ΤΕΙ ΣΕΡΡΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΟΜΕΑΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΒΙΟΜΗΧΑΝΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΠΡΟΗΓΜΕΝΑ ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΣΗΜΕΙΩΣΕΙΣ ΕΡΓΑΣΤΗΡΙΟΥ

ΣΤ' ΕΞΑΜΗΝΟ

Έκδοση 2η

Ιωάννης Καλόμοιρος

Επίκουρος καθηγητής ΤΕΙ Σερρών

Συνεργασία:

Μαδεμλής Ιωάννης

Εργαστηριακός Συνεργάτης

Σέρρες 2010

ΠΕΡΙΕΧΟΜΕΝΑ

ΜΕΡΟΣ Α

ΣΧΕΔΙΑΣΗ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΣΤΟΝ ΠΡΟΣΟΜΟΙΩΤΗ MULTISIM 7.0

ΕΡΓΑΣΤΗΡΙΟ 1

ΣΧΕΔΙΑΣΜΟΣ ΑΠΛΩΝ ΚΥΚΛΩΜΑΤΩΝ ΣΥΝΔΥΑΣΤΙΚΗΣ ΛΟΓΙΚΗΣ

1.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Τα συνδυαστικά λογικά κυκλώματα αποτελούνται από λογικές πύλες και υλοποιούν μία ή περισσότερες συναρτήσεις της άλγεβρας Boole. Έχουν έναν αριθμό εισόδων και μία ή περισσότερες εξόδους. Οι τιμές των εξόδων στα συνδυαστικά κυκλώματα εξαρτώνται μόνον από τις τρέχουσες τιμές των εισόδων. Άρα, λοιπόν, διαφέρουν από τα ακολουθιακά κυκλώματα, τα οποία περιέχουν και στοιχεία μνήμης, ώστε οι τιμές των εξόδων τους εξαρτώνται και από προηγούμενες τιμές των εισόδων. Η συμπεριφορά των ακολουθιακών κυκλωμάτων καθορίζεται από τη χρονική ακολουθία των καταστάσεων από τις οποίες περνούν διαδοχικά.

Προκειμένου να σχεδιάσουμε ένα συνδυαστικό ψηφιακό σύστημα ξεκινούμε από τις προδιαγραφές της λειτουργίας του. Ως πρώτο βήμα προσδιορίζουμε τις απαραίτητες εισόδους και εξόδους του κυκλώματος. Στη συνέχεια, με βάση τη σχέση εισόδων-εξόδων που περιγράφουν οι προδιαγραφές δημιουργούμε τον πίνακα αληθείας του κυκλώματος.

Από τον πίνακα αληθείας εξάγουμε τις λογικές συναρτήσεις που περιγράφουν το κύκλωμα Για παράδειγμα εξάγουμε τη λογική συνάρτηση ως άθροισμα ελαχίστων όρων (των λογικών γινομένων κάθε δυνατού συνδυασμού όλων των μεταβλητών της συνάρτησης ή των συμπληρωμάτων τους). Ακολουθεί η ελαχιστοποίηση της συνάρτησης με κάποιον από τους γνωστούς τρόπους, για παράδειγμα με τη βοήθεια του πίνακα του Karnaugh. Τέλος, σχεδιάζουμε το κύκλωμα που προκύπτει και το υλοποιούμε με κυκλώματα της τυπικής λογικής ή με τη βοήθεια κάποιας προγραμματιζόμενης διάταξης.

Στο εργαστηριακό μέρος της άσκησης θα εργαστούμε στον προσομοιωτή ηλεκτρονικών κυκλωμάτων Multisim 7.0, και θα σχεδιάσουμε απλά ψηφιακά κυκλώματα προκειμένου να εξοικειωθούμε με τα εργαλεία σχεδίασης του λογισμικού.

1.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

1.2.1 Γνωριμία με τον προσομοιωτή ηλεκτρονικών κυκλωμάτων Multisim 7

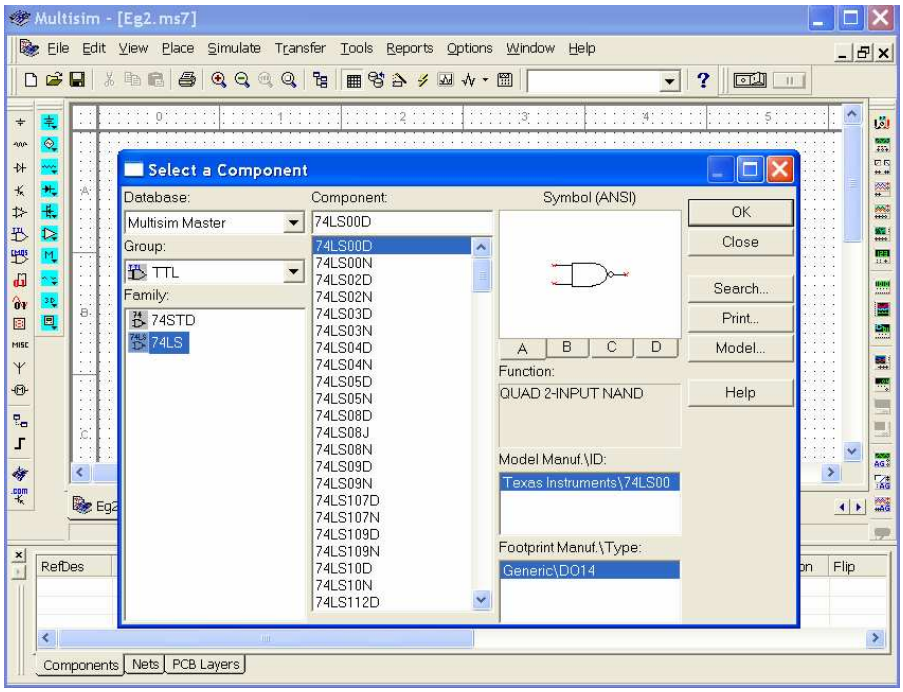
Ο προσομοιωτής Multisim 7 είναι ένα περιβάλλον λογισμικού που επιτρέπει τη σχεδίαση και προσομοίωση της λειτουργίας ηλεκτρονικών κυκλωμάτων. Η σχεδίαση γίνεται με τη βοήθεια βιβλιοθηκών, οι οποίες περιέχουν μοντέλα των πιο βασικών ηλεκτρονικών εξαρτημάτων και ολοκληρωμένων κυκλωμάτων που υπάρχουν σήμερα στην ηλεκτρονική αγορά.

Στο εργαστήριο των προηγμένων ψηφιακών θα χρησιμοποιήσουμε τον προσομοιωτή για να μελετήσουμε τη λειτουργία βασικών ψηφιακών κυκλωμάτων, πριν τα υλοποιήσουμε στο ράστερ με πραγματικά εξαρτήματα. Η προσομοίωση είναι μια βασική σχεδιαστική πρακτική, καθώς όσο πιο περίπλοκη και απαιτητική είναι μια εφαρμογή, τόσο μεγαλύτερη ανάγκη υπάρχει να την μελετήσει κανείς θεωρητικά πριν την υλοποιήσει πρακτικά.

Οι σπουδαστές θα χρησιμοποιήσουν τον προσομοιωτή και για την εκπόνηση των εργασιών τους.

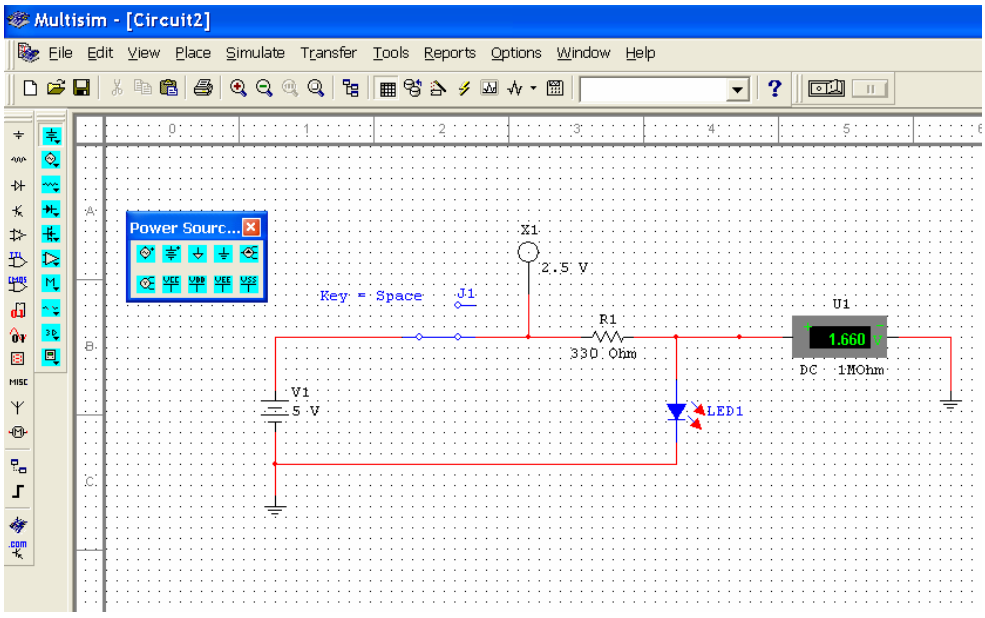
1.2.2 Οι βιβλιοθήκες του Multisim

1. Η παρακάτω εικόνα παρουσιάζει το σχεδιαστικό περιβάλλον του Multisim. Αριστερά φαίνονται οι βιβλιοθήκες σχεδίασης, κατά ομάδες κυκλωμάτων και εξαρτημάτων. Δεξιά φαίνονται τα εικονικά όργανα μετρήσεων. Είναι ανοιχτή η βιβλιοθήκη των κυκλωμάτων TTL.



Σχήμα 1.1 *Σχεδιαστικό περιβάλλον του Multisim 7. Φαίνεται η βιβλιοθήκη των κυκλωμάτων TTL.*

Να περιηγηθείτε μία-μία τις βιβλιοθήκες, ώστε να κατανοήσετε το περιεχόμενο τους. Να μελετήσετε κυρίως τις βιβλιοθήκες των ολοκληρωμένων κυκλωμάτων TTL και CMOS. Επίσης παρατηρείστε προσεχτικά το περιεχόμενο της βασικής βιβλιοθήκης (basic), της βιβλιοθήκης των πηγών (power source) και των οργάνων μέτρησης (measurement components).



Σχήμα 1.2 *Απλή εφαρμογή με LED και φωτεινό ενδείκτη*

1.2.3 Συνδυαστική σχεδίαση με απλές πύλες

1. Να σχεδιάσετε το κύκλωμα που φαίνεται στο παραπάνω σχήμα 1.2. Εντοπίστε τα εξαρτήματα στις βιβλιοθήκες που αναφέρθηκαν στην προηγούμενη παράγραφο. Δώστε προσοχή στις συνδέσεις, καθώς οι κακές συνδέσεις ανάμεσα στα εξαρτήματα μοιραία θα εμποδίσουν τη σωστή λειτουργία του κυκλώματος. Παρατηρείστε την ενεργοποίηση της ενδεικτικής λυχνίας και του LED.

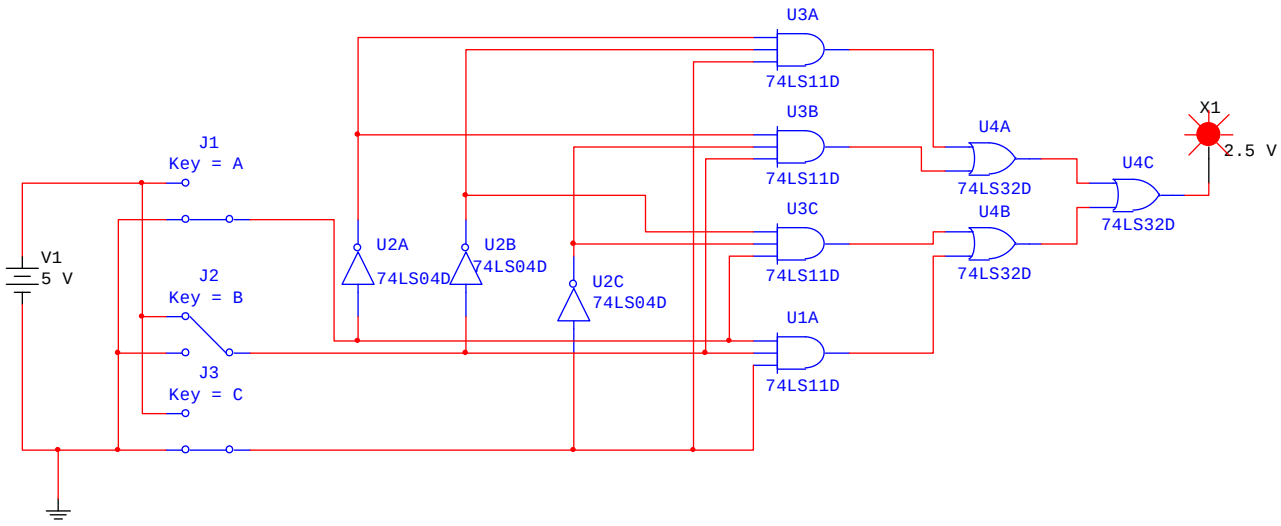
2. Έστω ο πίνακας αληθείας

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Να απλοποιήσετε τη συνάρτηση με τη βοήθεια του πίνακα Karnaugh.
Να σχεδιάσετε το συνδυαστικό κύκλωμα που προκύπτει στο Multisim και να επιβεβαιώσετε τη λειτουργία του.

1.2.4 Απλά συστήματα συνδυαστικής λογικής – Λαμπτήρας τριών δρόμων

Να σχεδιαστεί ένα ψηφιακό σύστημα που να εκτελεί την παρακάτω λογική: Ένας λαμπτήρας πρέπει να ελέγχεται από τρεις διακόπτες σε διαφορετικά σημεία του χώρου, ώστε να είναι δυνατή η ενεργοποίηση ή αποενεργοποίηση του λαμπτήρα, αλλάζοντας τη θέση (on ή off) οποιουδήποτε από τους τρεις διακόπτες.



Σχήμα 1.3 *Υλοποίηση του συστήματος λαμπτήρα τριών δρόμων με τη μορφή αθροίσματος γινομένων*

α) Να υπολογιστεί ο πίνακας αληθείας που περιγράφει τη λειτουργία του συστήματος.

β) Να γραφεί ή συνάρτηση που περιγράφει τη λειτουργία του κυκλώματος ως άθροισμα γινομένων και ως γινόμενο αθροισμάτων.

γ) Να σχεδιαστεί στο Multisim η παραπάνω υλοποίηση του σχ. 1.3 με τη μορφή γινομένου αθροισμάτων και να επιβεβαιωθεί η λειτουργία του κυκλώματος με τη βοήθεια της προσομοίωσης.

1.2.5 Σύστημα συναγερμού

Ένα κύκλωμα συναγερμού περιγράφεται ως εξής: Η έξοδος του συστήματος ονομάζεται «συναγερμός» και οι είσοδοι είναι: «Πανικός», «ενεργοποίηση», «έξοδος», καθώς και οι είσοδοι από τα ανοίγματα του σπιτιού, δηλαδή, «παράθυρο», «πόρτα» και «γκαράζ».

Αν θέσουμε την είσοδο «πανικός» σε 1, τότε ο συναγερμός χτυπά σε κάθε περίπτωση. Αλλιώς, για να λειτουργήσει ο συναγερμός πρέπει να τον έχουμε ενεργοποιήσει («ενεργοποίηση»=1) και να έχουμε βγει από το σπίτι («έξοδος»=1) και κάποιο από τα τρία ανοίγματα να μην είναι ασφαλισμένο (δηλαδή «πόρτα» ή «παράθυρο» ή «γκαράζ» = 0).

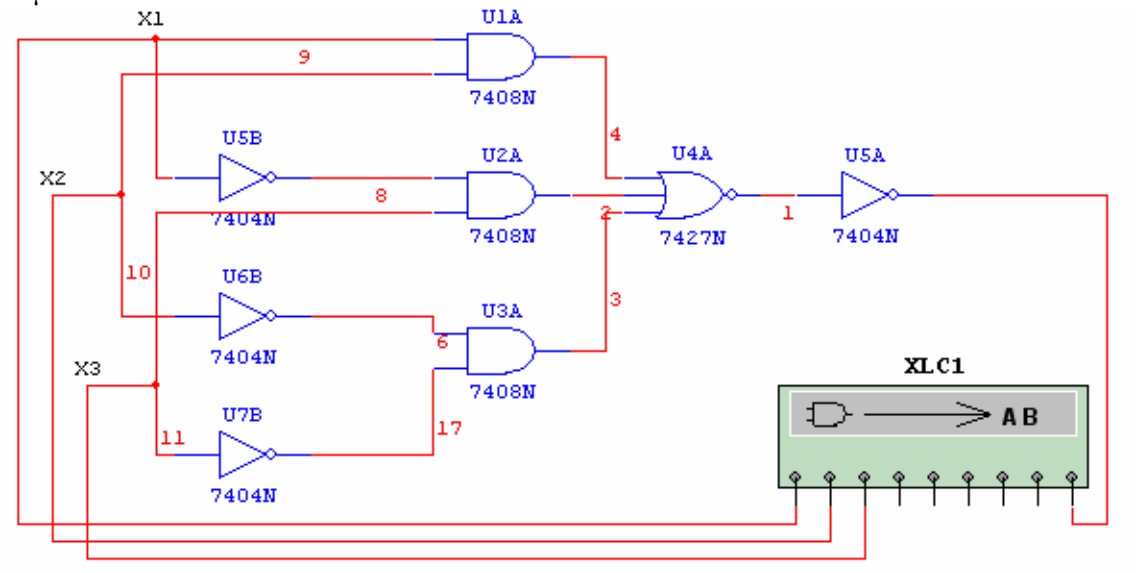
Να σχεδιάσετε ένα κύκλωμα που να υλοποιεί την παραπάνω λογική και να το προσομοιώσετε στο Multisim.

1.2.6 Ο λογικός μετατροπέας του Multisim

Ο Λογικός Μετατροπέας (Logic Converter) του Multisim είναι ένα εικονικό ηλεκτρονικό όργανο, που βρίσκεται στη σειρά εργαλείων Instruments Toolbar. Είναι εικονικό εργαλείο και δεν έχει αντίστοιχο στον πραγματικό κόσμο. Μπορεί, μεταξύ άλλων, από ένα ψηφιακό κύκλωμα που έχουμε σχεδιάσει να εξάγει αυτόματα:

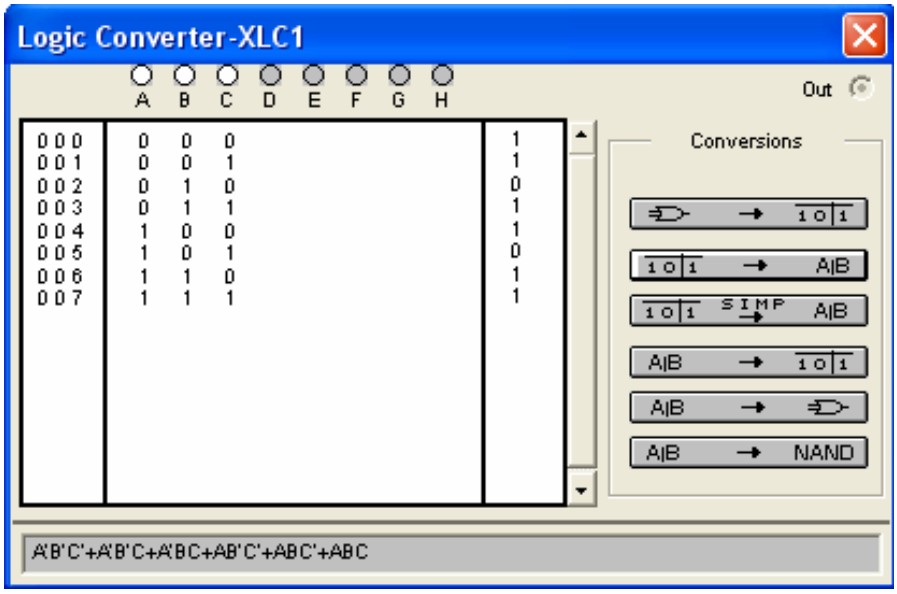
1. Τον πίνακα αληθείας
2. Την συνάρτηση Boole του κυκλώματος
3. Την απλοποιημένη συνάρτηση Boole
4. Το ίδιο κύκλωμα υλοποιημένο μόνον με πύλες NAND

Στο σχ. 1.4 φαίνεται ο τρόπος σύνδεσης του Λογικού Μετατροπέα σε ένα συνδυαστικό κύκλωμα.



Σχήμα 1.4 *Σύνδεση Λογικού Μετατροπέα σε κύκλωμα 3 εισόδων και 1 εξόδου*

Διπλοπατώντας πάνω στο εικονίδιο του μετατροπέα ανοίγει το παράθυρο με τις διάφορες λειτουργίες του (σχ. 1.5). Από εκεί μπορούμε να δούμε τον πίνακα αληθείας του κυκλώματος, την συνάρτηση Boole και την απλοποιημένη συνάρτηση Boole.



Σχήμα 1.5 *Το παράθυρο λειτουργιών του Λογικού Μετατροπέα*

Να σχεδιάσετε το κύκλωμα αυτό στο Multisim και να εξάγετε, με τη βοήθεια του Λογικού Μετατροπέα, τον πίνακα αληθείας, την συνάρτηση Boole, την απλοποιημένη συνάρτηση Boole και το κύκλωμα υλοποιημένο με πύλες NAND.

1.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

Να παραδώσετε μια επιμελημένη εργαστηριακή αναφορά απαντώντας στα παρακάτω ερωτήματα:

1. Να σχεδιάσετε ένα συνδυαστικό κύκλωμα στο Multisim η έξοδος Y του οποίου θα ενεργοποιεί (με λογικό “1”) έναν συναγερμό αυτοκινήτου. Το κύκλωμα έχει τρεις εισόδους A, B, C, που συνδέονται με τις εξόδους τριών αισθητήρων sA, sB, sC, οι οποίοι ελέγχουν αντίστοιχα: αν η μηχανή είναι σε λειτουργία, αν κάποια πόρτα είναι ανοιχτή και αν το καπό της μηχανής είναι ανοιχτό. Ο συναγερμός πρέπει να ενεργοποιείται όταν η μηχανή είναι σε λειτουργία και ταυτόχρονα είτε κάποια πόρτα είναι ανοιχτή είτε το καπό είναι ανοιχτό.
   - Να κατασκευάσετε τον πίνακα αληθείας του προβλήματος.
   - Να γράψετε τη συνάρτηση εξόδου Y ως άθροισμα ελαχίστων όρων των μεταβλητών εισόδου A, B, C
   - Να απλοποιήσετε τη συνάρτηση με τη βοήθεια του πίνακα του Karnaugh
   - Να σχεδιάσετε στο Multisim το τελικό κύκλωμα με πύλες AND-OR-NOT. Οι αισθητήρες να προσομοιωθούν ως απλοί διακόπτες
   - Να σχεδιάσετε στο Multisim τέλος το ίδιο κύκλωμα χρησιμοποιώντας μόνον πύλες NAND

2. Ένα βενζινάδικο έχει τέσσερις δεξαμενές καυσίμων. Στη δεξαμενή της βενζίνης SUPER υπάρχει ένας αισθητήρας SA που δίνει "1", όταν η στάθμη πέσει κάτω από κάποιο προκαθορισμένο όριο. Στη δεξαμενή της βενζίνης UNLEADED υπάρχει ένας αισθητήρας SB που δίνει "1", όταν η στάθμη πέσει κάτω από κάποιο προκαθορισμένο όριο. Στη δεξαμενή της βενζίνης SUPER UNLEADED υπάρχει ένας αισθητήρας SC που δίνει "1", όταν η στάθμη πέσει κάτω από κάποιο προκαθορισμένο όριο. Στη δεξαμενή του πετρελαίου DIESEL υπάρχει ένας αισθητήρας SD που δίνει "1", όταν η θερμοκρασία υπερβεί κάποιο προκαθορισμένο όριο.

Να σχεδιάσετε ένα συνδυαστικό κύκλωμα με τέσσερις εισόδους A, B, C και D, που συνδέονται με τέσσερις διακόπτες, στη θέση των τεσσάρων αισθητήρων, και μία έξοδο Y που δίνει "1", όταν η στάθμη τουλάχιστον μίας από τις δεξαμενές βενζίνης πέσει κάτω από το προκαθορισμένο όριο και ταυτόχρονα η θερμοκρασία της δεξαμενής του πετρελαίου κίνησης υπερβεί το προκαθορισμένο όριο.

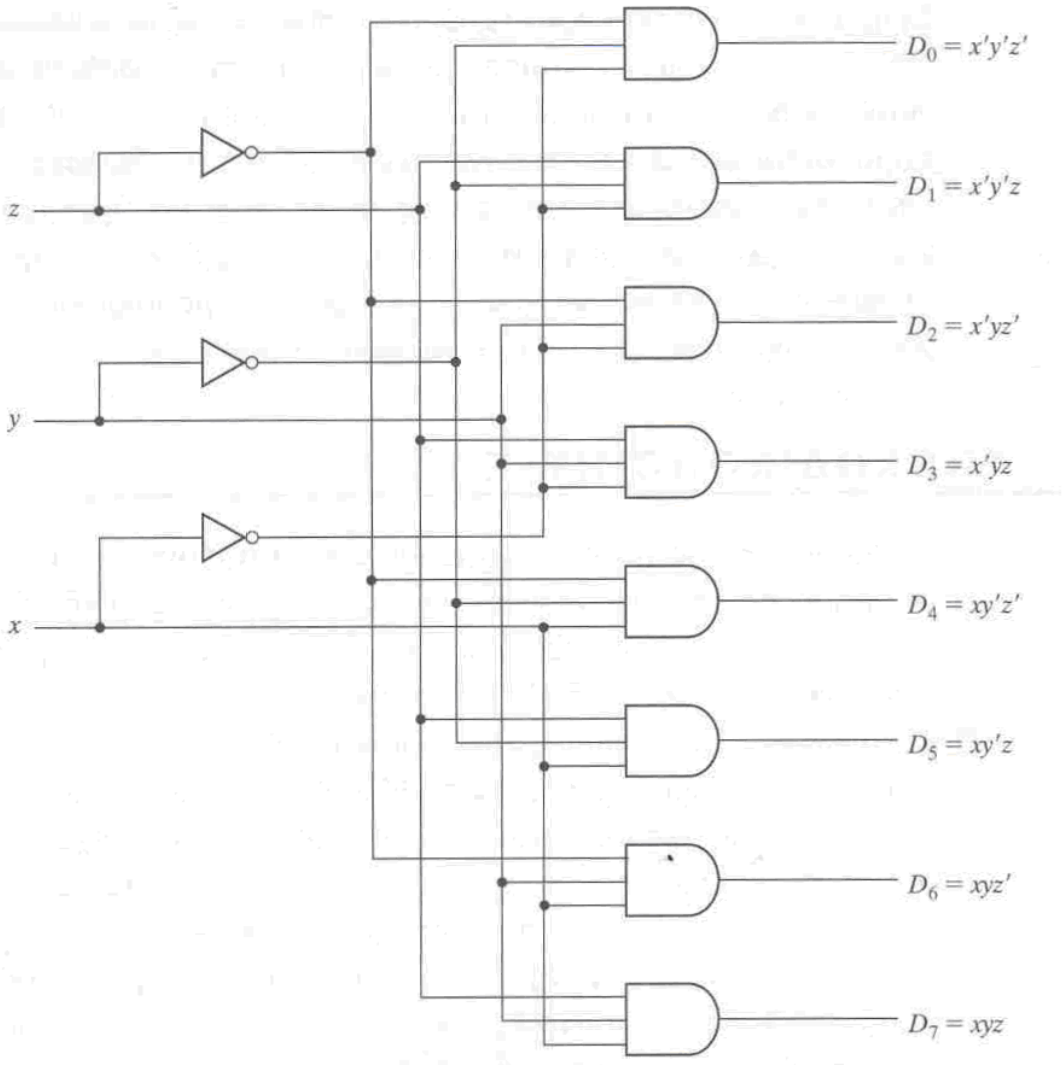
- Να κατασκευάσετε τον πίνακα αληθείας του προβλήματος.
- Να γράψετε τη συνάρτηση εξόδου Y ως άθροισμα ελαχίστων όρων των μεταβλητών εισόδου A, B, C, D
- Να απλοποιήσετε τη συνάρτηση με τη βοήθεια του πίνακα του Karnaugh
- Να σχεδιάσετε στο Multisim το τελικό κύκλωμα με πύλες AND-OR-NOT. Οι αισθητήρες να προσομοιωθούν ως απλοί διακόπτες

ΕΡΓΑΣΤΗΡΙΟ 2

ΠΟΛΥΠΛΕΚΤΕΣ – ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΕΣ
ΑΠΟΜΟΝΩΤΕΣ ΤΡΙΩΝ ΚΑΤΑΣΤΑΣΕΩΝ

2.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Ο ***αποκωδικοποιητής*** είναι ένα κύκλωμα με *n* εισόδους και *m* εξόδους, όπου $m \leq 2^n$. Για κάθε συνδυασμό των εισόδων, ο αποκωδικοποιητής επιλέγει μία από τις *m* εξόδους και τη φέρνει σε λογικό 1, ενώ οι υπόλοιπες παραμένουν σε λογικό μηδέν. Ένας αποκωδικοποιητής 3 σε 8 φαίνεται στο παρακάτω σχήμα 2.1. Οι τρεις είσοδοι αποκωδικοποιούνται σε οκτώ εξόδους, που η κάθε μια αντιπροσωπεύει έναν από τους ελάχιστους όρους (minterms) των *n* μεταβλητών εισόδου. Για παράδειγμα, όταν οι είσοδοι x,y,z είναι 000, τότε ο ελάχιστος όρος που παράγεται είναι ο x΄y΄z΄, στην έξοδο D0. Ο αντίστοιχος πίνακας αληθείας φαίνεται στο σχ. 2.2.



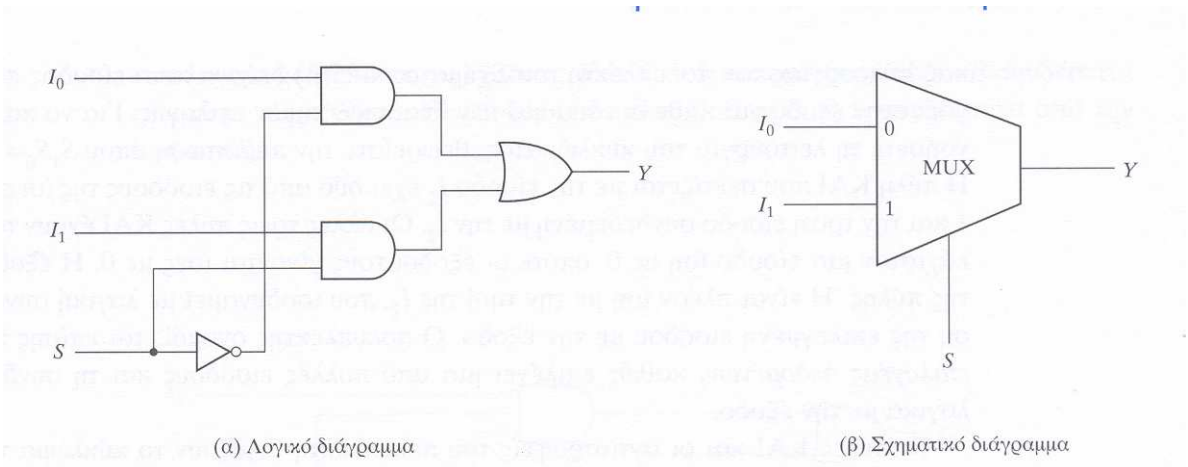
Σχήμα 2.1 *Κύκλωμα αποκωδικοποιητή 3 σε 8. Στις εξόδους παράγονται οι ελάχιστοι όροι που αντιστοιχούν στις μεταβλητές εισόδου.*

Πίνακας αληθείας ενός αποκωδικοποιητή 3-σε-8

Είσοδοι			Έξοδοι							
x	*y*	*z*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Σχήμα 2.2 *Πίνακας αληθείας αποκωδικοποιητή 3 σε 8.*

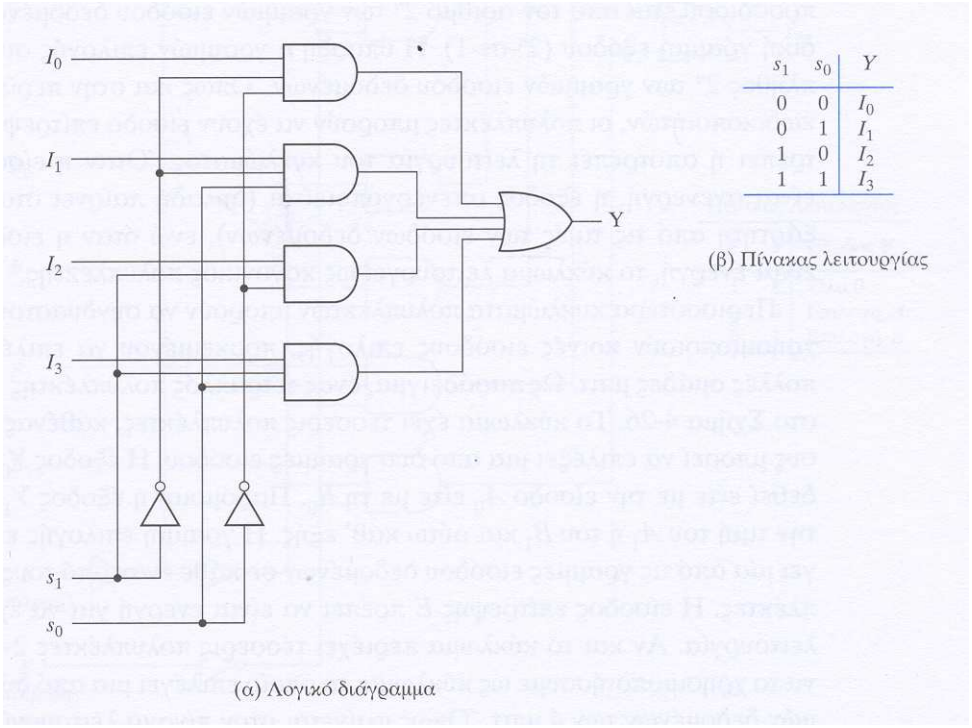
Ο ***πολυπλέκτης*** είναι ένα συνδυαστικό κύκλωμα, το οποίο επιλέγει τη δυαδική πληροφορία μιας από πολλές γραμμές εισόδου και την κατευθύνει σε μια μοναδική γραμμή εξόδου. Η επιλογή της συγκεκριμένης γραμμής εξόδου ελέγχεται από ειδικές γραμμές επιλογής. Κανονικά, υπάρχουν 2^n γραμμές εισόδου και n γραμμές επιλογής, των οποίων οι τιμές καθορίζουν ποια είσοδος επιλέγεται.



Σχήμα 2.3 *Πολυπλέκτης 2-σε-1 (α) Λογικό διάγραμμα, (β) Σχηματικό διάγραμμα.*

Στο παραπάνω σχήμα 2.3 δύο πηγές του ενός bit συνδέονται σε μία έξοδο Y, ανάλογα με τις τιμές της εισόδου επιλογής S.

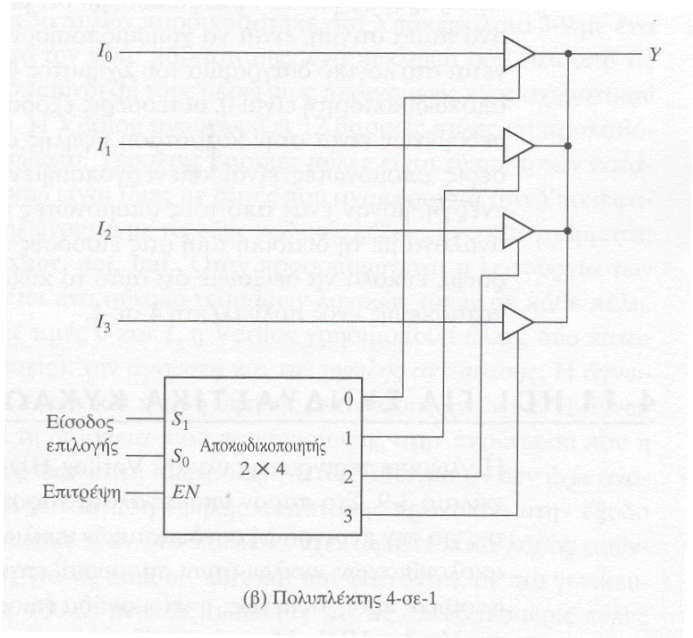
Στο παρακάτω σχήμα 4 φαίνεται ένας πολυπλέκτης 4-σε-1. Κάθε μια από τέσσερις εισόδους I_0 έως I_3 συνδέεται σε μια από τις εισόδους μιας πύλης AND. Οι γραμμές επιλογής S_1 και S_2 αποκωδικοποιούνται έτσι, ώστε να επιλέξουν μία μόνο πύλη AND. Οι έξοδοι των AND τροφοδοτούν τις εισόδους μιας μοναδικής πύλης OR, της οποίας η έξοδος είναι και έξοδος του κυκλώματος.



Σχήμα 2.4 *Πολυπλέκτης 4-σε-1 και πίνακας αληθείας*

Ένας πολυπλέκτης μπορεί να λειτουργήσει και σαν γεννήτρια λογικών συναρτήσεων. Η λειτουργία αυτή φαίνεται με την άσκηση 1.

Τέλος, ένας πολυπλέκτης μπορεί να κατασκευαστεί με τη βοήθεια ενός αποκωδικοποιητή και με απομονωτές τριών καταστάσεων, όπως φαίνεται στο σχήμα 2.5. Μια τέτοια διάταξη υλοποιείται και στην άσκηση 3.



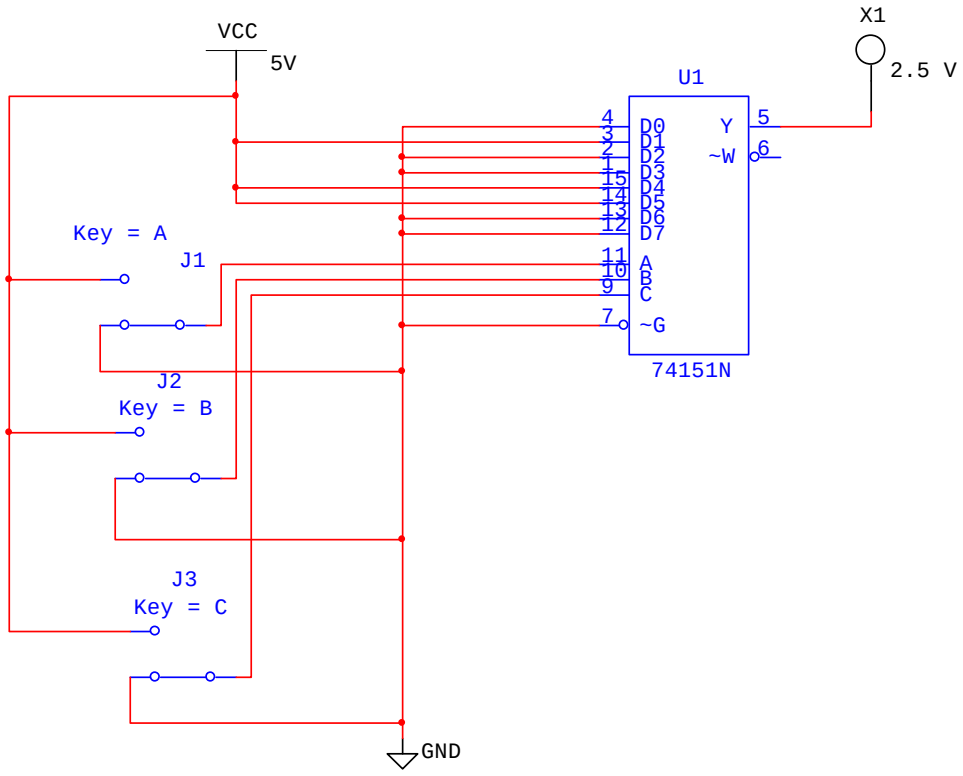
Σχήμα 2.5 *Υλοποίηση πολυπλέκτη με αποκωδικοποιητή και απομονωτές τριών καταστάσεων*

2.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

2.2.1. Πολυπλέκτες

Γίνεται χρήση του πολυπλέκτη 74LS151 και μελετάται ο ρόλος του ως γεννήτριας συναρτήσεων.

Α. Να υλοποιήσετε στο Multisim 7.0 το παρακάτω κύκλωμα με τον πολυπλέκτη 74LS151.



Σχήμα 2.6 *Ο πολυπλέκτης 74LS151 παράγει έναν προκαθορισμένο πίνακα αληθείας, με μεταβλητές εισόδου τις τιμές των εισόδων επιλογής*

Β. Στη συνέχεια να συμπληρώσετε τον παρακάτω πίνακα, δίνοντας τις κατάλληλες τιμές στις γραμμές επιλογής A,B,C.

C	B	A	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Γ. Να σχεδιάσετε με τον πολυπλέκτη 74LS151 ένα κύκλωμα που υλοποιεί τη λογική συνάρτηση με τον παρακάτω πίνακα αληθείας

C	**B**	**A**	**Y**
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

2.2.2. Απομονωτές τριών καταστάσεων και δυαδικοί αποκωδικοποιητές.

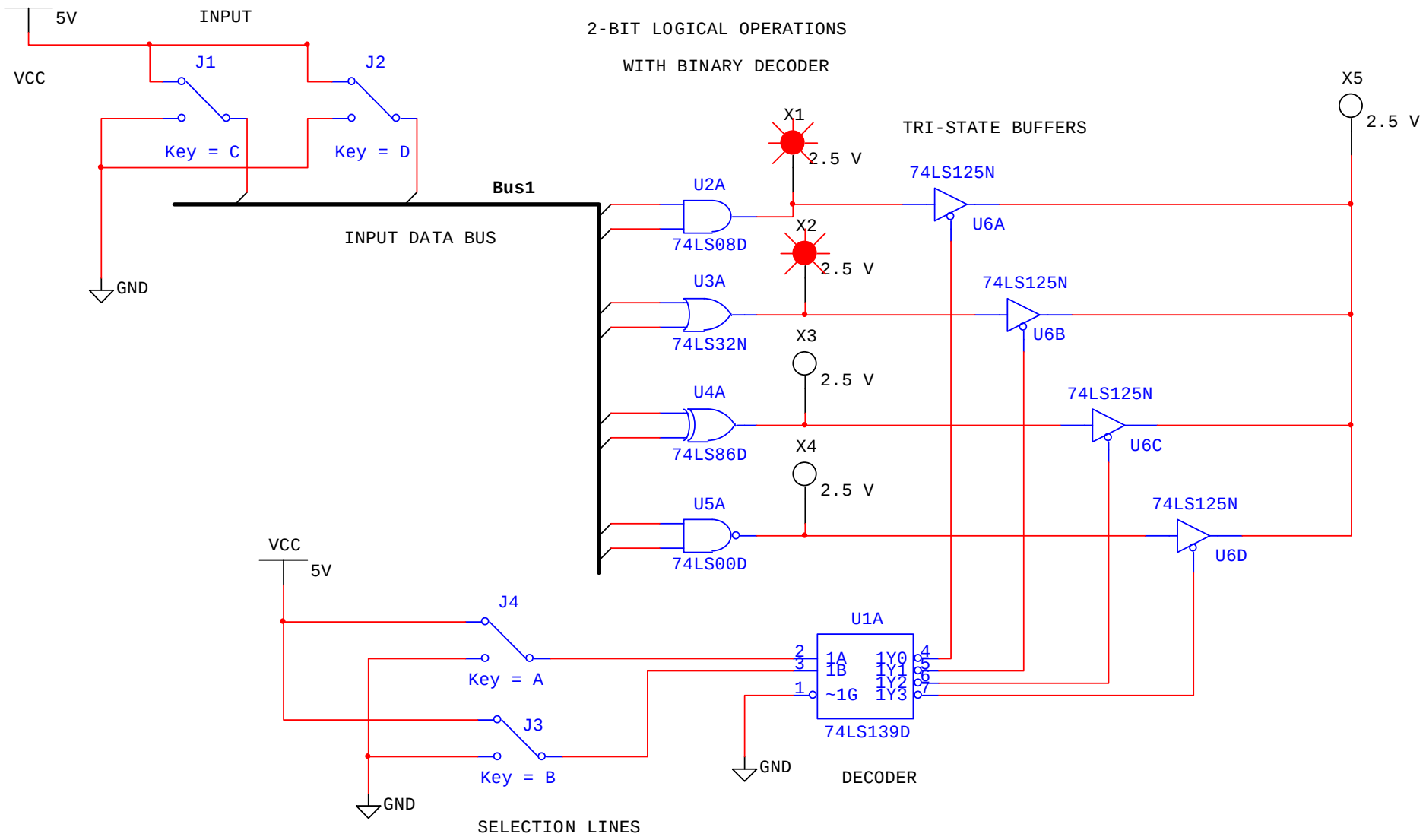
Στο εργαστήριο αυτό μελετούμε τον ρόλο του δυαδικού αποκωδικοποιητή και των απομονωτών τριών καταστάσεων (tri-state buffers) σε ένα κύκλωμα που επιτελεί τέσσερις βασικές λογικές πράξεις 2-bits (AND, OR, XOR, NAND).

Χρησιμοποιούμε ένα πληκτρολόγιο δύο bits για την είσοδο των δύο bits. Για λόγους εξοικονόμησης γραμμών χρησιμοποιούμε ένα bus 2-bits για τη μεταφορά της πληροφορίας στις τέσσερις πύλες. Το bus τοποθετείται με δεξί κλικ και επιλέγοντας Place Bus. Ακολουθήστε τις προφορικές οδηγίες του διδάσκοντα για τη χρήση του διαδρόμου.

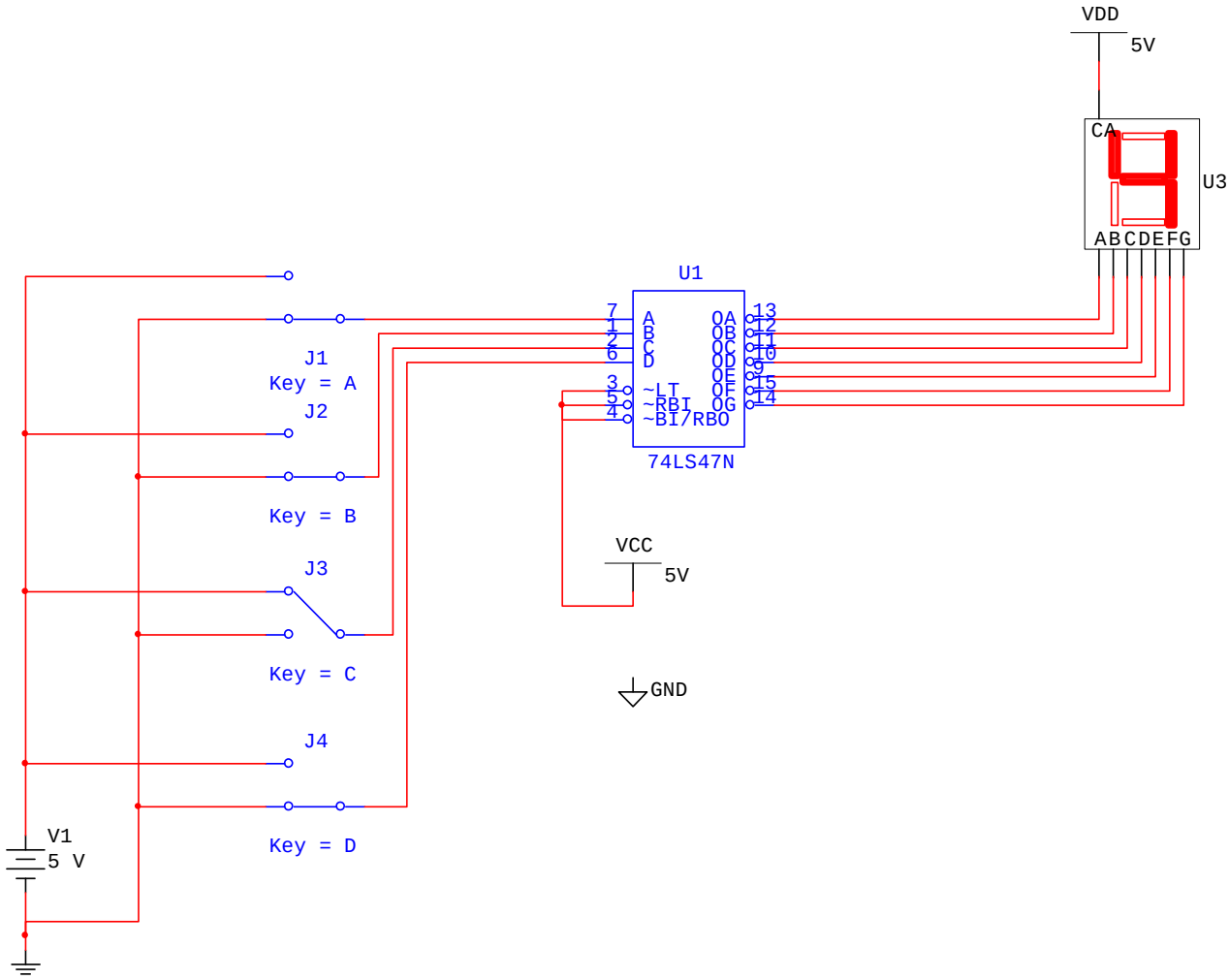
Αρχικά ελέγξτε τη λειτουργία των πυλών με τη βοήθεια των ενδεικτικών λυχνιών (probes). Στη συνέχεια θα δημιουργήσουμε μια διάταξη με την οποία θα επιλέγουμε την πράξη της οποίας το αποτέλεσμα θα εμφανίζεται στην έξοδο του κυκλώματος. Η έξοδος του κυκλώματος θα είναι 1-bit που θα απεικονίζεται σε μια ενδεικτική λυχνία.

Για την επιλογή της πράξης θα χρησιμοποιήσουμε τον δυαδικό αποκωδικοποιητή 74LS139. Οι έξοδοι του αποκωδικοποιητή ελέγχουν τις εισόδους enable τεσσάρων απομονωτών τριών καταστάσεων (tri-state). Η επιλογή της πράξης γίνεται με τις εισόδους επιλογής του αποκωδικοποιητή. Το κύκλωμα αυτό παίζει το ρόλο του αποκωδικοποιητή εντολών σε μια αριθμητική μονάδα τεσσάρων λογικών πράξεων των 2-bits.

Το δεύτερο κύκλωμα που παρουσιάζεται είναι ένα παράδειγμα της λειτουργίας του αποκωδικοποιητή BCD-to-seven-segment. Πρόκειται για τον αποκωδικοποιητή 74LS47, που δέχεται είσοδο BCD και οδηγεί τον ενδείκτη επτά τομέων (οκταράκι) κοινής ανόδου.



Σχήμα 2.7 *Χρήση των απομονωτών τριών καταστάσεων και του αποκωδικοποιητή 74LS139 σε ένα κύκλωμα επιλογής ανάμεσα σε τέσσερις λογικές πράξεις*



Σχήμα 2.8 *Αποκωδικοποιητής 74LS47 (BCD-to-7-segment) για οδήγηση απεικόνισης επτά τομέων*

2.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

1. Να βρείτε στο διαδίκτυο τα φύλλα δεδομένων του πολυπλέκτη **CD4051**. Να παρουσιάσετε με δικά σας λόγια τα βασικά χαρακτηριστικά του και τις βασικές εφαρμογές-λειτουργίες όπου μπορεί να χρησιμοποιηθεί.

2. Να υλοποιήσετε πολυπλέκτη **16:1** χρησιμοποιώντας δύο ολοκληρωμένα κυκλώματα **74LS151**. Να αναφερθείτε στην παράγραφο 5.7.2 του εγχειριδίου της θεωρίας σας (Wakerly). Να εξηγήσετε λεπτομερώς τη λειτουργία του κυκλώματος που σχεδιάσατε.

3. Να υλοποιήσετε την παρακάτω λογική συνάρτηση τριών μεταβλητών με πολυπλέκτη:

   $$F(x,y,z)=\Sigma(1,2,6,7)$$

   όπου στο δεξί μέρος έχουμε το άθροισμα των αντίστοιχων ελαχίστων όρων.

ΕΡΓΑΣΤΗΡΙΟ 3

ΑΘΡΟΙΣΤΕΣ-ΑΦΑΙΡΕΤΕΣ

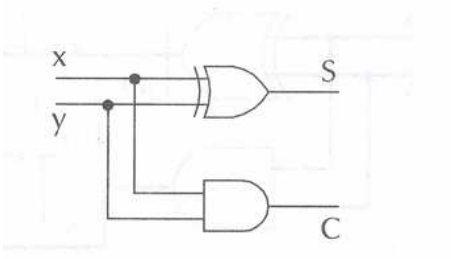
3.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Στο εργαστήριο αυτό παρουσιάζονται τα βασικά αριθμητικά κυκλώματα της πρόσθεσης και της αφαίρεσης και προσομοιώνονται στο περιβάλλον του Multisim. Επίσης, παρουσιάζεται η δημιουργία ιεραρχικής βαθμίδας στο Multisim, η οποία στη συνέχεια μπορεί να χρησιμοποιηθεί ως αρχείο βιβλιοθήκης, προκειμένου να κατασκευαστούν σύνθετα κυκλώματα.

3.1.1. Ο ημιαθροιστής και ο πλήρης αθροιστής

Ο ημιαθροιστής είναι ένα από τα πιο βασικά αριθμητικά κυκλώματα. Εκτελεί την πρόσθεση δύο δυαδικών ψηφίων και παράγει ως έξοδο το άθροισμα και το κρατούμενο.

x	y	C	S
0	0	0	0
0	0	0	1
1	0	0	1
1	1	1	0



Σχήμα 3.1 *Πίνακας αληθείας και σχηματικό κύκλωμα του ημιαθροιστή*

Ο πλήρης αθροιστής (Full Adder) είναι ένα συνδυαστικό κύκλωμα που εκτελεί την πρόσθεση δύο δυαδικών ψηφίων λαμβάνοντας υπόψη και την ύπαρξη κρατούμενου προηγούμενης τάξης. Ο πίνακας αληθείας του πλήρη αθροιστή φαίνεται στον πίνακα 3.1. Στον πίνακα αυτό με z συμβολίζεται το κρατούμενο εισόδου.

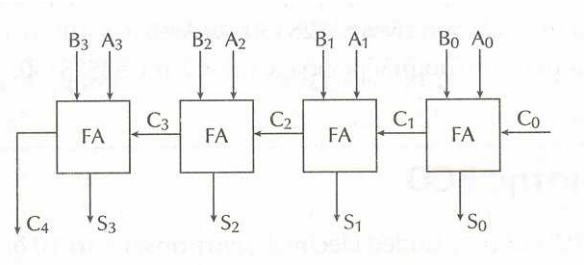
ΠΙΝΑΚΑΣ 3.1

x	y	z	C	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

Το κύκλωμα του πλήρη αθροιστή φαίνεται στο σχ. 3.4.

Για την πρόσθεση αριθμών των τεσσάρων bits χρησιμοποιούμε το κύκλωμα του παράλληλου πλήρη αθροιστή με διάδοση κρατούμενου (ripple carry) που φαίνεται στο παρακάτω σχήμα 3.2. Ο αθροιστής αυτός προσθέτει δύο ψηφιολέξεις $A_3A_2A_1A_0$ και $B_3B_2B_1B_0$.

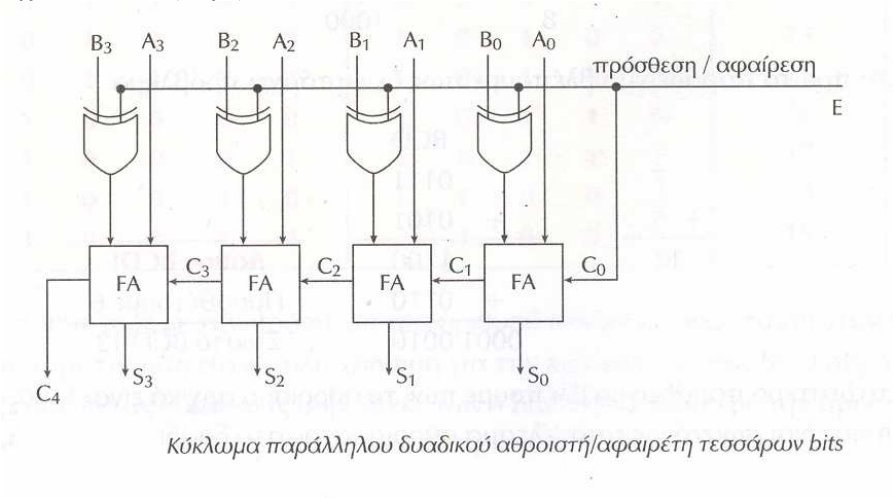
Αποτελείται από τέσσερις πλήρεις αθροιστές, που ο καθένας αθροίζει δύο bits. Το κρατούμενο που προκύπτει κατά την άθροιση των δύο ελάχιστα σημαντικών bits, εφαρμόζεται στην είσοδο του κρατουμένου του επόμενου αθροιστή. Παρομοίως, το νέο κρατούμενο διαδίδεται προς την επόμενη βαθμίδα. Με τον ίδιο τρόπο μπορούμε να δημιουργήσουμε αθροιστές περισσότερων bits.



Σχήμα 3.2 *Κύκλωμα παράλληλου αθροιστή τεσσάρων bits*

3.1.2 Αφαίρεση με το συμπλήρωμα ως προς 2

Η πράξη της αφαίρεσης μπορεί να γίνει με κυκλώματα αφαιρετών, όμως μπορεί να γίνει και με αθροιστές, χρησιμοποιώντας το συμπλήρωμα ως προς 2 του αφαιρετέου. Με τον τρόπο αυτόν μπορούμε να χρησιμοποιήσουμε το ίδιο κύκλωμα τόσο για την πράξη της πρόσθεσης όσο και για την πράξη της αφαίρεσης. Το συμπλήρωμα ως προς 2 παράγεται ως γνωστό, βρίσκοντας πρώτα το συμπλήρωμα ως προς 1 και προσθέτοντας τη μονάδα. Το συμπλήρωμα ως προς 1 σημαίνει λογική αντιστροφή των του αριθμού, ενώ η πρόσθεση της μονάδας μπορεί να γίνει εφαρμόζοντας στην είσοδο του κρατουμένου του πρώτου αθροιστή (LSB) τη μονάδα. Στο σχ. 3.3 φαίνεται ένα κύκλωμα που επιτελεί πρόσθεση ή αφαίρεση ανάλογα με τη λογική κατάσταση της εισόδου E (0 ή 1).



Σχήμα 3.3 *Κύκλωμα αθροιστή/αφαιρέτη με βάση το συμπλήρωμα ως προς 2*

3.1.3 Ιεραρχική σχεδίαση

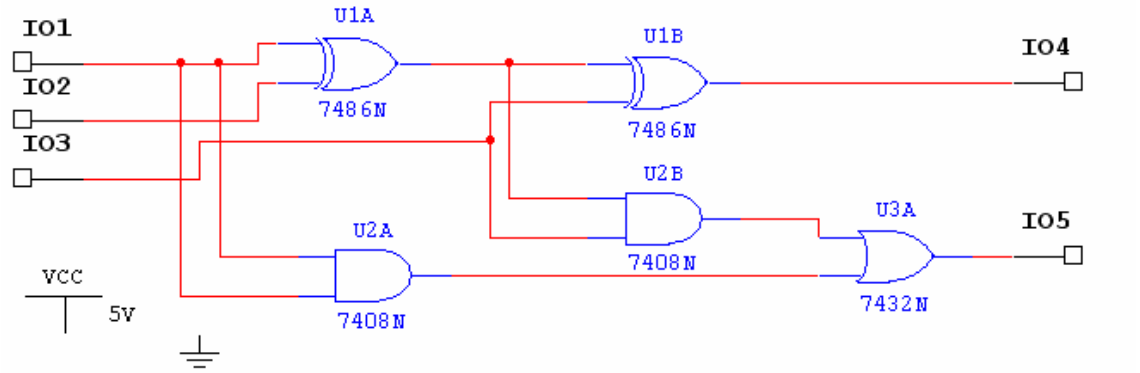
Στο εργαστήριο αυτό θα εξοικειωθούμε και με μία ακόμη έννοια της ψηφιακής σχεδίασης, την λεγόμενη ιεραρχική σχεδίαση. Κάθε απλό κύκλωμα είναι δυνατό να συμπεριληφθεί σε μια

βαθμίδα υλικού, με τη μορφή ενός "block" και να αποθηκευτεί με το δικό του όνομα. Πολλές τέτοιες βαθμίδες είναι δυνατό να συμπεριληφθούν σε μια ανώτερη σύνθετη βαθμίδα και να αποθηκευτούν ως ένα νέο σχεδιαστικό block, ως αρχείο βιβλιοθήκης. Με τον τρόπο αυτό είναι δυνατό να υλοποιήσουμε σύνθετα κυκλώματα, χρησιμοποιώντας ελάχιστο αριθμό βαθμίδων, που η κάθε μια περιλαμβάνει επί μέρους κυκλώματα. Έτσι, απλοποιείται σημαντικά η διαδικασία της σχεδίασης, και το τελικό κύκλωμα αποκτά μια πολύ πιο κατανοητή μορφή.

Στην ιεραρχική σχεδίαση κάθε σύνθετο κύκλωμα αποτελεί μια κορυφαία ("top") ιεραρχική οντότητα, η οποία με τη σειρά της αποτελείται από έναν αριθμό υποκυκλωμάτων.

3.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

1. Να υλοποιήσετε το παρακάτω κύκλωμα του πλήρη αθροιστή δύο bits ως ιεραρχικό block ακολουθώντας τα παρακάτω βήματα:



Σχήμα 3.4 *Πλήρης αθροιστής δύο bits*

Α) Δημιουργούμε ένα νέο σχηματικό αρχείο (add_sub.ms7) και το αποθηκεύουμε. Για να δημιουργήσουμε μια ιεραρχική βαθμίδα, που επιτελεί συγκεκριμένη λογική εργαζόμαστε στην συνέχεια ως εξής:

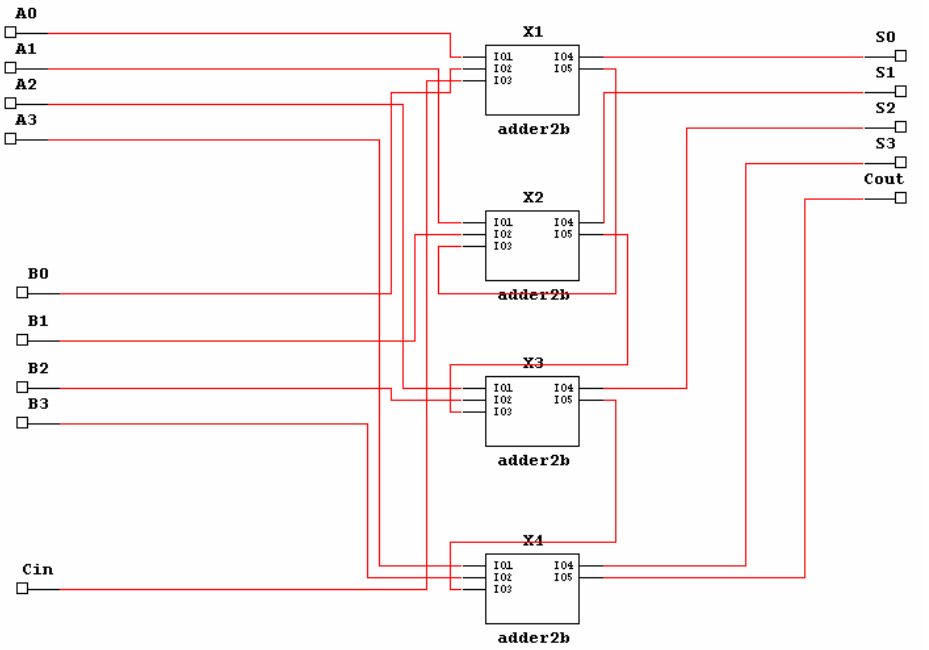
Επιλέγουμε στο βασικό menu: Place, Create Hierarchical Block. Ονομάζουμε τη βαθμίδα (π.χ adder2b) και ορίζουμε τον αριθμό των εισόδων και των εξόδων της. Έτσι, ο πλήρης αθροιστής των δύο bits θα έχει προφανώς τρεις εισόδους (μία για κάθε bit και μία για το κρατούμενο προηγούμενης τάξης) και δύο εξόδους (το άθροισμα και το επόμενο κρατούμενο).

Β) Τοποθετούμε το block στην επιφάνεια εργασίας. Διπλοπατούμε πάνω στο block και επιλέγουμε Edit Subcircuit. Ανοίγει ένα νέο σχηματικό διάγραμμα, που περιέχει αυτόματα έναν αριθμό εισόδων και εξόδων, όπου μπορούμε να σχεδιάσουμε το κύκλωμα της ιεραρχικής βαθμίδας. Είναι σκόπιμο να προσθέσουμε την τάση τροφοδοσίας Vcc και τη γείωση. **Αποθηκεύοντας (Save as: adder2b.ms7)** το σχέδιό μας, έχουμε δημιουργήσει μια ιεραρχική βαθμίδα.

Γ) Πριν προχωρήσετε στη δημιουργία του επόμενου ιεραρχικού αρχείου, ελέγξτε τη σωστή λειτουργία της βαθμίδας που υλοποιήσατε, συνδέοντάς την με 3 διακόπτες για τις εισόδους και 2 probe για τις εξόδους, επαληθεύοντας τον πίνακα αληθείας του πλήρους αθροιστή (Πίνακας 3.1)

Για να κάνουμε την εισαγωγή ενός ιεραρχικού block επιλέγουμε από το μενού Place, Hierarchical block και εισάγουμε το block όπως κάθε σχηματικό αρχείο.

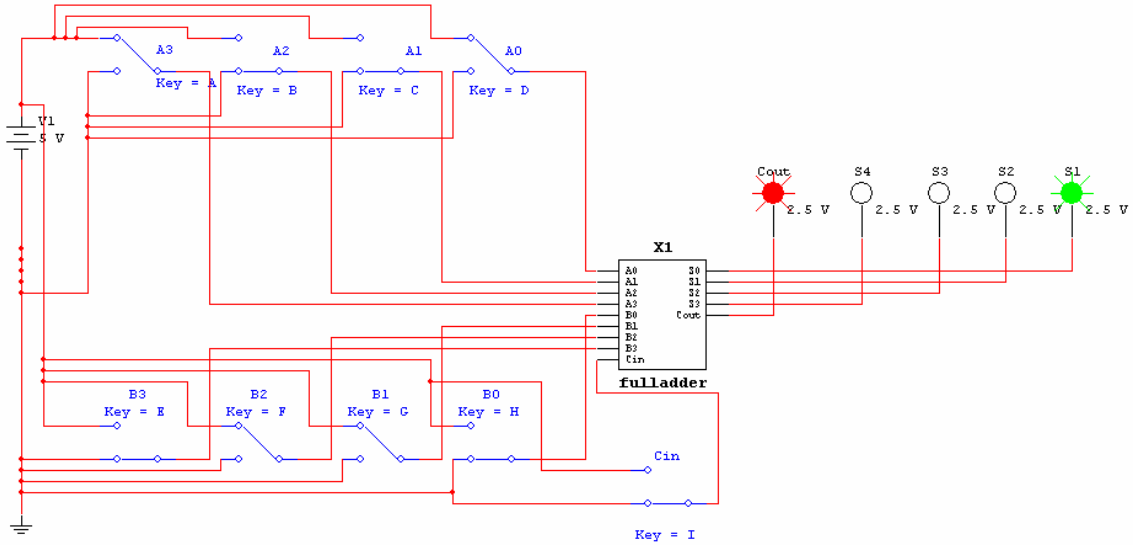
Δ) Η βαθμίδα που δημιουργήσαμε μπορεί να χρησιμοποιηθεί όσες φορές θέλουμε (copy-paste) για να παράγει τη συγκεκριμένη λογική. Έτσι, στο σχ. 3.5 χρησιμοποιούμε 4 φορές τον πλήρη αθροιστή δύο bits, που υλοποιήσαμε πριν, για να δημιουργήσουμε σε ένα νέο ιεραρχικό αρχείο (**Fulladder.ms7** – 9 είσοδοι, 5 έξοδοι) έναν παράλληλο αθροιστή δύο αριθμών των τεσσάρων bits ο καθένας. Η διαδικασία δημιουργίας του ιεραρχικού αυτού αρχείου είναι παρόμοια με αυτήν του πλήρους αθροιστή. Ο τελικός αθροιστής εμφανίζεται πλέον ως ιεραρχικό block και μπορούμε να τον εισάγουμε σε κάθε σχέδιο ως αρχείο βιβλιοθήκης (σχ. 3.6).



.

Σχήμα 3.5 *Πλήρης αθροιστής δύο αριθμών των τεσσάρων bits*

Κατόπιν δημιουργούμε την εξής εφαρμογή μέσα στο κύκλωμα add_sub.ms7 που κατασκευάσαμε στην αρχή:



Σχήμα 3.6 *Πλήρες κύκλωμα της εφαρμογής*

Ε) Να προσθέσετε τους αριθμούς που φαίνονται στον παρακάτω πίνακα και να συμπληρώσετε την στήλη της εξόδου και του κρατουμένου με τη βοήθεια του κυκλώματος του σχ. 3.6:

ΠΙΝΑΚΑΣ 3.2

A		**B**		**C_0**	**S**		**C_{out}**
Δεκαδική τιμή	**Δυαδική τιμή**	**Δεκαδική τιμή**	**Δυαδική τιμή**		**Δεκαδική τιμή**	**Δυαδική τιμή**	
3		5		0			
9		6		1			
2		1		0			
4		5		0			
14		15		0			

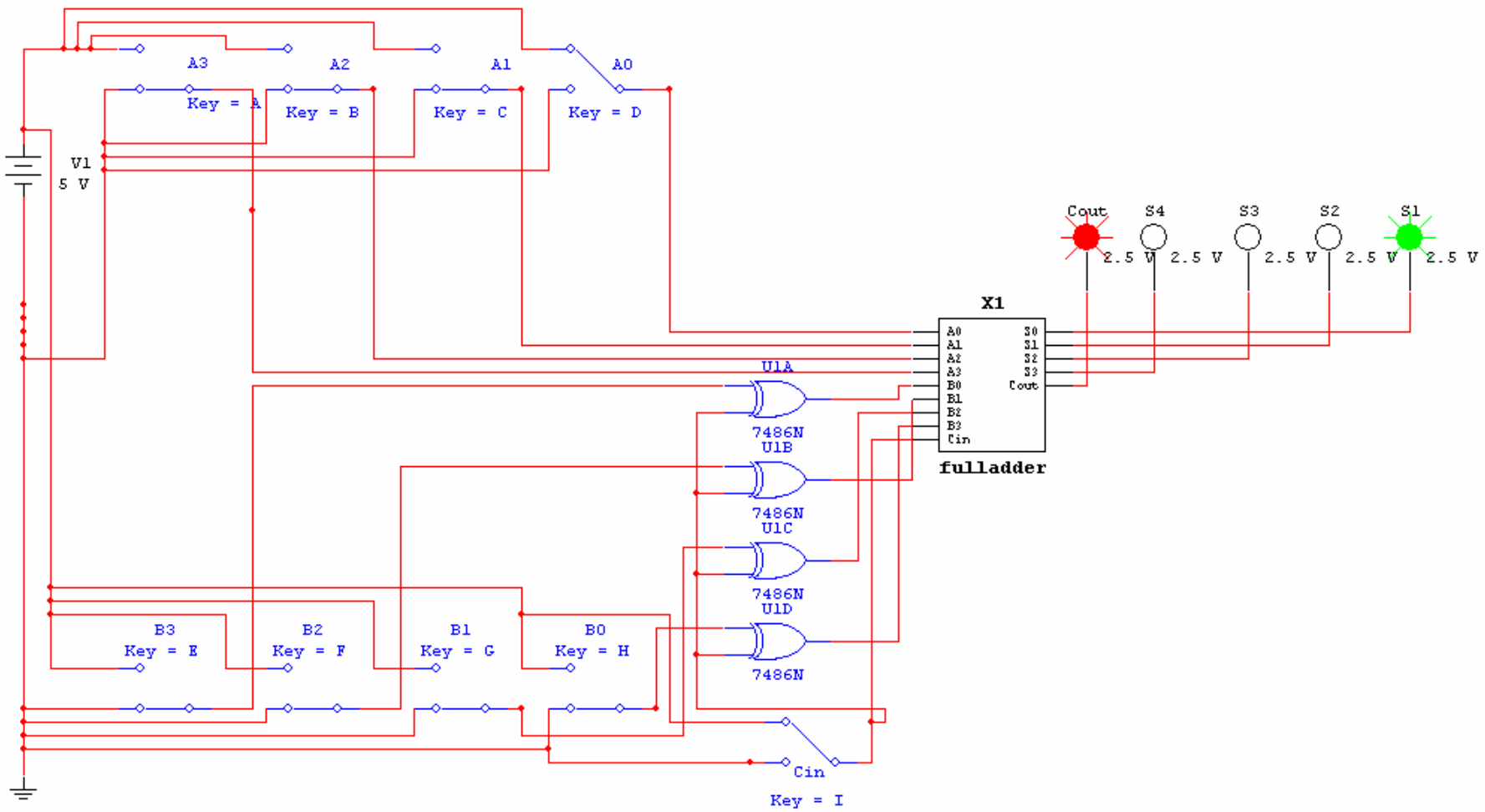
Να επιβεβαιώσετε την ορθότητα των πράξεων που επιτελεί το κύκλωμά σας. Πως εξηγείτε τα φαινομενικά λανθασμένα αποτελέσματα σε ορισμένα ζεύγη τιμών;

ΣΤ) Να μετατρέψετε το κύκλωμά σας σε αφαιρέτη λαμβάνοντας υπόψη το κύκλωμα της αφαίρεσης του σχ. 3.3. Το τελικό κύκλωμα φαίνεται στο σχ. 3.7

Να επιβεβαιώσετε ότι το παρακάτω κύκλωμα του αθροιστή 4-bit, με πύλες XOR στις εισόδους B επιτελεί την αφαίρεση ως πρόσθεση, με βάση την προσήμανση του αριθμού με το συμπλήρωμα ως προς 2. Για τον σκοπό αυτόν να υλοποιήσετε στο Multisim το κύκλωμα του σχήματος 4 και να συμπληρώσετε τον πίνακα που ακολουθεί.

ΠΙΝΑΚΑΣ 3.3

A		**B**		**Cin**	**S**		**Cout**
Δεκαδική τιμή	**Δυαδική τιμή**	**Δεκαδική τιμή**	**Δυαδική τιμή**		**Δεκαδική τιμή**	**Δυαδική τιμή**	
4		5		1			
5		4		0			
12		10		1			
4		15		0			
11		4		1			
7		1		0			
13		0		1			
10		0		0			



Σχήμα 3.7 *Κύκλωμα αφαιρέτη με βάση το συμπλήρωμα ως προς δύο.*

3.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

1. Να αναζητήσετε στο Internet τα φύλλα δεδομένων του αθροιστή **74xx283** και να περιγράψετε στην εργασία σας με δικά σας λόγια τους ακροδέκτες και τη λειτουργία του κυκλώματος.

2. Να χρησιμοποιήσετε το κύκλωμα **74HC283N** από τη βιβλιοθήκη CMOS του προσομοιωτή Multisim, προκειμένου να υλοποιήσετε έναν αθροιστή/αφαιρέτη 4bits. Να συμπληρώσετε τον πίνακα 3.3 για το κύκλωμα που θα σχεδιάσετε.

3. Με τη βοήθεια του αθροιστή **74xx283** να υλοποιήσετε έναν αθροιστή δύο αριθμών με εύρος 8 bits ο καθένας.

ΕΡΓΑΣΤΗΡΙΟ 4

ΚΥΚΛΩΜΑΤΑ ΣΥΓΚΡΙΤΩΝ

4.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

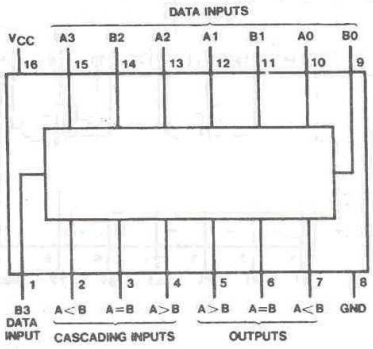
Η σύγκριση δύο δυαδικών λέξεων είναι μια πράξη που χρησιμοποιείται σε όλα τα συστήματα ηλεκτρονικών υπολογιστών, καθώς και στα περισσότερα σύνθετα ψηφιακά συστήματα. Ένα κύκλωμα που συγκρίνει δύο δυαδικές λέξεις και αναφέρει αν είναι ίσες ή αν η μία είναι μεγαλύτερη από την άλλη, ονομάζεται συγκριτής.

Ένας συγκριτής μπορεί να συγκρίνει απλώς 2 bits. Το απλούστερο κύκλωμα σύγκρισης δύο bits είναι η πύλη OR ή η πύλη NOR. Ένα πιο περίπλοκο κύκλωμα που μας αναφέρει αν τα συγκρινόμενα bits είναι ίσα ή αν το ένα είναι μεγαλύτερο από το άλλο, φαίνεται στο κύκλωμα του σχ. 4.3. Με κατάλληλη επανάληψη της βασικής βαθμίδας και σύνδεση των βαθμίδων σε συνδεσμολογία καταρράκτη είναι δυνατό να δημιουργήσουμε συγκριτές λέξεων πολλών bits.

Οι εφαρμογές των συγκριτών είναι αρκετά διαδεδομένες, με αποτέλεσμα να έχουν αναπτυχθεί και να κυκλοφορούν στην αγορά πολλοί συγκριτές μέσης κλίμακας ολοκλήρωσης. Ένα τέτοιο κύκλωμα είναι το TTL 74LS85 και το αντίστοιχο ισοδύναμο CMOS 4585. Τα κυκλώματα αυτά έχουν εισόδους 4-bits A3 A2 A1 A0 και B3 B2 B1 B0 για τη σύγκριση δύο λέξεων και εξόδους O_AGTB (out A Geater Than B), O_ALTB (Out A Less Than B) και O_AEQB (Out A Equal to B). Ανάλογα με το αποτέλεσμα της σύγκρισης ενεργοποιείται η αντίστοιχη έξοδος.

Connection Diagram

Dual-In-Line Package



Function Table

Comparing Inputs				Cascading Inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	A = B	A > B	A < B	A = B
A3 > B3	X	X	X	X	X	X	H	L	L
A3 < B3	X	X	X	X	X	X	L	H	L
A3 = B3	A2 > B2	X	X	X	X	X	H	L	L
A3 = B3	A2 < B2	X	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	X	X	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	H	L	L	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	L	H	H	L

H = High Level, L = Low Level, X = Don't Care

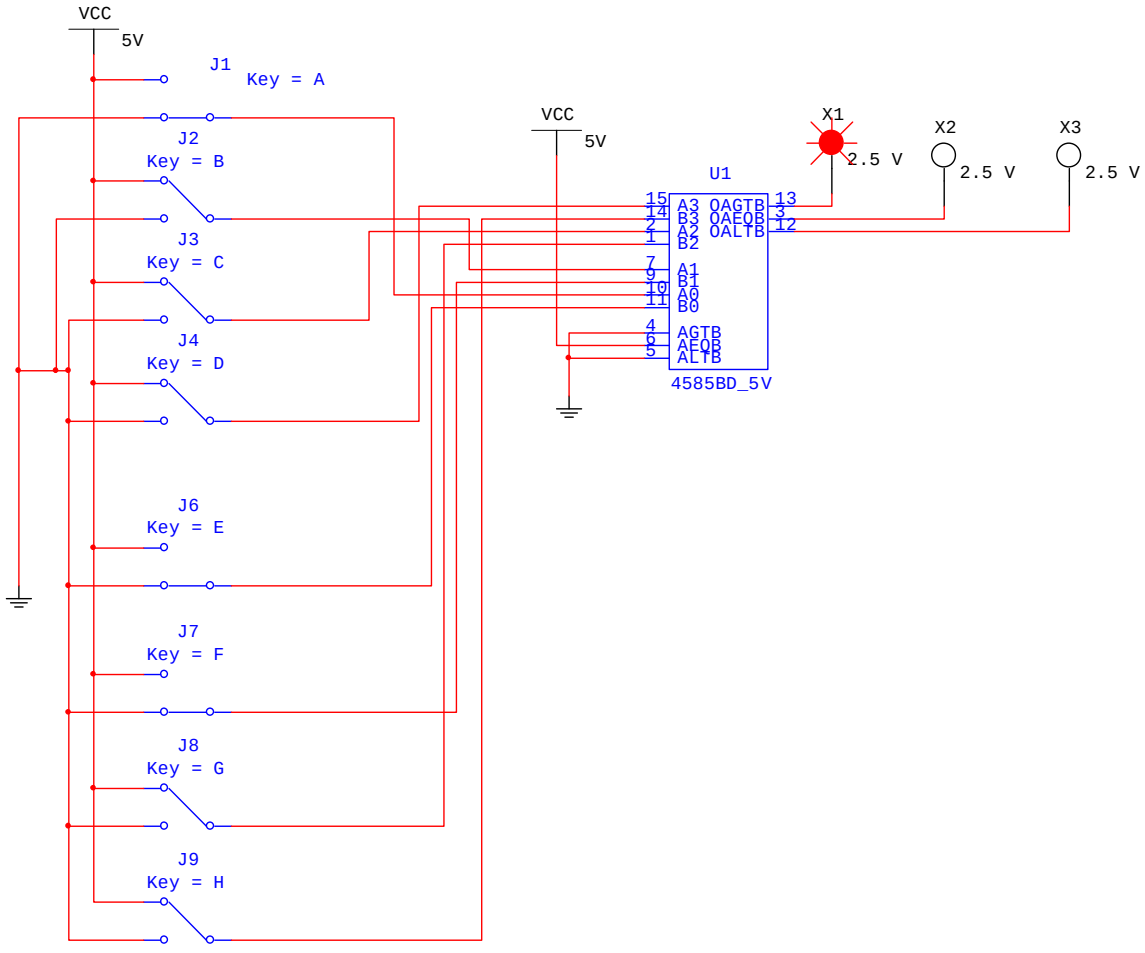
Σχήμα 4.1 *Διάγραμμα ακροδεκτών και πίνακας αληθείας του ολοκληρωμένου κυκλώματος 74LS85*

Επίσης, το ολοκληρωμένο κύκλωμα ’85 έχει αλυσιδωτές εισόδους (AGTB, AEQB, ALTB), με σκοπό τη δημιουργία συγκριτών για περισσότερα από τέσσερα bits. Να κοιτάξετε στη σελίδα 500 του εγχειριδίου σας και να μελετήσετε το κύκλωμα του συγκριτή 12 bits που χρησιμοποιεί το 74LS85.

Στο σχήμα 4.1 δίνεται το διάγραμμα ακροδεκτών και ο πίνακας αληθείας του ολοκληρωμένου κυκλώματος 74LS85, σύμφωνα με τα φύλλα δεδομένων του κατασκευαστή.

4.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

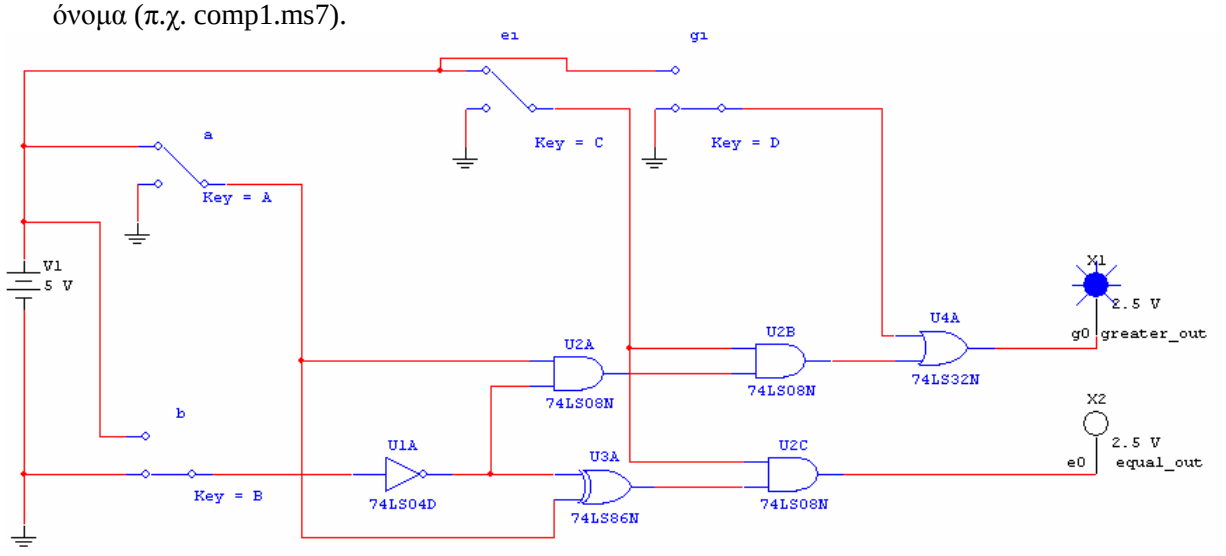
1. Να σχεδιάσετε το παραπάνω κύκλωμα συγκριτή 4-bits που στηρίζεται στο ολοκληρωμένο CMOS 4585, που είναι ισοδύναμο με το TTL 74LS85. Προσέξτε τη σωστή σύνδεση των εισόδων στους ακροδέκτες του ολοκληρωμένου.



Σχήμα 4.2 *Κύκλωμα συγκριτή με το MSI ολοκληρωμένο CMOS 4585 (ισοδύναμο με το TTL 74LS85)*

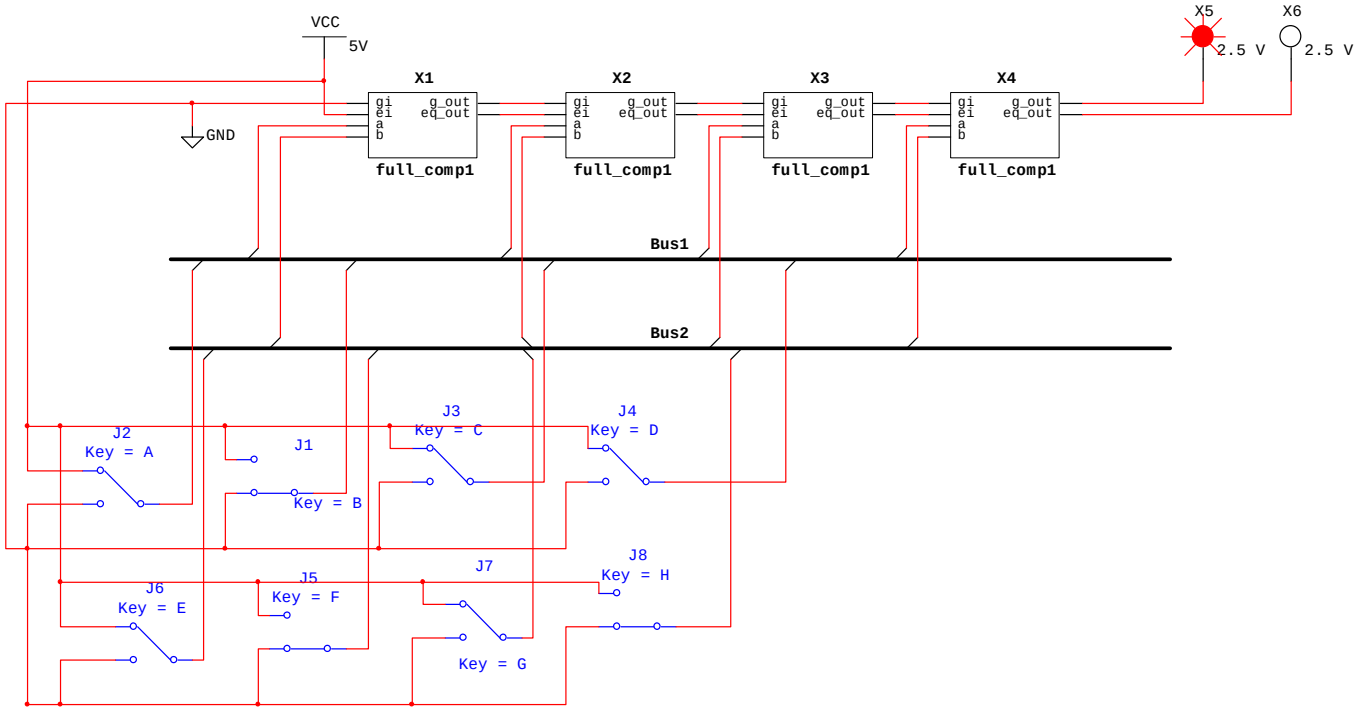
2. Να σχεδιάσετε στο περιβάλλον MultiSim 7.0 το παρακάτω κύκλωμα συγκριτή δύο αριθμών (a, b) του ενός bit, ο οποίος επιπλέον μπορεί να συνδεθεί με άλλους παρόμοιους σε συνδεσμολογία καταρράκτη ώστε να δημιουργηθεί ένας συγκριτής περισσότερων bits. Ο συγκριτής αυτός συγκρίνει τα bits a και b και παράγει εξόδους go και eo. Το go γίνεται 1 όταν a>b, ενώ το eo=1 όταν a=b. Όταν a<b τότε go=eo=0. Οι είσοδοι gi και ei συνδέονται με συγκριτές προηγούμενης τάξης όταν έχουμε συγκριτή πολλών bits σε συνδεσμολογία καταρράκτη. Όταν χρησιμοποιούμε μόνον έναν συγκριτή θα πρέπει να είναι ei=1 και gi=0 ώστε

να μην επηρεάζουν την έξοδο του συγκριτή. Να αποθηκεύσετε το σχέδιό σας με κατάλληλο όνομα (π.χ. comp1.ms7).



Σχήμα 4.3 *Κύκλωμα συγκριτή δύο bits*

3. Να υλοποιήσετε έναν συγκριτή 2 αριθμών των 4 bits με βάση τον συγκριτή 2 bits που σχεδιάσατε στην παράγραφο 2. Κάθε block στο διάγραμμα παριστάνει έναν τέτοιο συγκριτή 2-bits. Να εφαρμόσετε τα βήματα ιεραρχικής σχεδίασης με βαθμίδες, που μάθατε στο εργαστήριο 3. Να υλοποιήσετε το κύκλωμα στο Multisim και να περιγράψετε τη λειτουργία του με τη βοήθεια ενός πίνακα αληθείας. Βοηθητικά, το κύκλωμα που προκύπτει στο Multisim φαίνεται στο σχ. 4.4. Για τη δημιουργία των διαδρόμων (Bus1 και Bus2) συμβουλευτείτε τους διδάσκοντες. Σημειώστε ότι ο 1^{ος} συγκριτής X1 συγκρίνει τα πιο σημαντικά bit των αριθμών, ο X2 τα επόμενα bits κ.ο.κ.



Σχήμα 4.4 *Κύκλωμα συγκριτή δύο λέξεων των 4-bits ως ιεραρχικό διάγραμμα βαθμίδων του Multisim 7.0*

4.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

1. Να υλοποιήσετε έναν συγκριτή δύο λέξεων **8-bits**, χρησιμοποιώντας το ολοκληρωμένο κύκλωμα **4585**. Λάβετε υπόψιν ότι όταν συνδέονται συγκριτές 4585 σε συνδεσμολογία καταρράκτη για σύγκριση αριθμών μεγαλύτερων των 4 bits, ο πρώτος συγκριτής συγκρίνει τα 4 λιγότερο σημαντικά bit των αριθμών, ο επόμενος τα επόμενα 4 bits κ.ο.κ.

2. Ένας αισθητήρας θερμοκρασίας μετρά τη θερμοκρασία από **0 έως 63 βαθμούς Κελσίου** με ανάλυση τεσσάρων βαθμών Κελσίου και παράγει ένα ψηφιακό σήμα 4-bits. Ένας δεύτερος αισθητήρας μετρά την πίεση από **1 έως 16 ατμόσφαιρες** με ανάλυση μιας ατμόσφαιρας και παράγει πάλι ψηφιακό σήμα 4-bits. Να σχεδιάσετε ένα ψηφιακό κύκλωμα που να δέχεται ως είσοδο τα δύο σήματα εξόδου των αισθητηρίων και να θέτει σε λειτουργία έναν συναγερμό όταν η θερμοκρασία και η πίεση ξεπεράσουν ταυτόχρονα τους **32 βαθμούς Κελσίου** και τις **10 ατμόσφαιρες**, αντίστοιχα. Να υλοποιήσετε το κύκλωμα στο Multisim προσομοιώνοντας τους αισθητήρες με διακόπτες (τέσσερις διακόπτες ανά αισθητήρα).

ΕΡΓΑΣΤΗΡΙΟ 5

ΑΠΑΡΙΘΜΗΤΕΣ

5.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

5.1.1 Flip-Flops

Το flip=flop είναι ένα σύγχρονο ακολουθιακό κύκλωμα, οι έξοδοι του οποίου ανταποκρίνονται στις εισόδους του όταν εφαρμόζονται παλμοί ρολογιού. Οι παλμοί ρολογιού (clock pulses) εφαρμόζονται σε μια είσοδο του flip-flop, που ονομάζεται είσοδος ρολογιού (CP). Οι πλέον συχνά χρησιμοποιούμενοι τύποι flip-flops είναι οι ακόλουθοι:

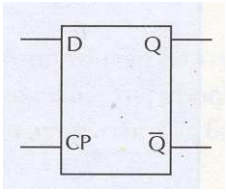
- R-S flip-flop
- D flip-flop
- J-K flip-flop
- T flip-flop

5.1.2 D Flip-Flop

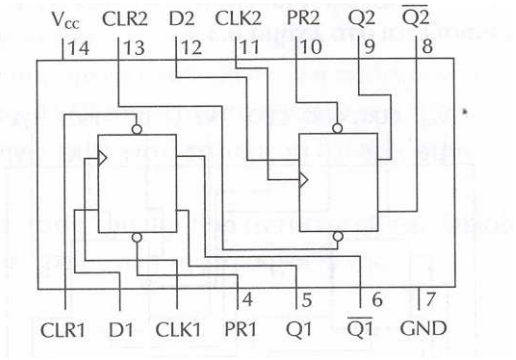
Η λειτουργία του D Flip-Flop περιγράφεται παρακάτω:

- Αν ο παλμός ρολογιού είναι CP=0, τότε το flip-flop δεν μπορεί να αλλάξει κατάσταση, ανεξάρτητα από την τιμή της εισόδου D.
- Αν ο παλμός του ρολογιού είναι CP=1, τότε η είσοδος D περνάει στην έξοδο. Η έξοδος παραμένει αμετάβλητη, διατηρώντας μνήμη της προηγούμενης κατάστασης. Η έξοδος ανανεώνεται στον επόμενο θετικό παλμό του ρολογιού.

Q(n)	D	Q(n+1)
0	0	0
0	1	1
1	0	0
1	1	1



Σχήμα 5.1 *Χαρακτηριστικός πίνακας του D Flip-Flop και σχηματικό σύμβολο*



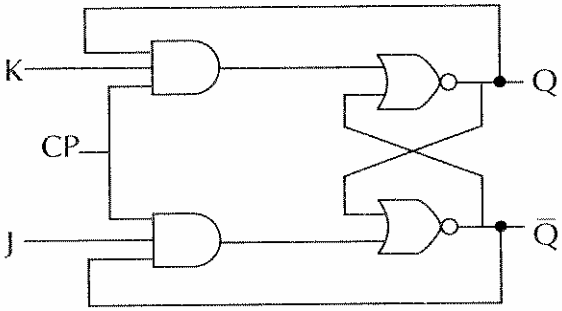
Σχήμα 5.2 *Σχηματικό διάγραμμα και ακροδέκτες του ολοκληρωμένου κυκλώματος 7474*

5.1.3 J-K Flip-Flop

Η υλοποίηση του J-K flip-flop φαίνεται στο παρακάτω σχήμα 5.3. Η λειτουργία του περιγράφεται ως εξής:

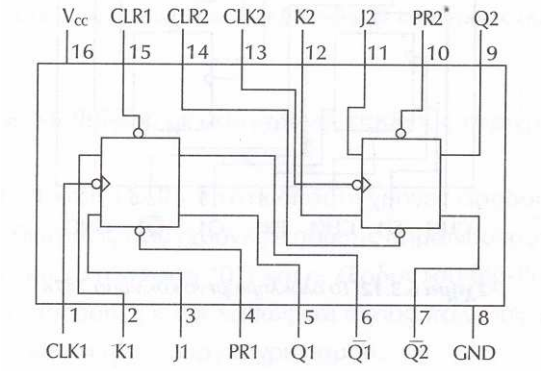
- Όταν J=0 και K=0, τότε η επόμενη κατάσταση είναι ίδια με την προηγούμενη.
- Όταν J=0 και K=1, τότε η επόμενη κατάσταση είναι Q=0.
- Όταν J=1 και K=0, τότε η επόμενη κατάσταση είναι Q=1.
- Όταν J=1 και K=1, τότε η κατάσταση του flip-flop αντιστρέφεται (toggle).

Ο χαρακτηριστικός πίνακας του J-K flip-flop παρουσιάζεται παρακάτω.



Q(n)	J	K	Q(n+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Σχήμα 5.3 *Κύκλωμα του JK FF με πύλες και χαρακτηριστικός πίνακας*

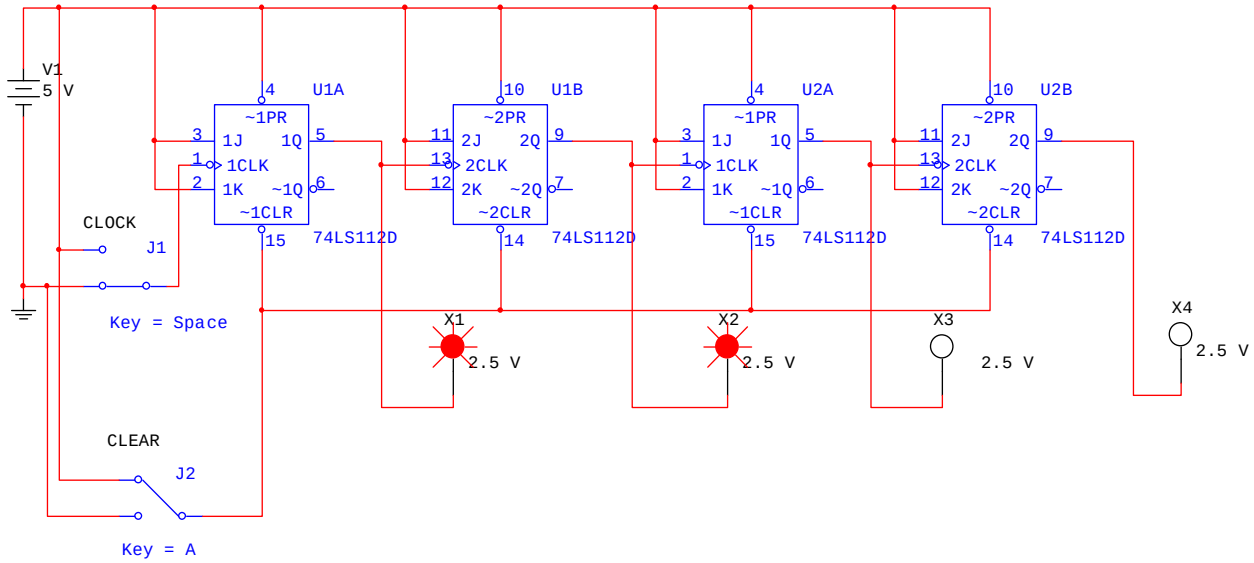


Σχήμα 5.4 *Σχηματικό διάγραμμα του ολοκληρωμένου 74LS112*

5.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

5.2.1 Κύκλωμα ασύγχρονου δυαδικού απαριθμητή με JK Flip-Flops.

Να δημιουργήσετε στο περιβάλλον Multisim 7.0 το παρακάτω κύκλωμα του ασύγχρονου δυαδικού απαριθμητή, με το J-K flip-flop 74LS112. Παρατηρείστε την αλλαγή των εξόδων για κάθε παλμό ρολογιού (CLOCK).



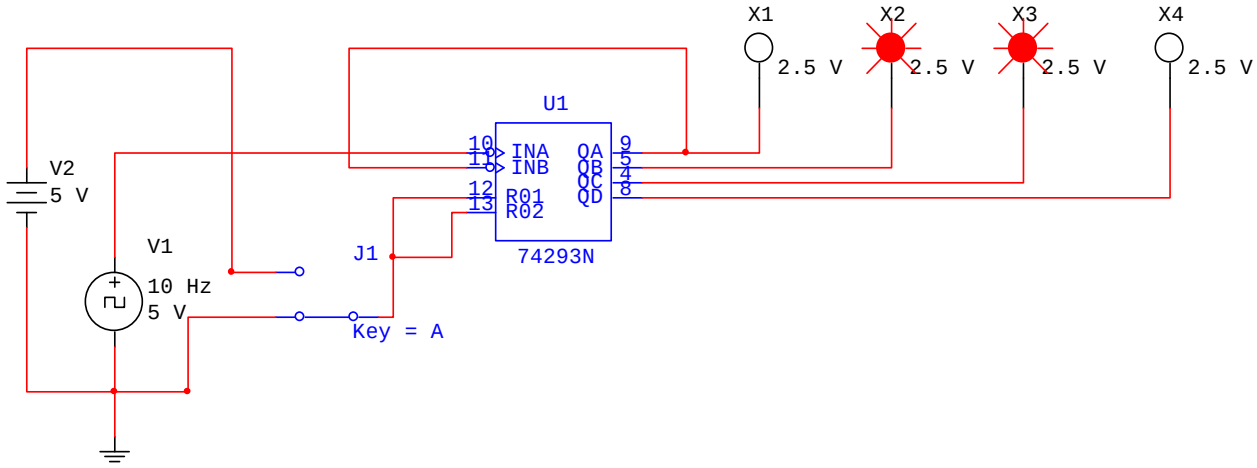
Σχήμα 5.5 *Ασύγχρονος δυαδικός απαριθμητής με J-K flip-flops*

Προσπαθήστε να εξηγήσετε τη λειτουργία του κυκλώματος και συμπληρώστε τον παρακάτω πίνακα καταγράφοντας την ακολουθία μέτρησης του απαριθμητή.

Παλμός ρολογιού	**Έξοδοι flip-flops**			
	Q4	**Q3**	**Q2**	**Q1**
Αρχική κατάσταση	0	0	0	0
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				

5.2.2 Κύκλωμα δυαδικού απαριθμητή (ολοκληρωμένο 74LS293)

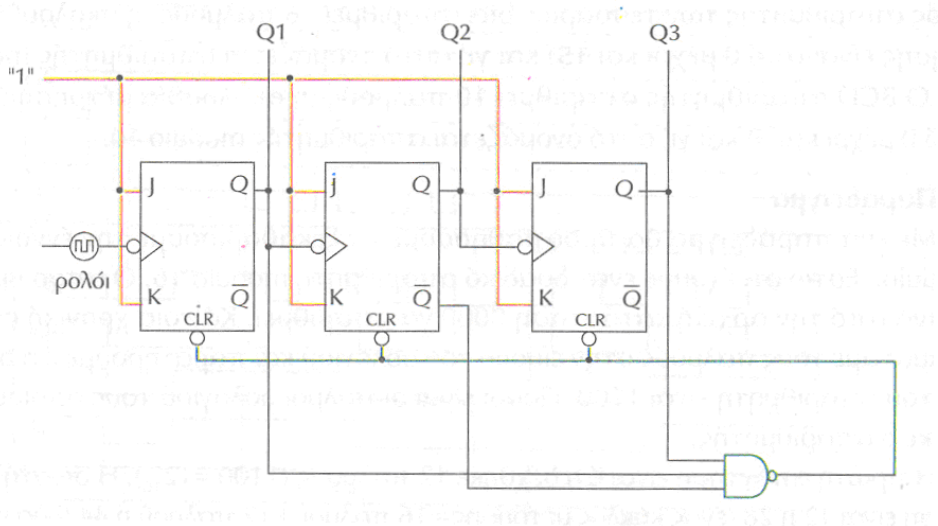
Να σχεδιάσετε στο περιβάλλον Multisim7.0 το παρακάτω κύκλωμα του ασύγχρονου δυαδικού απαριθμητή και να επιβεβαιώσετε τη λειτουργία του.



Σχήμα 5.6 *Δυαδικός απαριθμητής με το ολοκληρωμένο κύκλωμα 74LS293*

5.2.3 Αλλαγή του modulo ενός απαριθμητή.

Το παρακάτω κύκλωμα δείχνει πως μπορούμε να αλλάξουμε τον μέγιστο αριθμό παλμών που απαριθμεί ένας απαριθμητής, πριν μηδενιστεί. Χωρίς την πύλη NAND, ο απαριθμητής μετρά από το 000 μέχρι το 111. Όταν όμως η πύλη NAND λάβει στην είσοδο 111, δηλαδή όταν οι έξοδοι Q των FF γίνουν 101 (προσέξτε πως συνδέονται οι έξοδοι στην NAND), τότε η έξοδος της NAND γίνεται μηδέν και μηδενίζει τον απαριθμητή επιδρώντας στις εισόδους CLEAR των FF. Άρα αυτός ο απαριθμητής μετρά μέχρι το 101 (δηλαδή το 5) μέχρι να μηδενιστεί ξανά. Λέγεται λοιπόν απαριθμητής modulo 5.



Σχήμα 5.7 *Αλλαγή του modulo δυαδικού απαριθμητή (up-counter) από 8 σε 5*

Με βάση την παραπάνω ιδέα να αλλάξετε το κύκλωμα του σχ. 5.5 (παράγραφος 5.2.1), ώστε να μετατρέψετε το modulo του απαριθμητή από 16 σε 12.

5.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

1. Να υλοποιήσετε ένα ψηφιακό κύκλωμα που να επιτελεί τα εξής:
    - Να καταμετρά συνολικά 12 καταστάσεις.
    - Όταν καταμετρηθούν **6 παλμοί** και μέχρι τον **12^{ο} παλμό**, να θέτει την έξοδο στο **λογικό 1**. Σε κάθε άλλη περίπτωση να την θέτει στο **λογικό 0**.

ΕΡΓΑΣΤΗΡΙΟ 6

ΚΑΤΑΧΩΡΗΤΕΣ, ΜΝΗΜΕΣ

6.1. ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

6.1.1 Γενικός καταχωρητής 74LS194

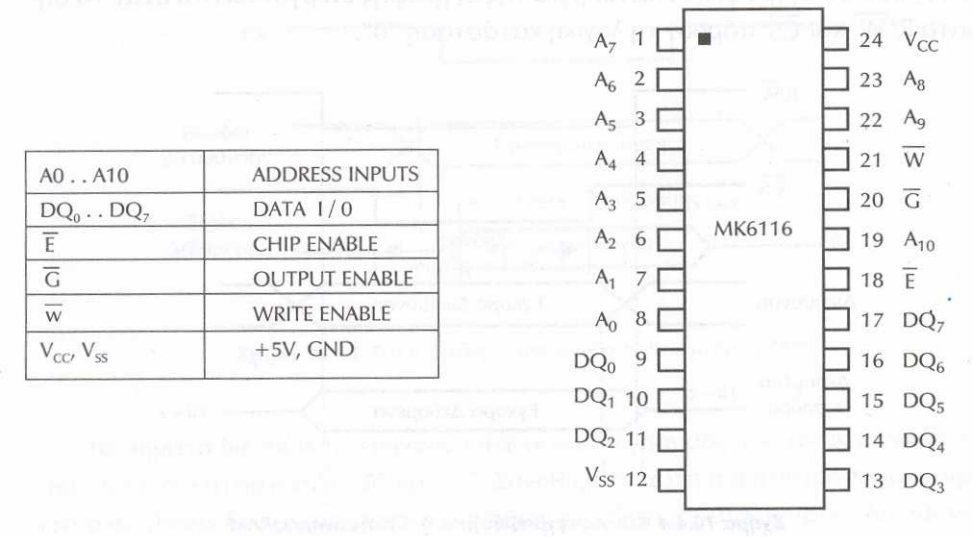
Το ολοκληρωμένο κύκλωμα 74LS194 είναι ένας γενικός καταχωρητής 4-bits, που επιτελεί παράλληλη φόρτωση και σειριακή ολίσθηση, προς τα δεξιά ή προς τα αριστερά, ανάλογα με τις τιμές που παίρνουν οι είσοδοι S0 και S1, σύμφωνα με τον παρακάτω πίνακα:

Mode (Λειτουργία)	S1	S2	QA(n+1)	QB(n+1)	QC(n+1)	QD(n+1)
Hold (Διατήρηση)	0	0	QA(n)	QB(n)	QC(n)	QD(n)
Shift Right (Ολίσθηση δεξιά)	0	1	SRSI	QA(n)	QB(n)	QC(n)
Shift Left (Ολίσθηση αριστερά)	1	0	QB(n)	QC(n)	QD(n)	SLSI
Load (Παράλληλη φόρτωση)	1	1	A	B	C	D

Σχήμα 6.1 *Πίνακας λειτουργίας του ολοκληρωμένου καταχωρητή γενικού σκοπού 74LS194*

6.1.2 Στατική μνήμη RAM σε ολοκληρωμένο κύκλωμα

Η στατική μνήμη SRAM MK6116 έχει χωρητικότητα 2048 λέξεων των 8 bits. Οι ακροδέκτες του κυκλώματος φαίνονται στο παρακάτω Σχήμα 6.2. Η σημασία των ακροδεκτών φαίνεται στον πίνακα. Η διαδικασία ανάγνωσης και εγγραφής περιγράφεται στο εργαστηριακό μέρος.

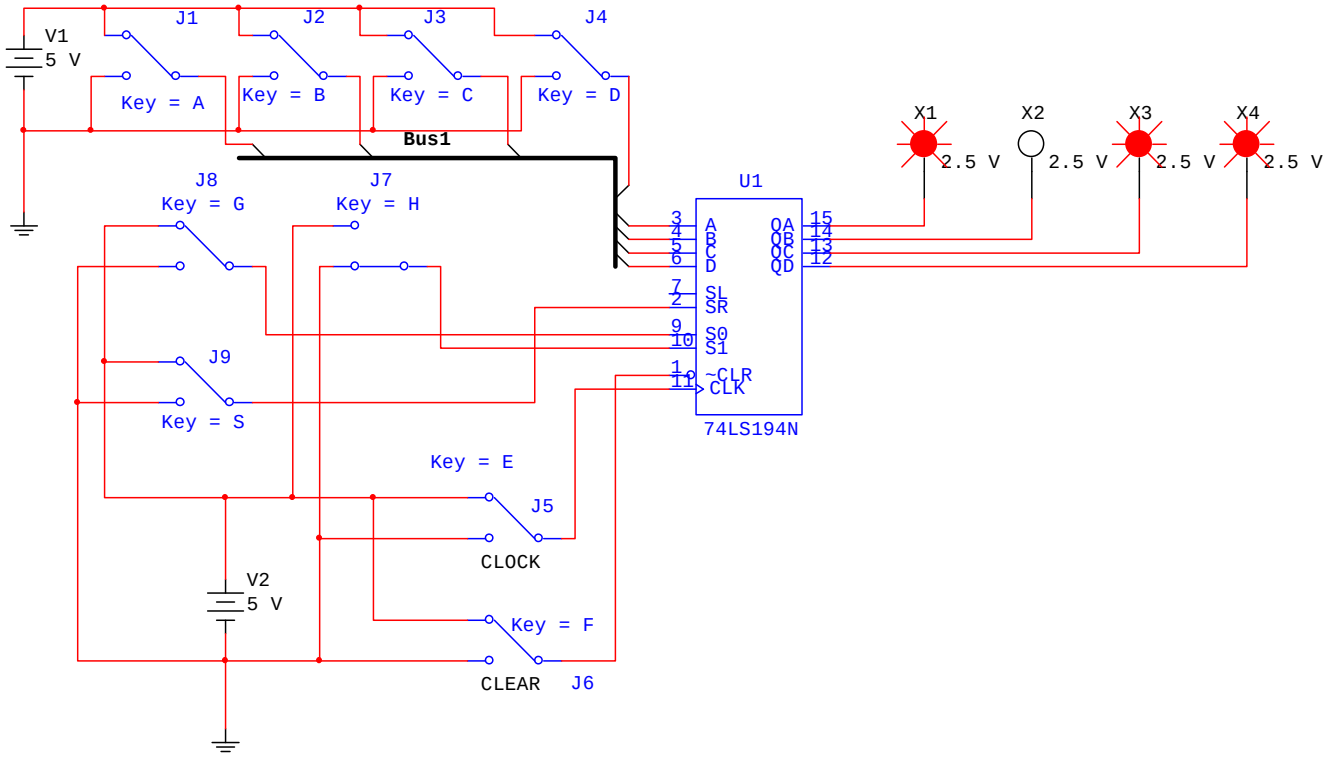


Σχήμα 6.2 *Διάγραμμα και λειτουργία των ακροδεκτών της μνήμης MK6116*

6.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

6.2.1 Ο καταχωρητής 74LS194

Να υλοποιήσετε το παρακάτω κύκλωμα του καταχωρητή στο Multisim 7.0 και να ελέγξετε τη λειτουργία της παράλληλης φόρτωσης και της σειριακής ολίσθησης. Στο σχήμα 6.3 η σειριακή είσοδος δίνεται από τον ακροδέκτη 2 (SR) και η ολίσθηση γίνεται προς τα δεξιά.



Σχήμα 6.3 *Γενικός καταχωρητής 74LS194*

6.2.2 Στατική μνήμη RAM

Να σχεδιάσετε το παρακάτω κύκλωμα της στατικής μνήμης RAM 2K x 8 bits και να επιβεβαιώσετε την λειτουργία του ως εξής:

1. Για εγγραφή δεδομένων πρέπει:
   - Στο δίαυλο δεδομένων να υπάρχουν έγκυρα δεδομένα. Αν και ο διάδρομος είναι προφανώς 8 bits, χρησιμοποιήστε μόνον τους διακόπτες A, B, C για να μεταβάλετε τα δεδομένα εισόδου, ώστε να μην έχετε πολλούς διακόπτες στο κύκλωμα. Τα υπόλοιπα bits ας μείνουν σε λογικό 0.
   - Να ορίσετε τη διεύθυνση μνήμης. Για απλότητα, να χειριστείτε μόνον τις τέσσερις πρώτες διευθύνσεις (από τις 2K συνολικά διευθύνσεις που διαθέτει το κύκλωμα), με τη βοήθεια δύο διακοπτών (D, E).
   - Ο ακροδέκτης CS να είναι σε λογικό 1
   - Να δώσετε στον ακροδέκτη WE (διακόπτης F) θετικό μέτωπο παλμού.

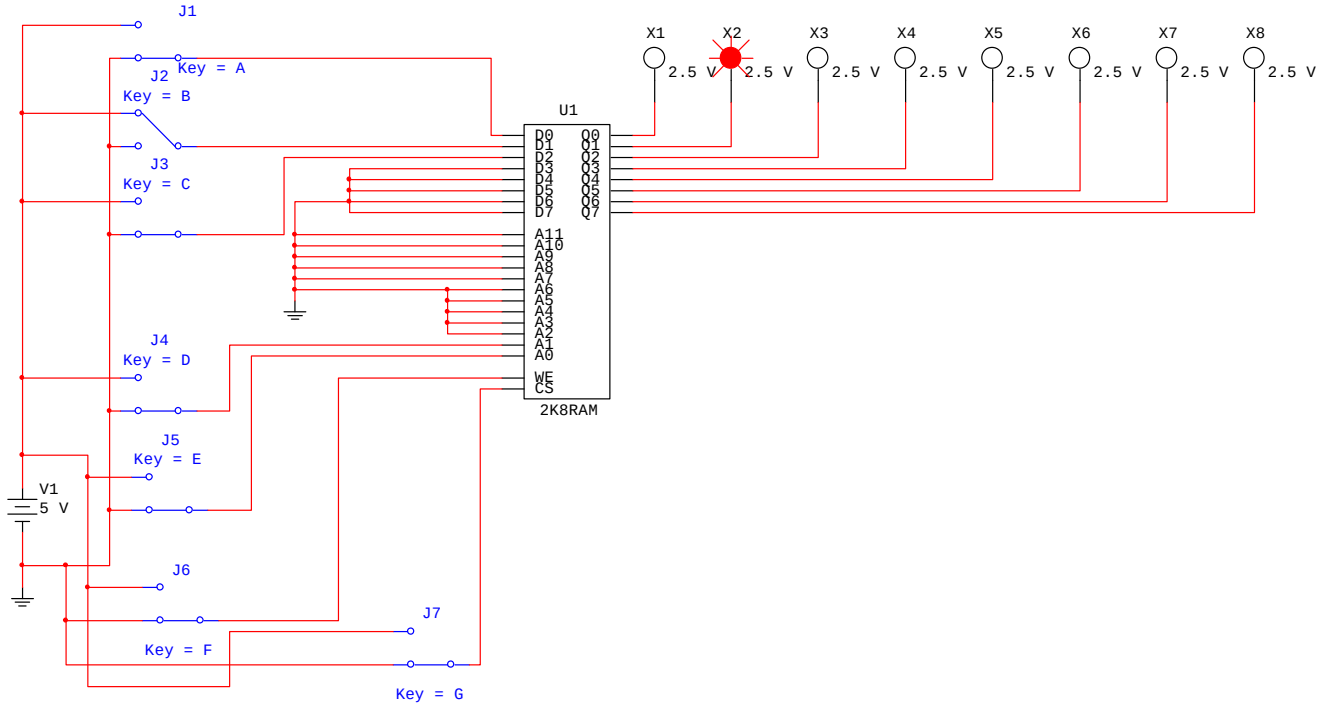
Εγγράψτε τα εξής δεδομένα στις τέσσερις πρώτες διευθύνσεις:

A1	**A0**	**Δεδομένα εισόδου**
0	0	00000001
0	1	00000010
1	0	00000100
1	1	00000101

2. Για ανάγνωση δεδομένων πρέπει:

- Το CS να είναι σε λογικό 1 και το WE σε λογικό 0 (διαδικασία Read).
- Να σαρώστε με τη βοήθεια των διακοπτών D, E τις τέσσερις πρώτες διευθύνσεις μνήμης (A1A0 = 00, 01, 10, 11).

Καταγράψτε τα αποτελέσματα και διαπιστώστε ότι συμφωνούν με τα δεδομένα που εγγράψατε κατά το βήμα της εγγραφής.



Σχήμα 6.4 *Κύκλωμα στατικής μνήμης RAM (ισοδύναμο με τη μνήμη MK6116 της SGS-Thomson)*

6.3 ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ ΠΡΟΣ ΠΑΡΑΔΟΣΗ

1. Ένα υπολογιστικό σύστημα διαθέτει **Data Bus 8 bits**, **Address Bus 13 bits** και διαθέτει συνολική μνήμη **RAM 8 KByte**, που υλοποιείται με **4** ολοκληρωμένα κυκλώματα της στατικής μνήμης της παραγράφου 6.2.2. Να σχεδιάσετε το κύκλωμα αποκωδικοποίησης διευθύνσεων και να υπολογίσετε ποιες διευθύνσεις μνήμης αντιστοιχούν σε καθ' ένα από τα 4 ολοκληρωμένα κυκλώματα μνήμης.

Υπόδειξη: χρησιμοποιείστε έναν αποκωδικοποιητή 2 προς 4 που σαν είσοδο θα έχει 2 από τα bit του address bus (ποια?) και οι έξοδοί του θα οδηγούνται στα σήματα CS των ολοκληρωμένων κυκλωμάτων μνήμης.

ΜΕΡΟΣ Β

ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΣΕ ΚΥΚΛΩΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗΣ ΛΟΓΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ 7

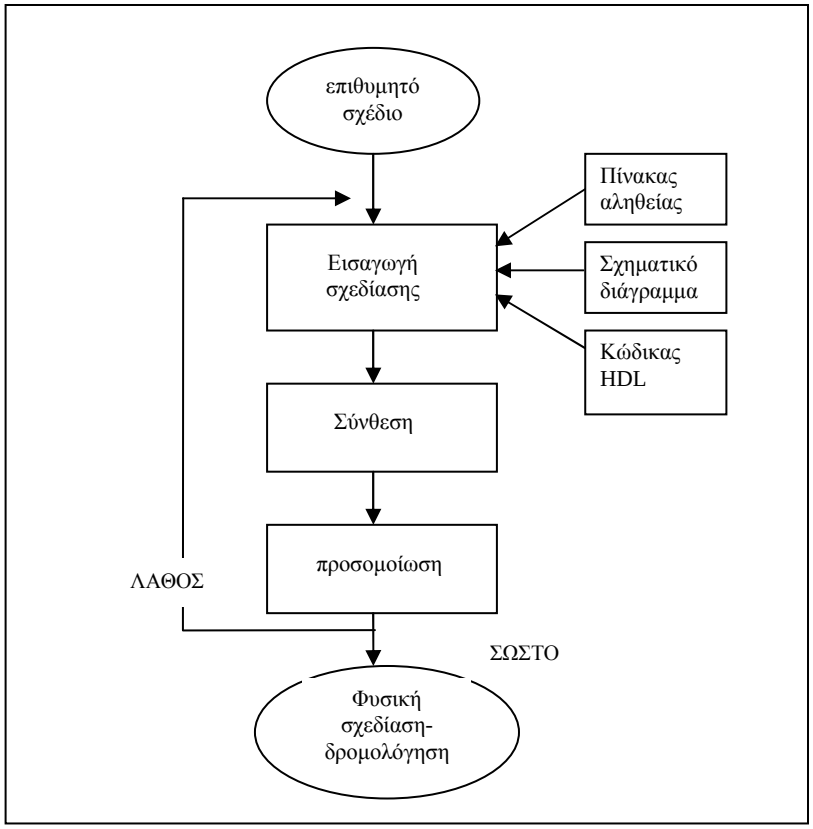
ΕΡΓΑΛΕΙΑ ΣΧΕΔΙΑΣΗΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΜΕ ΣΤΟΧΟ ΔΙΑΜΟΡΦΟΥΜΕΝΑ ΚΥΚΛΩΜΑΤΑ

7.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

7.1.1 Εισαγωγή

Η ανάγκη για σχεδίαση σύνθετων λογικών κυκλωμάτων κάνει αναγκαία τη χρήση ειδικών εργαλείων λογισμικού για τη σχεδίαση και προσομοίωση (CAD tools). Ο σκοπός των εργαλείων αυτών είναι η λεπτομερής προσομοίωση του κυκλώματος, καθώς και η δημιουργία των απαραίτητων αρχείων για τον προγραμματισμό κατάλληλου υλικού. Εδώ, αναφερόμαστε σε προγραμματιζόμενο υλικό, όπως είναι τα ολοκληρωμένα κυκλώματα CPLDs και FPGAs. Για τη δομή και τη λειτουργία τέτοιων κυκλωμάτων, ο σπουδαστής οφείλει να ανατρέξει στα ειδικά κεφάλαια της θεωρίας.

Τα εργαλεία CAD παρέχουν εναλλακτικούς τρόπους εισαγωγής της περιγραφής του κυκλώματος. Ένας τρόπος εισαγωγής του κυκλώματος είναι η σχεδίασή του με τη βοήθεια βαθμίδων (blocks) που λαμβάνονται από βιβλιοθήκες, όπως μάθαμε να κάνουμε στο σχεδιαστικό περιβάλλον Multisim. Ένας άλλος πολύ διαδεδομένος τρόπος εισαγωγής είναι τα αρχεία γλώσσας περιγραφής υλικού (Hardware Discription Language–HDL), όπως είναι για παράδειγμα η γλώσσα VHDL. Μια σύντομη εισαγωγή στη VHDL ο σπουδαστής μπορεί να βρει στο παράρτημα. Απλά παραδείγματα χρήσης της γλώσσας θα δοθούν στα εργαστήρια που ακολουθούν.



Σχήμα 7.1 *Ροή Εργασιών Σε Ένα Σύστημα Σχεδίασης*

Τα βασικά βήματα σχεδίασης με τη χρήση εργαλείων CAD φαίνονται στο σχ. 7.1. Η σειρά εργασιών που δείχνει το σχήμα είναι τυπική για όλα τα σχετικά εργαλεία σχεδίασης. Εκτός από την εισαγωγή του κυκλώματος, που αναφέρθηκε παραπάνω, τα επόμενα βήματα περιλαμβάνουν τη σύνθεση (synthesis), την βελτιστοποίηση και την φυσική σχεδίαση ή δρομολόγηση (place and route).

Η σύνθεση είναι μια αυτόματη διαδικασία που επιτελεί το σχεδιαστικό λογισμικό, ώστε να επανασχεδιάσει το κύκλωμά μας σύμφωνα με τους κανόνες της τεχνολογίας για την οποία προορίζεται το σχέδιό μας. Έτσι, άλλο θα είναι το αποτέλεσμα της σύνθεσης αν πρόκειται να προγραμματίσουμε ένα κύκλωμα CPLD και άλλο αν πρόκειται να προγραμματίσουμε ένα FPGA. Ο λόγος είναι ότι τα κυκλώματα αυτά περιέχουν το καθένα τις δικές τους διαφορετικές βαθμίδες για τη δημιουργία λογικών συναρτήσεων. Ένα CPLD θα πρέπει να υλοποιήσει τις λογικές συναρτήσεις με βάση λογικούς πίνακες πυλών (Logic Arrays), που κατασκευάζονται με επίπεδα πυλών AND και OR, ενώ ένα FPGA χρησιμοποιεί τους λεγόμενους πίνακες αναφοράς (Look-up Tables).

Η προσομοίωση ακολουθεί την ιδέα που γνωρίσαμε και στο Multisim, είναι όμως πιο λεπτομερής και ακριβής. Υλοποιείται με τη βοήθεια κατάλληλων εισόδων, που ενεργούν στους ακροδέκτες εισόδου του κυκλώματος, οπότε τα παραγόμενα σήματα εξόδου μπορούν να συγκριθούν με τα αναμενόμενα. Τα σήματα εισόδου σχεδιάζονται ως κατάλληλες κυματομορφές, με τη βοήθεια ειδικών εργαλείων σχεδίασης σημάτων. Τα σχετικά αρχεία που δημιουργούνται για την ενεργοποίηση των εισόδων, καλούνται “waveform vectors” ή διανύσματα κυματομορφών.

Αν η προσομοίωση δεν είναι ικανοποιητική, θα χρειαστεί επανασχεδιασμός του κυκλώματος. Αυτό σημαίνεται με το βέλος ανάδρασης που υπάρχει στο παραπάνω σχ. 7.1. Όταν τελικά η προσομοίωση είναι μέσα στα επιθυμητά πλαίσια, τότε προχωρούμε στο επόμενο βήμα, που είναι η δρομολόγηση του κυκλώματος (place and route ή fitting). Σε μεγάλα κυκλώματα η δρομολόγηση είναι πολύωρη διαδικασία. Στο στάδιο αυτό δημιουργούνται οι απαραίτητες συνδέσεις στον πίνακα διασυνδέσεων του κυκλώματος (interconnection matrix), με τις οποίες συνδέονται με βέλτιστο τρόπο μεταξύ τους τα λογικά στοιχεία που επιτελούν τις επιμέρους λειτουργίες του κυκλώματος.

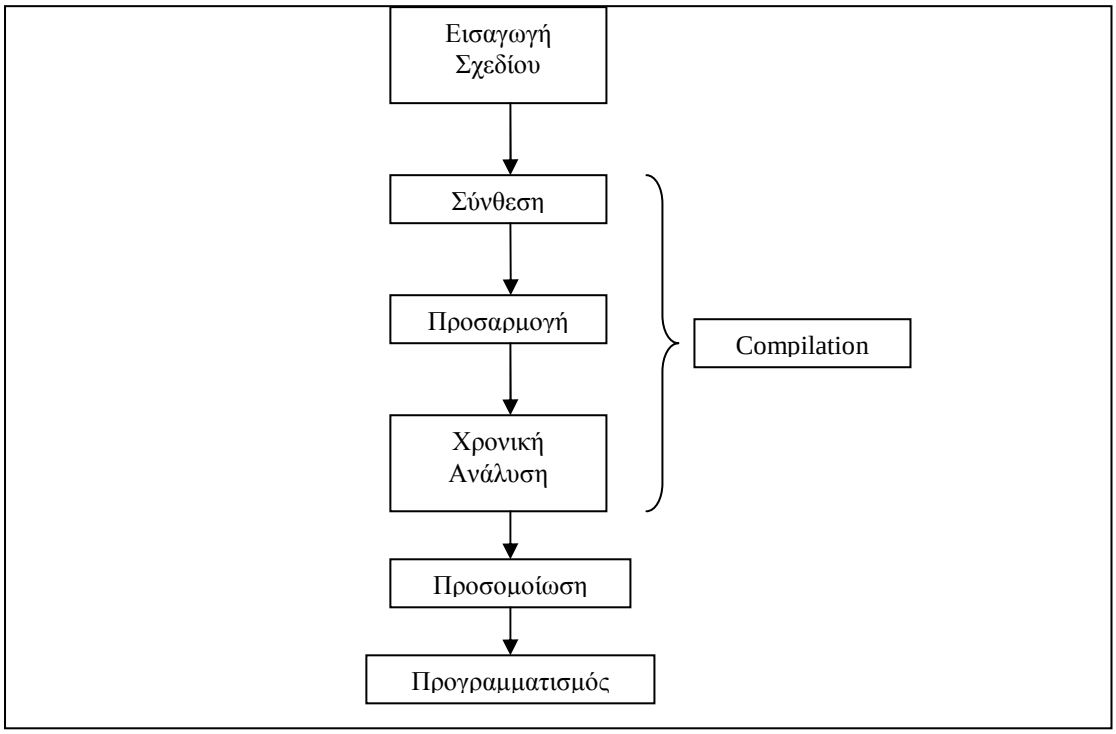
Ορισμένα λογισμικά σχεδίασης επιτρέπουν στο τέλος τον προγραμματισμό του κυκλώματος, ώστε το σχέδιό μας να αποτυπωθεί στο ολοκληρωμένο κύκλωμα για το οποίο προορίζεται, διαμορφώνοντάς το κατάλληλα.

7.1.2 Το QUARTUS II

Το λογισμικό QUARTUS II είναι ένα από τα γνωστότερα προγράμματα σχεδίασης CAD. Είναι πνευματική ιδιοκτησία της εταιρίας ALTERA, η οποία κατασκευάζει ορισμένες από τις πιο διαδεδομένες οικογένειες κυκλωμάτων CPLDs και FPGAs. Ενσωματώνει σε ένα ολοκληρωμένο περιβάλλον λογισμικού όλες τις λειτουργίες που αναφέρθηκαν παραπάνω. Στο τελικό στάδιο επιτρέπει την εύρεση της καταλληλότερης διαμόρφωσης (fitting), την χρονική ανάλυση ώστε να διαπιστωθεί η ορθή χρονική απόκριση του κυκλώματος στους παλμούς εισόδου καθώς και τον προγραμματισμό (διαμόρφωση) της διάταξης.

Το QUARTUS II χρησιμοποιείται για την ανάπτυξη και τον προγραμματισμό όλων των αναπτυξιακών κυκλωμάτων της εταιρείας ALTERA δηλαδή των διατάξεων CPLDs και FPGAs που κατασκευάζει η εταιρία. Στις παρακάτω παραγράφους θα αναπτυχθούν όλα τα στάδια της σχεδίασης με το QUARTUS II, και θα παρουσιαστεί ένα αναλυτικό παράδειγμα. Στο παράδειγμα αυτό θα δείξουμε τη σειρά λειτουργιών του προγράμματος, από την αρχική

σχεδίαση του κυκλώματος ως τον προγραμματισμό του. Η ροή διεργασιών στο QUARTUS II φαίνεται στο παρακάτω σχήμα 7.2.



Σχήμα 7.2 *Ροή Διεργασιών στο QUARTUS II*

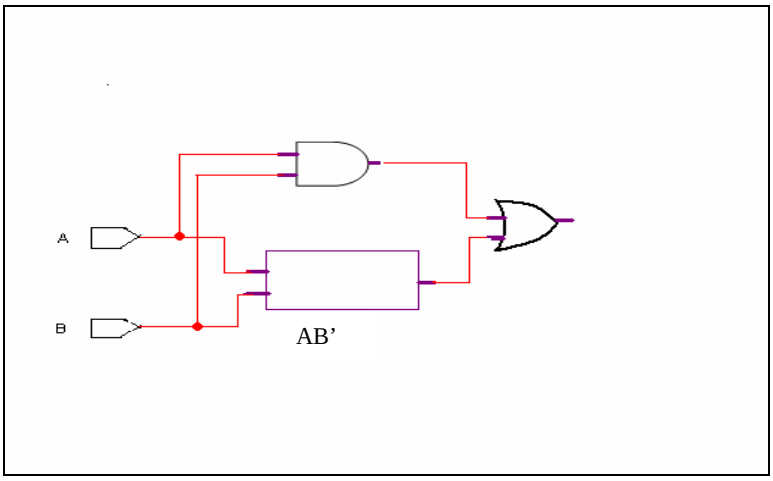
7.1.3 Εισαγωγή Σχεδίασης

Κατά την εισαγωγή σχεδίασης, ο χρήστης περιγράφει το κύκλωμα που θέλει να υλοποιήσει. Υπάρχουν τρεις τρόποι περιγραφής ενός κυκλώματος, που υποστηρίζουν τα περισσότερα προγράμματα σχεδίασης. Οι τρόποι αυτοί είναι η σχεδίαση με την βοήθεια πινάκων αληθείας, η σχεδίαση με βάση σχηματικά διαγράμματα και τέλος η σχεδίαση με τη βοήθεια κάποιας γλώσσας περιγραφής υλικού (HDL).

Κατά τον πρώτο τρόπο σχεδίασης, δηλαδή με την εισαγωγή πινάκων αληθείας, ο χρήστης μέσω ενός επεξεργαστή κυματομορφών εισάγει στο πρόγραμμα τις τιμές των εισόδων και τις επιθυμητές τιμές των εξόδων που θέλει να έχει το κύκλωμα του, και το πρόγραμμα σχεδίασης αναλαμβάνει να δημιουργήσει αυτό το κύκλωμα. Αυτός ο τρόπος εισαγωγής κυκλώματος δεν είναι εύχρηστος γι' αυτό και προτιμάται μόνο για απλά και μικρά κυκλώματα.

Ο δεύτερος τρόπος εισαγωγής είναι το σχηματικό διάγραμμα. Σε αυτόν τον τρόπο η σχεδίαση γίνεται με την βοήθεια σχεδιαστικών εργαλείων που παρέχονται από το πρόγραμμα σχεδίασης και με την χρήση εγκατεστημένων βιβλιοθηκών, οι οποίες περιέχουν έναν απλές πύλες ή και σύνθετα κυκλώματα. Οι πύλες αυτές περιέχονται σε διάφορους τύπους και με διαφορετικό αριθμό εισόδων. Για την αποφυγή δημιουργίας μεγάλων κυκλωμάτων, παρέχεται από το σχεδιαστικό πρόγραμμα η δυνατότητα χρήσης ιεραρχικών βαθμίδων (blocks). Δηλαδή μπορούμε να δημιουργούμε κυκλώματα, που στο εσωτερικό τους περιέχουν άλλα μικρότερα κυκλώματα. Αυτός ο τρόπος σχεδίασης λέγεται και ιεραρχική σχεδίαση. Ένα παράδειγμα ιεραρχικής σχεδίασης φαίνεται στο σχήμα 7.3 που υπάρχει παρακάτω. Το κύκλωμα αυτό είναι μια σχηματική απεικόνιση της συνάρτησης F=AB + AB', όπου το AB' έχει γίνει με την χρήση

της ιεραρχικής σχεδίασης. Ο τρόπος αυτός εισαγωγής κυκλώματος είναι κατάλληλος και για σύνθετα κυκλώματα.



Σχήμα 7.3 *Κύκλωμα Με Χρήση Ιεραρχικής Σχεδίασης*

Τέλος, ο άλλος τρόπος είναι η σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής κυκλωμάτων. Με αυτόν τον τρόπο σχεδίασης περιγράφουμε την λειτουργία του κυκλώματος με τη βοήθεια κατάλληλου κώδικα (βλέπε παράρτημα). Η γλώσσα που θα χρησιμοποιήσουμε εμείς είναι η VHDL. Υπάρχουν και άλλες γλώσσες περιγραφής υλικού (παράδειγμα η Verilog), από τις οποίες άλλες υποστηρίζονται από το Ινστιτούτο Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών (IEEE) και άλλες όχι. Η VHDL είναι μία από τις γλώσσες που υποστηρίζονται. Η σχεδίαση ενός κυκλώματος με μια πρότυπη γλώσσα περιγραφής υλικού παρέχει στον χρήστη τη δυνατότητα της μεταφοράς σε κάποιο άλλο σύστημα σχεδίασης, χωρίς να χρειάζεται αλλαγή στο κώδικα. Αυτό είναι ένα πλεονέκτημα αυτού του τρόπου σχεδίασης, σε αντίθεση με τους άλλους δύο που περιγράψαμε, οι οποίοι δεν είναι σίγουρο ότι μπορούν να μεταφερθούν από το ένα σύστημα στο άλλο. Ένα κοινό σημείο που έχει η σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής υλικού με το σχηματικό διάγραμμα είναι ότι ο κώδικας HDL μπορεί να πάρει ιεραρχική μορφή, δίνοντας μας την δυνατότητα να σχεδιάσουμε μεγάλα και πολύπλοκα κυκλώματα.

Με βάση τα προηγούμενα, ας δούμε αναλυτικά τους τρόπους εισόδου που παρέχει το πρόγραμμα Quartus. Πιο συγκεκριμένα, για να δώσουμε είσοδο κυκλώματος στο Quartus II χρησιμοποιούμε έναν από τους εξής Editors:

α) **Waveform Editor**. Αυτόν τον Editor τον χρησιμοποιούμε όταν ως είσοδο χρησιμοποιούμε τον πίνακα αληθείας του κυκλώματος.

β) **Block Editor**. Στην περίπτωση αυτή περιγράφουμε τον τρόπο διασύνδεσης βασικών δομικών μονάδων (π.χ. λογικών πυλών). Δηλαδή σαν είσοδο έχουμε ένα σχηματικό διάγραμμα.

γ) **Text Editor** μέσω του οποίου περιγράφουμε το ψηφιακό σύστημα με βάση τη συμπεριφορά του, δηλαδή σαν είσοδο χρησιμοποιούμε μία από τις γλώσσες περιγραφής υλικού (harware description languages) που αναφέραμε παραπάνω.

7.1.4 Σύνθεση

Το επόμενο βήμα μετά την ολοκλήρωση της περιγραφής της ψηφιακής λογικής είναι η διαδικασία της σύνθεσης. Στο QUARTUS II η διαδικασία της σύνθεσης αναφέρεται ως «Analysis & Synthesis». Κατά την διάρκεια αυτής της διαδικασίας η είσοδός μας μετατρέπεται

στις κατάλληλες λογικές συναρτήσεις, με τρόπο που να ταιριάζει στην τεχνολογία της συγκεκριμένης διάταξης που τελικά θα διαμορφώσουμε. Για παράδειγμα, έστω ότι έχουμε σαν είσοδο κάποιον πίνακα αληθείας, και σας στόχο μια διάταξη CPLD. Κατά την διάρκεια της σύνθεσης δημιουργούνται λογικές συναρτήσεις που εκφράζουν τους πίνακες αληθείας, με τρόπο κατάλληλο ώστε τελικά να απεικονιστούν σε μια διάταξη λογικών πινάκων (Logic Arrays). Στους λογικούς πίνακες AND-OR στηρίζονται οι γεννήτριες συναρτήσεων των CPLDs. Αν έχουμε σαν είσοδο σχηματικό διάγραμμα και στόχο ένα FPGA, κατά την σύνθεση παίρνουμε λογικές εξισώσεις ισοδύναμες με το κύκλωμα του σχηματικού διαγράμματος, οι οποίες μπορούν να μεταφερθούν σε πίνακες αναφοράς (look-up tables). Τέλος, όταν η εισαγωγή γίνεται μέσω μιας γλώσσας περιγραφής υλικού, κατά την διαδικασία της σύνθεσης τίθεται σε λειτουργία ένα άλλο τμήμα της σύνθεσης, ο μεταφραστής, και σαν έξοδο έχουμε την περιγραφή του κυκλώματος σε χαμηλό επίπεδο, κατάλληλο για την τεχνολογία της διάταξης-στόχου.

Με λίγα λόγια δηλαδή, η σύνθεση χρησιμοποιείται ως ένα αυτόματο εργαλείο υλοποίησης του αρχικού κυκλώματος που δίνει ο σχεδιαστής, με τρόπο κατάλληλο για την τεχνολογία του στόχου. Ταυτόχρονα, το κύκλωμα που παίρνουμε στην έξοδο της σύνθεσης είναι καλύτερο από το αρχικό. Δηλαδή, κατά την σύνθεση έχουμε και βελτιστοποίηση του κυκλώματος.

7.1.5 Προσαρμογή (fitting)

Η διαδικασία της προσαρμογής (fitting) λέγεται αλλιώς και δρομολόγηση (place and route). Στη φάση αυτή χρησιμοποιείται η βάση δεδομένων που έχει δημιουργηθεί κατά την ανάλυση και σύνθεση, και αντιστοιχίζεται η ψηφιακή λογική στους ελεύθερους πόρους της διάταξης-στόχου. Μάλιστα, λαμβάνονται αυστηρά υπόψη και οι χρονικές απαιτήσεις που τέθηκαν από τον σχεδιαστή. Δηλαδή, στη φάση αυτή καθορίζεται το πόσα και ποια συγκεκριμένα λογικά στοιχεία (logic elements-LE's) του ολοκληρωμένου κυκλώματος (chip) θα χρησιμοποιηθούν. Επίσης επιλέγονται συνδέσεις από τον προγραμματιζόμενο πίνακα διασυνδέσεων (programmable interconnect), ώστε να διασυνδεθούν τα απαραίτητα LE's μεταξύ τους.

7.1.6 Χρονική ανάλυση και παραγωγή αρχείων προγραμματισμού

Η χρονική ανάλυση (timing analysis), παράγεται ως αποτέλεσμα της προσαρμογής. Στην χρονική ανάλυση παράγονται οι καλύτεροι και οι χειρότεροι χρόνοι του κυκλώματος, με βάση τις προβλεπόμενες καθυστερήσεις κατά μήκος των διαδρομών. Δηλαδή, υπολογίζονται οι καθυστερήσεις που υφίστανται τα σήματα από την είσοδο τους στο κύκλωμα ως την έξοδο τους. Οι καθυστερήσεις οφείλονται κυρίως στο μήκος των καλωδίων και στον αριθμό των ενδιάμεσων βαθμίδων. Η χρονική ανάλυση είναι ένα σημαντικό σημείο κατά τη σχεδίαση ενός κυκλώματος, καθώς θα πρέπει οι χρόνοι που παίρνουμε από την εκτέλεση της να είναι συμβατοί με αυτούς που καθορίζονται από το ρολόι του συστήματος. Σε περίπτωση μη συμβατότητας των χρόνων θα πρέπει να διορθώσουμε τις αποκρίσεις αυτές, κάνοντας κατάλληλες χρονικές εκχωρήσεις (timing assignments) στο λογισμικό και επαναλαμβάνοντας την δρομολόγηση.

Η τελευταία φάση (assembling) είναι αυτή της παραγωγής συμβολικού (.hex) και δυαδικού (.sof) κώδικα. Κατ' αυτήν δημιουργούνται τα προγραμματιστικά αρχεία, που θα χρησιμοποιηθούν για την τελική διαμόρφωση.

Οι παραπάνω διαδικασίες, δηλαδή η σύνθεση, η προσαρμογή, η χρονική ανάλυση καθώς και η διαδικασία του «Assembler», συνθέτουν την διαδικασία της μετάφρασης (Compilation). Δηλαδή απεικονίζουν το επιθυμητό κύκλωμα σε μία προγραμματιζόμενη διάταξη (FPGA ή

CPLD) και δημιουργούν κώδικα για την προσομοίωση λειτουργίας (Simulation), και τον προγραμματισμό των διατάξεων (device programming) .

7.1.7 Προσομοίωση

Κατά την διάρκεια της λειτουργίας της προσομοίωσης (Simulation) ελέγχουμε κατά πόσο το κύκλωμα μας λειτουργεί σωστά. Δεν αρκεί μόνο η λειτουργία της σύνθεσης κατά την οποία γίνεται βελτιστοποίηση του κυκλώματος μας για να συμπεράνουμε ότι το κύκλωμα που σχεδιάσαμε είναι σωστό. Εξάλλου όπως αναφέραμε και πριν γίνεται βελτιστοποίηση του ήδη υπάρχοντος κυκλώματος χωρίς να υπάρχει κάποιος ελεγκτής για την διόρθωση σχεδιαστικών λαθών.

Ο έλεγχος της σωστής λειτουργίας γίνεται στο τμήμα των διεργασιών που ονομάζεται προσομοίωση. Εδώ συνδυάζονται δύο παράμετροι. Ο ένας είναι το αρχικό μας σχέδιο και ο άλλος είναι κάποιες τιμές που δίνει ο χρήστης στις εισόδους του κυκλώματος ώστε να ελέγξει αν οι τιμές που θα πάρει στην έξοδο του ανταποκρίνονται στις αρχικές προδιαγραφές που είχαμε θέσει για το κύκλωμα μας. Ο προσομοιωτής εξάγει τις τιμές της εξόδου του κυκλώματος μας μέσω ενός πίνακα αληθείας ή μέσω ενός διαγράμματος χρονισμού. Στην περίπτωση του διαγράμματος χρονισμού πρέπει να λάβουμε υπόψη μας ότι ο προσομοιωτής θεωρεί ότι ο χρόνος που χρειάζεται να μεταβούν τα σήματα των εισόδων στις πύλες είναι μηδενικός, κάτι που στην πραγματικότητα δεν ισχύει. Αυτό έχει σαν αποτέλεσμα να υπάρχει ένα ποσοστό αμφιβολίας ακόμα και μετά την προσομοίωση για το αν το κύκλωμα που σχεδιάσαμε μπορεί να υλοποιηθεί.

Τέλος υπάρχουν δύο τύποι προσομοίωσης η λειτουργική (**functional**) προσομοίωση και η προσομοίωση χρονισμού (**timing**). Στην πρώτη περίπτωση της προσομοίωσης δε λαμβάνονται υπόψη οι καθυστερήσεις των στοιχείων (πυλών και διασυνδέσεων) του κυκλώματος αλλά απλά επαληθεύεται ότι η λογική συνάρτηση που υλοποιεί το κύκλωμα είναι η σωστή. Στην δεύτερη περίπτωση προσομοίωσης, επαληθεύουμε την ορθότητα του κυκλώματος με βάση τους χρονικούς περιορισμούς του. Η χρονική προσομοίωση εμφανίζει τη χειρότερη περίπτωση καθυστέρησης στην έξοδο.

7.1.8 Προγραμματισμός και διαμόρφωση της συσκευής

Η διαδικασία του προγραμματισμού είναι η τελευταία ενέργεια που γίνεται για την ολοκλήρωση της δημιουργίας του κυκλώματος μας. Ως γνωστό τα CPLDs ή τα FPGAs πρέπει να προγραμματιστούν για να υλοποιήσουν το κύκλωμα που σχεδιάσαμε. Τα απαραίτητα αρχεία για τον προγραμματισμό και τη διαμόρφωση της συσκευής, έχουν δημιουργηθεί κατά την διαδικασία του assembler, που είναι το τελευταίο τμήμα της μετάφρασης (compilation) που κάναμε, σύμφωνα με τα προηγούμενα.

Η Altera επιτρέπει τον προγραμματισμό των συσκευών της με δύο τρόπους. Ο ένας είναι μέσω του κυκλώματος διεπαφής JTAG και ο άλλος ο Active Serial (AS) mode. Τα διαμορφωμένα αρχεία μεταφέρονται από τον υπολογιστή του χρήστη στο board μέσω ενός καλωδίου το οποίο συνδέεται σε θύρα του υπολογιστή μας (παράλληλη ή USB) και στο board όπου βρίσκεται το CPLD ή το FPGA. Η σύνδεση αυτή γίνεται μέσω του κατάλληλου οδηγού (driver) USB-Blaster ή BYTE-BLASTER.

Στη περίπτωση που χρησιμοποιήσουμε την JTAG διεπαφή, τα δεδομένα μας πηγαίνουν κατευθείαν στο ολοκληρωμένο κύκλωμα. Είναι ένας απλός και γρήγορος τρόπος για να διαμορφώσουμε ένα λογικό κύκλωμα. Με αυτόν τον τρόπο το ολοκληρωμένο κύκλωμα (αν είναι FPGA) διατηρεί την διαμόρφωση που του έχουμε δώσει για όσο διαρκεί η τροφοδοσία

του. Αν σταματήσει να τροφοδοτείται χάνεται και η διαμόρφωση του. Αυτό δεν ισχύει για τα κυκλώματα CPLDs, τα οποία διαμορφώνονται μόνιμα, με δυνατότητα επανεγγραφής, όπως οι μνήμες flash EEPROM.

Στην άλλη περίπτωση (AS) μια διάταξη με μνήμες flash, που βρίσκεται πάνω στην ίδια πλακέτα με το FPGA, χρησιμοποιείται για να αποθηκεύει τα αρχεία διαμόρφωσης. Σε αυτήν την περίπτωση το Quartus II στέλνει τα δεδομένα μας στη μνήμη Flash και αυτή με τη σειρά της στο chip FPGA. Στην περίπτωση αυτή δεν μας ενδιαφέρει αν υπάρχει τροφοδοσία ή όχι. Η επιλογή για το ποιόν από τους δύο τρόπους θα χρησιμοποιήσουμε γίνεται συνήθως μέσω ενός διακόπτη RUN/PROG που βρίσκεται πάνω στο board. Η αυτόματη (default) επιλογή είναι για διαμόρφωση με JTAG, ενώ η δεύτερη για Active Serial (AS).

Τέλος για να επιβεβαιώσουμε ότι το κύκλωμα μας λειτουργεί σωστά πρέπει να το δοκιμάσουμε δίνοντας κατάλληλες εισόδους 0 ή 1από τους διακόπτες ή άλλες συσκευές εισόδου που βρίσκονται πάνω στο board. Το αποτέλεσμα των εξόδων εμφανίζεται στους κατάλληλους ακροδέκτες εξόδου, που πρέπει να τους συνδέσουμε με συσκευές απεικόνισης πάνω στην αναπτυξιακή πλακέτα.

7.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

7.2.1 Εισαγωγή, Σύνθεση και Προσομοίωση του Ψηφιακού Κυκλώματος

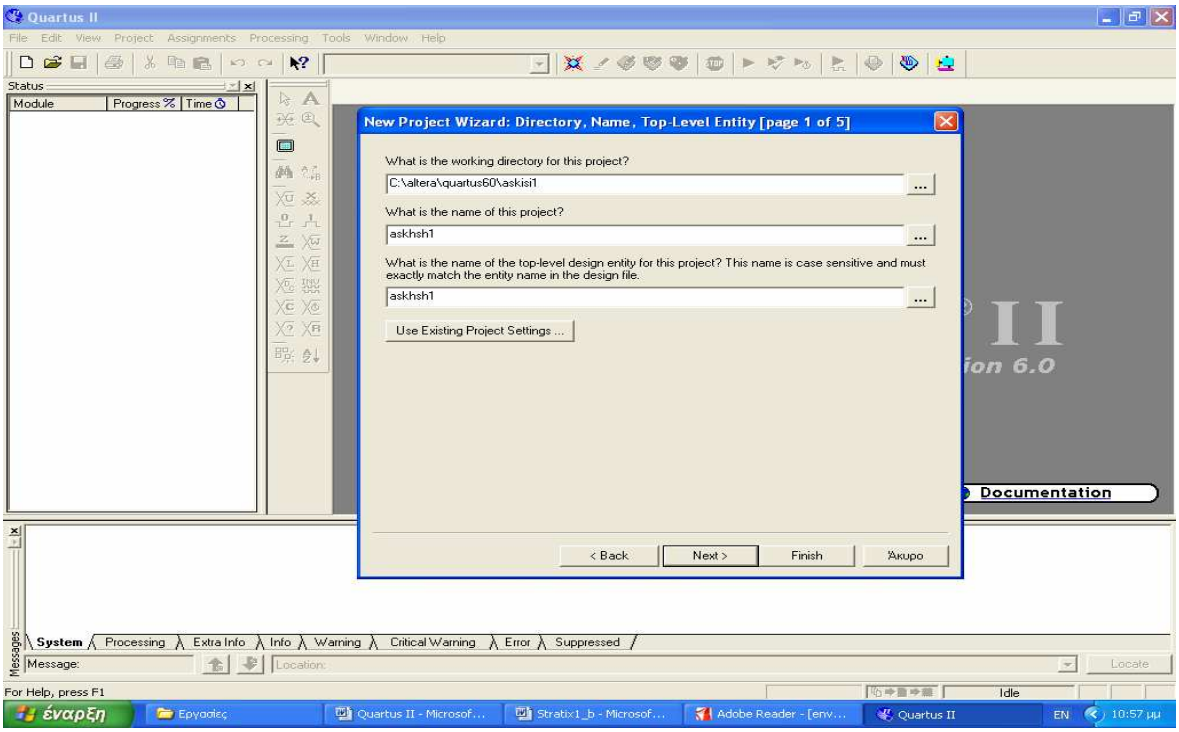
Σε αυτή την ενότητα θα περιγράψουμε με ένα παράδειγμα την χρήση και τις βασικές λειτουργίες του εργαλείου σχεδίασης Quartus II, της εταιρίας Altera. Τα βήματα που ακολουθούμε ξεκινούν από τον τρόπο σχεδίασης ενός ψηφιακού κυκλώματος (μέσω σχηματικού διαγράμματος) και φτάνουν στον προγραμματισμό ενός FPGA, ώστε να εκτελεί την λειτουργία του κυκλώματος που σχεδιάσαμε. Στο παράδειγμα αυτό θα υλοποιήσουμε τη συνάρτηση $\mathbf{F=X_1X_2+X_1´X_3+X_2´X_3´}$.

7.2.2 Ορισμός του σχεδίου (Project)

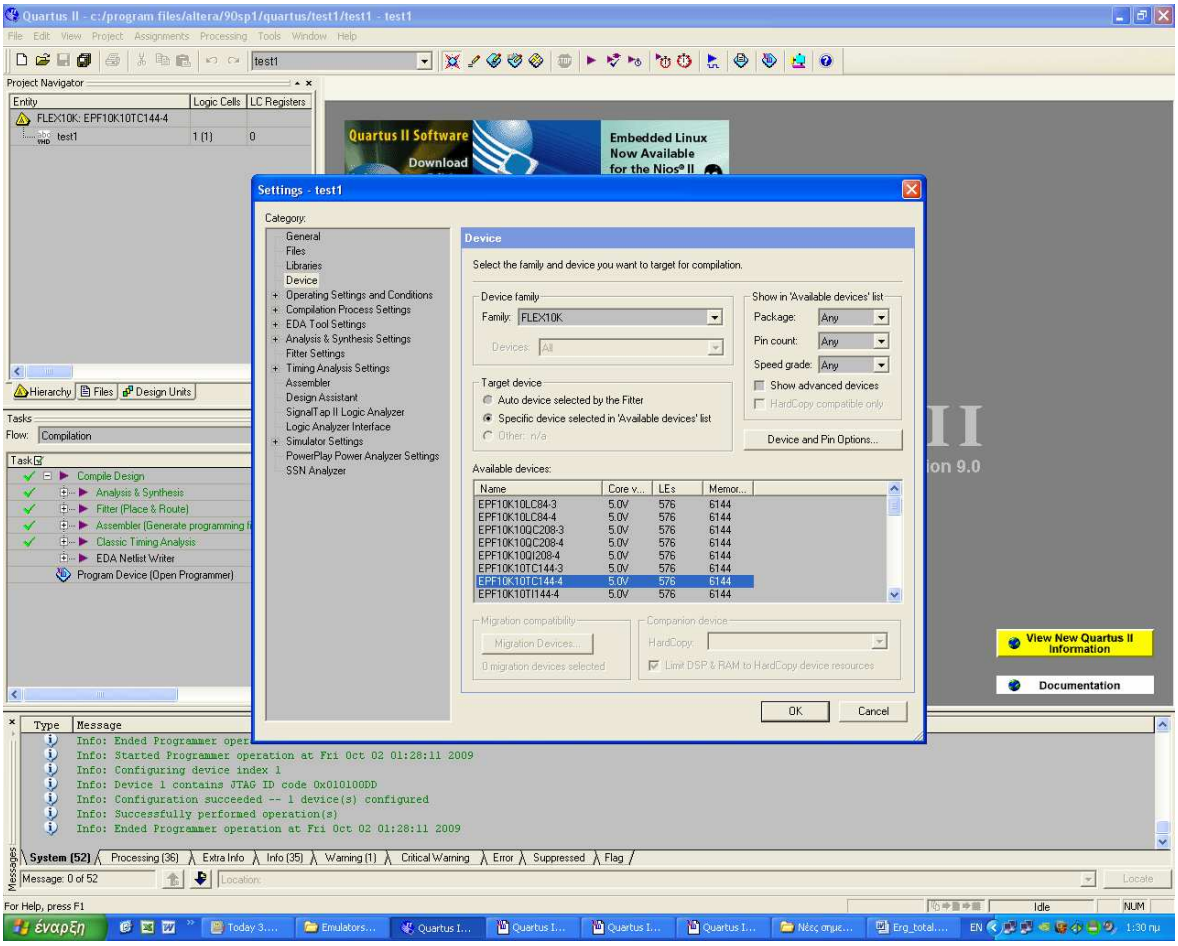
Ανοίγοντας το σχεδιαστικό περιβάλλον του Quartus II μας ζητείται η ονομασία του project. Η έννοια του project περιλαμβάνει το σύνολο των αρχείων που δημιουργούμε εμείς (το αρχικό σχέδιο και τις κυματομορφές εισόδου για την προσομοίωση), καθώς και τα αρχεία που δημιουργεί το πρόγραμμα για να εκτελέσει τις διάφορες λειτουργίες του. Με λίγα λόγια, σαν project θεωρούμε το σύνολο των αρχείων που δημιουργούνται για μια εφαρμογή. Η δημιουργία ενός project γίνεται από το **Menu File/New Project Wizard**.

Στη συνέχεια, ανοίγει ένα παράθυρο στο οποίο δηλώνουμε το όνομα του Project, καθώς και το όνομα του φακέλου που θα το περιέχει (σχήμα 7.4). Δημιουργείστε έναν διαφορετικό φάκελο, για κάθε ξεχωριστή εφαρμογή. Ποτέ μην αποθηκεύετε στον ίδιο φάκελο δύο διαφορετικά σχέδια (projects). ***Μια καλή συμβουλή είναι να χρησιμοποιείτε το ίδιο όνομα για τα τρία πεδία της καρτέλας του σχ. 7.4***.

Πατώντας το NEXT στην ίδια καρτέλα, το πρόγραμμα μας ζητάει να επιλέξουμε από λίστα την συγκεκριμένη διάταξη FPGA που θα χρησιμοποιήσουμε. Στο παράδειγμά μας, το όνομα του Project είναι askhsh1 και το FPGA που θα χρησιμοποιήσουμε ανήκει στην οικογένεια **FLEX10K** και είναι το **EPF 10K10TC144-4** (βλέπε σχ. 7.5).



Σχήμα 7.4 *Δημιουργία Φακέλου και Ονομασία του Project*



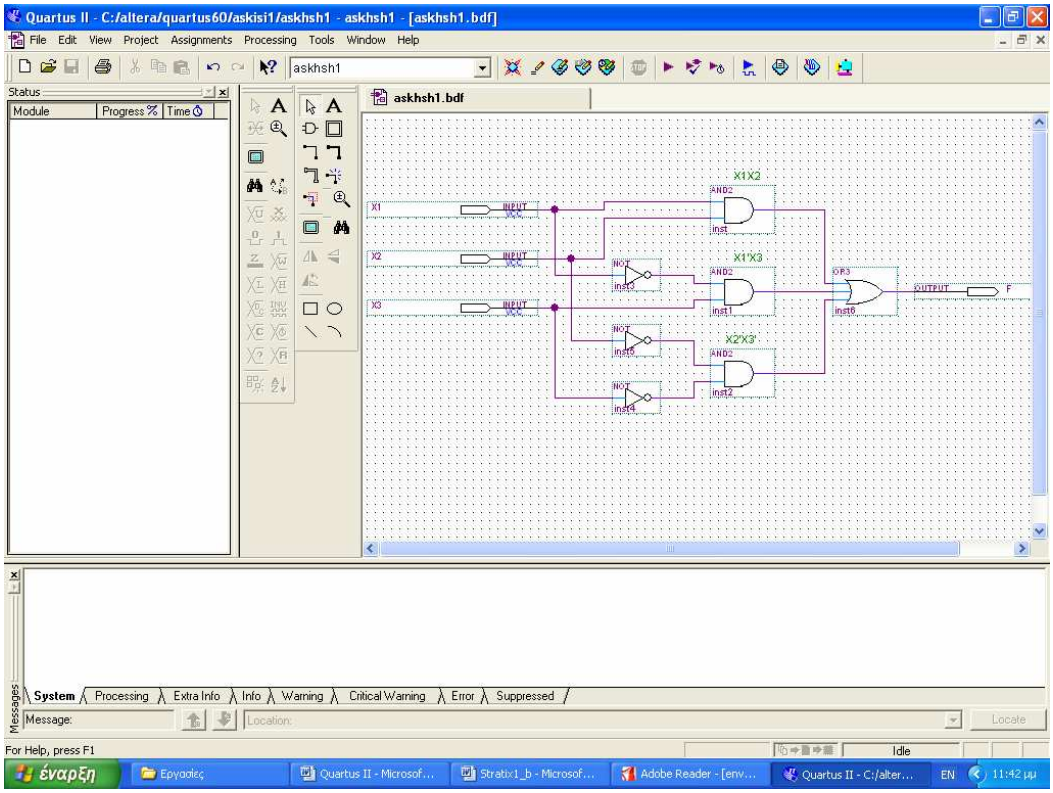
Σχήμα 7.5 *Επιλογή του FPGA που θα προγραμματίσουμε*

7.2.3 Εισαγωγή σχηματικού αρχείου

Αφού έχουμε δώσει όνομα στο Project μπορούμε να συνεχίσουμε κάνοντας την εισαγωγή του κυκλώματος μας με όποιον τρόπο επιθυμούμε. Η επιλογή αυτή γίνεται από την καρτέλα **New** στο **Menu File**. Εκεί έχουμε να επιλέξουμε ανάμεσα σε διαφορετικούς τρόπους δημιουργίας του κυκλώματος μας. Αν θέλουμε να το περιγράψουμε σχηματικά διαλέγουμε την επιλογή **Block Diagram/Schematic File**, ενώ αν θέλουμε να το περιγράψουμε με κάποια γλώσσα περιγραφής υλικού διαλέγουμε μία από τις ακόλουθες επιλογές: **Verilog HDL File** ή **VHDL File**.

Στην περίπτωση που το περιγράφουμε σχηματικά, το Quartus II μας παρέχει ένα μεγάλο αριθμό από πύλες, από ακροδέκτες εισόδου/εξόδου καθώς και από άλλα κυκλώματα όπως flip-flop ώστε να κάνει την σχεδίαση του κυκλώματος μας πιο εύκολη. Όλα αυτά υπάρχουν στο **Menu Edit/Insert Symbol**. Εναλλακτικά, οι βιβλιοθήκες ανοίγουν πιέζοντας το πλήκτρο Symbol Tool, στα αριστερά του επεξεργαστή. Οι βασικές πύλες βρίσκονται στη βιβλιοθήκη **Primitives/logic**, ενώ οι ακροδέκτες I/O βρίσκονται στη βιβλιοθήκη **Primitives/Pins**. Στο παρακάτω σχήμα 7.6 φαίνεται το κύκλωμα μας. Τα ονόματα των ακροδεκτών μπορούμε να τα ορίσουμε κατάλληλα κάνοντας διπλό κλικ πάνω στους ακροδέκτες.

Αφού σχεδιάσουμε το κύκλωμα, πρέπει να σώσουμε το αρχείο με το ίδιο όνομα που είχαμε δώσει στο Project, δηλαδή askhsh1.

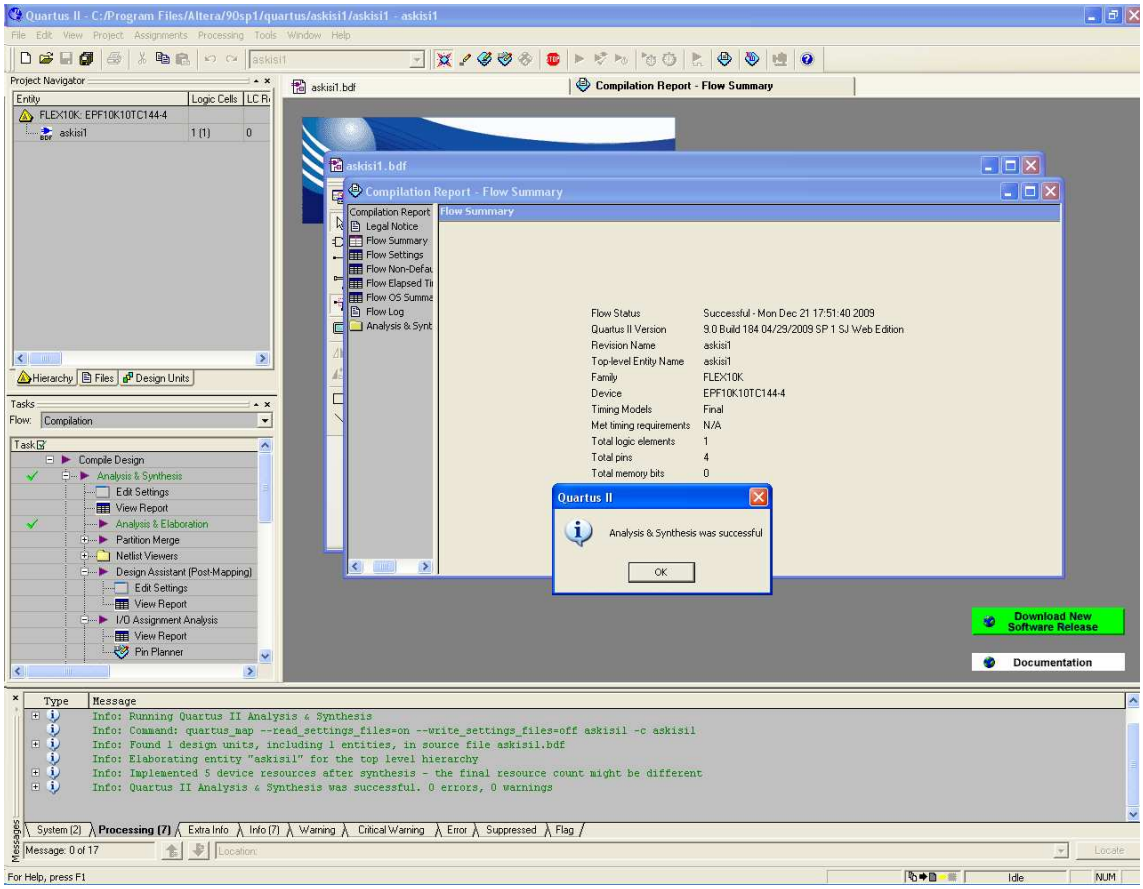


Σχήμα 7.6 *Το παραπάνω κύκλωμα υλοποιεί τη Συνάρτηση* $F=X_1X_2+X_2'X_3+X_1'X_3'$

7.2.4 Ανάλυση και σύνθεση

Αφού έχουμε τελειώσει με την περιγραφή του κυκλώματος μας και το έχουμε αποθηκεύσει, ακολουθεί η διαδικασία της Ανάλυσης και Σύνθεσης. Η διαδικασία αυτή ξεκινά από το **Menu Processing/Start/Start Analysis & Synthesis**.

Μετά το τέλος της διαδικασίας, εμφανίζεται μια αναφορά, που μας ενημερώνει για τις απαιτήσεις σε λογικά στοιχεία (LEs), bits μνήμης και ακροδέκτες, που έχει η εφαρμογή μας, όπως φαίνεται στο σχήμα. 7.7.



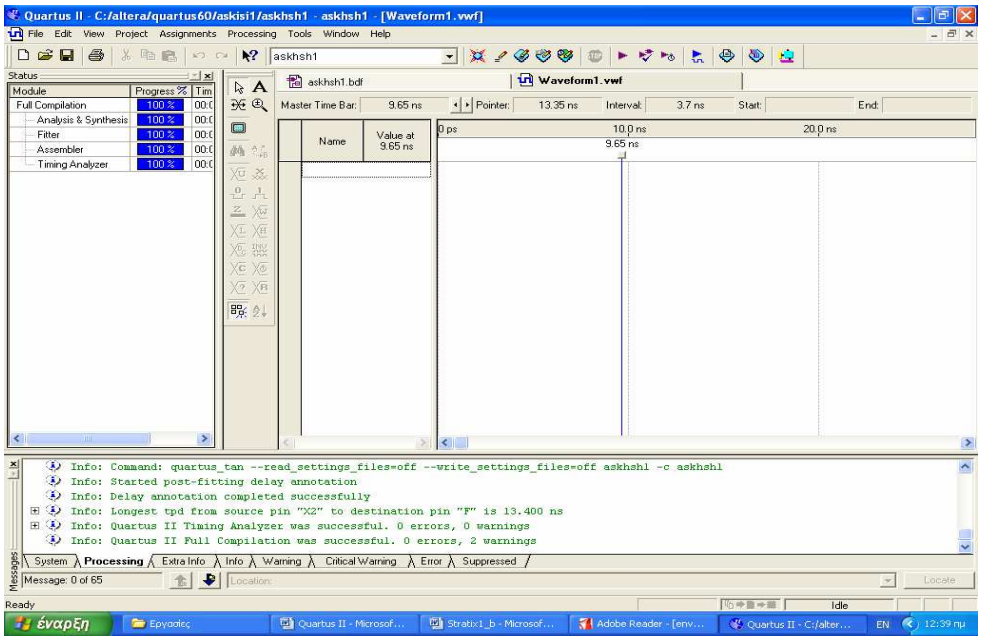
Σχήμα 7.7 *Η Διαδικασία της Ανάλυσης & Σύνθεσης (Analysis & Synthesis)*

7.2.5 Η διαδικασία της προσομοίωσης

Όταν τελειώσει η διαδικασία της μετάφρασης επιτυχώς, ακολουθεί η διαδικασία της προσομοίωσης. Στην προσομοίωση θέλουμε να διαπιστώσουμε εάν το κύκλωμα που σχεδιάσαμε επαληθεύει τον επιθυμητό πίνακα αληθείας, ο οποίος για το συγκεκριμένο κύκλωμα είναι ο ακόλουθος:

X1	X2	X3	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Η διαδικασία αυτή γίνεται μέσα από τον Waveform Editor, με τον οποίο εισάγουμε τις κατάλληλες κυματομορφές εισόδου (waveform vectors) που είναι απαραίτητες για την προσομοίωση. Ο Waveform Editor καλείται από την εισαγωγική καρτέλα, επιλέγοντας **Menu File/New/Other Files/Vector Waveform File**. Στην επιφάνεια εργασίας του Quartus εμφανίζεται το παράθυρο που φαίνεται στο σχήμα 7.8. Κάνοντας διπλό κλικ κάτω από Name, εμφανίζεται το παράθυρο **Insert Node or Bus**. Στο παράθυρο αυτό πατούμε στο κουμπί **Node Finder** και στη συνέχεια επιλέγουμε **List** (προσέχοντας να έχουμε βάλει στην επιλογή Filter: Pins All). Επιλέγουμε τις εισόδους και εξόδους του κυκλώματος, των οποίων τη λειτουργία επιθυμούμε να προσομοιώσουμε.

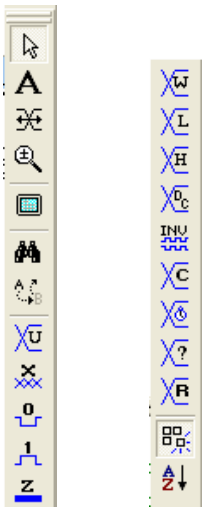


Σχήμα 7.8 *Παράθυρο επεξεργαστή κυματομορφών πριν την εισαγωγή εισόδων/εξόδων*

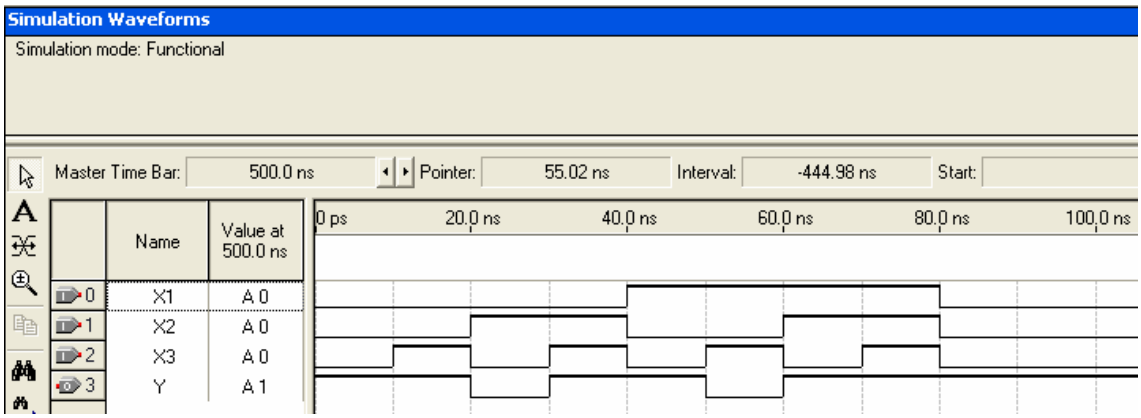
Μετά την ολοκλήρωση της παραπάνω διαδικασίας, στο παράθυρο του επεξεργαστή έχουν εμφανιστεί τα ονόματα των εισόδων και των εξόδων του κυκλώματος. Πριν θέσουμε τιμές στις εισόδους μας, θα πρέπει να επιλέξουμε **Menu Processing/Generate Functional Simulation Netlist** ώστε να δημιουργηθεί ένα αρχείο λειτουργικής περιγραφής του κυκλώματος. Επίσης από το **Menu Assignments/Settings/Simulator Settings** μπορούμε να διαλέξουμε αν η προσομοίωση θα είναι Timing ή Functional (χρονική ή λειτουργική). Επιλέγουμε **Functional**. Στην ίδια καρτέλα θέτουμε τη συνολική διάρκεια της προσομοίωσης στο 1μs. (**End Simulation at 1μs**).

Κατόπιν, χρησιμοποιώντας τα εργαλεία του Editor (σχ. 7.9), μπορούμε να δημιουργήσουμε τις κατάλληλες κυματομορφές εισόδου, θέτοντας τις τιμές των εισόδων σε 1 ή 0, για συγκεκριμένα χρονικά διαστήματα. Ακολουθήστε το υπόδειγμα του σχ. 7.10 για τις εισόδους. Αποθηκεύουμε το αρχείο των διανυσμάτων εισόδου (Waveform vector file) με όνομα askisi1.wvf.

Τέλος, πηγαίνοντας στο **Menu Processing/Start Simulation** εκτελούμε την προσομοίωση. Στο τέλος της προσομοίωσης, εμφανίζονται οι τιμές των εξόδων για τις τιμές των εισόδων που σχεδιάσαμε στα παραπάνω βήματα.



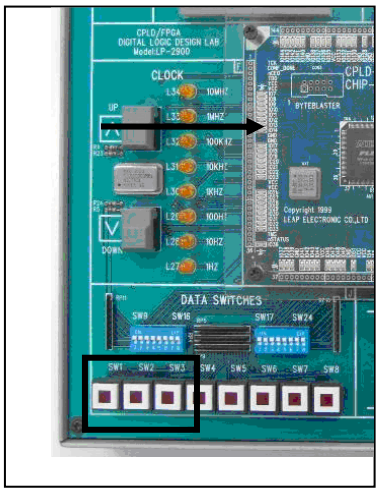
Σχήμα 7.9 *Εργαλεία του Waveform Editor. Αφού επιλέξουμε ένα χρονικό διάστημα της εισόδου που επιθυμούμε να επεξεργαστούμε, θέτουμε την κυματομορφή εισόδου σε 1 ή 0 με τα αντίστοιχα εργαλεία.*



Σχήμα 7.10 *Το παράθυρο της προσομοίωσης με τιμές εισόδων και εξόδων*

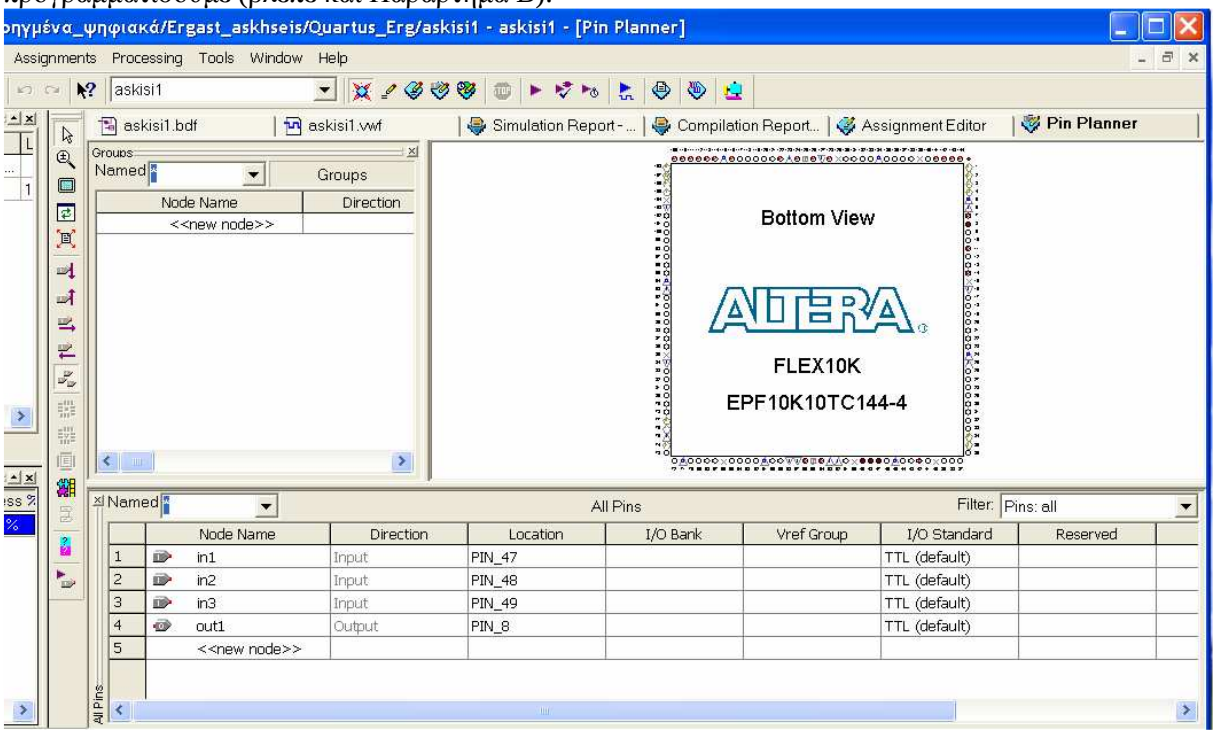
7.2.6 Ορισμός ακροδεκτών (pin assignments)

Το επόμενο βήμα είναι να αντιστοιχίσουμε τις εισόδους και τις εξόδους του κυκλώματός μας με συγκεκριμένους ακροδέκτες της διάταξης που πρόκειται να προγραμματίσουμε. Έχουμε ήδη ορίσει ότι η διάταξή μας είναι ένα FPGA που ανήκει στην οικογένεια FLEX10K της εταιρίας ALTERA και συγκεκριμένα η διάταξη EPF10K10TC144-4 (βλέπε στο **Menu Assignments/Device**). Προκειμένου να ορίσουμε σε ποιους ακροδέκτες του τσιπ θα συνδεθούν οι τρεις είσοδοι και η μία έξοδος του κυκλώματος που σχεδιάσαμε κάνουμε τις εξής επιλογές:



Επιλέγουμε **Menu Assignments/Pins**. Ανοίγει το παράθυρο του σχ. 7.11. Στο πεδίο Node Name εμφανίζονται τα ονόματα που έχουμε δώσει στους ακροδέκτες μας στο σχηματικό διάγραμμα. Διπλοπατώντας πάνω στο πεδίο Location εμφανίζεται η αρίθμηση όλων των ακροδεκτών της συγκεκριμένης διάταξης. Εκεί, κάνουμε τις επιλογές που φαίνονται στο σχήμα 7.11 δηλαδή οι είσοδοι in1, in2, in3

αντιστοιχίζονται στα pin 47, pin 48, pin 49 που αντιστοιχούν στους διακόπτες τύπου push-button SW1, SW2 και SW3, αντίστοιχα. Η έξοδος out1 αντιστοιχίζεται στο pin 8 που αντιστοιχεί σε ένα SMD led πάνω στην πλακέτα (Chip-Board) του FPGA. Οι επιλογές αυτές προκύπτουν από τα σχηματικά διαγράμματα του κυκλώματος που πρόκειται να προγραμματίσουμε (βλέπε και Παράρτημα Β).



Σχήμα 7.11 *Παράθυρο Ορισμού Ακροδεκτών για το FLEX10K. Ορίστε τους ακροδέκτες με τον τρόπο που φαίνεται στο κάτω μέρος της εικόνας.*

Εναλλακτικά, η αντιστοίχιση ενός pin του FPGA με μια είσοδο ή έξοδο μπορεί να γίνει διπλοπατώντας στο pin που θέλουμε να αντιστοιχίσουμε πάνω στην απεικόνιση του ολοκληρωμένου κυκλώματος, στο πάνω δεξιά πλαίσιο στο παράθυρο του ορισμού ακροδεκτών και συμπληρώνοντας τα κατάλληλα στοιχεία στη φόρμα που θα εμφανιστεί.

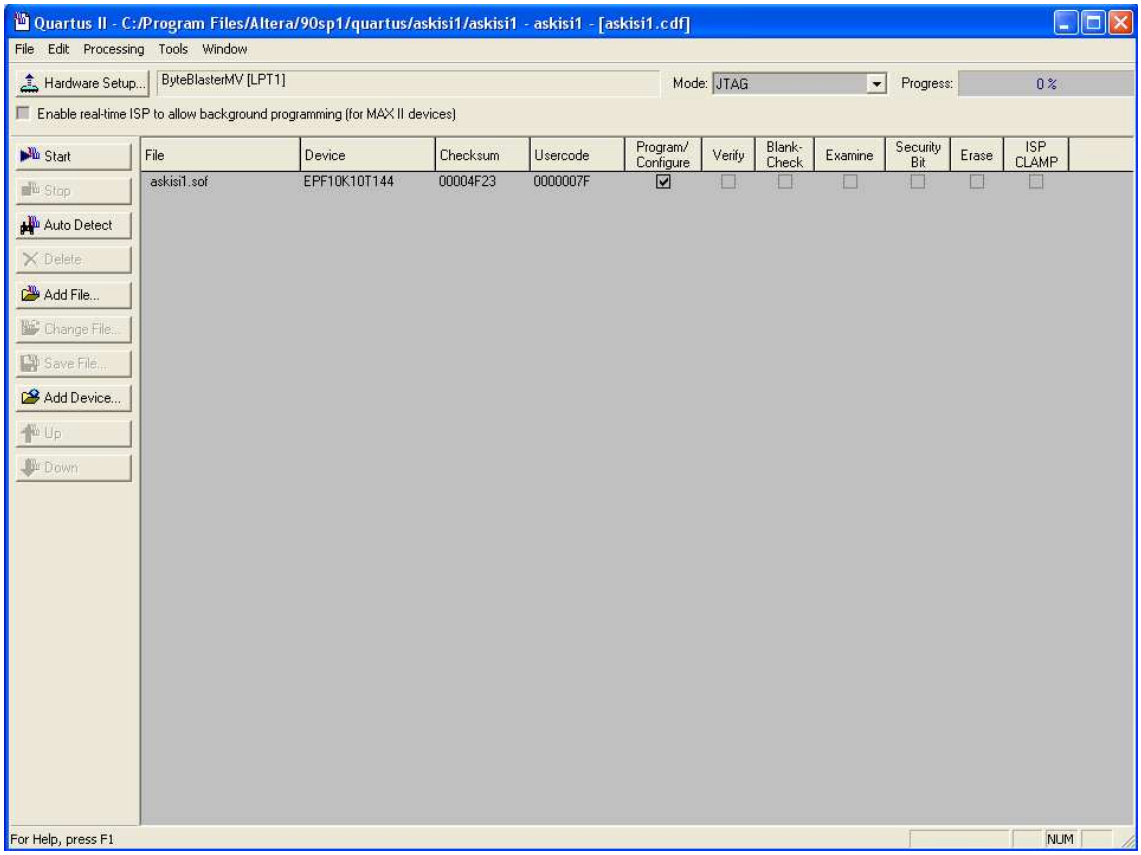
7.2.7 Προγραμματισμός (διαμόρφωση) του κυκλώματος

Αφού ολοκληρωθεί ο ορισμός ακροδεκτών και πριν πραγματοποιηθεί η διαμόρφωση του FPGA θα πρέπει να γίνει Μετάφραση (Compilation) ώστε να δημιουργηθεί το αρχείο *.sof το οποίο περιέχει όποιες πληροφορίες χρειάζονται για την διαμόρφωση του FPGA. Η Μετάφραση γίνεται από το Menu **Processing/Start Compilation**. Υπενθυμίζουμε ότι η μετάφραση περιλαμβάνει την Ανάλυση & Σύνθεση, τη Δρομολόγηση, την Χρονική Ανάλυση και την παραγωγή των αρχείων προγραμματισμού του FPGA.

Μετά την επιτυχή ολοκλήρωση της Μετάφρασης μπορεί να γίνει η διαμόρφωση του FPGA με τον Programmer, ο οποίος βρίσκεται στο **Menu Tools/Programmer**. Το παράθυρο που εμφανίζεται είναι αυτό που ακολουθεί.

Την πρώτη φορά που θα χρησιμοποιήσουμε τον programmer θα πρέπει να κάνουμε τις κατάλληλες επιλογές του υλικού (Hardware Setup) για την επιλογή του κατάλληλου κυκλώματος προγραμματισμού. Συνήθεις επιλογές είναι ο οδηγός **BYTE-BLASTER** για

προγραμματισμό μέσω της παράλληλης θύρας ή ο **USB-BLASTER** για προγραμματισμό μέσω της θύρας USB. Το αναπτυξιακό **LP-2900** που χρησιμοποιείται στο εργαστήριο χρησιμοποιεί τον οδηγό BYTE-BLASTER.



Σχήμα 7.12 *Το παράθυρο του Programmer*

Αν δεν είναι ήδη επιλεγμένο θα πρέπει να επιλέξουμε το τελικό αρχείο διαμόρφωσης (.sof) που δημιουργήθηκε κατά το τελικό στάδιο της μετάφρασης. Κατόπιν, επιλέγοντας Start αρχίζει η διαδικασία διαμόρφωσης, που συνήθως διαρκεί μερικά δευτερόλεπτα. Η πρόοδος της διαδικασίας εμφανίζεται στην μπλε μπάρα στα δεξιά της οθόνης.

ΕΡΓΑΣΤΗΡΙΟ 8

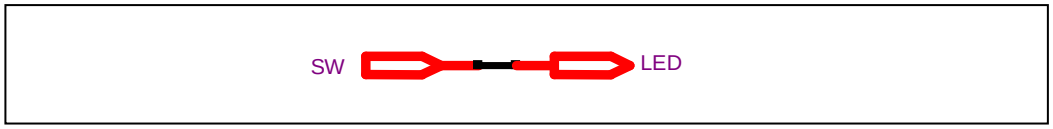
ΕΙΣΑΓΩΓΗ ΣΤΗ ΓΛΩΣΣΑ VHDL

8.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

8.1.1 Απλές Εντολές Αντιστοίχησης - Άναμμα LED

Για να συνθέσουμε ένα λογικό κύκλωμα, που προορίζεται να προγραμματίσει διαμορφούμενα κυκλώματα, πρέπει να περιγράψουμε το κύκλωμα χρησιμοποιώντας κάποιο εργαλείο σχεδίασης (CAD). Ένας τρόπος εισαγωγής του κυκλώματος είναι με χρήση σχηματικού διαγράμματος, όπως κάναμε στο προηγούμενο εργαστήριο. Σ' αυτήν και στις υπόλοιπες εργαστηριακές ασκήσεις που ακολουθούν, η περιγραφή του κυκλώματος θα γίνεται με την χρήση της γλώσσας περιγραφής υλικού VHDL, και του εργαλείου σχεδίασης QUARTUS II της εταιρίας ALTERA.

Για να δείξουμε τον τρόπο περιγραφής ενός κυκλώματος με την γλώσσα VHDL θα υλοποιήσουμε για αρχή ένα απλό κύκλωμα το οποίο θα περιέχει απλές εντολές αντιστοίχησης ενός διακόπτη σε ένα LED. Θα χρησιμοποιήσουμε 1 διακόπτη ως είσόδο του κυκλώματος μας και 1 LED ως έξοδο. Η είσοδος θα έχει όνομα sw, και θα είναι 1 bit. Η έξοδος θα έχει όνομα led, και θα είναι κι αυτή του 1 bit. Το κύκλωμα που θα υλοποιήσουμε θα έχει την μορφή που φαίνεται στο παρακάτω σχήμα 8.1.

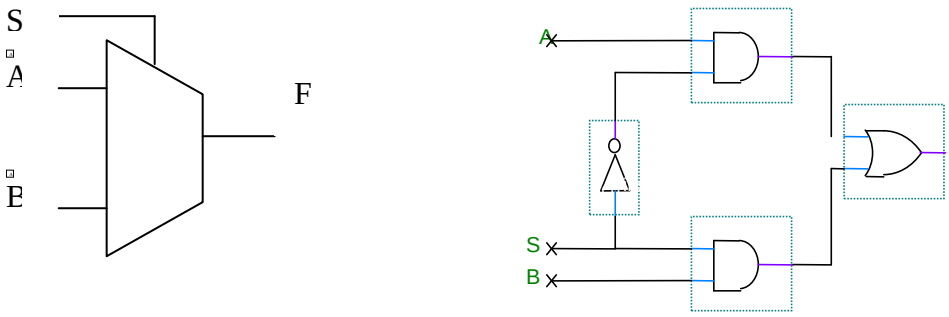


Σχήμα 8.1 *Κύκλωμα αντιστοίχησης διακόπτη σε LED*

8.1.2 Πολυπλέκτης 2:1

Σε αυτή την ενότητα θα σχεδιάσουμε έναν πολυπλέκτη 2 προς 1. Ως γνωστό, ο πολυπλέκτης είναι ένα κύκλωμα το οποίο αποτελείται από έναν αριθμό εισόδων, στην περίπτωση μας δύο, από κάποιες εισόδους επιλογής και από μια έξοδο. Ο αριθμός των διακοπτών επιλογής βρίσκεται από τον τύπο $\mathbf{2^v=x,}$ όπου x ο αριθμός των εισόδων και ν ο αριθμός των διακοπτών επιλογής. Στο δικό μας παράδειγμα, όπου έχουμε 2 εισόδους, χρειαζόμαστε έναν διακόπτη επιλογής αφού έχουμε $\mathbf{2^1=2.}$

Στο παρακάτω σχήμα φαίνεται ο πίνακας αληθείας, το λογικό κύκλωμα και το σύμβολο ενός πολυπλέκτη. Όπως βλέπουμε, ανάλογα με την τιμή του **S** (διακόπτης επιλογής) ως έξοδο λαμβάνουμε την αντίστοιχη είσοδο. Ο σπουδαστής μπορεί να ανατρέξει και στο εργαστήριο 4.



α) Σύμβολο β) Λογικό Κύκλωμα

S	F
0	A
1	B

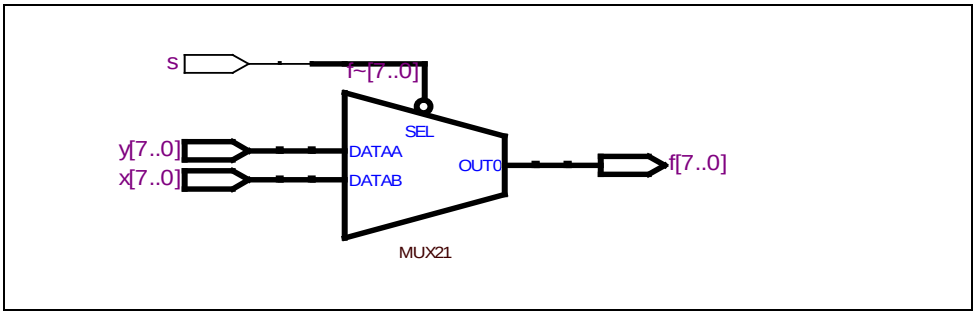
γ) Πίνακας Αληθείας

Σχήμα 8.2 *α) Σύμβολο Πολυπλέκτη β) Κύκλωμα Πολυπλέκτη γ) Πίνακας Αληθείας*

8.1.3 Πολυπλέκτης 8 bits

Σε αυτή την ενότητα θα αναφερθούμε στη σχεδίαση ενός πολυπλέκτη 2 προς 1, με την διαφορά όμως ότι τα κανάλια του δεν είναι του 1 bit, όπως στην προηγούμενη παράγραφο, αλλά των 8 bits. Η λογική της σχεδίασης ενός τέτοιου πολυπλέκτη δεν διαφέρει από αυτή του απλού πολυπλέκτη. Κι εδώ έχουμε δύο εισόδους, μία έξοδο και μια γραμμή επιλογής. Ο αριθμός των διακοπτών επιλογής ορίζεται από τον τύπο $2^v = x$. Άρα και εδώ αφού έχουμε δύο κανάλια εισόδου (x=2), ο διακόπτης επιλογής θα είναι ένας (v=1).

Λέγοντας ότι ο πολυπλέκτης είναι των 8 bits εννοούμε ότι κάθε είσοδος του καθώς και η έξοδος του έχουν εύρος 8 bits. Η είσοδος x θα γραφεί ως διάνυσμα x [7…0]. Αντίστοιχα, η άλλη είσοδος του πολυπλέκτη θα γραφτεί ως διάνυσμα y [7…0] και η έξοδος f [7…0]. Ο διακόπτης επιλογής s και σε αυτόν τον πολυπλέκτη θα είναι 1 bit. Στο παρακάτω σχήμα φαίνεται ένας πολυπλέκτης 2 προς 1, εύρους 8 bits.



Σχήμα 8.3 *Πολυπλέκτης 2 προς 1, 8 bits*

8.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

8.2.1 Απλές Εντολές Αντιστοίχησης - Άναμμα LED

Ο κώδικας περιγραφής ενός κυκλώματος σε VHDL χωρίζεται σε τρία τουλάχιστον τμήματα. Το πρώτο είναι το τμήμα όπου δηλώνονται οι βιβλιοθήκες. Οι βιβλιοθήκες μας περιλαμβάνουν έτοιμο κώδικα και δηλώσεις που θα χρησιμοποιήσουμε κι εμείς στο πρόγραμμά μας. Στο δεύτερο τμήμα περιγράφεται το κύκλωμα ως απλή οντότητα (ENTITY) με εισόδους και εξόδους. Στο τμήμα αυτό δηλώνονται τα σήματα εισόδου και εξόδου, με το όνομά τους και το εύρος τους σε bits. Το τρίτο τμήμα είναι το τμήμα της αρχιτεκτονικής. Εδώ περιγράφεται η λογική του κυκλώματος (ARCHITECTURE).

Στον παρακάτω κώδικα περιγράφουμε μια οντότητα με μια είσοδο sw του 1 bit και μια έξοδο με όνομα led, επίσης του 1 bit. Στο τμήμα της αρχιτεκτονικής ο κώδικας επιτελεί μια απλή αντιστοίχηση ανάμεσα στην είσοδο και την έξοδο.

```
--τμήμα βιβλιοθηκών
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- τμήμα οντότητας
ENTITY vhdl1 IS
PORT ( sw:   IN      std_logic;
          led :  OUT   std_logic);
 END vhdl1;

--τμήμα αρχιτεκτονικής
ARCHITECTURE behaviour OF vhdl1 IS
BEGIN
     led <= sw;
END behaviour ;
```

Από ότι βλέπουμε, ο κώδικας χωρίζεται πράγματι σε τρία τμήματα. Στο πρώτο τμήμα δηλώνουμε μόνο μια βιβλιοθήκη, την ieee.std logic 1164.all, η οποία περιέχει τον ορισμό του τύπου STD LOGIC VECTOR. Σε αυτό το σημείο του κώδικα μπορούμε να δηλώσουμε όχι μόνο βιβλιοθήκες που περιέχει η γλώσσα αλλά και βιβλιοθήκες που δημιουργεί ο χρήστης. Στο δεύτερο τμήμα υλοποιούμε μια βαθμίδα ψηφιακού κυκλώματος, που ονομάζεται vhdl1 και το οποίο δέχεται ως σήμα εισόδου το sw και παράγει ως σήμα εξόδου το led. Στο τρίτο τμήμα του κώδικα περιγράφεται η αρχιτεκτονική του ψηφιακού συστήματος που θα υλοποιηθεί. Έχει το όνομα Behaviour και αναφέρεται στην οντότητα vhdl1 (ARCHITECTURE Behaviour OF vhdl1 IS). Σ' αυτό το τμήμα γίνεται η αντιστοίχηση του διακόπτη σε LED (led <= sw;).

Για να γράψουμε κώδικα VHDL στο περιβάλλον του Quartus II θα δημιουργήσουμε κατ' αρχήν ένα νέο project, σύμφωνα με όσα περιγράψαμε στο προηγούμενο εργαστήριο (File/New Project Wizard). Αποθηκεύουμε το νέο project σε έναν νέο φάκελο, με όνομα "vhdl1". Δίνουμε στο project το ίδιο όνομα με τον φάκελο (vhdl1).

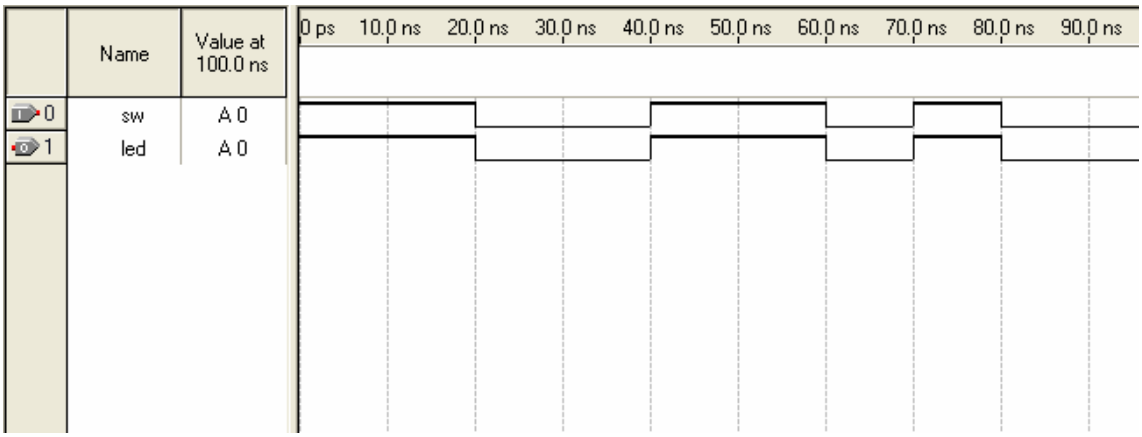
Για να ανοίξει ο επεξεργαστής VHDL επιλέγουμε Menu File/New και στο παράθυρο που ανοίγει επιλέγουμε VHDL File.

Αφού ανοίξει ο επεξεργαστής, πληκτρολογούμε τον παραπάνω κώδικα, αποφεύγοντας τα ορθογραφικά και συντακτικά λάθη.

Στη συνέχεια επιλέγουμε Menu Processing/Start/Analysis and Synthesis. Θα γίνει η λεπτομερής επεξεργασία του κώδικα και θα παραχθούν μηνύματα για τυχόν λάθη ή ασυμβατότητες.

Όταν διορθώσουμε όλα τα λάθη μπορούμε να προσομοιώσουμε τη λειτουργία του κυκλώματος που περιγράψαμε με τον κώδικα ακολουθώντας τα βήματα της προσομοίωσης, όπως και στο προηγούμενο εργαστήριο. Είναι απαραίτητο να δημιουργήσουμε το αρχείο διανυσμάτων κυματομορφών (Waveform Vector File- *.wvf). Για τον σκοπό αυτόν χρησιμοποιούμε τα εργαλεία του επεξεργαστή κυματομορφών (waveform Editor). Δεν παραλείπουμε να ορίσουμε τις σωστές επιλογές προσομοίωσης (Assignments/ Settings/Simulation), όπως κάναμε και στο προηγούμενο εργαστήριο. Μπορούμε να ξεκινήσουμε την προσομοίωση επιλέγοντας Menu Processing/Start/Simulation.

Στο σχήμα 8.4 που ακολουθεί φαίνεται η προσομοίωση του κυκλώματος. sw είναι το σήμα εισόδου και led η έξοδος του κυκλώματος. Όπως παρατηρούμε από τα αποτελέσματα της προσομοίωσης, όταν θέτουμε τον διακόπτη εισόδου σε λογική κατάσταση 1, τίθεται στην ίδια κατάσταση και η έξοδος. Παραδείγματος χάριν όταν το sw= ‘1’ τότε και το led= ‘1’.



Σχήμα 8.4 *Προσομοίωση Κυκλώματος*

8.2.1.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Το τμήμα αυτό της άσκησης προϋποθέτει στοιχειώδη κατανόηση της αναπτυξιακής πλακέτας (LP-2900), η οποία περιγράφεται στο παράρτημα Β. Εκεί, παρουσιάζονται σε πίνακες οι ακροδέκτες του διαμορφούμενου κυκλώματος (FPGA) που συνδέονται με διακόπτες εισόδου και Leds απεικόνισης της εξόδου. Για την αντιστοίχηση των εισόδων και των εξόδων της οντότητας που περιγράψαμε με τους ακροδέκτες του διαμορφούμενου κυκλώματος επιλέγουμε στο Quartus τον επεξεργαστή αντιστοίχησης (Menu/Assignments/Pins). Στη συνέχεια αντιστοιχήστε τους ακροδέκτες, όπως φαίνεται στον παρακάτω πίνακα:

SW	→ Pin 47 (SW1)
Led	→ Pin 8

Υπενθυμίζουμε ότι το Pin 8 αντιστοιχεί σε ένα SMD led πάνω στην πλακέτα (Chip-Board) του FPGA ενώ το Pin 47 αντιστοιχεί σε έναν από τους διακόπτες δεδομένων τύπου push-button, τον SW1. Επίσης, στο Pin 47 υπάρχει ένα SMD led που δείχνει την λογική κατάσταση του διακόπτη αυτού.

Μετά την αντιστοίχηση των ακροδεκτών εκτελέστε εκ νέου compilation (Menu Processing/Start Compilation) ώστε να δημιουργηθεί το αρχείο *.sof που περιέχει όλες τις απαραίτητες πληροφορίες για την διαμόρφωση του κυκλώματος. Τέλος, διαμορφώστε το ολοκληρωμένο (Menu Tools/Programmer/Start) και ελέγξτε στην αναπτυξιακή πλακέτα LP-2900 τη σωστή λειτουργία του κυκλώματος.

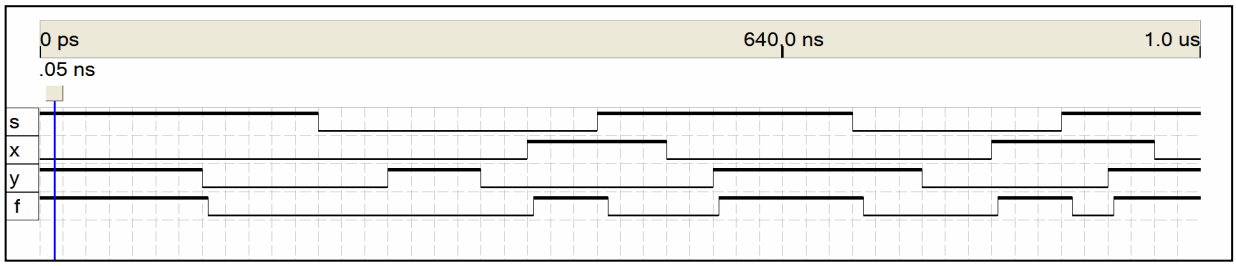
8.2.2 Πολυπλέκτης 2:1

Στον κώδικα που ακολουθεί παρουσιάζεται ένας από τους τρόπους που προσφέρονται για να περιγράψουμε σε VHDL έναν πολυπλέκτη 2 προς 1. Η βιβλιοθήκη που χρησιμοποιούμε είναι η ieee.std_logic_1164.all. Η οντότητα έχει όνομα mux2_1 με εισόδους x, y και s και έξοδο την f.

Το σώμα της αρχιτεκτονικής δίνουμε την ονομασία structural. Μέσα στο σώμα της αρχιτεκτονικής ορίζουμε δύο σήματα (m1, m2). Τα σήματα αυτά που στην ουσία είναι εσωτερικά καλώδια του κυκλώματος είναι ίδιου τύπου και ίδιου μεγέθους (ενός bit) με τις εισόδους αλλά και με την έξοδο. Ο τρόπος περιγραφής της σχέσης μεταξύ εισόδων και εξόδου, που χρησιμοποιούμε στο τμήμα της αρχιτεκτονικής, ονομάζεται «δομικός», κι αυτό επειδή η περιγραφή αυτή βασίζεται στα δομικά στοιχεία τις πύλες AND, OR και NOT που απαρτίζουν το κύκλωμα.

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

ENTITY mux2_1 IS
PORT (x,y,s : IN std_logic;
            f : OUT std_logic;
            gnd: OUT bit);
END mux2_1;

ARCHITECTURE structural OF mux2_1 IS
SIGNAL m1, m2 : std_logic;
BEGIN
  gnd<='1';
  m1<= x AND NOT(s);
  m2<= y AND s;
  f<= m1 OR m2;
END structural;
```

Στο σχήμα 8.5 που ακολουθεί, φαίνεται η προσομοίωση της λειτουργίας του πολυπλέκτη που αναλύθηκε παραπάνω. Παρατηρούμε ότι κάθε φορά που αλλάζουμε τιμή στην είσοδο επιλογής s, στην έξοδο του κυκλώματος παίρνουμε την αντίστοιχη τιμή της εισόδου. Παραδείγματος χάριν, στην αρχή της προσομοίωσης για s=’1’ η τιμή της εξόδου f είναι ίση με την τιμή της εισόδου y, δηλαδή ίση με ‘1’.



Σχήμα 8.5 *Προσομοίωση Λειτουργίας Πολυπλέκτη*

8.2.2.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Η αντιστοίχηση των ακροδεκτών γίνεται με τον ίδιο τρόπο που περιγράψαμε και στην άσκηση 8.1. Για να έχουμε απεικόνιση της εξόδου σε ένα από τα led L1 έως L12 προσθέτουμε τον ακροδέκτη gnd OUT τύπου bit στο τμήμα της οντότητας του κώδικα και στη συνέχεια στο τμήμα της αρχιτεκτονικής τον θέτουμε ίσο με '1'. Ο ακροδέκτης αυτός αντιστοιχίζεται στο Pin 141 το οποίο συνδέεται μέσω μια πύλης NOT στην κάθοδο των led L1 έως L12. Οι είσοδοι x,y αντιστοιχίζονται στους διακόπτες τύπου dip SW9 και SW17, η είσοδος s αντιστοιχίζεται στον διακόπτη τύπου push-button SW1 και η έξοδος f αντιστοιχίζεται στο led L1.

Η αντιστοίχιση ακροδεκτών θα είναι η ακόλουθη:

x	→ Pin 64 (SW9)
y	→ Pin 78 (SW17)
s	→ Pin 47 (SW1)
f	→ Pin 7 (L1)
gnd	→ Pin 141

Μετά την αντιστοίχηση των ακροδεκτών εκτελέστε εκ νέου compilation (Menu Processing/Start Compilation) ώστε να δημιουργηθεί το αρχείο *.sof που περιέχει όλες τις απαραίτητες πληροφορίες για την διαμόρφωση του κυκλώματος. Τέλος, διαμορφώστε το ολοκληρωμένο (Menu Tools/Programmer/Start) και ελέγξτε στην αναπτυξιακή πλακέτα LP-2900 τη σωστή λειτουργία του κυκλώματος.

8.2.3 Πολυπλέκτης 8 bits

Στην περίπτωση αυτή, η περιγραφή του πολυπλέκτη θα γίνει με διαφορετικό τρόπο από ότι στην περίπτωση της προηγούμενης παραγράφου. Αυτό συμβαίνει όχι επειδή υπάρχει κάποια διαφορά στην λειτουργία ανάμεσα στους δύο πολυπλέκτες, αλλά για να υποδείξουμε έναν διαφορετικό τρόπο προσέγγισης του ίδιου κυκλώματος. Στο μέρος της δήλωσης των βιβλιοθηκών δεν έχουμε καμία αλλαγή, αφού ο τύπος STD_LOGIC_VECTOR που χρησιμοποιούμε για τα σήματα των 8 bits περιέχεται στην ίδια βιβλιοθήκη (ieee.std_logic_1164.all). Στο μέρος της δήλωσης της οντότητας βλέπουμε πώς μπορούμε να δηλώσουμε τέτοιου είδους σήματα (f : out std_logic_vector (7 downto 0)).

Η είσοδος s είναι ο διακόπτης επιλογής του πολυπλέκτη και με την λέξη WITH (λέξη-κλειδί στη VHDL) δηλώνουμε την λειτουργία της αυτή. Οι δύο επόμενες εντολές που περιέχουν την λέξη WHEN δηλώνουν ότι η έξοδος f παίρνει την τιμή x αν το s='0' και την τιμή y για άλλη περίπτωση (δηλαδή για s='1'). Η χρήση της λέξης OTHERS είναι αναγκαία γιατί η γλώσσα VHDL θεωρεί ότι πρέπει να υπάρχει μια WHEN για κάθε πιθανή τιμή του σήματος. Το

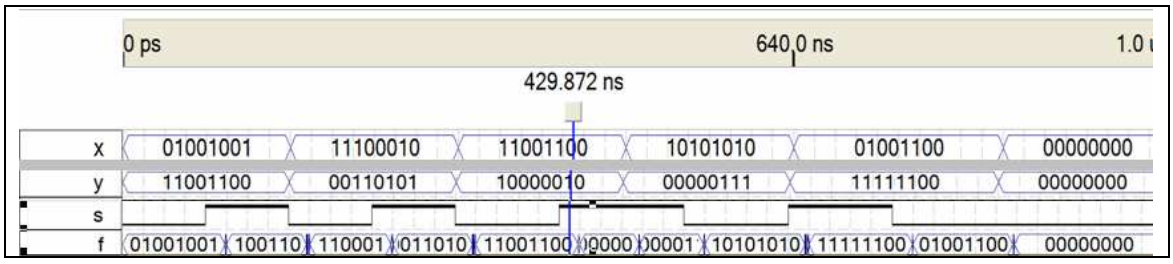
σήμα εισόδου s έχει δηλωθεί σαν std_logic, που σημαίνει ότι οι δυνατές τιμές που μπορεί να πάρει είναι ‘0’ , ‘1’ ή ‘Z’, άρα με την χρήση της λέξης OTHERS προνοούμε για όλες τις υπόλοιπες τιμές της s. Στον κώδικα που ακολουθεί περιγράφονται όλα αυτά που αναφέρθηκαν παραπάνω.

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

ENTITY part2 IS
PORT (x,y : IN std_logic_vector (7 downto 0);
        s : IN std_logic;
        f : OUT std_logic_vector (7 DOWNTO 0);
        gnd : OUT bit);
END part2;

ARCHITECTURE behaviour OF part2 IS
BEGIN
 gnd<=’1’;
 WITH s SELECT
   f<= x WHEN '0',
       y WHEN OTHERS;
END behaviour;
```

Στο σχήμα που ακολουθεί φαίνεται η προσομοίωση ενός τέτοιου πολυπλέκτη. Όπως βλέπουμε, κάθε bit στην έξοδο παίρνει την τιμή του αντίστοιχου bit της εισόδου ανάλογα με την τιμή του διακόπτη επιλογής s.

Για να εμφανίσουμε συνολικά αποτελέσματα εκεί που έχουμε σήματα πολλών bits μπορούμε να ορίσουμε τα σήματα εισόδου και εξόδου στον waveform Editor ως εξής:

Ανοίγουμε νέο αρχείο WVF. Με δεξί κλικ στην οθόνη του Editor, επιλέγουμε Zoom/Fit in Window. Με δεξί κλικ στη στήλη “Name” των σημάτων I/O Επιλέγουμε Insert Node or Bus/Node Finder/List/. Κατόπιν επιλέγουμε τα συνολικά σήματα x, y, και f (όχι τα επιμέρους bits) και τα εισάγουμε στην δεξιά στήλη. Στο παράθυρο Insert Node or bus επιλέγουμε το εύρος του σήματος σε bits. (για τα σήματα x, y και f 8 bits). Κατόπιν, επιλέγουμε περιοχές και με διπλό κλικ δίνουμε τιμή (Arbitrary value) στο δυαδικό σύστημα. Το αποτέλεσμα της προσομοίωσης φαίνεται παρακάτω.



Σχήμα 8.6 *Προσομοίωση Πολυπλέκτη 8 bit*

8.2.3.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Η αντιστοίχηση των ακροδεκτών γίνεται με τον ίδιο τρόπο που περιγράψαμε στις προηγούμενες ασκήσεις. Οι είσοδοι x[0], x[1],....x[7] αντιστοιχίζονται στους διακόπτες τύπου dip SW9 έως SW16, οι είσοδοι y[0], y[1],....y[7] αντιστοιχίζονται στους διακόπτες τύπου dip SW17 έως SW24, η είσοδος s αντιστοιχίζεται στον διακόπτη τύπου push-button SW1 και οι έξοδοι f[0], f[1],...f[7] αντιστοιχίζονται στα led L1 έως L8. Τέλος, ο ακροδέκτης gnd συνδέεται στο Pin 141, όπως εξηγήθηκε στην προηγούμενη άσκηση.

Η αντιστοίχιση ακροδεκτών θα είναι η ακόλουθη:

x[0] → Pin 64 (SW9)	**y[0]** → Pin 78(SW17)	**f[0]** → Pin 7(L1)	**s** → Pin 47 (SW1)
x[1] → Pin 65 (SW10)	**y[1]** → Pin 79 [SW18)	**f[1]** → Pin 8 (L2)	**gnd**→ Pin 141
x[2] → Pin 67 (SW11)	**y[2]** → Pin 80(SW19)	**f[2]** → Pin 9(L3)	
x[3] → Pin 68 (SW12)	**y[3]** → Pin 81 (SW20)	**f[3]** → Pin 10 (L4)	
x[4] → Pin 69 (SW13)	**y[4]** → Pin 82 (SW21)	**f[4]** → Pin 11 (L5)	
x[5] → Pin 70 (SW14)	**y[5]** → Pin 83 (SW22)	**f[5]** → Pin 12 (L6)	
x[6] → Pin 72 (SW15)	**y[6]** → Pin 86 (SW23)	**f[6]** → Pin 13 (L7)	
x[7] → Pin 73 (SW16)	**y[7]** → Pin 87 (SW24)	**f[7]** → Pin 14 (L8)	

Μετά την αντιστοίχηση των ακροδεκτών εκτελέστε εκ νέου compilation ώστε να δημιουργηθεί το αρχείο *.sof που περιέχει όλες τις απαραίτητες πληροφορίες για την διαμόρφωση του κυκλώματος. Τέλος, διαμορφώστε το ολοκληρωμένο και ελέγξτε στην αναπτυξιακή πλακέτα τη σωστή λειτουργία του κυκλώματος

ΕΡΓΑΣΤΗΡΙΟ 9

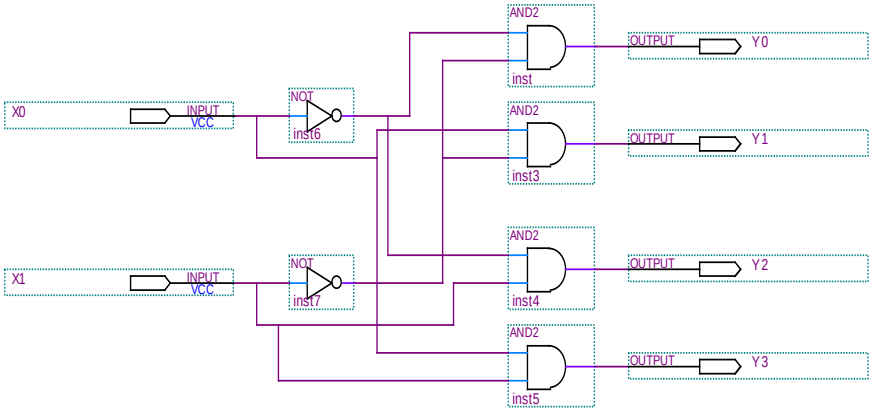
ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΕΣ – ΚΩΔΙΚΟΠΟΙΗΤΕΣ – ΣΥΓΚΡΙΤΕΣ

Σ’ αυτό το εργαστήριο θα περιγράψουμε έναν δυαδικό αποκωδικοποιητή 2 προς 4, έναν αποκωδικοποιητή BCD σε 7 segment και τέλος έναν συγκριτή 4 bits.

9.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

9.1.1 Αποκωδικοποιητής 2 προς 4

Τα κυκλώματα των αποκωδικοποιητών χρησιμοποιούνται για να αποκωδικοποιούν μια πληροφορία η οποία βρίσκεται σε κωδικοποιημένη μορφή. Γενικά για την δημιουργία ενός αποκωδικοποιητή ν εισόδων χρειάζονται $2^{ν}$ έξοδοι. Στο σχήμα που ακολουθεί φαίνεται ο πίνακας αληθείας ενός αποκωδικοποιητή 2 προς 4 καθώς και το κύκλωμά του.



α) Κύκλωμα Αποκωδικοποιητή

X_1	X_0	Y_0	Y_1	Y_2	Y_3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**

β) Πίνακας αληθείας Αποκωδικοποιητή

Σχήμα 9.1 *Αποκωδικοποιητής 2 προς 4*

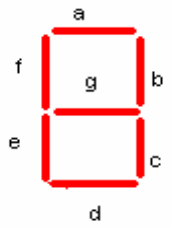
9.1.2 Αποκωδικοποιητής BCD σε 7segment

Ένας αποκωδικοποιητής κώδικα BCD σε ενδείκτη επτά τομέων (7 segment display), μετατρέπει ένα δεκαδικό ψηφίο σε σήματα που οδηγούν τις διόδους φωτοεκπομπής (LEDs) του

ενδείκτη. Οι δίοδοι αυτοί συμβολίζονται με τα γράμματα a ως g. Ο μετατροπέας αυτός αντιστοιχίζει κάθε συνδυασμό των εισόδων στον αντίστοιχο αριθμό του δεκαδικού συστήματος. Δηλαδή θέτει σε λειτουργία τα απαραίτητα LEDs ώστε να σχηματιστεί ο αριθμός. Ο πίνακας αληθείας του μετατροπέα καθώς και ο ενδείκτης επτά τομέων φαίνονται στο σχήμα 9.2 που ακολουθεί.

X_3	X_2	X_1	X_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

α) Πίνακας αληθείας



β) Ενδείκτης Επτά Τομέων

Σχήμα 9.2 *Πίνακας Αληθείας Μετατροπέα και Ενδείκτης Επτά Τομέων*

9.1.3 Συγκριτές

Οι συγκριτές περιγράφονται αναλυτικά στο Εργαστήριο 3.

9.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

9.2.1 Αποκωδικοποιητής 2 προς 4

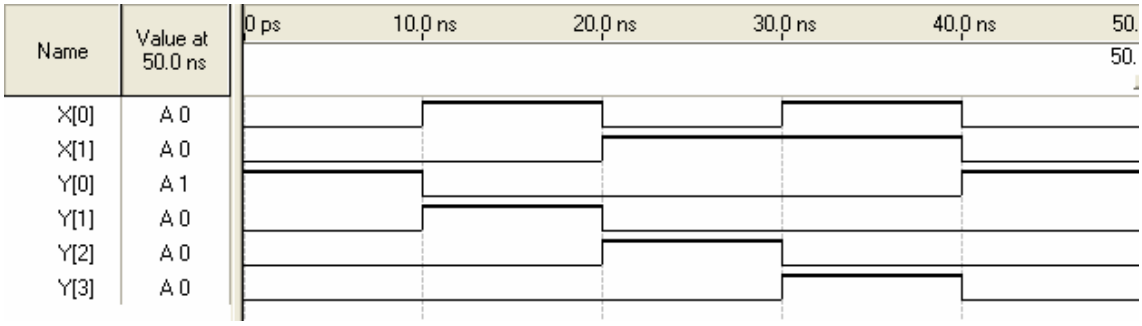
Και στην περίπτωση του αποκωδικοποιητή 2 προς 4 δεν βλέπουμε καμία διαφορά στη συγγραφή του κώδικα του σε σχέση με τα προηγούμενα κυκλώματα. Και εδώ χρησιμοποιούμε τις ίδιες βιβλιοθήκες και τους ίδιους τύπους σημάτων. Σαν είσοδο έχουμε δηλώσει έναν ακροδέκτη ο οποίος όμως έχει μέγεθος 2 bits, κάτι το οποίο μας δίνει την δυνατότητα να μπορούμε να ελέγξουμε το κάθε bit του ξεχωριστά σαν να επρόκειτο για δύο ακροδέκτες. Την ίδια λογική έχουμε ακολουθήσει και στην δήλωση των τεσσάρων εξόδων.

Στο σώμα της αρχιτεκτονικής έχουμε περιγράψει τον αποκωδικοποιητή με βάση την συμπεριφορά του και όχι με βάση την δομή του. Απ' ότι βλέπουμε και στον κώδικα που ακολουθεί έχουμε αντιστοιχήσει μία τιμή για κάθε σήμα της εξόδου ανάλογα με την τιμή που έχουμε στο σήμα της εισόδου. Έτσι για την τιμή ¨00¨ παίρνουμε στην έξοδο το ¨0001¨, για ¨01¨ παίρνουμε ¨0010¨, για ¨10¨ το ¨0100¨και τέλος για ¨11¨ παίρνουμε στην έξοδο το ¨1000¨.

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

```
ENTITY Decoder_2to4 IS
PORT(X :IN std_logic_vector (1 DOWNTO 0);
     Y :OUT std_logic_vector (3 DOWNTO 0));
END Decoder_2to4;

ARCHITECTURE behaviour OF Decoder_2to4 IS
BEGIN
 WITH X SELECT
  Y<= "0001" WHEN "00",
      "0010" WHEN "01",
      "0100" WHEN "10",
      "1000" WHEN  OTHERS;
END behaviour;
```

Στο σχήμα που ακολουθεί φαίνεται η προσομοίωση του αποκωδικοποιητή 2 προς 4. Όπως παρατηρούμε στο σχήμα, οι τιμές στις εξόδους σε σχέση με τις τιμές που έχουμε στην είσοδο επαληθεύουν το πίνακα αληθείας του αποκωδικοποιητή.



Σχήμα 9.3 *Προσομοίωση Αποκωδικοποιητή 2 προς 4*

9.2.1.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Να κάνετε μόνοι σας την αντιστοίχιση ακροδεκτών και την διαμόρφωση του κυκλώματος. Αντιστοιχήστε τις εισόδους x[0] και x[1] στους διακόπτες push-button SW1 και SW2 και τις εξόδους y[0], y[1], y[2] και y[3] στα led L1, L2, L3 και L4, κάνοντας τις απαραίτητες τροποποιήσεις στον κώδικα σε VHDL. Για διευκόλυνσή σας συμπληρώστε τον πίνακα που ακολουθεί:

x[0] → Pin ..... (SW1)	**y[0]** → Pin ..... (L1)
x[1] → Pin ..... (SW2)	**y[1]** → Pin ..... (L2)
	y[2] → Pin ..... (L3)
gnd → Pin 141	**y[3]** → Pin ..... (L4)

Αφού διαμορφώσετε το κύκλωμα, ελέγξτε τη σωστή λειτουργία του.

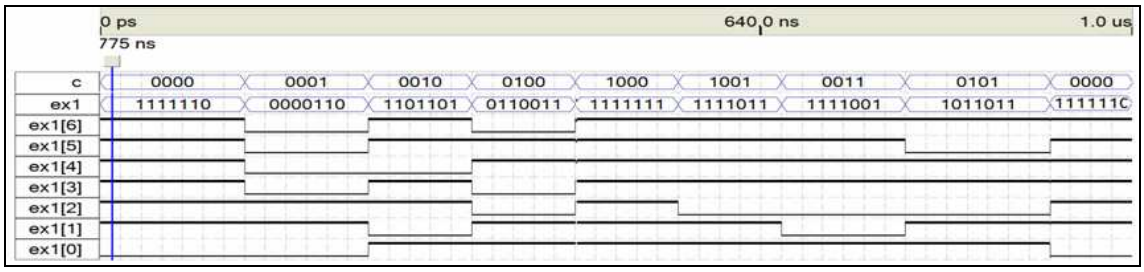
9.2.2 Αποκωδικοποιητής BCD σε 7segment

Η περιγραφή του κυκλώματος του μετατροπέα γίνεται με την ανάλυση της συμπεριφοράς του. Η περιγραφή του κυκλώματος με βάση τη δομή του είναι πολύπλοκη και θα μας υποχρέωνε στη συγγραφή πολλών γραμμών κώδικα, με μεγάλη πιθανότητα λάθους. Η δυνατότητα περιγραφής του κυκλώματος με βάση τη συμπεριφορά, έστω κι αν δεν γνωρίζουμε την εσωτερική δομή, μας διευκολύνει και αποτελεί σημαντικό πλεονέκτημα της γλώσσας VHDL. Στον κώδικα που ακολουθεί, ο μετατροπέας αποτελείται από μία είσοδο c η οποία έχει μέγεθος 4 bits, και μία έξοδο ex1 μεγέθους 7 bits. Το κάθε bit της εξόδου αντιστοιχεί σε ένα led του ενδείκτη επτά τομέων.

Ο μετατροπέας αυτός μπορεί να απεικονίσει τιμές από το 0 ως το 9, αφού θέλουμε την έξοδο σε δεκαδική μορφή. Στην είσοδο η τιμές που δίνουμε είναι από 0000 ως 1001 για να πάρουμε τα επιθυμητά αποτελέσματα ενώ για οποιαδήποτε άλλη τιμή στην είσοδο, στην έξοδο παίρνουμε μια παύλα (-).

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

ENTITY seg_7 IS
PORT(c : IN std_logic_vector (3 DOWNTO 0);
      ex1: OUT std_logic_vector(6 DOWNTO 0));
END seg_7;

ARCHITECTURE behaviour OF seg_7 IS
BEGIN
  WITH c SELECT
   ex1<= "1111110" WHEN "0000",
         "0000110" WHEN "0001",
         "1101101" WHEN "0010",
         "1111001" WHEN "0011",
         "0110011" WHEN "0100",
         "1011011" WHEN "0101",
         "1011111" WHEN "0110",
         "1110000" WHEN "0111",
         "1111111" WHEN "1000",
         "1111011" WHEN "1001",
         "0000001" WHEN OTHERS;
 END behaviour;
```

Στην προσομοίωση του κυκλώματος που ακολουθεί βλέπουμε την αντιστοίχηση εισόδων-εξόδων. Τα αποτελέσματα στην έξοδο επαληθεύουν τον πίνακα αληθείας του μετατροπέα.



Σχήμα 9.4 *Προσομοίωση Μετατροπέα BCD σε 7segment*

9.2.2.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Η αντιστοίχηση των ακροδεκτών γίνεται σύμφωνα με τον παρακάτω πίνακα.

c[0] → Pin 51 (SW4)	**ex1[0]** → Pin 31 (g)
c[1] → Pin 49 (SW3)	**ex1[1]** → Pin 30 (f)
c[2] → Pin 48 (SW2)	**ex1[2]** → Pin 29 (e)
c[3] → Pin 47 (SW1)	**ex1[3]** → Pin 28 (d)
	ex1[4] → Pin 27 (c)
	ex1[5] → Pin 26 (b)
	ex1[6] → Pin 23 (a)

Η 4 bit είσοδος c αποτελεί το σήμα BCD της εισόδου, όπου αντιστοιχίζουμε τους τέσσερις push-button διακόπτες SW1, SW2, SW3 και SW4. Οι ακροδέκτες ex1[0] ως ex1[6] είναι τα 7 σήματα της εξόδου του μετατροπέα, ένα για κάθε φωτεινό τομέα. Η έξοδος ex1[6] αντιστοιχεί στο γράμμα 'a' (σχήμα 9.2 β), το ex1[5] στο 'b' και ο τελευταίος ακροδέκτης ex1[0] στο 'g'.

Το σήμα της εξόδου ex1[0…6] αντιστοιχίζεται στους ακροδέκτες που οδηγούν τους ενδείκτες επτά τομέων. Την πληροφορία αυτή την αναζητούμε στο σχηματικό διάγραμμα του αναπτυξιακού κυκλώματος. Ο ενδείκτης 7 τομέων που χρησιμοποιούμε στο LP-2900 είναι κοινής ανόδου.

Σημείωση για περαιτέρω μελέτη: Στην αναπτυξιακή πλακέτα υπάρχουν συνολικά 6 ενδείκτες 7 τομέων. Το ποιος θα είναι ενεργοποιημένος κάθε φορά επιλέγεται από την έξοδο ενός αποκωδικοποιητή 3 σε 8 (74138). Την είσοδο του αποκωδικοποιητή αποτελούν τα σήματα DE1, DE2 και DE3 τα οποία είναι αντιστοιχημένα στα pins 33, 36 και 37 αντίστοιχα. Άρα, ανάλογα με τις τιμές που παίρνουν τα σήματα αυτά επιλέγεται ο ενδείκτης που θα είναι ενεργός κάθε φορά. Τα σήματα αυτά μπορούν να παίρνουν τιμές μέσα στο τμήμα architecture του VHDL κώδικα ή να αντιστοιχηθούν με διακόπτες ώστε να επιλέγεται δυναμικά ο ενδείκτης.

Στον κώδικα που ακολουθεί φαίνονται οι αλλαγές που έγιναν ώστε να επιλέγεται δυναμικά, μέσω 3 διακοπτών, ο ενεργός κάθε φορά ενδείκτης. Τις εξόδους de1, de2, de3 αντιστοιχήστε τις στα pin 33,36,37 αντίστοιχα και τις εισόδους s1,s2,s3 στους push-button διακόπτες SW8, SW7, SW6 αντίστοιχα. Κατόπιν διαμορφώστε το κύκλωμα και ελέγξτε τη σωστή λειτουργία του.

```
.....................................................
ENTITY seg_7 IS
  PORT(c : IN std_logic_vector (3 DOWNTO 0);
        ex1: OUT std_logic_vector(6 DOWNTO 0);
        de1,de2,de3:OUT bit;
        s1,s2,s3:IN bit);
END seg_7;

ARCHITECTURE behavior OF seg_7 IS
  BEGIN
  de1<=s1;
  de2<=s2;
  de3<=s3;
   WITH c SELECT
     ex1<= "1111110" WHEN "0000",
.....................................................
```

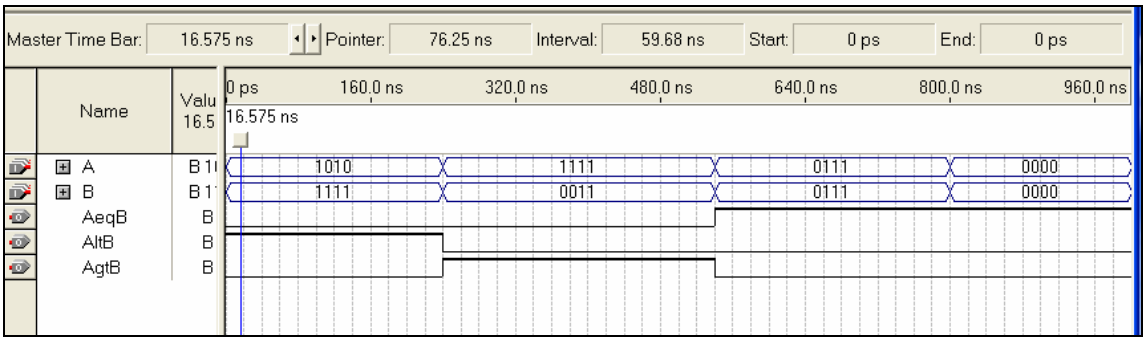
9.2.3 Συγκριτές

Να υλοποιήσετε στο Quartus II το κύκλωμα συγκριτή τεσσάρων bits, που περιγράφεται από τον παρακάτω κώδικα σε γλώσσα VHDL.

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

    ENTITY compare IS
         PORT(A,B:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                  AeqB, AgtB, AltB: OUT STD_LOGIC);
    END compare;

    ARCHITECTURE behaviour OF compare IS
    BEGIN
         AeqB<='1' WHEN A=B ELSE '0';
         AgtB<='1' WHEN A>B ELSE '0';
         AltB<='1' WHEN A<B ELSE '0';
    END behaviour;
```

1. Ποιοι είναι οι ακροδέκτες του κυκλώματος και ποια ακριβώς η λειτουργία που επιτελεί το κύκλωμα;

2. Να εκτελέσετε τη λειτουργική προσομοίωση του κυκλώματος, χρησιμοποιώντας αρχείο εισόδου με τις παρακάτω κυματομορφές:



Σχήμα 9.5 *Λειτουργική προσομοίωση του συγκριτή*

Προσέξτε στο παραπάνω αρχείο πως ορίσαμε τις εισόδους A και B σύμφωνα με τις αριθμητικές τους τιμές. Για το σκοπό αυτό εισάγουμε στον Node finder τις συνολικές μεταβλητές εισόδου A και B και αφού επιλέξουμε τις περιοχές ορισμού τους δίνουμε Numeric Value.

3. Να κάνετε την αντιστοίχηση ακροδεκτών. Τις εισόδους A[0], A[1], A[2], A[3] να τις αντιστοιχήσετε στους διακόπτες τύπου dip SW12, SW11, SW10, SW9 αντίστοιχα, τις εισόδους B[0], B[1], B[2], B[3] να τις αντιστοιχήσετε στους διακόπτες τύπου dip SW20, SW19, SW18, SW17 αντίστοιχα και τις εξόδους AeqB, AltB, AgtB στα led L1, L2, L3 αντίστοιχα, κάνοντας τις απαραίτητες τροποποιήσεις στον κώδικα σε VHDL. Συμπληρώστε τον παρακάτω πίνακα για διευκόλυνσή σας:

A[0] → Pin ..... (SW12)	**B[0]** → Pin ..... (SW20)
A[1] → Pin ..... (SW11)	**B[1]** → Pin ..... (SW19)
A[2] → Pin ..... (SW10)	**B[2]** → Pin ..... (SW18)
A[3] → Pin ..... (SW9)	**B[3]** → Pin ..... (SW17)
gnd → Pin 141	
	AeqB → Pin ..... (L1)
	AltB → Pin ..... (L2)
	AgtB → Pin ..... (L3)

4. Διαμορφώστε το ολοκληρωμένο και ελέγξτε τη σωστή λειτουργία του συγκριτή 4 bit.

ΕΡΓΑΣΤΗΡΙΟ 10

ΑΘΡΟΙΣΤΕΣ
ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ ΜΕ ΧΡΗΣΗ ΥΠΟΚΥΚΛΩΜΑΤΩΝ

10.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

10.1.1 Υποκυκλώματα

Η VHDL είναι μια γλώσσα που επιτρέπει την ιεραρχική σχεδίαση κυκλωμάτων. Έτσι, ο χρήστης μπορεί να περιγράψει ένα σύνθετο κύκλωμα με βάση τα υποκυκλώματα που το αποτελούν. Όπως αναφέραμε, τα υποκυκλώματα που συμπεριλαμβάνονται σε μια σχεδίαση ονομάζονται *συνιστώσες* (components). Αυτές οι συνιστώσες είναι μικρότερα κυκλώματα, σε VHDL, που έχουν ήδη περιγραφεί και μπορούν να κληθούν ως αρχεία βιβλιοθήκης. Τέτοια αρχεία μπορεί να είναι έτοιμα αρχεία από κάποιες υπάρχουσες βιβλιοθήκες, ή μπορεί να τα σχεδιάσει ο ίδιος ο χρήστης. Τα κύρια χαρακτηριστικά της σχεδίασης με υποκυκλώματα είναι η δήλωση των συνιστωσών που περιέχονται στο κύκλωμα, καθώς και η περιγραφή του τρόπου με τον οποίο συνδέονται μεταξύ τους.

Τις συνιστώσες πρέπει να τις δηλώνουμε. Με την δήλωση της κάθε συνιστώσας καθορίζουμε το όνομα της καθώς και τα ονόματα των εισόδων και των εξόδων της. Η γενική μορφή μιας τέτοιας δήλωσης παρουσιάζεται παρακάτω.

```
COMPONENT όνομα συνιστώσας
    [ GENERIC (όνομα παραμέτρου : integer := τιμή{ ;
                όνομα παραμέτρου : integer := τιμή }) ; ]
      PORT (όνομα ακροδέκτη : mode τύπος ;
            όνομα ακροδέκτη : mode τύπος ) ;
END COMPONENT;
```

Σχήμα 10.1 *Γενική Μορφή Δήλωσης Συνιστώσας*

Η δήλωση μιας συνιστώσας μπορεί να γίνει ή στην περιοχή δήλωσης της αρχιτεκτονικής ή σε μία δήλωση πακέτου (βλέπε παρακάτω). Στη συνέχεια, στο κύριο σώμα της αρχιτεκτονικής, πρέπει να δημιουργήσουμε *στιγμιότυπα* (instants) για τις συνιστώσες που έχουμε δηλώσει, τα οποία λειτουργούν ως υποκυκλώματα της σχεδίασής μας. Η δημιουργία των στιγμιότυπων γίνεται πάντα μετά τη δήλωση μιας συνιστώσας. Τα στιγμιότυπα είναι της μορφής :

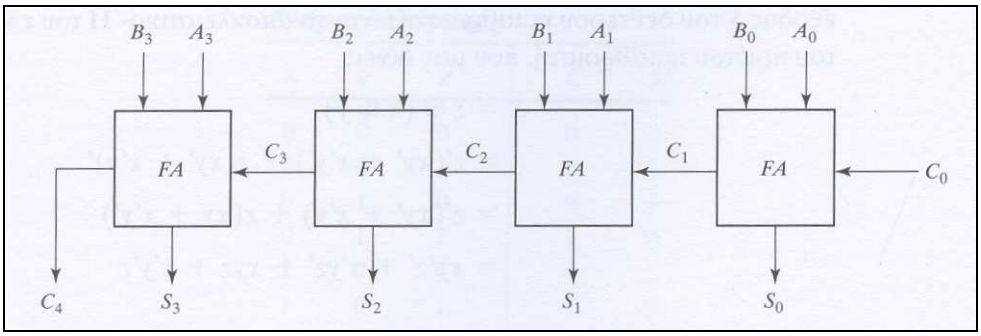
όνομα στιγμιότυπου: **όνομα συνιστώσας** PORT MAP (**ονόματα σημάτων**) ;

10.1.2 Χρήση Υποκυκλωμάτων (Components)

Όταν χρησιμοποιούμε υποκυκλώματα, ξεκινάμε πρώτα με την δήλωση των στοιχείων (components) που θα χρησιμοποιήσουμε. Συνεχίζουμε με την δήλωση των σημάτων που θα χρησιμοποιήσουμε για τις εσωτερικές συνδέσεις των στοιχείων, καθώς και βιβλιοθηκών ή

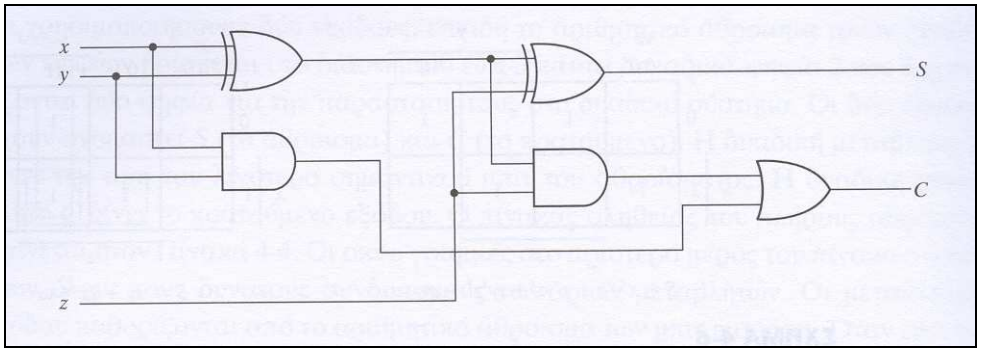
πακέτων. Και τέλος, περιγράφουμε τον τρόπο σύνδεσης του κάθε στοιχείου της σχεδίασης με τα υπόλοιπα.

Στην εργαστηριακή άσκηση που ακολουθεί θα δημιουργήσουμε ένα ιεραρχικό κύκλωμα πλήρους αθροιστή δύο αριθμών των τεσσάρων bits, με τη βοήθεια στιγμιοτύπων του απλού πλήρους αθροιστή 2-bits. Όπως είναι γνωστό ένας πλήρης αθροιστής τεσσάρων bits μπορεί να δημιουργηθεί με την χρήση τεσσάρων αθροιστών 2-bits, σε συνδεσμολογία όπως αυτή του σχ. 10.2.



Σχήμα 10.2 *Σύνδεση βαθμίδων πλήρη αθροιστή 2-bits για την υλοποίηση αθροιστή δύο αριθμών των τεσσάρων bits*.

Το κύκλωμα του πλήρη αθροιστή μπορεί να υλοποιηθεί με πύλες, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 10.3 *Πλήρης αθροιστής 2-bits με πύλες.*

10.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

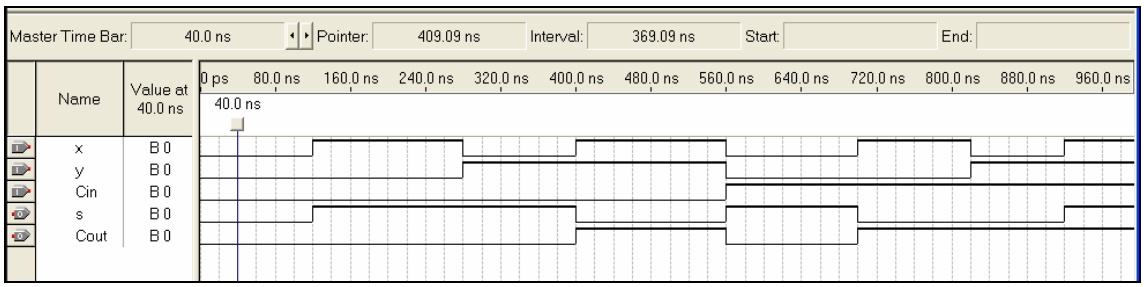
10.2.1 Σχεδίαση του πλήρη αθροιστή δύο bits

Ο παρακάτω κώδικας VHDL υλοποιεί έναν πλήρη αθροιστή 2-bits με δομική σχεδίαση. Θυμίζουμε ότι η δομική σχεδίαση στη VHDL αναφέρεται σε σχεδίαση σε επίπεδο λογικών πυλών.

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

```
ENTITY fulladder1 IS
PORT(Cin,x,y :IN STD_LOGIC;
    s, Cout  :OUT STD_LOGIC);
END fulladder1;

ARCHITECTURE structural OF fulladder1 IS
BEGIN
    s<=x XOR y XOR Cin;
    Cout<=(x AND y) OR (Cin AND x) OR (Cin AND y);
END Structural;
```

Στο παρακάτω σχήμα φαίνεται η προσομοίωση του πλήρη αθροιστή 2-bits στο Quartus II.



Σχήμα 10.4 *Προσομοίωση του πλήρη αθροιστή 2-bits*

10.2..2 Πλήρης αθροιστής αριθμών 4-bits με υποκυκλώματα

Ο παρακάτω κώδικας υλοποιεί τον πλήρη αθροιστή δύο αριθμών των τεσσάρων bits με τη χρήση στιγμιοτύπων του πλήρη αθροιστή 2-bits, που σχεδιάσαμε στην προηγούμενη παράγραφο.

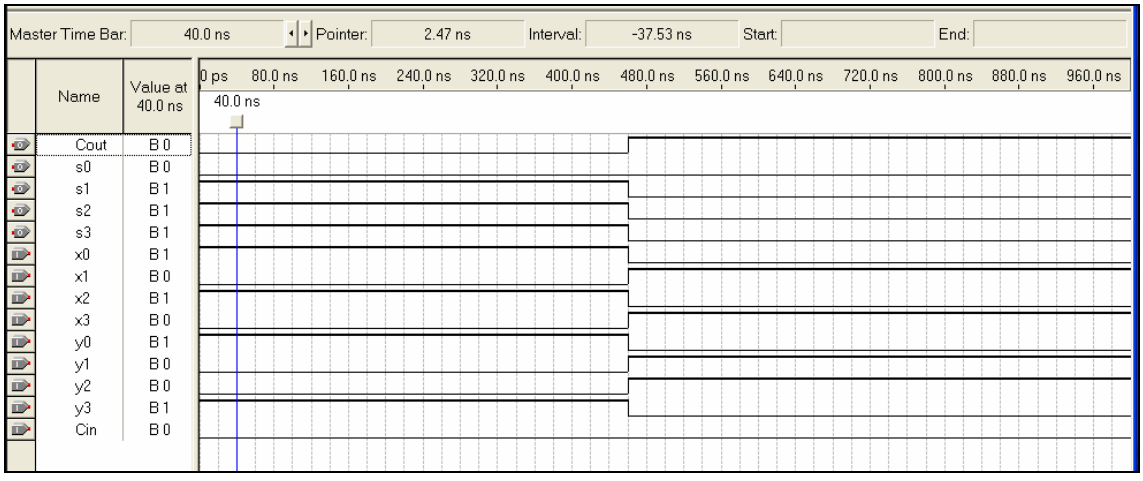
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder4 IS
    PORT(Cin, x3,x2,x1,x0, y3,y2,y1,y0:IN std_logic;
    s3,s2,s1,s0,Cout:OUT std_logic);
END adder4;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1,c2,c3: std_logic;

    COMPONENT fulladder1
    PORT(Cin,x,y: IN std_logic;
            s, Cout :OUT std_logic);
    END COMPONENT;
```

```
BEGIN
    Stage0: fulladder1 PORT MAP(Cin,x0,y0,s0,c1);
    Stage1: fulladder1 PORT MAP (c1,x1,y1,s1,c2);
    Stage2: fulladder1 PORT MAP (c2,x2,y2,s2,c3);
    Stage3: fulladder1 PORT MAP (c3,x3,y3,s3,Cout);
END Structure;
```

Το σώμα της οντότητας δεν αναφέρεται καθόλου στα υποκυκλώματα. Θα συναντήσουμε τα υποκυκλώματα για πρώτη φορά στο χώρο δηλώσεων, μέσα στο σώμα της αρχιτεκτονικής. Με τη λέξη “component” δηλώνουμε την χρήση του υποκυκλώματος με όνομα fulladder1 (Component fulladder1). Το αρχείο fulladder1 πρέπει να βρίσκεται αποθηκευμένο στον ίδιο φάκελο, όπου και το project adder4. Εκτός από το όνομα της οντότητας χρειάζεται να γράψουμε και τα ονόματα των ακροδεκτών του υποκυκλώματος, προσδιορίζοντας αν είναι είσοδοι ή έξοδοι, καθώς και το μέγεθος τους. Τόσο τα ονόματα των ακροδεκτών όσο και τα υπόλοιπα χαρακτηριστικά τους (τύπος και μέγεθος), πρέπει να είναι ίδια με αυτά που είχαν δηλωθεί στο αρχικό κύκλωμα, που αποθηκεύσαμε ως βιβλιοθήκη.

Στο τμήμα της δήλωσης, εκτός από τα υποκυκλώματα θα πρέπει να δηλώσουμε και τα σήματα που θα χρησιμοποιήσουμε για την εσωτερική διασύνδεση των υποκυκλωμάτων. Στην προκειμένη περίπτωση έχουμε δηλώσει τα σήματα c1, c2, c3.

Ξεκινώντας το κύριο σώμα της αρχιτεκτονικής παρατηρούμε την χρήση τεσσάρων στιγμιοτύπων με ονόματα stage0, stage1, stage2, stage3. Με τη βοήθεια αυτών των στιγμιοτύπων καλούμε τα υποκυκλώματα και αντιστοιχίζουμε τους ακροδέκτες τους με τους ακροδέκτες του ανώτερου ιεραρχικά κυκλώματος. Εδώ, θα πρέπει να επισημάνουμε την μεγάλη σημασία που έχει η σειρά των ακροδεκτών στα στιγμιότυπα, η οποία πρέπει να είναι ίδια με αυτήν που έχουμε δηλώσει στο «component».



Σχήμα 10.5 *Προσομοίωση του αθροιστή 4-bits.*

10.2.2.1 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Να κάνετε την αντιστοίχηση ακροδεκτών. Τις εισόδους x3, x2, x1, x0 να τις αντιστοιχίσετε στους διακόπτες τύπου dip SW9, SW10, SW11, SW12 αντίστοιχα, τις εισόδους y3, y2, y1, y0 να τις αντιστοιχίσετε στους διακόπτες τύπου dip SW17, SW18, SW19, SW20 αντίστοιχα, την

είσοδο κρατουμένου Cin να την αντιστοιχήσετε στο διακόπτη τύπου push button SW1 και τις εξόδους Cout, s3, s2, s1, s0 να τις αντιστοιχίσετε στα led L1, L2, L3, L4, L5 αντίστοιχα, κάνοντας τις απαραίτητες τροποποιήσεις στον κώδικα σε VHDL. Συμπληρώστε τον παρακάτω πίνακα για διευκόλυνσή σας:

x3 → Pin ..... (SW9)	**y3** → Pin ..... (SW17)
x2 → Pin ..... (SW10)	**y2** → Pin ..... (SW18)
x1 → Pin ..... (SW11)	**y1** → Pin ..... (SW19)
x0 → Pin ..... (SW12)	**y0** → Pin ..... (SW20)
gnd → Pin 141	**s3** → Pin ..... (L2)
	s2 → Pin ..... (L3)
	s1 → Pin ..... (L4)
	s0 → Pin ..... (L5)
Cin → Pin ..... (SW1)	**Cout** → Pin ..... (L1)

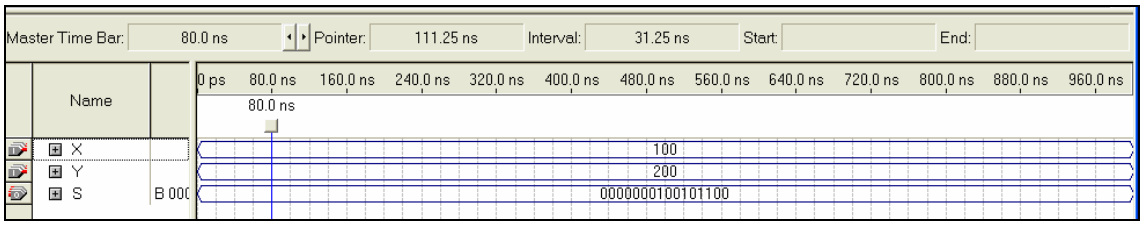
Κατόπιν, διαμορφώστε το ολοκληρωμένο και ελέγξτε τη σωστή λειτουργία του πλήρους αθροιστή 4 bit.

10.2.3 Χρήση αριθμητικών βιβλιοθηκών

Ο παρακάτω κώδικας περιγράφει έναν αθροιστή προσημασμένων αριθμών με εύρος 16-bits, ο οποίος σχεδιάστηκε με τη χρήση της αριθμητικής βιβλιοθήκης ieee.std_logic_signed.all.

Παρατηρούμε ότι η λειτουργία του αθροιστή περιγράφεται απλώς με τον αριθμητικό τελεστή «+».

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

ENTITY adder16 IS
    PORT(X,Y :IN std_logic_vector(15 DOWNTO 0);
             S   :OUT std_logic_vector (15 DOWNTO 0) );
END adder16;

ARCHITECTURE arithm OF adder16 IS
BEGIN
    S<=X+Y;
END arithm;
```



Σχήμα 10.6 *Προσομοίωση του αθροιστή 16-bits*

ΕΡΓΑΣΤΗΡΙΟ 11

ΔΟΜΕΣ ΕΝΤΟΛΩΝ ΣΕ VHDL
ΒΑΣΙΚΑ ΑΚΟΛΟΥΘΙΑΚΑ ΚΥΚΛΩΜΑΤΑ

11.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

11.1.1 Τύποι Εντολών Στη VHDL

Οι εντολές στην VHDL χωρίζονται σε:

- Εντολές **Σύμφωνης αντιστοίχησης**
- Εντολές **Ακολουθιακής αντιστοίχησης**.

Οι εντολές ακολουθιακής αντιστοίχησης λέγονται **Sequential Assignment Statements**, και η σειρά γραφής τους μέσα στον κώδικα έχει σημασία, αφού τα σήματα αλλάζουν τιμή με τη σειρά που εκτελούνται αυτές οι εντολές. Αντιθέτως, με τις εντολές σύμφωνης αντιστοίχησης η εκχώρηση των τιμών στα σήματα γίνεται μετά το πέρας της διαδικασίας (process) μέσα στην οποία εκτελούνται οι εντολές, οπότε η σειρά γραφής τους δεν είναι καθοριστική. Εντολές σύμφωνης αντιστοίχησης είναι οι εξής :

IF, **CASE** και οι εντολές βρόχων (**FOR** και **WHILE**).

11.1.2 Δομή Διαδικασίας (PROCESS)

Η δομή διαδικασίας χρησιμοποιείται για να διαχωρίζει τις εντολές ακολουθιακής αντιστοίχησης από αυτές της σύμφωνης αντιστοίχησης. Η εντολή **PROCESS** γράφεται στο κύριο σώμα της αρχιτεκτονικής και μέσα της περιέχονται άλλες εντολές (IF, CASE, FOR και WHILE). Η PROCESS συνοδεύεται πάντα από μια παρένθεση μέσα στην οποία περιέχεται η λεγόμενη "**λίστα ευαισθησίας**" (sensitivity list). Η λίστα ευαισθησίας περιέχει ένα ή περισσότερα σήματα, με κάθε αλλαγή των οποίων ενεργοποιείται η PROCESS και κατ' επέκταση και οι εντολές που περιέχει. Τότε, οι εντολές που περιλαμβάνονται σ' αυτήν εκτελούνται με τη σειρά. Όταν υπολογιστούν οι τιμές όλων των σημάτων που περιλαμβάνονται στη διαδικασία, τότε οι τελικές αντιστοιχήσεις παράγουν ορατό αποτέλεσμα στα σήματα εξόδου.

Η γενική μορφή της δομής διαδικασίας φαίνεται παρακάτω.

```
PROCESS (ονόματα σημάτων)
    BEGIN
            IF…
            CASE…
            FOR…
            WHILE…
END PROCESS;
```

Η δομή διαδικασίας μπορεί να χρησιμοποιηθεί για να περιγράψει τόσο συνδυαστικά όσο και ακολουθιακά κυκλώματα, λόγω της ευκολίας που μας παρέχει στην περιγραφή του κυκλώματος.

11.1.3 Η Εντολή IF

Η εντολή IF λειτουργεί ως μια εντολή διακλάδωσης υπό συνθήκη και η γενική μορφή της δίνεται στο σχήμα που ακολουθεί.

```
IF έκφραση THEN
    Εντολή ;
ELSEIF έκφραση THEN
    Εντολή ;
END IF;
```

Η εντολή αυτή βρίσκεται πάντα μέσα σε μία εντολή διαδικασίας (PROCESS). Για να γίνει πιο κατανοητή η χρήση αυτού του είδους εντολών ακολουθεί ένα παράδειγμα διαδικασίας που περιγράφει έναν πολυπλέκτη δύο προς ένα.

```
Process (sel, x1, x2)
  BEGIN
    IF sel='0' THEN
       F<=x1;
    ELSE
       F<=x2;
    END IF;
END process;
```

Στο παραπάνω παράδειγμα με sel συμβολίζουμε τον διακόπτη επιλογής του πολυπλέκτη, με x1 και x2 τις δύο εισόδους του και με F την έξοδο του. Παρατηρούμε ότι τα σήματα sel, x1 και x2 περιέχονται στην λίστα ευαισθησίας της διαδικασίας. Αυτό σημαίνει ότι κάθε φορά που θα έχουμε αλλαγή της κατάστασης τους θα τίθεται σε λειτουργία η εντολή IF και θα έχουμε αλλαγή στην κατάσταση της εξόδου. Στην συγκεκριμένη περίπτωση η χρήση της IF είναι να διαχωρίζει τις πιθανές τιμές του σήματος sel. Αν sel ισούται με μηδέν στην έξοδο F παίρνουμε την τιμή της εισόδου x1, αλλιώς αν sel ισούται με ένα στην έξοδο F παίρνουμε την τιμή της εισόδου x2.

11.1.4 Παράδειγμα διαδικασίας (Process)

Στα παραδείγματα που ακολουθούν χρησιμοποιούμε την εντολή IF-THEN-ELSE της γλώσσας VHDL, με την οποία μπορούμε να ορίσουμε τις τιμές που παίρνουν κάποια σήματα, υπό συνθήκη. Η εντολή αυτή είναι ιδιαίτερα χρήσιμη, αλλά ας προσέξουμε ότι πρέπει να την τοποθετούμε πάντα μέσα σε μια δομή, που καλείται «διαδικασία», και ορίζεται με τη λέξη-

κλειδί PROCESS. Όπως αναφέρθηκε, η λέξη PROCESS σε ένα πρόγραμμα VHDL ακολουθείται από ένα σύνολο σημάτων, μέσα σε παρενθέσεις, που ονομάζονται «λίστα ευαισθησίας». Σε ένα συνδυαστικό κύκλωμα, η λίστα ευαισθησίας περιλαμβάνει όλα τα σήματα εισόδου που χρησιμοποιούνται στη διαδικασία. Όλος ο κώδικας που περιλαμβάνεται σε μια διαδικασία μεταφράζεται από τον μεταφραστή της γλώσσας σε λογικές εξισώσεις που επιτελούν τη συνάρτηση που περιγράφει η διαδικασία.

Στην περίπτωση ενός πολυπλέκτη δύο εισόδων, μιας εξόδου και μιας γραμμής επιλογής, η διαδικασία περιλαμβάνει μια εντολή IF-THEN-ELSE που περιγράφει τη συνάρτηση του πολυπλέκτη. Ο κατάλογος ευαισθησίας περιλαμβάνει τα σήματα w0, w1 (κανάλια εισόδου) και τη γραμμή επιλογής s.

Μέσα σε μια διαδικασία μπορεί να υπάρχουν διάφορες εντολές. Όταν αλλάξει τιμή κάποιο από τα σήματα που περιλαμβάνονται στον κατάλογο ευαισθησίας λέμε ότι η διαδικασία (PROCESS) ενεργοποιείται και τότε, οι εντολές που περιλαμβάνονται σ' αυτήν εκτελούνται με τη σειρά. Όταν υπολογιστούν οι τιμές όλων των σημάτων που περιλαμβάνονται στη διαδικασία, τότε οι τελικές αντιστοιχήσεις παράγουν ορατό αποτέλεσμα στα σήματα εξόδου.

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

ENTITY mux2to1 IS
    PORT ( w0, w1, s  : IN     STD_LOGIC ;
           f          : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        IF s = '0' THEN
            f <= w0 ;
        ELSE
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Σχήμα 11.1 *Περιγραφή πολυπλέκτη 2:1 με τη βοήθεια εντολών IF-THEN-ELSE μέσα σε μια διαδικασία (PROCESS).*

11.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

11.2.1 Βασικά Ακολουθιακά Κυκλώματα Εισαγωγή

Σε όλες τις προηγούμενες ασκήσεις τα κυκλώματα που σχεδιάσαμε ήταν συνδυαστικά κυκλώματα. Συνδυαστικά λέγονται τα κυκλώματα που οι έξοδοι τους σε κάθε χρονική στιγμή εξαρτώνται μόνο από τις τιμές των εισόδων τους εκείνη την στιγμή και όχι από την κατάσταση του κυκλώματος πριν αυτές εφαρμοσθούν.

Σ' αυτήν την ενότητα θα ασχοληθούμε με κυκλώματα τα οποία χαρακτηρίζονται ως ακολουθιακά. Σ' αυτά τα κυκλώματα οι τιμές των εξόδων σε κάθε χρονική στιγμή δεν εξαρτώνται μόνο από τις τιμές που έχουν οι είσοδοί τους εκείνη την χρονική στιγμή, αλλά και από τις τιμές των εξόδων σε προηγούμενες χρονικές στιγμές. Τα ακολουθιακά κυκλώματα χωρίζονται σε δύο κατηγορίες: στα **ασύγχρονα** και στα **σύγχρονα**. Σύγχρονο λέγεται ένα ακολουθιακό κύκλωμα το οποίο αλλάζει κατάσταση σε συγκεκριμένη χρονική στιγμή, με τον παλμό ενός ρολογιού. Αντίθετα, όταν ένα ακολουθιακό κύκλωμα αλλάζει κατάσταση με βάση την σειρά που αλλάζουν τιμές οι είσοδοι, λέγεται ασύγχρονο ακολουθιακό κύκλωμα.

Τα κυκλώματα που θα περιγράψουμε σ' αυτή την ενότητα είναι ένας μανδαλωτής τύπου D (D-latch), ένα D flip flop, ένας καταχωρητής εύρους 8 bits και ένας απαριθμητής 4 bits.

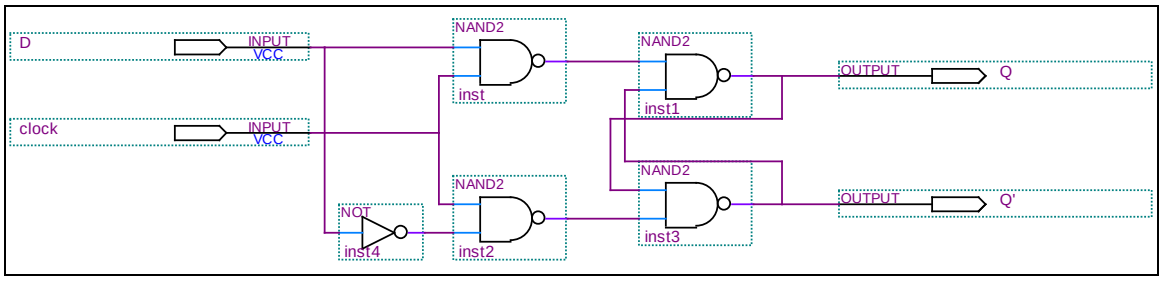
11.2.2 Ιδιότητες σημάτων (attributes) στη VHDL

Στο παρακάτω παράδειγμα υλοποιούμε ένα D-FF που διεγείρεται με θετικό μέτωπο παλμού ρολογιού. Χρησιμοποιούμε πάλι μια διαδικασία (PROCESS), η οποία τώρα περιέχει στον κατάλογο ευαισθησίας μόνο το ωρολογιακό σήμα, αφού αυτό είναι το μόνο που μπορεί να προκαλέσει αλλαγή στην έξοδο Q. Η εντολή IF-THEN-ELSE περιέχει εκτός από σήματα και μια ιδιότητα σήματος, την **EVENT**. Εδώ, η ιδιότητα EVENT αναφέρεται στο σήμα Clock και συντάσσεται ως **Clock'EVENT**. Η σύνταξη αυτή αναφέρεται σε οποιαδήποτε αλλαγή της κατάστασης του σήματος Clock. Ο συνδυασμός της συνθήκης **Clock'EVENT** και **Clock='1'** σημαίνει ότι η κατάσταση του σήματος Clock έχει μόλις αλλάξει και έχει γίνει 1. Άρα, η συνθήκη αναφέρεται σε **θετικό μέτωπο** ωρολογιακού παλμού. Αφού, λοιπόν, η έξοδος Q αλλάζει μόνον ως αποτέλεσμα ενός θετικού μετώπου παλμού του ρολογιού, ο κώδικας που παρουσιάζουμε περιγράφει ένα D Flip-Flop με θετική διέγερση μετώπου (positive edge triggering).

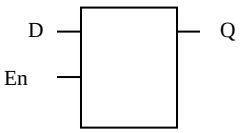
11.2.3 Μανδαλωτής τύπου D

Σ' αυτή την άσκηση θα περιγράψουμε έναν μανδαλωτή τύπου D (D-latch). Οι μανδαλωτές ανήκουν στα ασύγχρονα ακολουθιακά κυκλώματα. Τόσο οι μανδαλωτές όσο και τα flip flops, που θα περιγραφούν παρακάτω, έχουν την δυνατότητα της αποθήκευσης μιας τιμής. Οι μανδαλωτές τύπου D αποτελούνται από μία είσοδο δεδομένων, μία είσοδο ενεργοποίησης (Enable) και μία έξοδο Q. Η έξοδος ακολουθεί την είσοδο του μανδαλωτή όταν η είσοδος ενεργοποίησης βρίσκεται σε κατάσταση High (enable=1). Όσο η είσοδος ενεργοποίησης είναι σε κατάσταση low η τιμή της εισόδου δεν επηρεάζει την έξοδο, αλλά αυτή παραμένει στην προηγούμενη κατάσταση. Στο σχήμα που ακολουθεί φαίνεται το κύκλωμα ενός τέτοιου μανδαλωτή, το γραφικό του σύμβολο και ο πίνακας αληθείας του.

Όπως βλέπουμε στον πίνακα αληθείας του κυκλώματος, όταν το enable βρίσκεται σε κατάσταση '1', στην έξοδο του μανδαλωτή παίρνουμε την τιμή που έχει η είσοδος D ενώ όταν βρίσκεται σε κατάσταση '0', ο μανδαλωτής κατά κάποιον τρόπο παγώνει, αφού στην έξοδο εμφανίζει την τελευταία τιμή που είχαμε στην είσοδο D. Σ' αυτή την κατάσταση θα παραμείνει, ώσπου το ρολόι παράγει πάλι κατάσταση '1', οπότε θα εμφανιστεί στην έξοδο η νέα τιμή που έχει η είσοδος.



α) Κύκλωμα μανδαλωτή



β) Σύμβολο

Enable	D	Q(t+1)
0	x	Q(t)
1	0	0
1	1	1

γ) Πίνακας Αληθείας

Σχήμα 11.2 *Μανδαλωτής Τύπου D*

Η υλοποίηση του μανδαλωτή θα γίνει με την χρήση της γλώσσας VHDL, όπως και στις προηγούμενες ασκήσεις. Όπως παρατηρούμε στον κώδικα που ακολουθεί, το κύκλωμα έχει το όνομα part2 και αποτελείται από δύο εισόδους και μία έξοδο, όλες μεγέθους 1 bit.

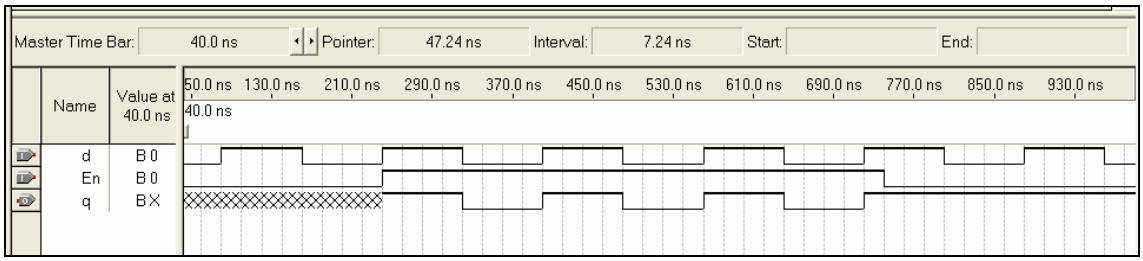
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dlatch1 IS
PORT (d,En: IN std_logic;
        q: OUT std_logic;
      gnd: OUT bit);
END dlatch1;

ARCHITECTURE behaviour OF dlatch1 IS
BEGIN
  gnd<='1';
  PROCESS (d, En)
  BEGIN
        IF En='1' THEN
              q<=d;
        END IF;
  END process;
END behaviour;
```

Στο σώμα της αρχιτεκτονικής περιέχεται μία διαδικασία (Process). Μια διαδικασία συνοδεύεται πάντα από μία παρένθεση που περιέχει την «λίστα ευαισθησίας». Η λίστα αυτή περιέχει σήματα του κυκλώματος, είτε αυτά είναι οι είσοδοι του, είτε καλώδια που συνδέουν εσωτερικά τμήματα του κυκλώματος. Το νόημα της ύπαρξης αυτής της λίστας είναι η άμεση ενεργοποίηση της διαδικασίας, όποτε έχουμε μεταβολή της κατάστασης των σημάτων που περιέχονται σ' αυτήν. Στην προκειμένη περίπτωση, οποιαδήποτε μεταβολή συμβεί στις εισόδους d και En, σκανδαλίζει την διαδικασία.

Η αρχή και το τέλος της διαδικασίας πρέπει να δηλώνονται. Ξεκινάει με την εντολή «**BEGIN**», αμέσως μετά την δήλωση της διαδικασίας και της λίστας ευαισθησίας της, και τελειώνει με την εντολή «**END PROCESS**». Η εντολή IF, που ακολουθεί τη Begin, ορίζει την τιμή της εξόδου Q ίση με την είσοδο D, σε περίπτωση που η είσοδος ενεργοποίησης είναι ίση με '1' (En='1'). Η εντολή ELSE παραλείπεται επειδή θέλουμε η έξοδος να συνεχίσει να

βρίσκεται στην παρούσα κατάσταση, όταν η είσοδος ενεργοποίησης γίνει ίση με ‘0’. Το κύκλωμα που μόλις περιγράψαμε ανήκει στα ασύγχρονα ακολουθιακά κυκλώματα, αφού οι αλλαγές στις εξόδους του δεν συγχρονίζονται από κάποιο ρολόι. Αυτού του είδους τα μάνδαλα (D-Latches) μπορούν εύκολα να μετατραπούν σε σύγχρονα, απλά αλλάζοντας την είσοδο enable με μια είσοδο clock.

Στο σχήμα 11.3 που ακολουθεί φαίνεται η προσομοίωση του D-Latch. Όταν η είσοδος En είναι ίση με ‘1’, η έξοδος Q παίρνει την τιμή που έχει η είσοδος D, ενώ όταν έχει την τιμή ‘0’ (En=0) η έξοδος Q διατηρεί την τελευταία τιμή που είχε, προτού η είσοδος ενεργοποίησης γίνει ‘0’.



Σχήμα 11.3 *Προσομοίωση D-Latch*

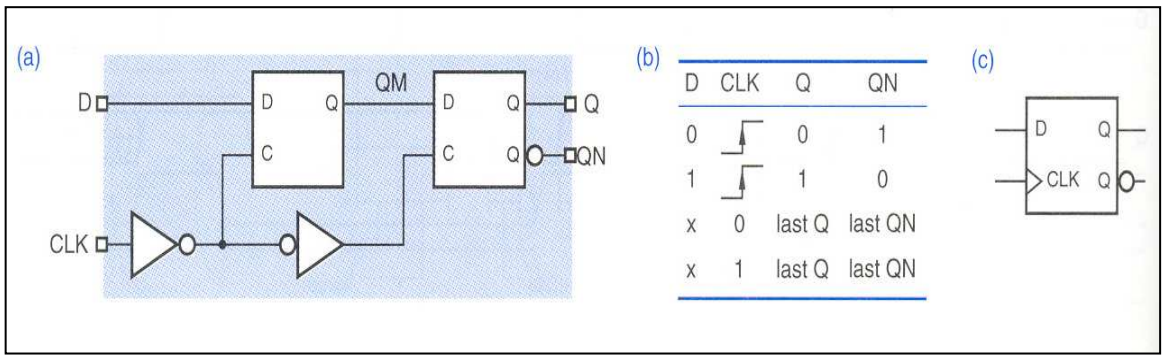
Η αντιστοίχηση των ακροδεκτών ακολουθεί την ίδια φιλοσοφία, όπως και στις προηγούμενες ασκήσεις. Οι είσοδοι En και d αντιστοιχίζονται στον Pulse Switch 3 (PS3) και στον διακόπτη τύπου push-button SW1, αντίστοιχα ενώ η έξοδος q στο led L1.

En → Pin 124 (PS3)	**q** → Pin 7 (L1)
d → Pin 47 (SW1)	**gnd** → Pin 141

Αφού κάνετε την αντιστοίχιση και τη διαμόρφωση του κυκλώματος, ελέγξτε στο αναπτυξιακό σύστημα τη σωστή λειτουργία του D-Latch.

11.2.4 Flip Flop τύπου D

Τα Flip Flop είναι κυκλώματα τα οποία αποτελούνται από μία είσοδο δεδομένων D, μία είσοδο ρολογιού clk και δύο εξόδους Q και QN. Τόσο τα Flip Flop όσο και οι μανδαλωτές που



ΣΧΗΜΑ 11.4 *α) Κύκλωμα D Flip Flop β) Πίνακας Αληθείας και γ) Σύμβολο D Flip Flop*

περιγράφηκαν προηγουμένως, αποτελούν τα πιο μικρά στοιχεία αποθήκευσης, χωρητικότητας 1 bit. Υπάρχουν δύο είδη Flip-Flops. Το πρώτο είδος είναι αυτά που η έξοδος τους επηρεάζεται από την είσοδο κατά το θετικό μέτωπο του παλμού του ρολογιού, δηλαδή η ανανέωση της εξόδου γίνεται την στιγμή της μετάβασης του ρολογιού από την κατάσταση Low στην κατάσταση High. Το δεύτερο είδος FF είναι αυτά που η έξοδος τους επηρεάζεται από την είσοδο κατά την αρνητική ακμή του ρολογιού, δηλαδή, κατά την μετάβαση του ρολογιού από την High κατάσταση στην Low.

Στο παραπάνω σχήμα βλέπουμε το κύκλωμα ενός D Flip Flop το οποίο έχει θετική διέγερση μετώπου, τον πίνακα αληθείας του και το γραφικό του σύμβολο. Όπως βλέπουμε, ένα D Flip Flop αποτελείται από δύο σύγχρονα D-Latch, από τα οποία το πρώτο λέγεται master και το δεύτερο slave.

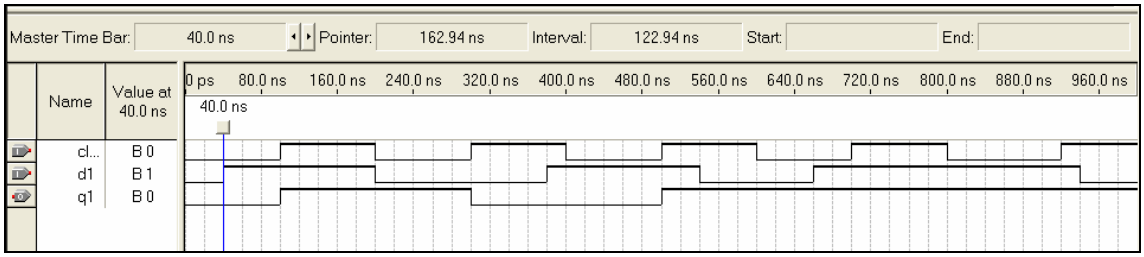
Όπως αναφέραμε και πριν, ένα D Flip Flop αποτελείται από δύο D-Latch συνδεδεμένα στη σειρά. Ένας τρόπος περιγραφής του είναι, λοιπόν, να το περιγράψουμε με την χρήση πακέτων ή στιγμιότυπων, όπως κάναμε και στη περίπτωση του πλήρη αθροιστή. Όπως βλέπουμε και στον κώδικα που ακολουθεί, έχουμε περιγράψει το κύκλωμα με βάση την συμπεριφορά του.

Καταρχήν ορίζουμε την οντότητα που θα περιέχει το D Flip Flop και της δίνουμε όνομα **dff_pos**. Η οντότητα περιέχει τις δύο εισόδους του κυκλώματος, από τις οποίες η μία δέχεται παλμούς ρολογιού, και μία έξοδο. Στη συνέχεια, χρησιμοποιούμε μία διαδικασία (process). Στην λίστα ευαισθησίας της περιέχεται μόνο το σήμα του ρολογιού, μιας και μόνο από αυτό επηρεάζεται η έξοδος του FF.

Η πρώτη εντολή που συναντούμε μέσα στην διαδικασία είναι μία IF με την εξής συνθήκη: **clock1 'event and clock1='1'**. Αυτή η γραμμή του κώδικα περιγράφει την συνθήκη: «**αν το ρολόι μεταβαίνει στην κατάσταση '1' από την κατάσταση '0'**». Στην περίπτωση αυτή, δηλαδή κατά το θετικό μέτωπο ωρολογιακών παλμών, θα έχουμε στην έξοδο την τιμή που έχει εκείνη την στιγμή η είσοδος: «**then q<=d**»;. Στη συνέχεια του προγράμματος παρατηρούμε ότι κι εδώ παραλείπεται η χρήση της εντολής ELSE, μιας και στο D Flip Flop επιθυμούμε την διατήρηση της τιμής της εξόδου μέχρι το επόμενο θετικό μέτωπο παλμών, οπότε η έξοδος θα ανανεωθεί.

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

ENTITY dff_pos IS
  PORT (d1, clock1: IN    std_logic;
                  q1 : OUT std_logic);
END dff_pos;

ARCHITECTURE behaviour OF dff_pos IS
 BEGIN
   PROCESS (clock1)
   BEGIN
    IF clock1 'event AND clock1='1' THEN
                    q1<=d1;
            END IF;
   END process;
END behaviour;
```

Στην προσομοίωση που ακολουθεί (σχήμα 11.5) φαίνεται η σωστή λειτουργία του D Flip-Flop. Όπως παρατηρούμε και στο σχήμα, όταν το ρολόι μεταβαίνει από την κατάσταση Low στην κατάσταση High, η έξοδος αλλάζει τιμή και παίρνει εκείνη της εισόδου κατά την συγκεκριμένη χρονική στιγμή. Σε όλες τις άλλες περιπτώσεις δεν συμβαίνει καμιά αλλαγή, δηλαδή η έξοδος κρατάει την τιμή που είχε πάρει κατά το τελευταίο θετικό μέτωπο του ωρολογιακού παλμού.



Σχήμα 11.5 *Προσομοίωση Ενός D Flip Flop*

11.2.5 Καταχωρητής 8 bits

Σ' αυτή την παράγραφο θα περιγράψουμε την υλοποίηση ενός καταχωρητή. Ως καταχωρητές θεωρούμε ένα σύνολο από Flip Flops, τα οποία είναι κατάλληλα συνδεδεμένα μεταξύ τους και δέχονται ένα κοινό ωρολογιακό σήμα. Σε κάθε Flip Flop μπορεί να αποθηκευτεί 1 bit πληροφορίας. Έτσι, σ' έναν καταχωρητή ο οποίος αποτελείται από n Flip Flops μπορούμε να αποθηκεύσουμε n bits πληροφορίας. Άρα, για την δημιουργία ενός καταχωρητή χωρητικότητας 8 bits θα πρέπει να χρησιμοποιήσουμε 8 Flip-Flops.

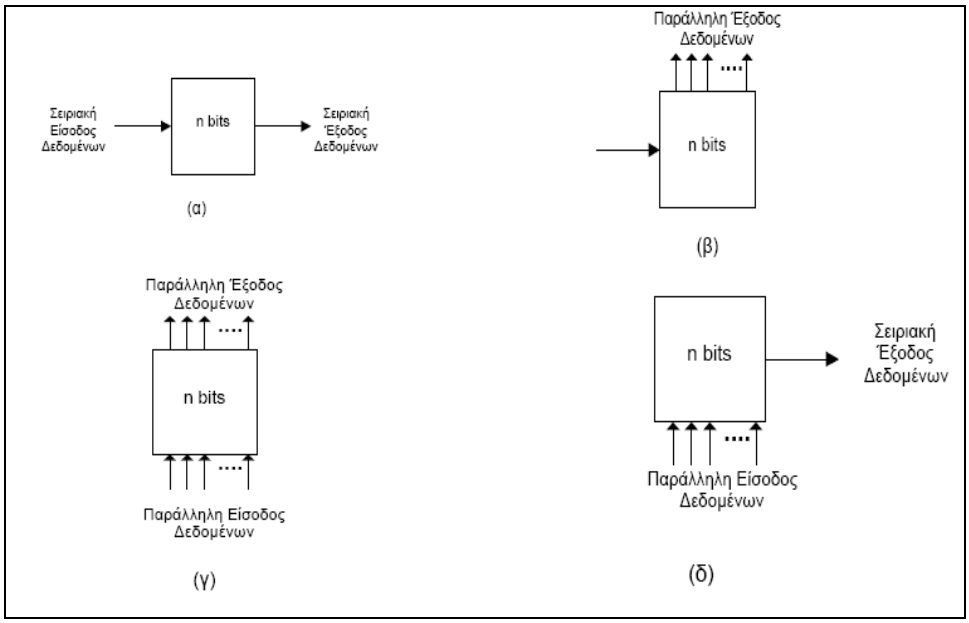
Η εισαγωγή των δεδομένων σ' έναν καταχωρητή μπορεί να γίνει είτε σειριακά, είτε παράλληλα. Στην πρώτη περίπτωση σε κάθε ωρολογιακό παλμό έχουμε είσοδο 1 bit, ενώ στην δεύτερη περίπτωση εισάγονται και τα n bits με ένα ωρολογιακό παλμό. Το ίδιο ισχύει και για την εξαγωγή των δεδομένων. Μπορούμε να εξάγουμε τα δεδομένα είτε παράλληλα είτε σειριακά.

Με βάση τους παραπάνω τρόπους εισαγωγής και εξαγωγής των δεδομένων, μπορούμε να χωρίσουμε τους καταχωρητές στις εξής τέσσερις κατηγορίες:

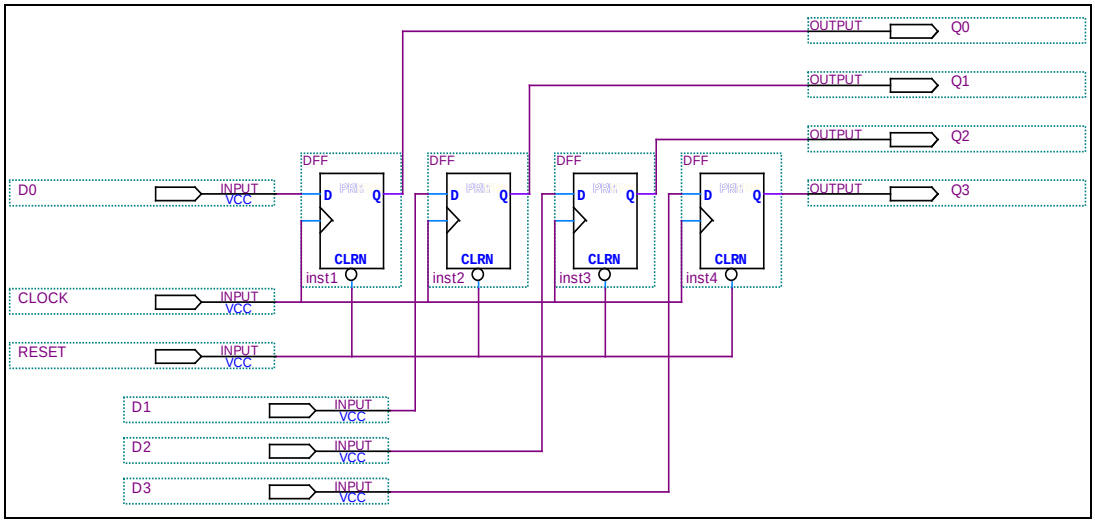
- Καταχωρητές σειριακής εισόδου-σειριακής εξόδου
- Καταχωρητές σειριακής εισόδου-παράλληλης εξόδου
- Καταχωρητές παράλληλης εισόδου-σειριακής εξόδου
- Καταχωρητές παράλληλης εισόδου-παράλληλης εξόδου

Οι τέσσερις τύποι των παραπάνω καταχωρητών φαίνονται στο σχήμα 11.6. Εμείς θα σχεδιάσουμε έναν καταχωρητή παράλληλης εισόδου - παράλληλης εξόδου 8 bits. Ένας τέτοιος καταχωρητής χωρητικότητας 4 bits φαίνεται στο σχήμα 11.7.

Στον κώδικα που ακολουθεί φαίνεται η περιγραφή της λειτουργίας ενός καταχωρητή παράλληλης εισόδου και παράλληλης εξόδου.



Σχήμα 11.6 *Τύποι καταχωρητών: α) σειριακής εισόδου-σειριακής εξόδου, β) σειριακής εισόδου- παράλληλης εξόδου, γ) παράλληλης εισόδου-παράλληλης εξόδου, δ) παράλληλης εισόδου-σειριακής εξόδου.*



Σχήμα 11.7 *Σχηματικό Διάγραμμα Καταχωρητή Παράλληλης Εισόδου-Παράλληλης Εξόδου 4 bits*

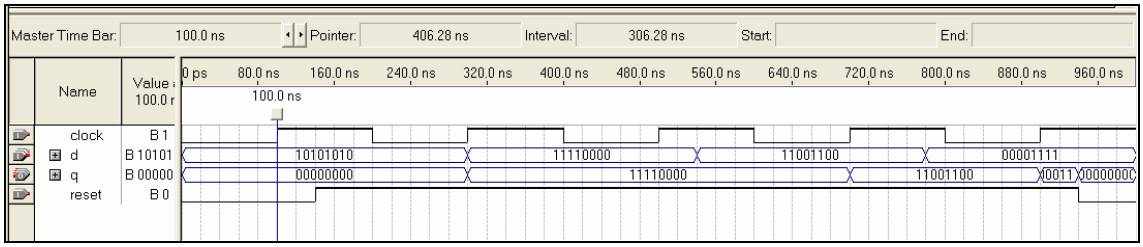
Το όνομα της οντότητας του κυκλώματος είναι **reg**. Αποτελείται από τρεις εισόδους εκ των οποίων οι δύο (**clock** και **reset**) έχουν εύρος 1 bit και η τρίτη, αυτή των δεδομένων (**d**), έχει εύρος 8 bits. Η έξοδος του κυκλώματος είναι και αυτή των 8 bits, όπως και η είσοδος. Η είσοδος reset λειτουργεί σαν μια ασύγχρονη είσοδος μηδενισμού.

Στο μέρος της αρχιτεκτονικής έχουμε μια διαδικασία (process), όπως σε όλα τα σύγχρονα ακολουθιακά κυκλώματα. Στην λίστα ευαισθησίας περιέχονται οι είσοδοι clock και reset. Αν η είσοδος reset είναι ίση με το ‘0’ τότε ο καταχωρητής δεν λειτουργεί και σαν έξοδο παίρνουμε την μηδενική τιμή, ανεξαρτήτως της εισόδου. Αν η είσοδος reset ισούται με ‘1’ τότε ο καταχωρητής λειτουργεί κανονικά και η τιμή της εξόδου εξαρτάται πλέον από το ρολόι του κυκλώματος. Δηλαδή, κατά το θετικό μέτωπο ωρολογιακών παλμών θα έχουμε στην έξοδο την τιμή που έχει εκείνη την στιγμή η είσοδος (q<=d;). Στη συνέχεια του προγράμματος

παρατηρούμε ότι κι εδώ παραλείπεται η χρήση της εντολής ELSE, μιας και επιθυμούμε την διατήρηση της τιμής της εξόδου μέχρι το επόμενο θετικό μέτωπο παλμών, οπότε η έξοδος θα ανανεωθεί. Η λογική ανανέωσης της εξόδου σε έναν καταχωρητή είναι ίδια με αυτή που περιγράψαμε προηγουμένως και στο απλό Flip Flop.

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

ENTITY reg IS
 PORT (d  : IN std_logic_vector(7 DOWNTO 0);
      clock, reset: IN std_logic;
      q: OUT std_logic_vector(7 DOWNTO 0);
      gnd: OUT bit);
 END reg;

ARCHITECTURE behaviour OF reg IS
  BEGIN
  gnd<='1';
    PROCESS (reset, clock)
    BEGIN
            IF reset='0' THEN
                    q<="00000000";
            ELSIF clock 'event AND clock= '1' THEN
                    q<=d;
            END IF;
    END process;
  END behaviour;
```

Στο Σχήμα 11.8 φαίνεται η προσομοίωση της λειτουργίας του καταχωρητή, οποία είναι σύμφωνη με τις θεωρητικές απαιτήσεις μας. Δηλαδή, όταν η είσοδος reset είναι ίση με '0' στην έξοδο έχουμε πάντα '0'. Αντίθετα, όταν η είσοδος reset παίρνει την τιμή '1' και έχουμε θετικό μέτωπο ωρολογιακών παλμών, η έξοδος παίρνει την τιμή που έχει η είσοδος την συγκεκριμένη χρονική στιγμή.



Σχήμα 11.8 *Προσομοίωση Καταχωρητή 8 bits*

Τέλος, να κάνετε την αντιστοίχηση ακροδεκτών. Τις εισόδους d[7], d[6]...,d[0] να τις αντιστοιχίσετε στους διακόπτες τύπου dip SW9, SW10, ..., SW16 αντίστοιχα, τις εισόδους clock και reset να τις αντιστοιχίσετε στους διακόπτες PS3 και PS1 αντίστοιχα, και τις εξόδους

q[7], q[6],...,q[0] να τις αντιστοιχίσετε στα led L1, L2, ...,L8 αντίστοιχα. Συμπληρώστε τον παρακάτω πίνακα για διευκόλυνσή σας:

d[7] → Pin ..... (SW9)	**q[7]** → Pin ..... (L1)
d[6] → Pin ..... (SW10)	**q[6]** → Pin ..... (L2)
d[5] → Pin ..... (SW11)	**q[5]** → Pin ..... (L3)
d[4] → Pin ..... (SW12)	**q[4]** → Pin ..... (L4)
d[3] → Pin ..... (SW13)	**q[3]** → Pin ..... (L5)
d[2] → Pin ..... (SW14)	**q[2]** → Pin ..... (L6)
d[1] → Pin ..... (SW15)	**q[1]** → Pin ..... (L7)
d[0] → Pin ..... (SW16)	**q[0]** → Pin ..... (L8)
gnd → Pin 141	**clock** → Pin ..... (PS3)
	reset → Pin ..... (PS1)

Κατόπιν, διαμορφώστε το ολοκληρωμένο και ελέγξτε τη σωστή λειτουργία του καταχωρητή.

11.2.6 Απαριθμητής

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

ENTITY counter IS
        PORT(clk,reset,e : IN std_logic;
                q:OUT std_logic_vector (3 DOWNTO 0);
                gnd:OUT bit);
END counter;

ARCHITECTURE behaviour OF counter IS
SIGNAL m : std_logic_vector (3 DOWNTO 0);
BEGIN
 gnd<='1';
 PROCESS(clk,reset)
 BEGIN
        IF reset='0' THEN
                m<="0000";
        ELSIF clk 'event AND clk='1'  THEN
                IF e='1' THEN
                        m<= m+1;
                ELSE
                        m<=m;
                END IF;
        END IF;
 END process;
 q<=m;
END behaviour;
```

Σ' αυτή την ενότητα θα παρουσιάσουμε ένα κύκλωμα απαριθμητή. Απαριθμητές ονομάζονται τα κυκλώματα που μπορούν να αυξήσουν ή να μειώσουν την τιμή της εξόδου τους κατά ένα, όταν στην είσοδο ρολογιού δέχονται παλμό clock. Τα κυκλώματα των απαριθμητών μπορούν να χρησιμοποιηθούν για να μετρούν πόσες φορές εμφανίστηκε κάποιο γεγονός ή για να δημιουργούν χρονικές καθυστερήσεις για τον έλεγχο διαφόρων εργασιών σε ένα σύστημα.

Η κατασκευή τέτοιων κυκλωμάτων γίνεται συνήθως με την χρήση καταχωρητών, αθροιστών, αφαιρετών ή με flip-flops διαφόρων τύπων. Ο απαριθμητής που θα σχεδιάσουμε εδώ λειτουργεί καταμετρώντας προς τα πάνω (up counter), είναι των 4 bits και διαθέτει μια είσοδο

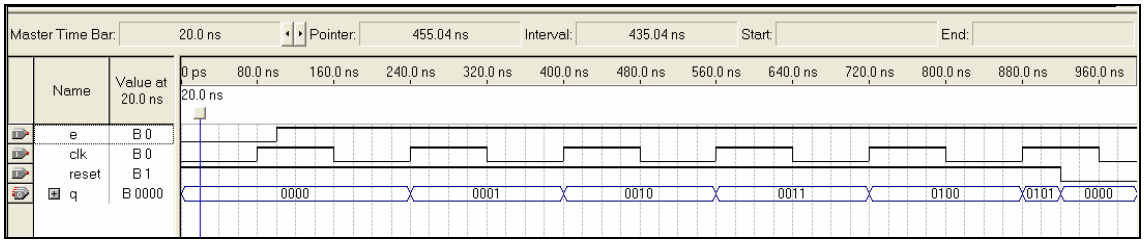
μηδενισμού και μια είσοδο ενεργοποίησης.

Στον κώδικα που ακολουθεί, ο μετρητής αποτελείται από τρεις εισόδους του 1 bit και μία έξοδο των 4 bits. Αφού η έξοδος έχει εύρος 4 bits σημαίνει ότι μπορεί να πάρει μέχρι και την τιμή ¨1111¨, που στο δεκαδικό σύστημα είναι ο αριθμός 15. Δηλαδή καταμετρά συνολικά δεκαέξι καταστάσεις.

Οι είσοδοι «**reset**» και «**e**» είναι οι είσοδοι μηδενισμού και ενεργοποίησης. Εάν η είσοδος reset βρίσκεται σε λογική κατάσταση μηδέν, η έξοδος του κυκλώματος (count) θα μηδενίζεται. Αν η είσοδος reset είναι ίση με ένα και η είσοδος ενεργοποίησης e είναι και αυτή ένα, τότε η τιμή της εξόδου του μετρητή θα αυξάνεται κατά ένα σε κάθε θετικό ωρολογιακό μέτωπο, αλλιώς (αν e=0) η τιμή της εξόδου θα παραμένει αμετάβλητη.

Οι παραπάνω περιπτώσεις των εισόδων περιγράφονται στον κώδικα μέσω μιας διαδικασίας (process). Η λίστα ευαισθησίας περιέχει την είσοδο για το ρολόι και την είσοδο μηδενισμού. Οποιαδήποτε αλλαγή στις τιμές αυτών των δύο εισόδων προκαλεί την αλλαγή της κατάστασης της εξόδου.

Στη προσομοίωση που ακολουθεί φαίνεται η αύξηση της τιμής της εξόδου κατά ένα, σε κάθε θετικό ωρολογιακό παλμό όταν η είσοδος reset ισούται με ένα και η είσοδος e επίσης ισούται με ένα.



Σχήμα 11. 9 *Προσομοίωση Απαριθμητή 4 bits*

Στη συνέχεια, να κάνετε την αντιστοίχηση ακροδεκτών. Τις εισόδους clock, reset και e να τις αντιστοιχίσετε στους διακόπτες PS3, PS1 και SW1 αντίστοιχα, και τις εξόδους q[3], q[2], q[1], q[0] να τις αντιστοιχίσετε στα led L1, L2, L3, L4 αντίστοιχα. Συμπληρώστε τον παρακάτω πίνακα για διευκόλυνσή σας:

q[3] → Pin ..... (L1)	**e** → Pin ..... (SW1)
q[2] → Pin ..... (L2)	**clock** → Pin ..... (PS3)
q[1] → Pin ..... (L3)	**reset** → Pin ..... (PS1)
q[0] → Pin ..... (L4)	**gnd** → Pin 141

Τέλος, διαμορφώστε το ολοκληρωμένο και ελέγξτε τη σωστή λειτουργία του απαριθμητή. Ενδεχομένως, λόγω αναπήδησης του διακόπτη PS2, να βλέπετε την ακολουθία απαρίθμησης να μην είναι σωστή, αυτό όμως οφείλεται στις μηχανικές ατέλειες του διακόπτη και όχι στον κώδικα σας.

ΕΡΓΑΣΤΗΡΙΟ 12

ΑΠΑΡΙΘΜΗΤΗΣ ΣΕ ΑΠΕΙΚΟΝΙΣΗ ΕΠΤΑ ΤΟΜΕΩΝ

12.1 ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Η VHDL είναι μια γλώσσα που επιτρέπει την ιεραρχική σχεδίαση κυκλωμάτων. Έτσι, ο χρήστης μπορεί να περιγράψει ένα σύνθετο κύκλωμα με βάση τα υποκυκλώματα που το αποτελούν. Τα υποκυκλώματα που συμπεριλαμβάνονται σε μια σχεδίαση ονομάζονται **συνιστώσες** (components). Αυτές οι συνιστώσες είναι μικρότερα δομικά κυκλώματα σε VHDL, που έχουν ήδη περιγραφεί και μπορούν να κληθούν ως αρχεία βιβλιοθήκης. Τέτοια αρχεία μπορεί να είναι έτοιμα αρχεία από κάποιες βιβλιοθήκες ή αρχεία που σχεδιάζει ο ίδιος ο χρήστης. Τα κύρια χαρακτηριστικά της σχεδίασης με υποκυκλώματα είναι η δήλωση των συνιστωσών που περιέχονται στο κύκλωμα, καθώς και η περιγραφή του τρόπου με τον οποίο οι συνιστώσες συνδέονται μεταξύ τους.

Με την δήλωση της κάθε συνιστώσας καθορίζουμε το όνομα της καθώς και τα ονόματα των εισόδων και των εξόδων της. Στη συνέχεια, στο κύριο σώμα της αρχιτεκτονικής, πρέπει να δημιουργήσουμε στιγμιότυπα για τις συνιστώσες που έχουμε δηλώσει, τα οποία λειτουργούν σαν υποκυκλώματα της σχεδίασής μας. Η δημιουργία των στιγμιοτύπων γίνεται πάντα μετά τη δήλωση μιας συνιστώσας. Τα στιγμιότυπα είναι της μορφής :

όνομα στιγμιότυπου: **όνομα συνιστώσας** PORT MAP (**ονόματα σημάτων**) ;

12.2 ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Στην άσκηση αυτή αρχικά θα δημιουργήσουμε ένα κύκλωμα που στηρίζεται σε δύο υποκυκλώματα. Το πρώτο είναι ένας αύξων απαριθμητής (upcounter), ίδιος με αυτόν που δημιουργήσαμε σε προηγούμενη άσκηση. Το δεύτερο είναι ο κωδικοποιητής BCD-to-seven-segment, που επίσης υλοποιήσαμε σε προηγούμενη άσκηση. Τα δύο αυτά συστατικά κυκλώματα θα ενσωματωθούν σε μια ανώτερη οντότητα VHDL (top entity), η οποία θα εξάγει σε ενδείκτη επτά τομέων την δεκαεξαδική τιμή που απαριθμεί ο counter.

Στη συνέχεια θα σχεδιάσουμε μια παραλλαγή του κυκλώματος όπου οι παλμοί ρολογιού δεν θα δίνονται χειροκίνητα με διακόπτη αλλά θα λαμβάνονται από το ενσωματωμένο ρολόι του αναπτυξιακού, μέσω ενός διαιρέτη συχνότητας, και ο απαριθμητής θα μπορεί να λειτουργεί είτε σαν up-counter είτε σαν down-counter, με επιλογή από τον χρήστη μέσω ενός διακόπτη.

Οι παρακάτω κώδικες πρέπει να δημιουργηθούν και να αποθηκευτούν στον ίδιο φάκελο.

12.2.1 Ο απαριθμητής

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

```
ENTITY upcount IS
PORT(clock, Resetn, E: IN     std_logic;
                        Q: OUT std_logic_vector (3 DOWNTO 0));
END upcount;

ARCHITECTURE behaviour OF upcount IS
SIGNAL Count: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
 PROCESS(Clock, Resetn)
 BEGIN
 IF Resetn='0' THEN
          Count<="0000";
 ELSIF Clock'EVENT AND Clock='1' THEN
          IF E='1' THEN
                   IF Count<9 THEN
                            Count<=Count+1;
                   ELSE
                            Count<="0000";
                   END IF;
          ELSE
                   Count<=Count;
          END IF;
 END IF;
 END PROCESS;
 Q<=Count;
END behaviour;
```

12.2.2 Ο αποκωδικοποιητής BCD-to-7 segment

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

ENTITY seg_7 IS
 PORT(c : IN     std_logic_vector (3 DOWNTO 0);
       ex1: OUT std_logic_vector (6 DOWNTO 0));
END seg_7;

ARCHITECTURE behaviour OF seg_7 IS
BEGIN
WITH c SELECT
   ex1<= "1111110" WHEN "0000", --0
          "0000110" WHEN "0001", --1
          "1101101" WHEN "0010", --2
          "1111001" WHEN "0011", --3
          "0110011" WHEN "0100", --4
```

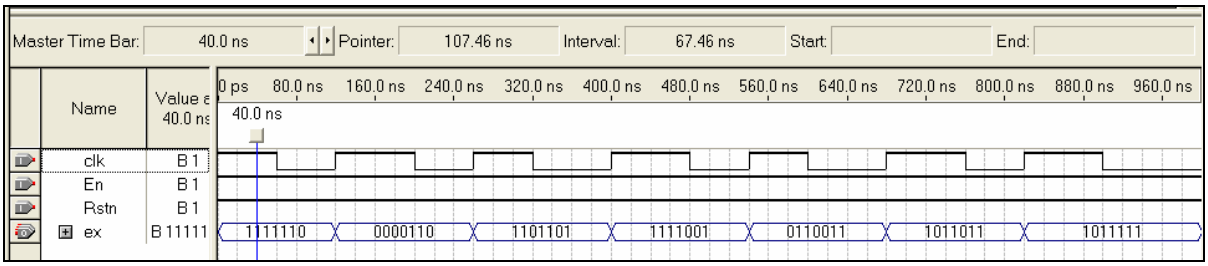
```
            "1011011" WHEN "0101", --5
            "1011111" WHEN "0110", --6
            "1110000" WHEN "0111", --7
            "1111111" WHEN "1000", --8
            "1111011" WHEN "1001", --9
            "0000001" WHEN OTHERS;
END behaviour;
```

12.2.3 Τελική οντότητα VHDL που ενσωματώνει τον απαριθμητή και τον αποκωδικοποιητή BCD-7-segment.

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

ENTITY count_sev_segm IS
PORT(clk, Rstn, En: IN     std_logic;
                   ex: OUT std_logic_vector (6 DOWNTO 0));
END count_sev_segm;

ARCHITECTURE structure OF count_sev_segm IS
SIGNAL QS: std_logic_vector (3 DOWNTO 0);
SIGNAL ex2: std_logic_vector (6 DOWNTO 0);

COMPONENT upcount
PORT(clock, Resetn, E : IN std_logic;
                    Q : OUT std_logic_vector (3 DOWNTO 0));
END COMPONENT;

COMPONENT seg_7
PORT(c : IN std_logic_vector (3 DOWNTO 0);
        ex1: OUT std_logic_vector(6 DOWNTO 0));
END COMPONENT;

BEGIN
        stage1:upcount PORT MAP(clk,Rstn,En,QS);
        stage2:seg_7 PORT MAP(QS,ex2);
        ex(6 DOWNTO 0)<=ex2(6 DOWNTO 0);
END structure;
```

12.2.4 Προσομοίωση

Να κάνετε λειτουργική προσομοίωση του κυκλώματος. Το αποτέλεσμα πρέπει να είναι όπως παρακάτω.



Σχήμα 12.1 *Η προσομοίωση του κυκλώματος που περιγράφει ο κώδικας της παραγράφου 12.2.3. Σε κάθε παλμό clock η έξοδος στην απεικόνιση επτά τομέων αυξάνει κατά 1.*

12.2.5 Αντιστοίχηση ακροδεκτών και διαμόρφωση του κυκλώματος

Να κάνετε την αντιστοίχηση ακροδεκτών. Τις εισόδους Clk, Rstn και En να τις αντιστοιχίσετε στους διακόπτες PS3, PS1 και SW1 αντίστοιχα ενώ τις εξόδους ex1[0] ως ex1[6] να τις αντιστοιχίσετε στα led ενός ενδείκτη 7 τμημάτων όπου η έξοδος ex1[6] αντιστοιχεί στο γράμμα ‘a’ (σχήμα 9.2 β), το ex1[5] στο ‘b’ και ο τελευταίος ακροδέκτης ex1[0] στο ‘g’.

Clk → Pin ..... (PS3)	**ex1[0]** → Pin ..... (g)
Rstn → Pin ..... (PS1)	**ex1[1]** → Pin ..... (f)
En → Pin ..... (SW1)	**ex1[2]** → Pin ..... (e)
	ex1[3] → Pin ..... (d)
	ex1[4] → Pin ..... (c)
	ex1[5] → Pin ..... (b)
	ex1[6] → Pin ..... (a)

Κατόπιν, διαμορφώστε το ολοκληρωμένο και ελέγξτε τη σωστή λειτουργία του κυκλώματος. Όπως έχουμε ήδη αναφέρει, ενδεχομένως, λόγω αναπήδησης του διακόπτη PS2, να βλέπετε την ακολουθία απαρίθμησης να μην είναι σωστή, αυτό όμως οφείλεται στις μηχανικές ατέλειες του διακόπτη και όχι στον κώδικα σας.

12.2.6 Τροποποίηση του κυκλώματος ώστε να δέχεται είσοδο clock από το ρολόι του αναπτυξιακού και να καταμετρά προς τα πάνω ή προς τα κάτω

Το αναπτυξιακό LP-2900 διαθέτει έναν ενσωματωμένο κρυσταλλικό ταλαντωτή που παράγει σήμα ρολογιού με συχνότητα 10 MHz. Το σήμα ρολογιού αυτό συνδέεται στο Pin 55 του FPGA και μπορεί να χρησιμοποιηθεί αυτό για είσοδο clock αντί του διακόπτη PS3. Θα πρέπει όμως, σε σχέση με το προηγούμενο project, να προστεθεί μια συνιστώσα διαίρεσης συχνότητας ώστε η συχνότητα να γίνει περίπου 1 Hz. Επίσης, ο διακόπτης SW2 θα επιλέγει αν ο απαριθμητής θα μετρά προς τα πάνω ή προς τα κάτω, μέσω της εισόδου dir.

Δημιουργήστε ένα νέο project με όνομα count_sev_segm1. Στο project αυτό δημιουργήστε τα ακόλουθα αρχεία VHDL:

- **UpDownCount.vhd** : Ακολουθεί ο κώδικας:

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

ENTITY UpDownCount IS
 PORT(clock, Resetn, E, dir: IN    std_logic;
                          Q: OUT std_logic_vector (3 DOWNTO 0));
END UpDownCount;

ARCHITECTURE behaviour OF UpDownCount IS
SIGNAL Count: std_logic_vector (3 DOWNTO 0);
BEGIN
 PROCESS(Clock, Resetn, dir)
 BEGIN
 IF Resetn='0' THEN
        IF dir='0' THEN             --count-up
                Count<="0000";
        ELSE
                Count<="1001";  --count-down
        END IF;
 ELSIF (Clock'EVENT AND Clock='1') THEN
        IF E='1' THEN
                IF dir='0' THEN          --count-up
                        IF Count<9 THEN
                                Count<=Count+1;
                        ELSE
                                Count<="0000";
                        END IF;
                ELSE                        --count-down
                        IF Count>0 THEN
                                Count<=Count-1;
                        ELSE
                                Count<="1001";
                        END IF;
                END IF;
        ELSE
                Count<=Count;
        END IF;
 END IF;
 END PROCESS;
 Q<=Count;
END behaviour;
```

Έχει προστεθεί η είσοδος dir η οποία θα αντιστοιχηθεί με τον διακόπτη SW2 και θα καθορίζει την φορά απαρίθμησης. Επίσης, έχει αλλάξει η λειτουργία του reset καθώς όταν ο

απαριθμητής απαριθμεί προς τα πάνω μηδενίζει την έξοδος ενώ όταν απαριθμεί προς τα κάτω του δίνει την τιμή 9 (1001).

- **Seg_7.vhd**: Ίδιο με το προηγούμενο project
- **Freqdiv.vhd**: Προκαλεί τη διαίρεση συχνότητας από τα 10 MHz στο 1 MHz και ο κώδικάς του ακολουθεί:

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

ENTITY freqdiv IS
PORT( clk    : IN std_logic;
      clkout: OUT std_logic;
      buz    : BUFFER bit);
END freqdiv;

ARCHITECTURE behaviour OF freqdiv IS
BEGIN
PROCES(clk)
VARIABLE cnt : INTEGER RANGE 0 to 12587500; --for 25MHZ to 1 HZ
BEGIN
   IF(clk'event and clk='1') THEN
         IF(cnt=10000000)THEN
               cnt:=0;
               clkout<='1';
               buz<=not(buz);
         ELSE
               cnt := cnt+1;
               clkout<='0';
         END IF;
   END IF;
END process;
END behaviour;
```

Έχει προστεθεί η έξοδος **buz** η οποία οδηγεί τον βομβητή του αναπτυξιακού, ώστε να υπάρχει και ηχητική ένδειξη της απαρίθμησης, και η οποία θα πρέπει να αντιστοιχηθεί στο Pin 46.

- **count_sev_segm1.vhd**: Το top-entity αρχείο, ο κώδικας του οποίου ακολουθεί:

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

ENTITY count_sev_segm1 IS
PORT(clk, Rstn, En : IN std_logic;
                ex : OUT std_logic_vector (6 DOWNTO 0);
               dir : IN std_logic;
               buz : BUFFER bit);
END count_sev_segm1;
```

```
ARCHITECTURE structure OF count_sev_segm1 IS
SIGNAL QS     : std_logic_vector (3 DOWNTO 0);
SIGNAL ex2    : std_logic_vector (6 DOWNTO 0);
SIGNAL clkout: std_logic;

COMPONENT upcount
PORT(clock, Resetn, E, dir : IN std_logic;
                          Q  : OUT std_logic_vector (3 DOWNTO 0));
END COMPONENT;

COMPONENT seg_7
PORT(   c : IN std_logic_vector (3 DOWNTO 0);
        ex1: OUT std_logic_vector(6 DOWNTO 0));
END COMPONENT;

COMPONENT freqdiv
PORT( clk : IN std_logic; clkout : OUT std_logic; buz : BUFFER bit);
END COMPONENT;

BEGIN
    stage1:freqdiv PORT MAP(clk,clkout,buz);
    stage2:upcount PORT MAP(clkout,Rstn,En,dir,QS);
    stage3:seg_7 PORT MAP(QS,ex2);
    ex(6 DOWNTO 0)<=ex2(6 DOWNTO 0);
END structure;
```

Για την αντιστοίχιση ακροδεκτών, συμπληρώστε για διευκόλυνση σας τον παρακάτω πίνακα:

Clk → Pin 55	**ex1[0]** → Pin ..... (g)
Rstn → Pin ..... (PS1)	**ex1[1]** → Pin ..... (f)
En → Pin ..... (SW1)	**ex1[2]** → Pin ..... (e)
Dir → Pin ..... (SW2)	**ex1[3]** → Pin ..... (d)
Buz → Pin 46	**ex1[4]** → Pin ..... (c)
	ex1[5] → Pin ..... (b)
	ex1[6] → Pin ..... (a)

Κατόπιν, διαμορφώστε το κύκλωμα και ελέγξτε τη σωστή λειτουργία του.

ΑΣΚΗΣΕΙΣ ΕΠΑΝΑΛΗΨΗΣ

1. α. Να δημιουργήσετε στο περιβάλλον του Multisim έναν καταχωρητή 4 bits

β. Να γράψετε κώδικα σε VHDL που να περιγράφει έναν καταχωρητή 4-bits και να προσομοιώσετε τη λειτουργία του στο Quartus.

2. α. Να δημιουργήσετε στο περιβάλλον του Multisim ένα κύκλωμα που να επιτελεί τις πράξεις της πρόσθεσης και της αφαίρεσης ανάμεσα σε αριθμούς 4-bits.

β. Να δημιουργήσετε αντίστοιχο κώδικα σε VHDL και να προσομοιώσετε την λειτουργία του στο Quartus.

3. α. Να προσομοιώσετε στο Multisim την λειτουργία ενός συγκριτή δύο αριθμών των 4-bits.

β. Να περιγράψετε το κύκλωμα με κώδικα VHDL και να το προσομοιώσετε στο Quartus.

4. α. Να δημιουργήσετε στο Multisim έναν απαριθμητή modulo 8.

β. Να περιγράψετε τον απαριθμητή με κώδικα VHDL και να προσομοιώσετε την λειτουργία του στο Quartus.

5. α. Να δημιουργήσετε κύκλωμα που να επιλέγει ανάμεσα στις εξής τέσσερις πράξεις: AND, OR, XOR, NAND. Οι πράξεις να γίνονται ανάμεσα σε 2 bits. Η επιλογή μιας πράξης να γίνεται με τη βοήθεια πληκτρολογίου 2-bits.

β. Να περιγράψετε το παραπάνω κύκλωμα επιλογής σε VHDL και να το προσομοιώσετε στο Quartus II.

6. Να δημιουργήσετε μια βαθμίδα που να λαμβάνει την έξοδο του απαριθμητή του θέματος 4 και να την προβάλλει με τη μορφή δεκαδικού ψηφίου σε απεικόνιση επτά τομέων (7-segment).

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. John Wakerly, Ψηφιακή Σχεδίαση, Αρχές και Πρακτικές, Εκδόσεις Κλειδάριθμος, 2002.

2. Morris Mano, Ψηφιακή Σχεδίαση, Εκδόσεις Παπασωτηρίου, 2006.

3. S. Brown, Z. Vranesic, Σχεδίαση Ψηφιακών συστημάτων με τη γλώσσα VHDL, Εκδόσεις Τζιόλα, Θεσσαλονίκη 2001.

4. Ν. Ασημάκης, Γ. Μουστάκας, Π. Παπαγέωργας, Ψηφιακά Ηλεκτρονικά, Μέρος Β, Οργανισμός Εκδόσεων Διδακτικών Βιβλίων, Αθήνα.

5. Ιστοσελίδα εταιρίας Altera: www.altera.com

6. Φύλλα δεδομένων, που υπάρχουν ελεύθερα στο διαδίκτυο.

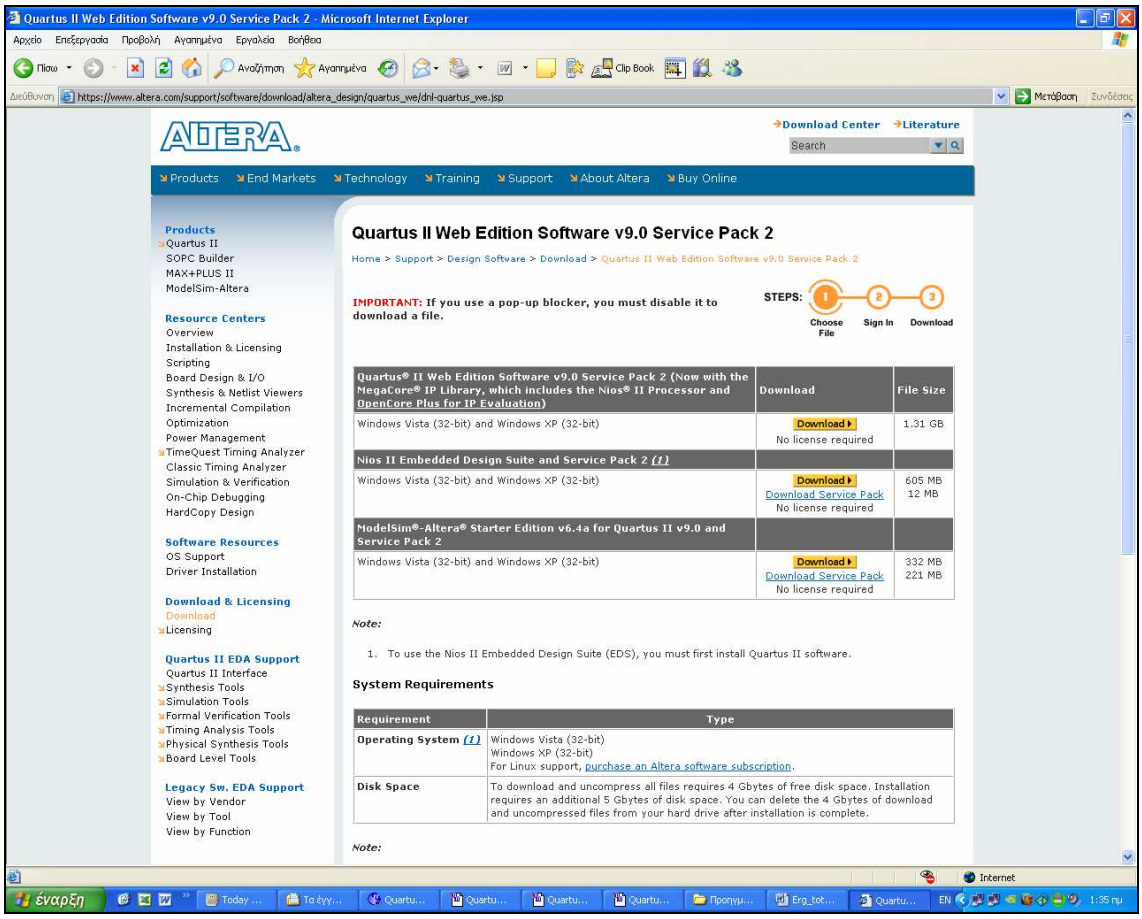
ΠΑΡΑΡΤΗΜΑ Α’

ΟΔΗΓΙΕΣ ΓΙΑ ΤΗΝ ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΑΔΕΙΟΔΟΤΗΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ALTERA QUARTUS II WEB EDITION

1. Η ιστοσελίδα για τη φόρτωση (downloading) του **Quartus II Web Edition** από τον δικτυακό τόπο της Altera είναι η:

https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp

Αφού φορτώσετε το αρχείο στον υπολογιστή σας εκτελέστε το και το Quartus II Web Edition θα εγκατασταθεί στον υπολογιστή σας.



2. Αδειοδότηση παλαιότερων εκδόσεων του Quartus II Web Edition

Σε περίπτωση που έχετε κάποια έκδοση του Quartus Web Edition παλαιότερη της 8.1 απαιτείται η λήψη άδειας χρήσης η οποία είναι δωρεάν και λαμβάνεται από την διεύθυνση **http://www.altera.com/support/licensing/lic-choose.html** επιλέγοντας “*Quartus II Web Edition software*” και ακολουθώντας τα βήματα που υποδεικνύονται. Θα σας ζητηθούν τα στοιχεία σας και το NIC-ID (MAC Address) του υπολογιστή σας. Αυτό το βρίσκετε πληκτρολογώντας στο command promt:> ipconfig /all. Προσέξτε το κενό ανάμεσα στο ipconfig και στο slash. Το ζητούμενο είναι ένας δωδεκαψήφιος αριθμός στο πεδίο «φυσική διεύθυνση» (π.χ. 00-3E-12-5F-00-72). Θα λάβετε την άδεια με e-mail. Αποθηκεύστε την άδεια σε έναν φάκελο (π.χ. c:\altera\license\mylicense.dat). Κατόπιν, ανοίξτε το Quartus και εγκαταστήστε την άδεια υποδεικνύοντας την διαδρομή του αρχείου στον δίσκο σας (Μενού Tools/license setup).

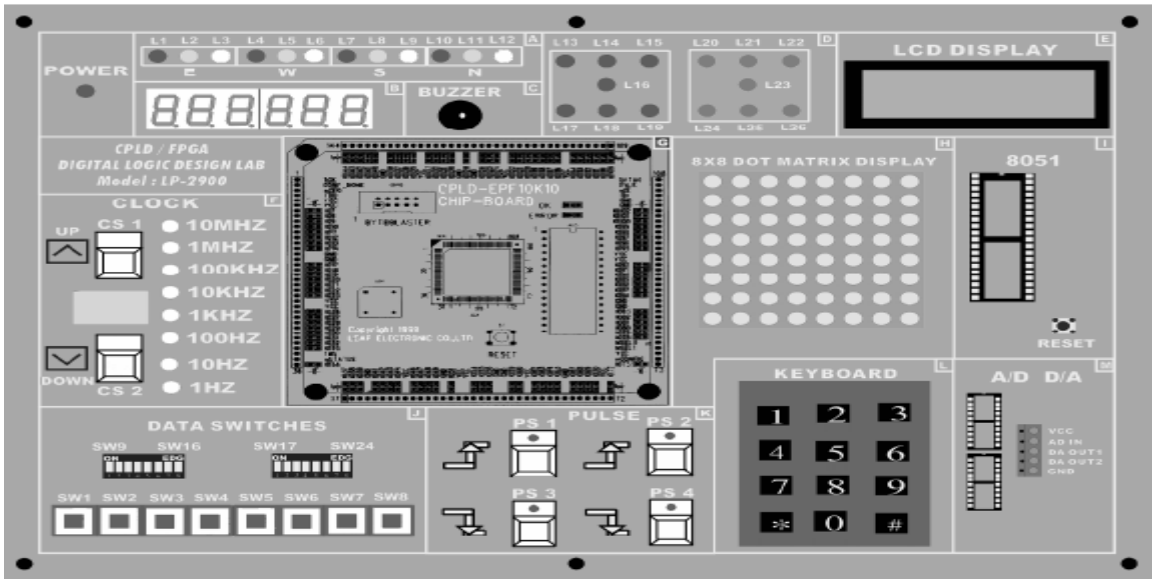
ΠΑΡΑΡΤΗΜΑ Β'

ΑΝΑΠΤΥΞΙΑΚΟ ΚΥΚΛΩΜΑ LEAP LP-2900

Β.1 Γενικά

Η αναπτυξιακή πλακέτα που θα χρησιμοποιήσουμε για την υλοποίηση των εφαρμογών μας είναι η LP-2900 της εταιρίας Leap Electronic Co. (http://www.leap.com.tw) Η πλακέτα αυτή βασίζεται στο ολοκληρωμένο FPGA **EPF10K10TC144-4** της εταιρίας **Altera**. Αυτό το FPGA έχει **144 pins εισόδου/εξόδου**, από τα οποία τα 102 είναι ελεύθερα για χρήση. Αποτελείται από **10.000 λογικές πύλες**, οργανωμένες σε **576 Λογικά Στοιχεία** (Logic Elements, LE), 720 Flip-Flops και 6.144 bits μνήμης.

Το LP-2900 είναι ένα εκπαιδευτικό αναπτυξιακό κύκλωμα, το οποίο παρέχει στους χρήστες του έναν αριθμό από περιφερειακά συστήματα. Είναι κατάλληλο για ανάπτυξη και επαλήθευση απλών πρωτότυπων ψηφιακών κυκλωμάτων. Στο σχήμα Β.1 που ακολουθεί φαίνεται η αναπτυξιακή πλακέτα LP-2900.



Σχήμα Β.1 Το LP-2900

Το LP-2900 χωρίζεται σε τέσσερα μέρη. Στο μέρος όπου περιέχεται το FPGA, στο μέρος τροφοδοσίας, στις συσκευές εισόδου/εξόδου και στη θύρα επικοινωνίας με τον υπολογιστή.

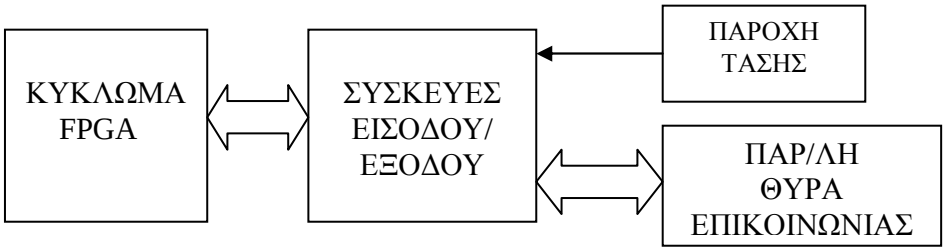
Οι συσκευές εισόδου/εξόδου του αναπτυξιακού είναι:

- 4 σετ από κόκκινα, κίτρινα, και πράσινα leds κοινής καθόδου
- ενδείκτες επτά τομέων (7segment displays) κοινής ανόδου
- 1 buzzer
- 2 ηλεκτρονικά ζάρια (dices) κοινής καθόδου
- 8 διακόπτες τύπου push-button
- 2X8 διακόπτες τύπου dip switches
- 4 διακόπτες παλμών

- 1 8x8 dot matrix display
- 1 πληκτρολόγιο 4x3
- 1 οθόνη LCD
- 1 μετατροπέας σήματος από αναλογικό σε ψηφιακό(A/D και D/A)
- 1 ολοκληρωμένο κύκλωμα 8051 (microprocessor)

Στο υποκύκλωμα όπου είναι τοποθετημένο το FPGA (chipboard) εκτός από τη διάταξη FPGA υπάρχει μία μνήμη EPROM, ένας διακόπτης reset και 144 μικρά leds που είναι συνδεδεμένα στους ακροδέκτες του FPGA, ώστε να μπορούμε να δούμε ποιοι από αυτούς χρησιμοποιούνται, αλλά και ποια είναι η τιμή τους (1 ή 0).

Η επικοινωνία με τον υπολογιστή γίνεται μέσω της παράλληλης θύρας του υπολογιστή. Για την σύνδεση του υπολογιστή με την αναπτυξιακή πλακέτα χρειαζόμαστε ένα καλώδιο που το ένα άκρο του το συνδέουμε στην παράλληλη θύρα του υπολογιστή και το άλλο στη θύρα του αναπτυξιακού. Επίσης είναι ανάγκη να εγκαταστήσουμε τους οδηγούς BYTEBLASTER της ALTERA ώστε να εγκατασταθεί στο QUARTUS η κατάλληλη θύρα προγραμματισμού (LPT1).

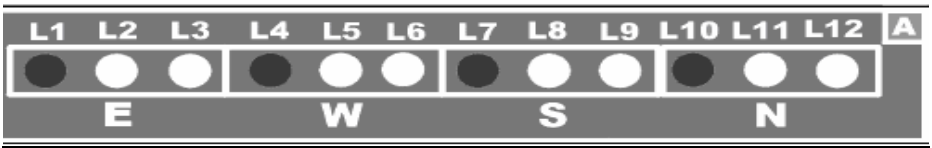


Σχήμα Β.2 Σχηματικό Διάγραμμα του LP-2900

Το σχεδιαστικό πρόγραμμα με το οποίο συνεργάζεται το LP-2900 είναι το QUARTUS II. Μέσω αυτού του προγράμματος θα δημιουργηθούν τα προγραμματιστικά αρχεία του κάθε κυκλώματος ώστε να παραχθούν τα κατάλληλα σήματα που θα διαμορφώσουν τη διάταξη, μέσω της διασύνδεσης JTAG. Στο παραπάνω σχήμα (σχήμα Β.2) φαίνεται το διάγραμμα βαθμίδων του αναπτυξιακού κυκλώματος LP-2900.

Στα παρακάτω, θα παραθέσουμε την αντιστοίχηση των ακροδεκτών της κάθε συσκευής εισόδου/εξόδου με τους ακροδέκτες του FPGA. Λόγω των πολλών συσκευών εισόδου/εξόδου του αναπτυξιακού κυκλώματος θα αναφερθούμε μόνο σε αυτές που θα χρησιμοποιήσουμε στις εργαστηριακές ασκήσεις που θα ακολουθήσουν.

Β.2 LED κοινής καθόδου



L1	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**
Pin7	Pin8	Pin9	Pin10	Pin11	Pin12	Pin13	Pin14	Pin17

L10	**L11**	**L12**
Pin18	Pin19	Pin20

Επειδή τα leds είναι κοινής καθόδου, για να δούμε τα αποτελέσματα σε αυτά θα πρέπει να φέρουμε την κοινή κάθοδο των LEDs στο δυναμικό της γης. Ας σημειώσουμε ότι η κοινή κάθοδος συνδέεται μέσω ενός αντιστροφέα με τον ακροδέκτη 141 του FPGA. Άρα πρέπει να γίνει η αντιστοίχηση pin 141 => +Vcc. Προφανώς, κάθε δίοδος φωτοεκπομπής ανάβει όταν λάβει λογικό 1 στην άνοδο.

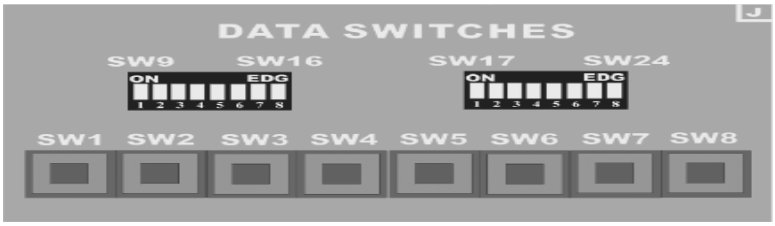
Εκτός από τα led L1 έως L12 υπάρχει στην πλακέτα του FPGA (Chip-Board) ένας μεγάλος αριθμός από SMD led (102 συνολικά). Η κάθοδος των led αυτών είναι μόνιμα συνδεδεμένη στο δυναμικό της γης και ανάβουν δίνοντας στον αντίστοιχο ακροδέκτη του led λογικό 1. Ορισμένα τέτοια led που μπορείτε να χρησιμοποιήσετε είναι αντιστοιχισμένα στα pin 7 έως και 14, 17 έως και 23, 26 έως και 33 και 95 έως και 102.

Β.3 Ενδείκτες 7 τομέων (7 Segment Display)

A	B	C	D	E	F	G	DP
Pin23	Pin26	Pin27	Pin28	Pin29	Pin30	Pin31	Pin32

Η επιλογή για το ποιος ενδείκτης επτά τομέων (7 Segment Display) θα εμφανίσει τα αποτελέσματα γίνεται από έναν αποκωδικοποιητή 3 προς 8 (74HCT138) του οποίου οι είσοδοι (DE1, DE2, DE3) συνδέονται με τους ακροδέκτες 33, 36 και 37.

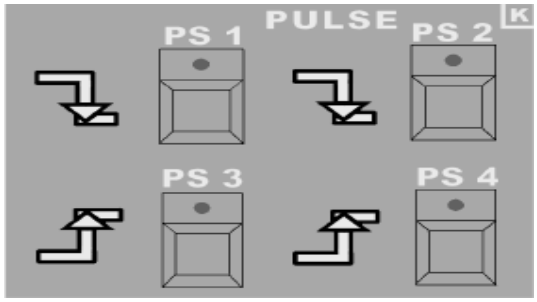
Β.4 Διακόπτες Δεδομένων



SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
Pin47	Pin48	Pin49	Pin51	Pin59	Pin60	Pin62	Pin63
SW9	**SW10**	**SW11**	**SW12**	**SW13**	**SW14**	**SW15**	**SW16**
Pin64	Pin65	Pin67	Pin68	Pin69	Pin70	Pin72	Pin73
SW17	**SW18**	**SW19**	**SW20**	**SW21**	**SW22**	**SW23**	**SW24**
Pin78	Pin79	Pin80	Pin81	Pin82	Pin83	Pin86	Pin87

Οι διακόπτες SW1 ως και SW8 αντιστοιχούν σε διακόπτες τύπου push-button ενώ οι διακόπτες SW9 ως και SW24 σε διακόπτες τύπου dip switches. Τέλος, μια άλλη σειρά από διακόπτες χρησιμοποιείται κυρίως για να δίνουμε παλμούς ρολογιού. Αυτοί είναι οι διακόπτες που παρουσιάζονται παρακάτω.

Β.5 Διακόπτες Παλμών



PS1	PS2	PS3	PS4
Pin54	Pin56	Pin124	Pin126

Β.6 Buzzer (Βομβητής)

SP1
Pin46

Β. 7 Clock 10 MHz

Στην αναπτυξιακή πλακέτα υπάρχει ένας ενσωματωμένος ταλαντωτής στα 10MHz που συνδέεται σαν είσοδος (OSC) στο Pin 55 και μπορεί να χρησιμοποιηθεί σαν CLOCK στα ακολουθιακά κυκλώματα. Ο χρήστης μπορεί να πάρει από τον ταλαντωτή αυτόν μικρότερες συχνότητες. υλοποιώντας σε VHDL διαιρέτες συχνότητας.

Υπάρχει επίσης μια σειρά από 8 πορτοκαλί led (L27 έως L34) για την ένδειξη της συχνότητας και δύο διακόπτες (UP, DOWN) για την αύξηση ή μείωση της συχνότητας.

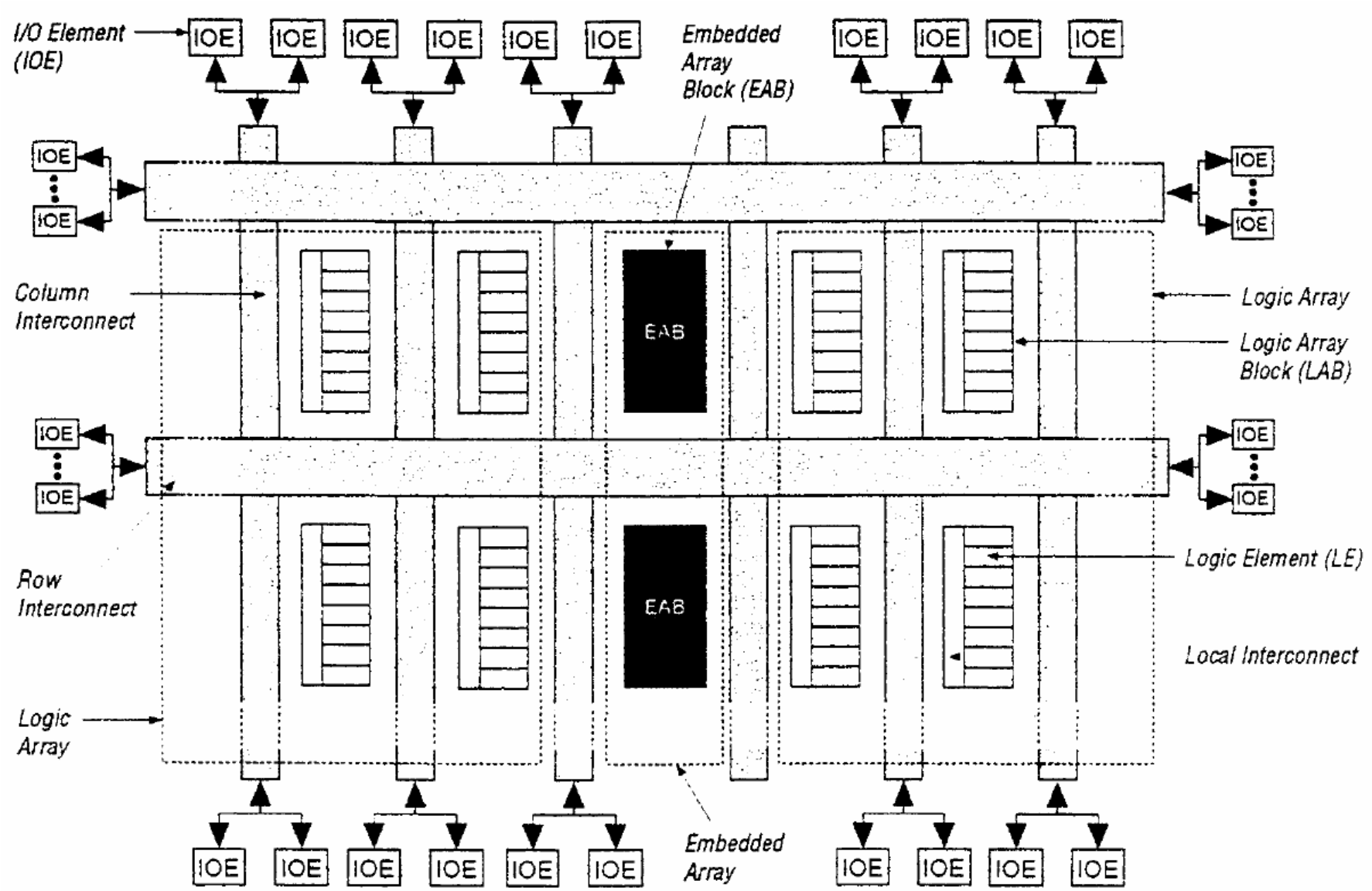
ΠΡΟΣΟΧΗ: Η αύξηση/μείωση της συχνότητας και η ένδειξή της στο αντίστοιχο led δεν γίνεται αυτόματα αλλά θα πρέπει να γραφεί από τον χρήστη κώδικας σε VHDL για την υλοποίησή της λειτουργίας αυτής.

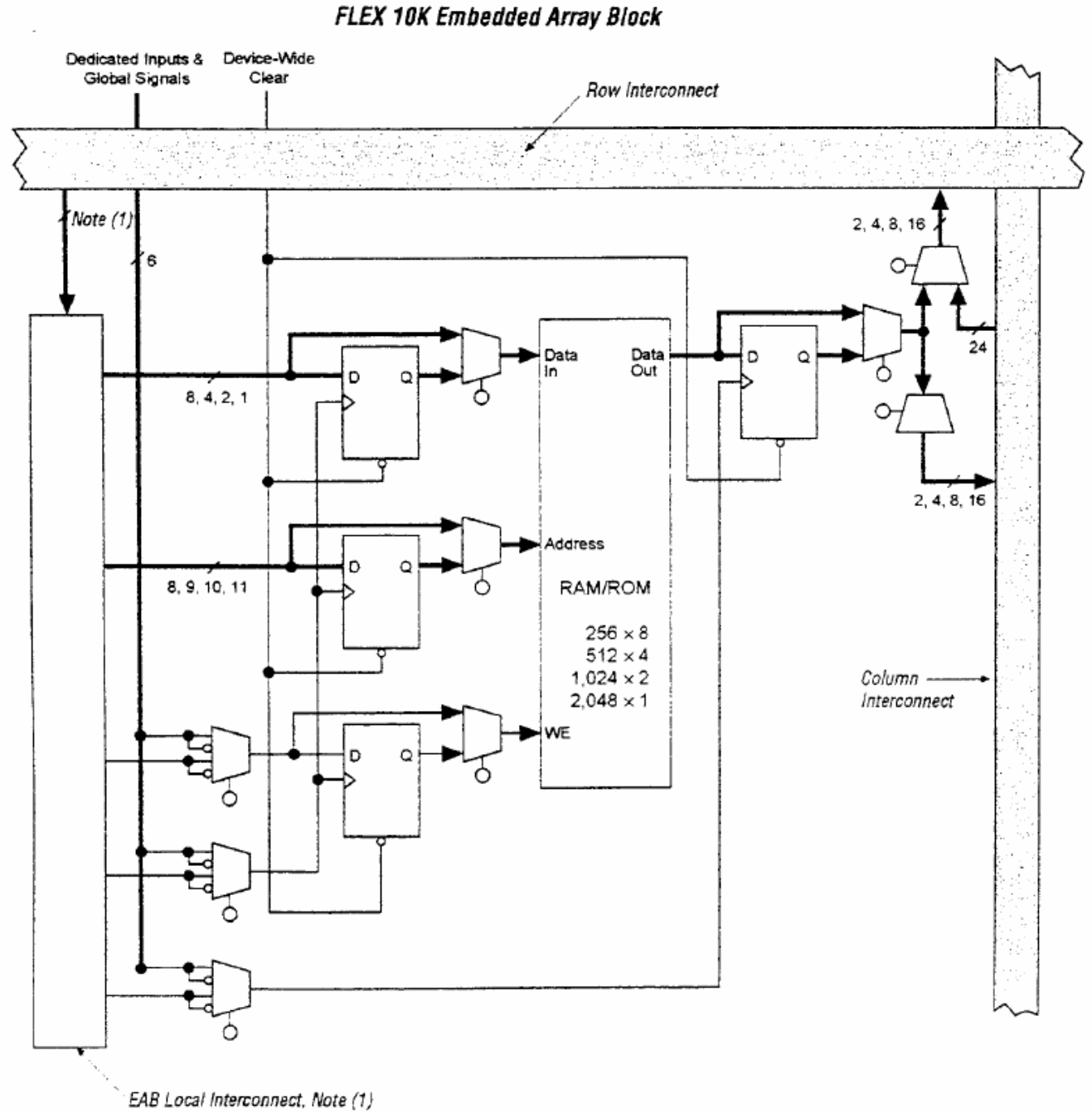
L27	L28	L29	L30	L31	L32	L33	L34
Pin23	Pin26	Pin27	Pin28	Pin29	Pin30	Pin31	Pin32
OSC	**UP**	**DOWN**	**DE1**	**DE2**	**DE3**		
Pin55	Pin121	Pin125	33	36	37		

Οι κοινές κάθοδοι των led L27 έως L34 συνδέονται στην έξοδο Y6 ενός αποκωδικοποιητή 3 προς 8 (74HCT138). Οι είσοδοι του αποκωδικοποιητή είναι τα σήματα DE1, DE2, DE3 και για την ενεργοποίηση της εξόδου Y6 θα πρέπει να έχουν τιμή DE1=0, DE2=1, DE3=1.

ΠΑΡΑΡΤΗΜΑ Γ'

Η ΕΣΩΤΕΡΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΟΙΚΟΓΕΝΕΙΑΣ FLEX10K




FLEX 10K Embedded Array Block
Dedicated Inputs & Global Signals
Device-Wide Clear
Row Interconnect
Note (1)
6
2, 4, 8, 16
24
2, 4, 8, 16
8, 4, 2, 1
8, 9, 10, 11
Data In
Data Out
Address
RAM/ROM
256 × 8
512 × 4
1,024 × 2
2,048 × 1
WE
D
Q
Column Interconnect
EAB Local Interconnect, Note (1)