ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΕΠΙΚΟΙΩΝΙΩΝ

ΕΞΟΜΟΙΩΤΗΣ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ

Πτυχιακή εργασία των

Αμυγδαλούδη Ελευθέριου (Α.Ε.Μ.: 190)

Θεοδοσιάδη Παύλου (Α.Ε.Μ.: 317)

Επιβλέποντες: Ιωάννης Κοτζιάμπασης – Πάρις Μαστοροκώστας

ΣΕΡΡΕΣ
ΙΟΥΝΙΟΣ 2010

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ

Βεβαιώνουμε ότι είμαστε συγγραφείς αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια, την οποία είχαμε για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχουμε αναφέρει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνουμε ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμάς προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Πληροφορικής & Επικοινωνιών του Τ.Ε.Ι. Σερρών.

Αμυγδαλούδης Ελευθέριος Θεοδοσιάδης Παύλος

ΠΕΡΙΛΗΨΗ

Το σύγγραμμα αυτό περιγράφει την υλοποίηση της πτυχιακής εργασίας Εξομοιωτή Ψηφιακών Κυκλωμάτων, η οποία μάς δίνει ως αποτέλεσμα τη δημιουργία του Εξομοιωτή Ψηφιακών Κυκλωμάτων μέσω της γλώσσας προγραμματισμού C++ Builder 6. Επίσης γίνεται αναφορά στα αξιώματα και θεωρήματα που περιβάλουν τα ψηφιακά κυκλώματα ως μια επιστημονική έννοια. Τα Κεφάλαια 1, 2 και 3 αποτελούν τη γνωριμία του αναγνώστη του συγγράμματος αλλά και χρήστη του Εξομοιωτή Ψηφιακών Κυκλωμάτων με τα ψηφιακά κυκλώματα, τις θεμελιώδεις αρχές και τα θεωρήματά τους. Ειδικότερα στο Κεφάλαιο 1 γίνεται αναφορά στη δυαδική κωδικοποίηση, στο Κεφάλαιο 2 γίνεται αναφορά στην Άλγεβρα Boole και στο Κεφάλαιο 3 στην Απλοποίηση Συνδυαστικών Κυκλωμάτων και στα Σύνθετα Λογικά Κυκλώματα. Στο Κεφάλαιο 4 παρουσιάζεται και αναλύεται το περιβάλλον του προγράμματος μέσω εικόνων-αποσπασμάτων (screenshots) του προγράμματος. Τέλος στο Κεφάλαιο 5 παρουσιάζεται, αναλύεται και επεξηγήται ο κώδικας που αναπτύχθηκε, μέσω της γλώσσας προγραμματισμού C++ Builder 6, ώστε να επιτευχθεί η υλοποίηση του προγράμματος Προσομοιωτής Ψηφιακών κυκλωμάτων.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

ΚΕΦΑΛΑΙΟ 1

ΔΥΑΔΙΚΗ ΚΩΔΙΚΟΠΟΙΗΣΗ

1.1 Αριθμητικά συστήματα 9

1.1.1 Θεσιακά συστήματα 10

1.1.2 Δυαδικό σύστημα 11

1.1.3 Προσημασμένοι αριθμοί - Συμπλήρωμα ως προς 2 17

1.2 Μέθοδοι κωδικοποίησης 20

1.2.1 Κωδικοποίηση με Βάρη 22

1.2.2 Κώδικας BCD 23

1.2.3 Κωδικοποίηση χωρίς Βάρη - Κώδικας Gray 24

1.3 Κώδικες ανίχνευσης και διόρθωσης λαθών 29

1.3.1 Κώδικες ανίχνευσης λαθών 31

ΚΕΦΑΛΑΙΟ 2

ΑΛΓΕΒΡΑ BOOLE

2.1 Άλγεβρα Boole 35

2.1.1 Αρχή του Δυϊσμού 39

2.1.2 Λογική Παράσταση 39

2.1.3 Βασικά θεωρήματα 40

2.2 Δίτιμη Άλγεβρα Boole 47

2.2.1 Δίτιμη Άλγεβρα και λογισμός των προτάσεων 49

2.2.2 Δίτιμη Άλγεβρα και κυκλώματα διακοπτών (άλγεβρα των διακοπτών) 53

2.3 Λογικές Συναρτήσεις 57

2.3.1 Παράσταση Συναρτήσεων 59

2.3.2 Θεωρήματα συναρτήσεων 63

2.4 Συναρτήσεις Ελαχίστου και Μεγίστου όρου 66

2.5 Κανονική Παράσταση Συνάρτησης 69

2.6 Συναρτήσεις Δυο Μεταβλητών - Λογικές Πύλες 76

2.6.1 Συναρτήσεις δυο Μεταβλητών 77

2.6.2 Λογικές Πύλες 78

2.6.3. Ισόμορφα Συστήματα 81

2.6.4 Πράξεις Συναρτησιακώς Πλήρεις 81
2.6.5 Πύλες με Πολλαπλές Εισόδους 82
ΚΕΦΑΛΑΙΟ 3
ΑΠΛΟΠΟΙΗΣΗ ΣΥΝΔΥΑΣΤΙΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΚΑΙ ΣΤΑ ΣΥΝΘΕΤΑ ΛΟΓΙΚΑ ΚΥΚΛΩΜΑΤΑ
3.1 Μέθοδος απλοποίησης με το χάρτη Karnaugh 85
3.1.1 Χάρτης Karnaugh 86
3.1.2 Απλοποίηση με τον χάρτη Karnaugh 89
3.2 Αθροιστές 92
3.2.1 Παράλληλος Αθροιστής 94
3.2.2 Αθροιστής / Αφαιρετής 96
3.3 Σχεδίαση χωρίς Hazards 97
ΚΕΦΑΛΑΙΟ 4
ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΤΟΥ ΕΞΟΜΟΙΩΤΗ ΨΗΦΙΑΚΩΝ ΚΥΚΛΩΜΑΤΩΝ
4.1 Περιβάλλον του προγράμματος Εξομοιωτή Ψηφιακών Κυκλωμάτων
4.1.1 Μενού File και αντιστοιχία των κουμπιών 101
4.1.2 Μενού Edit 102
4.1.3 Μενού Draw και αντιστοιχία των κουμπιών 103
4.1.4 Μενού Simulation και αντιστοιχία των κουμπιών 107
4.2 Τα κουμπιά του προγράμματος Εξομοιωτή Ψηφιακών Κυκλωμάτων και οι λειτουργίες τους 108
4.2.1 Κουμπιά που αντιστοιχούν σε εντολές του Μενού 108
4.2.2 Κουμπιά ελέγχου των πηγών-εισόδων 108
4.2.3 Κουμπία Move, Erase Gate και Erase Line 109
ΚΕΦΑΛΑΙΟ 5 114
ΠΑΡΟΥΣΙΑΣΗ ΚΑΙ ΑΝΑΛΥΣΗ ΤΟΥ ΚΩΔΙΚΑ ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C++ Builder 5
ΤΕΛΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ 114
ΒΙΒΛΙΟΓΡΑΦΙΑ 165
166

ΕΙΣΑΓΩΓΗ

Στο σύγγραμμα που ακολουθεί θα παρουσιαστεί ένας Εξομοιωτής Ψηφιακών Κυκλωμάτων. Σκοπός του προγράμματος αυτού είναι να διευκολύνει τη Εξομοίωση Ψηφιακών Κυκλωμάτων, η οποία γινόταν και γίνεται με διάφορα φυσικά μέσα σε εργαστήρια ηλεκτρικών και ηλεκτρονικών κυκλωμάτων.

Τα Κεφάλαια 1, 2 και 3 αποτελούν μια πρώτη γνωριμία με τα ψηφιακά κυκλώματα, τις θεμελιώδης αρχές και θεωρήματά τους.

Συγκεκριμένα στο Κεφάλαιο 1 γίνεται αναφορά στη Δυαδική Κωδικοποίηση. Η κωδικοποίηση των κάθε είδους δεδομένων σε δυαδική μορφή, είτε αυτά είναι αριθμητικής φύσεως είτε άλλης προέλευσης, είναι απαραίτητη για τα ψηφιακά συστήματα και τους ψηφιακούς ηλεκτρονικούς υπολογιστές, γιατί τα συστήματα αυτά λειτουργούν εσωτερικά με δυαδικά σήματα. Σκοπός του κεφαλαίου αυτού είναι να μας παρουσιάσει μεθόδους κωδικοποίησης για αριθμούς απλούς ή προσημασμένους (δυαδικοί, BCD, συμπλήρωμα ως προς 2), καθώς και για μη αριθμητικές εφαρμογές (πίνακες, κώδικες με και χωρίς βάρη, Gray). Αναλύεται, επίσης, το σημαντικότατο θέμα της προστασίας των δεδομένων από τις κακές συνέπειες του θορύβου και παρουσιάζονται στοιχειώδεις μέθοδοι κωδικοποίησης για την ανίχνευση (ψηφίο ισοτιμίας) και τη διόρθωση (κώδικας Hamming) αλλοιώσεων εξ αιτίας του θορύβου.

Στο Κεφάλαιο 2 γίνεται αναφορά στην Άλγεβρα Boole. Η επιτυχία των ψηφιακών υπολογιστικών συστημάτων στηρίζεται αφενός μεν στην ευκολία της κατασκευής τους και αφετέρου στον απλό και συστηματικό τρόπο σχεδίασής τους. Η εύκολη σχεδίαση πολύπλοκων ψηφιακών συστημάτων οφείλεται κατ' αρχήν στη δυνατότητα που έχουμε να παραστήσουμε τις επιθυμητές λειτουργίες τους με συμβολικό τρόπο. Η δυνατότητα αυτή προήλθε από την αντιστοιχία που έχουν τα βασικά στοιχεία ενός ψηφιακού κυκλώματος, οι Λογικές Πύλες, με τις μαθηματικές - λογικές πράξεις ενός αλγεβρικού συστήματος που ονομάζεται "άλγεβρα Boole". Οι θεωρητικές γνώσεις αυτής της άλγεβρας μας επιτρέπουν να σχεδιάζουμε τα κυκλώματά μας γρήγορα, χωρίς λάθη και με οικονομία. Για να μάθουμε, όμως, τους κανόνες της σχεδίασής τους, που ονομάζεται Ψηφιακή Σχεδίαση, πρέπει πρώτα να κατανοήσουμε τις πράξεις και τις ιδιότητες της άλγεβρας Boole. Αρχικά, η μελέτη και σχεδίαση των Ψηφιακών Συστημάτων γινόταν με εμπειρικό τρόπο, θεωρώντας τα

συστήματα αυτά σαν απλά ηλεκτρικά κυκλώματα. Το 1938 ο C. Shannon εισήγαγε το "Λογισμό των Προτάσεων" σαν εργαλείο για τη συμβολική περιγραφή της λειτουργίας τους. Ο Λογισμός των Προτάσεων με τη σειρά του προέρχεται από την άλγεβρα Boole. Η άλγεβρα Boole, τα θεωρήματά της και οι ειδικές εφαρμογές της έχουν καταστεί πλέον το μαθηματικό εργαλείο για την περιγραφή και σχεδίαση των Ψηφιακών Συστημάτων, όταν αναφερόμαστε στο επίπεδο της "λογικής" συμπεριφοράς τους.

Στο Κεφάλαιο αυτό θα παρουσιαστούν τα κυριότερα σημεία της άλγεβρας Boole, οι εφαρμογές της οι σχετικές με τη Λογική (Λογισμός των Προτάσεων) και τα Ψηφιακά Συστήματα (Άλγεβρα Διακοπτών) καθώς και η υλοποίηση των πράξεων της άλγεβρας αυτής με τις Λογικές Πύλες.

Στο Κεφάλαιο 3 γίνεται αναφορά στην Απλοποίηση Συνδυαστικών Κυκλωμάτων και στα Σύνθετα Λογικά Κυκλώματα. Η πολυπλοκότητα του ψηφιακού κυκλώματος που υλοποιεί μια λογική συνάρτηση συνδέεται άμεσα με την πολυπλοκότητα της αλγεβρικής παράστασης της συνάρτησής του. Μια συνάρτηση μπορεί να απεικονίζεται από διαφορετικές παραστάσεις και τα θεωρήματα της άλγεβρας Boole μας επιτρέπουν να μετασχηματίζουμε την μια παράσταση σε μια άλλη ισοδύναμή της, που να είναι π.χ. απλούστερη και επομένως να παράγει ένα οικονομικότερο κύκλωμα. Μια τεχνική απλοποίησης όμως που στηρίζεται μόνο στα θεωρήματα είναι όχι μόνο ανεπαρκής για μεγάλες παραστάσεις αλλά και παραπλανητική, μπορεί να μας παρασύρει σε φαινομενικά μόνο ελάχιστες παραστάσεις. Για τους λόγους αυτούς προτιμούνται συστηματικοί αλγόριθμοι, οι οποίοι εγγυώνται ότι σε κάθε περίπτωση θα μας οδηγήσουν στην εύρεση της ελάχιστης δυνατής παράστασης. Η ελάχιστη μορφή ενός ψηφιακού κυκλώματος είναι μία από τις πολλές απαιτήσεις βελτιστοποίησης που μπορεί να θέσει ο σχεδιαστής. Στο Κεφάλαιο αυτό θα ασχοληθούμε με τη βελτιστοποίηση ως προς την ελάχιστη μορφή ενός ψηφιακού κυκλώματος δύο επιπέδων. Θα παρουσιαστεί ο χάρτης Karnaugh, που η πιστή τήρησή του εγγυάται ότι θα δώσουν πάντοτε το ελάχιστο κύκλωμα. Μια παρενέργεια της ελαχιστοποίησης είναι τα hazards που εξετάζονται στο τέλος του Κεφαλαίου.

Στο Κεφάλαιο 4 παρουσιάζεται και αναλύεται το περιβάλλον του προγράμματος. Γίνεται γνωριμία με το περιβάλλον του προγράμματος το Μενού του και τις δυνατές επιλογές - εντολές που έχει στη δυνατότητα του ο χρήστης μέσω αυτού. Έχουν χρησιμοποιηθεί εικόνες - αποσπάσματα (screenshots) έτσι ώστε η γραπτή επεξήγηση να αποκτά εικόνα και να γίνεται ευκολότερη η περιήγηση και η χρήση του προγράμματος. Το

Κεφάλαιο αυτό μπορεί να θεωρηθεί ως οι οδηγίες χρήσεως του Εξομοιωτή Ψηφιακών Κυκλωμάτων.

Τέλος στο Κεφάλαιο 5 παρουσιάζεται και αναλύεται ο κώδικας που συγγράφηκε, μέσω της γλώσσας προγραμματισμού C++ Builder 6. Σε κάθε εντολή, συνθήκη ή συνάρτηση που εμφανίζεται στον κώδικα του προγράμματος προηγείται μια αναλυτική επεξήγηση με σκοπό ο αναγνώστης του συγγράματος και χρήστης του προγράμματος να καταννοήσει και προγραμματιστηκά τον τρόπο που λειτουργεί ο Εξομοιωτής Ψηφιακών Κυκλωμάτων.

1.1 Αριθμητικά συστήματα

ΚΕΦΑΛΑΙΟ 1

Αν και το δεκαδικό αριθμητικό σύστημα είναι γνωστό σε όλους, ίσως ορισμένοι να μη γνωρίζουν ότι είναι μία μορφή κώδικα με βάρη. Θα εξηγήσουμε την έννοια και τη δομή των *θεσιακών συστημάτων*, όπως είναι το γνωστό μας δεκαδικό σύστημα, και μέσω αυτής τη δομή του *δυαδικού* συστήματος αριθμών, καθώς και άλλων παρομοίων συστημάτων.

Από την αρχαιότητα έχουν αναπτυχθεί πολλά αριθμητικά συστήματα, όπως το εξηνταδικό (Βαβυλώνιοι), το Λατινικό κ.λπ., με πιο διαδεδομένα όσα στηρίζονται στη δεκαδική αρίθμηση. Η μεγάλη διάδοση των δεκαδικών αριθμών συνδέεται ιστορικώς με την ευκολία με την οποία οι απλοί άνθρωποι έκαναν πράξεις με τα δάχτυλά τους. Εξάλλου η αγγλική λέξη "digit" προέρχεται από τη Λατινική λέξη "digitus" που σημαίνει "δάκτυλο".

Το δεκαδικό αριθμητικό σύστημα, που χρησιμοποιούμε σήμερα, είναι Ινδουιστικής προέλευσης (700 μ.Χ.). Οι Ινδουιστές με την έξυπνη χρήση ενός ιδιόρρυθμου συμβόλου, του *μηδενός*, εκμεταλλεύτηκαν τη *θέση* των ψηφίων μέσα στον αριθμό για να δηλώσουν την αξία (βάρος) των ψηφίων τους (αρχή του *θεσιακού συστήματος*). Έτσι, απέφυγαν το πρόβλημα που είχε το Ελληνικό και το Λατινικό σύστημα να χρειάζονται νέα σύμβολα, καθώς μεγαλώνει η τάξη του αριθμού. Επιπλέον με την απλοποίηση που επέφεραν στην κωδικοποίηση, απλοποιήθηκαν και οι αριθμητικές πράξεις. Για ιστορικούς λόγους αναφέρουμε εδώ ότι τον 2ο αιώνα μ.Χ. ο Έλληνας αστρονόμος Πτολεμαίος χρησιμοποιούσε στους αστρονομικούς πίνακες του το “Ο”, από τη λέξη *ουδέν*, για τον ίδιο σκοπό. Η καινοτομία των Ινδών μεταφέρθηκε στους Άραβες. Από αυτούς διαδόθηκε στην Ευρώπη από το μαθηματικό Λεονάρδο της Πίζας, γνωστότερο ως Fibonacci (“Κεφάλας”), ο οποίος με το βιβλίο του Liber Abaci (1202 μ.Χ.) έβγαλε την Ευρώπη

από το μαθηματικό μεσαίωνα των λατινικών αριθμών. Παρά την επιτυχία του δεκαδικού συστήματος, η κατασκευή ψηφιακών υπολογιστικών μηχανών με δυαδικά σήματα δημιούργησε την ανάγκη χρήσης, στο εσωτερικό τους, αριθμητικών συστημάτων που ταιριάζουν καλύτερα προς την τεχνολογία τους. Σήμερα οι ψηφιακοί ηλεκτρονικοί υπολογιστές, εκτός ειδικών περιπτώσεων, χρησιμοποιούν το ονομαζόμενο δυαδικό σύστημα.

1.1.1 Θεσιακά συστήματα

Αν αναλύσουμε το δεκαδικό αριθμητικό σύστημα, θα παρατηρήσουμε ότι είναι ένας τρόπος κωδικοποίησης που στηρίζεται στη μέθοδο των βαρών. Για παράδειγμα όταν γράφουμε τον αριθμό 1908, εννοούμε στην πραγματικότητα πλήθος μονάδων ίσο με:

$$1*10^3+9*10^2*+0*10^1+8*10^0$$

Εδώ τα βάρη είναι οι αριθμοί 103, 102, 101 και 100, δηλαδή τα βάρη σχηματίζονται από διαδοχικές δυνάμεις του 10, το οποίο ονομάζεται **βάση** του συστήματος, και το “0” είναι απαραίτητο για να διατηρούνται τα ψηφία στη “σωστή” θέση τους. Γενικά, σε ένα θεσιακό σύστημα ένας θετικός αριθμός N που παριστάνεται από μία ακολουθία ψηφίων:

$$\alpha n\ \alpha n{-}1 \ldots \alpha 0\ .\ \alpha{-}1 \ldots \alpha{-}m$$

αντιπροσωπεύει πλήθος μονάδων:

$$N = \sum_{i=-m}^{n} a_i b^i$$

όπου:

β είναι η **βάση** του αριθμού (ακέραιος, β>1)

αi είναι τα **ψηφία** του αριθμού (0 ≤ αi < β)

i είναι η **τάξη** του ψηφίου

n+1 είναι το πλήθος των ακεραίων ψηφίων

m είναι το πλήθος των κλασματικών ψηφίων.

Η ακολουθία των ψηφίων α_n α_{n-1} … α_0 αποτελεί το *ακέραιο* μέρος του αριθμού, ενώ η ακολουθία α_{-1} α_{-2} … α_{-m} το *κλασματικό* μέρος του. Το αn ονομάζεται το **Πλέον Σημαντικό Ψηφίο** (Most Significant Digit), για συντομία ΠΣΨ (MSD), ενώ το α–m είναι το **Ελάχιστα Σημαντικό Ψηφίο** (Least Significant Digit), για συντομία ΕΣΨ (LSD). Η μικρότερη βάση, που έχει νόημα, είναι ο αριθμός 2. Όταν η βάση είναι β=2 το σύστημα ονομάζεται **δυαδικό**, όταν β=3 **τριαδικό**, όταν β=8 **οκταδικό**, όταν β=10 **δεκαδικό**, όταν β=16 **δεκαεξαδικό** κ.λπ. Όταν είναι β>10 χρειάζονται πρόσθετα σύμβολα για την παράσταση των επιπλέον ψηφίων. Για τα ψηφία 10 έως το 15 του *δεκαεξαδικού* συστήματος χρησιμοποιούνται διεθνώς τα πρώτα γράμματα του λατινικού αλφαβήτου:

Δεκαδικό: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14, 15

Δεκαεξαδικό: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Παράδειγμα 1.1

Ο δεκαεξαδικός αριθμός 1ABC αντιστοιχεί στο δεκαδικό σύστημα με τον αριθμό:

163*1+162*(10)+161*(11)+160*(12) => 684410

Για να δηλωθεί, όπου χρειάζεται, ότι η βάση κάποιου αριθμού **N** είναι η **β,** γράφουμε Nβ π.χ. 148.510 = 10010100.12 .

1.1.2 Δυαδικό σύστημα

Το δυαδικό σύστημα έχει μόνο δύο ψηφία, που συμβολίζονται με τους χαρακτήρες 0 και 1. Χρησιμοποιείται στα ψηφιακά συστήματα λόγω της άμεσης αναπαράστασης των ψηφίων του από δυαδικά σήματα. Τα ψηφία αυτά, όπως και οι καταστάσεις του δυαδικού σήματος, ονομάζονται διεθνώς **bits**. Η μεγαλύτερη

αριθμητική τιμή που μπορεί να παρασταθεί από ένα δυαδικό αριθμό μήκους n ψηφίων (bits) είναι $2^{η}$-1 γιατί

$$1*2^{η-1} + 1*2^{η-2} + \ldots + 1*2^{!} + 1*2^{\circ} = 2^{η}-1$$

Κατά συνέπεια, εάν n είναι το πλήθος των ψηφίων ενός δυαδικού αριθμού που αντιστοιχεί σ' ένα δεκαδικό αριθμό με d ψηφία, τότε:

$$2^{η}-1 > 10^{d}-l \quad \text{και} \quad η = d/\log^{10}2 = d/0.3$$

δηλ. η παράσταση του δυαδικού θα έχει περίπου τριπλάσια ψηφία από αυτή του δεκαδικού. Οι αριθμητικές πράξεις στο δυαδικό σύστημα είναι πολύ απλούστερες απ' ότι στα άλλα συστήματα.

Ο Πίνακας 1.1 περιέχει τις βασικές πράξεις για τους μονοψήφιους δυαδικούς αριθμούς (αριθμοί του ενός bit):

Πίνακας 1.1 Πράξεις μονοψήφιων δυαδικών αριθμών.

Bits		α+β		α–β		α*β	α/β
α	β	άθροισμ	κρατούμ.	διαφ.	κρατούμ.		
0	0	0	0	0	0	0	–
0	1	1	0	1	1	0	0
1	0	1	0	1	0	0	–
1	1	1	1	0	0	1	1

Ο ρόλος του κρατούμενου εξηγείται καλύτερα, αν λάβουμε υπ' όψιν τη θεσιακή δομή των δυαδικών αριθμών (βάση 2). Στο Παράδειγμα 1.2 εξηγείται ο ρόλος του κρατουμένου στην πρόσθεση.

Παράδειγμα 1.2

Έστω N το *άθροισμα* των αριθμών 1011 και 111:

$$
\begin{aligned}
N &= (2^3*1+2^2*0+2^1*1+2^0*1) + (2^2*1+2^1*1+2^0*1) \\
&= 2^3*1 \quad +2^2*(0+1) \quad +2^1*(1+1) \quad +2^0*(1+1) \\
&= 2^3*1 \quad +2^2*1 \quad +2^1*2 \quad +2^0*2 \\
&= 2^3*1 \quad +2^2*1 \quad +2^1*(2+1) \\
&= 2^3*1 \quad +2^2*(1+1) \quad +2^1*1 \\
&= 2^3*1 \quad +2^2*2 \quad +2^1*1 \\
&= 2^3*(1+1) \quad +2^1*1 \\
&= 2^3*2 \quad +2^1 \\
&= 2^4 \quad +2^1 \\
&= 2^4*1 \quad +2^3*0 \quad +2^2*0 \quad +2^1*1 \quad +2^0*0
\end{aligned}
$$

Συνεπώς τα ψηφία του N είναι 100102.

Παρατηρούμε στο Παράδειγμα 1.2 ότι κάθε φορά που τα ψηφία μιας στήλης συμπληρώνουν άθροισμα ίσο με τη βάση, δηλ. 2, προστίθεται μία μονάδα (κρατούμενο) στα ψηφία της στήλης που κατέχει την αμέσως μεγαλύτερη τάξη. Για την αφαίρεση ισχύει η ίδια μεθοδολογία, όπως φαίνεται στο παράδειγμα 1.3

Παράδειγμα 1.3

Έστω N η *διαφορά* των αριθμών 1011 και 111:

$$
\begin{aligned}
N &= (2^3*1+2^2*0+2^1*1+2^0*1) - (2^2*1+2^1*1+2^0*1) \\
&= 2^3*1 \quad +2^2*(0-1) \quad +2^1*(1-1) \quad +2^0*(1-1) \\
&= 2^3*1 \quad +2^2*(-1) \quad +2^1*0 \quad +2^0*0 \\
&= 2^3*1 \quad +2^2*(2-1-2) \\
&= 2^3*1 \quad +2^2*(1-2) \\
&= 2^3*(1-1) \quad +2^2*1 \\
&= 2^3*(0) \quad +2^2*1 \\
&= 2^3*0 \quad +2^2*1 \quad +2^1*0 \quad +2^0*0
\end{aligned}
$$

Επομένως τα ψηφία της διαφοράς είναι 01002.

Στην αφαίρεση, όπου χρειάζεται, δανειζόμαστε ποσό ίσο με τη βάση, δηλ. 2, και κατόπιν μεταφέρουμε αρνητικό κρατούμενο στην αμέσως μεγαλύτερης τάξης στήλη. Συμπερασματικά, *οι πράξεις στο δυαδικό σύστημα γίνονται όπως στο δεκαδικό με τη διαφορά ότι κρατούμενο δημιουργείται όταν συμπληρώνεται άθροισμα ίσο με τη βάση, δηλ. 2, ή όταν στην αφαίρεση δανειζόμαστε ποσό ίσο με τη βάση*:

Κρατούμενο		1	1	1	
		1	0	1	1
	+		1	1	1
Αθροισμα:	1	0	0	1	0

Κρατούμενο		−1	−1	−1	
		1	0	1	1
	−		1	1	1
Διαφορά:		0	0	1	0

Στο δυαδικό σύστημα οι πράξεις του πολλαπλασιασμού και της διαίρεσης είναι πολύ απλές και ανάγονται σε απλές διαδοχικές *προσθέσεις* για τον πολλαπλασιασμό και σε *αφαιρέσεις* για τη διαίρεση:

Πολλαπλασιασμός:

1100	
101	
1100	γιατί 1x1100
0000	μηδέν ή παραλείπεται
1100	
111100	Γινόμενο

Διαίρεση:

111100	1100	
1100	101	πηλίκο
0011		
110		κατεβ. το επόμενο ψηφίο
1100		μέχρις ότου > διαιρέτη.
1100		αφαιρ. διαιρέτη.
0000		υπόλοιπο.

Παράδειγμα 1.4

Οι τέσσερις πράξεις στο δεκαδικό και στο δυαδικό σύστημα:

Πρόσθεση:

$$\begin{array}{r} 9_{10} \\ +\,5_{10} \\ \hline 14_{10} \end{array} \qquad \begin{array}{r} 1001_2 \\ +\,0101_2 \\ \hline 1110_2 \end{array} \qquad \begin{array}{r} 15.25_{10} \\ +\,7.50_{10} \\ \hline 22.75_{10} \end{array} \qquad \begin{array}{r} 1111.01_2 \\ +\,111.10_2 \\ \hline 10110.11_2 \end{array}$$

Αφαίρεση:

$$\begin{array}{r} 25_{10} \\ -\,7_{10} \\ \hline 18_{10} \end{array} \qquad \begin{array}{r} 11001_2 \\ -\,00111_2 \\ \hline 10010_2 \end{array} \qquad \begin{array}{r} 18.75_{10} \\ -\,12.50_{10} \\ \hline 6.25_{10} \end{array} \qquad \begin{array}{r} 10010.11_2 \\ -\,1100.10_2 \\ \hline 00110.01_2 \end{array}$$

Πολλαπλασιασμός:

$$\begin{array}{r} 9_{10} \\ \times\,5_{10} \\ \hline 45_{10} \end{array} \qquad \begin{array}{r} 1001_2 \\ \times\,101_2 \\ \hline 1001_2 \\ 0000 \\ 1001 \\ \hline 101101_2 \end{array} \qquad \begin{array}{r} 25.5_{10} \\ \times\,6.5_{10} \\ \hline 1275 \\ 1530 \\ \hline 165.75_{10} \end{array} \qquad \begin{array}{r} 11001.1_2 \\ \times\,110.1_2 \\ \hline 110011 \\ 000000 \\ 110011 \\ 110011 \\ \hline 10100101.11_2 \end{array}$$

Διαίρεση:

$$\begin{array}{r|l} 46 & 9 \\ \hline 1 & 5 \end{array} \qquad \begin{array}{r|l} 101110 & 1001 \\ \hline -\,1001 & 101_2 \\ \hline 00101 & \\ 001010 & \\ -\,1001 & \\ \hline \text{υπόλοιπο}\quad 0001 & \end{array} \qquad \begin{array}{r|l} 2925 & 27 \\ \hline 225 & 108 \\ \hline 09 & \end{array} \qquad \begin{array}{r|l} 101101101101_2 & 11011_2 \\ \hline -\,11011 & 01101100 \\ \hline 0100101 & \\ -\,11011 & \\ \hline 000101001 & \\ -\,11011 & \\ \hline 0011101 & \\ -\,11011 & \\ \hline \text{υπόλοιπο}\quad 0001001 & \end{array}$$

Η μετατροπή ενός δυαδικού αριθμού σε δεκαδική μορφή είναι πολύ απλή: αρκεί να γράψουμε τον αριθμό με την πλήρη του μορφή και να εκτελέσουμε τις πράξεις.

Παράδειγμα 1.5

Μετατροπή του δυαδικού αριθμού 1101 σε δεκαδική μορφή:

$1101_2 \Rightarrow 2^3*1+2^2*1+2^1*0+2^{\circ}*1 = 8+4+0+1 \Rightarrow 13_{\mathbf{10}}$

Η μετατροπή ενός δεκαδικού σε δυαδικό γίνεται με διαδοχικές διαιρέσεις του αριθμού με το 2. Από κάθε διαίρεση προκύπτει ένα πηλίκο Π και ένα ακέραιο υπόλοιπο Υ. Η μέθοδος είναι η εξής:

α) Θέτουμε αρχικά το Π0 να ισούται με τον υπό μετατροπή αριθμό στη δεκαδική του μορφή και, επίσης, τον δείκτη k=0.

β) Διαιρούμε το Πk με το 2 και θέτουμε Υk το υπόλοιπο και Πk+1 το νέο πηλίκο. Το Υk είναι το ζητούμενο δυαδικό ψηφίο τάξεως k.

γ) Θέτουμε k=k+1 και επαναλαμβάνουμε από την (β) μέχρις ότου Πk =0.

Παράδειγμα 1.6

Μετατροπή του αριθμού 4510 σε δυαδικό:

Θέτουμε Π0 = 45. Από τις διαδοχικές διαιρέσεις με το 2 προκύπτουν:

k=0 => (Υ0=1, Π1=22), k=1 => (Υ1=0, Π2=11),

k=2 => (Υ2=1, Π3=5), k=3 => (Υ3=1, Π4=2),

k=4 => (Υ4=0, Π5=1), k=5 => (Υ5=1, Π6=0)

Η ζητούμενη μετατροπή είναι: 4510 => 1011012.

1.1.3 Προσημασμένοι αριθμοί – Συμπλήρωμα ως προς 2

Στο δυαδικό σύστημα, για να διακρίνουμε τους αριθμούς σε θετικούς και αρνητικούς, χρησιμοποιούμε σαν πρόσημα τα ψηφία 0 και 1. Κατά σύμβαση ένας αριθμός είναι **θετικός** όταν το πρώτο bit από αριστερά είναι **0** και **αρνητικός** όταν το bit αυτό είναι **1**, π.χ. ο αριθμός 01011 είναι θετικός και ο 11010 είναι αρνητικός. Επειδή η χρησιμοποίηση των 0 και 1 ταυτόχρονα σαν προσήμων και σαν ψηφίων του αριθμού δημιουργεί σύγχυση, είναι απαραίτητο, όταν αναφερόμαστε σε προσημασμένους αριθμούς, να αναφέρουμε και το πλήθος των ψηφίων τους. Για παράδειγμα ο αριθμός 11010 είναι αρνητικός όταν θεωρηθεί ότι έχει 5 ψηφία και θετικός, όταν έχει 6 ψηφία δηλ. γραφεί ως 011010.

Τα ηλεκτρονικά συστήματα που κάνουν πράξεις με προσημασμένους αριθμούς μπορούν να απλοποιηθούν εάν όχι μόνο το πρόσημο αλλά και η όλη παράσταση των προσημασμένων αριθμών εκλεγεί κατάλληλα. Ένα τέτοιο σύστημα είναι το ονομαζόμενο *Συμπλήρωμα ως προς 2*. Στο δυαδικό σύστημα ορίζεται σαν **Συμπλήρωμα ως προς 2** (2's Complement) ενός αριθμού N με n πλήθος ψηφίων ο αριθμός:

$$N' = 2^{\eta} - N$$

Ο N' ονομάζεται *συμπληρωματικός* του N και παίζει το ρόλο του αντιθέτου του N. Οι αριθμοί με Πλέον Σημαντικό Ψηφίο 0 θεωρούνται **θετικοί** και αυτοί με Πλέον Σημαντικό Ψηφίο 1 θεωρούνται **αρνητικοί**. Η περιοχή των αριθμών που καλύπτεται από ένα αριθμό με n bits στο συμπλήρωμα ως προς 2 είναι:

$$-2^{\eta-1} \leq N < 2^{\eta-1}$$

Το πλεονέκτημα αυτής της κωδικοποίησης είναι ότι η πράξη της *αφαίρεσης* δύο αριθμών A και B ανάγεται σε πρόσθεση του συμπληρωματικού του B, όπου το νέο πρόσημο προκύπτει από την πρόσθεση των προσήμων σαν να ήταν αριθμοί:

$$A - B = A + (2^{\eta} - B) - 2^{\eta} = A + B' - 2^{\eta} \Rightarrow A + B'$$

Ο αριθμός $-2^{η}$ που περισσεύει δεν αλλοιώνει το αποτέλεσμα. Η μέθοδος μπορεί να εξηγηθεί απλά με το παράδειγμα του σχήματος 1.1: Στο παράδειγμα χρησιμοποιούνται αριθμοί των τριών bit, συνεπώς n=3 και $2^{η}$=8. Το σχήμα 1.1 παριστά έναν *κυκλικό* πίνακα αντιστοιχίας, στον οποίο οι προσημασμένοι αριθμοί (εσωτερικό κύκλου) αντιστοιχούνται στους μη προσημασμένους αριθμούς (έξω από τον κύκλο). Στο σύστημα αυτό *σαν άθροισμα δύο προσημασμένων αριθμών λαμβάνεται ο προσημασμένος αριθμός που αντιστοιχεί στο άθροισμα των αντιστοίχων τους μη προσημασμένων*.

Παράδειγμα 1.7

Πρόσθεση στο Συμπλήρωμα ως προς 2:

α) (–1)+(–2): Ξεκινάμε από το μη προσημασμένο αριθμό 7 και κινούμαστε 6 θέσεις προς την *θετική* φορά ("+" στο σχήμα 1.1), καταλήγοντας στο μη προσημασμένο αριθμό 5, που αντιστοιχεί στον –3. Ισοδυναμεί με τη διαδικασία:

(–1)+(–2) => 6 + 7 = 13 => 13 – 8 = 5 => –3

β) (+1)+(–2): Ξεκινάμε από το μη προσημασμένο αριθμό 1 και κινούμαστε 6 θέσεις προς την θετική φορά, καταλήγοντας στο μη προσημασμένο αριθμό 7, που αντιστοιχεί στον –1. Ισοδυναμεί με την διαδικασία:

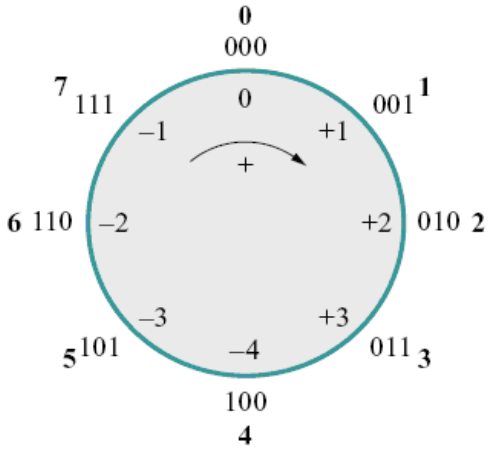
(+1)+(–2) => 1 + 6 = 7 => –1

γ) (–1)+(+2): ξεκινάμε από το μη προσημασμένο αριθμό 7 και κινούμεθα 2 θέσεις προς την θετική φορά, καταλήγοντας στο μη προσημασμένο αριθμό 1, που αντιστοιχεί στον +1. Ισοδυναμεί με τη διαδικασία:

(–1)+(+2) => 7 + 2 = 9 => 9 – 8 = 1 => +1

Παράδειγμα 1.8

Αφαίρεση στο Συμπλήρωμα ως προς 2 (βλέπε σχήμα 1.1):



Σχήμα 1.1 Σχηματική Διάταξη Αριθμών στο Συμπλήρωμα ως προς 2

α) (–1)–(–2): Πρώτα βρίσκουμε τον *αντίθετο* του (–2) που είναι 8 – 6 = 2 =>

(+2). Κατόπιν εκτελούμε την πρόσθεση (–1)+(+2) =+1.

β) (+1)–(+2): Βρίσκουμε τον αντίθετο του (+2) που είναι 8 – 2 = 6 => (–2).

Εκτελούμε την πρόσθεση (+1)+(–2) =–1.

Το πρόβλημα, λοιπόν, της αφαίρεσης μετατοπίζεται στην εύρεση του αντιθέτου του αριθμού. Η *εύρεση του αντιθέτου* ενός αριθμού στο συμπλήρωμα ως προς 2 γίνεται πολύ εύκολα στο δυαδικό σύστημα με τον εξής τρόπο:

α. αντιστρέφουμε όλα τα bit του αριθμού, δηλ. όπου 0 —>1 και 1 —> 0, και

β. στο αποτέλεσμα προσθέτουμε τη μονάδα.

Παράδειγμα 1.9

Αφαίρεση με υπολογισμό του αντιθέτου:

Έστω η πράξη (+1)–(+2). Ο αντίθετος του (+2) σε δυαδική μορφή είναι:

(+2) => 010 αντιστροφή bit 101

πρόσθεση 1 + 1

110 => (-2)

Η όλη διαδικασία της αφαίρεσης συντομεύεται, εάν γίνει κατευθείαν η πρόσθεση των αριθμών:

(+1) => 001

(- 2) => 101 αντιστροφή bit

+ 1 πρόσθεση 1

111 => (-1)

Παράδειγμα 1.10

Αφαίρεση με υπολογισμό του αντιθέτου:

Έστω η αφαίρεση 46 – 17 στο συμπλήρωμα ως προς 2 με δυαδικούς αριθμούς των 8 bit. Είναι 4610 => 001011102 και 1710 => 000100012:

(+46) => 00101110

(- 17) => 11101110

+ ____ 1

00011101 => (+ 29)

1.2 Μέθοδοι κωδικοποίησης

Οι μέθοδοι που χρησιμοποιούνται για την κωδικοποίηση χωρίζονται σε δύο μεγάλες κατηγορίες: σ’ αυτές που χρησιμοποιούν **πίνακες αντιστοιχίας** και σ’ αυτές που χρησιμοποιούν **κανόνες**. Η επιλογή μιας μεθόδου είναι, καταρχήν, αυθαίρετη,

τα δε κριτήρια επιλογής καθορίζονται από το πρακτικό αποτέλεσμα που επιδιώκει κανείς, π.χ. τη δημιουργία λέξεων που να είναι εύκολες στην απομνημόνευση (μνημονικά ονόματα), στην αποθήκευση (μέσο αποθήκευσης μαγνητικό ή χαρτί), την επεξεργασία (π.χ. το Ινδουιστικό σύστημα αριθμών για τις αριθμητικές πράξεις), τη μεταφορά κ.λπ. Είναι, επίσης, πολύ συνηθισμένο η ίδια πληροφορία να κωδικοποιείται κατά διαφόρους τρόπους, δηλ. να έχει πολλές παραστάσεις, ανάλογα με την εξυπηρέτηση που προσφέρει η κάθε παράσταση.

Ο *πίνακας αντιστοιχίας* είναι η απλούστερη μέθοδος κωδικοποίησης. Παράδειγμα ο Τηλεφωνικός Κατάλογος, όπου αντιστοιχείται το όνομα του συνδρομητή με τον αριθμό του τηλεφώνου του.

Στους υπολογιστές χρησιμοποιείται ευρύτατα για τη δυαδική παράσταση αλφαριθμητικών πληροφοριών ένας διεθνώς καθιερωμένος πίνακας αντιστοιχίας, ο ονομαζόμενος **πίνακας ASCII**. Στην τυπική του μορφή, χρησιμοποιεί 7 bits, ήτοι 128 λέξεις, για να κωδικοποιήσει σε δυαδική μορφή:

α) *αλφαριθμητικούς* χαρακτήρες, δηλ.:

τα γράμματα του λατινικού αλφαβήτου μικρά και κεφαλαία, τα σημεία στίξεως, τα αριθμητικά ψηφία 0 … 9, και

β) χαρακτήρες *ελέγχου,* δηλ. λέξεις που χρησιμοποιούνται για το συντονισμό της επικοινωνίας δύο συσκευών (αρχή μηνύματος, τέλος, stop κ.λπ.).

Οι *μέθοδοι κωδικοποίησης με κανόνες* βασίζονται στις ιδιότητες των πληροφοριών που θέλουμε να κωδικοποιήσουμε, π.χ. εάν αποτελούν ένα διατεταγμένο σύνολο κ.λπ. Έχουν το πλεονέκτημα ότι δεν απαιτούν μεγάλο χώρο αποθήκευσης, όπως οι πίνακες, και το μειονέκτημα ότι η κωδικοποίηση μ’ αυτούς δεν είναι τόσο απλή. Οι μέθοδοι με κανόνες χωρίζονται σε δύο κατηγορίες: στις μεθόδους κωδικοποίησης με **βάρη** και στις μεθόδους **χωρίς βάρη**.

1.2.1 Κωδικοποίηση με Βάρη

Χρησιμοποιούνται, κατά κανόνα, για την κωδικοποίηση πληροφοριών που αποτελούν ένα διατεταγμένο σύνολο ή μπορούν να αντιστοιχηθούν με ακέραιους αριθμούς, π.χ. ό,τι παριστάνεται με κάποιον αύξοντα αριθμό ή αριθμό μητρώου κ.λπ. Η μέθοδος είναι η εξής: Εάν σ είναι ο αντίστοιχος ακέραιος αριθμός που παριστάνει την πληροφορία στην αρχική της μορφή τότε στη νέα μορφή της συμβολίζεται με την ακολουθία συμβόλων:

αnαn–1**…**…α1α0

όπου τα αi (i=0, …, n) παίρνουν τιμές από ένα σύνολο ακέραιων αριθμών {0, 1, …, K} και οι τιμές τους επιλέγονται έτσι, ώστε να πληρούν τη σχέση:

σ = αnwn + … + α1w1 + α0w0

Οι συντελεστές wi (i=0, …, n) ονομάζονται **βάρη** και έχουν προκαθορισμένες αριθμητικές τιμές. Τα σύμβολα αi μπορούν να έχουν οποιαδήποτε μορφή, αρκεί να δοθεί μία αντιστοιχία τους με τους ακεραίους αριθμούς {0, 1, …,K}. Στην περίπτωση της *δυαδικής* κωδικοποίησης τα αi παίρνουν τις τιμές {0,1},που ονομάζονται **bits**.

Παράδειγμα 1.11

Έστω ότι έχουμε δέκα δεδομένα, που παριστάνονται από τους αύξοντες αριθμούς 0 έως 9, και θέλουμε να τα κωδικοποιήσουμε σε δυαδική μορφή με βάρη τους αριθμούς 8, 4, 2, 1. Λόγω της δυαδικής μορφής, που ζητείται, τα σύμβολα αi του νέου κώδικα θα είναι τα bit “0” και “1”, τα οποία, για το σκοπό της κωδικοποίησης, αντιστοιχούνται με τους αριθμούς 0 και 1. Οι δυαδικές λέξεις που σχηματίζονται φαίνονται στη στήλη (8 4 2 1) του Πίνακα 1.2. Στον ίδιο πίνακα δίνονται οι κωδικοποιήσεις με βάρη τους αριθμούς (2, 4, 2, 1) και τους (6, 4, 2, –3). Για παράδειγμα το δεδομένο με αύξοντα αριθμό 8 κωδικοποιείται αντίστοιχα ως 1000 ή 1110 ή 1010, διότι:

8 = 8*1+4*0+2*0+1*0 = 2*1+4*1+2*1+1*0 = 6*1+4*0+2*1+(–3)*0.

Πίνακας 1.2 Κωδικοποίηση με τη μέθοδο των Βαρών.

α/α	8 4 2 1	2 4 2 1	6 4 2 –3
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 0 1
2	0 0 1 0	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1	1 0 0 1
4	0 1 0 0	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1	1 0 1 1
6	0 1 1 0	1 1 0 0	0 1 1 0
7	0 1 1 1	1 1 0 1	1 1 0 1
8	1 0 0 0	1 1 1 0	1 0 1 0
9	1 0 0 1	1 1 1 1	1 1 1 1

Η παράσταση τόσο των δυαδικών όσο και των δεκαδικών αριθμών δεν είναι τίποτε άλλο παρά μία μέθοδος κωδικοποίησης με βάρη αριθμητικών ποσοτήτων. Ο κώδικας με βάρη 8, 4, 2, 1, δηλ2^3, 2^2, 2^1, 2° είναι ο συνήθης τρόπος παράστασης δυαδικών αριθμών και για συντομία θα τον ονομάζουμε Bin8421.

Είναι δυνατόν οι παραστάσεις μερικών δεκαδικών αριθμών σε κάποιο κώδικα με βάρη να μην είναι μοναδικές. Στο παράδειγμα με βάρη 2, 4, 2, 1 του Πίνακα 1.2, ο δεκαδικός 7 μπορεί να παρασταθεί ως 1101, όπως και ως 0111. Στην περίπτωση αυτή διαλέγουμε τη μορφή που εμείς προτιμάμε.

1.2.2 Κώδικας BCD

Μία επέκταση της κωδικοποίησης αριθμών με βάρη τα 8, 4, 2, 1 είναι ο κώδικας BCD (Binary Coded Decimal). Στον κώδικα αυτό τα ψηφία α10 β10 γ10… ενός *δεκαδικού* αριθμού αντικαθίστανται από την αντίστοιχη δυαδική τους παράσταση (Πίνακας 1.2, βάρη 8 4 2 1).

Παράδειγμα 1.12

Η BCD κωδικοποίηση του δεκαδικού αριθμού 12345 είναι:

1234510 <=> 000100100011010001 01BCD διότι

$$\overbrace{0001}^{1}\overbrace{0010}^{2}\overbrace{0011}^{3}\overbrace{0100}^{4}\overbrace{0101}^{5}$$

Ας σημειωθεί ότι τέσσερα bits παράγουν 16 συνδυασμούς, από το 0000 έως το 1111, ενώ ο κώδικας BCD αξιοποιεί μόνο δέκα από αυτούς. Οι πλεονάζοντες συνδυασμοί 1010, 1011, 1100, 1101, 1110 και 1111 δεν αποτελούν λέξεις του κώδικα BCD.

Είναι εξίσου απλό να μετατραπεί ένας αριθμός από BCD σε δεκαδική μορφή. Αρκεί να χωριστεί ο BCD αριθμός σε τετράδες bits και να αντικατασταθεί κάθε τετράδα bits με το αντίστοιχο δεκαδικό ψηφίο.

Παράδειγμα 1.13

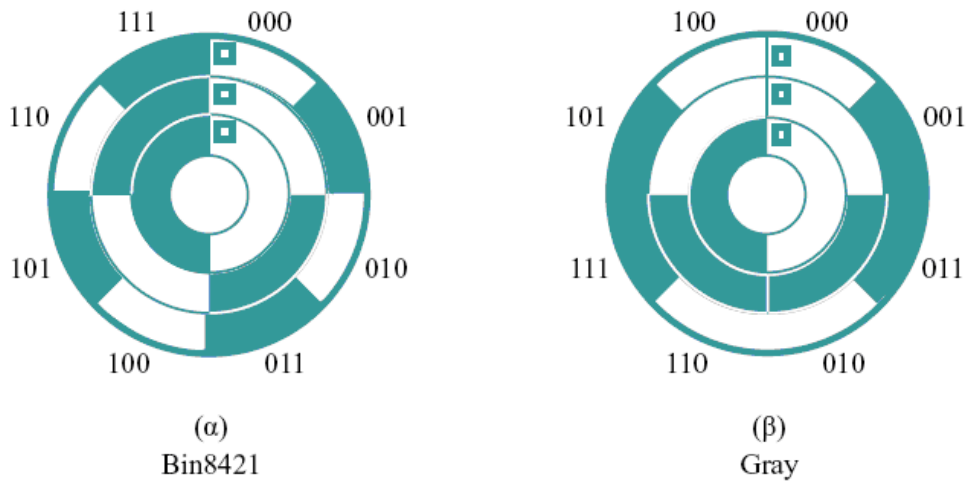
Ο ακόλουθος BCD αριθμός αντιστοιχεί στο δεκαδικό 2469:

$$\underbrace{0010}_{2}\underbrace{0100}_{4}\underbrace{0110}_{6}\underbrace{1001}_{9}$$

1.2.3 Κωδικοποίηση χωρίς Βάρη - Κώδικας Gray

Είναι μέθοδοι κωδικοποίησης με κανόνες που ακολουθούν άλλους τρόπους εκτός των βαρών. Σε πολλές πρακτικές εφαρμογές είναι π.χ. επιθυμητό να χρησιμοποιηθούν κώδικες, στους οποίους οι *διαδοχικές* λέξεις τους διαφέρουν μεταξύ τους σε ένα μόνο ψηφίο. Ένας κώδικας τέτοιας μορφής είναι ο κώδικας Gray. Για να εξηγηθεί η χρησιμότητά του, παρουσιάζονται στο σχήμα 1.2 δύο μηχανισμοί ψηφιακής κωδικοποίησης γωνιών: ο πρώτος (σχήμα 1.2.α) στον κώδικα Bin8421 (Πίνακας 1.2 – βάρη 8,4,2,1) και ο δεύτερος (σχήμα 1.2.β) στον κώδικα Gray, που θα δούμε παρακάτω. Χάριν απλότητας η γωνία των 360° αναλύεται μόνο

σε οκτώ στάθμες, ώστε να επαρκούν 3 bits για την κωδικοποίησή τους. Οι κωδικοποιητές αποτελούνται από κυκλικούς δίσκους που περιστρέφονται ανάλογα με τη γωνία. Κάθε δίσκος χωρίζεται σε τρεις κυκλικούς δακτυλίους, έναν για κάθε bit. Οι δακτύλιοι χωρίζονται σε κυκλικούς τομείς αγώγιμους (μαύρο χρώμα στο σχήμα 1.2) ή όχι (άσπρο). Τρεις ακίνητες επαφές με ελατήρια (τα τετραγωνάκια στο σχήμα 1.2), μία για κάθε δακτύλιο – bit του κώδικα, παράγουν τα αντίστοιχα σήματα ανάλογα με τη γωνιακή θέση που βρίσκεται ο δίσκος. Καθώς ο δίσκος περιστρέφεται, η τιμή του κάθε σήματος, “1” ή “0”, εξαρτάται από το εάν το ελατήριό του εφάπτεται σε αγώγιμο τομέα ή όχι. Ο μηχανισμός αυτός μετατροπής ενός συνεχούς μηχανικού μεγέθους (γωνία) σε ψηφιακή μορφή παρουσιάζει σφάλματα ανάγνωσης (εκτός από το σφάλμα ψηφιοποίησης), όταν κατά την περιστροφή του δίσκου τα ελατήρια εφάπτονται στη διαχωριστική γραμμή δύο κυκλικών τομέων (αγώγιμη – μη αγώγιμη κατάσταση). Ας δούμε τι συμβαίνει, όταν τα ελατήρια του κωδικοποιητή Bin8421 (σχήμα 1.2α) διασχίζουν τη διαχωριστική γραμμή από την τιμή 011 στην 100. Τα τρία σήματα πρέπει να αλλάξουν κατάσταση ταυτόχρονα. Η ταυτόχρονη, όμως, αλλαγή απαιτεί απόλυτη ευθυγράμμιση του συστήματος, που είναι τεχνικά αδύνατη. Έτσι, εάν π.χ. το σημείο επαφής του πρώτου ελατηρίου προηγείται ή καθυστερεί ως προς τα άλλα, τότε θα προκύψουν εσφαλμένες τιμές, που μπορεί να κυμαίνονται από το 111 (σφάλμα 3 μονάδες) έως το 000 (σφάλμα 4 μονάδες) αντίστοιχα. Ο κωδικοποιητής Gray (σχήμα 1.2β) περιορίζει τα σφάλματα ευθυγράμμισης των επαφών στο ελάχιστο. Όπως φαίνεται στο σχήμα 1.2β, διαδοχικές τιμές της γωνίας διαφέρουν μόνο σε ένα bit (αλλαγή ενός σήματος), συνεπώς το μέγιστο σφάλμα σε κάθε περίπτωση είναι μία μονάδα επάνω ή κάτω.



Σχήμα 1.2 Κωδικοποιητές Γωνίας.

Ο κώδικας Gray είναι μέλος μιας κατηγορίας κωδίκων, που ονομάζονται κώδικες **ανακλάσεως** (reflection codes). Ένας κώδικας ανακλάσεως των n bit παράγεται από έναν κώδικα ανακλάσεως των n–1 bit με την εξής μέθοδο:

α) οι λέξεις του κώδικα των n–1 bit γράφονται κατά σειρά η μία κάτω από την άλλη και κατόπιν κατοπτρίζονται ως προς ένα επίπεδο, δηλ. ξαναγράφονται κάτω από το επίπεδο με αντίστροφη σειρά,

β) μπροστά από τις λέξεις τοποθετείται ένα νέο bit, που έχει την τιμή 0 για τις λέξεις που βρίσκονται πάνω από το επίπεδο κατοπτρισμού και την τιμή 1 για τις κάτω.

Στο σχήμα 1.3 φαίνεται ο τρόπος που παράγεται με διαδοχικές ανακλάσεις ο κώδικας Gray για 2 και 3 bit, ξεκινώντας από ένα κώδικα του ενός bit.

Κώδικας Gray με:	1 bit	2 bit	3 bit
	0	00	000
	1	01	001
		11	011
επίπεδο ανακλάσεως		10	010
			110
			111
			101
			100

Σχήμα 1.3 Παραγωγή των λέξεων του κώδικα Gray με ανάκλαση.

Στην ανωτέρω σημαντική ιδιότητα πρέπει να προστεθεί και ο απλός τρόπος μετατροπής από Bin8421 σε κώδικα Gray και αντίστροφα. Στο σχήμα 1.4 δίνεται η αντιστοιχία μεταξύ των ψηφίων bi (i=0,1,2) του Bin8421 και των ψηφίων gi (i=0,1,2) του Gray.

Η μετατροπή γίνεται ως εξής: έστω ότι gngn–1 … g1g0 είναι μία λέξη του κώδικα Gray με (n+1) bits και έστω ότι bnbn–1 … b1b0 είναι η αντίστοιχη Bin8421 μορφή. Τότε το ψηφίο gi του κώδικα Gray υπολογίζεται από τα ψηφία του Bin8421 με τον εξής κανόνα:

$$g_n = b_n \qquad (2.1.\alpha)$$

$$g_i = b_i \oplus b_{i+1}, \quad 0 \le i < n \qquad (2.1.\beta)$$

όπου το σύμβολο $\oplus$ δηλώνει το άθροισμα <u>modulo 2</u>, που ορίζεται ως εξής:

$$0 \oplus 0 = 0,\ 0 \oplus 1 = 1,\ 1 \oplus 0 = 1,\ 1 \oplus 1 = 0$$

α/α	Bin8421 $b_2b_1b_0$	Gray $g_2g_1g_0$
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

Σχήμα 1.4 Αντιστοιχία ψηφίων κωδικών 8421 με Gray.

Παράδειγμα 1.14

Η λέξη του κώδικα Gray που αντιστοιχεί στον Bin8421 αριθμό 101101, προκύπτει ότι είναι η 111011:

$$g_5 = b_5 \quad => \quad 1$$

$$g_4 = b_4 \oplus b_5 \quad => \quad 0 \oplus 1 = 1$$

$$g_3 = b_3 \oplus b_4 \quad => \quad 0 \oplus 1 = 1$$

$$g_2 = b_2 \oplus b_3 \quad => \quad 1 \oplus 1 = 0$$

$$g_1 = b_1 \oplus b_2 \quad => \quad 0 \oplus 1 = 1$$

$$g_0 = b_0 \oplus b_1 \quad => \quad 1 \oplus 0 = 1$$

Αντίστροφα, η μετατροπή από κώδικα Gray σε Bin8421 δίνεται από τον κανόνα:

$$b_n = g_n$$

$$b_i = g_i \oplus g_{i+1} \oplus g_{i+2} \oplus \ldots \oplus g_n, \quad 0 \le i < n$$

Ο κανόνας αυτός οδηγεί σε μια πρακτική μέθοδο, κατά την οποία ξεκινώντας από το πλέον σημαντικό ψηφίο, το bn, παράγουμε διαδοχικά τα μικρότερης τάξης ψηφία:

$$b_n = g_n$$

$$b_i = g_i \oplus bi{+}1, \quad 0 \le i < n.$$

Παράδειγμα 1.15

Η λέξη 1101 του κώδικα Gray αντιστοιχεί στον Bin8421 αριθμό 1001:

$$b_3 = g_3 \quad => \quad 1$$

$$b_2 = g_2 \oplus b_3 \quad => \quad 1 \oplus 1 = 0$$

$$b_1 = g_1 \oplus b_2 \quad => \quad 0 \oplus 0 = 0$$

$$b_0 = g_0 \oplus b_1 \quad => \quad 1 \oplus 0 = 1$$

1.3 Κώδικες ανίχνευσης και διόρθωσης λαθών

Όταν θέλουμε να προστατεύσουμε μία πληροφορία από το θόρυβο, εκτός από τη διεύρυνση των σταθμών πλάτους του σήματος που δεν είναι πάντοτε μέσα στις δυνατότητές μας, μπορούμε να επιτύχουμε το ίδιο ή καλύτερο αποτέλεσμα με την κωδικοποίηση. Με την προσθήκη πλεοναζόντων bits στις λέξεις ενός κώδικα μπορούμε να ανιχνεύουμε, με αρκετή πιθανότητα, πότε μία λέξη έχει αλλοιωθεί

από το θόρυβο και επίσης μπορούμε από την αλλοιωμένη λέξη να ανακτούμε τη σωστή πληροφορία. Στην παρούσα ενότητα αναλύονται οι συνέπειες του θορύβου και παρουσιάζονται πρακτικές τεχνικές κωδικοποίησης για την ανίχνευση και τη διόρθωση λαθών από το θόρυβο.

Ένα μεγάλο πρόβλημα, που παρουσιάζεται κατά τη μετάδοση ή αποθήκευση πληροφοριών, είναι η αλλοίωσή τους (του σήματος φορέα) λόγω της πανταχού παρουσίας του **θορύβου**. Ο θόρυβος είναι εγγενής σε κάθε φυσικό σύστημα.

Οι επιπτώσεις, που μπορεί να έχει ο θόρυβος σ' ένα ψηφιακό σύστημα στην πράξη, είναι:

(α) καμία αλλοίωση της κατάστασης των σημάτων, διότι η τρέχουσα στάθμη του θορύβου είναι μέσα στα όρια ανοχής του συστήματος,

(β) αλλοίωση ενός ή περισσοτέρων bits μιας λέξης. Λόγω της δυαδικής μορφής των σημάτων αλλοίωση ενός bit σημαίνει αντιστροφή της κατάστασής του.

Οι μέθοδοι κωδικοποίησης που περιγράψαμε μέχρι εδώ, αν και είναι επαρκείς για άλλες χρήσεις, δεν προστατεύουν από λάθη λόγω θορύβου. Στα επόμενα θα μελετηθούν τρόποι κωδικοποίησης που να επιτρέπουν την ανίχνευση και την διόρθωση τέτοιων λαθών.

Ο θόρυβος από μαθηματικής πλευράς είναι **στατιστικό φαινόμενο**. Θεωρητικά, το πλάτος (ύψος) του θορύβου μπορεί να γίνει απεριόριστα μεγάλο και η χρονική συμπεριφορά του είναι απρόβλεπτη. Στην περίπτωση των ψηφιακών σημάτων η επίδραση του θορύβου μετράται με την πιθανότητα p να συμβεί ένα λάθος, δηλ. να αλλοιωθεί ένα bit κατά τη μετάδοση μιας λέξης. Τότε η πιθανότητα να συμβούν N λάθη *ταυτόχρονα* στην ίδια λέξη είναι p^N, η οποία είναι μεν υπαρκτή, είναι όμως πολύ μικρή. Για τους λόγους αυτούς η μελέτη των κωδίκων θα περιοριστεί κυρίως στην περίπτωση *ενός* λάθους, που είναι η συνηθέστερη στην πράξη. Σ' έναν κώδικα, που αποτελείται από λέξεις των n bit η καθεμία, ονομάζεται **απόσταση Hamming** ή για συντομία *απόσταση* μεταξύ δύο λέξεών του το πλήθος των bits που πρέπει να αλλάξουν τιμή για να ταυτισθεί η μία λέξη με την άλλη, π.χ.

οι λέξεις 0110 και 0101 έχουν απόσταση δύο. Ονομάζεται **απόσταση του κώδικα** η ελάχιστη των αποστάσεων μεταξύ όλων των λέξεών του.

1.3.1 Κώδικες ανίχνευσης λαθών

Εάν χρησιμοποιείται ένας κώδικας με λέξεις των n bit και κάθε ένας από τους 2^n συνδυασμούς bits αποτελεί και μία λέξη του κώδικα, δηλ. μεταφέρει μία διαφορετική πληροφορία, τότε η αλλοίωση (αντιστροφή) ενός bit κάποιας λέξης του, λόγω θορύβου, θα προκαλέσει την εμφάνιση στον παραλήπτη μιας άλλης λέξης του κώδικα, η οποία μεταφέρει διαφορετική πληροφορία. Η νέα λέξη είναι λανθασμένη ως προς το σκοπό που στάλθηκε, εξακολουθεί όμως να είναι μία έγκυρη λέξη του δοθέντος κώδικα και ο παραλήπτης δεν έχει τη δυνατότητα να καταλάβει ότι έγινε λάθος κατά τη μεταβίβαση. Για παράδειγμα στον κώδικα 00, 01, 10, 11, εάν σταλεί η λέξη 00 και παραληφθεί η 01 ή η 10, ο παραλήπτης δεν έχει κανένα τρόπο να γνωρίζει ότι στάλθηκε αρχικά η 00 και επομένως θα θεωρήσει τη λανθασμένη λέξη σαν σωστή. Εάν ένας κώδικας ήταν κατασκευασμένος έτσι, ώστε, όταν συμβεί ένα οποιοδήποτε λάθος σε μία λέξη του, να μετασχηματίζει την έγκυρη κωδική λέξη σε άσχετη, εκτός κώδικα, λέξη (*παράνομη*, άκυρη λέξη), τότε ο παραλήπτης θα καταλάβαινε αμέσως ότι δεν μπορεί να είχε σταλεί τέτοια λέξη. Ένας κώδικας που έχει αυτή την ιδιότητα για την περίπτωση που θα συμβούν μέχρι N λάθη σε οποιαδήποτε λέξη του ονομάζεται κώδικας **ανίχνευσης N λαθών**. Π.χ. ο κώδικας με μόνες λέξεις τις 00 και 11 είναι ένας κώδικας ανίχνευσης ενός λάθους, διότι, εάν υποτεθεί ότι στάλθηκε αρχικά η λέξη 00 και ότι παραλήφθηκε η λέξη 01, αμέσως γίνεται φανερή η ύπαρξη λάθους, αν και ο παραλήπτης δεν γνωρίζει σε ποια λέξη έγινε το λάθος, δηλ. δεν μπορεί να αντιληφθεί εάν αρχικά στάλθηκε η λέξη 00 και το λάθος έγινε στο δεύτερο bit (00 => 01) ή εάν στάλθηκε η λέξη 11 και το λάθος έγινε στο πρώτο bit (11 => 01). Έστω οι συνδυασμοί των τριών bit τοποθετημένοι ως ακολούθως, ώστε η κάθε σειρά να έχει *απόσταση* από τις γειτονικές της κατά ένα:

1)		000	
2)	001	010	100
3)	011	101	110
4)		111	

Εάν επιλεγούν σαν λέξεις του κώδικα οι 000, 011, 101 και 110 (υπογραμμισμένες), τότε είναι φανερό ότι, εάν συμβεί ένα λάθος σε οποιαδήποτε έγκυρη λέξη, τότε θα φθάσει στον παραλήπτη μία από τις λέξεις των σειρών 2 ή 4, που δεν ανήκουν στον κώδικα και συνεπώς είναι άκυρες. Η παρουσία ενός λάθους μετασχηματίζει τις έγκυρες κωδικές λέξεις των σειρών 1 και 3 σε λέξεις (άκυρες), που απέχουν κατά ένα (σειρές 2 και 4). Συνεπώς, ένας κώδικας, για να ανιχνεύει ένα λάθος, πρέπει να έχει απόσταση μεγαλύτερη από ένα. Στο παράδειγμά μας, οι σειρές 1 και 3 έχουν μεταξύ τους απόσταση δύο. Γενικώς, *ένας κώδικας, για να ανιχνεύει N λάθη, πρέπει να έχει απόσταση μεγαλύτερη του N*. Εάν στον κώδικα του ανωτέρω παραδείγματος συνέβαιναν δύο λάθη, η ανίχνευσή τους θα ήταν αδύνατη, διότι οι λέξεις των σειρών 1 και 3 απέχουν μεταξύ τους κατά δύο και αλλαγή σε δύο bits θα τις έκανε να συμπέσουν. Οι κωδικές λέξεις σ' έναν κώδικα ανίχνευσης ενός λάθους εκλέγονται με τέτοιο τρόπο, ώστε για να αλλάξει μία έγκυρη κωδική λέξη σε μία άλλη έγκυρη λέξη, πρέπει να αλλάξουν τιμή τουλάχιστον δύο ψηφία (απόσταση δύο).

Ένας συστηματικός τρόπος για την εκλογή των κωδικών λέξεων βασίζεται στην παρατήρηση ότι, εάν από όλες τις δυνατές λέξεις επιλεγούν μόνον αυτές που έχουν π.χ. άρτιο πλήθος από “1”, τότε αλλαγή της τιμής ενός οποιουδήποτε ψηφίου μιας απ’ αυτές τις λέξεις οδηγεί σε συνδυασμό που περιέχει περιττό πλήθος από “1” και επομένως δεν περιέχεται στον κώδικα. Ο κώδικας των τριών ψηφίων του προηγούμενου παραδείγματος ανιχνεύει ένα λάθος, γιατί οι λέξεις του έχουν άρτιο πλήθος 1 (ο συνδυασμός 000 θεωρείται άρτιος). Ένας γενικός κανόνας για τη μετατροπή οποιουδήποτε κώδικα, που στο εξής θα ονομάζεται **πηγαίος κώδικας**, σε κώδικα ανίχνευσης ενός λάθους βασίζεται στην προηγούμενη παρατήρηση και έχει ως ακολούθως:

1) σε κάθε λέξη του πηγαίου κώδικα τοποθετείται ένα επιπλέον ψηφίο, που

ονομάζεται **ψηφίο ισοτιμίας** (parity bit),

2) στο ψηφίο ισοτιμίας δίνεται τέτοια τιμή, ώστε το συνολικό πλήθος των 1 στη λέξη να είναι άρτιο (κώδικας με *άρτια ισοτιμία*) ή περιττό (*περιττή ισοτιμία*).

Η θέση του ψηφίου ισοτιμίας είναι αυθαίρετη για ένα κώδικα, αρκεί να τηρηθεί η θέση αυτή σε όλες τις λέξεις του, συνήθως τοποθετείται στο τέλος της λέξης. Η εκλογή άρτιας ή περιττής ισοτιμίας είναι επίσης αυθαίρετη για έναν κώδικα. Εννοείται ότι ο παραλήπτης πρέπει να γνωρίζει εκ των προτέρων τη θέση του ψηφίου και το είδος της ισοτιμίας.

Παράδειγμα 1.16

Έστω ο ακόλουθος πηγαίος κώδικας και ζητείται να παραχθεί από αυτόν ένας κώδικας που να έχει περιττή ισοτιμία:

πηγαίος κώδικας	κώδικας με περιττή ισοτιμία
Α = 00	001 (ψηφίο ισοτιμίας στο τέλος)
Β = 01	010
Γ = 10	100
Δ = 11	111

Εάν συμβεί ένα μόνο λάθος σε μία λέξη, π.χ. στην 001, τότε αυτή μετατρέπεται σε παράνομη (101 ή 011 ή 000 => άρτιο πλήθος 1) και το λάθος ανιχνεύεται εύκολα μετρώντας το πλήθος των 1. Αν και αυτή η κωδικοποίηση ισοτιμίας προορίζεται για την ανίχνευση ενός μόνο λάθους, εύκολα φαίνεται ότι στην πραγματικότητα ανιχνεύει οποιοδήποτε περιττό πλήθος λαθών. Η προσθήκη του ψηφίου ισοτιμίας φαινομενικά αποτελεί επιβάρυνση του κώδικα, διότι αυξάνει το μήκος των λέξεών του, και συνεπώς αυξάνει το κόστος μεταφοράς και επεξεργασίας, χωρίς να μεταφέρει πληροφορία χρήσιμη στον παραλήπτη. Δηλαδή ένας τέτοιος κώδικας N ψηφίων μεταφέρει μόνο 2^{N-1} πληροφορίες αντί των 2^N (διπλάσιες) που θα

μπορούσε να στείλει. Στην πραγματικότητα το ψηφίο ισοτιμίας μεταφέρει και αυτό πληροφορίες, που είναι χρήσιμες για την ανίχνευση του λάθους, αλλά πλεονάζουν ως προς την πηγαία πληροφορία και γι' αυτό ονομάζονται **πλεονάζουσες πληροφορίες** (redundant information).

2.1 Άλγεβρα Boole

Η σχεδίαση οποιουδήποτε ψηφιακού κυκλώματος απαιτεί την καλή σχεδίαση του Συνδυαστικού του μέρους. Μετά τη χρησιμοποίηση του "Λογισμού των Προτάσεων" από τον Shannon (1938) για τη μελέτη των Κυκλωμάτων διακοπτών, η άλγεβρα Boole έχει καταξιωθεί σαν το βασικό μαθηματικό εργαλείο για την ανάλυση και σχεδίαση κάθε ψηφιακού κυκλώματος.

Η άλγεβρα Boole δημιουργήθηκε από τον G. Boole, ο οποίος προσπάθησε να διατυπώσει τους κανόνες της Αριστοτελικής Λογικής με συστηματικό και συμβολικό τρόπο, ώστε να διευκολυνθεί η παραγωγή των λογικών συμπερασμάτων και να αποφεύγονται τα λάθη στους συλλογισμούς.

Η άλγεβρα του Boole ορίζεται σαν μία *αλγεβρική δομή,* δηλ. ένα μαθηματικό σύστημα, το οποίο περιέχει τα εξής:

α) Ένα *σύνολο από στοιχεία*, που έχουν κοινές ιδιότητες. Το γεγονός ότι ένα στοιχείο x ανήκει στο σύνολο B συμβολίζεται ως: x∈B. Ένα σύνολο B, που αποτελείται από πεπερασμένο πλήθος στοιχείων, π.χ. τα 1, 2, 3 και 4, μπορεί να οριστεί περικλείοντας τα στοιχεία του μέσα σε αγκύλες: B = {1, 2, 3, 4}.

β) Ένα σύνολο από *πράξεις*. Μια πράξη είναι στην ουσία ένας κανόνας που μας λέει ότι, όταν συνδυάσουμε σύμφωνα με αυτή την πράξη τα στοιχεία x και y του συνόλου B, θα προκύψει το στοιχείο z, π.χ. η πράξη της πρόσθεσης των αριθμών μας λέει ότι, όταν τον αριθμό 3 το συνδυάσω (προσθέσω) με τον 4, θα πάρω τον 7.

γ) Ένα σύνολο από *αξιώματα*. Αποτελούν τις βασικές παραδοχές του συστήματος, από τις οποίες προκύπτουν όλες οι άλλες ιδιότητες της άλγεβρας, που ονομάζονται θεωρήματα.

Από την εποχή του Boole έχουν δημιουργηθεί διάφορες παραλλαγές της άλγεβρας αυτής (άλγεβρα Boole–Schroeder). Εδώ θα παρουσιάσουμε μία παραλλαγή της, που είναι και η σημαντικότερη, και στηρίζεται στην αξιωματική θεμελίωση της άλγεβρας του Boole που προτάθηκε από τον Huntington (1904).

ΟΡΙΣΜΟΣ ΤΗΣ ΑΛΓΕΒΡΑΣ BOOLE

Η **Άλγεβρα Boole** ορίζεται, σύμφωνα με τη θεμελίωση του Huntington, σαν μια αλγεβρική δομή **B**:

$$\mathbf{B} = < B, +, ., –, 0, 1 >,$$

Όπου :

α) Β είναι ένα **σύνολο στοιχείων**, που περιέχει *δύο τουλάχιστον στοιχεία*,

β) Τα σύμβολα "+", ".", "–" δηλώνουν **πράξεις** επάνω στα στοιχεία του Β, όπου:

η πράξη "+" ονομάζεται **λογικό άθροισμα**,

η πράξη "." ονομάζεται **λογικό γινόμενο**, και

η πράξη "–" ονομάζεται **λογική αντιστροφή**.

γ) Τα "0" και "1" παριστάνουν ειδικά στοιχεία του Β (0,1∈Β), που ονομάζονται **σταθερές** του συστήματος.

δ) Για κάθε στοιχείο a, b, c του συνόλου Β οι ανωτέρω πράξεις ικανοποιούν τα **αξιώματα** του Πίνακα 2.1:

Πίνακας 2.3 Αξιώματα της άλγεβρας Boole (Huntighton).

A0.	$a, b \in B$ και $a \neq b$,	το σύνολο Β έχει δύο τουλάχιστον στοιχεία.	
A1.	$a+b \in B$,	$a.b \in B$	*Κλειστότης*
A2.	$a+b=b+a$,	$a.b=b.a$	*Αντιμεταθετικότης*
A3.	$a+(b.c)=(a+b).(a+c)$,	$a.(b+c)=(a.b)+(a.c)$	*Επιμεριστικότης*
A4.	$a+0=a$,	$a.1=a$	*Ουδέτερα Στοιχεία*
A5.	$a+\bar{a}=1$,	$a.\bar{a}=0$	*Ύπαρξη Συμπληρώματος*

Στα ακόλουθα, όπου δεν δημιουργείται σύγχυση, το σύμβολο του λογικού γινομένου θα παραλείπεται, δηλ. η πράξη a.b θα γράφεται απλώς ab. Επίσης, το αποτέλεσμα της πράξης "–" επί ενός στοιχείου a ($\bar{a}$ αντίστροφο ή συμπληρωματικό του a), που κανονικά παριστάνεται σαν $\bar{a}$, θα παριστάνεται εδώ, όπως γίνεται συνήθως στη βιβλιογραφία για τυπογραφικούς λόγους, είτε ως $\bar{a}$ είτε ως a'.

Παρατηρήσεις:

Παρατηρούμε ότι τα αξιώματα της άλγεβρας Boole, εκτός του A0, εμφανίζουν μία συμμετρία ως προς τις πράξεις, η πρώτη στήλη αφορά στην πρόσθεση και η δεύτερη στον πολλαπλασιασμό, και γι' αυτό παρουσιάζονται ανά ζεύγη. Αν αναλύσουμε τα αξιώματα αυτά με απλούστερα λόγια, μας λένε τα εξής:

α) Το αξίωμα Α0 μας λέει ότι το σύνολο Β πρέπει να περιέχει τουλάχιστον δύο στοιχεία. Υπόψη ότι στα στοιχεία του Β προσμετρούνται και οι σταθερές “0” και “1”. Γενικά το Β, ανάλογα με την εφαρμογή, μπορεί να περιέχει μέχρι άπειρο πλήθος (αριθμήσιμο) στοιχείων.

β) Σύμφωνα με το αξίωμα Α1 οι πράξεις του λογικού αθροίσματος και του γινομένου είναι κλειστές, δηλ. οι πράξεις αυτές για κάθε ζεύγος στοιχείων a, b του Β καθορίζουν (αντιστοιχούν) πάντοτε ένα στοιχείο c (που μπορεί να ταυτίζεται με το a ή το b), που ανήκει στο σύνολο Β. Συγκρίνετε το αξίωμα αυτό με την πράξη, π.χ. της διαίρεσης των ακεραίων αριθμών, όπου το πηλίκο δεν ανήκει πάντα στους ακέραιους αριθμούς.

γ) Από το αξίωμα Α2 προκύπτει ότι το αποτέλεσμα των πράξεων είναι το ίδιο είτε στο στοιχείο a προστεθεί (πολλαπλασιαστεί) το b είτε στο στοιχείο b προστεθεί (πολλαπλασιαστεί) το a.

δ) Το αξίωμα Α3 δηλώνει ότι ισχύει ο επιμερισμός όχι μόνο του γινομένου στο άθροισμα, a.(b+c), αλλά επιπλέον το άθροισμα επιμερίζεται στο γινόμενο, a+(b.c). Αυτό δεν συμβαίνει στην άλγεβρα των αριθμών και πρέπει να μας κάνει προσεκτικούς στις πράξεις με αυτή την άλγεβρα.

ε) Το αξίωμα Α4 μας λέει ότι, όσον αφορά στις πράξεις του λογικού αθροίσματος και του λογικού γινομένου, υπάρχουν δύο ειδικά στοιχεία του συνόλου Β, το “0” και το “1” αντίστοιχα, που ονομάζονται **ουδέτερα** στοιχεία ή στοιχεία **ταυτότητος**. Όσες φορές προσθέτουμε το “0” ή πολλαπλασιάζουμε με το “1”, το αποτέλεσμα δεν αλλάζει.

στ) Το αξίωμα Α5 μας λέει ότι για κάθε στοιχείο a του Β υπάρχει πάντοτε ένα στοιχείο, που ονομάζεται **συμπληρωματικό** ή **αντίστροφο** του a και συμβολίζεται $\bar{a}$, το οποίο συνδυαζόμενο με το a δίνει σαν αποτέλεσμα την σταθερά “1” ή “0”. Προσοχή: το $\bar{a}$ δεν είναι αντίθετο του a, δηλ. $a + \bar{a} \neq 0$

Συγκρίνοντας την άλγεβρα Boole με την άλγεβρα των αριθμών παρατηρούμε ότι έχουν αρκετές ομοιότητες. Η άλγεβρα Boole, όμως, διαφέρει σε ορισμένα βασικά σημεία από την άλγεβρα των αριθμών, τα οποία θα πρέπει να τα προσέχουμε για να μην παρασυρόμαστε σε λανθασμένα αποτελέσματα:

α) Δεν έχει αντίστροφες πράξεις, π.χ. δεν υπάρχει πράξη αντίστοιχη της αφαίρεσης ή της διαίρεσης, επομένως δεν μπορούμε να μεταφέρουμε στοιχεία από το ένα μέλος μιας εξίσωσης στο άλλο με τον τρόπο που γίνεται στην άλγεβρα των αριθμών.
β) Ισχύει ο *επιμερισμός του αθροίσματος στον πολλαπλασιασμό* (αξίωμα Α3), δηλ. a+(b.c)=(a+b).(a+c), που δεν υπάρχει στην άλγεβρα των αριθμών. Είναι ένα αξίωμα, που οδηγεί σε απρόσμενα πλην όμως ενδιαφέροντα αποτελέσματα.
γ) Δεν υπάρχουν *αντίθετα* στοιχεία, επομένως π.χ., εάν ένα στοιχείο εμφανίζεται και στα δύο μέλη μιας εξίσωσης δεν μπορούμε να το απαλείψουμε, δηλ. η σχέση a+c = b+c δεν συνεπάγεται τη σχέση a = b.
δ) Τα *συμπληρωματικά* ή *αντίστροφα* στοιχεία (αξίωμα Α5) δεν έχουν το αντίστοιχό τους στην άλγεβρα των αριθμών.

Εκτός από τις ανωτέρω παρατηρήσεις, θα πρέπει να αναφέρουμε ότι στον ορισμό της άλγεβρας αυτής δεν καθορίζονται:
α) το είδος και το πλήθος των στοιχείων του συνόλου Β και
β) το περιεχόμενο των πράξεών της, δηλ. ο κανόνας σύμφωνα με τον οποίο συνδυάζονται δύο στοιχεία για να δώσουν το αποτέλεσμα. Ο λόγος εδώ είναι ότι το αλγεβρικό σύστημα το οποίο παρουσιάσαμε και επεκράτησε να ονομάζεται "άλγεβρα Boole", από καθαρά μαθηματικής απόψεως δεν ορίζει μία συγκεκριμένη άλγεβρα αλλά μία οικογένεια από άλγεβρες ("άλγεβρες του Boole"). Επαφίεται στον κάθε μελετητή να "ορίσει" με το δικό του τρόπο την άλγεβρα που επιθυμεί, δηλ. να ορίσει το περιεχόμενο του συνόλου Β και των πράξεων, φροντίζοντας όμως πάντοτε να εκπληρούνται τα αξιώματα της άλγεβρας Boole. Μια τέτοια περίπτωση είναι η Δίτιμη Άλγεβρα, που παρουσιάζεται στην Ενότητα 2.2. Εδώ, για να γίνουν κατανοητά τα όσα αναφέραμε, θα δώσουμε ένα μικρό παράδειγμα ενός συστήματος Boole βασισμένο στις ιδιότητες των συνόλων:

2.1.1 Αρχή του Δυϊσμού

Στον Πίνακα 2.1 τα αξιώματα Α1 – Α5 είναι τοποθετημένα ανά ζεύγη και παρατηρούμε ότι υπάρχει μια "συμμετρία" εντός του κάθε ζεύγους. Η πρώτη στήλη αφορά στις ιδιότητες της πράξης του λογικού αθροίσματος και, εάν στα σημεία ακριβώς, όπου εμφανίζεται η πράξη του λογικού αθροίσματος, τοποθετήσουμε την πράξη του λογικού γινομένου, και αντίστροφα, τότε προκύπτουν τα αξιώματα της

δεύτερης στήλης. Π.χ. η εναλλαγή των πράξεων “+” και “.” στο πρώτο από τα αξιώματα Α3 οδηγεί στο δεύτερο:

$$a+(b.c) = (a+b).a+c)$$

$$a.(b+c) = (a.b)+(a.c)$$

Η συμμετρία αυτή, που υπάρχει στα αξιώματα της άλγεβρας Boole, αποτελεί μια βαθύτερη ιδιότητα αυτής της άλγεβρας και ονομάζεται Αρχή του Δυϊσμού. Η **Αρχή του Δυϊσμού** (Duality principle) μας λέει το εξής: Μία σχέση Α ονομάζεται **δυϊκή** (dual) μιας άλλης Β, εάν η Α προκύπτει από τη Β:

α) με αμοιβαία εναλλαγή των πράξεων του γινομένου και του αθροίσματος και

β) με αμοιβαία εναλλαγή των σταθερών 0 και 1.

Η αρχή του δυϊσμού στην άλγεβρα Boole οδηγεί στο εξής θεώρημα: **Θεώρημα του Δυϊσμού**: Εάν μια σχέση της άλγεβρας Boole είναι αληθής, τότε θα είναι αληθής και η δυϊκή της. Π.χ.: εάν (a+a).(b'+1) = a, τότε θα αληθεύει και η σχέση (a.a)+(b'.0) = a

Το θεώρημα του Δυϊσμού αποδεικνύεται, εάν παρατηρήσουμε ότι:

α) καθένα από τα αξιώματα Α1 – Α5 της πρώτης στήλης έχει το δυϊκό του στη δεύτερη στήλη, συνεπώς ισχύει για τα αξιώματα,

β) η απόδειξη μιας σχέσης Α ακολουθεί τα ίδια βήματα με την απόδειξη της δυϊκής της Β, με τη διαφορά ότι στη θέση των αξιωμάτων και θεωρημάτων που χρησιμοποιούνται για τη Β εφαρμόζονται τα δυϊκά τους.

Το θεώρημα του δυϊσμού μας διευκολύνει να αποδείξουμε μια σχέση, εάν έχει αποδειχθεί η δυϊκή της.

2.1.2 Λογική Παράσταση

Μια παράσταση της μορφής π.χ. a+(b+c).a+a.b+(b+c).b, που παραθέτει στη σειρά σύμβολα λογικών μεταβλητών, δηλ. στοιχεία που παίρνουν τιμές από το σύνολο Β, π.χ. a, b, c, και μεταξύ των μεταβλητών καθορίζει ορισμένες λογικές πράξεις, αποτελεί μια λογική παράσταση. Ειδικότερα, **Λογική Παράσταση** ονομάζεται κάθε συνδυασμός πεπερασμένου πλήθους μεταβλητών a,b,c … ∈Β ή/και σταθερών 0, 1 της άλγεβρας, που συνδέονται μεταξύ τους μέσω των πράξεων “+”, “.”, “–” . Π.χ. η a+b.c.(a+c)' είναι μια λογική παράσταση, όπως και η a.b+(a'+b').(a'+b.c').

Η λογική παράσταση είναι ο τρόπος με τον οποίο διατυπώνονται τα θεωρήματα και οι λοιπές σχέσεις της άλγεβρας Boole.

Όταν υπολογίζουμε μια λογική παράσταση, πρέπει να ακολουθούμε την προτεραιότητα των τελεστών, που είναι εξ ορισμού: παρενθέσεις, αντιστροφή, γινόμενο και τελευταία η πρόσθεση. Π.χ. η παράσταση a.b+a.c ισοδυναμεί με την (a.b)+(a.c). Υπενθυμίζεται ότι η παράσταση ab σημαίνει την a.b.

2.1.3 Βασικά θεωρήματα

Από τον ορισμό και τα αξιώματα A0 – A5 της άλγεβρας, που παρουσιάστηκαν στην Ενότητα 2.1, παράγονται θεωρήματα, τα σπουδαιότερα από τα οποία παρουσιάζονται σ' αυτή την Υποενότητα. Τα θεωρήματα παρατίθενται κατά δυικά ζεύγη και στις αποδείξεις που ακολουθούν σημειώνονται δίπλα τα αξιώματα και τα θεωρήματα τα οποία χρησιμοποιήθηκαν.

ΘΕΩΡΗΜΑ 1

a+a = a,

Είναι:

a = a+0	A4
= a+a.a'	A5
= (a+a).(a+a')	A3
= (a+a).1	A5
= a+a	A4

a.a = a

Είναι:

a = a.1	A4
= a.(a+a')	A5
= a.a+a.a'	A3
= a.a+0	A5
= a.a	A4

ΘΕΩΡΗΜΑ 2

a+1 = 1,

Είναι:

1 = a+a'	A5
= a+(1.a')	A4
= (a+1).(a+a')	A3
= (a+1).1	A5
= a+1	A4

a.0 = 0

Είναι:

0 =a.a'	A5
= a.(0+a')	A4
= a.0+a.a'	A3
= a.0+0	A5
= a.0	A4

ΘΕΩΡΗΜΑ 3

a+(a.b) = a,

Είναι:

$$\begin{aligned} a &= a.1 && A4 \\ &= a.(b+1) && \Theta 2 \\ &= a.(1+b) && A1 \\ &= (a.1)+(a.b) && A3 \\ &= a+(a.b) && A4 \end{aligned}$$

a.(a+b) = a

Είναι:

$$\begin{aligned} a &= a+0 && A4 \\ &= a+(b.0) && \Theta 2 \\ &= a+(0.b) && A1 \\ &= (a+0).(a+b) && A3 \\ &= a.(a+b) && A4 \end{aligned}$$

ΘΕΩΡΗΜΑ· 4α

Προσεταιριστική ιδιότητα αθροίσματος:

(a+b)+c = a+(b+c)

Ισχύει:

$$[a+(b+c)][(a+b)+c] = [(a+b)+c][a+(b+c)] \qquad A2$$

$$\{[a+(b+c)](a+b)\}+\{[a+(b+c)]c\} = \{[(a+b)+c]a\}+\{[(a+b)+c](b+c)\} \qquad A3$$

$$[a+(b+c)a]+[ab+(b+c)b]\}+\{[ac]+[(b+c)c]\} =$$
$$=\{[(a+b)a]+[ac]\}+\{[(a+b)b+bc]+[(a+b)c+c]\}$$

$$\{[a]+[ab+(b+bc)]\}+\{[ac]+[bc+c]\} = \{[a+ab]+[ac]\}+\{[(ab+b)+bc]+[c]\} \Theta 3$$

$$\{[a]+[ab+b]\}+\{[ac]+[c]\} = \{[a]+[ac]\}+\{[b+bc]+[c]\} \qquad \Theta 3$$

$$\{a+b\}+\{c\} = \{a\}+\{b+c\} \qquad \Theta 3$$

$$(a+b)+c = a+(b+c)$$

Λόγω του Θεωρήματος αυτού μπορούμε να γράφουμε τα αθροίσματα μεταξύ τους χωρίς παρενθέσεις: (a+b)+c = a+(b+c) = a+b+c.

ΘΕΩΡΗΜΑ 4β

Προσεταιριστική ιδιότητα γινομένου:

(a.b).c =a.(b.c)

Ισχύει:

$$[a.(b.c)]+[(a.b).c] = [(a.b).c)]+[a.(b.c)] \qquad A2$$

$$\{[a(bc)]+(ab)\}\{[a(bc)]+c\} = \{[(ab)c]+a\}\{[(ab)c]+(bc)\} \qquad A3$$

$$\{a[(bc)+b]\}\{[a+c][(bc)+c]\} = \{[(ab)+a][a+c]\}\{[(ab)+b]c\} \qquad A3$$

$$\{a[b]\}\{[a+c][c]\} = \{[a][a+c]\}\{[b]c\} \qquad \Theta 3$$

$$\{ab\}\{ac+c\} = \{a+ac\}\{bc\} \qquad \Theta 3$$

$$\{ab\}\{c\} = \{a\}\{bc\} \qquad \Theta 3$$

$$(ab)c = a(bc)$$

Λόγω του Θεωρήματος αυτού μπορούμε να γράφουμε τα γινόμενα μεταξύ τους χωρίς παρενθέσεις: (a.b).c = a.(b.c) = a.b.c.

ΘΕΩΡΗΜΑ 5

Το στοιχείο a' είναι το μοναδικό αντίστροφο του a:

α) Θα αποδείξουμε πρώτα ότι το a' είναι μοναδικό.

Έστω ότι υπάρχουν δύο αντίστροφα στοιχεία για το a, τα $a_1'=a$ και $a_2'=a$, θα αποδείξουμε ότι ταυτίζονται. Πρέπει: $a+a_1'=1$, $a+a_2'=1$ και $a.a_1'=0$, $a.a_2'=0$.

Είναι $a_2' = 1.a_2'$ A4

$= (a+a_1').a_2'$ A5

$= a.a_2'+a_1'.a_2'$ A3

$= 0+a_1'.a_2'$ A5

$= a.a_1'+a_1'.a_2'$ A5

$= a_1'.(a+a_2')$ A3

$= a_1'$

β) Θα αποδείξουμε ότι το αντίστροφο του a' είναι το a, δηλ. (a')'=a.

Εξ ορισμού a'=(a)', άρα το a είναι το μοναδικό αντίστροφο του a'.

ΘΕΩΡΗΜΑ 6

Θεωρήματα **de Morgan**:

α) (a+b)' = a'.b', και **β) (ab)' = a'+b'**

Το θεώρημα 6α) μας λέει ότι το (a'.b') είναι το αντίστροφο του (a+b). Αρκεί επομένως να αποδειχθεί ότι τα στοιχεία (a+b) και (a'.b') πληρούν τα αξιώματα Α5:

(a+b)+a'.b' = 1 και (a+b).(a'.b') = 0

Είναι:

(a+b)+a'.b'= [(a+b)+a'].[(a+b)+b'] και (a+b).(a'.b') = a.a'.b'+b.a'.b'

= (1+b).(a+1) = a.a'.b'+a'.b.b'

= 1 = 0

Το θεώρημα 6β) αποδεικνύεται ομοίως, εάν στη θέση των a και b τοποθετήσουμε τα a' και b'.

Τα θεωρήματα του de Morgan έχουν τη σημαντική ιδιότητα να μας επιτρέπουν να μετατρέπουμε μία παράσταση διατυπωμένη σαν άθροισμα κάποιων όρων σε παράσταση γινομένου αυτών των όρων και αντίστροφα.

ΘΕΩΡΗΜΑ 7

Θεώρημα **Απορρόφησης**.

a+a'.b = a+b, **a.(a'+b) = a.b**

Είναι:

a+b = 1.(a+b)	A4	a.b = 0+a.b	A4
= (a+a').(a+b)	A5	= a.a'+a.b	A5
= a+a'.b	A3	= a(a'+b)	A3

Τα Θεωρήματα Θ1 – Θ7 λόγω της χρησιμότητάς τους συνοψίζονται στον Πίνακα 2.2.

Πίνακας 2.4 Βασικά Θεωρήματα της Άλγεβρας Boole.

Θ1.	$a+a = a$	$a.a = a$
Θ2.	$a+1 = 1$	$a.0 = 0$
Θ3.	$a+(a.b) = a$	$a.(a+b) = a$
Θ4.	$(a+b)+c = a+(b+c)$	$(a.b).c = a.(b.c)$
Θ5.	το a' είναι μοναδικό	$(a')' = a$
Θ6.	$(a+b)' = a'.b'$	$(ab)' = a'+b'$
Θ7.	$a+a'.\beta = a+b$	$a.(a'+\beta) = a.b$

Τα Θεωρήματα Θ1 – Θ7, αν και δεν είναι τα μοναδικά, αποτελούν βασικά θεωρήματα αυτής της άλγεβρας και η εξοικείωση με αυτά είναι πολύ σημαντική.

ΠΑΡΑΔΕΙΓΜΑ 2.1

Θα αποδείξουμε τη σχέση: ab+a'c+bc = ab+a'c.

Απάντηση:

ab+a'c = (ab+**abc**)+(a'c+**a'bc**) Θ3

= ab+a'c+abc+a'bc A2

= ab+a'c+(a+a')bc A3

= ab+a'c+1.bc A5

= ab+a'c+bc A4

ΠΑΡΑΔΕΙΓΜΑ 2.2

Θα απλοποιήσουμε την παράσταση:

F = (x'+xyz')+(x'+xyz')(x+x'y'z)

Απάντηση:

F = (x'+xyz')+(x'+xyz')(x+x'y'z)

= (x'+xyz').[1+(x+x'y'z)] Θ3

= x'+xyz' Θ2

= x'+yz' Θ7

ΠΑΡΑΔΕΙΓΜΑ 2.3

Θα αποδείξουμε ότι: (abcd)' = a'+b'+c'+d'

Απάντηση:

$$a'+b'+c'+d' = (a'+b')+(c'+d')$$
$$= (ab)'+(cd)' \qquad Θ6β$$
$$= [(ab).(cd)]' \qquad Θ6β$$
$$= (abcd)'$$

ΠΑΡΑΔΕΙΓΜΑ 2.4

Θα βρούμε το συμπλήρωμα της: F = xy+wxyz'+x'y

Απάντηση:

$$F' = (xy+wxyz'+x'y)'$$
$$= ([xy+wxyz']+[x'y])'$$
$$= [xy+wxyz']'.[x'y]' \qquad Θ6α$$
$$= (xy)'.(wxyz')'.[x'y]' \qquad Θ6α$$
$$= (x'+y').(w'+x'+y'+z).[x+y'] \qquad Θ6β$$
$$= (x'+y').[x+y'].(w'+x'+y'+z) \qquad A2$$
$$= [xx'+x'y'+xy'+y'y'].(w'+x'+y'+z) \qquad A3$$
$$= [0+x'y'+xy'+y'].(w'+x'+y'+z) \qquad A5,Θ1$$
$$= [(x'+x+1)y'].(w'+x'+y'+z) \qquad Θ3$$
$$= [y'].(w'+x'+y'+z) \qquad Θ2$$
$$= (w'y'+x'y'+y'y'+y'z) \qquad A3$$
$$= (w'+x'+1+z).y' \qquad Θ1,A3$$
$$= y' \qquad Θ2$$

ΠΑΡΑΔΕΙΓΜΑ 2.5

Θα αποδείξουμε ότι:

$$x'y' + x'z + xz' = y'z' + x'z + xz'$$

Απάντηση: Στο παράδειγμα 2.1 αποδείξαμε ένα πολύ σημαντικό αποτέλεσμα, ότι:

$$ab+a'c = ab+a'c+bc$$

Θα εφαρμόσουμε την ιδιότητα αυτή σε καθένα μέλος χωριστά και θα προχωρούμε μέχρι να καταλήξουμε σε ταυτόσημες παραστάσεις. Θα εφαρμόσουμε την ιδιότητα

επιδιώκοντας να σχηματίσουμε τον όρο y'z', που λείπει από το πρώτο μέλος, και τον όρο x'y', που λείπει από το δεύτερο μέλος:

$$x'y'+x'z+xz' \qquad y'z'+x'z+xz'$$

$$[x'y'+xz']+x'z \qquad [y'z'+x'z]+xz'$$

$$[x'y'+xz'+y'z']+x'z \qquad [y'z'+x'z+x'y']+xz'$$

$$x'y'+xz'+x'z+y'z' \;=\; y'z'+x'z+xz'+x'y'$$

Προσοχή: Ο τρόπος αυτός απόδειξης ισχύει, γιατί οι πράξεις έγιναν χωριστά σε κάθε μέλος και ανεξάρτητα από το άλλο. Χρησιμοποιείται εδώ για λόγους οικονομίας χώρου. Η σωστή, τυπικά, απόδειξη είναι εκείνη κατά την οποία ξεκινάμε από το ένα μέλος και καταλήγουμε στο άλλο:

$$x'y'+x'z+xz' = [x'y'+xz']+x'z$$

$$= \ldots\ldots\ldots$$

$$= x'y'+xz'+x'z+y'z'$$

$$= y'z'+x'z+xz'+x'y'$$

$$= \ldots\ldots\ldots$$

$$= y'z'+x'z+xz'$$

ΠΑΡΑΔΕΙΓΜΑ 2.6

Θα αποδειχθεί η σχέση:

$$xy+x'y'+x'yz = xyz'+x'y'+yz$$

Απάντηση: Θα εφαρμόσουμε τη σχέση του Παραδείγματος 2.1:

ab+a'c = ab+a'c+bc

και θα προχωρήσουμε όπως στο παράδειγμα 2.5:

$$xy+x'y'+x'yz \qquad xyz'+x'y'+yz$$

$$[xy+x'yz]+x'y' \qquad [xyz'+yz]+x'y'$$

$$[xy+x'yz+yz]+x'y' \qquad [xyz'+yz+xy]+x'y'$$

$$[xy+yz]+x'y' \qquad [yz+xy]+x'y'$$

$$xy+yz+x'y' \;=\; yz+xy+x'y'$$

ΠΑΡΑΔΕΙΓΜΑ 2.7

Θα αποδείξουμε ότι:

Εάν x+y = 0, **τότε** x = 0 και y = 0

Απάντηση: x+y = 0 και x+y = 0

$x+y+y' = 0+y'$ $\quad$ $x'+x+y = 0+x'$

$x+1 = y'$ $\quad$ $1+y = x'$ $\quad$ A5

$1 = y'$ $\quad$ $1 = x'$

Επίσης: $x+y = 0$ και $x+y = 0$

$(x+y).y' = 0.y'$ $\quad$ $(x+y).x' = 0.x'$

$xy'+yy' = 0$ $\quad$ $xx'+yx' = 0$ $\quad$ Θ2

$x.1+0 = 0$ $\quad$ $0+y.1 = 0$ (γιατί $y'=1$, $x'=1$)

Επομένως: $x = 0$ και $y = 0$

ΠΑΡΑΔΕΙΓΜΑ 2.8

Θα αποδείξουμε ότι:

Εάν $xy'+x'y = xz'+x'z$, **τότε** $y = z$

Απάντηση: $xy'+x'y = xz'+x'z$ και $xy'+x'y = xz'+x'z$

$(xy'+x'y)x = (xz'+x'z)x$ $\quad$ $(xy'+x'y)x' = (xz'+x'z)x'$

$xy' = xz'$ $\quad$ $x'y = x'z$

$(xy')' = (xz')'$

$x'+y = x'+z$ $\quad$ Θ6β

$(x'+y)x = (x'+z)x$

$xy = xz$ και $x'y = x'z$

Προσθέτουμε κατά μέλη τις δύο σχέσεις:

$xy+x'y = xz+x'z$

$(x+x')y = (x+x')z$

$y = z$

2.2 Δίτιμη Άλγεβρα Boole

Η ονομαζόμενη “άλγεβρα Boole” είναι μια οικογένεια από άλγεβρες. Από αυτήν προκύπτουν συγκεκριμένες άλγεβρες ορίζοντας κατάλληλα τις πράξεις της (συμβατές με τα αξιώματα). Η Δίτιμη Άλγεβρα αποτελεί ένα παράδειγμα άλγεβρας με δύο στοιχεία (δίτιμη) και κατάλληλο ορισμό των πράξεων. Η Δίτιμη Άλγεβρα μας εισάγει στο Λογισμό των Προτάσεων και στην Άλγεβρα των Διακοπτών.

Η αλγεβρική μορφή της παράστασης έχει άμεση επίπτωση στη μορφή του κυκλώματος και επιτρέπει την εύκολη κατασκευή ή απλοποίηση ενός κυκλώματος μέσω των γνώσεών μας από την άλγεβρα.

Η Δίτιμη Άλγεβρα είναι μια ειδική εφαρμογή της Άλγεβρας Boole, στην οποία το σύνολο Β περιέχει μόνο δύο στοιχεία και οι πράξεις έχουν αποκτήσει συγκεκριμένο περιεχόμενο. Εννοείται ότι εξακολουθούν να ισχύουν όλα τα αξιώματα και θεωρήματα της άλγεβρας Boole.

Η *Δίτιμη Άλγεβρα* ορίζεται σαν μια άλγεβρα Boole, στην οποία:

α) το σύνολο Β περιέχει μόνο δύο στοιχεία, τα 0 και 1 δηλ. Β={0,1},

β) οι πράξεις του αθροίσματος "+", του γινομένου "." και της αντιστροφής "–" ορίζονται, σχετικά με το σύνολο Β={0,1}, από τον ακόλουθο πίνακα αντιστοιχίας (Πίνακας 2.3),που ονομάζεται **Πίνακας Αληθείας :**

Πίνακας 2.5 Πίνακας Αληθείας των Πράξεων.

	OR	AND		NOT
α β	α+β	α.β	α	α'
0 0	0	0	0	1
0 1	1	0	1	0
1 0	1	0		
1 1	1	1		

Οι πράξεις που ορίστηκαν στον Πίνακα 2.2 έχουν και μια εναλλακτική ονομασία, που είναι δανεισμένη από το Λογισμό των Προτάσεων, στον οποίο αρχικά εφαρμόστηκε η άλγεβρα Boole. Ονομάζονται (Πίνακας 2.3) αντίστοιχα OR (ή), AND (και) και NOT (όχι). Οι πράξεις OR, AND, NOT, όπως ορίστηκαν από τους πίνακες αληθείας τους, είναι εύκολο να διαπιστωθεί ότι πληρούν τα αξιώματα της άλγεβρας Boole. Οι μεταβλητές σε μια παράσταση της Δίτιμης Άλγεβρας μπορούν να έχουν μόνο δύο δυνατές τιμές, γιατί το σύνολο Β έχει δύο μόνο στοιχεία. Το γεγονός αυτό διευκολύνει πολλές φορές την απόδειξη των θεωρημάτων της Δίτιμης Άλγεβρας, γιατί αρκεί να ελέγξουμε το θεώρημα, απαριθμώντας όλους τους δυνατούς συνδυασμούς τιμών των μεταβλητών, που στην περίπτωση αυτή είναι

πεπερασμένου πλήθους. Στο ακόλουθο παράδειγμα θα αποδείξουμε το θεώρημα του de Morgan (Θ6α, Υποενότητα 2.1.3).

ΠΑΡΑΔΕΙΓΜΑ 2.9

Να αποδειχθεί το θεώρημα του de Morgan για τη Δίτιμη Άλγεβρα:

$$(a+b)' = a'b'$$

Απάντηση: Θα σχηματίσουμε έναν πίνακα που να περιέχει όλους τους συνδυασμούς τιμών των μεταβλητών a και b και για κάθε συνδυασμό θα συγκρίνουμε το αποτέλεσμα που προκύπτει από κάθε μέλος του θεωρήματος:

a b	(a+b)'	a'b'
0 0	1	1
0 1	0	0
1 0	0	0
1 1	0	0

Παρατηρούμε ότι για κάθε σειρά του πίνακα και τα δύο μέλη έχουν την ίδια τιμή, επομένως το θεώρημα αληθεύει. Οι εφαρμογές της Δίτιμης Άλγεβρας είναι πολλές και εξαρτώνται πλέον από το "φυσικό" περιεχόμενο, που δίνουμε τόσο στις πράξεις όσο και στα στοιχεία "0" και "1", δηλ. από το τι μεγέθη ή καταστάσεις θέλουμε να απεικονίζουν.

2.2.1 Δίτιμη Άλγεβρα και λογισμός των προτάσεων.

Η εφαρμογή της Δίτιμης Αλγεβρας στο Λογισμό των Προτάσεων έχει σκοπό την εύρεση της αλήθειας σύνθετων προτάσεων. Σύμφωνα με τη Λογική μια σύνθετη πρόταση σχηματίζεται από απλές προτάσεις συνδεδεμένες μεταξύ τους με τα συνδετικά **και**, **ή**, **όχι**. Έστω δύο προτάσεις Α και Β, ο τελεστής:

1) **OR** (Α **OR** Β) συμβολίζει την αλήθεια της σύνθετης πρότασης Γ=(Α **ή** Β),

2) **AND** (Α **AND** Β) συμβολίζει τη Γ=(Α **και** Β),

3) **NOT** συμβολίζει την αντίθετη πρόταση, **(όχι** Α), και παριστάνεται **NOT** Α.

Σύμφωνα με την Αριστοτελική Λογική, μια πρόταση θα έχει μία από τις εξής δύο καταστάσεις, δηλ. θα είναι ή αληθής (**T**rue) ή ψευδής (**F**alse). Η αλήθεια ή το ψεύδος μιας σύνθετης πρότασης καθορίζεται από την αλήθεια των επιμέρους προτάσεων Α και Β σύμφωνα με τον Πίνακα 2.4:

Πίνακας 2.6 Πίνακας Αληθείας σύνθετων Λογικών Προτάσεων.

A B	OR	AND	A	NOT
F F	F	F	F	T
F T	T	F	T	F
T F	T	F		
T T	T	T		

Εάν η κατάσταση αληθείας μιας πρότασης συμβολιστεί: (α) με το **1,** όταν η πρόταση είναι **αληθής**, και (β) με το **0,** όταν είναι **ψευδής**, τότε ο Πίνακας 2.4 συμπίπτει με τον Πίνακα 2.3, δηλ. η αλήθεια μιας σύνθετης πρότασης ακολουθεί τους νόμους της Δίτιμης Άλγεβρας Boole. Χρησιμοποιώντας τον ανωτέρω συμβολισμό, με το “0” και το “1” και τις πράξεις “+”=OR, “.”=AND και “–”=NOT, είναι δυνατόν η λεκτική περιγραφή μιας σύνθετης πρότασης να μετατραπεί σε μορφή αλγεβρικής παράστασης. Κατόπιν, εκμεταλλευόμενοι τα θεωρήματα της άλγεβρας, η πρόταση μπορεί να ελεγχθεί, να απλοποιηθεί, να βρεθούν ισοδύναμες προτάσεις κ.λπ. με πολύ ευκολότερο τρόπο από ότι στη λεκτική της περιγραφή. Στο παράδειγμα που ακολουθεί θα μετατρέψουμε ένα σύνολο από προτάσεις σε μορφή Δίτιμης Άλγεβρας και κατόπιν, εκμεταλλευόμενοι τα θεωρήματα της άλγεβρας (Υποενότητα 2.1.3.), θα βρούμε ένα σύνολο από απλούστερες, αλλά ισοδύναμες, προτάσεις.

ΠΑΡΑΔΕΙΓΜΑ 2.10

Έστω ότι κάποια επιχείρηση θέτει τους ακόλουθους όρους για την πρόσληψη ενός υπαλλήλου, δηλ. ότι ο υποψήφιος πρέπει να είναι:

1) άνδρας, παντρεμένος και κάτω των 35 ετών ή

2) άνδρας, ανύπανδρος και κάτω των 35 ή

3) γυναίκα, παντρεμένη και κάτω των 35 ή

4) γυναίκα, ανύπανδρη και άνω των 35.

Ζητείται να βρεθούν ισοδύναμοι όροι που να είναι διατυπωμένοι απλούστερα.

Απάντηση: Για να εφαρμοσθεί η άλγεβρα, πρέπει αρχικά κάθε βασική έννοια, που περιέχεται στους όρους αυτούς, να την παραστήσουμε με κάποιο σύμβολο (μεταβλητή). Θα χρησιμοποιήσουμε τον ακόλουθο συμβολισμό:

Άνδρας Α, *Παντρεμένος/η* Π, *Κάτω των 35* Κ

Γυναίκα Α', *Ανύπανδρος/η* Π', *Άνω των 35* Κ'

Τότε, εφαρμόζοντας το συμβολισμό για τα συνδετικά “και”, “ή”, θα μετατραπεί ο κάθε όρος σε αλγεβρική μορφή:

1) Α **AND** Π **AND** Κ 3) Α' **AND** Π **AND** Κ

2) Α **AND** Π' **AND** Κ 4) Α' **AND** Π' **AND** Κ'

Η εκπλήρωση του συνόλου των όρων θα σχηματίζεται από το λογικό άθροισμα “OR” των όρων 1), 2), 3) και 4) και θα συμβολίζεται με τη πρόταση Σ:

Σ = (Α **AND** Π **AND** Κ) **OR** (Α **AND** Π' **AND** Κ) **OR**

(Α' **AND** Π **AND** Κ) **OR** (Α' **AND** Π' **AND** Κ')

Μεταγράφοντας την πρόταση Σ με το συμβολισμό της Δίτιμης Άλγεβρας έχουμε:

Σ = (Α.Π.Κ)+(Α.Π'.Κ)+(Α'.Π.Κ)+(Α'.Π'.Κ')

Σ = Α.Π.Κ+Α.Π'.Κ+Α'.Π.Κ+Α'Π'Κ'

Σ = **Α.Π.Κ**+Α.Π'.Κ+**Α.Π.Κ**+Α'.Π.Κ+Α'Π'Κ' (Θεώρημα Θ1)

Σ = Α.Κ.(Π+Π')+(Α+Α').Π.Κ+Α'Π'Κ' (Αξίωμα Α3)

Σ = Α.Κ.1+1.Π.Κ+Α'Π'Κ' (Αξίωμα Α5)

Σ = Α.Κ+Π.Κ+Α'Π'Κ'

Η απλοποιημένη μορφή της πρότασης Σ σημαίνει ότι οι αρχικοί όροι προσλήψεως απλοποιούνται στους:

1) άνδρας και κάτω των 35 ή

2) παντρεμένος/η και κάτω των 35 ή

3) γυναίκα, ανύπανδρη και άνω των 35.

Έχοντας διατυπώσει το πρόβλημα σε αλγεβρική μορφή, είναι εξίσου εύκολο να διατυπώσουμε το ίδιο πρόβλημα στην αντίθετη μορφή του, δηλ. να βρούμε τους όρους για τους οποίους απορρίπτεται ένας υποψήφιος. Εφόσον η Σ είναι η πρόταση που εκφράζει την πρόσληψη ενός υποψηφίου, τότε Σ' = ΝΟΤ(Σ), δηλ. η αντίθετη πρόταση εκφράζει την απόρριψή του:

Σ' = (Α.Κ+Π.Κ+Α'Π'Κ')'

Σκοπός μας είναι να καταλήξουμε σε μια απλή παράσταση παρόμοιας μορφής με τη Σ.

Μία συνήθης “δομή” λογικής προτάσεως, που θα συναντήσουμε συχνά, είναι ο *συμπερασματικός λόγος*

“**Εάν** … **Τότε** …”

Διατυπωμένος υπό τη μορφή της λογικής προτάσεως

Y= (**Εάν** A **Τότε** B),

όπου A και B είναι προτάσεις, δηλώνει τον ισχυρισμό ότι, όταν αληθεύει η *συνθήκη* A, τότε ισχύει (αληθεύει) το *συμπέρασμα* B. Αυτό που μας ενδιαφέρει σε μια τέτοια πρόταση δεν είναι πότε αληθεύει η A, αλλά πότε αληθεύει η όλη πρόταση (ισχυρισμός) Y. Εάν μελετήσουμε τον πίνακα αληθείας της πρότασης Y (Πίνακας 2.5), παρατηρούμε ότι η μόνη περίπτωση που λέμε ψέματα είναι, όταν αληθεύει η A χωρίς να ισχύει η B. Επομένως η πρόταση Y, σε αλγεβρική μορφή, αληθεύει, όταν:

Y = A'.B' + A'.B + A.B

= A'.(B'+B) + A.B = A' + A.B = (A'+A).(A'+B) = A' + B,

δηλ. η αλήθεια (Y=1) ή το ψεύδος (Y=0) του ισχυρισμού Y= Εάν A Τότε B διατυπώνεται πολύ απλά σε αλγεβρική μορφή:

Y = A' + B (3.1)

Η πρόταση "Εάν…Τότε…, που ονομάσαμε ισχυρισμό, σε πολλά προβλήματα μπορεί να διατυπωθεί σαν *απαίτηση* για να συμβεί κάτι. Εφαρμόζοντας την αλγεβρική της μορφή (σχέση 3.1), έχουμε τη δυνατότητα να απλοποιήσουμε σύνθετες προτάσεις, που περιέχουν τη δομή αυτή. Στο επόμενο παράδειγμα θα επαναλάβουμε το Παράδειγμα 2.10 παραλλαγμένο, ώστε να περιλαμβάνει μια τέτοια δομή.

Πίνακας 2.7 Y = Εάν A Τότε B

A	B	Y
F	F	T
F	T	T
T	F	F
T	T	T

ΠΑΡΑΔΕΙΓΜΑ 2.11

Είναι το πρόβλημα του Παραδείγματος 2.10 με τη διαφορά ότι η πρόταση 1) έχει διατυπωθεί με τη μορφή "Εάν…Τότε…": Έστω ότι κάποια επιχείρηση θέτει τους

ακόλουθους όρους για την πρόσληψη ενός υπαλλήλου, δηλ. ότι ο υποψήφιος πρέπει:

1) **εάν** είναι άνδρας και παντρεμένος, **τότε** πρέπει να είναι κάτω των 35 ετών, ή

2) να είναι άνδρας, ανύπανδρος και κάτω των 35, ή

3) να είναι γυναίκα, παντρεμένη και κάτω των 35 ή

4) να είναι γυναίκα, ανύπανδρη και άνω των 35.

Ζητείται να βρεθεί μια απλούστερη διατύπωση των όρων.

Απάντηση: Ακολουθώντας τα βήματα του Παραδείγματος 2.10, η κάθε πρόταση σε αλγεβρική μορφή είναι:

1) (Α.Π)' + Κ 2) Α.Π'.Κ

3) Α'.Π.Κ 4) Α'.Π'.Κ'

Η εκπλήρωση συνολικά όλων των όρων θα δίνεται από την πρόταση Σ:

Σ = [(Α.Π)' + Κ] + [Α.Π'.Κ] + [Α'.Π.Κ] + [Α'.Π'.Κ']

Σ = [Α'+Π'+Κ] + Α.Π'.Κ + Α'.Π.Κ + Α'.Π'.Κ'

Σ = Α'+Π'+Κ.[1 + Α.Π' + Α'.Π] +Α'Π'.Κ'

Σ = Α'+Π'+Κ + Α'.Π'.Κ'

Σ = Α'.[1 + Π'.Κ'] + Π' + Κ

Σ = Α' + Π' + Κ

Η τελική μορφή της πρότασης Σ δηλώνει ότι οι όροι απλοποιούνται στους:

1) γυναίκα, ή 2) ανύπαντρος/η, ή 3) κάτω των 35.

Επειδή Σ = Α'+Π'+Κ = (Α.Π)'+Κ μια εναλλακτική διατύπωση είναι:

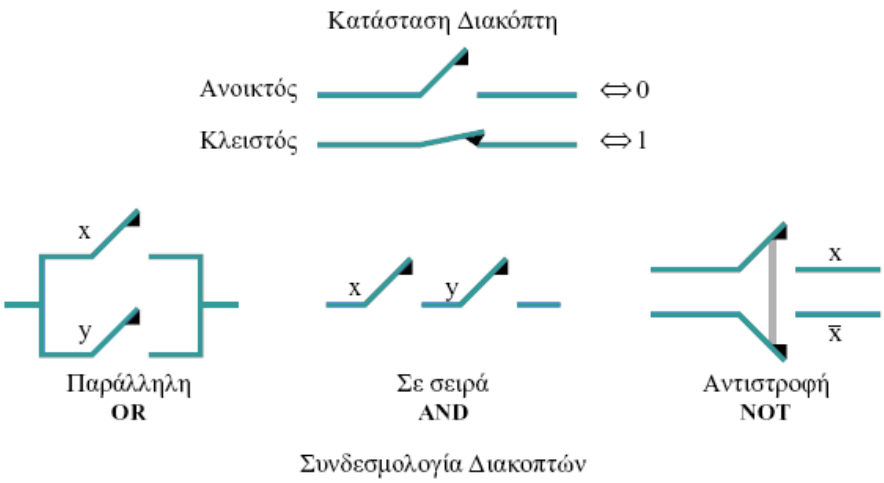
Εάν Άνδρας και Παντρεμένος, **Τότε** να είναι Κάτω των 35.

2.2.2 Δίτιμη Άλγεβρα και κυκλώματα διακοπτών (άλγεβρα των διακοπτών).

Ο Λογισμός των Προτάσεων, που όπως είδαμε αποτελεί μια ειδική εφαρμογή της άλγεβρας Boole, προτάθηκε το 1938 από τον C. Shannon σαν ένας συμβολικός τρόπος για την περιγραφή και μελέτη της λειτουργίας Κυκλωμάτων, που αποτελούνταν από ηλεκτρομαγνητικούς διακόπτες. Ουσιαστικά, ήταν η Δίτιμη Άλγεβρα Boole που εφαρμόστηκε και από τότε η Δίτιμη Άλγεβρα είναι γνωστή και σαν **Άλγεβρα των Διακοπτών**.

Για να καταλάβουμε πώς εφαρμόζεται η Δίτιμη Άλγεβρα σε κυκλώματα με διακόπτες, θα προσπαθήσουμε, να δώσουμε “φυσικό” περιεχόμενο σ' αυτή την

άλγεβρα, δηλ. να αντιστοιχίσουμε τις πράξεις της άλγεβρας και τα δύο στοιχεία της με λειτουργίες και καταστάσεις που χαρακτηρίζουν ένα κύκλωμα με διακόπτες. Ένας διακόπτης (Σχήμα 2.1) έχει δύο καταστάσεις: ή είναι **ανοικτός,** οπότε έχουμε διακοπή του κυκλώματος, ή είναι **κλειστός,** οπότε υπάρχει συνέχεια (αγωγή) στο κύκλωμα. Αντίστοιχα, μεταξύ δύο σημείων Α και Β ενός κυκλώματος, που αποτελείται από διακόπτες, υπάρχει **διακοπή** ή **αγωγή** ανάλογα με την κατάσταση των διακοπτών του. Ένα κύκλωμα που αποτελείται από δύο διακόπτες x και y θα τους έχει συνδεδεμένους είτε παράλληλα (Σχήμα 2.1, **OR**) είτε σε σειρά (Σχήμα 2.1, **AND**).



Σχήμα 2.5 Αντιστοιχία με Διακόπτες.

Ο Πίνακας 2.6 παρουσιάζει τον Πίνακα Αληθείας των διακοπτών του σχήματος 2.1, δηλ. τη συμπεριφορά του κυκλώματος για τις περιπτώσεις που οι διακόπτες x και y είναι **Α**νοικτοί ή **Κ**λειστοί. Για την εφαρμογή της πράξης NOT υπάρχουν ειδικοί διακόπτες, που λειτουργούν αντίστροφα (Σχήμα 2.1 NOT).

Πίνακας 2.8 Πίνακας Αληθείας κυκλώματος Διακοπτών.

.

x y	OR	AND	x	NOT
A A	A	A	A	K
A K	K	A	K	A
K A	K	A		
K K	K	K		

Εάν στον Πίνακα 2.6 η κατάσταση **αγωγής** (**Κ**λειστό), είτε αναφέρεται σε ένα διακόπτη (κλειστός) είτε στη διαδρομή ενός κυκλώματος, συμβολιστεί με **1** (Κ=1) και η κατάσταση **διακοπής** (**Α**νοικτό) συμβολιστεί με **0** (Α=0), τότε από τον Πίνακα 2.6 προκύπτει ο Πίνακας 2.3, που ορίζει τις πράξεις OR, AND, NOT. Συνεπώς ένα κύκλωμα, που αποτελείται (Σχήμα 2.1):

α) από δύο διακόπτες συνδεδεμένους παράλληλα, έχει τον ίδιο πίνακα αληθείας με την πράξη OR και

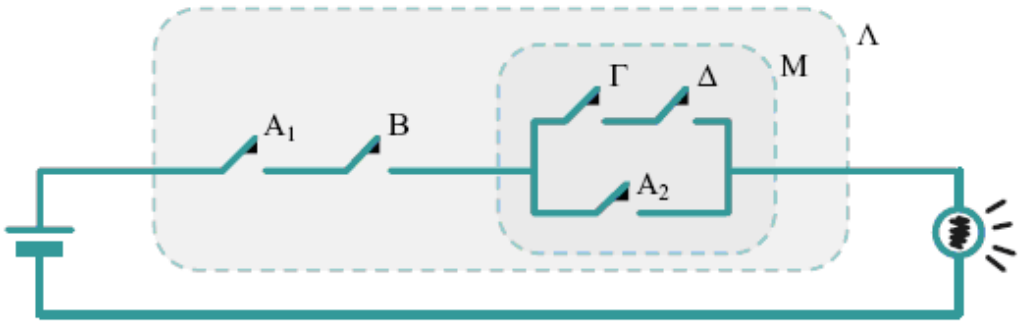
β) από δύο διακόπτες στη σειρά, έχει τον ίδιο πίνακα αληθείας με την πράξη AND.

γ) από ένα διακόπτη αντιστροφής, έχει τον ίδιο πίνακα αληθείας με την πράξη NOT.

Με την αντιστοιχία που ορίσαμε για τις καταστάσεις (Κ=1, Α=0) και τη σύνδεση (παράλληλη = OR, σειρά = AND) των διακοπτών, η λειτουργία ενός σύνθετου κυκλώματος με διακόπτες μπορεί να περιγραφεί από μια λογική παράσταση της Δίτιμης Άλγεβρας ή Άλγεβρας των Διακοπτών. Οποιοδήποτε μηχάνημα, ηλεκτρικό ή άλλου είδους, παρουσιάζει παρόμοια συμπεριφορά, η λειτουργία του μπορεί να παρασταθεί από τους νόμους της Άλγεβρας των Διακοπτών.

ΠΑΡΑΔΕΙΓΜΑ 2.12

Στο Σχήμα 2.2 παρουσιάζεται το σχεδιάγραμμα ενός ηλεκτρικού κυκλώματος. Στο κύκλωμα αυτό υπάρχουν δύο διακόπτες με την ονομασία Α, ο Α1 και ο Α2. Θα θεωρήσουμε ότι, για κάποιο λόγο, οι διακόπτες Α1 και Α2 παίρνουν ταυτόχρονα την ίδια κατάσταση Α (ανοικτός ή κλειστός), δηλ. όσον αφορά στις καταστάσεις τους: Α1=Α2=Α. Για να ανάψει το λαμπάκι, πρέπει να υπάρχει αγωγή ρεύματος, δηλ. να είναι κλειστοί κάποιοι από τους διακόπτες του κυκλώματος. Θα χρησιμοποιήσουμε την Άλγεβρα των Διακοπτών, για να μελετήσουμε το κύκλωμα, δηλ. να βρούμε (α) ποιοι συνδυασμοί διακοπτών ανάβουν το λαμπάκι και (β) εάν μπορεί να απλοποιηθεί το κύκλωμα.



Σχήμα 2.6 Κύκλωμα με διακόπτες.

Απάντηση :

α) Θα παραστήσουμε με Α, Β, Γ, Δ την κατάσταση του αντίστοιχου διακόπτη. Το τμήμα Μ του κυκλώματος περιλαμβάνει τους διακόπτες Γ, Δ, Α2 και το τμήμα Λ περιλαμβάνει όλους τους διακόπτες. Παρατηρούμε ότι οι διακόπτες Α1 και Β είναι συνδεδεμένοι στη σειρά, επομένως το αποτέλεσμά τους θα αντιστοιχεί στη σύνδεση που ονομάσαμε **AND**, δηλ. η συμπεριφορά αυτού του ζεύγους των διακοπτών θα είναι: Α1 **AND** Β. Παρατηρούμε τώρα ότι το ζεύγος (Α1, Β) είναι στη σειρά με το τμήμα Μ, επομένως η συμπεριφορά του τμήματος Λ θα είναι:

Λ = (Α1 **AND** Β) **AND** Μ.

Το τμήμα Μ αποτελείται από την ομάδα των διακοπτών Γ και Δ, που είναι στη σειρά, δηλ. Γ **AND** Δ, και από το διακόπτη Α2, που είναι παράλληλα στην ομάδα (Γ, Δ). Επομένως:

Μ = (Γ **AND** Δ) **OR** Α2

Αντικαθιστάμε το Μ στο Λ και έχουμε:

Λ = (Α1 **AND** Β) **AND** ((Γ **AND** Δ) **OR** Α2).

Επειδή από πλευράς καταστάσεων είναι Α1=Α2=Α, η συμπεριφορά του κυκλώματος Λ θα είναι:

Λ = (Α **AND** Β) **AND** ((Γ **AND** Δ) **OR** Α).

Μετά την αντικατάσταση των **AND** και **OR** με το συμβολισμό της Δίτιμης Άλγεβρας, η Λ γίνεται:

Λ = (Α.Β).((Γ.Δ)+Α)

= (Α.Β).(Γ.Δ+Α) = (Α.Β).Γ.Δ+(Α.Β).Α = Α.Β.Γ.Δ+Α.Β.Α

= Α.Β.Γ.Δ+Α.Β

Η παράσταση αυτή μας λέει ότι, για να άγει το κύκλωμα Λ και να ανάβει το λαμπάκι, πρέπει: ή και οι πέντε διακόπτες Α1, Β, Γ, Δ, Α2 να είναι κλειστοί ή μόνο οι Α1, Β, Α2. Υπενθυμίζουμε ότι Α1=Α2=Α.

β) Η παράσταση του κυκλώματος Λ, στην οποία καταλήξαμε, απλοποιείται:

Λ = Α.Β.Γ.Δ+Α.Β

= Α.Β.(Γ.Δ+1) = Α.Β.1= Α.Β,

δηλ. μας αρκούν μόνο δύο διακόπτες στη σειρά, για να κάνουμε την ίδια δουλειά.

2.3 Λογικές συναρτήσεις.

Μετά την εφαρμογή της Δίτιμης Άλγεβρας στο Λογισμό των Προτάσεων και στα Κυκλώματα Διακοπτών, μέσω των κατάλληλων απεικονίσεων στα στοιχεία αυτής της άλγεβρας, ολόκληρη η λειτουργία ενός πολύπλοκου κυκλώματος ή η λογική μιας σύνθετης πρότασης μπορεί να απεικονισθεί με μία Λογική Συνάρτηση. Κατόπιν η μελέτη της συμπεριφοράς του κυκλώματος ή της πρότασης ανάγεται στη μελέτη των ιδιοτήτων της Λογικής Συνάρτησής του.

Η σχεδίαση ενός ψηφιακού κυκλώματος ανάγεται σε διαδοχικούς μετασχηματισμούς της Λογικής του Συνάρτησης. Οι λογικές παραστάσεις είναι συνήθως μακροσκελείς και γι' αυτό χρειάζεται προσοχή στην εκτέλεση των πράξεων. Ένα συνηθισμένο λάθος είναι η παράλειψη, εκ παραδρομής, του συμβόλου της αντιστροφής σε κάποια μεταβλητή.

Ονομάζεται **Καρτεσιανό Γινόμενο** δύο συνόλων Α και Β και συμβολίζεται ΑxΒ το σύνολο που περιέχει σαν στοιχεία του όλα τα διατεταγμένα ζεύγη (x, y), όπου x∈A και y∈B, δηλ. AxB = {(x, y) | x∈A και y∈B}. Π.χ. εάν έχουμε τα σύνολα $A=\{a_1, \ldots, a_m\}$ και $B=\{b_1, \ldots, b_n\}$, τότε το καρτεσιανό γινόμενό τους είναι το σύνολο που έχει σαν στοιχεία τα (a_i, b_j), i=1,…, m, j=1,…, n, δηλ.:

$$A \times B = \{(a_1,b_1), (a_1,b_2), \ldots, (a_m,b_1), \ldots, (a_m,b_n)\}$$

Το καρτεσιανό γινόμενο ενός συνόλου Α επί τον εαυτό του συμβολίζεται για συντομία ως Α2 και γενικώς An = AxAx…xA, n φορές.

<u>Λογική Συνάρτηση</u>

Λογική συνάρτηση f με μεταβλητές $x_1, x_2, \ldots, x_n$:

$$y = f(x_1, x_2, \ldots, x_n)$$

ορίζεται στη Δίτιμη Αλγεβρα η μονοσήμαντη απεικόνιση:

$$f: B_n \rightarrow B \text{ όπου } B=\{0,1\},$$

δηλ. μια σχέση που αντιστοιχεί μονοσήμαντα κάθε συνδυασμό τιμών των μεταβλητών $x_1, x_2, \ldots, x_n$ σε κάποιο στοιχείο y του συνόλου B.

Το σύνολο B^n των συνδυασμών τιμών των ανεξάρτητων μεταβλητών ονομάζεται **πεδίο ορισμού** της συνάρτησης, ενώ το σύνολο B στο οποίο απεικονίζονται οι ανεξάρτητες μεταβλητές (αντιστοιχούνται οι συνδυασμοί) ονομάζεται **πεδίο τιμών** της συνάρτησης. Η απεικόνιση, δηλ. η αντιστοίχηση των συνδυασμών τιμών των μεταβλητών στις τιμές της συνάρτησης μπορεί να γίνει με πολλούς τρόπους. Ο απλούστερος τρόπος είναι να σχηματίσουμε έναν πίνακα αντιστοιχίας, που για παραδοσιακούς λόγος ονομάζεται **Πίνακας Αληθείας**. Π.χ. η αντιστοιχία που απεικονίζεται στον Πίνακα 2.7 ορίζει τις δίτιμες συναρτήσεις $f_1(x_1, x_2)$ και $f_2(x_1, x_2)$. Ο Πίνακας Αληθείας που συναντήσαμε για τις πράξεις **OR**, **AND**, **NOT** (Πίνακας 2.3) μπορεί να θεωρηθεί ότι ορίζει τις συναρτήσεις **OR**(α,β), **AND**(α,β), **NOT**(α).

Πίνακας 2.9 Ορισμός των συναρτήσεων $f_1(x_1,x_2)$ και $f_2(x_1,x_2)$.

πεδίο ορισμού x_1 x_2	πεδίο τιμών f_1	πεδίο τιμών f_2
0 0	0	0
0 1	1	0
1 0	1	0
1 1	1	1

Μια μεταβλητή της Δίτιμης άλγεβρας μπορεί να πάρει δύο τιμές (0 ή 1), επομένως το πεδίο ορισμού μιας συνάρτησης n μεταβλητών περιέχει, το πολύ, 2^n στοιχεία, διότι υπάρχουν 2^n διαφορετικοί συνδυασμοί των 0 και 1. Επίσης το πεδίο τιμών της συνάρτησης περιέχει δύο στοιχεία, τα {0, 1}. Τα 2^n στοιχεία του πεδίου ορισμού μιας συνάρτησης n μεταβλητών μπορούν να συνδυαστούν (απεικονιστούν) με $2^{(2n)}$ τρόπους με τα δύο στοιχεία του πεδίου τιμών, επομένως μπορούν να κατασκευαστούν το πολύ $2^{(2n)}$ διαφορετικές συναρτήσεις n μεταβλητών.

2.3.1 Παράσταση Συναρτήσεων.

Όπως είδαμε στην Ενότητα 2.3 ο ορισμός μιας Λογικής Συνάρτησης απαιτεί έναν τρόπο που να μας λέει σε ποιο στοιχείο του συνόλου Β (πεδίο τιμών) αντιστοιχείται ο κάθε συνδυασμός τιμών των μεταβλητών της (πεδίο ορισμού). Υπάρχουν πολλοί τρόποι για να περιγραφεί μια συνάρτηση, δηλ. να ορισθούν οι τιμές της, οι κυριότεροι από τους οποίους δίδονται παρακάτω:

***ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΑΣ* (Truth Table)**

Η αντιστοιχία μεταξύ των τιμών των ανεξαρτήτων μεταβλητών και των τιμών της συνάρτησης δίνεται υπό μορφή πίνακα που ονομάζεται *Πίνακας Αληθείας* της συνάρτησης. Π.χ. η συνάρτηση w=f(x,y,z) ορίζεται από τον Πίνακα Αληθείας (Πίνακας 2.8):

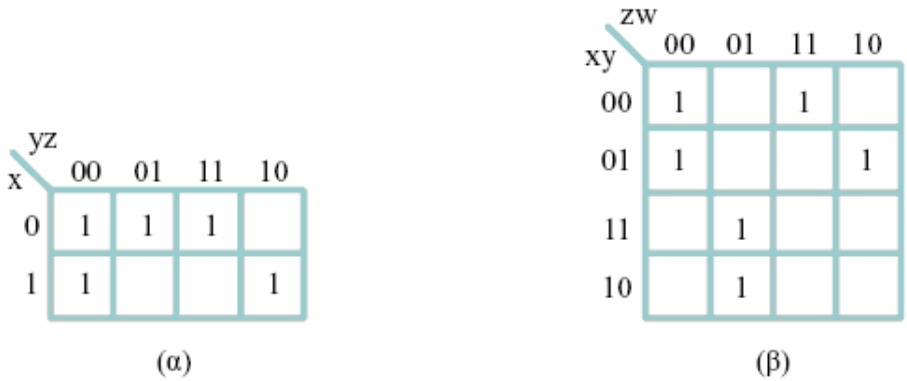
Πίνακας 2.10 Πίνακας Αλήθειας.

α/α	x	y	z	w
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Ο Πίνακας Αληθείας μιας συνάρτησης έχει 2^n σειρές, όσοι είναι οι συνδυασμοί τιμών των n μεταβλητών της συνάρτησης. Όταν θέλουμε να αναφερθούμε σε μια συγκεκριμένη σειρά του Πίνακα, είναι πολλές φορές βολικό να το κάνουμε με τη βοήθεια κάποιου αύξοντα αριθμού. Ένας συστηματικός τρόπος που ακολουθείται είναι να θεωρήσουμε τις τιμές των μεταβλητών της συνάρτησης σαν τα bits ενός δυαδικού αριθμού και να αριθμήσουμε τις σειρές του Πίνακα Αληθείας γράφοντας τον αντίστοιχο δεκαδικό αριθμό (α/α, Πίνακας 2.8). Η αρίθμηση θα είναι από 0 έως 2^n–1.

ΠΙΝΑΚΑΣ KARNAUGH

Είναι ένας δισδιάστατος πίνακας αληθείας. Οι μεταβλητές χωρίζονται σε δύο ίσες κατά το δυνατόν ομάδες και σχηματίζεται ο πίνακας τοποθετώντας οριζοντίως τους συνδυασμούς τιμών της μιας ομάδας, κατακορύφως της άλλης και στην διασταύρωσή τους τις αντίστοιχες τιμές της συνάρτησης. Π.χ. για τη συνάρτηση του προηγούμενου παραδείγματος με Πίνακα Αληθείας τον Πίνακα 2.8, ο χάρτης Karnaugh παρουσιάζεται στο Σχήμα 2.3α. Οι κενές θέσεις θεωρείται ότι περιέχουν το 0. Χαρακτηριστικό του χάρτη Karnaugh είναι ότι η διάταξη των τιμών 00, 01, 11, 10 των μεταβλητών στο χάρτη δεν ακολουθεί την κανονική αύξουσα σειρά, αλλά ακολουθεί τη σειρά ενός ανακλαστικού κώδικα. Ο χάρτης Karnaugh μιας συνάρτησης τεσσάρων μεταβλητών u=f(x,y,z,w) έχει τη μορφή του Σχήματος 2.3β, όπου τόσο οι στήλες όσο και οι γραμμές του πίνακα ακολουθούν τη σειρά ενός ανακλαστικού κώδικα.



Σχήμα 2.7 Χάρτης Karnaugh (α) τριών και (β) τεσσάρων μεταβλητών.

ΛΟΓΙΚΗ ΠΑΡΑΣΤΑΣΗ

Βρίσκεται μια λογική παράσταση με παραμέτρους τις μεταβλητές της συνάρτησης τέτοια ώστε για κάθε συνδυασμό τιμών των μεταβλητών, εάν οι τιμές αυτές αντικατασταθούν στην παράσταση και εκτελεστούν οι πράξεις, να παίρνει την τιμή της συνάρτησης. Π.χ. η συνάρτηση w=f(x,y,z) του προηγουμένου παραδείγματος, που έχει Πίνακα Αληθείας τον Πίνακα 2.8, μπορεί να παρασταθεί σε αλγεβρική μορφή με την παράσταση:

$$f(x,y,z) = x'y' + x'z + xz'$$

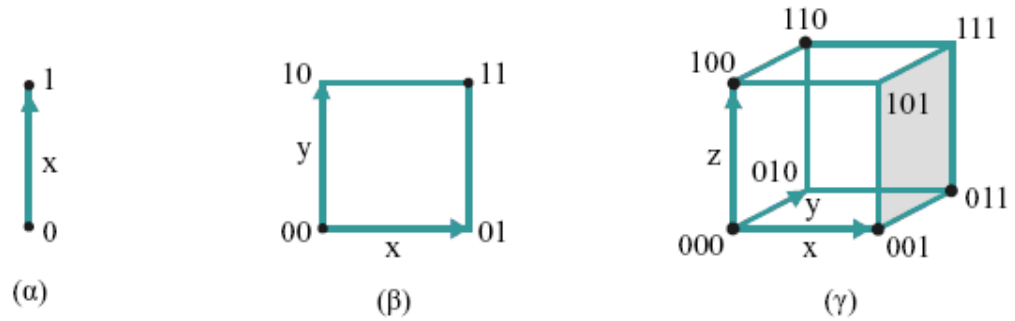
ή με την παράσταση:

$$f(x,y,z) = y'z' + x'z + xz'$$

Οι δύο μορφές παράστασης της συνάρτησης μπορούν να επαληθευτούν συγκρίνοντας τις τιμές τους, εξαντλώντας όλους τους συνδυασμούς τιμών, με αυτές του πίνακα αληθείας του παραδείγματος (Πίνακας 2.8). Ας σημειωθεί ότι η ίδια συνάρτηση μπορεί να αντιπροσωπεύεται από πολλές λογικές παραστάσεις.

N-ΚΥΒΟΣ

Είναι ένας τρόπος γραφικής παράστασης της συνάρτησης. Για μια συνάρτηση N μεταβλητών σχηματίζεται ένας κύβος N διαστάσεων (N– κύβος). Κάθε κορυφή του κύβου αντιστοιχεί σε κάποιο συνδυασμό τιμών των μεταβλητών (Σχήμα 2.4). Σημειώνονται με κύκλο οι κορυφές για τις οποίες η συνάρτηση παίρνει την τιμή 1, ενώ στις μη σημειωμένες κορυφές θεωρείται ότι παίρνει την τιμή 0. Π.χ. η συνάρτηση του παραδείγματος του Πίνακα 2.8 παριστάνεται υπό μορφή τρισδιάστατου κύβου στο Σχήμα 2.4γ:



Σχήμα 2.8 N-κύβος συνάρτησης: (α) μιας (β) δύο και (γ) τριών μεταβλητών.

Όταν οι μεταβλητές είναι περισσότερες από τρεις, η παράσταση του κύβου είναι πολύ δύσκολη. Εδώ θα παρουσιάσουμε, σύντομα, την έννοια του υποκύβου (subcube), που αναφέρεται συχνά στην ορολογία της ψηφιακής σχεδίασης. Δύο "γειτονικές" κορυφές ή γενικώς 2^k "γειτονικές" κορυφές ενός N–κύβου ($0 \le k < N$) λέμε ότι σχηματίζουν έναν υποκύβο. Π.χ. στο Σχήμα 2.4γ το ζεύγος των κορυφών {000, 001} σχηματίζει ένα μονοδιάστατο υποκύβο εντός του οποίου η συνάρτηση έχει την τιμή "1", επίσης η τετράδα {001, 011, 111, 101} (σκιασμένη πλευρά) σχηματίζει ένα δισδιάστατο υποκύβο.

ΑΘΡΟΙΣΜΑ ΟΡΩΝ

Ο τρόπος αυτός χρησιμοποιείται κυρίως για τυπογραφικούς λόγους. Εάν θεωρήσουμε τις τιμές των μεταβλητών της συνάρτησης σαν τα bits ενός δυαδικού αριθμού, όπως εξηγήθηκε στην παράγραφο για τον Πίνακα Αληθείας, τότε μπορούμε, για συντομία, στη θέση των τιμών των μεταβλητών να δηλώνουμε τον αντίστοιχο δεκαδικό αριθμό. Π.χ.

εάν f(0,1,1)=1, τότε γράφουμε f(3)=1,

εάν f(1,0,1)=0, τότε γράφουμε f(5)=0.

Με το σύστημα αυτό μπορούμε να ορίσουμε συνολικά μια δίτιμη συνάρτηση, αρκεί να δηλώσουμε τα σημεία στα οποία η συνάρτηση παίρνει την τιμή "1". Συγκεκριμένα, μια συνάρτηση f(x,y,z) παριστάνεται με τον τρόπο αυτό ως εξής:

$$f(x,y,z) = \Sigma(k_1, k_2,\ldots, k_n),$$

όπου k_1, k_2,…, k_n είναι οι αντίστοιχοι δεκαδικοί αριθμοί των συνδυασμών για τους οποίους η συνάρτηση παίρνει την τιμή "1". Π.χ. η συνάρτηση του παραδείγματος του Πίνακα 2.8 παριστάνεται σαν άθροισμα όρων ως εξής: f(x,y,z) = Σ(0,1,3,4,6).

Συνοψίζοντας τους τρόπους παράστασης μιας Λογικής Συνάρτησης, οι κύριοι τρόποι είναι ο Πίνακας Αληθείας και η Λογική Παράσταση. Εάν συγκριθούν οι δύο αυτοί τρόποι μεταξύ τους, προκύπτουν τα εξής πλεονεκτήματα και μειονεκτήματα:

α) Το μέγεθος του Πίνακα Αληθείας αυξάνει εκθετικά με το πλήθος n των μεταβλητών της συνάρτησης, δηλ. περιέχει 2^n σειρές. Συνεπώς, για συναρτήσεις με πολλές μεταβλητές το μέγεθος του πίνακα γίνεται απαγορευτικά μεγάλο. Π.χ. ένα μέτριο VLSI κύκλωμα με 30 μεταβλητές (εισόδους) χρειάζεται έναν πίνακα με 2^{30}_≈ 10^9 σειρές. Δεν ισχύει το ίδιο για τη Λογική Παράσταση (πλεονέκτημα της Λογικής Παράστασης).

β) Ο Πίνακας Αληθείας είναι μοναδικός για κάθε συνάρτηση, ενώ υπάρχουν πολλές (διαφορετικές) Λογικές Παραστάσεις που ανταποκρίνονται στην ίδια συνάρτηση. Συνεπώς, για την ταυτοποίηση δύο συναρτήσεων είναι αρκετό να συγκριθούν οι πίνακες αληθείας τους, για να διαπιστωθεί εάν οι συναρτήσεις ταυτίζονται μεταξύ τους ή όχι, ενώ η απλή σύγκριση των λογικών τους παραστάσεων δεν επαρκεί (μειονέκτημα λογικής παράστασης).

2.3.2 Θεωρήματα Συναρτήσεων.

Εκτός από τα βασικά θεωρήματα της άλγεβρας Boole, που είδαμε στην Υποενότητα 3.2.1, υπάρχουν και θεωρήματα που αφορούν στις Λογικές Συναρτήσεις. Τα κυριότερα από αυτά είναι τα δύο θεωρήματα **Ανάπτυξης Συναρτήσεων** του Shannon και το γενικευμένο **θεώρημα του de Morgan**.

***ΘΕΩΡΗΜΑ ΑΝΑΠΤΥΞΗΣ ΣΥΝΑΡΤΗΣΕΩΝ* (Shannon)**

Πρόκειται για δύο δυικά μεταξύ τους θεωρήματα, τα οποία προτάθηκαν από τον C. Shannon, σύμφωνα με τα οποία κάθε λογική συνάρτηση:

$$y = f(x_1,\ldots,x_k,\ldots,x_n)$$

που δίνεται υπό οποιαδήποτε μορφή, αλγεβρική ή πίνακα αληθείας, μπορεί να αναπτυχθεί ως προς κάποια μεταβλητή της x_k και η παράστασή της να διατυπωθεί με τις εξής δύο μορφές:

α) $f(x_1,\ldots,x_k,\ldots,x_n) = x_k.f(x1,\ldots,1,\ldots,x_n) + .f(x1,\ldots,0,\ldots,x_n)$ (3.2α)

και

β) $f(x_1,\ldots,x_k,\ldots,x_n) = [x_k+ f(x_1,\ldots,0,\ldots,x_n)]. + f(x_1,\ldots,1,\ldots,x_n)]$ (3.2β)

Το θεώρημα (3.2α) μας λέει ότι κάθε συνάρτηση μπορεί να παρασταθεί σαν άθροισμα δύο γινομένων ως προς κάποια μεταβλητή x_k.

ΠΑΡΑΔΕΙΓΜΑ 2.13

Να αναπτυχθεί ως προς τη μεταβλητή x η συνάρτηση f(x,y,z), που ορίζεται από τον Πίνακα Αληθείας του Πίνακα 2.8.

Απάντηση : Η f(x,y,z), κατά το θεώρημα 3.2α, μπορεί να παρασταθεί υπό τη μορφή:

$$f(x,y,z) = x.f(1, y, z) + x'.f(0, y, z),$$

όπου οι συναρτήσεις f(0, y, z) και f(1, y, z) παράγονται από την f και έχουν Πίνακες Αληθείας (βλέπε Πίνακα 2.8, για x=0 και για x=1) αντίστοιχα:

x y z	f(0, y, z)	x y z	f(1, y, z)
0 0 0	1	1 0 0	1
0 0 1	1	1 0 1	0
0 1 0	0	1 1 0	1
0 1 1	1	1 1 1	0

ΠΑΡΑΔΕΙΓΜΑ 2.14

Η συνάρτηση f(x,y,z) = x'y'+x'z+xz' να αναπτυχθεί διαδοχικά ως προς τις μεταβλητές x, y και z.

Απάντηση : Κατά το θεώρημα 3.2α μπορεί να παρασταθεί υπό τη μορφή:

$$f(x,y,z) = x.f(1, y, z) + x'.f(0, y, z),$$

όπου οι συναρτήσεις f0 και f1 είναι:

f(0, y, z) = 0'y'+0'z+0.z' = 1.y'+1.z+0 = y'+z

και f(1, y, z) = 1'y'+1'z+1.z' = 0.y'+0.z+1.z' = z'

Οι νέες συναρτήσεις f(0, y, z) και f(1, y, z) μπορούν να αναλυθούν με τη σειρά τους ως προς τη μεταβλητή y ως εξής:

f(0, y, z) = y.f(0, 1, z) + y'.f(0, 0, z)

και f(1, y, z) = y.f(1, 1, z) + y'.f(1, 0, z),

όπου:

f(0, 0, z) = 0'+z = 1 και f(0, 1, z) = 1'+z = z

f(1, 0, z) = z' και f(1, 1, z) = z'

Επομένως, εφαρμόζοντας διαδοχικά το θεώρημα, η συνάρτηση f(x,y,z) μπορεί να αναλυθεί ως προς περισσότερες από μία μεταβλητές:

f(x,y,z) = x.f(1, y, z) + x'f(0, y, z)

f(x,y,z) = x.[y.f(1, 1, z) + y'.f(1, 0, z)] + x'.[y.f(0, 1, z) + y'f(0, 0, z)]

f(x,y,z) = x.[y.z' + y'.z'] + x'.[y.z + y'.1]

και τελικά:

f(x,y,z) = x.y.z' + x.y'.z' + x'.y.z + x'.y'

Το δεύτερο θεώρημα (3.2β), που είναι δυϊκό του πρώτου, λέει ότι κάθε συνάρτηση μπορεί να γραφεί σαν γινόμενο δύο αθροισμάτων ως προς την

μεταβλητή x_k. Προσοχή πότε τοποθετείται η σταθερά “1” και πότε η σταθερά “0” στη συνάρτηση f.

ΓΕΝΙΚΕΥΜΕΝΟ ΘΕΩΡΗΜΑ ΤΟΥ DE MORGAN

Ορίζεται σαν **συμπληρωματική** ή **αντίστροφη** συνάρτηση f' μιας λογικής συνάρτησης f η συνάρτηση που παίρνει την τιμή 0, όπου η f παίρνει την τιμή 1 και αντίστροφα. Στην Υποενότητα 2.1.3. γνωρίσαμε το θεώρημα του de Morgan

(Θεώρημα 6), που δίνει το αντίστροφο του αθροίσματος και του γινομένου δύο μεταβλητών. Το θεώρημα αυτό μπορεί να γενικευθεί για οποιαδήποτε παράσταση. Σύμφωνα με το *γενικευμένο θεώρημα του de Morgan* η αντίστροφη συνάρτηση f' μπορεί να παραχθεί από τη λογική παράσταση της f κατά τον εξής τρόπο:

f'(x1,…,xn, +, ., 0, 1) = f(,…, , . , +, 1, 0)

Ο συμβολισμός αυτός σημαίνει ότι στη θέση:

α) της κάθε μεταβλητής θα τοποθετηθεί η συμπληρωματική της,

β) της πράξης "+" θα τοποθετηθεί το "." και αντίστροφα,

γ) της σταθεράς, όπου υπάρχει, "0" το "1" και αντίστροφα.

Εάν γίνουν αυτές οι ενέργειες, η νέα παράσταση θα αντιπροσωπεύει την f'. Το θεώρημα αυτό αφορά όχι μόνον στις συναρτήσεις αλλά και κάθε λογική παράσταση και αποτελεί γενίκευση, για οσεσδήποτε μεταβλητές, του θεωρήματος του de Morgan (Θ6), που παρουσιάστηκε στην Υποενότητα 2.1.3. για δύο μόνο μεταβλητές.

ΠΑΡΑΔΕΙΓΜΑ 2.15

Στο Παράδειγμα 2.14 είδαμε ότι η συνάρτηση f(x,y,z), που ορίζεται από τον Πίνακα 2.8, μπορεί να παρασταθεί τελικά υπό την αλγεβρική μορφή:

f(x,y,z) = [x.y.z'] + [x.y'.z'] + [x'.y.z] + [x'.y']

Οι αγκύλες τοποθετήθηκαν εδώ για να δηλωθεί η προτεραιότητα των γινομένων έναντι των αθροισμάτων. Με τη βοήθεια του γενικευμένου θεωρήματος του de Morgan θα βρούμε την αντίστροφη συνάρτηση της f(x,y,z), την f'(x,y,z).

Απάντηση : Σύμφωνα με το θεώρημα η παράσταση της f'(x,y,z) θα προκύψει εάν στην παράσταση της f(x,y,z) αντιστρέψουμε τις μεταβλητές και τις πράξεις:

f'(x,y,z) = [(x)'+(y)'+(z')'] . [(x)'+(y')'+(z')'] . [(x')'+(y)'+(z)'] . [(x')'+(y')']

Εκτελώντας τις πράξεις παίρνουμε

f'(x,y,z) = ([x'+y'+z] . [x'+y+z]) . ([x+y'+z'] . [x+y])

= (x'x'+x'y+x'z+x'y'+y'y+y'z+x'z+yz+z) . (xx+xy'+xz'+xy+y'y+yz')

= (x'+x'y+x'z+x'y'+0+y'z+x'z+yz+z) . (x+xy'+xz'+xy+0+yz')

= (x'.[1+y+z+y']+z[y'+x'+y+1]) . (x[1+y'+z'+y]+yz')

= (x'+z) . (x+yz')

= x'yz'+xz

Η f' είναι η αντίστροφη συνάρτηση της f. Η αντιστροφή μπορεί να επαληθευτεί, εάν κατασκευάσουμε τον Πίνακα Αληθείας της f' από την αλγεβρική παράστασή της και τον συγκρίνουμε με αυτόν της f (Πίνακας 2.8):

x	y	z	f	f' = x'yz'+xz
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

Παρατηρούμε ότι στις θέσεις του πίνακα, όπου f=1, έχουμε f'=0 και αντίστροφα.

2.4 Συναρτήσεις Ελαχίστου και Μεγίστου όρου.

Οι συναρτήσεις *Ελαχίστου* και *Μεγίστου* όρου αποτελούν δύο ειδικές κατηγορίες απλών συναρτήσεων, που είναι χρήσιμες για τον ορισμό και την παράσταση σύνθετων συναρτήσεων.

Ο ορισμός των συναρτήσεων Ελαχίστου και Μεγίστου όρου διευκολύνεται, όταν χρησιμοποιήσουμε τον ακόλουθο συμβολισμό με *εκθέτες,* για να δηλώσουμε την αντιστροφή ή όχι μιας μεταβλητής x:

$$x^0 = \overline{x} \text{ και } x^1 = x.$$

Παρακάτω παρουσιάζονται οι δύο κατηγορίες συναρτήσεων:

***ΕΛΑΧΙΣΤΟΙ ΟΡΟΙ* (minterms)**

Μία συνάρτηση $m(x_1,x_2,\ldots,x_n)$ ονομάζεται συνάρτηση **ελαχίστου όρου** (minterm), όταν ορίζεται από μια λογική παράσταση *γινομένου* της μορφής:

$$m(x_1, x_2, \ldots, x_n) = x_1^{a_1} \cdot x_2^{a_2} \cdot \ldots x_n^{a_n}$$

όπου τα a_i (i=1, …, n) είναι δυαδικές σταθερές, a_i={0,1} και η συνάρτηση παριστάνεται με μικρό “m”.

Συνολικά υπάρχουν 2^n διαφορετικές συναρτήσεις ελαχίστου όρου των n μεταβλητών, δηλ. όσοι οι συνδυασμοί τιμών των σταθερών a_i. Εάν k είναι ο δεκαδικός αριθμός που αντιστοιχεί σε μια δοθείσα διάταξη τιμών των σταθερών a_i:

$$k_{10} = a_1a_2\ldots a_n,$$

τότε ο αντίστοιχος *ελάχιστος όρος* παριστάνεται με m_k.

ΠΑΡΑΔΕΙΓΜΑ 2.16

Όλες οι συναρτήσεις ελαχίστου όρου των δύο μεταβλητών m(x,y) είναι οι εξής:

0. $m_0(x,y) = x^0y^0 = x'y'$ διότι $a_1a_2 = 00 \Rightarrow 0$,
1. $m_1(x,y) = x^0y^1 = x'y$ $01 \Rightarrow 1$,
2. $m_2(x,y) = x^1y^0 = x\,y'$ $10 \Rightarrow 2$,
3. $m_3(x,y) = x^1y^1 = x\,y$ $11 \Rightarrow 3$.

Ο Πίνακας Αληθείας τους είναι ο ακόλουθος:

		minterms			
		m_0	m_1	m_2	m_3
α/α	x y	x'y'	x'y	x y'	x y
0	0 0	**1**	0	0	0
1	0 1	0	**1**	0	0
2	1 0	0	0	**1**	0
3	1 1	0	0	0	**1**

Θεμελιώδης ιδιότητα των ελαχίστων όρων είναι ότι για κάθε συνάρτηση ελαχίστου όρου *υπάρχει μόνο ένας συνδυασμός τιμών των ανεξαρτήτων μεταβλητών* x_1, x_2, …, x_n, *για τον οποίο η συνάρτηση παίρνει την τιμή 1 και για οποιονδήποτε άλλο συνδυασμό η συνάρτηση παίρνει την τιμή 0*. Ο συνδυασμός αυτός συμπίπτει με τον αύξοντα αριθμό της συνάρτησης, δηλ. για τη συνάρτηση m_k είναι:

$$m_k(k)=0 \text{ και } m_k(j)=1,\ j \neq k$$

Στο Παράδειγμα 2.16 είναι $m_0(0,0)=1$, ενώ $m_0(0,1)=0$ κ.λπ.

***ΜΕΓΙΣΤΟΙ ΟΡΟΙ* (Maxterms)**

Μία συνάρτηση $M(x_1,x_2,\ldots,x_n)$ ονομάζεται συνάρτηση **μεγίστου όρου** (Maxterm), όταν ορίζεται από μια λογική παράσταση *αθροίσματος* της μορφής:

$$M(x_1,x_2,\ldots,x_n) = x_1^{\bar{a}_1} + x_2^{\bar{a}_2} + \ldots + x_n^{\bar{a}_n}$$

όπου τα a_i (i=1, …, n) είναι δυαδικές σταθερές, $a_i=\{0,1\}$ και η συνάρτηση παριστάνεται με κεφαλαίο "M".

Συνολικά υπάρχουν 2^n διαφορετικές συναρτήσεις ελαχίστου όρου των n μεταβλητών. Αντίστοιχα με τους ελαχίστους όρους, εάν k είναι ο δεκαδικός αριθμός που αντιστοιχεί σε μια δοθείσα διάταξη τιμών των σταθερών a_i δηλ.

$$k_{10} = a_1\, a_2 \ldots a_n,$$

τότε ο αντίστοιχος *μέγιστος όρος* παριστάνεται με M_k. Προσοχή: οι συναρτήσεις μεγίστου όρου (maxterms) παριστάνονται με κεφαλαίο “M”, ενώ οι συναρτήσεις ελαχίστου όρου (minterms) παριστάνονται με μικρό “m”.

ΠΑΡΑΔΕΙΓΜΑ 2.17

Οι συναρτήσεις μεγίστου όρου δύο μεταβλητών M(x,y) είναι οι εξής:

$$0.\ M_0(x,y) = x^{\bar{0}} + y^{\bar{0}} = x + y \quad \text{διότι } a_1a_2 = 00 \Rightarrow 0,$$
$$1.\ M_1(x,y) = x^{\bar{0}} + y^{\bar{1}} = x + y' \quad 01 \Rightarrow 1,$$
$$2.\ M_2(x,y) = x^{\bar{1}} + y^{\bar{0}} = x' + y \quad 10 \Rightarrow 2,$$
$$3.\ M_3(x,y) = x^{\bar{1}} + y^{\bar{1}} = x' + y' \quad 11 \Rightarrow 3.$$

Ο Πίνακας Αληθείας τους είναι:

		Maxterms			
		M_0	M_1	M_2	M_3
α/α	x y	x+y	x+y'	x'+y	x'+y'
0	0 0	**0**	1	1	1
1	0 1	1	**0**	1	1
2	1 0	1	1	**0**	1
3	1 1	1	1	1	**0**

Θεμελιώδης ιδιότητα κάθε συνάρτησης μεγίστου όρου είναι ότι από τους 2^n δυνατούς συνδυασμούς τιμών των ανεξαρτήτων μεταβλητών της *υπάρχει μόνο ένας συνδυασμός τιμών, για τον οποίο η συνάρτηση παίρνει την τιμή 0 και για οποιονδήποτε άλλο συνδυασμό η συνάρτηση παίρνει την τιμή 1*. Ο συνδυασμός αυτός συμπίπτει με τον αύξοντα αριθμό της συνάρτησης, δηλ. για τη συνάρτηση M_k είναι:

$$M_k\,(k)=0 \text{ και } M_k(j)=1,\ j \neq k$$

Στο Παράδειγμα 2.17 η συνάρτηση $M_0(0,0)=0$, ενώ $M_0(0,1)=1$ κ.λπ.

ΘΕΩΡΗΜΑΤΑ ΣΥΝΑΡΤΗΣΕΩΝ ΕΛΑΧΙΣΤΟΥ ΚΑΙ ΜΕΓΙΣΤΟΥ ΟΡΟΥ

Στα ακόλουθα θεωρήματα, για συναρτήσεις n μεταβλητών, οι δείκτες i και j καλύπτουν τις τιμές $\{0, \ldots, 2^n-1\}$. Επίσης, όπου στα αθροίσματα Σ, στα γινόμενα Π ή αλλού σημειώνεται $j \neq i$ θεωρείται ότι υπολογίζονται όλες οι τιμές j εκτός από την τιμή που συμπίπτει με το i:

Θεώρημα 1. $m_i.m_j=0$, $M_i + M_j=1$ για την περίπτωση $j \neq i$

Θεώρημα 2. $\sum_i m_i = 1$, $\prod_i M_i = 0$

Θεώρημα 3. $m_i = \prod_{j \neq i} (M_j)$, $M_i = \sum_{j \neq i} (m_i)$,

Θεώρημα 4. $m_i = \overline{M_i}$, $M_i = \overline{m_i}$.

2.5 Κανονική παράσταση συνάρτησης.

Με τη βοήθεια των συναρτήσεων Ελαχίστου και Μεγίστου όρου μπορούμε να σχηματίσουμε μια τυποποιημένη μορφή Λογικής Παράστασης μιας συνάρτησης που έχει το πλεονέκτημα ότι είναι μοναδική.

Οι συναρτήσεις Ελαχίστου και Μεγίστου όρου είναι ιδιαίτερα χρήσιμες για τη μελέτη των λογικών συναρτήσεων. Όπως είδαμε στην Υποενότητα 2.4, μια Λογική Συνάρτηση μπορεί να παρασταθεί με τον Πίνακα Αληθείας ή με διάφορες λογικές παραστάσεις. Με τη βοήθεια των συναρτήσεων Ελαχίστου και Μεγίστου όρου είναι δυνατόν αφενός μεν από τον Πίνακα Αληθείας να βρούμε κατ' ευθείαν μια Λογική Παράσταση της συνάρτησης αφετέρου δε η παράσταση αυτή είναι μοναδική για κάθε συνάρτηση. Οι ιδιότητες αυτές στηρίζονται στα δύο θεωρήματα που ακολουθούν:

ΚΑΝΟΝΙΚΟ ΑΘΡΟΙΣΜΑ ΓΙΝΟΜΕΝΩΝ

Κανονικό Άθροισμα Γινομένων (Canonical Sum of Products) ονομάζεται κάθε λογική παράσταση A με n μεταβλητές που διατυπώνεται σαν άθροισμα κάποιων minterms:

$$A(x_1,\ldots,x_n) = \Sigma_i\, m_i(x_1,\ldots,x_n)$$

Το θεώρημα του *Κανονικού Αθροίσματος Γινομένων* μιας συνάρτησης μας λέει ότι κάθε λογική συνάρτηση f(x1,…,xn) μπορεί να παρασταθεί σαν άθροισμα γινομένων της μορφής:

$$f(x_1,\ldots,x_n) = \Sigma_i\, [f(_i)\, .\, m_i(x_1,\ldots,x_n)],$$

όπου i παριστά ένα συνδυασμό τιμών των μεταβλητών $x_1,\ldots,x_n$ θεωρούμενο σαν δεκαδικό αριθμό, f(i) είναι η τιμή της συνάρτησης για τον συνδυασμό i και mi ο

αντίστοιχος minterm. Το θεώρημα αποδεικνύεται εύκολα, αν λάβουμε υπόψη ότι για κάθε συνδυασμό τιμών k των μεταβλητών $x_1,\ldots,x_n$:

α) μόνο ο αντίστοιχος minterm m_k θα είναι "1", ενώ όλοι οι άλλοι θα είναι "0" [(Θεμελιώδης ιδιότητα Ελαχίστων Όρων – Ενότητα 2.4)], δηλ.

$$m_k(k) = 1 \text{ και } m_i(k) = 0 \quad (i \neq k) \text{ και}$$

β) το θεώρημα ανάγεται στη σχέση

$$f(k) = f(k) \,.\, m_k(k) = f(k).1$$

Εάν στο ανωτέρω θεώρημα παραλειφθούν οι όροι του αθροίσματος, για τους οποίους f(k)=0, η παράσταση μιας συνάρτησης υπό μορφή κανονικού αθροίσματος γινομένων απλοποιείται στην εξής:

$$f(x_1,\ldots,x_n) = \sum_{i \in f(i)=1} m_i(x_1,\ldots,x_n)$$

δηλ. *ισούται με το άθροισμα όλων των minterms m_i, για τους οποίους f(i)=1*.

ΠΑΡΑΔΕΙΓΜΑ 2.18

Έστω η συνάρτηση f(x,y), που έχει τον ακόλουθο πίνακα αληθείας, και οι minterms m_0, m_1, m_2, m_3:

			minterms			
			m_0	m_1	m_2	m_3
α/α	x y	f	x'y'	x'y	x y'	x y
0	0 0	1	1	0	0	0
1	0 1	0	0	1	0	0
2	1 0	1	0	0	1	0
3	1 1	1	0	0	0	1

Όπως φαίνεται από τον πίνακα, ο κάθε minterm γίνεται 1 μόνο για ένα συνδυασμό τιμών των x, y και η συνάρτηση γράφεται:

$f(x,y) = 1.m_0(x,y) + 0.m_1(x,y) + 1.m_2(x,y) + 1.m_3(x,y)$

$= x'y' + xy' + xy$

Το θεώρημα του Κανονικού Αθροίσματος Γινομένων μπορεί να προκύψει επίσης από το πρώτο θεώρημα ανάπτυξης του Shannon με επανειλημμένη

εφαρμογή του μέχρις ότου εξαντληθούν όλες οι μεταβλητές της συνάρτησης. Τα συμπεράσματα από αυτό το θεώρημα είναι ότι:

α) Είναι εύκολο από τον Πίνακα Αληθείας μιας συνάρτησης f να βρούμε μια Λογική Παράστασή της, αρκεί να αθροίσουμε όλους τους minterms που αντιστοιχούν στις θέσεις f=1.

β) Η παράσταση μιας συνάρτησης σαν κανονικό άθροισμα γινομένων είναι μοναδική, δηλ. δεν υπάρχει άλλη τέτοια παράσταση που να παριστά την ίδια συνάρτηση. Γιατί μια άλλη παράσταση, με διαφορετικούς minterms, θα προερχόταν από ένα διαφορετικό Πίνακα Αληθείας και άρα από διαφορετική συνάρτηση. Η ιδιότητα αυτή χρησιμοποιείται για τον έλεγχο της ισοδυναμίας (ταυτοποίηση) δύο συναρτήσεων.

ΚΟΙΝΟ ΓΙΝΟΜΕΝΟ ΑΘΡΟΙΣΜΑΤΩΝ

Κανονικό Γινόμενο Αθροισμάτων (Canonical Product of Sums) ονομάζεται κάθε λογική παράσταση Α με n μεταβλητές που διατυπώνεται σαν γινόμενο κάποιων maxterms:

$$A(x_1,\ldots,x_n) = \Pi_i\, M_i(x_1,\ldots,x_n)$$

Το θεώρημα του Κανονικού Γινομένου Αθροισμάτων μιας συνάρτησης μας λέει ότι κάθε λογική συνάρτηση $f(x_1,\ldots,x_n)$ μπορεί να παρασταθεί σαν γινόμενο αθροισμάτων της μορφής:

$$f(x_1,\ldots,x_n) = \Pi_i\, [f(i) + M_i(x_1,\ldots,x_n)],$$

όπου i παριστά ένα συνδυασμό τιμών των μεταβλητών $x_1,\ldots,x_n$ θεωρούμενο σαν δεκαδικό αριθμό, f(i) είναι η τιμή της συνάρτησης για το συνδυασμό i και Mi ο αντίστοιχος maxterm. Το θεώρημα αποδεικνύεται, αν λάβουμε υπόψη ότι για κάθε τιμή k των μεταβλητών $x_1,\ldots,x_n$:

α) μόνο ο αντίστοιχος maxterm M_k θα είναι 0, ενώ όλοι οι άλλοι θα είναι 1 [(Θεμελιώδης ιδιότητα Μεγίστων Όρων – Ενότητα 3.5)], δηλ.

$$M_k(k) = 0 \text{ και } M_i(k) = 1 \ (\ i \neq k\ )$$

και

β) το θεώρημα ανάγεται στη σχέση

$$f(k) = 1.1\ldots[f(k) + 0]\ldots1.1 = f(k)+0.$$

Αντίστοιχα με το προηγούμενο θεώρημα, εάν παραλειφθούν οι όροι του γινομένου, για τους οποίους f(k)=1, η παράσταση της συνάρτησης υπό μορφή κανονικού γινομένου αθροισμάτων απλοποιείται στην εξής:

$$f(x_1,\ldots,x_n) = \sum_{i \in f(i)=0} M_i(x_1,\ldots,x_n)$$

δηλ. ισούται με το *γινόμενο όλων των maxterms M_i για τους οποίους f(i) = 0.*

ΠΑΡΑΔΕΙΓΜΑ 2.19

Έστω η συνάρτηση f(x,y) του προηγούμενου παραδείγματος (Παράδειγμα 2.18), που την επαναλαμβάνουμε εδώ μαζί με τους maxterms M0, M1,M2, M3:

			Maxterms			
			M_0	M_1	M_2	M_3
α/α	x y	f	x+y	x+y'	x'+y	x'+y'
0	0 0	**1**	**0**	1	1	1
1	0 1	**0**	1	**0**	**1**	1
2	1 0	**1**	1	1	0	1
3	1 1	**1**	1	1	1	**0**

Από τον πίνακα φαίνεται ότι κάθε maxterm γίνεται 0 μόνο για ένα συνδυασμό τιμών των x, y και η συνάρτηση γράφεται:

$f(x,y) = [1+M_0(x,y)][0+M_1(x,y)][1+M_2(x,y)][1+M_3(x,y)] = M1(x,y)$

$= x + y'$

Το θεώρημα του Κανονικού Γινομένου Αθροισμάτων μπορεί να προκύψει επίσης από το δεύτερο θεώρημα ανάπτυξης του Shannon με επανειλημμένη εφαρμογή του για όλες τις μεταβλητές. Όπως και στο προηγούμενο θεώρημα, τα συμπεράσματα από αυτό το θεώρημα είναι τα εξής:

α) Είναι εύκολο από τον Πίνακα Αληθείας μιας συνάρτησης f να βρούμε μια Λογική Παράστασή της, αρκεί να πολλαπλασιάσουμε όλους τους maxterms που αντιστοιχούν στις θέσεις f=0.

β) Η παράσταση μιας συνάρτησης σαν κανονικό γινόμενο αθροισμάτων είναι μοναδική, δηλ. δεν υπάρχει άλλη τέτοια παράσταση που να παριστά την ίδια συνάρτηση. Γιατί μια άλλη παράσταση, με διαφορετικούς maxterms, θα

προερχόταν από ένα διαφορετικό Πίνακα Αληθείας και άρα από διαφορετική συνάρτηση. Η ιδιότητα αυτή χρησιμοποιείται επίσης για τον έλεγχο της ισοδυναμίας (ταυτοποίηση) δύο συναρτήσεων.

ΘΕΩΡΗΜΑΤΑ ΚΑΝΟΝΙΚΗΣ ΠΑΡΑΣΤΑΣΗΣ ΣΥΝΑΡΤΗΣΕΩΝ

Τα θεωρήματα αφορούν στην εύρεση της συμπληρωματικής f' μιας συνάρτησης f, καθώς και την ταυτοποίηση δύο συναρτήσεων:

Θεώρημα 1. Έστω η συνάρτηση f υπό μορφή Κανονικής Παράστασης:

$f(x_1,\ldots,x_n) = \Sigma_i\ [f(i).m_i(x_1,\ldots,x_n)]$

ή $f(x_1,\ldots,x_n) = \Pi_i\ [f(i) + M_i(x_1,\ldots,x_n)]$

τότε η αντίστροφη συνάρτηση της f παριστάνεται αντίστοιχα:

$f'(x_1,\ldots,x_n) = \Pi_i\ [f'(i) + M_i(x_1,\ldots,x_n)]$

ή $f'(x_1,\ldots,x_n) = \Sigma_i\ [f'(i).m_i(x_1,\ldots,x_n)]$

Το θεώρημα αποδεικνύεται εύκολα από τα θεωρήματα του de Morgan και το Θεώρημα 4 της Ενότητας 2.4.

Θεώρημα 2. Δύο συναρτήσεις είναι ισοδύναμες (ταυτίζονται), όταν και μόνον όταν έχουν την ίδια Κανονική Παράσταση είτε υπό μορφή αθροίσματος γινομένων είτε γινομένου αθροισμάτων. Οι παραστάσεις ενδέχεται να διαφέρουν ως προς τη διάταξη των όρων τους.

ΑΝΑΠΤΥΞΗ ΣΥΝΑΡΤΗΣΗΣ ΣΕ ΚΑΝΟΝΙΚΟ ΑΘΡΟΙΣΜΑ ΓΙΝΟΜΕΝΩΝ

Η ανάπτυξη μιας συνάρτησης $f(x_1,\ldots,x_n)$ σε κανονικό άθροισμα γινομένων είδαμε πώς μπορεί να γίνει, αν μας δοθεί ο Πίνακας Αληθείας της (βλέπε Παράδειγμα 2.18). Εδώ θα δούμε επιπλέον πώς αναπτύσσεται, όταν μας δίνεται μια τυχούσα λογική παράστασή της. Υπάρχουν, γενικά, οι εξής τρεις τρόποι:

1. Να εφαρμοσθεί επαναληπτικά το πρώτο θεώρημα ανάπτυξης του Shannon για όλες τις μεταβλητές της συνάρτησης.

2. Από τον πίνακα αληθείας της συνάρτησης, να σχηματισθεί το άθροισμα όλων των minterms, για τους οποίους f(i)=1, σύμφωνα με το θεώρημα.

3. Από μια τυχούσα λογική παράσταση της συνάρτησης, να ακολουθηθεί η εξής μέθοδος:

α. η παράσταση αναπτύσσεται σε άθροισμα γινομένων (όχι απαραίτητα κανονικό),

β. ελέγχεται κάθε όρος του αθροίσματος και, εάν αποτελεί minterm, προχωρούμε στον επόμενο όρο,

γ. σε κάθε όρο, που δεν είναι minterm, προσδιορίζονται οι μεταβλητές της συνάρτησης που λείπουν και για κάθε μεταβλητή xi που λείπει πολλαπλασιάζεται ο όρος αυτός με την παράσταση (x_i+x_i'),

δ. στο τέλος αναπτύσσονται όλα τα γινόμενα και απαλείφονται οι όροι που πλεονάζουν.

ΠΑΡΑΔΕΙΓΜΑ 2.20

Ζητείται το κανονικό άθροισμα γινομένων της συνάρτησης:

$$f(x,y,z) = z' + y(x'+xz)$$

Απάντηση : Βρίσκεται με την τρίτη μέθοδο, (α) μετατρέποντας την παράσταση σε άθροισμα γινομένων και (β) παρατηρώντας κατόπιν ότι από τον πρώτο όρο λείπουν οι μεταβλητές x και y, από το δεύτερο η z και από τον τρίτο καμία:

$$\begin{aligned} f(x,y,z) &= z' + y(x'+xz) \\ &= z'+ x'y + xyz \\ &= (x+x')(y+y')z' + x'y(z+z') + xyz \\ &= xyz' + xy'z' + x'yz' + x'y'z' + x'yz + x'yz' + xyz \\ &= x'y'z' + x'yz' + x'yz + xy'z' + xyz' + xyz \end{aligned}$$

ΑΝΑΠΤΥΞΗ ΣΥΝΑΡΤΗΣΗΣ ΣΕ ΚΑΝΟΝΙΚΟ ΑΘΡΟΙΣΜΑ ΓΙΝΟΜΕΝΩΝ

Η ανάπτυξη μιας συνάρτησης $f(x_1,\ldots,x_n)$ σε *κανονικό γινόμενο αθροισμάτων* μπορεί να γίνει με τους εξής τρεις τρόπους:

1. Αν εφαρμοσθεί επαναληπτικά το δεύτερο θεώρημα ανάπτυξης του Shannon για όλες τις μεταβλητές της συνάρτησης.

2. Από τον πίνακα αληθείας της συνάρτησης, αν σχηματισθεί το γινόμενο όλων των Maxterms, για τους οποίους f(i)=0, σύμφωνα με το θεώρημα.

3. Από μια τυχούσα λογική παράσταση της συνάρτησης, εάν ακολουθηθεί η εξής μέθοδος:

α. η παράσταση αναπτύσσεται σε γινόμενο αθροισμάτων, όχι απαραίτητα κανονικό, βάσει του αξιώματος του επιμερισμού του γινομένου (Αξίωμα Α3, Ενότητα 2.1),

β. ελέγχεται κάθε όρος του γινομένου και, εάν αποτελεί Maxterm, προχωρούμε στον επόμενο όρο,

γ. σε κάθε όρο, που δεν είναι Maxterm, προσδιορίζονται οι μεταβλητές της συνάρτησης που λείπουν και για κάθε μεταβλητή xi που λείπει προστίθεται στον όρο αυτό η παράσταση (x_i . x_i'),

δ. στο τέλος μετατρέπονται όλα τα αθροίσματα που περιέχουν x_i . x_i' σε γινόμενα και απαλείφονται οι όροι που πλεονάζουν.

ΠΑΡΑΔΕΙΓΜΑ 2.21

Ζητείται το κανονικό γινόμενο αθροισμάτων της συνάρτησης του προηγουμένου παραδείγματος:

f(x,y,z) = z' + y(x'+xz)

Απάντηση : Εκτελούνται οι πράξεις που αφορούν παρενθέσεις. Προκύπτει η παράσταση:

f(x,y,z) = x'y + z' + xyz

Εφαρμόζεται η τρίτη μέθοδος ως εξής:

f(x,y,z) = x'y + z' + xyz

= x'y + [z'+xyz]

= [x'+z'+xyz] [y+z'+xyz]

= [(x'+z'+x)(x'+z'+y)(x'+z'+z)] [(y+z'+x)(y+z'+y)(y+z'+z)]

= [(1)(x'+z'+y)(1)] [(y+z'+x)(y+z'+y)(1)]

= (x'+y+z')(x+y+z')(y+z')

= (x'+y+z')(x+y+z')(xx'+y+z')

= (x'+y+z')(x+y+z')(x+y+z')(x'+y+z')

= (x'+y+z')(x+y+z')

ΜΕΤΑΤΡΟΠΗ ΚΑΝΟΝΙΚΗΣ ΜΟΡΦΗΣ

Όταν είναι γνωστή η μια από τις δύο μορφές Κανονικής Παράστασης μιας συνάρτησης $f(x_1,\ldots,x_n)$, τότε η μετατροπή της στην *άλλη κανονική μορφή* γίνεται με διπλή εφαρμογή του θεωρήματος του de Morgan, δεδομένου ότι f = (f')', ως εξής:

1. Αναπτύσσεται στην ίδια κανονική μορφή η αντίστροφη συνάρτηση $f'(x_1,\ldots,x_n)$
2. Αντιστρέφεται η παράσταση της συνάρτησης (f').

ΠΑΡΑΔΕΙΓΜΑ 2.22

Στο Παράδειγμα 2.21 είχαμε τη συνάρτηση:

f(x,y,z) = z' + y(x'+xz)

και η παράσταση αθροίσματος γινομένων της, στην οποία καταλήξαμε, ήταν:

$$f(x,y,z) = x'y'z' + x'yz' + x'yz + xy'z' + xyz' + xyz$$

Ζητείται να βρεθεί η άλλη Κανονική μορφή της.

Απάντηση : Κατά το θεώρημα του de Morgan, η παράσταση της αντίστροφης συνάρτησης είναι:

$$\begin{aligned} f'(x,y,z) &= (x+y+z)\,(x+y'+z)\,(x+y'+z')\,(x'+y+z)\,(x'+y'+z)\,(x'+y'+z') \\ &= (x+xy'+y'z+z)\,(x+y'+z')\,(x'+x'y'+y'z+z)\,(x'+y'+z') \\ &= (x+xz+y'z)\,(x'+x'z+y'z) \\ &= (x+y'z)\,(x'+y'z) \\ &= xy'z+x'y'z+y'z \\ &= y'z.(x+x') \\ &= xy'z+x'y'z \end{aligned}$$

Κατόπιν αντιστρέφεται η f' και επειδή:

$$f = (f')' = (xy'z+x'y'z)' = (x'+y+z')(x+y+z')$$

που ταυτίζεται με το αποτέλεσμα του Παραδείγματος 2.21.

2.6 Συναρτήσεις Δυο Μεταβλητών – Λογικές Πύλες.

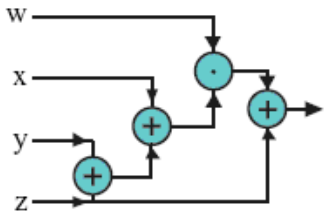
Ο υπολογισμός της τιμής μιας συνάρτησης με τη βοήθεια της Λογικής Παράστασής της γίνεται με την τοποθέτηση των τιμών των ανεξάρτητων μεταβλητών της στην παράσταση και εκτέλεση διαδοχικά των πράξεων που δηλώνονται. Ας πάρουμε σαν παράδειγμα τη συνάρτηση:

$$f(w,x,y,z) = w(z+y+x)+z$$

Το διάγραμμα του Σχήματος 2.5 απεικονίζει σχηματικά τις πράξεις και τη σειρά εκτέλεσής τους για τον υπολογισμό της συνάρτησης f(w,x,y,z). Τοποθετούμε τις τιμές στις θέσεις των μεταβλητών w, x, y, z και ακολουθούμε τη σειρά που δηλώνεται στο διάγραμμα. Επομένως, εάν υπήρχαν υλικές συσκευές που θα μπορούσαν να κάνουν την αντίστοιχη πράξη, τότε συνδέοντάς τις σύμφωνα με το διάγραμμα, που προκύπτει εύκολα από τη Λογική Παράσταση, θα είχαμε αυτομάτως στην έξοδο την τιμή της συνάρτησης.

Οποιαδήποτε συνάρτηση, οσοδήποτε σύνθετη και αν είναι, μπορεί να αναλυθεί σε απλές πράξεις. Η Λογική Παράσταση της συνάρτησης και τα

Θεωρήματα Ανάπτυξης (Υποενότητα 2.3.2) και Κανονικής Παράστασης (Ενότητα 2.5) μαρτυρούν γι' αυτό. Οι απλές πράξεις, με την έννοια ότι απεικονίζουν τιμές εισόδου σε μία έξοδο (Ενότητα 2.3), θα μπορούσαν να θεωρηθούν και αυτές σαν λογικές συναρτήσεις.



Σχήμα 2.9 Σειρά των πράξεων για την συνάρτηση *f(w,x,y,z)=w(z+y+x)+z.*

2.6.1 Συναρτήσεις Δυο Μεταβλητών.

Μία συνάρτηση μπορεί να είναι εξαιρετικά περίπλοκη ή να είναι τόσο απλή που να παριστάνει μια απλή πράξη της άλγεβρας Boole. Οι πιο απλές λογικές συναρτήσεις είναι οι συναρτήσεις που σχηματίζονται από δύο μεταβλητές και ορίζουν ένα σύνολο από συναρτήσεις – πράξεις, που θα ονομάζονται βασικές λογικές πράξεις. Μέσα σ' αυτές περιλαμβάνονται και οι πράξεις της άλγεβρας Boole. Το πλήθος των συναρτήσεων δύο μεταβλητών είναι $(2^2)^2$=16. Στον Πίνακα 2.9 δίνονται οι πίνακες αληθείας των 16 συναρτήσεων δύο μεταβλητών.

Πίνακας 2.11 Συναρτήσεις δύο μεταβλητών.

x y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Η σημασία των βασικών λογικών πράξεων έγκειται στο ότι, εάν κατασκευαστούν φυσικές μονάδες (υλικές συσκευές) που να υλοποιούν αυτές τις απλές συναρτήσεις – πράξεις, τότε οποιαδήποτε λογική συνάρτηση, επειδή αναλύεται σε βασικές πράξεις, μπορεί να υλοποιηθεί συνθέτοντας τις αντίστοιχες μονάδες μεταξύ τους. Οι φυσικές μονάδες που υλοποιούν τις βασικές λογικές πράξεις ονομάζονται **λογικές πύλες**.

Λόγω της χρησιμότητας που έχουν οι συναρτήσεις αυτές, έχει ορισθεί ειδικός συμβολισμός και όνομα για την κάθε βασική πράξη, καθώς και ειδικό

σχηματικό σύμβολο για την αντίστοιχη λογική πύλη. Στο Σχήμα 2.6 δίνεται η αλγεβρική περιγραφή των συναρτήσεων του Πίνακα 2.9 και οι αντίστοιχοι συμβολισμοί τους.

Από τις 16 βασικές συναρτήσεις – πράξεις οι πιο συχνά χρησιμοποιούμενες στην πράξη είναι οι **AND**, **OR**, **NOT**, **NAND**, **NOR** και **XOR** (Σχήμα 2.6). Λόγω της μεγάλης χρησιμότητάς τους έχουν κατασκευαστεί λογικές πύλες για καθεμία από αυτές τις συναρτήσεις – πράξεις.

2.6.2 Λογικές πύλες.

Οι λογικές πύλες είναι φυσικά συστήματα που υλοποιούν τις βασικές λογικές πράξεις. Αντίστοιχα με τις ανεξάρτητες μεταβλητές των συναρτήσεων, οι πύλες έχουν εισόδους στις οποίες καταλήγουν τα ψηφιακά σήματα που θέλουμε να επεξεργαστούμε. Στην έξοδο της πύλης παράγεται ένα σήμα, που η τιμή του είναι το αποτέλεσμα της επεξεργασίας, σύμφωνα με τις παρούσες τιμές των σημάτων εισόδου και το είδος της πύλης. Όταν θέλουμε να δείξουμε σχηματικά τα σήματα που κυκλοφορούν σ' ένα ψηφιακό σύστημα, χρησιμοποιούμε ειδικά σχήματα για να παραστήσουμε την κάθε πύλη. Στη στήλη “Λογική Πύλη” του Σχήματος 2.6 δίνεται το Κυκλωματικό διάγραμμα των βασικών πυλών, δηλ. το σχηματικό διάγραμμα που χρησιμοποιείται για να παραστήσει το κάθε είδος πύλης σ' ένα ψηφιακό κύκλωμα.

Το Κυκλωματικό διάγραμμα ενός ψηφιακού κυκλώματος αποτελείται από ένα σύνολο από γραμμές, που δηλώνουν τις διαδρομές των σημάτων, και από τα σχήματα των λογικών πυλών, που δηλώνουν το είδος της επεξεργασίας αυτών των σημάτων. Στις εισόδους κάθε πύλης συνδέονται τα σήματα που θέλουμε να επεξεργαστούμε και από την έξοδό της ξεκινά ένα σήμα, που η τιμή του είναι το αποτέλεσμα της επεξεργασίας. Λόγω της άμεσης αντιστοιχίας των πυλών με τις βασικές πράξεις είναι πολύ εύκολη η κατασκευή ενός ψηφιακού κυκλώματος όταν γνωρίζουμε μια λογική παράσταση της συνάρτησής του.

Συνάρτηση	Αλγεβρικό σύμβολο	Λειτουργία	Λογική Πύλη
$f_0 = 0$	0	Σταθερή 0	0
$f_1 = x \cdot y$	$x \cdot y$	AND	x y
$f_2 = x \cdot y'$	x/y	x "πλην" y	x y
$f_3 = x$	x	Μεταφορά	x
$f_4 = x'y$	y/x	y "πλην" x	x y
$f_5 = y$	y	Μεταφορά	y
$f_6 = xy'+x'y$	$x \oplus y$	XOR	x y
$f_7 = x+y$	$x+y$	OR	x y
$f_8 = (x+y)'$	$x'y'$	NOR	x y
$f_9 = x'y'+xy$	$x \odot y$	XNOR	x y
$f_{10} = y'$	y'	NOT	y
$f_{11} = x+y'$	$x+y'$	Εάν y Τότε x	x y
$f_{12} = x'$	x'	NOT	x
$f_{13} = x'+y$	$x'+y$	Εάν x Τότε y	x y
$f_{14} = (xy)'$	$x'+y'$	NAND	x y
$f_{15} = 1$	1	Σταθερή 1	1

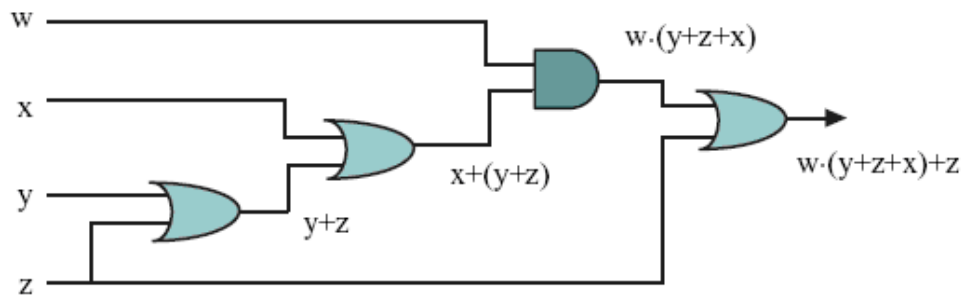
Σχήμα 2.10 Συμβολισμός συναρτήσεων δύο μεταβλητών.

ΠΑΡΑΔΕΙΓΜΑ 2.25

Θέλουμε να κατασκευάσουμε ένα ψηφιακό κύκλωμα που να υπολογίζει τη συνάρτηση:

$f(w,x,y,z) = w(z+y+x)+z$

Η σειρά των πράξεων για τον υπολογισμό της $f(w,x,y,z)$ παρίσταται στο Σχήμα 2.5. Απάντηση : Στο Σχήμα 2.5 δεν έχουμε παρά να τοποθετήσουμε στη θέση των πράξεων την αντίστοιχη λογική πύλη, όπως φαίνεται στο Σχήμα 2.7:



Σχήμα 2.11 Υλοποίηση συνάρτησης με λογικές πύλες.

ΠΥΛΕΣ NAND ΚΑΙ NOR

Καθεμία από αυτές τις πύλες μπορεί να θεωρηθεί ότι αποτελείται από δύο πύλες στη συσκευασία μιας πύλης. Συνδυάζουν η πρώτη μια πύλη **AND**, που ακολουθείται από μια πύλη αντιστροφής **NOT**, και η δεύτερη μια πύλη **OR**, που ακολουθείται από μια πύλη **NOT**. Ορίζονται αλγεβρικά με τις σχέσεις:

NAND(x,y) = (x.y)' και **NOR**(x,y) = (x+y)'

Έχουν ιδιότητες που τις κάνουν αρκετά χρήσιμες. Π.χ. έχουν το πλεονέκτημα ότι είναι "πρωτόγονες" πράξεις και επίσης χρησιμεύουν για την υλοποίηση των θεωρημάτων του de Morgan.

*ΠΥΛΗ **XOR***

Η συνάρτηση **XOR**, η οποία, όπου χρησιμοποιείται σαν "πράξη", θα συμβολίζεται για συντομία $\oplus$, έχει πολλές χρήσιμες ιδιότητες:

1) $x \oplus y = \begin{cases} 1 \text{ όταν } x \neq y \\ 0 \text{ όταν } x = y \end{cases}$

2) $x \oplus 1 = \overline{x}$ και $x \oplus 0 = x$.

3) $x \oplus y = y \oplus x$ αντιμεταθετική ιδιότητα

4) $x(y \oplus z) = (x \cdot y) \oplus (x \cdot z)$ επιμεριστική ιδιότητα γινομένου

5) $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$ προσεταιριστική ιδιότητα

Η πρώτη ιδιότητα 1) είναι πολύ σημαντική για να ελέγχουμε εάν δύο συναρτήσεις ταυτίζονται. Εάν παραστήσουμε με x και y τις συναρτήσεις που θέλουμε να ελέγξουμε, τότε, εάν είναι (x **XOR** y) = 0, οι x και y ταυτίζονται. Το αποτέλεσμα αυτό μπορεί να προκύψει είτε από ένα κύκλωμα, όπου στις εισόδους της πύλης **XOR** είναι συνδεδεμένα τα σήματα που παράγονται από τις συναρτήσεις x και y, είτε

μπορεί να προκύψει αλγεβρικά, εάν στις παραστάσεις των συναρτήσεων x και y εφαρμόσουμε την “πράξη” **XOR**.

2.6.3 Ισόμορφα Συστήματα.

Δύο συστήματα Σ1=<S1,F1,A1> και Σ2=<S2,F2,A2>, που αποτελούνται αντίστοιχα από τα σύνολα στοιχείων S1 και S2, από τις πράξεις F1 και F2 και τα αξιώματα A1 και A2, ονομάζονται ισόμορφα, όταν μεταξύ των στοιχείων των S1 και S2 και μεταξύ των πράξεων F1 και F2 υπάρχουν αντιστοιχίες τέτοιες, ώστε:

εάν S1<->S2 και F1<->F2, **τότε** A1<->A2.

Οι Λογικές Πύλες επίσης αποτελούν μια περίπτωση ισομορφίας που εκμεταλλευόμαστε για την κατασκευή των ψηφιακών Κυκλωμάτων.

Η μελέτη των ψηφιακών Κυκλωμάτων βασίζεται στο γεγονός ότι υπάρχουν απεικονίσεις που κάνουν την Άλγεβρα των Διακοπτών ισόμορφη με αυτά τα κυκλώματα και συνεπώς οι ιδιότητες της άλγεβρας εκφράζουν, κατάλληλα μεταφραζόμενες, και ιδιότητες των Κυκλωμάτων. Χάρη σ' αυτή την ισομορφία είμαστε σε θέση να μελετάμε και να συνθέτουμε εξαιρετικά πολύπλοκα ψηφιακά κυκλώματα.

Οι λογικές πύλες μπορούν να είναι μηχανικά συστήματα ή ηλεκτρονικά κυκλώματα ή υδραυλικές βαλβίδες ή οποιαδήποτε άλλα υλικά συστήματα, που με την κατάλληλη απεικόνιση των φυσικών τους καταστάσεων στις λογικές καταστάσεις καθίστανται ισόμορφα με τις αντίστοιχες λογικές συναρτήσεις.

2.6.4 Πράξεις Συναρτησιακώς Πλήρεις.

Ένα σύνολο από πράξεις ονομάζεται **συναρτησιακώς πλήρες** ή **καθολικό,** *εάν και μόνο εάν κάθε λογική συνάρτηση μπορεί να παρασταθεί αποκλειστικά με ένα συνδυασμό πράξεων από αυτό το σύνολο*. Οι πράξεις που ανήκουν σ' ένα τέτοιο σύνολο ονομάζονται **πρωτόγονες**. Σύμφωνα με τον ορισμό αυτό οι πράξεις της άλγεβρας Boole σχηματίζουν ένα συναρτησιακώς πλήρες σύνολο.

Γενικά για να αποδειχθεί ότι ένα σύνολο πράξεων είναι συναρτησιακώς πλήρες, αρκεί να αποδειχθεί ότι οι πράξεις του κατάλληλα συνδυαζόμενες μπορούν να αντικαταστήσουν *κάθε πράξη της άλγεβρας Boole*.

ΠΑΡΑΔΕΙΓΜΑ 2.26

Τα σύνολα των πράξεων {“+”, “–”} και {“.”, “–”} είναι συναρτησιακώς πλήρη. *Απάντηση* : Θα αποδείξουμε ότι υπάρχουν συνδυασμοί τους που παράγουν καθεμία από τις πράξεις της άλγεβρας Boole:

Πράξη Boole	:	Σύνολο:	{ +, ¯ },	{ ., ¯ }
x + y	:		x+y,	(x'.y')'
x . y	:		(x'+y')',	x.y
x'	:		x',	x'

Η σημασία των πρωτόγονων πράξεων συνίσταται στο ότι ένα οποιοδήποτε ψηφιακό κύκλωμα μπορεί να κατασκευαστεί χρησιμοποιώντας μια μικρή ποικιλία από διαφορετικές λογικές πύλες. Π.χ. οποιοδήποτε κύκλωμα μπορεί να κατασκευαστεί χρησιμοποιώντας μόνο πύλες **NAND** ή μόνο πύλες **NOR**. Η δυνατότητα αυτή είναι σημαντική, γιατί κάθε τεχνολογία κατασκευής πυλών ευνοεί από πλευράς κόστους, πυκνότητας, ταχύτητας κ.λπ. ορισμένους μόνο τύπους πυλών.

2.6.5 Πύλες με πολλαπλές Εισόδους.

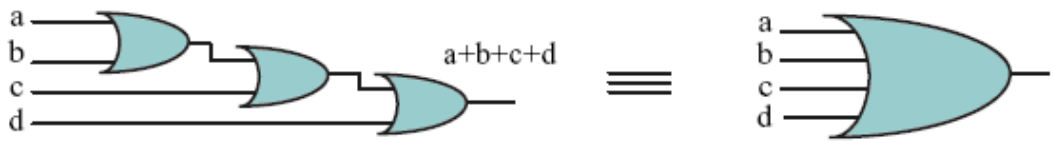
Η σύνθεση ενός ψηφιακού συστήματος με πολλές εισόδους, δηλ. η υλοποίηση μιας λογικής συνάρτησης με πολλές μεταβλητές, κάθε είσοδος αντιστοιχεί σε μια μεταβλητή, μπορεί να γίνει συνδυάζοντας λογικές πύλες δύο εισόδων (μεταβλητών). Κατά την υλοποίηση, όμως, ενός τέτοιου συστήματος είναι πολλές φορές επιθυμητό – για λόγους απλότητας και οικονομίας – ορισμένα είδη πυλών να έχουν περισσότερες εισόδους, δηλ. να εφαρμόζουν τη βασική τους λειτουργία σε συνδυασμούς περισσότερων μεταβλητών ταυτόχρονα.

Μια πύλη **OR** τεσσάρων εισόδων μπορεί να σχηματισθεί από διαδοχική σύνδεση πυλών **OR** δύο εισόδων, όπως στο Σχήμα 2.8. Το αποτέλεσμα θα είναι η παράσταση:

$$((a+b)+c)+d = a+b+c+d = c+b+a+d$$

Όπως προκύπτει, λόγω της προσεταιριστικής ιδιότητας του αθροίσματος (Θ4α, Υποενότητα 2.1.3) η σειρά των πυλών δεν έχει σημασία και επίσης λόγω της αντιμεταθετικής ιδιότητας (Α2, Ενότητα 2.1) το αποτέλεσμα είναι ανεξάρτητο από τη σειρά σύνδεσης των σημάτων στις εισόδους. Επομένως, μια πύλη **OR** τεσσάρων

εισόδων είναι “λογική” και επιπλέον, από φυσικής πλευράς, έχει ταχύτερη ανταπόκριση στα σήματα εισόδου της από ότι τρεις πύλες στη σειρά. Η λογική αυτή μπορεί να επεκταθεί σε πύλες **OR** οσωνδήποτε εισόδων.

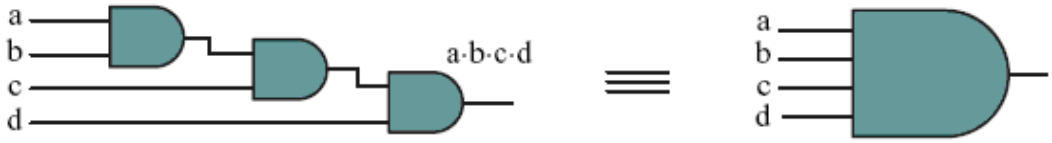


Σχήμα 2.12 Σύνθεση πύλης OR τεσσάρων εισόδων.

Συνδέοντας διαδοχικά πύλες **AND** δύο εισόδων, όπως στο Σχήμα 2.9, μπορεί να σχηματισθεί μια πύλη **AND** τεσσάρων εισόδων. Το αποτέλεσμα θα είναι η παράσταση.

$$((a.b).c).d = a.b.c.d = c.b.a.d$$

Λόγω της προσεταιριστικής και της αντιμεταθετικής ιδιότητας του γινομένου μπορεί να κατασκευαστεί μια πύλη **AND** τεσσάρων εισόδων και η λογική αυτή μπορεί να επεκταθεί σε πύλες **AND** οσωνδήποτε εισόδων.



Σχήμα 2.13 Σύνθεση πύλης AND τεσσάρων εισόδων.

Γενικά, μια πύλη μπορεί να επεκταθεί σε περισσότερες εισόδους, εάν η βασική της λειτουργία έχει: (α) την αντιμεταθετική και (β) την προσεταιριστική ιδιότητα.

Οι πύλες **AND** και **OR** έχουν, όπως είδαμε, αυτές τις ιδιότητες και επεκτείνονται για οσοδήποτε αριθμό εισόδων. Στην πράξη κατασκευάζονται πύλες **AND** και **OR** με 2, 3, 4 και 8 εισόδους. Οι πύλες **NAND** και **NOR** δεν έχουν την προσεταιριστική ιδιότητα, διότι π.χ. η

$$\textbf{NAND}(\ \textbf{NAND}\ (x,y), z\ ) = ((x\ y)'\ z)' = ((x'+y')\ z)' = (x'z + y'z)'$$

είναι διάφορη της:

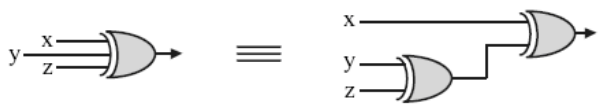
$$\textbf{NAND}\ (\ x, \textbf{NAND}(y,z)\ ) = (x\ (y\ z)'\ )' = (x\ (y'+z')\ )' = (xy' + xz')'$$

Οι πύλες όμως αυτές μπορούν να επεκταθούν σε πύλες με πολλαπλές εισόδους, εάν ορισθούν οι λειτουργίες τους λίγο διαφορετικά, ως εξής:

NAND(x,y,z,w,...) = (x y z w ...)'

και **NOR**(x,y,z,w,...) = (x+y+z+w+...)'

Με αυτή τη λογική κατασκευάζονται πύλες **NAND** και **NOR** με πολλές εισόδους. Η πύλη **XOR** έχει και τις δύο ιδιότητες και μπορεί να επεκταθεί σε οσοδήποτε αριθμό εισόδων, αν και συνήθως κατασκευάζεται με δύο εισόδους. Στο σχήμα 2.10 παρουσιάζεται το ισοδύναμο κύκλωμα για τρεις εισόδους. Μία πύλη **XOR** με πολλαπλές εισόδους παίρνει την τιμή "1", όταν εμφανίζεται περιττό πλήθος από "1" στις εισόδους της. Ο πίνακας αληθείας της για τρεις μεταβλητές είναι ο παρακάτω.



Πίνακας XOR τριών εισόδων

x y z	XOR
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Σχήμα 2.14 Ισοδύναμο κύκλωμα πύλης XOR με τρεις εισόδους.

3.1 Μέθοδος απλοποίησης με το χάρτη Karnaugh

Είναι μια απλή και εύχρηστη μέθοδος που επιδιώκει να εντοπίσει με γραφικό τρόπο (βασίζεται στην οπτική παρατήρηση επί του χάρτου) το μικρότερο αριθμό από υποκύβους (N–Κύβος – Υποενότητα 2.3.1.), δηλ. λογικά γινόμενα, που ορίζουν μια δοθείσα λογική συνάρτηση. Η παράσταση της συνάρτησης, κατόπιν, σχηματίζεται από το λογικό άθροισμα αυτών των υποκύβων.

Η μέθοδος του Χάρτη Karnaugh στηρίζεται στα διαγράμματα Venn και αρχικά είχε προταθεί από τον Veitch, αργότερα δε τελειοποιήθηκε από τον Karnaugh. Είναι απλή και γρήγορη, είναι όμως κατάλληλη για μικρό πλήθος μεταβλητών, το πολύ 5.

Όπως είχαμε αναφέρει στην Υποενότητα 2.3.1 (N–Κύβος), μία συνάρτηση n μεταβλητών $f(x_0, x_1 \ldots, x_{n-1})$ μπορεί να παρασταθεί με έναν n–κύβο (2^n κορυφές), στον οποίον έχουμε σημειώσει τις κορυφές του, που η συνάρτηση παίρνει την τιμή “1”. Κάθε σημειωμένη κορυφή αντιστοιχεί σε ένα minterm της συνάρτησης (Ενότητα 2.4). Ένας *m–υποκύβος* (υποσύνολο) ενός n–κύβου (m<n) σχηματίζεται από ένα υποσύνολο των 2^n κορυφών τέτοιο, ώστε n–m μεταβλητές να έχουν την ίδια τιμή σε όλες τις κορυφές του υποκύβου, οι δε υπόλοιπες m μεταβλητές να εμφανίζουν όλους τους δυνατούς συνδυασμούς τιμών. Δηλαδή, από τις $x_0, x_1 \ldots, x_{n-1}$ μεταβλητές, εάν l_j είναι ο j minterm των n–m μεταβλητών, π.χ. $x_0, \ldots, x_{n-m-1}$, και m_k είναι ο k minterm των υπολοίπων μεταβλητών $x_{n-m} \ldots x_{n-1}$, τότε ένας m–υποκύβος ορίζεται αλγεβρικά με μία συνάρτηση της μορφής:

$$\begin{aligned} \text{m – υποκύβος} &= l_j \sum_{0}^{2^m-1} m_k \\ &= l_j \end{aligned}$$

Αντίστοιχα, ο m–υποκύβος μιας συνάρτησης $f(x_0, x_1, \ldots, x_{n-1})$ ορίζεται από το υποσύνολο των κορυφών της, των οποίων οι αντίστοιχοι minterms έχουν n–m μεταβλητές με τον ίδιο εκθέτη (Ενότητα 2.4) και οι υπόλοιπες m μεταβλητές εμφανίζουν όλους τους δυνατούς συνδυασμούς εκθετών. Π.χ. ένας 0–υποκύβος αντιστοιχεί σε ένα minterm της συνάρτησης $f(x_0, x_1 \ldots, x_{n-1})$, ένας 1–υποκύβος ισοδυναμεί σε κάποιο γινόμενο n–1 μεταβλητών της (άθροισμα δύο minterms στους οποίους μία μεταβλητή έχει εκθέτη 0 και

1) και ο n–υποκύβος αντιστοιχεί στη συνάρτηση που είναι $f(x_0, x_1, \ldots, x_{n-1})=1$ για οποιονδήποτε συνδυασμό τιμών.

Επειδή κάθε συνάρτηση μπορεί να περιγραφεί ως Κανονικό Άθροισμα Γινομένων (Ενότητα 2.4), ένας m–υποκύβος της συνάρτησης ισοδυναμεί με μία σύμπτυξη 2^m minterms της και αντικατάστασή τους από ένα γινόμενο n–m μεταβλητών, δηλ. οικονομία 2^m όρων (γινομένων). Η μέθοδος του Χάρτη Karnaugh μας βοηθά (α) να εντοπίσουμε, με οπτική παρατήρηση επί του χάρτου, τους μεγαλύτερους δυνατούς υποκύβους μιας δοθείσας λογικής συνάρτησης και (β) να σχηματίσουμε τη συνάρτηση αθροίζοντας τον ελάχιστο αριθμό (μικρότερο άθροισμα) από υποκύβους. Παρακάτω περιγράφονται ο Χάρτης Karnaugh και η μέθοδος που βασίζεται σ' αυτόν.

3.1.1 Χάρτης Karnaugh

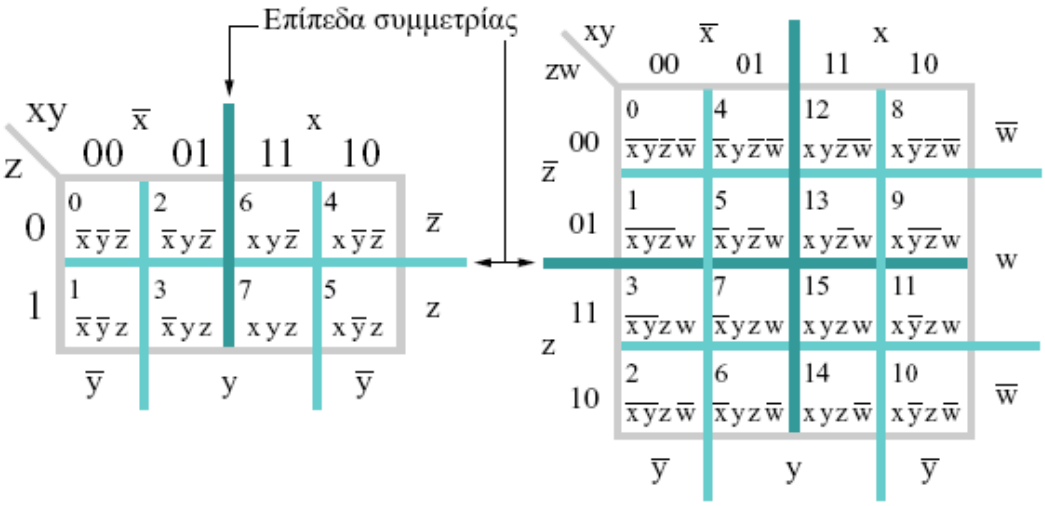
Σκοπός του χάρτη Karnaugh είναι να παραστήσει στο επίπεδο έναν κύβο διαστάσεων, διατηρώντας τη σχέση “γειτονικότητας” μεταξύ των κορυφών του. (Η δομή του περιγράφηκε στην Υποενότητα 2.3.1.) είναι ένας Πίνακας Αληθείας δύο διαστάσεων (Σχήμα 3.1). Επαναλαμβάνουμε ότι η διάταξη των τιμών των συντεταγμένων (μεταβλητών) στον πίνακα δεν ακολουθεί την κανονική αύξουσα σειρά, αλλά τη σειρά ενός ανακλαστικού κώδικα (κώδικας GRAY – Υποενότητα 1.2.3).

Σε κάθε χάρτη Karnaugh (Σχήμα 3.1) υπάρχουν τόσα τετραγωνίδια όσοι οι δυνατοί συνδυασμοί τιμών των μεταβλητών του και κάθε τετραγωνίδιο αντιστοιχεί σε ένα minterm, δηλ. αν υπάρχουν k μεταβλητές σχηματίζονται 2^k τετραγωνίδια και αντιστοίχως minterms.

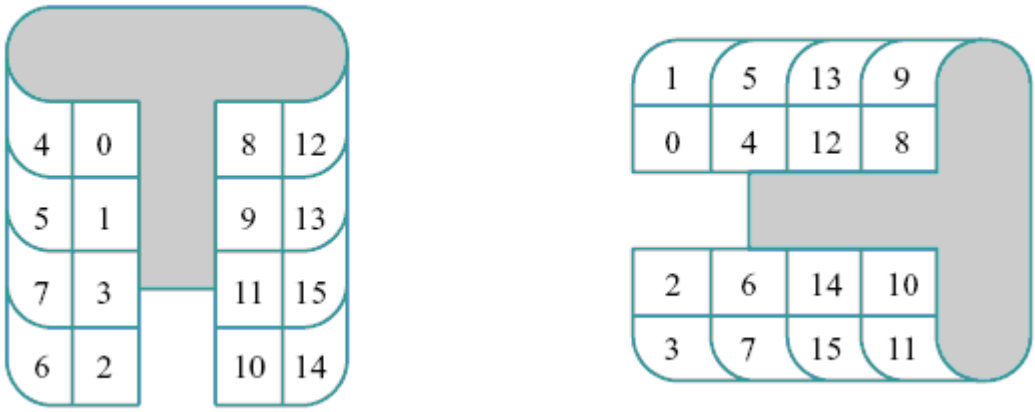
Η επιλογή ενός κώδικα με ανακλαστικές ιδιότητες για τη διάταξη του χάρτη έχει ως συνέπεια να εμφανίζονται *επίπεδα συμμετρίας* (Σχήμα 3.1), που χωρίζουν το χάρτη σε περιοχές, όπου μια μεταβλητή εμφανίζεται από τη μια πλευρά ως απλή και από την άλλη με το συμπλήρωμά της, π.χ. x και x', y και y', z και z'. Λόγω δε των κυκλικών ιδιοτήτων που ταυτόχρονα κατέχει ο ίδιος κώδικας, οι συντεταγμένες κάθε τετραγωνιδίου διαφέρουν από αυτές των γειτονικών του, οριζόντια ή κάθετα, όχι όμως διαγώνια, τετραγωνιδίων μόνο κατά την τιμή μιας μεταβλητής.

Για τους παραπάνω λόγους ο minterm του κάθε τετραγωνιδίου διαφέρει από τους "γεωμετρικά" γειτονικούς του minterms μόνο κατά τον εκθέτη (αντιστροφή ή όχι – Ενότητα 2.4) μιας μεταβλητής του (Σχήμα 3.1). Η ταύτιση μάλιστα της "γεωμετρικής" γειτονικότητας με τη "λογική" γειτονικότητα (διαφορά μόνο σε μια μεταβλητή) επεκτείνεται και πέρα από τα άκρα του πίνακα. Τα δύο ακραία τετραγωνίδια κάθε σειράς του πίνακα, οριζοντίως ή καθέτως, διαφέρουν μεταξύ τους μόνο κατά την τιμή μιας μεταβλητής, καθώς και οι minterms που περιέχουν. Για λόγους συμβατότητας με τη "γεωμετρική" γειτονικότητα, ο

χάρτης Karnaugh θεωρείται ότι αναδιπλώνεται είτε οριζοντίως είτε κατακορύφως (Σχήμα 3.2), ειδικότερα δε ότι έχει σχήμα σαμπρέλας αυτοκινήτου. Η απεικόνιση μιας συνάρτησης στο χάρτη Karnaugh μπορεί να γίνει είτε με απ' ευθείας καταγραφή των "1", όταν δίδεται ο πίνακας αληθείας της, είτε με την ανάπτυξή της σε κανονικό άθροισμα γινομένων, όταν δίνεται μια παράστασή της και κατόπιν καταγραφή των minterms που αντιστοιχούν στη συνάρτηση. Ένας τρόπος για να αποφύγει κανείς την ανάπτυξη σε κανονικό άθροισμα γινομένων, όταν δίνεται ένα οποιοδήποτε άθροισμα γινομένων, είναι ο εξής:



Σχήμα 3.15 Χάρτης Karnaugh (α) τριών και (β) τεσσάρων μεταβλητών.



Σχήμα 3.16 Αναδίπλωση Χάρτη Karnaugh.

Έστω η συνάρτηση:

$$f(x,y,z,w) = x z' + \ldots$$

Για τον όρο xz' τοποθετούνται "1" σε όλες τις περιοχές του πίνακα (βάσει επιπέδων συμμετρίας) που περιέχουν τις μεταβλητές x και z', και το ίδιο για τους υπόλοιπους όρους.

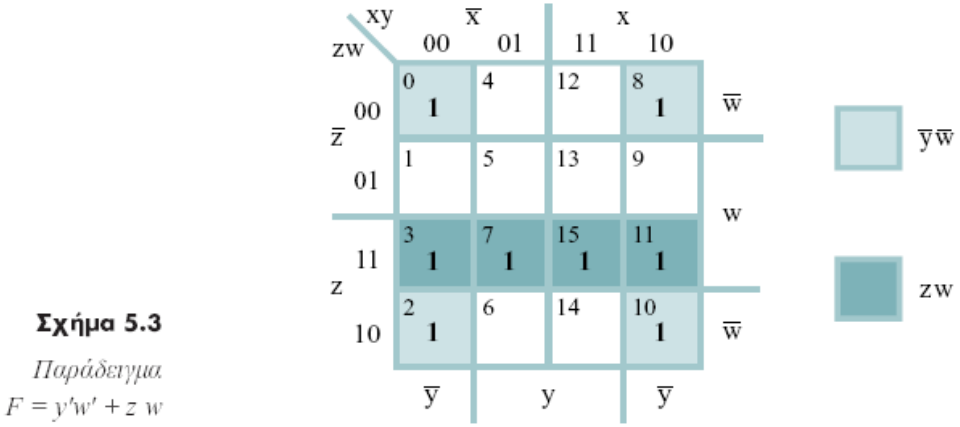
Παράδειγμα 3.1

Έστω η συνάρτηση:

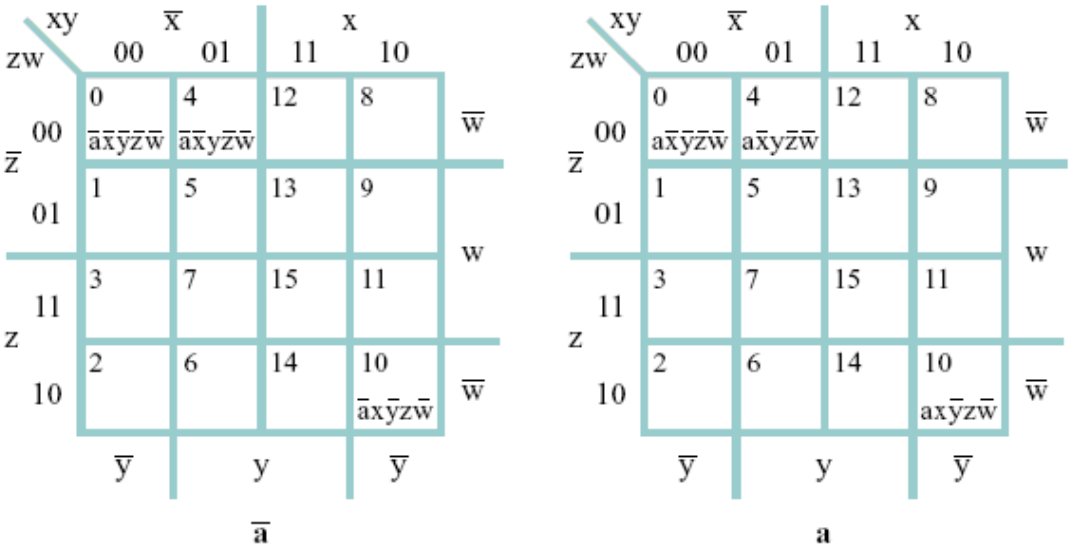
$$F(x,y,z,w) = \Sigma(0,2,3,7,8,10,11,15) \quad \text{(κανονική μορφή)}$$

$$= y'\,w' + z\,w \quad \text{(λογική παράσταση)}$$

Ο χάρτης της συνάρτησης δίνεται στο Σχήμα 3.3, όπου φαίνονται σκιασμένες οι περιοχές που αντιστοιχούν στους όρους y'w' και zw.



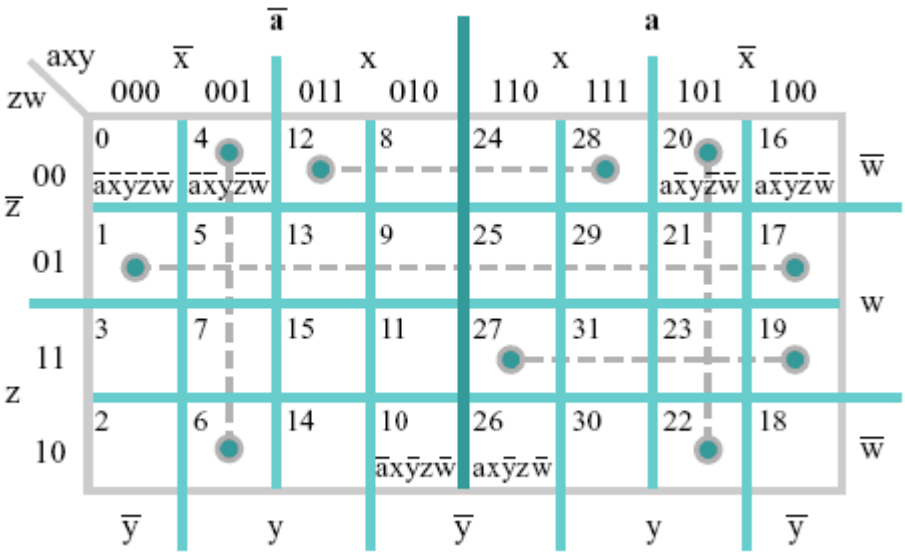
Σχήμα 3.17 ***Παράδειγμα F = y'w' + zw.***



Σχήμα 3.18 Χάρτης Karnaugh 5 Μεταβλητών.

Συναρτήσεις με 5 μεταβλητές, π.χ. F(a,x,y,z,w), απεικονίζονται με ένα ζεύγος πινάκων (Σχήμα 3.4), όπου ο πρώτος πίνακας περιλαμβάνει τους όρους με τη μεταβλητή a'

και ο δεύτερος πίνακας τους όρους με την a. Μία εναλλακτική κατασκευή χάρτη Karnaugh 5 μεταβλητών δίνεται στο Σχήμα 3.5. Μπορεί να θεωρηθεί ότι σχηματίζεται από τη συνένωση δύο χαρτών των 4 μεταβλητών, τοποθετημένων συμμετρικά ως προς το κεντρικό επίπεδο συμμετρίας (κάθετη παχιά γραμμή). Αριστερά βρίσκονται οι όροι με a' και δεξιά οι όροι με a. Τετράγωνα που είναι συμμετρικά ως προς τις παχιές γραμμές του χάρτη (επίπεδα συμμετρίας) διαφέρουν κατά μία μεταβλητή.



Σχήμα 3.19 Εναλλακτική μορφή χάρτη Karnaugh 5 μεταβλητών.

Η αντίστροφη εργασία, δηλ. η εύρεση της παράστασης μιας συνάρτησης, όταν δίνεται ο χάρτης Karnaugh που αντιστοιχεί σ' αυτήν, είναι πολύ εύκολη. Η ζητούμενη παράσταση μπορεί να δοθεί είτε σαν άθροισμα ελάχιστων όρων, όταν ληφθούν όλοι οι minterms που αντιστοιχούν στα "1" του χάρτη, είτε σαν γινόμενο μεγίστων όρων, όταν ληφθούν όλοι οι maxterms που αντιστοιχούν στα "0" του χάρτη.

3.1.2 Απλοποίηση με τον χάρτη Karnaugh

Η απλοποίηση στηρίζεται στη συστηματική εφαρμογή του θεωρήματος a'B+aB=B. Η ιδιότητα του χάρτη Karnaugh να τοποθετεί σε "γεωμετρικώς" γειτονικές θέσεις στο χάρτη τους λογικά γειτονικούς minterms παρέχει τη δυνατότητα να εφαρμοσθεί το θεώρημα με γραφικό τρόπο, χωρίς αλγεβρικές πράξεις. Απλοποιούνται, κατόπιν συμπτύξεως ανά ζεύγη, εκείνοι οι όροι από την παράσταση της συναρτήσεως που αντιστοιχούν σε "1" τοποθετημένα σε γειτονικά τετραγωνίδια.

Η απλοποίηση γίνεται ως εξής:

α) Τοποθετούνται τα "1" της συναρτήσεως στο χάρτη.

β) Σχηματίζονται ομάδες (υποκύβοι) από ζεύγη γειτονικών "1". Κάθε ζεύγος απαλείφει μια μεταβλητή, αυτή με το διαφορετικό εκθέτη. Λαμβάνεται επίσης υπόψη και η γειτονικότητα των ακραίων τετραγωνιδίων.

γ) Εάν δύο ζεύγη είναι γειτονικά, δηλ. ευρίσκονται συμμετρικά ως προς

κάποιο επίπεδο συμμετρίας, τότε οι όροι τους διαφέρουν ως προς τον εκθέτη μιας μεταβλητής, η οποία και απαλείφεται. Τα ζεύγη αυτά σχηματίζουν μια μεγαλύτερη ομάδα (υποκύβο).

δ) Ομάδες (υποκύβοι) που βρίσκονται τοποθετημένες συμμετρικά ως προς κάποιο επίπεδο συμμετρίας και μοιάζουν, όπως ένα αντικείμενο με το είδωλό του στον καθρέπτη, ονομάζονται “κατοπτρικές”. Οι κατοπτρικές ομάδες διαφέρουν ως προς τον εκθέτη μιας μεταβλητής, η οποία και απαλείφεται. Ακολουθώντας τη λογική της συνενώσεως των κατοπτρικών ομάδων μπορούν να σχηματισθούν και μεγαλύτερες ομάδες, όπου είναι δυνατόν, απαλείφοντας κάθε φορά από μια μεταβλητή. Επιδιώκεται δε πάντοτε ο σχηματισμός των μεγαλύτερων δυνατών ομάδων (υποκύβων).

ε) Στο τέλος επιλέγεται εκείνο το υποσύνολο μεγάλων ομάδων που προκαλεί τις *λιγότερες* επικαλύψεις στα “1” της συνάρτησης.

Κάθε ομάδα (υποκύβος), οσοδήποτε μεγάλη, αποτελείται από γειτονικά τετραγωνίδια, στα οποία η συνάρτηση έχει την τιμή "1". Λόγω του τρόπου σχηματισμού των ομάδων (κατοπτρισμός), κάθε ομάδα περιέχει πάντοτε 2^k, k=0,1,2,… πλήθος "1". Οι 2^k minterms που περιέχονται στην ομάδα συμπτύσσονται σ' ένα γινόμενο που περιέχει k λιγότερες μεταβλητές.

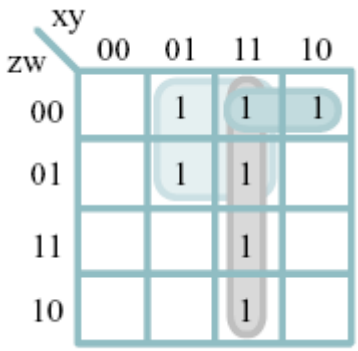
Παράδειγμα 3.2

Έστω η συνάρτηση f(x,y,z,w), η οποία παριστάνεται στον παρακάτω χάρτη:

Στο χάρτη σχηματίζονται οι μεγαλύτερες δυνατές ομάδες, ακόμη και αν απαιτείται μερική επικάλυψη των ομάδων. Η επικάλυψη επιτρέπεται με βάση το θεώρημα a=a+a (Θ1, Υποενότητα 2.1.3). Η τετράγωνη ομάδα περιέχει 22 "1", απαλείφει τις δύο μεταβλητές x και w και δίνει τον όρο yz'. Η κατακόρυφη ομάδα περιέχει 22 "1", απαλείφει τις μεταβλητές z και w και παράγει τον όρο xy. Η μικρή ομάδα απαλείφει την y και παράγει τον όρο xz'w'. Το άθροισμα των όρων που προκύπτουν από τις ομάδες αποτελεί την ελάχιστη παράσταση της συναρτήσεως:

$$f(x,y,z,w)= xy + yz'+ xz'w'$$

Η παραπάνω γραφική απλοποίηση αντιστοιχεί στην εξής αλγεβρική απλοποίηση, όπου προηγουμένως η συνάρτηση αναπτύχτηκε σε κανονικό άθροισμα γινομένων:



$$f(x,y,z,w) = \overbrace{x\bar{y}\bar{z}\bar{w} + \bar{x}y\bar{z}\bar{w} + \bar{x}y\bar{z}w + xy\bar{z}\bar{w}}^{x\,\bar{z}\,\bar{w}} + \overbrace{xy\bar{z}w + xyzw + xyz\bar{w}}^{x\,y}$$

$$\underbrace{\bar{x}y\bar{z}\bar{w} + \bar{x}y\bar{z}w + xy\bar{z}\bar{w} + xy\bar{z}w}_{y\,\bar{z}}$$

$$= x\,y + y\,\bar{z} + x\,\bar{z}\,\bar{w}$$

Η μέθοδος ελαχιστοποίησης με το χάρτη Karnaugh εξαρτάται πολύ από την ικανότητα του παρατηρητή να σχηματίζει τον καλύτερο συνδυασμό από ομάδες. Είναι εύκολο να παρασυρθεί και να διαλέξει ομάδες που παράγουν μικρή παράσταση, αλλά όχι και την ελάχιστη.

Για τη σωστή απλοποίηση με τη βοήθεια του χάρτη, πρέπει να επιδιώκονται τα εξής:

α) Να σχηματίζονται οι μεγαλύτερες δυνατές ομάδες επί του χάρτη, ώστε οι όροι που προκύπτουν από αυτές να έχουν τις λιγότερες μεταβλητές.

β) Να επιλέγεται ο ελάχιστος αριθμός ομάδων που καλύπτει το χάρτη, δηλ. να περιλαμβάνονται όλα τα "1" σε όσο το δυνατόν λιγότερες ομάδες, ώστε το πλήθος των όρων που παράγονται να είναι, κατά το δυνατόν, μικρό.

Οι παραπάνω κανόνες οδηγούν στην εξής τακτική που πρέπει να ακολουθείται:

1) Κάθε τετραγωνίδιο με "1" πρέπει να μετέχει σε μια τουλάχιστον ομάδα.

2) Πρέπει να σχηματίζονται οι μεγαλύτερες δυνατές ομάδες από 2k γειτονικά τετραγωνίδια με "1".

3) Κάθε τετραγωνίδιο με "1" πρέπει να μετέχει στον ελάχιστο αριθμό ομάδων.

Σύμφωνα με την απαίτηση 1) μία ομάδα, μικρή ή μεγάλη, που περιέχει κάποιο “1” που *δεν περιέχεται* σε άλλη ομάδα, πρέπει αναγκαστικά να επιλεγεί. Μια τέτοια ομάδα (υποκύβος) ονομάζεται *ουσιώδης*. Εάν κάποιο 1”

παραμείνει στο τέλος που να μην συμμετέχει σε ομάδα με άλλα “1”, τότε σχηματίζει από μόνο του μία ομάδα που είναι ουσιώδης.

3.2 Αθροιστές.

Οι αθροιστές είναι τα βασικότερα και συνηθέστερα κυκλώματα λόγω των πολλών εφαρμογών τους όχι μόνο στους καθ' αυτό υπολογιστές αλλά και σε πολλές άλλες κατηγορίες Κυκλωμάτων. Θα περιγράψουμε τις βασικές μονάδες για την εκτέλεση της απλής πρόσθεσης και από αυτές τις μονάδες σταδιακά θα αναπτύξουμε μεγαλύτερες μονάδες για την πρόσθεση οσωνδήποτε μεγάλων δυαδικών αριθμών. Θα εξηγήσουμε τα πλεονεκτήματά τους και, όπου μειονεκτούν, θα παρουσιάσουμε συμπληρωματικά κυκλώματα για τη βελτίωσή τους.

Θα ξεκινήσουμε από την απλούστερη μονάδα, τον *ημιαθροιστή*, και, αφού εξηγήσουμε το μειονέκτημά του, θα οδηγηθούμε στη σχεδίαση μιας καλύτερης μονάδας, τον *πλήρη αθροιστή*. Με τη βοήθεια του πλήρη αθροιστή 1 bit θα κατασκευάσουμε έναν αθροιστή για αριθμούς με οσοδήποτε πλήθος bit (αθροιστής n bit). Λόγω της μικρής ταχύτητας του απλού αθροιστή n bit θα τον συμπληρώσουμε με τις *μονάδες πρόβλεψης* κρατουμένου για τη βελτίωση της ταχύτητάς του. Η πράξη της προσθέσεως δύο δυαδικών αριθμών αποτελεί τη βάση για τις άλλες αριθμητικές πράξεις και εμφανίζεται σε πάρα πολλές εφαρμογές. Είναι σημαντικό να έχουμε ένα καλό κύκλωμα αθροιστού τόσο από πλευράς κόστους όσο και ταχύτητας. Έστω οι δυαδικοί αριθμοί, χωρίς πρόσημο, $X = x_{n-1} \ldots x_1x_0$ και $Y = y_{n-1} \ldots y_1y_0$.

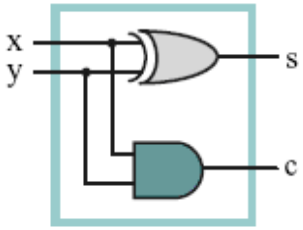
Η πρόσθεση εκτελείται προσθέτοντας διαδοχικά τα ζεύγη ψηφίων x_i και y_i (i=0, …, n–1), από τα οποία δημιουργούνται το άθροισμα s_i και το κρατούμενο c_i. Το απλούστερο κύκλωμα γι' αυτή την πράξη είναι ο *ημιαθροιστής*.

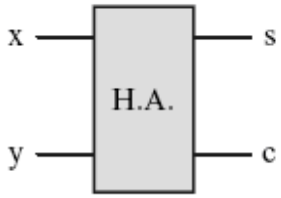
ΗΜΙΑΘΡΟΙΣΤΗΣ

Ο *ημιαθροιστής* (Half Adder) προσθέτει το ζεύγος των ψηφίων x και y και παράγει το άθροισμα s και το κρατούμενο c. Οι αντίστοιχοι Πίνακες Αληθείας και τα κυκλώματα για τις συναρτήσεις s και c δίδονται στο Σχήμα 3.6.

Ο ημιαθροιστής έχει το μειονέκτημα ότι είναι κατάλληλος μόνο για την πρόσθεση ενός ζεύγους bit. Όταν υπάρχει κρατούμενο c–1 από προηγούμενη πρόσθεση, δεν μπορεί να ληφθεί υπόψη.

x y	s	c
0 0	0	0
0 1	1	0
1 0	1	0
1 1	0	1





Σχήμα 3.20 Πίνακας Αληθείας, κύκλωμα και σχηματικό διάγραμμα ημιαθροιστή.

ΠΛΗΡΗΣ ΑΘΡΟΙΣΤΗΣ

Όταν προσθέτουμε αριθμούς με πολλά bit, πρέπει να υπάρχει δυνατότητα για την ταυτόχρονη πρόσθεση και του κρατουμένου c–1 της προηγούμενης πρόσθεσης. Το πρόβλημα λύνεται με ένα κύκλωμα που παράγει, για τα ψηφία τάξεως i, το άθροισμα s_i και το κρατούμενο c_i της πρόσθεσης των τριών bit x_i, y_i και c_i–1 (Σχήμα 3.7). Το κύκλωμα (Σχήμα 3.7) έχει τρεις εισόδους (x_i, y_i, c_i–1) και δύο εξόδους (s_i, c_i) και ονομάζεται *πλήρης αθροιστής* (Full Adder).

x_i	y_i	c_{i-1}	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

c_{i-1} \ $x_i y_i$	00	01	11	10
0		1		1
1	1		1	

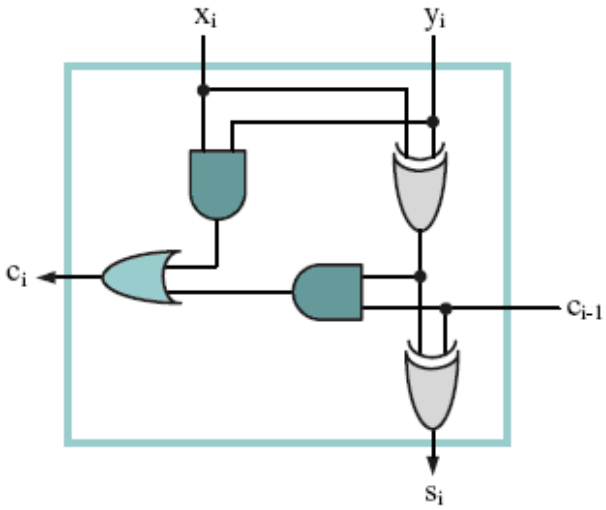
$s_i = x_i \oplus y_i \oplus c_{i-1}$

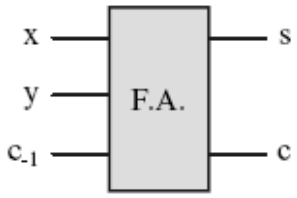
Άθροισμα

c_{i-1} \ $x_i y_i$	00	01	11	10
0			1	
1		1	1	1

$c_i = x_i \cdot y_i + c_{i-1} \cdot (x_i \oplus y_i)$

Κρατούμενο

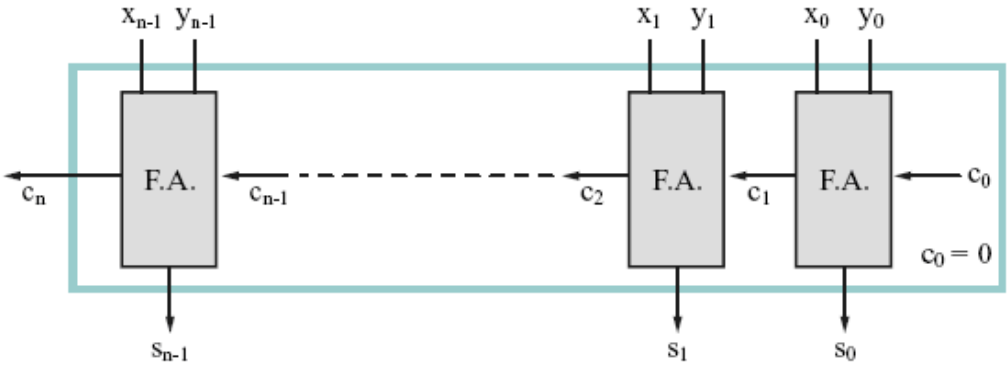




Σχήμα 3.21 Λογικές συναρτήσεις, κύκλωμα και σχηματική παράσταση πλήρη αθροιστή.

3.2.1 Παράλληλος Αθροιστής

Ο πλήρης αθροιστής που γνωρίσαμε προσθέτει ταυτόχρονα τρία bits. Ένας απλός τρόπος κατασκευής ενός κυκλώματος για την ταυτόχρονη (παράλληλη) πρόσθεση δυαδικών αριθμών των n bits (n>1) είναι να συνδέσουμε διαδοχικά n μονάδες πλήρων αθροιστών, όπως φαίνεται στο Σχήμα 3.8. Στην είσοδο c0, που υπάρχει λόγω της κατασκευής του πλήρη αθροιστή, τοποθετούμε την τιμή c_0 = 0. Η έξοδος c_n δίνει το τελικό κρατούμενο της πρόσθεσης. Σε ορισμένες εφαρμογές το bit c_n χρησιμοποιείται για να ελέγξουμε την *υπερχείλιση* του αθροιστή (c_n =1).



Σχήμα 3.22 Παράλληλος αθροιστής n bit.

Παρατηρούμε από το Σχήμα 3.8 ότι η σχεδίαση μιας κατάλληλης βασικής μονάδας, του πλήρη αθροιστή, μας διευκολύνει στη συνέχεια να πραγματοποιήσουμε παράλληλους αθροιστές οσωνδήποτε bits. Αναφερόμενοι σ' έναν *αθροιστή n bit*, θα εννοούμε έναν παράλληλο αθροιστή με δυνατότητα ταυτόχρονης (παράλληλης) πρόσθεσης δύο δυαδικών αριθμών των n bit ο καθένας. Εάν στον αθροιστή του Σχήματος 3.8 τα bits x_i και y_i δημιουργήσουν ένα κρατούμενο c_i, τότε αυτό ενδέχεται να επηρεάσει τα επόμενα αθροίσματα μέχρι τις τελικές εξόδους s_{n-1} και c_n. Επομένως πρέπει να περιμένουμε λίγο χρόνο, μέχρι να διαδοθεί το κρατούμενο, για να πάρουμε το σωστό τελικό άθροισμα. Επειδή δεν γνωρίζουμε πότε και πού δημιουργείται κρατούμενο, είμαστε υποχρεωμένοι να περιμένουμε τη χειρότερη καθυστέρηση, δηλ. όταν το κρατούμενο πηγάζει από τα bits x_0, y_0, η οποία είναι n–πλάσια της καθυστέρησης ενός πλήρη αθροιστή. Η κυματομορφή του κρατουμένου, καθώς διαδίδεται διαδοχικά από τη μία μονάδα στην επόμενη, μοιάζει με ένα κυματάκι που εξαπλώνεται και γι' αυτό ο τύπος αυτός του παράλληλου αθροιστή ονομάζεται *κυματικός*. (Σχηματικά, τη χρονική διάδοση του κρατουμένου μπορούμε να την παρακολουθήσουμε στο ακόλουθο παράδειγμα.)

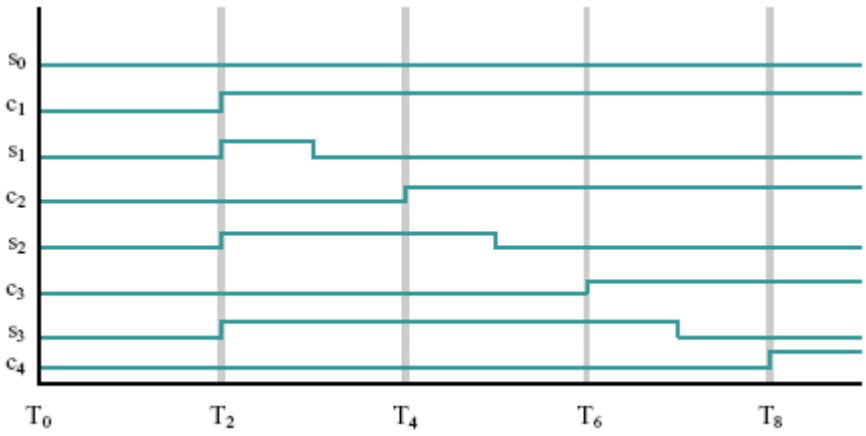
Παράδειγμα 3.3

Έστω ότι προσθέτουμε τους δυαδικούς αριθμούς 1111 και 0001 σε έναν παράλληλο αθροιστή 4 bit, όπως του Σχήματος 3.8. Εάν θεωρήσουμε ότι κάθε λογική πύλη καθυστερεί το αποτέλεσμά της κατά μία μονάδα χρόνου, τότε τα αποτελέσματα του αθροίσματος s (από τις εισόδους x_i, y_i έως s_i) και

του κρατουμένου c (από την είσοδο c_{i-1} έως c_i) σ' ένα πλήρη αθροιστή 1 bit

(Σχήμα 3.7) καθυστερούν δύο μονάδες χρόνου. Οι κυματομορφές των αντίστοιχων bit του αθροίσματος s_0, s_1, s_2, s_3 και του κρατουμένου c_1, c_2, c_3, c_4, που σχηματίζονται στον

παράλληλο αθροιστή στις διάφορες χρονικές στιγμές, από T_0 έως T_8, παρουσιάζονται στο Σχήμα 3.9.



Σχήμα 3.23 Χρονική κυμάτωση των τιμών του αθροίσματος και του κρατουμένου.

Παρατηρούμε ότι το εσωτερικό κρατούμενο (c_1, c_2, c_3) διαδίδεται σαν το μέτωπο ενός κύματος και το τελικό κρατούμενο c_4 παίρνει την οριστική τιμή του μετά από 8 μονάδες χρόνου. Εκτός από το bit s_0 που παραμένει "0", τα bits s_1, s_2, s_3 γίνονται όλα "1" στη δεύτερη χρονική στιγμή (T_2) και κατόπιν οριστικοποιούνται διαδοχικά στο "0" καθώς περνά το κρατούμενο.

3.2.2 Αθροιστής / Αφαιρετής

Στην Υποενότητα 1.1.3 αναφερθήκαμε στους προσημασμένους αριθμούς και στην παράστασή τους στο Συμπλήρωμα ως προς 2. Εξηγήσαμε ότι η αφαίρεση ενός δυαδικού αριθμού από έναν άλλον μπορεί να μετατραπεί σε πρόσθεση του αντιθέτου του και δώσαμε έναν κανόνα που κάνει εύκολη την εύρεση του αντιθέτου. Ενώ είναι γενικά δυνατό να κατασκευασθεί εξαρχής ένα κύκλωμα αφαιρέτου, είναι ευκολότερο και προτιμότερο να εκμεταλλευτούμε τις γνώσεις μας για τους αριθμούς στο Συμπλήρωμα ως προς 2 καθώς και για τα έτοιμα κυκλώματα αθροιστών, για να κατασκευάσουμε ένα ενιαίο κύκλωμα που να συνδυάζει τις πράξεις της πρόσθεσης και της αφαίρεσης. Ο αντίθετος ενός αριθμού $Y = y_{n-1} \ldots y_0$ στο Συμπλήρωμα ως προς 2 βρίσκεται (βλέπε Υποενότητα 1.1.3) εάν

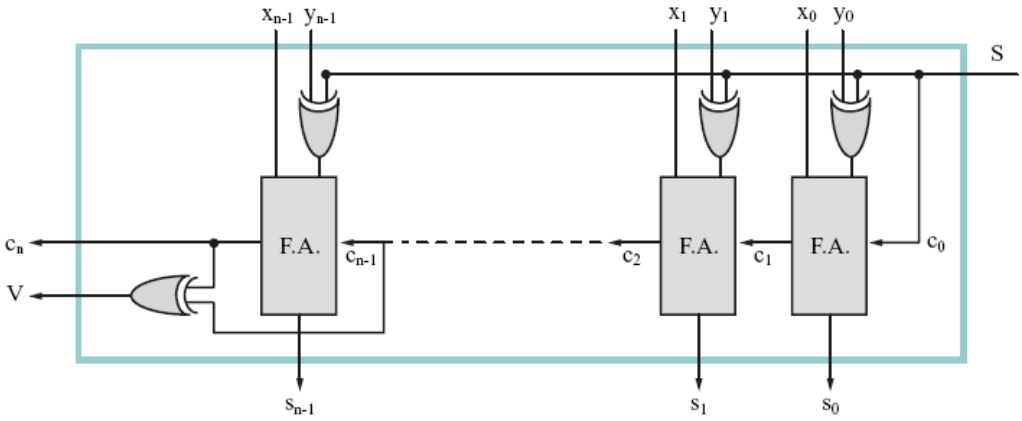
(α) αντιστρέψουμε όλα τα bit του Y και

(β) στο αποτέλεσμα προσθέσουμε το 1. Συνολικά, η αφαίρεση $Z = X–Y$,

όπου $X = {}_{xn-1} \ldots x_0$, συντομεύεται εάν γίνει η πρόσθεση:

$$
\begin{array}{cccc}
 & x_{n-1} & \cdots & x_1 & x_0 \\
 & y'_{n-1} & \cdots & y'_1 & y'_0 \\
+ & & & & 1 \\
\hline
 & z_{n-1} & \cdots & z_1 & z_0
\end{array}
$$

Η αντιστροφή των bits γίνεται πολύ εύκολα μέσω μιας λογικής πύλης **XOR** (Υποενότητα 2.6.5). Στο Σχήμα 3.10 παρουσιάζεται το πλήρες κύκλωμα του αθροιστή/αφαιρέτη. Στις εισόδους x_i (i=0, …, n–1) εισάγονται τα bits του X, στις εισόδους y_i (i=0, …, n–1) εισάγονται τα bits του Y και το ειδικό *σήμα ελέγχου* S καθορίζει εάν θα γίνει πρόσθεση (S=0) ή αφαίρεση (S=1). Όταν το σήμα ελέγχου είναι S=0, τότε παράγεται το άθροισμα X+Y, γιατί τα bits y_i (i=0, …, n–1) του αριθμού Y μεταφέρονται στον αθροιστή όπως είναι (**XOR**(y_i,0) = y_i) και επίσης c_0 = S = 0. Όταν είναι S=1, τότε παράγεται η διαφορά X–Y, γιατί τα bits y_i (i=0, …, n–1) του αριθμού Y εισάγονται αντεστραμμένα στον αθροιστή (**XOR**(y_i,1) = y'_i) και η μονάδα προστίθεται μέσω της εισόδου c_0 = S = 1. Υπενθυμίζεται ότι το πρόσημο z_{n-1} της διαφοράς Z υπολογίζεται αυτομάτως από την πρόσθεση των προσήμων x_{n-1} και y_{n-1}. Η έξοδος V σηματοδοτεί την υπερχείλιση (V=1) των προσημασμένων αριθμών.



Σχήμα 3.24 Αθροιστής / Αφαιρετής

(Στα επόμενα, αναφερόμενοι γενικά σε αθροιστές θα εννοούμε το συνδυασμό αθροιστή/αφαιρέτη.)

3.3 Σχεδίαση χωρίς Hazards

Μέχρι τώρα ασχοληθήκαμε με τη σχεδίαση στο επίπεδο της λογικής, θεωρώντας το κύκλωμα ιδανικό και αγνοώντας τυχόν "φυσικές" επιδράσεις που μπορεί να υφίστανται τα σήματα που διατρέχουν το πραγματικό κύκλωμα. Μια τέτοια επίδραση είναι η χρονική καθυστέρηση που προκαλείται, καθώς ένα σήμα περνά από μία πύλη. Η καθυστέρηση αυτή

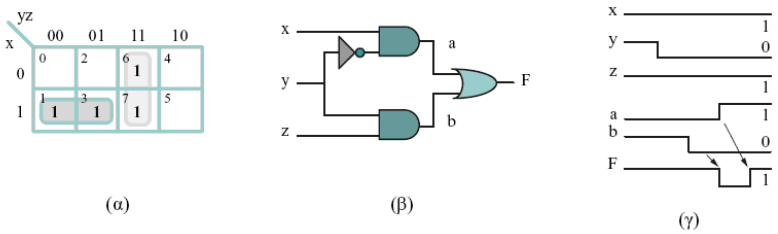
μπορεί στιγμιαία να προκαλέσει διαφορετική λογική τιμή στην έξοδο του κυκλώματος. Η διαταραχή αυτή, που ονομάζεται *Hazard*, αν και δεν αλλάζει μακροπρόθεσμα τη λογική ενός συνδυαστικού κυκλώματος, μπορεί να είναι επιζήμια σε ακολουθιακά κυκλώματα.

Κατά τη σχεδίαση ενός κυκλώματος, εκτός από την παραγωγή του ελαχίστου μεγέθους του, πρέπει να λαμβάνονται υπόψη και άλλοι παράγοντες, ώστε να εξασφαλίζεται η προβλεπόμενη λειτουργία του κυκλώματος κάτω από όλες τις περιστάσεις.

Ένας παράγοντας που μπορεί να προκαλέσει ανωμαλία στη συμπεριφορά ενός κυκλώματος κάτω από ορισμένες συνθήκες είναι το λεγόμενο *hazard*. Τα σήματα, καθώς περνούν μέσα από τις λογικές πύλες σε κάθε φυσικό κύκλωμα, υφίστανται αναπόφευκτες καθυστερήσεις. Τα hazards προκαλούνται όταν ένα σήμα διχάζεται σε κάποιο σημείο του κυκλώματος και κατόπιν, ακολουθώντας διαφορετικές διαδρομές, τα δύο παράγωγα σήματά του καταλήγουν στην ίδια πύλη. Θα εξηγήσουμε τη δημιουργία ενός hazard με το ακόλουθο παράδειγμα.

ΠΑΡΑΔΕΙΓΜΑ 3.4

Έστω το κύκλωμα του Σχήματος 3.11β, του οποίου η ελάχιστη λογική συνάρτηση είναι F(x, y, z) = xy'+yz. Τότε λογικά η συνάρτηση στην περίπτωση F(1, y, 1) = y'+y = 1 είναι ανεξάρτητη από την τιμή της y. Σ' ένα πραγματικό κύκλωμα όμως πρέπει να λάβουμε υπόψη και τις καθυστερήσεις των σημάτων. Το σήμα y (Σχήμα 3.11β), στη διαδρομή a καθυστερεί δύο μονάδες χρόνου (Σχήμα 3.11γ) ενώ στη διαδρομή b καθυστερεί μία μονάδα χρόνου. Όταν τα σήματα a και b ξανασυναντώνται στην πύλη OR, τότε λόγω των διαφορετικών καθυστερήσεων μεταξύ τους παραβιάζεται για ένα σύντομο χρονικό διάστημα, ίσο με τη διαφορά των καθυστερήσεων, η αναμενόμενη λογική σχέση F(1, y, 1) = 1 και προκύπτει για το διάστημα αυτό F(1, y, 1) = 0 (Σχήμα 3.11γ). Η εκτροπή F(1, y, 1)=0 αποτελεί ένα hazard.



Σχήμα 3.25 Δημιουργία Hazard στο κύκλωμα.

Ο απρόβλεπτος παλμός 101 που δημιουργείται στην έξοδο F, όταν λογικά (στατική κατάσταση) η F πρέπει να είναι συνεχώς 111, ονομάζεται *στατικό 1–hazard*. Στο Χάρτη Karnaugh του κυκλώματος (Σχήμα 3.11α) παρατηρούμε ότι το hazard αυτό εμφανίζεται κατά τη μετάβαση από τον minterm m3 στον m7 και αντίστροφα, οφείλεται δε στον τρόπο

που σχεδιάστηκε το κύκλωμα, δηλ. στο ότι δεν υπάρχει κοινή ομάδα που να καλύπτει τη μετάβαση ανάμεσα στους minterms m3 και m7. Γενικώς εμφανίζεται στατικό 1–hazard, όταν το κύκλωμα μεταβαίνει από ένα minterm σε γειτονικό του, που όμως ανήκει σε διαφορετική ομάδα.

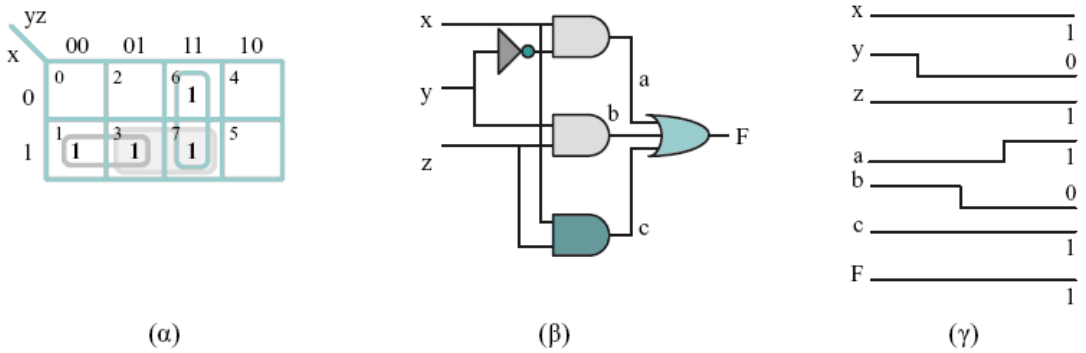
Όμοια με τα παραπάνω, ορίζουμε το *στατικό 0–hazard*, όταν εμφανίζεται στην έξοδο ο παλμός 010, ενώ η στατική κατάσταση πρέπει να είναι 000. Οφείλεται στη μετάβαση του κυκλώματος από κάποιο maxterm σε γειτονικό του, που όμως δεν καλύπτεται από κοινή ομάδα.

Η εξήγηση της αιτίας των hazards υποδεικνύει και τον τρόπο για την εξουδετέρωσή τους. Για να αποφύγουμε ένα πιθανό hazard, αρκεί να προσθέσουμε στην παράσταση της συνάρτησης επιπλέον πρώτους συνεπαγωγούς, όσοι και όποιοι χρειάζονται για να καλυφθεί κάθε μετάβαση από μία ομάδα στη γειτονική της.

ΠΑΡΑΔΕΙΓΜΑ 3.5

Θα διορθώσουμε τα hazards του κυκλώματος του Παραδείγματος 3.4 (Σχήμα 3.11).

Απάντηση: Στο Σχήμα 3.12β παρουσιάζεται το διορθωμένο κύκλωμα. Η λογική συνάρτηση του νέου κυκλώματος είναι F(x, y, z) = xy'+yz+xz (Σχήμα 3.12α). Με την προσθήκη του πλεονάζοντος πρώτου συνεπαγωγού xz καλύπτεται η μετάβαση μεταξύ m3 και m7 και αποφεύγεται το στατικό 1–hazard. Η χρονική συμπεριφορά του νέου κυκλώματος (έξοδος F) φαίνεται από την ανάλυση των κυματομορφών του Σχήματος 3.12γ.



Σχήμα 3.26 Κύκλωμα χωρίς Hazard.

Με την προσθήκη πλεοναζόντων όρων δημιουργείται ένα λογικά ισοδύναμο κύκλωμα, που δεν είναι μεν το ελάχιστο δυνατό, συμπεριφέρεται όμως καλύτερα. Γενικώς, τα hazards στα συνδυαστικά κυκλώματα δεν είναι τόσο κρίσιμα ως προς την λειτουργία του κυκλώματος, γιατί το κύκλωμα με την πάροδο μικρού χρόνου θα σταθεροποιηθεί στη σωστή κατάστασή του. Αποτελούν, όμως, σοβαρό πρόβλημα για τα ακολουθιακά κυκλώματα, στα οποία λανθασμένοι παλμοί μπορεί να τα οδηγήσουν σε λανθασμένες καταστάσεις, από τις οποίες είναι αδύνατον να επανέλθουν στην ορθή πορεία τους.

Στο κεφάλαιο αυτό θα παρουσιαστούν διάφορες εικόνες-αποσπάσματα (screenshots) του περιβάλλοντος του προγράμματος με σκοπό την γνωριμία και την περιήγηση σε αυτό καθώς και την παρουσίαση των δυνατοτήτων του.

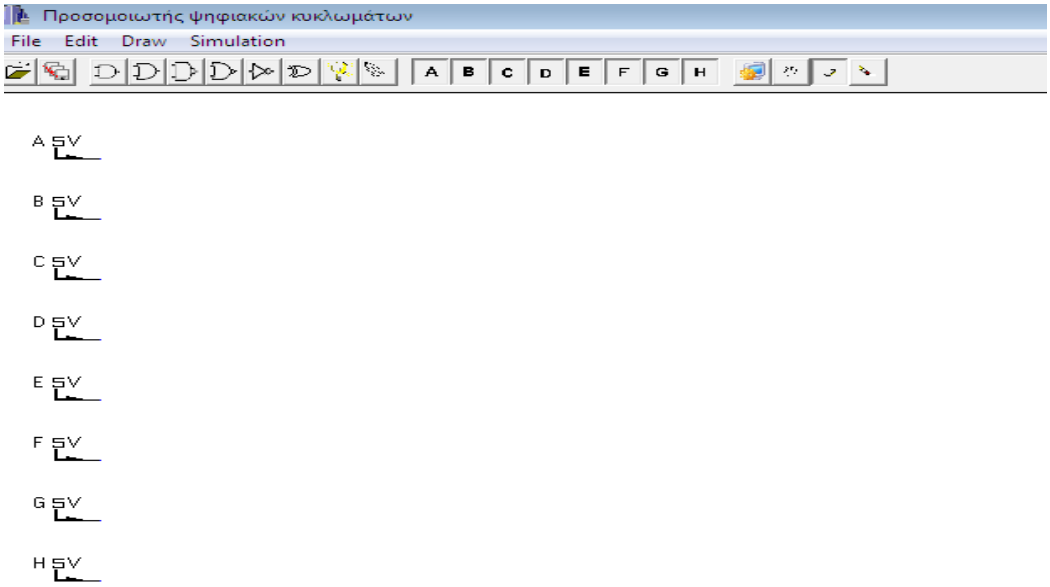
Όπως αναφέρθηκε και στην εισαγωγή του συγγράμματος πρόκητε για ένα Εξομοιωτή Ψηφιακών Κυκλωμάων με σκοπό την διευκόλυνση του χρήστη των παραδοσιακών - φυσικών μεθόδων εξομοίωσης ψηφιακών Κυκλωμάτων σε εργαστήρια ηλεκτρονικών.

ΚΕΦΑΛΑΙΟ 4

4.1 Περιβάλλον του προγράμματος Εξομοιωτής Ψηφιακών Κυκλωμάτων.

Αρχικά θα συναντήσουμε την εικόνα (Εικόνα 4.1) του αρχικού παραθύρου του προγράμματος το οποίο θα είναι και το περιβάλλον εργασίας και υλοποίησης των εξομοιώσεων του εκάστοτε χρήστη. Στο αρχικό περιβάλλον έχουν τοποθετηθεί οκτώ (8) πηγές-είσοδοι οι οποίες θα χρησιμοποιηθούν για να δίνονται οι είσοδοι στις διάφορες πύλες.

Εικόνα 4.1 Αρχικό περιβάλλον



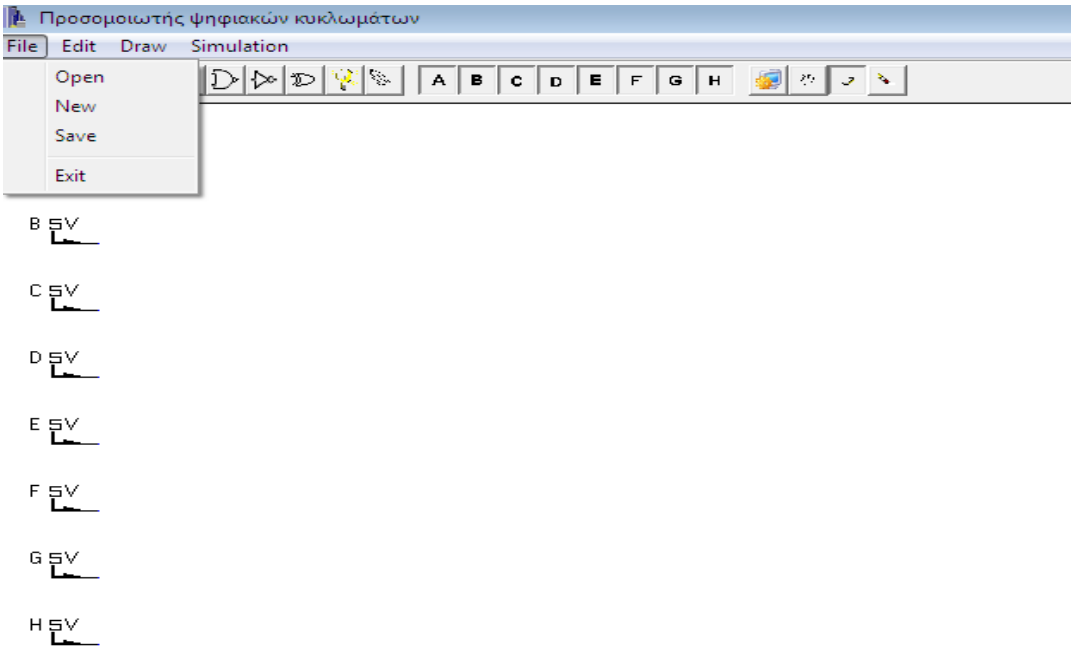
Παρατηρώντας την εικόνα εντοπίζουμε στο πάνω μέρος του προγράμματος την μπάρα των Μενού **File, Edit, Draw, Simulation** από την οποία μπορούν να γίνουν οι επιλογές σχεδίασης, επεξεργασίας και προσομοίωσης των Κυκλωμάτων**.**

Ακριβώς κάτω από την μπάρα των μενού, εντοπίζονται διάφορα κουμπιά, τα οποία τοποθετήθηκαν με σκοπό την ταχύτερη διαχείριση των επιλογών και δυνατοτήτων του προγράμματος και αντιστοιχούνται με αυτές των επιλογών της μπάρας του Μενού. Ανάλυση των αντιστοιχιών Μενού και κουμπιών ακολουθεί παρακάτω.

4.1.1 Μενού File και αντιστοιχία των κουμπιών.

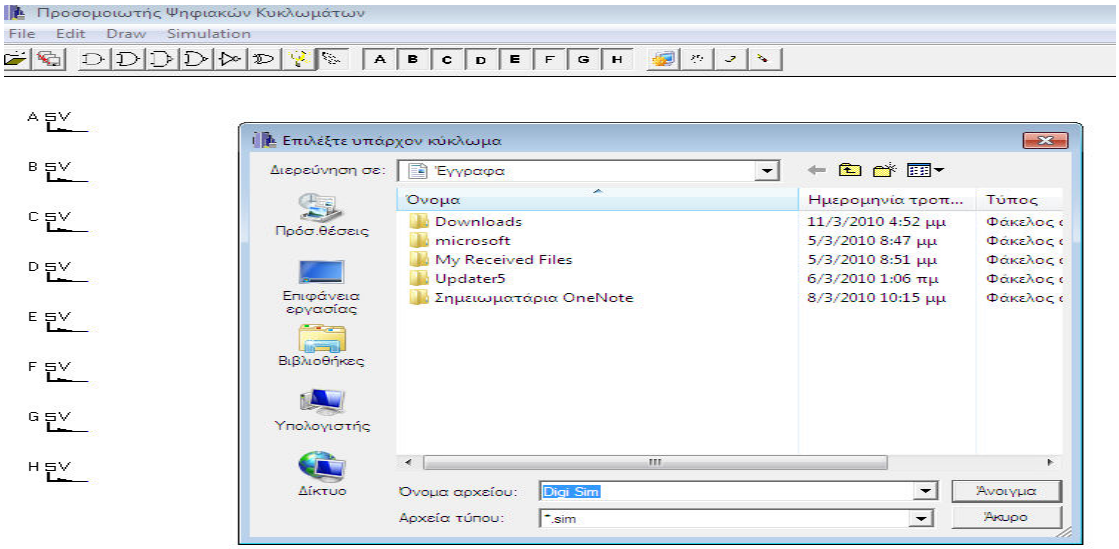
Στην εικόνα 4.2 βλέπουμε σε ανάπτυξη την επιλογή File του Μενού. Εμφανίζονται οι εντολές **Open, New, Save** και **Exit.**

Εικόνα 4.2 Μενού File.



Χρησιμοποιώντας την εντολή **Open,** ή σε αντιστοιχία το κουμπί  μας δίνετε δυνάτοτητα να επιλέξουμε και να ανοίξουμε μέσω ενός παραθύρου ένα υπάρχων κύκλωμα το οποίο έχουμε σχεδιάση παλαιότερα, όπως βλέπουμε και παρακάτω στην εικόνα 4.3

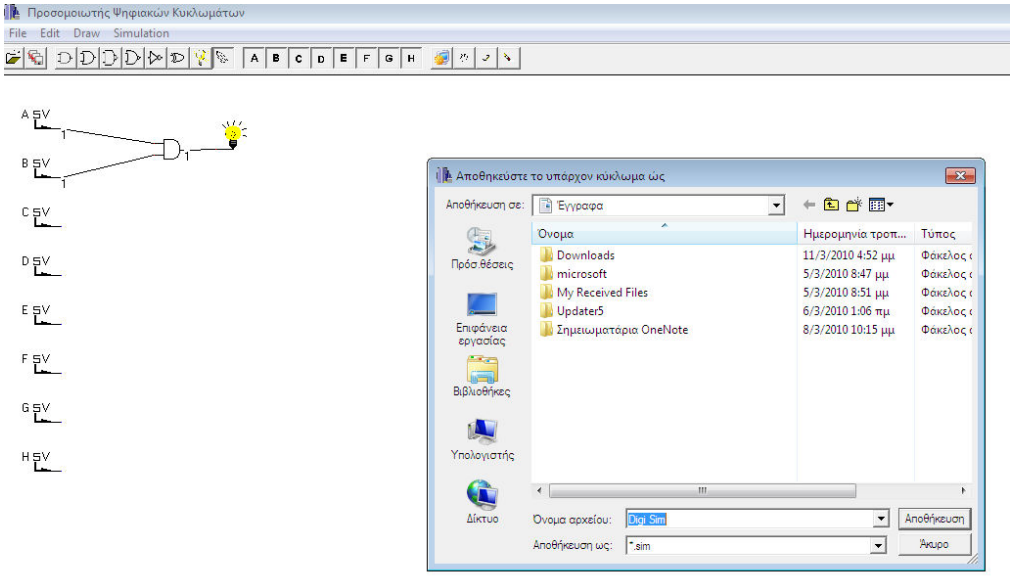
Εικόνα 4.3 Εντολή Open.



Συνεχίζοντας συναντούμε την εντολή **New,** με την οποία ξεκινούμε ένα νέο κύκλωμα από την αρχή παραθέτοντας μας στο αρχικό περιβάλλον του προγράμματος όπως είδαμε στην Εικόνα 4.1.

Με την επόμενη εντολή, την εντολή **Save** ή σε αντιστοιχία το κουμπί  , σώζουμε μέσω ενός παραθύρου το οποιοδήποτε κύκλωμα που έχουμε ήδη σχεδίασει οπουδήποτε θέλουμε. Το παράθυρο αυτό εμφανίζεται παρακάτω στη εικόνα 4.4.

Εικόνα 4.4 Εντολή Save.

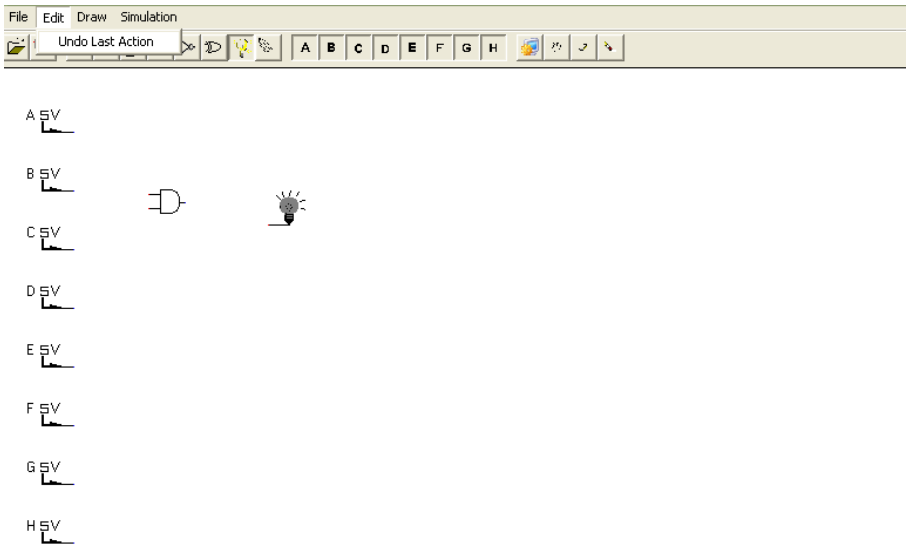


Τέλος με την εντολή **Exit** κλείνουμε το πρόγραμμα μας.

4.1.2 Μενού Edit.

Το μενού **Edit** περιλαμβάνει την εντολή **Undo Last Action** (Εικόνα 4.5) η οποία μας δίνει τη δυνατότητα να ακυρώσουμε την οποιαδήποτε τελευταία ενέργεια και μόνο.

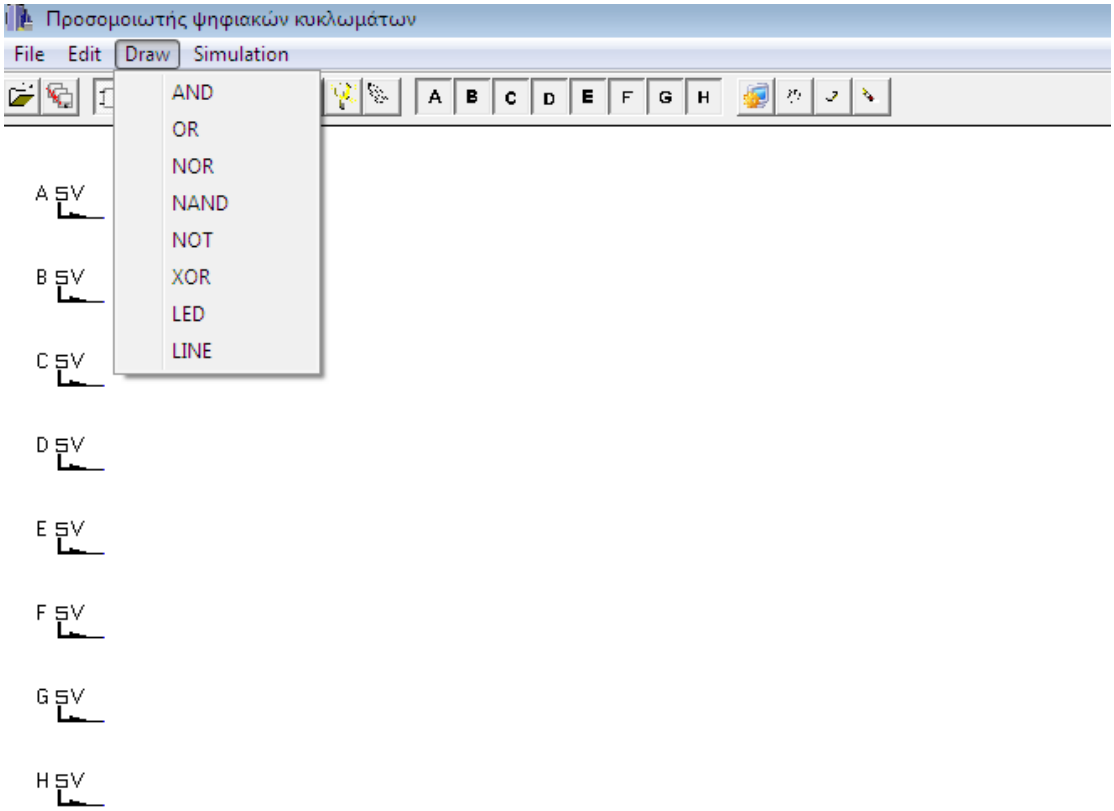
Εικόνα 4.5 Εντολή Undo Last Action.



4.1.3 Μενού Draw και αντιστοιχία των κουμπιών.

Με το μενού **Draw** (Εικόνα 4.6) πλέον μας δίνεται η δυνατότητα να επιλέξουμε ανάμεσα στις λογικές πύλες **AND**, **OR**, **NOT, NOR, NAND, XOR,** της τελικής εξόδου του κυκλώματος για τη οποία χρησιμοποιήτε ένα λαμπάκι τύπου **LED** και της σχεδίασης γραμμής-καλωδίου μέσω της εντολής **LINE** και έτσι να ξεκινήσουμε με την σχεδίαση και υλοποίηση διαφόρων ψηφιακών Κυκλωμάτων.

Εικόνα 4.6 Μενού Draw.

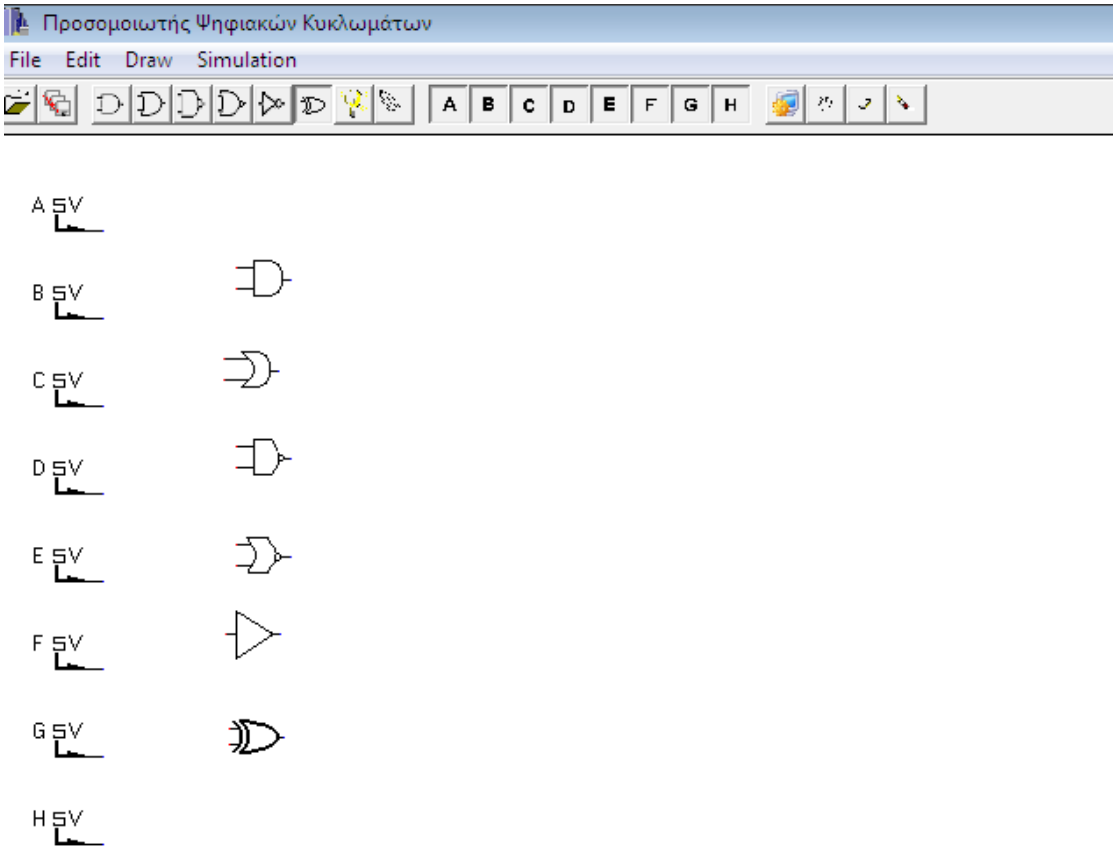


Επιλέγοντας την εντολή **AND,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη **AND** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη **AND.**

Επιλέγοντας την εντολή **OR,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη **OR** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη **OR.**

Επιλέγοντας την εντολή N**OR,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη N**OR** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη N**OR.**

Εικόνα 4.7 Πύλες του προγράμματος Προσομιωτής Ψηφιακών Κυκλωμάτων.



Επιλέγοντας την εντολή **NAND,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη **NAND** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη **NAND.**

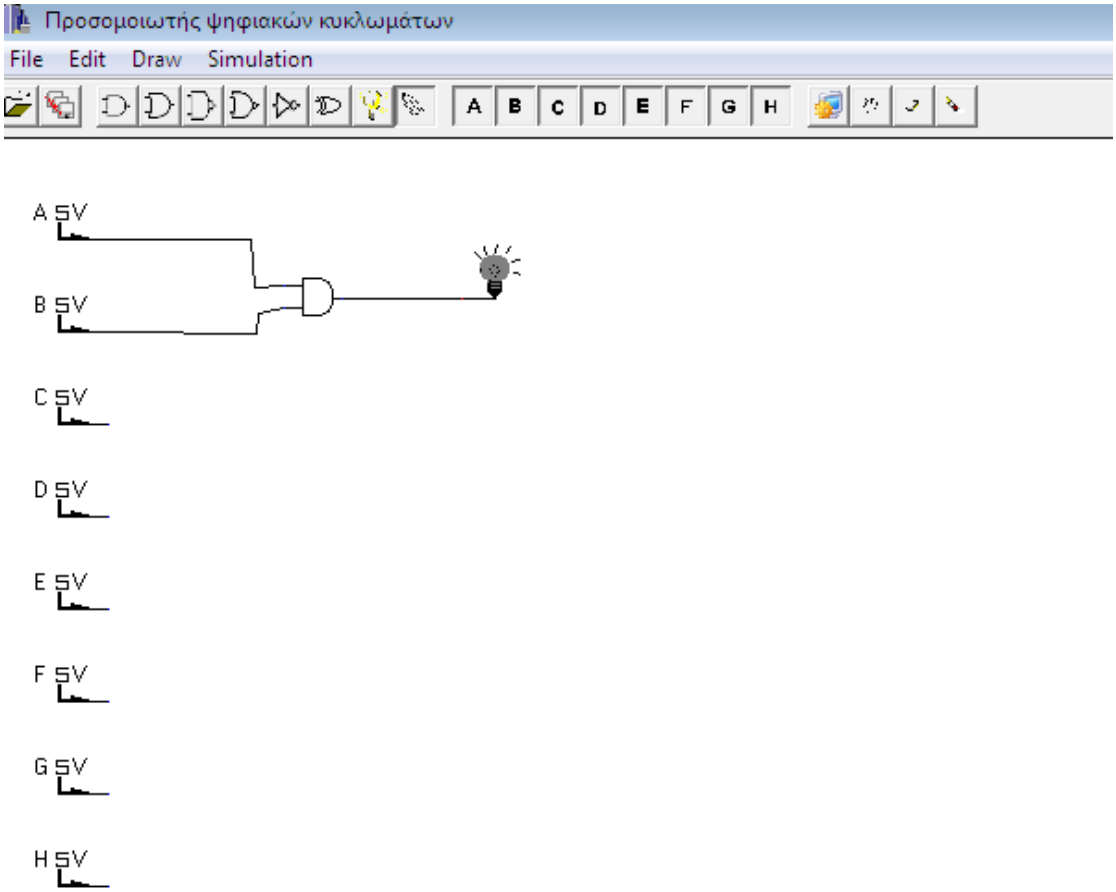
Επιλέγοντας την εντολή **NOT,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη **NOT** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη **NOT.**

Επιλέγοντας την εντολή **XOR,** η σε αντιστοιχία το κουμπί  , ο κέρσορας μας μετατρέπεται σε μια πύλη **XOR** και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι μία πύλη **XOR.**

Στην εικόνα (Εικόνα 4.7), εμφανίζονται κατά σειρά επιλογής, από ένα αντίγραφο της κάθε πύλης τοποθετημένα μέσα στο πρόγραμμα.

Επιλέγοντας την εντολή **LED,** η σε αντιστοιχία το κουμπί , ο κέρσορας μας μετατρέπεται σε ένα **LED** σβηστό και μπορούμε πλέον να σχεδιάσουμε οπουδήποτε μέσα στο πρόγραμμα κάνοντας ένα κλίκ με το ποντίκι ένα **LED.** Στην παρακάτω εικόνα (Εικόνα 4.8) εμφανίζεται ένα **LED** συνδεδεμένο με μια πύλη **AND.**

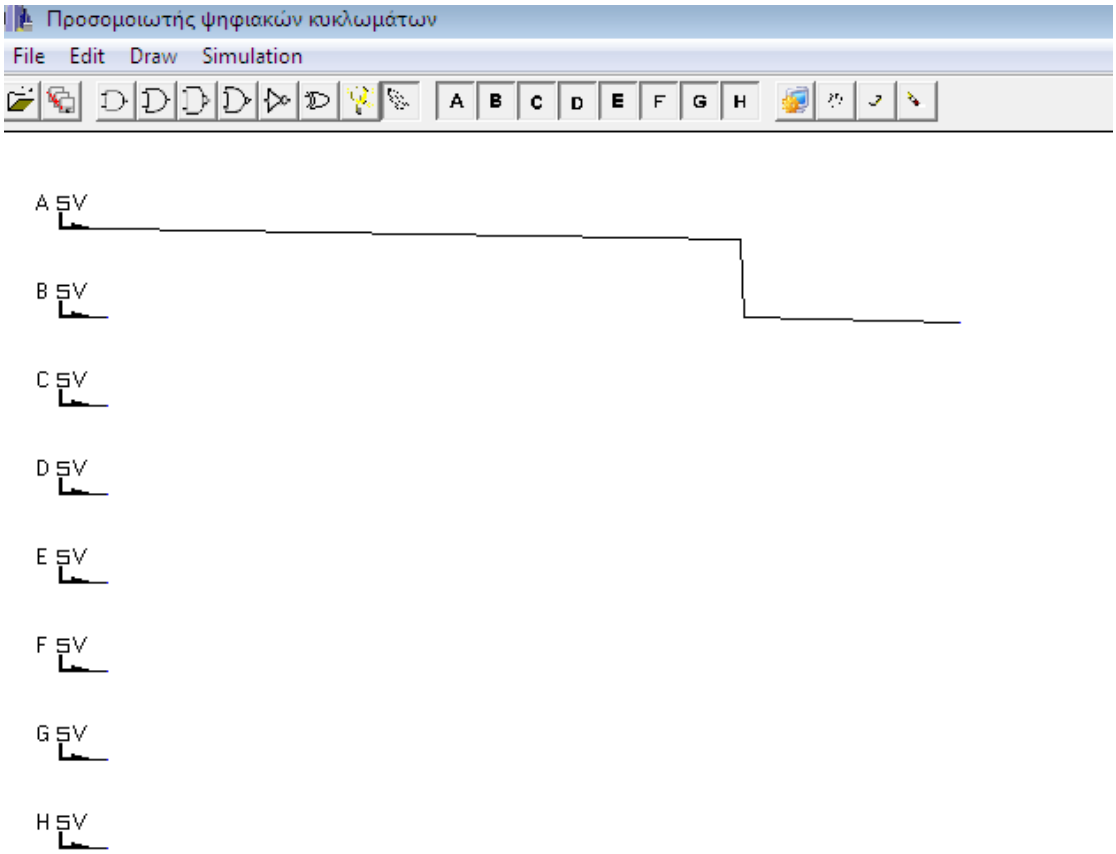
Εικόνα 4.8 Παράδειγμα LED και AND.



Τέλος επιλέγοντας την εντολή **LINE,** η σε αντιστοιχία το κουμπί , ο κέρσορας μας μετατρέπεται σε ένα σταυρό + και μπορούμε πλέον να σχεδιάσουμε γραμμές-καλώδια κάνοντας ένα κλίκ με το ποντίκι σε οποιαδήποτε είσοδο, πύλη ή **LED** συνδεόντας τα μεταξύ τους και υλοποιόντας το κύκλωμα της επιθυμίας μας**.** Όταν σύρουμε το ποντίκι μας σε κάποιο σημείο έναρξης ή τερματισμού μιας γραμμής-καλωδίου, ο κέρσορας μας μετατρέπεται σε . Στην Εικόνα 4.8 βλέπουμε μια πύλη **AND** συνδεδεμένη με γραμμές από τις εισόδους της με της πηγές A και B και από την έξοδο της με ένα **LED** μέσω γραμμών-καλωδίων.

Αν θέλουμε να σχεδιάσουμε μια τεθλασμένη γραμμή-καλώδιο αρκεί πριν φτάσουμε στο σημείο τερματισμού της γραμμής, το οποίο μπορεί να είναι η είσοδος μιας πύλης ή ενός **LED,** κάνουμε κλίκ μέσα στο πρόγραμμα και ακολουθούμε διαφορετική κατεύθυνση από την προηγούμενη. Σαν αποτέλεσμα θα έχουμε την εικόνα 4.9

Εικόνα 4.9 Παράδειγμα τεθλασμένης γραμμής-καλωδίου.



4.1.4 Μενού Simulation και αντιστοιχία των κουμπιών.

Με το Μενού **Simulation** και την εντολή του **Start** (Εικόνα 4.10)**,** ή το κουμπί  , μπορούμε αφού έχουμε τελειώσει με την σχεδίαση του κυκλώματος μας να προχωρήσουμε στην προσομοίωση του και να πάρουμε τις μετρήσεις και τα αποτελέσματα μας.

Αξίζει να σημειωθεί ότι με την προσομοίωση του κυκλώματος εμφανίζονται οι τιμές, ανάλογα 0 ή 1 σε κάθε έξοδο πηγής, πύλης και **LED**, όπως βλέπουμε και παρακάτω στην εικόνα 4.11.

4.2 Τα κουμπιά του προγράμματος Εξομοιωτή Ψηφιακών Κυκλωμάτων και οι λειτουργιές τους.

Στο αρχικό περιβάλλον του προγράμματος (Εικόνα 4.1) εμφανίζονται μια σειρά από κουμπιά ακριβώς κάτω από τη μπάρα του Μενού.

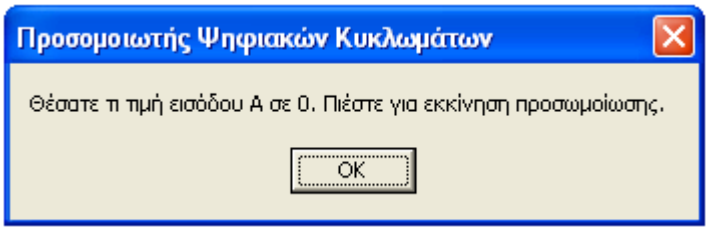
4.2.1 Κουμπιά που αντιστοιχούν σε εντολές του Μενού.

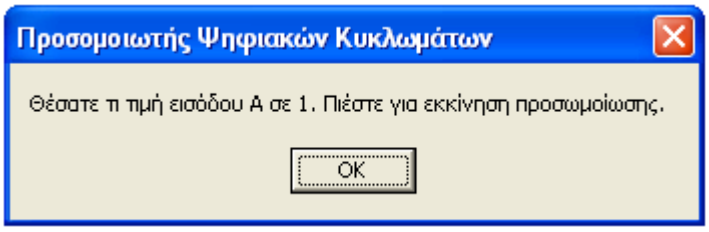
Όπως είδαμε παραπάνω κάποια από τα κουμπιά αντοιστοιχούν σε εντολές του Μενού. Ύπαρχουν τα κουμπία διαχείρησης του προγάμματος όπως είναι το κουμπί  για το **Open** και το κουμπί  για το **Save**. Επίσης υπάρχουν τα κουπία για την επιλογή σχεδίασης των πυλών, όπως είναι το κουμπί  για την πύλη **AND**, το κουμπί  για την πύλη **OR**, το κουμπί  για την πύλη **NAND**, το κουμπί  για τη πύλη **NOR**, το κουμπί  για την πύλη **NOT** και το κουμπί  για την πύλη **XOR**, καθώς και τα κουμπί  για την σχεδιάση **LED** και  το κουμπί για τη σχεδίαση των γραμμών. Τέλος να μην ξεχάσουμε το κουμπί  για την έναρξη της προσομοίωσης. Στη συνέχεια θα αναφερθούμε στα υπόλοιπα κουμπία που εμφανίζονται στο περιβάλλον του προγράμματος.

4.2.2 Κουμπιά ελέγχου των πηγών-εισόδων.

Στο αρχικό περιβάλλον του προγράμματος (Εικόνα 4.1) εμφανίζονται 8 (οκτώ) (**A, B, C, D, E, F, G** και **H**) πηγές-είσοδοι με τάση 5V, δηλαδή μας δίνουν είσοδο 1. Για να αλλάξουμε την τάση τους και μαζί την έξοδου από 5V σε 0V, δηλαδή από 1 σε 0, δεν έχουμε παρά να πατήσουμε πάνω στο κουμπί που έχει το ίδιο όνομα με την πύλη-είσοδο της οποίας τάση θέλουμε να αλλάξουμε. Όταν επιλέξουμε τάση 5V ή έξοδο 1 η πηγή εμφανίζεται ως  και όταν επιλέξουμε τάση 0V ή έξοδο 0 η πηγή εμφανίζεται ως  .

Κάθε φορά που κάνουμε μια τέτοια αλλάγη σε μια πηγή εμφανίζεται και το αντίστοιχο από τα παρακάτω μηνύματα για την κάθε πηγή ππου αλλάζουμε€, προειδοποιώντας μας ότι θα πρέπει να ξανακάνουμε προσομοίωση το κύκλωμα.



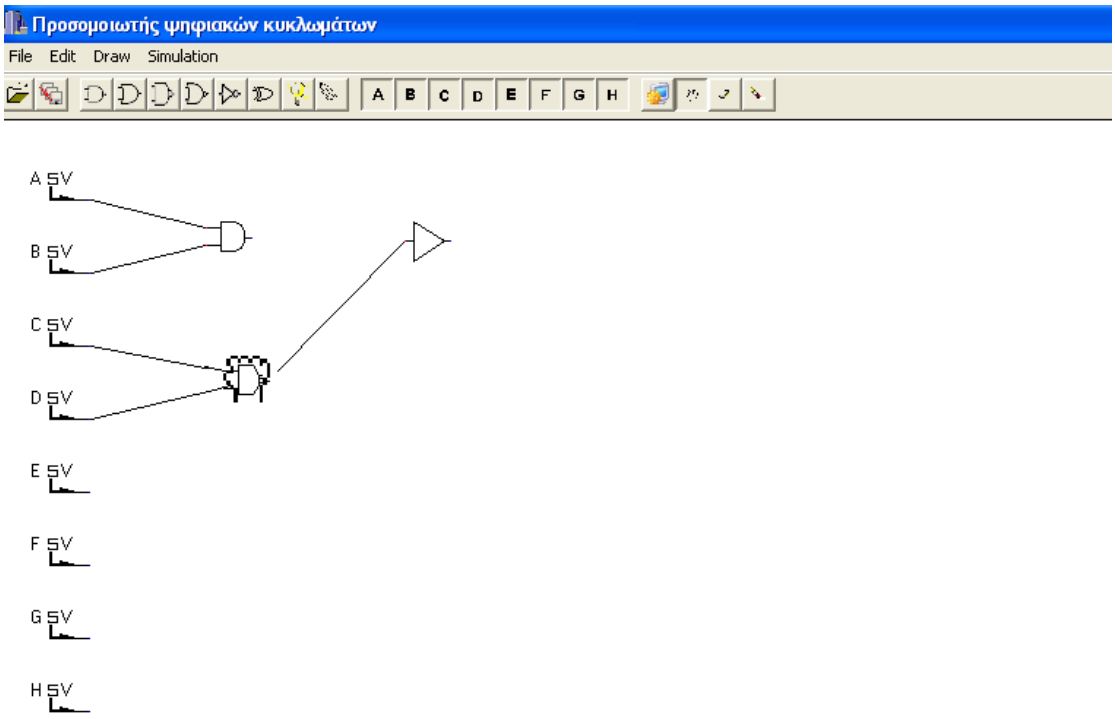


4.2.3 Κουμπία Move, Erase Gate και Erase Line.

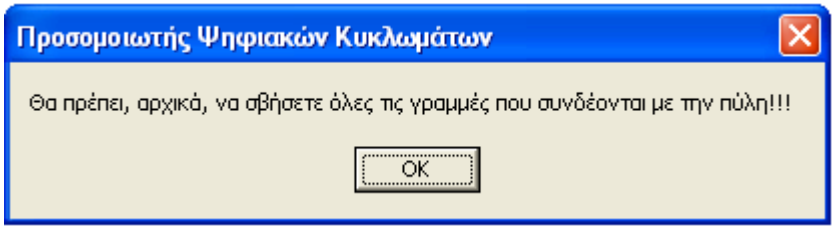
Τα υπόλοιπα κουμπία που εμφανίζονται στο πρόγραμμα Εξομοιωτής Ψηφιακών Κυκλωμάτων είναι τα **Move Gate** , **Erase Gate** , και **Erase Line** .

Με το κουμπί **Move Gate** έχουμε την δυνατίτητα να μετακινήσουμε οποιαδήποτε πύλη ή **LED** θέλουμε οπουδήποτε θέλουμε μέσα στο κύκλωμα που σχεδιάζουμε είτε αυτά είναι συνδεδεμένα με γραμμές-καλώδια είτε όχι. Αφού πατήσουμε αυτό το κουμπί ο κέρσορας του ποντικιού μας μετατρέπεαι σε ένα χέρι ( ). Στη συνέχεια κάνουμε ένα κλίκ στη πύλη ή **LED** που θέλουμε να μετακινήσουμε, αυτομάτος ο κέρσορας μας μετατρέπεται σε ένα χέρι που κρατάει την πύλη ή **LED** (Εικόνα 4.12) που έχουμε επιλέξει. Χώρίς να κρατάμε πατημένο το κλίκ του ποντικιου, δηλαδή δεν κάνουμε **Grab** and **Drop**, κάνουμε ένα κλίκ στο νέο σημείο που θέλουμε να τποθετήσουμε την πύλη ή **LED** μας. Αν η πύλη ή **LED** ήταν συνδεδεμένη με γραμμές-καλώδια οι συνδέσεις παραμένουν μεγαλώνοντας τις γραμμές ή μικραίνοντας τις ανάλογα.

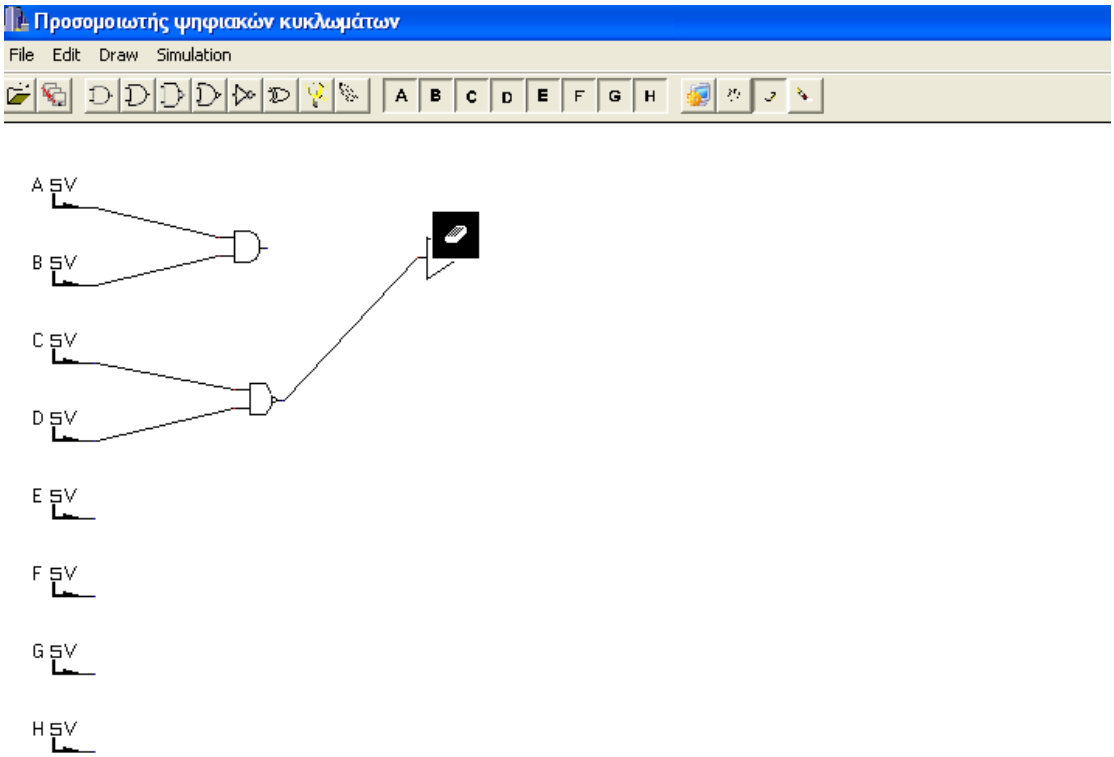
4.12 Παράδειγμα Move Gate μιας πύλης NAND.



Το επόμενο κουμπί που συναντούμε είναι το **Erase Gate** , με το οποίο έχουμε την δυνατότητα να διαγράψουμε όποια πύλη ή **LED** επιθυμούμε. Μόλις πατήσουμε το κουμπί αυτό ο κέρσορας του ποντικιού μας μετατρέπετε σε μια γόμα, κάπως έτσι . Μόλις σύρουμε το ποντίκι μας πάνω σε μια πύλη ή **LED** μετατρέπετε σε που μας δείχνει ότι μπορούμε να σβήσουμε την πύλη ή **LED** που επιθυμούμε με ένα κλίκ. Αν η πύλη ή **LED** που επιθυμούμε να σβήσουμε είναι συνδεδεμένη με γραμμές-καλώδια και κάνουμε κλίκ πάνω της, τότε θα μας εμφανιστεί το παρακάτω μήνυμα.

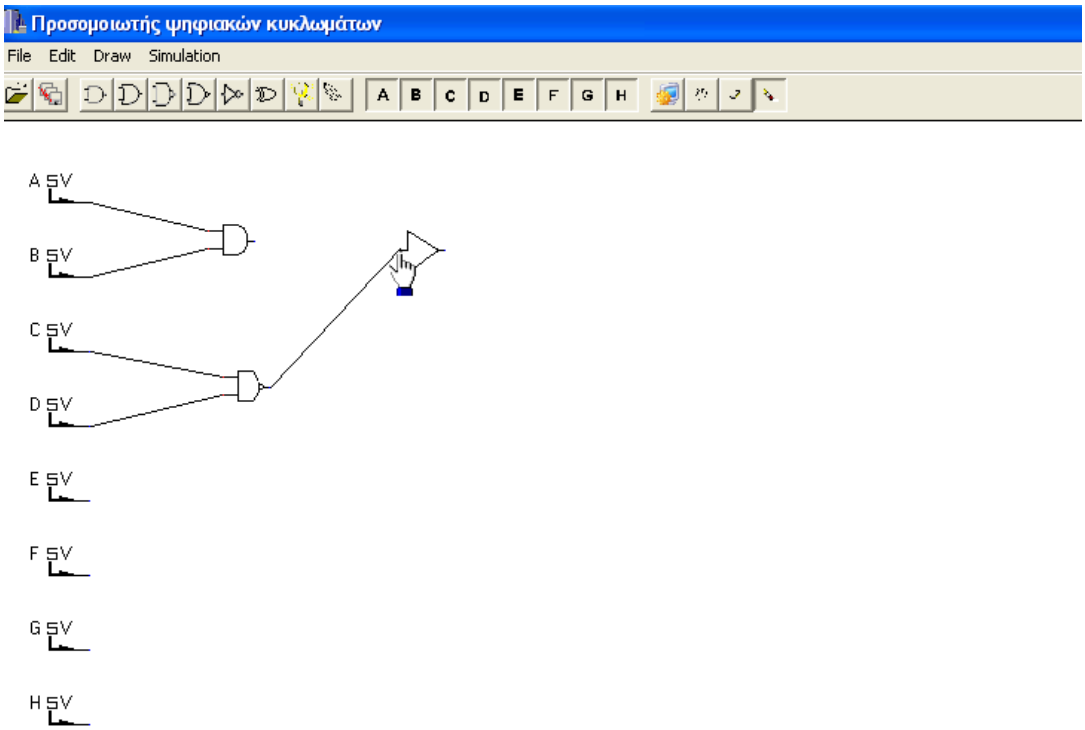


Εικόνα 4.13 Παράδειγμα Erase Gate.



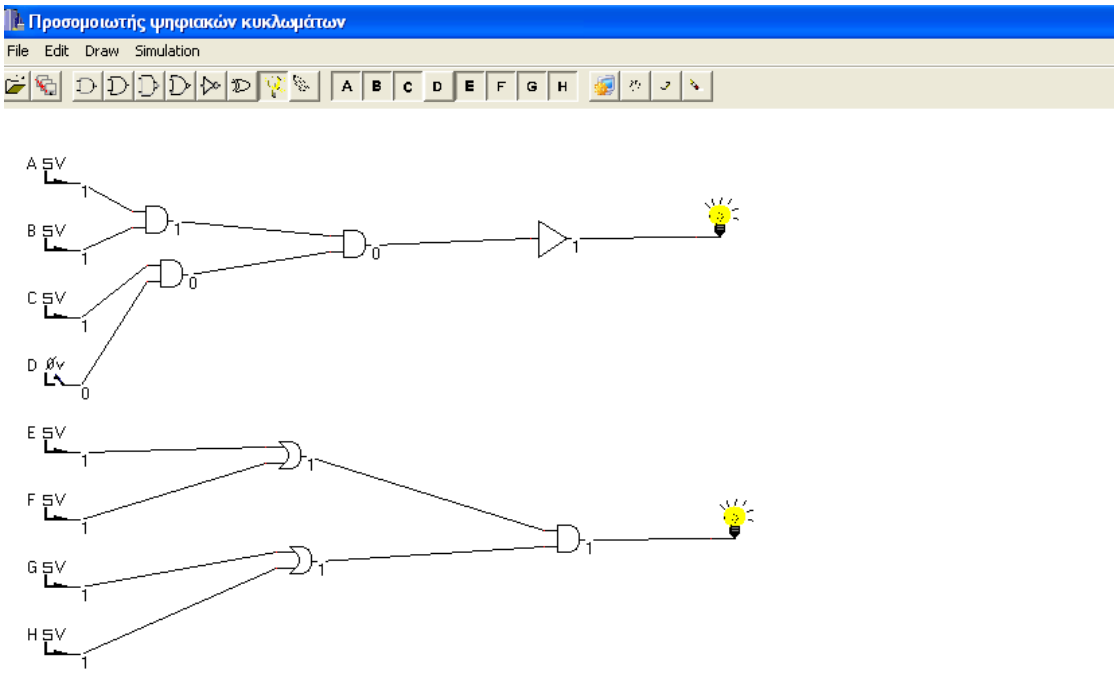
Αυτό το μήνυμα μας παραπέμπτει στη διαγραφή μιας γραμμής-καλωδίου,το τελευταίο μας κουμπί. Επιλέγουμε το κουμπί  και ο κέρσορας μας μετατρέπεται σε  . Για να διαγράψουμε μια γραμμή θα πρέπει να σύρουμε το ποντίκι μας σε καποιο σημείο έναρξης ή τερματισμού μιας γραμμής-καλωδίου που θέλουμε να διαγράψουμε, αυτό το καταλαβαίνουμε γιατί ο κέρσορας μετατρέπεται σε  (Εικόνα 4.14), και κάνοντας ένα κλίκ η γραμμή διαγράφεται.

Εικόνα 4.14 Παράδειγμα Erase Line.



Στην Εικόνα 4.15 βλέπουμε ένα αποτέλεσμα προσομοίωσης ψηφιακού κυκλώματος στο οποίο έχουν χρησιμοποιηθεί οι περισσότερες από τις δυνατότητες του προγράμματος.

Εικόνα 4.15 Παράδειγμα προσομοίωσης ψηφιακού κυκλώματος.


Προσομοιωτής ψηφιακών κυκλωμάτων
File Edit Draw Simulation
A B C D E F G H
A 5V
B 5V
C 5V
D 0V
E 5V
F 5V
G 5V
H 5V

Στο κεφάλαιο αυτό θα γίνει μια πλήρη παρουσίαση, αναφορά και επεξήγηση του κώδικα που συγγράφηκε με την γλώσσα προγραμματισμού C++ Builder 6 για να δημιουργηθεί το πρόγραμμα Εξομοιωτής Ψηφιακών Κυκλωμάτων. Η σειρά που παρουσιάζεται και επεξηγήτε ο κώδικας είναι με τη σειρά εμφάνισης στο C++ Builder 6.


```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "DigiObject.h"
#include "LineObject.h"
#include "Resource.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
```

Η C++ Builder διαθέτει (εξ ορισμού) τον πίνακα Cursors, στον οποίο μπορούμε να αποθηκεύσουμε όλους τους δείκτες ποντικιού που μπορεί να χρησιμοποιήσει η εφαρμογή μας. Κατά συνέπεια, για να μπορέσουμε να αλλάζουμε δείκτες ποντικιού δεν έχουμε παρά να επιλέξουμε την αντίστοιχη θέση του πίνακα Cursors. Για παράδειγμα, αν θέλουμε να χρησιμοποιήσουμε τον 1ο διαθέσιμο δείκτη, θα χρησιμοποιήσουμε το κελί Cursors[0], ενώ αν θέλουμε να χρησιμοποιήσουμε τον 2ο διαθέσιμο δείκτη, θα χρησιμοποιήσουμε το κελί Cursors[1]. Η μεταβλητή crMyCursor θα μας χρησιμεύσει για αυτή την αλλαγή του δείκτη στο κελί του πίνακα Cursors.

```
const crMyCursor = 5;
```

Πίνακας από αντικείμενα DigiObject, στα οποία αποθηκεύουμε τις πύλες. Συνολικά μπορούμε να βάλουμε ως και 110 πύλες.

```
DigiObject Pili[110];
```

Μέγιστος αριθμός πυλών, μπορεί ελεύθερα να αυξηθεί.

```
int maxobjects=110;
```

Τρέχον πλήθος πυλών (στην αρχή είναι 0).

```
int arithmospilon=0;
```

Πίνακας από αντικείμενα LineObject, στα οποία αποθηκεύουμε τις γραμμές.

Συνολικά μπορούμε να βάλουμε ως και 200 γραμμές.

```
LineObject Lines[200];
```

Μέγιστος αριθμός γραμμών (διασυνδέσεων), μπορεί ελεύθερα να αυξηθεί.

```
int maxlines=200;
```

Τρέχον πλήθος γραμμών (στην αρχή είναι 0).

```
int arithmosgrammon=0
```

Όταν κινείται το ποντίκι εντός της εφαρμογής, ενεργοποιείται το event mouse move της πλακέτας.

Στις μεταβλητές Xtoadd και Ytoadd αποθηκεύονται οι συντεταγμένες του ποντικιού.

```
int Xtoadd=0;
int Ytoadd=0;
```

Η μεταβλητή ZografiseGrammi χρησιμοποιείται για να καταλαβαίνει το πρόγραμμα αν βρίσκεται σε κατάσταση σχεδίασης γραμμής (true, όταν έχουν βρεθεί και η αρχή και το τέλος της γραμμής) ή όχι (false).

```
bool ZografiseGrammi=true;
```

Η μεταβλητή EndiamesesGrammes χρησιμοποιείται για να καταλαβαίνει το πρόγραμμα αν βρίσκεται σε κατάσταση σχεδίασης τεθλασμένων γραμμών (true) ή όχι (false).

Πολλές φορές είναι δυνατόν να μην συνδεθεί απευθείας μια πύλη με μια άλλη πύλη, αλλά η γραμμή που τις συνδέει να ακολουθεί κάποιο μονοπάτι. Σε αυτή την περίπτωση χρησιμοποιούμε την τεθλασμένη γραμμή.

```
bool EndiamesesGrammes=false;
```

Όταν κάνουμε αριστερό κλικ σε κάποιο σημείο του παραθύρου, ενεργοπείται το event mouse down της εφαρμογής. Σε αυτή την περίπτωση, το πρόγραμμα ελέγχει αν το pixel στο οποίο κάναμε κλικ είναι μπλε (σημάδι ότι βρέθηκε έξοδος πύλης).

Αν υπάρχει, οι συντεταγμένες του pixel αυτού αποθηκεύονται στις μεταβλητές XGrammiArxi και YGrammiArxi.

```
int XGrammiArxi=0;
int YGrammiArxi=0
```

Οι 4 παρακάτω μεταβλητές είναι παρόμοιες με τις XGrammiArxi και YGrammiArxi, αλλά χρησιμοποιούνται για τις τεθλασμένες γραμμές.

```
int arxi_XEndiamesiGrammiArxi=0;
```

```
int arxi_YEndiamesiGrammiArxi=0;
int XEndiamesiGrammiArxi=0;
int YEndiamesiGrammiArxi=0
```

Τρέχον πλήθος τεθλασμένων γραμμών.

```
int plithos_tethlasmenon=0;
```

Μπορούμε να σχεδιάσουμε ως και 2000 τεθλασμένες γραμμές. Κάθε τεθλασμένη γραμμή αποτελείται από ένα σύνολο μικρότερων ευθειών. Ο πίνακας αυτός περιέχει τα πλήθη αυτών των μικρότερων ευθειών (Αν π.χ. η 1η τεθλασμένη γραμμή αποτελείται από 4 μικρότερες ευθείες, θα ισχύει index_tethlasmenis[0]=4.

```
int index_tethlasmenis[2000];
```

Κάθε τεθλασμένη γραμμή μπορεί να αποτελείται από, το πολύ, 50 μικρότερες ευθείες.

(Μια ευθεία αποτελείται από 2 σημεία, άρα 50 ευθείες αποτελούνται από 51 σημεία.)

Ο πίνακας αυτός περιέχει τις συντεταγμένες των σημείων που σχηματίζουν την εκάστοτε τεθλασμένη γραμμή.

```
int pinakas_tethlasmenis[2000][51][2];
```

Ενεργοποιεί την κατάσταση διαγραφής πυλών.

```
bool for_erase=false;
```

Ενεργοποιεί την κατάσταση διαγραφής γραμμών.

```
bool line_for_erase=false;
```

Ενεργοποιεί την κατάσταση διαγραφής τεθλασμένων γραμμών.

```
bool tethlasmeni_for_erase=false;
```

Ενεργοποιεί την κατάσταση μετακίνησης πυλών.

```
bool for_move=false;
```

Ποια πύλη θα διαγραφτεί; (αρχική τιμή=-1, δηλ. δεν διαγράφεται καμία πύλη).

```
int index_for_erase=-1;
```

Ποια γραμμή θα διαγραφτεί; (αρχική τιμή=-1, δηλ. δεν διαγράφεται καμία γραμμή) .

Ποια τεθλασμένη θα διαγραφτεί; (αρχική τιμή=-1, δηλ. δεν διαγράφεται καμία τεθλασμένη).

Ποια πύλη θα μετακινηθεί; (αρχική τιμή=-1, δηλ. δεν διαγράφεται καμία πύλη).

Αλφαριθμητικό που περιέχει τον φάκελο στον οποίο βρίσκεται η εφαρμογή μας.

Χρησιμοποιείται για να φορτώσουμε τις εικόνες.

Τρέχουσα πύλη.

```
int line_index_for_erase=-1;
```

```
int tethlasmeni_index_for_erase=-1;
int index_for_move=-1;
AnsiString AppDir;
int numberOfGate;
bool dragStart;
int lastaction;
bool isSelection = false;
```

Η συνάρτηση αυτή εκτελείται όταν ξεκινάει το πρόγραμμα.

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
```

Δημιουργείται ένα λευκό ορθογώνιο, διαστάσεων ίσων με αυτές του παραθύρου.

```
    Plaketa->Canvas->Rectangle(0,0,Plaketa->Width,Plaketa->Height);
```

Στην αρχική οθόνη προστίθενται αυτόματα 8 διακόπτες (switches). Αυτοί οι διακόπτες θεωρούνται από το πρόγραμμα ως πύλες-είσοδοι .

Τα αρχικά ονόματα των εισόδων.

```
    char SwitchName[8]={'A','B','C','D','E','F','G','H'};
 int X;
 int Y;
 int diffY;
 Image1->Visible=false;
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
```

Εξ ορισμού, θεωρούμε ότι οι διακόπτες είναι ανοικτοί. Συνεπώς, το πρόγραμμα φορτώνει την εικόνα switchon.bmp (ανοικτός διακόπτης) και την αποθηκεύει στην μεταβλητή Bitmap1.

```
    Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
```

Η τοποθέτηση των διακοπτών ξεκινάει από το σημείο (30,50), ενώ κάθε διακόπτης τοποθετείται 50 pixels πιο κάτω από τον προηγούμενος. Δηλ. ο 2ος διακόπτης θα μπει στο (30,80), ο 3ος στο (30, 130) κ.ο.κ.

```
 X=30;
 Y=50;
 diffY=50;
 for(int i=0;i<8;i++)
 {
```

Αρχικά, σχεδιάζεται η εικόνα του διακόπτη, στην προβλεπόμενη θέση (με την Draw).

Στην συνέχεια "σχεδιάζεται" η ονομασία του διακόπτη (A, B, C κλπ., με την TextOut).

Τέλος, από την στιγμή που ο διακόπτης θεωρείται ως πύλη, τα στοιχεία του αποθηκεύονται στον πίνακα Pili (με την Store) και αυξάνεται το πλήθος των πυλών (arithmospilon).

Η διαδικασία αυτή θα εκτελεσθεί 8 φορές (8 διακόπτες)

```
 Plaketa->Canvas->Draw(X,Y+diffY*i-Bitmap1->Height*0.5,Bitmap1);
 Plaketa->Canvas->TextOut(X-10,Y+diffY*i-Bitmap1>Height*0.5,SwitchName[i]);
 Pili[arithmospilon].Store(6,X,Y+diffY*i-Bitmap1->Height*0.5,0);
 Pili[arithmospilon].outX=X+30;
 Pili[arithmospilon].outY=Y+diffY*i-Bitmap1->Height*0.5+21;
 Pili[arithmospilon].in1val=1;
 arithmospilon++;
}
```

Από την στιγμή που οι διακόπτες μας θεωρούνται, αρχικά, ότι είναι ανοικτοί, τα κουμπιά της μπάρας εργασιών που ανοίγουν ή κλείνουν τους διακόπτες είναι πατημένα (άρα οι διακόπτες είναι ανοικτοί)

```
 ToolButtonPowerA->Down=true;
 ToolButtonPowerB->Down=true;
 ToolButtonPowerC->Down=true;
 ToolButtonPowerD->Down=true;
 ToolButtonPowerE->Down=true;
 ToolButtonPowerF->Down=true;
 ToolButtonPowerG->Down=true;
 ToolButtonPowerH->Down=true;
}
```

Η συνάρτηση αυτή εκτελείται όταν ο χρήστης πατήσει το αριστερό πλήκτρο του mouse. Ανάλογα με το ποιό κουμπί (από την μπάρα εργασιών) είναι πατημένο, το πρόγραμμα εκτελεί το ανάλογο κομμάτι κώδικα.

```
void __fastcall TForm1::PlaketaMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)
```

Στην αρχή, το πρόγραμμα ελέγχει αν ο αριθμός των πυλών έχει ξεπεράσει το επιτρεπτό όριο (maxobjects) και ΔΕΝ έχει επιλεχθεί το κουμπί σχεδίασης γραμμών (δηλ. έχει επιλεχθεί κάποιο κουμπί σχεδίασης πύλης). Στην περίπτωση αυτή, το πρόγραμμα ενημερώνει τον χρήστη πως έχει ξεπεράσει το επιτρεπτό όριο πυλών.

```
    if((arithmospilon>maxobjects)&&(ToolButtonLINE->Down==false))
    {
        ShowMessage("Σφάλμα, μέγιστος αριθμός πυλών=" + maxobjects);
    }
    else
    {
        Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
        if ((ToolButtonSelect->Down==true) && (index_for_move!=-1))
```

```
{
```

Ο συγκεκριμένος κώδικας εκτελείται όταν έχει πατηθεί το κουμπί Select (κουμπί επιλογής πύλης για μετακίνηση) και όταν έχει επιλεχτεί κάποια πύλη (index_for_move δεν είναι ίσο με -1).

Πρακτικά, αυτός ο κώδικας εκτελείται όταν θέλουμε να μετακινήσουμε μια πύλη.

Υπάρχουν, ουσιαστικά, δύο στάδια για την μετακίνηση μιας πύλης.

Το 1ο στάδιο εκτελείται όταν ο χρήστης κάνει το πρώτο κλικ πάνω σε μια πύλη για την επιλέξει.

Στην περίπτωση αυτή, ενώ αρχικά η μεταβλητή for_move έχει την τιμή false, μετά θα έχει την τιμή true, ενώ επιπλέον ο δείκτης του ποντικιού θα μετατραπεί σε χεράκι συνοδευόμενο από το εικονίδιο της πύλης που έχει επιλεχτεί.

Το 2ο στάδιο εκτελείται όταν η μεταβλητή for_move έχει την τιμή true (δηλ. όταν έχουμε επιλέξει την πύλη μας, έχουμε μετακινήσει το ποντίκι και κάνουμε κλικ στην νέα θέση της πύλης.

Εκτός από την θέση της πύλης, αλλάζουνε και οι συντεταγμένες των γραμμών που τυχόν συνδέονται με την πύλη μας.

1ο στάδιο

```
if (for_move==false)
{
    for_move=true;

    switch (Pili[index_for_move].type)
    {
case 0:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_ANDGRABBED_CURSOR));break;
case 1:Screen->Cursors[crMyCursor] = LoadCursor(HInstance   MAKEINTRESOURCE(IDC_ORGRABBED_CURSOR));break;
case 2:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NANDGRABBED_CURSOR));break;
case 3:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NORGRABBED_CURSOR));break;
case 4:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NOTGRABBED_CURSOR));break;
case 5:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_XORGRABBED_CURSOR));break;
case 7:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_LEDGRABBED_CURSOR));break;
    }
    Cursor=crMyCursor;
}
```

2ο στάδιο

```
else
{
    X=Xtoadd;
    Y=Ytoadd;
```

```
int palio_telos_x1=Pili[index_for_move].inX1;
int palio_telos_x2=Pili[index_for_move].inX2;
int palio_telos_Y1=Pili[index_for_move].inY1;
int palio_telos_Y2=Pili[index_for_move].inY2;
int palia_arxi_X=Pili[index_for_move].outX;
int palia_arxi_Y=Pili[index_for_move].outY;
for_move=false;
```

Γίνεται η τοποθέτηση της πύλης στη νέα της θέση

```
Pili[index_for_move].left=X;
Pili[index_for_move].top=Y;

Pili[index_for_move].inX1=X+0;
if (Pili[index_for_move].type==4) Pili[index_for_move].inY1=Y- Bitmap1->Height*0.5+15;
else if (Pili[index_for_move].type==7)   Pili[index_for_move].inY1=Y-Bitmap1->Height*0.5+31;
else Pili[index_for_move].inY1=Y+11;

Pili[index_for_move].inX2=X+0;

Pili[index_for_move].outX=X+31;

If (Pili[index_for_move].type==5) Pili[index_for_move].inY2=Y+20;
else Pili[index_for_move].inY2=Y+23;

if (Pili[index_for_move].type==5) Pili[index_for_move].outY=Y+16;
else if (Pili[index_for_move].type==7) Pili[index_for_move].outY=Y-Bitmap1->Height*0.5+15;
else Pili[index_for_move].outY=Y+18;
```

Ο δείκτης του ποντικιού λαμβάνει την μορφή ενός χεριού.

```
Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_GRAB_CURSOR));
Cursor=crMyCursor;
```

Ελέγχονται όλες οι γραμμές του σχεδίου, και όσες κριθεί ότι πρέπει να αλλαχτούν, αλλάζουν (και η αρχή και το τέλος του)

```
for (int i=0;i<arithmosgrammon;i++)
{
    if ((Lines[i].eosX==palio_telos_x1) && (Lines[i].eosY==palio_telos_Y1))
    {
        Lines[i].eosX=Pili[index_for_move].inX1;
        Lines[i].eosY=Pili[index_for_move].inY1;
    }
    else if ((Lines[i].eosX==palio_telos_x2) && (Lines[i].eosY==palio_telos_Y2))
    {
        Lines[i].eosX=Pili[index_for_move].inX2;
        Lines[i].eosY=Pili[index_for_move].inY2;
```

```
            }
            else if ((Lines[i].apoX==palia_arxi_X) && (Lines[i].apoY==palia_arxi_Y))
            {
                Lines[i].apoX=Pili[index_for_move].outX;
                Lines[i].apoY=Pili[index_for_move].outY;
            }
        }
```

Το ίδιο γίνεται και με τις τεθλασμένες γραμμές

```
        for (int i=0;i<=plithos_tethlasmenon;i++)
        {
            if ((pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]==palio_telos_x1) &&
(pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][1]==palio_telos_Y1))
            {
                pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]=Pili[index_for_move].inX1;
                pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][1]=Pili[index_for_move].inY1;
            }
            else if ((pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]==palio_telos_x2) &&
(pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][1]==palio_telos_Y2))
            {
                pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]=Pili[index_for_move].inX2;
                pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][1]=Pili[index_for_move].inY2;
            }
            else if ((pinakas_tethlasmenis[i][0][0]==palia_arxi_X) && (pinakas_tethlasmenis[i][0][1]==palia_arxi_Y))
            {
                pinakas_tethlasmenis[i][0][0]=Pili[index_for_move].outX;
                pinakas_tethlasmenis[i][0][1]=Pili[index_for_move].outY;
            }
        }

        index_for_move=-1;
```

Το σχέδιο επανασχεδιάζεται από την αρχή

```
        int temparithmospilon=arithmospilon;
        int temparithmosgrammon=arithmosgrammon;

        bool tempPower[8];

        tempPower[0]=ToolButtonPowerA->Down;
        tempPower[1]=ToolButtonPowerB->Down;
        tempPower[2]=ToolButtonPowerC->Down;
        tempPower[3]=ToolButtonPowerD->Down;
        tempPower[4]=ToolButtonPowerE->Down;
        tempPower[5]=ToolButtonPowerF->Down;
        tempPower[6]=ToolButtonPowerG->Down;
        tempPower[7]=ToolButtonPowerH->Down;

        Undo1->Enabled = false;
```

```
                arithmospilon=0;
                arithmosgrammon=0;
                FormShow(this);
                arithmosgrammon=temparithmosgrammon;
                arithmospilon=temparithmospilon;

                lastaction=0;
                ToolButtonPowerA->Down=tempPower[0];
                ToolButtonPowerB->Down=tempPower[1];
                ToolButtonPowerC->Down=tempPower[2];
                ToolButtonPowerD->Down=tempPower[3];
                ToolButtonPowerE->Down=tempPower[4];
                ToolButtonPowerF->Down=tempPower[5];
                ToolButtonPowerG->Down=tempPower[6];
                ToolButtonPowerH->Down=tempPower[7];
                Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
                for (int i=0;i<8;i++)
                {
                    if (tempPower[i]==true) Bitmap1->
->LoadFromFile(AppDir+"\\switchon.bmp");
                    else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
                    Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
                }
                for(int i=0;i<arithmospilon;i++)
                {
                    switch(Pili[i].type)
                    {
                  case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
                  case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
                  case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
                  case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
                  case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
                  case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
                  case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
                    }
                    Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
                }
                for(int i=0;i<arithmosgrammon;i++)
                {
                    if (Lines[i].LineStatus==0)
                    {
                      Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
                      Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
                    }
                }
                for (int i=0;i<=plithos_tethlasmenon;i++)
                {
```

```
                    for (int k=0;k<index_tethlasmenis[i]-1;k++)
                    {
                 Plaketa->Canvas->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
                 Plaketa->Canvas->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
                Plaketa->Canvas->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
                    }
                }
            }
        }
```

Ο συγκεκριμένος κώδικας εκτελείται όταν έχει πατηθεί το κουμπί Erase (κουμπί διαγραφής πύλης) και όταν έχει επιλεχτεί κάποια πύλη για διαγραφή (for_erase=true).

```
        if ((ToolButtonErase->Down==true) && (for_erase==true))
        {
            for_erase=false;
```

Αρχικά ελέγχεται αν υπάρχει κάποια γραμμή που να συνδέεται με την πύλη που θέλουμε να διαγράψουμε.

Αν υπάρχει, εμφανίζεται προειδοποιητικό μήνυμα.

```
            for (int i=0;i<arithmosgrammon;i++)
            {
if (((Lines[i].eosX==Pili[index_for_erase].inX1) && (Lines[i].eosY==Pili[index_for_erase].inY1)) ||
((Lines[i].eosX==Pili[index_for_erase].inX2) && (Lines[i].eosY==Pili[index_for_erase].inY2)) ||
((Lines[i].apoX==Pili[index_for_erase].outX) && (Lines[i].apoY==Pili[index_for_erase].outY)))
                {
                    ShowMessage("Θα πρέπει, αρχικά, να σβήσετε όλες τις γραμμές που συνδέονται με την πύλη!!!");
                    return;
                }
            }
```

Ο ίδιος ακριβώς έλεγχος γίνεται και για τις τεθλασμένες γραμμές.

```
            for (int i=0;i<plithos_tethlasmenon;i++)
            {
                if ((pinakas_tethlasmenis[i][0][0]==Pili[index_for_erase].outX) &&
(pinakas_tethlasmenis[i][0][1]==Pili[index_for_erase].outY))
                {
                    ShowMessage("Θα πρέπει, αρχικά, να σβήσετε όλες τις γραμμές που συνδέονται με την πύλη!!!");
                    return;
                }
            }
```

1ο στάδιο διαγραφής.

Έστω ότι έχουμε σχεδιάσει 10 πύλες, και θέλουμε να σβήσουμε την 5η πύλη.

Στον πίνακα Pili (στον οποίο είναι καταχωρημένες οι πύλες) γίνεται μια ολίσθηση προς τα αριστερά.

Τα στοιχεία της 6ης πύλης θα αποθηκευτούν στο 5ο κελί, τα στοιχεία της 7ης πύλης στο 6ο κελί κ.ο.κ.

Στη συνέχεια, το πλήθος των πυλών θα μειωθεί κατά ένα..

Με αυτό τον τρόπο τα στοιχεία της προς διαγραφή πύλης θα διαγραφτούν.

```
for (int i=index_for_erase;i<arithmospilon-1;i++)
{
    Pili[i].type=Pili[i+1].type;
    Pili[i].left=Pili[i+1].left;
    Pili[i].top=Pili[i+1].top;
    Pili[i].width=Pili[i+1].width;
    Pili[i].numinputs=Pili[i+1].numinputs;
    Pili[i].inX1=Pili[i+1].inX1;
    Pili[i].inY1=Pili[i+1].inY1;
    Pili[i].inX2=Pili[i+1].inX2;
    Pili[i].inY2=Pili[i+1].inY2;
    Pili[i].outX=Pili[i].outX;
    Pili[i].outY=Pili[i].outY;
    Pili[i].in1val=Pili[i+1].in1val;
    Pili[i].in2val=Pili[i+1].in2val;
    Pili[i].outval=Pili[i+1].outval;
}

int temparithmospilon=arithmospilon;
int temparithmosgrammon=arithmosgrammon;
```

2ο στάδιο διαγραφής: Το σχέδιο επανασχεδιάζεται από την αρχή.

```
bool tempPower[8];

tempPower[0]=ToolButtonPowerA->Down;
tempPower[1]=ToolButtonPowerB->Down;
tempPower[2]=ToolButtonPowerC->Down;
tempPower[3]=ToolButtonPowerD->Down;
tempPower[4]=ToolButtonPowerE->Down;
tempPower[5]=ToolButtonPowerF->Down;
tempPower[6]=ToolButtonPowerG->Down;
tempPower[7]=ToolButtonPowerH->Down;

Undo1->Enabled = false;
arithmospilon=0;
arithmosgrammon=0;
FormShow(this);
arithmosgrammon=temparithmosgrammon;
```

Μείωση του πλήθους των πυλών κατά 1.

```
arithmospilon=temparithmospilon-1;

lastaction=0;
ToolButtonPowerA->Down=tempPower[0];
ToolButtonPowerB->Down=tempPower[1];
ToolButtonPowerC->Down=tempPower[2];
ToolButtonPowerD->Down=tempPower[3];
ToolButtonPowerE->Down=tempPower[4];
ToolButtonPowerF->Down=tempPower[5];
ToolButtonPowerG->Down=tempPower[6];
ToolButtonPowerH->Down=tempPower[7];
Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
```

Σχεδίαση διακοπτών.

```
for (int i=0;i<8;i++)
{
if (tempPower[i]==true) Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
    Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
}

for(int i=0;i<arithmospilon;i++)//Σχεδίαση πυλών
{
    switch(Pili[i].type)
    {
      case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
      case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
      case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
      case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
      case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
      case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
      case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
    }
    Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
}
```

Σχεδίαση γραμμών.

```
for(int i=0;i<arithmosgrammon;i++)
{
    if (Lines[i].LineStatus==0)
    {
        Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
        Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
    }
}
```

Σχεδίαση τεθλασμένων γραμμών

```
        for (int i=0;i<=plithos_tethlasmenon;i++)
        {
            for (int k=0;k<index_tethlasmenis[i]-1;k++)
            {
             Plaketa->Canvas->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
            Plaketa->Canvas->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
            Plaketa->Canvas->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
            }
        }
    }
```

Ο συγκεκριμένος κώδικας εκτελείται όταν έχει πατηθεί το κουμπί LineErase (κουμπί διαγραφής γραμμής) και όταν έχει επιλεχτεί κάποια τεθλασμένη γραμμή για διαγραφή (tethlasmeni_for_erase=true).

Η διαδικασία είναι παρόμοια με αυτή για την διαγραφή των πυλών.

```
    if ((ToolButtonLineErase->Down==true) && (tethlasmeni_for_erase==true))
    {
        tethlasmeni_for_erase=false;
```

1ο στάδιο: Διαγραφή των στοιχείων της τεθλασμένης (με ολίσθηση προς τα αριστερά του pinakas_tethlasmenis).

```
        for (int i=tethlasmeni_index_for_erase;i<=plithos_tethlasmenon-1;i++)
        {
            index_tethlasmenis[i]=index_tethlasmenis[i+1];
            for (int k=0;k<index_tethlasmenis[i];k++)
            {
                pinakas_tethlasmenis[i][k][0]=pinakas_tethlasmenis[i+1][k][0];
                pinakas_tethlasmenis[i][k][1]=pinakas_tethlasmenis[i+1][k][1];
            }
        }
Μείωση κατά 1
        plithos_tethlasmenon--;
```

//2ο στάδιο διαγραφής: Το σχέδιο επανασχεδιάζεται από την αρχή.

```
        int temparithmospilon=arithmospilon;
        int temparithmosgrammon=arithmosgrammon;

        bool tempPower[8];

        tempPower[0]=ToolButtonPowerA->Down;
        tempPower[1]=ToolButtonPowerB->Down;
        tempPower[2]=ToolButtonPowerC->Down;
        tempPower[3]=ToolButtonPowerD->Down;
        tempPower[4]=ToolButtonPowerE->Down;
        tempPower[5]=ToolButtonPowerF->Down;
```

```
tempPower[6]=ToolButtonPowerG->Down;
tempPower[7]=ToolButtonPowerH->Down;

Undo1->Enabled = false;
arithmospilon=0;
arithmosgrammon=0;
FormShow(this);
arithmosgrammon=temparithmosgrammon-1;
arithmospilon=temparithmospilon;

lastaction=0;
ToolButtonPowerA->Down=tempPower[0];
ToolButtonPowerB->Down=tempPower[1];
ToolButtonPowerC->Down=tempPower[2];
ToolButtonPowerD->Down=tempPower[3];
ToolButtonPowerE->Down=tempPower[4];
ToolButtonPowerF->Down=tempPower[5];
ToolButtonPowerG->Down=tempPower[6];
ToolButtonPowerH->Down=tempPower[7];
Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
for (int i=0;i<8;i++)
{
if (tempPower[i]==true) Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
}

for(int i=0;i<arithmospilon;i++)
{
    switch(Pili[i].type)
    {
      case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
      case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
      case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
      case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
      case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
      case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
      case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
    }
    Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
}
for(int i=0;i<arithmosgrammon;i++)
{
    if (Lines[i].LineStatus==0)
    {
        Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
        Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
```

```
            }
        }
        for (int i=0;i<=plithos_tethlasmenon;i++)
        {
            for (int k=0;k<index_tethlasmenis[i]-1;k++)
            {
              Plaketa->Canvas->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
              Plaketa->Canvas->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
             Plaketa->Canvas->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
            }
        }
        if (plithos_tethlasmenon==-1) plithos_tethlasmenon=0;

    }
```

Ο συγκεκριμένος κώδικας εκτελείται όταν έχει πατηθεί το κουμπί LineErase (κουμπί διαγραφής γραμμής) και όταν έχει επιλεχτεί κάποια γραμμή για διαγραφή (line_for_erase=true).

Η διαδικασία είναι παρόμοια με αυτή για την διαγραφή των τεθλασμένων γραμμών.

Η μοναδική διαφοροποίηση έγκειται στο γεγονός ότι, από την στιγμή που οι τεθλασμένες γραμμές νοούνται από το πρόγραμμα σαν απλές, ελέγχεται επιπλέον αν η γραμμή που πάμε να σβήσουμε είναι "virtual" (έχει δημιουργηθεί, δηλαδή, εικονικά για να εξομοιώσει την τεθλασμένη γραμμή, και δεν σχεδιάζεται).

Αν είναι, δεν θα διαγραφεί μονάχα η απλή γραμμή αλλά και η αντίστοιχη τεθλασμένη.

```
    if ((ToolButtonLineErase->Down==true) && (line_for_erase==true))
    {
        int deiktis_tethlasmenis=-1;
        line_for_erase=false;
```

Έλεγχος για το αν η γραμμή μας είναι "virtual"

```
        for (int i=0;i<=plithos_tethlasmenon;i++)
        {
            if ((Lines[line_index_for_erase].eosX==pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]) &&
(Lines[line_index_for_erase].eosY= =pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][1]))
            {
                deiktis_tethlasmenis=i;
                i=plithos_tethlasmenon;
            }
        }
```

Είναι virtual.

```
if (deiktis_tethlasmenis!=-1)
{
    for (int i=deiktis_tethlasmenis;i<=plithos_tethlasmenon-1;i++)
    {
        index_tethlasmenis[i]=index_tethlasmenis[i+1];
        for (int k=0;k<index_tethlasmenis[i];k++)
        {
            pinakas_tethlasmenis[i][k][0]=pinakas_tethlasmenis[i+1][k][0];
            pinakas_tethlasmenis[i][k][1]=pinakas_tethlasmenis[i+1][k][1];
        }
    }
}

plithos_tethlasmenon--;
```

Ολίσθηση του πίνακα Lines προς τα αριστερά (για διαγραφή των στοιχείων της γραμμής).

```
for (int i=line_index_for_erase;i<arithmosgrammon-1;i++)
{
    Lines[i].apoX=Lines[i+1].apoX;
    Lines[i].apoY=Lines[i+1].apoY;
    Lines[i].eosX=Lines[i+1].eosX;
    Lines[i].eosY=Lines[i+1].eosY;
    Lines[i].ApoPiliIndex=Lines[i+1].ApoPiliIndex;
    Lines[i].EosPiliIndex=Lines[i+1].EosPiliIndex;
    Lines[i].EosPiliNum=Lines[i+1].EosPiliNum;
    Lines[i].LineStatus=Lines[i+1].LineStatus;
}
```

Επανασχεδίαση του σχεδίου.

```
int temparithmospilon=arithmospilon;
int temparithmosgrammon=arithmosgrammon;

bool tempPower[8];

tempPower[0]=ToolButtonPowerA->Down;
tempPower[1]=ToolButtonPowerB->Down;
tempPower[2]=ToolButtonPowerC->Down;
tempPower[3]=ToolButtonPowerD->Down;
tempPower[4]=ToolButtonPowerE->Down;
tempPower[5]=ToolButtonPowerF->Down;
tempPower[6]=ToolButtonPowerG->Down;
tempPower[7]=ToolButtonPowerH->Down;

Undo1->Enabled = false;
arithmospilon=0;
arithmosgrammon=0;
```

```
        FormShow(this);
        arithmosgrammon=temparithmosgrammon-1;
        arithmospilon=temparithmospilon;

        lastaction=0;
        ToolButtonPowerA->Down=tempPower[0];
        ToolButtonPowerB->Down=tempPower[1];
        ToolButtonPowerC->Down=tempPower[2];
        ToolButtonPowerD->Down=tempPower[3];
        ToolButtonPowerE->Down=tempPower[4];
        ToolButtonPowerF->Down=tempPower[5];
        ToolButtonPowerG->Down=tempPower[6];
        ToolButtonPowerH->Down=tempPower[7];
        Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
        for (int i=0;i<8;i++)
        {
 if (tempPower[i]==true) Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
 else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
        }

        for(int i=0;i<arithmospilon;i++)
        {
            switch(Pili[i].type)
            {
            case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
            case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
            case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
            case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
            case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
            case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
            case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
            }
            Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
        }
        for(int i=0;i<arithmosgrammon;i++)
        {
            if (Lines[i].LineStatus==0)
            {
                Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
                Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
            }
        }
        for (int i=0;i<=plithos_tethlasmenon;i++)
        {
            for (int k=0;k<index_tethlasmenis[i]-1;k++)
```

```
                {
                    Plaketa->Canvas->
->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
                    Plaketa->Canvas->
->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
                    Plaketa->Canvas->
->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
                }
            }
        }
```

Πάμε να σχεδιάσουμε μια πύλη XOR.

```
        if(ToolButtonXOR->Down==true)
        {
            X=Xtoadd;
            Y=Ytoadd;
            Image1->Visible=true;
            Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");
            Plaketa->Canvas->Draw(X,Y,Bitmap1);
            Pili[arithmospilon].Store(5,X,Y,2);
            Pili[arithmospilon].inX1=X+0;
            Pili[arithmospilon].inY1=Y+11;
            Pili[arithmospilon].inX2=X+0;
            Pili[arithmospilon].inY2=Y+20;
            Pili[arithmospilon].outX=X+31;
            Pili[arithmospilon].outY=Y+16;
            arithmospilon++;
            lastaction=100;
            Undo1->Enabled = true;
        }
```

Πάμε να σχεδιάσουμε μια πύλη AND.

```
        if(ToolButtonAND->Down==true)
        {
            X=Xtoadd;
            Y=Ytoadd;
            Image1->Visible=true;
            Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");
            Plaketa->Canvas->Draw(X,Y,Bitmap1);
            Pili[arithmospilon].Store(0,X,Y,2);
            Pili[arithmospilon].inX1=X+0;
            Pili[arithmospilon].inY1=Y+11;
            Pili[arithmospilon].inX2=X+0;
            Pili[arithmospilon].inY2=Y+23;
            Pili[arithmospilon].outX=X+31;
            Pili[arithmospilon].outY=Y+18;
            arithmospilon++;
            lastaction=100;
```

```
        Undo1->Enabled = true;
    }
```

Πάμε να σχεδιάσουμε μια πύλη OR.

```
    if(ToolButtonOR->Down==true)
    {
        X=Xtoadd;
        Y=Ytoadd;
        Image1->Visible=false;
        Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");
        Plaketa->Canvas->Draw(X,Y,Bitmap1);
        Pili[arithmospilon].Store(1,X,Y,2);
        Pili[arithmospilon].inX1=X+0;
        Pili[arithmospilon].inY1=Y+11;
        Pili[arithmospilon].inX2=X+0;
        Pili[arithmospilon].inY2=Y+23;
        Pili[arithmospilon].outX=X+31;
        Pili[arithmospilon].outY=Y+18;
        arithmospilon++;
        lastaction=100;
        Undo1->Enabled = true;
    }
```

Πάμε να σχεδιάσουμε μια πύλη NAND.

```
    if(ToolButtonNAND->Down==true)
    {
        X=Xtoadd;
        Y=Ytoadd;
        Image1->Visible=false;
        Cursor=crNone;
        Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");
        Plaketa->Canvas->Draw(X,Y,Bitmap1);
        Pili[arithmospilon].Store(2,X,Y,2);
        Pili[arithmospilon].inX1=X+0;
        Pili[arithmospilon].inY1=Y+11;
        Pili[arithmospilon].inX2=X+0;
        Pili[arithmospilon].inY2=Y+23;
        Pili[arithmospilon].outX=X+31;
        Pili[arithmospilon].outY=Y+18;
        arithmospilon++;
        lastaction=100;
        Undo1->Enabled = true;
    }
```

Πάμε να σχεδιάσουμε μια πύλη NOR.

```
    if(ToolButtonNOR->Down==true)
    {
        X=Xtoadd;
```

```
        Y=Ytoadd;
        Image1->Visible=false;
        Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");
        Plaketa->Canvas->Draw(X,Y,Bitmap1);
        Pili[arithmospilon].Store(3,X,Y,2);
        Pili[arithmospilon].inX1=X+0;
        Pili[arithmospilon].inY1=Y+11;
        Pili[arithmospilon].inX2=X+0;
        Pili[arithmospilon].inY2=Y+23;
        Pili[arithmospilon].outX=X+31;
        Pili[arithmospilon].outY=Y+18;
        arithmospilon++;
        lastaction=100;
        Undo1->Enabled = true;
      }
```

Πάμε να σχεδιάσουμε μια πύλη NOT.

```
    if(ToolButtonNOT->Down==true)
    {
        X=Xtoadd;
        Y=Ytoadd;
        Image1->Visible=false;
        Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");
        Plaketa->Canvas->Draw(X,Y-Bitmap1->Height*0.5,Bitmap1);
        Pili[arithmospilon].Store(4,X,Y-Bitmap1->Height*0.5,1);
        Pili[arithmospilon].inX1=X+0;
        Pili[arithmospilon].inY1=Y-Bitmap1->Height*0.5+15;
        Pili[arithmospilon].outX=X+31;
        Pili[arithmospilon].outY=Y-Bitmap1->Height*0.5+15;
        arithmospilon++;
        lastaction=100;
        Undo1->Enabled = true;
    }
```

Πάμε να σχεδιάσουμε ένα LED (λαμπτήρας).

```
    if(ToolButtonLED->Down==true)
    {
        X=Xtoadd;
        Y=Ytoadd;
        Image1->Visible=false;
        Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");
        Plaketa->Canvas->Draw(X,Y-Bitmap1->Height*0.5,Bitmap1);
        Pili[arithmospilon].Store(7,X,Y-Bitmap1->Height*0.5,1);
        Pili[arithmospilon].inX1=X+0;
        Pili[arithmospilon].inY1=Y-Bitmap1->Height*0.5+31;
        arithmospilon++;
        lastaction=100;
```

```
        Undo1->Enabled = true;
    }
```

Ο κώδικας αυτός εκτελείται όταν έχουμε επιλέξει το κουμπί για την σχεδίαση γραμμής.

Η διαδικασία για την σχεδίαση γραμμής είναι η εξής:

Σε κάθε πύλη, το πλέον δεξί pixel της εξόδου είναι μπλε (το ίδιο ισχύει και για τους διακόπτες, αλλά και για τις άκρες των τεθλασμένων γραμμών), ενώ το πλέον αριστερό pixels και στις δύο εισόδους είναι κόκκινο.

Αρχικά, λοιπόν, ελέγχεται αν το αριστερό κλικ του ποντικιού έχει γίνει σε μπλε pixel.

Αν ο έλεγχος είναι πετυχημένος (το κλικ έγινε σε μπλε pixel), γίνονται οι εξής ενέργειες:

1. Η μεταβλητή ZografiseGrammi παίρνει την τιμή true (σημάδι ότι είμαστε σε κατάσταση σχεδίασης γραμμής).

2. Οι X και Y συντεταγμένες αποθηκεύονται στις μεταβλητές XGrammiArxi και YGrammiArxi.

Αν ο έλεγχος είναι αποτυχημένος (το κλικ ΔΕΝ έγινε σε μπλε pixel), γίνεται ένας επιπλέον έλεγχος για το αν το αριστερό κλικ έγινε σε κόκκινο pixel (είσοδος πύλης).

Αν ο νέος έλεγχος είναι πετυχημένος, τότε εξετάζεται η τιμή της μεταβλητής ZografiseGrammi.

Αν η ZografiseGrammi έχει την τιμή true (είμαστε σε κατάσταση σχεδίασης γραμμής) γίνονται οι εξής ενέργειες:

1. Σχεδιάζεται μια γραμμή η οποία συνδέει είτε δύο πύλες είτε το άκρο μιας τεθλασμένης με την είσοδο μιας πύλης (αυτό εξαρτάται από το αν η τιμή της μεταβλητής EndiamesesGrammes είναι true, σημάδι ότι βρισκόμαστε σε κατάσταση σχεδίασης τεθλασμένων γραμμών, ή όχι).

2. Η νέα γραμμή προστίθεται στον πίνακα Lines (αν είναι τεθλασμένη, προστίθενται και τα στοιχεία της τεθλασμένης στους πίνακες pinakas_tethlasmenis και index_tethlasmenis).

Σε αντίθετη περίπτωση, δεν γίνεται τίποτα.

Αν, όμως, ο έλεγχος για το κόκκινο pixel είναι αποτυχημένος, τότε εξετάζεται εκ νέου η τιμή της μεταβλητής ZografiseGrammi.

Αν είναι true, θεωρούμε ότι είμαστε σε κατάσταση σχεδίασης τεθλασμένης γραμμής, και γίνονται οι εξής ενέργειες:

1. Συνεχίζεται κανονικά η σχεδίαση της τεθλασμένης γραμμής

2. Ενημερώνονται οι πίνακες pinakas_tethlasmenis και index_tethlasmenis

```
if(ToolButtonLINE->Down==true)
{
    X=Xtoadd;
    Y=Ytoadd;
```

Μπλε pixel

```
    if(Plaketa->Canvas->Pixels[X][Y]==clBlue)
    {
        XGrammiArxi=X;
        YGrammiArxi=Y;
        XEndiamesiGrammiArxi=X;
        YEndiamesiGrammiArxi=Y;
        arxi_XEndiamesiGrammiArxi=X;
        arxi_YEndiamesiGrammiArxi=Y;
        ZografiseGrammi=true;
        index_tethlasmenis[plithos_tethlasmenon]=0;
        int tmp=index_tethlasmenis[plithos_tethlasmenon];
        pinakas_tethlasmenis[plithos_tethlasmenon][tmp][0]=X;
        pinakas_tethlasmenis[plithos_tethlasmenon][tmp][1]=Y;
        index_tethlasmenis[plithos_tethlasmenon]++;
        plithos_tethlasmenon++;
    }
    else
    {
```

Κόκκινο pixel

```
        if(Plaketa->Canvas->Pixels[X][Y]==clRed)
        {
            if((ZografiseGrammi==true))
            {
```

Για να μην παίρνει μια πύλη 2 εισόδους

```
                Plaketa->Canvas->Pixels[X][Y]=clBlack;
    if (EndiamesesGrammes==true) Plaketa->Canvas->->MoveTo(XEndiamesiGrammiArxi,YEndiamesiGrammiArxi);
else Plaketa->Canvas->MoveTo(XGrammiArxi+1,YGrammiArxi);
                Plaketa->Canvas->LineTo(X,Y);
Lines[arithmosgrammon].Store(XGrammiArxi,YGrammiArxi,X,Y);
                arithmosgrammon++;
                if (EndiamesesGrammes==true)
                {
            lastaction=400;
            Lines[arithmosgrammon-1].LineStatus=1;
```

```
                    int tmp=index_tethlasmenis[plithos_tethlasmenon-1];
                    pinakas_tethlasmenis[plithos_tethlasmenon-1][tmp][0]=X;
                    pinakas_tethlasmenis[plithos_tethlasmenon-1][tmp][1]=Y;
                   index_tethlasmenis[plithos_tethlasmenon-1]++;
                      }
                      else lastaction=200;
                      Undo1->Enabled = true;
                      ZografiseGrammi=false;
                      EndiamesesGrammes=false;
                      Plaketa->Canvas->Pixels[X][Y]=clRed;
                  }
              }
              else
              {
                  if (ZografiseGrammi==true)
                  {
      EndiamesesGrammes=true;
     Plaketa->Canvas->MoveTo(XEndiamesiGrammiArxi,YEndiamesiGrammiArxi);
     Plaketa->Canvas->LineTo(X,Y);
```

Για να μην παίρνει μια πύλη 2 εισόδους

```
    Plaketa->Canvas->Pixels[X][Y]=clBlue;

                      arxi_XEndiamesiGrammiArxi=XEndiamesiGrammiArxi;
                      arxi_YEndiamesiGrammiArxi=YEndiamesiGrammiArxi;
                      XEndiamesiGrammiArxi=X;
                      YEndiamesiGrammiArxi=Y;

                      lastaction=300;
                      Undo1->Enabled = true;
                      int tmp=index_tethlasmenis[plithos_tethlasmenon-1];
  pinakas_tethlasmenis[plithos_tethlasmenon-1][tmp][0]=XEndiamesiGrammiArxi;
  pinakas_tethlasmenis[plithos_tethlasmenon-1][tmp][1]=YEndiamesiGrammiArxi;
                      index_tethlasmenis[plithos_tethlasmenon-1]++;
                  }
              }
          }
      }
```

Όλες οι εικόνες φορτώνονται στην μεταβλητή Bitmap1, η οποία και διαγράφεται από την μνήμη.

```
      delete Bitmap1;
   }
}
```

Η συνάρτηση αυτή εκτελείται όταν ο χρήστης μετακινήσει το ποντίκι.

Ανάλογα με το ποιό κουμπί (από την μπάρα εργασιών) είναι πατημένο, το πρόγραμμα εκτελεί το ανάλογο κομμάτι κώδικα.

Κατά την μετακίνηση του mouse πάνω από την πλακέτα, ανάλογα με το αν έχει επιλεχτεί κάποια πύλη ή το εργαλείο σχεδίασης γραμμών (από την μπάρα εργασιών), για διευκόλυνση του χρήστη εμφανίζουμε το αντίστοιχο σύμβολο (ποια πύλη είναι κλπ.), ή αν έχουμε επιλέξει το εργαλείο των γραμμών το πρόγραμμα μάς κατευθύνει στην πιο κοντινή πύλη.

```
void __fastcall TForm1::PlaketaMouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
```

Τα όρια της πλακέτας καταχωρούνται σε μεταβλητές

```
    int deksiorio=Plaketa->Width-Image1->Width;
    int katoorio=Plaketa->Height-Image1->Height;
```

Αρχικά ελέγχεται αν μπορούν να προστεθούν και άλλες πύλες

```
    AnsiString tmp;
    if(arithmospilon==maxobjects)
    {
        tmp.printf("Δε μπορείτε να προσθέσετε άλλες πύλες        ");
    }
    else
    {
        if(arithmospilon==maxobjects-1)
        {
            tmp.printf("Μπορείτε να προσθέσετε 1 πύλη ακόμη        ");
        }
        else
        {
 tmp.printf("Μπορείτε να προσθέσετε %d πύλες ακόμη      ",maxobjects-arithmospilon);
        }
    }
```

Εμφάνιση μηνύματος για το πλήθος των πυλών στο κάτω μέρος της οθόνης

```
    Plaketa->Canvas->TextOutA(Plaketa->Width/2,katoorio-10,tmp);
```

Είμαστε εντός των ορίων της πλακέτας

```
    if((X<deksiorio)&&(Y<katoorio))
    {
```

Έχουμε πατήσει το κουμπί επιλογής πύλης. Το πρόγραμμα ελέγχει αν το ποντίκι μας έχει εισέλθει εντός των ορίων μιας πύλης.

Αν έχει εισέλθει και, επιπλέον, δεν είμαστε σε κατάσταση μετακίνησης πύλης (for_move = false), ο δείκτης του ποντικιού λαμβάνει την μορφή ενός χεριού που

δείχνει προς μια κατεύθυνση (παρόμοιο με το χεράκι που εμφανίζεται πάνω από τα hyperlinks κατά την πλοήγηση σε μια ιστοσελίδα).

Ο έλεγχος της τιμής της for_move εξασφαλίζει το ότι, σε περίπτωση που έχουμε επιλέξει να μετακινήσουμε π.χ. μια πύλη AND και, κατά την μετακίνησή μας, το ποντίκι περάσει από μια άλλη πύλη (είτε AND είτε κάποια άλλη) το πρόγραμμα θα συνεχίσει να μετακινεί την παλιά πύλη (στην περίπτωση αυτή, η τιμή της for_move θα είναι true).

Αν δεν έχει εισέλθει, ελέγχεται η τιμή της for_move.

Αν αυτή είναι true (μετακινούμε πύλη), ο δείκτης του mouse λαμβάνει την μορφή ενός χεριού που μετακινεί μια πύλη (το ποια πύλη θα είναι αυτή εξαρτάται από το ποια πύλη έχουμε επιλέξει να μετακινήσουμε).

Σε αντίθετη περίπτωση, ο δείκτης του mouse λαμβάνει την μορφή του απλού, γυμνού (χωρίς πύλη) χεριού.

```
if (ToolButtonSelect->Down==true)
{
    int left, top, bottom, right;
```

Ελέγχονται όλες οι πύλες.

```
    for (int i=8;i<arithmospilon;i++)
    {
        left=Pili[i].left-16;
        top=Pili[i].top-16;
        bottom=top+32;
        right=left+32;
```

Εντός των ορίων μιας πύλης, δεν μετακινούμε κάποια πύλη.

```
        if ((X>=left) && (X<=right) && (Y>=top) && (Y<=bottom) && (for_move==false))
        {
Screen->Cursors[crMyCursor]=LoadCursor(HInstance,MAKEINTRESOURCE(IDC_HANDPOINT_CURSOR));
            index_for_move=i;
            i=arithmospilon;
        }
```

Εκτός των ορίων των πυλών.

```
        else
        {
```

Μετακινούμε πύλη.

```
            if (for_move==true)
            {
                switch (Pili[index_for_move].type)
```

Αλλάζουμε τον δείκτη του mouse, ανάλογα με το ποια πύλη μετακινούμε.

```
                    {
case 0:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_ANDGRABBED_CURSOR));break;
case 1:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_ORGRABBED_CURSOR));break;
case 2:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NANDGRABBED_CURSOR));break;
case 3:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NORGRABBED_CURSOR));break;
case 4:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NOTGRABBED_CURSOR));break;
case 5:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_XORGRABBED_CURSOR));break;
case 7:Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_LEDGRABBED_CURSOR));break;
                    }
                }
```

Δεν μετακινούμε πύλη.

```
                else
                {
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_GRAB_CURSOR));
                    index_for_move=-1;
                }
            }
            Cursor = crMyCursor;
        }
    }
```

Έχουμε πατήσει το κουμπί σχεδίασης γραμμής.

Το πρόγραμμα ελέγχει, αρχικά, μια περιοχή διαστάσεων 10x10 pixels γύρω από το σημείο στο οποίο βρίσκεται ο δείκτης του ποντικιού (5 pixels δεξιά, 5 pixels αριστερά, 5 pixels πάνω και 5 pixels κάτω από το σημείο στο οποίο βρίσκεται ο δείκτης).

Αν εντοπιστεί κάποιο μπλε pixel στην περιοχή αυτή, ελέγχεται η τιμή της μεταβλητής ZografiseGrammi. Αν είναι false (σημάδι ότι ΔΕΝ βρισκόμαστε σε κατάσταση σχεδίασης γραμμής), ο δείκτης του ποντικιού μετατρέπεται σε χεράκι με τον δείκτη προτεταμένο (σημάδι ότι βρέθηκε σημείο εκκίνησης για σχεδίαση γραμμής) και μας δίνεται η δυνατότητα να ξεκινήσουμε την σχεδίαση μιας γραμμής. Σε αντίθετη περίπτωση (σημάδι ότι αυτό το μπλε pixel αποτελεί την έξοδο μιας άλλης πύλης), ο δείκτης του ποντικιού μετατρέπεται σε εικονίδιο απαγόρευσης (δεν μπορούμε να συνδέσουμε την έξοδο μιας πύλης ή το τέλος μιας τεθλασμένης γραμμής - μπλε pixels - με την έξοδο μιας άλλης πύλης ή το τέλος μια άλλης τεθλασμένης γραμμής).

Αν δεν εντοπιστεί κάποιο μπλε pixel, γίνεται εκ νέου αναζήτηση στην προαναφερθείσα περιοχή, αυτή τη φορά για το ενδεχόμενο ύπαρξης κόκκινου pixel. Σε περίπτωση που αυτό βρεθεί, ελέγχεται η τιμή της μεταβλητής ZografiseGrammi. Αν αυτή είναι true (είμαστε σε κατάσταση σχεδίασης γραμμών), ο δείκτης του ποντικιού μετατρέπεται σε χεράκι με τον δείκτη προτεταμένο. Αλλιώς, ο δείκτης του ποντικιού μετατρέπεται σε εικονίδιο απαγόρευσης.

Αν δεν βρεθεί ούτε και το κόκκινο pixel, δεν γίνεται τίποτα (ο δείκτης του ποντικιού παραμένει ο σταυρός της σχεδίασης γραμμών).

```
else if(ToolButtonLINE->Down==true)
{
    for(int k=-5;k<5;k++)//Ερευνα για μπλε pixel
    {
        for(int w=-5;w<5;w++)
        {
            if(Plaketa->Canvas->Pixels[X+k][Y+w]==clBlue)
            {
                X=X+k;
                Y=Y+w;
                k=5;
                w=5;
            }
        }
    }
```

Μπλε pixel.

```
    if(Plaketa->Canvas->Pixels[X][Y]==clBlue)
    {
```

Δεν σχεδιάζουμε γραμμή.

```
    if(ZografiseGrammi==false) Cursor=crHandPoint;
```

Σε κατάσταση σχεδίασης γραμμής.

```
    else Cursor=crNoDrop;
    }
```

Δεν βρήκαμε μπλε pixel.

```
    else
    {
```

Έρευνα για κόκκινο pixel.

```
        for(int k=-5;k<5;k++)
        {
            for(int w=-5;w<5;w++)
            {
                if(Plaketa->Canvas->Pixels[X+k][Y+w]==clRed)
```

```
                    {
                        X=X+k;
                        Y=Y+w;
                        k=5;
                        w=5;
                    }
                }
            }
```

Κόκκινο pixel.

```
            if(Plaketa->Canvas->Pixels[X][Y]==clRed)
            {
```

Σε κατάσταση σχεδίασης γραμμής.

```
                if((ZografiseGrammi==true)) Cursor=crHandPoint;
```

Δεν σχεδιάζουμε γραμμή.

```
                else Cursor=crNoDrop;
            }
```

Δεν βρέθηκε τίποτα.

```
            else Cursor=crCross;
        }
    }
```

Έχουμε πατήσει το κουμπί διαγραφής πύλης.

Εδώ, ελέγχονται οι συντεταγμένες του ποντικιού σε σχέση με μια περιοχή διαστάσεων 32x64 pixels γύρω από κάθε πύλη (16 pixels αριστερά, 16 pixels δεξιά, 32 pixels πάνω και 32 pixels κάτω).

Αν οι συντεταγμένες του ποντικιού βρεθούν εντός κάποιας πύλης, ο δείκτης του ποντικιού λαμβάνει την εικόνα του αρχείου eraser_found.cur (η οποία, μέσα στο αρχείο Resources1.rc -το αρχείο που περιέχει τον κατάλογο με όλα τα resources που χρησιμοποιεί το πρόγραμμα- έχει αντιστοιχιστεί στην ονομασία IDC_INVERA_CURSOR), και η αναζήτηση σταματάει, ενώ η μεταβλητή for_erase λαμβάνει την τιμή true (σημάδι ότι ετοιμαζόμαστε να διαγράψουμε πύλη). Σε αντίθετη περίπτωση, ο δείκτης του ποντικιού λαμβάνει την εικόνα του αρχείου eraser_found.cur (η οποία, μέσα στο αρχείο Resources1.rc έχει αντιστοιχιστεί στην ονομασία IDC_ERA_CURSOR), ενώ η μεταβλητή for_erase λαμβάνει την τιμή false.

```
    else if (ToolButtonErase->Down==true)
    {
        int left=0,top=0,bottom=0,right=0;

        for (int i=8;i<arithmospilon;i++)
```

```
        {
            left=Pili[i].left-16;
            top=Pili[i].top-16;
            bottom=top+32;
            right=left+32;

            if ((X>=left) && (X<=right) && (Y>=top) && (Y<=bottom))
            {
           Screen->Cursors[crMyCursor] = LoadCursor(HInstance,MAKEINTRESOURCE(IDC_INVERA_CURSOR));
                for_erase=true;
                index_for_erase=i;
                i=arithmospilon;
            }
            else
            {
              Screen->Cursors[crMyCursor] = LoadCursor(HInstance,MAKEINTRESOURCE(IDC_ERA_CURSOR));
                for_erase=false;
                index_for_erase=0;
            }
            Cursor = crMyCursor;
        }
    }
```

Έχουμε πατήσει το κουμπί διαγραφής γραμμής.

Η διαδικασία είναι παρόμοια με αυτήν για το κουμπί διαγραφής πύλης.

Ελέγχεται, λοιπόν, μια περιοχή 4x4 pixels (2 pixels αριστερά, 2 pixels δεξιά, 2 pixels πάνω και 2 pixels κάτω) γύρω από την θέση του ποντικιού.

Αν, εντός της περιοχής αυτής, βρεθεί κόκκινο pixel (τέλος γραμμής που συνδέεται με πύλη), καταγράφεται ο αριθμός σειράς της γραμμής αυτής (μεταβλητή = line_index_for_erase).

Αν, όμως, βρεθεί μπλε pixel (τέλος τεθλασμένης γραμμής), καταγράφεται ο αριθμός σειράς της γραμμής αυτής (μεταβλητή = tethlasmeni_index_for_erase). Και στις δύο αυτές περιπτώσεις, ο δείκτης του ποντικιού μετατρέπεται σε χεράκι με τον δείκτη προτεταμένο.

Αν, τυχόν, δεν βρεθεί κανένα μπλε ή κόκκινο pixel, ο δείκτης του ποντικιού μετατρέπεται σε εικονίδιο απαγόρευσης.

```
    else if (ToolButtonLineErase->Down==true)
    {
        bool tmp=false;
        for(int k=-2;k<2;k++)
        {
```

```
                for(int w=-2;w<2;w++)
                {
                    if (Plaketa->Canvas->Pixels[X+k][Y+w]==clRed)
                    {
                        for (int i=0;i<arithmosgrammon;i++)
                        {
                        if ((Lines[i].eosX==(X+k)) && (Lines[i].eosY==(Y+w)))
                            {
                                line_for_erase=true;
                                line_index_for_erase=i;
                                i=arithmosgrammon;
                            }
                        }
                        tmp=true;
                        k=2;w=2;
                    }
                    else if (Plaketa->Canvas->Pixels[X+k][Y+w]==clBlue)
                    {
                        for (int i=0;i<=plithos_tethlasmenon;i++)
   {   if   ((pinakas_tethlasmenis[i][index_tethlasmenis[i]-1][0]==(X+k))   &&   (pinakas_tethlasmenis[i][index_tethlasmenis[i]-
1][1]==(Y+w)))
                            {
                                tethlasmeni_for_erase=true;
                                tethlasmeni_index_for_erase=i;
                                i=plithos_tethlasmenon;
                            }
                        }
                        tmp=true;
                        k=2;w=2;
                    }
                }
            }
            if (tmp==true) Cursor=crHandPoint;
            else Cursor=crNo;
        }
        Image1->Top=Y+Plaketa->Top;
        Image1->Left=X+Plaketa->Left;

        Ytoadd=Y;
        Xtoadd=X;
    }
}
```

Η μέθοδος αυτή εκτελείται όταν πατήσουμε την εντολή Start (έναρξη εξομοίωσης κυκλώματος).

```
void __fastcall TForm1::Start1Click(TObject *Sender)
{
```

```
int tmp2=0;
AnsiString msg;
AnsiString tmpString;
```

1ο στάδιο: Βρίσκουμε ποιες γραμμές συνδέονται με ποια αντικείμενα (πύλες, διακόπτες, LEDs).

Για κάθε γραμμή, ελέγχουμε τις συντεταγμένες των σημείων εισόδου και εξόδου σε όλες τις πύλες.

Αν κάποιες από αυτές ταυτίζονται με τις συντεταγμένες της αρχής ή του τέλους κάποιας γραμμής, τότε ο αριθμός σειράς της αντίστοιχης πύλης καταχωρείται ή στην μεταβλητή ApoPiliIndex (αν μιλάμε για έναρξη γραμμής) ή στην μεταβλητή EosPiliIndex (αν μιλάμε για τέλος γραμμής). Στην 2η περίπτωση εξετάζεται, παράλληλα, αν η γραμμή μας τελειώνει στην 1η ή στην 2η είσοδο μιας πύλης.

Το αποτέλεσμα αυτό καταχωρείται στην μεταβλητή EosPiliNum.

```
for(int i=0;i<arithmosgrammon;i++)
{
    for(int z=0;z<arithmospilon;z++)
    {
        if((Lines[i].apoX==Pili[z].outX)&&(Lines[i].apoY==Pili[z].outY)) Lines[i].ApoPiliIndex=z;
        else
        {
            if((Lines[i].eosX==Pili[z].inX1)&&(Lines[i].eosY==Pili[z].inY1))
            {
                Lines[i].EosPiliIndex=z;
                Lines[i].EosPiliNum=1;
            }
            else
            {
                if((Lines[i].eosX==Pili[z].inX2)&&(Lines[i].eosY==Pili[z].inY2))
                {
                    Lines[i].EosPiliIndex=z;
                    Lines[i].EosPiliNum=2;
                }
            }
        }
    }
}
```

2ο στάδιο: Ελέγχουμε όλες τις γραμμές, ξεκινώντας από τις πύλες τύπου switch (type=6) στις οποίες είναι δεδομένες οι τιμές τους. Αν, λοιπόν, η πύλη από την οποία ξεκινάει η γραμμή μας είναι τύπου switch, τότε ελέγχεται αν το τέλος της γραμμής μας βρίσκεται στην 1η ή στην 2η είσοδο της άλλης πύλης. Σε κάθε

περίπτωση, εκτελείται η μέθοδος logic εκείνης της πύλης (του αντικειμένου της κλάσης DigiObject) και το αποτέλεσμά της καταχωρείται, αντίστοιχα, ή στην μεταβλητή in1val ή στην in2val (ιδιότητες της γραμμής).

```
for(int i=0;i<arithmosgrammon;i++)
{
    if (Pili[Lines[i].ApoPiliIndex].type==6)
    {
        if (Lines[i].EosPiliNum==2) Pili[Lines[i].EosPiliIndex].in2val=Pili[Lines[i].ApoPiliIndex].logic();
        else
        {
            if(Lines[i].EosPiliNum==1) Pili[Lines[i].EosPiliIndex].in1val=Pili[Lines[i].ApoPiliIndex].logic();
        }
    }
}
```

3ο στάδιο: Αφού υπολογίσαμε τις εισόδους των πυλών που συνδέονται με switch, ξεκινάει ένας βρόχος while για τον υπολογισμό των εισόδων/εξόδων των υπόλοιπων πυλών.

Η διαδικασία είναι ίδια με αυτή που ακολουθήθηκε στο 2ο στάδιο.

```
int tmpcounter=0;
bool found=true;

while(found)
{
    tmpcounter++;
```

Προσθέτουμε ένα όριο για κάθε ενδεχόμενο, ώστε να μην κρασάρει η εφαρμογή.

```
    if(tmpcounter==900)
    {
        msg.SetLength(0);
        msg.printf("Σφάλμα - Ελέξτε το κύκλωμα!");
        ShowMessage(msg);
        return;
    }
    found=false;
```

Για κάθε μια από τις γραμμές υπολογίζουμε τις εισόδους και εξόδους της.

Στην αρχή, όλες οι τιμές των πυλών είναι -1. Στο τέλος, δεν θα πρέπει να υπάρχουν είσοδοι με τιμή -1.

```
    for(int i=0;i<arithmosgrammon;i++)//Για όλες τις γραμμές
    {
```

Είναι κάποια πύλη, αλλά όχι διακόπτης ή LED.

```
        if(Pili[Lines[i].ApoPiliIndex].type<6          {
```

Επεξήγηση του παρακάτω if:

Η ροή του προγράμματος θα μπει εντός της if αν ισχύσει ένα εκ των παρακάτω ενδεχόμενων:

1ο ενδεχόμενο - Θα πρέπει να ισχύσουν ταυτόχρονα τα παρακάτω:

α) Ο τύπος (type) της πύλης από την οποία ξεκινάει η γραμμή μας (Pili[Lines[i].ApoPiliIndex]) δεν είναι 4 (δηλ. είναι οποιαδήποτε άλλη πύλη εκτός από NOT).

β) Η τιμή της in1val δεν είναι -1.

γ) Η τιμή της in2val δεν είναι -1.

(Τα δύο τελευταία δηλώνουν ότι η πύλη αυτή δεν είναι ξεκρέμαστη, δηλ. έχει σύνδεση με άλλες πύλες)

2ο ενδεχόμενο - Θα πρέπει να ισχύσουν ταυτόχρονα τα παρακάτω:

α) Ο τύπος (type) της πύλης από την οποία ξεκινάει η γραμμή μας (Pili[Lines[i].ApoPiliIndex]) είναι 4 (δηλ. η πύλη είναι NOT).

β) Η τιμή της in1val δεν είναι -1.

(Η τιμή της in2val αναγκαστικά είναι -1, οπότε αποφεύγουμε να την ελέγχουμε).

```
if (((Pili[Lines[i].ApoPiliIndex].type!=4) && (Pili[Lines[i].ApoPiliIndex].in1val!=-1) &&
(Pili[Lines[i].ApoPiliIndex].in2val!=-1)) || ((Pili[Lines[i].ApoPiliIndex].type==4) && (Pili[Lines[i].ApoPiliIndex].in1val!=-1)))
{
```

Από την στιγμή που υπάρχουν είσοδοι στην πύλη μας, θα υπάρχει και έξοδος.

Αυτή υπολογίζεται με την μέθοδο logic και το αποτέλεσμα καταχωρείται στην μεταβλητή outval.

```
Pili[Lines[i].ApoPiliIndex].outval=Pili[Lines[i].ApoPiliIndex].logic();
```

Γίνεται έλεγχος για το αν η γραμμή μας τελειώνει στην 2η είσοδο μιας άλλης πύλης (EosPiliNum==2).

Αν τελειώνει, τότε η 2η είσοδος της άλλης πύλης θα λάβει την τιμή της εξόδου της 1ης πύλης (in2val=outval).

```
if(Lines[i].EosPiliNum==2) Pili[Lines[i].EosPiliIndex].in2val=Pili[Lines[i].ApoPiliIndex].outval;
else
{
```

Γίνεται έλεγχος για το αν η γραμμή μας τελειώνει στην 1η είσοδο μιας άλλης πύλης (EosPiliNum==1).

Αν τελειώνει, τότε η 1η είσοδος της άλλης πύλης θα λάβει την τιμή της εξόδου της 1ης πύλης (in1val=outval).

```
                    if(Lines[i].EosPiliNum==1) Pili[Lines[i].EosPiliIndex].in1val=Pili[Lines[i].ApoPiliIndex].outval;
                }
            }
```

Αν έχει βρεθεί κάποια πύλη με είσοδο -1.

```
            else found=true;
        }
    }
}
```

Εκτυπώνουμε στις εξόδους και εισόδους το αποτέλεσμα του αλγορίθμου.

```
for(int i=0;i<arithmosgrammon;i++)
{
tmpString.SetLength(0);
tmpString.printf("%d",Pili[Lines[i].ApoPiliIndex].outval);
Plaketa->Canvas->TextOutA(Pili[Lines[i].ApoPiliIndex].outX,Pili[Lines[i].ApoPiliIndex].outY,tmpString);
    tmpString.SetLength(0);
}
```

Ξαναελέγχονται όλες οι γραμμές, και ελέγχεται αν το τέρμα τους οδηγεί σε LED (Pili[Lines[i].EosPiliIndex].type==7).

Αν είναι, ελέγχεται η είσοδος του LED. Αν είναι 1, το λαμπάκι φωτίζεται (φορτώνεται η εικόνα ledon.bmp), ενώ αν είναι 0, το λαμπάκι παραμένει σβηστό (φορτώνεται η εικόνα ledoff.bmp).

```
Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
for(int i=0;i<arithmosgrammon;i++)
{
    if(Pili[Lines[i].EosPiliIndex].type==7)//Eina tipou LED
    {
        if(Pili[Lines[i].EosPiliIndex].in1val==1)
        {
            Bitmap1->LoadFromFile(AppDir+"\\ledon.bmp");
            Plaketa->Canvas->Draw(Pili[Lines[i].EosPiliIndex].left,Pili[Lines[i].EosPiliIndex].top,Bitmap1);
        }
        else
        {
            Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");
            Plaketa->Canvas->Draw(Pili[Lines[i].EosPiliIndex].left,Pili[Lines[i].EosPiliIndex].top,Bitmap1);
        }
    }
```

```
    }
}
```

Μέθοδος για το άνοιγμα ενός αρχείου (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί Open της μπάρας εργασιών).

```
void __fastcall TForm1::Open1Click(TObject *Sender)
{
    ToolButtonOpenClick(Sender);
}
```

Μέθοδος για την δημιουργία νέου σχεδίου.

```
void __fastcall TForm1::New1Click(TObject *Sender)
{
```

Αρχικά το πρόγραμμα ρωτάει αν θέλουμε να δημιουργηθεί νέο αρχείο.

```
    if (MessageDlg("Είστε σίγουροι ότι θέλετε νέο αρχείο;", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == mrYes)
    {
```

Αν θέλουμε να σώσουμε το αρχείο μας.

```
        if (MessageDlg("Επιθυμείτε να αποθηκεύσετε το προηγούμενο αρχείο;", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == mrYes) ToolButtonSAVEClick(Sender);
```

Αρχικοποιούνται όλες οι μεταβλητές του προγράμματος.

```
        arithmospilon=0;
        arithmosgrammon=0;
        for (int i=0;i<200;i++) Lines[i].LineStatus=0;
        for (int i=0;i<2000;i++) index_tethlasmenis[i]=0;
        ZografiseGrammi=true;
        EndiamesesGrammes=false;
        for_erase=false;
        line_for_erase=false;
        index_for_erase=-1;
        line_index_for_erase=-1;
        plithos_tethlasmenon=0;
        FormShow(this);
        Undo1->Enabled=false;
    }
}
```

Μέθοδος για την αποθήκευση ενός αρχείου (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί Save της μπάρας εργασιών).

```
void __fastcall TForm1::Save1Click(TObject *Sender)
{
    ToolButtonSAVEClick(Sender);
}
```

Η μέθοδος αυτή εκτελείται όταν ο χρήστης πατήσει το πλήκτρο AND στην μπάρα εργασιών.

Αρχικά ο δείκτης του ποντικιού μετατρέπεται σε πύλη AND.

Στη συνέχεια, όλα τα άλλα πλήκτρα της μπάρας εργαλείων (πύλες, γραμμές) απενεργοποιούνται (δεν είναι πατημένα) και ενεργοποιείται το πλήκτρο AND (πατημένο).

```
void __fastcall TForm1::ToolButtonANDClick(TObject *Sender)
{
    Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_AND_CURSOR));
    Cursor = crMyCursor;
    isSelection = false;
    ToolButtonAND->Down=true;
    ToolButtonOR->Down=false;
    ToolButtonNAND->Down=false;
    ToolButtonNOR->Down=false;
    ToolButtonNOT->Down=false;
    ToolButtonLED->Down=false;
    ToolButtonLINE->Down=false;
    ToolButtonXOR->Down=false;
    ToolButtonErase->Down=false;
    ToolButtonLineErase->Down=false;
    ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου OR (όπως με το AND).

```
void __fastcall TForm1::ToolButtonORClick(TObject *Sender)
{
isSelection = false;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_OR_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=true;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου NAND (όπως με το AND).

```
void __fastcall TForm1::ToolButtonNANDClick(TObject *Sender)
```

```
{
isSelection = false;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NAND_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=true;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου NOR (όπως με το AND).

```
void __fastcall TForm1::ToolButtonNORClick(TObject *Sender)
{
isSelection = false;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NOR_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=true;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου NOT (όπως με το AND).

```
void __fastcall TForm1::ToolButtonNOTClick(TObject *Sender)
{
isSelection = false;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_NOT_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=true;
 ToolButtonLED->Down=false;
```

```
  ToolButtonLINE->Down=false;
  ToolButtonXOR->Down=false;
  ToolButtonErase->Down=false;
  ToolButtonLineErase->Down=false;
  ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου XOR (όπως με το AND).

```
void __fastcall TForm1::ToolButtonXORClick(TObject *Sender)
{
isSelection = false;
  Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_XOR_CURSOR));
  Cursor = crMyCursor;
  ToolButtonAND->Down=false;
  ToolButtonOR->Down=false;
  ToolButtonNAND->Down=false;
  ToolButtonNOR->Down=false;
  ToolButtonNOT->Down=false;
  ToolButtonLED->Down=false;
  ToolButtonLINE->Down=false;
  ToolButtonXOR->Down=true;
  ToolButtonErase->Down=false;
  ToolButtonLineErase->Down=false;
  ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου LED (όπως με το AND).

```
void __fastcall TForm1::ToolButtonLEDClick(TObject *Sender)
{
isSelection = false;
  Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_LED_CURSOR));
  Cursor = crMyCursor;
  ToolButtonAND->Down=false;
  ToolButtonOR->Down=false;
  ToolButtonNAND->Down=false;
  ToolButtonNOR->Down=false;
  ToolButtonNOT->Down=false;
  ToolButtonLED->Down=true;
  ToolButtonLINE->Down=false;
  ToolButtonXOR->Down=false;
  ToolButtonErase->Down=false;
  ToolButtonLineErase->Down=false;
  ToolButtonSelect->Down=false;
//  Cursor=crNone;

}
```

Μέθοδος για το πάτημα του πλήκτρου LINE (όπως με το AND).

```
void __fastcall TForm1::ToolButtonLINEClick(TObject *Sender)
{
isSelection = false;
 Cursor=crNone;
 ZografiseGrammi=false;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=true;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=false;
Cursor=crCross;
}
```

Μέθοδος για το πάτημα του πλήκτρου Select (για επιλογή πυλών, όπως με το AND).

```
void __fastcall TForm1::ToolButtonSelectClick(TObject *Sender)
{
 isSelection = true;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_GRAB_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=true;

}
```

Μέθοδος για το πάτημα του πλήκτρου Erase (για διαγραφή πυλών, όπως με το AND).

```
void __fastcall TForm1::ToolButtonEraseClick(TObject *Sender)
{
isSelection = false;
 Screen->Cursors[crMyCursor] = LoadCursor(HInstance, MAKEINTRESOURCE(IDC_ERA_CURSOR));
 Cursor = crMyCursor;
 ToolButtonAND->Down=false;
```

```
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=true;
 ToolButtonLineErase->Down=false;
 ToolButtonSelect->Down=false;
}
```

Μέθοδος για το πάτημα του πλήκτρου LineErase (για διαγραφή γραμμών, όπως με το AND).

```
void __fastcall TForm1::ToolButtonLineEraseClick(TObject *Sender)
{
isSelection = false;
 Cursor=crCross;
 ToolButtonAND->Down=false;
 ToolButtonOR->Down=false;
 ToolButtonNAND->Down=false;
 ToolButtonNOR->Down=false;
 ToolButtonNOT->Down=false;
 ToolButtonLED->Down=false;
 ToolButtonLINE->Down=false;
 ToolButtonXOR->Down=false;
 ToolButtonErase->Down=false;
 ToolButtonLineErase->Down=true;
 ToolButtonSelect->Down=false;
}
```

Μέθοδος για την έναρξη της εξομοίωσης ενός κυκλώματος (από την μπάρα εργασιών μέθοδος με αυτήν που εκτελείται όταν πατάμε την εντολή Simulate του μενού) Start1Click.

```
void __fastcall TForm1::ToolButtonSimulateClick(TObject *Sender)
{
    Start1Click(Sender);
}
```

Η μέθοδος αυτή εκτελείται όταν ο χρήστης πατήσει το πλήκτρο A της μπάρας εργασιών.

Αν το πλήκτρο A παραμείνει πατημένο, η τιμή της εξόδου του διακόπτη A θα είναι 1 αλλιώς θα είναι 0. Αμέσως μετά, ο χρήστης ενημερώνεται ότι πρέπει να ξεκινήσει εκ νέου την διαδικασία εξομοίωσης.

```
void __fastcall TForm1::ToolButtonPowerAClick(TObject *Sender)
```

```
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[0].in1val==0)
      {
 Pili[0].in1val=1;
 ToolButtonPowerA->Down=true;
 ShowMessage("Θέσατε τι τιμή εισόδου A σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
 Plaketa->Canvas->Draw(Pili[0].left,Pili[0].top,Bitmap1);
      }
 else
      {
 Pili[0].in1val=0;
 ToolButtonPowerA->Down=false;
 ShowMessage("Θέσατε τι τιμή εισόδου A σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
 Plaketa->Canvas->Draw(Pili[0].left,Pili[0].top,Bitmap1);
      }
}
```

Μέθοδος για το πάτημα του πλήκτρου B (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerBClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[1].in1val==0)
 {
  Pili[1].in1val=1;
  ToolButtonPowerB->Down=true;
  ShowMessage("Θέσατε τι τιμή εισόδου B σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
  Plaketa->Canvas->Draw(Pili[1].left,Pili[1].top,Bitmap1);
 }
else
{
  Pili[1].in1val=0;
  ToolButtonPowerB->Down=false;
  ShowMessage("Θέσατε τι τιμή εισόδου B σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
  Plaketa->Canvas->Draw(Pili[1].left,Pili[1].top,Bitmap1);
 }
// Start1Click(Sender);
}
```

Μέθοδος για το πάτημα του πλήκτρου C (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerCClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[2].in1val==0)
```

```
 {
  Pili[2].in1val=1;
  ToolButtonPowerC->Down=true;
  ShowMessage("Θέσατε τι τιμή εισόδου C σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
  Plaketa->Canvas->Draw(Pili[2].left,Pili[2].top,Bitmap1);
 }
else
{
  Pili[2].in1val=0;
  ToolButtonPowerC->Down=false;
  ShowMessage("Θέσατε τι τιμή εισόδου C σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
  Plaketa->Canvas->Draw(Pili[2].left,Pili[2].top,Bitmap1);
 }
// Start1Click(Sender);
}
```

Μέθοδος για το πάτημα του πλήκτρου D (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerDClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[3].in1val==0)
 {
  Pili[3].in1val=1;
  ToolButtonPowerD->Down=true;
  ShowMessage("Θέσατε τι τιμή εισόδου D σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
  Plaketa->Canvas->Draw(Pili[3].left,Pili[3].top,Bitmap1);
 }
else
{
  Pili[3].in1val=0;
  ToolButtonPowerD->Down=false;
  ShowMessage("Θέσατε τι τιμή εισόδου D σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
  Plaketa->Canvas->Draw(Pili[3].left,Pili[3].top,Bitmap1);
 }
// Start1Click(Sender);
}
```

Μέθοδος για το πάτημα του πλήκτρου E (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerEClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[4].in1val==0)
 {
  Pili[4].in1val=1;
```

```
 ToolButtonPowerE->Down=true;
 ShowMessage("Θέσατε τι τιμή εισόδου E σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
 Plaketa->Canvas->Draw(Pili[4].left,Pili[4].top,Bitmap1);
}
else
{
 Pili[4].in1val=0;
 ToolButtonPowerE->Down=false;
 ShowMessage("Θέσατε τι τιμή εισόδου E σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
 Plaketa->Canvas->Draw(Pili[4].left,Pili[4].top,Bitmap1);
}
}
```

Μέθοδος για το πάτημα του πλήκτρου F (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerFClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[5].in1val==0)
 {
 Pili[5].in1val=1;
 ToolButtonPowerF->Down=true;
 ShowMessage("Θέσατε τι τιμή εισόδου F σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
 Plaketa->Canvas->Draw(Pili[5].left,Pili[5].top,Bitmap1);
}
else
{
 Pili[5].in1val=0;
 ToolButtonPowerF->Down=false;
 ShowMessage("Θέσατε τι τιμή εισόδου F σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
 Plaketa->Canvas->Draw(Pili[5].left,Pili[5].top,Bitmap1);
}
}
```

Μέθοδος για το πάτημα του πλήκτρου G (όπως το A).

```
void __fastcall TForm1::ToolButtonPowerGClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[6].in1val==0)
 {
 Pili[6].in1val=1;
 ToolButtonPowerG->Down=true;
 ShowMessage("Θέσατε τι τιμή εισόδου G σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
 Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
 Plaketa->Canvas->Draw(Pili[6].left,Pili[6].top,Bitmap1);
```

```
 }
else
{
  Pili[6].in1val=0;
  ToolButtonPowerG->Down=false;
  ShowMessage("Θέσατε τι τιμή εισόδου G σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
  Plaketa->Canvas->Draw(Pili[6].left,Pili[6].top,Bitmap1);
 }
}
```

Μέθοδος για το πάτημα του πλήκτρου Η (όπως το Α).

```
void __fastcall TForm1::ToolButtonPowerHClick(TObject *Sender)
{
 Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
 if(Pili[7].in1val==0)
 {
  Pili[7].in1val=1;
  ToolButtonPowerH->Down=true;
  ShowMessage("Θέσατε τι τιμή εισόδου Η σε 1. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
  Plaketa->Canvas->Draw(Pili[7].left,Pili[7].top,Bitmap1);
 }
else
{
  Pili[7].in1val=0;
  ToolButtonPowerH->Down=false;
  ShowMessage("Θέσατε τι τιμή εισόδου Η σε 0. Πιέστε για εκκίνηση προσωμοίωσης.");
  Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
  Plaketa->Canvas->Draw(Pili[7].left,Pili[7].top,Bitmap1);
 }
}
```

Μέθοδος για την επιλογή της πύλης AND (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί AND της μπάρας εργασιών).

```
void __fastcall TForm1::AND1Click(TObject *Sender)
{
 ToolButtonANDClick(Sender);
}
```

Μέθοδος για την επιλογή της πύλης OR (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί OR της μπάρας εργασιών).

```
void __fastcall TForm1::OR1Click(TObject *Sender)
{
 ToolButtonORClick(Sender);
}
```

Μέθοδος για την επιλογή της πύλης NOR (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί NOR της μπάρας εργασιών).

```
void __fastcall TForm1::NOR1Click(TObject *Sender)
{
 ToolButtonNORClick(Sender);
}
```

Μέθοδος για την επιλογή της πύλης NAND (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί NAND της μπάρας εργασιών).

```
void __fastcall TForm1::NAND1Click(TObject *Sender)
{
 ToolButtonNANDClick(Sender);
}
```

Μέθοδος για την επιλογή της πύλης NOT (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί NOT της μπάρας εργασιών)

```
void __fastcall TForm1::NOT1Click(TObject *Sender)
{
 ToolButtonNOTClick(Sender);
}
```

Μέθοδος για την επιλογή της πύλης XOR (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί XOR της μπάρας εργασιών).

```
void __fastcall TForm1::XOR1Click(TObject *Sender)
{
 ToolButtonXORClick(Sender);
}
```

Μέθοδος για την επιλογή του LED (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί LED της μπάρας εργασιών).

```
void __fastcall TForm1::LED1Click(TObject *Sender)
{
 ToolButtonLEDClick(Sender);
}
```

Μέθοδος για την επιλογή του εργαλείου γραμμών (από το μενού, εκτελείται η ίδια μέθοδος με αυτήν που εκτελείται όταν πατάμε το κουμπί Line της μπάρας εργασιών).

```
void __fastcall TForm1::LINE1Click(TObject *Sender)
{
ToolButtonLINEClick(Sender);
}
```

Μέθοδος για την αποθήκευση του κυκλώματος σε αρχείο τύπου sim.

```
void __fastcall TForm1::ToolButtonSAVEClick(TObject *Sender)
{
```

```
    AnsiString filename;
```

Το όνομα του αρχείου.

```
    if(SaveDialog1->Execute()) filename = SaveDialog1->FileName;
    else return;

    FILE *stream;
    if((stream = fopen(filename.c_str(),"wb"))==NULL) ShowMessage("Δεν ήταν δυνατή η εγγραφή του αρχείου ή επιλέξατε ακύρωση.");
```

Αποθηκεύονται τα στοιχεία του σχεδίου.

```
    else
    {
        fwrite(&arithmospilon,sizeof(int),1,stream);
        fwrite(&Pili,sizeof(Pili),1,stream);
        fwrite(&arithmosgrammon,sizeof(int),1,stream);
        fwrite(&Lines,sizeof(Lines),1,stream);
        fwrite(&plithos_tethlasmenon,sizeof(int),1,stream);
        fwrite(&index_tethlasmenis,sizeof(index_tethlasmenis),1,stream);
        fwrite(&pinakas_tethlasmenis,sizeof(pinakas_tethlasmenis),1,stream);
        fclose(stream);
    }
}
```

Μέθοδος για τον τερματισμό της εφαρμογής (το πρόγραμμα ρωτάει αν θέλουμε να σώσουμε το σχέδιό μας).

```
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
    if (MessageDlg("Επιθυμείτε να αποθηκεύσετε το τρέχων κύκλωμα;", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == mrYes) ToolButtonSAVEClick(Sender);
    exit(0);
}
```

Μέθοδος για το άνοιγμα αρχείου sim. Αφού ανοιχτεί το αρχείο, το κύκλωμα σχεδιάζεται στην οθόνη μας.

```
void __fastcall TForm1::ToolButtonOpenClick(TObject *Sender)
{
    int storedarithmospilon;
    int storedarithmosgrammon;
    int storedtethlasmenes;
    DigiObject PiliTemp[110];
    LineObject LineTemp[200];
    AnsiString filename;
    if (MessageDlg("Επιθυμείτε να αποθηκεύσετε το προηγούμενο αρχείο;", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) == mrYes)
    {
        ToolButtonSAVEClick(Sender);
    }
```

```
if(OpenDialog1->Execute()) filename = OpenDialog1->FileName;
else return;
FILE *stream;
if((stream = fopen(filename.c_str(),"rb"))==NULL) ShowMessage("Δεν ήταν δυνατή η ανάγνωση του αρχείου");
else
{
```

Φορτώνονται τα στοιχεία του αρχείου.

```
Plaketa->Canvas->FillRect(Plaketa->Canvas->ClipRect);
fseek(stream, SEEK_SET, 0);
fread(&storedarithmospilon,sizeof(int),1,stream);
fread(&Pili, sizeof(DigiObject), 110, stream);
fread(&storedarithmosgrammon,sizeof(int),1,stream);
fread(&Lines, sizeof(LineObject), 200, stream);
fread(&storedtethlasmenes,sizeof(int),1,stream);
fread(&index_tethlasmenis,sizeof(int),2000,stream);
fread(&pinakas_tethlasmenis,sizeof(int),120000,stream);
```

Το κύκλωμα σχεδιάζεται από την αρχή.

```
arithmosgrammon=storedarithmosgrammon;
arithmospilon=storedarithmospilon;
plithos_tethlasmenon=storedtethlasmenes;

char SwitchName[8]={'A','B','C','D','E','F','G','H'};
Image1->Visible=false;
Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();

Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
int X=30;
int Y=50;
int diffY=50;
for(int i=0;i<8;i++)
{
 Plaketa->Canvas->Draw(X,Y+diffY*i-Bitmap1->Height*0.5,Bitmap1);
 Plaketa->Canvas->TextOut(X-10,Y+diffY*i-Bitmap1->Height*0.5,SwitchName[i]);
 Pili[i].Store(6,X,Y+diffY*i-Bitmap1->Height*0.5,0);
Pili[i].outX=X+30;
Pili[i].outY=Y+diffY*i-Bitmap1->Height*0.5+21;
}

if (Pili[0].in1val==0) ToolButtonPowerA->Down=false;
else ToolButtonPowerA->Down=true;
if (Pili[1].in1val==0) ToolButtonPowerB->Down=false;
else ToolButtonPowerB->Down=true;
if (Pili[2].in1val==0) ToolButtonPowerC->Down=false;
else ToolButtonPowerC->Down=true;
if (Pili[3].in1val==0) ToolButtonPowerD->Down=false;
else ToolButtonPowerD->Down=true;
```

```
    if (Pili[4].in1val==0) ToolButtonPowerE->Down=false;
    else ToolButtonPowerE->Down=true;
    if (Pili[5].in1val==0) ToolButtonPowerF->Down=false;
    else ToolButtonPowerF->Down=true;
    if (Pili[6].in1val==0) ToolButtonPowerG->Down=false;
    else ToolButtonPowerG->Down=true;
    if (Pili[7].in1val==0) ToolButtonPowerH->Down=false;
    else ToolButtonPowerH->Down=true;

    for(int i=0;i<storedarithmospilon;i++)
    {
        switch(Pili[i].type)
        {
            case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
            case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
            case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
            case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
            case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
            case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
            case 6:
     if (Pili[i].in1val==1) Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
     else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
     break;
            case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
        }
        Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
    }

    for(int i=0;i<storedarithmosgrammon;i++)
    {
        if (Lines[i].LineStatus==0)
        {
            Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
            Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
        }
    }
    for (int i=0;i<=plithos_tethlasmenon;i++)
    {
        for (int k=0;k<index_tethlasmenis[i]-1;k++)
        {
             Plaketa->Canvas->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
            Plaketa->Canvas->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
            Plaketa->Canvas->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
        }
    }
    fclose(stream);
}
```

```
}
```

Εκτελείται όταν ξεκινήσει το πρόγραμμα, ορίζει τον τρέχοντα φάκελο ως τον κυρίως φάκελο της εφαρμογής.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    AppDir=GetCurrentDir();
}

void __fastcall TForm1::Undo1Click(TObject *Sender)
{
```

Αναίρεση της τελευταίας ενέργειας. Όλες οι ενέργειες είναι αποθηκευμένες σε πίνακα.

Κάθε φορά που εκτελείται το undo ενημερώνονται οι πίνακες στους οποίους έχουμε αποθηκεύσει τα δεδομένα μας, και το σχέδιο ξανασχεδιάζεται από την αρχή.

```
    int temparithmospilon=arithmospilon;
    int temparithmosgrammon=arithmosgrammon;

    bool tempPower[8];

    tempPower[0]=ToolButtonPowerA->Down;
    tempPower[1]=ToolButtonPowerB->Down;
    tempPower[2]=ToolButtonPowerC->Down;
    tempPower[3]=ToolButtonPowerD->Down;
    tempPower[4]=ToolButtonPowerE->Down;
    tempPower[5]=ToolButtonPowerF->Down;
    tempPower[6]=ToolButtonPowerG->Down;
    tempPower[7]=ToolButtonPowerH->Down;

    Undo1->Enabled = false;
    arithmospilon=0;
    arithmosgrammon=0;
    FormShow(this);
    arithmosgrammon=temparithmosgrammon;
    arithmospilon=temparithmospilon;

    switch (lastaction)
    {
```

Η τελευταία ενέργεια ήταν προσθήκη πύλης.

```
        case 100:arithmospilon--;break;
```

Η τελευταία ενέργεια ήταν προσθήκη γραμμής.

```
        case 200:arithmosgrammon--;break;
```

Η τελευταία ενέργεια ήταν κλείσιμο τεθλασμένης γραμμής.

```
        case 300:index_tethlasmenis[plithos_tethlasmenon]--;break;
```

Η τελευταία ενέργεια ήταν συνέχιση (και όχι κλείσιμο) τεθλασμένης γραμμής.

```
        case 400:
        {
            plithos_tethlasmenon--;
            index_tethlasmenis[plithos_tethlasmenon]--;
            arithmosgrammon--;
        }
    }
```

Το κύκλωμα σχεδιάζεται από την αρχή.

```
    lastaction=0;
    ToolButtonPowerA->Down=tempPower[0];
    ToolButtonPowerB->Down=tempPower[1];
    ToolButtonPowerC->Down=tempPower[2];
    ToolButtonPowerD->Down=tempPower[3];
    ToolButtonPowerE->Down=tempPower[4];
    ToolButtonPowerF->Down=tempPower[5];
    ToolButtonPowerG->Down=tempPower[6];
    ToolButtonPowerH->Down=tempPower[7];
    Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
    for (int i=0;i<8;i++)
    {
      if (tempPower[i]==true) Bitmap1->LoadFromFile(AppDir+"\\switchon.bmp");
      else Bitmap1->LoadFromFile(AppDir+"\\switchoff.bmp");
     Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
    }


    for(int i=0;i<arithmospilon;i++)
    {
        switch(Pili[i].type)
        {
            case 0:Bitmap1->LoadFromFile(AppDir+"\\and1.bmp");break;
            case 1:Bitmap1->LoadFromFile(AppDir+"\\or1.bmp");break;
            case 2:Bitmap1->LoadFromFile(AppDir+"\\nand1.bmp");break;
            case 3:Bitmap1->LoadFromFile(AppDir+"\\nor1.bmp");break;
            case 4:Bitmap1->LoadFromFile(AppDir+"\\not1.bmp");break;
            case 5:Bitmap1->LoadFromFile(AppDir+"\\xor1.bmp");break;
            case 7:Bitmap1->LoadFromFile(AppDir+"\\ledoff.bmp");break;
        }
        Plaketa->Canvas->Draw(Pili[i].left,Pili[i].top,Bitmap1);
    }
    for(int i=0;i<arithmosgrammon;i++)
    {
        if (Lines[i].LineStatus==0)
```

```
        {
            Plaketa->Canvas->MoveTo(Lines[i].apoX,Lines[i].apoY);
            Plaketa->Canvas->LineTo(Lines[i].eosX,Lines[i].eosY);
        }
    }
    for (int i=0;i<=plithos_tethlasmenon;i++)
    {
        for (int k=0;k<index_tethlasmenis[i]-1;k++)
        {
            Plaketa->Canvas->MoveTo(pinakas_tethlasmenis[i][k][0],pinakas_tethlasmenis[i][k][1]);
            Plaketa->Canvas->LineTo(pinakas_tethlasmenis[i][k+1][0],pinakas_tethlasmenis[i][k+1][1]);
            Plaketa->Canvas->Pixels[pinakas_tethlasmenis[i][k+1][0]][pinakas_tethlasmenis[i][k+1][1]]=clBlue;
        }
    }
}
```

Κλείσιμο του προγράμματος (με ερώτηση αν θέλουμε να σώσουμε το κύκλωμά μας).

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    if (MessageDlg("Επιθυμείτε να αποθηκεύσετε το τρέχων κύκλωμα;", mtConfirmation, TMsgDlgButtons() << mbYes <<
mbNo, 0) == mrYes)
    {
        ToolButtonSAVEClick(Sender);
    }
    exit(0);
}
```

ΤΕΛΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ

Η χρήση των ψηφιακών Κυκλωμάτων είναι ευρέως διαδεδομένη και συναντάται στον κόσμο μας από τα πιο απλά αντικείμενα (π.χ. οικιακές συσκευές) έως τα πιο περίπλοκα (π.χ. υπερυπολογιστές). Η δημιουργία ενός Εξομοιωτή Ψηφιακών Κυκλωμάτων είναι επιτακτική έτσι ώστε να διευκολυνθεί η έρευνα των διαφόρων επιστημών που ασχολούνται με τη σχεδίαση και υλοποίηση Ψηφιακών Κυκλωμάτων.

Οι δυνατότητες του παρόντος Εξομοιωτή Ψηφιακών Κυκλωμάτων περιορίζεται από το μικρό αριθμό πηγών - εισόδων που προσφέρει και από τον αριθμό των Λογικών πυλών που παρέχει στον χρήστη. Ο εμπλουτισμός του με Συνδυαστικά Κυκλώματα, όπως ο αθροιστής, θα μπορούσε να διευρύνει τις δυνατότητες του προγράμματος σε επιστημονικές χρήσεις και μελέτες.

Ένα περιβάλλον στο οποίο ο παρών Εξομοιωτής θα μπορούσε να αποτελέσει εργαλείο - βοήθεια στον εκάστοτε χρήστη του μπορεί να θεωρηθεί η χρήση στο εργαστηριακό κομμάτι του μαθήματος Ψηφιακά Κυκλώματα που εντάσσεται στο πρόγραμμα σπουδών του τμήματος Πληροφορικής & Επικοινωνιών του Α.Τ.Ε.Ι. Σερρών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Floyd T., “Digital Fundamentals”, Macmillan, 1990
- Leach D., Malvino A., “Ψηφιακά Ηλεκτρονικά”, Εκδόσεις ΤΖΙΟΛΑ, 1996
- Tokheim R., “Ψηφιακά Ηλεκτρονικά”, Εκδόσεις ΤΖΙΟΛΑ
- Hall D., “Digital Circuits and Systems”, McGraw - Hill, 1989
- Glaser A., “A History of Binary and Other Nondecimal Numeration”, Southampton PA, Tomash, 1971
- Gregg J., “Ones and Zeroes Understanding Boolean Algebra, Digital Circuits and the Logic of Sets”, IEEE Press, 1998
- Ifrah G., “From One to Zero: a Universal History of Numbers”, New York, Viking, 1985
- Kohavi Z., “Switching and Finite Automata Theory”, McGraw Hill
- Menninger K., “Number Words and Number Symbols: A Cultural History of Numbers”, Cambridge, MIT, 1969
- Hamming R., Error Detecting and Error Correcting Codes, Bell System Tech. J., vol 29, pp.147–160, April 1950
- Rao T., Fujiwara E., “Error–Control Coding for Computer Systems”, Prentice Hall, 1989
- Sandige R., “Modern Digital Design”, McGraw–Hill, 1990
- Davio, Deschamps, Thayse, “Digital Systems with algorithm implementation”, Wiley & Sons (1983)
- Bryant R.E., “Graph–based algorithms for Boolean function manipulation”,
- IEEE Trans Computers, vol C–35, no 8, pp. 677–691, August 1986
- Harrison M., “Introduction to Switching and Automata Theory”, McGraw - Hill, 1965
- Rosenbloom P., “The Elements of Mathematical Logic”, Dover, 1950
- Boole G., “An Investigation of the Laws of Thought”, Dover, 1954

- Huntington E., “Sets of independent postulates for the algebra of logic”, Trans. of the American Mathematical Society vol 5 (1904)
- Shannon C., “A Symbolic Analysis of Relay and Switching Circuits”, Trans. of the AIEE, vol 57 (1938)
- De Morgan Augustus, “On the Syllogism and Other Logical Writings”, New Haven, Yale, 1966
- Couturat L., “The Algebra of Logic”, Chicago, Open Court, 1914
- Morris Mano, “Ψηφιακή Σχεδίαση”, Prentice - Hall 1991 (Εκδόσεις Παπασωτηρίου)
- Schaum's Series, “Boolean Algebra and Switching Circuits”, McGraw Hill
- Gajski D., “Principles of Digital Design”, Prentice - Hall 1997
- Unger, “The Essence of Logic Circuits”, IEEE Press 1997
- M. Breuer, A. Friedman, “Diagnosis and Reliable Design of Digital Systems”, Computer Science Press (1976)
- M. Abramovici, M. Breuer, A. Friedman, “Digital Systems Testing and Testable Design”, IEEE Press (1990)
- McCalla T. R., “Digital Logic and Computer Design”, Maxwell - Macmillan (1992)
- Karnaugh M., “A map method for synthesis of combinatorial logic circuits”, Trans. AIEE, Communications and Electronics, vol. 72, Part I (November 1953)
- McCluskey E., “Logic Design Priciples”, Prentice Hall 1986
- Quine W., “The problem of simplifying truth functions”, American Mathematical Monthly, vol. 59 (October 1952)
- Brayton R., Hachtel G., McMullen C., Sangiovanni - Vincentelli A. “Logic Minimization Algorithms for VLSI Synthesis”, Kluwer 1984
- Sandige R., “Modern Digital Design” McGraw - Hill, 1990