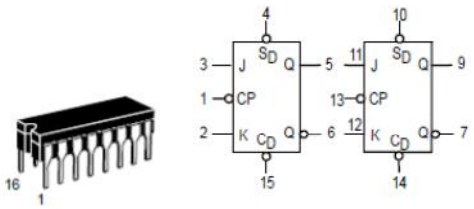


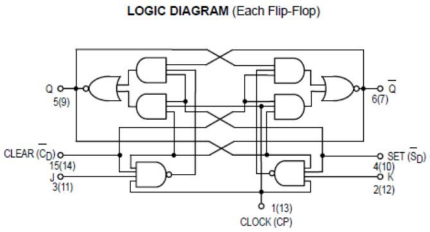
Εικόνα 39.



Εικόνα 40.

8. **SN74112 - DUAL JK NEGATIVE EDGE-TRIGGERED FLIP-FLOP WITH PRESET AND CLEAR.** Κάντε τις κατάλληλες συνδεσμολογίες στο Pencil Box για την αναπαραγωγή του πίνακας αληθείας (MODE SELECT – TRUTH TABLE) του SN74112 (**Εικόνα 41**). Κάντε στη συνέχεια προσομοίωση επαλήθευσης στο TINA (**Εικόνα 42**).





MODE SELECT — TRUTH TABLE

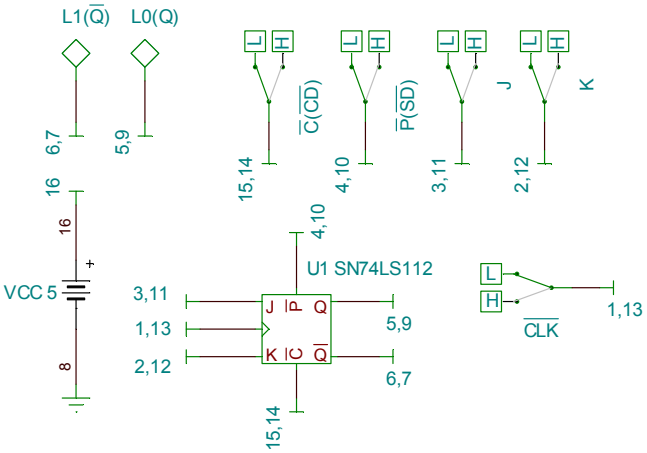
OPERATING MODE	INPUTS				OUTPUTS	
	$\overline{S}_D$	$\overline{C}_D$	J	K	Q	$\overline{Q}$
Set	L	H	X	X	H	L
Reset (Clear)	H	L	X	X	L	H
*Undetermined	L	L	X	X	H	H
Toggle	H	H	h	h	q	$\overline{q}$
Load "0" (Reset)	H	H	l	h	L	H
Load "1" (Set)	H	H	h	l	H	L
Hold	H	H	l	l	q	$\overline{q}$

* Both outputs will be HIGH while both $\overline{S}_D$ and $\overline{C}_D$ are LOW, but the output states are unpredictable if $\overline{S}_D$ and $\overline{C}_D$ go HIGH simultaneously.

H, h = HIGH Voltage Level
L, l = LOW Voltage Level
X = Don't Care
l, h (q) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the HIGH to LOW clock transition.

Εικόνα 41. SN74112 - DUAL JK NEGATIVE EDGE-TRIGGERED FLIP-FLOP WITH PRESET AND CLEAR.

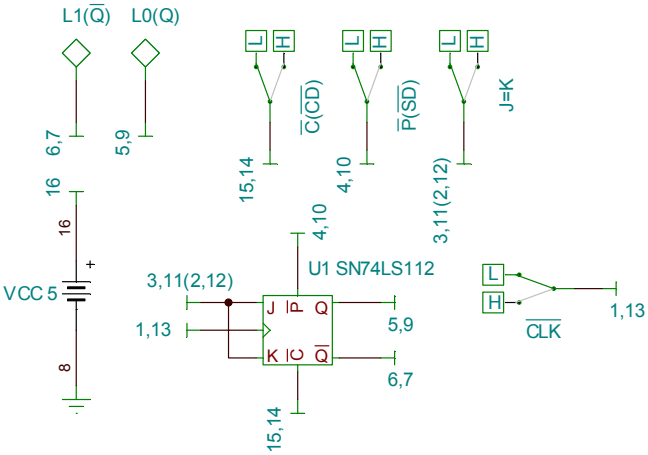
Λύση.



Εικόνα 42. Τρόπος συνδεσμολογίας με σύμβολα του TINA.

9. **SN74112 – Συνδεσμολογία ΩΣ T-FF.** Κάντε τις κατάλληλες συνδεσμολογίες στο Pencil Box ώστε το ένα από τα δύο JK του SN74112 να λειτουργεί ως T-FF. Στη συνέχεια επαληθεύστε με προσομοίωση στο TINA **(Εικόνα 43).**

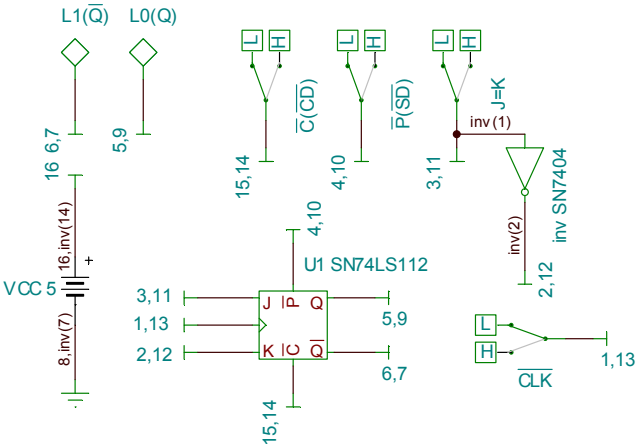
Λύση.



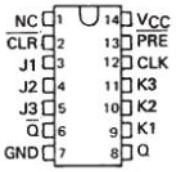
Εικόνα 43. Τρόπος συνδεσμολογίας με σύμβολα του TINA.

10. **SN74112 – Συνδεσμολογία ΩΣ D-FF.** Κάντε τις κατάλληλες συνδεσμολογίες στο Pencil Box ώστε το ένα από τα δύο JK του SN74112 να λειτουργεί ως D-FF. Θα χρειαστείτε και έναν αντιστροφέα (SN7404). Στη συνέχεια επαληθεύστε με προσομοίωση στο TINA **(Εικόνα 44)**.

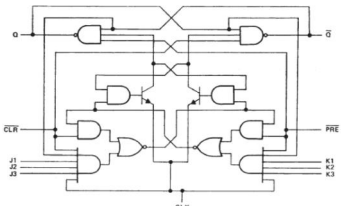
Λύση.



Εικόνα 44. Τρόπος συνδεσμολογίας με σύμβολα του TINA.



NC(1) = No Internal Connection



J = J1 J 2 J3 - K = K1 K2 K3

FUNCTION TABLE

INPUTS					OUTPUTS	
$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H†	H†
H	H	⎍	L	L	Q_0	$\overline{Q}_0$
H	H	⎍	H	L	H	L
H	H	⎍	L	H	L	H
H	H	⎍	H	H	TOGGLE	

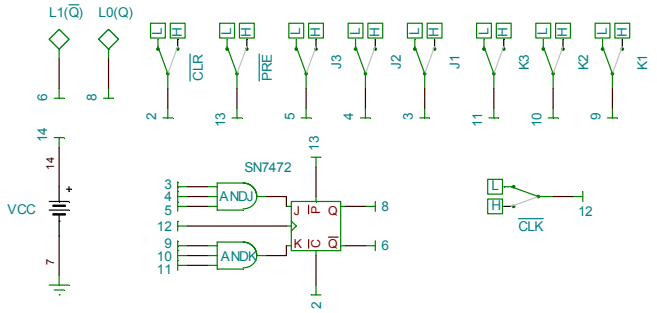
† This configuration is nonstable; that is, it will not persist when either preset or clear returns to its inactive (high) level.

Εικόνα 45. Χαρακτηριστικά SN7472 – AND-GATED JK MASTER-SLAVE NEGATIVE EDGE-TRIGGERED FLIP-FLOP

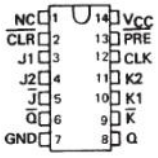
WITH PRESET AND CLEAR από το φυλλάδιο του κατασκευαστή.

11. **SN7472 – AND-GATED JK MASTER-SLAVE NEGATIVE EDGE-TRIGGERED FLIP-FLOP WITH PRESET AND CLEAR.** Κάντε τις κατάλληλες συνδεσμολογίες στο Pencil Box για την αναπαραγωγή του πίνακας αληθείας του SN7472. (**Εικόνα 45**). (Σημείωση: το 7472 που διατίθεται στο εργαστήριο Ψηφιακών Ηλεκτρονικών του τμήματος Ηλεκτρονικών Μηχανικών του ΤΕΙ Αθήνας έχει το πλεονέκτημα ότι οι πύλες and έχουν τις εισόδους τους σε λογικό 1. Οπότε δε χρειάζεται να τις συνδέσουμε στη VCC. Το ίδιο ισχύει και για τα preset και clear. Είναι δηλαδή συνδεδεμένα σε λογικό 1). Στη συνέχεια επαληθεύστε με προσομοίωση στο TINA (**Εικόνα 46**).

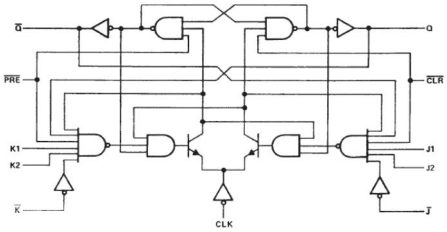
Λύση.



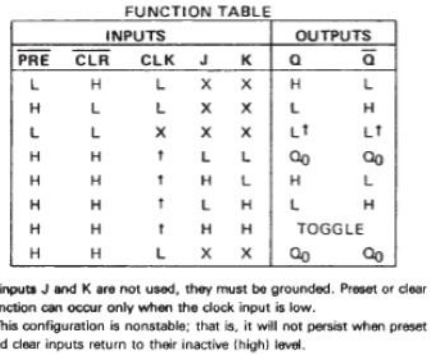
Εικόνα 46. Τρόπος συνδεσμολογίας με σύμβολα του TINA.



NC(1) = No Internal Connection



J = J1 J 2 J' - K = K1 K2 K'



FUNCTION TABLE

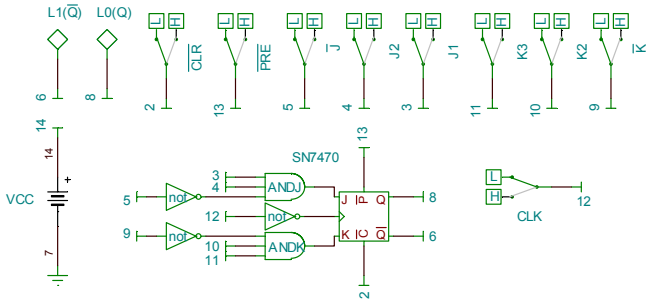
INPUTS					OUTPUTS	
PRE	CLR	CLK	J	K	Q	Q̄
L	H	L	X	X	H	L
H	L	L	X	X	L	H
L	L	X	X	X	L†	L†
H	H	↑	L	L	Q_0	Q_0
H	H	↑	H	L	H	L
H	H	↑	L	H	L	H
H	H	↑	H	H	TOGGLE	
H	H	L	X	X	Q_0	Q_0

If inputs J and K are not used, they must be grounded. Preset or clear function can occur only when the clock input is low.
†This configuration is nonstable; that is, it will not persist when preset and clear inputs return to their inactive (high) level.

Εικόνα 47. Χαρακτηριστικά SN7470 – AND-GATED JK MASTER-SLAVE POSITIVE EDGE-TRIGGERED FLIP-FLOP WITH PRESET AND CLEAR από το φυλλάδιο του κατασκευαστή.

12. **SN7470 – AND-GATED JK MASTER-SLAVE POSITIVE EDGE-TRIGGERED FLIP-FLOP WITH PRESET AND CLEAR.** Κάντε τις κατάλληλες συνδεσμολογίες στο Pencil Box για την αναπαραγωγή του πίνακας αληθείας του SN7470. (**Εικόνα 47**). Στη συνέχεια επαληθεύστε με προσομοίωση στο TINA (**Εικόνα 48**).

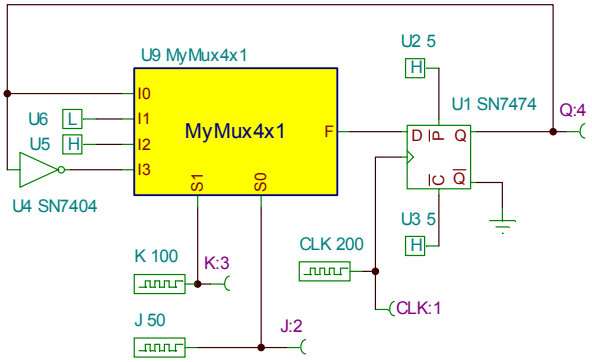
Λύση.



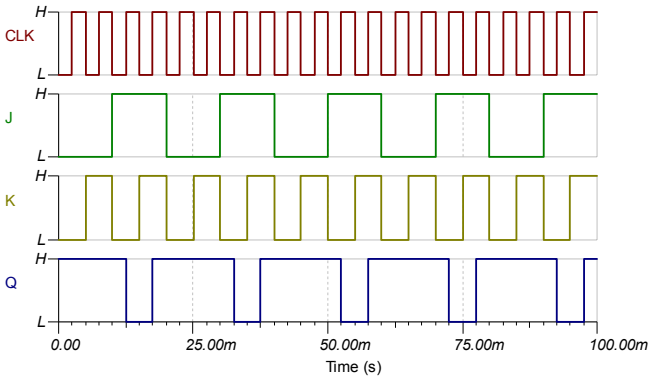
Εικόνα 48. Τρόπος συνδεσμολογίας με σύμβολα του TINA.

13. **Υλοποιήστε ένα JK-FF χρησιμοποιώντας ένα D-FF, ένα MUX 4x1 και μια πύλη NOT.**

Λύση. Βλέπε **Εικόνα 49** και **Εικόνα 50**.



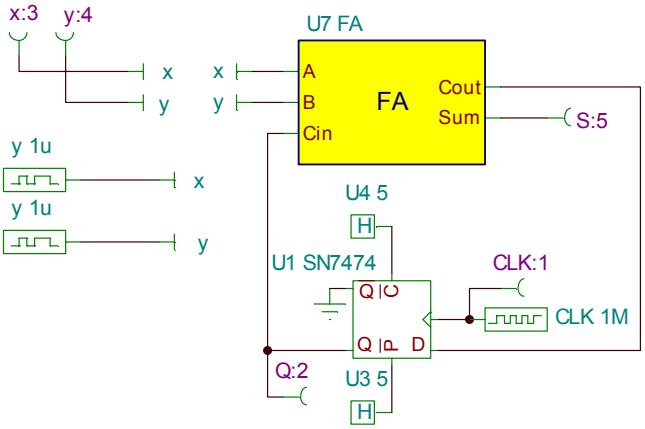
Εικόνα 49. JK-FF χρησιμοποιώντας ένα D-FF, ένα MUX 4x1 και μια πύλη NOT.



Εικόνα 50. Χρονική προσομοίωση προηγούμενου κυκλώματος.

14. Ένα ακολουθιακό κύκλωμα έχει ένα FF Q, δύο εισόδους x, y και μια έξοδο S. Αποτελείται από έναν πλήρη αθροιστή συνδεδεμένο με ένα D-FF (Εικόνα 51). Να εξάγετε τον πίνακα καταστάσεων και το διάγραμμα καταστάσεων.

Λύση.



Εικόνα 51. Ακολουθιακό κύκλωμα της άσκησης.

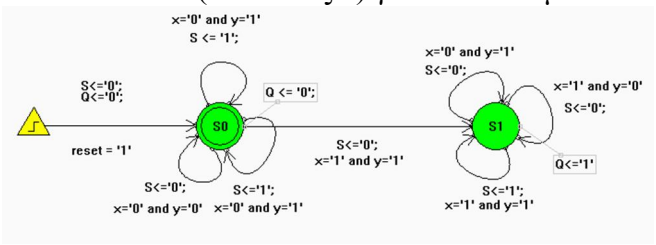
Οι εξισώσεις για το άθροισμα και το κρατούμενο εξόδου του πλήρους αθροιστή είναι: $Sum = x \oplus y \oplus Q$ και $Cout = x\,y + x\,Q + y\,Q$. Η εξίσωση εισόδου του FF είναι: $D_Q = Cout = x\,y + x\,Q + y\,Q$.

Η χαρακτηριστική εξίσωση επόμενης κατάστασης είναι : $Q(t+1) = D = Cout = x\,y + x\,Q + y\,Q$. Ο πίνακας καταστάσεων δίνεται στον **Πίνακα 7.**

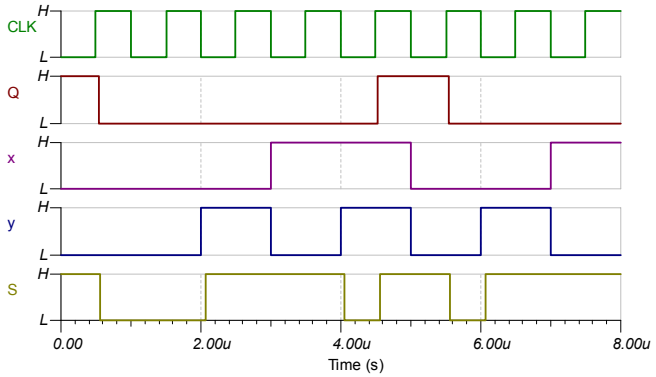
Π.Κ.	**Είσοδοι**		**Ε.Κ.**	**Έξοδοι**
Q	**x**	**y**	**Q(t+1)**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Πίνακας 7.

Το διάγραμμα καταστάσεων είναι συνεπώς αυτό που φαίνεται στην **Εικόνα 52** (Έχει σχεδιαστεί με τον **State Editor** του TINA). Η χρονική ανάλυση του κυκλώματος επαληθεύει τα προηγούμενα και φαίνεται στην **Εικόνα 53**. Ο State Editor του TINA, παράγει αυτόματα τον VHDL κώδικα (**Κώδικας 1**) για το κύκλωμα.



Εικόνα 52. Διάγραμμα καταστάσεων.



Εικόνα 53. Χρονική ανάλυση.

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

entity SAMPLE is
        port (
                reset: in std_logic;
                CLK: in std_logic;
                x: in std_logic;
                y: in std_logic;
                S: out std_logic;
                Q: out std_logic);
end;

architecture SAMPLE_arch of SAMPLE is
type SMachine0_type is (S0,S1);
signal SMachine0: SMachine0_type := S0;
begin
SMachine0_machine: process (CLK)
begin
if reset = '1' then
        S<='0';
        Q<='0';
        SMachine0 <= S0;
elsif CLK'event and CLK = '1' then
        case SMachine0 is
                when S0 =>
                        Q <= '0';
                        if x='1' and y='1' then
                                S<='0';
                                SMachine0 <= S1;
                        elsif x='0' and y='1'then
                                S<='1';
                                SMachine0 <= S0;
                        elsif x='0' and y='0'then
                                S<='0';
                                SMachine0 <= S0;
```

```
                        elsif x='0' and y='1'then
                                S <= '1';
                                SMachine0 <= S0;
                        end if;
                when S1 =>
                        Q<='1'
                        if x='1' and y='1' then
                                S<='1';
                                SMachine0 <= S1;
                        elsif x='1' and y='0'then
                                S<='0';
                                SMachine0 <= S1;
                        elsif x='0' and y='1'then
                                S<='0';
                                SMachine0 <= S1;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;
end SAMPLE_arch;
```

Κώδικας 1. Αυτόματα παραγόμενος VHDL κώδικας περιγραφής του κυκλώματος.

15. Ένα ακολουθιακό κύκλωμα έχει δύο JK-FF A και B και μια είσοδο x. Το κύκλωμα περιγράφεται με τις επόμενες εξισώσεις για τις εισόδους των FF: $J_A = x$, $K_A = B'$, $J_B = x$, $K_B = A$. Να εξάγετε τις εξισώσεις επόμενης κατάστασης A(t+1) και B(t+1) από τον πίνακα καταστάσεων και να σχεδιάσετε το διάγραμμα καταστάσεων.

Λύση.

Η εξίσωση επόμενη κατάστασης για ένα JK-FF είναι:

Q(t+1) = JQ' + K'Q.

Συνεπώς η χαρακτηριστική εξίσωση επόμενης κατάστασης για τα A και B FF θα είναι αντίστοιχα:

A(t+1) = x A' + B A και

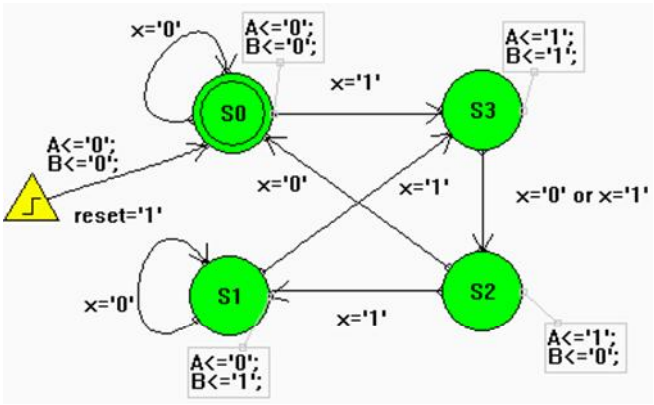
B(t+1) = x B' + A' B.

Ο πίνακας καταστάσεων φαίνεται στη συνέχεια (**Πίνακας 8**).

Το διάγραμμα καταστάσεων φαίνεται στην **Εικόνα 54** (έχει σχεδιαστεί με τον State Editor του TINA). Ο αυτόματα παραγώμενος VHDL κώδικας φαίνεται στον **Κώδικα 2**.

Π.Κ.		**Είσοδος**	**Ε.Κ.**		**Είσοδοι FF**			
A	**B**	**x**	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	1	0	0
0	0	1	1	1	1	1	1	0
0	1	0	0	1	0	0	0	0
0	1	1	1	1	1	0	1	0
1	0	0	0	0	0	1	0	1
1	0	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0	1
1	1	1	1	0	1	0	1	1

Πίνακας 8.



Εικόνα 54. Διάγραμμα καταστάσεων κυκλώματος.

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

entity SAMPLE is
        port (
                reset: in std_logic;
                CLK: in std_logic;
                x: in std_logic;
                A: out std_logic;
                B: out std_logic);
end;

architecture SAMPLE_arch of SAMPLE is
type SMachine0_type is (S0,S3,S2,S1);
signal SMachine0: SMachine0_type := S0;
begin

SMachine0_machine: process (CLK)
begin
if reset='1' then
        A<='0';
        B<='0';
        SMachine0 <= S0;
elsif CLK'event and CLK = '1' then
        case SMachine0 is
                when S0 =>
                        A<='0';
                        B<='0';
                        if x='1' then
                                SMachine0 <= S3;
                        elsif x='0'then
                                SMachine0 <= S0;
                        end if;
                when S3 =>
                        A<='1';
                        B<='1';
                        if x='0' or x='1' then
                                SMachine0 <= S2;
                        end if;
                when S2 =>
                        A<='1';
                        B<='0';
                        if x='0' then
                                SMachine0 <= S0;
                        elsif x='1'then
                                SMachine0 <= S1;
                        end if;
                when S1 =>
                        A<='0';
                        B<='1';
                        if x='0' then
                                SMachine0 <= S1;
                        elsif x='1'then
                                SMachine0 <= S3;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;

end SAMPLE_arch;
```

Κώδικας 2. Αυτόματα παραγόμενος VHDL κώδικας κυκλώματος.

16. Ελαττώστε το πλήθος των καταστάσεων που φαίνονται στον επόμενο πίνακα καταστάσεων (Πίνακας 9) και να παρουσιάσετε το νέο πίνακα καταστάσεων. (Π.Κ. = Παρούσα Κατάσταση, Ε.Κ. = Επόμενη Κατάσταση).

Λύση.

Οι καταστάσεις b, e είναι οι ίδιες γιατί έχουν τις ίδιες επόμενες καταστάσεις και τις ίδιες εξόδους για τις ίδιες τιμές εισόδων.

Οπότε αντικαθιστούμε την κατάσταση e (όπου αυτή εμφανίζεται) με την κατάσταση b. Ομοίως, οι καταστάσεις d και h είναι ισοδύναμες. Οπότε μπορούμε να αντικαταστήσουμε την h (όπου αυτή εμφανίζεται) με τη d (**Πίνακας 10**).

	Ε.Κ.		**Έξοδος**	
Π.Κ.	**X=0**	**X=1**	**X=0**	**X=1**
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

Πίνακας 9.

	Ε.Κ.		**Έξοδος**	
Π.Κ.	**X=0**	**X=1**	**X=0**	**X=1**
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

Πίνακας 10.

Ο πίνακας καταστάσεων μετά τις αλλαγές γίνεται όπως στον **Πίνακα 11**.

Τώρα παρατηρούμε ότι οι καταστάσεις a και c είναι ίδιες, οπότε αντικαθιστούμε την c (όπου αυτή εμφανίζεται) με την a.

Τελικά ο πίνακας καταστάσεων γίνεται όπως φαίνεται στον **Πίνακα 12**.

Παρατηρούμε ότι όλες οι καταστάσεις που περιέχονται σε αυτόν είναι πλέον ανεξάρτητες μεταξύ τους.

	Ε.Κ.		**Έξοδος**	
Π.Κ.	**X=0**	**X=1**	**X=0**	**X=1**
a	f	b	0	0
b	d	c	0	0
c	f	b	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

Πίνακας 11.

	Ε.Κ.		**Έξοδος**	
Π.Κ.	**X=0**	**X=1**	**X=0**	**X=1**
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

Πίνακας 12.

17. Ξεκινώντας από την κατάσταση a και την ακολουθία εισόδου 01110010011, να προσδιορίσετε την ακολουθία εξόδου: για τον αρχικό πίνακα καταστάσεων της προηγούμενης άσκησης καθώς και για τον απλοποιημένο πίνακα καταστάσεων την προγηούμενης άσκησης. Να δείξετε ότι η ακολουθία εξόδου είναι και στις δύο περιπτώσεις η ίδια.

Λύση.

Αν χρησιμοποιήσουμε τον αρχικό πίνακα καστάσεων έχουμε την επόμενη ακολουθία καταστάσεων και εξόδου:

state	a	f	b	c	e	d	g	h	g	g	h	a
input	0	1	1	1	0	0	1	0	0	1	1	
output	0	1	0	0	0	1	1	1	0	1	0	

Πίνακας 13.

Αν χρησιμοποιήσουμε τον απλοποιημένο πίνακα καστάσεων έχουμε την επόμενη ακολουθία καταστάσεων και εξόδου:

state	a	f	b	a	b	d	g	d	g	g	d	a
input	0	1	1	1	0	0	1	0	0	1	1	
output	0	1	0	0	0	1	1	1	0	1	0	

Πίνακας 14.

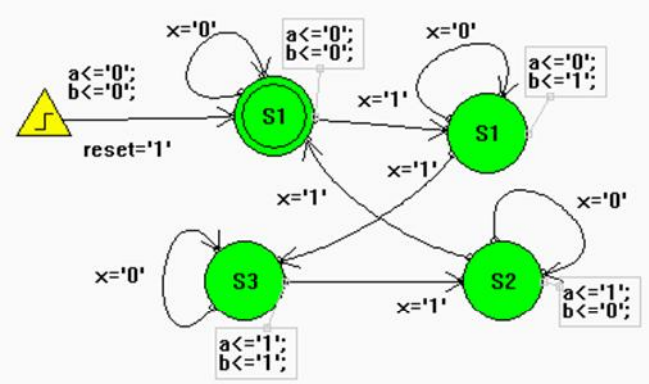
Παρατηρούμε ότι η ακολουθία εξόδου είναι η ίδια και στις δύο περιπτώσεις.

18. Σχεδιάστε ένα ακολουθιακό κύκλωμα με δύο D-FF A και B και μια είσοδο x. Όταν το x=0 η κατάσταση του κυκλώματος παραμένει η ίδια. Όταν x=1 το κύκλωμα μεταπηδά από τις καταστάσεις 00 στη 01 στην 11 στην 10 στην 00 και επαναλαμβάνει.

Λύση.

Με βάση την περιγραφή των μεταβάσεων από την εκφώνηση της άσκησης, το διάγραμμα καταστάσεων του κυκλώματος είναι όπως φαίνεται στην **Εικόνα 55**.

Από το διάγραμμα καταστάσεων προκύπτει ο πίνακας καταστάσεων (**Πίνακας 15**).



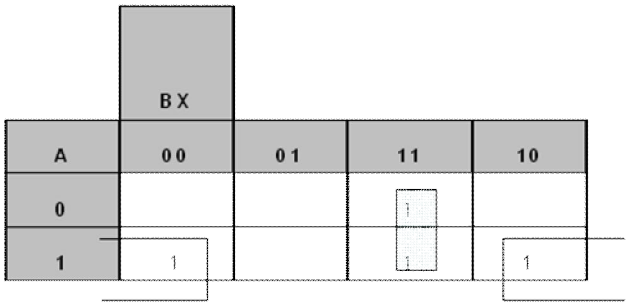
Εικόνα 55.

Π.Κ.		Έξοδος	Ε.Κ.	
A	**B**	**x**	**A**	**B**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

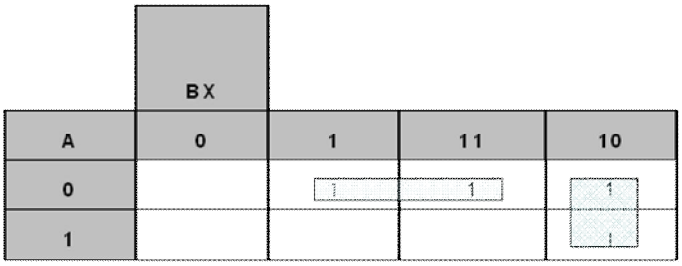
Πίνακας 15.

Η χαρακτηριστική εξίσωση για το D-FF είναι η: Q(t+1) = D.

Από τον πίνακα καταστάσεων έχουμε ότι οι εξισώσεις κατάστασης είναι: A(t+1) = D_A (A,B,X) = ∑ (3,4,6,7) και B(t+1) = D_B (A,B,X) = ∑ (1,2,3,6). Η απεικόνισή τους σε χάρτες Karnaugh και οι απλοποιήσεις φαίνονται στη συνέχεια:

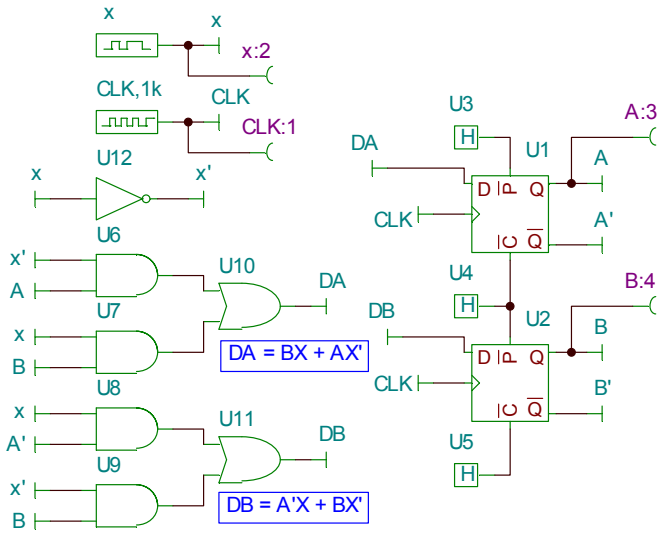


D_A = BX + AX'



D_B = A'X + BX'

Το κύκλωμα φαίνεται στην **Εικόνα 56**. Ο VHDL κώδικας όπως προκύπτει από τον FSM Editor του TINA φαίνεται στη λίστα **Κώδικας 3**.



Εικόνα 56.

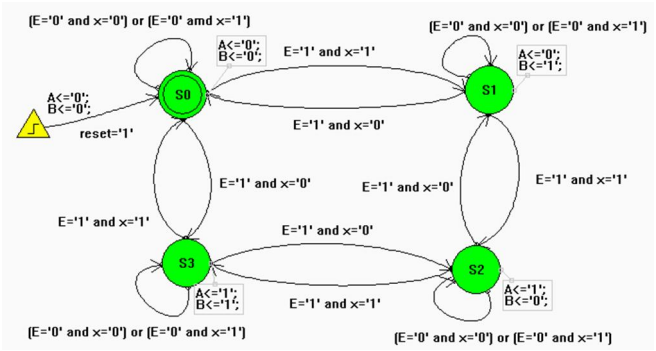
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity SAMPLE is
        port (
                reset: in std_logic;
                CLK: in std_logic;
                x: in std_logic;
                a: out std_logic;
                b: out std_logic);
end;

architecture SAMPLE_arch of SAMPLE is
type SMachine0_type is (S1,S1,S3,S2);
signal SMachine0: SMachine0_type := S1;
begin
SMachine0_machine: process (CLK)
begin

if reset='1' then
        a<='0';
        b<='0';
        SMachine0 <= S1;
elsif CLK'event and CLK = '1' then
        case SMachine0 is
                when S1 =>
                        a<='0';
                        b<='0';
                        if x='0' then
                                SMachine0 <= S1;
                        elsif x='1'then
                                SMachine0 <= S1;
                        end if;
                when S1 =>
                        a<='0';
                        b<='1';
                        if x='1' then
                                SMachine0 <= S3;
                        elsif x='0'then
                                SMachine0 <= S1;
```

```
                        end if;
                when S3 =>
                        a<='1';
                        b<='1';
                        if x='0' then
                                SMachine0 <= S3;
                        elsif x='1'then
                                SMachine0 <= S2;
                        end if;
                when S2 =>
                        a<='1';
                        b<='0';
                        if x='0' then
                                SMachine0 <= S2;
                        elsif x='1'then
                                SMachine0 <= S1;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;
end SAMPLE_arch;
```

Κώδικας 3. VHDL κώδικας παραγόμενος από τον State Editor του TINA.

19. Σχεδιάστε ένα ακολουθιακό κύκλωμα με δύο JK-FFs A και B και δύο εισόδους E και x. Αn E=0 το κύκλωμα θα παραμένει στην ίδια κατάσταση ανεξάρτητα από την τιμή της εισόδου x. Όταν E=1 και x=1 το κύκλωμα θα μεταβαίνει από την κατάσταση 00 στην 01 στην 10 στην 11 και πίσω στην 00 και θα ξεκινά από την αρχή τις μεταβάσεις. Όταν E=1 και x=0 το κύκλωμα θα μεταβαίνει από την 00 στην 11 στην 10 στην 01 ξανά στην 00 και θα επαναλαμβάνει τις μεταβάσεις.

Λύση. Σχεδιάζουμε τη διαδικασία που περιγράφεται στην εκφώνηση στον FSM Editor του TINA όπως φαίνεται στην **Εικόνα 57**.



Εικόνα 57.

Ο **Πίνακας 16** είναι ο πίνακας καταστάσεων του κυκλώματος.

Π.Κ.		**ΕΙΣΟΔΟΣ**		**Ε.Κ**		**ΕΙΣΟΔΟΙ FLIP-FLOPS**			
A	**B**	**E**	**X**	**A**	**B**	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	0	0	X	0	X
0	0	1	0	1	1	1	X	1	X
0	0	1	1	0	1	0	X	1	X
0	1	0	0	0	1	0	X	X	0
0	1	0	1	0	1	0	X	X	0
0	1	1	0	0	0	0	X	X	1
0	1	1	1	1	0	1	X	X	1
1	0	0	0	1	0	X	0	0	X
1	0	0	1	1	0	X	0	0	X
1	0	1	0	0	1	X	1	1	X
1	0	1	1	1	1	X	0	1	X
1	1	0	0	1	1	X	0	X	0
1	1	0	1	1	1	X	0	X	0
1	1	1	0	1	0	X	0	X	1
1	1	1	1	0	0	X	1	X	1

Πίνακας 16.

Οι επόμενοι ΧΚ και οι απλοποιήσεις δίνουν τις εισόδους των FFs.

J_A

AB \ EX	00	01	11	10
0	0	0	0	1
1	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$J_A = BEX + B'EX' = E\,(B \oplus X)'$

K_A

AB \ EX	0	1	11	10
0	X	X	X	X
1	X	X	X	X
11	0	0	1	0
10	0	0	0	1

$K_A = BEX + B'EX' = E\,(B \oplus X)'$

J_B

AB \ EX	0	1	11	10
0	0	0	1	1
1	X	X	X	X
11	X	X	X	X
10	0	0	1	1

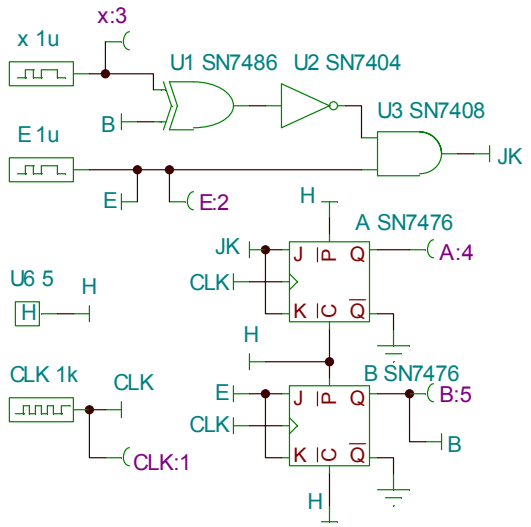
$J_B = E$

K_B

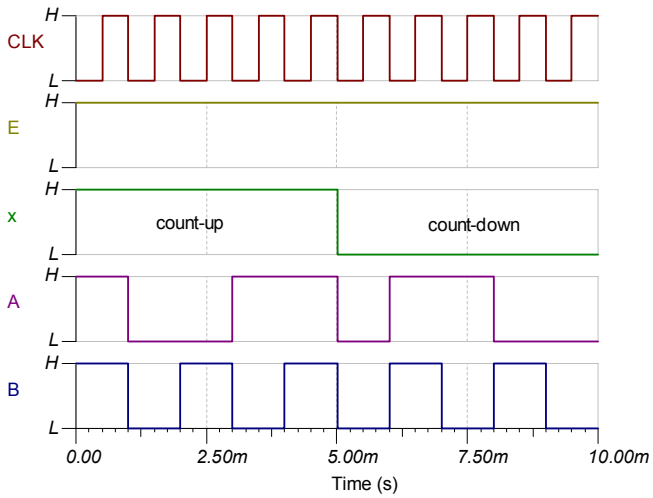
AB \ EX	0	1	11	10
0	X	X	X	X
1	0	0	1	1
11	0	0	1	1
10	X	X	X	X

$K_B = E$

Το τελικό κύκλωμα φαίνεται στην **Εικόνα 58**. Το κύκλωμα συμπεριφέραι ως 2bit μετρητής προς τα πάνω με X=1, και προς τα κάτω με X=0, με το E σε ρόλο επίτρεψης. Στην **Εικόνα 59** φαίνεται το διάγραμμα χρονισμού του. Στον **Κώδικα 4** φαίνεται ο αυτόματα παραγόμενος VHDLκώδικας που υλοποιεί το κύκλωμα.



Εικόνα 58.



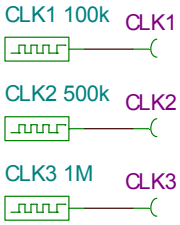
Εικόνα 59. Διάγραμμα χρονισμού.

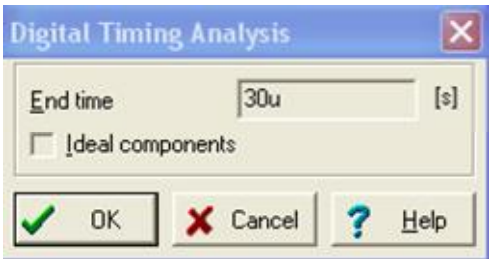
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity SAMPLE is
        port (
                reset: in std_logic;
                CLK: in std_logic;
                x: in std_logic;
                E: in std_logic;
                A: out std_logic;
                B: out std_logic);
end;

architecture SAMPLE_arch of SAMPLE is
type SMachine0_type is (S0,S1,S2,S3);
signal SMachine0: SMachine0_type := S0;
begin

SMachine0_machine: process (CLK)
begin
if reset='1' then
        A<='0';
        B<='0';
        SMachine0 <= S0;
elsif CLK'event and CLK = '1' then
        case SMachine0 is
                when S0 =>
                        A<='0';
                        B<='0';
                        if E='1' and x='0' then
                                SMachine0 <= S3;
                        elsif E='1' and x='1'then
                                SMachine0 <= S1;
                        elsif  (E='0'  and  x='0')  or
(E='0' amd x='1')then
                                SMachine0 <= S0;
                        end if;
                when S1 =>
                        A<='0';
                        B<='1';
                        if E='1' and x='0' then
                                SMachine0 <= S0;
                        elsif E='1' and x='1'then
                                SMachine0 <= S2;
                        elsif  (E='0'  and  x='0')  or
(E='0' and x='1')then
                                SMachine0 <= S1;
                        end if;
                when S2 =>
                        A<='1';
                        B<='0';
                        if E='1' and x='0' then
                                SMachine0 <= S1;
                        elsif E='1' and x='1'then
                                SMachine0 <= S3;
                        elsif  (E='0'  and  x='0')  or
(E='0' and x='1')then
                                SMachine0 <= S2;
                        end if;
                when S3 =>
                        A<='1';
                        B<='1';
                        if E='1' and x='0' then
                                SMachine0 <= S2;
                        elsif E='1' and x='1'then
                                SMachine0 <= S0;
                        elsif  (E='0'  and  x='0')  or
(E='0' and x='1')then
                                SMachine0 <= S3;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;
end SAMPLE_arch;
```

Κώδικας 4. VHDL κώδικας παραγόμενος από τον State Editor του TINA.

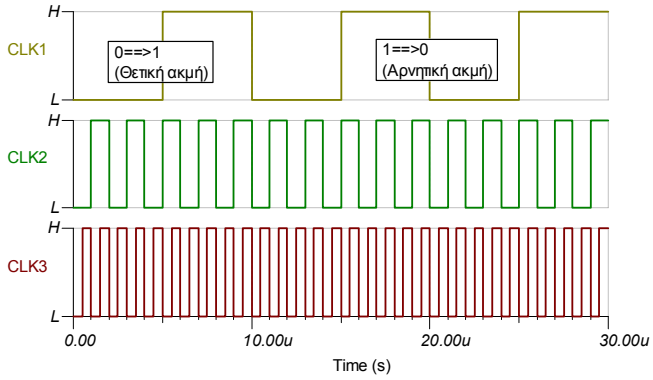
20. a) Σχεδιάστε στο TINA τρία ρολόγια ψηφιακών παλμών με συχνότητες 0.1, 0.5 και 1 MHz. b) Απεικονίστε σε κοινό διάγραμμα τους παλμούς τους για χρόνο από 0 μέχρι 30μs. c) Στην κυματομορφή του ρολογιού συχνότητας 0.1MHz να σημειώσετε τις θετικές και τις αρνητικές ακμές των παλμών.

Λύση. Επιλέγουμε το Clock2 από τα Sources και Voltage Pin από τα Meters. Σχεδιάζουμε το κύκλωμα της **Εικόνας 60**. Πραγματοποιούμε Analysis ➔Digital Timing Analysis για 30μs (30u στο παράθυρο διαλόγου), όπως φαίνεται στην **Εικόνα 61**.



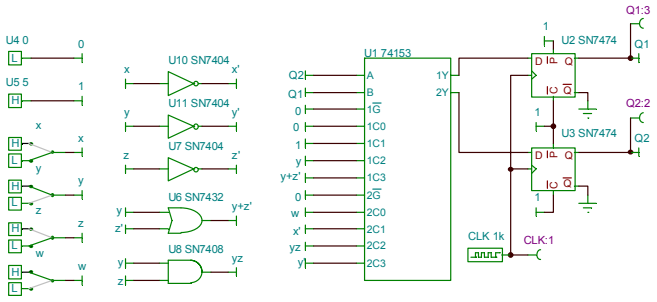


Εικόνα 60.



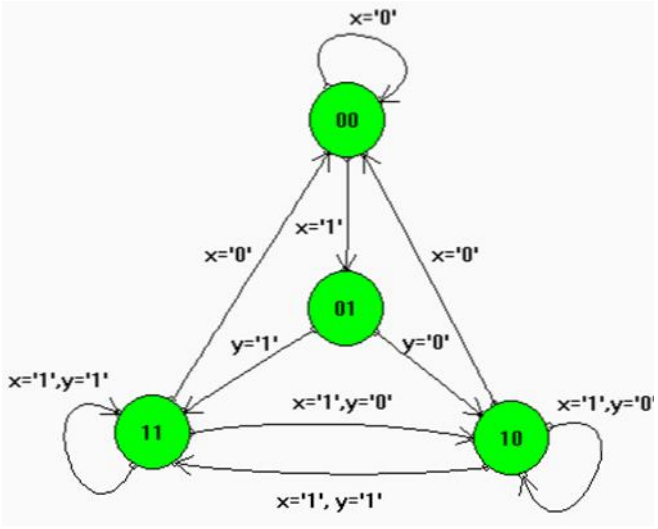
Εικόνα 61.

21. Σχεδιάστε το διάγραμμα καταστάσεων του επόμενου ακολουθιακού κυκλώματος. Σε ποια κατάσταση βρίσκεται για τις δεδομένες τιμές των εισόδων που φαίνονται στην Εικόνα 62.



Εικόνα 62.

22. Σχεδιάστε με πολυπλέκτες και D-FFs το κύκλωμα που υλοποιεί το επόμενο διάγραμμα κατάστασης (Εικόνα 63).



Εικόνα 63.

Υλοποίηση μανδαλωτή με GUARDED BLOCK

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

ENTITY latch IS
PORT (d, clk: IN STD_LOGIC;
q: OUT STD_LOGIC);
END latch;

ARCHITECTURE latch OF latch IS
BEGIN
b1: BLOCK (clk='1')
- guard expression
BEGIN
q <= GUARDED d;
-- guarded statement
END BLOCK b1;
END latch;
```

Υλοποίηση DFF με GUARDED BLOCK

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

ENTITY dff IS
PORT ( d, clk, rst: IN STD_LOGIC;
q: OUT STD_LOGIC);
END dff;

ARCHITECTURE dff OF dff IS
BEGIN
b1: BLOCK (clk'EVENT AND clk='1')
BEGIN
q <= GUARDED '0' WHEN rst='1' ELSE
d;
END BLOCK b1;
END dff;
```

Ακολουθιακός κώδικας

Ο κώδικας VHDL είναι από τη φύση του παράλληλος. Οι δομές PROCESS, FUNCTION και PROCEDURE είναι τα μόνα τμήματα κώδικα που μπορούν να εκτελεστούν σειριακά (ακολουθιακά). Ωστόσο, στο σύνολό τους καθένα από αυτά τα τμήματα κώδικα εξακολουθεί να τρέχει παράλληλα με τις υπόλοιπες εκφράσεις που είναι τοποθετημένες έξω από αυτά.

Μια σημαντική πτυχή του ακολουθιακού κώδικα είναι ότι δεν περιορίζεται μόνο στην ακολουθιακή λογική. Πράγματι, μπορούμε με αυτόν να υλοποιήσουμε ακολουθιακά κυκλώματα, αλλά μπορούμε να υλοποιήσουμε και συνδυαστικά. Ο ακολουθιακός κώδικας είναι γνωστός και ως κώδικας συμπεριφοράς (behavioral code).

Στη συνέχεια θα συζητήσουμε για δηλώσεις που είναι ακολουθιακές, δηλαδή επιτρέπονται μόνον μέσα στα σώματα PROCESS, FUNCTION και PROCEDURE). Πρόκειται για τις IF, WAIT, CASE και LOOP.

Οι VARIABLES χρησιμοποιούνται επίσης με τον περιορισμό να βρίσκονται μέσα σε ακολουθιακό κώδικα (δηλαδή μέσα σε PROCESS, FUNCTION και PROCEDURE). Έτσι, σε αντίθεση με ένα SIGNAL, μια VARIABLE δε μπορεί ποτέ να είναι καθολική, οπότε η τιμή της δε μπορεί να περάσει άμεσα.

Προς το παρόν θα ασχοληθούμε με τις PROCESSES. Οι FUNCTIONS και οι PROCEDURES είναι παρόμοιες αλλά προορίζονται για σχεδίαση σε επίπεδο συστήματος.

PROCESS

Η PROCESS είναι ένα ακολουθιακό τμήμα κώδικα της VHDL. Χαρακτηρίζεται από την παρουσία των IF, WAIT, CASE, ή LOOP και από μια λίστα ευαισθησίας (εκτός και αν χρησιμοποιείται η WAIT). Μια PROCESS πρέπει να εγκαθίσταται στον κύριο κώδικα, και εκτελείται κάθε φορά που ένα σήμα από τη λίστα ευαισθησίας της αλλάζει (ή όταν η συνθήκη που σχετίζεται με το WAIT ικανοποιείται). Η σύνταξή της έχει ως εξής:

```
[label:] PROCESS (sensitivity list)
[VARIABLE name type [range]
[:= initial_value;]]
BEGIN
(sequential code)
END PROCESS [label];
```

Οι VARIABLES είναι κατ' επιλογή. Αν χρησιμοποιηθούν όμως, θα πρέπει να δηλωθούν στο χώρο δηλώσεων πριν το BEGIN. Η αρχική τιμή τους δεν είναι συνθέσιμη και χρησιμοποιείται μόνο για τις προσομοιώσεις.

Η χρήση της label είναι επίσης κατ' επιλογή. Ο σκοπός της είναι να κάνει τον κώδικα ευανάγνωστο. Μπορεί να είναι οποιαδήποτε λέξη εκτός των δεσμευμένων λέξεων της VHDL.

Για να υλοποιήσουμε ένα σύγχρονο κύκλωμα, χρειάζεται να παρατηρούμε ένα σήμα (π.χ. το σήμα ενός ρολογιού). Ένας συνηθισμένος τρόπος ανίχνευσης αλλαγής ενός σήματος είναι με την ιδιότητα EVENT. Π.χ. αν το clk είναι το σήμα που παρακολουθούμε, τότε το clk'EVENT επιστρέφει TRUE όταν συμβεί αλλαγή στο clk

(είναι στην ακμή ανόδου είτε στην ακμή καθόδου).

Παράδειγμα ακολουθιακού κώδικα. DFF με ασύγχρονο Reset

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

ENTITY dff IS
PORT (d, clk, rst: IN STD_LOGIC;
q: OUT STD_LOGIC);
END dff;

ARCHITECTURE behavior OF dff IS
BEGIN
PROCESS (clk, rst)
BEGIN
IF (rst='1') THEN
q <= '0';
ELSIF (clk'EVENT AND clk='1') THEN
q <= d;
END IF;
END PROCESS;
END behavior;
```

Σήματα και μεταβλητές

Η VHDL έχει δύο τρόπου για να περάσει μη-στατικές τιμές: μέσω της SIGNAL ή μέσω της VARIABLE. Η SIGNAL ορίζεται σε ένα PACKAGE, ENTITY ή ARCHITECTURE, στο τμήμα δηλώσεών τους, ενώ η VARIABLE μπορεί να δηλωθεί μόνο στο τμήμα δηλώσεων ενός ακολουθιακού κώδικα (π.χ. μιας PROCESS). Έτσι, ενώ η τιμή της SIGNAL μπορεί να είναι καθολική, η τιμή της VARIABLE είναι πάντα τοπική.

Η τιμή της VARIABLE, δε μπορεί ποτέ να περάσει έξω από την PROCESS άμεσα. Αν κάτι τέτοιο είναι απαραίτητο, θα πρέπει να ανατεθεί σε μια SIGNAL. Από την άλλη η ανανέωση της VARIABLE είναι άμεσα, δηλαδή στην επόμενη γραμμή κώδικα. Αυτό δε συμβαίνει με τη SIGNAL (όταν χρησιμοποιείται σε μια PROCESS), γιατί η νέα τιμή της είναι γενικά βέβαιη μετά την ολοκλήρωση της τρέχουσας PROCESS.

Ο τελεστής ανάθεσης τιμής για μια SIGNAL είναι ο «<=» ενώ για μια VARIABLE είναι = «:=».

IF

Η IF, WAIT, CASE και LOOP είναι δηλώσεις ακολουθιακού κώδικα. Άρα μπορούν να βρεθούν μόνο μέσα σε μια PROCESS, FUNCTION ή PROCEDURE.

Οι χρήστες τείνουν να χρησιμοποιούν περισσότερο την IF. Παρότι, αυτό γενικά, μπορεί να έχει αρνητικές επιπτώσεις (γιατί η IF/ELSE μπορεί να υπονοήσει την υλοποίηση ενός μη απαραίτητου αποκωδικοποιητή προτεραιότητα), ο συνθέτης θα βελτιστοποιήσει τη δομή και θα αποφευχθεί η χρήση επιπλέον υλικού. Η δομή της IF έχει ως εξής:

```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

Παράδειγμα:

```
IF (x<y) THEN temp:="11111111";
ELSIF (x=y AND w='0') THEN
temp:="11110000";
ELSE temp:=(OTHERS =>'0');
```

WAIT

Η WAIT μερικές φορές είναι παρόμοια με την IF. Ωστόσο, υπάρχουν περισσότερες από μία εκδόσεις της WAIT. Επιπλέον σε αντίθεση με την περίπτωση που χρησιμοποιούμε IF, CASE ή LOOP, η PROCESS δε μπορεί να έχει λίστα ευαισθησίας όταν κάνει χρήση της WAIT. Οι τρεις δυνατές μορφές σύνταξης της WAIT φαίνονται στη συνέχεια:

```
WAIT UNTIL signal_condition;
WAIT ON signal1 [, signal2, ... ];
WAIT FOR time;
```

Η WAIT UNTIL δήλωση δέχεται μόνον ένα σήμα, δηλαδή είναι καταλληλότερη για υλοποίηση σύγχρονου κώδικα, παρά ασύγχρονου. Επειδή η PROCESS δεν έχει λίστα ευαισθησίας σε αυτή την περίπτωση, η WAIT UNTIL πρέπει να είναι η πρώτη δήλωση στην PROCESS. Η PROCESS θα πρέπει να εκτελείται κάθε φορά που ικανοποιείται η συνθήκη.

Παράδειγμα: 8bit καταχωρητής με σύγχρονο reset:

```
PROCESS -- no sensitivity list
BEGIN
WAIT UNTIL (clk'EVENT AND clk='1');
IF (rst='1') THEN
output <= "00000000";
ELSIF (clk'EVENT AND clk='1') THEN
output <= input;
END IF;
END PROCESS;
```

Η WAIT ON από την άλλη, δέχεται πολλά σήματα. Η PROCESS τίθεται σε αναμονή μέχρι κάποιο από τα σήματα στη λίστα να αλλάξει. Στο επόμενο παράδειγμα, η PROCESS θα συνεχίσει

κάθε φορά που συμβεί μια αλλαγή στο rst ή στο clk.

Παράδειγμα: 8bit καταχωρητής με ασύγχρονο reset:

```
PROCESS
BEGIN
WAIT ON clk, rst;
IF (rst='1') THEN
output <= "00000000";
ELSIF (clk'EVENT AND clk='1') THEN
output <= input;
END IF;
END PROCESS;
```

Τέλος, η WAIT FOR προορίζεται μόνο για προσομοίωση (παραγωγή κυματομορφών για testbenches). Π.χ.:

```
WAIT FOR 5ns;
```

Παράδειγμα. DFF με ασύγχρονο Reset και χρήση WAIT ON αντί για IF

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

ENTITY dff IS
PORT (d, clk, rst: IN STD_LOGIC;
q: OUT STD_LOGIC);
END dff;

ARCHITECTURE dff OF dff IS
BEGIN
PROCESS
BEGIN
WAIT ON rst, clk;
IF (rst='1') THEN
q <= '0';
ELSIF (clk'EVENT AND clk='1') THEN
q <= d;
END IF;
END PROCESS;
END dff;
```

CASE

Η CASE, είναι ακόμα μια δήλωση, που αφορά ακολουθιακό κώδικα (μαζί με τις IF, WAIT και LOOP). Η σύνταξή της φαίνεται στη συνέχεια:

```
CASE identifier IS
WHEN value => assignments;
WHEN value => assignments;
...
END CASE;
```

Παράδειγμα:

```
CASE control IS
WHEN "00" => x<=a; y<=b;
WHEN "01" => x<=b; y<=c;
WHEN OTHERS => x<="0000"; y<="ZZZZ";
END CASE;
```

Η CASE (για τα ακολουθιακά) είναι παρόμοια με την WHEN (για τα συνδυαστικά). Και εδώ, πρέπει να ελεγχθούν όλες οι περιπτώσεις, οπότε η λέξη OTHERS είναι συχνά χρήσιμη.

Μια άλλη χρήσιμη λέξη είναι η NULL (το αντίστοιχο της UNAFFECTED), η οποία θα πρέπει να χρησιμοποιείται όταν δεν πρέπει να γίνει καμία ενέργεια από το κύκλωμα. Π.χ. :

```
WHEN OTHERS => NULL;.
```

Ωστόσο, η CASE επιτρέπει πολλαπλές αναθέσεις για κάθε συνθήκη ελέγχου (όπως φαίνεται στο προηγούμενο παράδειγμα), ενώ η WHEN επιτρέπει μόνο μία.

Όπως στην περίπτωση της WHEN, η «WHEN value» μπορεί να πάρει τρεις μορφές:

```
WHEN value
-- single value
```

```
WHEN value1 to value2
-- range, for enumerated data types
-- only
```

```
WHEN value1 | value2 |...
-- value1 or value2 or ...
```

Παράδειγμα. DFF με ασύγχρονο reset με χρήση της CASE

```
LIBRARY ieee;
-- Unnecessary declaration,  because
-- BIT was used instead of STD_LOGIC

USE ieee.std_logic_1164.all;

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

ARCHITECTURE dff3 OF dff IS
BEGIN
PROCESS (clk, rst)
BEGIN
CASE rst IS
WHEN '1' => q<='0';
WHEN '0' =>
IF (clk'EVENT AND clk='1') THEN
q <= d;
END IF;
WHEN OTHERS => NULL;
-- Unnecessary, rst is of type BIT
END CASE;
END PROCESS;
END dff3;
```

LOOP

Η LOOP είναι χρήσιμη στην περίπτωση που ένα κομμάτι κώδικα πρέπει να επαναληφθεί αρκετές φορές. Όπως οι IF, WAIT, CASE, η LOOP αφορά ακολουθιακό κώδικα και μπορεί να υπάρξει μόνο μέσα σε PROCESS, FUNCTION ή PROCEDURE.

Οι διάφοροι τρόποι με τους οποίους μπορεί να χρησιμοποιηθεί η LOOP φαίνονται στη συνέχεια:

FOR / LOOP: Ο βρόγχος επαναλαμβάνεται συγκριμένο πλήθος επαναλήψεων.

`[label:] FOR identifier IN range LOOP`
`(sequential statements)`
`END LOOP [label];`

WHILE / LOOP: Ο βρόγχος επαναλαμβάνεται μέχρι την ικανοποίηση κάποιας συνθήκης.

`[label:] WHILE condition LOOP`
`(sequential statements)`
`END LOOP [label];`

EXIT: Χρησιμοποιείται για τον τερματισμό του βρόγχου.
`[label:] EXIT [label] [WHEN condition];`
NEXT: Used for skipping loop steps.
`[label:] NEXT [loop_label] [WHEN condition];`

Παράδειγμα. Μέτρηση αρχικών μηδενικών

Ο επόμενος κώδικας μετράει το πλήθος των αρχικών μηδενικών σε ένα διάνυσμα δυαδικών ξεκινώντας από τα αριστερά. Η συγκεκριμένη λύση χρησιμοποιεί τις LOOP/EXIT. Η EXIT υπενθυμίζουμε ότι δεν κάνει escape από την συγκεκριμένη επανάληψη αλλά δηλώνει μια καθορισμένη έξοδο από αυτή (δηλαδή ακόμα και αν η εμβέλεια επαναλήψεων της LOOP δεν έχει ολοκληρωθεί, θα θεωρηθεί ότι ολοκληρώθηκε). Στο συγκεκριμένο παράδειγμα, το LOOP θα ολοκληρωθεί όταν ένα 1 βρεθεί στο διάνυσμα των δεδομένων.

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

ENTITY LeadingZeros IS
PORT ( data: IN
STD_LOGIC_VECTOR (7 DOWNTO 0);
zeros: OUT
INTEGER RANGE 0 TO 8);
END LeadingZeros;

ARCHITECTURE behavior OF
LeadingZeros IS
BEGIN
PROCESS (data)
VARIABLE count:
INTEGER RANGE 0 TO 8;
BEGIN
count := 0;
FOR i IN data'RANGE LOOP
CASE data(i) IS
WHEN '0' => count := count + 1;
WHEN OTHERS => EXIT;
END CASE;
END LOOP;
zeros <= count;
END PROCESS;
END behavior;
```

Σύγκριση CASE με IF

Γενικά, η ύπαρξη του ELSE στην IF/ELSE μπορεί να υπονοήσει την υλοποίηση ενός αποκωδικοποιητή προτεραιότητας (κάτι που δε θα συμβεί ποτέ με την CASE), αλλά αυτό γενικά δε θα συμβεί. Για παράδειγμα, όταν η IF (μια ακολουθιακή δήλωση) χρησιμοποιηθεί για την υλοποίηση ενός πλήρως συνδυαστικού κυκλώματος, μπορεί να υπονοηθεί η υλοποίηση ενός πολυπλέκτη. Συνεπώς, μετά τη βελτιστοποίηση, η τάση είναι ένα κύκλωμα που συντίθεται από VHDL κώδικα που βασίζεται στην IF να μη διαφέρει από αντίστοιχη υλοποίηση που βασίζεται στην CASE.

Για παράδειγμα, οι δύο επόμενοι κώδικες υλοποιούν το ίδιο κύκλωμα πολυπλέκτη:

```
---- With IF: --------------
IF (sel="00") THEN x<=a;
ELSIF (sel="01") THEN x<=b;
ELSIF (sel="10") THEN x<=c;
ELSE x<=d;
```

```
---- With CASE: ------------
CASE sel IS
WHEN "00" => x<=a;
WHEN "01" => x<=b;
WHEN "10" => x<=c;
WHEN OTHERS => x<=d;
END CASE;
```

Σύγκριση CASE με WHEN

Οι CASE με τη WHEN είναι παρόμοιες. Ωστόσο, η WHEN είναι παράλληλη ενώ η CASE ακολουθιακή. Στον επόμενο πίνακα συνοψίζονται οι κύριες ομοιότητες και διαφορές τους:

	WHEN	CASE
Τύπος	Παράλληλη	Ακολουθιακή
Χρήση	Μόνον εκτός PROCESS, FUNCTION ή PROCEDURE	Μόνο εντός PROCESS, FUNCTION ή PROCEDURE
Εξέταση όλων	Ναι, με WITH /	Ναι

των συνδυασμών	SELECT / WHEN	
Μέγιστο πλήθος αναθέσεων ανά τεστ	1	Οποιοσδήποτε
Λέξη για αδράνεια	UNAFFECTED	NULL

Από συναρτησιακή άποψη, οι δύο παρακάτω κώδικες είναι ισοδύναμοι:

```
---- With WHEN: ----------------
WITH sel SELECT
x <= a WHEN "000",
b WHEN "001",
c WHEN "010",
UNAFFECTED WHEN OTHERS;
```

```
---- With CASE: ----------------
CASE sel IS
WHEN "000" => x<=a;
WHEN "001" => x<=b;
WHEN "010" => x<=c;
WHEN OTHERS => NULL;
END CASE;
```

Σήματα και μεταβλητές

Η VHDL παρέχει δύο αντικείμενα για το χειρισμό μη-στατικών τιμών: τα SIGNAL και VARIABLE. Μας επιτρέπει επίσης να εισάγουμε στατικές τιμές με τις CONSTANT και GENERIC.

Η CONSTANT και η SIGNAL είναι καθολικές (δηλαδή φαίνονται σε όλο τον κώδικα). Μπορούν να χρησιμοποιηθούν και από την παράλληλο και από τον ακολουθιακό κώδικα. Η VARIABLE είναι τοπικής χρήσης γιατί μπορεί να χρησιμοποιηθεί μέσα σε ακολουθιακό κώδικα (δηλαδή μέσα σε PROCESS, FUNCTION ή PROCEDURE) και η τιμής της δε μπορεί να περαστεί άμεσα.

Η επιλογή μεταξύ SIGNAL και VARIABLE δεν είναι πάντα εύκολη υπόθεση.

CONSTANT

Η CONSTANT χρησιμοποιείται για να αποδώσουμε σταθερή τιμή σε σήμα ή μεταβλητή:

```
CONSTANT name : type := value;
```

Παραδείγματα:

```
CONSTANT set_bit : BIT := '1';
CONSTANT datamemory : memory :=
(('0','0','0','0'),
('0','0','0','1'),
('0','0','1','1'));
```

Η CONSTANT μπορεί να δηλωθεί σε ένα PACKAGE, ENTITY ή ARCHITECTURE. Όταν η δήλωσή της γίνεται μέσα σε ένα πακέτο, είναι καθολική, γιατί το πακέτο μπορεί να χρησιμοποιηθεί από διάφορες οντότητες. Όταν δηλωθεί σε μια ENTITY (μετά την PORT) είναι καθολική σε όλες τις αρχιτεκτονικές που ακολουθούν αυτήν την ENTITY. Τέλος, όταν δηλωθεί μέσα σε ARCHITECTURE (στο τμήμα των ορισμών) είναι καθολική μόνο στον κώδικα της συγκεκριμένης ARCHITECTURE. Συνήθως, την CONSTANT τη συναντάμε μέσα σε ARCHITECTURE ή PACKAGE.

SIGNAL

Η SIGNAL χρησιμοποιείται για να περνά τιμές εντός και εκτός κυκλώματος, καθώς και μεταξύ εσωτερικών μονάδων. Δηλαδή παριστάνει τα καλώδια που συνδέουν τα διάφορα μέρη του κυκλώματος. Π.χ. όλα τα PORTS σε μια ENTITY είναι εξορισμού SINGAL. Η σύνταξή της έχει ως εξής:

```
SIGNAL name : type [range] [:=
initial_value];
```

Παραδείγματα:

```
SIGNAL control: BIT := '0';
SIGNAL count:
INTEGER RANGE 0 TO 100;
SIGNAL y:
STD_LOGIC_VECTOR (7 DOWNTO 0);
```

Η δήλωση ενός SIGNAL μπορεί να γίνει στα ίδια μέρη που μπορεί και η δήλωση της CONSTANT.

Ένα ιδιαιτέρως σημαντικό χαρακτηριστικό της SIGNAL όταν χρησιμοποιείται μέσα σε ακολουθιακό κώδικας (π.χ. σε μια PROCESS) είναι ότι η ανανέωσή της δεν είναι άμεση. Δηλαδή, η νέα της τιμή δε θα πρέπει να περιμένουμε ότι θα είναι έτοιμη πριν την ολοκλήρωση της αντίστοιχης PROCESS, FUNCTION ή PROCEDURE.

Ο τελεστής απόδοσης τιμής για μια SIGNAL είναι ο «<=». Η αρχική τιμή στην σύνταξη, δεν είναι συνθέσιμη και χρησιμοποιείται μόνο για τις προσομοιώσεις.

Προσοχή χρειάζεται στην περίπτωση που στο ίδιο SIGNAL γίνονται πολλαπλές αναφορές. Ο μεταγλωττιστής μπορεί να τυπώσει μήνυμα σφάλματος και να σταματήσει τη μεταγλώττιση, ή μπορεί να θεωρήσει λανθασμένο κύκλωμα (π.χ. θεωρώντας μόνο την τελική ανάθεση σήματος). Συνεπώς, η απόδοση αρχικών τιμών σε σήματα, καλύτερα να γίνεται με χρήση VARIABLE.

Παράδειγμα. Προβληματική υλοποίηση κυκλώματος μέτρησης 1

Ας υποθέσουμε ότι θέλουμε να σχεδιάσουμε ένα κύκλωμα που να μετρά το

πλήθος των 1 σε ένα δυαδικό διάνυσμα. Θα θεωρήσουμε ότι κάνουμε μόνο χρήση σημάτων. Θα δούμε στον επόμενο κώδικα, ότι υπάρχουν πολλαπλές αποδόσεις τιμής στο ίδιο σήμα. Επιπλέον, επειδή η τιμή του σήματος δεν ανανεώνεται αμέσως, παρά μετά το τέλος της PROCEDURE, υπάρχει μεγάλη πιθανότητα σφάλματος στα αποτελέσματα. Σε τέτοιες περιπτώσεις ενδείκνυται η χρήση VARIABLE αντί για SIGNAL.

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

ENTITY count_ones IS
PORT ( din: IN STD_LOGIC_VECTOR (7
DOWNTO 0);
ones: OUT INTEGER RANGE 0 TO 8);
END count_ones;

ARCHITECTURE not_ok OF count_ones IS
SIGNAL temp: INTEGER RANGE 0 TO 8;
BEGIN
PROCESS (din)
BEGIN
temp <= 0; -- not immediate
FOR i IN 0 TO 7 LOOP
IF (din(i)='1') THEN
temp <= temp + 1; -- not immmediate
END IF;
END LOOP;
ones <= temp;
END PROCESS;
END not_ok;
```

Η χρήση του temp φαίνεται υπερβολή, γιατί θα μπορούσαμε να χρησιμοποιήσουμε κατευθείαν το ones. Ωστόσο, για να γινόταν αυτό θα έπρεπε να αλλάξουμε τον τρόπο του ones από OUT σε BUFFER, γιατί το ones δέχεται τιμή και ταυτόχρονα χρησιμοποιείται για ανάγνωση εσωτερικά στο κύκλωμα. Επειδή το ones είναι σήμα δύο κατευθύνσεων (OUT), η χρήση ενός βοηθητικού σήματος (temp) είναι μια επαρκή σχεδιαστική πρακτική.

VARIABLE

Σε αντίθεση με την CONSTANT και SIGNAL, η VARIABLE αναπαριστά μόνον τοπική πληροφορία. Χρησιμοποιείται μόνον μέσα σε PROCESS, FUNCTION, ή PROCEDURE (δηλαδή σε ακολουθιακό κώδικα) και η τιμή της δε μπορεί να περάσει εκτός, άμεσα. Από την άλλη, η ανανέωση τιμών της είναι άμεση, δηλαδή στην επόμενη γραμμή κώδικα.

Για να δηλώσουμε VARIABLE, χρησιμοποιούμε την επόμενη σύνταξη:

```
VARIABLE name : type [range] [:=
init_value];
```

Παραδείγματα:

```
VARIABLE control: BIT := '0';
VARIABLE count:
INTEGER RANGE 0 TO 100;
VARIABLE y:
STD_LOGIC_VECTOR (7 DOWNTO 0) :=
"10001000";
```

Επειδή η VARIABLE μπορεί να χρησιμοποιηθεί μόνο μέσα σε ακολουθιακό κώδικα, η δήλωσή της μπορεί να γίνει στο σώμα δηλώσεων μιας PROCESS, FUNCTION ή PROCEDURE.

Ο τελεστής απόδοσης τιμής για VARIABLE είναι ο «:=». Επίσης, όπως και στην περίπτωση της SIGNAL, η αρχική τιμή στον τρόπο σύνταξης δεν είναι συνθέσιμη, αλλά χρησιμοποιείται μόνο στις προσομοιώσεις.

Παράδειγμα. Ορθή υλοποίηση κυκλώματος μέτρησης 1

Η διαφορά με την παραπάνω υλοποίηση είναι τώρα, η χρήση της εσωτερικής VARIABLE στη θέση της SIGNAL. Η ανανέωση της VARIABLE είναι άμεση, οπότε η αρχική τιμή ανεβαίνει αμέσως και δεν υπάρχει πρόβλημα με τις πολλαπλές αναθέσεις στην ίδια VARIABLE.

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

ENTITY count_ones IS
PORT ( din: IN STD_LOGIC_VECTOR (7
DOWNTO 0);
ones: OUT INTEGER RANGE 0 TO 8);
END count_ones;

ARCHITECTURE ok OF count_ones IS
BEGIN
PROCESS (din)
VARIABLE temp:
INTEGER RANGE 0 TO 8;
BEGIN
temp := 0;
FOR i IN 0 TO 7 LOOP
IF (din(i)='1') THEN
temp := temp + 1;
END IF;
END LOOP;
ones <= temp;
END PROCESS;
END ok;
```

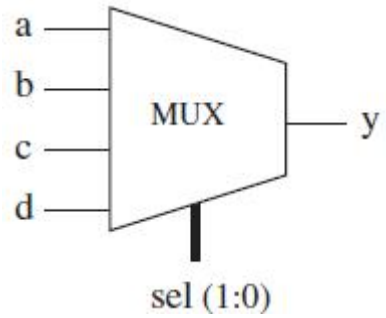
Σχέση SIGNAL και VARIABLE

	SIGNAL	VARIABLE
Ανάθεση	<=	:=
Χρήση	Παριστάνει τις διασυνδέσεις του	Παριστάνει τοπική

	κυκλώματος	πληροφορία
Εμβέλεια	Μπορεί να είναι καθολική.	Τοπική (φανερή μέσα σε PROCESS, FUNCTION, PROCEDURE).
Συμπεριφορά	Η ανανέωση δεν είναι άμεση σε ακολουθιακό κώδικα (πρέπει να ολοκληρωθεί η περιβάλλουσα PROCESS, FUNCTION, ή PROCEDURE).	Η ανανέωση είναι άμεση (η νέα τιμή είναι διαθέσιμη στην επόμενη γραμμή κώδικα).
Χρήση	Μέσα σε PACKAGE, ENTITY, ARCHITECTURE. Σε μια ENTITY όλες οι PORTS είναι SIGNAL εξορισμού.	Μόνο σε ακολουθιακό κώδικα (δηλαδή μέσα σε PROCESS, FUNCTION ή PROCEDURE).

Παράδειγμα. Κακή και καλή προγραμματιστική υλοποίηση πολυπλέκτη.

Θα υλοποιήσουμε το επόμενο πολυπλέκτη:



Υλοποίηση με χρήση της SINGAL (δεν είναι καλή τεχνική):

```
-- Solution 1:
-- using a SIGNAL (not ok) --
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux IS
PORT ( a, b, c, d, s0, s1: IN
STD_LOGIC;
y: OUT STD_LOGIC);
END mux;

ARCHITECTURE not_ok OF mux IS
SIGNAL sel : INTEGER RANGE 0 TO 3;
BEGIN
PROCESS (a, b, c, d, s0, s1)
BEGIN
sel <= 0;
IF (s0='1') THEN sel <= sel + 1;
END IF;
IF (s1='1') THEN sel <= sel + 2;
END IF;
CASE sel IS
WHEN 0 => y<=a;
WHEN 1 => y<=b;
WHEN 2 => y<=c;
WHEN 3 => y<=d;
END CASE;
END PROCESS;
END not_ok;
```

Η SIGNAL χρειάζεται συγκεκριμένο χρόνο για ανανέωση. Συνεπώς, η απόδοση sel<=sel+1, επειδή είναι μετά την sel<=0, δεν είναι βέβαιο ότι θα δώσει τιμή 1 στο sel. Τι ίδιο ισχύει και για την sel<=sel+2. Αυτό δεν αποτελεί πρόβλημα με τη χρήση VARIABLE, γιατί η ανάθεση τιμής γίνεται αμέσως σε αυτή την περίπτωση.

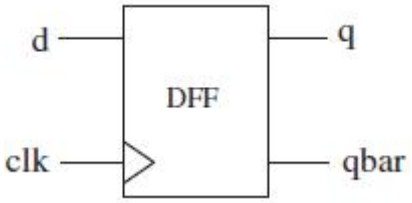
Ένα δεύτερο θέμα με την προηγούμενη υλοποίηση είναι αυτό των πολλαπλών αναθέσεων στο ίδια SIGNAL (στο sel). Γενικά μόνο μία ανάθεση σε ένα σήμα επιτρέπεται μέσα σε μια PROCESS. Δηλαδή το λογισμικό είτε θα θεωρήσει μόνο την sel<=sel+2 ή θα τυπώσει μήνυμα σφάλματος και θα σταματήσει τη μεταγλώττιση. Και πάλι, αυτό δεν αποτελεί πρόβλημα με την VARIABLE.

Υλοποίηση με χρήση της VARIABLE:

```
-- Solution 2: using a VARIABLE (ok)
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux IS
PORT ( a, b, c, d, s0, s1: IN
STD_LOGIC;
y: OUT STD_LOGIC);
END mux;

ARCHITECTURE ok OF mux IS
BEGIN
PROCESS (a, b, c, d, s0, s1)
VARIABLE sel :
INTEGER RANGE 0 TO 3;
BEGIN
sel := 0;
IF (s0='1') THEN sel := sel + 1;
 END IF;
IF (s1='1') THEN sel := sel + 2;
END IF;
CASE sel IS
WHEN 0 => y<=a;
WHEN 1 => y<=b;
WHEN 2 => y<=c;
WHEN 3 => y<=d;
END CASE;
END PROCESS;
END ok;
```

Υλοποίηση DFF με χρήση SIGNAL

Θα υλοποιήσουμε το επόμενο DFF:



Υπενθυμίζουμε ότι μια PORT είναι εξορισμού SIGNAL.

Στην επόμενη υλοποίηση οι αναθέσεις q<=d και qbar<=NOT q είναι σύγχρονες, οπότε οι νέες τιμές τους θα είναι διαθέσιμες με την ολοκλήρωση της PROCESS. Αυτό αποτελεί πρόβλημα για το qbar, γιατί η νέα τιμή του q δεν έχει εφαρμοστεί σωστά. Οπότε το qbar θα πάρει ως τιμή το αντίστροφο του παλιού q. Δηλαδή η σωστή τιμή του qbar θα καθυστερεί κατά έναν κύκλο ρολογιού, με συνέπεια το κύκλωμα να λειτουργεί λανθασμένα.

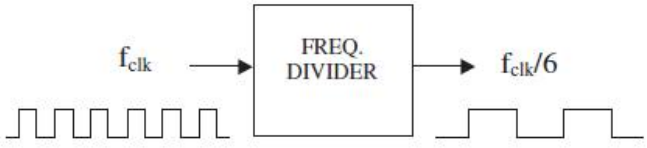
```
---- Solution 1: not OK -----------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
PORT ( d, clk: IN STD_LOGIC;
q: BUFFER STD_LOGIC;
qbar: OUT STD_LOGIC);
END dff;

ARCHITECTURE not_ok OF dff IS
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
q <= d;
qbar <= NOT q;
END IF;
END PROCESS;
END not_ok;
```

Στην επόμενη υλοποίηση έχουμε βάλει το qbar <= NOT q, εκτός της PROCESS, οπότε η λειτουργία του είναι παράλληλη.

```
---- Solution 2: OK --------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
PORT ( d, clk: IN STD_LOGIC;
q: BUFFER STD_LOGIC;
qbar: OUT STD_LOGIC);
END dff;

ARCHITECTURE ok OF dff IS
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
q <= d;
END IF;
END PROCESS;
qbar <= NOT q;
END ok;
```

Υλοποίηση διαιρέτη συχνότητας

Θα υλοποιήσουμε ένα κύκλωμα που διαιρεί τη συχνότητα του ρολογιού με 6:



Θα έχει μια έξοδο count1 ορισμένη ως SIGNAL και μια έξοδο count2 ορισμένη ως VARIABLE. Να συμπληρώσετε τα κενά ώστε να λειτουργεί ορθά ο επόμενος κώδικας:

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

ENTITY freq_divider IS
PORT ( clk : IN STD_LOGIC;
out1, out2 : BUFFER STD_LOGIC);
END freq_divider;

ARCHITECTURE example OF freq_divider
IS
SIGNAL count1 :
INTEGER RANGE 0 TO 7;
BEGIN
PROCESS (clk)
VARIABLE count2 :
INTEGER RANGE 0 TO 7;
BEGIN
IF (clk'EVENT AND clk='1') THEN
count1 <= count1 + 1;
count2 := count2 + 1;
IF (count1 = ? ) THEN
out1 <= NOT out1;
count1 <= 0;
END IF;
IF (count2 = ? ) THEN
out2 <= NOT out2;
count2 := 0;
END IF;
END IF;
END PROCESS;
END example;
```

Πλήθος καταχωρητών που υπολογίζει ο μεταγλωττιστής

Θα δούμε στη συνέχεια πως ο μεταγλωττιστής εκτιμά το πλήθος των flip-flops που χρειάζονται για την υλοποίηση ενός κυκλώματος. Ο στόχος μας είναι κατανοήσουμε

τις διαδικασίες που χρειάζονται για να ελαττώσουμε το πλήθος των καταχωρητών, αλλά και να εξακριβώσουμε ότι πράγματι ο μεταγλωττιστής θα υλοποιήσει το σωστό κύκλωμα.

Ένα SIGNAL παράγει ένα flip-flop κάθε φορά που μια ανάθεση γίνεται στη μετάβαση ενός άλλου σήματος. Δηλαδή κάθε φορά που γίνεται μια σύγχρονη ανάθεση. Επειδή μια τέτοια ανάθεση είναι σύγχρονη, μπορεί να γίνει μόνο μέσα σε μια PROCESS, FUNCTION ή PROCEDURE (συνήθως ακολουθεί τη δήλωση της IF signal'EVENT… ή της WAIT UNTIL…

Μια VARIABLE, δε θα δημιουργήσει απαραίτητα flip-flops αν η τιμή της δε φύγει από την PROCESS (ή FUNCTION ή PROCEDURE). Αν όμως τιμή ανατεθεί σε μια VARIABLE στη μετάβαση ενός άλλου σήματος, και μια τέτοια τιμή τελικά περάσει σε ένα σήμα (το οποίο φεύγει από τη διαδικασία), τότε ο μεταγλωττιστής θα υποθέσει flip-flops. Μια VARIABLE θα δημιουργήσει καταχωρητή και όταν χρησιμοποιείται πριν μια τιμή της ανατεθεί.

Στο επόμενο παράδειγμα, στη διαδικασία, οι output1 και output2 θα αποθηκευτούν και οι δύο (δηλαδή θα υποτεθούν flip-flops), γιατί και οι δύο τίθενται κατά τη μετάβαση σήματος (clk):

```
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
output1 <= temp; -- output1 stored
output2 <= a; -- output2 stored
END IF;
END PROCESS;
```

Στην επόμενη διαδικασία, μόνο η output1 θα αποθηκευτεί (η output2 θα κάνει χρήση πυλών λογικής):

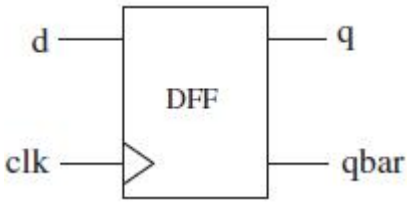
```
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
output1 <= temp; -- output1 stored
END IF;
output2 <= a; -- output2 not stored
END PROCESS;
```

Στην επόμενη διαδικασία, η μεταβλητή temp θα προκαλέσει αποθήκευση του x (ένα σήμα):

```
PROCESS (clk)
VARIABLE temp: BIT;
BEGIN
IF (clk'EVENT AND clk='1') THEN
temp <= a;
END IF;
x <= temp;
-- temp causes x to be stored
END PROCESS;
```

Υλοποίηση DFF και πλήθος υποτιθέμενων FF

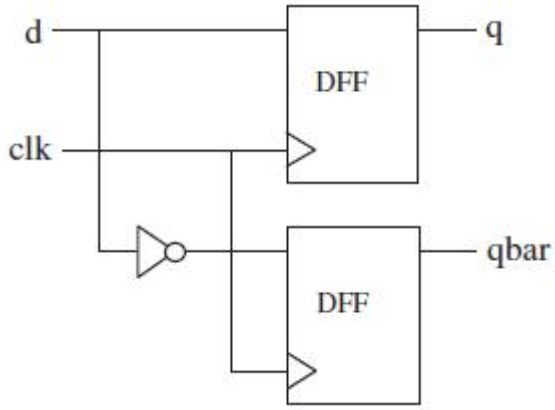
Θα χρησιμοποιήσουμε ως παράδειγμα αναφοράς την υλοποίηση του επόμενου DFF:



Η επόμενη υλοποίηση χρησιμοποιεί δύο αναθέσεις SIGNAL, οπότε ο μεταγλωττιστής θα υποθέσει την ύπαρξη δύο FFs:

```
-- Solution 1: Two DFFs
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
PORT ( d, clk: IN STD_LOGIC;
q: BUFFER STD_LOGIC;
qbar: OUT STD_LOGIC);
END dff;

ARCHITECTURE two_dff OF dff IS
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
q <= d;
-- generates a register
qbar <= NOT d;
-- generates a register
END IF;
END PROCESS;
END two_dff;
```

Δηλαδή το κύκλωμα που ο μεταγλωττιστής υποθέτει είναι το επόμενο:

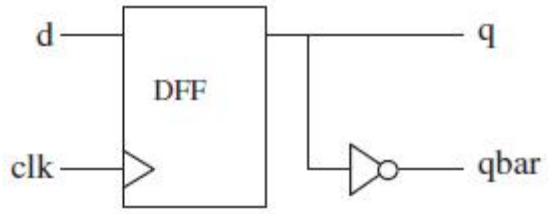


Στην επόμενη υλοποίηση, η μία από τις δύο αναθέσεις τιμών σήματος, δεν είναι πλέον σύγχρονη:

```
-- Solution 2: One DFF
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
PORT ( d, clk: IN STD_LOGIC;
q: BUFFER STD_LOGIC;
```

```
qbar: OUT STD_LOGIC);
END dff;

ARCHITECTURE one_dff OF dff IS
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
q <= d; -- generates a register
END IF;
END PROCESS;
qbar <= NOT q;
-- uses logic gate (no register)
END one_dff;
```

Το κύκλωμα που θα υλοποιήσει ο μεταγλωττιστής είναι στην περίπτωση αυτή, το επόμενο:



Είναι σημαντικό να σημειώσουμε εδώ ότι σε ορισμένους τύπους CPLD/FPGAs, όταν τα σήματα q και qbar συνδέονται άμεσα σε pins του τσιπ, ο fitter (place & route) μπορεί να δει ως βέλτιστη λύση τα δύο FFs ως φυσική υλοποίηση. Αυτό όμως δε σημαίνει πως είναι πραγματικά απαραίτητα. Η αναφορά του fitter (place & route) θα μας δηλώσει δύο καταχωρητές, αλλά η αναφορά της σύνθεσης θα δηλώσει ότι χρειάζεται μόνο ένας.

Παράρτημα. VHDL Κώδικες Κεφαλαίου

Από το βιβλίο του E. O. Hwang, "Digital Logic and Microprocessor Design withVHDL".

Latches and Flip-Flops

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Clockdiv IS PORT (
Clk25Mhz: IN STD_LOGIC;
Clk: OUT STD_LOGIC);
END Clockdiv;
ARCHITECTURE Behavior OF Clockdiv IS
CONSTANT max: INTEGER := 25000000;
CONSTANT half: INTEGER := max/2;
SIGNAL count: INTEGER RANGE 0 TO max;
BEGIN
PROCESS
BEGIN
WAIT UNTIL Clk25Mhz'EVENT and Clk25Mhz =
'1';
IF count < max THEN
count <= count + 1;
ELSE
count <= 0;
END IF;
IF count < half THEN
Clk <= '0';
ELSE
Clk <= '1';
END IF;
END PROCESS;
END Behavior;
```

Code. VHDL behavioral description of a clock divider circuit.