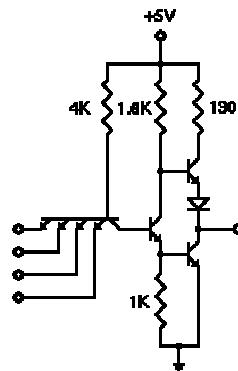
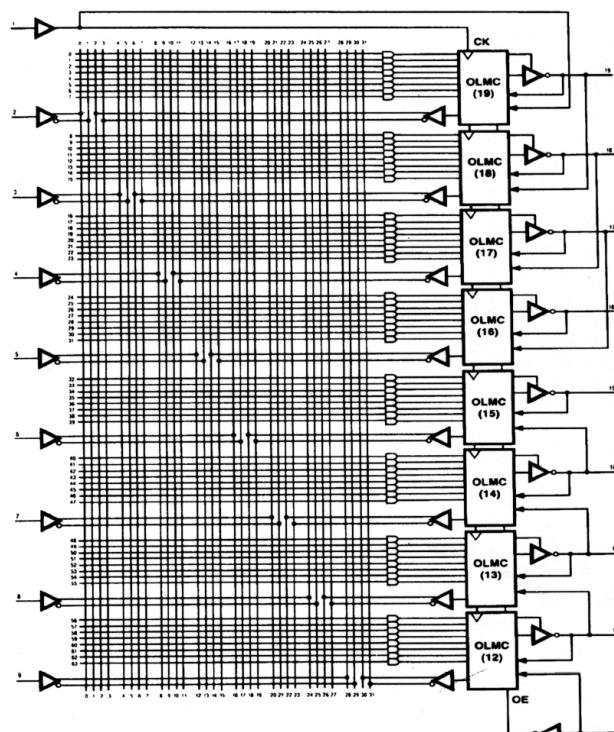


ΧΑΡΙΔΗΜΟΣ Θ. ΒΕΡΓΟΣ  
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ  
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΟΛΟΓΙΑΣ & ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ



**Πανεπιστημιακές Παραδόσεις στο μάθημα  
ΣΧΕΔΙΑΣΗ ΣΥΣΤΗΜΑΤΩΝ ΜΕ  
ΧΡΗΣΗ ΥΠΟΛΟΓΙΣΤΩΝ (Ε - CAD)**



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ  
ΕΚΔΟΣΗ 2.0  
ΑΥΓΟΥΣΤΟΣ 2004



## ΠΡΟΛΟΓΟΣ

Ενας σχεδιασμός του σήμερα για να είναι ανταγωνιστικός πρέπει πέρα από καινοτομικός να πληρεί μια σειρά από απαιτήσεις εμβαδού, χρονισμού και κατανάλωσης ισχύος. Σχεδιασμοί πολυπλοκότητας εκατομμυρίων τρανζίστορ είναι απαραίτητο να ολοκληρώνονται μέσα σε ελάχιστο χρονικό διάστημα από τη σύλληψη της ιδέας. Οι σχεδιασμοί αυτοί συνήθως περικλείουν ψηφιακά και αναλογικά κομμάτια ή/και κομμάτια που σχεδιάζονται από τρίτους σχεδιαστές. Η αγαστή συνεργασία των επιμέρους σχεδιασμών είναι μια αναγκαιότητα για τη βιωσιμότητα του σχεδιασμού όπως απαραίτητος είναι και ο πλήρης έλεγχος της λειτουργικότητας του σχεδιασμού πριν αυτός παραχθεί σε εκατομμύρια κομμάτια.

Τα σύγχρονα εργαλεία σχεδιασμού που περιγράφονται στο παρόν σύγγραμμα και με τα οποία θα έρθετε σε επαφή στο συνοδευτικό του μαθήματος εργαστήριο δεν έχουν σα στόχο να σας κάνουν καλύτερους σχεδιαστές. Στόχος τους είναι να κάνουν τη ζωή του σχεδιαστή πολύ πιο εύκολη, επιταχύνοντας τις διαδικασίες σχεδίασης, κάνοντάς τες πιο αξιόπιστες και επαναχρησιμοποιήσιμες.

Οι πανεπιστημιακές παραδόσεις που κρατάτε στα χέρια σας ζεκίνησαν να γράφονται το 1999. Ανακαινίστηκαν σημαντικά το Νοέμβριο του 2000, βάσει των υποδείξεων των πρώτων αναγνωστών (φοιτητών και μεταπτυχιακών φοιτητών), τους οποίους και ευχαριστώ θερμά. Η έκδοση 2.0 που κρατάτε στα χέρια σας είναι η πρώτη πλήρης έκδοση μιας και περικλείει όλη την ύλη που θίγουμε στο αμφιθέατρο. Επίσης διορθώνει αρκετές από τις ατέλειες της προηγούμενης, όπως αναδείχτηκαν από τις συζητήσεις με τους συναδέλφους σας.

Σκοπός των σημειώσεων είναι να αποτελέσουν ένα εργαλείο σύντομης αναδρομής από τους φοιτητές. Σαν τέτοιο, δεν μπορούν και δεν στοχεύουν άλλωστε να υποκαταστήσουν τα όσα διδάσκονται στο αμφιθέατρο κυρίως αδυνατούν να συμπεριλάβουν τις γόνιμες συζητήσεις που διεξάγονται εκεί. Εύχομαι να αποδειχθούν ένα χρήσιμο εργαλείο.

X. Βέργος



**Μέρος Α'**

**Πανεπιστημιακές Παραδόσεις**



---

# Κεφάλαιο 1

## Βασικές Γνώσεις

Στο κεφάλαιο αυτό προσπαθούμε να κάνουμε μια γρήγορη αναδρομή στα όσα έχετε συναντήσει σε διάφορα μαθήματα προηγουμένων εξαμήνων και τα οποία θα χρησιμοποιήσουμε πιο κάτω.

Ενα ψηφιακό κύκλωμα αποτελείται από δύο διαφορετικές ομάδες στοιχείων : τα συνδυαστικά κυκλώματα και τα ακολουθιακά κυκλώματα τα οποία πολύ συχνά αποκαλούνται και στοιχεία μνήμης του κυκλώματός μας.

Το συνδυαστικό υποκύκλωμα με τη σειρά του είναι μια συλλογή από μικρότερα λογικά στοιχεία, τις γνωστές σας λογικές πύλες. Αν και κάθε τεχνολογία κατασκευής χρησιμοποιεί ως δομικά στοιχεία των συνδυαστικών κυκλωμάτων διαφορετικά σύνολα από λογικές πύλες, οι συναρτήσεις αυτών των πυλών είναι κοινές σε όλες τις τεχνολογίες κατασκευής και προκύπτουν από τις λογικές συναρτήσεις των στοιχειωδών λογικών πυλών, τις οποίες εξετάσατε στο μάθημα του Λογικού Σχεδιασμού και είναι :

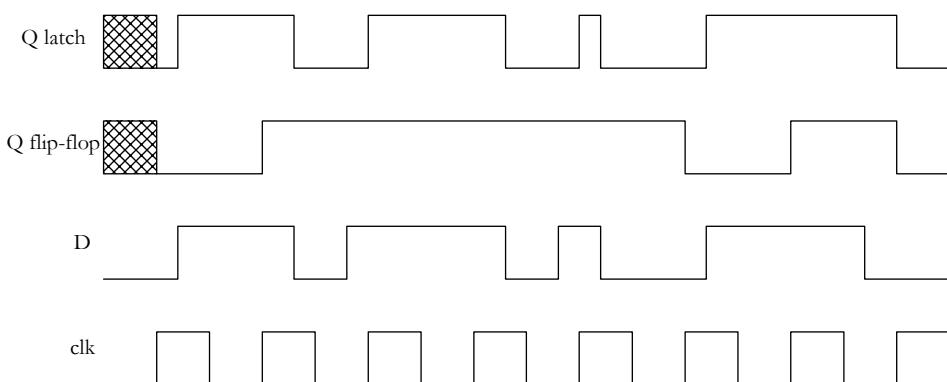
- ♦ η σύζευξη (AND). Η έξοδος αυτής της λογικής συνάρτησης είναι το 1 όταν όλες οι είσοδοι είναι στο λογικό 1.

- ♦ η διάλευξη (OR). Η έξοδος αυτής της λογικής συνάρτησης είναι το 0 όταν όλες οι είσοδοι είναι στο λογικό 0.
- ♦ η αποκλειστική διάλευξη (XOR). Η έξοδος αυτής της λογικής συνάρτησης είναι το 1 όταν περιττός αριθμός εισόδων είναι στο 1.
- ♦ η αντιστροφή (NOT). Η έξοδος αυτής της λογικής συνάρτησης είναι το 1 όταν η είσοδος είναι στο 0.
- ♦ η αντεστραμμένη σύζευξη (NAND). Η έξοδος αυτής της λογικής συνάρτησης είναι το 0 όταν όλες οι είσοδοι είναι στο λογικό 1.
- ♦ η αντεστραμμένη διάλευξη (OR). Η έξοδος αυτής της λογικής συνάρτησης είναι το 1 όταν όλες οι είσοδοι είναι στο λογικό 0.
- ♦ η ισοδυναμία (XNOR). Η έξοδος αυτής της λογικής συνάρτησης είναι το 1 όταν άριθμος εισόδων είναι στο 1.

Τα ακολουθιακά υποκυλώματα κατασκευάζονται από δύο διαφορετικές κατηγορίες λογικών στοιχείων (ή ισοδύναμα στοιχείων μνήμης). Οι κατηγορίες αυτές είναι οι μανταλωτές (latches) και τα φλιπ-φλοπς (flip-flops). Από τα διαφορετικά στοιχεία αυτών των οικογενειών (π.χ. T, D, JK κλπ flip-flops) στο μάθημα αυτό θα μας απασχολήσουν μόνο δύο στοιχεία :

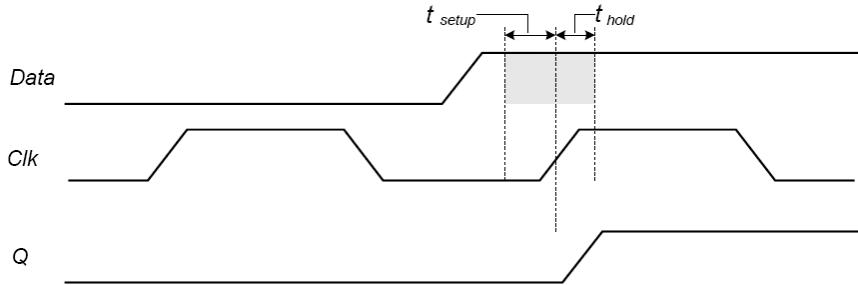
1. Ο μανταλωτής τύπου D και
2. Το flip-flop τύπου D.

Τα στοιχεία αυτά είναι στοιχεία μνήμης καθότι διατηρούν την έξοδό τους όταν δε λαμβάνουν σήμα ανανέωσης, όταν δηλαδή το ρολόι τους δεν είναι ενεργό. Οταν το σήμα ενεργοποίησής τους είναι ενεργό αντιγράφουν την είσοδό τους (D) στην έξοδό τους (Q). Η διαφορά μεταξύ αυτών των δύο στοιχείων έγκειται στο πως ενεργοποιούνται και πιο συγκεκριμένα, ο μανταλωτής είναι ενεργός καθ' όλη τη διάρκεια που το δυναμικό του ρολογιού είναι στην ενεργή στάθμη ενώ το flip-flop μόνο κατά την ακμή ενεργοποίησής. Ας δούμε στο παρακάτω σχήμα τη συμπεριφορά ενός flip-flop και ενός μανταλωτή που λαμβάνουν την ίδια είσοδο D και το ίδιο σήμα ρολογιού (clk).



Οπως βλέπουμε και από το σχήμα (τα σκιασμένα τετράγωνα υποδεικνύουν απροσδιοριστία) η είσοδος του μανταλωτή αντιγράφεται στην έξοδό του (Qlatch) καθ' όλη τη διάρκεια που το ρολόι μας είναι ενεργό, δηλαδή όσο βρίσκεται στο λογικό 1. Αντίθετα η είσοδος D δειγματοληπτείται με ηάθε ανοδική ακμή του ρολογιού στην περίπτωση του flip-flop, το δείγμα μεταφέρεται στην έξοδό του και η έξοδός του παραμένει σταθερή παρά τις όποιες αλλαγές της εισόδου κατά τη διάρκεια που το ρολόι είναι στο λογικό 1. Συνοπτικά θα μπορούσαμε να πούμε ότι ο μανταλωτής είναι ενεργοποιούμενος με δυναμικό ενώ το flip-flop ακμοπυροδότητο.

Ενα άλλο πρόβλημα είναι τι γίνεται όταν η ενεργοποίηση / απενεργοποίηση ενός στοιχείου μνήμης συμβεί ταυτόχρονα με την αλλαγή στην είσοδό του. Γνωρίζετε ότι κάτι τέτοιο μπορεί να αποβεί μοιραίο για την αναμενόμενη λειτουργία του νυκλώματός σας, καθώς εισάγει απροσδιοριστία. Για την αποφυγή αυτών των προβλημάτων, σε όλα τα ακολουθιακά στοιχεία ορίζονται δύο χρόνοι, ο χρόνος προετοιμασίας και ο χρόνος αποκατάστασης οι οποίοι φαίνονται στο σχήμα που ακολουθεί και ορίζονται ως :



Χρόνος Προετοιμασίας ( $t_{\text{setup}}$ ) :

είναι η χρονική διαφορά μεταξύ της αρχής δειγματοληψίας (συνήθως οριζόμενη στο 50% της αλλαγής του ρολογιού) της εισόδου δεδομένων (Data) και της αρχότερης χρονικής στιγμής που αυτή πρέπει να πάρει σταθερή τιμή ώστε να μη προκληθεί απροσδιοριστία.

Χρόνος Αποκατάστασης ( $t_{\text{hold}}$ ) :

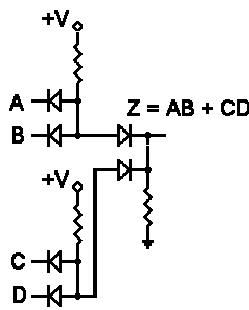
είναι η χρονική διαφορά μεταξύ της αρχής δειγματοληψίας της εισόδου δεδομένων (Data) και της χρονικής στιγμής που αυτή μπορεί να πάρει διαφορετική τιμή από αυτήν της δειγματοληψίας χωρίς να προκαλέσει απροσδιοριστία.

Αυτοί οι δύο χρόνοι ορίζουν τις χρονικές απαιτήσεις της εισόδου δεδομένων ενός στοιχείου μνήμης σε σχέση με το ρολόι του και καθορίζουν ένα παράθυρο χρόνου μέσα στο οποίο η είσοδος δεδομένων πρέπει να πάρει τη τελική της τιμή και να τη διατηρήσει αναλλοίωτη έτσι ώστε να αποφευχθεί απροσδιοριστία της εξόδου. Ο χρόνος  $t_{\text{setup}}$  είναι πάντοτε μεγαλύτερος του  $t_{\text{hold}}$ . Στην περίπτωση των flip-flops η

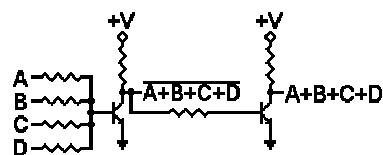
σχέση είναι περίπου 2:1. (Υποθέσαμε παραπάνω ότι τα στοιχεία μνήμης ενεργοποιούνται με το θετικό παλμό / ακμή του ρολογιού, αλλά πρέπει να είναι προφανές ότι τα ίδια ισχύουν στην περίπτωση ενεργοποίησης με τον αρνητικό παλμό / ακμή του ρολογιού.)

Σε προηγούμενα μαθήματα αλλά και σε διάφορα μαθήματα επιλογής που τυχόν έχετε παρακολουθήσει, έχετε διδαχθεί ότι σήμερα υπάρχουν πολλές διαφορετικές οικογένειες ολοκληρωμένων κυκλωμάτων. Παρακάτω συνοψίζονται οι περισσότερες από αυτές :

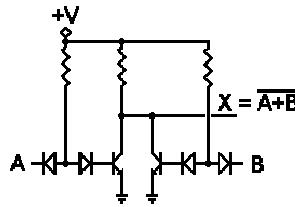
- ♦ Λογική Διόδων (Diode Logic). Οι δίοδοι (με την ιδιότητά τους να περνούν το ηλεκτρικό ρεύμα προς τη μία κατεύθυνση) χρησιμοποιούνται σε αυτή την οικογένεια για τις λογικές συναρτήσεις. Η γρήγορη εξασθένηση των σημάτων λόγω της τάσης πόλωσης των διόδων και η αδυναμία δημιουργίας συμπληρωματικών συναρτήσεων οδήγησε στη σπάνια χρήση αυτής της οικογένειας. Ενα παράδειγμα σύνθετης πύλης αυτής της οικογένειας είναι το παρακάτω :



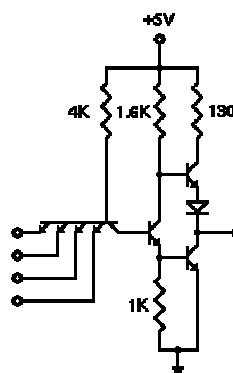
- ♦ Λογική Αντιστάτη – Τρανζίστορ (Resistor-Transistor Logic (RTL)). Στην οικογένεια αυτή χρησιμοποιούνται τρανζίστορ για το συνδυασμό πολλών εισόδων που παράλληλα ενισχύουν και αντιστρέφουν το σήμα. Το τρανζίστορ εξόδου αντιστρέφει και πάλι το σήμα. Η κατανάλωση ισχύος αυτών των πυλών μαζί με τη χαμηλή ταχύτητά τους είναι τα σοβαρότερα προβλήματά τους και οι λόγοι που δε χρησιμοποιούνται ευρέως. Το παρακάτω σχήμα δείχνει ένα παράδειγμα σύνθετης πύλης αυτής της οικογένειας.



- ♦ Λογική διόδου – τρανζίστορ (Diode-Transistor Logic (DTL)). Οι πύλες αυτής της οικογένειας χρησιμοποιούν διόδους για τη δημιουργία των λογικών συναρτήσεων. Οι έξοδοι οδηγούνται σε τρανζίστορ για την αποκατάσταση των τάσεων και τη τυχόν δημιουργία συμπληρωματικής λογικής.



- Λογική τρανζίστορ (Transistor-Transistor Logic (TTL)). Μια από τις πλέον διαδεδομένες οικογένειες ψηφιακών κυκλωμάτων προκύπτει από την αντικατάσταση των διόδων της DTL με τρανζίστορ πιθανά πολλαπλών εκπομπών. Το μεγαλύτερο πρόβλημα αυτής της οικογένειας πυλών είναι η μικρή οδηγητική ικανότητα, η σημαντική κατανάλωση ισχύος και οι μικρές για την εποχή μας ταχύτητες. Διάφορες υποοικογένειες της ίδιας τεχνολογίας προσπάθησαν να λύσουν κάποια από τα παραπάνω προβλήματα με τη χρήση διαφορετικών τιμών αντιστάσεων ή / και τρανζίστορ Schottky. Στην οικογένεια αυτή λοιπόν βρίσκουμε τις παρακάτω υποοικογένειες :



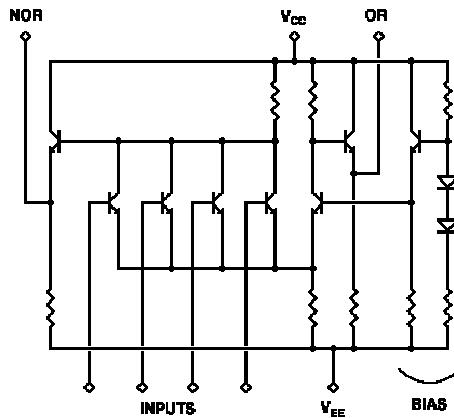
- Βασική σειρά (Standard). Το παραπάνω σχήμα δείχνει ένα παράδειγμα της πύλης NAND τεσσάρων εισόδων αυτής της υποοικογένειας.
- Σειρά χαμηλής κατανάλωσης (Low-Power Schottky-LS).
- Γρήγορη σειρά (Schottky-S).
- Σειρά ενδιάμεσης ταχύτητας και χαμηλής κατανάλωσης (Advanced Low-Power Schottky-ALS).
- Πολύ γρήγορη σειρά (Advanced Schottky-AS).

Ολες αυτές οι υποοικογένειες συχνά αναφέρονται ως η σειρά TTL 7400, από το κωδικό των ολοκληρωμένων κυκλωμάτων που τις υλοποιούν. Για παράδειγμα το ολοκληρωμένο με κωδικό 7404 είναι ένα ολοκληρωμένο που περιέχει έξι (6) αντιστροφείς TTL. Ενδιάμεσα γράμματα υποδεικνύουν την υποοικογένεια στην οποία ανήκει το εκάστοτε ολοκληρωμένο :

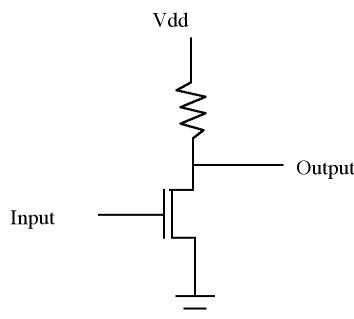
7404	Αντιστροφείς βασικής σειράς
74LS04	Αντιστροφείς σειράς χαμηλής κατανάλωσης
74S04	Αντιστροφείς γρήγορης σειράς, κ.ο.κ.

Αξίζει επίσης να θυμηθούμε ότι στη TTL τεχνολογία οι δύο στάθμες δυναμικών είναι 0 έως 0.8 Volt για το λογικό 0 και 2.7 έως 5 Volt για το λογικό 1.

- ♦ Λογική Συζευγμένου εκπομπού (Emitter-Coupled Logic (ECL)). Η οικογένεια αυτή που είναι γνωστή ως και λογική ελέγχου ρεύματος (Current Mode Logic (CML)), στοχεύει την μέγιστη συχνότητα λειτουργίας ενός ψηφιακού κυκλώματος με το να εμποδίζει κάθε τρανζίστορ να εισέρχεται βαθιά στον κόρο. Δυστυχώς αυτό σημαίνει και πολύ μεγάλα ποσά ρεύματος για την ορθή λειτουργία του κάθε στοιχείου και συνεπώς αυξημένη κατανάλωση ισχύος. Το παρακάτω σχήμα δείχνει ένα παράδειγμα πύλης αυτής της οικογένειας.

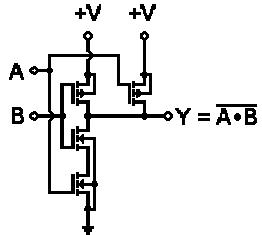


- ♦ Λογική NMOS / PMOS. Στις οικογένειες αυτές αντί για διπολικά τρανζίστορ γίνεται χρήση τρανζίστορ πεδίου (Metal Oxide Semiconductor Field Effect Transistor – MOSFET) με φορείς τύπου n στην περίπτωση των NMOS ή p στην περίπτωση των PMOS. Τα MOSFET δίνουν μεγαλύτερες δυνατότητες για μεγάλης ηλιμακας ολοκλήρωση με την οποία μπορούμε να επιτύχουμε μεγάλες ταχύτητες. Ωστόσο η διαρκής κατανάλωση ρεύματος πάνω στον αντιστάτη ακόμα και όταν δεν υπάρχουν εναλλαγές της εισόδου οδήγησε κι αυτή την οικογένεια σε μαρασμό. Το παρακάτω σχήμα παρουσιάζει έναν αντιστροφέα σε τεχνολογία NMOS.



- ♦ Λογική CMOS (Complementary Metal Oxide Semiconductor Logic). Πολλές εφαρμογές πέρα από την υψηλή ταχύτητα εισάγουν την απαίτηση για χαμηλή κατανάλωση ισχύος. Τέτοιες εφαρμογές είναι αυτές που σχετίζονται με φορητές

συσκευές που συνήθως λειτουργούν με μπαταρίες. Η λογική των στοιχείων CMOS βασίζεται στη χρήση τρανζίστορ προσαύξησης (enhancement mode transistors), συχνά συμπληρωματικής λογικής.



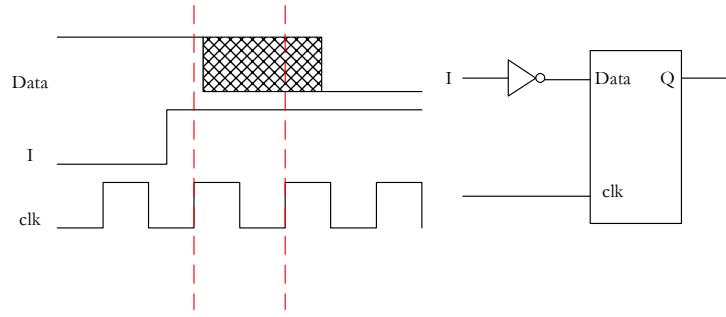
Αντίστοιχα με την οικογένεια TTL, έτσι και στην οικογένεια CMOS υπάρχουν διάφορες υποοικογένειες που έχουν σα στόχο κυρίως τη μεγαλύτερη ταχύτητα. Το σύνολο των ολοκληρωμένων αυτών των υποοικογενειών απαρτίζουν τη σειρά 74C00 όπου και πάλι τα γράμματα χρησιμοποιούνται για να ξεχωρίσουν τα ολοκληρωμένα συγκεκριμένης υποοικογένειας. Τα ολοκληρωμένα αυτά μάλιστα φτιάχτηκαν έτσι ώστε να μπορούν να αντικαταστήσουν τα αντίστοιχα της σειράς 7400, προσφέροντας παράλληλα μικρότερη κατανάλωση ισχύος. Ετσι για παράδειγμα το ολοκληρωμένο 74ACT04 είναι ένας άμεσος αντικαταστάτης του 74ALS04. Αυτό φυσικά επιτρέπεται γιατί τα δυναμικά εισόδου για την τεχνολογία CMOS είναι συμβατά με τις εξόδους των TTL ενώ και τα δυναμικά εξόδων της λογικής CMOS είναι συμβατές είσοδοι για τη λογική TTL.

Η οικογένεια CMOS πέρα από έτοιμα ολοκληρωμένα μικρής και μεσαίας κλίμακας ολοκλήρωσης είναι η μοναδική χρησιμοποιούμενη σήμερα λογική οικογένεια για ψηφιακά κυκλώματα πολύ μεγάλης κλίμακας ολοκλήρωσης (VLSI). Πέρα από έτοιμα ολοκληρωμένα, παρέχονται για τις διάφορες διαθέσιμες τεχνολογίες βιβλιοθήκες πυλών, στοιχείων μνήμης αλλά και έτοιμων σχεδιαστικών κομματών, τις οποίες μπορούμε να χρησιμοποιήσουμε για έναν πολύ μεγάλο σχεδιασμό του οποίο θα υλοποιήσουμε σαν ένα ολοκληρωμένο.

Μεγάλο ρόλο στην επιλογή του ολοκληρωμένου κάποιας από τις παραπάνω λογικές οικογένειες παίζει ο χρόνος υπολογισμού των λογικών συναρτήσεων που μπορεί να προσφέρει. Ετσι λοιπόν ανεξάρτητα από την λογική οικογένεια, κάθε ολοκληρωμένο κύκλωμα / σύστημα που κατασκευάζεται και διατίθεται στην αγορά υποχρεωτικά συνοδεύεται από ένα εγχειρίδιο που πέρα από τις λογικές συναρτήσεις θα πρέπει να παρέχει και αναλυτικές μετρήσεις του χρόνου καθυστέρησης. Οπως ήδη γνωρίζουμε όμως ο χρόνος αυτός δεν είναι σταθερός, αλλά εξαρτάται :

1. Από τη θερμοκρασία
2. Από τη τάση τροφοδοσίας
3. Από τη χωρητικότητα που οδηγεί το κύκλωμα και

Για το λόγο αυτό αντί μιας και μοναδικής τιμής συνήθως παρέχονται μια τριάδα τιμών {ελάχιστη, τυπική, μέγιστη} που μας δίνει τις καθυστερήσεις στην καλύτερη, τη συνήθη και τη χειρότερη περίπτωση αντίστοιχα. Προσέξτε ότι το γεγονός αυτό εισάγει έναν επιπλέον παράγοντα αποσδιοριστίας σε έναν σχεδιασμό. Το παρακάτω σχήμα δείχνει την έξοδο ενός αντιστροφέα που χρησιμοποιείται σαν είσοδος δεδομένων σε ένα flip-flop.



Ανάλογα με τη καθυστέρηση του αντιστροφέα, η είσοδος Data του flip-flop μπορεί να πάει στο λογικό 0 από λίγο μετά από τη δεύτερη ανοδική ακμή του ρολογιού στο σχήμα έως και μιάμιση περίοδο αργότερα. Εστω  $Q_2$  και  $Q_3$  η έξοδος του flip-flop μετά αντίστοιχα τη δεύτερη και τη τρίτη ανοδική ακμή. Ο σχεδιαστής ελπίζει στη τυπική καθυστέρηση του αντιστροφέα που θα οδηγήσει σε  $Q_2 = 1$ , και  $Q_3 = 0$ , χωρίς προβλήματα. Οπως όμως βλέπουμε από το σχήμα καμία από αυτές τις καταστάσεις δε μπορεί να θεωρηθεί σίγουρη. Αν ο αντιστροφέας λειτουργήσει με την ελάχιστη καθυστέρηση το  $Q_2 = 1$  κινδυνεύει από τη μη ύπαρξη αρκετού χρόνου αποκατάστασης. Αν πάλι ο αντιστροφέας λειτουργήσει με καθυστέρηση σημαντικά μεγαλύτερη της τυπικής το  $Q_3 = 0$  κινδυνεύει τόσο από έλλειψη χρόνου προετοιμασίας όσο και από έλλειψη χρόνου αποκατάστασης. Η μεγάλη αποσδιοριστία κάνει το παραπάνω κύκλωμα επισφαλές και συνεπώς απαράδεκτο.

Οπως αναλύθηκε προηγούμενα, στην περίπτωση της τεχνολογίας CMOS στην οποία παρέχεται στο σχεδιαστή η δυνατότητα για ολοκλήρωση πολύ μεγάλης κλίμακας, η παραπάνω περιγραφείσα πληροφορία για τη καθυστέρηση είναι φοβερά ανεπαρκής. Μιας και οι είσοδοι κάθε πύλης αυτής της τεχνολογίας δεν είναι ισοδύναμες από πλευράς καθυστέρησης καθώς και η καθυστέρηση μεταβάλλεται σημαντικά ανάλογα με τον αριθμό των οδηγούμενων πυλών, οι βιβλιοθήκες αυτής της τεχνολογίας συνήθως περιγράφουν την καθυστέρηση της εξόδου ανεξάρτητα για κάθε είσοδο. Επίσης περιλαμβάνουν τις διαφορετικές καθυστερήσεις ανάλογα με το οδηγούμενο φορτίο αλλά και τη δυνατότητα για πύλες με διαφορετικές ικανότητες οδήγησης. Ολη αυτή η πληροφορία είναι απαραίτητη για έναν σωστό σχεδιασμό ιδιαίτερα όταν αυτός γίνεται με αυτοματοποιημένο τρόπο μιας και τα εργαλεία έχουν τη δυνατότητα να εξετάζουν όλες αυτές τις πιθανότητες με πολύ μεγάλη ταχύτητα.

---

Θα κλείσουμε αυτή τη σύντομη αναδρομή μας με αναφορά στις διάφορες τεχνοτροπίες σχεδιασμού ενός κυκλώματος σαν ένα ολοκληρωμένο, που συνήθως θα υλοποιηθεί σε CMOS τεχνολογία. Ο σχεδιαστής πρέπει να επιλέξει κάποια από αυτές τις τεχνοτροπίες όταν ο στοχευόμενος σχεδιασμός δεν είναι διαθέσιμος σαν ένα εμπορικό ολοκληρωμένο, ή όταν τα διαθέσιμα δεν καλύπτουν κάποιες ή όλες από τις ανάγκες του σχεδιαστή για ταχύτητα, εμβαδόν και κατανάλωση ισχύος. Οι διαθέσιμες τεχνοτροπίες είναι :

- ◆ Πλήρως εξειδικευμένος σχεδιασμός (Full – custom design). Στην τεχνοτροπία αυτή ο σχεδιαστής θα πρέπει να σχεδιάσει εξ αρχής, ακόμη και τα βασικά δομικά στοιχεία, δηλαδή τις πύλες και τα στοιχεία μνήμης του σχεδιασμού του. Παρότι αυτός ο τρόπος παρέχει τη μέγιστη ευελιξία στον σχεδιαστή, είναι προφανές ότι είναι και ο πλέον επίπονος χρονικά. Επιπλέον η πιθανότητα για σχεδιαστικά λάθη είναι πολύ μεγάλη και το κόστος κατασκευής πολύ υψηλό. Η τεχνοτροπία αυτή σήμερα χρησιμοποιείται για μικρούς σχετικά σχεδιασμούς με πολύ αυξημένες απαιτήσεις σε ταχύτητα, εμβαδόν και κατανάλωση ισχύος.
- ◆ Μερικά εξειδικευμένος σχεδιασμός (Semi-custom design). Στην τεχνοτροπία αυτή παρότι μερικά σχεδιαστικά κομμάτια παρέχονται έτοιμα σα μια βιβλιοθήκη, ο σχεδιαστής έχει τη δυνατότητα αν δε καλύπτεται να σχεδιάσει τα δικά του και μετά να φτιάξει το σχεδιασμό του σα μίγμα έτοιμων και νέων υποσχεδιασμών. Προφανώς η ευελιξία που δίνεται στο σχεδιαστή είναι αντίστοιχη με αυτήν της προηγούμενης περίπτωσης, αλλά επίσης ο χρόνος σχεδιασμού μπορεί να μικρύνει σημαντικά. Χρησιμοποιώντας κατά πλειοψηφία έτοιμα σχεδιαστικά κομμάτια, η πιθανότητα σχεδιαστικών λαθών μικραίνει..
- ◆ Σχεδιασμός με έτοιμα σχεδιαστικά κομμάτια (Standard-cell design). Στην τεχνοτροπία αυτή ο σχεδιαστής εφοδιάζεται με μια σχεδιαστική βιβλιοθήκη και εκφράζει το σχεδιασμό του σαν την διασύνδεση στοιχείων αυτής της βιβλιοθήκης. Προφανώς η βιβλιοθήκη αυτή παρέχεται από τον τελικό κατασκευαστή του ολοκληρωμένου και μπορεί να περιέχει από πολύ λίγα έως πάρα πολλά και πολύ σύνθετα σχεδιαστικά κομμάτια. Σύμφωνα με τα όσα έχετε μάθει από το μάθημα της ψηφιακής σχεδίασης η βιβλιοθήκη θα μπορούσε να περιέχει μόνο τη πύλη NAND δύο εισόδων μιας και κάθε λογική συνάρτηση μπορεί να εκφραστεί βάσει αυτής. Ωστόσο για τη διευκόλυνση των σχεδιαστών οι βιβλιοθήκες που παρέχονται σήμερα περιέχουν όλες τις λογικές πύλες (και μάλιστα σε διάφορες εκδόσεις ταχύτητας, εμβαδού και οδηγητικής μακρότητας), στοιχεία μνήμης, μικρά έως μεσαία συνδυαστικά κυκλώματα (αθροιστές, πολλαπλασιαστές, κλπ), μικρά έως μεσαία ακολουθιακά κυκλώματα (καταχωρητές, ολισθητές, μετρητές κλπ). Το

---

μεγάλο πλεονέκτημα χρησιμοποίησης αυτής της τεχνοτροπίας είναι προφανώς ο χρόνος ολοκλήρωσης του σχεδιασμού. Ωστόσο, ζεφεύγει πλέον από τα χέρια του σχεδιαστή η δυνατότητα καθορισμού των ηλεκτρικών χαρακτηριστικών των στοιχειωδών σχεδιαστικών κομμάτιών, με αποτέλεσμα τόσο οι μέγιστες ταχύτητες, όσο και το ελάχιστο εμβαδόν και η ελάχιστη κατανάλωση ισχύος που μπορεί να επιτευχθεί να μη μπορούν να καθοριστούν άμεσα από αυτόν. Σημειώνουμε ότι στις μέρες μας ο σχεδιασμός με έτοιμα σχεδιαστικά κομμάτια είναι η κυρίαρχη τεχνοτροπία σχεδιασμού.

Ενας σχεδιασμός από έτοιμα σχεδιαστικά κομμάτια μπορεί να υλοποιηθεί σαν ολοκληρωμένο με διάφορους τρόπους. Οι τρόποι αυτοί στην ουσία καθορίζουν και το κόστος κατασκευής του ολοκληρωμένου. Παρακάτω αναφέρονται μερικοί από τους πλέον διαδεδομένους:

1. **Υλοποίηση σαν σχεδιασμός από έτοιμα σχεδιαστικά κομμάτια (Standard-cell implementation).** Η υλοποίηση αυτή συνήθως χρησιμοποιεί την πλέον πρόσφατη τεχνολογία και απαιτεί την χρησιμοποίηση ενός πολύ μεγάλου αριθμού μασκών οι οποίες προκύπτουν από φωτολιθογραφικές μεθόδους. Με αυτό το τρόπο μπορούμε να εκμεταλλευτούμε πλήρως τις δυνατότητες της κάθε τεχνολογίας, να επιτύχουμε τη μέγιστη ταχύτητα που μας προσφέρει, αλλά ταυτόχρονα θα πρέπει να είμαστε προετοιμασμένοι να πληρώσουμε υψηλό τίμημα. Υπολογίζεται ότι το πρώτο ολοκληρωμένο σε αυτό το τρόπο υλοποίησης μπορεί να κοστίσει \$200.000 και κάθε επόμενο σημαντικά λιγότερο. Αυτό προκύπτει από το μεγάλο κόστος κατασκευής των μασκών και συνεπώς αυτός ο τρόπος υλοποίησης είναι ελκυστικός μόνο όταν ο αριθμός των ολοκληρωμένων που θα κατασκευαστεί είναι τουλάχιστον 20.000. Επίσης ο χρόνος παράδοσης θα πρέπει να υπολογίζεται σε 1,5 έως 2 μήνες.
2. **Απεικόνιση σε πίνακα πυλών (Gate Array, Sea of Gates, etc.).** Αντί να φτιάχνονται όλες οι μάσκες από την αρχή για το ολοκληρωμένο που σχεδιάσαμε, υπάρχουν ολοκληρωμένα στα οποία ήδη έχει υλοποιηθεί ένας πίνακας από τρανζίστορ. Αυτά τα ολοκληρωμένα έχουν κατασκευαστεί σε εξαιρετικά μεγάλες ποσότητες και συνεπώς το κόστος τους είναι πολύ χαμηλό. Ο σχεδιασμός μας εκφράζεται σαν τη διασύνδεση αυτών των υπαρχόντων τρανζίστορ, ενώ κάποια τρανζίστορ μπορεί να μείνουν και αχρησιμοποίητα. Συνήθως ένα ποσοστό χρήσης των υπαρχόντων τρανζίστορ της τάξης του 70-80% είναι εξαιρετικό αν μπορεί να επιτευχθεί. Με αυτό το τρόπο υλοποίησης απαιτούνται να κατασκευαστούν μάσκες μόνο για τη διασύνδεση, γεγονός που περιορίζει σημαντικά το κόστος αλλά και το χρόνο κατασκευής του ολοκληρωμένου (1 έως 2 εβδομάδες). Η απόδοση του τελικού

---

προϊόντος μπορεί να είναι αρκετά καλή, αλλά προφανώς χειρότερη από αυτήν του προηγούμενου τρόπου υλοποίησης. Αυτός ο τρόπος κατασκευής είναι ελκυστικός όταν ο αριθμός των ολοκληρωμένων που θα κατασκευαστεί κυμαίνεται μεταξύ 1.000 έως 25.000.

3. Απεικόνιση σε προγραμματιζόμενο πίνακα πυλών (LPGA, FPGA, etc.). Αυτός ο τρόπος κατασκευής είναι εξέλιξη του προηγουμένου και έχει σκοπό τη περαιτέρω μείωση του χρόνου και του κόστους κατασκευής. Βασίζεται σε ολοκληρωμένα που πέρα από τρανζίστορ, περιέχουν και προ-υλοποιημένο πίνακα πιθανών διασυνδέσεων μεταξύ τους. Ο σχεδιασμός μας εκφράζεται με τη χρήση κάποιων από αυτά τα τρανζίστορ και κάποιων από τις πιθανές διασυνδέσεις ή με άλλα λόγια με την απεικόνιση του στοχευόμενου σχεδιασμού πάνω στους υλοποιημένους πίνακες. Η απεικόνιση αυτή στην περίπτωση των LPGA (Laser Programmable Gate Arrays) γίνεται με τη χρήση laser, γεγονός που μεταφράζεται σε χρόνο υλοποίησης λιγότερο της μίας εβδομάδας. Στην περίπτωση των FPGA (Field Programmable Gate Arrays) η στοχευόμενη απεικόνιση μπορεί να διαβαστεί από μια εξωτερική μνήμη και συνεπώς ο χρόνος υλοποίησης στην ουσία καταλήγει στον προγραμματισμό αυτής της μνήμης, δηλαδή είναι της τάξης των μερικών λεπτών και μπορεί να γίνει από τον ίδιο το σχεδιαστή. Μιας και η χρήση αυτών των ολοκληρωμένων μας επιτρέπει την πολύ γρήγορη πρωτότυποποίηση των σχεδιασμών μας, τα ολοκληρωμένα αυτά εξετάζονται με πολύ μεγάλη λεπτομέρεια σε επόμενο κεφάλαιο.



---

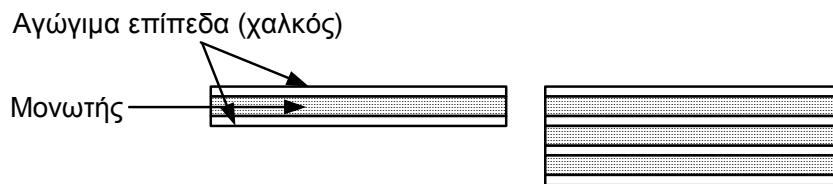
## Κεφάλαιο 2

# PCBs και PLDs

Στο πρώτο μέρος αυτού του κεφαλαίου εξετάζουμε λεπτομερειακά το τι απαιτείται από το σχεδιαστή ώστε να καταλήξουμε σε ένα σχεδιασμό υλοποιημένο σε μία πλακέτα (Printed Circuit Board – PCB). Ανεξάρτητα αν κάποιο μέρος ή και όλος ο σχεδιασμός μας υλοποιηθεί σε ένα και μόνο ολοκληρωμένο κύκλωμα, αυτό το ολοκληρωμένο δε μπορεί να θεωρηθεί σα τελικό προϊόν. Αργά ή γρήγορα θα χρειαστεί να σχεδιαστεί μια πλακέτα στην οποία θα τοποθετηθεί αυτό το ολοκληρωμένο μόνο του ή και με άλλα ολοκληρωμένα ώστε να αποτελέσει το τελικό σύστημα. Είναι προφανές λοιπόν ότι ο σχεδιασμός μιας πλακέτας ήταν και θα είναι πάντα επίκαιος.

Ας δούμε πρώτα το πως κατασκευάζεται μια πλακέτα έτσι ώστε να καταλήξουμε στο τι θα πρέπει να παραδώσει ο σχεδιαστής για τη κατασκευή της. Μια πλακέτα αποτελείται από ένα ή περισσότερα φύλλα, τα οποία είναι επιστρωμένα (συνήθως με χαλκό) και από τις δύο πλευρές τους. Στην περίπτωση των περισσοτέρων του ενός φύλλων χαλκού, αυτά προσκολλώνται μεταξύ τους αφού πρώτα εναποτεθεί ενδιάμεσα ένα στρώμα μονωτικού υλικού. Τα φύλλα αυτά είναι διαθέσιμα σε διάφορα πάχη των

οποίων το κόστος αυξάνει σημαντικά καθώς κινούμαστε προς τα πιο λεπτά φύλλα. Ο αριθμός των επιφανειών χαλκού (αγώγιμων επιφανειών) είναι προφανώς διπλάσιος του αριθμού των φύλλων που θα χρησιμοποιήσουμε. Από πρακτικής άποψης (με στόχο δηλαδή τη διατήρηση του συνολικού πάχους σε επίπεδα καθημερινής χρήσης) χρησιμοποιούνται το πολύ τέσσερα φύλλα, προσφέροντας το πολύ οκτώ αγώγιμα επίπεδα. Το παρακάτω σχήμα δείχνει τη κάθετη τομή μιας πλακέτας δύο και τεσσάρων επιπέδων.

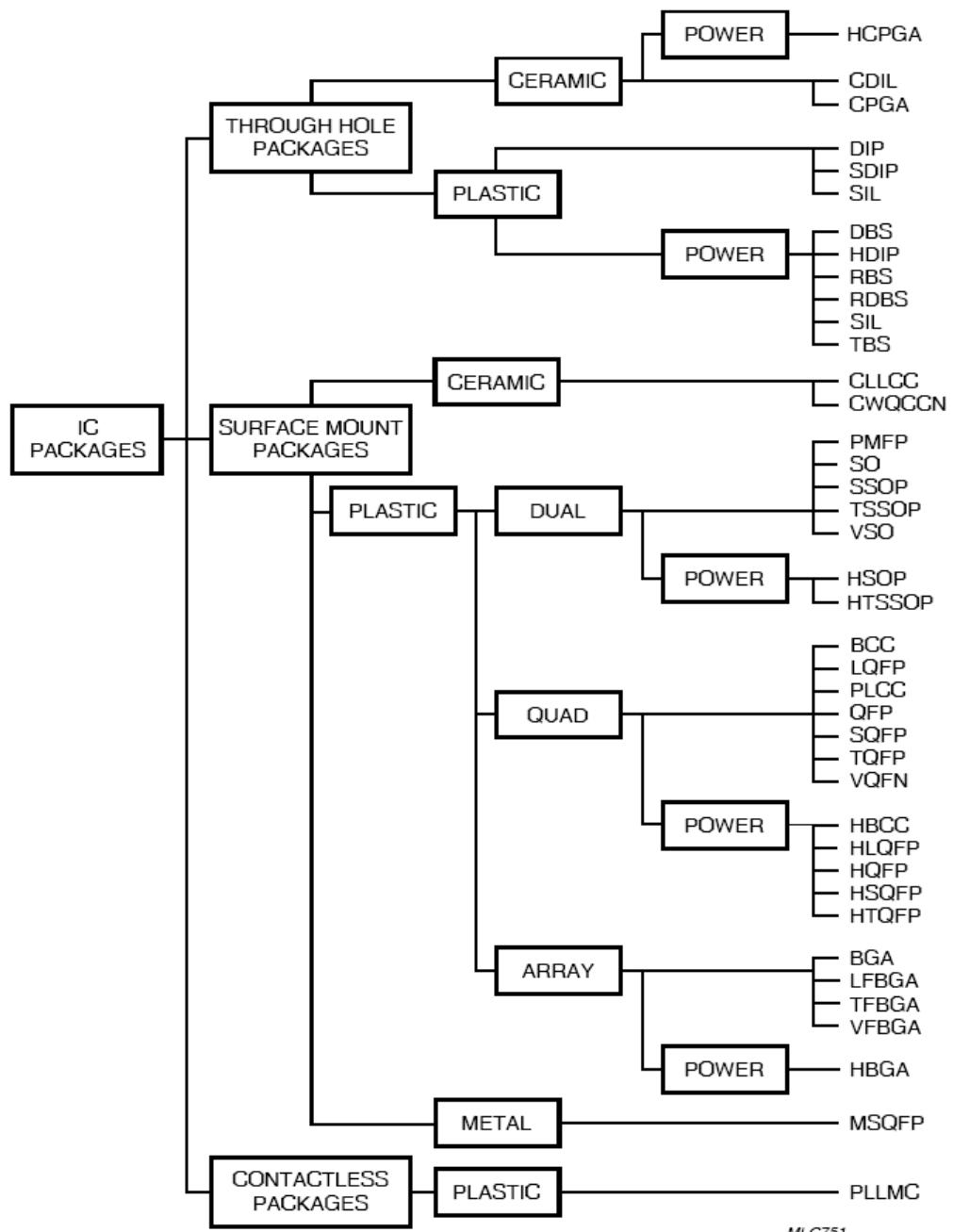


Πάνω στην πλακέτα χρειάζονται να οριστούν οι θέσεις στις οποίες θα τοποθετηθούν τα ολοκληρωμένα και τα διάκριτα (αντιστάσεις, πυκνωτές, διακόπτες κλπ) στοιχεία. Ομως κάθε στοιχείο είναι πιθανό να παράγεται σε ένα από πολλά διαφορετικά περιβλήματα (package). Η ανάγκη για τον ορισμός της θέσης ενός στοιχείου άμεσα οδηγεί και στην ανάγκη καθορισμού από το σχεδιαστή του περιβλήματος που θα χρησιμοποιηθεί για το συγκεκριμένο στοιχείο καθώς τα διαφορετικά περιβλήματα έχουν και διαφορετικές ανάγκες για τη τοποθέτησή τους.

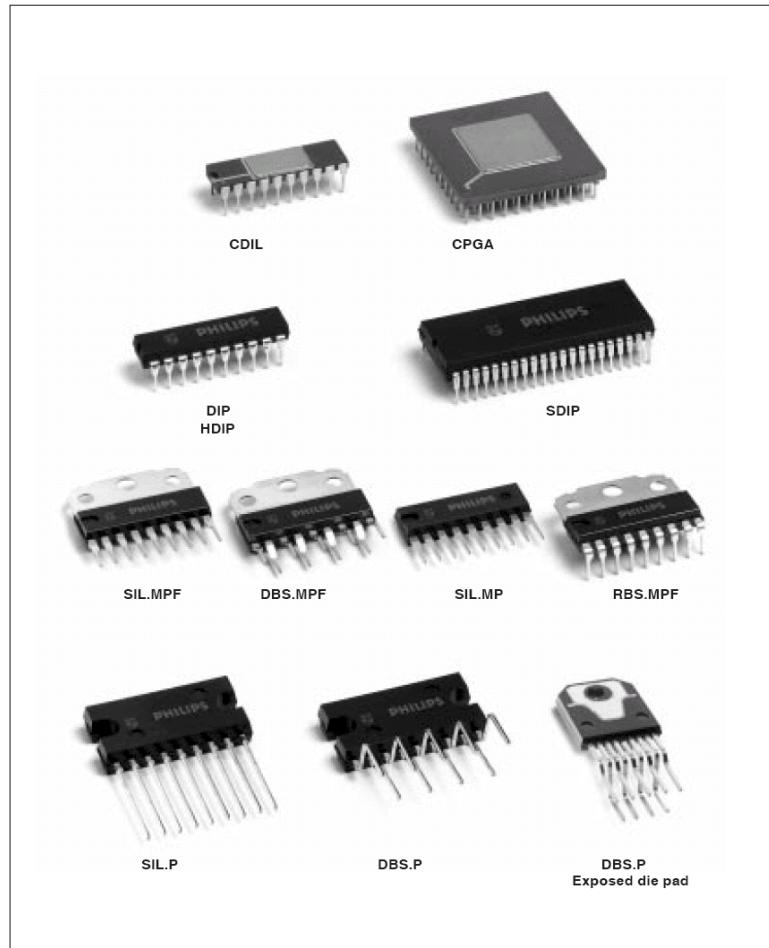
Ας εξετάσουμε για λίγο τα διαφορετικά περιβλήματα που συνήθως είναι διαθέσιμα για το ίδιο ολοκληρωμένο. Η παρακάτω ταξινόμηση (όπως προτείνεται από τον κατασκευαστή ολοκληρωμένων Philips) κατατάσσει τα διαθέσιμα περιβλήματα ανάλογα με :

1. Από το αν για τη προσαρμογή του στοιχείου πάνω στην πλακέτα απαιτείται η διάνοιξη οπών (through hole packages) ή όχι. Στη δεύτερη περίπτωση τα στοιχεία μας προσαρμόζονται πάνω στην επιφάνεια του χαλκού και γι' αυτό ονομάζονται (surface mount devices – SMDs). Είναι προφανές ότι όταν ο σχεδιαστής χρησιμοποιήσει through hole στοιχεία (στην επόμενη σελίδα φαίνεται φωτογραφία των διατιθέμενων περιβλημάτων αυτής της κατηγορίας από τον κατασκευαστή Philips) θα πρέπει να παραδώσει στον κατασκευαστή της πλακέτας αρχείο με τις θέσεις των οπών που θα πρέπει να διανοιχτούν πάνω στην πλακέτα καθώς και το διαμέτρημα αυτών. Τα στοιχεία αυτά υπάρχουν καταχωρημένα σε διάφορες βάσεις δεδομένων που οι κατασκευαστές στοιχείων παρέχουν στους σχεδιαστές. Σε περίπτωση πάντως που το περιβλήμα του στοιχείου που θα χρησιμοποιηθεί δεν υπάρχει στη χρησιμοποιούμενη βάση δεδομένων, τότε ο σχεδιαστής θα πρέπει μόνος του να περιγράψει το αποτύπωμα (footprint) του στοιχείου, χρησιμοποιώντας κατάλληλους editors. Προφανώς όσο περισσότερα

διαφορετικά διαμετρήματα χρησιμοποιήσει ο σχεδιαστής τόσα διαφορετικά τρυπάνια θα πρέπει να χρησιμοποιήσει ο κατασκευαστής με αποτέλεσμα να αυξάνεται η δυσκολία και ο χρόνος κατασκευής και συνεπώς και το κόστος της πλακέτας. Κάθε τέτοια οπή υπόκειται στη διαδικασία της επιμετάλλωσης (metallization), έτσι ώστε να προσφέρει ηλεκτρική αγωγή με όλα τα στρώματα χαλκού τα οποία διαπερνάει και φυσικά με τον ακροδέκτη του στοιχείου που θα τοποθετηθεί σ' αυτήν.



TROUGH-HOLE MOUNT PACKAGES

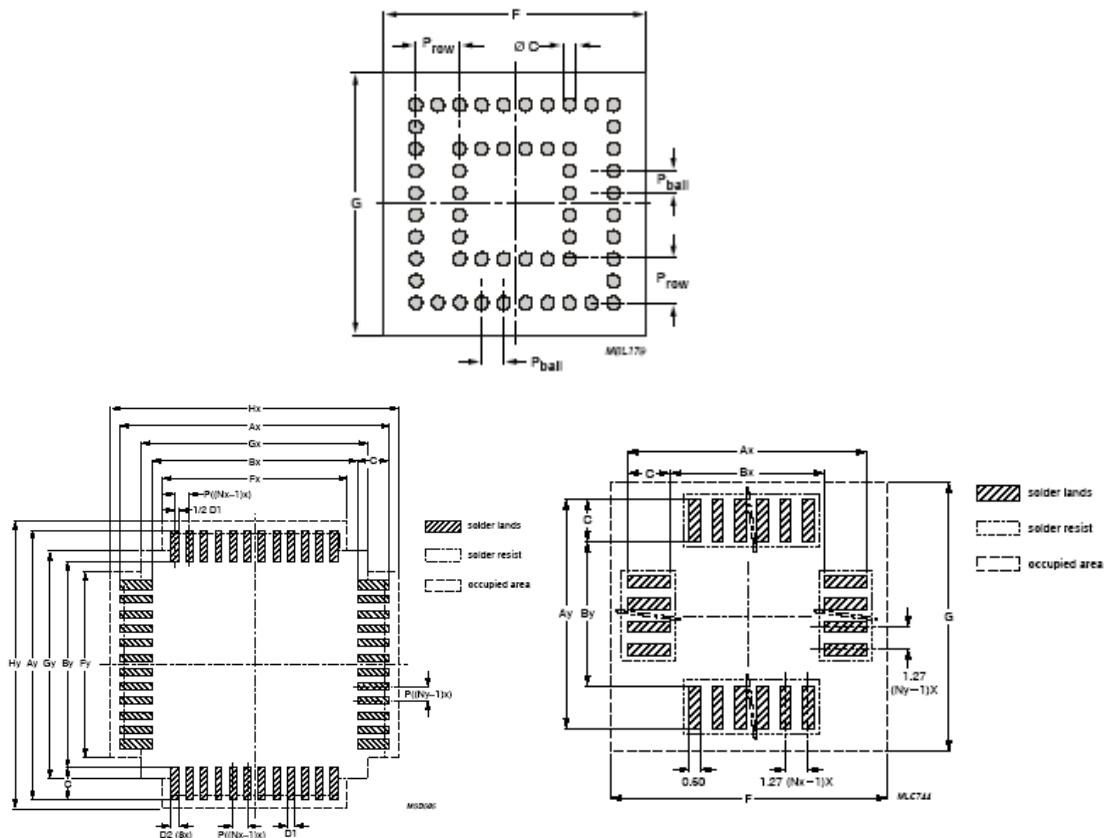


Trough-hole mount packages

PACKAGE NAME	NUMBER OF LEADS																				
	3	7	8	9	13	14	16	17	18	20	22	23	24	27	28	32	40	42	48	52	56
DBS.MP			X																		
DBS.MPF			X																		
DBS.P	X		X	X			X					X		X							
DIP		X			X	X			X	X	X		X		X	X	X	X			X
HDIP								X													
RBS.MPF		X																			
RDBS.P				X		X								X		X		X	X	X	
SDIP																					
SIL.MP		X																			
SIL.MPF		X																			
SIL.P		X	X																		
TBS.P				X																	
TO-92	X																				

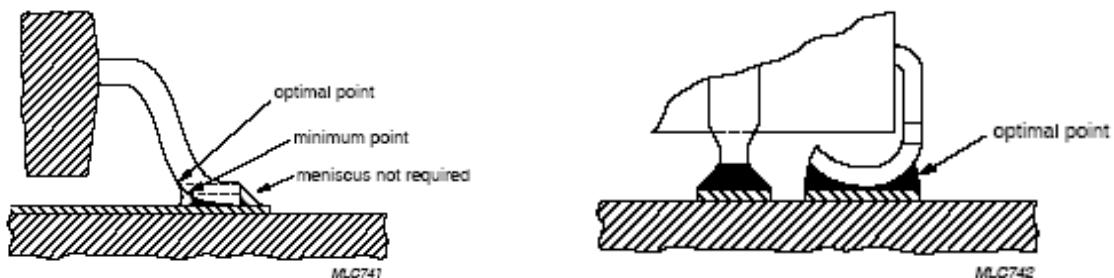
Στον παραπάνω πίνακα φαίνονται οι πιθανοί συνδυασμοί αριθμού ακροδεκτών και διαφορετικών through hole περιβλημάτων που προσφέρει ο κατασκευαστής Philips. Οπως βλέπουμε δεν προσφέρεται κάθε συνδυασμός περιβλήματος – ακροδεκτών αλλά το πιο σημαντικό είναι ότι ο αριθμός των ακροδεκτών είναι περιορισμένος. Στην περίπτωση που ο σχεδιασμός μας χρειάζεται περισσότερους ακροδέκτες, τότε θα πρέπει να καταφύγουμε στη λύση των SMD. Στη περίπτωση των SMD στοιχείων, η τοποθέτησή τους πάνω στη πλακέτα γίνεται με τη κόλλησή τους πάνω σε κατάλληλα ίχνη (tracks). Το σύνολο των tracks ενός στοιχείου αποτελεί το αποτύπωμα (footprint) του στοιχείου. Ο σχεδιαστής με

την επιλογή ενός SMD είναι υποχρεωμένος να καθιορίσει και το κατάλληλο αποτύπωμα. Προς βοήθειά του έχει και πάλι βάσεις δεδομένων που περιέχουν αποτυπώματα για διάφορα περιβλήματα SMD, ενώ στην περίπτωση που το αποτύπωμα δεν υπάρχει θα πρέπει να σχεδιαστεί. Ο κατασκευαστής του ολοκληρωμένου σε αυτή τη περίπτωση θα πρέπει να παρέχει σαφέστατα διαγράμματα με το μέγεθος των επαφών των ακροδεκτών, των αποστάσεων μεταξύ τους, την κατανομή τους κλπ. Παραδείγματα τέτοιων διαγραμμάτων παρουσιάζονται στις παρακάτω εικόνες.



1. Από την ικανότητα απαγωγής της θερμότητας που παρέχει το κάθε είδος περιβλήματος. Εδώ διακρίνουμε τις περιπτώσεις κυρίως κεραμικών και πλαστικών περιβλημάτων με τα πρώτα να προσφέρουν σημαντικά καλύτερη δυνατότητα απαγωγής της θερμότητας με ταυτόχρονο μεγαλύτερο κόστος. Τελευταία σε εφαρμογές που η απαγωγή της θερμότητας παίζει μεγάλο ρόλο έχουν εισαχθεί και στοιχεία με μεταλλικό περιβλήμα, που συνήθως εύκολα προσκολλώνται σε άλλα ψυκτικά μέσα.
2. Τα SMD διαιρίνονται περαιτέρω ανάλογα με τη κατανομή και το σχήμα των ακροδεκτών τους. Η κατανομή μπορεί να είναι στις δύο μόνο πλευρές του περιβλήματος, στις τέσσερις πλευρές του περιβλήματος ή σε μορφή πίνακα σε όλο

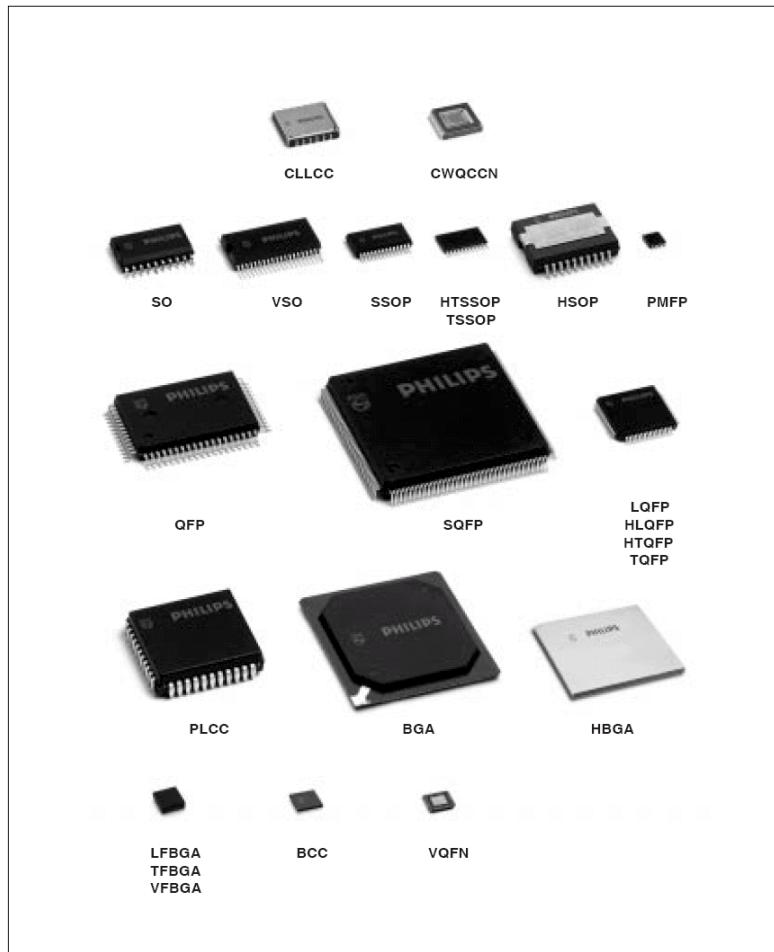
το κάτω μέρος του περιβλήματος. Η μορφή των ακροδεκτών στην περίπτωση που έχουμε ακροδέκτες στις δύο ή τις τέσσερις πλευρές του στοιχείου, μπορεί να είναι μία από τις δύο που παρουσιάζονται στην παρακάτω εικόνα. Οι ακροδέκτες στην αριστερή εικόνα έχουν τη μορφή του αγγλικού γράμματος J και γι' αυτό είναι γνωστοί ως και J-lead. Στην δεξιά εικόνα, ο ακροδέκτης διπλώνει προς τη μέση πλευρά του στοιχείου εξασφαλίζοντας έτσι μια μικρή απόσταση μεταξύ πλακέτας και στοιχείου.



Ανάλογα με τον αριθμό των ακροδεκτών και τη κατανομή τους, είναι πιθανό η πυκνότητά τους να μην επιτρέπει τη συγκόλληση αυτών των στοιχείων με χειρωνακτικό τρόπο. Στην περίπτωση αυτή τα στοιχεία συγκολλούνται από τον ίδιο το κατασκευαστή της πλακέτας (στην περίπτωση που το περιβλήμα έχει ακροδέκτες σε μορφή πίνακα στο κάτω μέρος του η συγκόλληση από τον κατασκευαστή είναι η μόνη δυνατότητα) ο οποίος έχει και την ευθύνη ελέγχου της ορθής συγκόλλησης με το αντίστοιχο φυσικά αυξημένο κόστος.

3. Το συνολικό πάχος του στοιχείου. Υπάρχουν εφαρμογές στις οποίες το πάχος είναι καθοριστικός παράγοντας. Θα μπορούσαμε για παράδειγμα να αναφέρουμε τις κάρτες PCMCIA που συνήθως συναντάμε στους φορητούς υπολογιστές. Υπάρχουν για τους λόγους αυτούς μικρά, πολύ μικρά και μικροσκοπικά περιβλήματα (SOP, SSOP και TSSOP). Το πάχος ενός TSSOP περιβλήματος είναι τόσο μικρό που άνετα χωράει κάτω από ένα J-Lead ή PLCC στοιχείο. Στην παρακάτω εικόνα φαίνονται τα περιβλήματα που παρέχονται από ένα συγκεκριμένο κατασκευαστή.

SURFACE MOUNT PACKAGES



Και σ' αυτή τη περίπτωση δεν έχουμε όλους τους πιθανούς συνδυασμούς περιβλήματος – ακροδεκτών. Για το συγκεκριμένο κατασκευαστή οι πιθανοί συνδυασμοί είναι :

PACKAGE TYPE OVERVIEW WITH LEAD COUNT

Surface mount packages (part 1 of 2)

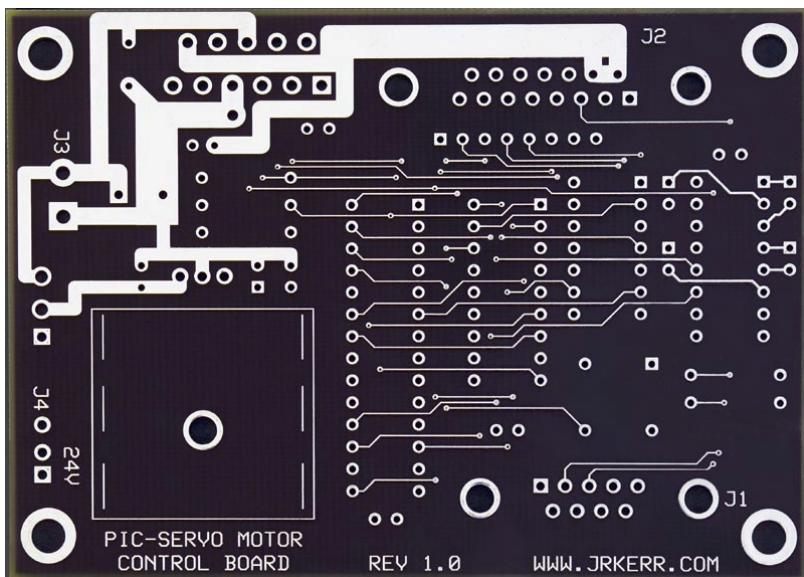
PACKAGE NAME	NUMBER OF LEADS																																					
	4	8	9	10	13	14	16	20	24	28	30	32	36	40	44	48	52	56	64	68	80	81	84	96	100	114	120	128	132	144	156	160						
BGA																												X										
HBCC							X	X		X						X																						
HBGA																																						
HLQFP																													X									
HQFP																																X						
HSOP									X	X																												
HSQFP																																						
HTQFP											X				X		X	X						X														
HTSSOP									X		X																											
LFBGA																		X	X			X	X	X	X													
LQFP												X			X	X			X	X																		
MSQFP																																						
PLCC													X																									
PMFP	X																																					
QFP																																						
SMS.P		X	X								X	X	X	X	X	X																						
SO	X	X									X	X	X	X	X	X																						
SOJ																				X																		
SQFP																																						
SSOP											X	X	X	X	X				X	X																		
TFBGA																					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
TQFP																				X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
TSSOP	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
VSO																				X			X															
VFBGA																																						
VQFN																				X	X																	

Surface mount packages (part 2 of 2)

PACKAGE NAME	NUMBER OF LEADS																									
	176	180	208	240	256	272	280	292	304	316	324	329	352	388	420	456	492	504	516	553	596	600	656	776	1140	1160
BGA			X	X	X		X	X	X		X	X	X		X	X	X	X	X	X	X					
HBCC																										
HBGA				X			X	X			X		X		X					X						
HLOFP																										
HQFP																										
HSOP																										
HSQFP			X	X																						
HTQFP																										
HTSSOP																										
LFBGA															X											
LQFP	X		X																							
MSQFP			X																							
PLCC																										
PMFP																										
QFP																										
SMS.P																										
SO																										
SOJ																										
SQFP		X	X												X											
SSOP																										
TFBGA	X	X													X											
TQFP																										
TSSOP																										
VSO																										
VFBGA																										
VQFN																										

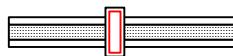
Οπως φαίνεται και από τους παραπάνω πίνακες, άλλα περιβλήματα είναι κατάλληλα για λίγους ακροδέκτες, ενώ στην περίπτωση των πάρα πολλών ακροδεκτών αυτοί μπορούν μόνο να βρίσκονται με τη μορφή πίνακα στο κάτω μέρος του ολοκληρωμένου.

Πέρα από τη θέση και τη προσαρμογή των ολοκληρωμένων και των διακριτών στοιχείων του κυκλώματος, ο σχεδιαστής θα πρέπει να περιγράψει στον κατασκευαστή της πλακέτας και το τρόπο διασύνδεσής τους. Αυτό σημαίνει ότι θα πρέπει να παραδώσει ένα αρχείο γραμμών διασύνδεσης, που για ικανόθε μία θα περιγράψει την αρχή, το τέλος της και το πάχος της. Εναλλακτικά μπορεί να δώσει ένα τοπογραφικό σχέδιο της επιθυμητής διασύνδεσης. Ενα τέτοιο σχέδιο φαίνεται στη παρακάτω εικόνα.



Προσέξτε ότι στη παραπάνω εικόνα χρησιμοποιούνται γραμμές διασύνδεσης με τρία διαφορετικά πάχη. Ο αριθμός των διαφορετικών παχών είναι κι αυτός ένας παράγοντας του κόστους της πλακέτας.

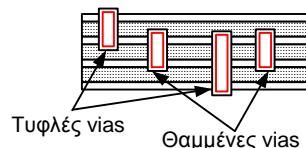
Τα στοιχεία συνήθως τοποθετούνται στη μία πλευρά της πλακέτας (πολλές φορές ονομαζόμενη top ή component side), αν και υπάρχουν πλακέτες με στοιχεία και στην αντίθετη πλευρά. Η διασύνδεση των στοιχείων ωστόσο, δεν μπορεί πάντοτε να ολοκληρωθεί στη μία πλευρά (στην περίπτωση πλακετών με δύο επίπεδα το δεύτερο επίπεδο αναφέρεται και σα routing side). Συχνά λοιπόν μια γραμμή διασύνδεσης που ξεκινά σε ένα επίπεδο χαλκού είναι απαραίτητο να συνεχιστεί σε άποιο άλλο. Αυτό γίνεται εφικτό με τη χρησιμοποίηση των vias. Μια via είναι μια επιμεταλλωμένη οπή πολύ μικρότερου διαμετρήματος από ότι μια οπή για through hole devices που σκοπό έχει τη διασύνδεση δύο κομματιών μιας γραμμής διασύνδεσης που βρίσκονται σε διαφορετικά αγώγιμα επίπεδα. Η τομή μιας via σε μια πλακέτα δύο επιπέδων φαίνεται στο παρακάτω σχήμα :



Η via δημιουργεί ένα αγώγιμο μονοπάτι μεταξύ των δύο αγώγιμων επιπέδων διαπερνώντας το μονωτή. Στις περιπτώσεις πλακετών με περισσότερα επίπεδα υπάρχει περίπτωση το ένα ή και τα δύο αγώγιμα επίπεδα που ενώνονται με τη via να μην είναι ορατά (να είναι δηλαδή ίντας από τα εσωτερικά επίπεδα της πλακέτας. Στις περιπτώσεις αυτές έχουμε :

- ♦ Τις τυφλές vias (blind vias) : είναι vias που ενώνουν ένα ορατό επίπεδο με ένα μη ορατό.
- ♦ Τις θαμμένες vias (buried vias) : είναι vias που ενώνουν δύο μη ορατά επίπεδα.

Στο παρακάτω σχήμα δίνονται παραδείγματα από τυφλές και θαμμένες vias.



Στις πλακέτες που υπάρχουν πολλά αγώγιμα επίπεδα, είναι πολλές φορές επικερδές να αφιερώσουμε ένα ολόκληρο τέτοιο επίπεδο στην τροφοδοσία και άλλο ένα αποκλειστικά στη γείωση. Είναι γνωστό ότι για να μπορεί ο σχεδιασμός μας να ανταποκριθεί στην υψηλή πυκνότητα ρεύματος που απαιτείται για υψηλές ταχύτητες λειτουργίας, οι γραμμές διασύνδεσης πρέπει να χαραχτούν με το μέγιστο δυνατό πάχος, έτσι ώστε αφενός να παρουσιάζουν τη μικρότερη αντίσταση

---

και αφετέρου να έχουν αρκετούς φορείς. Οι παχιές γραμμές ωστόσο αυξάνουν το εμβαδόν του σχεδιασμού μας. Με την αφέρωση ολόκληρου επιπέδου για τροφοδοσία και γείωση, τα SMD μπορούν να συνδεθούν με αυτά μέσω τυφλών vias χωρίς να χρειάζεται η χάραξη καμίας γραμμής διασύνδεσης. Αυτό προφανώς οδηγεί σε πλακέτες μικρότερου εμβαδού. Επίσης θα πρέπει να διευκρινιστεί ότι γύρω από τους ακροδέκτες των through hole devices σε αυτή τη περίπτωση χρειάζεται κόψιμο της αγωγιμότητας μέσω isolation rings.

Θα κλείσουμε το κομμάτι που αφορά στη κατασκευή πλακετών με μια σύντομη αναφορά του πως ένα τοπογραφικό σχέδιο της διασύνδεσης που δίνει ο σχεδιαστής τελικά απεικονίζεται πάνω στη πλακέτα. Υπάρχουν πάρα πολλοί τρόποι γι' αυτό. Οι πλέον συνήθεις είναι :

- ♦ Να χρησιμοποιήσουμε αυτό το σχέδιο σα μάσκα και να υποβάλλουμε το στρώμα χαλκού σε ίαποια φωτοχημική διαδικασία. Στο τέλος αυτής της διαδικασίας τα κομμάτια του χαλκού που δεν προστατεύηκαν από τη μάσκα έχουν αποκτήσει διαφορετικές χημικές ιδιότητες από τα υπόλοιπα. Χρησιμοποιώντας αυτές τις αποκτηθείσες ιδιότητες μπορούμε να τα απομακρύνουμε μέσω περαιτέρω χημικών διαδικασιών και έτσι να απομείνει μόνο ο χαλκός που η μάσκα μας άριζε.

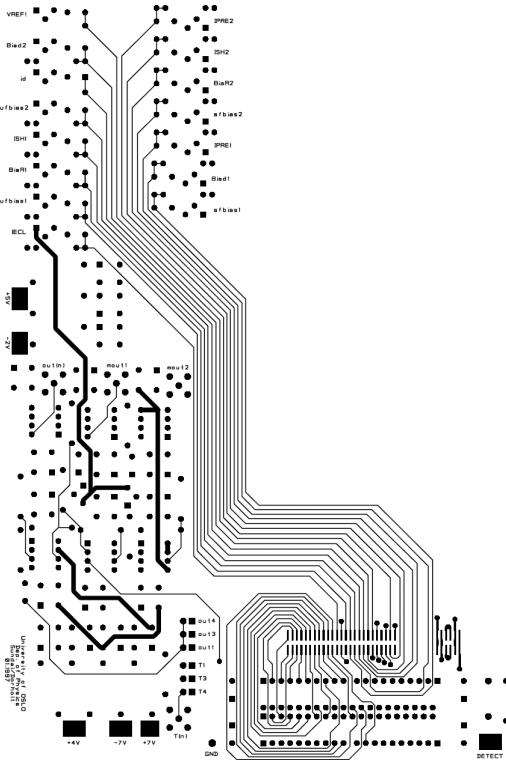
- ♦ Να χρησιμοποιήσουμε αυτό το σχέδιο για να καθοδηγήσουμε ένα βραχίονα ρομπότ που χαράζει το περίγραμμα γύρω από τις επιθυμητές γραμμές διασύνδεσης, απομονώνοντάς τις από τον υπόλοιπο χαλκό. Μετά με ηλεκτρόλυση μπορούμε να απομακρύνουμε όλο τον υπόλοιπο χαλκό που δε χρειάζεται.

Αφού η επιθυμητή διασύνδεση μεταφερθεί σε κάθε επίπεδο χαλκού, τα διαφορετικά φύλλα συγκολλώνται με τη προσθήκη μονωτικού ενδιάμεσα και μετά ακολουθεί η διαδικασία για τη διάνοιξη των οπών και των vias. Στο τέλος της κατασκευαστικής γραμμής, τα ορατά αγώγιμα επίπεδα καλύπτονται από ένα στρώμα υλικού ανθεκτικού στο φως (photoresist, είναι το υλικό που δίνει το χαρακτηριστικό πράσινο χρώμα στις περισσότερες πλακέτες) με σκοπό την αποφυγή της οξείδωσης του χαλκού με το πέρασμα του χρόνου.

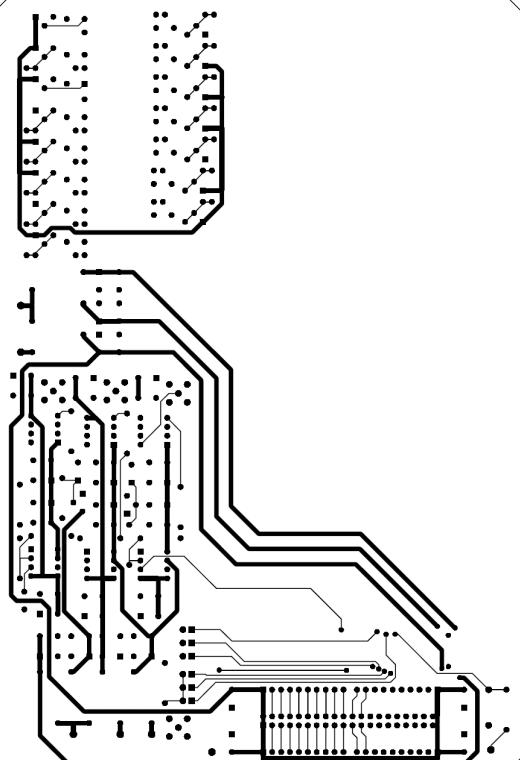
Συμπερασματικά μπορούμε να πούμε ότι για τη κατασκευή μιας πλακέτας ο σχεδιαστής πρέπει να παραδώσει τη πλήρη φυσική περιγραφή του σχεδιασμού του εκφρασμένη σε διάφορα τοπογραφικά αρχεία. Τα αρχεία αυτά περιλαμβάνουν :

- ♦ Αρχεία διασύνδεσης για κάθε αγώγιμο επίπεδο που χρησιμοποιείται για διασύνδεση (routing layer).
- ♦ Τα αποτυπώματα των SMD που χρησιμοποιεί για το component layer.
- ♦ Τις οπές που θα πρέπει να ανοιχτούν για τα through hole στοιχεία.
- ♦ Τις vias και τον τύπο τους.

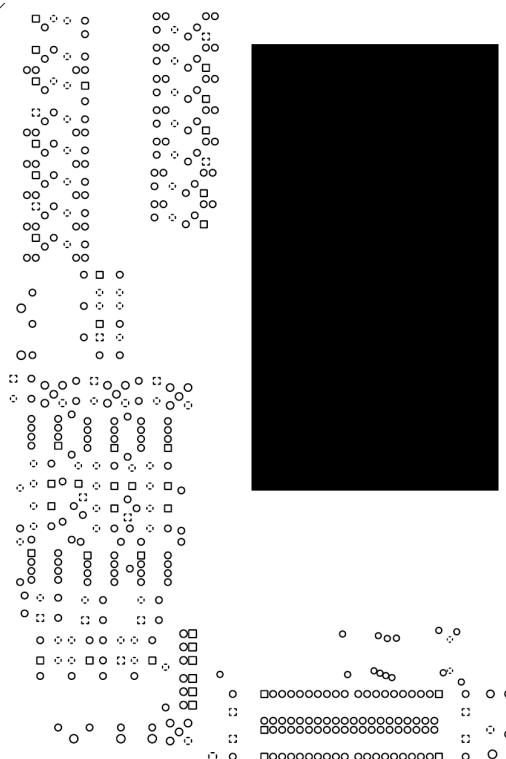
Στα παρακάτω σχήματα δίνουμε παραδείγματα τέτοιων αρχείων από ένα πραγματικό σχεδιασμό. Η πλακέτα είναι τεσσάρων επιπέδων με τα μη ορατά επίπεδα να χρησιμοποιούνται για τροφοδοσία και γείωση.



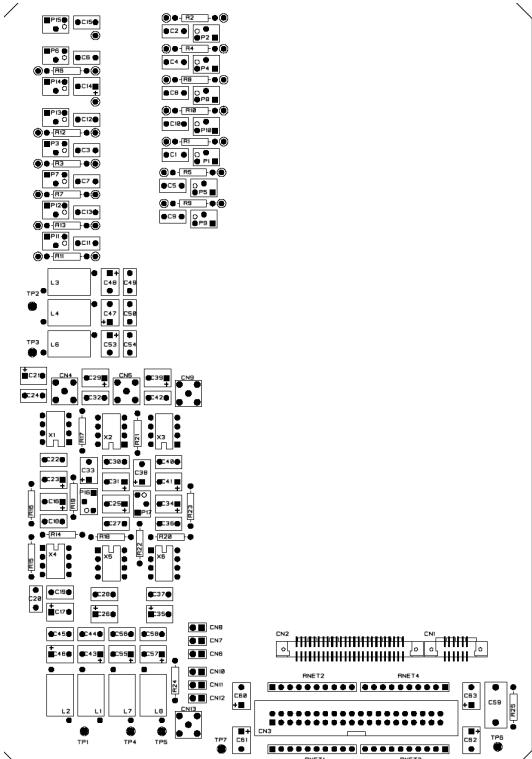
Η πάνω πλευρά (component side)



Η κάτω πλευρά (routing side)



Το επίπεδο γείωσης  
(αντεστραμμένη πολικότητα εικόνας)



Το επίπεδο μεταξοτυπίας

---

Τα πρώτα χρόνια κατασκευής πλακετών, αυτές περιείχαν μόνο πολύ απλά (small scale of integration – SSI) ολοκληρωμένα και τα απαραίτητα διάκριτα στοιχεία. Με την ολοένα αυξανόμενη πολυπλοκότητα των συστημάτων θα οδηγούμαστε ακολουθώντας αυτό το δρόμο σε πλακέτες πολύ μεγάλου εμβαδού. Ενα άλλο στοιχείο που θα έκανε τέτοιες πλακέτες μη εμπορικές θα ήταν και η ευκολία αντιγραφής τους. Σκοπός λοιπόν των κατασκευαστών ήταν να μπορέσουν να αναπτύξουν ολοκληρωμένα με τα οποία θα μπορούσαν σε πρώτη φάση να αντικαταστήσουν δύο ή περισσότερα SSI ολοκληρωμένα και τα οποία θα προσέφεραν και κάποιο είδος προστασίας από την αντιγραφή. Για να επιτύχουν αυτό το πρωταρχικό στόχο τους, οι κατασκευαστές ολοκληρωμένων βασίστηκαν στις προγραμματιζόμενες συσκευές (Programmable Logic Devices – PLDs).

Παρότι στο εμπόριο υπάρχουν διαθέσιμα πάνω από μερικά εκατομμύρια διαφορετικά ολοκληρωμένα προγραμματιζόμενης λογικής, που το καθένα μπορεί να υλοποιήσει μια πλειάδα συναρτήσεων και είναι φτιαγμένο με ξεχωριστό τρόπο από όλα τα υπόλοιπα, θα μπορούσαμε να πούμε ότι όλα αυτά τα ολοκληρωμένα βασίζονται στην ίδια ιδέα : στην υλοποίηση συνδυαστικών συναρτήσεων με τη χρήση μνήμης. Αν και αυτό είναι ένα θέμα που έχετε διδαχθεί αναλυτικά στο μάθημα της Ψηφιακής Σχεδίασης, η σύντομη αναφορά που γίνεται παρακάτω θα μας βοηθήσει στην καλύτερη εισαγωγή των PLDs.

Εστω λοιπόν ένα σύνολο n λογικών συναρτήσεων  $f_1(x_1, x_2, x_3, \dots, x_m)$ ,  $f_2(x_1, x_2, x_3, \dots, x_m)$ , ...,  $f_n(x_1, x_2, x_3, \dots, x_m)$ , κάθε μία με μια μόνο έξοδο και όπου m ο αριθμός των διαφορετικών λογικών μεταβλητών στο σύνολο αυτών των συναρτήσεων. Προφανώς αν μία από αυτές τις συναρτήσεις είναι συνάρτηση λιγότερων από m μεταβλητών, αυτή μπορεί να μετατραπεί σε συνάρτηση m μεταβλητών, όπου οι επιπλέον μεταβλητές καθίστανται αδιάφοροι όροι γι' αυτή τη συνάρτηση. Για παράδειγμα η  $f(a,b,c) = ab + c$  μπορεί να μετατραπεί στην  $f'(a,b,c,d) = (ab+c)d = (ab+c)(d + !d) = abd + cd + ab!d + c!d$  (χρησιμοποιούμε το ! για να υποδηλώσουμε το συμπλήρωμα μιας μεταβλητής), η οποία χρησιμοποιεί μία επιπλέον μεταβλητή (d) η οποία όμως είναι αδιάφορος όρος για την αλήθεια της  $f'$ , ή με άλλα λόγια η f και η  $f'$  είναι διαφορετικές εκφράσεις της ίδιας λογικής συνάρτησης.

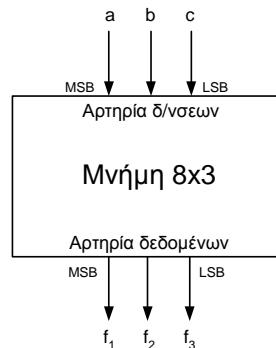
Μπορούμε να αντικαταστήσουμε τα λογικά κυκλώματα που χρειάζονται για την υλοποίηση των  $f_1$ ,  $f_2$ , ...,  $f_n$ , με μία μνήμη  $2^m$  διευθυνσιοδοτούμενων θέσεων που η κάθε θέση έχει μήκος n δυαδικών ψηφίων, αν σε κάθε θέση της μνήμης γράψουμε τα απαραίτητα δεδομένα. Εστω λοιπόν η ψηφιολέξη  $b_0b_1b_2\dots b_{m-1}$ . Αν  $f_1(b_0, b_1, b_2, \dots, b_{m-1})$ ,  $f_2(b_0, b_1, b_2, \dots, b_{m-1})$ , ...,  $f_n(b_0, b_1, b_2, \dots, b_{m-1}) = y_1, y_2, y_3, \dots, y_n$ , θα πρέπει στη θέση  $2^{b_0}b_1b_2\dots b_{m-1}$  της μνήμης να αποθηκεύσουμε τη ψηφιολέξη  $y_1, y_2, y_3, \dots, y_n$ .

Χρησιμοποιώντας ακολούθως τα  $x_1, x_2, x_3, \dots, x_m$  ως αρτηρία διευθύνσεων της μνήμης και τις εξόδους της μνήμης ως τις συναρτήσεις  $f_1, f_2, \dots, f_n$ , έχουμε πετύχει την απαραίτητη αντικατάσταση. Ας δούμε ένα παράδειγμα που συνοψίζει όλα τα προηγούμενα.

Εστω οι συναρτήσεις  $f_1(a) = a$ ,  $f_2(b, c) = bc$  και  $f_3(a, b) = a + b$ . Στη περίπτωση αυτή έχουμε  $m=3$  και χρειάζεται να μετασχηματίσουμε όλες τις συναρτήσεις μας. Παίρνουμε λοιπόν την  $f_1(a, b, c) = abc + ab!c + a!bc + a!b!c$ , την  $f_2(a, b, c) = abc + !abc$  και την  $f_3(a, b, c) = abc + ab!c + a!bc + a!b!c + !abc + !ab!c$ . Οι πίνακες αληθείας των συναρτήσεων αυτών φαίνονται στο επόμενο σχήμα.

a	b	c	$f'_1$	$f'_2$	$f'_3$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	1

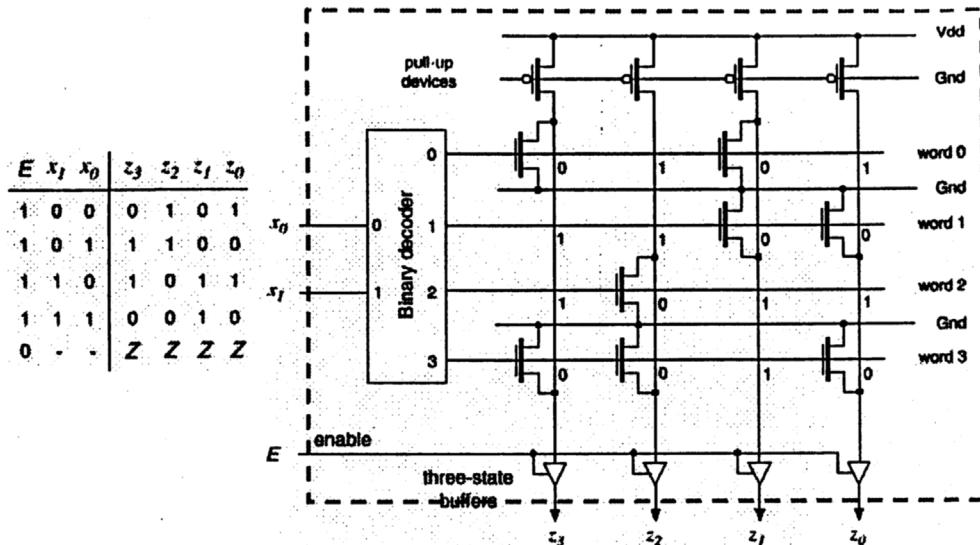
Σύμφωνα λοιπόν με τα παραπάνω μπορούμε να αντικαταστήσουμε αυτές τις συναρτήσεις με μία μνήμη οκτώ θέσεων με τρία δυαδικά ψηφία ανά θέση. Στη μνήμη αυτή θα πρέπει να αποθηκεύσουμε το δεξί μισό του παραπάνω σχήματος ενώ το ιώκλωμα που χρειάζεται για την αντικατάσταση των παραπάνω συναρτήσεων είναι το ακόλουθο :



Μια ιδιαίτερη περίπτωση της αντικατάστασης συναρτήσεων με μνήμη είναι αυτή που υπάρχει μόνο μία συνάρτηση προς αντικατάσταση. Τότε ο μονοδιάστατος πίνακας που θα πρέπει να αποθηκευτεί στη μνήμη μεγέθους  $2^m \times 1$  είναι γνωστός και σαν πίνακας – σύμβουλος (look-up table – LUT).

Στη συνέχεια θα εξετάσουμε πως εξελίχθηκαν ιστορικά τα ολοκληρωμένα προγραμματιζόμενης λογικής. Σαν πρώτα τέτοια ολοκληρωμένα θα πρέπει να θεωρήσουμε τις μνήμες ανάγνωσης μόνο (Read Only Memories – ROM). Ενα

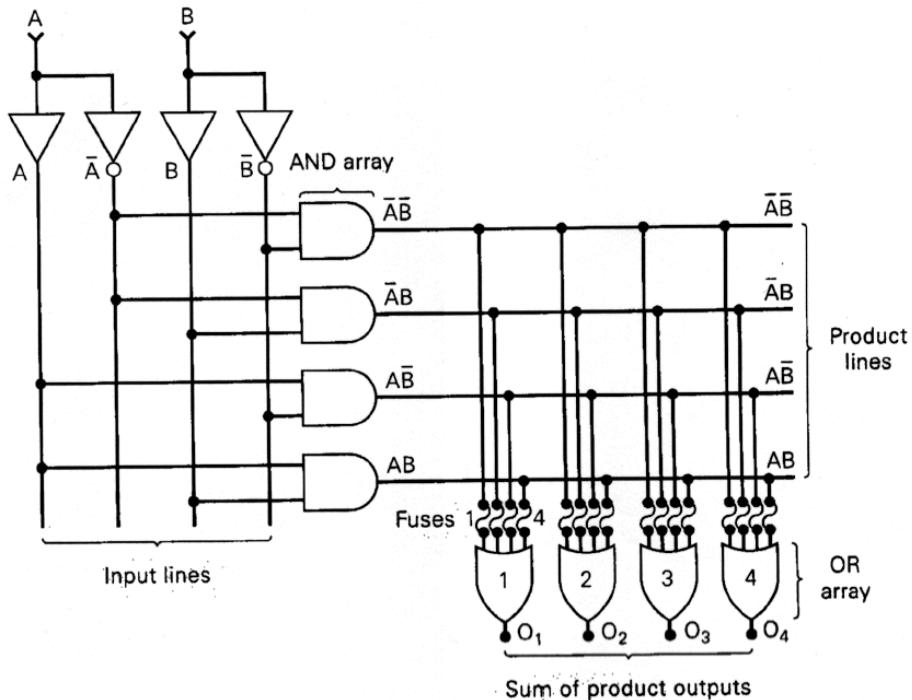
παράδειγμα της αρχιτεκτονικής που χρησιμοποιούν αυτά τα ολοκληρωμένα φαίνεται στο παρακάτω σχήμα :



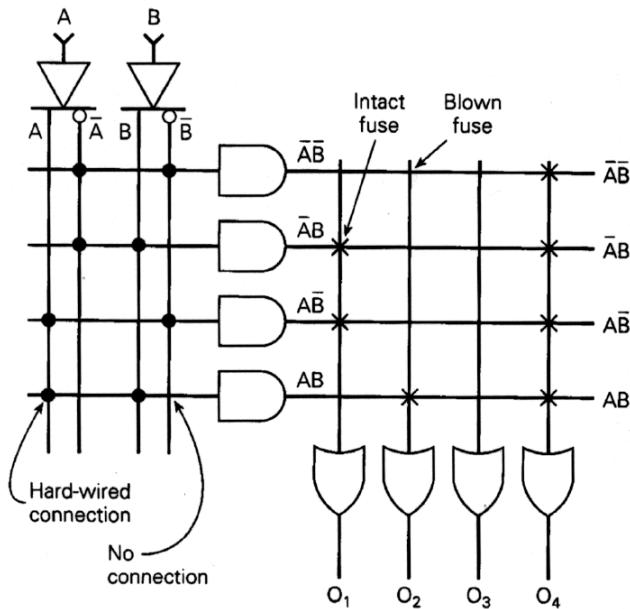
Στη ROM του σχήματος υλοποιούνται 4 συναρτήσεις των δύο μεταβλητών. Ο πίνακας αληθείας αυτών των συναρτήσεων φαίνεται στα αριστερά. Επίσης υπάρχει μια είσοδος για κατάσταση υψηλής εμπέδησης των εξόδων. Οπως μπορούμε να παρατηρήσουμε η αρχιτεκτονική μιας ROM αποτελείται από έναν αποκωδικοποιητή που ενεργοποιεί μία από τις γραμμές της μνήμης, ένα μονοδιάστατο πίνακα από ενεργητικά στοιχεία ανύψωσης δυναμικού (PMOS transistor), ένα δισδιάστατο πίνακα από ενεργά στοιχεία πτώσης δυναμικού (NMOS transistor) και από τους απομονωτές τριών καταστάσεων στην έξοδο. Ο προγραμματισμός έγκειται στη διαγραφή τρανζίστορ NMOS όπου στον πίνακα αληθείας των συναρτήσεων υπάρχει 1. Ας δούμε πως λειτουργεί η προγραμματισμένη ROM του σχήματος για να το καταλάβουμε καλύτερα. Εστω ότι η είσοδος μας είναι  $x_1x_0=01$ . Ο αποκωδικοποιητής θα ενεργοποιήσει τη γραμμή 1 στην έξοδό του ενώ όλες οι άλλες γραμμές θα είναι ανενεργές. Αποτέλεσμα αυτού είναι ότι μόνο όσα τρανζίστορ υπάρχουν στη γραμμή 1 έρχονται σε αγωγή. Τα τρανζίστορ PMOS που είναι μόνιμα σε αγωγή έχουν προφορτίσει τις εξόδους στο 1. Τα NMOS που έρχονται σε αγωγή και έχουν μεγαλύτερο κέρδος θα οδηγήσουν τις αντίστοιχες γραμμές στο 0. Άρα όπου στη γραμμή 1 υπάρχει NMOS θα οδηγήσει τις εξόδους στο 0 ενώ οι υπόλοιπες θα παραμείνουν στο 1. Βλέπουμε λοιπόν γιατί για κάθε 1 του πίνακα αληθείας θα πρέπει να διαγράψουμε το αντίστοιχο NMOS τρανζίστορ.

Η χρήση των ROM είναι ωστόσο ιδιαίτερα προβληματική. Το μέγεθος της απαιτούμενης ROM μεγαλώνει εκθετικά με τον αριθμό των μεταβλητών, υποστηρίζονται μόνο συνδυαστικά κυκλώματα και το χειρότερο από όλα ο προγραμματισμός του ολοκληρωμένου γίνεται μόνο από το εργοστάσιο κατασκευής

στο οποίο ο σχεδιαστής πρέπει να στείλει τα αρχεία του. Τα υπόλοιπα PLDs που θα δούμε παρακάτω βασίζονται σε μία άλλη φιλοσοφία, δηλαδή την έκφραση των λογικών συναρτήσεων με τη μορφή αθροίσματος γινομένων (Sum Of Products – SOP). Το βασικό κύκλωμα που υλοποιεί αυτή την έκφραση φαίνεται στο παρακάτω σχήμα για τέσσερις συναρτήσεις των δύο μεταβλητών.

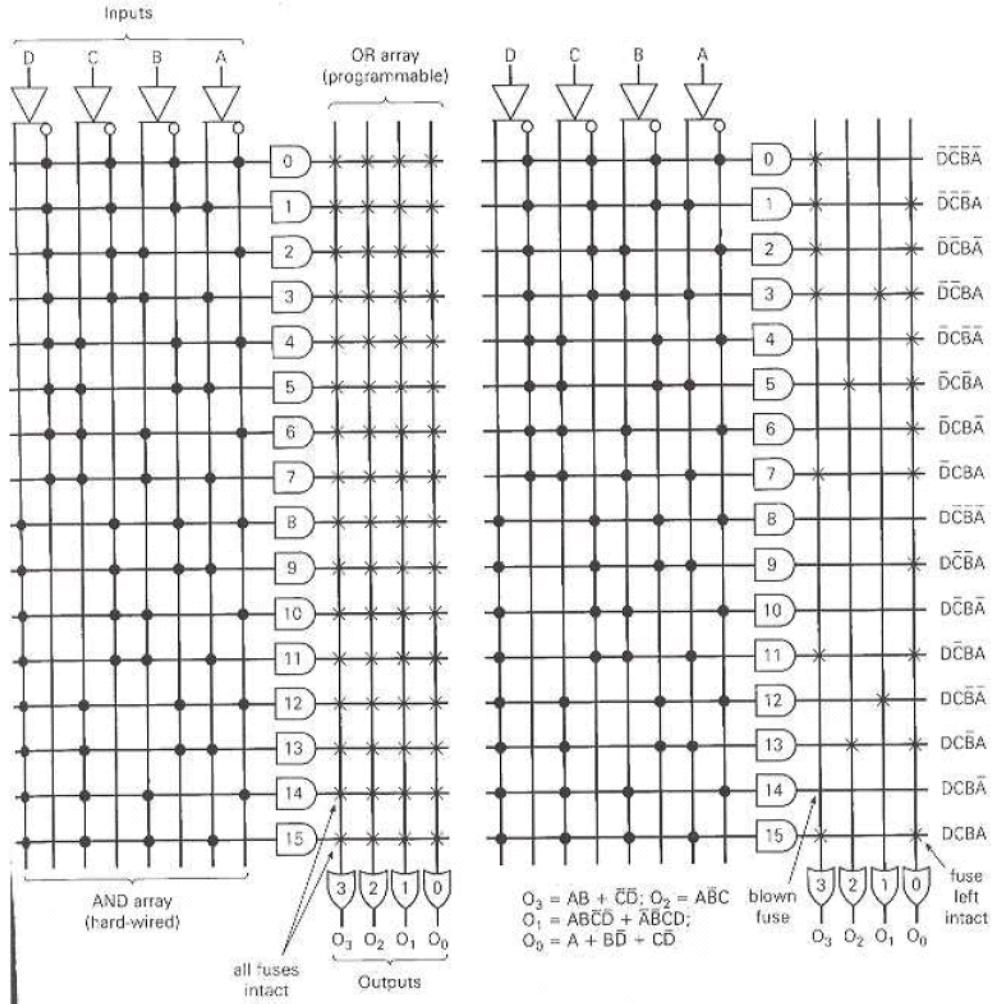


Το κύκλωμα αυτό αποτελείται από ένα πίνακα από AND πύλες οι οποίες φτιάχνουν όλα τα δυνατά γινόμενα (τέσσερα στην περίπτωσή μας) των μεταβλητών εισόδου. Κάποια (όλα στην περίπτωση του παραπάνω σχήματος) από αυτά τα γινόμενα οδηγούνται στο δεύτερο πίνακα της αρχιτεκτονικής που αποτελείται από πύλες OR οι οποίες κάνουν το άθροισμα των γινομένων. Προσέξτε ότι είναι πιθανόν σε ολοκληρωμένα με περισσότερες μεταβλητές στις πύλες OR να μην οδηγούνται όλα αλλά ικανοια και διαφορετικά από τα γινόμενα που σχηματίστηκαν από το πρώτο πίνακα. Στις εισόδους όμως των πυλών OR υπάρχουν μικροσκοπικές ασφάλειες που το κάψιμό τους απαγορεύει το συγκεκριμένο γινόμενο να πάρει μέρος στο λογικό OR. Η διαδικασία προγραμματισμού έγκειται συνεπώς στο κάψιμο ή μη αυτών των συγκεκριμένων ασφαλειών (fuses). Στο παρακάτω σχήμα φαίνεται η υλοποίηση τεσσάρων συναρτήσεων των δύο μεταβλητών χρησιμοποιώντας αυτή τη βασική αρχιτεκτονική. Με X σημειώνεται μια ασφάλεια που είναι άθικτη και συνεπώς το αντίστοιχο γινόμενο συμμετέχει στο άθροισμα ενώ με η απουσία του X δηλώνει μια καμένη ασφάλεια.



Προφανώς οι συναρτήσεις που υλοποιούνται στο παραπάνω σχήμα είναι οι  $O_1 = A \text{ XOR } B$ ,  $O_2 = A \text{ AND } B$ ,  $O_3 = 0$  και  $O_4 = 1$ . Αρχικά η διαδικασία αυτή μπορούσε να γίνει μόνο μία φορά. Τα ολοκληρωμένα αυτής της τεχνολογίας ονομάστηκαν PROMs (Programmable Read Only Memories). Το σχήμα που ακολουθεί δείχνει τη μορφή μιας PROM πριν και μετά τον προγραμματισμό της, που δεν είναι τίποτε άλλο παρά η επέκταση του βασικού σχήματος που εισάγαμε πιο πάνω.

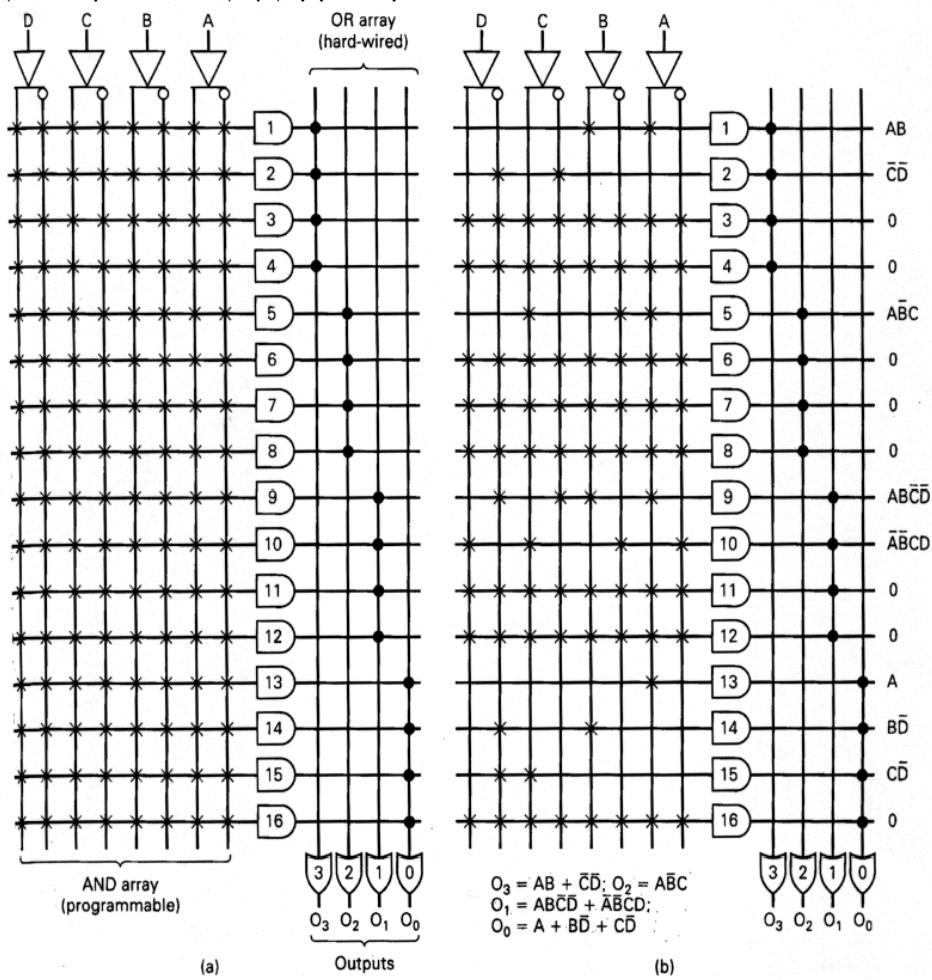
Η εξέλιξη αυτών των ολοκληρωμένων επέτρεψε σε πρώτη φάση το κάψιμο των ασφαλειών να μπορεί να γίνει και από το σχεδιαστή μέσω κατάλληλων τάσεων προγραμματισμού οδηγώντας μας στις FPGAs (Field Programmable Read Only Memories). Σε δεύτερη φάση, οι ασφάλειες αυτές φτιάχτηκαν από υλικά τα οποία με υπεριώδη ακτινοβολία μπορούσαν να αποκατασταθούν. Ετσι φτιάχτηκαν τα πρώτα ολοκληρωμένα που μπορούσαν να σβηστούν και να εγγραφούν ξανά (EPROMS – Erasable Programmable Read Only Memories). Τα ολοκληρωμένα αυτά ξεχωρίζουν εύκολα από τα υπόλοιπα μιας και στο πάνω μέρος τους υπάρχει ένα παράθυρο προστατευμένο με τζάμι από το οποίο περνάει η υπεριώδης ακτινοβολία. Αμεσοί απόγονοί τους είναι τα ολοκληρωμένα EEPROM –συχνά θα τα δείτε σαν E<sup>2</sup>PROM –(Electrically Erasable Programmable Read Only Memories) στα οποία οι ασφάλειες είναι ηλεκτρικά ελεγχόμενες. Σαν αποτέλεσμα ο προγραμματισμός του ολοκληρωμένου μπορεί να διαγραφεί άμεσα με την χρησιμοποίηση κατάλληλων τάσεων.



Ο χρόνος αποκατάστασης των ασφαλειών και οι χαμηλές ταχύτητες λειτουργίας είναι τα κύρια μειονεκτήματα των E<sup>2</sup>PROM. Άλλα μειονεκτήματά τους είναι ο εκθετικός αριθμός γινομένων σε σχέση με τις εισόδους που προκύπτει από το γεγονός ότι χρησιμοποιούν ένα σταθερό πίνακα AND πυλών και η αδυναμία τους για υλοποίηση ακολουθιακών κυκλωμάτων. Παρά τα μειονεκτήματά τους οι E<sup>2</sup>PROM χρησιμοποιούνται ευρέως ακόμη και σήμερα. Εναλλακτική πρόταση είναι οι κατά πολύ ταχύτερες στατικές μνήμες (SRAM) που έχουν το μειονέκτημα του πολύ αυξημένου κόστους για ίδιο μέγεθος και φυσικά ότι ο προγραμματισμός τους χάνεται με τη πτώση της τάσης. Εναλλακτική, ενδιάμεση των δύο προτάσεων από πλευράς ταχύτητας και κόστους λύση είναι οι μνήμες Flash. Οι μνήμες αυτές είναι non-volatile (διατηρούν δηλαδή τα δεδομένα τους χωρίς ηλεκτρική τάση αλλά για σημαντικά μικρότερο χρονικό διάστημα από ότι οι E<sup>2</sup>PROMS). Αν και ο διαχωρισμός του ποια ολοκληρωμένα είναι E<sup>2</sup>PROMS και ποια Flash δεν είναι σαφής, μιας και οι σημερινές E<sup>2</sup>PROMS έχουν αποκτήσει αρκετά χαρακτηριστικά των SRAM μνημών όπως για παράδειγμα τη διαγραφή μιας μόνο λέξης και την αντικατάστασή τους από άλλη, ο εμπορικός κόσμος συνήθως θεωρεί Flash εκείνα τα ολοκληρωμένα που παρέχουν

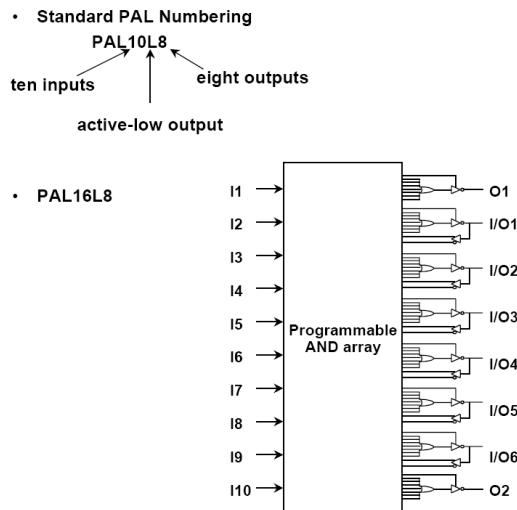
εκτός των άλλων πολύ γρήγορες διαδικασίες για περιοχές διευθύνσεων (fast page modes). Ετσι για παράδειγμα ένα ολοκληρωμένο στο οποίο ο χρόνος διαγραφής ενός κομματιού διευθύνσεων είναι πολύ μικρότερος από αυτόν που του αντιστοιχεί κατά τη διαγραφή όλων των δεδομένων του ολοκληρωμένου, μπορεί να θεωρηθεί σα ολοκληρωμένο τεχνολογίας Flash.

Τα πρώτα ολοκληρωμένα προγραμματιζόμενης λογικής που δεν ακολούθησαν τη φιλοσοφία των PROM, ήταν τα ολοκληρωμένα λογικής προγραμματιζόμενου πίνακα (Programmable Array Logic – PAL). Στα ολοκληρωμένα αυτά αντί να είναι προγραμματιζόμενο το OR επίπεδο, το επίπεδο αυτό κρατήθηκε σταθερό και ο προγραμματισμός μεταφέρθηκε στο επίπεδο AND. Το παρακάτω σχήμα δείχνει ένα PAL πριν και μετά το προγραμματισμό του.

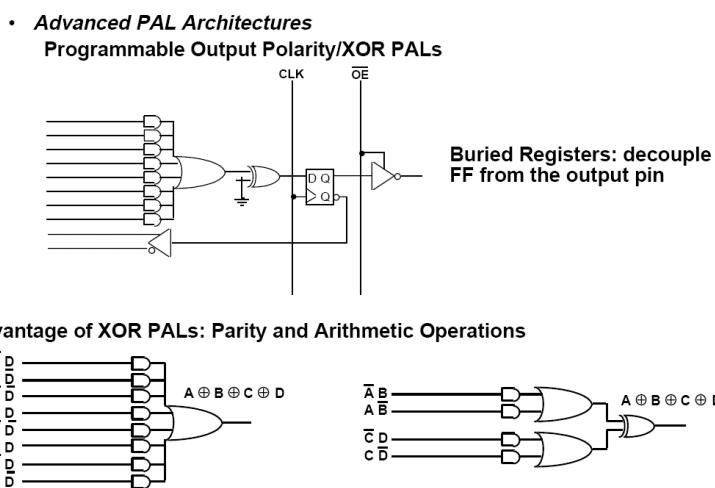


Συγκρίνοντας τα σχήματα των PAL και των PROM μπορεί να κανείς να διαπιστώσει ότι οι PAL προσφέρουν σαφώς καλύτερους χρόνους. Αυτό γιατί κάθε OR των PROM δέχεται όλα τα γινόμενα, που είναι εκθετικά σε αριθμό σε σχέση με τις εισόδους, ενώ στην περίπτωση των PAL κάθε AND τους δέχεται το διπλάσιο αριθμό των εισόδων. Οπως και στην περίπτωση των TTL υπάρχουν διαθέσιμα πάρα πολλά ολοκληρωμένα PAL. Η ονοματολογία τους εξηγείται στο παρακάτω σχήμα, όπου βλέπουμε ότι ο

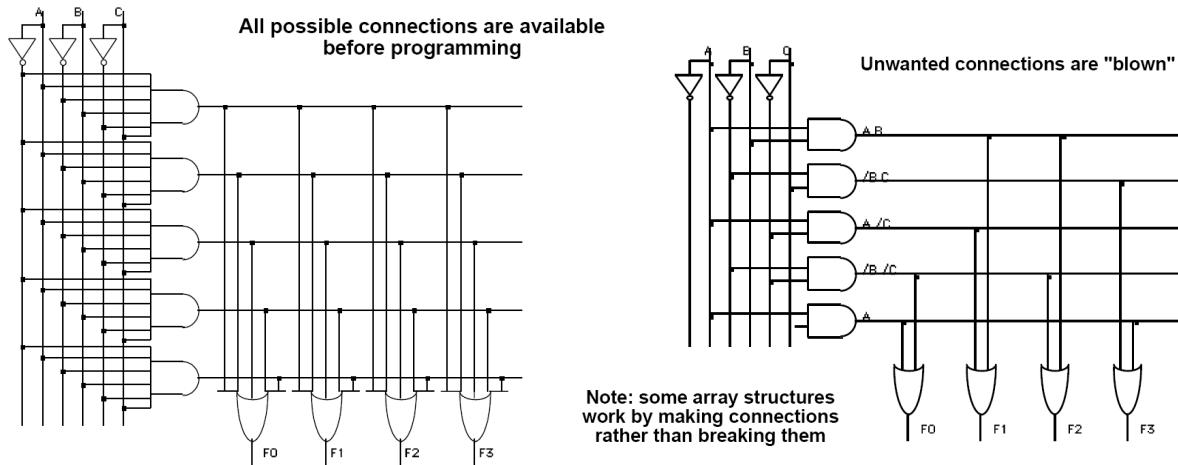
πρώτος αριθμός μας δείχνει τον αριθμό των εισόδων, ο τελευταίος αυτόν των εξόδων ενώ το ενδιάμεσο γράμμα καθορίζει αν χρησιμοποιείται θετική ή αρνητική λογική. Το σχήμα επίσης δείχνει μία από τις εξελίξεις αυτών των ολοκληρωμένων όπου βλέπουμε ένα υποσύνολο των εξόδων ότι μπορεί να θεωρείται σαν επιπλέον είσοδος στο κύκλωμα και συνεπώς να λαμβάνει μέρος με τη κανονική ή τη συμπληρωματική μορφή στο πίνακα AND. Τέτοιες ιδέες έχουν σκοπό την αύξηση της ευελιξίας και συνεπώς τη χρησιμοποίηση μεγαλύτερου ποσοστού των πόρων του ολοκληρωμένου.



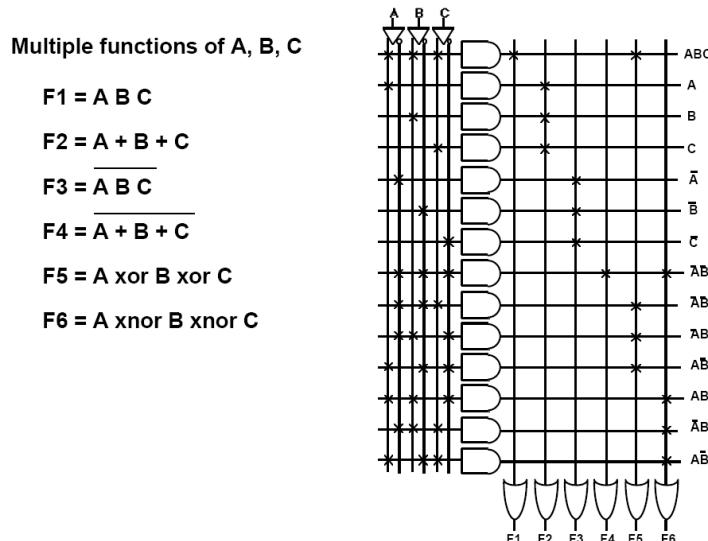
Μερικές άλλες ιδέες που έχουν χρησιμοποιηθεί φαίνονται στην παρακάτω εικόνα. Η χρήση μιας πύλης XOR αμέσως μετά το επίπεδο OR μπορεί να μας δώσει διπλή πολικότητα σε συνάρτηση με την τιμή της άλλης εισόδου της. Επιπλέον μπορεί να διευκολύνει πάρα πολύ στην υλοποίηση αριθμητικών συναρτήσεων όπως φαίνεται στο κάτω σχήμα. Πολύ σημαντική είναι επίσης η προσθήκη καταχωρητών στην έξοδο μιας PAL των οποίων η έξοδος χρησιμοποιείται σαν όλες τις υπόλοιπες είσοδες. Με τη προσθήκη αυτών των καταχωρητών οι PAL μπορούν πλέον να χρησιμοποιηθούν για την υλοποίηση και σύγχρονων ακολουθιακών κυκλωμάτων.



Στις PAL το ποσοστό χρησιμοποίησης των πόρων που υπάρχουν μέσα στο ολοκληρωμένο είναι μικρό, λόγω της πολύ μικρής ευελιξίας που υπάρχει στο επίπεδο OR. Με στόχο την αύξηση της ευελιξίας, τα επόμενα ολοκληρωμένα τα οποία ονομάζονται ολοκληρωμένα πίνακα προγραμματιζόμενης λογικής (Programmable Logic Array – PLAs) δίνουν τη δυνατότητα προγραμματισμού τόσο στο επίπεδο AND όσο και στο επίπεδο OR. Τα παρακάτω σχήματα δείχνουν ένα PLA πριν και μετά τον προγραμματισμό του.



Ενα άλλο παράδειγμα που μας δείχνει τον προγραμματισμό έξι συναρτήσεων σε ένα μόνο PLA φαίνεται στο παρακάτω σχήμα :



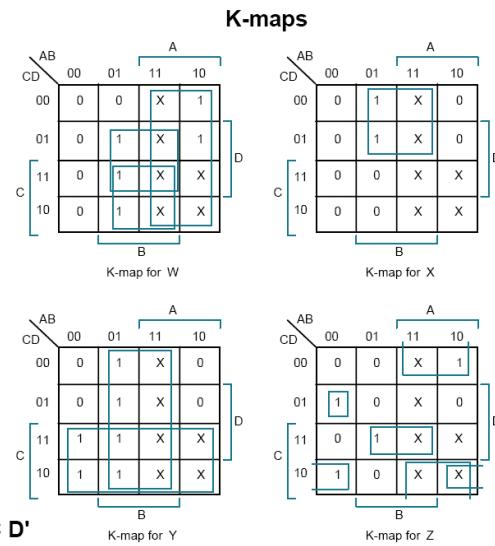
Αν και τα PLAs προσφέρουν σημαντικά μεγαλύτερη ευελιξία, ωστόσο πάσχουν από ένα πολύ σοβαρό πρόβλημα. Στο παραπάνω σχήμα αν και έχουμε τρεις μεταβλητές εισόδου έχουμε τη γέννηση 14 γινομένων ! Στην πραγματικότητα συνήθως συμβαίνει το αντίθετο. Στα PLAs ο αριθμός των γινομένων που παράγονται είναι μικρότερος του  $2^m$ , όπου m συμβολίζει τον αριθμό των εισόδων. Αποτέλεσμα αυτού είναι να μη

μπορούμε πάντα να παράγουμε όλες τις επιθυμητές συναρτήσεις ή να μη μπορούμε στο ίδιο ολοκληρωμένο να παράγουμε ταυτόχρονα όλες τις συναρτήσεις που επιθυμούμε.

Στα παρακάτω σχήματα συνοψίζονται με ένα παράδειγμα τα όσα έχουμε επιτύχει μέχρι ώρας. Στόχος μας είναι να κατασκευάσουμε ένα κύκλωμα το οποίο μετατρέπει BCD σε Gray. Στα σχήματα φαίνονται οι πίνακες αληθείας των συναρτήσεων, οι πίνακες Karnaugh για την απλοποίησή τους, η υλοποίησή τους με SSI ολοκληρωμένα καθώς και η υλοποίηση που μπορούμε να πετύχουμε με μία μόνο PAL. Οπως βλέπουμε η PAL μας επιτρέπει την αντικατάσταση 4 SSI από ένα μόνο ολοκληρωμένο.

#### Design Example: BCD to Gray Code Converter

Truth Table							
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	1	1	0	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



#### Minimized Functions:

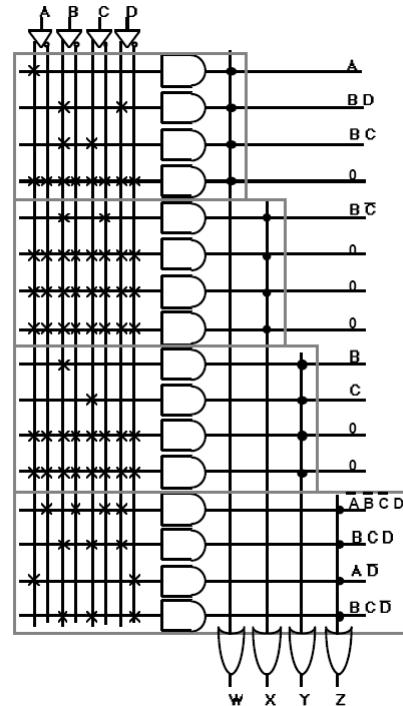
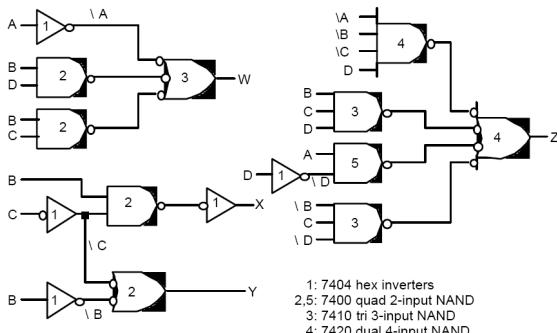
$$W = A + BD + BC$$

$$X = B'C'$$

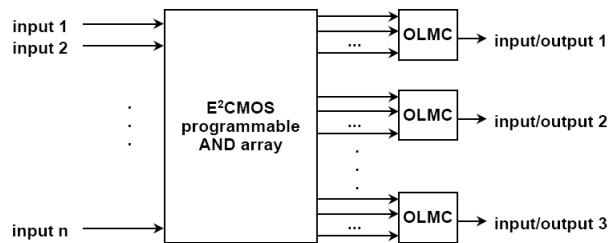
$$Y = B + C$$

$$Z = A'B'C'D + B'CD + AD' + B'C'D'$$

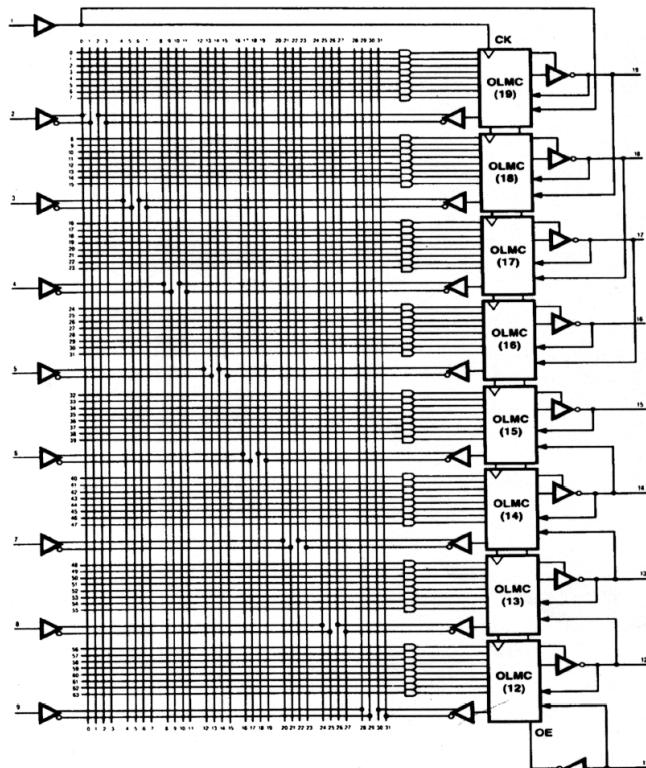
#### Code Converter Discrete Gate Implementation



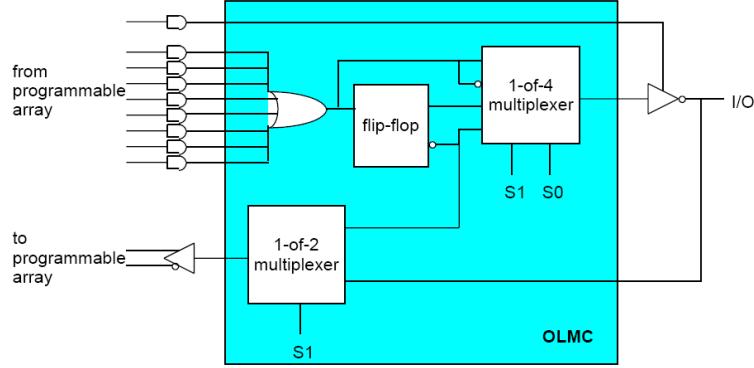
Είναι προφανές ωστόσο ότι ακόμα και η χρήση των PAL / PLAs δεν είναι ικανή να καλύψει τις ανάγκες μας για δραστική μείωση των ολοκληρωμένων που απαιτούνται στη πλακέτα ενός σύγχρονου σχεδιασμού. Το επόμενο ερώτημα λοιπόν που μας απασχολεί είναι πως μπορούμε να δώσουμε τη δυνατότητα στα PLDs να μπορούν να αντικαταστήσουν σημαντικά πιο πολύπλοκους σχεδιασμούς. Προσέξτε ότι αυτό δε μπορεί να επιτευχθεί με το να μεγαλώσουμε τους πίνακες που υπάρχουν σε μία PAL / PLA, μιας και τότε θα κληθούμε να πληρώσουμε το τίμημα του μεγαλύτερου χρόνου καθυστέρησης λόγω της χρήσης πυλών με περισσότερες εισόδους. Ενα πρώτο μικρό βήμα έγινε με την εισαγωγή των ολοκληρωμένων πίνακα πυλών (Gate Array Logic – GAL). Τα ολοκληρωμένα αυτά μπορούν να αντικαταστήσουν έναν αριθμό από PALS. Χρησιμοποιούν ένα επαναπρογραμματιζόμενο επίπεδο AND, ένα σταθερό επίπεδο OR αλλά εισάγουν την έννοια των κυττάρων εξόδου (Output Logic Macro Cells – OLMCs), τα οποία παρέχουν σημαντική ευελιξία. Συνοπτικά η αρχιτεκτονική τους φαίνεται στο παρακάτω σχήμα,



το οποίο οδηγεί σε υλοποιήσεις της μορφής :

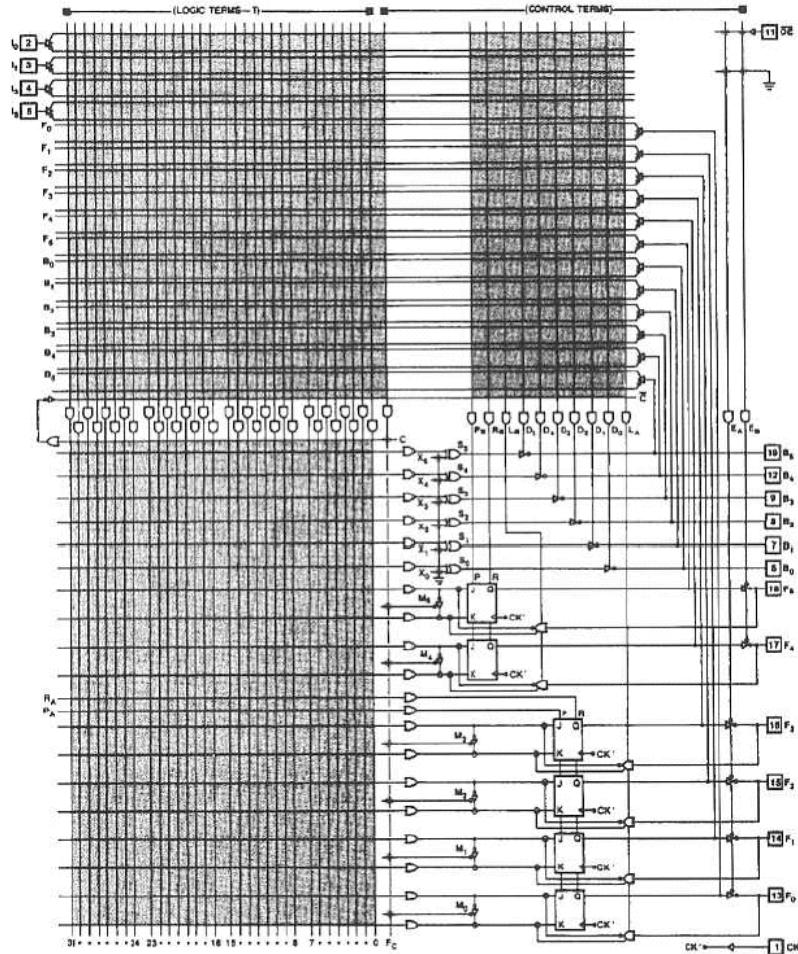


Μιας και τα υπόλοιπα σχεδιαστικά κομμάτια είναι γνωστά και πανομοιότυπα με αυτά μιας PAL, στη συνέχεια επικεντρωνόμαστε στα OLMC, των οποίων η αρχιτεκτονική φαίνεται στο παρακάτω σχήμα :



**s1 s0 = 00** --> registered mode with active-LOW output  
**= 01** --> registered mode with active-HIGH output  
**= 10** --> combinational mode with active-LOW output  
**= 11** --> combinational mode with active-HIGH output

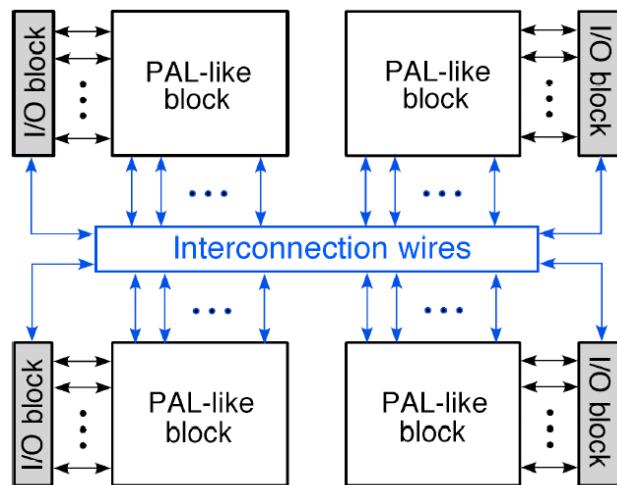
Οπως βλέπουμε ο σχεδιαστής έχει τη δυνατότητα μέσω του καθορισμού των σημάτων  $S_1$  και  $S_0$  να επιλέξει τόσο τη πολικότητα όσο και το αν το σήμα εξόδου του θα καταχωρηθεί στο ακολουθιακό στοιχείο, δίνοντας τη δυνατότητα για σύγχρονα ακολουθιακά κυκλώματα.



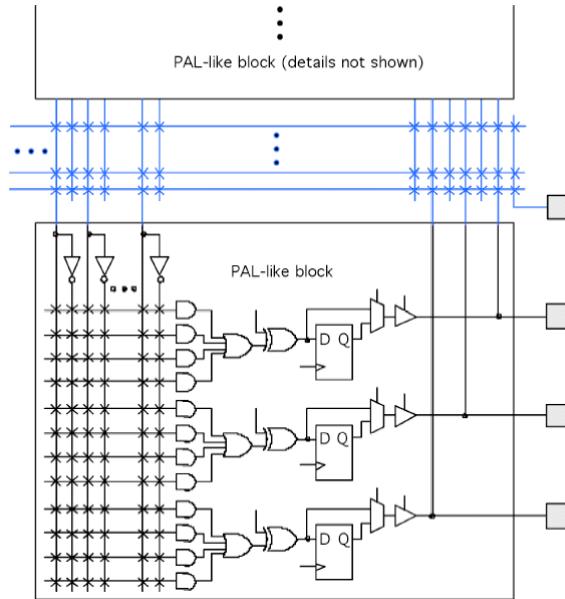
Οι προγραμματιζόμενοι λογικοί ακολουθητές προγράμματος (FPLS) είναι τα πρώτα ολοκληρωμένα που προχώρησαν στην εμφάλευση των καταχωρητών εντός του ολοκληρωμένου. Η έξοδος αυτών των καταχωρητών διαχειρίζεται από το ολοκληρωμένο σαν κάθε άλλη είσοδος. Ωστόσο έχει προστεθεί άλλος ένας πίνακας AND (ο πίνακας των στοιχείων ελέγχου) που έχει σα στόχο να διαχειρίζεται τα σήματα υψηλής εμπέδησης, τα σήματα καθαρισμού και θέσης των καταχωρητών κ.ο.κ. Στην πραγματικότητα τα FPLS ήταν τα πρώτα ολοκληρωμένα που έδιναν τη δυνατότητα να υλοποιήσουμε μια μηχανή πεπερασμένων καταστάσεων μέσα σε ένα ολοκληρωμένο.

Καθώς η τεχνολογία εξελισσόταν ήμασταν πλέον σε θέση να τοποθετήσουμε στο ίδιο ολοκληρωμένο μεγάλους αριθμούς από PALs, GALs ή FPLS. Κάτι τέτοιο είναι προφανώς θεμιτό μιας και μειώνει το χρόνο που χάνεται κατά την ανταλλαγή σημάτων μεταξύ ολοκληρωμένων, μειώνει τον αριθμό των ολοκληρωμένων στη πλακέτα με άμεσο αντίκτυπο στην αξιοπιστία (λιγότερες διασυνδέσεις και λιγότερος επηρεασμός από το περιβάλλον), στο εμβαδόν και στην κατανάλωση ισχύος. Το πρόβλημα λοιπόν τώρα παίρνει μια νέα διάσταση, καθώς θα πρέπει οι νέες αρχιτεκτονικές να προτείνουν τρόπους για την ανταλλαγή πληροφορίας μεταξύ αυτών των μεγακυττάρων. Προφανώς μια πλήρης διασύνδεση δεν είναι εφικτή λόγω του αριθμού των μεγακυττάρων. Εδώ παρουσιάστηκαν δύο φιλοσοφίες :

- Η χρήση ενός κεντροποιημένου συστήματος διασύνδεσης στα πολύπλοκα ολοκληρωμένα προγραμματιζόμενης λογικής (Complex PLDs – CPLDs) και
- Η χρήση ενός ιεραρχικού αλλά αποκεντρωμένου συστήματος διασύνδεσης στα προγραμματιζόμενα από το χρήστη ολοκληρωμένα πίνακα πυλών (Field Programmable Gate Arrays – FPGAs), που θα αποτελέσουν αντικείμενο του επόμενου κεφαλαίου, αλλά και πλατφόρμα για τις εργαστηριακές σας ασκήσεις.



Στο παραπάνω σχήμα φαίνεται η αρχιτεκτονική ενός CPLD. Αποτελείται από τις μονάδες εισόδου – εξόδου, τα μεγακύτταρα και το σύμπλεγμα των καλωδίων που τα ενώνει μεταξύ τους. Ολοι καταλαβαίνουμε ότι αυτή η αρχιτεκτονική προσομοιάζει εκείνη ενός υπολογιστικού συστήματος με τις μονάδες I/O και τις μονάδες επεξεργασίας που επικοινωνούν πάνω από μια διαμοιραζόμενη αρτηρία. Η μορφή και η διασύνδεση των μεγακυττάρων μεταξύ τους φαίνεται στο ακόλουθο σχήμα:



Κάθε μεγακύτταρο παίρνει τις εισόδους του από άλλα ή από τις μονάδες εισόδου, έχει ένα επίπεδο AND, ένα OR και μία πύλη XOR που καθορίζει τη πολικότητα της εξόδου. Η έξοδος μπορεί να οδηγηθεί σε καταχωρητή και ακολούθως να αποτελέσει είσοδο στο ίδιο ή άλλο μεγακύτταρο ή να περάσει στην έξοδο μέσω ενός απομονωτή τριών καταστάσεων.

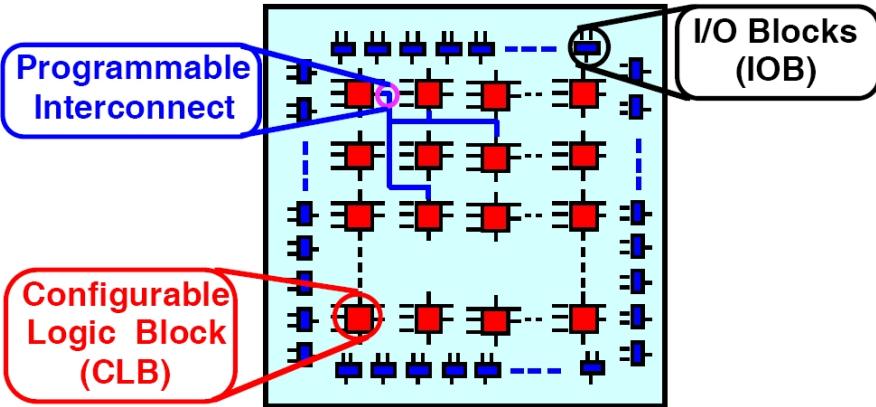


---

## Κεφάλαιο 3

# FPGAs

Τα FPGAs είναι σήμερα (2004) ότι πιο σύγχρονο έχει να επιδείξει η οικογένεια των ολοκληρωμένων προγραμματιζόμενης λογικής. Σε αντίθεση με τα CPLDs ακολουθείται ένα αρκετά πιο πολύπλοκο, ιεραρχικό μα ταυτόχρονα αποκεντρωμένο σύστημα διασύνδεσης των μεγακυττάρων προγραμματιζόμενης λογικής. Παράλληλα, τα μεγακύτταρα αυτά γίνονται σημαντικά απλούστερα και συνεπώς μικρότερα δίνοντας έτσι στις σύγχρονες φωτολιθογραφικές μεθόδους τη δυνατότητα να εμφωλεύσουν ακόμη και εκατομμύρια από αυτά μέσα στο ίδιο ολοκληρωμένο. Η ονομασία αυτών των υποσχεδιασμών προγραμματιζόμενης λογικής διαφέρει από κατασκευαστή σε κατασκευαστή. Εμείς στη συνέχεια ακολουθούμε την ονοματολογία του κατασκευαστή Xilinx που τους δίνει την ονομασία CLBs (Complex Logic Blocks). Το παρακάτω σχήμα δείχνει συνοπτικά τη γενική αρχιτεκτονική ενός FPGA.



Οπως βλέπουμε τα δομικά στοιχεία από τα οποία απαρτίζεται το FPGA είναι :

1. Οι υποσχεδιασμοί προγραμματιζόμενης λογικής (CLBs).
2. Οι υποσχεδιασμοί για την είσοδο – έξοδο του ολοκληρωμένου (Input – Output Blocks – IOB).
3. Η προγραμματιζόμενη διασύνδεση (programmable interconnect).

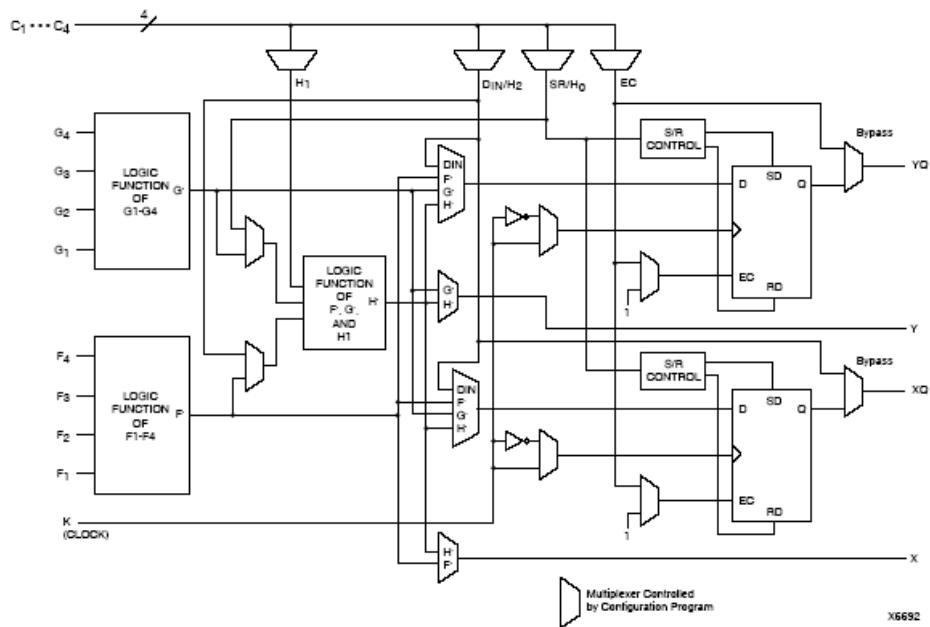
Κάθε κατασκευαστής ολοκληρωμένων προτείνει τις δικές του επιλογές για την αρχιτεκτονική του καθενός από αυτούς τους υποσχεδιασμούς. Πολλά δε επιστημονικά άρθρα έχουν παρουσιαστεί στα πλέον έγκυρα περιοδικά και συνέδρια της επιστήμης μας (IEEE Transactions on Computers, IEEE Micro, κλπ.) σχετικά με το ποια είναι η καλύτερη αρχιτεκτονική. Ωστόσο από το γεγονός και μόνο ότι ο ίδιος κατασκευαστής παρέχει πληθώρα διαφορετικών αρχιτεκτονικών στα ολοκληρωμένα που παράγει, συμπεραίνει κανείς ότι δεν υπάρχει μία και μόνο αρχιτεκτονική βέλτιστη για όλους τους σχεδιασμούς που προτιθέμεθα να εμφωλεύσουμε. Αυτό γεννάει την ανάγκη να εστιάσουμε σε μία από τις διαθέσιμες αρχιτεκτονικές, αν και η φιλοσοφία παραμένει σταθερή σε όλα τα ολοκληρωμένα. Παρακάτω συνεπώς αναλύουμε διεξοδικά την αρχιτεκτονική των CLB, IOB και προγραμματιζόμενης διασύνδεσης για τη σειρά ολοκληρωμένων XC4000E/X του κατασκευαστή Xilinx. Αν κι αυτή η σειρά θεωρείται πλέον απαρχαιωμένη, η εξέτασή της έχει σημαντική εκπαιδευτική αξία.

Στο παρακάτω σχήμα φαίνεται μια απλουστευμένη έκδοση του CLB γι' αυτή τη σειρά ολοκληρωμένων. Οπως μπορούμε να δούμε το CLB απαρτίζεται από τα ακόλουθα σχεδιαστικά κομμάτια :

- ♦ Δύο LUT των τεσσάρων μεταβλητών εισόδου το καθένα. Τα LUT αυτά ονομάζονται F και G αντίστοιχα.
- ♦ Ένα LUT των τριών μεταβλητών εισόδου. Το LUT αυτό ονομάζεται H. Στην περίπτωση που τα F, G και H LUT χρησιμοποιούνται για την υλοποίηση συνδυαστικών συναρτήσεων, τα δεδομένα τους γράφονται σε αυτά κατά την

εκκίνηση λειτουργίας του FPGA, βάσει μιας διαδικασίας που ονομάζεται διάρθρωσή (configuration) του.

- ♦ Δύο καταχωρητές (D flip-flops), τα οποία μπορούν να λειτουργήσουν είτε με τη θετική είτε με την αρνητική ακμή του σήματος ρολογιού (K).
- ♦ Ενα κύκλωμα ελέγχου του καθαρισμού ή της θέσης των καταχωρητών που κι αυτό καθορίζεται κατά τη διάρθρωση του FPGA.
- ♦ Μια πλειάδα πολυπλεκτών για τον ορισμό της επιθυμητής διασύνδεσης μεταξύ αυτών των υποσχεδιασμών και της λογικής ακμοπυροδότησης των καταχωρητών. Τα σήματα ελέγχου αυτών των πολυπλεκτών προέρχονται από μία μνήμη που εγγράφεται κατά τη φάση διάρθρωσης.



Είναι προφανές ότι με τη χρήση των LUT που υπάρχουν στο CLB και των σημάτων ελέγχου των πολυπλεκτών μπορούμε να υλοποιήσουμε :

- ♦ Οποιεσδήποτε δύο συναρτήσεις έως και τεσσάρων ανεξάρτητων μεταξύ τους μεταβλητών μαζί με οποιαδήποτε συνάρτηση τριών ανεξάρτητων μεταβλητών. (Προφανώς επειδή από το CLB υπάρχουν μόνο δύο συνδυαστικές έξοδοι μία εκ των τριών συναρτήσεων θα πρέπει να οδηγηθεί σε κάποιον από τους καταχωρητές).
- ♦ Οποιαδήποτε συνάρτηση πέντε μεταβλητών.
- ♦ Οποιαδήποτε συνάρτηση τεσσάρων μεταβλητών παράλληλα με μερικές συναρτήσεις των έξι μεταβλητών.
- ♦ Μερικές συναρτήσεις έως και εννέα μεταβλητών.

Η δυνατότητα που μας δίνεται από το συγκεκριμένο CLB για την υλοποίηση ακόμα και συναρτήσεων πολλών μεταβλητών, οδηγεί στη μείωση του αριθμού των CLBs που

---

χρειάζονται για την υλοποίηση ενός σχεδιασμού αλλά και σε αυξημένη ταχύτητα μιας και αποφεύγεται διάδοση των σημάτων μεταξύ CLBs. Θα πρέπει βέβαια πάντοτε να λαμβάνουμε υπόψη μας ότι μιας και οι λογικές συναρτήσεις εντός του CLB υλοποιούνται με μνήμη, ο χρόνος υπολογισμού για μια συνάρτηση δύο ή τεσσάρων μεταβλητών είναι ο ίδιος.

Ο πιο κάτω πίνακας είναι ο πίνακας αληθείας για τους δύο καταχωρητές που υπάρχουν μέσα σε κάθε CLB.

Mode	K	EC	SR	D	Q
Power-Up or GSR	X	X	X	X	SR
Flip-Flop	X	X	1	X	SR
	/	1*	0*	D	D
	0	X	0*	X	Q
Latch	1	1*	0*	X	Q
	0	1*	0*	D	D
Both	X	0	0*	X	Q

Legend:

X	Don't care
/	Rising edge
SR	Set or Reset value. Reset is default.
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

Οι καταχωρητές αυτοί δέχονται το ίδιο ρολόι και το ίδιο σήμα επίτρεψης ρολογιού (EC). Οπως βλέπουμε και από το πίνακα αληθείας οι καταχωρητές αυτοί μπορούν να λειτουργήσουν σα flip flops ή σα latches. Οπως αναφέραμε προηγούμενα η πολικότητα της ενεργοποίησης είναι προγραμματιζόμενη ανεξάρτητα για κάθε καταχωρητή στο 1 ή το 0 του ρολογιού όταν λειτουργούν σα latches και στην αρνητική ή τη θετική ακμή όταν λειτουργούν ως flip flops. Οι υποσχεδιασμοί S/R control, ελέγχουν την ασύγχρονη θέση ή καθαρισμό των καταχωρητών. Οι υποσχεδιασμοί αυτοί πέρα από τη χρήση τους για την ασύγχρονο καθορισμό της τιμής των καταχωρητών μέσω του σήματος SR / Η ο χρησιμοποιούνται και για να καθορίσουν την έξοδο των καταχωρητών στην κατάσταση αρχικοποίησης (η κατάσταση αυτή υποδηλώνεται με ένα παλμό σε ένα σχεδιασμό που ονομάζεται γενικευμένος σχεδιασμός θέσης / καθαρισμού – Global Set / Reset – GSR).

Οπως είπαμε πιο πάνω, τα περιεχόμενα των LUT, οι τιμές των σημάτων ελέγχου των πολυπλεκτών και των S/R υποκυλωμάτων μπορεί να καθοριστούν κατά τη φάση διάρθρωσης του FPGA. Σε πολλές εφαρμογές όμως παράλληλα με τον υπολογισμό συναρτήσεων, χρειάζεται και αποθήκευση επιμέρους αποτελεσμάτων. Αφού η απαιτούμενη μνήμη είναι διαθέσιμη εντός του CLB, θα ήταν παράλογο για την αποθήκευση αυτή να χρησιμοποιήσουμε κάποιο άλλο ολοκληρωμένο μνήμης.

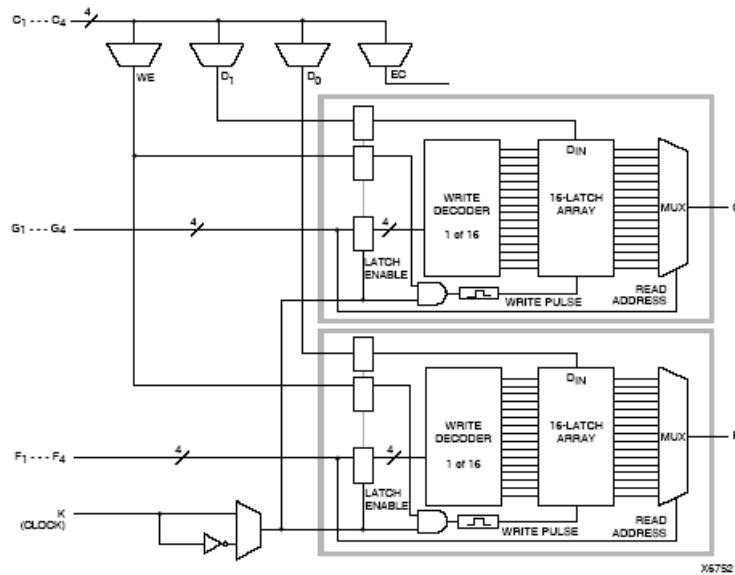
---

Συνεπώς το CLB θα πρέπει να μας δίνει την εναλλακτική δυνατότητα να μπορούμε να χρησιμοποιούμε τα LUT σα μονάδες μνήμης.

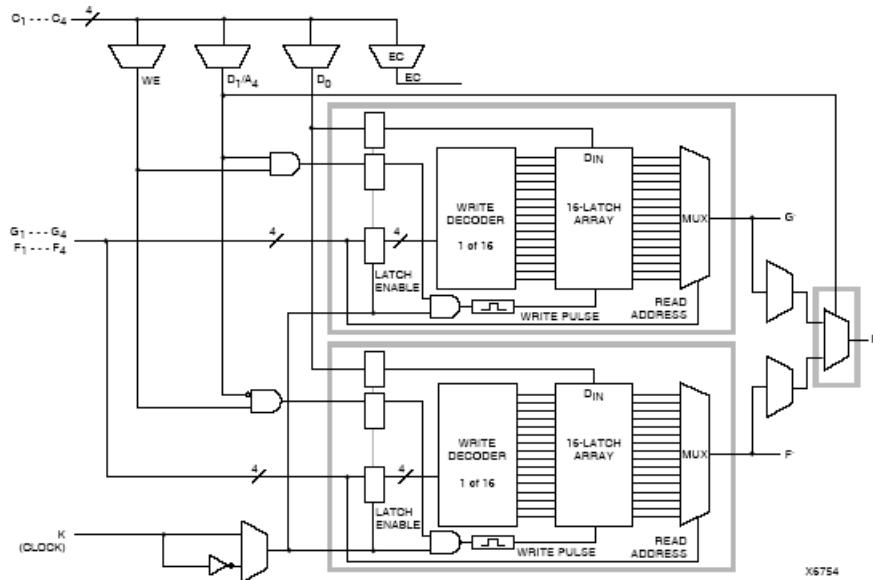
Το CLB που εξετάζουμε παρέχει πολλές δυνατότητες για χρήση του ως μνήμη. Μπορούμε χρησιμοποιώντας το να φτιάξουμε μνήμη μίας θύρας (one port memory – μονάδα μνήμης που προσφέρει μόνο ανάγνωση ή μόνο εγγραφή ανά πάσα στιγμή) που να λειτουργεί είτε με παλμό είτε με ακμή του ρολογιού, ή μνήμη δύο θυρών (dual port memory – μονάδα μνήμης στην οποία μπορούμε παράλληλα να διαβάζουμε από δύο ξεχωριστές διευθύνσεις). Οι δυνατότητες αυτές συνοψίζονται στο παρακάτω πίνακα :

	<b>16 x 1</b>	<b>16 x 2</b>	<b>32 x 1</b>	<b>Edge- Triggered Timing</b>	<b>Level- Sensitive Timing</b>
<b>Single-Port</b>	✓	✓	✓	✓	✓
<b>Dual-Port</b>	✓			✓	

Τα παρακάτω σχήματα μας δείχνουν συνοπτικά τις διαρθρώσεις του CLB για την επίτευξη μνήμης 16x1 (ή 16x2) και 32x1. Οι είσοδοι F και G γίνονται πλέον γραμμές διευθύνσεων που επιλέγουν ένα στοιχείο μνήμης μέσα σε κάθε LUT. Οι γραμμές H<sub>2</sub>, H<sub>1</sub> και H<sub>0</sub> δρουν σαν οι δύο γραμμές δεδομένων εισόδου (D<sub>0</sub> και D<sub>1</sub>) των μνημών και σαν το σήμα επίτρεψης εγγραφής (WE) στην περίπτωση διάρθρωσης σα μνήμη 16x2. Για τη διάρθρωση μνήμης 32x1 το H<sub>1</sub> γίνεται το πέμπτο ψηφίο της διευθυνσιοδότησης. Προσέξτε ότι τίποτε δεν αλλάζει τη λειτουργία των υπόλοιπων ιομματιών που υπάρχουν μέσα στο CLB. Για παράδειγμα στο πρώτο σχήμα θα μπορούσαμε να χρησιμοποιήσουμε μόνο το ένα LUT σα μνήμη 16x1 και να χρησιμοποιήσουμε τα υπόλοιπα δύο LUT για την υλοποίηση μιας συνάρτησης πέντε μεταβλητών. Επίσης στο δεύτερο σχήμα, τίποτε δε μας εμποδίζει τις εξόδους που διαβάζουμε από τις μνήμες να τις οδηγήσουμε στους καταχωρητές που υπάρχουν στο CLB.

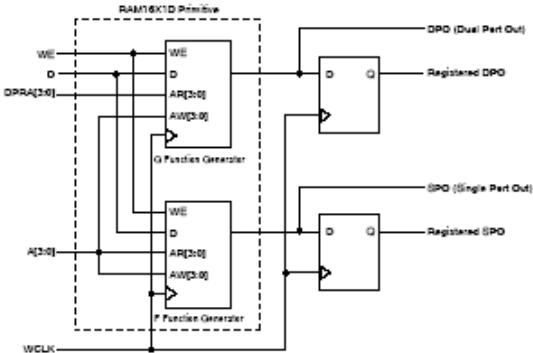


16x2 (or 16x1) Edge-Triggered Single-Port RAM



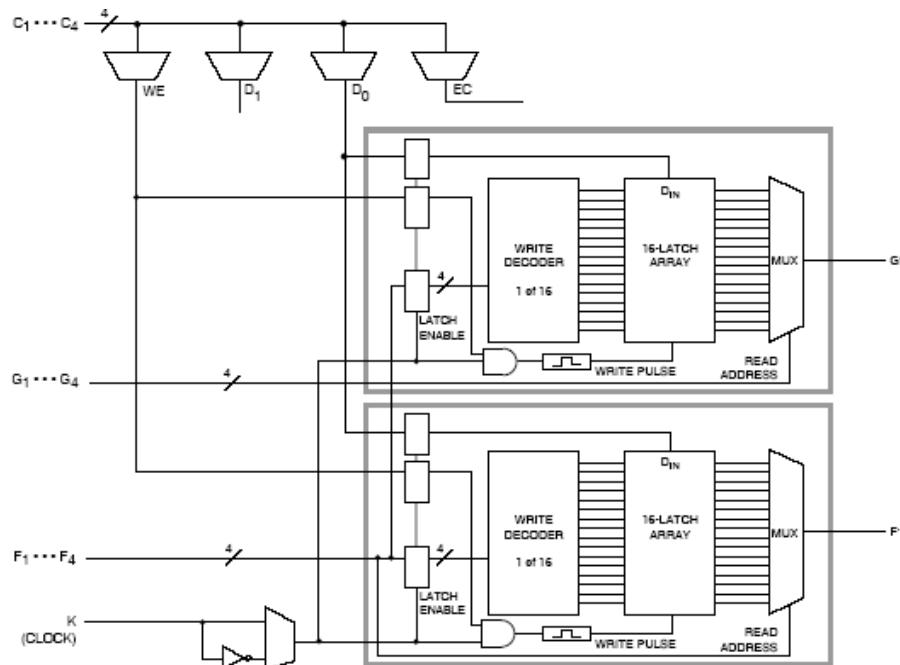
32x1 Edge-Triggered Single-Port RAM (F and G addresses are identical)

Για να φτιάξουμε τώρα μια μνήμη δύο θυρών χρησιμοποιούμε παράλληλα τα δύο 16x1 LUT. Το παρακάτω σχήμα είναι ένα απλό μοντέλο του πως διαρθρώνεται το CLB σε αυτή τη περίπτωση και ο διπλανός πίνακας μας δίνει την αντιστοιχία των σημάτων του μοντέλου με αυτά του CLB. Κάθε LUT έχει μια διεύθυνση εγγραφής και μία ανάγνωσης. Προσέξτε όμως ότι η διεύθυνση εγγραφής είναι κοινή και για τα δύο LUT (προέρχεται από τις εισόδους F) ενώ η διεύθυνση ανάγνωσης μπορεί να είναι διαφορετική (προέρχεται από τις εισόδους F και G για τα αντίστοιχα LUT). Σαν αποτέλεσμα αυτού είναι η κάθε πληροφορία να γράφεται παράλληλα και στα δύο LUT, αλλά με παράλληλες διευθύνσεις στις εισόδους F και G για τα αντίστοιχα LUT.



RAM Signal	CLB Pin	Function
D	D0	Data In
A[3:0]	F1-F4	Read Address for F, Write Address for F and G
DPRA[3:0]	G1-G4	Read Address for G
WE	WE	Write Enable
WCLK	K	Clock
SPO	F'	Single Port Out (addressed by A[3:0])
DPO	G'	Dual Port Out (addressed by DPRA[3:0])

Η πραγματική διάρθρωση του FPGA για την επίτευξη μνήμης δύο θυρών φαίνεται στο παρακάτω σχήμα.



Οπως είπαμε αρχικά η μέχρι ώρας περιγραφή του συγκεκριμένου CLB αποκρύπτει ορισμένες λεπτομέρειες. Από τα πλέον συνήθη κυκλώματα που καλούνται τα FPGA να φιλοξενήσουν είναι τα αριθμητικά κυκλώματα, κυρίως αυξητές, συγκριτές, προσθετές και αφαιρέτες. Για την επίτευξη μεγάλων ταχυτήτων σε αυτά τα κυκλώματα, τα CLBs συνήθως διαθέτουν επιπλέον λογική για γρήγορη διάδοση του κρατουμένου (γνωστή ως fast carry logic). Στη περίπτωση αυτή τα F και G LUT μπορούν να διαρθρωθούν σαν αθροιστές των δύο δυαδικών ψηφίων το καθένα με ειδική πρόνοια για τη διάδοση του κρατουμένου εξόδου σε επόμενο γειτονικό CLB. Σαν αποτέλεσμα αυτού προκύπτει το να είναι ανόητο να προσπαθήσουμε να εμφωλεύσουμε σε ένα FPGA άλλη αρχιτεκτονική για αθροιστές / αφαιρέτες πέραν αυτής με διάδοση κρατουμένου (ripple carry). Με άλλα λόγια ένας carry look ahead αθροιστής των 16 δυαδικών ψηφίων θα είναι αργότερος από έναν αντίστοιχο ripple

---

carry αν και οι δύο υλοποιηθούν σε FPGA και ο δεύτερος κάνει χρήση αυτών των ειδικών δυνατοτήτων των CLBs.

Θα πρέπει να σημειώσουμε ότι σε επίπεδο φυσικού σχεδιασμού το κάθε CLB υλοποιείται σαν ένα ορθογώνιο ή ένα τετράγωνο. Οι είσοδοι / έξοδοι του CLB είναι φυσικά διεσπαρμένες και στις τέσσερις πλευρές του CLB. Ετσι είναι πιθανό να χρησιμοποιήσουμε ένα CLB και για λόγους διαδρόμισης, δηλαδή για να περάσουμε απλά την είσοδο στην έξοδο που βρίσκεται σε κάποια άλλη πλευρά.

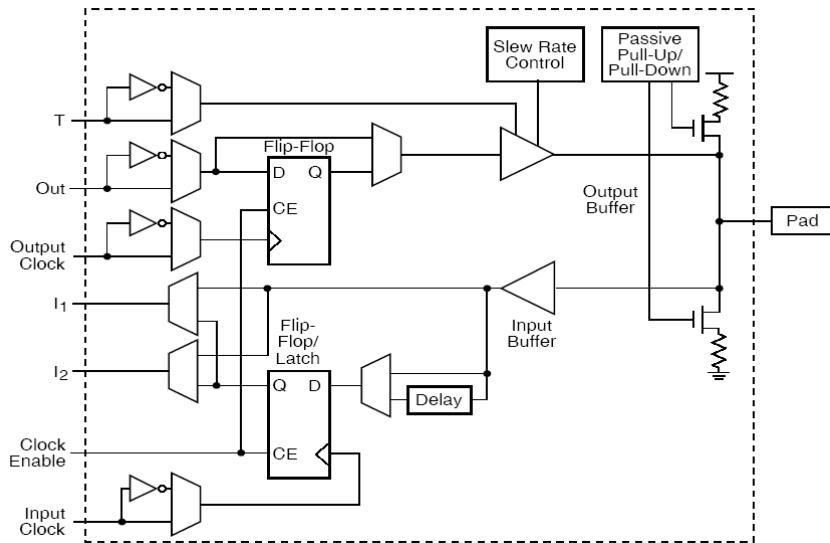
Κλείνοντας την αναφορά μας στα CLB, είναι καλό να δούμε πόσα CLBs συνήθως βρίσκουμε σε ένα τέτοιο ολοκληρωμένο. Ο παρακάτω πίνακας συνοψίζει τα διαθέσιμα ολοκληρωμένα αυτής της οικογένειας (υπενθυμίζεται ότι αυτή η οικογένεια εισήχθη το 1999). Μπορούμε να δούμε ότι το μεγαλύτερο ολοκληρωμένο της σειράς αυτής διαθέτει 3.136 CLBs σε ένα πίνακα 56x56, με 7.168 flip flops σε συσκευασία που προσφέρει έως και 448 ακροδέκτες στο χρήστη. Μπορούμε να δούμε ότι σύμφωνα με τις εκτιμήσεις του κατασκευαστή στο ολοκληρωμένο αυτό μπορεί να μπει λογική που αντιστοιχεί σε 85.000 πύλες (να αντικαταστήσει δηλαδή περίπου 21.250 SSI !!!) ή να χρησιμοποιηθεί σα μνήμη των 72 KB. Βεβαίως αυτά τα νούμερα έχουν περίπου δεκαπενταπλασιαστεί στις τρέχουσες οικογένειες ολοκληρωμένων.

Device	Logic Cells	Max Logic Gates (No RAM)	Max. RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total CLBs	Number of Flip-Flops	Max. User I/O
XC4002XL	152	1,600	2,048	1,000 - 3,000	8 x 8	64	256	64
XC4003E	238	3,000	3,200	2,000 - 5,000	10 x 10	100	360	80
XC4005E/XL	466	5,000	6,272	3,000 - 9,000	14 x 14	196	616	112
XC4006E	608	6,000	8,192	4,000 - 12,000	16 x 16	256	768	128
XC4008E	770	8,000	10,368	6,000 - 15,000	18 x 18	324	936	144
XC4010E/XL	950	10,000	12,800	7,000 - 20,000	20 x 20	400	1,120	160
XC4013E/XL	1368	13,000	18,432	10,000 - 30,000	24 x 24	576	1,536	192
XC4020E/XL	1862	20,000	25,088	13,000 - 40,000	28 x 28	784	2,016	224
XC4025E	2432	25,000	32,768	15,000 - 45,000	32 x 32	1,024	2,560	256
XC4028EX/XL	2432	28,000	32,768	18,000 - 50,000	32 x 32	1,024	2,560	256
XC4036EX/XL	3078	36,000	41,472	22,000 - 65,000	36 x 36	1,296	3,168	288
XC4044XL	3800	44,000	51,200	27,000 - 80,000	40 x 40	1,600	3,840	320
XC4052XL	4598	52,000	61,952	33,000 - 100,000	44 x 44	1,936	4,576	352
XC4062XL	5472	62,000	73,728	40,000 - 130,000	48 x 48	2,304	5,376	384
XC4085XL	7448	85,000	100,352	55,000 - 180,000	56 x 56	3,136	7,168	448

\* Max values of Typical Gate Range include 20-30% of CLBs used as RAM.

Στη συνέχεια εξετάζουμε τα IOBs. Οι υποσχεδιασμοί αυτοί παρέχουν τη διασύνδεση του κυκλώματος που υλοποιείται στο FPGA με άλλα κυκλώματα και γενικότερα με το περιβάλλον. Κάθε IOB ελέγχει ένα ακροδέκτη του περιβλήματος του FPGA και μπορεί να προγραμματιστεί έτσι ώστε αυτός ο ακροδέκτης να είναι μόνο είσοδος ή μόνο έξοδος ή τέλος είσοδος / έξοδος. Η αρχιτεκτονική του IOB φαίνεται στο απλοποιημένο σχήμα παρακάτω.

Οπως βλέπουμε το σήμα που φτάνει στον ακροδέκτη (PAD) μπορεί να γίνει είσοδος στο FPGA ακολουθώντας τα μονοπάτια προς το  $I_1$  ή το  $I_2$ . Το σήμα στον ακροδέκτη μπορεί επίσης να καταχωρηθεί προιν αποτελέσει είσοδο στο FPGA μας, σε ένα στοιχείο που μπορεί να δράσει σα latch ή σαν flip flop. Το IOB μπορεί επίσης να προγραμματιστεί έτσι ώστε να συνεργάζεται με σήματα TTL ή CMOS. Στη περίπτωση εισόδου από τον καταχωρητή μπορούμε να επιλέξουμε τη πρόσθεση ή όχι κάποιας μικρής καθυστέρησης έτσι ώστε να μπορούμε να χρησιμοποιήσουμε και σήματα που δεν έχουν αρκετό χρόνο αποκατάστασης (hold time).



Στην περίπτωση χρήσης του IOB για σήματα εξόδου, βλέπουμε ότι αυτά μπορούν αν θέλουμε να αντιστραφούν και ή να περάσουν κατεύθειαν στον ακροδέκτη ή μέσω ενός καταχωρητή εξόδου. Το σήμα  $T$  σε κανονική ή αντεστραμμένη μορφή μπορεί να χρησιμοποιηθεί για τον έλεγχο του στοιχείου τριών καταστάσεων έτσι ώστε το σήμα εξόδου μας να μπορεί να είναι ένας από τους πολλούς οδηγούς μιας διαμοιραζόμενης γραμμής ή στη περίπτωση που θέλουμε ο ακροδέκτης μας να χρησιμοποιείται σαν είσοδος / έξοδος (πρακτικά αυτό μεταφράζεται ότι πάντα είναι είσοδος, αλλά έξοδος μόνο όταν το  $T$  είναι σε τιμή που αποφεύγει τη κατάσταση υψηλής εμπέδησης). Επίσης μπορούμε αν θέλουμε να χρησιμοποιήσουμε παθητικά στοιχεία για την ανύψωση ή τη πτώση του δυναμικού έτσι ώστε να υλοποιήσουμε καλωδιωμένη (wired logic) λογική συνδυάζοντας την έξοδό μας με άλλες.

Είναι προφανές ότι το IOB είναι αρκετά ευέλικτο για να καλύψει όλες τις ανάγκες του σχεδιαστή για είσοδο / έξοδο. Παρατηρείστε ότι αυτός το μόνο που χρειάζεται να κάνει είναι να αποθηκεύσει τις κατάλληλες τιμές στη μνήμη που ελέγχει τα σήματα ελέγχου των πολυπλεκτών που υπάρχουν στο IOB. Η μνήμη αυτή αναλαμβάνει να διαρθρώσει το κάθε IOB κατά τη φάση της διάρθρωσης.

---

Κλείνοντας θα πρέπει να πούμε ότι η παραπάνω αναφορά μένει σε ένα αρκετά υψηλό επίπεδο, καθώς κάθε ακροδέκτης του ολοκληρωμένου μπορεί να χρησιμοποιείται και για άλλους συκοπούς στις διάφορες φάσεις του ολοκληρωμένου. Για παράδειγμα κατά τη φάση διάρθρωσης του ολοκληρωμένου, κάποιοι ακροδέκτες χρησιμοποιούνται για την ανάγνωση της επιθυμητής διάρθρωσης από κάποια εξωτερική μνήμη.

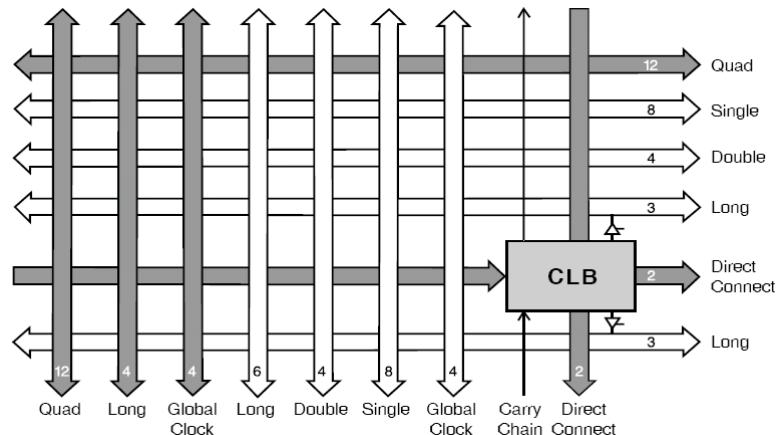
Οι διασυνδέσεις μεταξύ των σχεδιασμών είναι το τελευταίο σχεδιαστικό κομμάτι που θα μας απασχολήσει. Εσωτερικά στο FPGA υπάρχουν έτοιμα κομμάτια συνδέσεων τα οποία καταλήγουν σε στοιχεία που προγραμματίζονται για να πετύχουν τις επιθυμητές ενώσεις μεταξύ αυτών των κομματιών. Τα κομμάτια αυτά μαζί με τα προγραμματιζόμενα στοιχεία συνδέσεων (Programmable Switching Matrices – PSM) σχηματίζουν ένα κανονικό, ιεραρχικό πίνακα διασυνδέσεων.

Στη σειρά XC4000 της Xilinx υπάρχουν αρκετοί διαφορετικοί τύποι γραμμών διασύνδεσης:

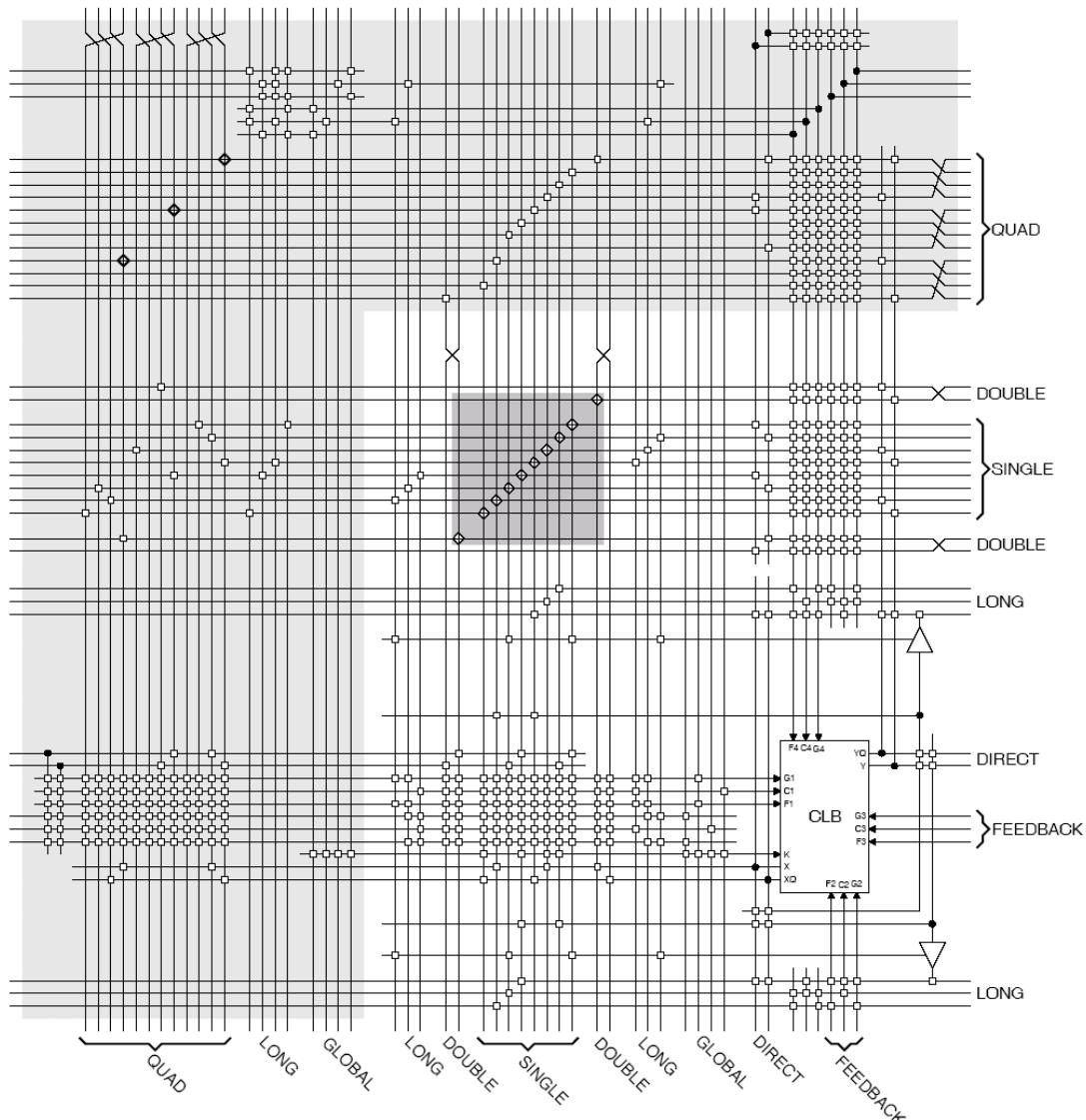
- ♦ Γραμμές διασύνδεσης σε κάθε στήλη και γραμμή του πίνακα των CLB.
- ♦ Γραμμές διασύνδεσης των IOB που ουσιαστικά δημιουργούν ένα δακτύλιο γύρω από τον πίνακα των CLB
- ♦ Συνολικές γραμμές διασύνδεσης, οι οποίες είναι εξειδικευμένες γραμμές που χρησιμοποιούνται για τη μετάδοση σημάτων ρολογιού με την ελάχιστη δυνατή παραμόρφωση. Υπάρχουν μερικές τέτοιες γραμμές διαθέσιμες στο σχεδιαστή για τη μετάδοση σημάτων που οδηγούν μεγάλο αριθμό εισόδων.

Οι γραμμές διασύνδεσης επίσης μπορούν να κατηγοριοποιηθούν ανάλογα με το μήκος των κομματιών σε απλές, διπλές, τετραπλές, οκταπλές (σε ορισμένα ολοκληρωμένα μόνο) και πολύ μακριές.

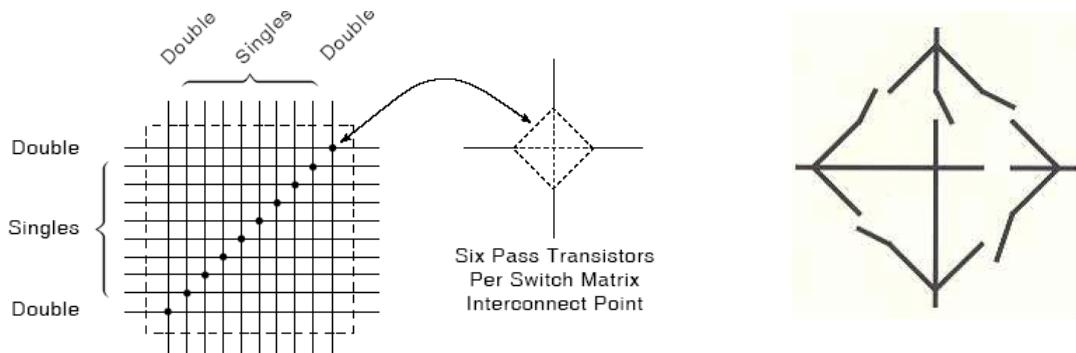
Το παρακάτω σχήμα είναι ένα μοντέλο των γραμμών διασύνδεσης που αντιστοιχούν σε κάθε γραμμή και στήλη ενός CLB. Οπως βλέπουμε Υπάρχει μια σαφής ιεραρχία όπου οι κοντές γραμμές είναι πολλές μιας και το CLB θα χρειαστεί να συνδεθεί με αρκετά από τα CLBs που υπάρχουν τριγύρω του, ενώ ο αριθμός των γραμμών ελαττώνεται καθώς αυξάνεται το μήκος τους, μιας και η πιθανότητα ένα CLB να χρειάζεται να συνδεθεί με κάποιο πολύ μακριά του είναι μικρή, υποθέτοντας φυσικά ότι έχουμε κάποια καλή τοποθέτηση της λογικής μας στο χώρο. Οι συιασμένες γραμμές της εικόνας υποδηλώνουν ότι οι γραμμές αυτές παρέχονται σε ορισμένα μόνο ολοκληρωμένα της σειράς που μπορεί με αυτόν το τρόπο να είναι πιο κατάλληλα για σχεδιασμούς που έχουν μεγάλες ανάγκες διασύνδεσης.



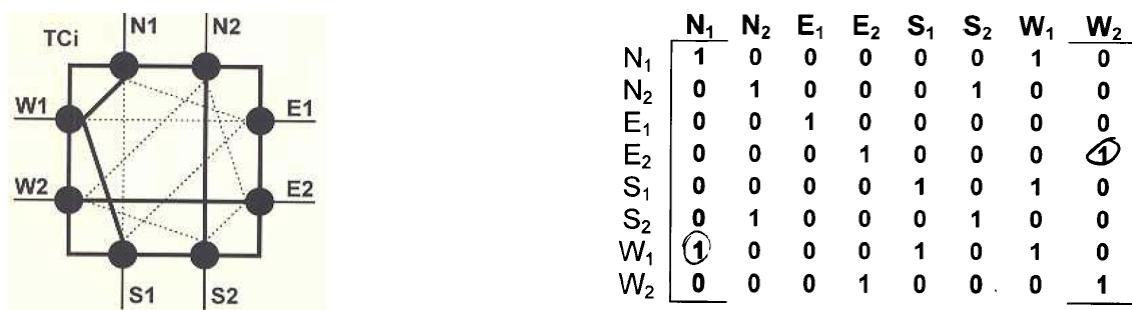
Το μοντέλο αυτό στο φυσικό επίπεδο έχει τη μορφή που δείχνει το παρακάτω σχήμα.



Όπως βλέπουμε οι είσοδοι / έξοδοι του CLB είναι κατανεμημένες και στις τέσσερις πλευρές του για μέγιστη ευελιξία και γενικότερα αυτό οδηγεί σε μία συμμετρική και κανονική δομή. Τα σήματα του CLB συνδέονται σε περισσότερες από μία απλές και διπλές γραμμές. Στο σχήμα με το σκούρο χρώμα σημειώνεται το PSM. Εκεί διασταυρώνονται οι οριζόντιες και κάθετες απλές και διπλές γραμμές. Κάθε PSM αποτελείται από προγραμματιζόμενα pass τρανζίστορα τα οποία χρησιμοποιούνται για να φτιάξουν τις διασυνδέσεις μεταξύ των κομματών. Το παρακάτω σχήμα μας δείχνει ένα μοντέλο του PSM.



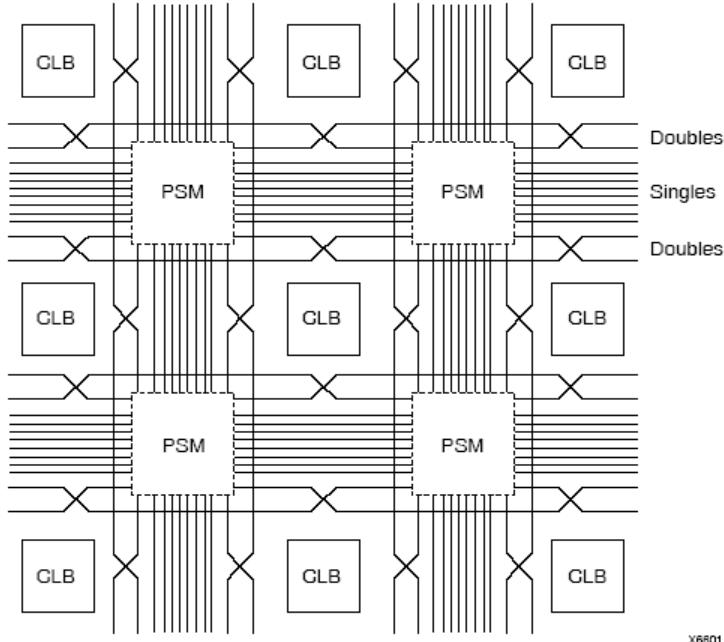
Το κάθε σημείο πιθανής διασύνδεσης υλοποιείται όπως βλέπουμε στο δεξιό σχήμα με έξι pass transistors, έτσι ώστε να είναι δυνατή η διάδοση ενός σήματος που έρχεται από μία κατεύθυνση προς οποιονδήποτε συνδυασμό των άλλων τριών κατευθύνσεων. Είναι προφανές ότι η επιθυμητή διασύνδεση στην ουσία μπορεί να οριστεί βάσει της τιμής στην πύλη κάθε pass τρανζίστορο. Το σύνολο αυτών των τιμών μπορεί προφανώς να οριστεί από το σχεδιαστή και να αποθηκευτεί σε μία μνήμη, η οποία καθορίζει την επιθυμητή διασύνδεση κατά τη φάση διάρθρωσης του FPGA. Υποθέστε για παράδειγμα ότι έχουμε σαν επιθυμητή διασύνδεση την ακόλουθη:



όπου το σήμα στο σημείο  $N_1$  συνδέεται με τα σημεία  $S_1$  και  $W_1$ , το σήμα στο σημείο  $W_2$  με το  $E_2$  και το σήμα στο σημείο  $N_2$  με το  $S_2$ . Αυτή η επιθυμητή διασύνδεση αντιστοιχεί στον πίνακα δεξιά στο σχήμα που δείχνει τις τιμές που πρέπει να πάρουν οι πύλες συγκεκριμένων pass τρανζίστορων. Προφανώς ο πίνακας είναι αυτός που τελικά

Θα χρησιμοποιήσει ο σχεδιαστής κατά τη φάση διάρθρωσης για να αποκαταστήσει την επιθυμητή διασύνδεση.

Η ιεραρχία των γραμμών επεκτείνεται και στον τρόπο που αυτές συνδέονται στα PSM (βλέπε παρακάτω εικόνα). Οι απλές γραμμές που έχουν στόχο να προσφέρουν τη μέγιστη ευελιξία μεταξύ μεταξύ γειτονικών CLBs οδηγούνται όλες σε κάθε PSM που υπάρχει γειτονικά σε ένα CLB. Κάθε pass transistor ωστόσο βάζει μια επιπλέον καθυστέρηση πάνω στα δύο κομμάτια απλών γραμμών που συνδέει. Ετσι οι γραμμές αυτές δεν είναι κατάλληλες για σύνδεση μακρινών CLB μιας και η καθυστέρηση της διασύνδεσης θα ήταν σημαντική.



Αντίθετα, οι διπλές γραμμές διασχίζουν μήκος δύο CLBs πριν οδηγηθούν σε ένα PSM. Ως αποτέλεσμα είναι πολύ περισσότερο κατάλληλες για μεσαίου μεγέθους διασυνδέσεις από ότι οι απλές γραμμές. Η ιεραρχία συνεχίζεται όμοια για τις γραμμές μεγαλύτερου μεγέθους με τις πολύ μακριές γραμμές να διασχίζουν όλο το μήκος ή το πλάτος του πίνακα.

Πριν κλείσουμε θα πρέπει να αναφέρουμε ότι αντίστοιχα είναι οργανωμένη και η διασύνδεση μεταξύ CLBs και IOBs. Πέρα από αυτές τις διασυνδέσεις υπάρχουν άμεσες διασυνδέσεις μεταξύ CLBs για το σκοπό της γρήγορης υλοποίησης αριθμητικών συναρτήσεων όπως αναφέρθηκαν πιο πάνω. Επίσης υπάρχουν σε κάθε τεταρτημόριο του ολοκληρωμένου δύο γραμμές που περνούν από όλα τα CLBs που βρίσκονται στο τεταρτημόριο. Οι γραμμές αυτές μπορούν να χρησιμοποιηθούν για τη διαμοίραση ρολογιών με πολύ μικρή παραμόρφωση ή σημάτων αρχικοποίησης.

Από τη μικρή αυτή περιήγησή μας στο κόσμο των FPGAs θα πρέπει να έχει γίνει κατανοητό ότι η εμφώλευση ενός σχεδιασμού σε αυτά έγκειται στην αποθήκευση

---

σε μία μνήμη (στο εξής θα την αποκαλούμε μνήμη διάρθρωσης – configuration memory), των τιμών που θέλουμε να περιέχουν τα LUT, των τιμών για τα σήματα ελέγχου των πολυπλεκτών στα CLBs και τα IOBs καθώς και των τιμών ελέγχου των pass τρανζίστορ στα PSMs. Άρα ο σχεδιαστής το μόνο που χρειάζεται να παράγει σε αυτή τη περίπτωση είναι το αρχείο για τον προγραμματισμό αυτής της μνήμης (configuration / programming file). Τις διαδικασίες που χρειάζεται να ολοκληρώσει ο σχεδιαστής για την εξαγωγή αυτού του αρχείου τις εξετάζουμε σε κατοπινό κεφάλαιο. Σκοπός των παρακάτω είναι να εξηγήσουμε το πως συνδέεται ένα FPGA με την μνήμη διάρθρωσης.

Τα σύγχρονα FPGAs δίνουν στους σχεδιαστές πολλούς τρόπους για τη διάρθρωσή τους (configuration modes). Πέρα από τη διάρθρωση που απαιτείται πριν το FPGA τεθεί σε λειτουργία μαζί με το υπόλοιπο σύστημα, οι σχεδιαστές έχουν την δυνατότητα να ζητούν την αναδιάρθρωση του FPGA κατά τη διάρκεια λειτουργίας του συστήματος (on-line reconfiguration). Τέτοιες λύσεις προσφέρουν μέγιστη ευελιξία στο τελικό προϊόν καθώς επιτρέπουν την παρέμβαση του σχεδιαστή σε αυτό, ακόμη κι όταν έχει περάσει στα χέρια του τελικού χρήστη μέσω απομακρυσμένων τρόπων αναδιάρθρωσης, αλλά και την εμφάλευση κάθε φορά στο FPGA μόνο εκείνων των σχεδιαστικών κομματιών που πραγματικά χρειάζονται.

Οι τρόποι που συνήθως προσφέρονται για τη διάρθρωση του FPGA καθορίζουν το τρόπο που αυτό θα διασυνδεθεί με τη πηγή των δεδομένων διάρθρωσης και τη τυχόν συνοδευτική λογική ελέγχου. Για τη σειρά XC4000 πιθανοί τρόποι είναι :

- ◆ Σειριακή φόρτωση των δεδομένων.
- ◆ Παράλληλη φόρτωση των δεδομένων και
- ◆ Φόρτωση μέσω της αλυσίδας ελέγχου (Boundary scan chain)

Κάθε τρόπος επιπλέον κατηγοριοποιείται από το αν το FPGA θα δρα ως μια συσκευή κύριος ή συλλάβος, από το αν χρειάζεται εξωτερική λογική ελέγχου της διαδικασίας διάρθρωσης, από το αν έχουμε μόνο ένα ή περισσότερα FPGAs που θα προγραμματιστούν ξλπ. Οπως αναφέραμε προηγούμενα, ορισμένοι από τους ακροδέκτες του FPGA κατά τη διαδικασία της διάρθρωσης καλούνται να παίζουν ξεχωριστό ρόλο. Η σειρά XC4000 χρησιμοποιεί τους ακροδέκτες M<sub>2</sub>M<sub>1</sub>M<sub>0</sub> για την επιλογή του τρόπου διάρθρωσης σύμφωνα με τον εξής πίνακα αληθείας :

Mode	M2	M1	M0	CCLK	Data
Master Serial	0	0	0	output	Bit-Serial
Slave Serial	1	1	1	input	Bit-Serial
Master Parallel Up	1	0	0	output	Byte-Wide, increment from 00000
Master Parallel Down	1	1	0	output	Byte-Wide, decrement from 3FFFF
Peripheral Synchronous*	0	1	1	input	Byte-Wide
Peripheral Asynchronous	1	0	1	output	Byte-Wide
Reserved	0	1	0	—	—
Reserved	0	0	1	—	—

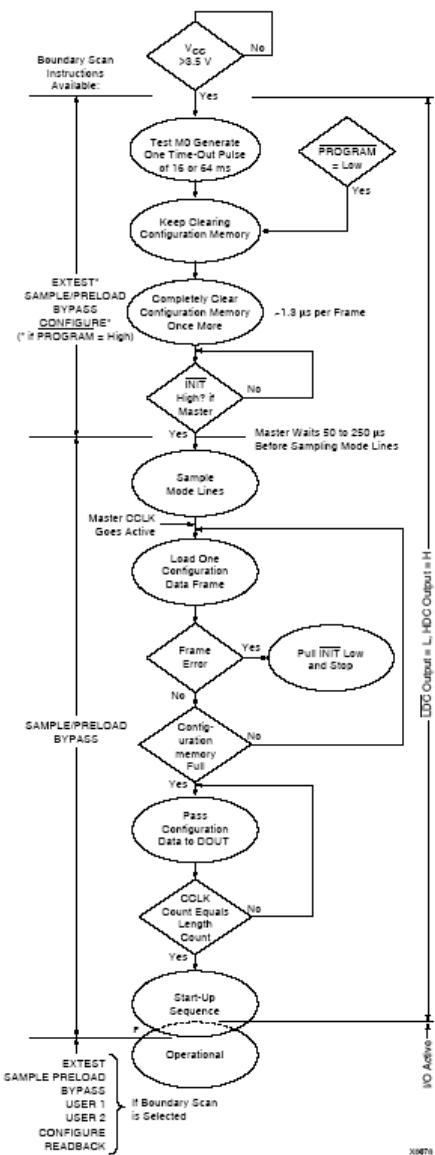
\* Can be considered byte-wide Slave Parallel

Στις τρεις περιπτώσεις που το FPGA καλείται να δράσει σα συσκευή κύριος, στην ουσία είναι αυτή η συσκευή που θα πρέπει να γεννήσει το ρολόι της διάρθρωσης (configuration clock – CCLK) με το οποίο θα λειτουργήσει η όλη διαδικασία, αλλά και τις διευθύνσεις για τη προσπέλαση της μνήμης των δεδομένων διάρθρωσης.

Οταν φυσικά μιλάμε για παράλληλη φόρτωση δεδομένων. Στην περίπτωση σειριακών δεδομένων συνήθως για την αποθήκευση των δεδομένων διάρθρωσης χρησιμοποιείται μια σειριακή EEPROM, η οποία είναι ένα ολοκληρωμένο το οποίο δε χρειάζεται εξωτερική προσπέλαση διευθύνσεων. Αντίθετα διαθέτει εσωτερικούς μετρητές που μηδενίζονται με την αρχή παροχής της τροφοδοσίας και με κάθε παλμό του ρολογιού (CCLK) αυξάνονται κατά ένα έτσι ώστε να δώσουν το επόμενο δυαδικό ψηφίο στην έξοδό τους.— Γι' αυτό άλλωστε και στον πίνακα το CCLK εμφανίζεται σαν έξοδος, κάθε φορά που επιλέγεται ένας τρόπος διάρθρωσης με το FPGA σα συσκευή κύριος. Οπως μπορούμε να δούμε στην περίπτωση παράλληλων δεδομένων η ανάγνωση της μνήμης δεδομένων διάρθρωσης μπορεί να γίνει από τις χαμηλότερες προς τις υψηλότερες διευθύνσεις ή ανάποδα, ανάλογα με την επιθυμία του σχεδιαστή. Άλλοι ακροδέκτες οι οποίοι χρησιμοποιούνται κατά τη διαδικασία της διάρθρωσης είναι :

- ◆ Program : Σήμα αρνητικής λογικής με το οποίο ξεκινά η διαδικασία διάρθρωσης.
- ◆ Din : Το σήμα σειριακών δεδομένων εισόδου.
- ◆ Dout : Το σήμα σειριακών δεδομένων εξόδου. Χρησιμοποιείται όταν έχουμε πολλά FPGAs και θέλουμε να χρησιμοποιήσουμε μόνο ένα ολοκληρωμένο μνήμης για την αποθήκευση των δεδομένων διάρθρωσής τους, οπότε και τα συνδέουμε με μια μορφή αλυσίδας (Daisy Chain). Η γραμμή Dout χρησιμεύει για το πέρασμα δεδομένων στο επόμενο FPGA της αλυσίδας.
- ◆ Init : Σήμα εισόδου / εξόδου ανοικτής καταβόθρας (open drain). Σκοπός του να δείχνει τη σταθεροποίηση της τροφοδοσίας και την ύπαρξη λάθους στη διαδικασία διάρθρωσης.
- ◆ Done. Σήμα εισόδου / εξόδου ανοικτής καταβόθρας. Δείχνει την ολοκλήρωση της διαδικασίας διάρθρωσης.

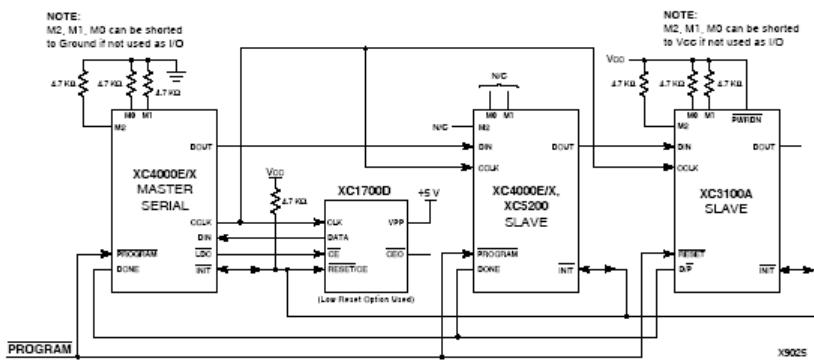
Το διάγραμμα ροής που ακολουθείται στη διαδικασία διάρροωσης είναι το ακόλουθο :



Οπως βλέπουμε τα πάντα ξεκινάνε με την οδήγηση του σήματος PROGRAM στο 0. Αυτό το γεγονός ουσία ξεκινά μια επαναλαμβανόμενη διαδικασία καθαρισμού των δεδομένων διάρροωσης του FPGA, μέχρι το σήμα να πάει στο ένα, οπότε μετά από έναν ακόμη καθαρισμό της μνήμης διάρροωσης εσωτερικά του FPGA, αλλάζει η πολικότητα του INIT ώστε να περάσουμε στη φάση ανάγνωσης δεδομένων. Ακολουθεί η αναγνώριση του τρόπου διάρροωσης μέσω της δειγματοληψίας των σημάτων M, και η ενεργοποίηση του CCLK στη περίπτωση συσκευής κυρίου. Ακολουθούν κύριοι ανάγνωσης πακέτων δεδομένων. Στο τέλος κάθε πακέτου γίνεται έλεγχος του CRC του για να αποφευχθούν λάθη. Οταν μία συσκευή διαβάσει όσα δεδομένα της αντιστοιχούν τότε είτε έχουμε τελειώσει τη διαδικασία ανάγνωσης ή

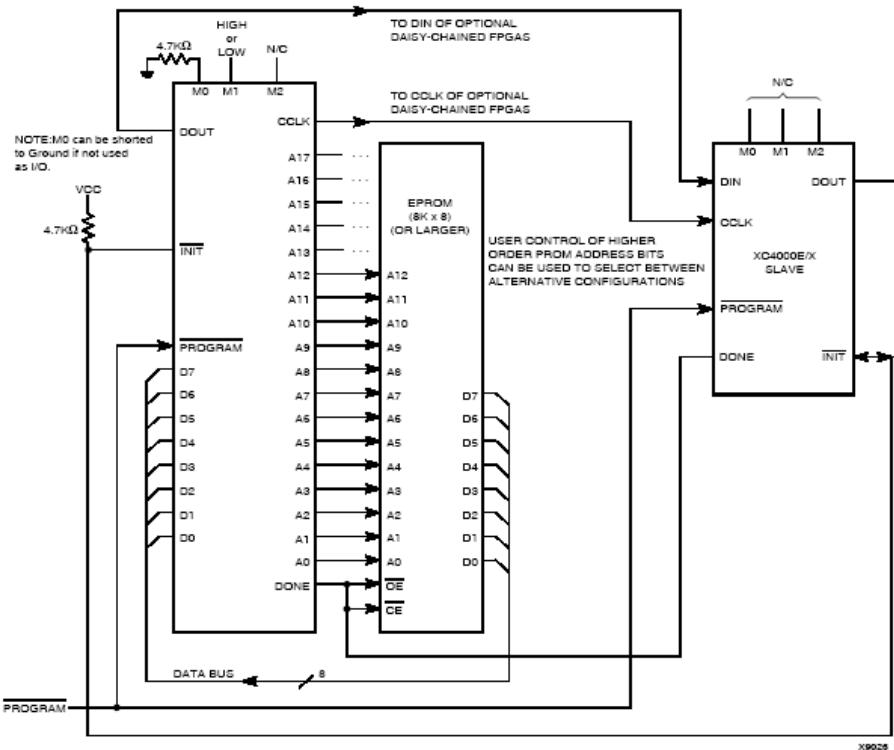
υπάρχουν και άλλες συσκευές πίσω από αυτήν στην αλυσίδα. Συνεπώς η συσκευή δείχνει ότι είναι έτοιμη μέσω του δικού της σήματος DONE αλλά αν υπάρχουν και άλλες στην αλυσίδα, η γραμμή DONE θα παραμείνει στο 0 και η συσκευή θα πρέπει να συνεχίσει να γεννάει CCLK αν είναι κύριος και θα πρέπει να αρχίσει να περνάει τα δεδομένα εισόδου στη γραμμή δεδομένων εξόδου. Οταν προγραμματιστεί σωστά και η τελευταία συσκευή της αλυσίδας, τότε πλέον περνάμε στη φάση αρχικοποίησης, όπου οι ακροδέκτες παίρνουν τη τελική τους κατάσταση σύμφωνα με τις επιθυμίες του σχεδιαστή και ενεργοποιείται το σήμα GSR (Global Set Reset) για να φέρει όλους τους καταχωρητές του ολοκληρωμένου στην αρχική τους κατάσταση. Πλέον το FPGA είναι έτοιμο για τη λειτουργία του.

Παρακάτω εξετάζουμε τα σχηματικά μερικών από τους πιο πάνω τρόπους για τον προγραμματισμό των FPGAs. Στο πιο κάτω σχήμα βλέπουμε το σειριακό τρόπο προγραμματισμού, όπου το αριστερότερο FPGA είναι μια συσκευή κύριος, ενώ το FPGA και το CPLD που φαίνονται δεξιά χρησιμοποιούνται ως σκλάβοι. Για το προγραμματισμό και των τριών χρησιμοποιείται μία και μόνο σειριακή E<sup>2</sup>PROM. Αξίζει να παρατηρήσετε την αλυσίδα που σχηματίζουν οι γραμμές DIN και DOUT των ολοκληρωμένων προγραμματιζόμενης λογικής. Η έξοδος της E<sup>2</sup>PROM οδηγεί τη γραμμή δεδομένων εισόδου της συσκευής κύριος. Η γραμμή δεδομένων εξόδου αυτής οδηγεί το σκλάβο FPGA που με τη σειρά του οδηγεί το σκλάβο CPLD. Επίσης αξίζει να προσέξετε ότι το ρολόι γεννιέται από το ολοκληρωμένο κύριο και διαδίδεται σε όλα τα υπόλοιπα ολοκληρωμένα, ενώ οι γραμμές που είναι ανοικτής ικανοτήτας διευκολύνουν εξαιρετικά τη διασύνδεση μεταξύ τους για τη δημιουργία καλωδιωμένης λογικής.



Οι απαραίτητες διασυνδέσεις για έναν παράλληλο τρόπο προγραμματισμού φαίνονται στο αριστερό τμήμα της παρακάτω εικόνας. Το FPGA στα αριστερά, όντας προγραμματισμένο σα συσκευή κύριος, γεννά τις απαραίτητες διευθύνσεις για τη προσπέλαση της E<sup>2</sup>PROM. Προσέξτε ότι μπορεί στην ίδια E<sup>2</sup>PROM να αποθηκεύσουμε περισσότερες της μίας πιθανές διαρθρώσεις για το FPGA μας και να

επιλέγουμε αυτή που επιθυμούμε χρησιμοποιώντας ψηφία διεύθυνσης υψηλής σημαντικότητας. Στο ίδιο σχήμα επίσης φαίνεται ότι μπορούμε να έχουμε παράλληλα και σειριακή διαμόρφωση άλλων σκλάβων συσκευών. Αφού προγραμματιστεί η συσκευή αύριος, συνεχίζει να παραλαμβάνει παράλληλα δεδομένα από τη μνήμη, τα σειριοποιεί και τα δίνει στο επόμενο στην αλυσίδα προγραμματιζόμενο ολοκληρωμένο μέσω της σειριακής εξόδου δεδομένων. Προφανώς παράγει και το απαραίτητο ρολόι διάρθρωσης γι' αυτές.



Θα κλείσουμε την αναφορά μας στα FPGAs κάνοντας μια γρήγορη αναφορά στο τι περιμένουμε να δούμε στο εγγύς μέλλον από πλευράς ολοκληρωμένων προγραμματιζόμενης λογικής. Θυμηθείτε ότι τα ολοκληρωμένα αυτά πρωτεισήχθησαν με σκοπό την αντικατάσταση μερικών SSI και τη προστασία από την αντιγραφή. Με βάση τον παρακάτω πίνακα της πλέον πρόσφατης οικογένειας FPGAs από τον κατασκευαστή Xilinx, βλέπουμε ότι έχουμε πλέον απομακρυθεί πάρα πολύ από αυτόν τον πρωταρχικό στόχο μας. Το μεγαλύτερο ολοκληρωμένο αυτής της σειράς διαθέτει 5.000.000 ισοδύναμες πύλες, 104 διαφορετικούς πολλαπλασιαστές των 18 δυαδικών ψηφίων ο καθένας, υποστηρίζει 23 διαφορετικούς τύπους δυναμικών σαν είσοδο/έξοδο και παρέχει 784 ακροδέκτες στο χρήστη του!

Device	XC 3S50	XC 3S200	XC 3S400	XC 3S1000	XC 3S1500	XC 3S2000	XC 3S4000	XC 3S5000
System Gates	50K	200K	400K	1000K	1500K	2000K	4000K	5000K
Logic Cells	1,728	4,320	8,064	17,280	29,952	46,080	62,208	74,880
18x18 Multipliers	4	12	16	24	32	40	96	104
Block RAM Bits	72K	216K	288K	432K	576K	720K	1,728K	1,872K
Distributed RAM Bits	12K	30K	56K	120K	208K	320K	432K	520K
DCMs	2	4	4	4	4	4	4	4
I/O Standards	23	23	23	23	23	23	23	23
Max Differential I/O Pairs	56	76	116	175	221	270	312	344
Max Single Ended I/O	124	173	264	391	487	565	712	784

Προφανώς το ολοκληρωμένο αυτό δίνει στο σχεδιαστή VLSI δυνατότητες, μιας και η κατασκευή του σε τεχνολογία 0.18 μμ επιτρέπει την επίτευξη ταχυτήτων της τάξης του GHz. Οπως λοιπόν φαίνεται μπορούμε πλέον να ενσωματώσουμε έναν ολόκληρο σχεδιασμό μέσα σε ένα ολοκληρωμένο προγραμματιζόμενης λογικής. Συνεπώς οι μελλοντικές εξελίξεις δεν μπορεί απλά να περιοριστούν στην παροχή ολοένα και περισσότερων λογικών μεγαυττάρων. Τα μελλοντικά ολοκληρωμένα προγραμματιζόμενης λογικής διαφαίνεται να ρίχνουν βάρος σε τρεις διαφορετικές κατευθύνσεις :

1. Στην ενσωμάτωση στο ίδιο ολοκληρωμένο μαζί με τη προγραμματιζόμενη λογική, ενός τουλάχιστον επεξεργαστή και στατικής μνήμης. Σκοπός αυτής της κίνησης, είναι η μείωση του χρόνου επικοινωνίας μεταξύ των πλέον συνηθισμένων κομματιών που απαντάμε σε ένα σχεδιασμό.
2. Στην ενσωμάτωση αναλογικών προγραμματιζόμενων στοιχείων, όπως τελεστικοί ενισχυτές, μεταβλητές αντιστάσεις ή πυκνωτές. Συνήθως σε ένα σχεδιασμό συνυπάρχουν ψηφιακά και αναλογικά στοιχεία, οπότε η παροχή τέτοιων προγραμματιζόμενων στοιχείων επιτρέπει την πραγματοποίηση του σχεδιασμού σε ένα ολοκληρωμένο.
3. Στην ενσωμάτωση σχεδιαστικών πυρήνων (intellectual properties – cores) που θα επιτρέψουν τη πραγματοποίηση συναρτήσεων που είναι δύσκολο ή εξαιρετικά δαπανηρό (σε χρόνο) να επιτευχθούν με τη χρήση προγραμματιζόμενων blocks. Μια ιδιαίτερη κατηγορία θα πρέπει να θεωρούνται cores που επιτρέπουν την πολύ γρήγορη είσοδο / έξοδο δεδομένων προς και από το ολοκληρωμένο, π.χ. optical transceiver cores.



---

## Κεφάλαιο 4

# Σχεδιαστική Ροή

Ενα από τα σημαντικότερα κριτήρια για την επιτυχία κάποιου συστήματος στην σημερινή ανταγωνιστική αγορά είναι οι επιδόσεις του. Για την επίτευξη υψηλών επιδόσεων, όλο και περισσότερες λειτουργίες που σε συστήματα περασμένων γενεών υλοποιούνταν από λογισμικό μεταφέρονται στο υλικό του συστήματος. Αμεση επίπτωση αυτού είναι το υλικό των συστημάτων να γίνεται ολοένα πιο πολύπλοκο.

Η διαρκώς αυξανόμενη πολυπλοκότητα έχει σαν συνέπειες :

- ♦ Να αυξάνεται σημαντικά ο χρόνος που απαιτείται για τον σχεδιασμό του συστήματος.
- ♦ Να γίνεται δυσκολότερος ο έλεγχος του σχεδιασμού, τόσο από λογικής δηλαδή αν πράγματι ο σχεδιασμός υλοποιεί όλες τις απαιτούμενες συναρτήσεις ακόμη και σε ακραίες καταστάσεις) όσο και από χρονικής πλευράς (δηλαδή αν ο σχεδιασμός ακόμη και κάτω από τις χειρότερες καταστάσεις ανταποκρίνεται μέσα στα πλαίσια της μέγιστης ανεκτής καθυστέρησης διάδοσης).

- 
- ♦ Να γίνεται πιο πολύπλοκη η επικοινωνία μεταξύ των διαφορετικών ανθρώπων που ασχολούνται με τον σχεδιασμό υποσυστημάτων που τελικά συνδεόμενα θα αποτελέσουν το συνολικό σύστημα.

Τα παραπάνω εξελίσσονται σε ανασταλτικούς παράγοντες για την επιτυχία του συστήματος αν επιπλέον λάβουμε υπόψη μας ότι :

- ♦ Ο μέσος χρόνος ανταγωνιστικής παρουσίας (πρακτικά και ο μέσος χρόνος ζωής) ενός συστήματος συρρικνώνεται διαρκώς. Παίρνοντας σαν παράδειγμα τους επεξεργαστές για προσωπικούς υπολογιστές, μπορούμε να δούμε ότι σχεδόν κάθε εξάμηνο προωθείται και ένας νανούργιος σχεδιασμός. Συνεπώς, όσο μικρότερος είναι ο χρόνος από την σύλληψη ή την δημοσίευση μιας ιδέας μέχρι την εμπορική παραγωγή (time-to-market), τόσο μεγαλύτερο θα είναι και το μερίδιο της αγοράς που θα διεκδικεί το σύστημα που σχεδιάσαμε και τόσο μεγαλύτερος ο μέσος χρόνος ζωής του.
- ♦ Σχεδιασμοί οι οποίοι δεν έχουν ελεγχθεί πλήρως για την ορθή λογική και χρονική λειτουργία τους, δεν έχουν κανένα περιθώριο βιωσιμότητας. Σαν παράδειγμα, αναφέρουμε τον σχεδιασμό του μαθηματικού συνεπεξεργαστή 80387 της Intel. Δύο περίπου χρόνια μετά την κυκλοφορία του και αφού ανακαλύφθηκε λογικό λάθος στον σχεδιασμό του, η εταιρεία αναγκάστηκε να αντικαταστήσει όλα τα ολοκληρωμένα που είχαν κυκλοφορήσει και να αποζημιώσει όλους όσους αποδεδειγμένα υπέστησαν φθορές από την χρήση του. Σήμερα πλέον, λόγω της διεθνοποίησης της αγοράς, αντίστοιχο λάθος θα ήταν μοιραίο. Σκεφτείτε για παράδειγμα ότι το εν λόγω ολοκληρωμένο θα μπορούσε να αποτελεί υποσύστημα κάποιου διορυφόρου ή ενός χειρουργικού εργαλείου. Ο προϋπολογισμός ενός διορυφόρου είναι από μόνος του σημαντικά μεγαλύτερος από τους προϋπολογισμούς των περισσοτέρων εταιρειών σχεδιασμού συστημάτων.
- ♦ Τα σημερινά συστήματα αποτελούνται πλέον από τόσο πολλά αλλά και τόσο ειδικευμένα υποσυστήματα που τουλάχιστον για το κάθε ένα από αυτά απαιτείται και η ενασχόληση ενός εξειδικευμένου σχεδιαστή. Όλοι αυτοί θα πρέπει να μπορούν να συνεργάζονται αρμονικά έχοντας σαν βάση κάποιο κοινό υπόβαθρο. Σκεφτείτε για παράδειγμα τον σχεδιασμό ενός συστήματος ασύρματης μικροκυματικής μετάδοσης εικόνας, το οποίο θα μπορεί να χρησιμοποιηθεί από χρήστες προσωπικών υπολογιστών. Στην περίπτωση αυτή, λόγω της πολυπλοκότητας και της εξειδίκευσης των υποσυστημάτων του συνολικού σχεδιασμού, θα πρέπει να εμπλακούν αρκετά άτομα. Για το παράδειγμά μας θα απαιτούνταν τουλάχιστον : α) κάποιος για την σχεδίαση του υποσυστήματος επικοινωνίας με προσωπικό υπολογιστή (για όλους τους πιθανούς διαύλους), β)

---

κάποιος για την σχεδίαση του υποσυστήματος διαχείρισης πληροφορίας εικόνας (π.χ. συμπίεση βάσει των καθιερωμένων αλγορίθμων) και γ) κάποιος για να σχεδιάσει το σύστημα μικροκυματικής ασύρματης μετάδοσης πληροφοριών.

Μοναδική διέξοδο στα παραπάνω προβλήματα αποτελεί ο σχεδιασμός συστημάτων με χρήση υπολογιστών και ειδικών εφαρμογών ανεπτυγμένων για το σκοπό αυτό (Electronic Computer Aided Design, E-CAD). Παρατηρείστε ότι έχει αναπτυχθεί μια αναδρομική σχέση: χρησιμοποιώντας τα συστήματα του χθες, σχεδιάσαμε τα συστήματα του σήμερα, χρησιμοποιώντας τα συστήματα του σήμερα σχεδιάζουμε αυτά του αύριο κοκ. Σημειώνεται ότι η χρήση υπολογιστών και E-CAD εργαλείων σε καμία περίπτωση δεν υποκαθιστά τον ανθρώπινο παράγοντα ούτε κάνει έναν σχεδιαστή καλύτερο· δεν είναι άλλωστε αυτός ο στόχος της. Αυτά που μπορούμε να επιτύχουμε με τη χρήση υπολογιστών και E-CAD εργαλείων είναι :

1. Να έχουμε ανά πάσα στιγμή μια τυπική αναπαράσταση (formal representation) του σχεδιασμού μας, κάτι που μειώνει την πολυπλοκότητα ενός ιδεατού σχεδιασμού, με τον ίδιο τρόπο που ένα διάγραμμα ροής μειώνει την πολυπλοκότητα ενός αλγορίθμου εκφρασμένου με λόγια ή ένα διάγραμμα οντοτήτων – σχέσεων μειώνει την πολυπλοκότητα περιγραφής μιας βάσης δεδομένων.
2. Να μπορούμε να περιγράψουμε τον σχεδιασμό μας σε διάφορα επίπεδα λεπτομέρειας ή αφαίρεσης (abstraction levels), με σκοπό τον εντοπισμό λογικών λαθών όσο πιο νωρίς μέσα στον σχεδιαστικό κύκλο (design cycle). Για παράδειγμα στο σύστημα ασύρματης μικροκυματικής μετάδοσης εικόνας, θα μπορούσαμε να αντικαταστήσουμε το σχεδιασμό του πομπού – δέκτη μικροκυμάτων και αυτόν του προσαρμογέα με τον προσωπικό υπολογιστή με μαύρα κουτιά που απλά παράγουν ή καταναλώνουν δεδομένα με σκοπό τον εντοπισμό λαθών στο υποσύστημα συμπίεσης - αποσυμπίεσης εικόνας.
3. Να μπορούμε να καθορίσουμε με ακρίβεια τις λειτουργίες που θα πρέπει να επιτελεί κάθε υποσύστημα του συνολικού σχεδιασμού αρκετά νωρίς μέσα στο σχεδιαστικό κύκλο, έτσι ώστε οι διάφοροι σχεδιαστές που εργάζονται σε διαφορετικά υποσυστήματα να είναι απεξαρτημένοι ο ένας από την πρόσθιο και τις ιδιαιτερότητες του σχεδιασμού του άλλου. Η διαδικασία αυτή είναι αντίστοιχη της συγγραφής υποπρογραμμάτων (υπορουτινών) από διάφορους προγραμματιστές στον σχεδιασμό συστημάτων λογισμικού.
4. Να σχεδιάζουμε χωρίς να ενδιαφερόμαστε για την τεχνολογία υλοποίησης παρά μόνο στα τελικά στάδια πρωτότυποίησης του σχεδιασμού μας. Η διαδικασία αυτή είναι αντίστοιχη της συγγραφής ενός προγράμματος σε υψηλή γλώσσα

---

προγραμματισμού κατά την οποία δεν μας ενδιαφέρει η πραγματική αρχιτεκτονική ή οποία τελικά θα εκτελέσει το πρόγραμμά μας.

5. Να μπορούμε να δοκιμάζουμε διάφορες τεχνολογίες υλοποίησης του σχεδιασμού μας, με σκοπό να καταλήξουμε σε αυτήν που θα ικανοποιεί τις λογικές και χρονικές απαιτήσεις που τέθηκαν με το μικρότερο δυνατό κόστος, την ελάχιστη κατανάλωση ισχύος και την μεγαλύτερη αξιοπιστία (reliability).
6. Να μπορούμε να παράγουμε με αυτόματο τρόπο σημαντικό μέρος από την τεκμηρίωση του σχεδιασμού μας σε διαφορετικές μορφές, ανάλογα με το υπόβαθρο και τις ανάγκες του τελικού αναγνώστη.
7. Να καταλήγουμε σε σχεδιασμούς άμεσα επαναχρησιμοποιούμενους. Στο παράδειγμα του συστήματος ασύρματης μικροκυματικής μετάδοσης εικόνας, τα υποσυστήματα ασύρματης μετάδοσης και προσαρμογής στον προσωπικό υπολογιστή θα μπορούσαν, αυτούσια ή με ελάχιστες αλλαγές, να επαναχρησιμοποιηθούν και για ένα σύστημα ασύρματης μικροκυματικής μετάδοσης ήχου, το οποίο θα μπορεί να χρησιμοποιηθεί από χρήστες προσωπικών υπολογιστών.

Από τα παραπάνω είναι προφανές ότι ένα E-CAD πρόγραμμα γίνεται ένα πολύ χρήσιμο εργαλείο στα χέρια ενός ικανού σχεδιαστή. Στην ευρύτερη αγορά πακέτων λογισμικού υπάρχει πλήθωρα E-CAD προγραμμάτων που απευθύνονται σε διαφορετικό αγοραστικό κοινό (από αυτούς που θέλουν να υλοποιήσουν μικροευρεσιτεχνίες μέχρι εταιρείες παγκόσμιας εμβέλειας) και με αντίστοιχα κόστη (από μερικές χιλιάδες δραχμές μέχρι εκατοντάδες εκατομμύρια). Παρακάτω αναλύουμε τις διαδικασίες που θα πρέπει να περάσει ο χρήστης αυτών των προγραμμάτων, προκειμένου από μια αρχική πλήρως αφαιρετική περιγραφή ενός στοχευόμενου συστήματος να καταλήξει σε ένα πλήρως λειτουργικό πρωτότυπο του σχεδιασμού του. Οι διαδικασίες αυτές φτιάχνουν τη ροή του σχεδιασμού (design flow).

Στο σύγγραμμα αυτό θεωρούμε αυτονόητη την ύπαρξη ενός στοιχειώδους λογικού διαγράμματος του στοχευόμενου σχεδιασμού. Στη πράξη αυτό σπάνια συμβαίνει. Συνήθως, ο σχεδιασμός ενός συστήματος ξεκινά από μια περιγραφή του σε ανθρώπινη γλώσσα. Η δυσκολία της μετάφρασης από την αρχική περιγραφή σε ένα στοιχειώδες λογικό διάγραμμα πουίλει ανάλογα με το πόσο κοντά ή μακριά βρίσκεται ο περιγράφων στην επιστήμη μας. Θα πρέπει να αναφερθεί ότι η αυτοματοποίηση αυτής της μετάφρασης αποτελεί στις μέρες μας αντικείμενο συνεχιζόμενης έρευνας (High Level Synthesis).

---

Συνήθως η ροή του σχεδιασμού χωρίζεται σε δύο φάσεις :

1. Την πρώτη φάση (Front – end). Κατά την φάση αυτή εκτελούνται η εισαγωγή του σχεδιασμού στον υπολογιστή σε διάφορα επίπεδα αφαίρεσης και ο έλεγχος λογικής και χρονικής λειτουργίας του συστήματος. Η φάση αυτή είναι συνήθως ανεξάρτητη από την τελική υλοποίηση του κυκλώματος αν και για την χρονική επαλήθευση του σχεδιασμού χρησιμοποιούνται χρονικά μοντέλα για το υλικό νάποιας συγκεκριμένης τεχνολογίας.
2. Την δεύτερη φάση (Back – end). Η φάση αυτή περιέχει τις διαδικασίες αντιστοίχησης του σχεδιασμού σε νάποια συγκεκριμένη τεχνολογία, την φυσική υλοποίησή του, την εξαγωγή πραγματικών χρονικών παραμέτρων και την χρονική επαλήθευση του υλοποιημένου σχεδιασμού.

Οι δύο αυτές φάσεις αναπτύσσονται παρακάτω με λεπτομέρεια. Για την σαφήνεια της ανάλυσης, εισάγουμε ένα παράδειγμα συστήματος το οποίο θα χρησιμοποιούμε στη συνέχεια ως βάση. Εστω ότι σκοπός μας είναι να σχεδιάσουμε ένα υποσύστημα ενός μικροεπεξεργαστή, το οποίο θα μπορεί να προσθέτει δύο δεδομένα των 8 δυαδικών ψηφίων σε ένα κύκλο μηχανής. Τα δεδομένα προέρχονται από την αρτηρία δεδομένων του μικροεπεξεργαστή των 16 δυαδικών ψηφίων. Υποθέτουμε ότι τα αποτελέσματα τοποθετούνται και πάλι στην αρτηρία δεδομένων και ότι η ίδια κύκλος μηχανής ορίζεται σαν το χρονικό διάστημα μεταξύ δύο ανερχόμενων ακμών (rising edges) του ρολογιού του μικροεπεξεργαστή.

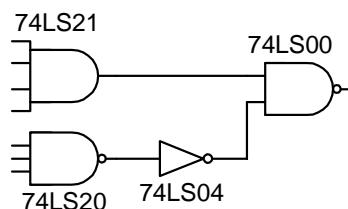
Οπως αναφέρθηκε και προηγούμενα η πρώτη φάση περιλαμβάνει την εισαγωγή του σχεδιασμού στον υπολογιστή σε διάφορα επίπεδα αφαίρεσης και τον έλεγχο λογικής και χρονικής λειτουργίας του συστήματος. Το πρώτο βήμα κατά την χρήση ενός E-CAD εργαλείου είναι η έκφραση του σχεδιασμού σε αναπαράσταση κατανοητή και εύχρηστη από τον υπολογιστή. Η διαδικασία της έκφρασης στα διάφορα επίπεδα αφαίρεσης μπορεί να γίνει με ποικίλους τρόπους :

1. Γραφικά.
2. Με λογικές συναρτήσεις ή με τους αντίστοιχους πίνακες αληθείας του σχεδιασμού.
3. Με εξειδικευμένες γλώσσες περιγραφής υλικού (Hardware Description Languages όπως για παράδειγμα οι καθιερωμένες Verilog και VHDL) ή κλασσικές γλώσσες με κατάλληλες δομές για την περιγραφή της παραλληλίας του υλικού (π.χ. Handel C).
4. Με μηχανές πεπερασμένων καταστάσεων (Finite State Machines – FSM).
5. Με γεννήτορες προκαθορισμένων σχεδιασμών (Block generators).
6. Με οποιονδήποτε συνδυασμό των παραπάνω τεχνικών.

---

Παρακάτω αναλύουμε σύντομα το γραφικό τρόπο εισαγωγής του σχεδιασμού. Διεξοδική ανάλυσή του γίνεται μέσω της εκπόνησης μέρους των εργαστηριακών ασκήσεων.

Κατά την γραφική μέθοδο περιγραφής, ο σχεδιαστής καλείται να "ζωγραφίσει" το κύκλωμά του σε έναν εξειδικευμένο γραφικό editor. Ο editor αυτός παρέχει γραφικά σύμβολα (προετοιμασμένα σχήματα) που αντιπροσωπεύουν στοιχειώδεις σχεδιασμούς μιας γενικευμένης τεχνολογίας. Για παράδειγμα παρέχει σύμβολα για τις πύλες AND, OR, NOT, XOR, αλλά και σύμβολα για μεγαλύτερα ολοκληρωμένα όπως για παράδειγμα ενός transceiver των οκτώ δυαδικών ψηφίων. Αν ευθύς εξαρχής ο σχεδιαστής το επιθυμεί μπορεί εναλλακτικά να προσκολλήσει το σχεδιασμό του σε μία ή σε ένα σύνολο από τεχνολογίες, οπότε στη διάθεσή του έχει μόνο σύμβολα που αντιπροσωπεύουν τα ολοκληρωμένα των τεχνολογιών που επέλεξε. Για παράδειγμα αν επιλέξει την υποοικογένεια χαμηλής κατανάλωσης της σειράς 7400, τότε θα έχει στη διάθεσή του NAND δύο, τριών, τεσσάρων ή οκτώ ακροδεκτών (ισοδύναμα τα ολοκληρωμένα 74LS00, 74LS10, 74LS20, 74LS30) αλλά όχι για παράδειγμα μία NAND επτά εισόδων, που ενδεχομένως να υπάρχει σε μια άλλη τεχνολογία. Φυσικά τίποτε δεν εμποδίζει το χρήστη να δημιουργήσει ένα δικό του νέο σύμβολο και να το επεξηγήσει γραφικά. Για παράδειγμα μπορεί να φτιάξει τη νέα LS TTL NAND των 7 εισόδων, φτιάχνοντας ένα νέο σύμβολο γι' αυτήν και επεξηγώντας γραφικά ότι το σύμβολο αυτό ισοδυναμεί με τον ακόλουθο υποσχεδιασμό :



Κάθε σύμβολο διαθέτει κάποια σημεία σύνδεσης, τα οποία προσομοιώνουν τους φυσικούς ακροδέκτες του ολοκληρωμένου ή της πύλης. Όλα τα σύμβολα οργανώνονται σε βιβλιοθήκες, ανάλογες με τις βιβλιοθήκες σχεδιασμού που προσφέρει κάθε κατασκευαστής. Για παράδειγμα η βιβλιοθήκη TTL του κατασκευαστή Texas Instruments, περιέχει γραφικά σύμβολα για όλα τα ολοκληρωμένα που κατασκευάζει η Texas Instruments σε τεχνολογία TTL. Κάθε ολοκληρωμένο δεν αποτελεί πάντα ένα αυτόνομο στοιχειώδες κομμάτι σχεδιασμού. Μπορεί να συγκροτείται από μικρότερα κομμάτια, τα οποία και διαχειρίζεται ο graphic editor. Για παράδειγμα όταν ζητήσουμε το part 74LS08 από την βιβλιοθήκη, τότε το πρόγραμμα θα μας παρουσιάσει μία πύλη AND δύο εισόδων και όχι ένα ολοκληρωμένο 74LS08 που περιέχει 4 αντίτυπα AND πυλών 2 εισόδων.

Ο σχεδιαστής υποδηλώνει στο εργαλείο την φυσική ένωση μεταξύ ακροδεκτών ολοκληρωμένων ή σχεδιασμών, σχεδιάζοντας γραμμές μεταξύ των σημείων σύνδεσης ή μεταξύ γραμμών. Επίσης έχει την δυνατότητα μία ομάδα τέτοιων γραμμών να την θεωρήσει σαν αρτηρία (bus) και να την διαχειρίζεται ανάλογα, δημιουργώντας έτσι ένα πρώτο επίπεδο αφαιρεσης. Επίσης ο σχεδιαστής μπορεί να δίνει ονόματα σε κάθε γραμμή, ώστε να διευκολυνθεί αργότερα στην εκσφαλμάτωση του σχεδιασμού του. Για να υποδηλώσει τις εισόδους και εξόδους του κυκλώματός του, συνήθως ο σχεδιαστής εισάγει ειδικά σύμβολα (ports), με τα οποία ενώνει τα σήματα εισόδου και εξόδου του σχεδιασμού του. Τέλος, θα πρέπει να σημειωθεί ότι υπάρχουν ειδικά σύμβολα για την τροφοδοσία και την γείωση του κυκλώματος, τα οποία όταν παραλείπονται εννοούνται από το εργαλείο. Για παράδειγμα, σε όλα τα σύμβολα του εργαλείου, συνήθως παραλείπονται οι ακροδέκτες της γείωσης και της τροφοδοσίας, αφού η ύπαρξη και η διασύνδεσή τους είναι προφανής.

Πριν συνεχίσουμε να εστιάζουμε στη γραφική εισαγωγή του σχεδιασμού, είναι καλό να ορίσουμε τις δύο βασικές φιλοσοφίες του. Αυτές είναι :

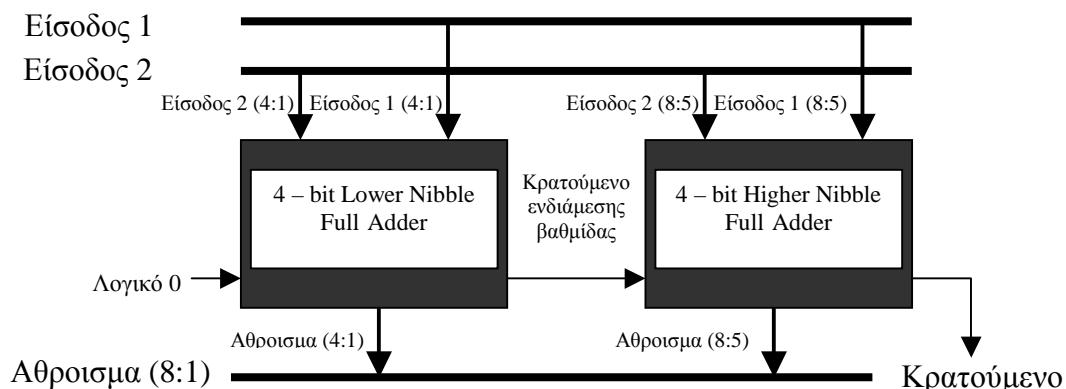
1. Η φιλοσοφία από το γενικότερο προς το ειδικότερο (Top – down) και
2. Η φιλοσοφία από το ειδικότερο προς το γενικότερο (Bottom – up).

Για να γίνουν κατανοητές οι δύο αυτές φιλοσοφίες ας ανατρέξουμε στο παράδειγμα που εισάγαμε πιο πάνω. Εστω ότι επιλέγουμε την πρώτη (top-down). Θα μπορούσαμε λοιπόν να ξεκινήσουμε την περιγραφή αυτού του απαιτούμενου αθροιστή των 8 δυαδικών ψηφίων σαν ένα μαύρο κουτί με δύο εισόδους των 8 δυαδικών ψηφίων και δύο εξόδους : το αποτέλεσμα 8 δυαδικών ψηφίων και το κρατούμενο. Με το γραφικό τρόπο αναπαράστασης τότε θα χρειαζόταν να σχεδιάσουμε κάτι ανάλογο με το παρακάτω σχήμα.



Το παραπάνω σχήμα από μόνο του είναι απλά μια γραφική αναπαράσταση και δεν ορίζει καμία σαφή λειτουργικότητα. Επειδή δηλαδή ζωγραφίσαμε ένα μαύρο κουτί και το ονομάσαμε αθροιστή δεν σημαίνει ότι πράγματι είναι αθροιστής. Οπως και στην περίπτωση της LS NAND των 7 εισόδων προηγούμενα, έτσι και ο αθροιστής

των 8 δυαδικών ψηφίων θα πρέπει κάπως να επεξηγηθεί περαιτέρω, έτσι ώστε να μειωθεί η αφαιρετικότητα του σχεδιασμού. Εστω ότι επιλέγουμε να τον υλοποιήσουμε σαν αθροιστή διάδοσης κρατουμένου (ripple carry adder) αποτελούμενο από δύο αντίγραφα αθροιστών των 4 δυαδικών ψηφίων. Φυσικά εσωτερικά ο κάθε αθροιστής των 4 δυαδικών ψηφίων, μπορεί και αυτός να υλοποιείται σαν αθροιστής διάδοσης κρατουμένου, ή σα αθροιστής με πρόβληψη κρατουμένου (carry look ahead adder). Τότε σε ένα επίπεδο αμέσως μεγαλύτερης σαφήνειας, θα καταλήγαμε στο επόμενο σχήμα, όπου χρησιμοποιούμε δύο αθροιστές των 4 δυαδικών ψηφίων με σύνδεση διάδοσης κρατουμένου για την υλοποίηση του αθροιστή των 8 δυαδικών ψηφίων. Δουλεύοντας αναδρομικά, μπορούμε να εκφράσουμε τον αθροιστή των 4 δυαδικών ψηφίων σαν σύνδεση δύο αθροιστών των 2 δυαδικών ψηφίων και έναν αθροιστή 2 δυαδικών ψηφίων σαν σύνδεση αθροιστών του 1 δυαδικού ψηφίου.



Θα πρέπει να σημειώσουμε ότι :

1. Με την παραπάνω διαδικασία δημιουργείται μια ιεραρχία επιπέδων σχεδιασμού, όπου το πιο πάνω επίπεδο γίνεται πολύ πιο συγκεκριμένο βάσει του αμέσως επομένου επιπέδου της ιεραρχίας. Ακόμη και αν έχουμε φτάσει σε μεγάλο επίπεδο λεπτομέρειας, ανά πάσα χρονική στιγμή μπορούμε να επιλέξουμε οποιοδήποτε από αυτά τα επίπεδα σαν το τρέχον επίπεδο για την επόμενη διαδικασία (αυτή της λογικής επαλήθευσης του σχεδιασμού).
2. Αυτόματα εισάγεται και η έννοια της επαναχρησιμοποίησης ενός υποσυστήματος (reusable design block). Για παράδειγμα, στο παραπάνω σχήμα δεν χρειάζεται να επεξηγήσουμε ιεραρχικά και τους δύο υποσχεδιασμούς για τους αθροιστές των 4 δυαδικών ψηφίων, εφόσον επιθυμούμε την ίδια λειτουργικότητα και για τους δύο. Αρκεί να αποσαφηνίσουμε (διατρέχοντας την ιεραρχία από το γενικότερο στο ειδικότερο) τον ένα από αυτούς και να δηλώσουμε στο E-CAD εργαλείο ότι και ο δεύτερος αποτελείται από μια ίδια ιεραρχία (hierarchical block).

---

3. Η διαδικασία αλλαγών και δοκιμών γίνεται σημαντικά ευκολότερη. Για παράδειγμα έστω ότι θέλουμε να μελετήσουμε τους χρόνους που προσφέρουν οι διάφορες αρχιτεκτονικές για τον αθροιστή μας. Το παραπάνω σχήμα θα χρειαστεί να υπάρχει στην ιεραρχία μας, ανεξάρτητα αν επιλέξουμε αρχιτεκτονική πρόβλεψης ή διάδοσης κρατουμένου για τους αθροιστές των 4 δυαδικών ψηφίων. Θα χρειαστεί να αλλάξει μόνο αν επιλέξουμε αρχιτεκτονική πρόβλεψης κρατουμένου για τον αθροιστή των 8 δυαδικών ψηφίων.

Η μέθοδος από το ειδικότερο προς το γενικότερο (που ιστορικά ήταν και η πρώτη που χρησιμοποιήθηκε) διατρέχει την ιεραρχία των επιπέδων σχεδιασμού κατά την αντίστροφη φορά. Ακολουθώντας αυτή τη μέθοδο για το παράδειγμά μας θα ξεκινούσαμε πρώτα από την περιγραφή (με κάποιον από τους πιθανούς τρόπους) του πλήρη αθροιστή του 1 δυαδικού ψηφίου. Θα χρησιμοποιούσαμε στη συνέχεια 2 τέτοιους για να φτιάξουμε τον πλήρη αθροιστή 2 δυαδικών ψηφίων και θα συνεχίζαμε κατ' αυτόν τον τρόπο μέχρι να φτάσουμε στο πιο κορυφαίο επίπεδο της ιεραρχίας. Προφανώς σε κάθε ενδιάμεσο επίπεδο θα είχαμε την ευκαιρία του ελέγχου της λογικής λειτουργίας του επιπέδου. Εδώ θα πρέπει να διευκρινίσουμε ότι ο ιεραρχικός τρόπος σχεδιασμού δεν θα πρέπει να συνδέεται με το επίπεδο αφαιρεσης κάθε περιγραφής. Αν και συνήθως τα πιο υψηλά στην ιεραρχία επίπεδα περιγράφουν το στοχευόμενο σύστημα με μεγαλύτερη αφαιρεση από ότι τα επόμενα, αυτό δεν είναι πάντα αληθές. Για παράδειγμα αν στη μέθοδο από το ειδικότερο στο γενικότερο περιγράψουμε τον πλήρη αθροιστή του 1 δυαδικού ψηφίου σαν ένα μαύρο κουτί, τίποτε δεν μας εμποδίζει να περιγράψουμε με απόλυτη σαφήνεια (μικρή αφαιρεση) όλα τα πιο πάνω επίπεδα της ιεραρχίας.

Ας δούμε στη συνέχεια πως εκφράζουμε την ιεραρχία ενός σχεδιασμού όταν χρησιμοποιούμε γραφική περιγραφή. Η γραφική περιγραφή ενός ιεραρχικού σχεδιασμού επιτυγχάνεται με τη χρήση ιεραρχικών blocks και ιεραρχικών parts. Στην πρώτη περίπτωση (που αντιστοιχεί στη φιλοσοφία top-down design) ο σχεδιαστής "ζωγραφίζει" κουτιά που αντιπροσωπεύουν άλλους σχεδιασμούς και τους επισυνάπτει σημεία σύνδεσης τα οποία αντικατοπτρίζουν εισόδους - εξόδους του block. Επιπλέον ένα τέτοιο block μπορεί να το διαχειρίζεται ανάλογα με όλα τα υπόλοιπα σύμβολα των βιβλιοθηκών σχεδιασμού (να το αντιγράφει, να το μετακινεί). Αργότερα, μπορεί να προσάψει λειτουργικότητα σε αυτό το block επεξηγώντας το σε μια καινούργια σελίδα σχηματικού σαν ένα σχεδιασμό από στοιχειώδη κομμάτια ή από άλλα ιεραρχικά blocks ή οποιονδήποτε συνδυασμό τους. Ασφαλώς η επεξήγηση του ιεραρχικού block δεν απαιτείται να γίνει με γραφικό τρόπο. Θα μπορούσε να χρησιμοποιηθεί οποιαδήποτε άλλη τεχνική περιγραφής κυριλωμάτων. Συνήθως η

---

δομή των αρχείων που δημιουργεί το E-CAD εργαλείο αντικατοπτρίζει και την ιεραρχία του σχεδιασμού. Στο γνωστό μας παράδειγμα, μια πιθανή δομή αρχείων θα ήταν η εξής :

```
... \ design \ schematic  
... \ design \ 8-bit full-adder \ schematic  
... \ design \ 8-bit full-adder \ 4-bit full-adder \ schematic  
... \ design \ 8-bit full-adder \ 4-bit full-adder \ 2-bit full-adder \ schematic  
... \ design \ 8-bit full-adder \ 4-bit full-adder \ 2-bit full-adder \ 1-bit full-adder \ schematic
```

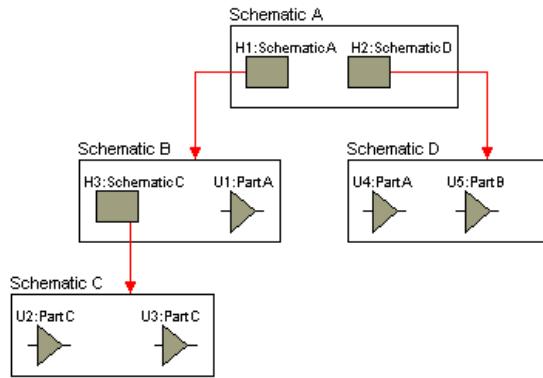
όπου ... \ design η ρίζα (root) του σχεδιασμού μας και schematic ένα αρχείο – σχηματικό που περιγράφει τον αντίστοιχο σχεδιασμό. Το παραπάνω επισημαίνει ότι για την επεξήγηση κάθε διαφορετικού ιεραρχικού block δημιουργείται και ένας υποκατάλογος (subdirectory) κάτω από το σχηματικό στο οποίο χρησιμοποιείται το ιεραρχικό block, έτσι ώστε να υπάρχει αντίστοιχη μεταξύ της ιεραρχίας του σχεδιασμού και της δομής των αρχείων. Συνήθως το εργαλείο παρέχει την ικανότητα της αυτόματης πλοήγησης μεταξύ των διαφορετικών επιπέδων της ιεραρχίας. Δηλαδή ενώ βρισκόμαστε στο σχηματικό του πλήρους αθροιστή οκτώ δυαδικών ψηφίων, επιλέγοντας το σύμβολο για τον πλήρη αθροιστή των τεσσάρων δυαδικών ψηφίων και εκτελώντας κάποια εντολή καθόδου στην ιεραρχία, μεταφερόμαστε στο σχηματικό που το επεξηγεί.

Στην δεύτερη περίπτωση (που αντιστοιχεί στη φιλοσοφία bottom-up design) ο σχεδιαστής αφού ολοκληρώσει κάποιο κομμάτι του σχεδιασμού του που θα ήθελε να χρησιμοποιήσει για να περιγράψει μεγαλύτερους σχεδιασμούς, το αποθηκεύει σαν καινούργιο σύμβολο (hierarchical part) των βιβλιοθηκών σχεδιασμού και μπορεί να το χρησιμοποιεί στα επόμενα βήματα σχεδίασης όπως ακριβώς τα υπόλοιπα στοιχειώδη σύμβολα της βιβλιοθήκης.

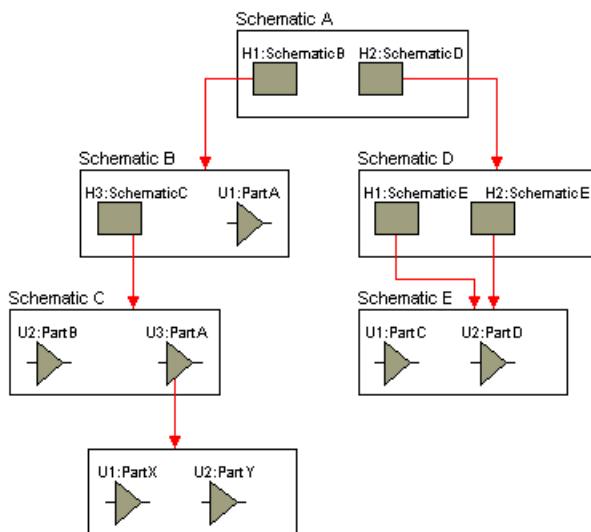
Ο σχεδιαστής προφανώς μπορεί να χρησιμοποιήσει οποτεδήποτε και οποιαδήποτε από τις δύο παραπάνω φιλοσοφίες. Στην πράξη καμία από τις παραπάνω φιλοσοφίες δεν έχει αποδειχθεί καλύτερη από την άλλη. Γι' αυτό άλλωστε όλα τα σύγχρονα E-CAD εργαλεία προσφέρουν την δυνατότητα να χρησιμοποιηθούν οι δύο πιο πάνω μέθοδοι σχεδιασμού εναλλακτικά και σε άριστη συνεργασία μεταξύ τους. Αυτό όμως που συνήθως καταγράφεται στην πράξη, είναι ότι σε ένα πολύπλοκο σύστημα για τον σχεδιασμό του οποίου θα εργαστούν διάφορα άτομα ακολουθείται η μέθοδος του γενικότερου προς το ειδικότερο μέχρι να καθοριστεί σαφώς το περιεχόμενο του σχεδιασμού που θα αναλάβει ο κάθε σχεδιαστής. Κατόπιν αυτού ο

κάθε σχεδιαστής είναι ελεύθερος να επιλέξει εκείνη την μεθοδολογία που του ταιριάζει καλύτερα ή να χρησιμοποιεί και τις δύο εναλλακτικά.

Πολλές φορές οι τεχνικές αυτές οδηγούν σε 1-1 αντιστοιχήσεις μεταξύ ιεραρχικών συμβόλων και σχηματικών σελίδων επεξήγησής τους. Τέτοιες ιεραρχίες ονομάζονται απλές, όπως για παράδειγμα αυτή του παρακάτω σχήματος.



Εναλλακτικά μπορεί να να οδηγηθούμε σε many to 1 αντιστοιχήσεις, δημιουργώντας πολύπλοκες ιεραρχίες όπως για παράδειγμα αυτή του σχήματος που ακολουθεί.



Θα πρέπει τέλος να αναφέρουμε ότι τα περισσότερα σύγχρονα E-CAD εργαλεία παρέχουν στον σχεδιαστή επιπλέον την δυνατότητα της οριζόντιας ιεραρχίας. Μεγάλοι σχεδιασμοί του ίδιου επιπέδου κάθετης ιεραρχίας που δεν χωρούν σε μία σελίδα σχηματικού ή που καταστούν την αναγνωσιμότητα του σχηματικού περιορισμένη μπορούν να κατατμηθούν σε μικρότερους δημιουργώντας έτσι την οριζόντια ιεραρχία. Η διασύνδεση αυτών των σελίδων του οριζόντιου σχεδιασμού γίνεται με ειδικά σύμβολα (interpage connectors). Ακόμη και στην ίδια σελίδα σχηματικού, διασυνδέσεις που χρειάζεται να περάσουν πάνω από άλλες και συνεπώς θα μπορούσαν να δημιουργήσουν παρανοήσεις μπορεί να κατατμηθούν σε

---

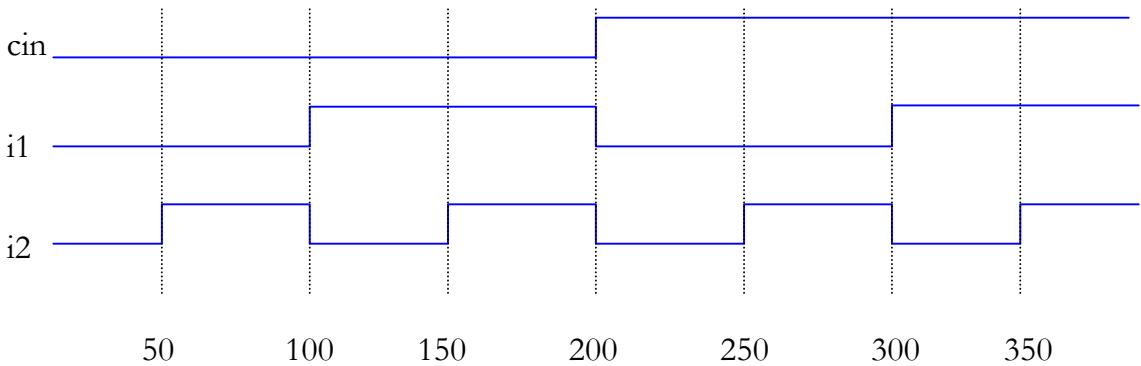
τμήματα μικρότερων διασυνδέσεων ιδεατά ενωμένων μέσω ειδικών συμβόλων (intrapage connectors).

Κλείνοντας αυτή την αναφορά στην γραφική εισαγωγή ενός σχεδιασμού, θα πρέπει να σημειώσουμε ότι αν και αυτή η μέθοδος αποτελεί ένα άριστο εκπαιδευτικό εργαλείο, σταδιακά εγκαταλείπεται από τις εταιρείες σχεδίασης συστημάτων λόγω του χρόνου που απαιτεί για την έκφραση μεγάλων σχεδιασμών, αλλά και της μειωμένης μεταφερσιμότητας των σχεδιασμών μεταξύ εργαλείων διαφορετικών εταιρειών συγγραφής E-CAD προγραμμάτων. Πιο κάτω στο σύγγραμμα αυτό εκετάζουμε νέους, πιο γρήγορους και πιο ευέλικτους τρόπους για την εισαγωγή ενός κυκλώματος.

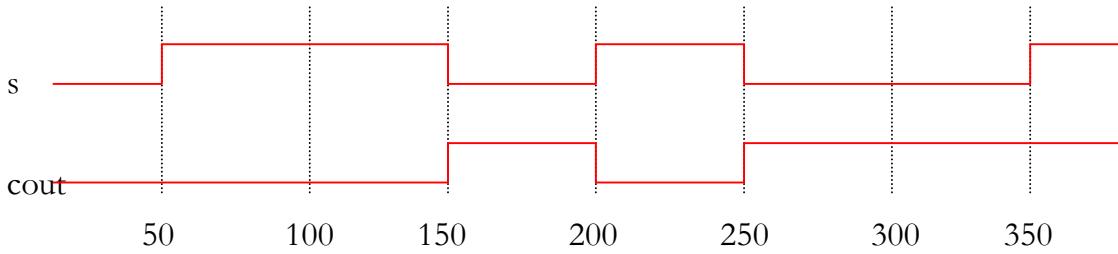
Αφού ολοκληρωθεί η εισαγωγή του σχεδιασμού σε κάποιο επίπεδο αφαιρεσης και με κάποια από τις τεχνικές που αναφέρθηκαν πιο πάνω, ακολουθεί το στάδιο του ελέγχου της λογικής λειτουργίας του. Στο στάδιο αυτό το ζητούμενο είναι να βεβαιωθούμε ότι όντως ο σχεδιασμός επιτελεί σωστά όλες τις επιθυμητές λειτουργίες (όπως αυτές τέθηκαν στις προδιαγραφές) οποιαδήποτε και αν είναι τα δεδομένα εισόδου του. Για τον έλεγχο χρησιμοποιείται ένα υποπρόγραμμα του E-CAD εργαλείου το οποίο ονομάζεται εξομοιωτής (simulator). Η διαδικασία χρήσης του εξομοιωτή για έλεγχο της λογικής λειτουργίας του σχεδιασμού ονομάζεται λογική εξομοίωση (logic simulation).

Οι είσοδοι που δέχεται ο εξομοιωτής είναι ο σχεδιασμός σε κάποια ειδική μορφή περιγραφής του δικτυώματος του υλικού (netlist) και ένα αρχείο καθορισμού των εισόδων του σχεδιασμού (stimulus file). Η κατασκευή του netlist είναι αρμοδιότητα του E-CAD εργαλείου. Δηλαδή, υπάρχει ένα ενδιάμεσο στάδιο μετάφρασης του σχεδιασμού από τις διάφορες μορφές που έχουμε χρησιμοποιήσει για την εισαγωγή του, σε αναπαράσταση κατανοητή από τον εξομοιωτή. Για παράδειγμα, αναφέρουμε ότι ένας εξομοιωτής ο οποίος δέχεται το netlist του κυκλώματος σε VHDL, μπορεί να δεχτεί κατ' ευθείαν σχεδιασμούς περιγραμμένους σε αυτή τη γλώσσα. Αντίθετα, θα πρέπει να μεταφραστούν σε αυτή τη γλώσσα κομμάτια του σχεδιασμού εισηγμένα γραφικά, με μηχανές πεπερασμένων καταστάσεων ή με κάθε άλλο τρόπο εισαγωγής σχεδιασμών. Αυτή η διαδικασία μετάφρασης συνήθως πραγματοποιείται με ένα υποπρόγραμμα του E-CAD εργαλείου που ονομάζεται netlister. Το αρχείο καθορισμού των εισόδων του σχεδιασμού (stimulus file) για τον εξομοιωτή δημιουργείται από τον σχεδιαστή με τη χρήση είτε γραφικών μεθόδων είτε με κάποια γλώσσα περιγραφής. Για παράδειγμα, έστω ότι θέλουμε να ελέγξουμε την λογική λειτουργία του σχεδιασμού για τον πλήρη αθροιστή του 1 δυαδικού ψηφίου που θα χρησιμοποιήσουμε για να φτιάξουμε τον πλήρη αθροιστή 8 δυαδικών ψηφίων. Τότε, αν θεωρήσουμε ότι ο αθροιστής αυτός

δέχεται σαν εισόδους τα σήματα `cin` (κρατούμενο εισόδου), `i1` (πρώτο έντελο), `i2` (δεύτερο έντελο) και παράγει δύο εξόδους `s` (άθροισμα) και `cout` (κρατούμενο εξόδου), ένα πιθανό `stimulus` file περιγεγραμμένο γραφικά που μπορεί να χρησιμοποιηθεί για πλήρη έλεγχο της λογικής λειτουργίας του κυκλώματος είναι αυτό του σχήματος που ακολουθεί, όπου ζητάμε σε διάφορες χρονικές στιγμές της εξομοίωσης τα σήματα εισόδου να πάρουν συγκεκριμένες στιγμές. Διευκρινίζεται ότι οι χρόνοι που φαίνονται στο κάτω μέρος δεν αντιπροσωπεύουν συγκεκριμένα χρονικά μεγέθη· θα μπορούσαμε να εννοούμε `ps`, `ns` ή δευτερόλεπτα. Αντιπροσωπεύουν απλά συγκεκριμένα πολλαπλάσια της εννοούμενης μονάδας χρόνου εξομοίωσης.



Θεωρώντας ότι έχουμε κάνει σωστά τον σχεδιασμό του πλήρους αθροιστή 1 δυαδικού ψηφίου και τρέχοντας την εξομοίωση για 400 χρονικές μονάδες, το αποτέλεσμα της λογικής εξομοίωσης περιγεγραμμένο με γραφικό τρόπο θα ήταν αυτό που παρουσιάζεται παρακάτω. Προφανώς ο εξομοιωτής μπορεί να μας δώσει τα αποτελέσματα και σε άλλες μη γραφικές μορφές.



Οι σύγχρονοι εξομοιωτές δεν περιορίζονται απλά στη διαχείριση σημάτων με τιμές το 0 και το 1. Αντίθετα, χρησιμοποιούν ένα επαυξημένο σύνολο δυνατών τιμών για τα σήματα που λαμβάνουν μέρος στην εξομοίωση και το οποίο λαμβάνει υπόψη του και την οδηγητική ικανότητα των σημάτων. Μερικές από τις πιθανές τιμές για ένα σήμα στους σύγχρονους εξομοιωτές πέρα από το 0 και 1 μπορεί να είναι:

- ◆ U (unknown). Το σήμα βρίσκεται σε κατάσταση απροσδιοριστίας.
- ◆ Z (high Z). Το σήμα παρουσιάζει υψηλή εμπέδηση.

- 
- ♦ R1 (resistive 1). Το σήμα βρίσκεται στο λογικό 1, συνδεδεμένο στη τροφοδοσία μέσω κάποιας αντίστασης. Προφανώς αυτό το σήμα έχει μικρότερη οδηγητική ικανότητα από ένα σήμα που βρίσκεται στο 1. Η βραχυχκύλωση ενός τέτοιου σήματος με το 0, θα δώσει 0 στην έξοδο και όχι κατάσταση απροσδιοριστίας.
  - ♦ R0 (resistive 0). Το σήμα βρίσκεται στο λογικό 0, συνδεδεμένο στη γείωση μέσω κάποιας αντίστασης. Προφανώς αυτό το σήμα έχει μικρότερη οδηγητική ικανότητα από ένα σήμα που βρίσκεται στο 0. Η βραχυχκύλωση ενός τέτοιου σήματος με το 1, θα δώσει 1 στην έξοδο και όχι κατάσταση απροσδιοριστίας.

Τα περισσότερα E-CAD εργαλεία της σημερινής γενιάς, χρησιμοποιούν κάποια κοινή βάση δεδομένων για όλα τα υποπρογράμματά τους. Αυτή η φιλοσοφία, επιτρέπει την απρόσκοπη επικοινωνία και συνεργασία μεταξύ των διαφορετικών εργαλείων που συνιστούν το E-CAD εργαλείο (inter-tool communication), παρέχοντας ταυτόχρονα στον σχεδιαστή σημαντικές ευκολίες για τον γρήγορο εντοπισμό λογικών λαθών και την εκσφαλμάτωση (debugging) του σχεδιασμού. Για παράδειγμα επιλέγοντας μία συγκεκριμένη χρονική στιγμή στις κυματομορφές του εξομοιωτή αυτόματα μπορούμε να δούμε στον γραφικό editor τις λογικές τιμές που έχουν όλες οι είσοδοι, οι έξοδοι και οι ενδιάμεσοι κόμβοι του κυκλώματός μας ή στον editor κειμένου τις τιμές που έχουν οι μεταβλητές που χρησιμοποιούμε για περιγραφή σε HDL και να εντοπίσουμε άμεσα τυχόν λογικά λάθη στο σχεδιασμό μας. Η παραπάνω διαδικασία αναφέρεται σαν διαγώνια έρευνα (cross – probing).

Το επόμενο βήμα στη διαδικασία σχεδιασμού είναι η χρονική εξομοίωση του σχεδιασμού. Στο στάδιο αυτό το ζητούμενο δεν είναι απλά να βεβαιωθούμε ότι όντως ο σχεδιασμός μας επιτελεί σωστά τις επιθυμητές λειτουργίες (όπως αυτές τέθηκαν στις προδιαγραφές) όποια και αν είναι τα δεδομένα εισόδου του, αλλά επιπλέον ότι τηρεί και τις χρονικές προδιαγραφές. Για τον έλεγχο χρησιμοποιείται και πάλι ο εξομοιωτής (simulator) του E-CAD εργαλείου, αυτή τη φορά όμως απαιτείται επιπλέον του netlist και του stimulus file να εισάγουμε και κάποια μοντέλα χρονικής καθυστέρησης για τα διάφορα κομμάτια που αποτελούν τον σχεδιασμό μας. Η διαδικασία χρήσης του εξομοιωτή για έλεγχο της χρονικής λειτουργίας του σχεδιασμού ονομάζεται χρονική εξομοίωση (timing simulation).

Οι τιμές των μοντέλων χρονικής καθυστέρησης μπορεί να είναι εντελώς ιδεατές (οριζόμενες από τον σχεδιαστή) ή αυστηρά προκαθορισμένες από την στοχευόμενη τεχνολογία ή συνδυασμός τους. Στην δεύτερη περίπτωση, το E-CAD εργαλείο θα πρέπει να διαθέτει όλα τα πιθανά χρονικά μοντέλα για κάθε στοιχειώδες κομμάτι του σχεδιασμού σε κάθε εξεταζόμενη τεχνολογία υλοποίησης, ώστε να δίνει στον σχεδιαστή την δυνατότητα να διερευνήσει πιθανές εναλλακτικές τεχνολογίες

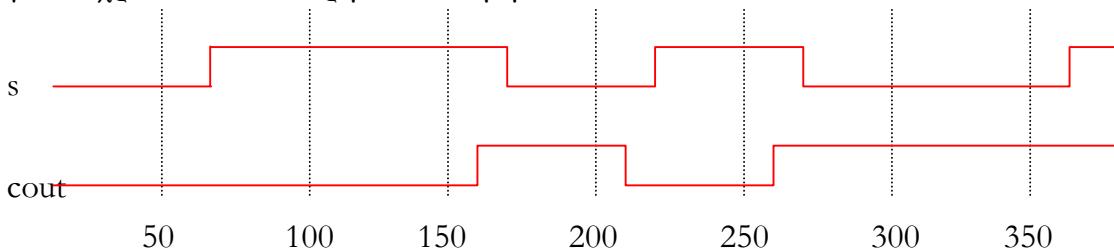
υλοποίησης. Θα εξηγήσουμε τις δύο αυτές εναλλακτικές λύσεις με βάση το παρόδειγμά μας.

Εστω ότι ο σχεδιαστής αποφασίζει να εισάγει από μόνος του τις πληροφορίες για την χρονική καθυστέρηση του πλήρους αθροιστή ενός δυαδικού ψηφίου και ότι αποφασίζει να το κάνει χρησιμοποιώντας περιγραφή σε Verilog. Τότε θα μπορούσε να γράψει τις κάτωθι γραμμές κώδικα :

```
#20 assign s = i1 ^ i2 ^ cin;
```

```
#10 assign cout = (i1 and i2) or (i1 and cin) or (i2 and cin);
```

όπου  $\wedge$  συμβολίζει τη συνάρτηση XOR, and και or οι γνωστές λογικές συναρτήσεις και  $\#x$  υποδεικνύει χρονική καθυστέρηση  $x$  χρονικών μονάδων. Οι παραπάνω γραμμές υποδεικνύουν στον εξομοιωτή ότι η συνάρτηση του αθροίσματος παρουσιάζει χρονική καθυστέρηση 20 χρονικών μονάδων και αυτή του κρατουμένου 10 χρονικών μονάδων. Χρησιμοποιώντας αυτό το μοντέλο χρονικής καθυστέρησης και τρέχοντας χρονική εξομοίωση 400 χρονικών μονάδων με το ίδιο stimulus file όπως και προηγούμενα θα προκύψουν τα αποτελέσματα του σχήματος που ακολουθεί. Προσέξτε ότι ο εξομοιωτής πλέον, λαμβάνοντας υπ' όψιν του την χρονική πληροφορία του σχεδιαστή έχει μεταθέσει τις μεταβάσεις των εξόδων του ικανώματος κατά αντίστοιχες χρονικές μονάδες εξομοίωσης. Προφανώς η λογική εξομοίωση είναι ένα υποσύνολο της χρονικής, αυτό στο οποίο όλες οι πληροφορίες για τις χρονικές καθυστερήσεις είναι μηδενικές.



Χρησιμοποιώντας ακόμη και αυτό το ιδεατό χρονικό μοντέλο, ο σχεδιαστής μπορεί να βγάλει χρήσιμα συμπεράσματα και για τα υψηλότερα επίπεδα ιεραρχίας του σχεδιασμού του. Για παρόδειγμα αν χρησιμοποιήσει δύο τέτοιους αθροιστές του 1 δυαδικού ψηφίου με διάδοση κρατουμένου για να φτιάξει έναν πλήρη αθροιστή των 2 δυαδικών ψηφίων, θα μετρήσει καθυστέρηση διάδοσης για την έξοδο του αθροίσματος ίση με 30 χρονικές μονάδες και για την έξοδο του κρατουμένου 20 χρονικές μονάδες. Τελικά, θα μπορέσει να καταλήξει στις κάτωθι σχέσεις για τις καθυστερήσεις των εξόδων ενός αθροιστή  $2^x$  δυαδικών ψηφίων σχεδιασμένο αναδρομικά με διάδοση κρατουμένου και δύο αθροιστών  $2^{x-1}$  δυαδικών ψηφίων:

$$\text{Καθυστέρηση αθροίσματος } (2^x) = 20 + (2^{x-1}) * \text{Καθυστέρηση κρατουμένου } (2^0).$$

$$\text{Καθυστέρηση κρατουμένου } (2^x) = 10 + (2^{x-1}) * \text{Καθυστέρηση κρατουμένου } (2^0).$$

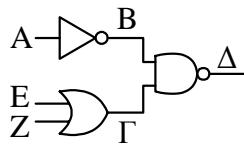
---

Αν λοιπόν οι χρονικές προδιαγραφές του συστήματος, καθορίζουν συχνότητα λειτουργίας 25 MHz, ο σχεδιαστής είναι σε θέση να συμπεράνει ότι θα πρέπει να κατευθύνει τον σχεδιασμό του σε μία τεχνολογία υλοποίησης που θα του επιτρέπει να υλοποιήσει τον αθροιστή 1 δυαδικού ψηφίου με καθυστέρηση διάδοσης της τάξης των 3 – 4 ns το πολύ. Σε περίπτωση που κάτι τέτοιο είναι αδύνατο λόγω του προϋπολογισμού του σχεδιασμού θα πρέπει από τώρα να ξεχάσει την λύση της διάδοσης κρατουμένου και να στραφεί προς αθροιστές με πρόβλεψη κρατουμένου.

Ακολουθώντας από την άλλη πλευρά την λύση των αυστηρά καθορισμένων μοντέλων χρονικής καθυστέρησης ο σχεδιαστής θα εξετάσει μία - μία τις διαθέσιμες τεχνολογίες υλοποίησης. Το πόσες και ποιες από αυτές χρειάζεται να εξετάσει είναι συνάρτηση γνώσεων και εμπειρίας. Για το παρόντευγμά μας, ο έμπειρος σχεδιαστής δεν θα ασχοληθεί διόλου με τις υποοικογένειες Standard και Low Power Schottky της σειράς TTL σαν πιθανές τεχνολογίες υλοποίησης, γιατί εφόσον στην καλύτερη περίπτωση μία πύλη αυτών των τεχνολογιών έχει καθυστέρηση τουλάχιστον 10 ns και απαιτούνται τουλάχιστον 2 επίπεδα πυλών για την πραγματοποίηση κάθε μιας από τις εξόδους του πλήρους αθροιστή ενός δυαδικού ψηφίου είναι αδύνατο να μπορέσει να εκπληρώσει την χρονική προδιαγραφή των 25 MHz σε οποιαδήποτε από αυτές τις τεχνολογίες υλοποίησης. Για κάθε μία από τις διαφορετικές τεχνολογίες, θα πρέπει να δηλώσει ότι τα στοιχειώδη κομμάτια (parts) του σχεδιασμού του ανήκουν στην επιλεγμένη τεχνολογία. Για παράδειγμα, αν επιλέξει την AS TTL και κάπου στο σχεδιασμό του χρησιμοποιεί μια πύλη AND θα πρέπει να υποδείξει στο E-CAD εργαλείο ότι πρόκειται για μία AS TTL AND. Ο netlister του εργαλείου, αυτόματα αναλαμβάνει να επισυνάψει τη πληροφορία χρονικής καθυστέρησης κάθε part στο netlist του κυκλώματος, έτσι ώστε αυτή να διαβαστεί με τη λογική του κυκλώματος από τον εξομοιωτή.

Τα σύγχρονα E-CAD εργαλεία οργανώνουν τις πληροφορίες για τις χρονικές καθυστερήσεις των parts σε βιβλιοθήκες ανάλογα με την κάθε τεχνολογία υλοποίησης. Επιπλέον θυμηθείτε για κάθε part θα πρέπει να παρέχουν τρεις πληροφορίες χρονικής καθυστέρησης : της καλύτερης, της τυπικής και της χειρότερης περίπτωσης (minimum, typical, maximum delay). Ο σχεδιαστής είναι εκείνος που θα επιλέξει ποιο από τα τρία αυτά μοντέλα ή ποιος συνδυασμός τους, θα χρησιμοποιηθεί κατά την χρονική εξομοιωση.

Ας ασχοληθούμε λίγο περισσότερο με τη λειτουργία του ίδιου του εξομοιωτή, για να καταλάβουμε το πώς λειτουργεί, χρησιμοποιώντας τον ακόλουθο σχεδιασμό.



Το πρώτο στάδιο στην διαδικασία εξομοίωσης, είναι η κατασκευή από τον εξομοιωτή των πινάκων εξάρτησης, για νάθε κόμβο του κυκλώματος εισόδου. Για παράδειγμα, η τιμή της γραμμής  $B$  του εξαρτάται μόνο από την τιμή της εισόδου  $A$ , ενώ η τιμή της γραμμής  $\Gamma$  εξαρτάται τόσο από την τιμή της εισόδου  $E$ , όσο και από την τιμή της εισόδου  $Z$ . Αναλύοντας όλο το κύκλωμα τελικά, θα καταλήξουμε στον παρακάτω πίνακα εξάρτησης για το κύκλωμα μας. Οι κύριες είσοδοι του κυκλώματος δεν παρουσιάζουν καμία εξάρτηση από άλλες γραμμές του κυκλώματος.

Γραμμή	Εξαρτάται Από
$A$	-
$B$	$A$
$\Gamma$	$E, Z$
$\Delta$	$B, \Gamma$
$E$	-
$Z$	-

Το δεύτερο στάδιο στην διαδικασία εξομοίωσης είναι η επαύξηση του πίνακα εξάρτησης με την χρονική πληροφορία για τις καθυστερήσεις των ενδιάμεσων κόμβων. Ας υποθέσουμε ότι όλες αυτές οι πληροφορίες είναι εκφρασμένες σε ns και ότι επειδή είναι πληροφορία την οποία έχει προσάψει ο χρήστης, δεν υπάρχουν διαφορετικές τιμές για νάθε είσοδο πύλης με περισσότερες από μία εισόδους. Τέλος ας υποθέσουμε ότι όλες οι χρονικές πληροφορίες είναι απόλυτες τιμές και όχι ένα διάστημα τιμών. Υποθέτουμε ότι η καθυστέρηση του αντιστροφέα είναι 5ns, της πύλης OR 8ns και της πύλης NAND 6ns. Ετσι μπορούμε να οδηγηθούμε στον παρακάτω επαυξημένο πίνακα εξάρτησης.

Το τρίτο και τελικό στάδιο της διαδικασίας εξομοίωσης είναι η ανάγνωση του αρχείου εισόδων (stimulus file), ανά χρονική στιγμή εξομοίωσης. Ας υποθέσουμε ότι έχουμε ορίσει τις εξής μεταβάσεις για τις εισόδους μας :

Είσοδος  $A$  : 0 την χρονική στιγμή 0 και 1 την χρονική στιγμή 50.

Είσοδος  $E$  : Σταθερά στο 0.

Είσοδος  $Z$  : 0 την χρονική στιγμή 0 και 1 την χρονική στιγμή 40.

Γραμμή	Εξαρτάται Από
A	-
B	A/5
Γ	E/8, Z/8
Δ	B/6, Γ/6
E	-
Z	-

Αρχικά όλοι οι κόμβοι του κυκλώματος βρίσκονται σε άγνωστη κατάσταση (U). Κατά την ανάγνωση του αρχείου εισόδου, ο εξομοιωτής παρατηρεί αλλαγή στις τιμές των σημάτων A, E, Z την χρονική στιγμή 0. Συμβουλευόμενος τον επαυξημένο πίνακα εξάρτησης, προγραμματίζει την μεταβολή των κόμβων του κυκλώματος ως εξής : Ο κόμβος B που εξαρτάται από το A, θα πρέπει μετά από 5 χρονικές μονάδες να μεταβεί στο 1 και ο κόμβος Γ θα πρέπει μετά από 8 χρονικές μονάδες να μεταβεί στο 0. Καμία μετάβαση δεν παρατηρείται στο κύκλωμα μέχρι την χρονική στιγμή 5. Στην χρονική στιγμή 5, υπάρχει αλλαγή στην τιμή του σήματος B που οδηγείται στο 1. Ωστόσο επειδή η γραμμή Γ, βρίσκεται ακόμη σε άγνωστη κατάσταση ένας «έξυπνος» εξομοιωτής δεν πρόκειται να προγραμματίσει καμία μεταβολή στην τιμή του σήματος Δ. Την χρονική στιγμή 8, υπάρχει προγραμματισμένη μεταβολή τιμής για την γραμμή Γ. Ταυτόχρονα με την οδήγηση της γραμμής Γ στο 0, ο εξομοιωτής προγραμματίζει μεταβολή στην τιμή της γραμμής Δ, μετά από 6 χρονικές στιγμές. Ετσι την χρονική στιγμή 14, η έξοδος του κυκλώματος οδηγείται στο 1. Καμία μεταβολή δεν παρατηρείται μέχρι την χρονική στιγμή 40, στην οποία υπάρχει μεταβολή στην γραμμή Z, η οποία θα προκαλέσει μεταβολή στην τιμή του σήματος Γ την χρονική στιγμή 48 και ταυτόχρονα ο εξομοιωτής θα προγραμματίσει μεταβολή στην τιμή της εξόδου την χρονική στιγμή 54. Ενδιάμεσα την χρονική στιγμή 50, υπάρχει μεταβολή στην τιμή του A, που θα προκαλέσει μεταβολή στην τιμή του B την χρονική στιγμή 55 και της εξόδου την χρονική στιγμή 61.

Γραμμή	Αρχική Κατάσταση	Χρονική Στιγμή (ns)									
		0	5	8	14	40	48	50	54	55	61
A	U	0	0	0	0	0	0	1	1	1	1
B	U	U	1	1	1	1	1	1	1	0	0
Γ	U	U	U	0	0	0	1	1	1	1	1
Δ	U	U	U	U	1	1	1	1	0	0	1
E	U	0	0	0	0	0	0	0	0	0	0
Z	U	0	0	0	0	1	1	1	1	1	1

---

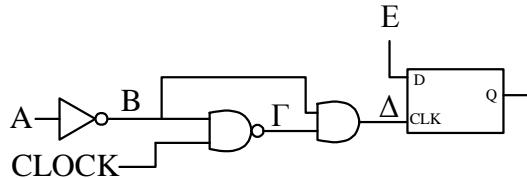
Οπως φαίνεται και από το παραπάνω παράδειγμα, η διαδικασία της εξομοίωσης είναι ένα αρκετά πολύπλοκο πρόβλημα. Ακόμη και για το στοιχειώδες παράδειγμά μας, η μνήμη που απαιτεί ο εξομοιωτής για να αποθηκεύσει όλα τα ενδιάμεσα αποτελέσματα ακόμη και μόνο για τις χρονικές στιγμές στις οποίες παρουσιάζονται αλλαγές είναι σημαντική. Για ένα κύκλωμα που αποτελείται από εκατομμύρια πύλες, η μνήμη που μπορεί να απαιτηθεί μπορεί να είναι υπερβολικά μεγάλη.

Μία λύση που προσφέρεται από τους περισσότερους εμπορικούς εξομοιωτές, είναι να αποθηκεύονται μόνο οι τιμές της εξομοίωσης για επιλεγόμενους από τον χρήστη κόμβους του κυκλώματος. Επειδή κάτι τέτοιο θα έκανε την διαδικασία εντοπισμού λαθών σημαντικά δυσκολότερη, συνήθως η εξομοίωση του κυκλώματος γίνεται σε δύο στάδια : κατά το πρώτο στάδιο ο σχεδιαστής ζητά από τον εξομοιωτή να αποθηκεύει όλη την πληροφορία και εκτελεί την εξομοίωση για πολύ μικρό χρονικό διάστημα έτσι ώστε να μπορεί να εντοπίσει τυχόν σχεδιαστικά λάθη γρήγορα και απλά και κατά το δεύτερο στάδιο που εκτελεί την εξομοίωση για μεγάλο χρόνο επιλέγει λίγους και σημαντικούς μόνο κόμβους του κυκλώματός του.

Επιπλέον αυτού, ο εξομοιωτής θα πρέπει να είναι σε θέση να διαχειρίζεται επαυξημένους πίνακες αληθείας για όλα τα στοιχεία που αποτελούν το κύκλωμά μας. Παρότι αναφερόμαστε σε ψηφιακούς εξομοιωτές, το δυαδικό σύστημα τιμών δεν είναι ικανό να μοντελοποιήσει όλα τα ψηφιακά κυκλώματα, όπως για παράδειγμα αρτηρίες τριών καταστάσεων με ή χωρίς παθητικά στοιχεία ανύψωσης ή πτώσης δυναμικού. Οι επαυξημένοι πίνακες αληθείας θα πρέπει να περιλαμβάνουν δυναμικά υψηλής εμπέδησης, παθητικό 0 ή 1 κλπ.

Επιπροσθέτως, για το παράδειγμά μας έχουμε επίσης νάνει ορισμένες παραδοχές οι οποίες στην πράξη δεν ισχύουν. Για παράδειγμα για τις πύλες δύο εισόδων του κυκλώματός μας έχουμε υποθέσει ίδια καθυστέρηση διάδοσης, άσχετα με το ποια από τις δύο εισόδους αλλάζει κατάσταση, πράγμα που σπανιότατα συμβαίνει στην πράξη. Επιπλέον έχουμε χρησιμοποιήσει μία σταθερή τιμή για την καθυστέρηση διάδοσης, αντί για ένα διάστημα τιμών. Στην πράξη, ο κατασκευαστής των ολοκληρωμένων, μας παρέχει την καθυστέρηση διάδοσης σαν τρεις διαφορετικές τιμές : της καλύτερης, της μέσης και της χειρότερης περίπτωσης. Για να είμαστε σίγουροι για τα αποτελέσματα της χρονικής εξομοίωσης, θα πρέπει τα χρονικά μοντέλα που χρησιμοποιούνται να λαμβάνουν υπόψη τους και τις τρεις αυτές τιμές και κατά την χρονική εξομοίωση να εξετάζονται όλες οι πιθανές περιπτώσεις. Αν στο παράδειγμα, η καθυστέρηση του αντιστροφέα ήταν οποιαδήποτε τιμή μεταξύ 1ns και 5ns, η μετάβαση της εξόδου στο λογικό 0 μπορεί να μην υπήρχε.

Ενα άλλο πρόβλημα που εμφανίζεται κατά την εξομοίωση (ιδιαίτερα την λογική) είναι η σωστή διάταξη της σειράς των γεγονότων που προκαλούνται από καθυστερήσεις που θεωρούνται μηδενικές και το γεγονός ότι ο εξομοιωτής είναι στην ουσία ένα ακολουθιακό πρόγραμμα που καλείται να εξομοιώσει την παραλληλία του υλικού κάποιου κυκλώματος. Για παράδειγμα θεωρείστε το παρακάτω σχήμα που μπορεί να είναι ένα κομμάτι σχεδιασμού για την παραγωγή του ρολογιού ενός συστήματος.



Ας υποθέσουμε ότι όλες οι πύλες έχουν μηδενική καθυστέρηση (όπως για παράδειγμα στην περίπτωση της λογικής εξομοίωσης) και ότι σε κάποια χρονική στιγμή της εξομοίωσης το σήμα  $\Gamma$  έχει πάρει την λογική τιμή 1, η είσοδος του ακολουθιακού στοιχείου  $E$  είναι στο λογικό 0 και η έξοδός του στο λογικό 1. Τότε όταν υπάρχει μια καθοδική ακμή στην είσοδο  $A$ , το σήμα  $B$  θα οδηγηθεί κατευθείαν στο λογικό 1 και ο εξομοιωτής θα προγραμματίσει μεταβολές τιμής και για το σήμα  $\Gamma$  και για το σήμα  $\Delta$  μετά από 0 χρονικές μονάδες. Τότε, ανάλογα με το ποιο από τα δύο σήματα θα υπολογιστεί πρώτο, υπάρχουν οι εξής δύο περιπτώσεις :

- ♦ Το σήμα  $\Gamma$  υπολογίζεται πρώτο. Σε αυτήν την περίπτωση το  $\Gamma$  θα οδηγηθεί στο 0 και θα προκαλέσει και μεταβολή του σήματος  $\Delta$  (σήμα χρονισμού για το ακολουθιακό στοιχείο) μετά από 0 χρονικές μονάδες. Με αυτή τη διάταξη των γεγονότων, δεν θα υπάρχει σήμα χρονισμού για το ακολουθιακό στοιχείο, αφού δεν υπάρχει ανοδική ακμή του ρολογιού του. Συνεπώς η έξοδός του θα συνεχίσει να βρίσκεται στο λογικό 1.
- ♦ Το σήμα  $\Delta$  υπολογίζεται πρώτο. Τότε επειδή το  $\Gamma$  βρίσκεται στο 1, το  $\Delta$  θα οδηγηθεί στο 1. Ο υπολογισμός του σήματος  $\Gamma$  που θα ακολουθήσει, θα οδηγήσει το  $\Delta$  στο 0. Με αυτήν την διάταξη των γεγονότων, υπάρχει έστω και μηδενικού χρόνου ανοδική ακμή στο σήμα χρονισμού του ακολουθιακού κυκλώματος, και συνεπώς η έξοδος του κυκλώματος θα αλλάξει από 1 σε 0.

Οπως είναι προφανές χωρίς κάποια σαφή διάταξη των γεγονότων, η εξομοίωση του παραπάνω κυκλώματος σε δύο διαφορετικούς εξομοιωτές μπορεί να οδηγήσει σε διαφορετικά αποτελέσματα. Για την επίλυση του παραπάνω προβλήματος, οι περισσότεροι εξομοιωτές, αντί μηδενικών καθυστερήσεων χρησιμοποιούν δ (delta) χρόνο εξομοίωσης, όπου δ ένα απειροστό χρονικό διάστημα. Κάθε αλλαγή μιας

---

γραμμής προκαλεί την αλλαγή στην έξοδο των πυλών που δέχονται αυτή τη γραμμή σαν είσοδο μετά από χρόνο δ. Με την χρήση των δ για εξομοίωση, για το κύκλωμά μας θα παίρναμε τα εξής αποτελέσματα (θεωρείστε ότι η καθοδική ακμή συμβαίνει την χρονική στιγμή 10 της εξομοίωσης) :

- ♦ Χρονική στιγμή 10 : αλλαγή του A σε 0
- ♦ Χρονική στιγμή 10 + δ : αλλαγή του B σε 1
- ♦ Χρονική στιγμή 10 + 2δ : αλλαγή του Δ σε 1 και του Γ σε 0
- ♦ Χρονική στιγμή 10 + 3δ : αλλαγή του Δ σε 0.

Κλείνοντας, χρειάζεται να τονίσουμε ότι ακόμη και όταν ο σχεδιασμός μας λειτουργεί τόσο από λογικής όσο και από χρονικής πλευράς σωστά κάτω από χρονική εξομοίωση χειρότερης περίπτωσης δεν μπορούμε να είμαστε σίγουροι ότι το σύστημά μας θα είναι πλήρως λειτουργικό κατά την παραγωγή του. Ακόμη δεν έχουμε λάβει υπόψη μας καμία παράμετρο της φυσικής υλοποίησης του συστήματος, όπως για παράδειγμα την αντίσταση και την χωρητικότητα των φυσικών γραμμών διασύνδεσης των ολοκληρωμένων ή τρανζίστορ του συστήματός μας. Αυτό θα συμβεί στις διαδικασίες της δεύτερης φάσης (back-end) της ροής του σχεδιασμού.

Η δεύτερη φάση του σχεδιασμού είναι άμεσα εξαρτημένη από την τεχνολογία υλοποίησής του. Πρώτος στόχος της δεύτερης φάσης είναι η φυσική υλοποίηση του σχεδιασμού είτε σαν πλακέτα (Printed Circuit Board) είτε σαν ολοκληρωμένο σε κάποια τεχνολογία γρήγορης πρωτότυποποίησης. Δεν θα ασχοληθούμε παρακάτω με τις κλασσικές τεχνολογίες υλοποίησης ολοκληρωμένων, καθώς αυτό αποτελεί αντικείμενο άλλων μαθημάτων. Ωστόσο, πρέπει να διευκρινίσουμε ότι για αρκετές από αυτές, εφαρμόζονται παρόμοιες (αν όχι οι ίδιες) διαδικασίες. Ενας δεύτερος στόχος της δεύτερης φάσης είναι η λογική και χρονική επαλήθευση του φυσικά υλοποιημένου σχεδιασμού. Τρίτος στόχος είναι η εξαγωγή κάποιων διανυσμάτων δοκιμής (test vectors), τα οποία θα χρησιμοποιήσουμε για την πιστοποίηση των πρωτότυπων. Τέλος, σαν τέταρτο στόχο θα πρέπει να αναφέρουμε την εξαγωγή πληροφορίας σχετική με τη φυσική υλοποίηση του σχεδιασμού, η οποία μπορεί να χρησιμοποιηθεί για την υλοποίηση του σχεδιασμού από κάποιο εργοστάσιο κατασκευής πρωτότυπων (ή από την ίδια τη σχεδιαστική ομάδα), για την εκτίμηση του κόστους για την κατασκευή κάθε πρωτότυπου και για την τεκμηρίωση του σχεδιασμού.

Η δεύτερη φάση ξεκινά με είσοδο ένα netlist του κυκλώματος, που στην περίπτωση αυτή θα πρέπει να περιγράφει με απόλυτη ακρίβεια το στοχευόμενο σύστημα στην συγκεκριμένη τεχνολογία υλοποίησης. Ολες οι τυχόν αφαιρέσεις που είχαν χρησιμοποιηθεί στα προηγούμενα στάδια του σχεδιασμού, θα πρέπει να

---

απαλειφθούν πριν μπορέσουμε να προχωρήσουμε στις διαδικασίες αυτής της φάσης. Τέτοιες ασάφειες προκύπτουν από την περιγραφή του κυκλώματος με γλώσσες περιγραφής υλικού, εξισώσεις, μηχανές πεπερασμένων καταστάσεων, ιεραρχιών γραφημάτων. Προκειμένου να μεταβούμε από τέτοια αφαιρετικά επίπεδα στο ζητούμενο netlist ακολουθούνται μια σειρά από διαδικασίες :

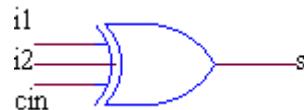
**Διαδικασία της σύνθεσης (synthesis) :** Σκοπός της διαδικασίας αυτής είναι να μετατρέψει ένα αφηρημένο μοντέλο κυκλώματος συχνά περιγεγραμμένο με κάποια γλώσσα περιγραφής κυκλωμάτων σε κύκλωμα μιας γενικευμένης τεχνολογίας (generic technology) ή σε κύκλωμα της συγκεκριμένης τεχνολογίας. Η διαδικασία αυτή πραγματοποιείται από ένα πρόγραμμα γνωστό ως συνθέτης (synthesis tool). Ο συνθέτης δεν είναι ικανός να παρέχει ένα μόνο κύκλωμα στη στοχευόμενη τεχνολογία. Είναι ικανός να παράγει διάφορα κυκλώματα τα οποία προφανώς υλοποιούν τις στοχευόμενες συναρτήσεις, αλλά το καθένα έχει διαφορετικά χαρακτηριστικά ταχύτητας λειτουργίας, εμβαδού υλοποίησης ή κατανάλωσης ισχύος. Ο σχεδιαστής είναι εκείνος που θα κατευθύνει τη διαδικασία της σύνθεσης και θα θέσει στον συνθέτη τους απαραίτητους στόχους – περιορισμούς (constraints) τους οποίους θα προσπαθήσει ο συνθέτης να εκπληρώσει. Μάλιστα τις περισσότερες φορές ο συνθέτης όχι μόνο θα εκπληρώσει τους περιορισμούς του σχεδιαστή αλλά θα τους υπερκεράσει, αφήνοντας έτσι ένα περιθώριο λάθους στις εκτιμήσεις του που είναι απαραίτητο μιας και ο σχεδιασμός δεν έχει πάρει ακόμη τη τελική φυσική του μορφή.

**Διαδικασία γέννησης Προκαθορισμένων Συναρτήσεων (Macro Block Generation) :** Τα περισσότερα σύγχρονα E-CAD πακέτα παρέχουν την δυνατότητα της αυτόματης παραγωγής μερικών ευρέως χρησιμοποιούμενων σχεδιασμών (adders / subtractors, multipliers / dividers, registers, shifters, ALUs, memories, UARTs, ...) μέσω γεννητόρων (generators).

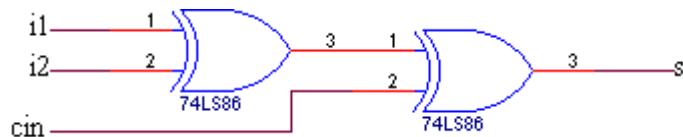
**Διαδικασία εξομάλυνσης του σχεδιασμού (Flattening) :** Κατά την διαδικασία αυτή αναλύεται όλη η ιεραρχία του σχεδιασμού και οι ιεραρχικοί σχεδιασμοί εξομαλύνονται με σκοπό να παραχθεί ένα netlist χωρίς καθόλου ιεραρχικές αναφορές. Στο παρόντο μας με τον αθροιστή, η διαδικασία της εξομάλυνσης θα παράγει δύο αντίτυπα του κυκλώματος γενικευμένης τεχνολογίας για τον αθροιστή των 2 δυαδικών ψηφίων, 4 αντίτυπα για τον αθροιστή των 4 δυαδικών ψηφίων και τελικά 8 αντίτυπα για τον αθροιστή των 8 δυαδικών ψηφίων.

**Διαδικασία αντιστοίχησης του κυκλώματος στην συγκεκριμένη τεχνολογία (Mapping) :** Στόχος αυτής της διαδικασίας είναι η μετάφραση των κομματιών

του κυκλώματος που είναι εκφρασμένα με στοιχειώδης σχεδιασμούς της γενικευμένης τεχνολογίας σε αντίστοιχους σχεδιασμούς της συγκεκριμένης τεχνολογίας. Η διαδικασία αυτή δεν απαιτείται εφόσον τόσο ο συνθέτης όσο και ο γεννήτορας του E-CAD είναι ικανοί να παράγουν το κύκλωμα απευθείας στην συγκεκριμένη τεχνολογία. Στο παρόντειγμά μας ο συνθέτης θα μπορούσε να παράγει το κάτωθι κύκλωμα για την έξιδο αθροίσματος του αθροιστή 1 δυαδικού ψηφίου σε μια γενικευμένη τεχνολογία :



Ωστόσο, όλες οι τεχνολογίες υλοποίησης δεν διαθέτουν XOR πύλες 3 εισόδων. Για παρόντειγμα αυτό ισχύει για όλες τις υποοικογένειες των TTL. Με τη διαδικασία αντιστοίχησης ο παραπάνω στοιχειώδης σχεδιασμός θα μεταφραζόταν στην τεχνολογία Low – Power Schottky TTL ως εξής :



**Διαδικασία συσκευασίας (packaging) :** Η διαδικασία αυτή χρειάζεται να εφαρμοστεί μόνο όταν τελικός στόχος μας είναι μια πλακέτα και μόνο για ορισμένα στοιχειώδη κομμάτια του σχεδιασμού. Για παρόντειγμα, μπορεί ο σχεδιασμός μας να απαιτεί 6 πύλες AND 2 εισόδων τεχνολογίας LS TTL. Οπως γνωρίζετε από τα Εργαστήρια Λογικού Σχεδιασμού όμως, τα διαθέσιμα ολοκληρωμένα αυτής της τεχνολογίας περιέχουν το καθένα 4 πύλες AND. Συνεπώς θα πρέπει να μεσολαβήσει μια διαδικασία αντιστοίχησης μεταξύ των 6 ιδεατών πυλών που χρησιμοποιήσαμε ήταν τον σχεδιασμό μας και των 8 πυλών που προσφέρονται από δύο ολοκληρωμένα TTL 74LS08. Η διαδικασία αυτή μπορεί να γίνεται εντελώς αυτόματα από το E-CAD εργαλείο ή να παρεμβάλλεται ο σχεδιαστής αν και όπου αυτός θεωρεί απαραίτητο.

Μετά την ολοκλήρωση των παραπάνω διαδικασιών το δημιουργηθέν netlist του κυκλώματος, περιγράφει με απόλυτη ακρίβεια το στοχευόμενο σύστημα στην συγκεκριμένη τεχνολογία υλοποίησης. Η επεξεργασία αυτού του netlist κατά την δεύτερη φάση του σχεδιασμού περιλαμβάνει τα στάδια της διάταξης στο χώρο (placement), της διασύνδεσης (routing), της εξαγωγής των φυσικών μεγεθών (extraction), της επαλήθευσης (verification) και της εξαγωγής των αρχείων κατασκευής.

---

Κατά τη διαδικασία της τοποθέτησης στο χώρο, σκοπός μας είναι να τοποθετήσουμε τα διάφορα στοιχειώδη κομμάτια που αποτελούν το σχεδιασμό μας με στόχους :

Ο συνολικός σχεδιασμός μας να καταλαμβάνει το δυνατόν μικρότερο φυσικό χώρο.

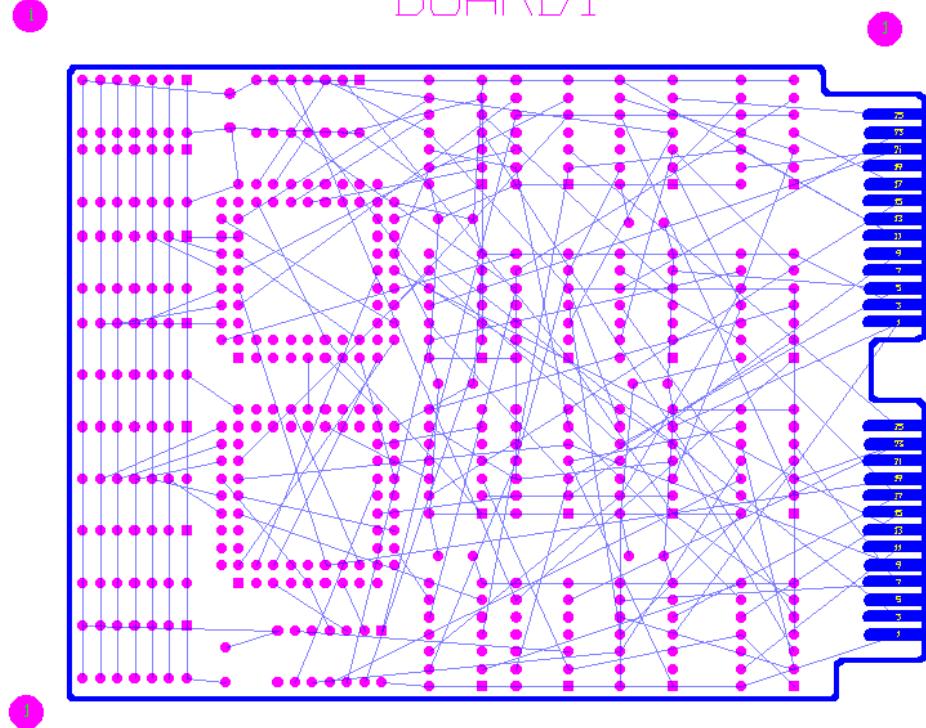
- ♦ Να μπορεί να γίνει απρόσκοπτα η διασύνδεση μεταξύ των στοιχειωδών κομματιών αν και όπου απαιτείται αυτή σύμφωνα με τα δεδομένα του netlist.
- ♦ Να εξασφαλίσουμε ότι διασυνδεδεόμενα κομμάτια του σχεδιασμού μας θα βρίσκονται σε μικρή απόσταση το ένα από το άλλο και συνεπώς οι γραμμές διασύνδεσής τους θα έχουν μικρή αντίσταση και θα εισάγουν μικρή καθυστέρηση και τον ελάχιστο δυνατό θόρυβο. Για παράδειγμα οι φυσικές υλοποιήσεις των αθροιστών ενός δυαδικού ψηφίου θα πρέπει να βρίσκονται γειτονικά στο χώρο.
- ♦ Να διατάξουμε τα υποσυστήματα του σχεδιασμού μας έτσι ώστε να διευκολύνουμε την διασύνδεσή του με άλλα συστήματα. Για το παράδειγμά μας, θα συγκεντρώναμε όλη την απαραίτητη λογική για την διασύνδεση με την αρτηρία στην ίδια πλευρά της πλακέτας ή του ολοκληρωμένου.

Εχει αποδειχθεί ότι το πρόβλημα της εύρεσης της βέλτιστης διάταξης στο χώρο είναι NP-complete. Τα περισσότερα E-CAD προγράμματα χρησιμοποιούν κάποιον ευριστικό (heuristic) αλγόριθμο για να προσεγγίσουν κάποια αξιόλογη λύση. Ωστόσο, τις περισσότερες φορές απαιτείται η παρέμβαση του σχεδιαστή για να εκπληρωθούν οι παραπάνω στόχοι. Το υποπρόγραμμα του E-CAD που εκτελεί τη διαδικασία της διάταξης στο χώρο ονομάζεται χωροδιατάκτης (placer).

Η έξοδος της διάταξης στο χώρο στην περίπτωση ενός PCB είναι ένας διατεταγμένος φυσικός σχεδιασμός, όπως για παράδειγμα αυτός του σχήματος που ακολουθεί. Στο σχήμα φαίνονται μόνο τα αποτυπώματα (foot prints) των ολοκληρωμένων καθώς και οι διασυνδέσεις τους οι οποίες δεν έχουν πάρει ακόμη την τελική φυσική μορφή τους.

Με τη διαδικασία της διασύνδεσης (routing), σκοπός μας είναι να διασυνδέσουμε με φυσικές γραμμές μετάλλου τα στοιχειώδη κομμάτια του σχεδιασμού μας έτσι ώστε να προκύψει η περιγραφόμενη από το netlist λειτουργικότητα. Η διασύνδεση θα πρέπει να λαμβάνει υπόψη της διάφορους περιορισμούς, όπως για παράδειγμα τον αριθμό των επιπέδων διασύνδεσης, το ελάχιστο πλάτος που θα πρέπει να έχει κάθε γραμμή καθώς και τον περιορισμένο χώρο μέσα στον οποίο θα πρέπει να ολοκληρωθεί.

## BOARD1

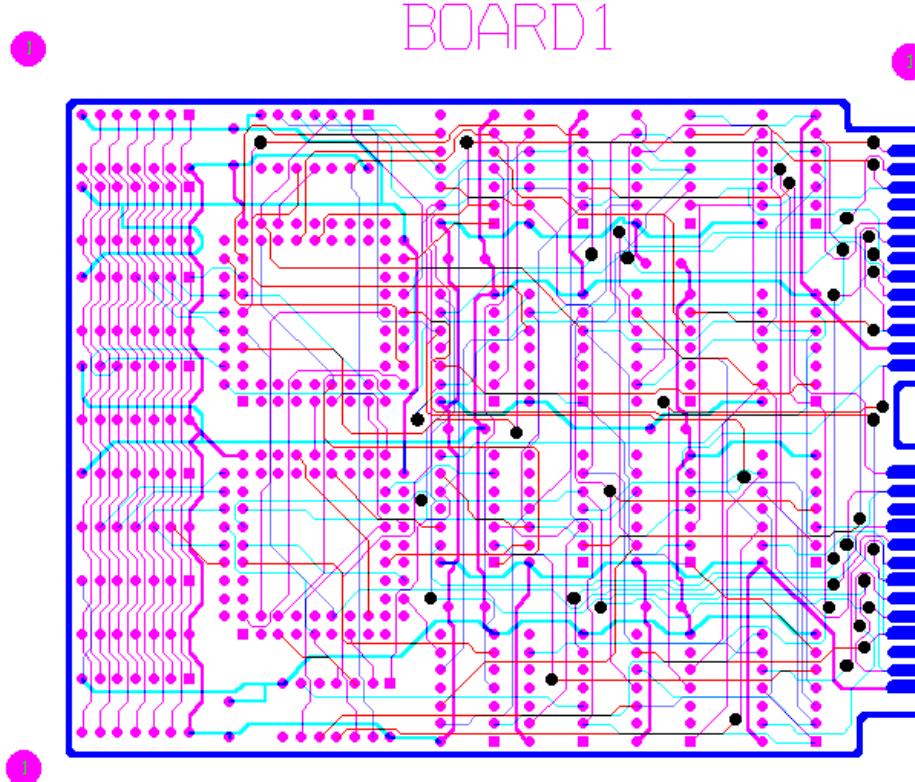


Προφανώς το πρόβλημα της διασύνδεσης για έναν σχεδιασμό σε PCB είναι σημαντικά δυσκολότερο από το αντίστοιχο ενός σχεδιασμού που πρόκειται να εμφωλευτεί σε ένα ολοκληρωμένο γρήγορης πρωτότυποποίησης. Στην πρώτη περίπτωση δεν υπάρχουν κάποια φυσικά μονοπάτια. Θα πρέπει να δημιουργηθούν από το πρόγραμμα διασύνδεσης (router). Στην δεύτερη περίπτωση υπάρχει ένα μεγάλο σύνολο φυσικών μονοπατιών (διασυνδέσεων) εκ των οποίων εμείς καλούμαστε να επιλέξουμε ποια θέλουμε να ενεργοποιηθούν και ποια όχι.

Τα περισσότερα E-CAD προγράμματα για PCB και για την περίπτωση της διασύνδεσης, χρησιμοποιούν κάποιον ευριστικό (heuristic) αλγόριθμο για να προσεγγίσουν κάποια αξιόλογη λύση και τις περισσότερες φορές κατορθώνουν να χαράξουν τα απαιτούμενα φυσικά μονοπάτια χωρίς την παρέμβαση του σχεδιαστή. Η ποιότητα ωστόσο της προκύπτουσας διασύνδεσης (όπως καθορίζεται από το συνολικό μήκος όλων των γραμμών και τις αποστάσεις ασφαλείας μεταξύ τους για να αποφευχθούν φαινόμενα crosstalk) δεν είναι πάντοτε η βέλτιστη. Αρκετές φορές χρειάζονται αρκετά περάσματα (routing passes) πάνω από τον διατεταγμένο σχεδιασμό, ώστε να επιτύχουμε υψηλό ποσοστό επιτυχούς διασύνδεσης (routing completion percentage) και μακροποιητική ποιότητα διασύνδεσης. Κάτω από αυτές τις δυσκολίες και δεδομένου του υψηλού υπολογιστικού φόρτου της διαδικασίας διασύνδεσης, ο σχεδιαστής αναγκάζεται να σταματήσει την διαδικασία διασύνδεσης όταν τα ποσοστά ολοκλήρωσής της ξεπεράσουν το 95 %. Αρκετές φορές θα κληθεί να

περάσει κάποιες γραμμές μόνος του (manual routing) καθώς και να εξωραΐσει τα αποτελέσματα (beautification) της αυτόματης διαδικασίας διασύνδεσης (automated routing).

Η έξοδος της διασύνδεσης είναι ένας πλήρως συνδεδεμένος φυσικός σχεδιασμός (layout του κυκλώματος), όπως για παράδειγμα αυτός του επόμενου σχήματος, ο οποίος αντιστοιχεί στον σχεδιασμό του οποίου τη τοποθέτηση είδαμε προηγούμενα.



Εχοντας πλέον ολοκληρώσει τον φυσικό σχεδιασμό του συστήματός μας, θέτουμε σαν στόχο την επαλήθευσή του τόσο από λογικής όσο και από χρονικής πλευράς. Ο σκοπός του σταδίου αυτού είναι να εξάγουμε από τον φυσικά υλοποιημένο σχεδιασμό όλα τα απαραίτητα μεγέθη που θα κάνουν τα χρονικά μοντέλα που χρησιμοποιούμε για την εξομοίωση του κυκλώματός μας το δυνατόν πιο ακριβή.

Εχοντας πια τις απαραίτητες πληροφορίες περί του μήκους αλλά και του πλάτους κάθε γραμμής διασύνδεσης, και διαθέτοντας μοντέλα για την αντίσταση και την χωρητικότητα γραμμών διασύνδεσης θεμελιωδών μεγεθών, μπορούμε με σχετική ακρίβεια να υπολογίσουμε τις τιμές για τις γραμμές διασύνδεσης που πραγματικά χρησιμοποιήθηκαν. Επιπλέον είναι γνωστό ότι η καθυστέρηση διάδοσης μεταξύ της αλλαγής μιας εισόδου και της αλλαγής της εξόδου δεν είναι πάντα η ίδια για όλες τις εισόδους του κυκλώματος ακόμη και όταν το κύκλωμα είναι ένας στοιχειώδης σχεδιασμός. Για παράδειγμα ακόμη και οι δύο εισόδοι μιας πύλης AND δεν είναι ακριβώς ισοδύναμες σε ότι αφορά την χρονική καθυστέρηση διάδοσης μεταβάσεων

---

πάνω σε αυτές, μιας και παρουσιάζουν διαφορετική χωρητικότητα οδήγησης. Τέλος είναι γνωστό ότι η καθυστέρηση κάθε σχεδιασμού είναι άμεση συνάρτηση του φορτίου που θα πρέπει να οδηγηθεί από τις εξόδους του. Γνωρίζοντας με ακρίβεια τον αριθμό των πυλών που κάθε πύλη του σχεδιασμού μας πρέπει να οδηγήσει, την χωρητικότητα της συγκεκριμένης εισόδου κάθε οδηγούμενης πύλης και την αντίσταση και χωρητικότητα των γραμμών διασύνδεσης, μπορούμε να καταλήξουμε σε μοντέλα που περιγράφουν με πολύ μεγαλύτερη ακρίβεια την χρονική συμπεριφορά του σχεδιασμού μας.

Ολα τα παραπάνω είναι μεγέθη, τα οποία υπολογίζονται από ένα υποπρόγραμμα του E-CAD εργαλείου μας το οποίο ονομάζουμε εξαγωγέα (extractor). Συνήθως οι τιμές των μεγεθών αυτών αποθηκεύονται σε ένα αρχείο (annotation file, Standard Delay File – SDF) με κάποιο τυποποιημένο τρόπο. Το δεύτερο βήμα είναι να επαυξήσουμε τα χρονικά μοντέλα που χρησιμοποιήσαμε για την χρονική εξομοίωση χρησιμοποιώντας την πληροφορία του εξαγωγέα με τη διαδικασία της προς τα πίσω ενημέρωσης (back – annotation) του σχεδιασμού μας.

Αξίζει να αναφέρουμε ότι ο εξαγωγέας κάποιων προηγμένων (και ταυτόχρονα πολύ ακριβών) E-CAD εργαλείων, μπορεί να μας παρέχει και άλλες σημαντικές πληροφορίες. Για παράδειγμα μπορούμε να ζητήσουμε πληροφορίες για τα θερμικά μεγέθη του σχεδιασμού μας με σκοπό σε επόμενο στάδιο να προχωρήσουμε σε θερμική εξομοίωση και να εντοπίσουμε σημεία στα οποία θα πρέπει να παρεμβάλλουμε μονάδες απαγωγής θερμότητας.

Η επαλήθευση του σχεδιασμού μας (verification) είναι το τελευταίο στάδιο ελέγχου του σχεδιασμού μας πριν την διαδικασία κατασκευής του συστήματός μας. Στόχοι του σταδίου αυτού είναι να επιβεβαιώσει τον σχεδιαστή :

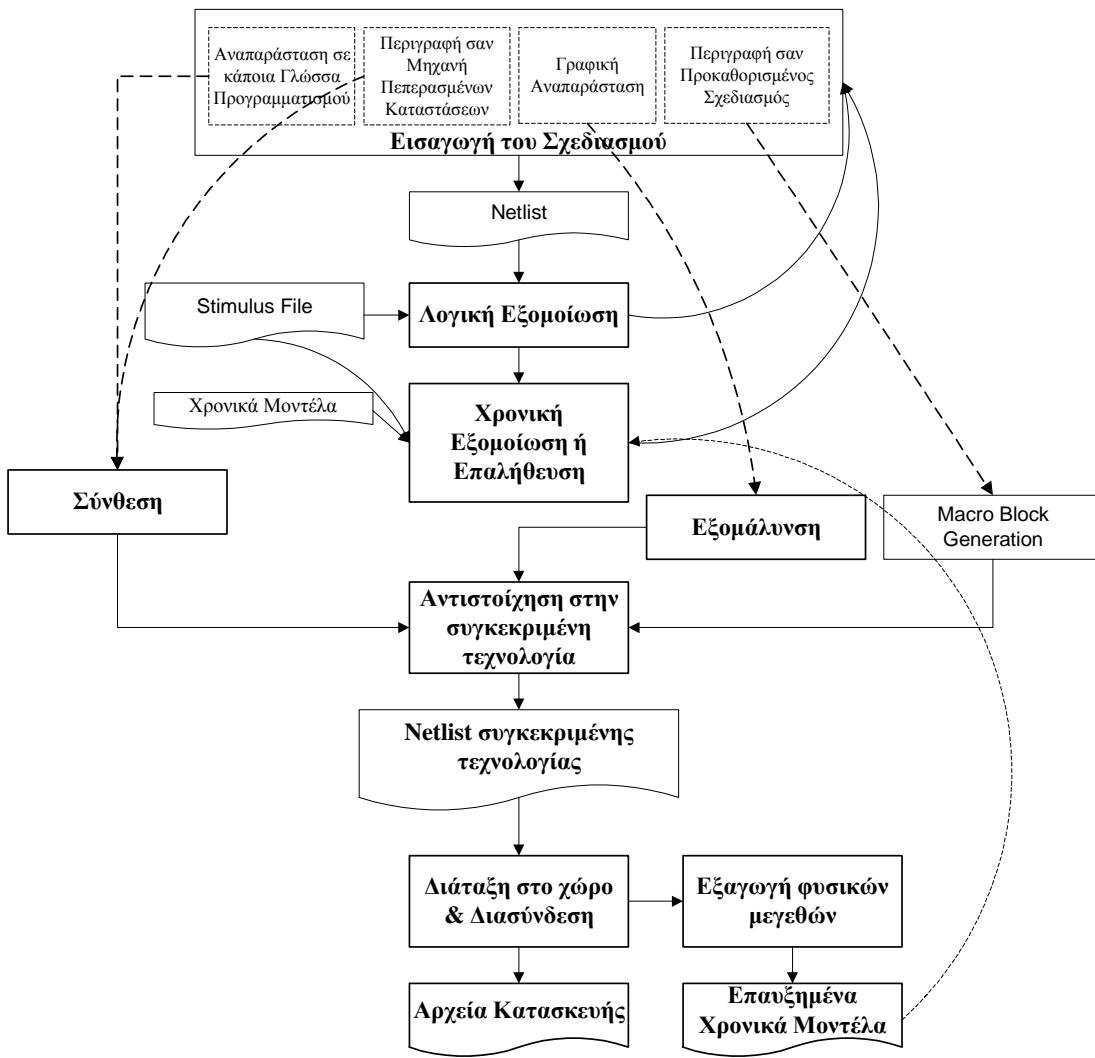
- ♦ Οτι δεν έχει γίνει καμία αλλοίωση της αρχικής πληροφορίας από τα προγράμματα που χρησιμοποιήθηκαν για τη δεύτερη φάση της ροής του σχεδιασμού μας. Σημειώνουμε ότι λόγω του μεγέθους και της πολυπλοκότητας αυτών των προγραμμάτων και κάτω από ακραίες καταστάσεις συχνά έχουν παρατηρηθεί φαινόμενα αλλοίωσης της πληροφορίας. Ο έλεγχος γίνεται με τη σύγκριση δύο netlist που στην ουσία είναι οι αναπαραστάσεις του σχεδιασμού μας στα διαφορετικά επίπεδα αφαιρεσης. Συνήθως η σύγκριση γίνεται μεταξύ του netlist που χρησιμοποιήσαμε για τη λογική και χρονική εξομοίωση του σχεδιασμού μας και ενός που παράγεται με είσοδο τον φυσικό σχεδιασμό (το κύκλωμα δηλαδή μετά τις διαδικασίες placement και routing). Αυτή η σύγκριση συνήθως αναφέρεται σαν LVS (Layout vs. Schematic).

- 
- ♦ Οτι ο φυσικός σχεδιασμός ανταποκρίνεται στις χρονικές προδιαγραφές που ετέθησαν από την αρχή. Η διαδικασία που ακολουθείται είναι η χρονική εξομοίωση με είσοδο το netlist του κυκλώματός μας και τα επαυξημένα χρονικά μοντέλα τα οποία μας παρείχε ο εξαγωγέας του εργαλείου για τις χειρότερες περιπτώσεις χρονοκαθυστέρησης.

Αφότου έχουμε περάσει επιτυχώς το στάδιο της επαλήθευσης, για την κατασκευή του σχεδιασμού μας, αρκεί να εξάγουμε κάποια αρχεία κατασκευής. Στην περίπτωση των PLDs αυτό είναι στην ουσία το αρχείο προγραμματισμού της διάρθρωσης του PLD. Στην περίπτωση των πλακετών, θα πρέπει να σημειώσουμε ότι υπάρχουν διεθνείς τυποποιημένες γλώσσες περιγραφής αυτών των αρχείων κατασκευής. Σε αυτή τη περίπτωση, πέρα από αυτά τα αρχεία, καλό επίσης θα ήταν να χρησιμοποιήσουμε το E-CAD εργαλείο για να εξάγουμε μία λίστα από τα υλικά (Bill of Materials - BOM) που απαιτούνται για τη συναρμολόγηση κάθε πρωτότυπου. Επιπλέον, ο κάθε κατασκευαστής για να μπορέσει να μας απαντήσει θετικά για το αν μπορεί να κατασκευάσει την πλακέτα που σχεδιάσαμε (αν δηλαδή οι επιλογές μας ταιριάζουν με τη διαθέσιμη γραμμή κατασκευής) και να τιμολογήσει το κόστος κατασκευής της, θα ζητήσει διάφορες πληροφορίες, όπως :

- ♦ Ποιο είναι το ελάχιστο πλάτος γραμμής που χρησιμοποιήσαμε.
- ♦ Τι είδους packages ολοκληρωμένων χρησιμοποιούμε (DIP, SMD, ...)
- ♦ Αν απαιτούμε οι τρύπες υποδοχής των ολοκληρωμένων να είναι επιχρυσωμένες ή επαργυρωμένες και πόσες είναι αυτές,
- ♦ Πόσες διαφορετικές διαμέτρους τρυπών χρησιμοποιούμε κλπ.

Συνοπτικά θα μπορούσαμε να εκφράσουμε τη ροή του σχεδιασμού με το flowchart του επόμενου σχήματος. Προσέξτε ότι στο σχήμα υπάρχουν αρκετές διαδικασίες οι οποίες χρειάζεται να επαναληφθούν σε περιπτώσεις λαθών. Για παράδειγμα αν στη λογική εξομοίωση προκύψουν λογικά λάθη στο σχεδιασμό μας, θα πρέπει να αλλάξουμε τη λογική ώστε να εκτελεί σωστά όλες τις συναρτήσεις που προδιαγράφηκαν. Το ίδιο μπορεί να συμβεί και όταν χρησιμοποιώντας τα επαυξημένα χρονικά μοντέλα μετά τη δεύτερη φάση του σχεδιασμού, προκύψει ότι πλέον ο σχεδιασμός δεν ικανοποιεί τις χρονικές προδιαγραφές που τέθηκαν.





---

## Κεφάλαιο 5

# Γλώσσες περιγραφής υλικού (Hardware Description Languages)

Στο κεφάλαιο αυτό γίνονται αναφορές στις δύο πιο διαδεδομένες γλώσσες περιγραφής υλικού. Οι αναφορές έχουν σα σκοπό να σας εντάξουν στη φιλοσοφία της γλώσσας και όχι να αποτελέσουν ένα πλήρες εγχειρίδιο. Ο ενδιαφερόμενος για περισσότερες λεπτομέρειες αναγνώστης ωστόσο θα μπορέσει να βρει παραπομπές σε αναγνωρισμένα πλήρη εγχειρίδια των γλωσσών περιγραφής υλικού στο τέλος του βιβλίου.

Οι περιορισμοί του γραφικού τρόπου εισαγωγής σχεδιασμών, οδήγησαν στην ανάπτυξη των γλωσσών περιγραφής υλικού (Hardware Description Languages – HDLs). Σήμερα υπάρχουν δύο καθιερωμένες τέτοιες γλώσσες : η VHDL και η Verilog. Και οι δύο γλώσσες έχουν μια παρόμοια φιλοσοφία σχεδιασμού και αναπαράστασης ενός κυκλώματος σε αυτές. Η VHDL που κληρονομεί τη φιλοσοφία της Ada (μία από τις πρώτες γλώσσες προγραμματισμού και πρόγονος της Pascal)

---

έχει πολύ πιο αυστηρές δομές. Θεωρείται ως η πλέον κατάλληλη για μεγάλους σχεδιασμούς μιας και προσφέρει μικρές επιπλέον δυνατότητες από ότι η Verilog. Επιπλέον, είναι στις μέρες μας το standard που έχει νιοθετήσει η βιομηχανία. Από την άλλη πλευρά, η Verilog θεωρείται πολύ πιο εύκολη στην εκμάθηση (καθώς μοιάζει αρκετά με την C) και διαρκώς κερδίζει έδαφος σε μεγάλους οίκους σχεδιασμού. Στην συνέχεια εξετάζουμε συνολικά οι δύο αυτές γλώσσες με κύριο σκοπό να αναδειχθεί η φιλοσοφία σχεδιασμού με την χρήση οποιασδήποτε από αυτές, χωρίς να υπεισερχόμαστε σε λεπτομέρειες. Ο ενδιαφερόμενος αναγνώστης μπορεί να βρει μια πλήρη και τυπική περιγραφή κάθε μίας από τις δύο γλώσσες συμβουλευόμενος τις αναφορές. Θα πρέπει να επισημανθεί ότι κατά καιρούς έχουν παρουσιαστεί διάφορες άλλες γλώσσες για περιγραφή υλικού, χωρίς ωστόσο να κερδίσουν σοβαρό μερίδιο της προσοχής των σχεδιαστών παγκοσμίως (π.χ. ABEL HDL). Πρόσφατα (τέλη 1997) παρουσιάστηκαν οι πρώτες ερευνητικές προσπάθειες για συσχέτιση γλωσσών προγραμματισμού υψηλού επιπέδου και γλωσσών περιγραφής υλικού. Οι προσπάθειες αυτές κινούνται προς δύο διαφορετικές κατευθύνσεις :

1. Κατάλληλη τροποποίηση των δομών των γλωσσών προγραμματισμού ώστε να μπορούν να περιγράφουν τις παραλληλες δομές του υλικού (π.χ. Handel C της εταιρείας Embedded Solutions).
2. Ανάπτυξη μεταφραστών από γλώσσες προγραμματισμού υψηλού επιπέδου σε γλώσσες περιγραφής υλικού (π.χ. C2Verilog της εταιρείας CompiLogic).

Τέτοιες προσπάθειες αναμένεται να τύχουν ευρείας αποδοχής στο μέλλον, κυρίως λόγω :

1. Της μεγάλης υπάρχουσας βάσης ατόμων ικανών να προγραμματίζουν σε γλώσσες υψηλού επιπέδου,
2. Γιατί μπορούν να αποτελέσουν κοινή πλατφόρμα συνεννόησης για σχεδιαστικές ομάδες με διαφορετικούς στόχους (π.χ. σχεδιασμός λογισμικού και υλικού του ίδιου συστήματος).

Ας δούμε πρώτα μερικές αρχές κοινές και για τις δύο γλώσσες. Η πρώτη αρχή είναι αυτή της παραλληλίας. Η παραλληλία είναι απαραίτητη στις HDL για τη μοντελοποίηση της παραλληλης εκτέλεσης που συμβαίνει στο υλικό. Αυτή η παραλληλία ανατρέπει βασικά πράγματα που έχουμε μάθει κατά τη συγγραφή των προγραμμάτων. Για παράδειγμα, ας θεωρήσουμε μια ιλασσική γλώσσα προγραμματισμού. Η σειρά αναγραφής των εντολών σε μια τέτοια γλώσσα έχει σημασία μιας και καθορίζει και τη σειρά εκτέλεσης. Για παράδειγμα, η σειρά αναγραφής των αναθέσεων  $A=B$  και  $C=A$  έχει σημασία. Αν αναγραφούν με τη σειρά που φαίνεται, στο τέλος της εκτέλεσης το C θα έχει πάρει τη νέα τιμή του A, δηλαδή

---

το B. Αν όμως αναγραφούν με σειρά C=A και A=B, το C θα έχει πάρει τη παλιά τιμή του A. Αντίθετα σε μια HDL η σειρά αναγραφής των εντολών δεν έχει καμιά σημασία μιας και όλες οι εντολές της γλώσσας εκτελούνται παράλληλα. Συνεπώς είτε με τον ένα είτε με τον άλλο τρόπο αναγράψουμε τις εντολές το αποτέλεσμα θα είναι το C να πάρει στο τέλος της εκτέλεσης τη νέα τιμή του A, δηλαδή το B.

Η παραλληλία εκτέλεσης στις HDL, διακόπτεται μόνο από ειδικές δομές. Η εκτέλεση εντός αυτών των δομών είναι ακολουθιακή. Οι δομές αυτές συνήθως χρησιμοποιούνται για να περιγράψουν στοιχεία μνήμης.

Μια δεύτερη αρχή των HDL είναι η αρχή της συνθεσιμότητας. Οποιοδήποτε συντακτικά σωστό πρόγραμμα γραμμένο σε μία HDL μπορεί να εξομοιωθεί, δηλαδή να αποτελέσει είσοδο σε έναν εξομοιωτή. Αντίθετα όμως ένα συντακτικά σωστό πρόγραμμα δεν είναι απαραίτητα ικανή είσοδος για έναν συνθέτη. Ενα πολύ μικρό υποσύνολο των δυνατών συντακτικά σωστών προγραμμάτων, μπορεί να αποτελέσει σωστή είσοδο σε ένα εργαλείο σύνθεσης. Το παραπάνω μπορεί να εξηγηθεί με ένα παράδειγμα. Υποθέστε την παρακάτω εντολή σε γλώσσα Verilog :

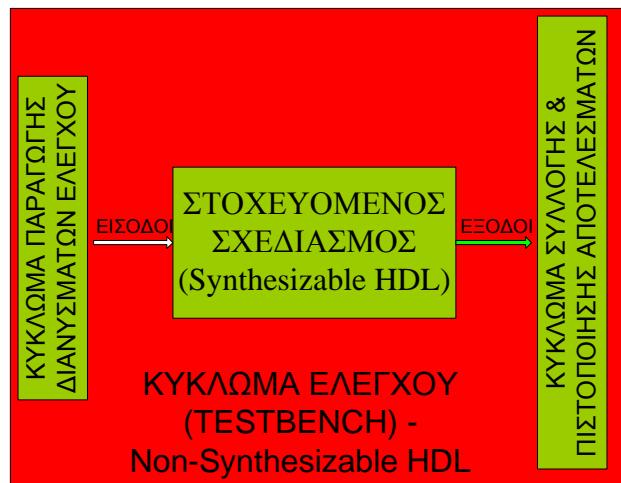
```
always #20 clk = ~clk
```

που ορίζει ότι η μεταβλητή clk κάθε 20 χρονικές στιγμές θα αλλάζει τιμή. Προφανώς η εντολή αυτή στον εξομοιωτή καθορίζει ένα σήμα ρολογιού με περίοδο 40 χρονικών μονάδων και συνεπώς είναι εξομοιώσιμη. Όμως παράλληλα αυτή η εντολή δε περιγράφει σε καμμία περίπτωση προς το εργαλείο σύνθεσης το πως είναι φτιαγμένο αυτό το ρολόι. Το εργαλείο σύνθεσης θα έπρεπε συνεπώς να είναι σε θέση να συνθέτει ρολόγια με περίοδο 40 ns, 40 ps ή και 40 αιώνων. Μιας που εμείς οι ίδιοι δεν είμαστε σε θέση να φτιάχνουμε τέτοια ρολόγια είναι αδύνατο να θέλουμε να τα φτιάχνει το πρόγραμμα σύνθεσης.

Αποτέλεσμα αυτής της αρχής είναι να μπορούμε με HDL να περιγράψουμε (δες και το παρακάτω σχήμα) :

- ◆ Τον στοχευόμενο σχεδιασμό. Για να μπορέσει αυτή η περιγραφή να αποτελέσει είσοδο στο πρόγραμμα σύνθεσης, είναι απαραίτητο να είναι κατάλληλα γραμμένη (synthesizable).
- ◆ Το κύκλωμα παραγωγής διανυσμάτων εισόδου και συλλογής – πιστοποίησης των αποκρίσεων του στοχευόμενου σχεδιασμού (κύκλωμα testbench). Το κύκλωμα αυτό προσομοιώνει το breadboard που χρησιμοποιούσατε στα εργαστήρια ψηφιακής σχεδιασης, δηλαδή ένα κύκλωμα που γεννά τις εισόδους και διαβάζει τις εξόδους του στοχευόμενου σχεδιασμού. Προφανώς αυτός ο σχεδιασμός δεν αποτελεί κομμάτι του στοχευόμενου σχεδιασμού και φυσικά δεν θα αποτελέσει

είσοδο στο εργαλείο της σύνθεσης. Ως εκ τούτου δεν υπάρχει ανάγκη να είναι synthesizable κώδικας.



Ποιοι όμως είναι οι κανόνες ώστε μια περιγραφή να είναι συνθέσιμη ? Συνθέσιμες είναι όλες οι δομικές (structural) περιγραφές καθώς και εκείνες από τις περιγραφές της συμπεριφοράς (behavioral) –οι έννοιες των structural και behavioral περιγραφών αναλύονται παρακάτω— του κυκλώματος που :

1. Περιγράφουν τη συνδυαστική μας λογική και
2. Περιγράφουν τις περιπτώσεις αλλαγών των ακολουθιακών στοιχείων του κυκλώματός μας.

Οι περιγραφές αυτές μιας και περιορίζονται στο να περιγράφουν τη ροή των σημάτων μεταξύ των στοιχείων μνήμης του κυκλώματός μας είναι γνωστές με την επωνυμία RTL (Register Transfer Level) περιγραφές.

Παρακάτω αναλύουμε αυτές τις έννοιες και τη φιλοσοφία περιγραφής ενός κυκλώματος με τις HDL. Ξεκινάμε πρώτα από την VHDL μιας που οι αυστηρές δομές της είναι ένα καλό εκπαιδευτικό εργαλείο.

Η VHDL (VHSIC –Very High Speed Integrated Circuit– Hardware Description Language) αναπτύχθηκε γύρω στα τέλη της δεκαετίας του 1970 και τις αρχές της δεκαετίας του 1980. Προτάθηκε σε κάποια τυποποιημένη μορφή το 1981 και έγινε standard του IEEE το 1986 με κωδικό 1076. Ακολούθησαν τροποποιήσεις για να ξεπεραστούν κάποιοι περιορισμοί και σήμερα ευρέως χρησιμοποιείται το δεύτερο standard της γλώσσας, το IEEE 1164. Η VHDL είναι μια γλώσσα με πολύ αυστηρές δομές και αρκετά δύσκολη στην εκμάθηση. Εμπεριέχει δομές τόσο για την αναπαράσταση του σχεδιασμού σε όλα τα επίπεδα αφαιρεσης, όσο και για την λογική και χρονική εξομοίωσή του. Οι βασικοί όροι που χρησιμοποιούνται κατά την περιγραφή ενός σχεδιασμού σε VHDL είναι :

1. Οντότητες (Entities) : Κάθε είδους σχεδιασμός εκφράζεται σαν οντότητα και αποτελεί ένα βασικό κομμάτι σχεδιασμού. Το πιο υψηλό επίπεδο ιεραρχίας είναι και η υψηλότερη οντότητα (top-level entity). Η περιγραφή μιας οντότητας υψηλού βαθμού ιεραρχίας μπορεί να περιέχει περιγραφές άλλων οντοτήτων χαμηλότερης ιεραρχίας. Για παράδειγμα, ο κώδικας VHDL που χρειάζεται για τη δήλωση μιας οντότητας για τον αθροιστή ενός δυαδικού ψηφίου είναι ο ακόλουθος :

```
ENTITY one_bit_full_adder IS
PORT (
    i1, i2, cin : IN BIT;
    s, cout : OUT BIT
);
END one_bit_full_adder;
```

Αναπαριστούμε τις δεσμευμένες λέξεις (keywords) της VHDL με κεφαλαία γράμματα, και όλο τον υπόλοιπο κώδικα με μικρά γράμματα. Η έννοια της οντότητας στην VHDL είναι ταυτόσημη με την έννοια του γραφικού συμβόλου στην γραφική αναπαράσταση. Με τον παραπάνω κώδικα δηλώνουμε μια καινούργια οντότητα με τρεις εισόδους και δύο εξόδους του ενός δυαδικού ψηφίου η οποία μετατρέπει την προσαρμογή της με τον υπόλοιπο σχεδιασμό. Ορίζονται δηλαδή ο αριθμός, η κατεύθυνση και το είδος κάθε σημείου εισόδου / εξόδου με την συγκεκριμένη οντότητα.

2. Αρχιτεκτονικές (Architectures) : Κάθε αρχιτεκτονική εκφράζει την συμπεριφορά μιας οντότητας. Μια οντότητα μπορεί να έχει διάφορες αρχιτεκτονικές που εκφράζουν τον σχεδιασμό της οντότητας σε διάφορα επίπεδα αφαίρεσης. Η σελίδα σχηματικού που επεξηγεί την λειτουργία του πλήρους αθροιστή ενός δυαδικού ψηφίου αντιστοιχεί με μία αρχιτεκτονική για την παραπάνω οντότητα. Για παράδειγμα για το κύκλωμα του πλήρους αθροιστή ενός δυαδικού ψηφίου, θα μπορούσαμε να ορίσουμε την ακόλουθη αρχιτεκτονική :

```
ARCHITECTURE structural OF one_bit_full_adder IS
    COMPONENT xor2
        PORT (a, b :IN BIT;
              c : OUT BIT);
    END COMPONENT;
    COMPONENT and2
        PORT (a, b :IN BIT;
```

---

```

        c : OUT BIT);
END COMPONENT;
COMPONENT or3
    PORT (a, b, c :IN BIT;
          d : OUT BIT);
    END COMPONENT;
SIGNAL temp1, temp2, temp3, temp4 : BIT;
BEGIN
    U1: xor2
        PORT MAP (i1, i2, temp1);
    U2: xor2
        PORT MAP (cin, temp1, s);
    U3: and2
        PORT MAP (i1, i2, temp2);
    U4: and2
        PORT MAP (i1, cin, temp3);
    U5: and2
        PORT MAP (i2, cin, temp4);
    U6: or3
        PORT MAP (temp2, temp3, temp4, cout);
END structural;

```

Στην παραπάνω αρχιτεκτονική χρησιμοποιούμε αντίτυπα στοιχειωδών σχεδιασμών – πυλών (components) για να περιγράψουμε τη δομή του αθροιστή γι' αυτό και το παραπάνω είδος αρχιτεκτονικής ονομάζεται δομική (structural). Επιπλέον περιγράφουμε την διασύνδεση μεταξύ αυτών των στοιχείων χρησιμοποιώντας τις εισόδους / εξόδους και τοπικές μεταβλητές (που αντιστοιχούν στις γραμμές ένωσης των διαφόρων πυλών του γραφικού τρόπου αναπαράστασης). Θα πρέπει να σημειωθεί ότι όλες οι εντολές μιας αρχιτεκτονικής (όπως περιλαμβάνονται μεταξύ των δεσμευμένων λέξεων BEGIN και END) εκτελούνται παράλληλα.

Οπως είναι προφανές με τη χρήση δομικών αρχιτεκτονικών ο σχεδιαστής ελάχιστα διαφοροποιείται από τον γραφικό τρόπο εισαγωγής του σχεδιασμού. Ενδεχόμενα μάλιστα η προσπάθεια που χρειάζεται να καταβληθεί να είναι σημαντικά μεγαλύτερη και η αναγνωσιμότητα περιορισμένη. Για τον σκοπό αυτό στις γλώσσες περιγραφής υλικού δίνεται η δυνατότητα περιγραφής του υλικού με

---

αναφορά στην συμπεριφορά του (behavioral description). Ο ίδιος σχεδιασμός θα μπορούσε κάλλιστα να περιγραφεί με την κάτωθι behavioral αρχιτεκτονική :

```
ARCHITECTURE behavioral OF one_bit_full_adder IS
BEGIN
    s <= i1 XOR i2 XOR cin;
    cout <=(i1 AND i2) OR (i1 AND cin) OR (i2 AND cin);
END behavioral;
```

Η χρήση του συμβόλου  $<=$  υποδηλώνει την παραλληλία της εκτέλεσης των δύο εντολών της αρχιτεκτονικής. Σε μια αρχιτεκτονική της VHDL δεν υπάρχει η έννοια της σειράς εκτέλεσης για εντολές εκχώρησης, αφού όλες εκτελούνται παράλληλα έτσι ώστε να αντικατοπτρίζουν την έμφυτη παραλληλία στο υλικό. Η σειρά εκτέλεσης αυτών των εντολών καθορίζεται μόνο από αλλαγές πάνω στα σήματα που χρησιμοποιούνται στην εκχώρηση. Για παράδειγμα η εντολή  $s <= i1 \text{ XOR } i2 \text{ XOR } cin$ ; θα εκτελείται οποτεδήποτε συμβεί αλλαγή στην τιμή των σημάτων  $i1$  ή  $i2$  ή  $cin$ . Με άλλα λόγια, η παραπάνω εντολή είναι ευαίσθητη (sensitive) στις αλλαγές των σημάτων που βρίσκονται δεξιά του  $<=$ . Τα σήματα αυτά καταρτίζουν την λίστα ευαίσθησίας (sensitivity list) της συγκεκριμένης εντολής.

Στην παραπάνω εντολή θα μπορούσαμε να προσάψουμε χρονική πληροφορία όπως για παράδειγμα με τον κώδικα :

```
s <= TRANSPORT (i1 XOR i2 XOR cin) AFTER 2ns;
```

Η παραπάνω εντολή υποδεικνύει στο εργαλείο εξομοίωσης ότι θα πρέπει να εκτελέσει την εντολή εκχώρησης 2 ns αργότερα από οποιαδήποτε αλλαγή τιμής πάνω στα σήματα της λίστας ευαίσθησίας της. Η τιμή ωστόσο που θα εκχωρηθεί υπολογίζεται σύμφωνα με τις τρέχουσες τιμές των σημάτων της λίστας ευαίσθησίας και όχι αυτές που τυχόν θα έχουν μετά από 2ns, έτσι ώστε ακόμα και αλλαγές με διάρκεια μικρότερη των 2ns να είναι ορατές στην έξοδο s (transport delay model). Παραλείποντας το keyword TRANSPORT αλλαγές στα σήματα εισόδου με χρονική διάρκεια μικρότερη των 2ns γίνονται αόρατες στην έξοδο (inertial delay model). Το τελευταίο μοντέλο μπορεί να είναι αρκετό για τις περισσότερες περιπτώσεις εξομοίωσης.

Πέρα από τις εντολές ανάθεσης και ολόκληρες διαδικασίες μπορεί να έχουν λίστα ευαίσθησίας. Κάτω από αυτή τη θεώρηση, ένας τρίτος τρόπος περιγραφής της αρχιτεκτονικής του πλήρους αθροιστή ενός δυαδικού ψηφίου θα ήταν ο εξής :

---

```

ARCHITECTURE behavioral2 OF one_bit_full_adder IS
BEGIN
    PROCESS (i1, i2, cin)
        BEGIN
            IF i1='0' AND i2='0' AND cin='0' THEN
                s <= '0';
                cout <= '0';
            ELSEIF i1='0' AND i2='0' AND cin='1' THEN
                s <= '1';
                cout <= '0';
            ELSEIF ...
            ELSEIF i1='1' AND i2='1' AND cin='1' THEN
                s <= '1';
                cout <= '1';
            ENDIF;
        END PROCESS;
    END behavioral2;

```

Η παραπάνω αρχιτεκτονική έχει μόνο μια εντολή, αυτήν της διαδικασίας. Ολες οι υπόλοιπες αποτελούν εντολές της διαδικασίας. Η πιο πάνω διαδικασία έχει την δική της λίστα ευαισθησίας και συνεπώς θα εκτελεστεί αν και μόνο αν υπάρχει αλλαγή στην τιμή κάποιου σήματος της λίστας. Ολες οι εντολές της διαδικασίας εκτελούνται ακολουθιακά, με τον ίδιο δηλαδή τρόπο που θα εκτελούνταν οι εντολές μιας κοινής γλώσσας προγραμματισμού. Αυτό σημαίνει ότι η σειρά αναγραφής αυτών των εντολών EXEI σημασία αφού καθορίζει και την σειρά εκτέλεσής τους και τα αποτελέσματα της εξομοίωσης.

3. Διαρθρώσεις (Configurations) : Μια διάρθρωση αντιστοιχίζει ένα κομμάτι σχεδιασμού σε μία δυάδα οντότητας – αρχιτεκτονικής. Με άλλα λόγια περιγράφει σε ποια οντότητα αναφέρεται κάθε κομμάτι σχεδιασμού και ποια αρχιτεκτονική θα χρησιμοποιηθεί για την συμπεριφορά του.

Μέχρι ώρας έχουμε δει τρεις διαφορετικές αρχιτεκτονικές για τον πλήρη αθροιστή του ενός δυαδικού ψηφίου. Το ποια θα επιλέξει ο σχεδιαστής εξαρτάται από την ακρίβεια που επιθυμεί και από το αν απαιτείται δομική πληροφορία. Εάν η στοχευόμενη τεχνολογία υλοποίησης είναι μια πλακέτα του συστήματος ενδεχόμενα η δομική αρχιτεκτονική να είναι η καταλληλότερη. Και οι δύο behavioral αρχιτεκτονικές είναι αποδοτικότερες από πλευράς χώρου μνήμης και χρόνου εκτέλεσης στον εξομοιωτή. Η επιλογή ανάμεσά τους είναι καθαρά θέμα

---

του στυλ που προτιμάει ο προγραμματιστής αν δηλαδή προτιμάει να χρησιμοποιεί παράλληλα ή ακολουθιακά εκτελούμενες εντολές. Θα δούμε αργότερα ότι υπάρχει και μια τέταρτη μορφή αρχιτεκτονικής που χρησιμοποιείται συχνά σαν είσοδος στο synthesis εργαλείο. Πρόκειται για την RTL (Register Transfer Level) περιγραφή σε VHDL και ονομάζεται έτσι διότι σε αυτήν περιγράφουμε μόνο όλους τους καταχωρητές του σχεδιασμού μας και την συνδυαστική λογική ανάμεσά τους. Στο παρόδειγμά μας, η αρχιτεκτονική behavioral είναι ταυτόχρονα και μια RTL περιγραφή.

Οι εντολές διάρθρωσης χρησιμοποιούνται από τον σχεδιαστή ώστε να επιλέξει για κάθε αντίτυπο των διαφορετικών οντοτήτων μία από τις διαφορετικές αρχιτεκτονικές που μπορεί να έχει κάθε οντότητα. Για το παρόδειγμά μας μια πιθανή διάρθρωση θα ήταν :

```
CONFIGURATION one_bit_full_adder_conf1 OF one_bit_full_adder IS
FOR structural
    FOR U1, U2 : xor2 USE ENTITY WORK.myxor2(version1);
        END FOR;
    FOR U3, U4, U5 : and2 USE ENTITY WORK.myand2(version1);
        END FOR;
    FOR U6 : or3 USE ENTITY WORK.myor3(version1);
        END FOR;
    END FOR;
END one_bit_full_adder_conf1;
```

όπου ορίζουμε την δομική αρχιτεκτονική για τον αθροιστή ενός δυαδικού ψηφίου και για τις πύλες XOR δύο εισόδων την χρήση της οντότητας myxor2 και αρχιτεκτονικής version1 που υπάρχουν στην βιβλιοθήκη WORK. Αντίστοιχα ορίζουμε τις οντότητες και αρχιτεκτονικές και των άλλων πυλών. Με τη μετάφραση των οντοτήτων, των αρχιτεκτονικών και των διαρθρώσεων, καταλήγουμε σε ένα μοντέλο του κυκλώματός μας, έτοιμο για εξομοίωση. Αν θέλαμε να επιλέξουμε την αρχιτεκτονική behavioral για εξομοίωση, θα αρκούσε ο παρακάτω κώδικας :

```
CONFIGURATION one_bit_full_adder_conf2 OF one_bit_full_adder IS
FOR behavioral
    END FOR;
END one_bit_full_adder_conf2;
```

---

Ο παραπάνω κώδικας δεν είναι αναγκαίος, όταν η επιθυμητή αρχιτεκτονική για κάποια οντότητα είναι η τελευταία που μεταφράστηκε. Ωστόσο εξυπηρετεί υπερβολικά στην αναγνωσιμότητα του σχεδιασμού.

4. Βιβλιοθήκες (Packages) : Μία βιβλιοθήκη είναι μια συλλογή από συχνά χρησιμοποιούμενες δομές και υποπρογράμματα. Αποτελεί μια συλλογή εργαλείων για την περιγραφή μεγαλύτερων σχεδιασμών.
5. Ιδιότητες (Attributes) : Μία ιδιότητα είναι ένα δεδομένο αντιστοιχισμένο σε κάποιο αντικείμενο της VHDL. Ιδιότητες αποτελούν για παράδειγμα η οδηγητική ικανότητα μίας πύλης ή η μέγιστη θερμοκρασία λειτουργίας μιας συσκευής.
6. Παράμετροι (Generics) : Χρησιμοποιούνται για το πέρασμα πληροφοριών σε μία οντότητα. Για παράδειγμα, αν η οντότητα είναι ένα μοντέλο πυλών με χρόνους ανόδου και καθόδου, αυτοί θα μπορούσαν να καθορίζονται μέσω παραμέτρων κάθε φορά. Οι παράμετροι είναι ένας γενικός μηχανισμός για πέρασμα πληροφοριών σε ένα αντίτυπο μιας οντότητας. Τις πιο πολλές φορές χρησιμοποιούνται για το πέρασμα πληροφοριών χρονικής καθυστέρησης. Ωστόσο, η χρησιμότητά τους είναι μεγάλη και κατά την διαδικασία της σύνθεσης για μεγέθη όπως το εύρος μιας αρτηρίας. Κάθε πληροφορία που ανατίθεται σε μια παράμετρο έχει ισχύ μόνο για το συγκεκριμένο αντίτυπο μιας οντότητας και είναι πάντα στατική (δεν μπορεί να είναι μια άλλη παράμετρος π.χ. της διαδικασίας εξομοίωσης). Ο σχεδιαστής έχει την δυνατότητα να αναθέσει διαφορετικές τιμές στην ίδια παράμετρο διαφορετικών αντιτύπων της ίδιας οντότητας. Για παράδειγμα ο παρακάτω κώδικας ορίζει τρεις παραμέτρους (οι δύο πρώτες τύπου χρόνου και η τρίτη ακεραίου) για μια πύλη AND 2 εισόδων :

```
ENTITY and2 IS
  GENERIC (rise, fall : TIME; load : INTEGER);
  PORT (a, b :IN BIT;
        c :OUT BIT);
END and2;
```

Μια πιθανή αρχιτεκτονική για την παραπάνω οντότητα θα μπορούσε να είναι :

```
ARCHITECTURE load_dependent OF and2 IS
  SIGNAL internal : BIT ;
BEGIN
  internal <= a AND b;
  c <= internal AFTER (rise + (load * 2ns)) WHEN internal = '1'
    ELSE internal AFTER (fall + (load * 3ns));
END
```

---

όπου χρησιμοποιούμε την τοπική μεταβλητή `internal` για να αποθηκεύσουμε την λογική τιμή της πύλης και την αναθέτουμε στην έξοδο μετά από χρόνο που δίνεται σαν συνάρτηση των παραμέτρων. Στη συνέχεια μπορούμε να καλούμε αντίτυπα αυτής της πύλης με διάφορα χρονικά χαρακτηριστικά όπως :

U1 : and2 GENERIC MAP (10 ns, 12 ns, 3) PORT MAP (`in1, in2, out1`);  
 U2 : and2 GENERIC MAP (9 ns, 11 ns, 5) PORT MAP (`in3, in4, out2`);

7. Διαδικασίες (Processes) : Η διαδικασία είναι η βασική μονάδα εκτέλεσης του εξομοιωτή στην VHDL. Ολες οι λειτουργίες που πραγματοποιούνται στην εξομοίωση μιας VHDL περιγραφής διαχωρίζονται σε απλές ή πολλαπλές διαδικασίες. Παρόλον διαδικασιών δίνουμε παρακάτω.

Διάφορες εντολές της γλώσσας μπορούν να ομαδοποιηθούν σε BLOCKS, και η εκτέλεση κάθε BLOCK μπορεί να οριοθετηθεί βάσει μιας λογικής συνθήκης (guard statement). Για παράδειγμα στον επόμενο κώδικα στον οποίο περιγράφεται η λειτουργία ενός latch, με εισόδους `clk` και `d` και εξόδους `q`, `qb` :

```
ARCHITECTURE guard_latch IS
BEGIN
    G1 : BLOCK (clk = '1')
    BEGIN
        q <= GUARDED d AFTER 5 ns;
        qb <= GUARDED NOT(d) AFTER 7 ns;
    END BLOCK G1;
END guard_latch
```

Οι δύο εντολές εκχώρησης θα εκτελεστούν (παράλληλα) αν και μόνο αν το σήμα `clk` έχει την τιμή 1. Προσέξτε ότι καθώς στην παραπάνω περιγραφή δεν δηλώνεται τι θα συμβεί όταν το σήμα `clk` πάρει την τιμή 0, θεωρείται ότι σε αυτήν την περίπτωση (λόγω του αποκλεισμού εκτέλεσης των δύο εντολών εκχωρήσεων) οι έξοδοι του latch θα διατηρήσουν τις τιμές που είχαν, οδηγώντας μας σε ακολουθιακό κύκλωμα κατά την διαδικασία της σύνθεσης.

Κλείνοντας την σύντομη αναφορά μας στην VHDL, θα πρέπει να σημειώσουμε ότι αν και όλες οι δομές μιας σωστής συντακτικά VHDL είναι εξομοιώσιμες, ελάχιστες από αυτές μπορούν να οδηγήσουν σε σωστή είσοδο σε ένα synthesis εργαλείο. Παρακάτω παρουσιάζονται οι RTL περιγραφές δύο διαφορετικών κυκλωμάτων, ώστε αφενός να διευκρινιστεί η έννοια της RTL περιγραφής και αφετέρου να παρουσιαστούν επιπλέον δομές της γλώσσας. Το πρώτο κύκλωμα που

---

περιγράφουμε είναι ένα θετικά ακμοπυροδότητο D flip-flop με ασύγχρονες εισόδους για preset και clear.

```
USE work.std_logic_1164.ALL;
ENTITY dff_pc IS
    PORT (preset, clear, clock, din : IN std_logic;
          dout : OUT std_logic);
END dff_pc;
ARCHITECTURE synthesisable OF dff_pc IS
BEGIN
    PROCESS (preset, clear, clock)
    BEGIN
        IF (preset = '1') THEN dout <= '1';
        ELSEIF (clear = '1') THEN dout <= '0';
        ELSEIF (clock'EVENT) and (clock = '1') THEN dout <= din;
        ENDIF;
    END PROCESS;
END synthesisable;
```

Στην παραπάνω περιγραφή δηλώνουμε αρχικά ότι θέλουμε να χρησιμοποιήσουμε τις συναρτήσεις της standard βιβλιοθήκης 1164. Αποτέλεσμα αυτού είναι να μπορούμε να χρησιμοποιήσουμε διάφορους ήδη ορισμένους τύπους δεδομένων, όπως για παράδειγμα την std\_logic (τύπος δεδομένων 9 τιμών για την ικανότητα οδήγησης κάθε σήματος που χρησιμοποιούμε. Οι τιμές αυτές είναι : 0, 1, X (άγνωστη) , Z (κατάσταση υψηλής εμπέδησης) κλπ). Ακολουθεί η δήλωση της οντότητάς μας με την δήλωση των εισόδων και των εξόδων της. Στην δήλωση της αρχιτεκτονικής μας ορίζουμε μια διαδικασία με λίστα ευαισθησίας τα τρία σήματα preset, clock και clear. Αποτέλεσμα αυτού είναι οι εντολές της διαδικασίας (στην προκειμένη περίπτωση μόνο μία, η εντολή IF) να εκτελεστούν μόνο όταν υπάρχει αλλαγή τιμής σε κάποιο από αυτά τα τρία σήματα. Η τελευταία περίπτωση του IF ορίζει ότι τα δεδομένα εισόδου θα μεταφερθούν στην είσοδο, μόνο αν υπήρχε μεταβολή στην τιμή του σήματος clock (clock'EVENT) και η νέα τιμή του είναι 1, με όλα λόγια εάν υπήρχε ανοδική ακμή του ρολογιού του flip-flop. Προσέξτε ότι μετά την διαδικασία της σύνθεσης το κύκλωμα που θα πάρουμε (υποθέτουμε ένα synthesis tool που υποστηρίζει πλήρως τη standard της γλώσσας), θα δίνει προτεραιότητα στην είσοδο preset σε σχέση με την είσοδο clear, γιατί οι εντολές εντός της διαδικασίας εκτελούνται ακολουθιακά. Με όλα λόγια το κύκλωμα που θα

---

προκύψει από την σύνθεση στην περίπτωση που preset = clear = 1, θα πρέπει να μας δίνει έξοδο 1. Εάν αυτό δεν είναι το επιθυμητό, τότε θα πρέπει να αλλάξουμε τον κώδικα μας γράφοντας την συνθήκη για το clear πριν από αυτή για το preset. Τέλος, αν επιθυμούμε σύγχρονο preset και clear, θα πρέπει να προσαρμόσουμε τον κώδικα μας όπως παρακάτω :

```

PROCESS (clock)
BEGIN
    IF (clock'EVENT) AND (clock = '1') THEN
        IF (preset = '1') THEN dout <= '1';
        ELSEIF (clear = '1') THEN dout <= '0';
        ELSE dout <= din;
        END IF;
    END IF;
END PROCESS;
```

Το δεύτερο κύκλωμα που θα περιγράψουμε είναι ένας μετρητής 4 δυαδικών ψηφίων. Θα χρησιμοποιήσουμε για την περιγραφή την τεχνική των δύο παράλληλων συνεργαζόμενων διαδικασιών. Η μία διαδικασία λειτουργεί βάσει του ρολογιού του συστήματος (σύγχρονη) και χρησιμοποιείται για να αποθηκεύει την τρέχουσα κατάσταση του μετρητή στους καταχωρητές, ενώ η δεύτερη διαδικασία περιγράφει το συνδυαστικό κύκλωμα που βάσει της τρέχουσας κατάστασης και των εισόδων υπολογίζει την επόμενη κατάσταση του μετρητή. Στην παρακάτω περιγραφή επιπλέον ορίζουμε και έναν καινούργιο τύπο δεδομένων (bit4) για μια καινούργια βιβλιοθήκη (package) με την ονομασία count\_types.

Η οντότητά μας στην περιγραφή παρακάτω έχει μία είσοδο ρολογιού, μια είσοδο για σύγχρονη φόρτωση του μετρητή, μια είσοδο για σύγχρονο καθαρισμό του μετρητή, τις εισόδους δεδομένων καθώς και τις εξόδους του μετρητή. Η περιγραφή των εξόδων γίνεται σαν είσοδοι / έξοδοι με σκοπό την εξέτασή τους για τον καθορισμό της επόμενης κατάστασης. Η πρώτη διαδικασία της περιγραφής καθορίζει την επόμενη τιμή του μετρητή με μια απλή εντολή IF εξετάζοντας όλες τις εισόδους και δίνοντας κάποιο καθορισμένο σχήμα προτεραιότητας. Η δεύτερη διαδικασία είναι αυτή που μεταφέρει τα αποτελέσματα που υπολογίστηκαν από την πρώτη διαδικασία στις εξόδους του μετρητή. Προσέξτε ότι αν και δεν υπάρχει λίστα ευαισθησίας στην δεύτερη διαδικασία, η εντολή εκχώρησης αυτής θα εκτελεστεί μόνο μετά από την εκτέλεση της πρώτης εντολής της διαδικασίας (λόγω της ακολουθιακής εκτέλεσης των εντολών) που ουσιαστικά «μπλοκάρει» την εκτέλεση της εντολής εκχώρησης μέχρι μια

---

ανοδική ακμή του ρολογιού. Επίσης προσέξτε ότι η εκτέλεση της εντολής εκχώρησης λόγω της μεταβολής στην τιμή των εξόδων του μετρητή που προκαλεί, αυτόματα προκαλεί και την εκτέλεση της πρώτης διαδικασίας. Υπενθυμίζεται ότι όλες οι διαδικασίες εκτελούνται παράλληλα. Ο συγχρονισμός σε αυτή την περιγραφή πραγματοποιείται με το μπλοκάρισμα των δύο διαδικασιών με διαφορετικές συνθήκες.

```
PACKAGE count_types IS
    TYPE bit4 IS range 0 to 15;
END count_types;

USE WORK.std_logic_1164.ALL;
USE WORK.count_types.ALL;
ENTITY count IS
    PORT (clock, load, clear : IN std_logic;
          din : IN bit4;
          dout : inout bit4);
END count;

ARCHITECTURE synthesisable OF COUNT IS
    SIGNAL count_val : bit4;
BEGIN
    PROCESS (load, clear, din, dout)
    BEGIN
        IF (load = '1') THEN count_val <= din;
        ELSEIF (clear = '1') THEN count_val <= 0;
        ELSEIF (dout >= 15) THEN count_val <= 0;
        ELSE count_val <= dout + 1;
        END IF;
    END PROCESS;

    PROCESS
    BEGIN
        WAIT UNTIL clock'EVENT and clock = '1';
        dout <= count_val;
    END PROCESS;
END synthesisable;
```

---

Η Verilog HDL είναι η δεύτερη ευρέως διαδεδομένη γλώσσα περιγραφής υλικού. Πολλοί πιστεύουν ότι είναι σημαντικά απλούστερη στην εκμάθηση από ότι η VHDL, καθώς μοιάζει αρκετά με την C. Εισήχθηκε από την εταιρεία Gateway Design System Corporation το 1985 και το 1995 έγινε standard της IEEE. Οπως και η VHDL υποστηρίζει την περιγραφή κυκλωμάτων σε όλα τα επίπεδα αφαιρεσης και παρέχει δομικούς και behavioral τρόπους περιγραφής τους. Επιπλέον είναι και αυτή πλήρως εξομοιούμενη, αλλά μόνο ένα μικρό υποσύνολό της μπορεί να αποτελέσει ικανή είσοδο σε ένα εργαλείο σύνθεσης.

Οι δομές που χρησιμοποιεί η Verilog είναι αντίστοιχες με αυτές της VHDL. Οι οντότητες αντικαθίστανται με κομμάτια (modules), οι μεταβλητές με registers, και οι συνδέσεις με wires. Επιπλέον, όπως και στην VHDL όλες οι εντολές που περιλαμβάνονται μέσα σε ένα module εκτελούνται παράλληλα, πλην των εντολών που περιλαμβάνονται μέσα σε always blocks (που αντιστοιχούν στις διαδικασίες της VHDL). Οπως θα φανεί και στα παραδείγματα που παραθέτονται πιο κάτω, η δομή της γλώσσας είναι λιγότερο αυστηρή από αυτήν της VHDL, επιτρέποντας στον σχεδιαστή να γράψει σημαντικά λιγότερο κώδικα για να εκφράσει αντίστοιχα κυκλώματα. Για παράδειγμα, ο ακόλουθος κώδικας είναι αρκετός για την behavioral περιγραφή του πλήρους αθροιστή του ενός δυαδικού ψηφίου :

```
module one_bit_full_adder (i1, i2, cin, s, cout);
    input i1, i2, cin ;
    output s, cout;
    assign s = i1 ^ i2 ^ cin;
    assign cout =(i1 & i2) || (i1 & cin) || (i2 & cin);
endmodule
```

Η περιγραφή ενός σχεδιασμού ξεκινάει με το keyword module και τελειώνει με το endmodule. Ακολουθεί η δήλωση των εισόδων – εξόδων του κυκλώματος και οι εντολές που περιγράφουν την λειτουργία του κυκλώματος. Μιας και το κύκλωμα μας είναι ακολουθιακό χρησιμοποιούμε εντολές ανάθεσης που ξεκινούν με το keyword assign. Οι εντολές αυτές είναι γνωστές και σα continuous assignments. Οι δύο αυτές εντολές εκτελούνται παράλληλα. Εχοντας δεδομένη την παραπάνω περιγραφή, μπορούμε πολύ εύκολα να γράψουμε τον structural κώδικα για τον αθροιστή των δύο δυαδικών ψηφίων με διάδοση κρατουμένου :

---

```

module two_bit_full_adder (a1, a2, cin, sum, cout);
    input [1:0] a1, a2;
    input cin;
    output [1:0] s;
    output cout;
        one_bit_full_adder add1 (a1[0], a2[0], cin, sum[0], temp);
        one_bit_full_adder add2 (a1[1], a2[1], temp, sum[1], cout);
    endmodule

```

Στον παραπάνω κώδικα ορίζουμε τρεις αρτηρίες ( $a_1$ ,  $a_2$ ,  $sum$ ) μεγέθους 2 δυαδικών ψηφίων η οποία μία. Με δεδομένο τον κώδικα για τον αθροιστή ενός δυαδικού ψηφίου που είδαμε προηγούμενα, χρησιμοποιούμε δύο αντίτυπα αυτού του υποσχεδιασμού (τα οποία ονομάζουμε  $add1$  και  $add2$ ) για να περιγράψουμε τον αθροιστή των δύο δυαδικών ψηφίων. Σημειώστε ότι το σήμα  $temp$ , αντιπροσωπεύει στην ουσία ένα καλώδιο που συνδέει τους δύο αθροιστές και δεν είναι αναγκαίο να δηλωθεί (προς χάριν της αναγνωσιμότητας θα μπορούσε να δηλωθεί σαν wire  $temp$ ). Γενικά στη Verilog τα wires δεν είναι απαραίτητο να δηλώνονται όταν πρώτα τους ανατίθεται μια τιμή και μετά τα χρησιμοποιούμε. Αναθέσεις σε wires αυτόματα συνεπάγονται την περιγραφή συνδυαστικής λογικής.

Η Verilog διαθέτει την δομή των always blocks για την περιγραφή ακολουθιακών κυκλωμάτων και τον έλεγχο του χρονισμού τους. Όλα τα always blocks εκτελούνται παράλληλα από έναν εξομοιωτή της γλώσσας, ενώ οι εντολές που περιέχονται σε αυτά εκτελούνται ακολουθιακά. Για παράδειγμα ας θεωρήσουμε το κύκλωμα του latch. Ενας synthesizable κώδικας σε Verilog ενός τέτοιου κυκλώματος θα ήταν ο ακόλουθος :

```

module latch (din, clock, q, qb);
    input din, clock;
    output q, qb;
    reg q, qb;
    always @(clock or din)
        if (clock)
            begin
                q<= din;
                qb <= ~din;
            end
endmodule

```

---

Στον παραπάνω κώδικα το always block θα εκτελείται μόνο όταν υπάρχουν αλλαγές στις τιμές των σημάτων clock ή din. Τα σήματα δηλαδή αυτά αποτελούν τη λίστα ευαισθησίας του always block. Παρατηρείστε ότι σύμφωνα με το κώδικα, αλλαγές στα σήματα εξόδου επιτρέπονται μόνο όταν το clock έχει τιμή 1. Επειδή δεν διευκρινίζουμε τι θα γίνει σε περίπτωση που το clock έχει τιμή 0, το εργαλείο σύνθεσης θεωρεί ότι οι έξοδοι θα κρατούν σε αυτή την περίπτωση τις προηγούμενες τιμές τους, οδηγούμενο έτσι στην παραγωγή ακολουθιακής λογικής. Ας σταματήσουμε για λίγο στη δήλωση των εξόδων q και qb σα registers μέσω της reg q, qb; Η δήλωση αυτή δεν είναι δεσμευτική για το πρόγραμμα σύνθεσης. Δηλαδή επειδή εμείς τα ορίζουμε σαν ακολουθιακή λογική δε σημαίνει ότι σώνει και καλά θα υλοποιηθούν σα τέτοια. Από την άλλη πλευρά, η δήλωση αυτή είναι απαραίτητη για το συντακτικό της γλώσσας. Χωρίς αυτήν δηλαδή παίρνουμε μηνύματα λάθους κάθε φορά που θα προσπαθήσουμε να περάσουμε το κώδικα μας από το μεταφραστή της γλώσσας.

Ενα άλλο ενδιαφέρον σημείο είναι η χρήση του συμβόλου  $\leq$  για την ανάθεση τιμών στα q και qb. Εναλλακτικά θα μπορούσαμε να χρησιμοποιήσουμε και το σύμβολο  $=$ . Ωστόσο οι δύο αυτές αναθέσεις διαφέρουν σημαντικά. Οπως αναφέραμε πιο πάνω εντός των always blocks η εκτέλεση των εντολών είναι ακολουθιακή. Αυτό σημαίνει ότι αν είχαμε και άλλη μια εντολή πέραν της if που υπάρχει στο παράδειγμά μας, θα έπρεπε να ήμασταν εξαιρετικά προσεκτικοί στη σειρά αναγραφής τους. Εντός όμως του κομματιού κώδικα που περιβάλλεται από τα begin – end ο σχεδιαστής έχει τη δυνατότητα να επιλέξει είτε ακολουθιακή είτε παράλληλη εκτέλεση. Στη περίπτωση που χρησιμοποιεί την ανάθεση μέσω του συμβόλου  $\leq$  επιλέγει παράλληλη εκτέλεση. Μιας και η εκτέλεση αυτής της εντολής δε σταματά την εκτέλεση άλλων εντολών η ανάθεση αυτή ονομάζεται non-blocking assignment. Στην αντίθετη περίπτωση, δηλαδή όταν χρησιμοποιεί την ανάθεση μέσω του συμβόλου  $=$ , ο σχεδιαστής επιλέγει την ακολουθιακή εκτέλεση. Η εκτέλεση των επομένων εντολών δε μπορεί να ξεκινήσει ποτέ ολοκληρωθεί η εκτέλεση αυτής της ανάθεσης. Γι' αυτό και η ανάθεση αυτή είναι γνωστή σα blocking assignment.

Τα primitives της γλώσσας negedge και posedge μπορούν να χρησιμοποιηθούν για την περιγραφή συνθέσιμων κυκλωμάτων ακολουθιακής λογικής με ακμοπυροδότητα flip – flops. Ο παρακάτω κώδικας θα μπορούσε να χρησιμοποιηθεί για την περιγραφή ενός αρνητικά ακμοπυροδότητου flip-flop με ασύγχρονη είσοδο για καθαρισμό :

---

```

module flip_flop1 (din, clk, clear, q);
    input din, clk, clear;
    output q;
    always @ (negedge clock or posedge clear)
        if (clear) q <= 0;
        else q <= din;
endmodule

```

Στον πιο πάνω κώδικα το always block θα εκτελεστεί είτε όταν υπάρξει θετική ακμή στο σήμα καθαρισμού είτε όταν υπάρξει αρνητική ακμή ρολογιού. Στην πρώτη περίπτωση θα εκτελεστεί η εντολή if (clear) q <= 0; ενώ στην δεύτερη τα δεδομένα εισόδου θα μεταφερθούν στην έξοδο. Πολύ εύκολα ο παραπάνω κώδικας μπορεί να αλλαχθεί για περιγραφή ενός flip-flop με σύγχρονη είσοδο καθαρισμού αλλάζοντας μόνο την συνθήκη του always block σε :

```
always @ (negedge clock)
```

Για να δείξουμε πόσο εύκολη είναι η περιγραφή ενός κυκλώματος σε Verilog παρακάτω παραθέτουμε τον synthesisable κώδικα που υλοποιεί τον μετρητή των 4 δυαδικών ψηφίων, που περιγράψαμε πιο πάνω σε VHDL.

```

module count4 (clock, load, clear, din, dout);
    input load, clear, clock;
    input [3:0] din;
    output [3:0] dout;
    reg [3:0] temp;

    always @ (load or clear or dout or din)
        if (load) temp <= din;
        else if (clear) temp <= 0;
        else temp <= dout+1;
    always @ (posedge clock) dout <= temp;

endmodule

```

Η παραπάνω περιγραφή είναι ακριβώς αντίστοιχη με αυτήν που δώσαμε για τη γλώσσα VHDL. Υπάρχουν δύο always blocks, εκ των οποίων το πρώτο υπολογίζει την επόμενη τιμή για τις εξόδους του κυκλώματος ανεξάρτητα από το ρολόι του συστήματος και η δεύτερη οδηγεί αυτήν την υπολογισμένη τιμή στις εξόδους του

---

κυκλώματος μετά από την ανερχόμενη ακμή του ρολογιού. Προσέξτε ότι η αρτηρία temp, παρότι δηλώνεται σα καταχωρητής στην ουσία χρησιμοποιείται εντός ενός always block που περιγράφει συνδυαστική λογική.

Συνοπτικά μπορούμε να εκφράσουμε τον εξής κανόνα για το πότε ένα always block περιγράφει ακολουθιακή και πότε συνδυαστική λογική. Ακολουθιακή λογική περιγράφεται όταν :

1. Χρησιμοποιούνται ακμές σημάτων π.χ.

always @ (posedge clock or posedge clear)

2. Οποτεδήποτε δεν εξαντλούνται όλες οι πιθανές περιπτώσεις κάποιας εντολής if, όπως για παράδειγμα στο κώδικα που δώσαμε προηγούμενα για το latch. Η περίπτωση αυτή διακρίνεται εύκολα από την απουσία τελικού else statement. Για παράδειγμα ο κώδικας :

always @ (load or clear or dout or din)  
if (load) temp <= din;  
else if (clear) temp <= 0;  
else temp <= dout+1;

δεν περιγράφει ακολουθιακή λογική αφού υπάρχει τελικό else. Καλό είναι η συνδυαστική λογική να μη κωδικοποιείται με always statements μιας και μπορεί εύκολα να προκληθεί σύγχυση σχετικά με το τι λογική περιγράφει ο κώδικας. Για παράδειγμα ο παραπάνω κώδικας μπορεί να γραφεί με τη μορφή continuous assignment, στον οποίο χρησιμοποιείται ο γνωστός σας από τη C three-way assignment τελεστής.

assign temp = (load) ? din :  
(clear) ? 4'b0 : dout+1;

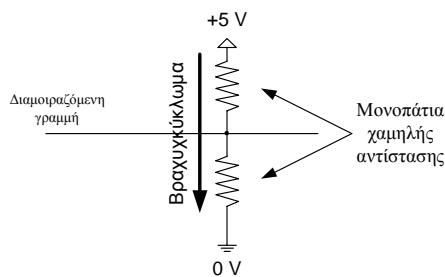
Στη συνέχεια θα δούμε μερικά πιο πολύπλοκα παραδείγματα. Σε κάποια από τις ασκήσεις που θα κληθείτε να κάνετε στο εργαστήριο ζητείται να σχεδιάσετε ένα ιλειδωτήρι λογισμικού (hasp) σε τεχνολογία PCB. Στο παράδειγμα αυτό θα περιγράψουμε με γλώσσα Verilog τον ίδιο σχεδιασμό για 2 λόγους :

- 1) Για να δείξουμε πόση λιγότερη προσπάθεια χρειάζεται για τη περιγραφή του ίδιου σχεδιασμού όταν ξεφεύγουμε από τη δομική περιγραφή και χρησιμοποιούμε περιγραφή βάσει συμπεριφοράς. Ο γραφικός τρόπος σχεδιασμός για το hasp αποτελούνταν από ένα ογκώδες φύλλο σχεδιασμού.

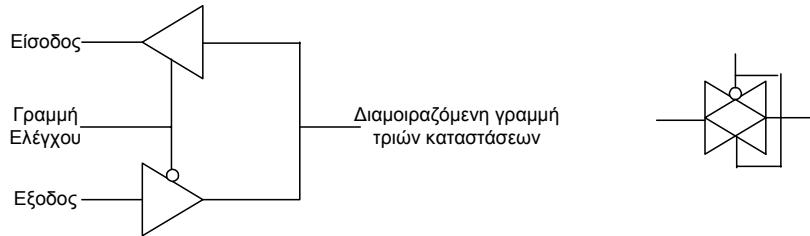
- 2) Γιατί είναι ένα εξαιρετικό παράδειγμα του χειρισμού που απαιτεί μια γραμμή τριών καταστάσεων.

Το κλειδωτήρι μας στην ουσία είναι ένας σχεδιασμός που συνδέεται στη παράλληλη θύρα του υπολογιστή μας. Ανά τακτά χρονικά διαστήματα, το λογισμικό μας στέλνει ένα byte προς το hasp. Αυτό με βάση αυτό το byte παράγει ένα κρυπτογραφημένο byte που για λόγους απλότητας θεωρούμε ότι είναι το byte που έλαβε αυξημένο κατά 127. Το λογισμικό μας ακολούθως διαβάζει από τη παράλληλη θύρα το κρυπτογραφημένο byte και το συγκρίνει με τη κρυπτογράφηση που εκτέλεσε το ίδιο. Αν τα δύο bytes δεν είναι ίδια τότε το hasp δεν υπάρχει στη παράλληλη θύρα και συνεπώς υπήρξε απόπειρα αντιγραφής του λογισμικού.

Ας θυμηθούμε πρώτα μερικά πράγματα για τις γραμμές τριών καταστάσεων. Στη γενική περίπτωση σε μια γραμμή τριών καταστάσεων υπάρχουν περισσότεροι του ενός πιθανοί οδηγοί της. Θα ήταν καταστροφικό να επιτρέψουμε σε περισσότερες της μιας συσκευές να οδηγούν **ταυτόχρονα** τη γραμμή αυτή μιας που το μόνο που θα επιτυγχάναμε θα ήταν ένα βραχυκύλωμα. Το παρακάτω σχήμα δείχνει τι θα συμβεί πάνω στη γραμμή αυτή όταν ένας εκ των οδηγών της προσπαθεί να θέσει αυτή τη γραμμή στο λογικό 0 και ένας δέυτερος ταυτόχρονα προσπαθεί να τη θέσει στο λογικό 1. Καθένας από αυτούς αποκαθιστά ένα αγώγιμο μονοπάτι χαμηλής αντίστασης με τη γειωση και τη τροφοδοσία αντίστοιχα. Αν υποθέσουμε ότι το λογικό 0 αντιστοιχεί στα 0V και το λογικό 1 στα 5V, τότε το ηλεκτρικό κύκλωμα που αντιστοιχεί θα είναι :



Βάσει των παραπάνω είναι προφανές ότι δε θα πρέπει να επιτρέπουμε **ταυτόχρονα** σε περισσότερες της μιας συσκευές να οδηγούν κάποια αρτηρία. Μόνο μία εκ των συσκευών θα πρέπει να οδηγεί την γραμμή ενώ όλες οι υπόλοιπες θα πρέπει να έμφανίζουν μονοπάτια μεγάλης σύνθετης αντίστασης (εμπέδησης) στην έξοδό τους. Αυτό βέβαια δεν αποκλείει τη δυνατότητα πολλών συσκευών να διαβάζουν τα δεδομένα της γραμμής. Η λογική αυτή μας οδηγεί στο παρακάτω λογικό διάγραμμα για κάθε υποσύστημα (δεξιά ένας από τους πλέον συνήθεις συμβολισμούς του) το οποίο διαμοιράζεται τη γραμμή τριών καταστάσεων :



Η τιμή της γραμμής ελέγχου καθορίζει αν ο συγκεκριμένος σχεδιασμός ή όχι θα οδηγήσει τη γραμμή τριών καταστάσεων. Οταν αυτή η γραμμή είναι στο 0, ο σχεδιασμός οδηγεί τη γραμμή και συνεπώς κάθε άλλος σχεδιασμός θα πρέπει να έχει αυτή τη γραμμή στο 1. Οταν κάποιος άλλος σχεδιασμός οδηγεί τη διαμοιραζόμενη γραμμή ο σχεδιαστής οφείλει να έχει αυτή τη γραμμή στο 1 έτσι ώστε η τιμή της διαμοιραζόμενης γραμμής να μπορεί να διαβαστεί σαν είσοδος. Φυσικά θα μπορούσαμε να μην έχουμε απομονωτή τριών καταστάσεων στο πάνω μέρος του σχήματος αφού κανείς δε μας απαγορεύει να διαβάζουμε πάντα από την αρτηρία (ακόμη κι όταν την οδηγούμε) όπως και θα μπορούσε τα σήματα ελέγχου των τυχόν δύο απομονωτών τριών καταστάσεων να μην είναι σχετιζόμενα.

Ας δούμε το κώδικα σε Verilog που απαιτείται για το ιλειδωτήρι λογισμικού μαζί με επεξηγηματικά σχόλια κυρίως για τη διαχείριση της αρτηρίας τριών καταστάσεων :

```

module hasp (PWEn, PREn, PCLK, PD);
    input PWEn, PREn, PCLK ;
    // Parallel Write Enable, Parallel Read Enable σήματα αρνητικής λογικής.
    // Parallel Clock, το οποίο που σχηματίζει ο λογισμικό για το hasp.
    inout [7:0] PD;
    // Η αρτηρία τριών καταστάσεων.
    // Το DIN είναι η είσοδος από τις γραμμές τριών καταστάσεων ή με άλλα λόγια η έξοδος του πάνω
    // buffer στο αριστερό πιο πάνω σχήμα.
    wire [7:0] DIN;
    // Το DOUT είναι η έξοδος προς τις γραμμές τριών καταστάσεων ή με άλλα λόγια η έξοδος του
    // κάτω buffer στο αριστερό πιο πάνω σχήμα. Στη παρίπτωσή μας, η έξοδος αυτή είναι έξοδος ενός
    // καταχωρητή.
    reg [7:0] DOUT;
    // Η παρακάτω ανάθεση αυτή δείχνει το χειρισμό της γραμμής τριών καταστάσεων. Τα DOUT
    // οδηγούνται πάνω στην αρτηρία μόνο όταν το σήμα PREn είναι ενεργό, δηλαδή όταν το λογισμικό
    // μας ζητήσει ανάγνωση δεδομένων από την παράλληλη θύρα. Σε κάθε άλλη περίπτωση, το hasp
    // δεν οδηγεί τα διαμοιραζόμενα σήματα. Ενδιαφέρον παρουσιάζει το πως διαχειρίζομαστε τις

```

---

```

// σταθερές στη Verilog. Οι σταθερές αντιπροσωπεύονται με literals της μορφής X'BV, όπου X ο
// αριθμός των σημάτων που παίρουν μέρος στην ανάθεση, B η βάση αναγραφής της σταθεράς και
// V η τιμή της. Εποιητικό θεώρετα η δυαδική τιμή 0 εκφρασμένη σε μήκος ενός δυαδικού
// ψηφίου, ενώ 4'hF, σημαίνει η δεκαεξαδική ποσότητα F εκφρασμένη σε 4 δυαδικά ψηφία.
assign PD = (PREn == 1'b0) ? DOUT[7:0] : 8'bz;

// Η παρακάτω ανάθεση αυτή δείχνει το διάβασμα από την αρτηρία. Φυσικά σε αυτή τη περίπτωση
// θα μπορούσαμε να μην έχουμε τον απομονωτή τριών καταστάσεων και απλά να συνδέαμε τις
// γραμμές DIN και PD. Ωστόσο η παρακάτω γραμμή κώδικα κάνει ότι ακριβώς και το
// σχηματικό της άσκησης (μη ξεχνάμε το 1o στόχο).
assign DIN = (PWEen == 1'b0) ? PD[7:0] : 8'bz;

// Το "κλειδωμα", έγκειται στη πρόσθεση του δεκαδικού 127 !!!
always @(posedge PCLK)
    DOUT[7:0] <= DIN[7:0] + 8'd127;
endmodule

```

Ενδιαφέρον στη περίπτωση αυτή παρουσιάζει και το κύκλωμα το οποίο γεννά τις κυματομορφές εισόδου καθώς και πιστοποιεί τις εξόδους του κυκλώματος hasp (κύκλωμα testbench). Μιας και το κύκλωμα testbench δεν απαιτείται να είναι synthesizable, προφανώς έχουμε τη δυνατότητα να χρησιμοποιήσουμε γι' αυτό όλο το σύνολο των συντακτικών περιγραφών της Verilog. Συνήθεις δομές που χρησιμοποιούμε είναι :

- ♦ initial blocks : αντίστοιχα των always blocks, με μόνη διαφορά ότι εκτελούνται μόνο μια φορά. Προφανώς όλα τα initial blocks εκτελούνται παράλληλα, ενώ σε κάθε initial block, οι εντολές εκτελούνται ακολουθιακά.
- ♦ Χρονοκαθυστερήσεις. Με το να χρησιμοποιούμε #X statements πριν από ηποια εντολή της γλώσσας, εισάγουμε χρονοκαθυστέρηση X μονάδων εξομοίωσης. Προφανώς η εντολή που ακολουθεί ηποια με χρονοκαθυστέρηση X μονάδων είτε εκτελείται την ίδια χρονική στιγμή, είτε αμέσως μετά την εκτέλεση της εντολής με τη χρονοκαθυστέρηση (X+δ) ανάλογα με το αν στην εντολή με τη χρονοκαθυστέρηση χρησιμοποιούμε blocking ή non-blocking assignments.

Στο συγκεκριμένο παράδειγμα για να μπορέσει το κύκλωμα testbench να δώσει δεδομένα εισόδου στο hasp θα πρέπει να μπορεί να οδηγήσει την αρτηρία τριών καταστάσεων. Επιπλέον για να διαβάσει τα "κλειδωμένα" δεδομένα θα πρέπει να

---

επιτρέψει στο hasp να οδηγήσει τη διαμοιραζόμενη αρτηρία ενώ αυτό θα έχει μπει στη θέση του ακροατή. Στον παρακάτω κώδικα δίνουμε ένα παράδειγμα τέτοιου testbench μαζί με ικανοποιητικά σύντομα σχόλια για τη πληρέστερη κατανόησή του.

```
module test ();
    reg      PWEn, PREn, PCLK ;
    reg [7:0] DATA;           // Δηλώσεις για τοπικούς καταχωρητές.
    tri [7:0] PD;
    // Η δήλωση για την αρτηρία τριών καταστάσεων. Προσέξτε ότι στο testbench δεν υπάρχει
    // δυνατότητα για inout και συνεπώς το μόνο που μπορεί να δηλωθεί το PD ώστε να
    // μπορεί να μπει σε κατάσταση υψηλής εμπέδησης είναι το tri.
    // Instantiation του hasp και διασύνδεσή του με τις τοπικές μεταβλητές
    hasp i0 (PWEn, PREn, PCLK, PD);

    // Η ανάθεση αυτή δείχνει το χειρισμό της γραμμής τριών καταστάσεων. Τα DATA
    // οδηγούνται πάνω στην αρτηρία μόνο όταν το σήμα PWEn είναι ενεργό.
    assign PD[7:0] = (PWEn == 1'b0) ? DATA[7:0] : 8'bz;

    initial
        begin
            PWEn <= 1'b1;
            PREn <= 1'b1;
            PCLK <= 1'b0;
            DATA <= 8'h55; //Οι 4 αυτές εντολές εκτελούνται τη χρονική στιγμή 0. Με τη τελευταία
            // βάζουμε τη τιμή 55HEX στα δεδομένα, ενώ και τα δύο modules είναι σε high Z κατάσταση
            # 50 PWEn <= 1'b0; // Η ενεργοποίηση του PWEn τη χρονική στιγμή 50 έχει σαν αποτέλεσμα
            // να περάσει στην αρτηρία και στα DIN του module hasp η τιμή των DATA.
            # 20 PCLK <= 1'b1; // Ενεργοποίηση του PCLK στη χρονική στιγμή 70 (50+20) προκαλεί
            // υπολογισμό της κρυπτογραφημένης τιμής στο DOUT του hasp.
            # 30 PCLK <= 1'b0;
            #100 PWEn <= 1'b1;
            // To testbench σταματάει να οδηγεί την αρτηρία και μπαίνει σε κατάσταση high Z.
            #50 PREn <= 1'b0; // Η αρτηρία υπό τον έλεγχο του hasp παίρνει τη τιμή των DOUT.
            #100 PREn <= 1'b1; // Η αρτηρία σε κατάσταση High-Z.
        end
    endmodule
```

---

Στο δεύτερο παράδειγμά μας σκοπός είναι η σχεδίαση ενός κυκλώματος για διόρθωση λαθών κωδικών λέξεων ενός Hamming κώδικα. Η είσοδός μας είναι 7 δυαδικά ψηφία (4 από αυτά είναι τα δεδομένα μας και τα υπόλοιπα 3 είναι τα δυαδικά ψηφία ελέγχου). Η έξοδός μας είναι το διορθωμένο μήνυμα και κάποια ένδειξη για το αν συνέβη ή όχι λάθος. Υποθέτουμε ότι ο σχεδιασμός μας δειγματοληπτεί τα δεδομένα με τη θετική ακμή του ρολογιού και παράγει τα αποτελέσματά του με την αρνητική ακμή του. Υπενθυμίζεται ότι μια κωδική λέξη 7 ψηφίων στο κώδικα Hamming έχει τη μορφή  $d_3\ d_2\ d_1\ c_2\ d_0\ c_1\ c_0$ , όπου  $c_0 = d_0 \wedge d_1 \wedge d_3$ ,  $c_1 = d_0 \wedge d_2 \wedge d_3$  και  $c_2 = d_1 \wedge d_2 \wedge d_3$ . Ο κώδικας που απαιτεί αυτός ο σχεδιασμός είναι ο ακόλουθος :

```

module HC (indata, corrected_data, clk, reset, cor_occurred);
    input [6:0] indata; //Είσοδος
    output [6:0] corrected_data; // Εξοδος : τα διορθωμένα δεδομένα
    input clk, reset; // Ρολόι και σήμα αρχικοποίησης
    output cor_occurred; //Ενδεικτης λάθους στην έξοδο

    reg [6:0] data; // Τοπική αποθήκη των δεδομένων
    reg [6:0] corrected_data;
    wire [2:0] syndrome; // Τα check bits που αντιστοιχούν στα δεδομένα που διαβάσαμε
    wire [6:0] invert_bit; // Μάσκα για την αντιστροφή του λανθασμένου bit

    always @(posedge clk or posedge reset) // Αποθήκευση των δεδομένων στον εσωτερικό
        if (reset) data <= 7'h00; // καταχωρητή με την θετική ακμή ρολογιού.
        else data <= indata;

    assign syndrome [2:0] = {data[6] ^ data[5] ^ data[4] ^ data[3], // Υπολογισμός των
                            data[6] ^ data[5] ^ data[2] ^ data[1], //check bits για τα
                            data[6] ^ data[4] ^ data[2] ^ data[0]}; // data

    // Το {} χρησιμοποιείται για το concatenation.

    assign cor_occurred = (syndrome[2:0] == 3'h0) ? 1'b0 : 1'b1;
    // Σύνδρομο ≠ 0 σημαίνει λάθος

    // Κατασκευή της μάσκας για την αντιστροφή του λάθους bit
    assign invert_bit [6:0] = (syndrome[2:0] == 3'h0) ? 7'h00 :
        (syndrome[2:0] == 3'h1) ? 7'h01 :
        (syndrome[2:0] == 3'h2) ? 7'h02 :
        (syndrome[2:0] == 3'h3) ? 7'h04 :
```

---

```

(syndrome[2:0] == 3'h4) ? 7'h08 :
(syndrome[2:0] == 3'h5) ? 7'h10 :
(syndrome[2:0] == 3'h6) ? 7'h20 : 7'h40;

// Διόρθωση τυχόν λάθους και οδήγηση των διορθωμένων δεδομένων με την αρνητική ακμή
always @(negedge clk or posedge reset)
    if (reset) corrected_data <= 7'h00;
    else corrected_data <= data ^ invert_bit;

endmodule

```

Αν και ο κώδικας για το σχεδιασμό αυτό είναι μάλλον απλός, η δυσκολία κατασκευής ενός testbench είναι σαφώς μεγαλύτερη. Κι αυτό γιατί ένα testbench θα πρέπει :

- ◆ Να γεννά κωδικές και όχι τυχαίες λέξεις στην είσοδο του σχεδιασμού.
- ◆ Να εισάγει / μην εισάγει λάθη.
- ◆ Οταν εισάγει λάθη να το κάνει με τυχαίο τρόπο.
- ◆ Να πιστοποιεί ότι οι αρχικές (χωρίς τα τυχαία λάθη λέξεις) και οι διορθωμένες είναι ίδιες και η ένδειξη λάθους συμβαδίζει με την εισαγωγή ή μη λάθους.

Παρακάτω δίνουμε το κώδικα για ένα testbench που υλοποιεί όλα πλην του τελευταίου στόχου.

```

module testbench();
    wire [6:0] data;
    reg      clk, reset;
    wire     cor_occurred;
    wire [6:0] corrected_data;
    reg [3:0] info;           // Τα αρχικά μας δεδομένα
    reg [2:0] error_position; // Τυχαία θέση λάθους
    reg      error_occur;    // 0 όχι λάθος, 1 λάθος
    integer   seed;          // Αρχική τιμή της γεννήτριας τυχαίων αριθμών
    wire [2:0] syndrome;    // Τα check bits
    wire [6:0] mask;         // Η μάσκα αντιστροφής ενός bit (εισαγωγής λάθους)

```

```

initial
begin
#5 clk = 0; reset = 0; info = 0; seed = 1;
#10 reset = 1; #10 reset = 0; // Αρχικοποίηση
end

```

---

```

HC CUT (data, corrected_data, clk, reset, cor_occured); // Ο σχεδιασμός που θα ελέγχουμε

always #40 clk = !clk; // Ρολόι με περίοδο 80 μονάδων χρόνου

always @ (negedge clk or posedge reset) // Επόμενο δεδομένο με κάθε αρνητική ακμή
    if (reset) info <= 4'b0;
    else info <= info + 1;

assign syndrome[2:0] = {info[1] ^ info [2] ^ info [3], // Δημιουργία check bits
                        info[0] ^ info [2] ^ info [3],
                        info[0] ^ info [1] ^ info [3]};

always @ (negedge clk or posedge reset) // Σε κάθε αρνητική ακμή με τυχαίο
    if (reset) // τρόπο αποφασίζεται ένα τυχαίο bit
        begin // και η θέση του
            error_position <= 3'b0;
            error_occur <= 1'b0;
            seed <= 1020;
        end
    else
        begin
            error_position <= $random (seed-200);
            error_occur <= $random (seed+30);
            seed <= $random (seed-12);
        end

assign mask [6:0] = (error_position[2:0] == 3'h0) ? 7'h00 :
                    (error_position[2:0] == 3'h1) ? {6'h0, error_occur} :
                    (error_position[2:0] == 3'h2) ? {5'h0, error_occur, 1'h0} :
                    (error_position[2:0] == 3'h3) ? {4'h0, error_occur, 2'h0} :
                    (error_position[2:0] == 3'h4) ? {3'h0, error_occur, 3'h0} :
                    (error_position[2:0] == 3'h5) ? {2'h0, error_occur, 4'h0} :
                    (error_position[2:0] == 3'h6) ? {1'h0, error_occur, 5'h0} :
                    {error_occur, 6'h0};

// Δημιουργία μάσκας όπου στην τυχαία θέση error_position περιέχει το τυχαίο bit error_occur
// Αν αυτό το bit είναι 1, τότε θα γίνει εισαγωγή λάθους. Αν όχι τότε η κωδική λέξη θα είναι σωστή

assign data [6:0] = {info[3:1], syndrome[2], info[0], syndrome[1:0]} ^ mask [6:0] ;

endmodule

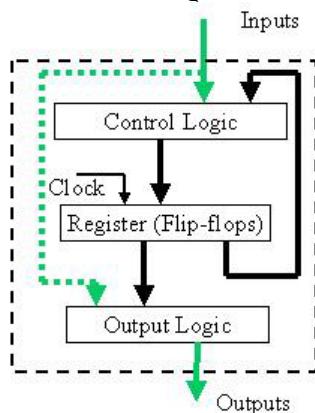
```

---

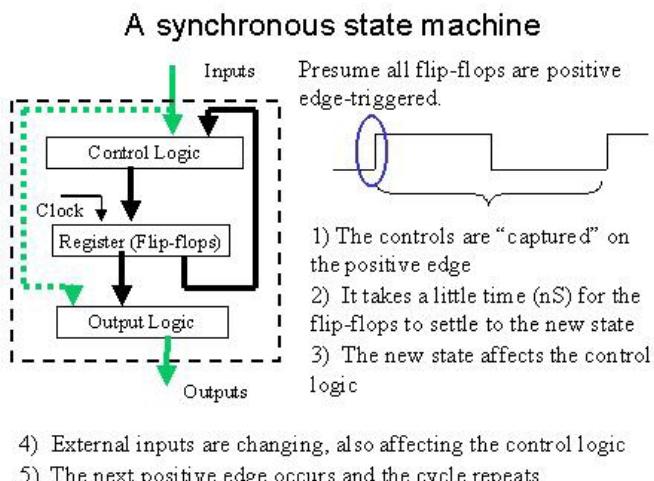
Κάθε σύγχρονο κύκλωμα με ένα μόνο ρολόι μπορεί να θεωρηθεί σα μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine – FSM). Τα ακολουθιακά στοιχεία του κυκλώματος αποτελούν τη μνήμη του κυκλώματος και το σύνολο των κατάστασεών τους αποτελεί τη κατάσταση της μηχανής. Η τρέχουσα κατάσταση μαζί με τις εισόδους, μέσω ενός συνδυαστικού κυκλώματος καθορίζουν την επόμενη κατάσταση της μηχανής. Οι έξοδοι του κυκλώματός μας μπορεί :

- ♦ Είτε να εξαρτώνται μόνο από την κατάσταση της μηχανής (Moore).
- ♦ Είτε να εξαρτώνται τόσο από τη κατάσταση της μηχανής όσο και από τις εισόδους του κυκλώματος (Mealy).

Σχηματικά δηλαδή έχουμε τις δύο αναπαραστάσεις :



όπου με πράσινο σημειώνεται το επιπλέον κύκλωμα που απαιτείται για τις μηχανές τύπου Mealy.

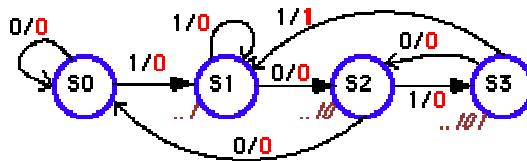


Το παραπάνω σχήμα απαριθμεί τα γεγονότα που συμβαίνουν κατά τη διάρκεια ενός κύκλου ενός σύγχρονου FSM, που λειτουργεί με την ανοδική ακμή του ρολογιού. Η ακμή αυτή πυροδοτεί τα flip-flop που μπαίνουν σε μια νέα τρέχουσα κατάσταση. Η νέα κατάσταση ορίζεται μαζί με τις εισόδους την έξοδο αλλά και τη λογική που γεννά

την επόμενη κατάσταση. Η επόμενη ανοδική ακμή επαναλαμβάνει το κύκλο αυτό από την αρχή.

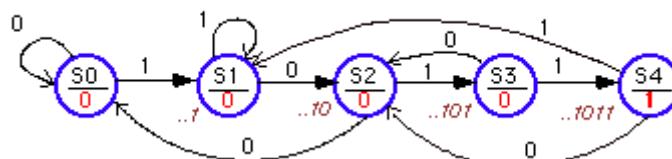
Τα FSMs συνήθως αναπαριστώνται γραφικά με διαγράμματα αλλαγής καταστάσεων (State Transition Diagram - STD). Το διάγραμμα αυτό αποτελείται από κόμβους που δείχνουν μία κατάσταση και ακμές που δείχνουν τις μεταβάσεις από τη μία κατάσταση στην άλλη. Πάνω στις ακμές αναγράφονται οι συνθήκες για τη μετάβαση, δηλαδή οι τιμές των εισόδων που προκαλούν αυτή τη μετάβαση και στην περίπτωση των Mealy μηχανών και τις εξόδους κατά τη μετάβαση.

Ας θεωρήσουμε το πρόβλημα αναγνώρισης του string 1011 από μία σειριακή είσοδο.



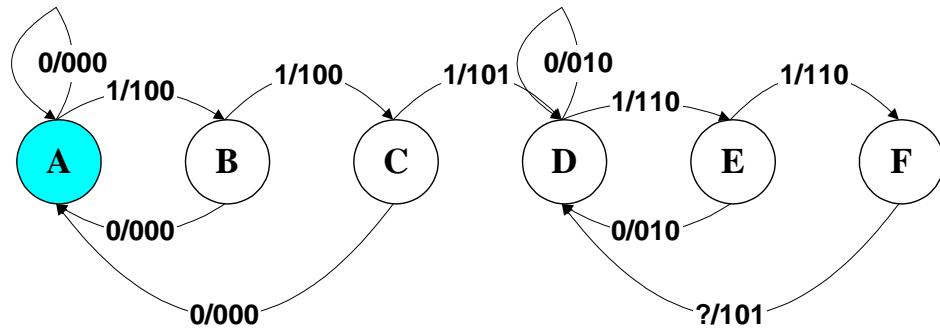
Το παραπάνω σχήμα δείχνει τη μηχανή Mealy που αναπαριστά ένα FSM για την επίλυση αυτού του προβλήματος. Η σημασιολογία των επιγραφών πάνω στις ακμές είναι ότι πριν τη κάθετο αναγράφονται οι είσοδοι και μετά την κάθετο οι έξοδοι του κυκλώματος. Ετσι όταν η μηχανή μας βρίσκεται στη κατάσταση S2 είσοδος 0 θα μας οδηγήσει στην κατάσταση S0 και θα μας δώσει έξοδο 0. Είσοδος 1 από την άλλη πλευρά θα μας οδηγήσει στην S3 και θα μας δώσει έξοδος 0. Από το σχήμα φαίνεται ότι έξοδο 1 (που σημαίνει αναγνώριση του string 1011) παράγονται μόνο κατά τη μετάβαση από την S3 στην S1 που γίνεται με έισοδο 1. Μιας και ο μοναδικός συνδυασμός ακμών που μας οδηγεί στην S3 από την S0 είναι ο 101 καταλαβαίνουμε ότι όντως η μηχανή μας αναγνωρίζει το string 1011 της εισόδου. Προσέξτε επίσης ότι η ακμή από το S3 στο S1 ταυτόχρονα υπονοεί ότι σε μια είσοδο της μορφής 1011011, το string 1011 θα αναγνωριστεί δύο φορές.

Η αντίστοιχη μηχανή Moore φαίνεται στο παρακάτω σχήμα.



Παρατηρείστε ότι σε αυτήν την περίπτωση η έξοδος εξαρτάται αποκλειστικά και μόνο από την κατάσταση στην οποία βρίσκεται το κύκλωμα και όχι από τις εισόδους. Αυτές μαζί με την τρέχουσα κατάσταση απλά καθορίζουν την επόμενη κατάσταση του κυκλώματος. Εντός των κόμβων του διαγράμματος αναγράφεται και η τιμή της εξόδου.

Παρακάτω αναζητούμε μηχανιστικούς τρόπους για τη περιγραφή FSM εκφρασμένων με STD σε συνθέσιμο ιώδικα Verilog. Είναι προφανές ότι από την παρατήρηση ενός STD, μπορούμε να βρούμε τον αριθμό των καταχωρητών κατάστασης που απαιτούνται (είναι το πάνω όριο του λογαρίθμου με βάση του 2 του αριθμού των κόμβων), τον αριθμό των εισόδων και των εξόδων του κυκλώματος. Ας θεωρήσουμε το STD που φαίνεται στο παρακάτω σχήμα. Η A είναι και η αρχική κατάσταση, δηλαδή η κατάσταση στην οποία πρέπει να μπει ο στοχευόμενος σχεδιασμός κατά την αρχικοποίηση του συστήματος.



Ο αριθμός των πιθανών καταστάσεων που μπορεί να βρεθεί η μηχανή μας είναι έξι (από A έως και F). Για την απομνημόνευση της κατάστασης στην οποία μπορεί να βρεθεί η μηχανή μας θα απαιτηθεί συνεπώς ένας καταχωρητής 3 δυαδικών ψηφίων. Δύο από τις καταστάσεις ωστόσο που θα κωδικοποιεί αυτός ο καταχωρητής θα είναι ανά πάσα στιγμή αδιάφορες. Είναι προφανές ότι μπορούμε να επιλέξουμε μία από πολλές διαφορετικές κωδικοποίησεις. Δύο από αυτές φαίνονται παρακάτω :

Κατάσταση	Πιθανή Κωδικοπίηση 1	Πιθανή κωδικοποίηση 2
A	000	010
B	110	101
C	100	000
D	011	110
E	111	001
F	001	011
Αδιάφορη 1	010	100
Αδιάφορη 2	101	111

Τέλος, τα τόξα (arcs) προδίδουν ότι το στοχευόμενο σύστημα έχει μία είσοδο και 3 εξόδους.

Ενας πιθανός κώδικας Verilog που περιγράφει το στοχευόμενο σύστημα θα μπορούσε να είναι ο ακόλουθος (έχουν προστεθεί σύντομα σχόλια για τη πληρέστερη περιγραφή) :

---

```

module fsm (i, clock, reset, out);
    input i, clock, reset;
    output [2:0] out;
    // Οι δηλώσεις για τις εισόδους και εξόδους του συστήματός μας

    // Δηλώσεις για τις μεταβλητές εξόδου (out), επόμενης (nstate) και τρέχουσας (cstate)
    // κατάστασης. Οι παρακάτω δηλώσεις μπορεί να είναι παραπλανητικές. Οι μεταβλητές
    // αυτές παρότι δηλώνονται ως καταχωρητές στη πραγματικότητα δεν είναι όλες. Η δήλωση
    // ωστόσο είναι συντακτικά απαραίτητη μιας και οι μεταβλητές αυτές εμφανίζονται στο
    // αριστερό μέρος non-blocking assignments που βρίσκονται μέσα σε always blocks. Εξ
    // αυτών μόνο ο καταχωρητής τρέχουσας κατάστασης (cstate) θα είναι ακολουθιακό
    // κύκλωμα.

    reg [2:0] out;
    reg [2:0] cstate, nstate;

    parameter [2:0] A=0, B=1, C=2, D=3, E=4, F=5;

    // Η δήλωση αυτή έχει σκοπό την αντικατάσταση των αριθμών με συμβολικά ονόματα. Εδώ
    // ωστόσο θα πρέπει να διευκρινιστεί ότι ούτε και οι αριθμοί έχουν απόλυτη σημασία.

    // Δηλαδή όταν το αρχείο Verilog περάσει από το πρόγραμμα σύνθεσης, αυτό θα καταλάβει
    // τη περιγραφή ενός FSM. Ανεξάρτητα των αριθμών ή των συμβολισμών που έχουν
    // χρησιμοποιηθεί το εργαλείο θα επιλέξει εκείνη τη κωδικοποίηση για τις καταστάσεις
    // A-F που εξυπρετεί τους στόχους του χρήστη, δηλαδή αυτή που οδηγεί σε μικρότερο
    // εμβαδό / καθυστέρηση / κατανάλωση ισχύος. Επιπλέον εκείνη τη στιγμή ο χρήστης
    // μπορεί να αποφασίσει να χρησιμοποιήσει μια κωδικοποίηση που θα χρησιμοποιεί
    // περισσότερα από τα ελάχιστα δυαδικά ψηφία (3 στο παράδειγμά μας) όπως για
    // παράδειγμα την one-hot-encoding, με σκοπό την ευκολότερη εκσφαλμάτωση του
    // σχεδιασμού του.

    // Το συνδυαστικό κύκλωμα του FSM.

    always @ (i or cstate)
        case (cstate)
            A : begin
                // Κάθε ζεύγος εντολών σαν αυτό που ακολουθεί περιγράφει το ζεύγος εξερχόμενων
                // arcs κάθε κόμβου του DFG.
                nstate = (i == 0) ? A : B;
                out = (i==0)? 3'h0 : 3'h4;
            end
            B : begin

```

---

```

nstate = (i ==0) ? A : C;
out = (i==0)? 3'h0 : 3'h4;
end

C : begin
    nstate = (i ==0) ? A : D;
    out = (i==0)? 3'h0 : 3'h5;
end

D : begin
    nstate = (i ==0) ? D : E;
    out = (i==0)? 3'h2 : 3'h6;
end

E : begin
    nstate = (i ==0) ? D : F;
    out = (i==0)? 3'h2 : 3'h6;
end

// Η επόμενη της F κατάσταση είναι η D ανεξαρτήτως εισόδου και η έξοδος θα
// είναι το δεκαεξαδικό 5.

F : begin
    nstate = D;
    out = 3'h5;
end

// Θα μπορούσαμε και να παραλείψουμε τη περιγραφή των αδιάφορων καταστάσεων. Σε
// περιπτώσεις εκσφαλμάτωσης όμως η προσθηκή του τι θα γίνει στη περίπτωση
// εμφάνισης των αδιάφορων καταστάσεων είναι σωτήρια.

default : begin
    nstate = A;
    out = 3'hX;
end

endcase

// Τέλος, το ακολουθιακό κύκλωμα του FSM. Ασύγχρονη είσοδος καθαρισμού και ανανέωση
// της κατάστασης του FSM με τη θετική ακμή του ρολογιού.

always @ (posedge clock or negedge reset)
    if (~reset) cstate <= A;
    else cstate <= nstate;

endmodule

```

---

---

Οπως βλέπουμε η απεικόνιση σε κώδικα Verilog ενός STD είναι μια πολύ απλή διαδικασία. Ωστόσο θα πρέπει να έχουμε υπ' όψιν μας ότι πολλές φορές υπάρχουν και άλλες λύσεις στο ίδιο πρόβλημα ενδεχόμενα πιο προσοδοφόρες. Ας επιστρέψουμε για λίγο στο παράδειγμα με την αναγνώριση του string 1011 που είδαμε πιο πάνω. Από το STD είναι πολύ εύκολο να φτάσουμε στον παρακάτω συνθέσιμο Verilog κώδικα (παρατίθεται χωρίς σχόλια), τον οποίο παρακάτω αναφέρουμε ως κώδικα 1 :

```
module str1011recognition1 (i, recognised, clk, reset);
    input i, clk, reset;
    output recognised;
    reg recognised;
    reg [1:0] cstate, nstate;

    parameter [1:0] S0=0, S1=1, S2=2, S3=3;

    always @(i or cstate)
        case (cstate)
            S0 : begin
                nstate = (i==0)? S0 : S1;
                recognised = 0;
            end
            S1 : begin
                nstate = (i==1)? S1 : S2;
                recognised = 0;
            end
            S2 : begin
                nstate = (i==0)? S0 : S3;
                recognised = 0;
            end
            S3 : begin
                nstate = (i==1)? S1 : S2;
                recognised = (i==1)? 1 : 0;
            end
        endcase

        always @(posedge clk or posedge reset)
            if (reset) cstate <= S0;
            else cstate <= nstate;

endmodule
```

---

Εναλλακτικά μπορούμε να χρησιμοποιήσουμε ένα serial to parallel καταχωρητή εισόδου με ένα συγκριτή στην έξοδό του. Ο κώδικας Verilog που χρειάζεται γι' αυτή τη περιγραφή είναι (στη συνέχεια αναφέρεται σα κώδικας 2) :

```
module str1011recognition2 (i, recognised, clk, reset);  
    input i, clk, reset;  
    output recognised;  
  
    reg [3:0] cons4inputs;  
  
    always @ (posedge clk or posedge reset)  
        if (reset) cons4inputs <= 4'h0;  
        else cons4inputs[3:0] <= {cons4inputs[2:0], i};  
  
    assign recognised = (cons4inputs == 4'b1011) ? 1'b1 : 1'b0;  
endmodule
```

Ας προσπαθήσουμε να δούμε τα πλεονεκτήματα και τα μειονεκτήματα κάθε κώδικα. Ο κώδικας 1 χρησιμοποιεί 2 flip-flops, ενώ ο 2 χρησιμοποιεί 4 flip-flops. Αντιθέτως, η συνδυαστική λογική που απαιτείται για το καθορισμό της επόμενης κατάστασης είναι στη περίπτωση του κώδικα 1 σημαντική ενώ σε αυτή του κώδικα 2 μηδενική. Από πλευράς εμβαδού, μιας και ένα flip-flop έχει σημαντικά μεγαλύτερο εμβαδόν όταν υλοποιείται σε VLSI CMOS τεχνολογία από ότι μια πύλη (περίπου τετραπλάσιο) συμπεριλαμβάνει ότι κατά πάσα πιθανότητα ο κώδικας 1 οδηγεί σε μικρότερο κύκλωμα από αυτό του κώδικα 2.

Στρέφοντας τη προσοχή μας στη συχνότητα λειτουργίας, μπορούμε να δούμε ότι στη περίπτωση του κώδικα 1, το κύκλωμά μας έχει μια συχνότητα που εξαρτάται από :

1. Το μονοπάτι υπολογισμού της επόμενης κατάστασης. Η καθυστέρηση αυτού του μονοπατιού είναι ίση με το χρόνο διάδοσης μέσω ενός flip-flop, το μεγαλύτερο χρόνο της συνδυαστικής λογικής για το καθορισμό της επόμενης κατάστασης και το χρόνο προετοιμασίας (setup) του flip-flop ή
2. Το μονοπάτι υπολογισμού των εξόδων. Η καθυστέρηση αυτού του μονοπατιού είναι ίση με το χρόνο διάδοσης μέσω ενός flip-flop και το μεγαλύτερο χρόνο της συνδυαστικής λογικής για το καθορισμό των εξόδων.

Στις περισσότερες περιπτώσεις το χειρότερο μονοπάτι είναι αυτό της περίπτωσης 1 πιο πάνω. Στη περίπτωση του κώδικα 2, υπάρχει μόνο το μονοπάτι υπολογισμού των εξόδων. Η καθυστέρηση αυτού του μονοπατιού είναι ίση με το χρόνο διάδοσης μέσω ενός flip-flop και το χρόνο μέσα από μια πύλη AND τεσσάρων εισόδων που κάνει τη

---

σύγκριση μεταξύ των παράλληλων δεδομένων και του επιθυμητού string. Συνεπώς αναμένουμε ότι ο κώδικας 2 θα οδηγεί σε πολύ ταχύτερες υλοποιήσεις.

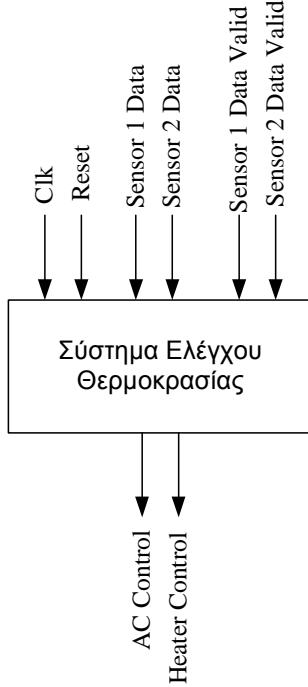
Ενας τρόπος για ασφαλείς συγκρίσεις είναι να συνθέσουμε τους δύο κώδικες στην ίδια τεχνολογία. Χρησιμοποιώντας μια τεχνολογία 0.5 micron, μπορούμε να πάρουμε τα εξής αποτελέσματα. Ο κώδικας 1 οδηγεί σε μια υλοποίηση που απαιτεί 38 ισοδύναμες πύλες και επιτυγχάνει συχνότητα 640 περίπου MHz. Η υλοποίηση του κώδικα 2 οδηγεί σε μια υλοποίηση που απαιτεί 45 ισοδύναμες πύλες και μπορεί να λειτουργήσει με συχνότητες έως και 971 MHz.

Θα κλείσουμε την αναφορά μας στη Verilog με τη παρουσίαση ενός πραγματικού παραδειγματος σχεδιασμού. Στο παρόντο παραδειγμα αυτό καλούμαστε να σχεδιάσουμε ένα σύστημα ελέγχου της θερμοκρασίας κάποιων χώρων, δηλαδή ένα σύστημα που θα ελέγχει τη λειτουργία των θερμαντικών και ψυκτικών μηχανημάτων των χώρων. Υποθέτουμε ότι :

- ♦ το σύστημα λαμβάνει τιμές θερμοκρασίας από 2 αισθητές και ότι σα τρέχουσα τιμή θερμοκρασίας θα πρέπει να θεωρεί τον μέσο όρο τους. Ενας A/D converter συνδέεται με κάθε αισθητή της θερμοκρασίας και η έξοδός του (ισοδύναμα η είσοδος του συστήματος μας) είναι σειριακή και έχει μήκος 6 δυαδικών ψηφίων με το περισσότερο σημαντικό δυαδικό ψηφίο να μεταδίδεται πρώτο (Sensor Data). Κάθε δυαδικό ψηφίο από κάθε αισθητή παράγεται με την ανοδική ακμή του ρολογιού και η όλη μετάδοση περικλείεται από ένα σήμα ετοιμότητας (Sensor Data Valid).
- ♦ υπάρχει ένα προγραμματιζόμενο κατώφλι θερμοκρασίας (στην παρακάτω λύση αυτό θεωρείται ότι είναι στους 21°C. Οταν η θερμοκρασία μας βρεθεί έως και 2 βαθμούς κάτω από αυτήν του κατώφλιού τότε το σύστημά μας θα πρέπει να απενεργοποιεί τα ψυκτικά μηχανήματα. Αντίστοιχα σε περίπτωση θερμοκρασίας έως και 2 βαθμούς πάνω από το κατώφλι θα πρέπει να απενεργοποιεί τα θερμαντικά σώματα. Σε περίπτωση 3 ή επιπλέον βαθμών κάτω (πάνω) από το κατώφλι το σύστημά μας θα πρέπει να ενεργοποιεί τα θερμαντικά (ψυκτικά) μηχανήματα.

Μια πρώτη προσέγγιση του στοχευόμενου συστήματος είναι να προχωρήσουμε στη σχεδίαση της διασυνδεσιμότητάς του με τα υπόλοιπα συστήματα. Σύμφωνα με τις υποθέσεις του παραδειγματος το σύστημά μας θα έχει 2 εισόδους (Sensor 1/2 Data) από τις οποίες θα διαβάζει τα σειριακά δεδομένα των A/D που υπάρχουν στους αισθητές, 2 εισόδους (Sensor 1/2 Data Valid) που δείχνουν πότε οι τιμές στις προηγούμενες εισόδους είναι έγκυρες, 1 ρολόι (clk), ένα σήμα αρχικοποίησης (reset) και 2 εξόδους, μία για τον έλεγχο των ψυκτικών (AC Control) και μία για τον έλεγχο

των θερμοκρασίας (Heater Control) μηχανημάτων. Συνεπώς γραφικά το στοχευόμενο σύστημα θα έχει την ακόλουθη μορφή :



Μερικές επιλέον υποθέσεις είναι απαραίτητες για το καθορισμό της λογικής των σημάτων αλλά και των χρονισμών. Στη λύση που ακολουθεί υποθέτουμε επιπλέον ότι :

- ♦ Χρησιμοποιείται θετική λογική για τον έλεγχο των θερμοκρασίας και ψυκτικών σωμάτων, δηλαδή το αντίσταοιχο σήμα οδηγείται στο 1 για να υποδειξει ενεργοποίηση.
- ♦ Για την ανάγνωση των σειριακών δεδομένων θεωρούμε ότι το πρώτο δεδομένο είναι συγχρονισμένο με την πρώτη ανοδική ακμή που ακολουθεί την άνοδο του σήματος εγκυρότητας. Το σήμα εγκυρότητας παραμένει στο 1 μέχρι και την ανάγνωση του διαδικού ψηφίου και κατόπιν οδηγείται στο 0.
- ♦ Το σύστημά μας πριν προβεί σε οποιαδήποτε ενέργεια θα πρέπει πρώτα να έχει διαβάσει ένα δείγμα θερμοκρασίας από κάθε αισθητή.
- ♦ Το σύστημά μας χρησιμοποιεί ένα μόνο ρολόι και η είσοδος αρχικοποίησης είναι σύγχρονη.

Εχοντας όλες αυτές τις υποθέσεις, δεύτερο βήμα προσέγγισης του στοχευόμενου σχεδιασμού είναι η δημιουργία της λειτουργικότητάς του σε ένα πολύ υψηλό επίπεδο. Στο παρόν παράδειγμα διαλέγουμε τη συγγραφή με ψευδοκώδικα :

---

<b>Βήμα</b>	<b>Ενέργεια</b>
Αρχικοποίηση	Απενεργοποίηση όλων των συσκευών
1	Αναμονή μέχρι την ένδειξη για έγκυρα δεδομένα από κάποιο αισθητή. (Προσοχή στο ότι μπορεί να υπάρξουν παράλληλα ενδείξεις και από τους δύο αισθητές).
2	Εχουμε διαβάσει δεδομένα και από τους δύο αισθητές ? Αν όχι πήγαινε στο Βήμα 1.
3	Υπολόγισε τη μέση θερμοκρασία από τα δεδομένα των δύο αισθητών.
4	Ανάλογα μα τη σχέση της μέσης θερμοκρασίας με το κατώφλι ενεργοποίησε / απενεργοποίησε τα θερμαντικά / ψυκτικά σώματα. Πήγαινε στο βήμα 2.

Είναι προφανές ότι μπορούμε πλέον να περάσουμε στη συγγραφή του κώδικα με βάση το παραπάνω ψευδοκώδικα. Ωστόσο προτιμούμε να διαπιστώσουμε ότι ο παραπάνω ψευδοκώδικας εμπεριέχει μια μηχανή πεπερασμένων καταστάσεων. Οι καταστάσεις αυτές είναι αυτή της αναμονής για νέο δεδομένο από τους αισθητές (start), της ανάγνωσης δεδομένων από τις σειριακές γραμμές (read\_data) και του υπολογισμού των σημάτων ελέγχου (comp\_out). Ας δούμε με λίγα επιπλέον σχόλια όλο το κώδικα σε γλώσσα Verilog γι' αυτό το παράδειγμα.

```
module TempControl (Sensor_Data, Sensor_Data_Valid, Clk, Reset, AC_ctrl, Heater_ctrl);
    input [1:0] Sensor_Data;
    input [1:0] Sensor_Data_Valid;
    input     Clk;
    input     Reset;
    output    AC_ctrl;
    output    Heater_ctrl;
    // Οι δηλώσεις για τις εισόδους και εξόδους του συστήματός μας
    reg AC_ctrl, Heater_ctrl;
    // Οι δύο εξοδοί μας θα φυλάγονται σε καταχωρητές.
    parameter threshold = 21;
    // Με απλή τροποποίηση αυτής της γραμμής καθορίζεται διαφορετικό κατώφλι.
    reg [1:0] current_state, next_state;
    // Χρησιμοποιούνται για τη μηχανή πεπερασμένων καταστάσεών μας. Προσοχή : παρότι το
    // next_state δηλώνεται σα register δεν υλοποιείται με καταχωρητές μα σα συνδυαστική
    // λογική. Η δήλωση γίνεται για συντακτικούς λόγους της Verilog.
```

---

```

reg [5:0] read_data_0, read_data_1;
// Οι δύο αυτοί καταχωρητές χρησιμοποιούνται για την αποθήκευση των τιμών που
// διαβάζονται από τις σειριακές εισόδους. Χρειάζονται δύο μιας και η διαδικασία
// ανάγνωσης από τους δύο αισθητές μπορεί να εξελίσσεται παράλληλα

wire [6:0] sum;
wire [5:0] difference;

assign sum = read_data_1 + read_data_0;
assign difference = sum[6:1] - threshold ;

// Η άθροιση των δύο δειγμάτων οδηγεί σε αποτέλεσμα 7 bit (sum). Τα 6 MSB μας δίνουν
// τη μέση τιμή. Η διαφορά με το κατώφλι θα μας οδηγήσει στη διαμόρφωση των εξόδων μας
// Εδώ ζεκινά η περιγραφή του FSM μας με τις τρεις καταστάσεις.
parameter [1:0] start = 0, read_data = 1, comp_out =2;

// Η ακολουθιακή λογική πρώτα. Το FSM τυχόν αλλάζει καταστάσεις με την ανοδική ακμή
// του ρολογιού.

always @(posedge Clk)
if (Reset) current_state <= start;
else current_state <= next_state;

// Η συνδυαστική λογική. Εδώ χρησιμοποιείται ένας νέος εισαγόμενος καταχωρητής. Ο
// καταχωρητής αυτός θα μας δείχνει πότε έχουμε ολοκληρώσει διάβασμα και από τους δύο
// αισθητές θερμοκρασίας.

reg all_complete;
always@(current_state or Sensor_Data_Valid or all_complete)
case (current_state)
start : next_state = (Sensor_Data_Valid) ? read_data : start;
read_data : next_state = (all_complete) ? comp_out : (Sensor_Data_Valid) ? read_data : start;
comp_out : next_state = start ;
default : next_state = start;
endcase
// Φεύγουμε από τη κατάσταση αρχικοποίησης, όταν ένας από τους δύο αισθητές μας δώσει
// ένδειξη νέου δείγματος και προχωράμε στον υπολογισμό νέων τιμών όταν διαβάσουμε και
// από τους δύο αισθητές.

```

---

```

// Ο all_complete καταχωρητής οδηγείται στο 1 μόνο αν έχουμε διαβάσει δείγμα και από
// τους δύο αισθητές. Εισάγουμε 2 καταχωρητές που ο καθένας δείχνει ότι διαβάσαμε
// δείγμα από τον αντίστοιχο αισθητή. Προσέξτε επίσης λίγο το χρονισμό. Η απόφαση για
// τη τιμή του all_complete παίρνεται στην καθοδική ακμή του ρολογιού, έτσι ώστε η
// επόμενη ανοδική ακμή να επιτρέψει στη μηχανή πεπερασμένων καταστάσεων να μπει στην
// κατάσταση comp_out.

reg s0_sample_complete, s1_sample_complete;

always @(negedge Clk)
  if (Reset) all_complete <= 1'b0;
  else if (s0_sample_complete & s1_sample_complete) all_complete <= 1'b1;

// Ο προφανής κώδικας για τον υπολογισμό των εξόδων
always @(negedge Clk)
  if (Reset) begin AC_cntl<= 1'b0; Heater_cntl<=1'b0; end
  else if (current_state == comp_out) begin
    if (difference >= 3) begin AC_cntl<= 1'b1; Heater_cntl<=1'b0; end
    else if (difference <= -3) begin AC_cntl<= 1'b0; Heater_cntl<=1'b1; end
    else begin AC_cntl<= 1'b0; Heater_cntl<=1'b0; end
  end
  // Προσέξτε και πάλι το χρονισμό. Η αλλαγή των εξόδων γίνεται μισό κύκλο (στην
  // καθοδική ακμή) αφότου το FSM μας μπει στη κατάσταση comp_out.

// Το μόνο που απομένει στην περιγραφή μας είναι ο ορισμός των s0/s1_sample_complete.
// Κάθε δείγμα αποτελείται από 6 δυαδικά ψηφία. Για τη μέτρησή τους θα
// χρησιμοποιήσουμε τους δύο ακόλουθους down counters.

reg [2:0] count0, count1;

// Στην αρχικοποίηση οι δύο μετρητές παίρνουν τιμή 6 και τα δείγματα τιμή 21
always @(posedge Clk)
  if (Reset)
    begin
      count0 <= 3'h6;
      count1 <= 3'h6;
      read_data_0 [5:0] <= 6'd21;
      read_data_1 [5:0] <= 6'd21;
      s0_sample_complete <= 1'b0;
      s1_sample_complete <= 1'b0;
    end

```

---

---

```

else
// Εξετάζουμε αν έχουμε ένδειξη εγκυρότητας
begin
if (Sensor_Data_Valid[0])
begin
// Αν δεν έχουμε ήδη πάρει 6 δυαδικά ψηφία
if (count0)
begin
// Serial to parallel με ολισθηση αριστερά σε κάθε κύκλο λόγω MSB first
read_data_0 [5:0] <= {read_data_0 [4:0], Sensor_Data[0]};
count0 <= count0 - 1;
end
else
// Εχουμε πάρει δείγμα από τον αισθητή 0. Μόλις πάρουμε και από τον 1
// πήγαινε στην αρχική κατάσταση
begin
if (all_complete)
begin
s0_sample_complete <= 1'b0;
count0 <= 3'h6;
end
else s0_sample_complete <= 1'b1;
end
end
else
begin
if (Sensor_Data_Valid[1])
begin
// Αν δεν έχουμε ήδη πάρει 6 δυαδικά ψηφία
if (count1)
begin
// Serial to parallel με ολισθηση αριστερά σε κάθε κύκλο λόγω MSB first
read_data_1 [5:0] <= {read_data_1 [4:0], Sensor_Data[1]};
count1 <= count1 - 1;
end
else

```

---

```
// Εχουμε πάρει δείγμα από τον αισθητή 1. Μόλις πάρουμε και από τον 0
// πήγαινε στην αρχική κατάσταση
begin
  if (all_complete)
    begin
      s1_sample_complete <= 1'b0;
      count1 <= 3'h6;
    end
    else s1_sample_complete <= 1'b1;
  end
end
end
endmodule
```

**Μέρος Β'**

**Εργαστηριακές Ασκήσεις**



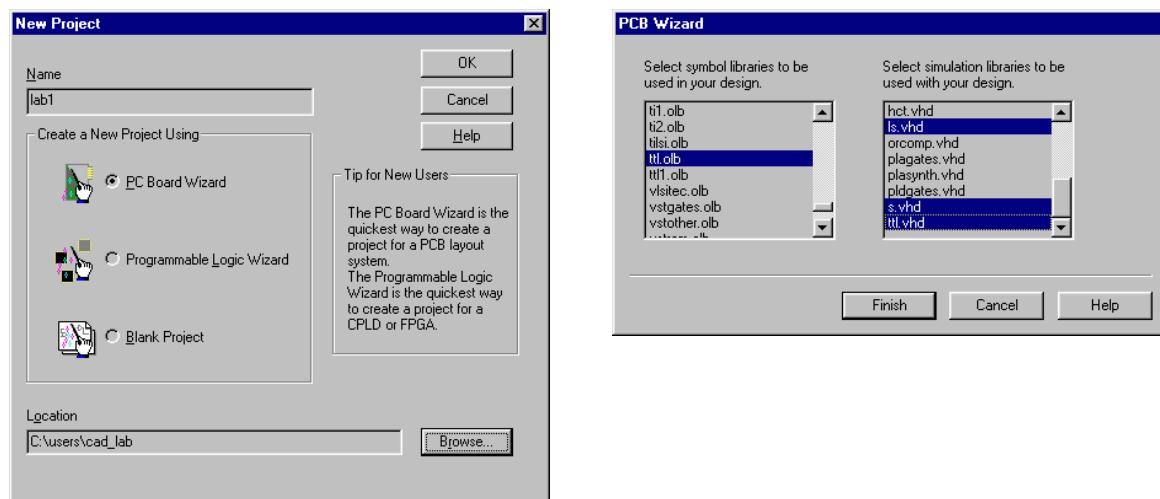
# ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 1

Σκοπός της εργαστηριακής άσκησης είναι η εξοικείωσή σας με την γραφική περιγραφή υλικού, την εξομοίωση ενός σχεδιασμού, την μέτρηση της καθυστέρησης διάδοσής του και τη διαδικασία εκσφαλμάτωσης. Στην άσκηση αυτή θα κληθείτε να εισάγετε γραφικά το σχηματικό ενός κυκλώματος μιας πύλης AND 74xx, 74Sxx, 74Asxx, 74LSxx και 74ALSxx.

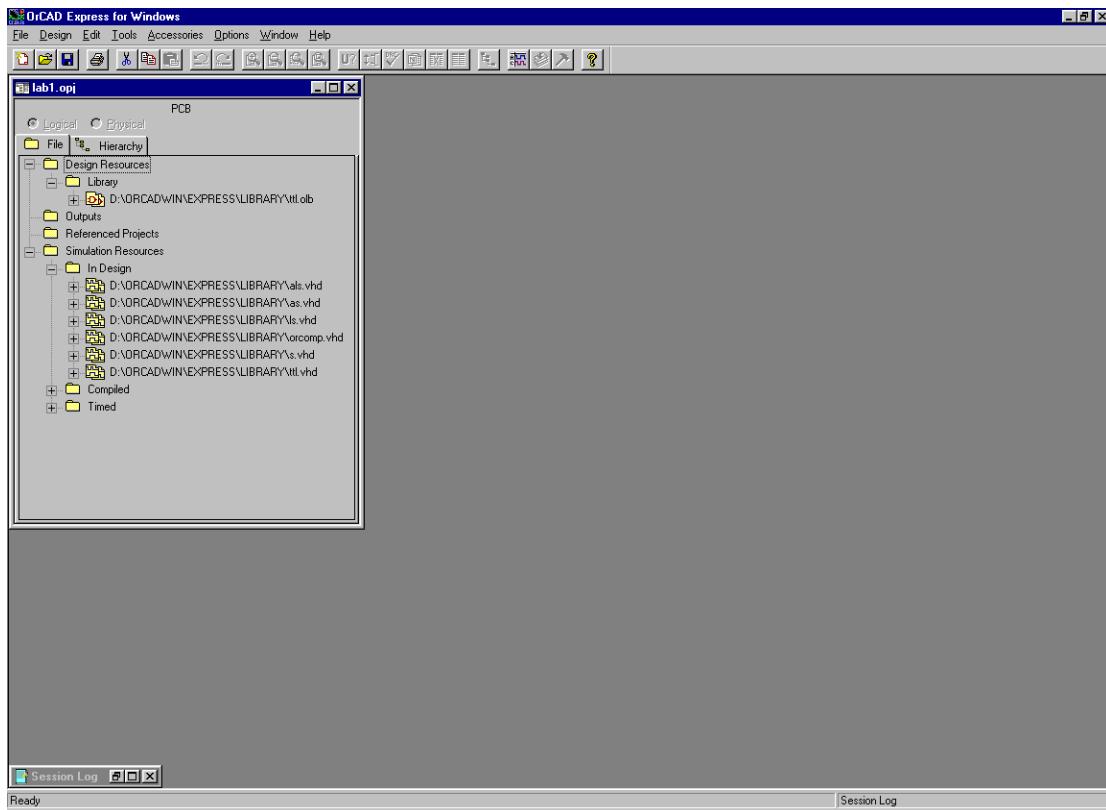
## Α. Εκκίνηση προγράμματος - Δημιουργία νέου σχεδιασμού

Εκτελούμε το πρόγραμμα Orcad Express for Windows. Το πρόγραμμα αυτό ακολουθεί την φιλοσοφία των σύγχρονων παραθυρικών εφαρμογών και πολλές λειτουργίες γίνονται όπως και στα άλλα προγράμματα που λειτουργούν κάτω από τα Microsoft Windows.

Όλα τα αρχεία που έχουν σχέση με ένα σχεδιασμό (σχηματικά, stimulus files, κον.) αποθηκεύονται από το Orcad με ενιαίο τρόπο με την μορφή project. Στην αρχική οθόνη που εμφανίζεται επιλέγουμε File -> New Project και δίνουμε το όνομα του project μας και το directory στο οποίο θέλουμε να βρίσκεται.



Από το μενού που προκύπτει επιλέγουμε PC Board Wizard. Με άλλα λόγια κατευθύνουμε το εργαλείο προς πλακέτα σαν πλατφόρμα υλοποίησης. Στην επόμενη οθόνη επιλέγουμε τις βιβλιοθήκες συμβόλων - symbol libraries που θέλουμε να χρησιμοποιήσουμε στον σχεδιασμό μας (στην συγκεκριμένη περίπτωση το ttl.old που περιέχει σχηματική πληροφορία για τις TTL πύλες) και τις βιβλιοθήκες εξομοίωσης - simulation libraries (στην συγκεκριμένη περίπτωση τις ttl.vhd, als.vhd, as.vhd, s.vhd, ls.vhd που περιέχουν τα μοντέλα χρονικής καθυστέρησης για τις οικογένειες standard TTL, ALS, AS, S και LS TTL αντίστοιχα).



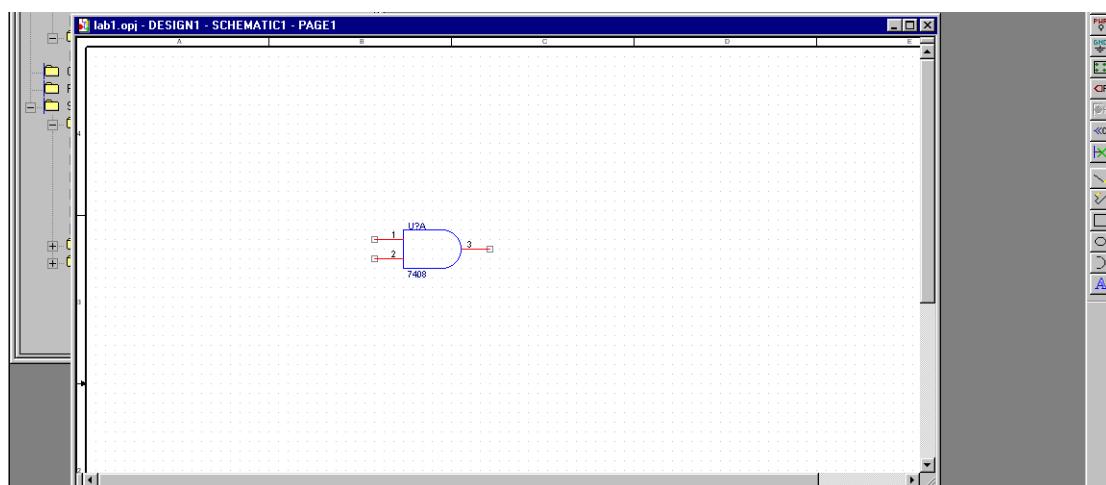
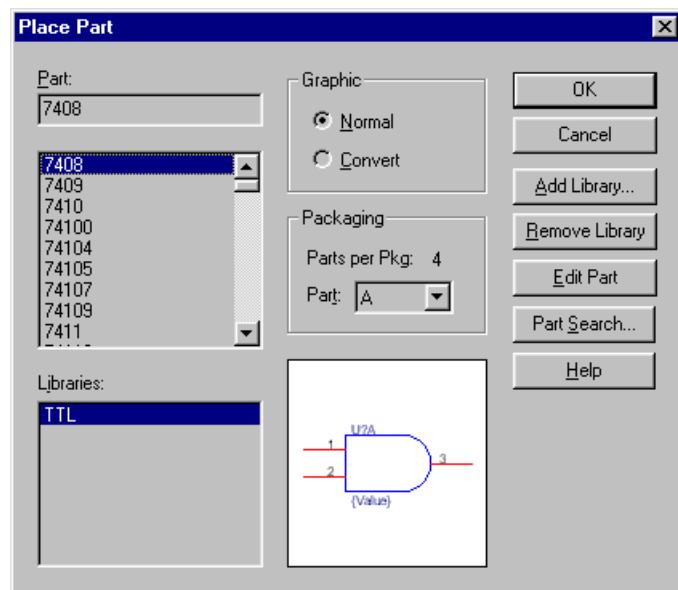
Στο αριστερό μέρος της οθόνης εμφανίζεται ο hierarchical browser. Σε αυτόν βρίσκεται συγκεντρωμένη όλη η πληροφορία που έχει σχέση με τον σχεδιασμό σας. Ο φάκελος Design Resources αποτελείται από τα αρχεία που περιέχουν την περιγραφή του κυκλώματος σε εσωτερική αναπαράσταση του πακέτου καθώς και τις βιβλιοθήκες συμβόλων που έχετε επιλέξει να χρησιμοποιήσετε σε αυτόν. Ο φάκελος Design Resources περιέχει μόνο σχηματική πληροφορία και πληροφορία συνδέσεων. Οι πληροφορίες για τα μοντέλα χρονικής καθυστέρησης βρίσκονται στον φάκελο Simulation Resources όπου θα βρείτε τρία directories (Compiled, In Design, Timed) που περιέχουν τις βιβλιοθήκες εξομοίωσης που είχατε επιλέξει προηγουμένως καθώς επίσης και την orcomp.vhd που χρησιμοποιείται εξ ορισμού από το Orcad.

## B. Γραφική εισαγωγή σχεδιασμού

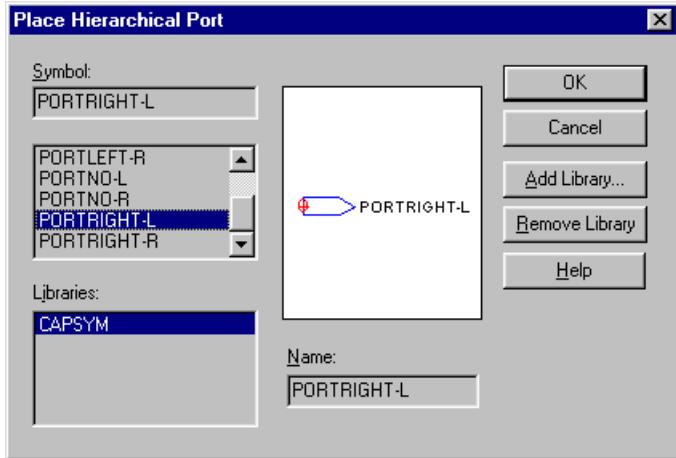
Προκειμένου να εισάγουμε το σχηματικό μας, επιλέγουμε το Design Resources και δίνουμε File -> New -> Design. Εμφανίζεται μία κενή σελίδα του graphic editor (αποτελεί την σελίδα 1 του σχηματικού 1 του σχεδιασμού μας) στην οποία θα σχηματίσουμε το κύκλωμα μας. Παρατηρήστε ότι ταυτόχρονα ενημερώνεται και ο hierarchical browser.

Κάθε κύκλωμα αποτελείται κυρίως από στοιχειώδη κομμάτια (parts), θύρες (ports) για τις εισόδους και τις εξόδους και καλώδια (wires) για τις συνδέσεις των παραπάνω. Τοποθετούμε τις πύλες μας στην σελίδα δίνοντας Place -> Part και επιλέγοντας την πύλη που θέλουμε. Στην συγκεκριμένη περίπτωση δίνουμε 7408. Επειδή κάθε part περιέχει 4

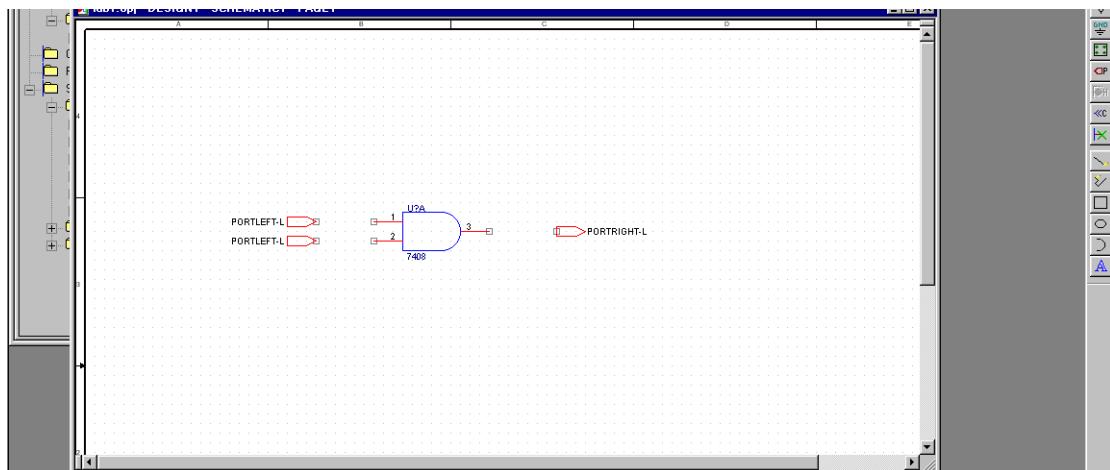
πύλες AND το εργαλείο αυτόματα ανακαλεί 1 πύλη AND και όχι ένα ολοκληρωμένο 7408. Για μεγαλύτερα στοιχειώδη κομμάτια (π.χ. 74244) κάθε part είναι και ένα αυτόνομο κομμάτι σχεδιασμού. Δίνοντας OK τοποθετούμε την πύλη AND στο σημείο που θέλουμε. Μπορούμε να τοποθετήσουμε δύες πύλες AND θέλουμε με αυτόν τον τρόπο στην σελίδα. Προκειμένου να τερματίσουμε την λειτουργία Place Part επιλέγουμε με το δεξί κουμπί του ποντικιού End Mode. Εάν κατά λάθος τοποθετήσαμε περισσότερες πύλες από αυτές που χρειαζόμαστε μπορούμε να επιλέξουμε με το ποντίκι τις περιττές πύλες και να τις διαγράψουμε με Delete. Παρατηρήστε ότι στο πάνω μέρος της πύλης εμφανίζεται το όνομα του ολοκληρωμένου –IC- U?Α ενώ στις εισόδους και εξόδους της εμφανίζονται οι αριθμοί των ακροδεκτών του IC.



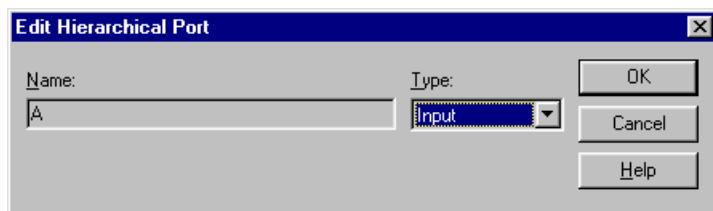
Στην συνέχεια δίνουμε Place -> Hierarchical Port για να τοποθετήσουμε τα I/O του κυκλώματός μας. Επιλέγουμε και τοποθετούμε το PORTRIGHT-L για την έξοδο και το PORTLEFT-L για τις δύο εισόδους. Με το δεξί κουμπί του ποντικιού και End Mode τερματίζουμε και αυτή την λειτουργία.



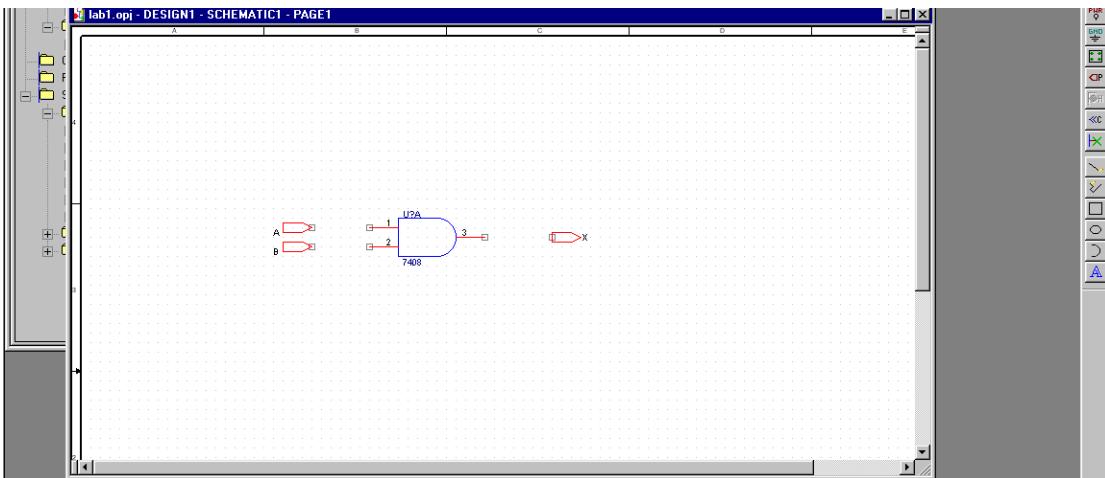
Προκειμένου να περιστρέψουμε οποιοδήποτε object στο σχήμα (port, part, wire) το επιλέγουμε και με το δεξί κουμπί του ποντικιού δίνουμε την αντίστοιχη λειτουργία (Rotate, Mirror Horizontally, Mirror Vertically).



Με διπλό κλικ πάνω στα ports δίνουμε όνομα A & B και τύπο Input για τις εισόδους και όνομα X και τύπο Output για την έξοδο.



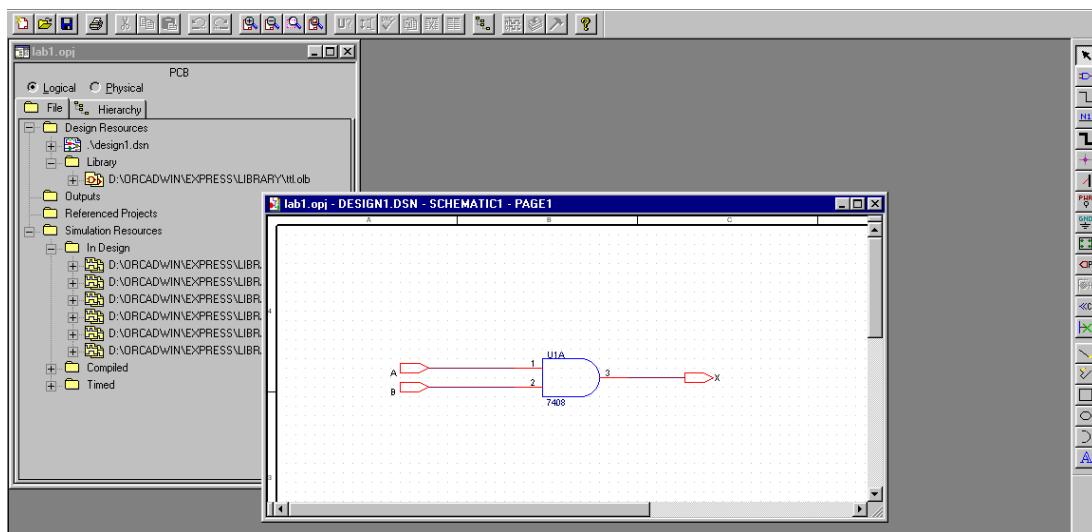
Το κύκλωμα μας θα πρέπει να φαίνεται πλέον όπως παρακάτω:



Στην συνέχεια συνδέουμε τα ports με την πύλη AND με wires. Δίνουμε Place-> Wire. Παρατηρήστε ότι στην σελίδα του σχηματικού μας ο cursor έχει μετατραπεί από βέλος σε σταυρό. Ενώνουμε τα άκρα που θέλουμε. Αυτό γίνεται κάνοντας κλικ με το ποντίκι εκεί όπου θέλουμε να ξεκινάει το καλώδιο και σύροντας το ποντίκι μέχρι εκεί που θέλουμε και κάνοντας κλικ για να τερματίσει. Τερματίζουμε την λειτουργία αυτή με End Wire (δεξιά κουμπί). Εάν κατά λάθος δημιουργήσατε καλώδια που δεν χρειάζονται μπορείτε να τα επιλέξετε με το ποντίκι και να τα διαγράψετε με Delete. Οι παραπάνω λειτουργίες μπορούν να γίνουν εκτός από το μενού Place και με την βοήθεια των εικονιδίων στην εργαλειοθήκη που βρίσκεται στο δεξιό μέρος της οθόνης.

Αφού τελειώσουμε τα παραπάνω, επιλέγουμε το Design Resources -> ./design1.dsn στον Hierarchical Browser και επιλέγουμε Tools -> Update Part References. Αφού δώσουμε OK και αποθηκεύσουμε το κύκλωμα και τις αλλαγές στο project μας παρατηρούμε ότι ενημερώνονται τα ονόματα των διαφόρων πυλών (π.χ. το U?A γίνεται U1A που σημαίνει η πρώτη πύλη από το A 7408 ολοκληρωμένο).

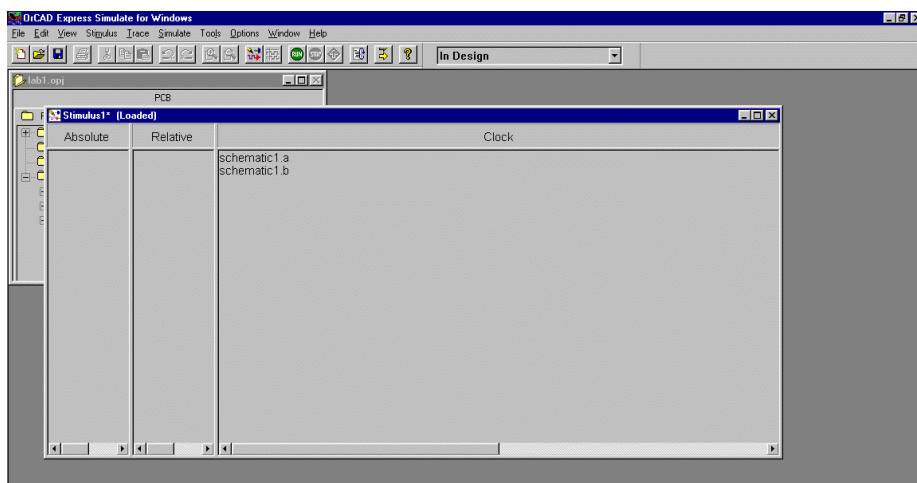
Το κύκλωμα πλέον θα πρέπει να φαίνεται όπως παρακάτω:



## Γ. Εξομοίωση

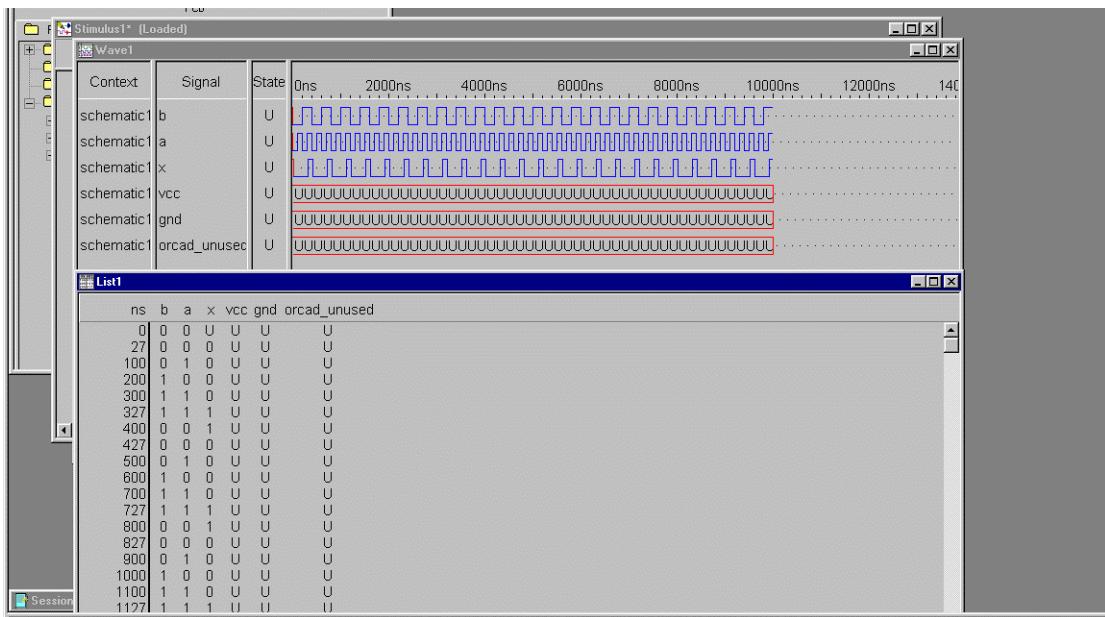
Αφού επιλέξουμε τον Hierarchical Browser δίνουμε Tools -> Simulate και αυτόματα εκτελείται ο Express Simulator. Επιλέγουμε In Design εξομοίωση και δίνουμε yes για να φορτωθεί ο σχεδιασμός μας. Στην προκειμένη περίπτωση και η επιλογή Compiled θα οδηγούσε στην ίδια εξομοίωση, αφού το κύκλωμά μας δεν περιλαμβάνει κανένα επίπεδο αφαίρεσης. Από το μενού Options->Preferences μπορούμε να επιλέξουμε το μοντέλο χρονικής καθυστέρησης (minimum, typical ή maximum). Επιλέξτε τυπική καθυστέρηση.

Πριν κάνουμε την εξομοίωση θα πρέπει να ετοιμάσουμε τις κυματομορφές των εισόδων μας. Αυτό γίνεται με την επιλογή Stimulus -> New Interactive. Μπορούμε να ορίσουμε τριάντα ειδών κυματομορφές: Absolute, Relative και Clock ανάλογα με το είδος της κυματομορφής εισόδου που θέλουμε. Οι κυματομορφές φαίνονται στο παρόντα Stimulus1.

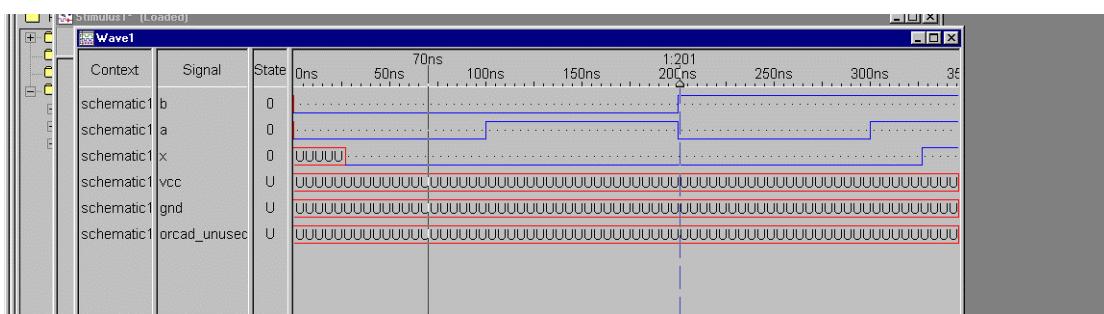


Επιλέγουμε το Clock Stimulus και αφού επιλέξουμε την είσοδο που θέλουμε π.χ. το Α ορίζουμε κυματομορφή με 100ns low και 100ns high που επαναλαμβάνεται συνέχεια και δίνουμε Add. Αντίστοιχα για την είσοδο Β δίνουμε 200ns low και 200ns high.

Αφού φορτώσουμε το stimulus file εκτελούμε την εξομοίωση για χρόνο 10000ns δίνοντας Simulate -> Run. Στο παρόντα Wave1 εμφανίζονται τα αποτελέσματα σε μορφή κυματομορφής ενώ στο παρόντα List1 εμφανίζονται οι χρόνοι στους οποίους γίνεται αλλαγή κατάστασης στα σήματα εισόδου ή εξόδου.



Με τις επιλογές View -> Zoom In και View -> Zoom Out μπορούμε να μεγεθύνουμε ή να μικρύνουμε την ορατή περιοχή για μεγαλύτερη λεπτομέρεια. Επιλέγουμε ένα σημείο της κυματομορφής και πατάμε το αριστερό κουμπί του ποντικιού. Εμφανίζεται ο marker που μας δείχνει σε ποιο χρονικό σημείο βρισκόμαστε ήδης φορά ενώ οι τιμές των σημάτων φαίνονται στην στήλη State. Επιλέγοντας τον marker, μπορούμε να τον σύρουμε σε όποιο χρονικό σημείο θέλουμε. Προκειμένου να μετρήσουμε την διαφορά μεταξύ δύο σημείων της κυματομορφής επιλέγουμε με το δεξί κουμπί του ποντικιού Add Delta Marker. Στο κάτω μέρος της οθόνης υπάρχει η απόσταση μεταξύ των δύο markers. Εάν θέλουμε μπορούμε να προσθέσουμε και δεύτερο Delta Marker. Οι Delta Markers μετράνε ήδης φορά την απόσταση από τον Marker.



Μετρήστε την καθυστέρηση της πύλης όταν μεταβαίνει στο λογικό μηδέν και όταν μεταβαίνει στο λογικό ένα και καταγράψτε την στον αντίστοιχο πίνακα.

Κλείστε τον εξομοιωτή κάνοντας Close και αφού αποθηκεύστε το stimulus file στο αρχείο stimulus.stm, το προσθέστε στο project και μετά το κάνετε unload από τον εξομοιωτή.

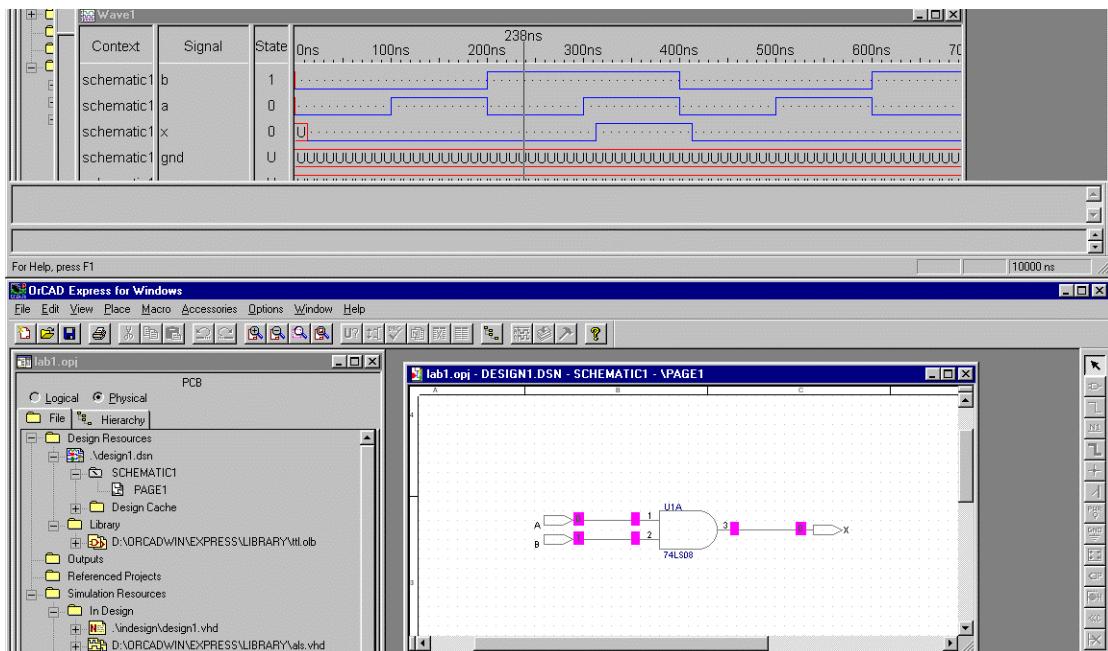
Επαναλάβατε την παραπάνω διαδικασία για τις πύλες 74LS08, 74S08, 74AS08 και 74ALS08 και μετρήστε την καθυστέρηση διάδοσης τους. Επιβεβαιώστε τους χρόνους που

μετρήσατε ανοίγοντας την αντίστοιχη βιβλιοθήκη εξομοίωσης και επιλέγοντας την πύλη που θέλετε μέσα από τον Hierarchical Browser. Συμπληρώστε τον παρακάτω πίνακα:

	7408	74ALS08	74AS08	74S08	74LS08
Χρόνος καθυστέρησης μετάβασης στο λογικό 0					
Χρόνος καθυστέρησης μετάβασης στο λογικό1					

### Δ. Διαδικασία Εκσφαλμάτωσης

Προκειμένου να ελέγχουμε την ορθότητα του σχεδιασμού μας και να εντοπίζουμε εύκολα και γρήγορα λάθη, μπορούμε να χρησιμοποιήσουμε την διαδικασία της διαγώνιας έρευνας (Cross Probing). Προκειμένου να το πετύχουμε αυτό αφού εκτελέσουμε την εξομοίωση επιλέγουμε Window -> Split Screen ώστε να φαίνονται ταυτόχρονα στην οθόνη τόσο οι κυματομορφές όσο και το σχηματικό του κυκλώματός μας. Επίσης επιλέγουμε στον Hierarchical Browser την φυσική απεικόνιση (Physical) και όχι την λογική (Logical). Παρατηρούμε ότι κινώντας τον marker στα διάφορα σημεία της κυματομορφής εμφανίζονται οι τιμές των σημάτων εισόδου και εξόδου πάνω στο σχηματικό μας. Αντίστοιχα εάν επιλέξουμε κάποιο από τα σήματα στην κυματομορφή αυτόματα επιλέγεται το αντίστοιχο σήμα στο σχηματικό.



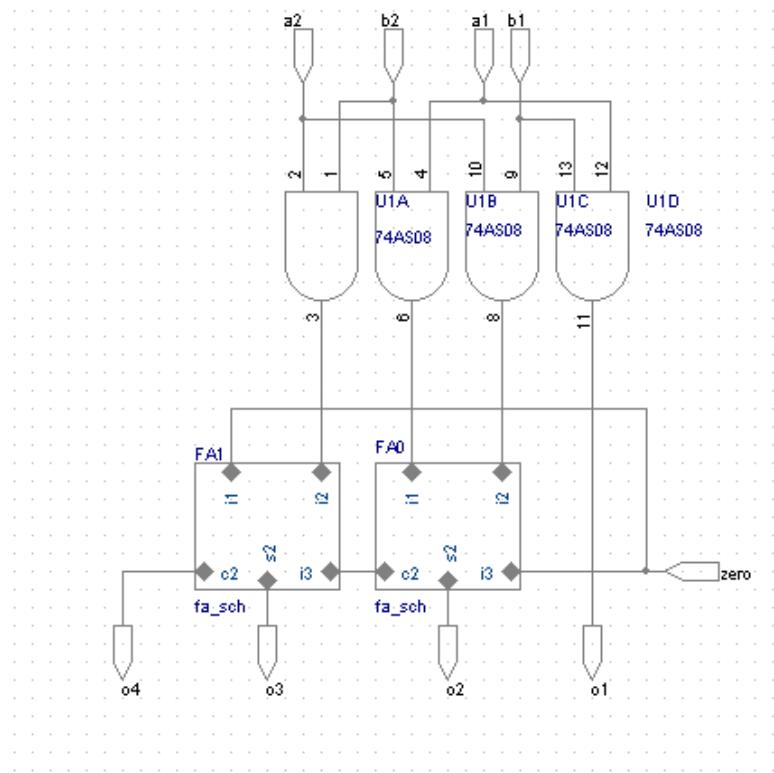
## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2

Σκοπός της εργαστηριακής άσκησης είναι η εξοικείωσή σας με τις φιλοσοφίες σχεδιασμού top-down και bottom-up. Για τη μεν top-down φιλοσοφία θα σχεδιάσετε έναν carry-save πολλαπλασιαστή 2 δυαδικών ψηφίων, για τη δε bottom-up έναν καταχωρητή 2 δυαδικών ψηφίων.

### A. Top-down design

Για τον σχεδιασμό σας θα χρησιμοποιήσετε την οικογένεια πυλών 74AS TTL και επομένως θα χρειαστείτε την ttl.old βιβλιοθήκη συμβόλων και την as.vhd βιβλιοθήκη εξομοίωσης.

Οπως ίσως γνωρίζετε από τις παραδόσεις Λογικού Σχεδιασμού, ένας carry-save πολλαπλασιαστής των δύο δυαδικών ψηφίων μπορεί να σχεδιαστεί με 2 πλήρεις αθροιστές και τέσσερις πύλες AND. Θεωρώντας προς το παρόν ότι ο πλήρης αθροιστής είναι ένα μαύρο κουτί με τρεις εισόδους ( $i_1, i_2, i_3$ ) και δύο εξόδους ( $c, s$ ) για το κρατούμενο και το αθροισμα αντίστοιχα, δημιουργήστε την πρώτη σελίδα του σχηματικού σύμφωνα με το παρακάτω σχήμα. Τα blocks FA0 και FA1 αποτελούν hierarchical blocks.

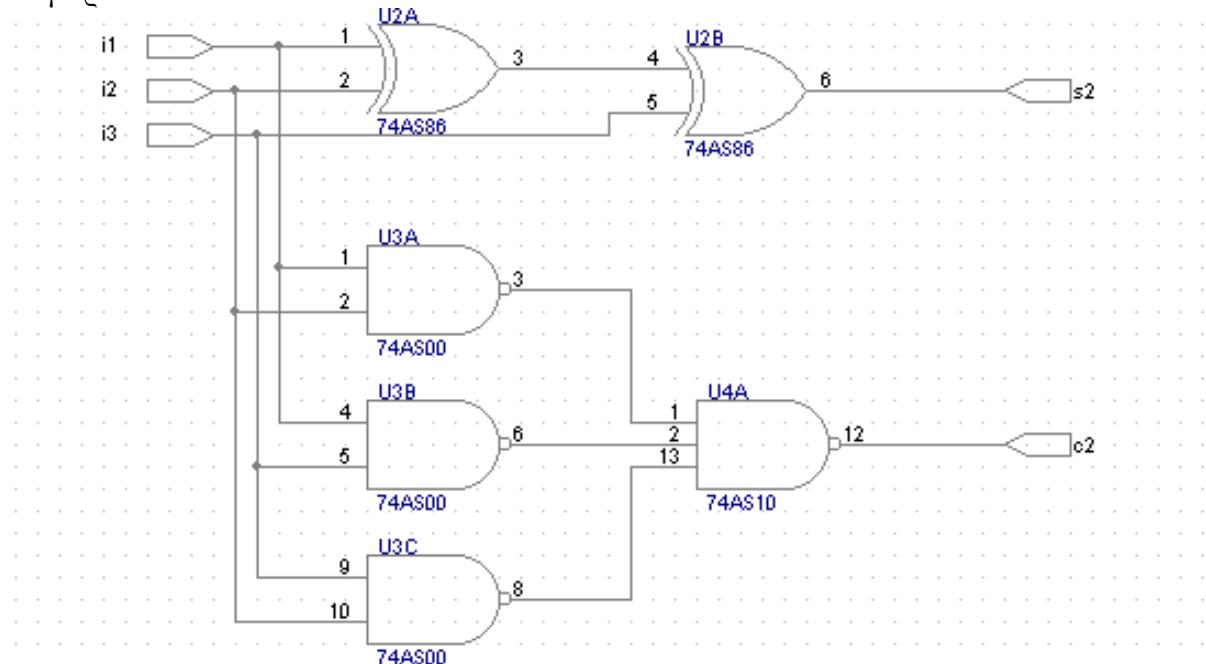


Τα hierarchical blocks αποτελούνται από:

- ένα μοναδικό όνομα αναφοράς μέσα στην σελίδα του σχηματικού (FA0 και FA1) για κάθε αντίγραφο,
- το όνομα του block (fa\_sch) και

- ♦ τα σήματα εισόδου και εξόδου.

Τοποθετούνται στην σελίδα επιλέγοντας Place -> Hierarchical Block. Στο πεδίο reference το όνομα αναφοράς του block το οποίο θα πρέπει να είναι μοναδικό στην σελίδα μας (π.χ. FA0, FA1, FA2, ...), στο πεδίο Implementation Type επιλέγουμε Schematic View (υπονοώντας ότι θα εισάγουμε τον πλήρη αθροιστή με γραφικό τρόπο) και στο Implementation Name δίνουμε το όνομα του block (π.χ fa\_sch). Στην συνέχεια ορίζουμε με το ποντίκι τις διαστάσεις του block. Τα σήματα εισόδου και εξόδου του hierarchical block αποτελούν hierarchical pins. Εισάγονται επιλέγοντας με το ποντίκι το hierarchical block και στην συνέχεια με την εντολή Place -> Hierarchical Pins και αφού δώσουμε το όνομα του σήματος, το είδος και το πλάτος του. Η λειτουργία τερματίζεται με δεξιό πλήκτρο του ποντικιού και End Mode.



Αφού τελειώσετε την εισαγωγή του πρώτου επιπέδου ιεραρχίας του σχεδιασμού σας, θα πρέπει να επεξηγήσετε την λειτουργία του hierarchical block fa\_sch. Προσέξτε ότι εφόσον τα δύο blocks έχουν την ίδια λειτουργία, μόνο μία επεξήγηση χρειάζεται. Το σχηματικό που θα πρέπει να φτιάξετε για να επεξηγήσετε τον πλήρη αθροιστή φαίνεται στο παραπάνω σχήμα. Επιλέξτε λοιπόν ένα από τα δύο hierarchical blocks που έχετε ζωγραφίσει και με το δεξιό πλήκτρο του ποντικιού επιλέξτε Descend Hierarchy. Το εργαλείο θα δημιουργήσει για εσάς μια νέη σελίδα μαζί με τους ακροδέκτες (I/Os) του hierarchical block. Εκεί εισάγετε το σχηματικό του πλήρους αθροιστή. Αφού ολοκληρώσετε την εισαγωγή του κυκλώματος, παρατηρείστε την δομή των Design Resources στον Hierarchical Browser.

---

Ακολουθήστε την διαδικασία που περιγράφεται στην προηγούμενη εργαστηριακή άσκηση για να εκτελέσετε την χρονική εξομοίωση ώστε να επαληθεύσετε την σωστή λειτουργία του σχηματικού σας, που είναι και το ζητούμενο του πρώτου μέρους.

## B. Bottom-up design

Για τον σχεδιασμό σας θα χρησιμοποιήσετε την οικογένεια πυλών 74AS TTL και επομένως θα χρειαστείτε την ttl.old βιβλιοθήκη συμβόλων και την as.vhd βιβλιοθήκη εξομοίωσης.

Ενας καταχωρητής 2 δυαδικών ψηφίων αποτελείται από 2 D flip-flops. Παρότι το D flip-flop υπάρχει στις δεδομένες βιβλιοθήκες, εμείς θα δημιουργήσουμε ένα νέο σε μία δική μας βιβλιοθήκη, Στο νέο αυτό flip-flop θα δώσουμε όνομα d\_ff.

Προκειμένου να δημιουργήσουμε μία νέα βιβλιοθήκη επιλέγουμε : File -> New -> Library. Με το δεξί πλήκτρο του ποντικιού πάνω στο εικονίδιο της νέας βιβλιοθήκης μέσα στον hierarchical browser επιλέγουμε New Part. Δίνουμε το όνομα του νέου part (π.χ. d\_ff) και στην επιλογή Attach implementation επιλέγουμε στο Type το Schematic View (το flip-flop θα επεξηγηθεί σχηματικά) ενώ στο Name δίνουμε το όνομα του σχηματικού που θα περιγράφει το part (π.χ. d\_ff\_schematic).

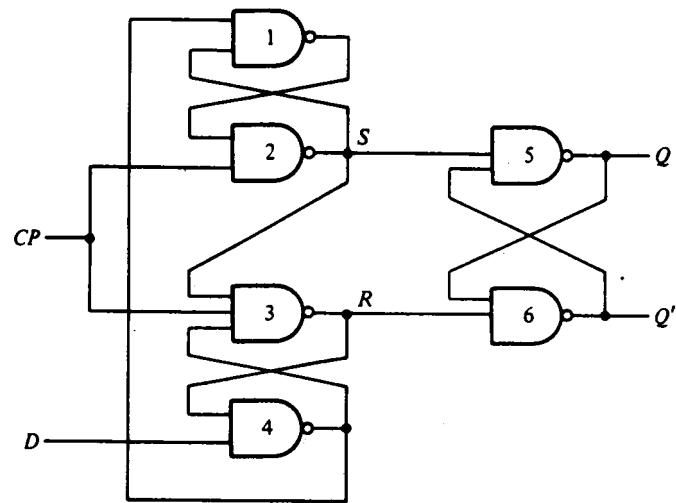
Στην σελίδα που εμφανίζεται δημιουργούμε το σχήμα του συμβόλου του νέου part (Place -> Rectangle, Place -> Ellipse, κ.ο.κ) και εισάγουμε πληροφορία για τα σήματα εισόδου και εξόδου του (Place -> Pin, Place -> Pin Array). Τελειώνοντας, αποθηκεύουμε τα σύμβολα της βιβλιοθήκης μας με κάποιο όνομα και τερματίζουμε το library project.

Προκειμένου να δημιουργήσουμε τον καταχωρητή των 2 δυαδικών ψηφίων δημιουργούμε ένα νέο project. Με το δεξί πλήκτρο του ποντικιού πάνω στο εικονίδιο Library μέσα στον hierarchical browser επιλέγουμε Add File και εισάγουμε την βιβλιοθήκη που δημιουργήσαμε νωρίτερα. Στην συνέχεια εισάγουμε το σχηματικό μας χρησιμοποιώντας αντίγραφα του νέου part που έχουμε δημιουργήσει. Με το δεξί κουμπί του ποντικιού πάνω στο σύμβολο του νέου part (d\_ff) και την επιλογή Descend Hierarchy μπορούμε να εισάγουμε την σχηματική περιγραφή του. Προσέξτε ότι το εργαλείο θεωρεί ότι τα σύμβολα συνήθως δεν ιράρουν ιεραρχία πίσω τους, αλλά προέρχονται από τις βιβλιοθήκες κάποιου κατασκευαστή. Για να μπορέσετε συνεπώς να εκτελέσετε την Descend Hierarchy θα πρέπει πρώτα να αλλάξετε τις ιδιότητες των συμβόλων σε non-primitives. Παρατηρήστε ότι τα σήματα εισόδου και εξόδου είναι ήδη γνωστά από την περιγραφή του συμβόλου στην βιβλιοθήκη. Διάφοροι σχεδιασμοί μπορούν να χρησιμοποιούν αντίγραφα του νέου flip-flop, αρκεί να περιλαμβάνουν την βιβλιοθήκη, την οποία δημιουργήσαμε στα αρχικά στάδια του δεύτερου μέρους.

Εισάγετε το παρακάτω σχηματικό για το d\_ff χρησιμοποιώντας πύλες της οικογένειας 74AS TTL. Τέλος, επαληθεύστε την ορθότητα του σχεδιασμού σας με την

---

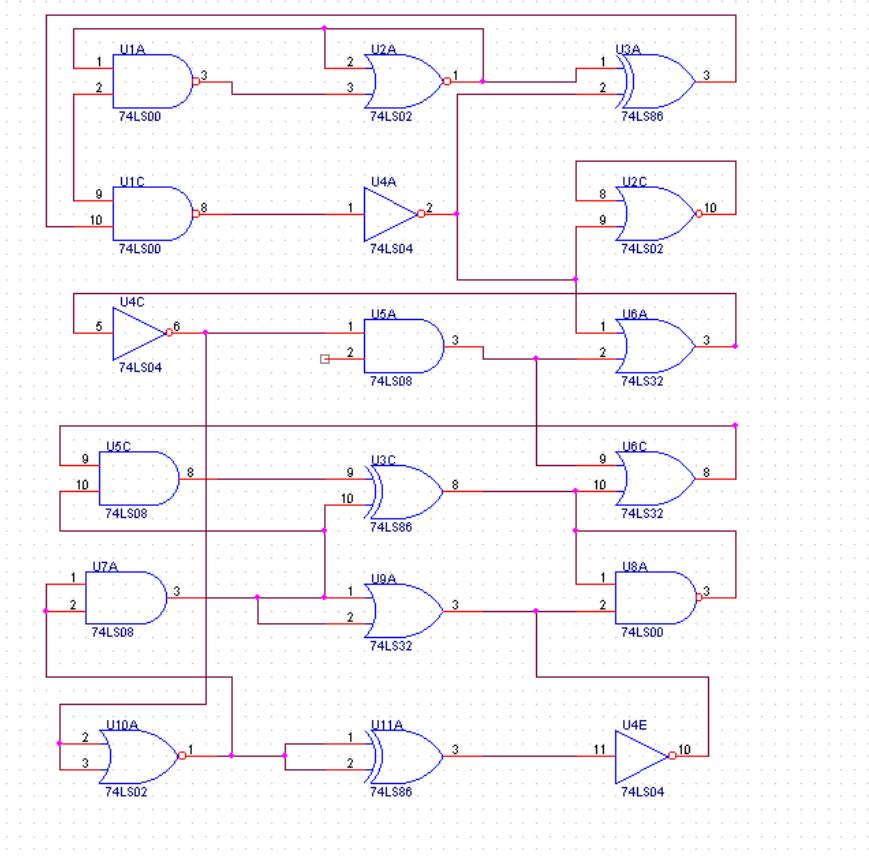
βοήθεια της χρονικής εξομοίωσης . Παραδοτέα για το 2<sup>o</sup> μέρος είναι το stimulus που χρησιμοποιήσατε καθώς και τα αποτελέσματα της εξομοίωσης.



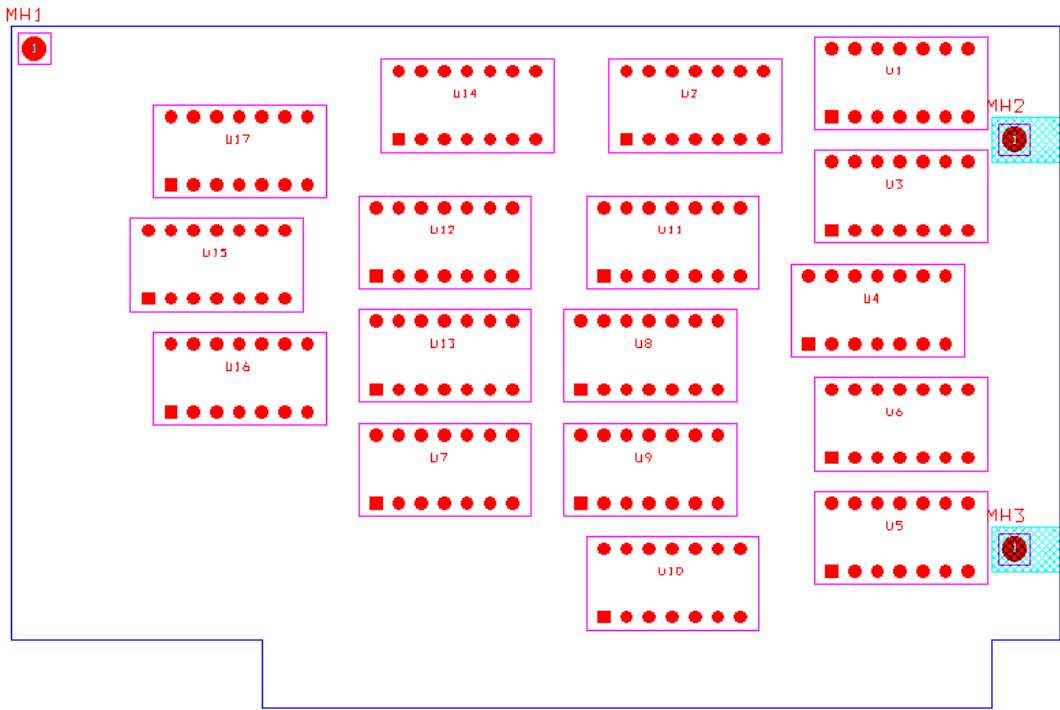
### ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 3

Σκοπός της άσκησης είναι η εξοικείωση με τις διαδικασίες back-end σε τεχνολογία πλακέτας. Το τελικό αποτέλεσμα της εργαστηριακής άσκησης θα είναι ένας πλήρως τοποθετημένος στο χώρο και διασυνδεδεμένος σχεδιασμός.

Το πρώτο βήμα είναι η εισαγωγή του σχεδιασμού μας με γραφικό τρόπο και η δημιουργία του netlist. Έστω ότι θέλουμε να κατασκευάσουμε το PCB για κύκλωμα αποτελούμενο από 4 σχεδιαστικά κύτταρα κάθε ένα της μορφής της παραπάνω εικόνας. Αφού σχεδιάστε το κύκλωμα στο Orcad Express, επιλέγετε Tools -> Update Part References και στην συνέχεια Tools -> Create Netlist. Επιλέγουμε το Layout και ενεργοποιούμε το User Properties are in inches. Το αποτέλεσμα αυτού είναι η δημιουργία του netlist σε ένα αρχείο με κατάληξη .MNL.



Στην συνέχεια τερματίζουμε το Orcad Express και εκτελούμε το Orcad Layout. Δίνοντας File -> New επιλέγουμε το design template που θέλουμε να ακολουθήσουμε (τα διάφορα design template που υπάρχουν ορίζουν παραμέτρους της πλακέτας όπως διαστάσεις, επίπεδα γραμμών κ.α.). Επιλέξτε το tutor.tch και στην συνέχεια ορίστε το αρχείο .MNL που δημιουργήσατε προηγουμένως. Τέλος δώστε το όνομα του αρχείου όπου θα αποθηκευτεί ο υλοποιημένος σχεδιασμός.



Στην οθόνη βλέπετε πλέον τις διαστάσεις της πλακέτας και τα ολοκληρωμένα κυκλώματα που πρέπει να εισαχθούν σε αυτή με τη μορφή footprints (αποτυπώματα). Το template tutor που χρησιμοποιούμε επιτρέπει μόνο τη χρήση DIP through hole ολοκληρωμένων.

Το επόμενο βήμα είναι η διάταξη στον χώρο των ολοκληρωμένων του σχεδιασμού. Η διαδικασία αυτή γίνεται αυτόματα επιλέγοντας Auto -> Batch Place. Εάν κάποια από τα ολοκληρωμένα δεν έχουν τοποθετηθεί στην επιθυμητή θέση την κρίση σας θέση μπορείτε να τα μετακινήσετε σύροντας τα με το ποντίκι. Παρατηρήστε ότι εκτός από τα ολοκληρωμένα κυκλώματα φαίνονται και οι γραμμές που ενώνουν τους διάφορους ακροδέκτες μεταξύ τους με τη μορφή ασύνδετων γραμμών (hairlines). Το επόμενο και τελευταίο βήμα είναι η διασύνδεση των ακροδεκτών των ολοκληρωμένων κυκλωμάτων. Αυτό μπορεί να γίνει αυτόματα με την επιλογή Auto -> Batch Route.

Η πλακέτα που κατασκευάσατε είναι δύο επιπέδων. Διασύνδεση των στοιχείων επιτρέπεται και στα δύο αυτά επίπεδα. Ανατρέχοντας στα manuals του εργαλείου (κάτω από το μενού help), απενεργοποιείστε το δεύτερο επίπεδο, έτσι ώστε η διαδρόμιση να γίνεται μόνο στο ένα επίπεδο. Επαναλάβετε την διαδικασία διαδρόμισης και εφόσον το εργαλείο δεν τα καταφέρει να σας παράγει έναν πλήρως διασυνδεδεμένο σχεδιασμό, πειραματιστείτε με διαφορετική διάταξη των ολοκληρωμένων, ή αλλάξτε το μέγεθος της πλακέτας. Παραδώστε τα σχηματικά των πλακετών ενός και δύο επιπέδων

## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 4

(Ιδέα & Υλοποίηση : Κ. Αδαός – Υπ. Διδάκτωρ ΤΜΗΥΠ)

Στην άσκηση αυτή θα φτιάξουμε ένα μικρό "κλειδωτήριο" (γνωστό ως hasp) για την προστασία του λογισμικού μας από πιθανούς αντιγραφείς. Ενα hasp διανέμεται μαζί με κάθε αντίγραφο του λογισμικού και τοποθετείται σε μια από τις θύρες του υπολογιστή μας. Το λογισμικό σε τακτά χρονικά διαστήματα, στέλνοντας και διαβάζοντας πληροφορίες μέσω της θύρας του υπολογιστή μας, ελέγχει την παρουσία ή την απουσία του hasp. Ετσι ο επίδοξος αντιγραφέας μπορεί μεν να αντιγράψει το λογισμικό, όμως αυτό δε θα μπορεί να λειτουργήσει σε άλλον υπολογιστή, παρά μόνο αν αντιγραφεί και το hasp.

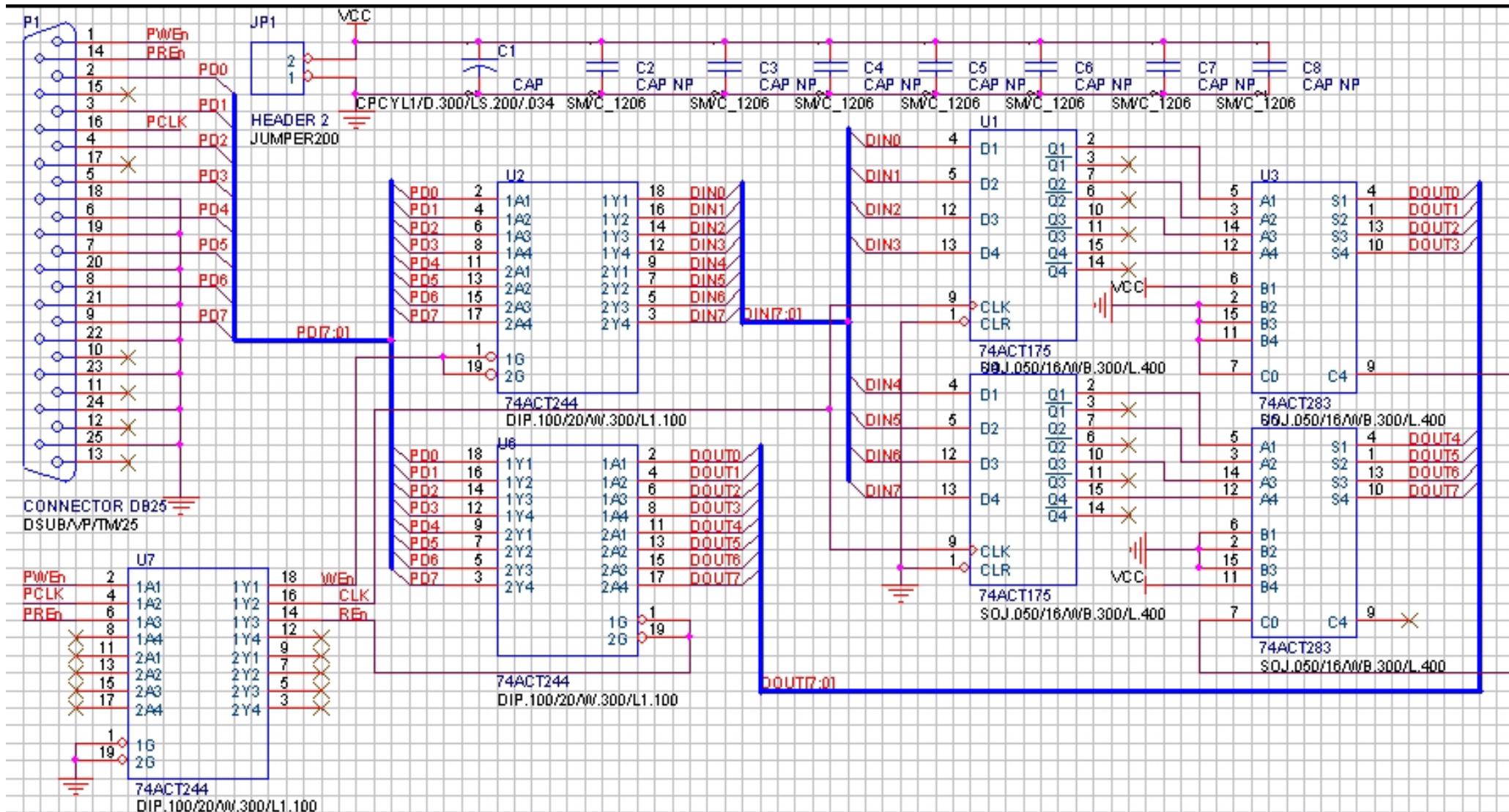
Για το παρόντα μας, υποθέτουμε :

- Οτι το hasp θα τοποθετηθεί σε μια παράλληλη θύρα του υπολογιστή μας και καμμία άλλη συσκευή δε θα χρησιμοποιεί τη συγκεκριμένη θύρα.
- Οτι το λογισμικό μας δεν ελέγχει απλά την παρουσία του hasp, αλλά του αναθέτει και κάποια μορφή επεξεργασίας της πληροφορίας. Υποθέτουμε ότι αυτή η επεξεργασία είναι η απεικόνιση κάθε byte σε κώδικα XS (Excess) 127.
- Το πρωτόκολλο επικοινωνίας με το hasp είναι το δυνατόν απλούστερο. Το λογισμικό αφού τοποθετήσει το byte στην παράλληλη θύρα, ενεργοποιεί ένα σήμα για να υποδειξει τη διαδικασία εγγραφής και παράγει ένα παλμό ρολογιού για το χρονισμό της εγγραφής. Το λογισμικό υποδεικνύει ανάγνωση των κωδικοποιημένων δεδομένων ενεργοποιώντας ένα σήμα ανάγνωσης.

Σκοπός της άσκησης είναι η εξοικείωσή σας με το χειρισμό αποκλειστικής πρόσβασης πάνω σε μια διαμοιραζόμενη αρτηρία, και της φάσης back-end..

### A. Εισαγωγή του σχεδιασμού

Η προτεινόμενη λύση για το παραπάνω πρόβλημα φαίνεται στο σχηματικό της επόμενης σελίδας. Τα σήματα PWEn, PCLK και PREn, είναι τα σήματα ελέγχου που μας παράγει το λογισμικό μας, ενώ η αρτηρία PD είναι τα παράλληλα δεδομένα που ανταλλάσσονται μεταξύ του hasp και του λογισμικού μας. Προσέξτε ότι η αρτηρία PD είναι δύο κατευθύνσεων (bidirectional) και συνεπώς θα πρέπει να υπάρχει αυστηρός έλεγχος σχετικά με το πότε θα πρέπει το hasp να οδηγήσει αυτή την αρτηρία. Το hasp οδηγεί αυτή την αρτηρία MONO όταν λάβει σήμα PREn (το ο δείχνει σήμα αρνητικής λογικής), ενώ σε κάθε άλλη περίπτωση διαβάζει από αυτή την αρτηρία.



---

Αυτό το σχήμα διαιτησίας επιτυγχάνεται μέσω των ολοκληρωμένων ACT244 που είναι απομονωτές και οδηγοί γραμμών τριών καταστάσεων. Τα ολοκληρωμένα αυτά αντιγράφουν τα δεδομένα εισόδου τους (γραμμές A) στις αντίστοιχες εξόδους (γραμμές Y) μόνο όταν τα αντίστοιχα σήματα ενεργοποιήσης G (αρνητικής λογικής) είναι στο λογικό 0. Αναφερόμενοι στο σχηματικό βλέπουμε ότι οι γραμμές ελέγχου επιτρέπεται διαρκώς να περνάνε στο hasp, ενώ τα δεδομένα εισόδου περνάνε στο hasp μόνο όταν ενεργοποιηθεί το σήμα PWEn. Το hasp οδηγεί με τα κωδικοποιημένα δεδομένα του την αρτηρία μόνο όταν το PEn είναι ενεργό (προσέξτε τον προσανατολισμό του U6).

Οταν νέα δεδομένα φτάσουν στο hasp, αυτά κλειδώνονται στα 175 (τετραπλά D flip flops) βάσει της ανοδικής ακμής του ρολογιού. Η ανοδική ακμή αυτή προκαλείται από το λογισμικό. Οι έξοδες των 175 μετατρέπονται σε XS127 κώδικα μέσω των δύο προσθετών (74ACT283) τεσσάρων δυαδικών ψηφίων που συνδέονται με ripple-carry σχήμα.

Στο όλο σχηματικό έχουν προστεθεί 7 decoupling πυκνωτές (C2 έως C8), ένας πυκνωτής τανταλίου (C1) για τη σταθεροποίηση της τάσης και μια υποδοχή (header 2 pins) για τροφοδοσία και γή. Βεβαιωθείτε ότι όλα τα σύμβολα τροφοδοσίας και γής στο σχηματικό σας έχουν μετονομαστεί σε VCC και GND. Αποθηκεύστε το σχεδιασμό σας, ανανεώστε τις αναφορές των ολοκληρωμένων που χρησιμοποιήσατε, περάστε Design Rule Check ... και πριν προχωρήσετε παρακάτω, αντιγράψτε το σχεδιασμό σας σε ένα νέο υποκατάλογο.

## B. Εξομοίωση του σχεδιασμού

Πριν την υλοποίηση του σχεδιασμού, θα πρέπει να εξομοιώσετε αναλυτικά τη λειτουργία του. Απομακρύνετε τον connector P1, τους πυκνωτές και τον header, από το ένα αντίγραφο. Αντικαταστήστε τα σήματα του P1 με ιεραρχικά ports. Εξομοιώστε το σχεδιασμό σας γράφοντας τις κατάλληλες κυματομορφές εισόδου, οδηγώντας όμως την αρτηρία PD MONO όταν ενεργοποιείτε το σήμα εγγραφής. Σε όλες τις υπόλοιπες περιπτώσεις οδηγήστε την στην κατάσταση υψηλής εμπέδησης (Z). Σημειώστε τους χρόνους που χρειάζεται το σχηματικό σας για την κωδικοποίηση των δεδομένων. Αυτοί οι χρόνοι είναι σημαντικοί, μιας και θα πρέπει να γίνονται σεβαστοί από το λογισμικό ΑΝΕΞΑΡΤΗΤΑ από το πόσο γρήγορα ή αργά τρέχει το λογισμικό μας. Παραδοτέο αυτής της υποενότητας είναι το αρχείο των κυματομορφών εισόδου-εξόδου στο οποίο έχετε επιτύχει τη μέγιστη συχνότητα λειτουργίας του σχεδιασμού. Ποιά είναι αυτή η συχνότητα? Σε περίπτωση που κάπου διαπιστώσατε λάθος στο σχεδιασμό σας, μη ξεχάστε να ανανεώσετε και τα δύο αντίγραφα.

## Γ. Φυσική υλοποίηση του σχεδιασμού

Επιλέξτε το αρχικό αντίγραφο του σχεδιασμού σας, δηλαδή εκείνο που έχει τους connectors και όχι τα iεραρχικά ports. Για τη φυσική υλοποίηση του σχεδιασμού, χρειάζεται να δημιουργήσουμε ένα netlist του σχεδιασμού μας και κατόπιν να καλέσουμε το εργαλείο Layout. Ωστόσο, ακόμη δεν έχουμε ορίσει τα footprints των components που έχουμε χρησιμοποιήσει. Αυτό μπορεί να γίνει είτε μέσω του Layout, όταν φορτώνουμε το netlist είτε στο ίδιο το σχηματικό. Εμείς ακολουθούμε το δεύτερο τρόπο αν και μπορείτε να πειραματείστε με τον άλλο.

Καλέστε το εργαλείο Layout. Δώστε File->New και cancel για να μην επιλέξετε καμμία έτοιμη τεχνολογία. Ενώ είναι επιλεγμένο το κενό σχεδιαστικό φύλλο, επιλέξτε Tool ->Library Manager. Στο καινούργιο φύλλο που ανοίγει, βλέπετε αριστερά τις βιβλιοθήκες footprints που υπάρχουν. Επιλέγοντας κάποια βιβλιοθήκη, κάτω αριστερά εμφανίζονται όλα τα footprints που υπάρχουν στη συγκεκριμένη βιβλιοθήκη. Χτυπήστε σε κάποιο από αυτά για να το δείτε στο δεξί παράθυρο.

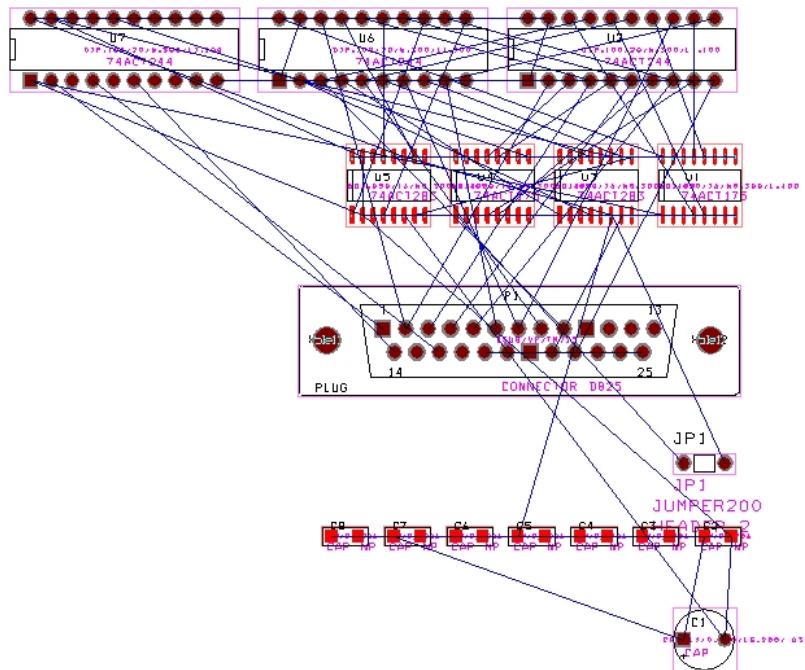
Ας γυρίσουμε στο σχεδιασμό μας, για να δούμε τι packages θα επιλέξουμε. Ξεκινώντας από τους πυκνωτές, βλέπουμε ότι θέλουμε through hole πυκνωτή για το C1, ενώ για τους C2 έως C8 μπορούμε να επιλέξουμε πυκνωτές surface mount για μείωση του εμβαδού της πλακέτας. Επιλέξτε συνεπώς στο Layout τη βιβλιοθήκη TM\_CAP\_P. Εκεί υπάρχουν δεκάδες footprints για διαφορετικούς πυκνωτές. Κάθε footprint ορίζει το σχήμα του πυκνωτή, την απόσταση μεταξύ των ακροδεκτών του, τις διαμέτρους των τρυπών ι.ο.κ. Επιλέγουμε ένα footprint που αντιστοιχεί σε κύλινδρικό πυκνωτή με μεγάλη απόσταση και μεγάλη διάμετρο τρυπών, π.χ. το CPCYL1/D.300/LS.200/.034. Προσέξτε ότι ο πυκνωτής έχει πολικότητα που υποδεικνύεται από τα +/- στο αποτύπωμα. Κάντε Control-C το όνομα του αποτυπώματος, γυρίστε στο σχεδιασμό και μέσω του Edit Properties, User Properties, PCB Footprint, New, Control-V, επισυνάψτε στο συγκεκριμένο component το συγκεκριμένο αποτύπωμα. Από τη βιβλιοθήκη SM (surface mount) επιλέξτε το αποτύπωμα για τους υπόλοιπους πυκνωτές (άνευ πολικότητας). Προτεινόμενο αποτύπωμα είναι το SM/C\_1206. Ωστόσο μπορείτε να επιλέξετε οποιοδήποτε άλλο νομίζετε κατάλληλο. Αφού επισυνάψετε κι αυτό το αποτύπωμα στο σχηματικό σας, μπορείτε να αντιγράψετε τον πυκνωτή στις υπόλοιπες θέσεις ώστε να αποφύγετε την επανάληψη της διαδικασίας για κάθε πυκνωτή.

Με αντίστοιχο τρόπο επιλέγονται τα υπόλοιπα αποτυπώματα που φαίνονται στο σχηματικό. Προσέξτε ότι για τα 244 έχουμε επιλέξει through hole packages ενώ για τα υπόλοιπα surface mount packages. Η επιλογή αυτή δεν είναι τυχαία. Προσέξτε ότι αν για κάποιο λόγο δύο πηγές ταυτόχρονα προσπαθήσουν να οδηγήσουν την

διαμοιραζόμενη αρτηρία PD, τότε αυτό πιθανότατα να οδηγήσει σε "κάψιμο" των 244. Επιλέγοντας through hole packages μπορούμε να μη κολλήσουμε τα ολοκληρωμένα κατευθείαν πάνω στην πλακέτα, αλλά να κολλήσουμε βάσεις ολοκληρωμένων στην πλακέτα και να τα αλλάξουμε κάθε φορά που καίγονται. Η πιθανότητα να καούν τα υπόλοιπα ολοκληρωμένα είναι μικρή και συνεπώς γι' αυτά επιλέγουμε surface mount packages για τη μείωση του εμβαδού. Τέλος σημειώνεται η χρήση της οικογένειας ACT (Advanced CMOS – TTL compatible) σε όλο το σχεδιασμό. Η οικογένεια αυτή προσφέρει κατανάλωση τεχνολογίας CMOS με εξόδους συμβατές με TTL και καθυστέρηση αντίστοιχη της ALS TTL τεχνολογίας.

Αφού ολοκληρώσετε την επιλογή των αποτυπωμάτων για όλα τα components του σχεδιασμού σας, αποθηκεύστε τον, ανανεώστε τις αναφορές σε αυτά και όταν το Design Rule Check... σας υποδειξεί ότι δεν υπάρχουν λάθη στο σχεδιασμό σας, δημιουργήστε ένα νέο netlist. Καλέστε το Layout.

Θα επιλέξουμε σαν Template υλοποίησης το Metric.TCH. Το template αυτό έχει ήδη οριστεί να παρέχει δύο επίπεδα διαδρόμισης και δύο επίπεδα για τροφοδοσία και γή. Οι γραμμές σε κάθε επίπεδο έχουν οριστεί να έχουν πλάτος 0,254mm. Σαν netlist επιλέξτε αυτό που δημιουργήσατε πιο πάνω. Μετά την ανάγνωση όλης της πληροφορίας, θα πάρετε ένα σχηματικό που έχει τη μορφή:



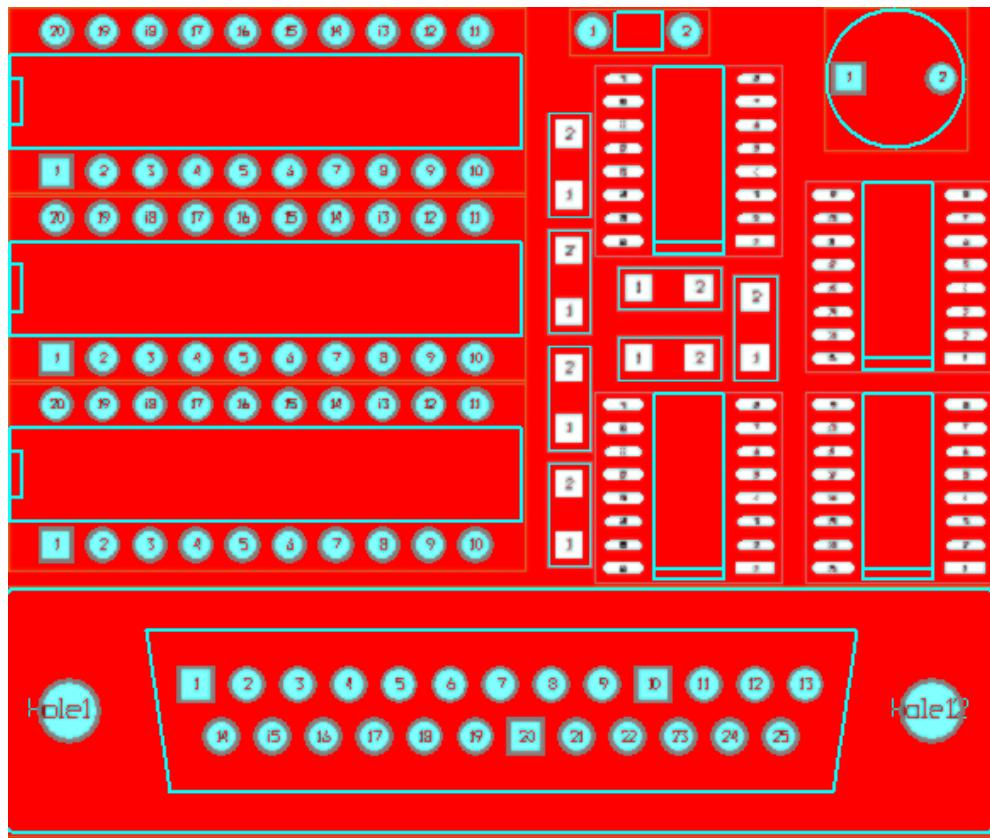
Στο παραπάνω σχηματικό φαίνονται τα αποτυπώματα που έχετε επιλέξει καθώς και οι γραμμές διασύνδεσης που πρέπει να διαδρομιστούν. Πριν από όλα όμως ας κάνουμε μια σύντομη γνωριμία με το Template METRIC.

Επιλέξτε window -> Database Spreadsheets και εκεί Layers. Παρατηρείστε ότι χρησιμοποιούνται δύο επίπεδα για διαδρόμιση και τα δύο επόμενα σαν επίπεδα τροφοδοσίας και γείωσης. Επιλέξτε τα δύο τελευταία επίπεδα και απενεργοποιείστε τα αλλάζοντας τον τύπο τους από Plane σε Unused. Στην ουσία έχετε πλέον ένα νέο Template με μόνο δύο επίπεδα.

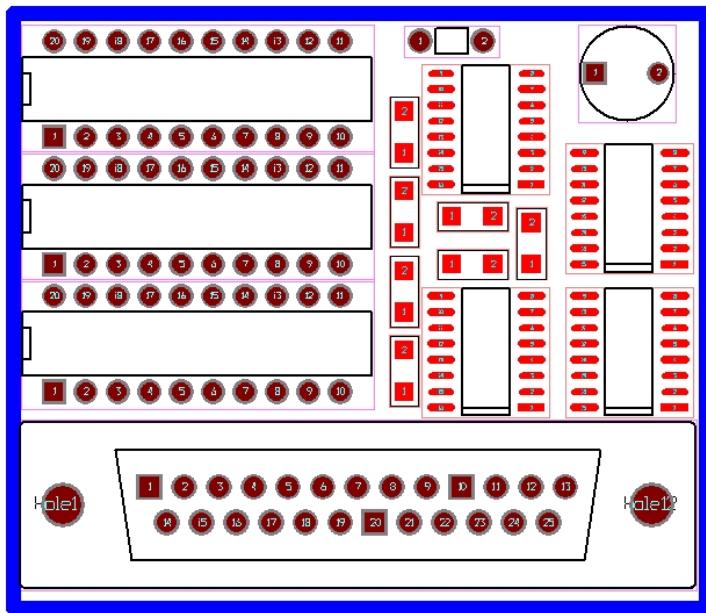
Επιλέγοντας Tool->Component, ξεκινείστε την τοποθέτηση του σχεδιασμού σας, έχοντας σα στόχους :

- α) Το μικρότερο δυνατό εμβαδόν
- β) Οι decoupling πυκνωτές να είναι το δυνατόν κοντύτερα στα ολοκληρωμένα
- γ) Ο ηλεκτρολυτικός πυκνωτής να είναι το δυνατόν κοντύτερα στο jumper τροφοδοσίας.

Μια πιθανή τοποθέτηση φαίνεται στο παρακάτω σχήμα :



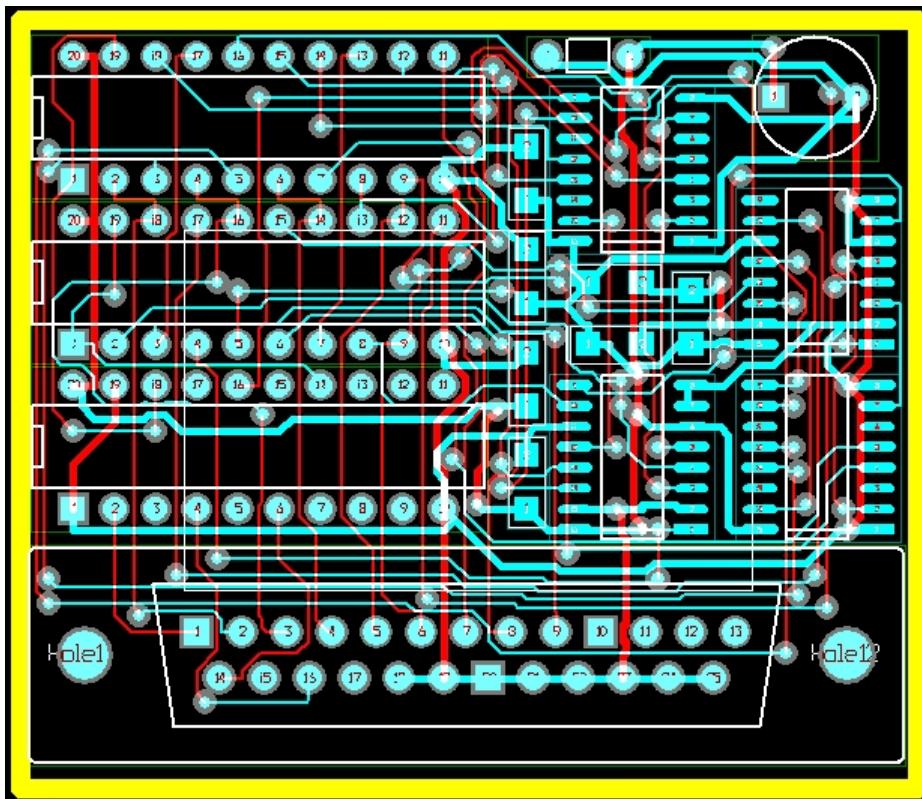
Πριν ξεκινήσουμε τη διαδρόμιση θα πρέπει να υποδείξουμε στο εργαλείο μας τα όρια της πλακέτας μας. Επιλέγουμε γι' αυτό Tool->Obstacle. Επίσης αλλάζουμε το επίπεδο που βλέπουμε στην οθόνη μας σε 0 (Global). Από το μενού που προκύπτει με δεξιά κλίκ επιλέγουμε insert ... και χαράζουμε τα όρια της πλακέτας μας, στα όρια των τοποθετημένων σχεδιασμών μας. Το σχηματικό μας θα πρέπει να είναι κάπως έτσι :



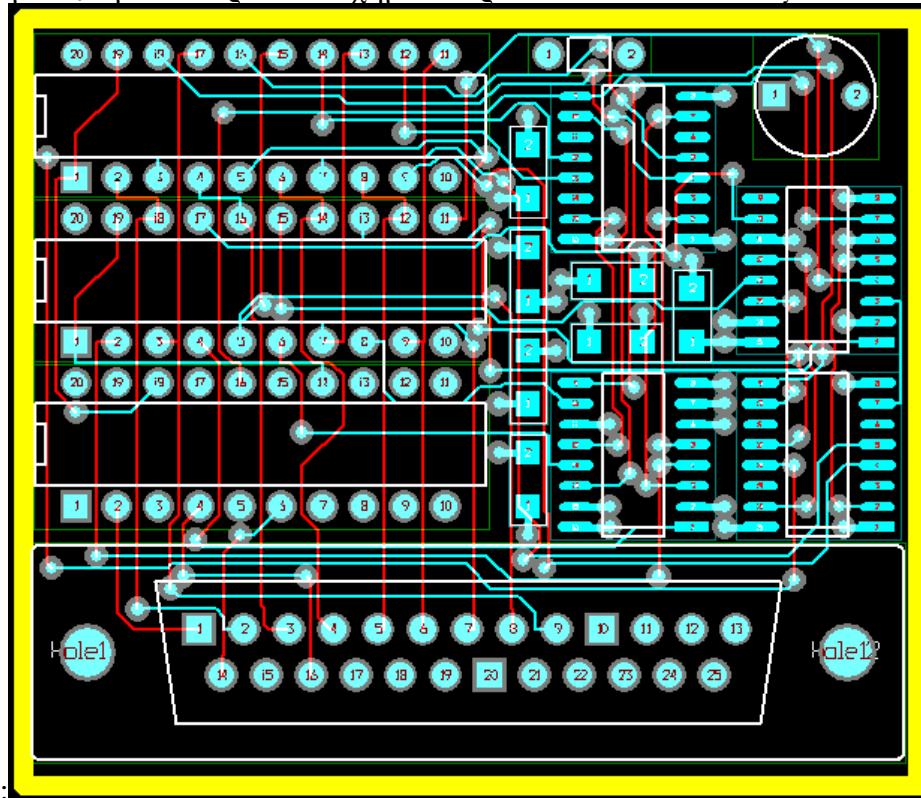
Ας ορίσουμε στη συνέχεια το πάχος των γραμμών που θέλουμε. Επιλέξτε Window->Database Spreadsheets και Nets. Από τα nets επιλέξτε τα σήματα VCC και GND και αλλάξτε το πλάτος τους σε .508 (δηλαδή το διπλό πλάτος). Επιλέξτε όλα τα σήματα χτυπώντας στη στήλη Net Name και αποεπιλέξτε το Routing Enabled. Επιλέξτε τα σήματα VCC και GND και αλλάξτε τα σε Routing Enabled, μιας και στο πρώτο πέρασμα θα ξάνουμε διαδρόμιση μόνο των δύο σημάτων. Γυρνώντας στο σχηματικό, δείτε ότι μόνο αυτά τα δύο σήματα εμφανίζονται πλέον.

Διαδρομίστε αυτά τα δύο σήματα με το πλάτος που έχετε επιλέξει. Γυρίστε στον πίνακα των σημάτων, αλλάξτε τα δεδομένα διαδρόμισης έτσι ώστε να διαδρομιστούν όλα τα υπόλοιπα σήματα πλην των VCC και GND. Δώστε Auto->Route Window για να διαδρομιστούν τα υπόλοιπα σήματα. Πλέον θα πρέπει να έχετε έναν πλήρως τοποθετημένο και διαδρομημένο σχεδιασμό σαν τον παρακάτω (αν δεν μπορεί η διαδρόμιση να ολοκληρωθεί αυτόματα, επιλέξτε διαφορετική τοποθέτηση ή μεγαλώστε το μέγεθος της πλακέτας σας).

Αφού αποθηκεύσετε τον σχεδιασμό σας, επιλέξτε όλα τα nets από το spreadsheet και μετά clear tracks. Ετσι θα σβηστεί όλη η προηγούμενη διαδρόμιση. Σκοπός μας είναι πλέον να ξάνουμε διαδρόμιση σε 4 επίπεδα, όπου τα δύο μεσαία επίπεδα θα είναι μόνο για τροφοδοσία και γή. Συνεπώς θα πρέπει να επιλέξουμε στα layers τα δύο επίπεδα σαν Planes. Στα nets επιλέγουμε μόνο το GND για routing και μέσω της επιλογής Net Layers αναθέτουμε αυτό το σήμα μόνο στο GND Plane Layer. Με αντίστοιχο τρόπο αναθέτουμε τη διαδρόμιση του VCC μόνο στο Power Plane Layer. Η διαδρόμιση σε αυτά τα επίπεδα γίνεται μέσω της εντολής Auto -> Fanout Board. Κάντε ορατά μόνο τα επίπεδα Power και GND (επίπεδα 3 και 4).



Εξηγείστε πως έχει γίνει η διαδρόμιση σε αυτά τα επίπεδα και γιατί είναι διαφορετική η τακτική για τα through hole και τα surface mount devices. Η διαδρόμιση στα υπόλοιπα επίπεδα γίνεται όπως προηγουμένων. Ο τελικός σχεδιασμός σας θα μοιάζει με το παρακάτω σχήμα. Παραδοτέα είναι τα δύο layouts.



## **ΕΡΓΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΣΕ FPGAs**

Για τις ασκήσεις 5, 6, 7 και 8 κατά την τρέχουσα ακαδημαϊκή χρονιά, θα χρησιμοποιήσουμε τα εργαλεία :

- (α) ModelSim 5.7f της Mentor Graphics και
- (β) WebpackISE 13.1 της εταιρείας Xilinx,  
καθώς και τις πλακέτες
  - (α) XSA-3S και
  - (β) XST-4 της εταιρείας Xess.com

Σκοπός όλων των ασκήσεων είναι το rapid system prototyping μέσω ανάπτυξης κώδικα HDL, σύνθεσης και προγραμματισμού ενός FPGA. Τόσο τα εργαλεία όσο και οι αναπτυξιακές πλακέτες έχουν τόσο πολλά χαρακτηριστικά που είναι αδύνατον να περιγραφούν πλήρως στα πλαίσια ενός μαθήματος. Παρακάτω συνεπώς χρησιμοποιούμε κάποια από εκείνα τα χαρακτηριστικά που είναι χρήσιμα για τις συγκεκριμένες ασκήσεις, αλλά θα πρέπει να είστε έτοιμοι να ανατρέξετε στα αντίστοιχα manuals ανά πάσα χρονική στιγμή.

Τα εργαλεία ModelSim και WebpackISE είναι διαθέσιμα για ακαδημαϊκούς λόγους από τα site [www.model.com](http://www.model.com) και [www.xilinx.com](http://www.xilinx.com) οπότε μπορείτε μεγάλο μέρος των ασκήσεων να τις προετοιμάσετε σπίτι σας και απλά να τις επιβεβαιώνετε στις αναπτυξιακές πλακέτες. Ωστόσο λόγω του μεγέθους αυτών των εργαλείων, μπορείτε να τα προμηθευτείτε μαζί με τις εκπαιδευτικές άδειές τους από το γραφείο μου, προσκομμίζοντας ένα DVD.

Πριν από οποιαδήποτε εργασία σας στις πλακέτες, πρέπει οπωσδήποτε να διαβάσετε και να καταλάβετε τα αντίστοιχα manuals, που επισυνάπτονται. Το κόστος αυτών των πλακετών είναι ιδιαίτερα υψηλό και θα πρέπει να διαφυλάξετε την πλήρη λειτουργικότητά τους σας κόρη οφθαλμού. Επίσης, θα πρέπει κατά την επαφή σας με τις πλακέτες να είστε ιδιαίτερα προσεκτικοί, ώστε να μη βραχυκυκλώνετε με το άγγιγμά σας σημεία τα οποία δε θα πρέπει.

Στο εργαστήριο υπάρχει πάντα ένα ελεγμένο έτοιμο ζεύγος πλακετών προς χρήση, μαζί με το απαραίτητο τροφοδοτικό και το καλώδιο προγραμματισμού. Σε καμία περίπτωση δε θα πρέπει να προσπαθήσετε να διαχωρίσετε τις δύο αυτές πλακέτες, να αλλάξετε σχήμα τροφοδοσίας ή να αλλάξετε τρόπο προγραμματισμού του FPGA. Για όποιες ομάδες το επιθυμούν, μπορούν να παραχωρηθούν πλακέτες + τροφοδοτικά + καλώδια προγραμματισμού για το σπίτι με υποχρέωση καταβολής του κόστους (περίπου 650 Euro) σε περίπτωση επιστροφής κατεστραμένων υλικών. (Η κατάσταση μπορεί να ελεγχθεί άμεσα με ηλεκτρονικό τρόπο).



## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 5

Σκοπός της εργαστηριακής ασκησης είναι μια πρώτη γνωριμία σας με την γλώσσα περιγραφής υλικού Verilog και την εξομοίωση. Στην άσκηση αυτή θα κληθείτε να περιγράψετε σε Verilog τόσο έναν μετρητή 8 δυαδικών ψηφίων όσο και τα διανύσματα εξομοίωσής του. (Στο εξής όλος ο κώδικας που παρατίθεται τις ασκήσεις, υπάρχει και στα αντίστοιχα subfolders του παρόντος).

### A. Περιγραφή του μετρητή

(αρχείο 5/counter.v)

```
module counter (clear, clock, load, start_stop, count, data);
    input [6:0] data;
    output [6:0] count;
    input start_stop;
    input load;
    input clock;
    input clear;
    reg [6:0] count;

    always @(posedge clock or posedge clear)
        if (clear) count <= 0;
        else if (load) count <= data;
        else if (start_stop) count <= count + 1;
endmodule
```

Ο κώδικας αυτός περιγράφει έναν μετρητή προς τα άνω των 7 δυαδικών ψηφίων με παράλληλη φόρτωση, είσοδο επίτρεψης και ασύγχρονη είσοδο καθαρισμού.

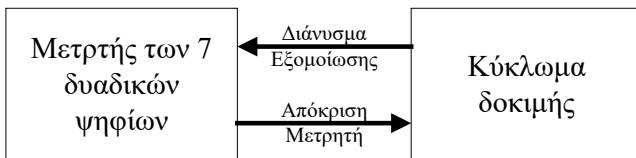
### B. Περιγραφή των διανυσμάτων εξομοίωσης

Για την εξομοίωση του κυκλώματος απαιτείται να περιγράψουμε και τα διανύσματα εξομοίωσης του μετρητή. Σε ένα καινούργιο αρχείο (5/testcounter.v) εισάγουμε τον ακόλουθο κώδικα :

```
module testcounter ();
    reg [7:0] d;
    wire [7:0] c;
    reg s_s, l, clk, clr;

    counter inst0 (clr, clk, l, s_s, c, d);
    initial
        begin
            # 5 clk <= 0; clr <= 0; l <= 0; s_s <= 0; d <= 8'hF0;
        end
    always # 40 clk <= !clk;
    initial
        begin
            # 100 clr <= 1;
            # 50 clr <= 0;
        end
    initial
        begin
            # 400 s_s <= 1;
            # 3000 s_s <= 0;
        end
    initial
        begin
            # 3500 s_s <= 1;
            # 3700 l <= 1;
            # 200 l <= 0;
        end
endmodule
```

Ο κώδικας αυτός περιγράφει έναν tester για τον μετρητή σύμφωνα με το παρακάτω σχήμα :



Για τον σκοπό αυτό χρησιμοποιεί ένα αντίγραφο του μετρητή (inst0), κάποιους καταχωρητές (clr, clk, l, s\_s,, d) για να εφαρμόσει το διάνυσμα δοκιμής και κάποια σήματα ώστε να διαβάσει την απόκριση του μετρητή (c). Ο υπόλοιπος κώδικας παράγει τα διανύσματα δοκιμής που ελέγχουν την λογική λειτουργία του μετρητή. Είμαστε πλέον έτοιμοι να προχωρήσουμε στην εξομοίωση του κυκλώματος.

### Γ. Εξομοίωση του Σχεδιασμού

Για την διαδικασία εξομοίωσης θα χρησιμοποιήσουμε το εργαλείο ModelSim. Παρότι γνωρίζετε αρκετά τη χρήση του εργαλείου, τα βασικά βήματα επαναλαμβάνονται παρακάτω. Από το μενού File -> Change working directory, υποδείξτε στο εργαλείο το folder που περιέχει τα αρχεία σας. Από το μενού Library->Create New Library, δημιουργείστε μια καινούργια βιβλιοθήκη με όνομα π.χ. work. Σκοπός της βιβλιοθήκης είναι να κρατά όλα τα objects που θα δημιουργηθούν κατά την διαδικασία μετάφρασης του κώδικά σας. Με το πάνω αριστερά εικονίδιο της γραμμής εργαλείων καλείτε τον Verilog compiler, που όπως κάθε άλλος compiler θα ελέγχει το συντακτικό και την λογική του κώδικά σας και θα δημιουργήσει τα objects της βιβλιοθήκης. Υποδείξτε τα δύο αρχεία που θέλετε να μεταφραστούν και εκτελέστε την διαδικασία μετάφρασης. Οταν αυτή ολοκληρωθεί χωρίς λάθη είστε έτοιμοι να προχωρήσετε στην διαδικασία παραγωγής κυματομορφών. Σε περίπτωση λαθών, ο μεταφραστής υποδεικνύει την γραμμή λάθους και μπορείτε να διορθώσετε τα λάθη σας με το button edit source. Ο editor που θα παρουσιαστεί επιτρέπει την διαδικασία cross probing με τον μεταφραστή και είναι ειδικά διαμορφωμένος ώστε να αναλύει τα keywords και τις σταθερές που χρησιμοποιείτε και να τις παρουσιάζει με διαφορετικά χρώματα σε σχέση με τις μεταβλητές σας.

Εχοντας ολοκληρώσει την διαδικασία μετάφρασης, στην βιβλιοθήκη σας έχουν δημιουργηθεί τα δύο objects testcounter και counter όπως μπορείτε να διαπιστώσετε με την εντολή Library -> View Library Contents. Από το μενού File -> Load New Design επιλέξτε το object testcounter. Επειδή στον κώδικα αυτού του object είχατε συμπεριλάβει και ένα αντίτυπο του μετρητή, το εργαλείο θα φορτώσει αυτόματα και το object του μετρητή σας. Από το μενού View επιλέξτε structure και θα εμφανιστεί ένα παράθυρο που σας δείχνει την ιεραρχία του σχεδιασμού σας. Επιλέγοντας signals θα δείτε όλα τα σήματα που αναφέρονται στο συγκεκριμένο object της ιεραρχίας που έχετε επιλέξει καθώς και οι τιμές τους την τρέχουσα χρονική στιγμή. Εφόσον δεν έχετε τρέξει εξομοίωση όλες οι τιμές είναι σε x (άγνωστη) κατάσταση. Επιλέξτε όλα τα σήματα του testcounter και στο παράθυρο των σημάτων δώστε την εντολή View -> Wave -> Selected Signals.

Αυτόματα ανοίγει ένα παράθυρο κυματομορφών για τα επιλεγμένα σήματα. Για να τρέξετε εξομοίωση χρησιμοποιείστε το πρώτο από τα τέσσερα τελευταία buttons του παραθύρου κυματομορφών. Με αυτό θα πάρετε εξομοίωση 100 ns και μπορείτε να το χρησιμοποιείτε επαναληπτικά. Με το View -> Source μπορείτε να βλέπετε τον κώδικά σας. Ακριβώς όπως ένα debugging εργαλείο σας επιτρέπει να κάνετε step κατά την εκτέλεση του κώδικά σας, το ίδιο ισχύει και για τον συγκεκριμένο εξομοιωτή. Μπορείτε να επιλέξετε step με ένα από τα δύο buttons του παραθύρου κώδικα και το βέλος στα αριστερά δείχνει το επόμενο σημείο του κώδικα προς εκτέλεση. Επιπλέον το εργαλείο σας ανακοινώνει περί της χρονικής στιγμής που θα συμβεί το επόμενο γεγονός. Λόγω της παραλληλίας του υλικού, πιθανόν να υπάρχουν περισσότερα του ενός βέλη. Συνεχίστε την εξομοίωσή σας μέχρι να έχετε αποτελέσματα για μια πλήρη περιοχή τιμών του μετρητή (μέχρι να πάρει όλες τις 128 διαφορετικές τιμές του. Τι συμβαίνει μετά την τιμή 127 και γιατί;). Ελέγχετε τα αποτελέσματα της εξομοίωσης και βεβαιωθείτε ότι όλες οι λογικές λειτουργίες που προβλέφθηκαν από τα διανύσματα εξομοίωσης υλοποιούνται σωστά. Αν θέλετε μπορείτε να προσθέσετε ή να τροποποιήσετε τα διανύσματα εξομοίωσης μέσω κώδικα στο αρχείο testcounter.v. Δεν χρειάζεται να βγείτε από το εργαλείο για κάτι τέτοιο. Ωστόσο, αφού αλλάξετε τον κώδικα, χρειάζεται να αποθηκεύσετε τις αλλαγές και να ξανακάνετε μετάφραση. Ακολούθως απλά επιλέξτε από το File -> Restart για να ξεκινήσει η διαδικασία εξομοίωσης πάνω στον καινούργιο κώδικα.

### Δ. Ζητούμενα της άσκησης

- (1) Τροποποιείστε τον δεδομένο κώδικα ώστε να υλοποιεί έναν μετρητή των 4 δυαδικών ψηφίων με τα ίδια χαρακτηριστικά.
- (2) Χρησιμοποιείστε δύο τέτοιους μετρητές όση λογική κρίνετε απαραίτητη για να υλοποιήσετε έναν μετρητή των 8 δυαδικών ψηφίων. Που θα πρέπει να συνδεθεί η είσοδος επίτρεψης του high order nibble ;
- (3) Δημιουργείστε ένα σύνολο διανυσμάτων εξομοίωσης για τον μετρητή των 8 δυαδικών ψηφίων.

- (4) Εξομοιώστε τον σχεδιασμό σας και επαληθεύστε την λογική λειτουργία του.
- (5) Τροποποιείστε τον κώδικά σας ώστε ο μετρητής να έχει σύγχρονη είσοδο καθαρισμού και επαναλάβετε τα β, γ και δ.
- (6) Εισάγετε μια καθυστέρηση 20 χρονικών στιγμών στη λειτουργία του μετρητή των 4 δυαδικών ψηφίων. Ποια είναι τότε η μέγιστη συχνότητα λειτουργίας του μετρητή των 8 δυαδικών ψηφίων ;



## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 6

Σκοπός της εργαστηριακής άσκησης είναι μια πρώτη γνωριμία σας με τις διαδικασίες του rapid system prototyping. Θα περιγράψουμε ένα κύκλωμα σε HDL, θα πάρουμε την ισοδύναμη υλοποίησή του σε CLBs μέσω σύνθεσης και τέλος θα φτιάξουμε ένα πρωτότυπο αυτού του κυκλώματος στις αναπτυξιακές μας πλακέτες.

Πριν από οτιδήποτε άλλο, πρέπει να διαβάσετε (ακόμη κι αν το έχετε κάνει ήδη πρέπει να το ξανακάνετε) τα manuals :

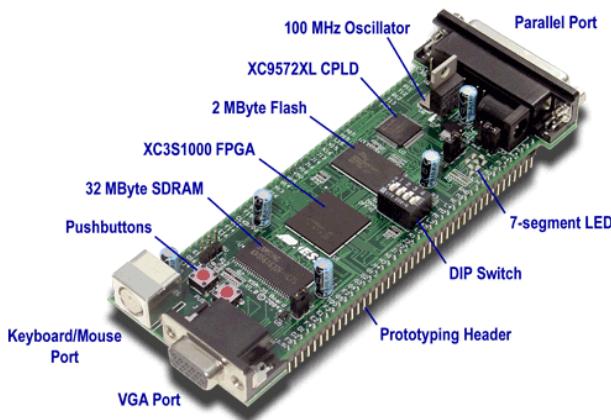
- (α) xsa-3S-manual-v1\_1.pdf
- (β) xst-manual-v4\_0.pdf

και να έχετε πάντοτε σε χρήση τα Excel αρχεία :

- (α) XSA-3S-pins.xls
- (β) XSA+XST4-pins.xls

### A. Σύντομη περιγραφή των αναπτυξιακών πλακετών XSA-3S και XSTend v.4.0

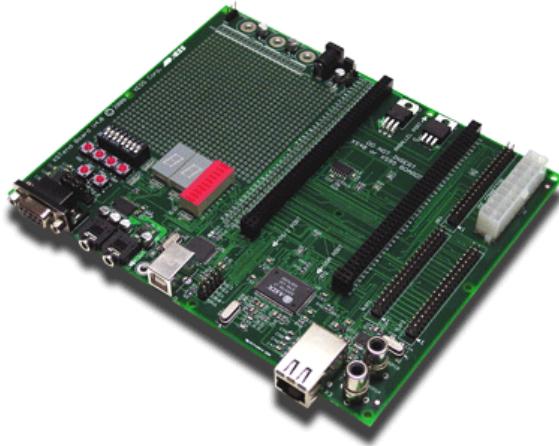
Για την πρωτότυποποίηση των σχεδιασμών μας θα χρησιμοποιήσουμε ένα FPGA χωρητικότητας 1.000.000 ισοδύναμων πυλών, της σειράς SPARTAN 3 της εταιρείας Xilinx. Το FPGA αυτό για εργαστηρικούς λόγους διατίθεται και στη μορφή της αναπτυξιακής πλακέτας XSA-3S που έχει κατασκευάσει η εταιρεία XESS και που απεικονίζεται στην παρακάτω εικόνα :



Στην πλακέτα αυτή προσφέρεται εκτός από το FPGA μια σειρά από interfaces διασύνδεσης (VGA, PS/2, Parallel Port) μαζί με τους απαραίτητους υποσχεδιασμούς τους καθώς και μια σειρά από σχεδιασμούς ελέγχου και παρατήρησης (dip switches, push buttons, 7 segment leds). Επίσης παρέχονται μνήμες :

(α) SDRAM, για bulk αποθήκευση ενδιάμεσων αποτελεσμάτων,  
(β) Flash, για αποθήκευση κώδικα προγραμματισμού του ολοκληρωμένου ή για reconfiguration και τέλος ένας προγραμματιζόμενος χρονιστής και ένα CPLD για τον έλεγχο των πηγών προγραμματισμού του FPGA και διάφορων άλλων λειτουργιών. Για όλες τις εργαστηριακές μας ασκήσεις καθώς και για την εξαμηνιαία εργασία σας, θα συνδέσουμε την πλακέτα αυτή με ένα προσωπικό υπολογιστή μέσω της παράλληλης θύρας και θα προγραμματίζουμε το FPGA με κώδικα που θα παράγεται στο PC και θα μεταβιβάζεται στο FPGA μέσω αυτής της σύνδεσης.

Παρότι η πλακέτα αυτή είναι standalone (χρειάζεται μόνο τροφοδοσία για να λειτουργήσει) μπορούμε να επεκτείνουμε επιπλέον τις δυνατότητές της προσκολλώντας την σε μια πλακέτα XSTend v.4.0 (XST4), η οποία μας δίνει επιπλέον interfaces (serial, audio USB, Ethernet, video), πάρα πολλούς ακροδέκτες παρατήρησης (headers) και σχεδιασμούς ελέγχου και παρατήρησης, χώρο για ανάπτυξη των δικών μας κυκλωμάτων και τη δυνατότητα τροφοδοσίας από ATX τροφοδοτικό. Η πλακέτα XST4 φαίνεται στην παρακάτω φωτογραφία:

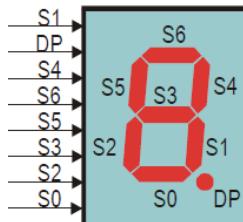


Στο εργαστήριο υπάρχει πάντα ένα ελεγμένο έτοιμο τέτοιο ζεύγος πλακετών προς χρήση, μαζί με το απαραίτητο τροφοδοτικό και το καλώδιο προγραμματισμού. Σε καμμία περίπτωση δε θα πρέπει να προσπαθήσετε να διαχωρίσετε τις δύο αυτές πλακέτες, να αλλάξετε σχήμα τροφοδοσίας ή να αλλάξετε τρόπο προγραμματισμού του FPGA.

Στην παρούσα εργαστηριακή άσκηση, θα χρησιμοποιήσουμε μόνο την πλακέτα XSA και θα χρησιμοποιούμε τη πλακέτα XST απλά για τροφοδοσία του συστήματός μας. Σκοπός του 1<sup>ου</sup> μέρους της άσκησης είναι να περιγράψουμε μια πύλη XOR 4 εισόδων, να τροφοδοτήσουμε τις εισόδους της από τα dip switches (υπενθυμίζεται : της πλακέτας XSA) και να βλέπουμε την έξοδο σαν αριθμό (0 ή 1) στο 7 segment.

#### B. HDL περιγραφή του σχεδιασμού

Το κύκλωμα που θα χρειστεί να περιγράψουμε έχει 4 εισόδους (a, b, c, d που θα συνδεθούν στα dip switches) και 8 εξόδους (s[7:0] και dp που θα οδηγούν τα 8 led του 7 segment).



Θεωρούμε ότι η έξοδος s[i] οδηγεί το λαμπάκι Si της παραπάνω εικόνας και ότι η έξοδος dp το DP του 7 segment. Άρα για να βλέπουμε το 0, θα πρέπει να είναι s[0]=s[1]=s[2]=s[4]=s[5]=s[6] = 1 και s[3]=0, ενώ για να βλέπουμε το 1, θα πρέπει να είναι s[1]=s[4]=1 και όλα τα υπόλοιπα 0. Σε κάθε περίπτωση υποθέστε ότι dp=1.

#### **(αρχείο 6/mlx.v)**

```
module mlx (a, b, c, d, s, dp);
    input a, b, c, d;
    output [6:0] s;
    output      dp;
    wire o;

    assign o = a^b^c^d;
    assign s[6:0] = o ? 7'b0010010 : 7'b1110111;
    assign dp = 1;
endmodule
```

#### C. Περιγραφή των διανυσμάτων εξομοίωσης

Για την εξομοίωση του κυκλώματός μας, απλά βάζουμε όλες τις πιθανές τιμές στις εισόδους του :

#### **(αρχείο 6/testmlx.v)**

```
module testmlx ();
    reg a, b, c, d;
    wire [6:0] s;
    wire      dp;

    mlx i0 (a, b, c, d, s, dp);
    initial {a,b,c,d} <= 0;
```

```

always #100 {a,b,c,d} <= {a,b,c,d} + 1;
endmodule

```

#### D. UCF

Για τη σύνθεση του κυκλώματός μας, θα χρησιμοποιήσουμε το ISE της εταιρείας Xilinx (συγκεκριμένα την έκδοση WebPack, που είναι περιορισμένη ως προς τις σειρές FPGAs που υποστηρίζει, αλλά διατίθεται δωρεάν). Το εργαλείο αυτό είναι στην πράξη μια ολόκληρη πλατφόρμα εργαλείων (γραφικός editor + HDL editor + simulator + synthesis + place & route + bitgen), οπότε θα επικεντρωθούμε μόνο στα σημεία ενδιαφέροντος. Ενδεχόμενα να χρειαστεί να καταφύγετε στην online βοήθεια αρκετές φορές για να εντοπίσετε συγκεκριμένες ενέργειες του εργαλείου.

Για να έχουμε την ικανότητα υλοποίησης του κυκλώματός μας στην πλακέτα XSA θα πρέπει πέρα από τη περιγραφή του σχεδιασμού μας, να υποδείξουμε στο εργαλείο σύνθεσης, σε ποιους ακροδέκτες του FPGA θα συνδεθούν τα σήματα του σχεδιασμού μας. Αυτό είναι απαραίτητο, καθώς στην πλακέτα XSA μόνο συγκεκριμένα pins του FPGA συνδέονται με συγκεκριμένους άλλους σχεδιασμούς. Αυτή η διασύνδεση περιγράφεται αναλυτικά τόσο στο εγχειρίδιο της XSA όσο και στο αντίστοιχο Excel αρχείο.

A	B	C	D	E	F	G	H
<b>Connections Between the FPGA, CPLD and other Components on the XSA-3</b>							
Net Name	FPGA Pin (U1)	CPLD Pin (U2)	Parallel Port	LEDs	Switch / Buttons	SDRAM (U4)	Flash (U3) O
FPGA-CCLK	T15	52					
FPGA-DIN-D0	M11	1		LED-C (S1)			DQ0 (29)
FPGA-D1	N11	2		LED-DP			DQ1 (31)
FPGA-D2	P10	4		LED-B (S4)			DQ2 (33)
FPGA-D3	R10	5		LED-A (S6)			DQ3 (35)
FPGA-D4	T7	7		LED-F (S5)			DQ4 (38)
FPGA-D5	R7	9		LED-G (S3)			DQ5 (40)
FPGA-D6	N6	10		LED-E (S2)			DQ6 (42)
FPGA-D7	M6	11		LED-D (S0)			DQ7 (44)
FPGA-DONE	R14	6					

Για παράδειγμα, στο απόσπασμα του Excel για την πλακέτα XSA της παραπάνω φωτογραφίας βρίσκουμε ότι το S0 led του 7 segment δυνδέεται με το ποδαράκι M6 του FPGA και συνεπώς πρέπει να ζητήσουμε από το εργαλείο της σύνθεσης να φτιάξει έτσι το κύκλωμά μας ώστε να συνδέσει το σήμα s[0] στο ποδαράκι M6. Όλοι αυτοί οι περιορισμοί που ζητάμε από το εργαλείο της σύνθεσης, αποτελούν ένα αρχείο περιορισμών (user constraints file) το οποίο θα πρέπει να επισυνάψουμε με το σχεδιασμό μας. Για το παράδειγμά μας, αυτό το αρχείο έχει ως εξής (αρχείο 6/mlx.ucf)

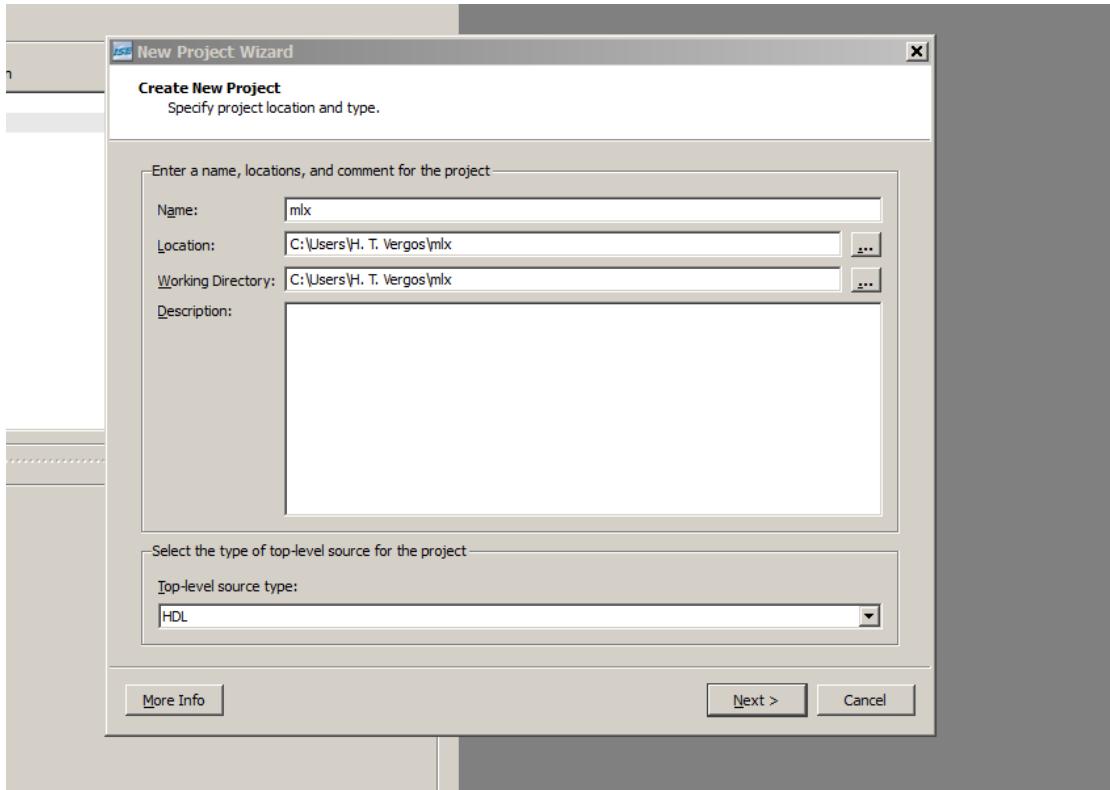
```

net a loc=k4;
net b loc=k3;
net c loc=k2;
net d loc=j4;
net s<6> loc=r10;
net s<5> loc=t7;
net s<4> loc=p10;
net s<3> loc=r7;
net s<2> loc=n6;
net s<1> loc=m11;
net s<0> loc=m6;
net dp loc=n11;

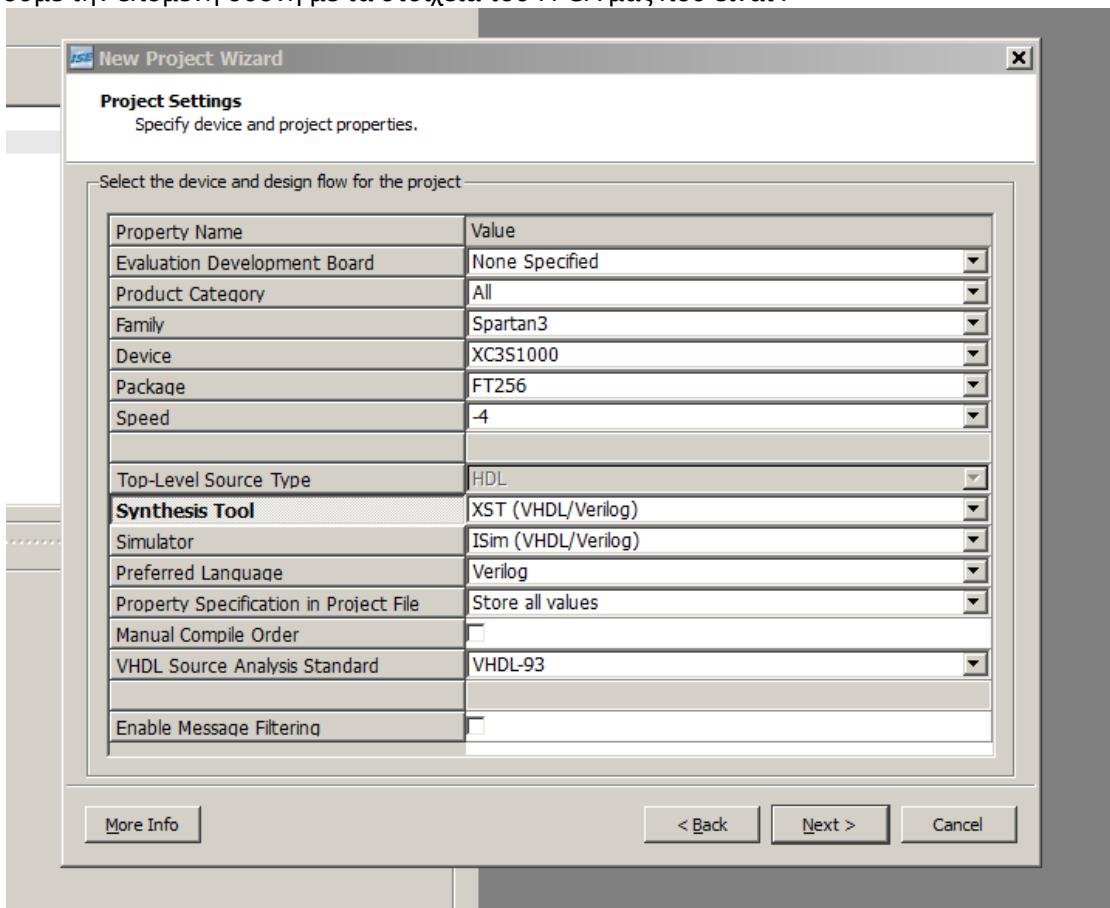
```

#### E. Σύνθεση του κυκλώματός μας και επισκόπηση των αποτελεσμάτων

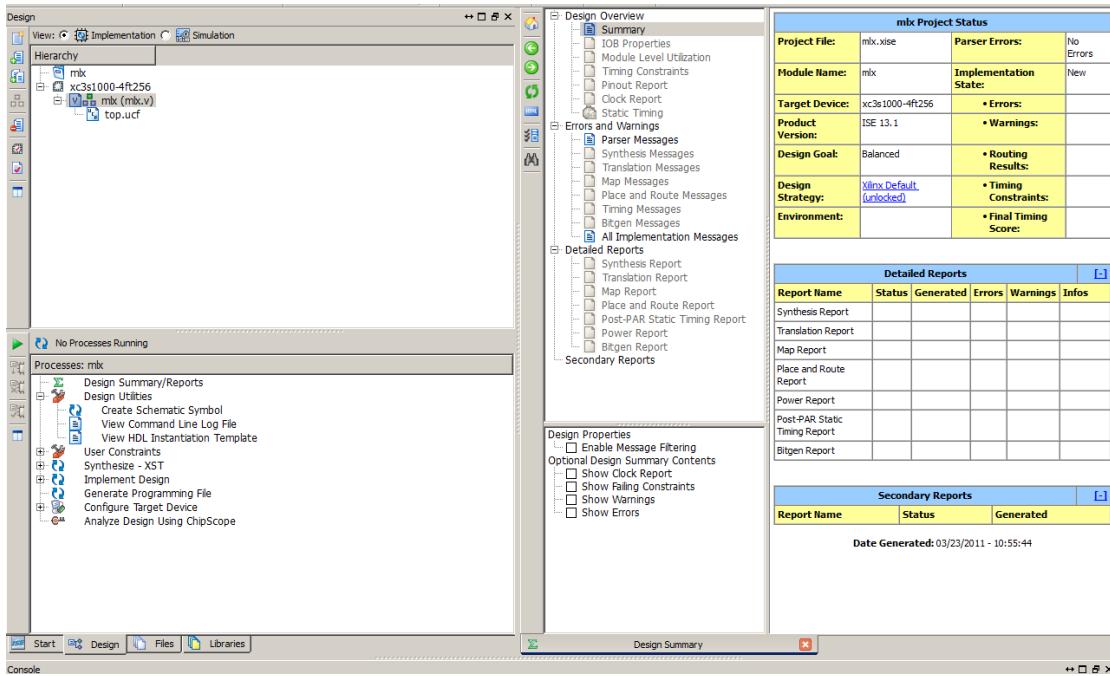
Ξεκινάμε το ISE Project Navigator που είναι ένα για όλο το φάσμα των εργαλείων του ISE. Επιλέγουμε New Project, δίνουμε ένα όνομα (π.χ. mlx) και έναν κατάλογο στον οποίο θα δημιουργηθεί ένα ολόκληρο folder με το παραπάνω όνομα για αυτό το project. Στο κάτω μέρος αφήνουμε τη default επιλογή ότι δηλαδή το κορυφαίο σχεδιαστικό μας κομμάτι θα είναι περιγεγραμμένο σε HDL (θυμηθείτε ότι το ISE έχει και γραφικό editor και πολλά άλλα).



Συμπληρώνουμε την επόμενη οθόνη με τα στοιχεία του FPGA μας που είναι :

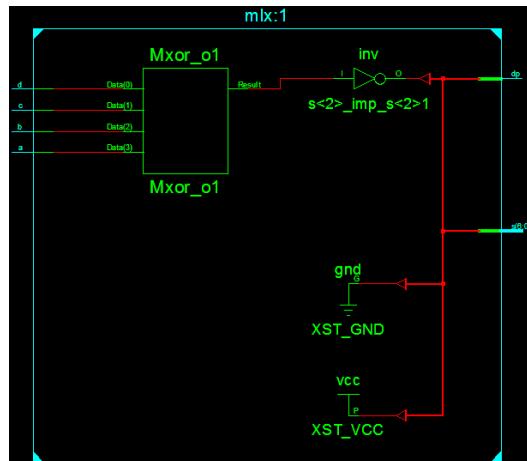


Λέμε δηλαδή στο εργαλείο ότι θα προγραμματίσουμε ένα XC3S1000 FPGA της Spartan 3 σειράς ταχύτητας 4 και σε tiny package 256 pins. Του λέμε επίσης ότι θα πρέπει να χρησιμοποιήσει το δικό του εργαλείο σύνθεσης και παρότι αφήνουμε το default εξομοιωτή, μπορούμε να του πούμε να διασυνδεθεί με την ήδη εγκατεστημένη έκδοση του ModelSim που τυχόν διαθέτουμε. Αφού αποδεχθούμε και την επόμενη διαλογική εικόνα, μεταφερόμαστε στο project navigator. Στη συνέχεια θα πρέπει να προσθέσουμε τα αρχεία μας στο project που μόλις δημιουργήσαμε μέσω του Project -> Add Source. Προσθέτουμε λοιπόν τόσο το mlx.v όσο και το mlx.ucf (όχι το testbench, αφού δεν περιγράφει τίποτε για τον σχεδιασμό μας) και έτσι καταλήγουμε στην εξής οθόνη :

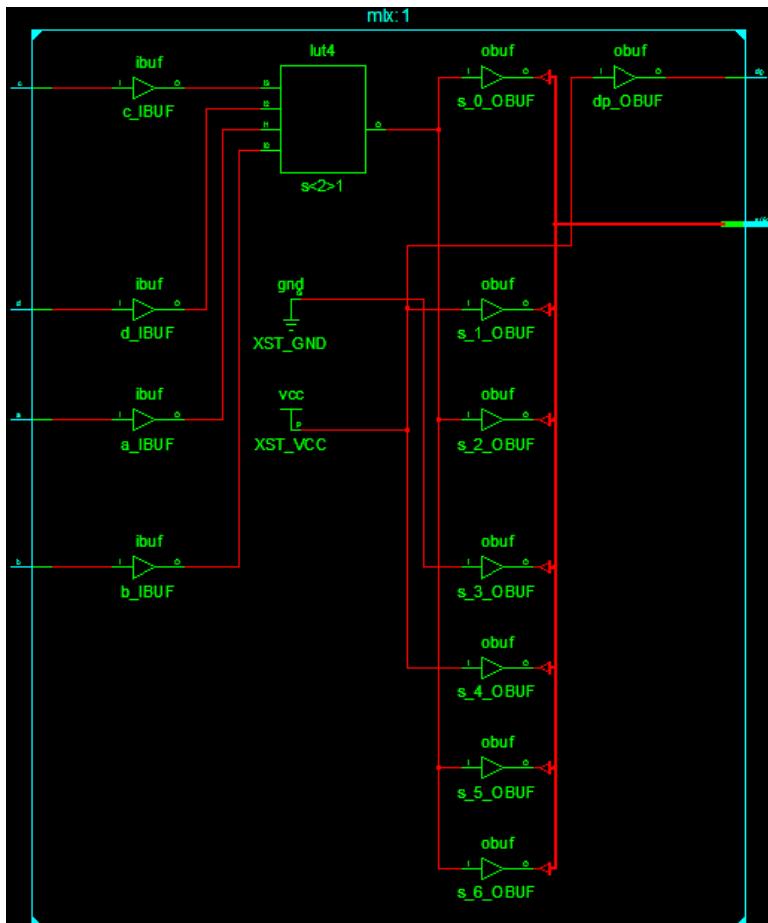


όπου πάνω αριστερά είναι η ιεραρχία του σχεδιασμού μας, κάτω αριστερά το τι μπορούμε να τρέξουμε ανάλογα με το τι θα επιλέξουμε από την ιεραρχία και δεξιά τα διάφορα reports που έχουν προκύψει.

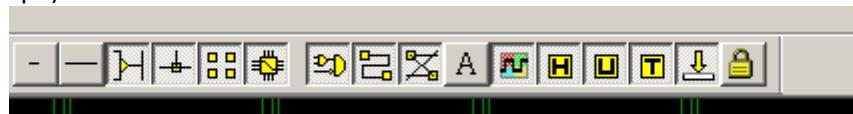
Επιλέγουμε Process -> Implement Top module, και αυτόματα θα κληθούν τα εργαλεία μετάφρασης, σύνθεσης, απεικόνισης σε CLBs και Place & Route. Ας καλέσουμε κάποια από τα διαφορετικά εργαλεία για να δούμε τι ακριβώς μετατροπές έχουν γίνει στον σχεδιασμό μας. Επιλέγουμε Tools -> Schematic Viewer -> RTL και αφού αποδεχθούμε τις αρχικές επιλογές θα δούμε ένα σχηματικό παράθυρο του top level module. Χτυπώντας πάνω στο σύμβολο, θα δούμε την υλοποίησή του :



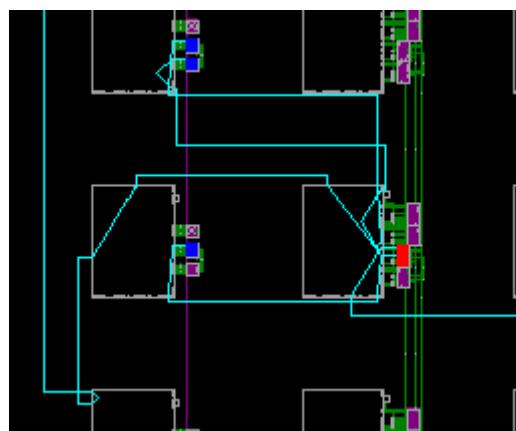
Κλείστε το παράθυρο του σχηματικού αυτού και επιλέξτε Tools -> Schematic Viewer -> Technology, αποδεχθείτε τις αρχικές επιλογές και πατήστε πάλι πάνω στο top level module του σχηματικού. Θα πάρετε έτσι το σχηματικό για τη συγκεκριμένη τεχνολογία που θα μιάζει με αυτό της εικόνας :



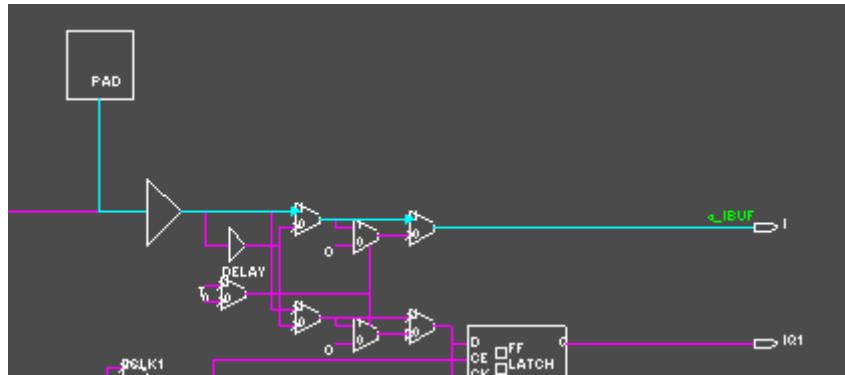
Δηλαδή, ο σχεδιασμός σας θα υλοποιηθεί με κάποιους input buffers, κάποιους output buffers και όλη σας η λογική με ένα LookUpTable 4 μεταβλητών. Ας δούμε στη συνέχεια, πως πράγματι αυτά μεταφράστηκαν στο πραγματικό κόσμο, εντός δηλαδή του FPGA που θέλαμε. Επιλέξτε Tools -> FPGA Editor -> Post Place & Route. Το εργαλείο που μας παρουσιάζεται, μας δείχνει την πραγματική υλοποίηση του κυκλώματος μας εντός του FPGA. Αφού μεγιστοποιήσετε το Array 1 υποπαράθυρο, από την πάνω πλευρά κάντε ορατά περισσότερα resources του FPGA κάνοντας τις εξής επιλογές :



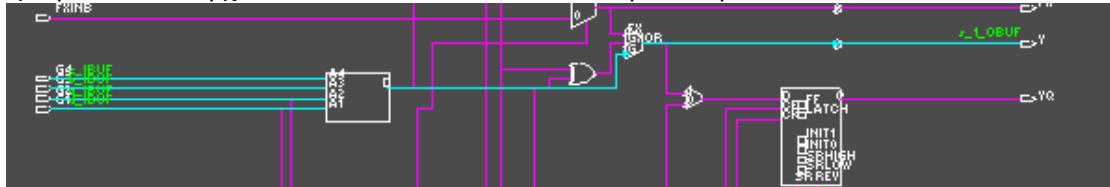
στα visibility options. Κάντε zoom in και προσπαθείστε να εντοπίσετε resources που χρησιμοποιούνται όπως αυτά της παρακάτω εικόνας στην αριστερή πλευρά του FPGA (τα πλέον αριστερά είναι IOB, ενώ το κόκκινο μέρος ενός CLB)



Χτυπώντας σε κάποιο από τα μπλε χρησιμοποιούμενα resources θα δείτε πως ακριβώς χρησιμοποιείται. Για παράδειγμα στην παρακάτω εικόνα :



φαίνεται ο σχεδιασμός που υλοποιείται εντός του ακροδέκτη k4. Υλοποιεί ένα ριπ εισόδου για το σήμα a όπως ακριβώς ζητήσαμε στο .ucf αρχείο. Σε ένα από τα διπλανά CLBs βλέπουμε να υλοποιείται :



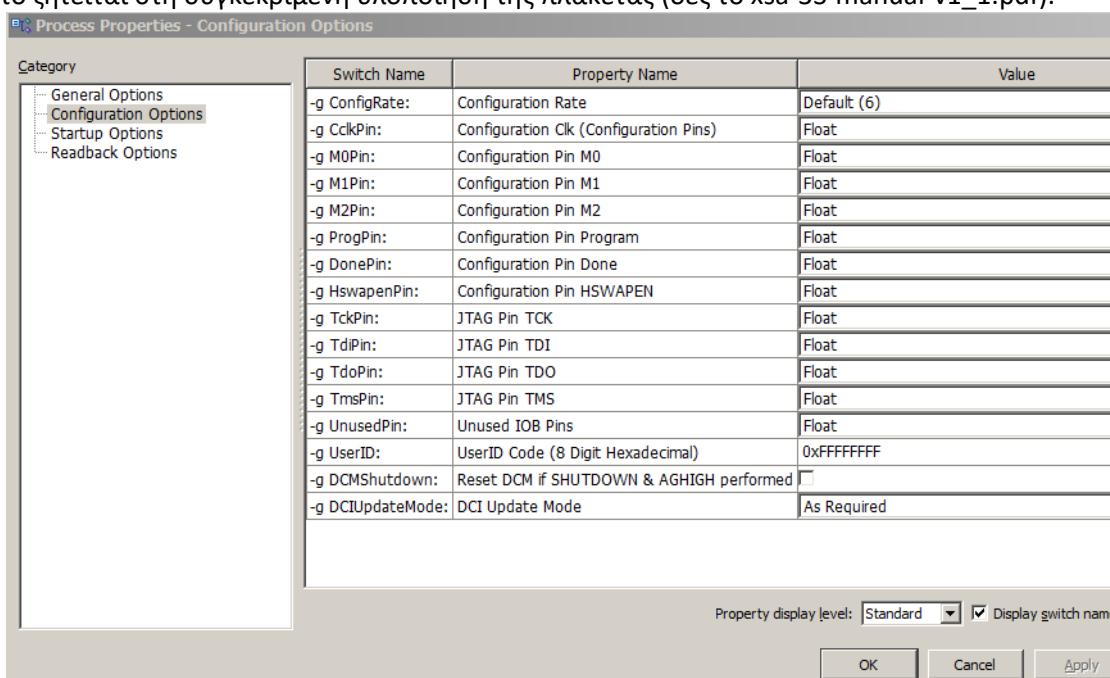
μια συνδυαστική συνάρτηση 4 μεταβλητών σε ένα LUT. Χτυπώντας πάνω στο LUT, παίρνουμε τη συνάρτησή του :

```
<G>=(A3@(A2@(A1@A4))); bel <Mxor_o_xo<0>1>.
```

Αφού κλείστε όλα τα εργαλεία τα οποία είδαμε ως τώρα, ας πάμε να προσδιορίσουμε τη συχνότητα λειτουργίας του σχεδιασμού μας. Θα χρησιμοποιήσουμε το Tools -> Timing Analyzer σε Post Place & Route mode. Στο παράθυρο που ανοίγει επεκτείνετε το Data Sheet report και επιλέξτε Pad to Pad. Εμφανίζονται τα χειρότερα μονοπάτια καθυστέρησης από κάθε είσοδο σε κάθε έξοδο (στο σχεδιασμό μας μόνο τα s[1] και s[4] αλλάζουν τιμές). Από εκεί προκύπτει ότι η χειρότερη καθυστέρηση του σχεδιασμού μας είναι 13.070ns.

### ΣΤ. Υλοποίηση του κυκλώματός μας

Αφού ο σχεδιασμός μας έχει υλοποιηθεί, το επόμενο βήμα είναι η δημιουργία του αρχείου προγραμματισμού. Αφού κλείστε όσα εργαλεία είναι ακόμη ανοιχτά, επιλέξτε στην ιεραρχία το mlx.v και στο κάτω αριστερά παράθυρο, κάντε δεξί κλικ στο Generate Programming File. Στο νέο παράθυρο που ανοίγει επιλέξτε Configuration Options και επιλέξτε να απομακρυνθούν όλες οι pull up αντιστάσεις που βάζει το εργαλείο στα configuration pins μιας και αυτό ζητείται στη συγκεκριμένη υλοποίηση της πλακέτας (δες το xsas3s-manual-v1\_1.pdf).



Αφού λοιπόν επιλέξουμε τις επιλογές που φαίνονται στην εικόνα, τις αποδεχόμαστε και κάνουμε διπλό κλικ στο Generate Programming file, οπότε και θα δημιουργηθεί ένα αρχείο mlx.bit στον κατάλογο του σχεδιασμού μας, που είναι το αρχείο προγραμματισμού. Ήρθε η ώρα να προγραμματίσουμε το FPGA μας.

Για την επικοινωνία της πλακέτας μας με το FPGA, θα χρησιμοποιήσουμε τα εργαλεία της XESS. Παρότι υπάρχουν αρκετά εμείς θα χρησιμοποιήσουμε μόνο το GXSLOAD. Δώστε λοιπόν τροφοδοσία στις πλακέτες σας και τρέξτε το GXSLOAD. Επιλέξτε Board Type XSA3S1000, Port επικοινωνίας το LPT1 και σύρτε το αρχείο mlx.bit εντός του χώρου FPGA/CPLD, για να δείξετε που θέλετε να αποθηκευτεί το αρχείο σας (με το ίδιο εργαλείο μπορούμε να προγραμματίσουμε και τις μνήμες της XSA). Πατείστε το Load και όταν ολοκληρωθεί η διαδικασία, έχετε πλέον ένα λειτουργικό πρωτότυπο !!! Χαρείτε το !!! Αλλάξτε κάποιο από τα dip switches (με τη βοήθεια της μύτης ενός μολυβιού) και δείτε πως αλλάζει και η ένδειξη στα leds. (Υπόδειξη: το λογικό 1 στα dip switches είναι μάλλον ανάποδα από ότι θα περιμένατε).

### Z. Ζητούμενα

(1) Περιγράψτε ένα κύκλωμα το οποίο θα διαβάζει έναν δυαδικό αριθμό από τα dip switches και θα τον απεικονίζει στα led bars. Παραδοτέα : ο κώδικας περιγραφής, η μέγιστη συχνότητα λειτουργίας του κυκλώματός σας και το αρχείο προγραμματισμού.

(2) Περιγράψτε ένα κύκλωμα το οποίο θα πολλαπλασιάζει 2 αριθμούς, ο καθένας των 2 δυαδικών ψηφίων και θα απεικονίζει το αποτέλεσμά τους στο 7 segment. Εξετάστε 2 πιθανότητες : (α) οι αριθμοί να είναι μη προσημασμένοι και (β) οι αριθμοί να είναι σε παράσταση συμπληρώματος του 2. Χρησιμοποιείστε το DP του 7 segment για να υποδείξετε το πρόσημο του αποτελέσματος. Παραδοτέα για κάθε υποπερίπτωση : ο κώδικας περιγραφής, η μέγιστη συχνότητα λειτουργίας και το αρχείο προγραμματισμού.

## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 7

Σκοπός της εργαστηριακής άσκησης είναι να φτιάξουμε ένα ακολουθιακό κύκλωμα χρησιμοποιώντας και τις 2 αναπτυξιακές μας πλακέτες ελαφρά μεγαλύτερης δυσκολίας από αυτό της προηγούμενης άσκησης. Αντικείμενο είναι αρχικά να φτιάξουμε έναν μετρητή δευτερολέπτων ο οποίος θα απεικονίζει τη μέτρησή του στα δύο 7 segment του XST board και κατόπιν να τον τροποποιήσετε ώστε να παρέχει επιπλέον δυνατότητες.

### A. Συχνότητα 1Hz

Η πλακέτα XSA έχει ενσωματωμένο έναν χρονιστή συχνότητας 100MHz. Συνεπώς για να πάρουμε την επιθυμητή συχνότητα του 1Hz θα πρέπει να διαιρέσουμε τη βασική με  $10^8$ . Προφανώς αυτό μπορούμε να το πετύχουμε ανιχνεύοντας πότε ένας μετρητής με  $\lceil \log_2 10^8 \rceil = 27$  δυαδικά ψηφία φθάνει σε μία συγκεκριμένη τιμή μέτρησης, αλλά κάτι τέτοιο θα απαιτούσε τη σύγκριση 27 ψηφίων, πράγμα που ενδεχόμενα να μας έριχνε σημαντικά τη μέγιστη συχνότητα λειτουργίας του σχεδιασμού μας και συνεπώς θα χάλαγε την ακρίβεια του ρολογιού μας. Συνεπώς θα φτιάξουμε μικρότερους μετρητές που η εξάντληση της ακολουθίας μέτρησης του ενός θα είναι η επίτρεψη μέτρησης του άλλου. Επειδή είναι  $10^8 = 25^4 * 2^8$  θα φτιάξουμε ένα μετρητή mod25 και έναν 8 bit και θα χρησιμοποιήσουμε 4 από το πρώτο είδος και 1 από το δεύτερο σε μια αλυσίδα επίτρεψης.

Εδώ υπάρχει ένα κρυφό και ταυτόχρονα καίριο σημείο. Γιατί να φτιάξουμε μετρητές με είσοδο επίτρεψης και όχι απλούς μετρητές παίρνοντας τη πιο σημαντική έξοδο του ενός ως ρολόι του επόμενου? Η απάντηση σε αυτό το ερώτημα έχει να κάνει με τους φυσικούς πόρους που διαθέτει το FPGA. Όπως εξηγήθηκε στο μάθημα, το FPGA διαθέτει ειδικούς buffers για σήματα ρολογιού, οι οποίοι όμως είναι περιορισμένοι. Αν συνεπώς χρησιμοποιήσουμε μόνο ένα ρολόι, θα χρησιμοποιηθεί γι' αυτό ο BUFGP (global buffer) με αποτέλεσμα να έχουμε το μικρότερο δυνατό skew ανάμεσα στο ρολόι που λαμβάνουν όλα τα flip flops του σχεδιασμού μας. Έτσι λοιπόν, ο κώδικας για τη συχνότητα 1 Hz είναι ο ακόλουθος:

#### (αρχείο 7/cnt25.v)

```
module cnt25 (reset, clk, enable, clkdiv25);
    input reset, clk, enable;
    output clkdiv25;
    reg [5:0] cnt;

    assign clkdiv25 = (cnt==5'd24);
    always @(posedge reset or posedge clk)
        if (reset) cnt <= 0;
        else if (enable)
            if (clkdiv25) cnt <= 0;
            else cnt <= cnt + 1;
    endmodule

module cnt8b (reset, clk, enable, clkdiv256);
    input reset, clk, enable;
    output clkdiv256;
    reg [7:0] cnt;

    assign clkdiv256 = (cnt==8'd255);
    always @(posedge reset or posedge clk)
        if (reset) cnt <= 0;
        else if (enable) cnt <= cnt + 1;
    endmodule

module OneHertz(reset, clk, en_nxt);
    input clk, reset;
    output en_nxt;
    wire clk1Hz;
    wire first, second, third, fourth;

    cnt25 i0 (reset, clk, 1'b1, first);
    cnt25 i1 (reset, clk, first, second);
```

```

cnt25 i2 (reset, clk, first & second, third);
cnt25 i3 (reset, clk, first & second & third, fourth);
cnt8b i4 (reset, clk, first & second & third & fourth, clk1Hz);
assign en_nxt = first & second & third & fourth & clk1Hz;
endmodule

```

Μπορούμε να επιβεβαιώσουμε τη λειτουργία του κυκλώματος παραγωγής της συχνότητας 1Hz, εξομοιώνοντάς το με τον ακόλουθο κώδικα :

```

module testOneHertz();
reg reset, clk;
wire en_nxt;

OneHertz CUT (reset, clk, en_nxt);
initial begin reset=0; clk = 0; # 10 reset = 1; # 9 reset = 0; end
always #1 clk=~clk;
endmodule

```

### B. Μετρητής δευτερολέπτων

Για να μετρήσουμε τα δευτερόλεπτα, θα μπορούσαμε απλά να χρησιμοποιήσουμε έναν δυαδικό μετρητή 6 δυαδικών ψηφίων με αναδίπλωση μετά την τιμή 59. Ωστόσο για την απεικόνιση αυτής της λύσης στα δύο 7-segments θα έπρεπε μετά να υπολογίζουμε το πηλίκο και το υπόλοιπο της μέτρησης ως προς 10. Μια εναλλακτική πρόταση είναι να χρησιμοποιήσουμε 2 εναλλακτικούς μετρητές έναν για τις μονάδες των δευτερολέπτων και έναν για τις δεκάδες. Ο πρώτος είναι ένας δεκαδικός μετρητής, ενώ ο δεύτερος ένας μετρητής modulo 6. Ο κώδικας γι' αυτή τη δεύτερη λύση, είναι ο ακόλουθος (προσέξτε ότι εξακολουθούμε να χρησιμοποιούμε παντού τη λογική του ενός ρολογιού) :

#### **(αρχείο 7/secondcounter.v)**

```

module singliseconds (reset, clk, enable, ss, nxt);
input reset, clk, enable;
output [3:0] ss;
output   nxt;
reg   [3:0] ss;

assign nxt= (ss == 4'd9);
always @(posedge clk or posedge reset)
  if (reset) ss <= 4'd0;
  else if (enable)
    if (nxt) ss <= 0;
    else ss <= ss + 1;
endmodule

module tenthsofseconds (reset, clk, enable, ts);
input reset, clk, enable;
output [2:0] ts;
wire   again;
reg   [2:0] ts;

assign again = (ts == 3'd5);
always @(posedge clk or posedge reset)
  if (reset) ts <= 4'd0;
  else if (enable)
    if (again) ts <= 0;
    else ts <= ts + 1;
endmodule

module secondcounter (reset, clk, enable, ts, ss);
input reset, clk, enable;
output [2:0] ts;

```

```

output [3:0] ss;
wire      ent;

singleseconds i0 (reset, clk, enable, ss, ent);
tenthsseconds i1 (reset, clk, enable & ent, ts);
endmodule

```

του οποίου μπορείτε να επαληθεύσετε την ορθή λειτουργία με τον ακόλουθο κώδικα εξομοίωσης :

```

module test();
reg reset, clk;
wire [2:0] ts;
wire [3:0] ss;

secondcounter CUT (reset, clk, 1'b1, ts, ss);

initial begin reset =0; clk = 0; #10 reset = 1; #9 reset = 0; end
always #4 clk = ~clk;
endmodule

```

#### Γ. Απεικόνιση στα Leds

Χρειαζόμαστε τέλος ένα συνδυαστικό κύκλωμα που θα παίρνει στην είσοδο τη δυαδική μέτρηση και θα βγάζει στην έξοδο εξόδους για τα 7 segments. Ένα τέτοιο κύκλωμα είναι το εξής :  
**(αρχείο 7/bin\_2\_7.v)**

```

module bin_2_7 (x, s);
input [3:0] x;
output [6:0] s;

assign s = (x == 4'd0 ) ? 7'b1111110 : (x == 4'd1 ) ? 7'b0110000 :
           (x == 4'd2 ) ? 7'b1101101 : (x == 4'd3 ) ? 7'b1111001 :
           (x == 4'd4 ) ? 7'b0110011 : (x == 4'd5 ) ? 7'b1011011 :
           (x == 4'd6 ) ? 7'b1011111 : (x == 4'd7 ) ? 7'b1110010 :
           (x == 4'd8 ) ? 7'b1111111 : (x == 4'd9 ) ? 7'b1111011 :
           (x == 4'd10) ? 7'b1110111 : (x == 4'd11) ? 7'b0011111 :
           (x == 4'd12) ? 7'b1001110 : (x == 4'd13) ? 7'b0111101 :
           (x == 4'd14) ? 7'b1001111 : 7'b1000111 ;
endmodule

```

#### Δ. Top Level Module

Το τελευταίο module που χρειαζόμαστε είναι αυτό που ενώνει όλους τους υπόλοιπους σχεδιασμούς και είναι το κορυφαίο στην ιεραρχία μας. Αυτό θα περιέχει τους διαιρέτες συχνότητας, τον μετρητή δευτερολέπτων και 2 αντίγραφα του κυκλώματος απεικόνισης. Το κύκλωμα αυτό λαμβάνει μόνο reset και ρολόι και βγάζει τις εξόδους των δύο 7 segments. Ο κώδικας συνεπώς που χρειάζεται **(αρχείο 7/tops.v)** είναι :

```

module tops (reset, clk, left, right);
input reset, clk;
output [6:0] left, right;
wire [2:0] ts;
wire [3:0] ss;

OneHertz   i0 (reset, clk, en_nxt);
secondcounter i1 (reset, clk, en_nxt, ts, ss);
bin_2_7    lt ({1'b0,ts}, left);
bin_2_7    rt (ss, right);
endmodule

```

και όλο το κύκλωμά μας μπορεί να ελεγχθεί με το ακόλουθο testbench :

```

module test();
reg reset, clk;

```

```

wire [6:0] left, right;

tops CUT (reset, clk, left, right);

initial
begin
    reset = 0; clk =0;
    #2 reset = 1;
    #2 reset = 0;
end

always #5 clk = ~clk;
endmodule

```

Πλέον έχουμε ολοκληρώσει το σχεδιασμό μας και μπορούμε να προχωρήσουμε στην υλοποίηση του.

#### E. Αρχείο φυσικών παραμέτρων

Στη συνέχεια μπορούμε να προχωρήσουμε στον ορισμό των pins που θα μας χρειαστούν. Προσοχή χρειάζεται στο ότι πρέπει να χρησιμοποιήσουμε το Excel αρχείο του XST+XSA. Για το reset θα επιλέξουμε το 1<sup>o</sup> dip switch της XSA πλακέτας και για ρολόι το pin t9 που οδηγείται από το CPLD (CLKA). Έτσι το αρχείο μας (**αρχείο 7/tops.ucf**) διαμορφώνεται στο εξής :

```

net clk loc=t9;
net reset loc=k4;
net left<6> loc=h14;
net left<5> loc=m4;
net left<4> loc=p1;
net left<3> loc=n3;
net left<2> loc=m15;
net left<1> loc=h13;
net left<0> loc=g16;
net right<6> loc=e2;
net right<5> loc=e1;
net right<4> loc=f3;
net right<3> loc=f2;
net right<2> loc=g4;
net right<1> loc=g3;
net right<0> loc=g1;

```

#### Z. Ζητούμενα

(1) Εφαρμόστε όλη τη ροή σύνθεσης και υλοποίησης του κυκλώματος όπως αυτή αναπτύχθηκε στην εργαστηριακή άσκηση 7 ώστε να καταλήξετε σε ένα λειτουργούν πρωτότυπο. Φωνάξτε τον επιβλέποντα, για να του το δείξετε.

(2) Επαυξήστε τις δυνατότητες του σχεδιασμού σας με τα ακόλουθα :

- Λειτουργία stop-watch : Η αλλαγή κάποιου διακόπτη παγώνει την ένδειξη δευτερολέπτων, αλλά το ρολόι σας συνεχίζει να μετράει κανονικά. Με την επαναφορά του διακόπτη, η ένδειξη συνεχίζει κανονικά από την πραγματική τιμή των δευτερολέπτων.
- Μέτρηση δεκάτων : Πέρα από δευτερόλεπτα το ρολόι σας είναι ικανό να μετρά και δέκατα του δευτερολέπτου. Η απεικόνιση γίνεται στα led bars δίπλα από τα 7 segment, όπου ακολουθείται η κωδικοποίηση θερμομέτρου, δηλαδή ο αριθμός των αναμένων leds αυξάνει προς κάποια διεύθυνση για κάθε δέκατο του δευτερολέπτου που περνά.

## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 8

Σκοπός της εργαστηριακής άσκησης είναι να χρησιμοποιήσουμε ένα από τα πολλά interfaces τα οποία παρέχουν οι αναπτυξιακές μας κάρτες για επικοινωνία με κάποια περιφερειακή συσκευή. Συγκεκριμένα θα χρησιμοποιήσουμε ένα πληκτρολόγιο PS/2 που θα αποτελέσει συσκευή εισόδου στο σύστημά μας. Θα πρέπει να διαβάζουμε κάποιους συγκεκριμένους χαρακτήρες και να τους απεικονίζουμε στα 7 segment. Αρχικά θα μελετήσουμε το πρωτόκολλο PS/2 για επικοινωνία με το πληκτρολόγιο.

### A. Πρωτόκολλο πληκτρολογίου

Περιγράφεται παρακάτω η ανάγνωση δεδομένων από ένα πληκτρολόγιο τεχνολογίας AT, η οποία είναι ακριβώς ίδια με αυτήν του PS/2 πρωτοκόλλου. Η περιγραφή περιορίζεται μόνο στα απαραίτητα για το project στοιχεία. Ο ενδιαφερόμενος αναγνώστης μπορεί να ανατρέξει και στα :

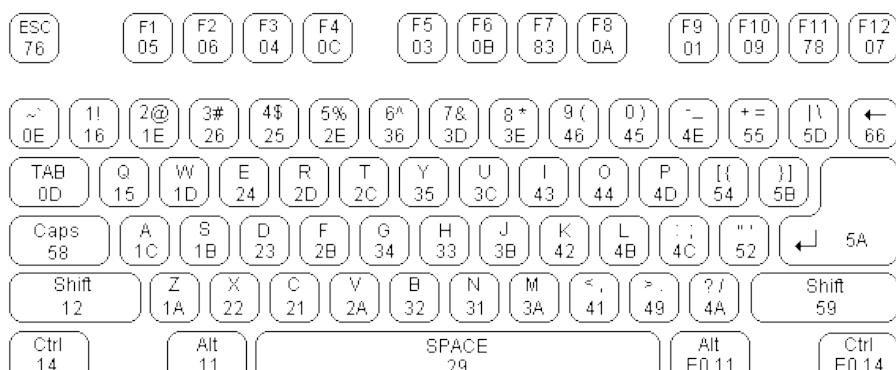
<http://www.beyondlogic.org/keyboard/keybrd.htm>

<http://www.barcodeman.com/altek/mule/scandoc.php3>

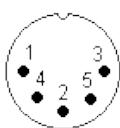
Σε κάθε πλήκτρο αντιστοιχεί ένας κωδικός 1 ή 2 ή 3 bytes που ονομάζεται κώδικας σάρωσης (scan code). Τα scan codes που μας ενδιαφέρουν στο παρόν project είναι :

Πλήκτρο	Scan Code (in HEX)
0	45
1	16
2	1E
3	26
4	25
5	2E
6	36
7	3D
8	3E
9	46

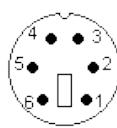
Οι κώδικες σάρωσης για ένα πληκτρολόγιο AT φαίνονται στη παρακάτω εικόνα.



Το πάτημα ενός πλήκτρου, ισοδυναμεί με τη δημιουργία και τη μεταφορά του αντίστοιχου κωδικού σάρωσης. Ο κωδικός σάρωσης θα συνεχίσει να παράγεται και να μεταδίδεται μέχρι να αφεθεί το πλήκτρο οπότε παράγεται ένας κωδικός αποτελούμενος από 2 bytes : το F0<sub>16</sub> και τον κωδικό σάρωσης. Συνεπώς ένα απλό πάτημα του "2", ισοδυναμεί με τη μεταφορά της πληροφορίας 1E F0 1E. Η μεταφορά των scan codes στο υπόλοιπο σύστημα γίνεται πάνω από μια σειριακή αρτηρία μέσω των γραμμών πληροφορίας (γραμμή KBD Data) όσο και ενός ρολογιού κωδικοποίησης (KBD Clock), μικρής συχνότητας (20 – 30 KHz). Τα σήματα που χρησιμοποιούνται σε 5 Pin DIN και PS/2 προσαρμογέις είναι όπως παρακάτω :



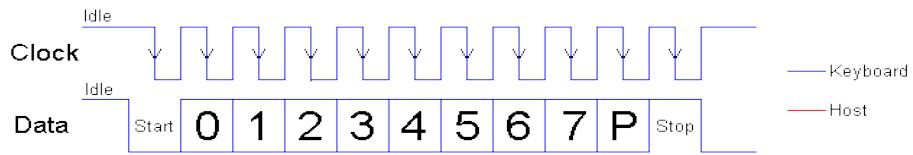
1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)



1. KBD Clock
2. GND
3. KBD Data
4. N/C
5. +5V (VCC)
6. N/C

Το σειριακό πρωτόκολλο ανταλλαγής δεδομένων στην περίπτωση του πληκτρολογίου ακολουθεί τους εξής κανόνες :

- ♦ Οταν δεν ανταλλάσσονται δεδομένα η γραμμή δεδομένων (Data) είναι ανενεργή (στο λογικό 1).
- ♦ Για την αποστολή ενός byte δεδομένων απαιτείται η αποστολή ενός πακέτου 11 bits με χρονισμό που φαίνεται στην παρακάτω εικόνα.



- Το πρώτο δυαδικό ψηφίο της γραμμής δεδομένων είναι ένα ψηφίο εκκίνησης (start bit), ακολουθούμενο από τα 8 δυαδικά ψηφία του κώδικα σάρωσης, ένα ψηφίο περιττής ισοτιμίας και ένα δυαδικό ψηφίο τερματισμού (stop bit).
- Το πρώτο δυαδικό ψηφίο πληροφορίας που αποστέλλεται είναι το λιγότερο σημαντικό (τόσο για τον κώδικα σάρωσης, όσο και για το  $F0_{16}$ ).
- Ο παραλήπτης της πληροφορίας για τη δειγματοληψία θα πρέπει να χρησιμοποιήσει την αρνητική ακμή του Clock. (Προσοχή αυτό δε σημαίνει ότι θα πρέπει στο σχεδιασμό σας να έχετε 2 ρολόγια, ένα του υπόλοιπου κυκλώματος και ένα για το πληκτρολόγιο).

## B. Υλοποίηση

Πριν τη συγγραφή του κώδικα, χρειάζεται να αναλυθούν περαιτέρω κάποια θέματα :

- Ο σχεδιασμός μας φαίνεται να έχει 2 ρολόγια, αυτό της αναπτυξιακής πλακέτας και αυτό του πληκτρολογίου. Ωστόσο, κάτι τέτοιο είναι απολύτως ανεπιθύμητο (μεγαλώνει το skew, χρησιμοποιούνται περισσότερα resources, είναι δύσκολο να προσδιοριστεί η συχνότητα λειτουργίας και είναι δύσκολος ο ακριβής συγχρονισμός μεταξύ υποσχεδιασμών με διαφορετικό χρονισμό). Επίσης, μιας και υπάρχει τάξεις μεγέθους διαφορά μεταξύ των συχνοτήτων αυτών των ρολογιών, υπάρχει περίπτωση ένα glitch στο αργό ρολόι να διαρκέσει δεκάδες κύκλους του γρήγορου. Για την αποφυγή όλων των παραπάνω, στον κώδικα μας πρέπει να έχουμε ένα μόνο ρολόι, αυτό της πλακέτας, το οποίο θα χρησιμοποιήσουμε και για δειγματοληψία του αργού (του πληκτρολογίου).
- Όπως αναλύθηκε παραπάνω, το γρήγορο πάτημα ενός χαρακτήρα, δημιουργεί την ακολουθία <scancode> <F0> <scancode>, ενώ το παρατεταμένο πάτημά του την ακολουθία <scancode><scancode> ... <scancode> <F0> <scancode>. Μια μεθοδολογία που μπορούμε να ακολουθήσουμε στο σχεδιασμό μας είναι να ανιχνεύουμε το <scancode> μόνο αμέσως το <F0>, έτσι ώστε να απεμπλακούμε από το φόρτο να κοιτάμε για πόσο πατήθηκε ένας χαρακτήρας ή να απορρίπτουμε τα 2 τελευταία σειριακά πακέτα κάθε πληκτρολόγησης.
- Το τελευταίο θέμα είναι τι γίνεται με τα υπόλοιπα πλήκτρα. Παρακάτω υποθέτουμε ότι γι' αυτά στην οθόνη μας όλα τα leds του 7 segment σβήνουν.

Μετά τα παραπάνω, ο κώδικας που χρειάζεται αρχικά να δοκιμάσουμε είναι ο :

(αρχείο 8/eight.v)

```
module kbd_protocol (reset, clk, ps2clk, ps2data, scancode);
    input      reset, clk, ps2clk, ps2data;
    output [7:0] scancode;
    reg      [7:0] scancode;
    // Synchronize ps2clk to local clock and check for falling edge;
    reg  [7:0] ps2clksamples; // Stores last 8 ps2clk samples
    always @(posedge clk or posedge reset)
        if (reset) ps2clksamples <= 8'd0;
        else ps2clksamples <= {ps2clksamples[7:0], ps2clk};
    wire fall_edge; // indicates a falling_edge at ps2clk
    assign fall_edge = (ps2clksamples[7:4] == 4'hF) & (ps2clksamples[3:0] == 4'h0);
    reg  [9:0] shift; // Stores a serial package, excluding the stop bit;
    reg  [3:0] cnt;  // Used to count the ps2data samples stored so far
    reg      f0;    // Used to indicate that f0 was encountered earlier
    /* A simple FSM is implemented here. Grab a whole package, check its parity validity and output it in
     * the scancode only if the previous read value of the package was F0 that is, we only trace when a
     * button is released, NOT when it is pressed. */
    always @(posedge clk or posedge reset)
        if (reset)
            begin
                cnt  <= 4'd0; scancode <= 8'd0; shift  <= 10'd0; f0 <= 1'b0;
            end
        else if (fall_edge)
            begin
```

```

if (cnt == 4'd10) // we just received what should be the stop bit
begin
    cnt <= 0;
    if ((shift[0] == 0) && (ps2data == 1) && (^shift[9:1]==1)) // A well received serial packet
    begin
        if (f0) // following a scancode of f0. So a key is released !
        begin
            scancode <= shift[8:1]; f0 <= 0;
            end
        else if (shift[8:1] == 8'hF0) f0 <= 1'b1;
        end // All other packets have to do with key presses and are ignored
    end
else
begin
    shift <= {ps2data, shift[9:1]}; // Shift right since LSB first is transmitted
    cnt <= cnt+1;
end
end
endmodule

module scan_2_7seg (scan, ss);
    input [7:0] scan;
    output [7:0] ss;

    assign ss =      (scan == 8'h45) ? 8'b01111110 :
                  (scan == 8'h16) ? 8'b00110000 :
                  (scan == 8'h1E) ? 8'b01101101 :
                  (scan == 8'h26) ? 8'b01111001 :
                  (scan == 8'h25) ? 8'b00110011 :
                  (scan == 8'h2E) ? 8'b01011011 :
                  (scan == 8'h36) ? 8'b01011111 :
                  (scan == 8'h3D) ? 8'b01110010 :
                  (scan == 8'h3E) ? 8'b01111111 :
                  (scan == 8'h46) ? 8'b01111011 : 8'b10000000 ;
endmodule

module eight (reset, clk, ps2clk, ps2data, left);
    input    reset, clk;
    input    ps2clk, ps2data;
    output [7:0] left;
    wire    [7:0] scan;
    kbd_protocol kbd (reset, clk, ps2clk, ps2data, scan);
    scan_2_7seg lft (scan, left);
endmodule

```

Μερικές επιπλέον διευκρινίσεις για τον παραπάνω κώδικα. Ο σχεδιασμός μας αποτελείται από 2 υποσχεδιασμούς. Ο δεύτερος (scan\_2\_7seg) απλά διαβάζει ένα scancode και το μετατρέπει σε πληροφορία απεικόνισης σε 7 segment (θα χρησιμοποιήσουμε το αριστερό 7 segment της XST).

Ο πρώτος (kbd\_protocol) υλοποιεί όλο το πρωτόκολλο ανάγνωσης από το πληκτρολόγιο. Χρησιμοποιούμε :

- τον καταχωρητή ps2clksamples για να κρατήσουμε τα τελευταία 8 δείγματα του ρολογιού του πληκτρολογίου. Παίρνουμε ένδειξη αρνητικής ακμής μόνον όταν αυτός έχει τη τιμή 11110000.
- έναν καταχωρητή ολίσθησης (shift) ο οποίος σε κάθε τέτοια αρνητική ακμή διαβάζει ένα δυαδικό ψηφίο δεδομένων.
- έναν μετρητή (cnt) για να διαπιστώσουμε πότε διαβάσαμε 11 ψηφία και εάν είναι όπως αναμενόταν (σωστά start – stop bit και σωστή ισοτιμία) τότε μόνο είναι είναι ένα σωστό σειριακό πακέτο με scancode υποψήφιο για έξοδο.

- ένα flag (f0) για να θυμόμαστε αν ο προηγούμενος scancode που ανιχνεύσαμε ήταν το f0. Μόνο σε αυτή την περίπτωση ανανεώνουμε την έξοδό μας.

### Γ. Αρχείο φυσικών παραμέτρων

Μπορούμε πλέον να προχωρήσουμε στον ορισμό των pins που θα μας χρειαστούν. Προσοχή χρειάζεται στο ότι πρέπει να χρησιμοποιήσουμε το Excel αρχείο του XST+XSA. Για το reset θα επιλέξουμε το 1<sup>o</sup> dip switch της XSA πλακέτας και για ρολό το pin t9 που οδηγείται από το CPLD (CLKA). Έτσι το αρχείο μας διαμορφώνεται στα αναγραφόμενα στο (**αρχείο 8/eight.ucf**)

### Δ. Ζητούμενα

- (1) Εφαρμόστε όλη τη ροή σύνθεσης και υλοποίησης του κυκλώματος όπως αυτή αναπτύχθηκε στην εργαστηριακή áσκηση 7 ώστε να καταλήξετε σε ένα λειτουργούν πρωτότυπο. Φωνάξτε τον επιβλέποντα, για να του το δείξετε. Κρατήστε συνεχώς ένα αριθμητικό πλήκτρο πατημένο και δείτε τι συμβαίνει. Εξηγήστε το γιατί.
- (2) Επαυξήστε τις δυνατότητες του σχεδιασμού σας με τα ακόλουθα :
  - Λειτουργία προηγούμενου πλήκτρου : Χρησιμοποιείστε και το δεύτερο 7-segment της XSA για να απεικονίζετε το προηγούμενο πατηθέν πλήκτρο.
  - Λειτουργία calculator : Δώστε τη δυνατότητα ανίχνευσης και των συμβόλων των αριθμητικών πράξεων με τους ακόλουθους scancodes (εισάγονται από το αριθμητικό πληκτρολόγιο πλην του /):

Πλήκτρο	Scan Code (in HEX)
+	79
-	7B
*	7C
/	4A

Όταν ανιχνευθεί ένας τέτοιος χαρακτήρας, εκτελείται η πράξη <πρόσφατο στοιχείο> <πράξη><προηγούμενο στοιχείο> και το αποτέλεσμα (για την πράξη \* μόνο το least significant digit, για την / μόνο το ακέραιο μέρος) εμφανίζεται στο 7 segment της XSA.