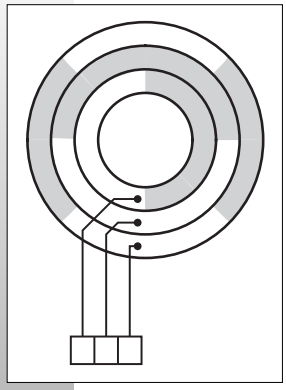


κεφάλαιο 2

Αριθμητικά συστήματα και κώδικες

Τα ψηφιακά συστήματα είναι κατασκευασμένα από κυκλώματα τα οποία επεξεργάζονται δυαδικά ψηφία — 0 και 1, όμως στην πράξη πολύ λίγα πραγματικά προβλήματα βασίζονται σε δυαδικούς αριθμούς ή σε αριθμούς γενικότερα. Γι' αυτόν το λόγο, ο σχεδιαστής ψηφιακών συστημάτων πρέπει να ορίσει κάποια αντιστοιχία μεταξύ των δυαδικών ψηφίων, τα οποία επεξεργάζονται τα ψηφιακά συστήματα, και των πραγματικών αριθμών, των συμβάντων, και των συνθηκών. Ο σκοπός αυτού του κεφαλαίου είναι να δείξει στον αναγνώστη τον τρόπο αναπαράστασης και χειρισμού των γνωστών αριθμητικών ποσοτήτων σε ένα ψηφιακό σύστημα, καθώς επίσης και τον τρόπο αναπαράστασης μη αριθμητικών δεδομένων, συμβάντων, και συνθηκών.

Οι πρώτες εννέα ενότητες περιγράφουν τα δυαδικά αριθμητικά συστήματα και δείχνουν πώς πραγματοποιούνται η πρόσθεση, η αφαίρεση, ο πολλαπλασιασμός, και η διαίρεση στα συστήματα αυτά. Οι ενότητες 2.10-2.13 δείχνουν πώς γίνεται η κωδικοποίηση άλλων πραγμάτων, όπως π.χ. δεκαδικών αριθμών, χαρακτήρων κειμένου, μηχανικών θέσεων, και οποιωνδήποτε συνθηκών, με τη βοήθεια ακολουθιών δυαδικών ψηφίων.

Η Ενότητα 2.14 παρουσιάζει τους "n-διάστατους κύβους", οι οποίοι παρέχουν έναν τρόπο εποπτείας της σχέσης μεταξύ διαφορετικών ψηφιοσειρών (bit strings). Οι n-διάστατοι κύβοι είναι ιδιαίτερα χρήσιμοι στη μελέτη των κωδίκων ανίχνευσης σφαλμάτων στην Ενότητα 2.15. Τέλος, το κεφάλαιο αυτό κλείνει με μια εισαγωγή στους κώδικες για τη μετάδοση και αποθήκευση δεδομένων κατά ένα bit τη φορά.

2.1 Αριθμητικά συστήματα θέσης

Τα παραδοσιακό αριθμητικό σύστημα που έχουμε μάθει στο σχολείο και χρησιμοποιούμε καθημερινά στη δουλειά μας ονομάζεται *αριθμητικό σύ-*

αριθμητικό σύστημα θέσης

βάρος

στημα θέσης. Σε ένα τέτοιο σύστημα, κάθε αριθμός αναπαρίσταται από μια ακολουθία ψηφίων, όπου η θέση του κάθε ψηφίου έχει ένα αντίστοιχο *βάρος*. Η τιμή ενός αριθμού είναι το σταθμισμένο άθροισμα των ψηφίων, δηλαδή:

$$1734 = 1\cdot1000 + 7\cdot100 + 3\cdot10 + 4\cdot1$$

Το κάθε βάρος είναι μια δύναμη του 10 που αντιστοιχεί στη θέση του ψηφίου. Η υποδιαστολή επιτρέπει τη χρήση τόσο αρνητικών όσο και θετικών δυνάμεων του 10:

$$5185{,}68 = 5\cdot1000 + 1\cdot100 + 8\cdot10 + 5\cdot1 + 6\cdot0{,}1 + 8\cdot0{,}01$$

Γενικά, ένας αριθμός D της μορφής d_1d_0, $d_{-1}d_{-2}$ έχει την τιμή

$$D = d_1\cdot10^1 + d_0\cdot10^0 + d_{-1}\cdot10^{-1} + d_{-2}\cdot10^{-2}$$

βάση

Εδώ το 10 ονομάζεται "βάση" του αριθμητικού συστήματος. Σε ένα γενικό αριθμητικό σύστημα θέσης, βάση μπορεί να είναι κάθε ακέραιος $r \geq 2$, ενώ το ψηφίο στη θέση i έχει βάρος ίσο με r^i. Η γενική μορφή ενός αριθμού σε ένα τέτοιο σύστημα είναι

$$d_{p-1}\, d_{p-2} \cdots d_1\, d_0\, ,\, d_{-1}\, d_{-2} \cdots d_{-n}$$

υποδιαστολή

όπου υπάρχουν p ψηφία στα αριστερά και n ψηφία στα δεξιά του κόμματος, που ονομάζεται "*υποδιαστολή*". Αν δεν υπάρχει υποδιαστολή, τότε αυτή θεωρείται ότι βρίσκεται στα δεξιά του τελευταίου ψηφίου. Η τιμή του αριθμού είναι το άθροισμα του κάθε ψηφίου πολλαπλασιαζόμενου με την αντίστοιχη δύναμη της βάσης:

$$D = \sum_{i=-n}^{p-1} d_i\cdot r^i$$

Εκτός από πιθανά αρχικά και τελικά μηδενικά, η αναπαράσταση ενός αριθμού σε ένα αριθμητικό σύστημα θέσης είναι μοναδική (προφανώς, το 0185,6300 ισούται με το 185,63 κ.ο.κ.). Το τελευταίο ψηφίο αριστερά σε έναν τέτοιο αριθμό ονομάζεται *ψηφίο υψηλής τάξης* ή *πλέον σημαντικό ψηφίο*, ενώ το τελευταίο ψηφίο δεξιά είναι το *ψηφίο χαμηλής τάξης* ή *λιγότερο σημαντικό ψηφίο*.

ψηφίο υψηλής τάξης
πλέον σημαντικό ψηφίο
ψηφίο χαμηλής τάξης
λιγότερο σημαντικό ψηφίο

Όπως θα δούμε και στο Κεφάλαιο 3, τα ψηφιακά συστήματα έχουν σήματα τα οποία βρίσκονται σε μια από δύο δυνατές καταστάσεις — χαμηλό ή υψηλό, φορτισμένο ή εκφορτισμένο, κλειστό ή ανοικτό. Τα σήματα σε αυτά τα κυκλώματα χρησιμοποιούνται για την αναπαράσταση των *δυαδικών ψηφίων* (ή *bit*) τα οποία έχουν μία από τις δύο δυνατές τιμές, 0 ή 1. Κατά συνέπεια, η *δυαδική βάση* χρησιμοποιείται κατά κανόνα για την αναπαράσταση των αριθμών σε ένα ψηφιακό σύστημα. Η γενική μορφή ενός δυαδικού αριθμού είναι

δυαδικό ψηφίο
bit
δυαδική βάση

$$b_{p-1}\, b_{p-2} \cdots b_1\, b_0\, ,\, b_{-1}\, b_{-2} \cdots b_{-n}$$

και η τιμή του αριθμού είναι

$$B = \sum_{i=-n}^{p-1} b_i \cdot 2^i$$

Σε ένα δυαδικό αριθμό, η υποδιαστολή ονομάζεται *δυαδική υποδιαστολή*. Όταν εργαζόμαστε με δυαδικούς και άλλους μη δεκαδικούς αριθμούς, χρησιμοποιούμε ένα δείκτη για να προσδιορίζουμε τη βάση του κάθε αριθμού, εκτός αν η βάση είναι σαφής από τα συμφραζόμενα. Παρακάτω δίνονται παραδείγματα δυαδικών αριθμών και των δεκαδικών τους ισοδυνάμων.

δυαδική υποδιαστολή

$$10011_2 = 1\cdot16 + 0\cdot8 + 0\cdot4 + 1\cdot2 + 1\cdot1 = 19_{10}$$
$$100010_2 = 1\cdot32 + 0\cdot16 + 0\cdot8 + 0\cdot4 + 1\cdot2 + 0\cdot1 = 34_{10}$$
$$101.001_2 = 1\cdot4 + 0\cdot2 + 1\cdot1 + 0\cdot0.5 + 0\cdot0.25 + 1\cdot0.125 = 5.125_{10}$$

Το τελευταίο bit στα αριστερά ενός δυαδικού αριθμού ονομάζεται bit υψηλής τάξης ή πλέον σημαντικό bit (MSB Most Significant Bit), ενώ το τελευταίο bit δεξιά είναι το bit χαμηλής τάξης ή λιγότερο σημαντικό bit (LSB Least Significant Bit).

MSB
LSB

2.2 Οκταδικοί και δεκαεξαδικοί αριθμοί

Η βάση 10 είναι σημαντική επειδή τη χρησιμοποιούμε στις καθημερινές μας συναλλαγές, αλλά και η βάση 2 είναι σημαντική επειδή η επεξεργασία των δυαδικών αριθμών μπορεί να γίνεται απευθείας από τα ψηφιακά κυκλώματα. Οι αριθμοί σε άλλες βάσεις δεν προσφέρονται για απευθείας επεξεργασία, αλλά μπορεί να είναι σημαντικοί για τεκμηρίωση ή και για άλλους σκοπούς. Ειδικότερα, οι βάσεις 8 και 16 παρέχουν βολικές και σύντομες αναπαραστάσεις αριθμών με πολλά bit σε ένα ψηφιακό σύστημα.

Το οκταδικό αριθμητικό σύστημα χρησιμοποιεί ως βάση το 8, ενώ το δεκαεξαδικό αριθμητικό σύστημα χρησιμοποιεί ως βάση το 16. Ο Πίνακας 2-1 παρουσιάζει τους δυαδικούς ακέραιους από το 0 έως το 1111, καθώς και τα οκταδικά, δεκαδικά, και δεκαεξαδικά ισοδύναμά τους. Το οκταδικό σύστημα χρειάζεται 8 ψηφία, έτσι χρησιμοποιεί τα ψηφία από το 0 έως το 7 του δεκαδικού συστήματος. Το δεκαεξαδικό σύστημα χρειάζεται 16 ψηφία, έτσι παίρνει τα δεκαδικά ψηφία από το 0 έως το 9 και τα συμπληρώνει με τα λατινικά γράμματα από *A* έως *F*.

οκταδικό αριθμητικό σύστημα

δεκαεξαδικό αριθμητικό σύστημα

δεκαεξαδικά ψηφία A-F

Το οκταδικό και το δεκαεξαδικό σύστημα είναι χρήσιμα για την αναπαράσταση αριθμών με πολλά bit, επειδή οι βάσεις τους είναι δυνάμεις του 2. Εφόσον μια ακολουθία από τρία bit μπορεί να αναπαραστήσει 8 διαφορετικούς συνδυασμούς bit, έπεται ότι κάθε ακολουθία 3 bit μπορεί να αναπαρασταθεί με μοναδικό τρόπο από ένα οκταδικό ψηφίο, σύμφωνα με την τρίτη και τέταρτη στήλη του Πίνακα 2-1. Όμοια, μια ακολουθία 4 bit μπορεί να αναπαρασταθεί από ένα δεκαεξαδικό ψηφίο σύμφωνα με την πέμπτη και έκτη στήλη του ίδιου πίνακα.

μετατροπή δυαδικού σε οκταδικό

Είναι λοιπόν πολύ εύκολο να μετατρέψουμε ένα δυαδικό αριθμό σε οκταδικό. Αρχίζοντας από τη δυαδική υποδιαστολή και κινούμενοι προς τα αριστερά, απλώς χωρίζουμε τα bit σε ομάδες των τριών και αντικαθιστούμε κάθε ομάδα με το αντίστοιχο οκταδικό ψηφίο:

$$100011001110_2 = 100\ 011\ 001\ 110_2 = 4316_8$$
$$11101101110101001_2 = 011\ 101\ 101\ 110\ 101\ 001_2 = 355651_8$$

μετατροπή δυαδικού σε δεκαεξαδικό

Η διαδικασία μετατροπής δυαδικού αριθμού σε δεκαεξαδικό είναι παρόμοια, μόνο που εδώ χρησιμοποιούμε ομάδες των τεσσάρων bit:

$$100011001110_2 = 1000\ 1100\ 1110_2 = 8CE_{16}$$
$$11101101110101001_2 = 0001\ 1101\ 1011\ 1010\ 1001_2 = 1DBA9_{16}$$

Σε αυτά τα παραδείγματα, προσθέσαμε όσα μηδενικά θέλαμε στα αριστερά προκειμένου να γίνει ο συνολικός αριθμός των bit πολλαπλάσιος του 3 ή του 4.

Πίνακας 2-1 Δυαδικοί, δεκαδικοί, οκταδικοί, και δεκαεξαδικοί αριθμοί

Δυαδικός	*Δεκαδικός*	*Οκταδικός*	*Σειρά 3 bit*	*Δεκαεξαδικός*	*Σειρά 4 bit*
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	—	8	1000
1001	9	11	—	9	1001
1010	10	12	—	A	1010
1011	11	13	—	B	1011
1100	12	14	—	C	1100
1101	13	15	—	D	1101
1110	14	16	—	E	1110
1111	15	17	—	F	1111

Αν ένας δυαδικός αριθμός έχει ψηφία στα δεξιά της δυαδικής υποδιαστολής, μπορούμε να τα μετατρέψουμε σε οκταδικά ή δεκαεξαδικά αρχίζοντας από τη δυαδική υποδιαστολή και κινούμενοι προς τα δεξιά. Τόσο στα αριστερά όσο και στα δεξιά του αριθμού, μπορούν να προστεθούν όσα μηδενικά χρειάζονται για να έχουμε πολλαπλάσια των τριών και τεσσάρων bit, όπως στο παρακάτω παράδειγμα:

$$10{,}1011001011_2 = 010\ ,\ 101\ 100\ 101\ 100_2 = 2{,}5454_8$$
$$= 0010\ ,\ 1011\ 0010\ 1100_2 = 2{,}B2C_{16}$$

Η αντίστροφη μετατροπή, δηλαδή από οκταδικό ή δεκαεξαδικό αριθμό σε δυαδικό, είναι πολύ εύκολη. Απλώς αντικαθιστούμε κάθε οκταδικό ή δεκαεξαδικό ψηφίο με την αντίστοιχη ακολουθία 3 ή 4 bit, όπως φαίνεται παρακάτω:

μετατροπή οκταδικού ή δεκαεξαδικού σε δυαδικό

$$1357_8 = 001\ 011\ 101\ 111_2$$
$$2046{,}17_8 = 010\ 000\ 100\ 110\ ,\ 001\ 111_2$$
$$BEAD_{16} = 1011\ 1110\ 1010\ 1101_2$$
$$9F{,}46C_{16} = 1001\ 1111\ ,\ 0100\ 0110\ 1100_2$$

Το οκταδικό αριθμητικό σύστημα ήταν πολύ δημοφιλές πριν από 25 χρόνια, εξαιτίας ορισμένων μίνι υπολογιστών οι οποίοι είχαν στην πρόσοψη φωτεινές ενδείξεις και διακόπτες, διατεταγμένους σε ομάδες των τριών. Ωστόσο, το οκταδικό αριθμητικό σύστημα δε χρησιμοποιείται πολύ σήμερα, λόγω της υπεροχής των μηχανών που επεξεργάζονται οκτάδες από bit, δηλαδή *byte*. Στην οκταδική αναπαράσταση, είναι δύσκολο να εξαγάγει κανείς μεμονωμένα byte από αριθμούς πολλών byte. Για παράδειγμα, ποιες θα ήταν οι οκταδικές τιμές των τεσσάρων 8μπιτων byte που αποτελούν τον αριθμό των 32 bit με οκταδική αναπαράσταση 12345670123_8;

Στο δεκαεξαδικό σύστημα δύο ψηφία αναπαριστούν ένα byte των 8 bit, ενώ 2*n* ψηφία αναπαριστούν μια λέξη των *n* byte. Κάθε ζευγάρι ψηφίων αποτελεί ένα ακριβώς byte. Για παράδειγμα, ο δεκαεξαδικός αριθμός $5678ABCD_{16}$ των 32 bit αποτελείται από τέσσερα byte με τιμές 56_{16}, 78_{16}, AB_{16} και CD_{16}. Σε αυτό το γενικότερο πλαίσιο, ένα δεκαεξαδικό ψηφίο των 4 bit μερικές φορές λέγεται *μισό byte ή nibble*. Ένας αριθμός των 32 bit (4 byte) έχει οκτώ nibble. Οι δεκαεξαδικοί αριθμοί χρησιμοποιούνται συχνά προκειμένου να περιγράψουν το χώρο διευθύνσεων μνήμης ενός υπολογιστή. Για παράδειγμα, ένας υπολογιστής με διευθύνσεις των 16 bit θα μπορούσε να έχει μνήμη ανάγνωσης/εγγραφής εγκατεστημένη στις διευθύνσεις $0\text{-}EFFF_{16}$ και μνήμη μόνο για ανάγνωση στις διευθύνσεις $F000\text{-}FFFF_{16}$. Πολλές γλώσσες προγραμματισμού υπολογιστών χρησιμοποιούν το πρόθεμα “0x” για να υποδηλώνουν ένα δεκαεξαδικό αριθμό, π.χ. 0xBFC0000.

byte

μισό byte

πρόθεμα 0x

ΟΤΑΝ ΘΑ ΕΙΜΑΙ 64 ΧΡΟΝΩΝ

Καθώς γερνάτε, θα διαπιστώνετε ότι το δεκαεξαδικό σύστημα δεν είναι χρήσιμο μόνο για τους υπολογιστές. Όταν έκλεισα τα 40, είπα στους φίλους μου ότι μόλις έκλεισα τα 28_{16}. Το “$_{16}$”, φυσικά, το είπα ψιθυριστά. Στην ηλικία των 50, θα είμαι μόλις $32_{16.}$

Οι άνθρωποι ενθουσιάζονται με τα δεκαετή γενέθλια, όπως τα γενέθλια των 20, 30, 40, 50..., αλλά μάλλον μπορείτε να πείσετε τους φίλους σας ότι το δεκαδικό σύστημα δεν έχει τίποτα το ιδιαίτερο. Οι πιο σημαντικές αλλαγές στη ζωή συμβαίνουν κοντά στα γενέθλια των 2, 4, 8, 16, 32, και 64 ετών, τότε δηλαδή που προστίθενται σημαντικά δυαδικά ψηφία στην ηλικία σας. Γιατί νομίζετε πως οι Beatles τραγουδούσαν “When I am sixty-*four*”;

2.3 Γενικές μετατροπές στα αριθμητικά συστήματα θέσης

Γενικά, η μετατροπή από τη μια βάση στην άλλη δεν μπορεί να γίνει με απλή αντικατάσταση; είναι αναγκαίο να γίνουν αριθμητικές πράξεις. Σε αυτή την ενότητα θα δείξουμε πώς μετατρέπεται ένας αριθμός οποιασδήποτε βάσης σε αριθμό με βάση το 10, καθώς και το αντίστροφο, κάνοντας χρήση της αριθμητικής με βάση το 10.

μετατροπή αριθμού με βάση το r σε δεκαδικό

Στην Ενότητα 2.1 δείξαμε ότι η τιμή ενός αριθμού σε μια οποιαδήποτε βάση δίνεται από τον τύπο

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

όπου r είναι η βάση του αριθμού και υπάρχουν p ψηφία στα αριστερά της υποδιαστολής και n ψηφία στα δεξιά της. Έτσι, η τιμή του αριθμού μπορεί να υπολογιστεί αν κάθε ψηφίο του αριθμού μετατραπεί στο ισοδύναμο του με βάση το 10, ενώ η παραπάνω σχέση επεκτείνεται με τη χρήση αριθμητικής με βάση το 10.
Ακολουθούν μερικά παραδείγματα:

$$1CE8_{16} = 1\cdot16^3 + 12\cdot16^2 + 14\cdot16^1 + 8\cdot16^0 = 7400_{10}$$
$$F1A3_{16} = 15\cdot16^3 + 1\cdot16^2 + 10\cdot16^1 + 3\cdot16^0 = 61859_{10}$$
$$436{,}5_8 = 4\cdot8^2 + 3\cdot8^1 + 6\cdot8^0 + 5\cdot8^{-1} = 286{,}625_{10}$$
$$132{,}3_4 = 1\cdot4^2 + 3\cdot4^1 + 2\cdot4^0 + 3\cdot4^{-1} = 30{,}75_{10}$$

τύπος ένθετου αναπτύγματος

Ένας συντομότερος τρόπος για τη μετατροπή ολόκληρων αριθμών σε αριθμούς με βάση το 10 θα ήταν να ξαναγράψουμε τη σχέση του αναπτύγματος σε ένθετη μορφή:

$$D = ((\cdots((d_{p-1}) \cdot r + d_{p-2}) \cdot r + \cdots) \cdot r + d_1) \cdot r + d_0$$

Δηλαδή, ξεκινάμε με άθροισμα μηδέν. Αρχίζοντας από το πρώτο ψηφίο από τα αριστερά, πολλαπλασιάζουμε το άθροισμα με το r και το προσθέτουμε στο επόμενο ψηφίο, επαναλαμβάνοντας το ίδιο για όλα τα ψηφία. Για παράδειγμα, μπορούμε να γράψουμε:

$$F1AC_{16} = (((15) \cdot 16 + 1) \cdot 16 + 10) \cdot 16 + 12$$

μετατροπή δεκαδικού σε αριθμό με βάση το r

Ο τύπος αυτός χρησιμοποιείται σε επαναληπτικούς προγραμματιζόμενους αλγόριθμους μετατροπής (όπως στον Πίνακα 4-38 στην Ενότητα 4.7.4). Αποτελεί επίσης τη βάση μιας πολύ εύχρηστης μεθόδου μετατροπής ενός δεκαδικού αριθμού D σε αριθμό με βάση το r. Κοιτάξτε τι συμβαίνει αν διαιρέσουμε τον παραπάνω τύπο με το r. Επειδή το τμήμα του τύπου που βρίσκεται μέσα στην παρένθεση είναι διαιρετό με το r, το πηλίκο θα είναι

$$Q = (\cdots((d_{p-1}) \cdot r + d_{p-2}) \cdot r + \cdots) \cdot r + d_1$$

και το υπόλοιπο θα είναι d_0. Κατά συνέπεια, το d_0 μπορεί να υπολογιστεί ως υπόλοιπο της διαίρεσης του D με το r. Επιπλέον, το πηλίκο Q έχει την ίδια μορφή με τον αρχικό τύπο. Συνεπώς, διαδοχικές διαιρέσεις με το r οδηγούν σε διαδοχικά ψηφία του D από τα δεξιά προς τα αριστερά, μέχρι να σχηματιστούν όλα τα ψηφία του D. Ακολουθούν μερικά παραδείγματα:

179 ÷ 2 = 89 υπόλοιπο 1 (LSB)
÷2 = 44 υπόλοιπο 1
÷2 = 22 υπόλοιπο 0
÷2 = 11 υπόλοιπο 0
÷2 = 5 υπόλοιπο 1
÷2 = 2 υπόλοιπο 1
÷2 = 1 υπόλοιπο 0
÷2 = 0 υπόλοιπο 1 (MSB)

$179_{10} = 10110011_2$

467 ÷ 8 = 58 υπόλοιπο 3 (λιγότερο σημαντικό ψηφίο)
÷8 = 7 υπόλοιπο 2
÷ 8 = 0 υπόλοιπο 7 (πλέον σημαντικό ψηφίο)

$467_{10} = 723_8$

3417 ÷ 16 = 213 υπόλοιπο 9 (λιγότερο σημαντικό ψηφίο)
÷ 16 = 13 υπόλοιπο 5
÷ 16 = 0 υπόλοιπο 13 (πλέον σημαντικό ψηφίο)

$3417_{10} = D59_{16}$

Οι μέθοδοι για τη μετατροπή μεταξύ των πιο συνηθισμένων βάσεων συνοψίζονται στον Πίνακα 2-2.

2.4 Πρόσθεση και αφαίρεση μη δεκαδικών αριθμών

Για την πρόσθεση και αφαίρεση μη δεκαδικών αριθμών με το χέρι χρησιμοποιούνται οι ίδιες τεχνικές που μάθαμε στο δημοτικό σχολείο για τους δεκαδικούς αριθμούς. Η μόνη διαφορά είναι ότι οι πίνακες πρόσθεσης και αφαίρεσης είναι διαφορετικοί.

Πρόσθεση δυαδικών

Ο Πίνακας 2-3 είναι ο πίνακας πρόσθεσης και αφαίρεσης για τα δυαδικά ψηφία. Για να αθροίσουμε δύο δυαδικούς αριθμούς X και Y, προσθέτουμε μαζί τα λιγότερο σημαντικά bit με αρχικό κρατούμενο (c_{in}) το μηδέν (0), παράγοντας bit "κρατουμένων" (c_{out}) και "αθροίσματος" (s) σύμφωνα με τον πίνακα. Συνεχίζουμε τη διαδικασία αυτή με φορά από τα δεξιά προς τα αριστερά, προσθέτοντας το αποτέλεσμα της κάθε στήλης στο άθροισμα της επόμενης.

Στην Εικόνα 2-1 δίνονται δύο παραδείγματα πρόσθεσης δεκαδικών και τα αντίστοιχα παραδείγματα πρόσθεσης δυαδικών, όπου το βέλος δείχνει το κρατούμενο 1. Τα ίδια παραδείγματα επαναλαμβάνονται και παρακάτω μαζί με δύο επιπλέον παραδείγματα των οποίων τα κρατούμενα εμφανίζονται ως ακολουθία bit C:

Πίνακας 2-2 Μέθοδοι μετατροπής για τις πιο συνηθισμένες βάσεις

Μετατροπή	*Μέθοδος*	*Παράδειγμα*
Δυαδικός σε		
Οκταδικό	Αντικατάσταση	$10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$
Δεκαεξαδικό	Αντικατάσταση	$10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$
Δεκαδικό	Άθροιση	$10111011001_2 = 1\cdot 1024 + 0\cdot 512 + 1\cdot 256 + 1\cdot 128 + 1\cdot 64 + 0\cdot 32 + 1\cdot 16 + 1\cdot 8 + 0\cdot 4 + 0\cdot 2 + 1\cdot 1 = 1497_{10}$
Οκταδικός σε		
Δυαδικό	Αντικατάσταση	$1234_8 = 001\ 010\ 011\ 100_2$
Δεκαεξαδικό	Αντικατάσταση	$1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$
Δεκαδικό	Άθροιση	$1234_8 = 1\cdot 512 + 2\cdot 64 + 3\cdot 8 + 4\cdot 1 = 668_{10}$
Δεκαεξαδικός σε		
Δυαδικό	Αντικατάσταση	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$
Οκταδικό	Αντικατάσταση	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$
Δεκαδικό	Άθροιση	$C0DE_{16} = 12\cdot 4096 + 0\cdot 256 + 13\cdot 16 + 14\cdot 1 = 49374_{10}$
Δεκαδικός σε		
Δυαδικό	Διαίρεση	$108_{10} \div 2 = 54$ υπόλοιπο 0 (LSB) $\div 2 = 27$ υπόλοιπο 0 $\div 2 = 13$ υπόλοιπο 1 $\div 2 = 6$ υπόλοιπο 1 $\div 2 = 3$ υπόλοιπο 0 $\div 2 = 1$ υπόλοιπο 1 $\div 2 = 0$ υπόλοιπο 1 (MSB) $108_{10} = 1101100_2$
Οκταδικό	Διαίρεση	$108_{10} \div 8 = 13$ υπόλοιπο 4 (λιγότερο σημαντικό ψηφίο) $\div 8 = 1$ υπόλοιπο 5 $\div 8 = 0$ υπόλοιπο 1 (πλέον σημαντικό ψηφίο) $108_{10} = 154_8$
Δεκαεξαδικό	Διαίρεση	$108_{10} \div 16 = 6$ υπόλοιπο 12 (λιγότερο σημαντικό ψηφίο $\div 16 = 0$ υπόλοιπο 6 (πλέον σημαντικό ψηφίο) $108_{10} = 6C_{16}$

Πίνακας 2-3 Πίνακας πρόσθεσης και αφαίρεσης δυαδικών

c_{in} ***ή*** b_{in}	x	y	c_{out}	s	b_{out}	d
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

								1	1	1	1									1		1	1	
X	190						1	0	1	1	1	1	1	0	X	173			1	0	1	0	1	1	0
Y	+ 141				+	1	0	0	0	1	1	0	1	Y	+ 44		+	0	0	1	0	1	1	0
$X+Y$	331				1	0	1	0	0	1	0	1	1	$X+Y$	217			1	1	0	1	1	0	0

Εικόνα 2-1 Παραδείγματα πρόσθεσης δεκαδικών και των αντίστοιχων δυαδικών

C		*101111000*	*C*		*001011000*
X	190	10111110	X	173	10101101
Y	+141	+ 10001101	Y	+ 44	+ 00101100
X+Y	331	101001011	*X + Y*	217	11011001

C		*011111110*	*C*		*000000000*
X	127	01111111	X	170	10101010
Y	+ 63	+ 00111111	Y	+ 85	+ 01010101
X+Y	190	10111110	*X + Y*	255	11111111

Η αφαίρεση δυαδικών αριθμών πραγματοποιείται με παρόμοιο τρόπο. Εδώ χρησιμοποιούμε "δανεικά" (b_{in} και b_{out}) αντί για κρατούμενα σε κάθε βήμα και δημιουργούμε ένα bit διαφοράς *d*. Στην Εικόνα 2-2 παρουσιάζονται δύο παραδείγματα αφαίρεσης δεκαδικών καθώς και τα αντίστοιχα παραδείγματα αφαίρεσης δυαδικών. Όπως και στην αφαίρεση δεκαδικών, έτσι και εδώ οι δυαδικές τιμές του μειωτέου στις στήλες μεταβάλλονται όταν προκύπτει ένα δανεικό, όπως φαίνεται και από τα αντίστοιχα βέλη και bit. Τα παραδείγματα της εικόνας αυτής επαναλαμβάνονται παρακάτω μαζί με άλλα δύο παραδείγματα, όπου τα δανεικά εμφανίζονται ως ακολουθία bit *B*:

Αφαίρεση δυαδικών

μειωτέος
αφαιρετέος

B		*001111100*	*B*		*011011010*
X	229	11100101	X	210	11010010
Y	– 46	– 00101110	Y	– 109	– 01101101
X–Y	183	10110111	X – Y	101	01100101

B		*010101010*	*B*		*000000000*
X	170	10101010	X	221	11011101
Y	– 85	– 01010101	Y	– 76	– 01001100
X–Y	85	01010101	X – Y	145	10010001

Μια συνήθης χρήση της αφαίρεσης στους υπολογιστές είναι η σύγκριση δύο αριθμών μεταξύ τους. Αν, για παράδειγμα, η πράξη *X-Y* δίνει ως αποτέλεσμα ένα δανεικό πέρα από τη θέση του πλέον σημαντικού bit, τότε το *X* είναι μικρότερο του *Y*, διαφορετικά το *X* είναι μεγαλύτερο ή ίσο με το *Y*. Στην Ενότητα 5.10 θα διερευνήσουμε περισσότερο τη σχέση μεταξύ κρατούμενων και δανεικών στους αθροιστές και στους μειωτές, αντίστοιχα.

σύγκριση αριθμών

Εικόνα 2-2
Παραδείγματα αφαιρέσεων δεκαδικών και των αντιστοίχων δυαδικών αριθμών

Απαιτείται δανεικό 1, και καταλήγουμε στη νέα αφαίρεση 10 - 1 = 1

Μετά από το πρώτο δανεικό, η νέα αφαίρεση για τη στήλη αυτή είναι 0 - 1 και έτσι απαιτείται πάλι δανεικό.

Το δανεικό μετακυλίεται σε τρεις στήλες για να φτάσει σε ένα ψηφίο με δυνατότητα δανεισμού 1, δηλαδή: 100 = 011 (τροποποιημένα bit) + 1 (δανεικό)

								0	10		1		1	10	10																	
μειωτέος	X	229					1	1	1	0	0		1	0	1		X	210					1	1	0	1	0	1	0	1	0	
αφαιρετέος	Y	− 46				−	0	0	1	0	1		1	1	0		Y	− 109				−	0	1	1	0	1	1	0	1		
διαφορά	$X - Y$	183					1	0	1	1	0		1	1	1		$X - Y$	101					0	1	1	0	0	1	0	1		

Είναι δυνατόν να δημιουργήσουμε πίνακες πρόσθεσης και αφαίρεσης για οκταδικά και δεκαεξαδικά ψηφία ή για οποιαδήποτε άλλη βάση θέλουμε. Όμως, ελάχιστοι μηχανικοί υπολογιστών ασχολούνται με την απομνημόνευση τέτοιων πινάκων. Αν όμως χρειαστεί να κάνετε πράξεις με μη δεκαδικούς αριθμούς, τότε είναι πιο εύκολο να τους μετατρέψετε σε δεκαδικούς, να κάνετε τις πράξεις, και να τους μετατρέψετε πάλι στην αρχική τους μορφή. Από την άλλη, αν πράγματι πρέπει να κάνετε συχνά υπολογισμούς με δυαδικούς, οκταδικούς, και δεκαεξαδικούς, τότε ίσως είναι καλύτερα να προμηθευτείτε μια αριθμομηχανή δεκαεξαδικών της Texas Instruments ή της Casio.

Αν η μπαταρία της αριθμομηχανής εξαντληθεί, μπορούν να χρησιμοποιηθούν κάποιοι μνημονικοί κανόνες για να διευκολύνουν τις πράξεις με μη δεκαδικούς αριθμούς. Γενικότερα, κάθε πρόσθεση (ή αφαίρεση) κατά στήλες γίνεται με μετατροπή των ψηφίων τής κάθε στήλης σε δεκαδικούς, κατόπιν με πρόσθεση των δεκαδικών, και στη συνέχεια με μετατροπή του αποτελέσματος κάθε πρόσθεσης και των κρατουμένων της ξανά στη μη δεκαδική βάση. (Κρατούμενο έχουμε κάθε φορά που το άθροισμα της στήλης είναι ίσο ή μεγαλύτερο από τη βάση.) Έτσι λοιπόν η πρόσθεση γίνεται στο δεκαδικό σύστημα, του οποίου τον πίνακα πρόσθεσης γνωρίζουμε καλά. Το μόνο καινούργιο που χρειάζεται να μάθουμε είναι η μετατροπή από δεκαδικά σε μη δεκαδικά ψηφία και το αντίστροφο. Η σειρά των βημάτων προκειμένου να προσθέσουμε νοητά δύο δεκαεξαδικούς αριθμούς είναι η ακόλουθη:

πρόσθεση δεκαεξαδικών

C	*1 1 0 0*	*1*	*1*	*0*	*0*
X	$19B9_{16}$	1	9	11	9
Y	+ $C7E6_{16}$	+12	7	14	6
$X + Y$	$E19F_{16}$	14	17	25	15
		14	16 + 1	16 + 9	15
		E	1	9	F

2.5 Αναπαράσταση αρνητικών αριθμών

Μέχρι τώρα ασχοληθήκαμε μόνο με θετικούς αριθμούς, αλλά υπάρχουν αρκετοί τρόποι για να αναπαραστήσουμε τους αρνητικούς αριθμούς. Στις καθημερινές μας συναλλαγές χρησιμοποιούμε το σύστημα προσημασμένου μεγέθους που περιγράφεται παρακάτω. Ωστόσο, οι περισσότεροι υπολογιστές χρησιμοποιούν ένα από τα αριθμητικά συστήματα συμπληρώματος που θα εξετάσουμε αργότερα.

2.5.1 Αναπαράσταση προσημασμένου μεγέθους

σύστημα προσημασμένου μεγέθους

Στο *σύστημα προσημασμένου μεγέθους*, κάθε αριθμός αποτελείται από το "μέγεθος" (ή, αλλιώς, "μέτρο") και από ένα σύμβολο που δείχνει αν το μέγεθος είναι θετικό ή αρνητικό. Έτσι, ερμηνεύουμε τους δεκαδικούς αριθμούς +98, -57, +123,5 και -13 με το συνήθη τρόπο, και δεχόμαστε επίσης ότι το πρόσημο είναι "+" αν δεν υπάρχει πρόσημο. Υπάρχουν δύο δυνατές αναπαραστάσεις για το μηδέν, το "+0" και το "-0", που έχουν όμως την ίδια τιμή.

bit προσήμου

Το σύστημα προσημασμένου μεγέθους βρίσκει εφαρμογή στους δυαδικούς αριθμούς, με τη χρήση μίας επιπλέον θέσης bit για την αναπαράσταση του προσήμου (το λεγόμενο "bit προσήμου"). Κατά παράδοση, το πλέον σημαντικό bit (MSB) μιας σειράς από bit χρησιμοποιείται ως bit προσήμου (0=συν, 1=μείον), ενώ τα χαμηλότερης τάξης ψηφία αναπαριστούν το μέγεθος. Κατά συνέπεια, μπορούμε να γράψουμε μερικούς 8μπιτους ακεραίους προσημασμένου μεγέθους και τους αντίστοιχους δεκαδικούς:

$01010101_2 = +85_{10}$ $\qquad$ $11010101_2 = -85_{10}$

$01111111_2 = +127_{10}$ $\qquad$ $11111111_2 = -127_{10}$

$00000000_2 = +0_{10}$ $\qquad$ $10000000_2 = -0_{10}$

Το σύστημα προσημασμένου μεγέθους έχει ίσο αριθμό θετικών και αρνητικών ακεραίων. Ένας ακέραιος προσημασμένου μεγέθους με *n bit* βρίσκεται μέσα στο διάστημα από $-(2^{n-1} - 1)$ έως $+(2^{n-1} - 1)$, και υπάρχουν δύο δυνατές αναπαραστάσεις για το μηδέν.

αθροιστής προσημασμένου μεγέθους

Ας υποθέσουμε τώρα ότι θέλουμε να φτιάξουμε ένα ψηφιακό λογικό κύκλωμα που να προσθέτει αριθμούς προσημασμένου μεγέθους. Το κύκλωμα πρέπει να εξετάζει τα πρόσημα των προσθετέων για να προσδιορίζει πώς θα ενεργήσει πάνω στα μεγέθη τους. Αν τα πρόσημα είναι ίδια, τότε πρέπει να προσθέσει τα μεγέθη και να δώσει στο αποτέλεσμα το ίδιο πρόσημο. Αν τα πρόσημα είναι διαφορετικά, τότε το κύκλωμα θα πρέπει να συγκρίνει τα μεγέθη, να αφαιρέσει το μικρότερο αριθμό από το μεγαλύτερο, και να δώσει στο αποτέλεσμα το πρόσημο του μεγαλύτερου. Όλες αυτές όμως οι υποθέσεις, οι προσθέσεις, οι αφαιρέσεις, και οι συγκρίσεις μεταφράζονται σε ένα πολύπλοκο λογικό κύκλωμα. Οι αθροιστές για τα αριθμητικά συστήματα συμπληρώματος είναι πολύ πιο απλοί, όπως θα δούμε πιο κάτω. Ίσως το πιο ενδιαφέρον

χαρακτηριστικό των αριθμητικών συστημάτων προσημασμένου μεγέθους είναι ότι, από τη στιγμή που γνωρίζουμε πώς να κατασκευάζουμε έναν αθροιστή προσημασμένου μεγέθους, η κατασκευή ενός αφαιρέτη προσημασμένου μεγέθους είναι εύκολη υπόθεση — αρκεί απλώς να αλλάξουμε το πρόσημο του αφαιρετέου και να τον στείλουμε μαζί με το μειωτέο σε έναν αθροιστή.

αφαιρέτης προσημασμένου μεγέθους

2.5.2 Αριθμητικά συστήματα συμπληρώματος

αριθμητικά συστήματα συμπληρώματος

Ενώ τα αριθμητικά συστήματα προσημασμένου μεγέθους καθιστούν έναν αριθμό αρνητικό αλλάζοντας το πρόσημο του, τα *αριθμητικά συστήματα συμπληρώματος* καθιστούν έναν αριθμό αρνητικό παίρνοντας το συμπλήρωμά του όπως αυτό ορίζεται από το σύστημα. Ο υπολογισμός του συμπληρώματος είναι πιο δύσκολος από την αλλαγή του προσήμου. Έτσι όμως, σε ένα αριθμητικό σύστημα συμπληρώματος, δύο αριθμοί είναι δυνατόν να προστεθούν ή να αφαιρεθούν απευθείας, χωρίς έλεγχο μεγέθους ή προσήμου όπως θα χρειαζόταν σε ένα αριθμητικό σύστημα προσημασμένου μεγέθους. Θα περιγράψουμε δύο αριθμητικά συστήματα συμπληρώματος: το "συμπλήρωμα ως προς τη βάση" και το "συμπλήρωμα ως προς τη βάση πλην ένα ".

Σε οποιοδήποτε αριθμητικό σύστημα συμπληρώματος, έχουμε να κάνουμε με ένα σταθερό αριθμό ψηφίων, έστω n. (Μπορούμε όμως να αυξήσουμε τον αριθμό των ψηφίων με "επέκταση προσήμου", όπως στην Άσκηση 2.23, καθώς και να τον μειώσουμε αποκόπτοντας ψηφία υψηλής τάξης, όπως στην Άσκηση 2.24.) Επιπλέον, ας υποθέσουμε ότι η βάση είναι r και ότι οι αριθμοί έχουν τη μορφή

$$D = d_{n-1}d_{n-2} \cdots d_1 d_0.$$

Η υποδιαστολή είναι στα δεξιά και έτσι ο αριθμός είναι ακέραιος. Αν μια πράξη δώσει αποτέλεσμα το οποίο χρειάζεται περισσότερα από n ψηφία, αγνοούμε τα επιπλέον ψηφία υψηλής τάξης. Αν πάρουμε δύο φορές το συμπλήρωμα ενός αριθμού D, το αποτέλεσμα είναι πάλι D.

2.5.3 Αναπαράσταση συμπληρώματος ως προς τη βάση

σύστημα συμπληρώματος ως προς τη βάση

συμπλήρωμα ως προς 10

Σε ένα *σύστημα συμπληρώματος ως προς τη βάση*, το συμπλήρωμα ενός n-ψήφιου αριθμού λαμβάνεται με αφαίρεσή του από τον r^n. Στο δεκαδικό αριθμητικό σύστημα, το συμπλήρωμα ως προς τη βάση ονομάζεται *συμπλήρωμα ως προς 10*. Στον Πίνακα 2-4 δίνονται μερικά παραδείγματα με τετραψήφιους δεκαδικούς αριθμούς (και αφαίρεση από το 10.000).

Εξ ορισμού, το συμπλήρωμα ενός n-ψήφιου αριθμού D ως προς τη βάση λαμβάνεται με αφαίρεση του αριθμού D από τον r^n. Αν ο D είναι μεταξύ 1 και r^n-1, τότε η αφαίρεση δίνει ως αποτέλεσμα έναν άλλο αριθμό μεταξύ του 1 και του r^n-1. Στην περίπτωση τώρα που ο D είναι το 0, το αποτέλεσμα της αφαίρεσης θα είναι r^n, το οποίο έχει τη μορφή 100...00 με n+1 ψηφία συνολικά. Αγνοούμε τα υψηλής τάξης ψηφία και παίρνου-

Αριθμός	*Συμπλήρωμα ως προς 10*	*Συμπλήρωμα ως προς 9*
1849	8151	8150
2067	7933	7932
100	9900	9899
7	9993	9992
8151	1849	1848
0	10000 (= 0)	9999

Πίνακας 2-4 Παραδείγματα συμπληρωμάτων ως προς 10 και ως προς 9

με ως αποτέλεσμα το 0. Κατά συνέπεια, υπάρχει μία μόνο αναπαράσταση του μηδενός σε ένα αριθμητικό σύστημα συμπληρώματος ως προς τη βάση.

υπολογισμός συμπληρώματος ως προς τη βάση

Φαίνεται από τον ορισμό ότι για τον υπολογισμό του συμπληρώματος του D χρειάζεται μια πράξη αφαίρεσης. Όμως μπορούμε να αποφύγουμε αυτή την αφαίρεση ξαναγράφοντας το r^n ως $(r^n-1)+1$ και το r^n-D ως $((r^n-1)-D)+1$. Ο αριθμός r^n-1 έχει τη μορφή $mm...mm$, όπου $m=r-1$ και με n το πλήθος αριθμούς m. Για παράδειγμα το 10.000 ισούται με 9.999+1. Αν ορίσουμε το συμπλήρωμα ενός ψηφίου d ως $r-1-d$, τότε, από το συμπλήρωμα των ψηφίων του D προκύπτει ο αριθμός $(r^n-1)-D$. Συνεπώς, το

	Συμπλήρωμα			
Ψηφίο	*Δυαδικό*	*Οκταδικό*	*Δεκαδικό*	*Δεκαεξαδικό*
0	1	7	9	F
1	0	6	8	E
2	-	5	7	D
3	-	4	6	C
4	-	3	5	B
5	-	2	4	A
6	-	1	3	9
7	-	0	2	8
8	-	-	1	7
9	-	-	0	6
A	-	-	-	5
B	-	-	-	4
C	-	-	-	3
D	-	-	-	2
E	-	-	-	1
F	-	-	-	0

Πίνακας 2-5 Συμπληρώματα ψηφίων

συμπλήρωμα ως προς τη βάση ενός αριθμού D λαμβάνεται με τον υπολογισμό του συμπληρώματος κάθε ψηφίου που απαρτίζει τον D και με πρόσθεση του 1. Για παράδειγμα, το συμπλήρωμα του 1849 ως προς 10 είναι 8150+1 δηλαδή 8151. Θα πρέπει να επαληθεύσετε ότι το τέχνασμα αυτό είναι εξίσου αποδοτικό και για τα προηγούμενα παραδείγματα συμπληρωμάτων ως προς 10. Στον Πίνακα 2-5 δίνονται τα ψηφία των συμπληρωμάτων για δυαδικούς, οκταδικούς, δεκαδικούς, και δεκαεξαδικούς αριθμούς.

2.5.4 Αναπαράσταση συμπληρώματος ως προς δύο

συμπλήρωμα ως προς δύο

Στους δυαδικούς αριθμούς, το συμπλήρωμα ως προς τη βάση ονομάζεται *συμπλήρωμα ως προς δύο*. Σε αυτό το σύστημα, το πλέον σημαντικό ψηφίο (MSB) ενός αριθμού χρησιμεύει ως bit προσήμου. Ένας αριθμός είναι αρνητικός αν και μόνο αν το MSB του είναι 1. Ο ισοδύναμος δεκαδικός αριθμός ενός δυαδικού αριθμού συμπληρώματος ως προς δύο υπολογίζεται με τον ίδιο τρόπο που είδαμε και για έναν απρόσημο αριθμό, με τη διαφορά ότι το βάρος του MSB είναι -2^{n-1} αντί για $+2^{n-1}$. Το πεδίο τιμών των αριθμών που μπορούν να αναπαρασταθούν είναι από $-(2^{n-1})$ ως $+(2^{n-1}-1)$. Μερικά παραδείγματα αριθμών των 8 bit παρουσιάζονται παρακάτω:

βάρος του πλέον σημαντικού ψηφίου

$17_{10} = 00010001_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 11101110 \\ +1 \\ \hline 11101111_2 \end{array} = -17_{10}$$

$-99_{10} = 10011101_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 01100010 \\ +1 \\ \hline 01100011_2 \end{array} = 99_{10}$$

$119_{10} = 01110111_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 10001000_2 \\ +1 \\ \hline 10001001_2 \end{array} = -119_{10}$$

$-127_{10} = 10000001_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 01111110_2 \\ +1 \\ \hline 01111111_2 \end{array} = 127_{10}$$

$0_{10} = 00000000_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 11111111 \\ +1 \\ \hline 1\ 00000000_2 \end{array} = 0_{10}$$

$-128_{10} = 10000000_2$ ⇓ τα bit συμπληρώματος

$$\begin{array}{r} 01111111 \\ +1 \\ \hline 10000000_2 \end{array} = -128_{10}$$

Όπως φαίνεται στο παραπάνω παράδειγμα, σε μια περίπτωση εμφανίζεται κρατούμενο πέρα από τη θέση του πλέον σημαντικού bit. Όπως γίνεται σε όλες τις πράξεις συμπληρώματος ως προς δύο, αυτό το bit αγνοείται και χρησιμοποιούνται μόνο τα n bit χαμηλής τάξης του αποτελέσματος.

Στο αριθμητικό σύστημα συμπληρώματος ως προς δύο, το μηδέν θεωρείται θετικός αριθμός επειδή το bit του προσήμου του είναι 0. Εφόσον λοιπόν στο σύστημα συμπληρώματος ως προς δύο υπάρχει μόνο μία ανα-

παράσταση του μηδενός καταλήγουμε με έναν επιπλέον αρνητικό αριθμό δηλαδή τον -2^{n-1}, ο οποίος δεν έχει αντίστοιχο θετικό.

επιπλέον αρνητικός αριθμός

Μπορούμε να μετατρέψουμε έναν αριθμό X συμπληρώματος ως προς δύο με n bit σε έναν άλλο με m bit, αλλά με την απαιτούμενη προσοχή. Αν $m > n$, πρέπει να προσθέσουμε m-n αντίγραφα του bit προσήμου του X στα αριστερά του X (δείτε την Άσκηση 2.23). Με άλλα λόγια, σε ένα θετικό αριθμό προσθέτουμε μια σειρά από 0, ενώ σε έναν αρνητικό αριθμό προσθέτουμε μια σειρά από 1. Αυτή η διαδικασία ονομάζεται *επέκταση προσήμου*. Αν τώρα $m < n$, απορρίπτουμε τα n-m bit στα αριστερά του X. Όμως το αποτέλεσμα είναι έγκυρο μόνο όταν όλα τα bit που απορρίψαμε είναι όμοια με το bit προσήμου του αποτελέσματος (δείτε την Άσκηση 2.24).

επέκταση προσήμου

Οι περισσότεροι υπολογιστές και άλλα ψηφιακά συστήματα χρησιμοποιούν το σύστημα συμπληρώματος ως προς δύο για την αναπαράσταση των αρνητικών αριθμών. Ωστόσο, χάριν πληρότητας, θα περιγράψουμε και τα αριθμητικά συστήματα "συμπληρώματος ως προς τη βάση πλην ένα" και "συμπληρώματος ως προς ένα".

*2.5.5 Αναπαράσταση συμπληρώματος ως προς τη βάση πλην ένα

Σε ένα *σύστημα συμπληρώματος ως προς τη βάση πλην ένα*, το συμπλήρωμα ενός n-ψήφιου αριθμού D λαμβάνεται με αφαίρεση του αριθμού από το r^n-1. Αυτό μπορεί να το πετύχουμε βρίσκοντας το συμπλήρωμα κάθε ψηφίου του D ξεχωριστά, χωρίς πρόσθεση του 1 όπως σε ένα αριθμητικό σύστημα συμπληρώματος ως προς τη βάση. Στο δεκαδικό σύστημα, αυτό ονομάζεται *συμπλήρωμα ως προς 9*. Μερικά παραδείγματα παρουσιάζονται στην τελευταία στήλη του Πίνακα 2-4, στην Ενότητα 2.5.3.

συμπλήρωμα ως προς τη βάση πλην ένα

συμπλήρωμα ως προς 9

*2.5.6 Αναπαράσταση συμπληρώματος ως προς ένα

Το αριθμητικό σύστημα συμπληρώματος ως προς τη βάση πλην ένα για τους δυαδικούς αριθμούς ονομάζεται *συμπλήρωμα ως προς ένα*. Όπως και στο συμπλήρωμα ως προς δύο, το πλέον σημαντικό bit είναι το πρόσημο, 0 αν είναι θετικός αριθμός και 1 αν είναι αρνητικός. Κατά συνέπεια, υπάρχουν δύο αναπαραστάσεις του μηδενός, το θετικό μηδέν (00...00) και το αρνητικό μηδέν (11...11). Οι αναπαραστάσεις των θετικών αριθμών είναι οι ίδιες για τα συμπληρώματα ως προς ένα και για τα συμπληρώματα ως προς δύο. Αντίθετα, οι αναπαραστάσεις των αρνητικών αριθμών διαφέρουν κατά 1. Στο πλέον σημαντικό bit, δίνεται βάρος ίσο με $-(2^{n-1}-1)$ αντί για -2^{n-1} όταν υπολογίζουμε το δεκαδικό ισοδύναμο ενός αριθμού συμπληρώματος ως προς ένα. Το διάστημα των αριθμών με αυτή την αναπαράσταση είναι από $-(2^{n-1}-1)$ έως $+(2^{n-1}-1)$. Ακολουθούν μερικοί αριθμοί των 8 bit και τα συμπληρώματα τους ως προς ένα:

συμπλήρωμα ως προς ένα

*Σε ολόκληρο το βιβλίο, οι προαιρετικές ενότητες σημειώνονται με αστερίσκο.

$$17_{10} = 00010001_2 \Downarrow 11101110_2 = -17_{10} \qquad -99_{10} = 10011100_2 \Downarrow 01100011_2 = 99_{10}$$

$$119_{10} = 01110111_2 \Downarrow 10001000_2 = -119_{10} \qquad -127_{10} = 10000000_2 \Downarrow 01111111_2 = 127_{10}$$

$$0_{10} = 00000000_2 \text{ (θετικό μηδέν)} \Downarrow 011111111_2 = 0_{10} \text{ (αρνητικό μηδέν)}$$

Τα κύρια πλεονεκτήματα του αριθμητικού συστήματος συμπληρώματος ως προς ένα είναι η συμμετρία του και η ευκολία με την οποία παίρνουμε το συμπλήρωμα. Ωστόσο, η σχεδίαση ενός αθροιστή για αριθμούς συμπληρώματος ως προς ένα είναι κάπως πιο περίπλοκη απ' ό,τι για έναν αθροιστή αριθμών συμπληρώματος ως προς δύο (δείτε την Άσκηση 7.72). Επίσης, τα κυκλώματα ανίχνευσης μηδενός σε ένα σύστημα συμπληρώματος ως προς ένα είτε πρέπει πάντα να ελέγχουν και τις δύο αναπαραστάσεις του μηδενός, είτε πρέπει πάντα να μετατρέπουν το 11...11 σε 00...00.

*2.5.7 Αναπαραστάσεις υπέρβασης

αναπαράσταση υπέρβασης κατά B

πόλωση

αριθμητικό σύστημα υπέρβασης κατά 2^{m-1}

Είναι αλήθεια ότι ο αριθμός των διαφορετικών συστημάτων για την αναπαράσταση των αρνητικών αριθμών είναι υπερβολικός, όμως απέμεινε μόνο ένα ακόμη για να καλύψουμε. Στην *αναπαράσταση υπέρβασης κατά B* μια ακολουθία *m* bit, της οποίας η απρόσημη ακέραια τιμή είναι M ($0 \le M < 2^m$), αναπαριστά τον προσημασμένο ακέραιο *M-B,* όπου το B ονομάζεται και *πόλωση* του αριθμητικού συστήματος.

Για παράδειγμα, σε ένα *σύστημα υπέρβασης κατά* 2^{m-1}, κάθε αριθμός *X* μέσα στο διάστημα από -2^{m-1} έως $+2^{m-1}-1$ αναπαρίσταται από τη δυαδική αναπαράσταση *m*-bit $X + 2^{m-1}$ (η οποία είναι πάντα μη αρνητική και μικρότερη του 2^m). Το πεδίο τιμών αυτής της αναπαράστασης είναι ακριβώς το ίδιο όπως και των αριθμών συμπληρώματος ως προς δύο με *m* bit. Μάλιστα, οι αναπαραστάσεις οποιουδήποτε αριθμού και στα δύο συστήματα είναι οι ίδιες εκτός από τα bit προσήμου τα οποία είναι πάντα αντίθετα. (Σημειώστε ότι αυτό ισχύει μόνο όταν η πόλωση είναι 2^{m-1}.)

Η πιο συνήθης χρήση των αναπαραστάσεων υπέρβασης είναι στα αριθμητικά συστήματα κινητής υποδιαστολής (δείτε τις Παραπομπές).

2.6 Πρόσθεση και αφαίρεση αριθμών συμπληρώματος ως προς δύο

2.6.1 Κανόνες πρόσθεσης

Ο Πίνακας 2-6, που περιέχει τους δεκαδικούς αριθμούς και τους ισοδύναμους τους σε διαφορετικά αριθμητικά συστήματα, φανερώνει το λόγο για τον οποίο το συμπλήρωμα ως προς δύο προτιμάται στις αριθμητικές πράξεις. Αν αρχίσουμε με το 1000_2 (-8_{10}) και προσθέτουμε κάθε φορά ένα μέχρι τον 0111_2 ($+7_{10}$), βλέπουμε ότι κάθε αριθμός συμπληρώματος ως προς δύο παράγεται με την πρόσθεση του 1 στον προηγούμενο, αγνοώντας τυχόν κρατούμενα πέραν της τέταρτης θέσης bit. Δεν ισχύει το ίδιο για τους αριθμούς προσημασμένου μεγέθους και τους αριθμούς συμπληρώματος ως προς ένα. Επειδή η κοινή πρόσθεση είναι απλώς μια επέκταση της απαρίθμησης, μπορούμε να προσθέτουμε τους αριθμούς συμπληρώματος ως προς δύο χρησιμοποιώντας την πρόσθεση δυαδικών, αγνοώντας κάθε κρατούμενο πέραν του πλέον σημαντικού ψηφίου. Το αποτέλεσμα θα είναι πάντα το σωστό άθροισμα, υπό τον όρο ότι δεν υπερβαίνουμε το πεδίο τιμών του αριθμητικού συστήματος. Μερικά παραδείγματα πρόσθεσης 4μπιτων δεκαδικών και των αντίστοιχων συμπληρωμάτων ως προς δύο επαληθεύουν όσα είπαμε.

πρόσθεση αριθμών συμπληρώματος ως προς δύο

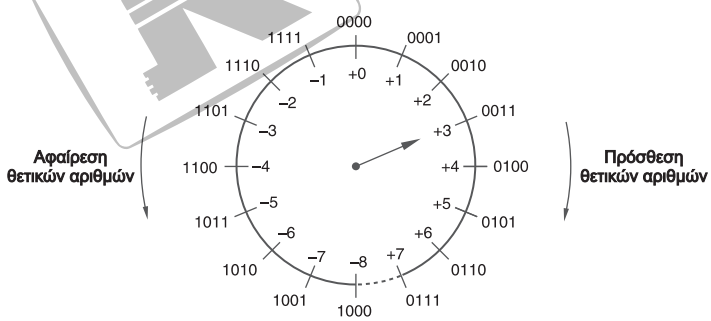
Πίνακας 2-6 Δεκαδικοί αριθμοί και αριθμοί των 4 bit

Δεκαδικός	*Συμπλήρωμα ως προς δύο*	*Συμπλήρωμα ως προς ένα*	*Προσημασμένο μέγεθος*	*Υπέρβαση κατά 2^{m-1}*
-8	1000	—	—	0000
-7	1001	1000	1111	0001
-6	1010	1001	1110	0010
-5	1011	1010	1101	0011
-4	1100	1011	1100	0100
-3	1101	1100	1011	0101
-2	1110	1101	1010	0110
-1	1111	1110	1001	0111
0	0000	1111 ή 0000	1000 ή 0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

+3	0011	–2	1110
+ +4	+ 0100	+ –6	+ 1010
+7	0111	–8	1 1000
+6	0110	+4	0100
+ –3	+ 1101	+ –7	+ 1001
+3	1 0011	–3	1101

2.6.2 Γραφική αναπαράσταση

Ένας άλλος τρόπος για να δει κανείς το σύστημα συμπληρώματος ως προς δύο είναι να χρησιμοποιήσει τον απαριθμητή των 4 bit που φαίνεται στην Εικόνα 2-3. Εδώ έχουμε τοποθετήσει τους αριθμούς σε κυκλική ή "αρθρωτή" αναπαράσταση. Η λειτουργία αυτού του απαριθμητή μιμείται αρκετά καλά αυτή ενός πραγματικού κυκλώματος κανονικής/αντίστροφης απαρίθμησης, το οποίο και θα εξετάσουμε στην Ενότητα 8.4. Αρχίζοντας με το βέλος να δείχνει σε οποιονδήποτε αριθμό, μπορούμε να προσθέσουμε +*n* σε αυτόν τον αριθμό απαριθμώντας *n* φορές ή με άλλα λόγια μετακινώντας το βέλος *n* θέσεις δεξιόστροφα. Επίσης, είναι προφανές ότι μπορούμε να αφαιρέσουμε *n* από έναν αριθμό απαριθμώντας αντίστροφα *n* φορές, δηλαδή μετακινώντας το βέλος *n* θέσεις αριστερόστροφα. Φυσικά, αυτές οι πράξεις δίνουν σωστά αποτελέσματα μόνο όταν το *n* είναι αρκετά μικρός αριθμός έτσι ώστε να μην συμβαίνει υπέρβαση της ασυνέχειας μεταξύ του -8 και του +7.



Εικόνα 2-3 Αρθρωτή αναπαράσταση απαρίθμησης αριθμών συμπληρώματος ως προς δύο των 4 bit.

Το πιο ενδιαφέρον είναι ότι μπορούμε επίσης να αφαιρέσουμε *n* (ή αλλιώς να προσθέσουμε -*n*) μετακινώντας το βέλος κατά *16-n* θέσεις δεξιόστροφα. Σημειώστε ότι η ποσότητα *16-n* είναι αυτό που ορίζουμε ως "συμπλήρωμα ως προς δύο των 4-bit" του *n*, δηλαδή αναπαράσταση συμπληρώματος ως προς δύο του -*n*. Αυτό υποστηρίζει και γραφικά τον προηγούμενο ισχυρισμό μας ότι ένας αρνητικός αριθμός, όταν αναπαρίσταται ως συμπλήρωμα ως προς δύο, μπορεί να προστεθεί σε έναν άλλο αριθμό με πρόσθεση των αναπαραστάσεων 4-bit και εφαρμογή των κα-

νόνων πρόσθεσης δυαδικών. Έτσι λοιπόν, στην Εικόνα 2-3, για να προσθέσουμε έναν αριθμό αρκεί να μετακινήσουμε το βέλος δεξιόστροφα κατά τόσες θέσεις όσες και ο αριθμός.

2.6.3 Υπερχείλιση

Αν μια πράξη πρόσθεσης δώσει αποτέλεσμα που υπερβαίνει το πεδίο τιμών του αριθμητικού συστήματος, τότε θα έχουμε *υπερχείλιση* (*overflow*). Στην αρθρωτή αναπαράσταση απαρίθμησης της Εικόνας 2-3, υπερχείλιση στην πρόσθεση θετικών αριθμών εμφανίζεται όταν αριθμούμε πάνω από το +7. Η πρόσθεση ετερόσημων αριθμών δεν προκαλεί ποτέ υπερχείλιση, αλλά η πρόσθεση δύο ομόσημων αριθμών μπορεί να προκαλέσει, όπως φαίνεται και στα παρακάτω παραδείγματα:

υπερχείλιση

−3	1101	+5	0101
+ −6	+ 1010	+ +6	+ 0110
−9	10111 = +7	+11	1011 = −5
−8	1000	+7	0111
+ −8	+ 1000	+ +7	+ 0111
−16	10000 = +0	+14	1110 = −2

Ευτυχώς, υπάρχει ένας απλός κανόνας για τον εντοπισμό της υπερχείλισης κατά την πρόσθεση: μια πρόσθεση εμφανίζει υπερχείλιση όταν τα πρόσημα των προσθετέων είναι το ίδια και το πρόσημο του αθροίσματος είναι διαφορετικό από το πρόσημο των προσθετέων. Μερικές φορές ο κανόνας της υπερχείλισης διατυπώνεται ως προς τα κρατούμενα που παράγονται κατά τη διάρκεια της πρόσθεσης: μια πράξη πρόσθεσης εμφανίζει υπερχείλιση αν τα κρατούμενα bit εκατέρωθεν της θέσης προσήμου (c_{in} και c_{out}) είναι διαφορετικά. Μια προσεκτική μελέτη του Πίνακα 2-3 στην Ενότητα 2.3 δείχνει ότι οι δύο κανόνες είναι ισοδύναμοι — υπάρχουν μόνο δύο περιπτώσεις όπου $c_{in} \neq c_{out}$, όταν δηλαδή $x=y$ και το bit του αθροίσματος είναι διαφορετικό.

κανόνες υπερχείλισης

2.6.4 Κανόνες αφαίρεσης

Οι αριθμοί συμπληρώματος ως προς δύο αφαιρούνται μεταξύ τους σαν να είναι κοινοί απρόσημοι δυαδικοί αριθμοί και, για την περίπτωση αυτή, θα μπορούσαν να διατυπωθούν κατάλληλοι κανόνες για τον εντοπισμό της υπερχείλισης. Ωστόσο, τα περισσότερα κυκλώματα αφαίρεσης αριθμών συμπληρώματος ως προς δύο δεν πραγματοποιούν την αφαίρεση απευθείας. Αντίθετα, μετατρέπουν τον αφαιρετέο σε αρνητικό παίρνοντας το συμπλήρωμα του ως προς δύο και μετά τον προσθέτουν στο μειωτέο, χρησιμοποιώντας τους συνήθεις κανόνες της πρόσθεσης.

Αφαίρεση αριθμών συμπληρώματος ως προς δύο

Με τη μετατροπή του αφαιρετέου σε αρνητικό αριθμό και κατόπιν την πρόσθεσή του στο μειωτέο, κάνουμε μόνο μία πράξη πρόσθεσης, με τον εξής τρόπο: Παίρνουμε το συμπλήρωμα bit προς bit του αφαιρετέου και προσθέτουμε το αποτέλεσμα στο μειωτέο με αρχικό κρατούμενο (c_{in}) αντί για 0. Ακολουθούν μερικά παραδείγματα:

		$1 — c_{in}$			$1— c_{in}$
+40	100	0100	+3	0011	0011
– +3	– 0011	+ 1100	– +4	– 0100	+ 1011
+1		**1**0001	– 1		1111

		$1 — c_{in}$			$1 — c_{in}$
+3	0011	0011	– 3	1101	1101
– –4	– 1100	+ 0011	– – 4	– 1100	+ 0011
+7		0111	+1		10001

Η υπερχείλιση στην αφαίρεση μπορεί να εντοπιστεί με την εξέταση του προσήμου του μειωτέου και του *συμπληρώματος* του αφαιρετέου, με εφαρμογή του ίδιου κανόνα όπως και στην πρόσθεση. Εναλλακτικά, με χρήση της τεχνικής που εφαρμόστηκε στα προηγούμενα παραδείγματα, παρατηρούνται τα κρατούμενα προς και από τη θέση προσήμου και η υπερχείλιση εντοπίζεται ανεξάρτητα των προσήμων των εισόδων και της εξόδου, πάλι με εφαρμογή του ίδιου κανόνα πρόσθεσης.

Αν επιχειρήσουμε να πάρουμε τον αντίθετο για τον "επιπλέον" αρνητικό αριθμό, θα συμβεί υπερχείλιση σύμφωνα με τους παραπάνω κανόνες, όταν προσθέσουμε 1 στη διαδικασία υπολογισμού του συμπληρώματος:

$$
\begin{array}{rl}
-(-8) = -1000 = & 0111 \\
 & \underline{+\,0001} \\
 & 1000 = -8
\end{array}
$$

Ωστόσο, ο αριθμός αυτός μπορεί ακόμη να χρησιμοποιηθεί σε προσθέσεις και αφαιρέσεις, με την προϋπόθεση ότι το τελικό αποτέλεσμα δεν υπερβαίνει το πεδίο τιμών του αριθμητικού συστήματος:

				$1— c_{in}$
+4	0100	–3	1101	1101
+ –8	+ 1000	– –8	– 1000	+ 0111
–4	1100	+ 5		10101

2.6.5 Αριθμοί συμπληρώματος ως προς δύο και απρόσημοι δυαδικοί αριθμοί

Αφού οι αριθμοί συμπληρώματος ως προς δύο προστίθενται και αφαιρούνται σαν να ήταν απρόσημοι αριθμοί του ίδιου μήκους, με εφαρμογή των βασικών αλγορίθμων πρόσθεσης και αφαίρεσης δυαδικών, ένας υπολογιστής ή ένα άλλο ψηφιακό σύστημα μπορεί να χρησιμοποιήσει το ίδιο κύκλωμα αθροιστή για να χειριστεί αριθμούς και των δύο τύπων. Όμως τα αποτελέσματα πρέπει να ερμηνεύονται διαφορετικά, ανάλογα με το αν το σύστημα έχει να κάνει με προσημασμένους (π.χ. -8 έως και +7) ή με απρόσημους (π.χ. 0 έως 15) αριθμούς.

προσημασμένοι και απρόσημοι αριθμοί

Στην Εικόνα 2-3 είδαμε μια γραφική αναπαράσταση του αριθμητικού συστήματος συμπληρώματος ως προς δύο των 4-bit. Μπορούμε να αλ-

λάξουμε τον τίτλο της εικόνας, όπως φαίνεται στην Εικόνα 2-4 για να πάρουμε μια αναπαράσταση των απρόσημων αριθμών των 4-bit. Οι δυαδικοί συνδυασμοί καταλαμβάνουν τις ίδιες θέσεις πάνω στον τροχό. Έτσι κάθε αριθμός εξακολουθεί να προστίθεται με μετακίνηση του βέλους δεξιόστροφα κατά αντίστοιχο αριθμό θέσεων, ενώ αφαιρείται με μετακίνηση του βέλους αριστερόστροφα.

Μια πράξη πρόσθεσης υπερβαίνει το διάστημα του αριθμητικού συστήματος των απρόσημων αριθμών των 4-bit της Εικόνας 2-4, αν το βέλος κινηθεί δεξιόστροφα μέσω της ασυνέχειας μεταξύ 0 και 15. Σε αυτή την περίπτωση θα εμφανιστεί ένα *κρατούμενο* πέραν της θέσης του πλέον σημαντικού bit.

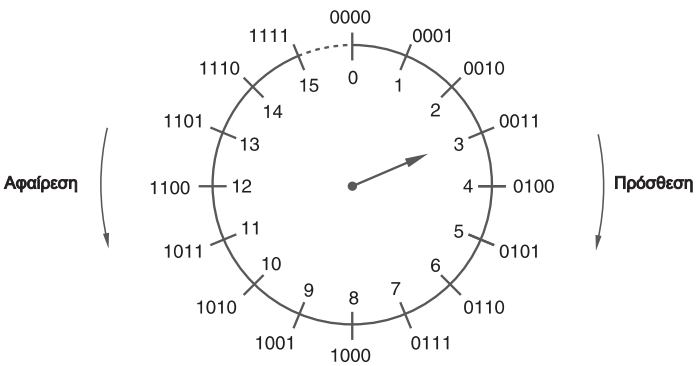
κρατούμενο

Ομοίως, μια πράξη αφαίρεσης υπερβαίνει το διάστημα του αριθμητικού συστήματος αν το βέλος κινηθεί αριστερόστροφα μέσω της ασυνέχειας. Σε αυτή την περίπτωση, θα εμφανιστεί ένα *δανεικό* πέραν της θέσης του πλέον σημαντικού bit.

δανεικό

Από την Εικόνα 2-4 είναι επίσης προφανές ότι μπορούμε να αφαιρέσουμε έναν απρόσημο αριθμό *n* απαριθμώντας *δεξιόστροφα 16-n* θέσεις. Αυτό είναι ισοδύναμο με το να *προσθέσουμε* το συμπλήρωμα ως προς δύο των 4-bit του *n*. Η αφαίρεση παράγει ένα δανεικό αν η αντίστοιχη πρόσθεση του συμπληρώματος ως προς δύο *δεν* παράγει κρατούμενο.

Συνοψίζοντας, στην πρόσθεση απρόσημων, το κρατούμενο ή το δανεικό στη θέση του πλέον σημαντικού bit δείχνει αποτέλεσμα εκτός του διαστήματος του αριθμητικού συστήματος. Στην πρόσθεση προσημασμένων αριθμών συμπληρώματος ως προς δύο, η συνθήκη υπερχείλισης που ορίστηκε προηγουμένως δείχνει αποτέλεσμα εκτός του διαστήματος του αριθμητικού συστήματος. Το κρατούμενο από τη θέση του πλέον σημαντικού bit δεν έχει χρησιμότητα στην πρόσθεση προσημασμένων αριθμών, με τη λογική ότι υπερχείλιση μπορεί να εμφανιστεί ή να μην εμφανιστεί ανεξάρτητα από το κατά πόσον υπάρχει ή όχι κρατούμενο.



Εικόνα 2-4
Αρθρωτή αναπαράσταση απαρίθμησης απρόσημων αριθμών των 4 bit.

Πίνακας 2-7 Περίληψη των κανόνων πρόσθεσης και αφαίρεσης δυαδικών αριθμών

Αριθμητικό σύστημα	***Κανόνες πρόσθεσης***	***Κανόνες αλλαγής προσήμου***	***Κανόνες αφαίρεσης***
Απρόσημοι αριθμοί	Προσθέστε τους αριθμούς. Το αποτέλεσμα είναι εκτός ορίων αν προκύψει κρατούμενο πέραν του MSB	Δεν έχει εφαρμογή	Αφαιρέστε τον αφαιρετέο από το μειωτέο. Το αποτέλεσμα είναι εκτός ορίων αν προκύψει δανεικό πέραν του MSB.
Προσημασμένου μεγέθους	(ομόσημοι) Προσθέστε τα μεγέθη. Υπερχείλιση αν προκύψει κρατούμενο πέραν του MSB. Το αποτέλεσμα έχει το ίδιο πρόσημο. (ετερόσημοι) Αφαιρέστε το μικρότερο μέγεθος από το μεγαλύτερο. Δεν υπάρχει περίπτωση υπερχείλισης. Το αποτέλεσμα έχει το πρόσημο του μεγαλύτερου.	Αλλάξτε το bit προσήμου του αριθμού	Αλλάξτε το bit προσήμου του αφαιρετέου και εκτελέστε πρόσθεση.
Συμπληρώματος ως προς δύο	Εκτελέστε πρόσθεση, αγνοώντας τυχόν κρατούμενα πέραν του MSB. Υπερχείλιση όταν τα κρατούμενα εκατέρωθεν του MSB είναι διαφορετικά.	Υπολογίστε το συμπλήρωμα όλων των bit του αριθμού. Προσθέστε 1 στο αποτέλεσμα.	Υπολογίστε το συμπλήρωμα όλων των bit του αφαιρετέου και προσθέστε το αποτέλεσμα στο μειωτέο με αρχικό κρατούμενο 1.
Συμπληρώματος ως προς ένα	Εκτελέστε πρόσθεση. Αν προκύψει κρατούμενο πέραν του MSB, προσθέστε 1 στο αποτέλεσμα. Υπερχείλιση παρουσιάζεται όταν τα κρατούμενα εκατέρωθεν του MSB είναι διαφορετικά.	Υπολογίστε το συμπλήρωμα όλων των bit του αριθμού.	Υπολογίστε το συμπλήρωμα όλων των bit του αφαιρετέου και εκτελέστε πρόσθεση.

*2.7 Πρόσθεση και αφαίρεση αριθμών συμπληρώματος ως προς ένα

Άλλη μια ματιά στον Πίνακα 2-6 θα μας βοηθήσει να εξηγήσουμε τον κανόνα για την πρόσθεση αριθμών συμπληρώματος ως προς ένα. Αρχίζοντας από το 1000_2 (-7_{10}) και, απαριθμώντας κανονικά, λαμβάνουμε κάθε επόμενο αριθμό συμπληρώματος ως προς ένα προσθέτοντας 1 στον προηγούμενο, *με εξαίρεση* τη μετάβαση από το 1111_2 (αρνητικό 0) στο 0001_2 ($+1_{10}$). Για να διατηρήσουμε τη σωστή απαρίθμηση, πρέπει να προσθέτουμε 2 αντί για 1 κάθε φορά που απαριθμούμε πέρα από το 1111_2. Αυτό δείχνει την εξής τεχνική για την πρόσθεση αριθμών συμπληρώματος ως προς ένα: πραγματοποιήστε τη συνήθη πρόσθεση δυαδικών αλλά προσθέστε ένα επιπλέον 1 κάθε φορά που απαριθμείτε πέρα από το 1111_2.

Η απαρίθμηση πέρα από το 1111_2 κατά τη διάρκεια μιας πρόσθεσης είναι δυνατόν να ανιχνευθεί με παρατήρηση της μεταφοράς της θέσης προσήμου. Κατά συνέπεια, ο κανόνας για την πρόσθεση αριθμών συμπληρώματος ως προς ένα μπορεί να διατυπωθεί αρκετά απλά ως εξής:

πρόσθεση αριθμών συμπληρώματος ως προς ένα

- Πραγματοποιήστε μια συνήθη πρόσθεση δυαδικών και, αν υπάρχει μεταφορά προσήμου, πρόσθεσε 1 στο αποτέλεσμα.

Αυτός ο κανόνας συχνά αποκαλείται και *κυκλικό κρατούμενο*. Παρακάτω ακολουθούν μερικά παραδείγματα πρόσθεσης αριθμών συμπληρώματος ως προς ένα, με τα τρία τελευταία να συμπεριλαμβάνουν κυκλικό κρατούμενο:

κυκλικό κρατούμενο

+ 3	0011	+ 4	0100	+ 5	0101
+ + 4	+ 0100	+ – 7	+ 1000	+ – 5	+ 1010
+7	0111	– 3	1100	– 0	1111
– 2	1101	+ 6	0110	– 0	1111
+ – 5	+ 1010	+ – 3	+ 1100	+ – 0	+ 1111
–7	1 0111	+ 3	1 0010	– 0	1 1110
	+ 1		+ 1		+ 1
	1000		0011		1111

Ακολουθώντας τον παραπάνω κανόνα πρόσθεσης δύο σταδίων, η πρόσθεση ενός αριθμού με το συμπλήρωμά του ως προς ένα δίνει ως αποτέλεσμα ένα αρνητικό 0. Στην πραγματικότητα, μια πράξη πρόσθεσης με χρήση του κανόνα αυτού δεν είναι ποτέ δυνατόν να δώσει ως αποτέλεσμα το θετικό 0, εκτός και αν και οι δύο προσθετέοι είναι θετικό 0.

Όπως και με το συμπλήρωμα ως προς δύο, ο ευκολότερος τρόπος για να αφαιρέσει κανείς έναν αριθμό με την τεχνική συμπληρώματος ως προς ένα είναι να υπολογίσει το συμπλήρωμα του αφαιρετέου και κατόπιν να τον προσθέσει στο μειωτέο. Οι κανόνες υπερχείλισης για την πρόσθεση και την αφαίρεση με την τεχνική συμπληρώματος ως προς ένα είναι οι ίδιοι με αυτούς για το συμπλήρωμα ως προς δύο.

αφαίρεση αριθμών με την τεχνική συμπληρώματος ως προς ένα

Στον Πίνακα 2-7 συνοψίζονται οι κανόνες που παρουσιάσαμε σε αυτή την ενότητα και στις προηγούμενες για την αλλαγή προσήμου, την πρόσθεση, και την αφαίρεση στα δυαδικά αριθμητικά συστήματα.

*2.8 Πολλαπλασιασμός δυαδικών

πολλαπλασιασμός με ολίσθηση και πρόσθεση

πολλαπλασιασμός απρόσημων δυαδικών

Στο δημοτικό σχολείο μάθαμε να πολλαπλασιάζουμε προσθέτοντας μια σειρά από πολλαπλασιαστέους ολίσθησης τους οποίους υπολογίζαμε σύμφωνα με τα ψηφία του πολλαπλασιαστή. Η ίδια μέθοδος μπορεί να χρησιμοποιηθεί και για τον υπολογισμό του γινομένου μεταξύ δύο απρόσημων δυαδικών αριθμών. Ο σχηματισμός των πολλαπλασιαστέων ολίσθησης είναι κάτι εύκολο στον πολλαπλασιασμό δυαδικών, μια και οι μόνες δυνατές τιμές που μπορούν να πάρουν τα ψηφία του πολλαπλασιαστή είναι το 0 και το 1. Ακολουθεί ένα παράδειγμα:

```
   11        1011      πολλαπλασιαστέος
 × 13       ×1101      πολλαπλασιαστής
   33        1011
  11         0000
  143       1011       πολλαπλασιαστέοι ολίσθησης
           1011
         10001111      γινόμενο
```

μερικό γινόμενο

Σε ένα ψηφιακό σύστημα, είναι πιο βολικό να προσθέτουμε κάθε πολλαπλασιαστέο ολίσθησης που δημιουργείται σε ένα *μερικό γινόμενο*, αντί να φτιάχνουμε μια λίστα με τους πολλαπλασιαστέους ολίσθησης και μετά να τους προσθέτουμε όλους μαζί. Εφαρμόζοντας αυτή την τεχνική στο προηγούμενο παράδειγμα, για τον πολλαπλασιασμό αριθμών των 4 bit χρειάζονται τέσσερις προσθέσεις και τέσσερα μερικά γινόμενα:

```
  11          1011      πολλαπλασιαστέος
× 13        × 1101      πολλαπλασιαστής
            0000        μερικό γινόμενο
            1011        πολλαπλασιαστέος ολίσθησης
           01011        μερικό γινόμενο
           0000↓        πολλαπλασιαστέος ολίσθησης
          001011        μερικό γινόμενο
          1011↓↓        πολλαπλασιαστέος ολίσθησης
          0110111       μερικό γινόμενο
          1011↓↓↓       πολλαπλασιαστέος ολίσθησης
         10001111       γινόμενο
```

Γενικά, όταν πολλαπλασιάζουμε έναν αριθμό των n bit με έναν αριθμό των m bit, το γινόμενο που προκύπτει χρειάζεται το πολύ $n+m$ bit για να αναπαρασταθεί. Ο αλγόριθμος με ολίσθηση και πρόσθεση χρειάζεται m μερικά γινόμενα και προσθέσεις για να πάρουμε το τελικό αποτέλεσμα, αλλά η πρώτη πρόσθεση θεωρείται τετριμμένη επειδή το πρώτο μερικό γινόμενο είναι μηδέν. Παρά το γεγονός ότι το πρώτο μερικό γινόμενο έχει μόνο n σημαντικά bit, μετά από κάθε βήμα πρόσθεσης το με-

ρικό γινόμενο αποκτά ένα επιπλέον πλέον σημαντικό bit, αφού κάθε πρόσθεση μπορεί να παράγει ένα κρατούμενο. Ταυτόχρονα, κάθε βήμα οδηγεί σε ένα ακόμη bit μερικού γινομένου, αρχίζοντας από το τελευταίο δεξιά και προχωρώντας προς τα αριστερά, το οποίο δεν αλλάζει. Ο αλγόριθμος με ολίσθηση και πρόσθεση μπορεί να εκτελεσθεί από ένα ψηφιακό κύκλωμα το οποίο περιέχει έναν καταχωρητή ολίσθησης, έναν αθροιστή, και ένα λογικό στοιχείο ελέγχου, όπως περιγράφεται στην Ενότητα 8.7.2.

πολλαπλασιασμός προσημασμένων αριθμών

Ο πολλαπλασιασμός μεταξύ προσημασμένων αριθμών μπορεί να εκτελεστεί αν κάνουμε χρήση του απρόσημου πολλαπλασιασμού και των κανόνων που μάθαμε στο δημοτικό: Πολλαπλασιάστε τα μεγέθη και θεωρήστε το γινόμενο θετικό αν οι τελεστέοι είναι ομόσημοι ή αρνητικό αν είναι ετερόσημοι. Αυτός ο τρόπος είναι πολύ βολικός σε συστήματα προσημασμένων μεγεθών, επειδή το πρόσημο και το μέγεθος είναι ανεξάρτητα μεταξύ τους.

πολλαπλασιασμός αριθμών συμπληρώματος ως προς δύο

Στα συστήματα συμπληρώματος ως προς δύο, ο υπολογισμός του μεγέθους ενός αρνητικού αριθμού και η προσθήκη αρνητικού προσήμου στο απρόσημο γινόμενο δε θεωρούνται τετριμμένες πράξεις. Αυτό μας αναγκάζει να βρούμε έναν πιο αποτελεσματικό τρόπο για τον πολλαπλασιασμό αριθμών συμπληρώματος ως προς δύο, ο οποίος περιγράφεται παρακάτω.

Καταρχήν, ο απρόσημος πολλαπλασιασμός πραγματοποιείται με μια αλληλουχία απρόσημων προσθέσεων των πολλαπλασιαστέων ολίσθησης. Σε κάθε βήμα, η ολίσθηση του πολλαπλασιαστέου αντιστοιχεί στο βάρος του bit του πολλαπλασιαστή. Τα bit ενός αριθμού συμπληρώματος ως προς δύο έχουν τα ίδια βάρη όπως και σε έναν απρόσημο αριθμό, εκτός του MSB το οποίο έχει αρνητικό βάρος (δείτε την Ενότητα 2.5.4). Έτσι μπορούμε να κάνουμε πολλαπλασιασμό αριθμών συμπληρώματος ως προς δύο μέσω μιας αλληλουχίας προσθέσεων των συμπληρωμάτων ως προς δύο των πολλαπλασιαστέων ολίσθησης, με εξαίρεση το τελευταίο βήμα όπου ο πολλαπλασιαστέος ολίσθησης που αντιστοιχεί στο MSB του πολλαπλασιαστή πρέπει να αλλάξει πρόσημο για να προστεθεί στο μερικό γινόμενο. Παρακάτω επαναλαμβάνεται το προηγούμενο παράδειγμα, αυτή τη φορά όμως ο πολλαπλασιαστής και ο πολλαπλασιαστέος ερμηνεύονται ως αριθμοί συμπληρώματος ως προς δύο:

-5	1011	πολλαπλασιαστέος
$\times -3$	$\times 1101$	πολλαπλασιαστής
	$\mathbf{0}0000$	μερικό γινόμενο
	$\mathbf{1}1011$	πολλαπλασιαστέος ολίσθησης
	$\mathbf{1}11011$	μερικό γινόμενο
	$\mathbf{0}0000\downarrow$	πολλαπλασιαστέος ολίσθησης
	$\mathbf{1}111011$	μερικό γινόμενο
	$\mathbf{1}1011\downarrow\downarrow$	πολλαπλασιαστέος ολίσθησης
	$\mathbf{1}1100111$	μερικό γινόμενο
	$\mathbf{0}0101\downarrow\downarrow\downarrow$	πολλαπλασιαστέος ολίσθησης με αλλαγμένο πρόσημο
	$\mathbf{0}0001111$	γινόμενο

Ο χειρισμός των MSB απαιτεί προσοχή, γιατί έχουμε αύξηση κατά ένα σημαντικό ψηφίο σε κάθε βήμα και εργαζόμαστε με προσημασμένους αριθμούς. Επομένως, πριν από την πρόσθεση κάθε πολλαπλασιαστέου ολίσθησης στο μερικό γινόμενο k bit, τους μετατρέπουμε έτσι ώστε να έχουν $k + 1$ σημαντικά bit κάνοντας επέκταση προσήμου, όπως φαίνεται με την έντονη γραφή του παραπάνω παραδείγματος. Έτσι, κάθε άθροισμα που προκύπτει έχει $k + 1$ bit, ενώ αγνοείται κάθε κρατούμενο πέραν του MSB του αθροίσματος $k + 1$ bit.

*2.9 Διαίρεση δυαδικών

διαίρεση με ολίσθηση και αφαίρεση
διαίρεση απρόσημων αριθμών

Ο απλούστερος αλγόριθμος διαίρεσης δυαδικών βασίζεται στη μέθοδο ολίσθησης και αφαίρεσης που μάθαμε στο δημοτικό σχολείο. Στον Πίνακα 2-8 δίνονται παραδείγματα αυτής της μεθόδου για απρόσημους δεκαδικούς και δυαδικούς αριθμούς.

Και στις δύο περιπτώσεις, συγκρίνουμε νοητά τον ανηγμένο διαιρετέο με πολλαπλάσια του διαιρέτη για να προσδιορίσουμε ποιο πολλαπλάσιο του διαιρέτη ολίσθησης θα αφαιρέσουμε. Στην περίπτωση του δεκαδικού, πρώτα επιλέγουμε το 11 ως το μέγιστο πολλαπλάσιο του 11 που είναι συγχρόνως μικρότερο του 21. Έπειτα επιλέγουμε το 99 ως το μέγιστο πολλαπλάσιο του 11 που είναι συγχρόνως μικρότερο του 107. Στην περίπτωση του δυαδικού, η επιλογή είναι κάπως πιο εύκολη μια και οι δύο επιλογές είναι μηδέν, όπως και ο ίδιος ο διαιρέτης.

Οι μέθοδοι διαίρεσης δυαδικών αριθμών είναι περίπου συμπληρωματικές των μεθόδων πολλαπλασιασμού δυαδικών. Ένας τυπικός αλγόριθμος διαίρεσης χρειάζεται ένα διαιρέτη ($n+m$) bit και ένα διαιρετέο n bit και παράγει ένα πηλίκο m bit και ένα υπόλοιπο n bit. Μια διαίρεση λέμε ότι *υπερχειλίζει* όταν ο διαιρέτης είναι μηδέν ή το πηλίκο χρειάζεται παραπάνω από m bit για να αναπαρασταθεί. Στα περισσότερα κυκλώματα διαίρεσης σε υπολογιστές, ισχύει ότι $n = m$.

υπερχείλιση διαίρεσης

Πίνακας 2-8
Παράδειγμα μεγάλης διαίρεσης

19	10011	πηλίκο
11) 217	1011)11011001	διαιρετέος
11	1011	διαιρέτης ολίσθησης
107	0101	ανηγμένος διαιρετέος
99	0000	διαιρέτης ολίσθησης
8	1010	ανηγμένος διαιρετέος
	0000	διαιρέτης ολίσθησης
	10100	ανηγμένος διαιρετέος
	1011	διαιρέτης ολίσθησης
	10011	ανηγμένος διαιρετέος
	1011	διαιρέτης ολίσθησης
	1000	υπόλοιπο

Η διαίρεση προσημασμένων αριθμών μπορεί να πραγματοποιηθεί με εφαρμογή της διαίρεσης απρόσημων αριθμών και των κανόνων που μάθαμε στο δημοτικό σχολείο: Πραγματοποιήστε μια απρόσημη διαίρεση των μεγεθών και κάντε το πηλίκο θετικό αν οι τελεστέοι είναι ομόσημοι και αρνητικό αν είναι ετερόσημοι. Το υπόλοιπο παίρνει το ίδιο πρόσημο με το διαιρετέο. Όπως και στον πολλαπλασιασμό, υπάρχουν ειδικές τεχνικές για την απευθείας διαίρεση αριθμών συμπληρώματος ως προς ένα. Αυτές οι τεχνικές συχνά υλοποιούνται σε κυκλώματα διαίρεσης υπολογιστών (δείτε τις Παραπομπές).

διαίρεση προσημασμένων αριθμών

2.10 Δυαδικοί κώδικες δεκαδικών αριθμών

Παρά το γεγονός ότι οι δυαδικοί αριθμοί είναι οι πλέον κατάλληλοι για τους εσωτερικούς υπολογισμούς που λαμβάνουν χώρα σε ένα ψηφιακό σύστημα, οι περισσότεροι άνθρωποι εξακολουθούν να προτιμούν τις συναλλαγές με δεκαδικούς αριθμούς. Γι' αυτόν το λόγο, οι εξωτερικές διασυνδέσεις ενός ψηφιακού συστήματος ενδέχεται να διαβάζουν ή να εμφανίζουν σε οθόνη δεκαδικούς αριθμούς, ενώ μερικές ψηφιακές διατάξεις στην πράξη επεξεργάζονται απευθείας δεκαδικούς αριθμούς.

Η ανάγκη του ανθρώπου να αναπαριστά δεκαδικούς αριθμούς δεν αλλάζει το βασικό χαρακτήρα των ψηφιακών ηλεκτρονικών κυκλωμάτων. Αυτά εξακολουθούν να επεξεργάζονται σήματα τα οποία βρίσκονται σε μια από δύο μόνο καταστάσεις: 0 και 1. Επομένως, ένας δεκαδικός αριθμός αναπαρίσταται σε ένα ψηφιακό σύστημα από μια ακολουθία από bit, όπου οι διαφορετικοί συνδυασμοί τιμών bit στην ακολουθία αναπαριστούν διαφορετικούς δεκαδικούς αριθμούς. Για παράδειγμα, αν χρησιμοποιήσουμε μια ακολουθία 4-bit για να αναπαραστήσουμε ένα δεκαδικό αριθμό, μπορούμε να αντιστοιχίσουμε το συνδυασμό bit 0000 στο δεκαδικό ψηφίο 0, το 0001 στο 1, το 0010 στο 2 κ.ο.κ.

Ένα σύνολο από ακολουθίες των n bit, στο οποίο διαφορετικές ψηφιοσειρές αναπαριστούν διαφορετικούς αριθμούς ή άλλα πράγματα ονομάζεται *κώδικας*. Κάθε συγκεκριμένος συνδυασμός των n-bit ονομάζεται *κωδικολέξη*. Όπως θα δούμε και στα παραδείγματα δεκαδικών κωδίκων στην ενότητα αυτή, είναι δυνατόν να υπάρχει ή και να μην υπάρχει αριθμητική σχέση μεταξύ των τιμών των bit σε μια κωδικολέξη και στο αντικείμενο που αυτή αναπαριστά. Επιπλέον, ένας κώδικας που χρησιμοποιεί ακολουθίες των n-bit δεν είναι απαραίτητο να περιέχει 2^n έγκυρες κωδικολέξεις.

κώδικας
κωδικολέξη

Τουλάχιστον τέσσερα bit χρειάζονται για να αναπαραστήσουμε τα δέκα δεκαδικά ψηφία. Υπάρχουν δισεκατομμύρια διαφορετικοί τρόποι για να διαλέξουμε δέκα κωδικολέξεις των 4-bit. Μερικοί από τους πιο συνήθεις δεκαδικούς κώδικες δίνονται στον Πίνακα 2-9.

Ίσως ο πιο "φυσιολογικός" δεκαδικός κώδικας είναι το *σύστημα δυαδικά κωδικοποιημένων δεκαδικών* (*BCD*), στο οποίο τα ψηφία από το 0 έως το 9 κωδικοποιούνται σύμφωνα με τις απρόσημες αναπαραστάσεις 4

σύστημα δυαδικά κωδικοποιημένων δεκαδικών (BCD)

Πίνακας 2-9 Δεκαδικοί κώδικες.

Δεκαδικό ψηφίο	*BCD (8421)*	*2421*	*Υπέρβαση κατά 3*	*Διπενταδικός*	*1 από 10*
0	0000	0000	0011	0100001	1000000000
1	0001	0001	0100	0100010	0100000000
2	0010	0010	0101	0100100	0010000000
3	0011	0011	0110	0101000	0001000000
4	0100	0100	0111	0110000	0000100000
5	0101	1011	1000	1000001	0000010000
6	0110	1100	1001	1000010	0000001000
7	0111	1101	1010	1000100	0000000100
8	1000	1110	1011	1001000	0000000010
9	1001	1111	1100	1010000	0000000001
Μη χρησιμοποιούμενες κωδικολέξεις					
	1010	0101	0000	0000000	0000000000
	1011	0110	0001	0000001	0000000011
	1100	0111	0010	0000010	0000000101
	1101	1000	1101	0000011	0000000110
	1110	1001	1110	0000101	0000000111
	1111	1010	1111	...	...

αναπαράσταση πακέτων BCD

bit από το 0000 έως το 1001. Οι κωδικολέξεις από 1010 έως 1111 δε χρησιμοποιούνται. Οι μετατροπές μεταξύ BCD και δεκαδικών αναπαραστάσεων είναι τετριμμένες, αφού πρόκειται για μια απευθείας αναπαράσταση τεσσάρων bit για κάθε δεκαδικό ψηφίο. Μερικά προγράμματα υπολογιστών τοποθετούν δύο ψηφία BCD σε ένα byte των 8-bit σε μια *συμπτυγμένη αναπαράσταση BCD* και κατά συνέπεια ένα byte μπορεί να αναπαριστά τις τιμές από 0 έως 99, ενώ αντίθετα ένας απρόσημος δυαδικός αριθμός των 8-bit μπορεί να αναπαριστά τις τιμές από το 0 έως το 255. Επίσης χρησιμοποιώντας ένα byte για κάθε δύο ψηφία, μπορούμε να πάρουμε αριθμούς BCD με οποιονδήποτε αριθμό ψηφίων.

Όπως συμβαίνει και με τους δυαδικούς αριθμούς, έτσι κι εδώ υπάρχουν πολλές δυνατές αναπαραστάσεις των αρνητικών αριθμών BCD. Οι

ΔΥΩΝΥΜΙΚΟΙ ΣΥΝΤΕΛΕΣΤΕΣ

Ο αριθμός των διαφορετικών τρόπων για να επιλέξουμε m αντικείμενα από ένα σύνολο n αντικειμένων δίνεται από έναν *διωνυμικό συντελεστή*, ο οποίος συμβολίζεται με $\binom{n}{m}$ και η τιμή του ισούται με $\frac{n!}{m! \cdot (n-m)!}$

Για έναν δεκαδικό κώδικα 4-bit, υπάρχουν $\binom{16}{10}$ διαφορετικοί τρόποι να επιλέξουμε 10 από τις 16 κωδικολέξεις των 4-bit και 10! τρόποι να αντιστοιχίσουμε την κάθε διαφορετική επιλογή στα 10 ψηφία. Έτσι, υπάρχουν $\frac{16!}{10! \cdot 6!}$ διαφορετικοί δεκαδικοί κώδικες των 4 bit.

προσημασμένοι αριθμοί BCD έχουν μια επιπλέον θέση ψηφίου για το πρόσημο. Τόσο οι αναπαραστάσεις προσημασμένου μεγέθους όσο και οι αναπαραστάσεις συμπληρώματος ως προς δέκα είναι ιδιαίτερα διαδεδομένες. Σε έναν προσημασμένου μεγέθους αριθμό BCD, η κωδικοποίηση της ακολουθίας bit προσήμου είναι αυθαίρετη. Στο σύστημα αριθμών συμπληρώματος ως προς δέκα, το 0000 υποδηλώνει το σημείο "συν" και το 1001 υποδηλώνει το σημείο "πλην".

πρόσθεση BCD

Η πρόσθεση ψηφίων BCD είναι παρόμοια με την πρόσθεση απρόσημων δυαδικών αριθμών των 4-bit, με τη διαφορά ότι πρέπει να γίνει μια διόρθωση αν το αποτέλεσμα υπερβαίνει το 1001. Το αποτέλεσμα διορθώνεται με πρόσθεση του 6. Ακολουθούν μερικά παραδείγματα:

5	0101		4	0100
+ 9	+ 1001		+ 5	+ 0101
14	1110		9	1001
	+ 0110	— διόρθωση		
10 + 4	1 0100			
8	1000		9	1001
+ 8	+ 1000		+ 9	+ 1001
16	1 0000		18	1 0010
	+ 0110	— διόρθωση		+ 0110 — διόρθωση
10 + 61	0110		10 + 8	1 1000

Σημειώστε ότι η πρόσθεση μεταξύ δύο ψηφίων BCD έχει αποτέλεσμα την εισαγωγή ενός κρατουμένου στην επόμενη θέση ψηφίου αν είτε η αρχική πρόσθεση δυαδικών είτε η πρόσθεση του παράγοντα διόρθωσης έχει αποτέλεσμα ένα κρατούμενο. Πολλοί υπολογιστές εκτελούν αριθμητικές πράξεις συμπτυγμένης αναπαράστασης BCD με τη βοήθεια ειδικών εντολών οι οποίες χειρίζονται αυτόματα τη διόρθωση κρατουμένου.

σταθμισμένος κώδικας

Το σύστημα BCD είναι ένας *σταθμισμένος κώδικας* επειδή μπορούμε να πάρουμε το κάθε δεκαδικό ψηφίο από την κωδικολέξη του, με την αντιστοίχιση ενός σταθερού βάρους σε κάθε bit κωδικολέξης. Τα βάρη των bit στο BCD είναι 8, 4, 2, και 1 και, για το λόγο αυτόν, ο κώδικας μερικές φορές ονομάζεται και *κώδικας 8421*. Ένα άλλο σύνολο από βάρη έχει αποτέλεσμα τον *κώδικα 2421*, όπως φαίνεται στον Πίνακα 2-9. Αυτός ο κώδικας έχει το πλεονέκτημα ότι είναι *αυτοσυμπληρούμενος* κώδικας, που σημαίνει ότι μπορούμε να πάρουμε την κωδικολέξη για το συμπλήρωμα ως προς 9 του κάθε ψηφίου υπολογίζοντας το συμπλήρωμα των επιμέρους bit της κωδικολέξης του ψηφίου.

κώδικας 8421
κώδικας 2421
αυτοσυμπληρούμενος κώδικας

κώδικας υπέρβασης κατά 3

Ένας άλλος αυτοσυμπληρούμενος κώδικας που παρουσιάζεται στον Πίνακα 2-9 είναι ο *κώδικας υπέρβασης κατά 3*. Παρά το γεγονός ότι αυτός ο κώδικας δεν είναι σταθμισμένος, έχει μια αριθμητική σχέση με τον κώδικα BCD: η κωδικολέξη για κάθε δεκαδικό ψηφίο είναι η αντίστοιχη κωδικολέξη BCD συν 0011_2. Επειδή λοιπόν οι κωδικολέξεις ακολουθούν μια τυπική ακολουθία απαρίθμησης δυαδικών, οι τυπικοί δυαδικοί απαριθμητές μπορούν εύκολα να κατασκευαστούν έτσι ώστε να απαριθμούν

σε κώδικα υπέρβασης κατά 3, όπως θα δούμε στην Εικόνα 8-37, στην Ενότητα 8.4.3.

διπενταδικός κώδικας

Οι δεκαδικοί κώδικες μπορούν να έχουν περισσότερα από τέσσερα bit. Για παράδειγμα, ο διπενταδικός κώδικας του Πίνακα 2-9 χρησιμοποιεί επτά. Τα πρώτα δύο bit μιας κωδικολέξης υποδηλώνουν αν ο αριθμός είναι στο διάστημα από 0 έως 4 ή από 5 έως 9, ενώ τα τελευταία 5-bit υποδηλώνουν για ποιον από τους πέντε αριθμούς στο επιλεγόμενο διάστημα πρόκειται.

Ένα εν δυνάμει πλεονέκτημα της χρήσης περισσότερων από τον ελάχιστο αριθμό bit σε έναν κώδικα είναι η ιδιότητα ανίχνευσης σφαλμάτων. Σε ένα διπενταδικό κώδικα, αν οποιοδήποτε bit μιας κωδικολέξης αλλάξει τυχαία και πάρει την αντίθετη τιμή, τότε αυτή η κωδικολέξη δεν αναπαριστά δεκαδικό ψηφίο και συνεπώς θα προσημειωθεί ως σφάλμα. Από τις 128 πιθανές κωδικολέξεις των 7-bit, μόνο 10 είναι έγκυρες και αναγνωρίζονται ως δεκαδικά ψηφία, ενώ οι υπόλοιπες αν εμφανιστούν μπορούν να προσημειωθούν ως σφάλματα.

κώδικας 1 από 10

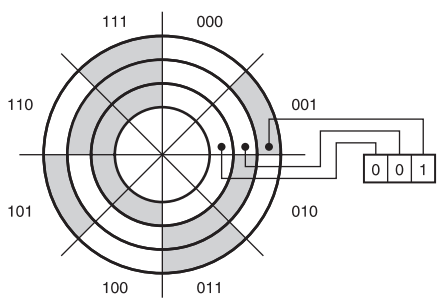
Ο *κώδικας 1 από 10*, όπως αυτός που περιλαμβάνεται στην τελευταία στήλη του Πίνακα 2-9, χρησιμοποιεί 10 από τις 1024 πιθανές κωδικολέξεις των 10-bit και είναι η πιο αραιή κωδικοποίηση για δεκαδικά ψηφία.

2.11 Κώδικας Gray

Στις ηλεκτρομηχανικές εφαρμογές των ψηφιακών συστημάτων, όπως για παράδειγμα στις εργαλειομηχανές, στα συστήματα φρένων αυτοκινήτων, και στα φωτοαντιγραφικά, είναι μερικές φορές απαραίτητο για έναν αισθητήρα εισόδου να δίνει μια ψηφιακή τιμή η οποία αναπαριστά μια μηχανική θέση. Για παράδειγμα, στην Εικόνα 2-5 βλέπουμε ένα εννοιολογικό σκαρίφημα ενός δίσκου κωδικοποίησης και ένα σύνολο επαφών που δίνουν μία από τις οκτώ δυαδικές κωδικολέξεις των 3-bit οι οποίες εξαρτώνται από τη θέση περιστροφής του δίσκου. Οι σκούρες περιοχές του δίσκου συνδέονται με μια πηγή σήματος και αντιστοιχούν στο λογικό 1, ενώ οι λευκές περιοχές δεν είναι συνδεδεμένες, γεγονός που οι επαφές ερμηνεύουν ως λογικό 0.

Ο κωδικοποιητής της Εικόνας 2-5 έχει πρόβλημα όταν ο δίσκος βρίσκεται στα συγκεκριμένα όρια μεταξύ των περιοχών. Για παράδειγμα, θεωρήστε το όριο μεταξύ των περιοχών του δίσκου 001 και 010, όπου

Εικόνα 2-5
Μηχανικός δίσκος κωδικοποίησης που χρησιμοποιεί δυαδικό κώδικα των 3 bit.



Δεκαδικός αριθμός	*Δυαδικός κώδικας*	*Κώδικας Gray*
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Πίνακας 2-10 Σύγκριση μεταξύ του δυαδικού κώδικα των 3 bit και του κώδικα Gray.

αλλάζουν δύο από τα κωδικοποιημένα bit. Ποια τιμή θα δώσει ο κωδικοποιητής αν ο δίσκος είναι τοποθετημένος ακριβώς πάνω στο θεωρητικό όριο; Από τη στιγμή που είμαστε πάνω στο όριο, τόσο το 001 όσο και το 010 είναι αποδεκτές τιμές. Όμως, επειδή το μηχανικό σύστημα δεν είναι τέλειο, οι δύο επαφές στα δεξιά είναι δυνατόν να έρθουν και οι δύο σε επαφή με μια περιοχή "1", δίνοντας μια εσφαλμένη ανάγνωση 011. Με παρόμοιο τρόπο, και μια ανάγνωση 000 είναι πιθανή. Γενικά, ένα τέτοιο πρόβλημα μπορεί να προκύψει σε κάθε όριο όπου έχουμε αλλαγή περισσοτέρων του ενός bit. Τα μεγαλύτερα προβλήματα εμφανίζονται όταν αλλάζουν και τα τρία bit όπως π.χ. στα όρια 000-111 και 011-100.

Το πρόβλημα του δίσκου κωδικοποίησης μπορεί να λυθεί με την εξεύρεση ενός ψηφιακού κώδικα στον οποίο αλλάζει μόνο ένα bit μεταξύ κάθε ζεύγους διαδοχικών κωδικολέξεων. Ένας τέτοιος κώδικας ονομάζεται *κώδικας Gray*. Στον Πίνακα 2-10 παρουσιάζεται ένας κώδικας Gray των 3 bit.

κώδικας Gray

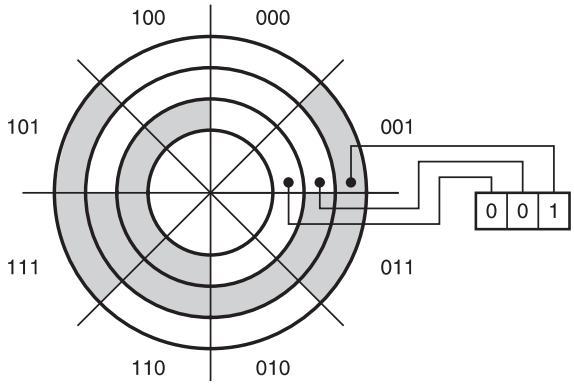
Ξανασχεδιάσαμε λοιπόν το δίσκο κωδικοποίησης κάνοντας χρήση του κώδικα Gray, όπως φαίνεται και στην Εικόνα 2-6. Σε αυτόν το νέο δίσκο, σε κάθε όριο αλλάζει μόνο ένα bit. Έτσι, η ανάγνωση πάνω στην οριογραμμή δίνει μια τιμή που βρίσκεται είτε στη μια είτε στην άλλη πλευρά του ορίου.

Υπάρχουν δύο βολικοί τρόποι για να κατασκευάσει κανείς έναν κώδικα Gray με όποιον αριθμό bit επιθυμεί. Η πρώτη μέθοδος βασίζεται στο γεγονός ότι ο κώδικας Gray είναι ένας *ανακλώμενος κώδικας*, δηλαδή μπορεί να ορισθεί και να κατασκευαστεί αναδρομικά με τη βοήθεια των ακόλουθων κανόνων:

ανακλώμενος κώδικας

1. Ένας κώδικας Gray του 1-bit έχει δύο κωδικολέξεις: το 0 και το 1.
2. Οι πρώτες 2^n κωδικολέξεις ενός κώδικα Gray των $(n + 1)$ bit ισούνται με τις κωδικολέξεις ενός κώδικα Gray των n bit, γραμμένες με κανονική σειρά με προσαρτημένο ένα 0 στην αρχή.
3. Οι τελευταίες 2^n κωδικολέξεις ενός κώδικα Gray των $(n + 1)$ bit ισούνται με τις κωδικολέξεις ενός κώδικα Gray των n bit, αλλά γραμμένες με αντίστροφη σειρά με προσαρτημένο ένα 1 στην αρχή.

Εικόνα 2-6
Μηχανικός δίσκος κωδικοποίησης με χρήση κώδικα Gray των 3 bit



Αν σχεδιάσουμε μια γραμμή μεταξύ της στήλης 3 και της στήλης 4 του Πίνακα 2-10, μπορούμε να δούμε ότι οι κανόνες 2 και 3 αληθεύουν για τον κώδικα Gray των 3-bit. Φυσικά, για να κατασκευάσει κανείς έναν κώδικα Gray των *n*-bit για μια οποιαδήποτε τιμή του *n* κάνοντας χρήση της παραπάνω μεθόδου, πρέπει επίσης να κατασκευάσει έναν κώδικα Gray για κάθε μήκος μικρότερο του *n*.

Η δεύτερη μέθοδος μάς επιτρέπει να πάρουμε μια κωδικολέξη ενός κώδικα Gray των *n*-bit απευθείας από την αντίστοιχη δυαδική κωδικολέξη των *n* bit:

1. Τα bit μιας δυαδικής κωδικολέξης ή μιας κωδικολέξης ενός κώδικα Gray των *n* bit αριθμούνται από τα δεξιά προς τα αριστερά, από το 0 έως το *n* - 1.
2. Το *i*-οστό bit μιας κωδικολέξης ενός κώδικα Gray είναι 0 αν τα bit *i* και *i* + 1 της αντίστοιχης δυαδικής κωδικολέξης είναι ίδια μεταξύ τους, διαφορετικά το *i*-οστό bit είναι 1. (Όταν *i* + 1= *n*, το bit *n* της δυαδικής κωδικολέξης θεωρείται ότι είναι 0.)

Μια προσεκτική ματιά στον Πίνακα 2-10 δείχνει ότι τα παραπάνω είναι αληθή για τον κώδικα Gray των 3 bit.

*2.12 Κώδικες χαρακτήρων

Όπως είδαμε στην προηγούμενη ενότητα, μια ακολουθία bit δεν είναι απαραίτητο να αναπαριστά έναν αριθμό και, στην πραγματικότητα, το μεγαλύτερο μέρος του συνόλου των πληροφοριών που επεξεργάζονται οι υπολογιστές δεν είναι αριθμητικές. Η πιο κοινή μορφή μη αριθμητικών δεδομένων είναι το *κείμενο*, δηλαδή αλφαριθμητικά που περιέχουν χαρακτήρες από κάποιο σύνολο χαρακτήρων. Κάθε χαρακτήρας αναπαρίσταται μέσα στον υπολογιστή από μια ψηφιοσειρά, σύμφωνα με μια καθορισμένη σύμβαση.

κείμενο

ASCII

Ο πιο διαδεδομένος κώδικας χαρακτήρων είναι ο *ASCII* (προφέρεται "άσκι" — ακρώνυμο του American Standard Code for Information Interchange). Στον κώδικα ASCII σε κάθε χαρακτήρα αντιστοιχεί μια

ακολουθία των 7-bit, και έτσι προκύπτει ένα σύνολο από 128 διαφορετικούς χαρακτήρες, όπως φαίνεται στον Πίνακα 2-11. Ο κώδικας περιέχει τα μικρά και κεφαλαία γράμματα του λατινικού αλφαβήτου, τους αριθμούς, τα σημεία στίξης, και διάφορους μη εκτυπώσιμους χαρακτήρες ελέγχου. Έτσι, το αλφαριθμητικό κειμένου "Yeccch!" αντιστοιχεί στην παρακάτω σειρά αριθμών των 7-bit:

1011001 1100101 1100011 1100011 1100011 1101000 0100001

2.13 Κώδικες για ενέργειες, συνθήκες και καταστάσεις

Οι κώδικες τους οποίους περιγράψαμε μέχρι τώρα χρησιμοποιούνται γενικά για την αναπαράσταση πραγμάτων που θα μπορούσαν να θεωρηθούν "δεδομένα", όπως οι αριθμοί, οι θέσεις, και οι χαρακτήρες. Οι προγραμματιστές γνωρίζουν ότι σε ένα πρόγραμμα υπολογιστή μπορούν να χρησιμοποιηθούν πολλοί διαφορετικοί τύποι δεδομένων.

Στη σχεδίαση ψηφιακών συστημάτων, συχνά βρισκόμαστε αντιμέτωποι με εφαρμογές που δεν αφορούν δεδομένα, όπου μια ακολουθία bit πρέπει να χρησιμοποιηθεί για τον έλεγχο μιας ενέργειας, την προσημείωση μιας συνθήκης, ή την αναπαράσταση της τρέχουσας κατάστασης του υλικού. Πιθανώς ο πιο διαδεδομένος τύπος κώδικα για μια τέτοια εφαρμογή είναι ο απλός δυαδικός κώδικας. Αν υπάρχουν n διαφορετικές ενέργειες, συνθήκες, ή καταστάσεις, μπορούμε να τις αναπαραστήσουμε με ένα δυαδικό κώδικα των b bit, όπου ο αριθμός των απαιτούμενων bit είναι $b=\lceil \log_2 n \rceil$. (Οι αγκύλες $\lceil\ \rceil$ υποδηλώνουν τη *συνάρτηση οροφής* - το μικρότερο ακέραιο που είναι μεγαλύτερος ή ίσος με την ποσότητα μέσα στις αγκύλες. Κατά συνέπεια, το b είναι ο μικρότερος ακέραιος τέτοιος ώστε $2^b \geq n$.)

$\lceil\ \rceil$
συνάρτηση οροφής

Για παράδειγμα, θεωρήστε έναν απλό ελεγκτή φαναριού ρύθμισης κυκλοφορίας. Τα σήματα στη διασταύρωση μιας οδού με διεύθυνση βορράς-νότος (Β-Ν) με μια οδό με διεύθυνση ανατολή-δύση (Α-Δ) είναι δυνατόν να βρίσκονται σε μία από τις έξι καταστάσεις που παρατίθενται στον Πίνακα 2-12.

Αυτές οι καταστάσεις είναι δυνατόν να κωδικοποιηθούν με 3-bit, όπως φαίνεται και στην τελευταία στήλη του παραπάνω πίνακα. Εδώ, χρησιμοποιούνται μόνο έξι από τις οκτώ πιθανές κωδικολέξεις των 3-bit, ενώ η αντιστοίχιση των έξι επιλεγμένων κωδικολέξεων σε καταστάσεις είναι αυθαίρετη, συνεπώς πολλές άλλες κωδικοποιήσεις είναι δυνατές. Ένας έμπειρος σχεδιαστής ψηφιακών συστημάτων θα επέλεγε μια συγκεκριμένη κωδικοποίηση για να ελαχιστοποιήσει το μέγεθος του κυκλώματος ή να βελτιστοποιήσει κάποιες άλλες παραμέτρους (όπως ο χρόνος σχεδίασης, αφού δεν υπάρχει κανένας λόγος να δοκιμάζει τεράστιους αριθμούς δυνατών κωδικοποιήσεων).

Μια άλλη εφαρμογή δυαδικού κώδικα παρουσιάζεται στην Εικόνα 2-7(α). Εδώ έχουμε ένα σύνολο που αποτελείται από n διατάξεις, κάθε

Πίνακας 2-11 Αμερικανικός Πρότυπος Κώδικας για την Ανταλλαγή Πληροφοριών (ASCII) Πρότυπο υπ' αριθμ. X3.4-1968 του Αμερικανικού Εθνικού Ιδρύματος Προτύπων.

		$b_6b_5b_4$ (στήλη)							
$b_3b_2b_1b_0$	Γραμμή (16δικός)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	'	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Κώδικες ελέγχου

NUL	Κενό	DLE	Διαφυγή σύνδεσης δεδομένων
SOH	Αρχή επικεφαλίδας	DC1	Έλεγχος διάταξης 1
STX	Αρχή κειμένου	DC2	Έλεγχος διάταξης 2
ETX	Τέλος κειμένου	DC3	Έλεγχος διάταξης 3
EOT	Τέλος μετάδοσης	DC4	Έλεγχος διάταξης 4
ENQ	Ερώτημα	NAK	Αρνητική επιβεβαίωση
ACK	Αναγνώριση	SYN	Συγχρονισμός
BEL	Κουδούνισμα	ETB	Τέλος μπλοκ μετάδοσης
BS	Οπισθοδρόμηση	CAN	Άκυρο
HT	Οριζόντιος στηλοθέτης	EM	Τέλος μέσου
LF	Αλλαγή γραμμής	SUB	Αντικατάσταση
VT	Κατακόρυφος στηλοθέτης	ESC	Διαφυγή
FF	Αλλαγή σελίδας	FS	Διαχωριστικό αρχείων
CR	Επαναφορά κεφαλής	GS	Διαχωριστικό ομάδων
SO	Shift ανασηκωμένο	RS	Διαχωριστικό εγγραφών
SI	Shift πατημένο	US	Διαχωριστικό μονάδων
SP	Κενό διάστημα	DEL	Διαγραφή

Πίνακας 2-12 Καταστάσεις σε έναν ελεγκτή φαναριού ρύθμισης κυκλοφορίας

	Φωτεινές ενδείξεις						
Κατάσταση	***Β-Ν Πράσινο***	***Β-Ν Πορτοκαλί***	***Β-Ν κόκκινο***	***Α-Δ Πράσινο***	***Α-Δ Πορτοκαλί***	***Α-Δ Κόκκινο***	***Κωδικο-λέξη***
Β-Ν ξεκίνα	αναμμένο	σβηστό	σβηστό	σβηστό	σβηστό	αναμμένο	000
Β-Ν περίμενε	σβηστό	αναμμένο	σβηστό	σβηστό	σβηστό	αναμμένο	001
Β-Ν σταμάτα	σβηστό	σβηστό	αναμμένο	σβηστό	σβηστό	αναμμένο	010
Α-Δ ξεκίνα	σβηστό	σβηστό	αναμμένο	αναμμένο	σβηστό	σβηστό	100
Α-Δ περίμενε	σβηστό	σβηστό	αναμμένο	σβηστό	αναμμένο	σβηστό	101
Α-Δ σταμάτα	σβηστό	σβηστό	αναμμένο	σβηστό	σβηστό	αναμμένο	110

μία από τις οποίες μπορεί να πραγματοποιεί μια συγκεκριμένη ενέργεια. Τα χαρακτηριστικά των διατάξεων είναι τέτοια έτσι ώστε αυτές να έχουν τη δυνατότητα να λειτουργούν μία-μία. Η μονάδα ελέγχου παράγει μια δυαδικά κωδικοποιημένη λέξη "επιλογής διάταξης" με $\lceil \log_2 n \rceil$ bit προκειμένου να υποδεικνύει τη διάταξη που ενεργοποιείται σε κάθε χρονική στιγμή. Η κωδικολέξη "επιλογής διάταξης" εφαρμόζεται σε κάθε διάταξη, η οποία τη συγκρίνει με το δικό της "αριθμό ταυτότητας διάταξης" για να διαπιστώσει κατά πόσον είναι ενεργοποιημένη. Παρά το γεγονός ότι οι κωδικολέξεις έχουν τον ελάχιστο αριθμό bit, ένας δυαδικός κώδικας δεν είναι πάντα η καλύτερη επιλογή για την κωδικοποίηση ενεργειών, συνθηκών, ή καταστάσεων. Η Εικόνα 2-7(β) δείχνει πώς μπορεί να ελέγξει κανείς *n* διατάξεις με έναν *κώδικα 1 από n*, δηλαδή έναν κώδικα των *n* bit όπου οι έγκυρες κωδικολέξεις έχουν ένα bit ίσο με 1 και τα υπόλοιπα bit ίσα με 0. Κάθε bit του κώδικα "1 από *n*" συνδέεται απευθείας με την ενεργοποιημένη είσοδο της αντίστοιχης διάταξης. Αυτό απλοποιεί τη σχεδίαση των διατάξεων, μια και δεν έχουν πλέον αριθμό ταυτότητας διάταξης, αλλά χρειάζονται μόνο ένα bit "έγκρισης" εισόδου.

κώδικας 1 από n

Στον Πίνακα 2-9 παρουσιάζονται οι κωδικολέξεις ενός κώδικα "1 από 10". Μερικές φορές, μια λέξη που αποτελείται μόνο από 0 μπορεί και αυτή να συμπεριληφθεί σε έναν κώδικα "1 από *n*" για να δηλώσει ότι δεν έχει επιλεγεί καμία διάταξη. Ένας άλλος συνήθης κώδικας είναι ο *κώδικας αντεστραμμένου 1 από n* στον οποίο οι έγκυρες κωδικολέξεις έχουν ένα bit 0 ενώ τα υπόλοιπα bit ισούνται με 1.

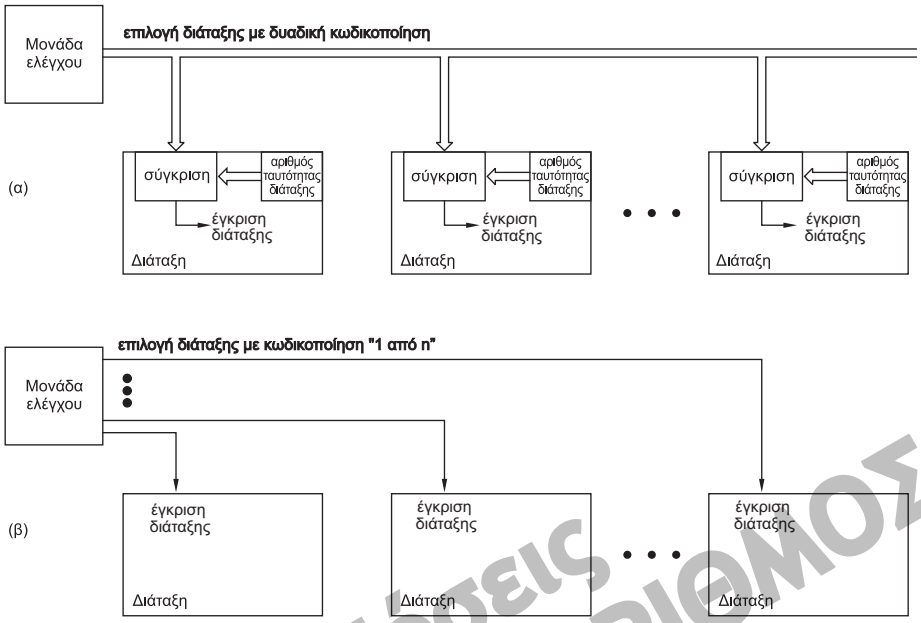
κώδικας αντεστραμμένου 1 από n

Στα σύνθετα συστήματα, είναι δυνατόν να χρησιμοποιηθεί ένας συνδυασμός τεχνικών κωδικοποίησης. Για παράδειγμα, θεωρήστε ένα σύστημα όμοιο με αυτό της Εικόνας 2-7(β), στο οποίο κάθε μία από τις *n* διατάξεις περιλαμβάνει έως και *s* υποδιατάξεις.

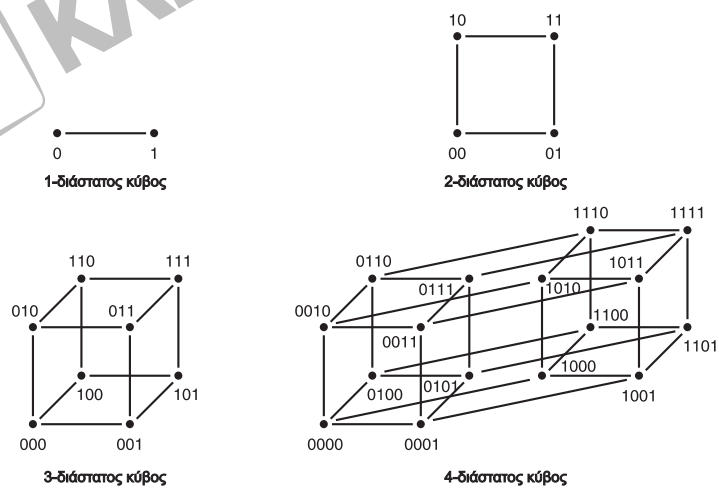
Η μονάδα ελέγχου μπορεί να παράγει έναν κώδικα επιλογής διάταξης με ένα πεδίο κωδικοποίησης "1 από *n*" για την επιλογή διάταξης και ένα πεδίο δυαδικής κωδικοποίησης των $\lceil \log_2 s \rceil$ bit για την επιλογή μίας από τις *s* υποδιατάξεις της επιλεγμένης διάταξης.

Ο *κώδικας "m από n"* είναι η γενίκευση του κώδικα "1 από *n*" στον οποίο οι έγκυρες κωδικολέξεις έχουν *m* bit ίσα με 1 και τα υπόλοιπα bit

κώδικας "m από n"



Εικόνα 2-7 Δομή ελέγχου για ένα ψηφιακό σύστημα *n* διατάξεων: (α) με χρήση δυαδικού κώδικα, (β) με χρήση κώδικα "1 από *n*"



Εικόνα 2-8 *n*-διάστατοι κύβοι για *n* =1, 2, 3 και 4.

ίσα με 0. Ένας κώδικας "*m* από *n*" μπορεί να εντοπιστεί από μια πύλη AND *m* εισόδων η οποία δίνει έξοδο 1 αν όλες της οι είσοδοι είναι 1. Τούτο είναι σχετικά απλό και οικονομικό για να γίνει, όμως για τις περισσότερες τιμές του *m* ένας κώδικας "*m* από *n*" έχει πολύ περισσότερες έγκυρες κωδικολέξεις από ό,τι ένας κώδικας "1 από *n*". Ο συνολικός αριθμός κωδικολέξεων δίνεται από το διωνυμικό συντελεστή $\binom{n}{m}$, ο οποίος έχει

την τιμή $\frac{n!}{m! \cdot (n-m)!}$. Έτσι, ο κώδικας “2 από 4” έχει 6 έγκυρες κωδικολέξεις, ενώ ο κώδικας “3 από 10” έχει 120.

Μια σημαντική παραλλαγή του κώδικα “*m* από *n*” είναι ο *κώδικας 8B10B* που χρησιμοποιείται στο πρότυπο Gigabit Ethernet 802.3z. Ο κώδικας αυτός χρησιμοποιεί 10 bit για την αναπαράσταση 256 έγκυρων κωδικολέξεων, δηλαδή δεδομένων που χρησιμοποιούν 8 bit. Οι περισσότεροι κώδικες χρησιμοποιούν κωδικοποίηση “5 από 10”. Ωστόσο, εφόσον το $\binom{10}{5}$ είναι μόνο 252, χρησιμοποιούνται και μερικές κωδικολέξεις “4 από 10” και “6 από 10” για τη συμπλήρωση του κώδικα με έναν ενδιαφέροντα τρόπο. Περισσότερα πάνω σε αυτό θα δούμε στην Ενότητα 2.16.2.

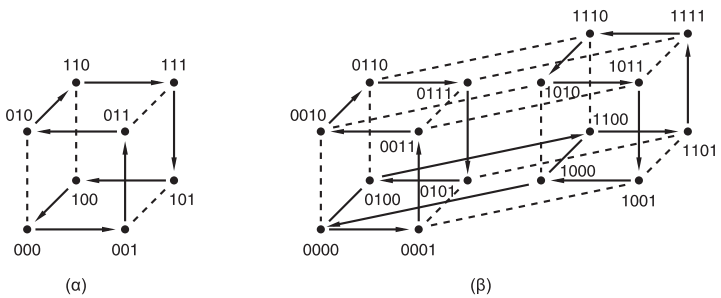
κώδικας 8B10B

2.14 *n*-διάστατοι κύβοι και απόσταση

Μια ακολουθία των *n* bit μπορεί να αναπαρασταθεί γεωμετρικά ως κορυφή ενός αντικειμένου που ονομάζεται *n-διάστατος κύβος*. Η Εικόνα 2-8 δείχνει *n*-διάστατους κύβους για *n* = 1, 2, 3, 4. Ένας *n*-διάστατος κύβος έχει 2^n κορυφές, κάθε μία από τις οποίες έχει ως ετικέτα μια ακολουθία *n* bit. Οι ακμές σχεδιάζονται έτσι ώστε κάθε κορυφή να είναι γειτονική σε *n* άλλες κορυφές των οποίων οι ετικέτες διαφέρουν από την εν λόγω κορυφή κατά μόνο ένα bit. Για τιμές του *n* μεγαλύτερες του 4, η σχεδίαση των *n*-διάστατων κύβων είναι πραγματικά δύσκολη.

n-διάστατος κύβος

Για λογικές τιμές του *n*, οι *n*-διάστατοι κύβοι καθιστούν εύκολη την οπτική αναπαράσταση ορισμένων προβλημάτων κωδικοποίησης και ελαχιστοποίησης λογικής. Για παράδειγμα, το πρόβλημα της σχεδίασης ενός κώδικα Gray των *n* bit είναι ισοδύναμο με το να βρεθεί μια διαδρομή κατά μήκος των ακμών ενός *n*-διάστατου κύβου, η οποία διέρχεται από κάθε κορυφή μία φορά ακριβώς. Οι διαδρομές για τους κώδικες Gray των 3 και 4-bit δίνονται στην Εικόνα 2-9.



Εικόνα 2-9
Διάσχιση *n*-διάστατου κύβου κατά σειρά κώδικα Gray:
(α) 3-διάστατος κύβος,
(β) 4-διάστατος κύβος

Οι κύβοι δίνουν επίσης και μια γεωμετρική ερμηνεία στην έννοια *απόσταση*, η οποία ονομάζεται και *απόσταση Hamming*. Η απόσταση μεταξύ δύο ακολουθιών *n* bit είναι ο αριθμός των θέσεων bit κατά τις οποίες αυτές διαφέρουν. Σε ένα *n*-διάστατο κύβο, η απόσταση αυτή είναι το ελά-

απόσταση
απόσταση Hamming

χιστο μήκος μιας διαδρομής μεταξύ των δύο αντίστοιχων κορυφών. Δύο γειτονικές κορυφές έχουν απόσταση 1. Οι κορυφές 001 και 100 σε έναν 3-διάστατο κύβο έχουν απόσταση 2. Η έννοια της απόστασης είναι σημαντική στη σχεδίαση και κατανόηση των κωδίκων ανίχνευσης σφαλμάτων που παρουσιάζονται στην επόμενη ενότητα.

m-διάστατος υποκύβος

Ένας *m-διάστατος υποκύβος* ενός *n*-διάστατου κύβου είναι ένα σύνολο από 2^m κορυφές, στις οποίες τα *n-m* εκ των συνολικών bit έχουν την ίδια τιμή σε κάθε κορυφή, ενώ τα υπόλοιπα *m*-bit καταλαμβάνουν όλους τους 2^m συνδυασμούς. Για παράδειγμα, οι κορυφές (000, 010, 100, 110) σχηματίζουν ένα 2-διάστατο υποκύβο του 3-διάστατου κύβου. Αυτός ο υποκύβος μπορεί επίσης να οριστεί και από μια απλή ακολουθία xx0, όπου το "x" ορίζει ότι το συγκεκριμένο bit είναι *άνευ σημασίας*. Κάθε κορυφή της οποίας τα bit ταιριάζουν με τα bit που δε βρίσκονται στις θέσεις x ανήκουν σε αυτόν τον υποκύβο. Η έννοια των υποκύβων είναι ιδιαίτερα χρήσιμη στην οπτική αναπαράσταση αλγορίθμων που ελαχιστοποιούν το κόστος των συναρτήσεων συνδυαστικής λογικής, όπως θα δείξουμε στην Ενότητα 4.4.

bit άνευ σημασίας

*2.15 Κώδικες για εντοπισμό και διόρθωση σφαλμάτων

σφάλμα

αστοχία

προσωρινή αστοχία

μόνιμη αστοχία

Σε ένα ψηφιακό σύστημα, *σφάλμα* είναι η αλλοίωση των σωστών τιμών των δεδομένων, τα οποία παίρνουν κάποιες άλλες τιμές. Ένα σφάλμα οφείλεται σε μια φυσική *αστοχία*. Οι αστοχίες μπορεί να είναι είτε προσωρινές είτε μόνιμες. Για παράδειγμα, μια κοσμική ακτίνα ή ένα σωμάτιο άλφα μπορούν να προκαλέσουν μια προσωρινή αστοχία σε ένα κύκλωμα μνήμης, αλλάζοντας την τιμή ενός bit που είναι αποθηκευμένο σε αυτή. Αν επιτρέψουμε σε ένα κύκλωμα να ζεσταθεί υπερβολικά ή αν το "χτυπήσουμε" με στατικό ηλεκτρισμό, ενδέχεται να προκληθεί μια μόνιμη αστοχία και το κύκλωμα να μην ξαναλειτουργήσει ποτέ σωστά.

μοντέλο σφαλμάτων

ανεξάρτητο μοντέλο σφαλμάτων

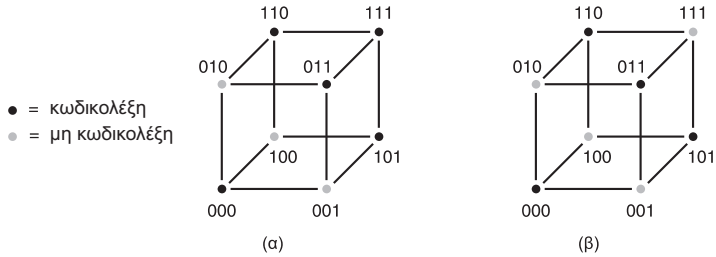
απλό σφάλμα

πολλαπλό σφάλμα

Οι επιπτώσεις της αστοχίας στα δεδομένα είναι δυνατόν να προβλεφθούν από *μοντέλα σφαλμάτων*. Το απλούστερο μοντέλο σφαλμάτων που μπορούμε να εξετάσουμε εδώ ονομάζεται *ανεξάρτητο μοντέλο σφαλμάτων*. Σε αυτό το μοντέλο, μια απλή φυσική αστοχία θεωρείται ότι επηρεάζει μόνο ένα bit από τα δεδομένα. Τα αλλοιωμένα δεδομένα θεωρούνται τότε ότι περιέχουν *ένα απλό σφάλμα*. Πολλαπλές αστοχίες μπορούν να προκαλέσουν *πολλαπλά σφάλματα*, δηλαδή δύο ή περισσότερα εσφαλμένα bit, αλλά τα πολλαπλά σφάλματα συνήθως θεωρούνται λιγότερο πιθανά από ό,τι τα απλά σφάλματα.

2.15.1 Κώδικες ανίχνευσης σφαλμάτων

Θυμηθείτε πάλι τους ορισμούς που δώσαμε στην Ενότητα 2.10 ότι ένας κώδικας που χρησιμοποιεί ακολουθίες των *n*-bit δεν είναι απαραίτητο να περιέχει 2^n έγκυρες κωδικολέξεις. Αυτή είναι ακριβώς η περίπτωση για τους κώδικες με τους οποίους θα ασχοληθούμε σε αυτή την ενότητα.



Εικόνα 2-10
Κωδικολέξεις μέσα σε δύο διαφορετικούς κώδικες 3 bit: (α) ελάχιστη απόσταση = 1, δεν ανιχνεύει όλα τα απλά λάθη
(β) ελάχιστη απόσταση = 2, ανιχνεύει όλα τα απλά λάθη.

Ένας *κώδικας ανίχνευσης σφαλμάτων* έχει την ιδιότητα ότι η αλλοίωση θα παράγει μια ακολουθία από bit που κατά πάσα πιθανότητα δε θα είναι κωδικολέξη (δηλαδή μια *μη κωδικολέξη*).

κώδικας ανίχνευσης σφαλμάτων
μη κωδικολέξη

Ένα σύστημα που χρησιμοποιεί έναν κώδικα ανίχνευσης σφαλμάτων παράγει, μεταδίδει, και αποθηκεύει μόνο κωδικολέξεις. Έτσι, τα τυχόν σφάλματα μέσα σε μια ακολουθία bit είναι δυνατόν να ανιχνευθούν με έναν απλό κανόνα: αν η ακολουθία bit είναι μια κωδικολέξη, τότε γίνεται η υπόθεση ότι είναι σωστή, ενώ αν είναι μια μη κωδικολέξη, τότε αυτή περιέχει σφάλμα.

Ο κώδικας n-bit, καθώς και οι ιδιότητες ανίχνευσης σφαλμάτων στα πλαίσια του ανεξάρτητου μοντέλου σφαλμάτων, εξηγούνται εύκολα με τη βοήθεια των n-διάστατων κύβων. Ο κώδικας είναι απλώς ένα υποσύνολο των κορυφών ενός n-διάστατου κύβου. Για να μπορεί ο κώδικας να ανιχνεύει όλα τα απλά σφάλματα, καμία κορυφή που αντιστοιχεί σε κωδικολέξη δε γειτονεύει άμεσα με άλλη κορυφή που αντιστοιχεί σε κωδικολέξη.

Για παράδειγμα, στην Εικόνα 2-10(α) φαίνεται ένας κώδικας 3-bit με πέντε κωδικολέξεις. Η κωδικολέξη 111 γειτονεύει άμεσα με τις κωδικολέξεις 110, 011 και 101. Εφόσον μια απλή αστοχία μπορεί να αλλάξει το 111 σε 110, 011, ή 101, αυτός ο κώδικας δεν ανιχνεύει όλα τα απλά σφάλματα. Αν καταστήσουμε το 111 μη κωδικολέξη, τότε παίρνουμε έναν κώδικα που έχει την ιδιότητα να ανιχνεύει απλά σφάλματα, όπως φαίνεται στο (β). Κανένα απλό σφάλμα δεν μπορεί να μετατρέψει μια κωδικολέξη σε άλλη.

Την ικανότητα ενός κώδικα να ανιχνεύει απλά σφάλματα μπορεί να την εκφράσει κανείς και ως προς την έννοια της απόστασης την οποία παρουσιάσαμε στην προηγούμενη ενότητα:

- Ένας κώδικας ανιχνεύει όλα τα απλά σφάλματα αν η *ελάχιστη απόσταση* μεταξύ όλων των πιθανών ζευγών κωδικολέξεων είναι 2.

ελάχιστη απόσταση

Γενικά, χρειαζόμαστε n+1 bit για να κατασκευάσουμε έναν κώδικα ανίχνευσης απλών σφαλμάτων με 2^n κωδικολέξεις. Τα πρώτα n bit μιας κωδικολέξης ονομάζονται *bit πληροφορίας* και μπορούν να είναι οποιαδήποτε από τις 2^n ακολουθίες των n bit. Για να προκύψει ένας κώδικας ελάχιστης απόστασης 2, προσθέτουμε ένα ακόμη bit που ονομάζεται *bit ισοτιμίας* και παίρνει την τιμή 0 αν το πλήθος των 1 είναι άρτιος αριθμός και την τιμή 1 σε αντίθετη περίπτωση. Αυτό φαίνεται στις δύο πρώ-

bit πληροφορίας

bit ισοτιμίας

Πίνακας 2-13 Κώδικες απόστασης 2 με τρία bit πληροφορίας.

Bit πληροφορίας	*Κώδικας άρτιας ισοτιμίας*	*Κώδικας περιττής ισοτιμίας*
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

τες στήλες του Πίνακα 2-13 για έναν κώδικα με τρία bit πληροφορίας. Κάθε κωδικολέξη των (n+1) bit έχει άρτιο αριθμό 1, ενώ ο κώδικας αυτός ονομάζεται *κώδικας άρτιας ισοτιμίας*.

κώδικας άρτιας ισοτιμίας

Επίσης μπορούμε να κατασκευάσουμε έναν κώδικα στον οποίο το συνολικό πλήθος των 1 που περιέχονται σε μια έγκυρη κωδικολέξη (n+1) bit να είναι περιττός αριθμός. Ένας τέτοιος κώδικας ονομάζεται *κώδικας περιττής ισοτιμίας* και φαίνεται στην τρίτη στήλη του Πίνακα 2-13. Αυτοί οι κώδικες μερικές φορές ονομάζονται και *κώδικες ισοτιμίας 1-bit*, επειδή ο κάθε ένας χρησιμοποιεί ένα απλό bit ισοτιμίας.

κώδικας περιττής ισοτιμίας

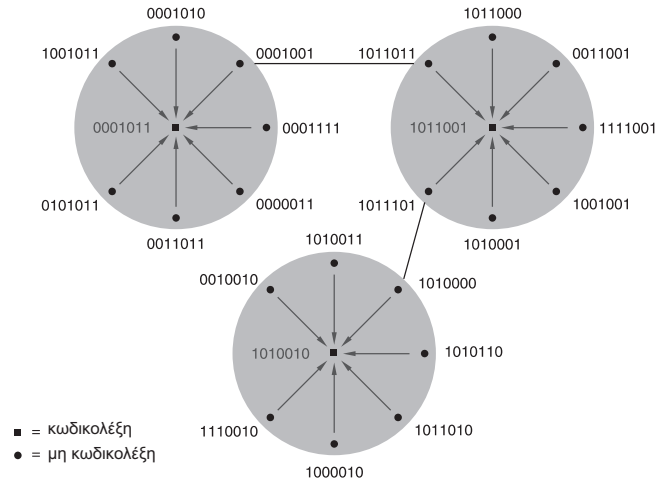
κώδικας ισοτιμίας 1 bit

Οι κώδικες ισοτιμίας 1-bit δεν ανιχνεύουν σφάλματα 2-bit, αφού η τυχόν αλλαγή δύο bit δεν επηρεάζει την ισοτιμία. Ωστόσο, οι κώδικες αυτοί μπορούν να ανιχνεύσουν σφάλματα, αρκεί ο αριθμός των bit να είναι οποιοσδήποτε *περιττός* αριθμός. Για παράδειγμα, αν αλλάξουν τρία bit σε μια κωδικολέξη, τότε η κωδικολέξη που προκύπτει έχει εσφαλμένη ισοτιμία και είναι μια μη κωδικολέξη. Αυτό, ωστόσο, δεν αποτελεί και μεγάλη βοήθεια. Στα πλαίσια του ανεξάρτητου μοντέλου σφαλμάτων, τα σφάλματα των 3-bit είναι πολύ λιγότερο πιθανά από τα σφάλματα των 2-bit, τα οποία δεν είναι ανιχνεύσιμα. Έτσι, από πρακτικής πλευράς, η ικανότητα ανίχνευσης σφαλμάτων των κωδίκων ισοτιμίας 1-bit παύει μετά από σφάλματα του 1-bit. Για την ανίχνευση πολλαπλών σφαλμάτων, μπορούν να χρησιμοποιηθούν άλλοι κώδικες με ελάχιστη απόσταση μεγαλύτερη από 2.

2.15.2 Κώδικες διόρθωσης σφαλμάτων και κώδικες ανίχνευσης πολλαπλών σφαλμάτων

bit ελέγχου

Κάνοντας χρήση περισσοτέρων του ενός bit ισοτιμίας, ή αλλιώς *bit ελέγχου,* και εφαρμόζοντας κάποιους καλά επιλεγμένους κανόνες, μπορούμε να δημιουργήσουμε έναν κώδικα του οποίου η ελάχιστη απόσταση είναι μεγαλύτερη του 2. Προτού δείξουμε όμως τον τρόπο με τον οποίο μπορεί να γίνει αυτό, ας δούμε πώς ένας τέτοιος κώδικας μπορεί να χρησιμοποιηθεί για να διορθώσει απλά σφάλματα ή να ανιχνεύσει πολλαπλά σφάλματα.



Εικόνα 2-11 Μερικές κωδικολέξεις και μη κωδικολέξεις σε έναν κώδικα απόστασης 3, των 7 bit.

Υποθέστε ότι ένας κώδικας έχει ελάχιστη απόσταση ίση με 3. Στην Εικόνα 2-11 φαίνεται ένα μέρος του n-διάστατου κύβου για έναν τέτοιο κώδικα. Όπως φαίνεται, υπάρχουν τουλάχιστον δύο μη κωδικολέξεις ανάμεσα σε κάθε ζευγάρι κωδικολέξεων. Τώρα, υποθέστε ότι μεταδίδουμε κωδικολέξεις και κάντε την υπόθεση ότι οι τυχόν αστοχίες επηρεάζουν το πολύ ένα bit σε κάθε λαμβανόμενη κωδικολέξη. Τότε μια λαμβανόμενη μη κωδικολέξη με ένα εσφαλμένο bit θα είναι πιο κοντά στην αρχικά μεταδιδόμενη κωδικολέξη από ό,τι σε οποιαδήποτε άλλη κωδικολέξη. Συνεπώς, όταν λαμβάνουμε μια μη κωδικολέξη μπορούμε να *διορθώσουμε* το σφάλμα αλλάζοντας τη λαμβανόμενη μη κωδικολέξη στην πλησιέστερη κωδικολέξη όπως δείχνουν τα βέλη στην εικόνα. Η διαδικασία με την οποία αποφασίζουμε ποια κωδικολέξη είχε αρχικά μεταδοθεί με βάση μια λαμβανόμενη λέξη ονομάζεται *αποκωδικοποίηση* και το υλικό που την υλοποιεί ονομάζεται *αποκωδικοποιητής διόρθωσης σφαλμάτων*.

διόρθωση σφαλμάτων

αποκωδικοποίηση
αποκωδικοποιητής

Ο κώδικας που χρησιμοποιείται για τη διόρθωση σφαλμάτων ονομάζεται *κώδικας διόρθωσης σφαλμάτων*. Γενικά, αν ένας κώδικας έχει ελάχιστη απόσταση $2c + 1$, αυτός μπορεί να χρησιμοποιηθεί για να διορθώσει σφάλματα που επηρεάζουν έως και c bit ($c = 1$ στο προηγούμενο παράδειγμα). Αν τώρα η ελάχιστη απόσταση ενός κώδικα είναι $2c + d + 1$, ο κώδικας αυτός μπορεί να χρησιμοποιηθεί για να διορθώσει σφάλματα έως και σε c bit και να ανιχνεύσει σφάλματα έως και σε d επιπλέον bit.

κώδικας διόρθωσης σφαλμάτων

Για παράδειγμα, στην Εικόνα 2-12(α) φαίνεται ένα μέρος του n-διάστατου κύβου για έναν κώδικα με ελάχιστη απόσταση ίση με 4 ($c = 1$, $d = 1$). Τα σφάλματα ενός bit τα οποία παράγουν τις μη κωδικολέξεις 00101010 και 11010011 είναι δυνατόν να διορθωθούν. Ωστόσο, ένα σφάλμα που παράγει τη λέξη 10100011 δεν είναι δυνατόν να διορθωθεί, επειδή κανένα σφάλμα ενός bit δεν παράγει αυτή τη μη κωδικολέξη, ενώ υπάρχουν δύο σφάλματα των 2-bit που είναι δυνατόν να την έχουν πα-

ραγάγει. Επομένως, ο κώδικας μπορεί να ανιχνεύσει ένα σφάλμα των 2-bit, αλλά δεν μπορεί να το διορθώσει.

Όταν λαμβάνουμε μια μη κωδικολέξη, δε γνωρίζουμε ποια κωδικολέξη είχε μεταδοθεί αρχικά. Το μόνο που γνωρίζουμε είναι ποια κωδικολέξη βρίσκεται πλησιέστερα σε εκείνη που έχουμε λάβει. Κατά συνέπεια, όπως φαίνεται και στην Εικόνα 2-12(β), ένα σφάλμα των 3-bit ενδέχεται να "διορθωθεί" σε μια εσφαλμένη τιμή. Η πιθανότητα να γίνει ένα τέτοιου είδους λάθος μπορεί να είναι αποδεκτή, αν η εμφάνιση σφαλμάτων των 3-bit θεωρείται πολύ απίθανη. Από την άλλη μεριά, αν μας ενδιαφέρουν τα σφάλματα των 3-bit, τότε μπορούμε να αλλάξουμε τον τρόπο αποκωδικοποίησης του κώδικα. Αντί να επιχειρούμε να διορθώσουμε τα σφάλματα, αρκεί απλώς να προσημειώσουμε όλες τις μη κωδικολέξεις ως μη διορθώσιμα σφάλματα. Έτσι, όπως φαίνεται και στο (γ), μπορούμε να χρησιμοποιήσουμε τον ίδιο κώδικα απόστασης 4 για να ανιχνεύσουμε σφάλματα έως και των 3 bit, αλλά δε θα διορθώνουμε τα σφάλματα ($c = 0$, $d = 3$).

2.15.3 Κώδικες Hamming

Το 1950 ο R.W. Hamming περιέγραψε μια γενική μέθοδο για την κατασκευή κωδίκων με ελάχιστη απόσταση ίση με 3, οι οποίοι ονομάζονται πλέον *κώδικες Hamming*. Για οποιαδήποτε τιμή του i, η μέθοδος αυτή έχει αποτέλεσμα έναν κώδικα (2^i - 1) bit με i bit ελέγχου και (2^i - 1 - i) bit πληροφορίας. Οι κώδικες απόστασης 3 με μικρότερο αριθμό bit πληροφορίας λαμβάνονται με διαγραφή bit πληροφορίας από έναν κώδικα Hamming με μεγαλύτερο αριθμό bit.

κώδικας Hamming

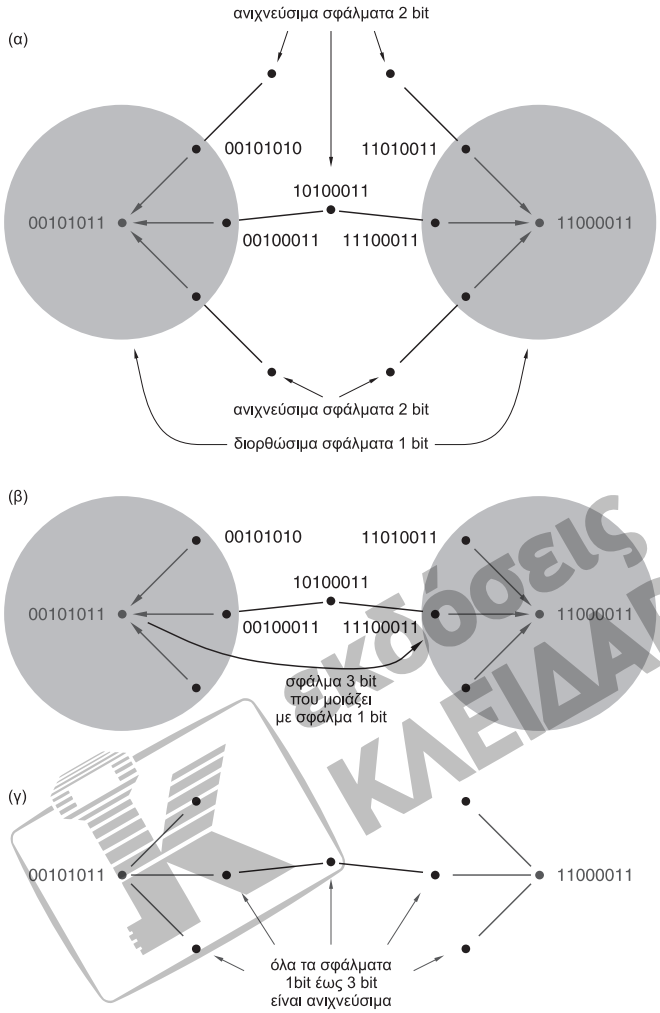
Οι θέσεις των bit σε μια κωδικολέξη Hamming αριθμούνται από το 1 έως και το 2^i - 1. Σε αυτή την περίπτωση, κάθε θέση της οποίας ο αριθμός είναι δύναμη του 2 αποτελεί bit ελέγχου, ενώ οι υπόλοιπες θέσεις αποτελούν bit πληροφορίας. Κάθε bit ελέγχου ομαδοποιείται μαζί με ένα υποσύνολο των bit πληροφορίας, όπως καθορίζεται σε έναν *πίνακα ελέγχου ισοτιμίας*.

πίνακας ελέγχου ισοτιμίας

ΑΠΟΦΑΣΕΙΣ, ΑΠΟΦΑΣΕΙΣ

Οι όροι *αποκωδικοποίηση* και *αποκωδικοποιητής* έχουν νόημα, εφόσον αυτοί είναι απλώς αλλοιώσεις απόστασης 1 των λέξεων *λήψη αποφάσεων* και *λήπτης αποφάσεων*. Η δήλωση αυτή είναι λογοπαίγνιο διότι στα αγγλικά οι λέξεις *decoding* και *decoder* διαφέρουν κατά ακριβώς ένα χαρακτήρα από τις *deciding* και *decider* αντίστοιχα.

Όπως φαίνεται και στην Εικόνα 2-13(α), κάθε bit ελέγχου ομαδοποιείται μαζί με τις θέσεις πληροφορίας των οποίων οι αριθμοί έχουν 1 στο ίδιο bit όταν εκφράζονται ως δυαδικοί. Για παράδειγμα, το bit ελέγχου 2 (010) ομαδοποιείται μαζί με τα bit πληροφορίας 3 (011), 6 (110) και 7 (111). Για ένα δεδομένο συνδυασμό τιμών των bit πληροφορίας, κάθε bit ελέγχου επιλέγεται με τρόπο ώστε να δίνει άρτια ισοτιμία, έτσι ώστε ο συνολικός αριθμός των 1 στην ομάδα του να είναι άρτιος.

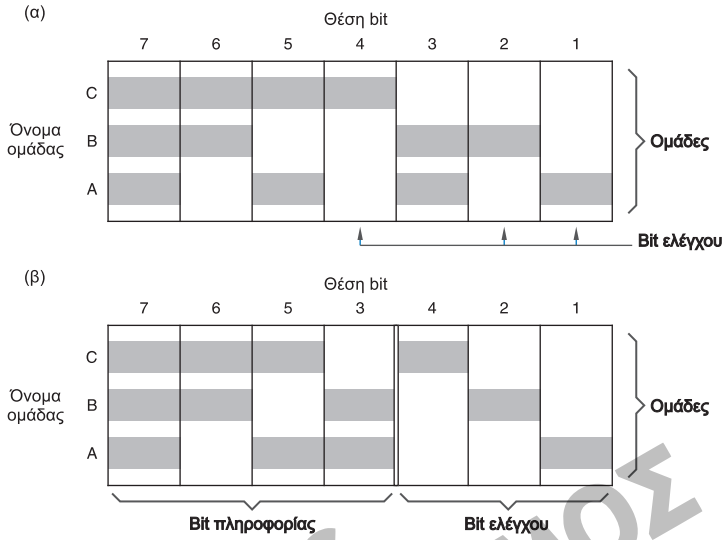


Εικόνα 2-12
Κάποιες κωδικολέξεις και μη κωδικολέξεις σε έναν κώδικα απόστασης 4 των 8 bit:
(α) διόρθωση σφαλμάτων 1 bit και ανίχνευση σφαλμάτων 2 bit
(β) εσφαλμένη "διόρθωση" ενός σφάλματος 3 bit
(γ) χωρίς διόρθωση σφαλμάτων αλλά με ανίχνευση έως και σφαλμάτων 3 bit.

Συνήθως οι θέσεις των bit ενός πίνακα ελέγχου ισοτιμίας και οι κωδικολέξεις που προκύπτουν αναδιατάσσονται έτσι ώστε όλα τα bit ελέγχου να είναι στα δεξιά, όπως στην Εικόνα 2-13(β). Οι πρώτες δύο στήλες του Πίνακα 2-14 περιέχουν τις κωδικολέξεις που προκύπτουν.

Μπορούμε να αποδείξουμε ότι η ελάχιστη απόσταση ενός κώδικα Hamming είναι 3 αποδεικνύοντας ότι πρέπει να γίνει τουλάχιστον μία αλλαγή 3 bit σε μια κωδικολέξη για να προκύψει μια άλλη κωδικολέξη. Θα αποδείξουμε δηλαδή ότι μια αλλαγή 1-bit ή 2-bit σε μια κωδικολέξη καταλήγει σε μια μη κωδικολέξη.

Αν αλλάξουμε ένα bit μιας κωδικολέξης, στη θέση j, τότε αλλάζουμε την ισοτιμία κάθε ομάδας που περιέχει τη θέση j. Αφού κάθε bit πληροφορίας περιλαμβάνεται σε τουλάχιστον μία ομάδα, τότε τουλάχιστον μία



Εικόνα 2-13 Πίνακες ελέγχου ισοτιμίας για κώδικες Hamming των 7 bit: (α) με τις θέσεις των bit σε αριθμητική σειρά (β) με τα bit ελέγχου ξεχωριστά από τα bit πληροφορίας

ομάδα θα έχει εσφαλμένη ισοτιμία και το αποτέλεσμα θα είναι μια μη κωδικολέξη.

Τι συμβαίνει αν αλλάξουμε δύο bit στις θέσεις *j* και *k*; Οι ομάδες ισοτιμίας που περιλαμβάνουν και τις δύο θέσεις *j* και *k* θα εξακολουθούν να έχουν σωστή ισοτιμία, εφόσον η ισοτιμία δεν επηρεάζεται όταν αλλάζει άρτιο πλήθος bit. Ωστόσο, επειδή τα *j* και *k* είναι διαφορετικά, οι δυαδικές αναπαραστάσεις τους διαφέρουν τουλάχιστον κατά ένα bit, που αντιστοιχεί σε μία από τις ομάδας ισοτιμίας. Αυτή η ομάδα παρουσιάζει αλλαγή μόνο σε ένα bit, με αποτέλεσμα την εσφαλμένη ισοτιμία και μια μη κωδικολέξη.

Αν κατανοήσατε αυτή την απόδειξη, θα πρέπει επίσης να έχετε κατανοήσει πώς οι κανόνες αρίθμησης θέσης για την κατασκευή ενός κώδικα Hamming είναι λογικό επακόλουθο αυτής της απόδειξης. Στο πρώτο μέρος της απόδειξης (σφάλματα 1-bit), υποθέσαμε ότι οι αριθμοί θέσεων είναι μη μηδενικοί. Και στο δεύτερο μέρος (σφάλματα 2-bit), υποθέσαμε ότι δύο θέσεις δεν είναι δυνατόν να έχουν τον ίδιο αριθμό. Κατά συνέπεια, με έναν αριθμό θέσεων *i*-bit, μπορείτε να κατασκευάσετε έναν κώδικα Hamming που θα έχει έως και $2^i - 1$ θέσεις bit.

αποκωδικοποιητής διόρθωσης σφαλμάτων

σύνδρομο

Η απόδειξη αυτή δείχνει επίσης πώς μπορούμε να σχεδιάσουμε έναν *αποκωδικοποιητή διόρθωσης σφαλμάτων* για μια κωδικολέξη Hamming που έχουμε λάβει. Πρώτα ελέγχουμε όλες τις ομάδες ισοτιμίας: αν όλες έχουν άρτια ισοτιμία, τότε η λέξη που έχει ληφθεί θεωρείται σωστή. Αν μία ή περισσότερες ομάδες έχουν περιττή ισοτιμία, τότε θεωρείται ότι έχει ανακύψει ένα απλό σφάλμα. Το υπόδειγμα των ομάδων που έχουν περιττή ισοτιμία (το οποίο ονομάζεται *σύνδρομο*) πρέπει να συμφωνεί με μία από τις στήλες του πίνακα ελέγχου ισοτιμίας. Η αντίστοιχη θέση bit θεωρείται ότι περιέχει τη λάθος τιμή και υπολογίζεται το συμπλήρωμά της. Για παράδειγμα, χρησιμοποιώντας τον κώδικα που ορίζεται στην

Πίνακας 2-14 Κωδικολέξεις απόστασης 3 και απόστασης 4 μέσα σε κώδικες Hamming με τέσσερα bit πληροφορίας.

Κώδικας ελάχιστης απόστασης 3		*Κώδικας ελάχιστης απόστασης 4*	
Bit πληροφορίας	***Bit ισοτιμίας***	***Bit πληροφορίας***	***Bit ισοτιμίας***
0000	000	0000	0000
0001	011	0001	0111
0010	101	0010	1011
0011	110	0011	1100
0100	110	0100	1101
0101	101	0101	1010
0110	011	0110	0110
0111	000	0111	0001
1000	111	1000	1110
1001	100	1001	1001
1010	010	1010	0101
1011	001	1011	0010
1100	001	1100	0011
1101	010	1101	0100
1110	100	1110	1000
1111	111	1111	1111

Εικόνα 2-13(β), υποθέστε ότι λαμβάνουμε τη λέξη 0101011. Οι ομάδες B και C έχουν περιττή ισοτιμία, που αντιστοιχεί στη θέση 6 του πίνακα ελέγχου ισοτιμίας (το σύνδρομο είναι 110, δηλαδή 6). Υπολογίζοντας το συμπλήρωμα του bit στη θέση 6 της λέξης που έχει ληφθεί, διαπιστώνουμε ότι η σωστή λέξη είναι η 0001011.

Ένας κώδικας Hamming απόστασης 3 μπορεί εύκολα να τροποποιηθεί για την αύξηση της ελάχιστης απόστασής του σε 4. Απλώς προσθέτουμε ένα ακόμη bit ελέγχου, επιλεγμένο έτσι ώστε η ισοτιμία όλων των bit συμπεριλαμβανομένου και του καινούργιου να είναι άρτια. Όπως και στον κώδικα άρτιας ισοτιμίας 1-bit, αυτό το bit εξασφαλίζει ότι όλα τα σφάλματα που επηρεάζουν έναν περιττό αριθμό bit είναι ανιχνεύσιμα. Πιο συγκεκριμένα, κάθε σφάλμα 3-bit είναι ανιχνεύσιμο. Δείξαμε ήδη ότι τα σφάλματα 1- και 2-bit ανιχνεύονται από τα υπόλοιπα bit ισοτιμίας,

Πίνακας 2-15 Μεγέθη λέξεων σε κώδικες Hamming απόστασης 3 και 4.

	Κώδικες ελάχιστης απόστασης 3		*Κώδικες ελάχιστης απόστασης 4*	
Bit πληροφορίας	***Bit ισοτιμίας***	***Συνολικά Bit***	***Bit Ισοτιμίας***	***Συνολικά Bit***
1	2	3	3	4
≤ 4	3	≤ 7	4	≤ 8
≤ 11	4	≤ 15	5	≤ 16
≤ 26	5	≤ 31	6	≤ 32
≤ 57	6	≤ 63	7	≤ 64
≤ 120	7	≤ 127	8	≤ 128

συνεπώς η ελάχιστη απόσταση του τροποποιημένου κώδικα πρέπει να είναι 4.

Οι κώδικες Hamming απόστασης 3 και 4 χρησιμοποιούνται ευρέως για την ανίχνευση και διόρθωση σφαλμάτων στα συστήματα μνήμης των υπολογιστών, και ιδιαίτερα στα μεγάλα συστήματα υπολογιστών (mainframe) όπου τα κυκλώματα μνήμης παρουσιάζουν το μεγαλύτερο μέρος των αστοχιών του συστήματος. Αυτοί οι κώδικες είναι ιδιαίτερα βολικοί για λέξεις μνήμης μεγάλου εύρους, επειδή ο απαιτούμενος αριθμός bit ισοτιμίας αυξάνει αργά ως προς το εύρος της λέξης μνήμης, όπως φαίνεται στον Πίνακα 2-15.

2.15.4 Κώδικες CRC

κώδικας κυκλικού ελέγχου πλεονασμού (CRC)

Εκτός από τους κώδικες Hamming, έχουν αναπτυχθεί και πολλοί άλλοι κώδικες ανίχνευσης και διόρθωσης σφαλμάτων. Οι πιο σημαντικοί κώδικες, που μάλιστα περιλαμβάνουν και τους κώδικες Hamming, είναι οι *κώδικες κυκλικού ελέγχου πλεονασμού* (*CRC Cyclic Redundancy Check*). Έχει αναπτυχθεί εκτενής θεωρία γι' αυτούς τους κώδικες, με επίκεντρο τόσο τις ιδιότητες ανίχνευσης και διόρθωσης όσο και τη σχεδίαση οικονομικών κωδικοποιητών και αποκωδικοποιητών για τέτοιους κώδικες (δείτε τις Παραπομπές).

Δύο σημαντικές εφαρμογές των κωδίκων CRC βρίσκονται στις μονάδες δίσκου και στα δίκτυα δεδομένων. Σε μια μονάδα δίσκου, κάθε μπλοκ δεδομένων (συνήθως 512 byte) προστατεύεται από έναν κώδικα CRC, έτσι ώστε τα σφάλματα μέσα σε ένα μπλοκ να είναι δυνατόν να ανιχνευθούν και, σε μερικές μονάδες, να διορθωθούν. Σε ένα δίκτυο δεδομένων, κάθε πακέτο δεδομένων τελειώνει με bit ελέγχου σε κώδικα CRC. Οι κώδικες CRC και για τις δύο αυτές εφαρμογές έχουν επιλεγεί για τις ιδιότητες τους να ανιχνεύουν σφάλματα ριπής. Εκτός από σφάλματα ενός bit, μπορούν να ανιχνεύουν σφάλματα πολλών bit που συστοιχίζονται μέσα στα μπλοκ του δίσκου ή του πακέτου. Τέτοια σφάλματα είναι πιο πιθανά από τα σφάλματα τυχαία κατανεμημένων bit, εξαιτίας των πιθανών φυσικών αιτίων των σφαλμάτων και στις δύο εφαρμογές, δηλαδή ελαττώματα στην επιφάνεια των μονάδων δίσκων και ριπές θορύβου σε συνδέσμους επικοινωνίας.

2.15.5 Διδιάστατοι κώδικες

διδιάστατος κώδικας

Ένας άλλος τρόπος προκειμένου να πάρουμε έναν κώδικα μεγάλης ελάχιστης απόστασης είναι να κατασκευάσουμε ένα *διδιάστατο κώδικα*, όπως φαίνεται στην Εικόνα 2-14(α). Τα bit πληροφορίας θεωρητικά διατάσσονται σε ένα διδιάστατο πίνακα, ενώ τα bit ισοτιμίας ελέγχουν τόσο τις γραμμές όσο και τις στήλες. Για τις γραμμές χρησιμοποιείται ένας κώδικας C_{row} ελάχιστης απόστασης d_{row}, ενώ για τις στήλες χρησιμοποιείται ο ενδεχομένως διαφορετικός κώδικας C_{col} ελάχιστης απόστασης d_{col}. Με άλλα λόγια, τα bit ισοτιμίας γραμμής επιλέγονται έτσι ώστε κάθε γραμμή να είναι μια κωδικολέξη στον κώδικα C_{row}, ενώ τα bit ισοτι-

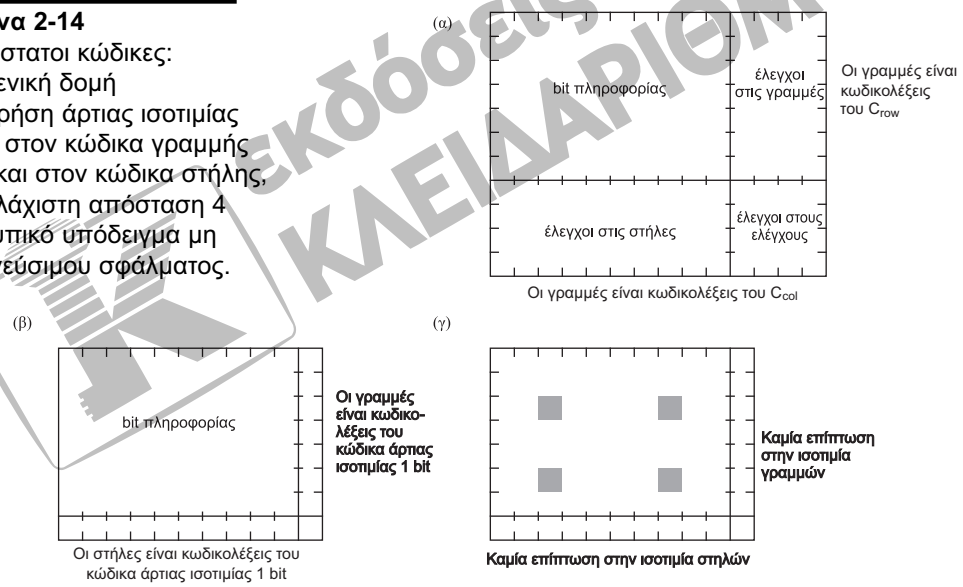
μίας στήλης επιλέγονται έτσι ώστε κάθε στήλη να είναι μια κωδικολέξη στον κώδικα C_{col}. (Τα "γωνιακά" bit ισοτιμίας μπορούν να επιλεγούν σύμφωνα με οποιονδήποτε από τους δύο κώδικες.) Η ελάχιστη απόσταση ενός διδιάστατου κώδικα είναι το γινόμενο του d_{row} επί το d_{col}. Μερικές φορές μάλιστα, οι διδιάστατοι κώδικες αποκαλούνται και *κώδικες γινομένου*.

κώδικας γινομένου

Όπως φαίνεται και στην Εικόνα 2-14(β), ο απλούστερος διδιάστατος κώδικας χρησιμοποιεί κώδικες άρτιας ισοτιμίας του 1-bit τόσο για τις γραμμές όσο και για τις στήλες και έχει ελάχιστη απόσταση ίση με $2 \cdot 2$ δηλαδή 4.

Μπορείτε εύκολα να αποδείξετε ότι η ελάχιστη απόσταση είναι 4 αν θεωρήσετε ότι κάθε υπόδειγμα ενός, δύο, ή και τριών εσφαλμένων bit προκαλεί εσφαλμένη ισοτιμία σε μια γραμμή ή στήλη ή και στις δύο. Για να πάρουμε ένα μη ανιχνεύσιμο σφάλμα, πρέπει να αλλάξουν τουλάχιστον τέσσερα bit σε ένα ορθογώνιο υπόδειγμα, όπως π.χ. στο (γ).

Εικόνα 2-14
Διδιάστατοι κώδικες:
(α) γενική δομή
(β) χρήση άρτιας ισοτιμίας τόσο στον κώδικα γραμμής όσο και στον κώδικα στήλης, για ελάχιστη απόσταση 4
(γ) τυπικό υπόδειγμα μη ανιχνεύσιμου σφάλματος.



Οι διαδικασίες ανίχνευσης και διόρθωσης σφαλμάτων ενός τέτοιου κώδικα προκύπτουν εύκολα. Ας υποθέσουμε ότι διαβάζουμε τις πληροφορίες μία-μία γραμμή. Καθώς διαβάζουμε την κάθε γραμμή, ελέγχουμε τον κώδικα γραμμής της. Αν ανιχνευθεί σφάλμα σε μια γραμμή, δεν μπορούμε να πούμε ποιο bit είναι λάθος μόνο και μόνο με τον έλεγχο της γραμμής. Όμως, αν υποθέσουμε ότι μόνο μία γραμμή έχει σφάλμα, μπορούμε να την ανακατασκευάσουμε υπολογίζοντας την αποκλειστική διάζευξη (EOR) bit-προς-bit των στηλών, παραλείποντας την εσφαλμένη γραμμή αλλά συμπεριλαμβάνοντας τη γραμμή ελέγχου στήλης.

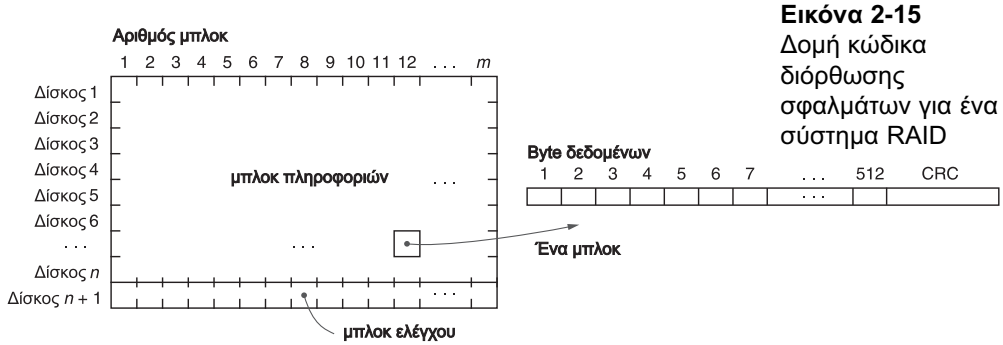
Για να πάρουμε ακόμη μεγαλύτερη ελάχιστη απόσταση, είναι δυνατόν να χρησιμοποιηθεί ένας κώδικας Hamming απόστασης 3 ή και 4 ως κώδικας γραμμής ή στήλης ή και των δύο. Είναι επίσης δυνατόν να κατασκευάσουμε έναν κώδικα τριών ή περισσότερων διαστάσεων, ο οποίος θα έχει ελάχιστη απόσταση ίση με το γινόμενο των ελαχίστων αποστάσεων σε κάθε διάσταση.

RAID

Μια σημαντική εφαρμογή των διδιάστατων κωδίκων είναι στα συστήματα αποθήκευσης RAID. Το ακρώνυμο *RAID* σημαίνει "Redundant Array of Inexpensive Disks" (πλεονάζουσα συστοιχία φθηνών δίσκων). Με αυτή τη μέθοδο, χρησιμοποιούνται $n + 1$ μονάδες δίσκου για να αποθηκεύσουν όγκο δεδομένων που χωρούν σε n δίσκους. Για παράδειγμα, οκτώ δίσκοι των 8 Gb είναι δυνατόν να χρησιμοποιηθούν για να αποθηκεύσουν 64-Gb μη πλεοναζόντων δεδομένων, ενώ ο ένατος δίσκος των 8-Gb θα χρησιμοποιηθεί για να αποθηκεύει πληροφορίες ελέγχου.

Στην Εικόνα 2-15 δίνεται ο γενικός συνδυασμός ενός διδιάστατου κώδικα για ένα σύστημα RAID. Κάθε μονάδα δίσκου θεωρείται ως μια γραμμή στον κώδικα. Κάθε μονάδα αποθηκεύει m μπλοκ δεδομένων, όπου κάθε μπλοκ περιέχει 512 byte. Για παράδειγμα, μια μονάδα των 8 Gb μπορεί να αποθηκεύσει 16 εκατομμύρια μπλοκ περίπου. Όπως φαίνεται και στην εικόνα, κάθε μπλοκ συμπεριλαμβάνει τα δικά του bit ελέγχου σε έναν κώδικα CRC, για να ανιχνεύει σφάλματα μέσα στο συγκεκριμένο μπλοκ. Οι πρώτες n μονάδες αποθηκεύουν τα μη πλεονάζοντα δεδομένα. Κάθε μπλοκ της μονάδας n +1 αποθηκεύει bit ισοτιμίας των αντίστοιχων μπλοκ που έχουν αποθηκευτεί στις πρώτες n μονάδες. Με άλλα λόγια, κάθε bit i της μονάδας n +1, στο μπλοκ b, επιλέγεται έτσι ώστε να υπάρχει άρτιο πλήθος 1 στο μπλοκ b, στη θέση bit i, σε όλες τις μονάδες.

Κατά τη λειτουργία, τα τυχόν σφάλματα στα μπλοκ πληροφοριών ανιχνεύονται από τον κώδικα CRC. Κάθε φορά που ανιχνεύεται ένα σφάλμα σε ένα μπλοκ σε μία από τις μονάδες, το σωστό περιεχόμενο αυτού του μπλοκ είναι δυνατόν να ανακατασκευαστεί απλώς με τον υπολογισμό της ισοτιμίας των αντίστοιχων μπλοκ σε όλες τις άλλες μονάδες, συμπεριλαμβανομένης και της μονάδας n + 1. Παρά το γεγονός ότι αυτή η διαδικασία απαιτεί n παραπάνω λειτουργίες ανάγνωσης δίσκου, είναι κα-



Εικόνα 2-15 Δομή κώδικα διόρθωσης σφαλμάτων για ένα σύστημα RAID

λύτερη από την απώλεια των δεδομένων σας! Οι λειτουργίες εγγραφής απαιτούν και αυτές περισσότερες προσπελάσεις του δίσκου, για την ενημέρωση του αντίστοιχου μπλοκ ελέγχου όταν εγγράφεται ένα μπλοκ πληροφοριών (δείτε την Άσκηση 2.46). Επειδή οι εγγραφές στο δίσκο είναι πολύ λιγότερο συχνές από τις αναγνώσεις στις συνήθεις εφαρμογές, αυτή η επιβάρυνση συνήθως δεν αποτελεί πρόβλημα.

2.15.6 Κώδικες αθροίσματος ελέγχου

Η πράξη ελέγχου ισοτιμίας που χρησιμοποιήσαμε στις προηγούμενες ενότητες είναι ουσιαστικά πρόσθεση modulo-2 των bit: το άθροισμα modulo-2 μιας ομάδας bit ισούται με 0 αν το πλήθος των 1 μέσα στην ομάδα είναι άρτιο ή 1 αν το πλήθος των 1 είναι περιττό. Η τεχνική της πρόσθεσης modulo μπορεί να επεκταθεί και σε άλλες βάσεις εκτός από το 2, για το σχηματισμό ψηφίων ελέγχου.

Για παράδειγμα, ένα υπολογιστής αποθηκεύει τις πληροφορίες σε σύνολα byte των 8-bit. Κάθε byte μπορεί να θεωρηθεί ότι έχει μια δεκαδική τιμή από το 0 έως το 255. Επομένως, για να ελέγχουμε τα byte, μπορούμε να χρησιμοποιήσουμε την πράξη πρόσθεσης modulo-256. Φτιάχνουμε λοιπόν ένα απλό byte ελέγχου, το *άθροισμα ελέγχου*, το οποίο είναι το άθροισμα modulo-256 όλων των byte πληροφορίας. Ο προκύπτων *κώδικας αθροίσματος ελέγχου* μπορεί να ανιχνεύσει οποιοδήποτε απλό εσφαλμένο byte, αφού ένα τέτοιο σφάλμα θα έχει αποτέλεσμα ότι το επανυπολογισμένο άθροισμα των byte δεν θα συμφωνεί με το άθροισμα ελέγχου.

άθροισμα ελέγχου

κώδικας αθροίσματος ελέγχου

Οι κώδικες αθροίσματος ελέγχου είναι επίσης δυνατόν να χρησιμοποιούν διαφορετικό υπόλοιπο. Πιο συγκεκριμένα, οι κώδικες αθροίσματος ελέγχου που χρησιμοποιούν πρόσθεση modulo-255 αριθμών συμπληρώματος ως προς ένα είναι σημαντικοί εξαιτίας των ιδιαίτερων ιδιοτήτων υπολογισμού και ανίχνευσης σφαλμάτων που διαθέτουν, καθώς και λόγω της χρήσης τους για τον έλεγχο των κεφαλίδων των πακέτων στο ευρέως διαδεδομένο Πρωτόκολλο του Internet (IP — δείτε τις Παραπομπές).

κώδικας αθροίσματος ελέγχου συμπληρώματος ως προς ένα

2.15.7 Κώδικες *m* από *n*

Οι κώδικες “1 από *n*” και “*m* από *n*” που είδαμε στην Ενότητα 2.13 έχουν ελάχιστη απόσταση 2, αφού η αλλαγή ενός μόνο bit προκαλεί αλλαγές στο συνολικό αριθμό των 1 σε μια κωδικολέξη και συνεπώς έχει αποτέλεσμα την παραγωγή μιας μη κωδικολέξης.

Αυτοί οι κώδικες έχουν μια επιπλέον χρήσιμη ιδιότητα ανίχνευσης σφαλμάτων: ανιχνεύουν τα μονοκατευθυντικά πολλαπλά σφάλματα. Σε ένα *μονοκατευθυντικό πολλαπλό σφάλμα*, όλα τα εσφαλμένα bit αλλάζουν προς στην ίδια κατεύθυνση (τα 0 αλλάζουν σε 1 ή το αντίθετο). Αυτή η ιδιότητα είναι πολύ χρήσιμη σε συστήματα όπου ο κυρίαρχος μηχανισμός σφαλμάτων τείνει να αλλάζει όλα τα bit προς την ίδια κατεύθυνση.

μονοκατευθυντικό πολλαπλό σφάλμα

2.16 Κώδικες για σειριακή μετάδοση και αποθήκευση

2.16.1 Παράλληλα και σειριακά δεδομένα

παράλληλα δεδομένα

Οι περισσότεροι υπολογιστές και άλλα ψηφιακά συστήματα μεταδίδουν και αποθηκεύουν δεδομένα σε *παράλληλη* μορφή. Στην παράλληλη μετάδοση δεδομένων, παρέχεται μια ανεξάρτητη γραμμή σήματος για κάθε bit μιας λέξης δεδομένων. Στην παράλληλη αποθήκευση δεδομένων, όλα τα bit μιας λέξης δεδομένων μπορούν να εγγραφούν ή να αναγνωστούν ταυτόχρονα.

Οι παράλληλες μορφές δεν είναι και τόσο οικονομικές για κάποιες εφαρμογές. Για παράδειγμα, η παράλληλη μετάδοση byte δεδομένων μέσω τηλεφωνικού δικτύου θα απαιτούσε οκτώ τηλεφωνικές γραμμές, ενώ η παράλληλη αποθήκευση byte δεδομένων σε ένα μαγνητικό δίσκο θα απαιτούσε μονάδα δίσκου με οκτώ διαφορετικές κεφαλές ανάγνωσης/εγγραφής. Οι *σειριακές* μορφές επιτρέπουν τη μετάδοση και αποθήκευση δεδομένων ένα-ένα bit, ελαττώνοντας έτσι το κόστος του συστήματος σε πολλές εφαρμογές.

σειριακά δεδομένα

Στην Εικόνα 2-16 δίνονται κάποιες από τις βασικές αρχές της σειριακής μετάδοσης δεδομένων. Ένα επαναλαμβανόμενο σήμα χρονισμού, το ονομαζόμενο CLOCK (ρολόι) στην εικόνα, καθορίζει το ρυθμό με τον οποίο μεταδίδονται τα δεδομένα, ένα bit για κάθε κύκλο του ρολογιού. Έτσι ο *ρυθμός μεταφοράς bit*, σε bit ανά δευτερόλεπτο (bps), ισούται αριθμητικά με τη συχνότητα του ρολογιού σε κύκλους ανά δευτερόλεπτο (Hertz ή Hz).

ρυθμός μεταφοράς bit, bps

διάρκεια bit

Ο αντίστροφος του ρυθμού μεταφοράς bit λέγεται *διάρκεια του bit* και ισούται αριθμητικά με την περίοδο του ρολογιού, σε δευτερόλεπτα (*s*). Αυτή η ποσότητα χρόνου δεσμεύεται στη γραμμή σειριακών δεδομένων (που στην εικόνα ονομάζεται SERDATA) για κάθε bit που μεταδίδεται. Ο χρόνος που καταλαμβάνει το κάθε bit ονομάζεται μερικές φορές και *κυψέλη του bit*. Η μορφή του πραγματικού σήματος που εμφανίζεται πάνω στη γραμμή κατά τη διάρκεια της κάθε κυψέλης bit εξαρτάται από τον *κώδικα γραμμής*. Στον απλούστερο κώδικα γραμμής, που λέγεται "*χωρίς επιστροφή στο μηδέν*" (*Non-Return-to-Zero*, *NRZ*), τα 1 μεταδίδονται με την τοποθέτησης ενός 1 πάνω στη γραμμή για όλο το διάστημα της κυψέλης bit, ενώ τα 0 μεταδίδονται ως 0. Πιο πολύπλοκοι κώδικες έχουν άλλους κανόνες που παρουσιάζονται στην επόμενη ενότητα.

κυψέλη bit

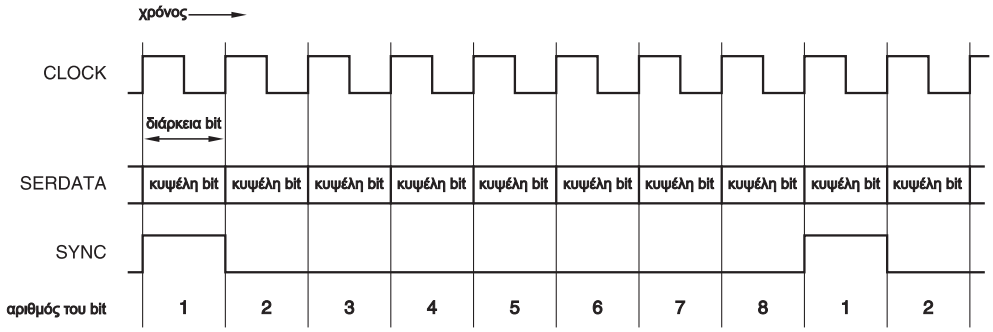
κώδικας γραμμής χωρίς επιστροφή στο μηδέν (NRZ)

Ανεξάρτητα από τον κώδικα γραμμής, ένα σύστημα σειριακής μετάδοσης ή αποθήκευσης δεδομένων χρειάζεται κάποιον τρόπο για να αναγνωρίσει τη σημαντικότητα του κάθε bit του σειριακού ρεύματος δεδομένων. Για παράδειγμα, υποθέστε ότι byte των 8-bit μεταδίδονται σειριακά. Πώς μπορούμε να αναγνωρίσουμε το πρώτο bit του κάθε byte; Ένα *σήμα συγχρονισμού*, το SYNC στην Εικόνα 2-16, παρέχει την απαιτούμενη πληροφορία και έχει την τιμή 1 για το πρώτο bit κάθε byte.

σήμα συγχρονισμού

Προφανώς, χρειαζόμαστε τουλάχιστον τρία σήματα για να ανακτήσουμε ένα σειριακό ρεύμα δεδομένων: ένα σήμα χρονισμού για να κα-

Εικόνα 2-16 Βασικές έννοιες στη σειριακή μετάδοση δεδομένων.



θορίσουμε τις κυψέλες bit, ένα σήμα συγχρονισμού για να καθορίσουμε τα όρια των λέξεων, και τα ίδια τα σειριακά δεδομένα. Σε κάποιες εφαρμογές, όπως για παράδειγμα στη διασύνδεση των υπομονάδων ενός υπολογιστή ή ενός συστήματος τηλεπικοινωνιών, χρησιμοποιείται ένας ξεχωριστός αγωγός για κάθε ένα από τα σήματα αυτά, γεγονός που ελαττώνει τον αριθμό των αγωγών ανά σύνδεση από *n* σε τρία. Στην Ενότητα 8.5.4 θα δώσουμε ένα παράδειγμα ενός σειριακού συστήματος δεδομένων 3 αγωγών.

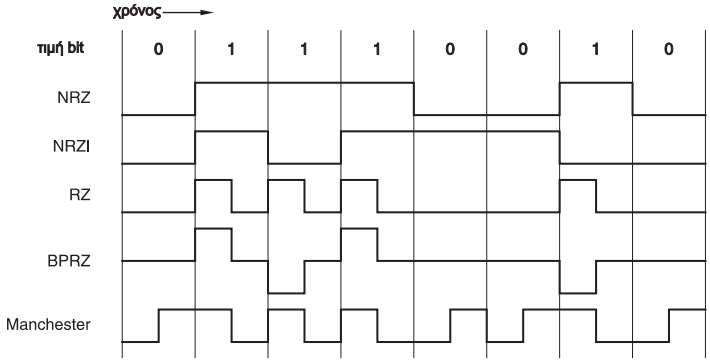
Σε πολλές εφαρμογές, το κόστος διάθεσης τριών ξεχωριστών σημάτων εξακολουθεί να είναι υπερβολικά υψηλό (π.χ. τρεις τηλεφωνικές γραμμές, τρεις κεφαλές ανάγνωσης/εγγραφής). Αυτού του τύπου τα συστήματα συνήθως συνδυάζουν και τα τρία σήματα σε ένα σειριακό ρεύμα δεδομένων και χρησιμοποιούν πολύπλοκα αναλογικά και ψηφιακά κυκλώματα για να ανακτήσουν το σήμα χρονισμού και τις πληροφορίες συγχρονισμού από το ρεύμα των δεδομένων.

*2.16.2 Κώδικες σειριακής γραμμής

Οι πιο ευρέως διαδεδομένοι κώδικες γραμμής για σειριακά δεδομένα δίνονται στην Εικόνα 2-17. Στον κώδικα NRZ, κάθε τιμή bit αποστέλλεται στη γραμμή για ολόκληρη την κυψέλη bit. Αυτή είναι η απλούστερη και πιο αξιόπιστη μέθοδος κωδικοποίησης για μετάδοση σε κοντινή απόσταση. Όμως, γενικά απαιτείται η αποστολή ενός σήματος ρολογιού μαζί με τα δεδομένα για τον καθορισμό των κυψελών bit. Διαφορετικά ο δέκτη δε θα είναι σε θέση να προσδιορίσει πόσα 0 και πόσα 1 αναπαρίστανται από μια συνεχή στάθμη 0 ή 1. Για παράδειγμα, χωρίς το σήμα χρονισμού για τον καθορισμό των κυψελών bit, η κυματομορφή NRZ της Εικόνας 2-17 μπορεί εσφαλμένα να εκληφθεί ως 01010.

ψηφιακός κλειστός βρόχος φάσης (DPLL)

Ο *ψηφιακός κλειστός βρόχος φάσης* (Digital Phase Locked Loop, DPLL) είναι ένα αναλογικό/ψηφιακό κύκλωμα που μπορεί να χρησιμοποιηθεί για την ανάκτηση του σήματος χρονισμού από ένα σειριακό ρεύμα δεδομένων. Το DPLL λειτουργεί μόνο αν το σειριακό ρεύμα δεδομένων περιέχει αρκετές μεταβάσεις από το 1 στο 0 και από το 0 στο 1 για να δώσει στο DPLL "υποδείξεις" σχετικά με το πότε πραγματοποιήθη-



Εικόνα 2-17
Συνήθεις κώδικες γραμμής για σειριακά δεδομένα.

καν οι αρχικές μεταβάσεις του σήματος χρονισμού. Για τη μετάδοση δεδομένων με κωδικοποίηση NRZ, το DPLL λειτουργεί μόνο αν τα δεδομένα δεν περιέχουν μεγάλα και συνεχή ρεύματα των 1 και 0.

μέσα ευαίσθητα στις μεταβάσεις

Κάποια μέσα σειριακής μετάδοσης και αποθήκευσης είναι ευαίσθητα στις μεταβάσεις, δηλαδή δεν μπορούν να μεταδώσουν ή να αποθηκεύσουν απόλυτες στάθμες του 0 και του 1 παρά μόνο μεταβάσεις μεταξύ δύο διακριτών σταθμών. Για παράδειγμα, ένας μαγνητικός δίσκος ή μαγνητική ταινία αποθηκεύει πληροφορίες με αλλαγή της πολικότητας μαγνήτισης του μέσου στις περιοχές που αντιστοιχούν στα αποθηκευμένα bit. Όταν οι πληροφορίες ανακτώνται, δεν είναι εφικτός ο προσδιορισμός της απόλυτης πολικότητας μαγνήτισης μιας περιοχής, αλλά μόνο η διαπίστωση της αλλαγής πολικότητας από τη μια περιοχή στην άλλη.

Τα δεδομένα που αποθηκεύονται σε μορφή NRZ (Non Return to Zero) πάνω σε μέσα ευαίσθητα στις μεταβάσεις δεν είναι δυνατόν να ανακτηθούν με σαφήνεια. Για παράδειγμα, τα δεδομένα της Εικόνας 2-17 είναι δυνατόν να ερμηνευθούν ως 01110010 ή ως 10001101. Ο κώδικας "*χωρίς επιστροφή στο μηδέν και με αντιστροφή στα 1*" (*Non-Return-to-Zero Invert-on-1s, NRZI*) ξεπερνά αυτή τη δυσκολία μεταδίδοντας ένα 1 ως το αντίθετο της στάθμης που μεταδόθηκε κατά τη διάρκεια της προηγούμενης κυψέλης bit, ενώ μεταδίδει ένα 0 ως την ίδια στάθμη. Ένα DPLL μπορεί να ανακτήσει το σήμα χρονισμού από δεδομένα με κωδικοποίηση NRZI εφόσον τα δεδομένα δεν περιλαμβάνουν κάποιο μεγάλης διάρκειας και συνεχές ρεύμα από 0.

χωρίς επιστροφή στο μηδέν και με αντιστροφή στα 1 (NRZI)

με επιστροφή στο μηδέν - RZ

Ο κώδικας "*με επιστροφή στο μηδέν*" (Return-to-Zero, RZ) είναι παρόμοιος με τον κώδικα NRZ με τη διαφορά ότι, για ένα bit-1, η στάθμη 1 μεταδίδεται μόνο για ένα κλάσμα της διάρκειας του bit, συνήθως 1/2. Με αυτόν τον κώδικα, τα υποδείγματα δεδομένων που περιέχουν πολλά 1 δημιουργούν πολλές μεταβάσεις σε ένα DPLL, που χρησιμοποιείται για την ανάκτηση του σήματος χρονισμού. Ωστόσο, όπως συμβαίνει και στους άλλους κώδικες γραμμής, μια ακολουθία από 0 δεν προκαλεί καμία μετάβαση, ενώ μια μεγάλη ακολουθία από 0 καθιστά την ανάκτηση του σήματος χρονισμού ανέφικτη.

Μια άλλη απαίτηση κάποιων μέσων μετάδοσης, όπως οι συνδέσεις υψηλών ταχυτήτων με οπτικές ίνες, είναι ότι το σειριακό ρεύμα δεδομένων πρέπει να είναι *εξισορροπημένο κατά DC*. Πρέπει δηλαδή να έχει ίσο αριθμό 1 και 0. Κάθε συνιστώσα DC μακράς διαρκείας μέσα στο ρεύμα των δεδομένων (η οποία δημιουργείται από την ύπαρξη περισσότερων 1 από τα 0 ή και το αντίστροφο) δημιουργεί μια πόλωση στο δέκτη η οποία μειώνει την ικανότητά του να διακρίνει αξιόπιστα μεταξύ των 0 και 1.

ισορροπία κατά DC

Συνήθως, τα δεδομένα με κωδικοποίηση NRZ, NRZI, ή RZ δεν εγγυώνται την ισορροπία κατά DC. Με άλλα λόγια, δεν υπάρχει κάτι που να εμποδίζει ένα ρεύμα δεδομένων χρήστη να έχει μεγάλης διάρκειας ακολουθίες λέξεων με περισσότερα 1 από τα 0 ή και το αντίστροφο. Όμως, η ισορροπία κατά DC μπορεί ακόμη να επιτευχθεί με τη χρήση λίγων επιπλέον bit για την κωδικοποίηση των δεδομένων του χρήστη σε έναν *εξισορροπημένο κώδικα*, όπου η κάθε κωδικολέξη θα έχει ίσο αριθμό 1 και 0 και κατόπιν οι κωδικολέξεις που θα προκύπτουν θα αποστέλλονται σε μορφή NRZ.

εξισορροπημένος κώδικας

Για παράδειγμα, στην Ενότητα 2.13 παρουσιάσαμε τον κώδικα 8B10B, ο οποίος κωδικοποιεί 8-bit από τα δεδομένα του χρήστη σε 10 bit, σε κώδικα το πολύ 5 από 10. Θυμηθείτε ότι υπάρχουν μόνο 252 κωδικολέξεις "5 από 10", αλλά και άλλες $\binom{4}{10} = 210$ κωδικολέξεις "4 από 10", καθώς και ένας ίσος αριθμός κωδικολέξεων "6 από 10". Φυσικά, αυτές οι κωδικολέξεις δεν είναι ακριβώς εξισορροπημένες κατά DC. Ο κώδικας 8B110B λύνει αυτό το πρόβλημα συσχετίζοντας κάθε τιμή των 8-bit που πρόκειται να κωδικοποιηθεί με ένα *ζευγάρι* μη εξισορροπημένων κωδικολέξεων, μια "4 από 10" ("μικρού βάρους") και μια "6 από 10" ("μεγάλου βάρους"). Ο κωδικοποιητής επίσης παρακολουθεί την *κινητή ανισοτιμία*, ένα απλό bit πληροφορίας το οποίο δείχνει αν η τελευταία μη εξισορροπημένη κωδικολέξη ήταν μεγάλου ή μικρού βάρους. Τη στιγμή που ο κωδικοποιητής πρέπει να μεταδώσει μια άλλη μη εξισορροπημένη κωδικολέξη, ο κωδικοποιητής επιλέγει εκείνη τη λέξη του ζευγαριού με το αντίθετο βάρος. Αυτό το απλό τέχνασμα δίνει στον κώδικα 8B10B συνολικά 252 +210 = 462 διαθέσιμες κωδικολέξεις για να κωδικοποιήσει 8 bit από τα δεδομένα του χρήστη. Μερικές από τις παραπάνω κωδικολέξεις χρησιμοποιούνται για τη βολική κωδικοποίηση καταστάσεων που δεν αποτελούν δεδομένα στη σειριακή γραμμή, όπως

κινητή ανισοτιμία

ΚΙΛΟ-, ΜΕΓΑ-, ΓΙΓΑ-, ΤΕΡΑ-

Τα προθέματα K (κιλο-), M (μεγα-), G (γιγα-), και T (τερα-) σημαίνουν 10^3, 10^6, 10^9 και 10^{12} αντίστοιχα, όταν αναφέρονται σε bps, Hertz, Ω, Watt και στα περισσότερα λοιπά μεγέθη που αφορούν τους μηχανικούς. Ωστόσο, όταν αναφερόμαστε σε μνήμες, τα προθέματα αυτά σημαίνουν 2^{10}, 2^{20}, 2^{30} και 2^{40}. Ιστορικά, τα προθέματα αυτά επιλέχθηκαν γι' αυτή τη χρήση επειδή τα μεγέθη των μνημών κανονικά είναι δυνάμεις του 2 και το 2^{10} (1024) είναι πολύ κοντά στο 1000.

Τώρα, αν κάποιος σας προσφέρει 50 κιλοευρώ το χρόνο για την πρώτη σας δουλειά μηχανικού, επαφίεται σε εσάς να διαπραγματευθείτε τη σημασία του προθέματος!

π.χ. IDLE, SYNC, και ERROR. Δε χρησιμοποιούνται όλες οι μη εξισορροπημένες κωδικολέξεις. Επίσης, δε χρησιμοποιούνται και κάποιες από τις εξισορροπημένες κωδικολέξεις, όπως το 0000011111, υπέρ των μη εξισορροπημένων ζευγαριών που περιέχουν περισσότερες μεταβάσεις.

με διπολική επιστροφή στο μηδέν (BPRZ)

με εναλλάξ αντιστροφή σημείων (AMI)

Όλοι οι παραπάνω κώδικες μεταδίδουν ή αποθηκεύουν μόνο δύο στάθμες σήματος. Ο κώδικας "με διπολική επιστροφή στο μηδέν" (Bipolar Return-to-Zero, BPRZ) μεταδίδει τρεις στάθμες σήματος: +1, 0 και -1. Αυτός ο κώδικας μοιάζει με τον NRZ, με τη διαφορά ότι τα 1 μεταδίδονται εναλλάξ ως +1 και -1. Γι' αυτόν το λόγο, είναι γνωστός και ως *"με εναλλάξ αντιστροφή σημείων"* (Alternate Mark Inversion, AMI).

Το μεγάλο πλεονέκτημα του κώδικα BPRZ έναντι του RZ είναι ότι είναι ισορροπημένος κατά DC. Αυτό μας δίνει τη δυνατότητα να στέλνουμε ροές δεδομένων BPRZ σε μέσα μετάδοσης που δεν μπορούν να ανεχθούν συνιστώσες DC, όπως π.χ. οι συζευγμένες με μετασχηματιστή τηλεφωνικές γραμμές. Πράγματι, ο κώδικας BPRZ χρησιμοποιείται στις ψηφιακές τηλεφωνικές συνδέσεις T1 εδώ και δεκαετίες, όπου τα αναλογικά φωνητικά σήματα μεταφέρονται ως ρεύμα 8000 ψηφιακών δειγμάτων των 8-bit ανά δευτερόλεπτο και μεταδίδονται σε μορφή BPRZ σε σειριακά κανάλια των 64-Kbps.

Όπως και με τον κώδικα RZ, η ανάκτηση ενός σήματος χρονισμού από ένα ρεύμα δεδομένων BPRZ είναι εφικτή εφόσον δεν υπάρχουν πάρα πολλά 0 στη σειρά. Παρά το γεγονός ότι η τηλεφωνική εταιρία δεν έχει τον έλεγχο σε όσα λέτε (τουλάχιστον, όχι ακόμη), έχει έναν εύκολο τρόπο για να περιορίσει τον αριθμό των συνεχόμενων 0. Αν ένα από τα byte των 8-bit που προκύπτει από τη δειγματοληψία του αναλογικού υποδείγματος της ομιλίας σας είναι όλο 0, τότε απλώς αλλάζουν το προτελευταίο λιγότερο σημαντικό bit σε 1! Αυτό ονομάζεται *συμπίεση μηδενικού κώδικα* (zero-code suppression) και στοιχηματίζω ότι δεν το είχατε αντιληφθεί μέχρι τώρα. Αυτό επίσης δικαιολογεί γιατί σε πολλές εφαρμογές δεδομένων των συνδέσεων T1 έχουμε μόνο 56 Kbps χρησιμοποιήσιμων δεδομένων στο κάθε κανάλι των 64-Kbps: το LSB κάθε byte γίνεται πάντα 1 για την αποτροπή της αλλαγής των υπολοίπων bit από τη συμπίεση μηδενικού κώδικα.

συμπίεση μηδενικού κώδικα

διφασικός κώδικας (Manchester)

Ο τελευταίος κώδικας στην Εικόνα 2-17 ονομάζεται κώδικας *Manchester* ή *διφασικός* κώδικας. Το μεγάλο πλεονέκτημα αυτού του κώδικα είναι ότι, ανεξάρτητα από το μεταδιδόμενο υπόδειγμα δεδομένων, δίνει τουλάχιστον μία μετάβαση ανά κυψέλη bit, διευκολύνοντας έτσι την ανάκτηση του σήματος χρονισμού. Όπως φαίνεται και στην εικόνα, το 0 κωδικοποιείται ως μετάβαση από το 0 στο 1 στη μέση της κυψέλης bit, ενώ το 1 κωδικοποιείται ως μετάβαση από το 1 στο 0. Όμως αυτό το μεγάλο πλεονέκτημα του κώδικα Manchester είναι ταυτόχρονα και το μεγάλο του μειονέκτημα. Εφόσον έχει περισσότερες μεταβάσεις ανά κυψέλη bit σε σύγκριση με άλλους κώδικες, χρειάζεται περισσότερο εύρος ζώνης του μέσου για να μεταδώσει δεδομένα με συγκεκριμένο ρυθμό bit. Ωστόσο, το εύρος ζώνης δεν αποτελεί πρόβλημα στα ομοαξονικά καλώδια τα οποία χρησιμοποιήθηκαν στα πρώτα τοπικά δίκτυα

ΣΧΕΤΙΚΑ ΜΕ ΤΙΣ ΤΗΛΕΦΩΝΙΚΕΣ ΕΤΑΙΡΙΕΣ

Δείτε την ταινία του James Coburn (1967) "Ο Αναλυτής του Προέδρου" για την αστεία πλευρά μιας τηλεφωνικής εταιρίας. Με την αυξανόμενη διάδοση της ψηφιακής τεχνολογίας και των φθηνών ασύρματων επικοινωνιών, η έννοια της δυνατότητας μιας καθολικής *προσωπικής* σύνδεσης με το τηλεφωνικό δίκτυο που παρουσιάζεται στον επίλογο της ταινίας έχει γίνει πλέον αρκετά κοντινή.

Ethernet για τη μεταφορά σειριακών δεδομένων με κωδικοποίηση Manchester με ρυθμό 10 Mbps (μεγαbit ανά δευτερόλεπτο).

Παραπομπές

Η παρουσίαση των πρώτων εννέα ενοτήτων αυτού του κεφαλαίου αυτού βασίστηκε στο Κεφάλαιο 4 του βιβλίου *Microcomputer Architecture and Programming*, του John F. Wakerly (Wiley, 1981). Ακριβείς, λεπτομερείς, και ενδιαφέρουσες πραγματείες αυτών των θεμάτων υπάρχουν επ-σης στο βιβλίο του Donald E. Knuth *Seminumerical Algorithms*, 3η έκδοση (Addison-Wesley, 1997). Οι αναγνώστες με προτίμηση στα μαθηματικά θα βρουν εξαιρετική την ανάλυση του Knuth πάνω στις ιδιότητες των αριθμητικών συστημάτων και της αριθμητικής, ενώ όλοι οι αναγνώστες θα απολαύσουν τη βαθιά γνώση και τα ιστορικά στοιχεία που υπάρχουν διάσπαρτα σε ολόκληρο το βιβλίο.

Οι περιγραφές των ψηφιακών λογικών κυκλωμάτων για αριθμητικές πράξεις όπως επίσης και μια εισαγωγή στις ιδιότητες των διάφορων αριθμητικών συστημάτων δίνονται στο βιβλίο του Kai Hwang, *Computer Arithmetic* (Wiley, 1979). Επίσης το βιβλίο του Hermann Schmid, *Decimal Computation* (Wiley, 1974) περιλαμβάνει μια λεπτομερή περιγραφή των τεχνικών της αριθμητική BCD.

Μια εισαγωγή στους αλγόριθμους που χρησιμοποιούνται για πολλαπλασιασμό και διαίρεση δυαδικών, καθώς και στην αριθμητική αριθμών κινητής υποδιαστολής, δίνεται στο βιβλίο του John F. Wakerly, *Microcomputer Architecture and Programming: The 68000 Family*. Μια πιο λεπτομερής περιγραφή των αριθμητικών τεχνικών και των αριθμητικών συστημάτων κινητής υποδιαστολής δίνεται στο βιβλίο *Introduction to Arithmetic for Digital Systems Designers* των Shlomo Waser και Michael J. Flynn (Holt, Rinehart and Winston, 1982).

πεπερασμένα πεδία

Οι κώδικες CRC βασίζονται στη θεωρία των *πεπερασμένων πεδίων* η οποία αναπτύχθηκε από το γάλλο μαθηματικό Evariste Galois (1811-1832) λίγο πριν σκοτωθεί σε μονομαχία με έναν πολιτικό του αντίπαλο. Το κλασικό βιβλίο πάνω στους κώδικες ανίχνευσης και διόρθωσης σφαλμάτων είναι το *Error-Correcting Codes* των W. W. Peterson και E. J. Weldon, Jr. (MIT Press, 1972, δεύτερη έκδοση). Ωστόσο, το βιβλίο αυτό συνιστάται μόνο σε αναγνώστες που έχουν εντρυφήσει στα μαθηματικά. Μια πιο προσιτή εισαγωγή στην κωδικοποίηση μπορεί κανείς να βρει στο βιβλίο *Error Control Coding: Fundamentals and Applications* των S.

Lin και D. J. Costello, Jr. (Prentice Hall, 1983). Μια άλλη εισαγωγή στη θεωρία των κωδίκων, με κατεύθυνση το χώρο των επικοινωνιών, δίνεται στο βιβλίο των A. M. Michelson και A. H. Levesque *Error-Control Techniques for Digital Communication* (Wiley-Intersience, 1985). Οι εφαρμογές υλικού των κωδίκων σε υπολογιστικά συστήματα εξετάζονται στο βιβλίο *Error-Detecting Codes, Self-Checking Circuits, and Applications* του John F. Wakerly (Elsevier/North-Holland, 1978).

Όπως περιγράφεται και στο παραπάνω βιβλίο του Wakelry, οι κώδικες αθροίσματος ελέγχου συμπληρώματος ως προς ένα έχουν την ικανότητα να ανιχνεύουν μεγάλης διάρκειας ριπές μονοκατευθυντικών σφαλμάτων. Αυτό είναι χρήσιμο στα κανάλια επικοινωνίας, όπου τα σφάλματα τείνουν όλα προς την ίδια κατεύθυνση. Οι ιδιαίτερες υπολογιστικές ιδιότητες που έχουν αυτοί οι κώδικες τους κάνουν επίσης ιδιαίτερα επιδεκτικούς στον αποτελεσματικό υπολογισμό αθροισμάτων ελέγχου από προγράμματα λογισμικού, τα οποία είναι σημαντικά λόγω της χρήσης τους στο Πρωτόκολλο Internet (IP). Δείτε τα RFC-1071 και RFC-1141. Οι Αιτήσεις για Σχόλια (Requests for Comments, RFC) υπάρχουν αρχειοθετημένες σε πολλά σημεία στον Ιστό (Web). Απλώς αναζητήστε τον όρο "RFC".

Μια εισαγωγή στις τεχνικές κωδικοποίησης για τη σειριακή μετάδοση δεδομένων, που περιλαμβάνει και τη μαθηματική ανάλυση των απαιτήσεων σε απόδοση και εύρος ζώνης για διάφορους κώδικες, παρουσιάζεται στο βιβλίο του R. M. Gagliardi, *Introduction to Communications Engineering* (Wiley-Interscience, 1998, δεύτερη έκδοση). Μια ακόμη καλή εισαγωγή στους σειριακούς κώδικες που χρησιμοποιούνται στους μαγνητικούς δίσκους και τις μαγνητικές ταινίες δίνεται από τον Richard Matick στο βιβλίο του *Computer Storage Systems and Technology* (Wiley-Interscience, 1977).

Η δομή του κώδικα 8B10B και η λογική που κρύβεται πίσω από αυτόν εξηγούνται στην πρωτότυπη ευρεσιτεχνία της IBM από τους Peter Franaszek και Albert Widmer, αριθμός ευρεσιτεχνίας στις Η.Π.Α. 4.486.739 (1984). Αυτή, καθώς και σχεδόν όλες οι ευρεσιτεχνίες στις Η.Π.Α. που έχουν εκδοθεί μετά από το 1971, υπάρχουν στον Ιστό (Web) στη διεύθυνση www.patents.ibm.com.

Προβλήματα για εξάσκηση

2.1 Πραγματοποιήστε τις παρακάτω μετατροπές αριθμητικών συστημάτων:

(α) $1101011_2 = ?_{16}$ (β) $174003_8 = ?_2$

(γ) $10110111_2 = ?_{16}$ (δ) $67{,}24_8 = ?_2$

(ε) $10100{,}1101_2 = ?_{16}$ (στ) $F3A5_{16} = ?_2$

(ζ) $11011001_2 = ?_8$ (η) $AB3D_{16} = ?_2$

(θ) $101111{,}0111_2 = ?_8$ (ι) $15C{,}38_{16} = ?_2$

2.2 Μετατρέψτε τους παρακάτω οκταδικούς αριθμούς σε δυαδικούς και δεκαεξαδικούς:

(α) $1023_8 = ?_2 = ?_{16}$ (β) $761302_8 = ?_2 = ?_{16}$
(γ) $163417_8 = ?_2 = ?_{16}$ (δ) $552273_8 = ?_2 = ?_{16}$
(ε) $5436{,}15_8 = ?_2 = ?_{16}$ (στ) $13705{,}207_8 = ?_2 = ?_{16}$

2.3 Μετατρέψτε τους παρακάτω δεκαεξαδικούς αριθμούς σε δυαδικούς και οκταδικούς:
(α) $1023_{16} = ?_2 = ?_8$ (β) $7E6A_{16} = ?_2 = ?_8$
(γ) $ABCD_{16} = ?_2 = ?_8$ (δ) $C350_{16} = ?_2 = ?_8$
(ε) $9E36{,}7A_{16} = ?_2 = ?_8$ (στ) $DEAD{,}BEEF_{16} = ?_2 = ?_8$

2.4 Ποιες είναι οι οκταδικές τιμές των τεσσάρων byte των 8 bit ενός 32μπιτου αριθμού με οκταδική αναπαράσταση 12345670123_8;

2.5 Μετατρέψτε τους παρακάτω αριθμούς σε δεκαδικούς:
(α) $1101011_2 = ?_{10}$ (β) $174003_8 = ?_{10}$
(γ) $10110111_2 = ?_{10}$ (δ) $67{,}24_8 = ?_{10}$
(ε) $10100{,}1101_2 = ?_{10}$ (στ) $F3A5_{16} = ?_{10}$
(ζ) $12010_3 = ?_{10}$ (η) $AB3D_{16} = ?_{10}$
(θ) $7156_8 = ?_{10}$ (ι) $15C{,}38_{16} = ?_{10}$

2.6 Πραγματοποιήστε τις παρακάτω μετατροπές μεταξύ των αριθμητικών συστημάτων:
(α) $125_{10} = ?_2$ (β) $3849_{10} = ?_8$
(γ) $209_{10} = ?_2$ (δ) $9714_{10} = ?_8$
(ε) $132_{10} = ?_2$ (στ) $23851_{10} = ?_{16}$
(ζ) $727_{10} = ?_5$ (η) $57190_{10} = ?_{16}$
(θ) $1435_{10} = ?_8$ (ι) $65113_{10} = ?_{16}$

2.7 Προσθέστε τα παρακάτω ζευγάρια δυαδικών αριθμών, δείχνοντας παράλληλα όλα τα κρατούμενα:

(α)	(β)	(γ)	(δ)
110101	101110	11011101	1110010
+ 11001	+ 100101	+ 1100011	+ 1101101

2.8 Επαναλάβετε το Πρόβλημα 2.7 κάνοντας αφαίρεση αντί για πρόσθεση και δείχνοντας τώρα τα δανεικά αντί για τα κρατούμενα.

2.9 Προσθέστε τα παρακάτω ζευγάρια οκταδικών αριθμών:

(α)	(β)	(γ)	(δ)
1372	47135	175214	110321
+ 4631	+ 5125	+ 15240	+ 56573

2.10 Προσθέστε τα παρακάτω ζευγάρια δεκαεξαδικών αριθμών:

(α)	(β)	(γ)	(δ)
1372	4F1A5	F35B	1B90F
+ 4631	+ B8D5	+ 27E6	+ C44E

2.11 Γράψτε τις αναπαραστάσεις προσημασμένου μεγέθους, συμπληρώματος ως προς δύο, και συμπληρώματος ως προς ένα των 8 bit για κάθε έναν από τους παρακάτω δεκαδικούς αριθμούς: +18, +115, +79, -49, -3, -100.

2.12 Δείξτε κατά πόσον θα υπάρξει υπερχείλιση όταν προσθέτετε τους παρακάτω αριθμούς συμπληρώματος ως προς δύο των 8 bit:

(α)	11010100	(β)	10111001	(γ)	11011101	(δ)	00100110
+	10101011	+	11010110	+	00100001	+	01011010

2.13 Πόσα σφάλματα είναι δυνατόν να ανιχνευθούν από έναν κώδικα ελάχιστης απόστασης d;

2.14 Ποιος είναι ο ελάχιστος αριθμός των bit ισοτιμίας που απαιτούνται για να πάρουμε ένα διδιάστατο κώδικα απόστασης 4 με n bit πληροφορίας;

Ασκήσεις

2.15 Να ένα πρόβλημα για να σας ανοίξει την όρεξη. Ποιο είναι το δεκαεξαδικό ισοδύναμο του 61453_{10} ;

2.16 Κάθε μία από τις παρακάτω αριθμητικές πράξεις είναι σωστή σε τουλάχιστον ένα αριθμητικό σύστημα. Προσδιορίστε τις πιθανές βάσεις των αριθμών για κάθε πράξη.

(α) 1234+5432 = 6666 (β) 41/3= 13
(γ) 33/3 = 11 (δ) 23+44+14+32 = 223
(ε) 302/20 = 12.1 (στ) $\sqrt{41}$ = 5

2.17 Η πρώτη αποστολή στον Άρη βρήκε μόνο τα ερείπια ενός πολιτισμού. Από τα χειροτεχνήματα και τις ζωγραφιές, οι εξερευνητές συμπέραναν ότι τα όντα που δημιούργησαν αυτόν τον πολιτισμό ήταν τετράποδα με ένα πλοκάμι που διακλαδωνόταν και κατέληγε σε έναν αριθμό από συλληπτήρια "δάχτυλα". Έπειτα από πολύ μελέτη, οι εξερευνητές κατόρθωσαν να μεταφράσουν τα Αρειανά μαθηματικά και βρήκαν την ακόλουθη εξίσωση:

$$5x^2 - 5x + 125 = 0$$

με τις ενδεικτικές λύσεις x=5 και x=8. Η τιμή x=5 φαινόταν λογική, αλλά η τιμή x=8 χρειαζόταν κάποια εξήγηση. Τότε οι εξερευνητές συλλογίστηκαν προς την κατεύθυνση κατά την οποία αναπτύχθηκε το γήινο αριθμητικό σύστημα και βρήκαν ενδείξεις ότι και το Αρειανό αριθμητικό σύστημα είχε παρόμοια ιστορία. Πόσα δάχτυλα πιστεύετε ότι είχαν οι Αρειανοί; (Από το *The Bent of Tau Beta Pi*, Φεβρουάριος 1956.)

2.18 Έστω ότι ο αριθμός B των $4n$ bit αναπαρίσταται από ένα δεκαεξαδικό αριθμό H των n ψηφίων. Αποδείξτε ότι το συμπλήρωμα ως προς 2 του B αναπαρίσταται από το συμπλήρωμα ως προς 16 του H. Διατυπώστε και αποδείξετε μια παρόμοια πρόταση για την οκταδική αναπαράσταση.

2.19 Επαναλάβετε την Άσκηση 2.18 χρησιμοποιώντας το συμπλήρωμα ως προς 1 του B και το συμπλήρωμα ως προς 15 του H.

2.20 Δίνεται ένας ακέραιος x στο διάστημα $-2^{n-1} \le x \le 2^{n-1}-1$ και ορίζουμε ως $[x]$ το συμπλήρωμα ως προς 2 του x, εκφραζόμενο ως θετικό αριθμό: $[x] = x$ για $x \ge 0$ και $[x]= 2^n - |x|$ για $x < 0$, όπου $|x|$ είναι η απόλυτη τιμή του x. Δίνεται ακόμη ένα άλλος ακέραιος y στο ίδιο διάστημα με τον x. Αποδείξτε ότι οι κανόνες της πρόσθεσης αριθμών συμπληρώματος ως προς δύο της Ενότητας 2.6 είναι σωστοί αποδεικνύοντας ότι η παρακάτω εξίσωση είναι πάντα αληθής:

$$[x + y] = [x] + [y] \text{ modulo } 2^n$$

(*Υπόδειξη*: Θεωρήστε τέσσερις περιπτώσεις βασισμένες στα πρόσημα των x και y. Χωρίς βλάβη της γενικότητας, μπορείτε να υποθέσετε ότι $|x| \ge |y|$.)

2.21 Επαναλάβετε την Άσκηση 2.20 χρησιμοποιώντας κατάλληλες αναπαραστάσεις και κανόνες για την πρόσθεση αριθμών συμπληρώματος ως προς ένα.

2.22 Διατυπώστε έναν κανόνα υπερχείλισης για την πρόσθεση αριθμών συμπληρώματος ως προς δύο σε σχέση με τις πράξεις απαρίθμησης στην αρθρωτή αναπαράσταση της Εικόνας 2-3.

2.23 Δείξτε ότι ένας αριθμός συμπληρώματος ως προς δύο μπορεί να μετατραπεί σε μια αναπαράσταση με περισσότερα bit με εφαρμογή της *επέκτασης προσήμου*. Δηλαδή, έστω ότι δίνεται ένας αριθμός X συμπληρώματος ως προς δύο των n-bit. Δείξτε ότι η αναπαράσταση συμπληρώματος ως προς δύο των m-bit του X, όπου $m > n$, μπορεί να προέλθει με προσάρτηση m-n αντιγράφων του bit προσήμου του X στα αριστερά της αναπαράστασης των n-bit του X.

2.24 Δείξτε ότι ένας αριθμός συμπληρώματος ως προς δύο μπορεί να μετατραπεί σε μια αναπαράσταση με λιγότερα bit με απαλοιφή bit υψηλής τάξης. Με άλλα λόγια, για έναν αριθμό X συμπληρώματος ως προς δύο των n-bit, δείξτε ότι ο αριθμός Y συμπληρώματος ως προς δύο των m-bit, που προκύπτει με απόρριψη d-bit από το αριστερό άκρο του X, αναπαριστά τον ίδιο αριθμό όπως και ο X όταν και μόνο όταν όλα τα απορριφθέντα bit είναι ίσα με το bit προσήμου του Y.

2.25 Γιατί η στίξη του "συμπληρώματος ως προς ένα" δεν είναι συνεπής με εκείνη του "συμπληρώματος ως προς δύο"; (Δείτε τις δύο πρώτες αναφορές στην ενότητα Παραπομπές.)

2.26 Ένας δυαδικός αθροιστής των n-bit μπορεί να χρησιμοποιηθεί για να πραγματοποιήσει μια πράξη απρόσημης αφαίρεσης X-Y των n-bit, με εκτέλεση της πράξης $X+\overline{Y}+1$, όπου το X και το Y είναι απρόσημοι αριθμοί των n -bit και το $\overline{Y}$ αναπαριστά το bit-προς-bit συμπλήρωμα του Y. Αποδείξτε τον παραπάνω ισχυρισμό ως εξής. Αποδείξτε πρώτα ότι $(X-Y) = (X+\overline{Y}+1)- 2^n$. Έπειτα, αποδείξτε ότι το κρατούμενο που προέρχεται από τον αθροιστή n-bit είναι το αντίθετο του δανεικού που προκύπτει από την αφαίρεση των n-bit. Δηλαδή, δείξτε ότι η πράξη X-Y παράγει ένα δανεικό πέραν της θέσης του MSB όταν και μόνο όταν η πράξη $X+\overline{Y}+1$ *δεν* παράγει κρατούμενο πέραν της θέσης του MSB.

2.27 Στις περισσότερες περιπτώσεις, για την αναπαράσταση του γινομένου μεταξύ δύο αριθμών συμπληρώματος ως προς δύο των n bit χρειάζονται λιγότερα από $2n$ bit. Υπάρχει, μάλιστα, μόνο μία περίπτωση στην οποία χρειάζονται $2n$ bit. Ποια είναι αυτή;

2.28 Αποδείξτε ότι ένα αριθμός συμπληρώματος ως προς δύο μπορεί να πολλαπλασιαστεί με το 2 με μετατόπιση των ψηφίων του κατά μια θέση bit προς τα αριστερά, με ένα κρατούμενο 0 στη θέση του λιγότερο σημαντικού bit και με παράβλεψη του τυχόν κρατουμένου πέρα της θέσης του πλέον σημαντικού bit, με την υπόθεση ότι δε συμβαίνει υπερχείλιση. Διατυπώστε τον κανόνα για την ανίχνευση υπερχείλισης.

2.29 Διατυπώστε σωστά και αποδείξτε μια τεχνική όμοια με εκείνη της Άσκησης 2.28 για τον πολλαπλασιασμό ενός αριθμού συμπληρώματος ως προς ένα με το 2.

2.30 Δείξτε πώς γίνεται η αφαίρεση μεταξύ αριθμών BCD, διατυπώνοντας τους κανόνες για τη δημιουργία δανεικών και την εφαρμογή ενός παρά-

γοντα διόρθωσης. Δείξτε επίσης πώς οι κανόνες σας εφαρμόζονται σε κάθε μία από τις εξής αφαιρέσεις: 9-3, 5-7, 4-9, 1-8.

2.31 Πόσες διαφορετικές κωδικοποιήσεις δυαδικής κατάστασης των 3-bit είναι εφικτές για τον ελεγκτή φαναριού ελέγχου κυκλοφορίας του Πίνακα 2-12;

2.32 Απαριθμήστε όλα τα "εσφαλμένα" όρια, εκεί δηλαδή όπου μπορεί να ανιχνευθεί εσφαλμένη θέση, στο μηχανικό δίσκο κωδικοποίησης της Εικόνας 2-5.

2.33 Πόσα "εσφαλμένα" όρια μπορούν να υπάρξουν σε ένα μηχανικό δίσκο κωδικοποίησης που χρησιμοποιεί δυαδικό κώδικα των n-bit, συναρτήσει του n;

2.34 Στα ιδιωτικά και επιβατικά αεροπλάνα, οι αναμεταδότες υψομέτρου χρησιμοποιούν τον κώδικα Gray για την κωδικοποίηση των μετρούμενων τιμών του υψομέτρου οι οποίες μεταδίδονται στους ελεγκτές εναέριας κυκλοφορίας. Γιατί;

2.35 Ένας λαμπτήρας πυρακτώσεως καταπονείται κάθε φορά που ανοίγουμε το φως, έτσι σε κάποιες εφαρμογές η διάρκεια ζωής του λαμπτήρα περιορίζεται από το πόσες φορές τον ανάβουμε και τον σβήνουμε, και όχι από τον ολικό χρόνο κατά τον οποίο είναι "αναμμένος". Χρησιμοποιήστε τη γνώση σας πάνω στους κώδικες προκειμένου να προτείνετε έναν τρόπο διπλασιασμού της διάρκειας ζωής ενός λαμπτήρα με τρεις θέσεις σε τέτοιες εφαρμογές.

2.36 Πόσα διαφορετικά και ξεχωριστά υποσυστήματα κύβων υπάρχουν σε ένα n-διάστατο κύβο, συναρτήσει του n;

2.37 Βρείτε τρόπο να ζωγραφίσετε σε ένα φύλλο χαρτιού έναν 3-διάστατο κύβο (ή κάποιο άλλο διδιάστατο αντικείμενο) έτσι ώστε όλες οι γραμμές να μην τέμνονται μεταξύ τους ή αποδείξτε ότι αυτό είναι αδύνατο.

2.38 Επαναλάβετε την Άσκηση 2.37 για έναν 4-διάστατο κύβο.

2.39 Γράψτε ένα μαθηματικό τύπο που να δίνει τον αριθμό των m-διάστατων υποκύβων ενός n-διάστατου κύβου για μια συγκεκριμένη τιμή του m. (Η απάντησή σας πρέπει να είναι συνάρτηση των m και n.)

2.40 Προσδιορίστε τα σύνολα ισοτιμίας ενός κώδικα Hamming απόστασης 3 με 11-bit πληροφορίας.

2.41 Γράψτε τις κωδικολέξεις ενός κώδικα Hamming με ένα bit πληροφορίας.

2.42 Παρουσιάστε το υπόδειγμα ενός σφάλματος των 3-bit το οποίο δεν ανιχνεύεται αν τα "γωνιακά" bit ισοτιμίας δεν περιλαμβάνονται στους διδιάστατους κώδικες της Εικόνας 2-14.

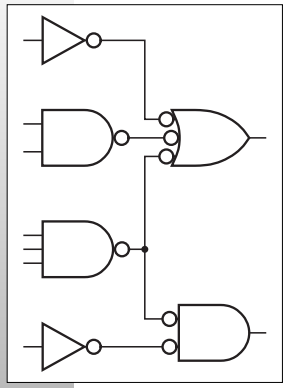
2.43 Ο *ρυθμός κώδικα* είναι ο λόγος του αριθμού των bit πληροφορίας προς το συνολικό αριθμό των bit σε μια κωδικολέξη. Οι υψηλοί ρυθμοί, που προσεγγίζουν το 1, είναι επιθυμητοί για την αποδοτική μετάδοση των πληροφοριών. Κατασκευάστε ένα γράφημα σύγκρισης των ρυθμών των κωδίκων ισοτιμίας απόστασης 2 και των κωδίκων Hamming απόστασης 3 και 4 με 100 bit πληροφορίας το πολύ.

2.44 Ποιος τύπος κώδικα απόστασης 4 έχει τον υψηλότερο ρυθμό: ένας διδιάστατος κώδικας ή ένας κώδικας Hamming; Τεκμηριώστε την απάντησή σας με έναν πίνακα του ιδίου στιλ με τον Πίνακα 2-15, συμπεριλαμβάνοντας το ρυθμό και τον αριθμό των bit ισοτιμίας και των bit πληροφορίας για τον κάθε κώδικα με 100 bit πληροφορίας το πολύ.

2.45 Δείξτε πώς κατασκευάζεται ένας κώδικας απόστασης 6 με τέσσερα bit πληροφορίας. Γράψτε έναν κατάλογο με τις κωδικολέξεις του κώδικα.

2.46 Περιγράψτε τις ενέργειες που πρέπει να γίνουν σε ένα σύστημα RAID για την εγγραφή νέων δεδομένων στο μπλοκ πληροφορίας *b* του δίσκου *d*, έτσι ώστε τα δεδομένα να είναι δυνατόν να ανακτηθούν στην περίπτωση σφάλματος στο μπλοκ *b* οποιουδήποτε δίσκου. Ελαχιστοποιήστε τον αριθμό των απαιτούμενων προσβάσεων στο δίσκο.

2.47 Στο ίδιο στιλ με την Εικόνα 2-17, σχεδιάστε τις κυματομορφές του υποδείγματος bit 10101110 όταν αυτό αποστέλλεται σειριακά με χρήση των κωδίκων NRZ, NRZI, RZ, BPRZ, και Manchester, υποθέτοντας ότι τα bit μεταδίδονται με τη σειρά, από τα αριστερά προς τα δεξιά.



κεφάλαιο 4

Αρχές συνδυαστικής λογικής σχεδίασης

Τα λογικά κυκλώματα κατατάσσονται σε δύο τύπους, τα "συνδυαστικά" και τα "ακολουθιακά". *Συνδυαστικό λογικό κύκλωμα* είναι αυτό του οποίου οι έξοδοι εξαρτώνται μόνο από τις τρέχουσες εισόδους. Το περιστροφικό κουμπί επιλογής καναλιών μιας παλιάς τηλεόρασης λειτουργεί όπως ένα συνδυαστικό κύκλωμα: η "έξοδός" του επιλέγει ένα κανάλι με βάση την τρέχουσα θέση του κουμπιού ("είσοδος").

Οι έξοδοι ενός *ακολουθιακού λογικού κυκλώματος* εξαρτώνται όχι μόνο από τις τρέχουσες εισόδους, αλλά και από την προηγούμενη ακολουθία εισόδων, η οποία μπορεί να φτάνει σε οποιοδήποτε βάθος χρόνου. Ο επιλογέας καναλιών που ελέγχεται από κουμπιά επιλογής πάνω και κάτω σε μια τηλεόραση ή ένα βίντεο είναι ένα ακολουθιακό κύκλωμα — η επιλογή των καναλιών εξαρτάται από την προηγούμενη ακολουθία πατημάτων των κουμπιών πάνω και κάτω, τουλάχιστον από τη στιγμή που αρχίσατε να βλέπετε 10 ώρες πριν, και πιθανώς από τότε που θέσατε σε λειτουργία τη συσκευή. Τα ακολουθιακά κυκλώματα περιγράφονται στα Κεφάλαια 7 και 8.

Ένα συνδυαστικό κύκλωμα είναι δυνατόν να περιέχει οποιονδήποτε αριθμό λογικών πυλών και αντιστροφέων, αλλά όχι βρόχους ανάδρασης. Ο *βρόχος ανάδρασης* είναι μια διαδρομή σήματος σε ένα κύκλωμα η οποία επιτρέπει στην έξοδο μιας πύλης να μεταδίδεται πίσω στην είσοδο της ίδιας πύλης. Ένας τέτοιος βρόχος δημιουργεί ακολουθιακή συμπεριφορά στο κύκλωμα.

Στην *ανάλυση* των συνδυαστικών κυκλωμάτων ξεκινάμε με ένα λογικό διάγραμμα και προχωράμε σε μια τυπική περιγραφή της λειτουργίας που εκτελείται από αυτό το κύκλωμα, όπως π.χ. με έναν πίνακα αληθείας ή με μια λογική παράσταση. Στη *σύνθεση* κάνουμε το αντίστροφο, ξεκινώντας με την τυπική περιγραφή και προχωρώντας σε ένα λογικό διάγραμμα.

ΣΥΝΘΕΣΗ ΚΑΙ ΣΧΕΔΙΑΣΗ

Η σχεδίαση λογικών κυκλωμάτων είναι ένα υπερσύνολο της σύνθεσης, εφόσον σε ένα πραγματικό πρόβλημα σχεδίασης ξεκινάμε συνήθως με μια άτυπη (λεκτική ή νοητή) περιγραφή του κυκλώματος. Συχνά το πιο δημιουργικό και δύσκολο μέρος της σχεδίασης είναι η διατύπωση της περιγραφής του κυκλώματος, με τον καθορισμό των σημάτων εισόδου και εξόδου του κυκλώματος και τον καθορισμό της λειτουργικής του συμπεριφοράς με τη βοήθεια πινάκων αληθείας και εξισώσεων. Αφού δημιουργήσουμε την τυπική περιγραφή του κυκλώματος, συνήθως συνεχίζουμε με μια διαδικασία σύνθεσης τύπου "περιστροφή της μανιβέλας" για να πάρουμε το λογικό διάγραμμα ενός κυκλώματος με την απαιτούμενη λειτουργική συμπεριφορά. Το υλικό που περιλαμβάνεται στις πρώτες τέσσερις ενότητες του κεφαλαίου αυτού αποτελεί τη βάση των διαδικασιών τύπου "περιστροφή της μανιβέλας", όπου η "μανιβέλα" γυρίζει με το χέρι ή με τον υπολογιστή. Οι δύο τελευταίες ενότητες περιγράφουν πραγματικές γλώσσες σχεδίασης, την ABEL και τη VHDL. Όταν δημιουργούμε μια σχεδίαση με μια από τις γλώσσες αυτές, το πρόγραμμα στον υπολογιστή μπορεί να εκτελεί τα βήματα της σύνθεσης για λογαριασμό μας. Στα επόμενα κεφάλαια θα συναντήσουμε πολλά παραδείγματα της πραγματικής διαδικασίας σχεδίασης.

Τα συνδυαστικά κυκλώματα είναι δυνατόν να έχουν μία ή περισσότερες εισόδους. Οι περισσότερες τεχνικές ανάλυσης και σύνθεσης είναι δυνατόν να επεκταθούν με προφανή τρόπο από κυκλώματα μίας εξόδου σε κυκλώματα πολλών εξόδων (π.χ. επαναλαμβάνοντας τα ίδια βήματα για κάθε έξοδο). Επίσης υποδεικνύουμε τον τρόπο επέκτασης μερικών τεχνικών με όχι και τόσο προφανή τρόπο για βελτιωμένη αποτελεσματικότητα στην περίπτωση των πολλών εξόδων.

Ο σκοπός αυτού του κεφαλαίου είναι να σας δώσει γερές θεωρητικές βάσεις όσον αφορά την ανάλυση και τη σύνθεση συνδυαστικών λογικών κυκλωμάτων, οι οποίες είναι σημαντικές για δύο λόγους, όπως θα δούμε όταν φτάσουμε στα ακολουθιακά κυκλώματα. Αν και οι περισσότερες από τις διαδικασίες ανάλυσης και σύνθεσης που αναφέρονται σε αυτό το κεφάλαιο είναι σήμερα αυτοματοποιημένες χάρη στη χρήση εργαλείων σχεδίασης με τη βοήθεια υπολογιστή, χρειάζεστε μια βασική κατανόηση των θεμελιωδών αρχών χρήσης των εργαλείων αυτών ώστε να εντοπίζετε το πρόβλημα όταν παίρνετε απρόσμενα ή ανεπιθύμητα αποτελέσματα.

Όταν θα έχετε πλέον κατανοήσει τις θεμελιώδεις αρχές, το επόμενο βήμα είναι να κατανοήσετε με ποιον τρόπο μπορούν να εκφραστούν και να αναλυθούν οι συνδυαστικές λειτουργίες με τη χρήση γλωσσών περιγραφής υλικού (HDL). Έτσι, οι δύο τελευταίες ενότητες του κεφαλαίου αυτού περιγράφουν τα βασικά χαρακτηριστικά των γλωσσών ABEL και VHDL, τις οποίες θα χρησιμοποιήσουμε για τη σχεδίαση διαφόρων λογικών κυκλωμάτων σε όλο το βιβλίο.

Προτού ξεκινήσουμε τη συζήτηση για τα συνδυαστικά λογικά κυκλώματα, πρέπει να κάνουμε μια εισαγωγή στην άλγεβρα μεταγωγής, το βα-

σικό μαθηματικό εργαλείο ανάλυσης και σύνθεσης κάθε τύπου λογικών κυκλωμάτων.

4.1 Άλγεβρα μεταγωγής

Οι τυπικές τεχνικές ανάλυσης ψηφιακών κυκλωμάτων έχουν τις ρίζες τους στην εργασία ενός Άγγλου μαθηματικού, του George Boole. Το 1854 ο Boole επινόησε ένα αλγεβρικό σύστημα δύο τιμών, που σήμερα λέγεται *άλγεβρα Boole*, με σκοπό να "εκφράσει τους θεμελιώδεις νόμους του συλλογισμού στη συμβολική γλώσσα ενός Λογισμού". Με το σύστημα αυτό ένας φιλόσοφος, κάποιος που ασχολείται με την επιστήμη της λογικής, ή ένας μόνιμος κάτοικος άλλου πλανήτη μπορεί να διατυπώνει προτάσεις που να είναι αληθείς ή ψευδείς, να τις συνδυάζει μεταξύ τους για να φτιάχνει νέες προτάσεις, καθώς και να προσδιορίζει την αλήθεια ή το ψεύδος των νέων προτάσεων. Για παράδειγμα, αν συμφωνήσουμε ότι "Οι άνθρωποι που δεν έχουν μελετήσει αυτή την ύλη είτε είναι αποτυχημένοι είτε δεν είναι σπασίκλες" και ότι "Κανένας σχεδιαστής υπολογιστών δεν είναι αποτυχημένος", μπορούμε να απαντήσουμε σε ερωτήσεις όπως π.χ. "Αν είσαι σπασίκλας σχεδιαστής υπολογιστών, τότε έχεις ήδη μελετήσει αυτό το αντικείμενο;".

άλγεβρα Boole

Αρκετά μετά τον Boole, το 1938, ο ερευνητής Claude E. Shannon των Εργαστηρίων Bell παρουσίασε τον τρόπο προσαρμογής της άλγεβρας Boole για την ανάλυση και την περιγραφή της συμπεριφοράς κυκλωμάτων που κατασκευάζονται από ηλεκτρονόμους (ρελέ), που ήταν τα πιο διαδεδομένα ψηφιακά λογικά στοιχεία εκείνης της εποχής. Στην *άλγεβρα μεταγωγής* του Shannon η κατάσταση της επαφής ενός ρελέ, δηλαδή ανοικτή ή κλειστή, αναπαρίσταται από μια μεταβλητή X η οποία είναι δυνατόν να έχει μια από δύο δυνατές τιμές, 0 ή 1. Στις σημερινές τεχνολογίες λογικής, αυτές οι τιμές αντιστοιχούν σε μεγάλη ποικιλία φυσικών συνθηκών, π.χ. υψηλή τάση (HIGH) ή χαμηλή τάση (LOW), αναμμένο ή σβηστό φως, πυκνωτής φορτισμένος ή εκφορτισμένος, ασφάλεια καμένη ή όχι κ.λπ., όπως είδαμε αναλυτικά στον Πίνακα 3-1, στην Ενότητα 3.1.

άλγεβρα μεταγωγής

Στο υπόλοιπο μέρος της ενότητας αυτής αναπτύσσουμε την άλγεβρα μεταγωγής, χρησιμοποιώντας τις "πρώτες βασικές αρχές" και τις γνώσεις που ήδη έχουμε σχετικά με τη συμπεριφορά των λογικών στοιχείων (πυλών και αντιστροφέων). Για περισσότερες ιστορικές ή/και μαθηματικές αναφορές σε αυτό το θέμα, συμβουλευθείτε τις Παραπομπές.

4.1.1 Αξιώματα

Στην άλγεβρα μεταγωγής χρησιμοποιούμε μια συμβολική μεταβλητή, όπως π.χ. X, για να αναπαραστήσουμε την κατάσταση ενός λογικού σήματος. Ένα λογικό σήμα είναι σε μία από δύο δυνατές καταστάσεις, π.χ. χαμηλή ή υψηλή τάση, αναμμένο ή σβηστό φως κ.λπ., ανάλογα με την τεχνολογία. Λέμε ότι το X έχει τιμή "0" για τη μια από αυτές τις καταστάσεις και "1" για την άλλη.

σύμβαση θετικής λογικής
σύμβαση αρνητικής λογικής

Για παράδειγμα, αναφορικά με τα λογικά κυκλώματα CMOS και TTL του Κεφαλαίου 3, η *σύμβαση θετικής λογικής* μάς επιβάλει να αντιστοιχίζουμε την τιμή "0" στην τάση LOW (χαμηλή) και την τιμή "1" στην τάση HIGH (υψηλή). Η *σύμβαση αρνητικής λογικής* κάνει την αντίθετη συσχέτιση: 0 = HIGH και 1 = LOW. Ωστόσο, η επιλογή της θετικής ή της αρνητικής λογικής δεν επηρεάζει τη δυνατότητά μας να αναπτύξουμε μια συνεπή αλγεβρική περιγραφή της συμπεριφοράς του κυκλώματος, παρά μόνο την αφαίρεση "από τη φυσική στην αλγεβρική μορφή", όπως θα εξηγήσουμε αργότερα στην ανάλυση της έννοιας της δυικότητας. Για την ώρα, μπορούμε να αγνοήσουμε τη φυσική πραγματικότητα των λογικών κυκλωμάτων και να προσποιηθούμε ότι λειτουργούν άμεσα με τα λογικά σύμβολα 0 και 1.

αξίωμα

Τα *αξιώματα* ενός μαθηματικού συστήματος είναι ένα ελάχιστο σύνολο βασικών ορισμών που θεωρούμε ότι είναι αληθείς, από τους οποίους μπορούν να παραχθούν όλες οι υπόλοιπες πληροφορίες του συστήματος. Τα δύο πρώτα αξιώματα της άλγεβρας μεταγωγής ενσωματώνουν την "ψηφιακή αφαίρεση", με την τυπική διατύπωση ότι η μεταβλητή X μπορεί να πάρει μόνο μία από δύο δυνατές τιμές:

(A1) X=0 αν X≠1 (A1′) X=1 αν X≠0

Σημειώστε ότι διατυπώσαμε τα αξιώματα ως ζεύγος, όπου η μόνη διαφορά ανάμεσα στο A1 και το A1′ είναι η εναλλαγή των συμβόλων 0 και 1. Αυτό είναι ένα χαρακτηριστικό όλων των αξιωμάτων της άλγεβρας μεταγωγής και αποτελεί τη βάση της αρχής της "δυικότητας" που θα μελετήσουμε αργότερα.

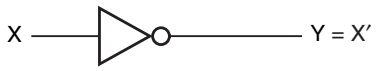
συμπληρωματικό
τόνος (′)

Στην Ενότητα 3.3.3 δείξαμε τη σχεδίαση ενός αντιστροφέα, ο οποίος είναι ένα λογικό κύκλωμα του οποίου η στάθμη του σήματος εξόδου είναι το αντίθετο (ή το *συμπληρωματικό*) της στάθμης του σήματος εισόδου. Ο *τόνος* (′) υποδηλώνει τη λειτουργία αντιστροφέα. Αυτό σημαίνει ότι αν η μεταβλητή X δείχνει ένα σήμα στην είσοδο ενός αντιστροφέα, τότε το X′ δείχνει την τιμή ενός σήματος στην έξοδο του αντιστροφέα. Αυτός ο συμβολισμός περιγράφεται τυπικά στο δεύτερο ζευγάρι αξιωμάτων:

(A2) αν X = 0, τότε X′ = 1 (A2′) αν X = 1, τότε X′ = 0

αλγεβρικός τελεστής
παράσταση
πράξη NOT

Όπως φαίνεται στην Εικόνα 4-1, η έξοδος ενός αντιστροφέα με σήμα εισόδου X μπορεί να έχει ένα οποιοδήποτε όνομα σήματος, π.χ. Y. Αλγεβρικά, ωστόσο, γράφουμε Y=X′ για να πούμε ότι "το σήμα Y έχει πάντα τιμή αντίθετη από εκείνη του σήματος X". Ο τόνος (′) είναι ένας *αλγεβρικός τελεστής*, ενώ το X′ είναι μια *παράσταση* την οποία μπορείτε να διαβάσετε ως "X τόνος" ή ως "NOT X". Η χρήση αυτή είναι ανάλογη με εκείνη που έχετε μάθει στις γλώσσες προγραμματισμού, όπου αν το J είναι μια ακέραια μεταβλητή, τότε το -J είναι μια παράσταση της οποίας η τιμή είναι 0 - J. Παρότι αυτά μπορεί να φαίνονται ασήμαντα, θα μάθουμε ότι η διάκριση ανάμεσα στα ονόματα των σημάτων (X, Y), τις παραστάσεις (X′) και τις εξισώσεις (Y=X′) έχει μεγάλη σημασία κατά τη με-



Εικόνα 4-1
Ονοματολογία και αλγεβρική σημειογραφία των σημάτων ενός αντιστροφέα.

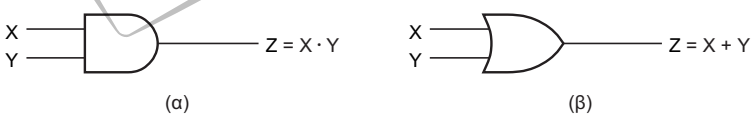
λέτη προτύπων τεκμηρίωσης και εργαλείων προγραμματισμού για τη σχεδίαση λογικών κυκλωμάτων.

Στην Ενότητα 3.3.6 είδαμε πώς κατασκευάζεται μια πύλη AND δύο εισόδων τύπου CMOS, ένα κύκλωμα του οποίου οι έξοδοι είναι 1 αν και οι δύο είσοδοι είναι 1. Η λειτουργία μιας πύλης AND δύο εισόδων λέγεται μερικές φορές *λογικός πολλαπλασιασμός* και συμβολίζεται αλγεβρικά με μια *τελεία πολλαπλασιασμού* (·). Αυτό σημαίνει ότι μια πύλη AND με εισόδους X και Y έχει ένα σήμα εξόδου του οποίου η τιμή είναι X·Y, όπως φαίνεται στην Εικόνα 4-2(α). Μερικοί συγγραφείς, ειδικά μαθηματικοί και επιστήμονες της λογικής, εκφράζουν το λογικό πολλαπλασιασμό με ένα σύμβολο γωνίας (X^Y). Εμείς τηρούμε την τυπική τεχνική πρακτική χρησιμοποιώντας την τελεία (X·Y). Κατά τη μελέτη γλωσσών περιγραφής υλικού (HDL) συναντάμε κάποια άλλα σύμβολα τα οποία χρησιμοποιούνται για να δείξουν το ίδιο πράγμα.

λογικός πολλαπλασιασμός
τελεία πολλαπλασιασμού (·)

Στην Ενότητα 3.3.6 περιγράψαμε επίσης πώς κατασκευάζεται μια πύλη OR δύο εισόδων τύπου CMOS, ένα κύκλωμα του οποίου η έξοδος είναι 1 αν οποιαδήποτε από τις εισόδους είναι 1. Η λειτουργία μιας πύλης OR δύο εισόδων μερικές φορές λέγεται *λογική πρόσθεση* και συμβολίζεται αλγεβρικά με το σύμβολο συν (+). Μια πύλη OR με εισόδους X και Y δίνει σήμα εξόδου με τιμή X+Y, όπως φαίνεται στην Εικόνα 4-2(β).

λογική πρόσθεση



Εικόνα 4-2
Ονοματολογία και αλγεβρική σημειογραφία σημάτων: (α) πύλη AND, (β) πύλη OR.

ΣΗΜΕΙΩΣΗ ΓΙΑ ΤΗ ΣΗΜΕΙΟΓΡΑΦΙΑ

Μερικοί συγγραφείς χρησιμοποιούν επίσης τις σημειογραφίες X, ~X και ¬X για το συμπλήρωμα του X. Η σημειογραφία με τη γραμμή από πάνω ($\overline{X}$) είναι μάλλον η ευρύτερα χρησιμοποιούμενη και η πιο εμφανίσιμη από τυπογραφική άποψη. Ωστόσο, εμείς χρησιμοποιούμε τη βασική σημειογραφία για να συνηθίσετε στη γραφή λογικών παραστάσεων σε μια γραμμή κειμένου, χωρίς την πιο γραφική γραμμή από πάνω, και για να αναγκαστείτε να χρησιμοποιείτε παρενθέσεις για τις σύνθετες συμπληρωματικές επιμέρους παραστάσεις, καθώς αυτό είναι που πρέπει να κάνετε όταν χρησιμοποιείτε γλώσσες HDL και άλλα εργαλεία.

Μερικοί συγγραφείς εκφράζουν τη λογική πρόσθεση με το σύμβολο "v" (XvY), αλλά εμείς ακολουθούμε την τυπική τεχνική πρακτική της χρήσης του συμβόλου συν (X+Y). Τονίζεται για άλλη μια φορά ότι μπορούν να χρησιμοποιηθούν και άλλα σύμβολα στις γλώσσες HDL. Συμβατικά, θεωρούμε ότι στις λογικές παραστάσεις που περιλαμβάνουν πολλαπλασιασμό και πρόσθεση ο πολλαπλασιασμός έχει *προτεραιότητα*, όπως οι ακέραιες παραστάσεις στις συμβατικές γλώσσες προγραμματισμού. Αυτό σημαίνει ότι η έκφραση W·X+Y·Z είναι ισοδύναμη με την (W·X)+(Y·Z).

προτεραιότητα

πράξη AND
πράξη OR

Τα τρία τελευταία ζεύγη αξιωμάτων διατυπώνουν τους τυπικούς ορισμούς των λειτουργιών AND και OR με την αναγραφή της εξόδου που παράγεται από κάθε πύλη για κάθε δυνατό συνδυασμό εισόδων:

(A3)	0·0=0	(A3′)	1+1=1
(A4)	1·1=1	(A4′)	0+0=0
(A5)	0·1=1·0=0	(A5′)	1+0=0+1=1

Τα πέντε ζεύγη αξιωμάτων A1-A5 και A1′-A5′ ορίζουν πλήρως την άλγεβρα μεταγωγής. Όλα τα υπόλοιπα συμβάντα του συστήματος αποδεικνύονται με τη χρήση των αξιωμάτων αυτών ως σημείων αρχής.

ΓΙΑ ΣΤΑΘΕΙΤΕ ΕΝΑ ΛΕΠΤΟ...

Στα παλαιότερα κείμενα χρησιμοποιείται η απλή *παράθεση* (XY) για να παρασταθεί ο λογικός πολλαπλασιασμός, εμείς όμως δεν τη χρησιμοποιούμε. Γενικά, η παράθεση είναι σαφής σημειογραφία μόνο όταν τα ονόματα των σημάτων περιορίζονται στον ένα χαρακτήρα. Διαφορετικά, τίθεται το ερώτημα: το XY είναι ένα λογικό γινόμενο ή είναι ένα όνομα σήματος με δύο χαρακτήρες; Τα ονόματα μεταβλητών με ένα χαρακτήρα είναι συνηθισμένα στην άλγεβρα, αλλά στα πραγματικά προβλήματα σχεδίασης προτιμάται η χρήση ονομάτων σημάτων με πολλούς χαρακτήρες οι οποίοι έχουν κάποιο νόημα. Έτσι, χρειαζόμαστε ένα διαχωριστικό ανάμεσα στα ονόματα, το οποίο μπορεί να είναι απλώς μια τελεία πολλαπλασιασμού αντί για ένα διάστημα. Το ισοδύναμο της HDL για την τελεία πολλαπλασιασμού (συνήθως * ή &) είναι απολύτως απαραίτητο όταν γράφονται λογικοί τύποι σε μια γλώσσα περιγραφής υλικού.

4.1.2 Θεωρήματα μίας μεταβλητής

Κατά τη διάρκεια της ανάλυσης ή της σύνθεσης λογικών κυκλωμάτων, συχνά γράφουμε αλγεβρικές παραστάσεις οι οποίες χαρακτηρίζουν την πραγματική ή την επιθυμητή συμπεριφορά. Τα *θεωρήματα* της άλγεβρας μεταγωγής είναι προτάσεις που γνωρίζουμε ότι είναι πάντα αληθείς και μας επιτρέπουν να χειριζόμαστε αλγεβρικές παραστάσεις οι οποίες επιτρέπουν την απλούστερη ανάλυση ή την αποτελεσματικότερη σύνθεση των αντίστοιχων κυκλωμάτων. Για παράδειγμα, το θεώρημα X+0=X μας επιτρέπει να αντικαθιστούμε κάθε εμφάνιση του X+0 σε μια παράσταση με το X.

Πίνακας 4-1
Θεωρήματα άλγεβρας μεταγωγής με μία μεταβλητή

(T1)	$X + 0 = X$	(T1′)	$X \cdot 1 = X$	(ταυτότητες)
(T2)	$X + 1 = 1$	(T2′)	$X \cdot 0 = 0$	(ουδέτερα στοιχεία)
(T3)	$X + X = X$	(T3′)	$X \cdot X = X$	(αυτοδυναμία)
(T4)	$(X')' = X$			(ενέλιξη)
(T5)	$X + X' = 1$	(T5′)	$X \cdot X' = 0$	(συμπληρώματα)

θεώρημα

Ο Πίνακας 4-1 παρουσιάζει τα θεωρήματα της άλγεβρας μεταγωγής που περιλαμβάνουν μία μεταβλητή X. Πώς ξέρουμε ότι αυτά τα θεωρήματα είναι αληθή; Μπορούμε είτε να τα αποδείξουμε μόνοι μας είτε να πάρουμε την απόδειξη από κάποιον που την έχει κάνει ήδη. Εντάξει, τώρα ακόμα μαθαίνουμε, ας δούμε πώς μπορούμε να τα αποδείξουμε.

τέλεια επαγωγή
πεπερασμένη επαγωγή

Τα περισσότερα θεωρήματα της άλγεβρας μεταγωγής αποδεικνύονται πάρα πολύ εύκολα με τη χρήση μιας τεχνικής που ονομάζεται *τέλεια επαγωγή*. Το αξίωμα A1 είναι το βασικό στοιχείο αυτής της τεχνικής: εφόσον μια μεταβλητή μεταγωγής μπορεί να πάρει μόνο δύο διαφορετικές τιμές, 0 και 1, μπορούμε να αποδείξουμε ένα θεώρημα που περιλαμβάνει μια μεταβλητή X αποδεικνύοντας ότι αυτό είναι αληθές τόσο για X=0 όσο και για X=1. Για παράδειγμα, για να αποδείξουμε το θεώρημα T1, κάνουμε δύο αντικαταστάσεις:

[X=0] 0+0=0 αληθές, σύμφωνα με το αξίωμα A4′
[X=1] 1+0=1 αληθές, σύμφωνα με το αξίωμα A5′

Όλα τα θεωρήματα του Πίνακα 4-1 είναι δυνατόν να αποδειχθούν με χρήση της τέλειας επαγωγής, όπως θα σας ζητηθεί να κάνετε στα Προβλήματα 4.2 και 4.3.

4.1.3 Θεωρήματα δύο και τριών μεταβλητών

Τα θεωρήματα της άλγεβρας μεταγωγής με δύο ή τρεις μεταβλητές παρουσιάζονται στον Πίνακα 4-2. Κάθε ένα από τα θεωρήματα αυτά αποδεικνύεται εύκολα με τη χρήση της τέλειας επαγωγής, με αξιολόγηση της πρότασης του θεωρήματος για τους τέσσερις δυνατούς συνδυασμούς των X και Y ή τους οκτώ δυνατούς συνδυασμούς των X, Y, και Z.

Τα δύο πρώτα ζεύγη θεωρημάτων αφορούν την αντιμεταθετικότητα και την προσεταιριστικότητα της λογικής πρόσθεσης και πολλαπλασιασμού, και είναι ταυτόσημα με τους νόμους της αντιμετάθεσης και του προσεταιρισμού της πρόσθεσης και του πολλαπλασιασμού ακεραίων και πραγματικών αριθμών. Η συνδυαστική χρήση τους δείχνει ότι η τοποθέτηση παρενθέσεων ή η σειρά των όρων σε ένα λογικό άθροισμα ή λογικό γινόμενο δεν έχει σημασία. Για παράδειγμα, από καθαρά αλγεβρική άποψη, μια παράσταση όπως η $W \cdot X \cdot Y \cdot Z$ είναι ασαφής. Μπορεί να γραφεί ως $(W \cdot (X \cdot (Y \cdot Z)))$ ή $(((W \cdot X) \cdot Y) \cdot Z)$ ή $(W \cdot X) \cdot (Y \cdot Z)$ (δείτε την Άσκηση 4.34). Ωστόσο, σύμφωνα με τα θεωρήματα, η ασαφής μορφή της παράστασης δε δημιουργεί πρόβλημα, αφού παίρνουμε τα ίδια αποτελέσματα σε κά-

Πίνακας 4-2 Θεωρήματα άλγεβρας μεταγωγής με δύο ή τρεις μεταβλητές.

(T6)	X+Y=Y+X	(T6′)	X·Y=Y·X	(Αντιμεταθετικότητα)
(T7)	(X+Y)+Z=X+(Y+Z)	(T7′)	(X·Y)·Z=X·(Y·Z)	(Προσεταιριστικότητα)
(T8)	X·Y+X·Z=X·(Y+Z)	(T8′)	(X+Y)·(X+Z)=X+Y·Z	(Επιμεριστικότητα)
(T9	X+X·Y=X	(T9′)	X·(X+Y)=X	(Κάλυψη)
(T10)	X·Y+X·Y′=X	(T10′)	(X+Y)·(X+Y′)=X	(Συνδυασμός)
(T11)	X·Y+X′·Z+Y·Z=X·Y+X′·Z			(Κοινή συναίνεση)
(T11′)	(X+Y)·(X′+Z)·(Y+Z)=(X+Y)·(X′+Z)			

θε περίπτωση. Ακόμη κι αν αλλάζαμε τη σειρά των μεταβλητών (π.χ. X·Z·Y·W), θα παίρναμε και πάλι τα ίδια αποτελέσματα.

δυαδικοί τελεστές

Όσο και αν αυτή η συζήτηση φαίνεται περιττή, το θέμα αυτό είναι πολύ σημαντικό καθώς διαμορφώνει τη θεωρητική βάση για τη χρήση λογικών πυλών με περισσότερες από δύο εισόδους. Έχουμε ορίσει τα σύμβολα · και + ως *δυαδικούς τελεστές*, δηλαδή τελεστές που συνδυάζουν *δύο* εισόδους. Ωστόσο, στην πράξη χρησιμοποιούμε λογικές πύλες AND και OR δύο, τριών, ή και περισσότερων εισόδων. Σύμφωνα με τα θεωρήματα, μπορούμε να συνδέσουμε τις εισόδους των πυλών με οποιαδήποτε σειρά. Πολλά προγράμματα πλακετών τυπωμένων κυκλωμάτων και διάταξης ASIC, μάλιστα, επωφελούνται από αυτό το πλεονέκτημα. Μπορούμε να χρησιμοποιήσουμε ισοδύναμα είτε μια πύλη *n* εισόδων είτε (*n* - 1) πύλες των 2 εισόδων η κάθε μία, αν και η καθυστέρηση και το κόστος είναι δυνατόν να είναι υψηλότερο στην περίπτωση των πολλών πυλών 2 εισόδων.

Το θεώρημα T8 είναι ταυτόσημο με τον επιμεριστικό νόμο των ακεραίων και των πραγματικών αριθμών, που σημαίνει ότι ο λογικός πολλαπλασιασμός επιμερίζεται πάνω στη λογική πρόσθεση. Συνεπώς, μπορούμε να "εκτελέσουμε τους επιμέρους πολλαπλασιασμούς" μιας παράστασης για να την πάρουμε σε μορφή αθροίσματος γινομένων, όπως στο παρακάτω παράδειγμα:

$$V\cdot(W+X)\cdot(Y+Z)=V\cdot W\cdot Y+V\cdot W\cdot Z+V\cdot X\cdot Y+V\cdot X\cdot Z$$

Ωστόσο, η άλγεβρα μεταγωγής έχει και την ασυνήθιστη ιδιότητα να αληθεύει και το αντίστροφο, δηλαδή η λογική πρόσθεση επιμερίζεται στο λογικό πολλαπλασιασμό, όπως αποδεικνύεται από το θεώρημα T8′. Έτσι μπορούμε να "εκτελέσουμε τις επιμέρους προσθέσεις" μιας παράστασης για να την πάρουμε σε μορφή γινομένου αθροισμάτων:

$$(V\cdot W\cdot X)+(Y\cdot Z)=(V+Y)\cdot(V+Z)\cdot(W+Y)\cdot(W+Z)\cdot(X+Y)\cdot(X+Z)$$

θεώρημα κάλυψης
κάλυψη
θεώρημα συνδυασμού

Τα θεωρήματα T9 και T10 χρησιμοποιούνται ευρέως στην ελαχιστοποίηση των λογικών συναρτήσεων. Αν, για παράδειγμα, εμφανίζεται η επιμέρους παράσταση X+X·Y σε μια λογική παράσταση, το *θεώρημα κάλυψης* T9 λέει ότι μόνο το X χρειάζεται να συμπεριληφθεί στην παράσταση. Το X λέγεται ότι *καλύπτει* το X·Y. Το *συνδυαστικό θεώρημα* T10

λέει ότι αν εμφανίζεται η επιμέρους παράσταση X·Y+X·Y′ σε μια παράσταση, μπορούμε να την αντικαταστήσουμε με το X. Εφόσον το Y μπορεί να είναι 0 ή 1, οποιαδήποτε από τις δύο περιπτώσεις της αρχικής επιμέρους παράστασης δίνουν 1 αν και μόνο αν το X είναι 1.

Παρά το γεγονός ότι μπορούμε εύκολα να αποδείξουμε το T9 με την τέλεια επαγωγή, η αλήθεια του T9 είναι προφανής αν την αποδείξουμε χρησιμοποιώντας τα θεωρήματα που αποδείξαμε ως τώρα:

X+X·Y	= X·1+X·Y	(σύμφωνα με το T1′)
	= X·(1+Y)	(σύμφωνα με το T8)
	= X·1	(σύμφωνα με το T2)
	= X	(σύμφωνα με το T1′)

Με παρόμοιο τρόπο, τα υπόλοιπα θεωρήματα είναι δυνατόν να χρησιμοποιηθούν για να αποδείξουμε το T10, όπου το κύριο βήμα είναι να χρησιμοποιήσουμε το T8 για να ξαναγράψουμε το αριστερό μέρος ως X·(Y+Y′).

Το θεώρημα T11 είναι γνωστό ως *θεώρημα κοινής συναίνεσης* (*consensus*). Ο όρος Y·Z λέγεται *όρος κοινής συναίνεσης* των X·Y και X′·Z. Με άλλα λόγια, αν το Y·Z είναι 1, τότε είτε το X·Y είτε το X′·Z πρέπει να είναι επίσης 1, εφόσον το Y και το Z είναι και τα δύο 1 και είτε το X είτε το X′ πρέπει να είναι 1. Έτσι, ο όρος Y·Z είναι πλεονάζων και μπορεί να απαλειφθεί από το δεξιό μέρος του T11. Το θεώρημα κοινής συναίνεσης έχει δύο σημαντικές εφαρμογές. Μπορεί να χρησιμοποιηθεί για την εξάλειψη ορισμένων κινδύνων χρονισμού σε συνδυαστικά λογικά κυκλώματα, όπως θα δούμε στην Ενότητα 4.5. Διαμορφώνει επίσης τη βάση της επαναληπτικής μεθόδου κοινής συναίνεσης για την εύρεση των πρωταρχικών όρων (prime implicants — δείτε τις Παραπομπές).

θεώρημα κοινής συναίνεσης

κοινή συναίνεση

Σε όλα τα θεωρήματα, κάθε μεταβλητή είναι δυνατόν να αντικατασταθεί με οποιαδήποτε λογική παράσταση. Μια απλή αντικατάσταση είναι να συμπληρώσετε μία ή περισσότερες μεταβλητές:

(X+Y′)+Z′ = X+(Y′+Z′) (βάσει του T7)

Είναι επίσης δυνατόν να αντικατασταθούν και πιο πολύπλοκες παραστάσεις:

(V′+X)·(W·(Y′+Z))+(V′+X)·(W·(Y′+Z))′ = V′ + X (βάσει του T10)

4.1.4 Θεωρήματα *n* μεταβλητών

Αρκετά σημαντικά θεωρήματα, που παρατίθενται στον Πίνακα 4-3, είναι αληθή για οποιονδήποτε αριθμό μεταβλητών, *n*. Τα περισσότερα από τα θεωρήματα αυτά είναι δυνατόν να αποδειχθούν με τη χρήση μιας μεθόδου δύο βημάτων που λέγεται *πεπερασμένη επαγωγή*: αρχικά αποδεικνύουμε ότι το θεώρημα είναι αληθές για *n*=2 (*βασικό βήμα*) και στη συνέχεια αποδεικνύουμε ότι αν το θεώρημα είναι αληθές για *n*=*i* τότε είναι επίσης αληθές για *n*=*i*+1 (*βήμα επαγωγής*). Για παράδειγμα, θεωρούμε το γενικευμένο θεώρημα αυτοδυναμίας T12. Για *n*=2, το T12 είναι ισοδύ-

βασικό βήμα

βήμα επαγωγής

ναμο με το T3 και επομένως είναι αληθές. Αν είναι αληθές για το λογικό άθροισμα των i X, τότε είναι επίσης αληθές για το άθροισμα των i+1 X, σύμφωνα με τον παρακάτω συλλογισμό:

X+X+X+...+X	= X+(X+X+...+X)	(i+1 X σε οποιαδήποτε πλευρά)
	= X+(X)	(αν το T12 είναι αληθές για n=i)
	= X	(σύμφωνα με το T3)

Συνεπώς, το θεώρημα είναι αληθές για όλες τις πεπερασμένες τιμές του n.

θεωρήματα DeMorgan

Τα *θεωρήματα του DeMorgan* (T13 και T13′) είναι ίσως τα πιο συχνά χρησιμοποιούμενα θεωρήματα της άλγεβρας μεταγωγής. Σύμφωνα με το θεώρημα T13, μια πύλη AND n εισόδων της οποίας η έξοδος υφίσταται συμπλήρωμα είναι ισοδύναμη με μια πύλη OR n εισόδων της οποίας οι είσοδοι υφίστανται συμπλήρωμα. Αυτό σημαίνει ότι οι Εικόνες 4-3(α) και (β) είναι ισοδύναμες.

Στην Ενότητα 3.3.4 δείξαμε πώς κατασκευάζεται μια πύλη NAND τύπου CMOS. Η έξοδος μιας πύλης NAND για οποιοδήποτε σύνολο εισόδων είναι το συμπλήρωμα της εξόδου μιας πύλης AND για τις ίδιες εισόδους. Συνεπώς, μια πύλη NAND μπορεί να έχει το λογικό σύμβολο της Εικόνας 4-3(γ). Ωστόσο, το κύκλωμα NAND τύπου CMOS δε σχεδιάζεται ως μια πύλη AND ακολουθούμενη από έναν αντιστροφέα με τρανζίστορ (πύλη NOT), αλλά είναι απλώς μια συλλογή από τρανζίστορ που συμβαίνει να εκτελούν τη λειτουργία AND-NOT. Το θεώρημα T13, μάλιστα, λέει ότι το λογικό σύμβολο στο (δ) υποδεικνύει την ίδια λογική λειτουργία (τα κυκλάκια στις εισόδους της πύλης OR υποδεικνύουν τη λογική αντιστροφή). Αυτό σημαίνει ότι μια πύλη NAND μπορεί θεωρηθεί ότι εκτελεί μια λειτουργία NOT-OR.

Παρατηρώντας τις εισόδους και την έξοδο μιας πύλης NAND, είναι αδύνατο να προσδιορίσουμε κατά πόσον έχει κατασκευαστεί εσωτερικά ως μια πύλη AND ακολουθούμενη από έναν αντιστροφέα, ως αντιστροφείς ακολουθούμενοι από μια πύλη OR, ή ως απευθείας υλοποίηση CMOS, *καθώς όλα τα κυκλώματα NAND εκτελούν ακριβώς την ίδια λει-*

Πίνακας 4-3 Θεωρήματα άλγεβρας μεταγωγής με n μεταβλητές.

(T12)	$X+X+...+X = X$	(Γενικευμένη αυτοδυναμία)
(T12′)	$X\cdot X\cdot...\cdot X = X$	
(T13)	$(X_1\cdot X_2\cdot...\cdot X_n)' = X_1'+X_2'+...+X_n'$	(Θεωρήματα DeMorgan)
(T13′)	$(X_1+X_2+...+X_n)' = X_1'\cdot X_2'\cdot...\cdot X_n'$	
(T14)	$[F(X_1,X_2,...,X_n,+,\cdot)]' = F(X_1',X_2',...,X_n',\cdot,+)$	(Γενικευμένο θεώρημα DeMorgan)
(T15)	$F(X_1,X_2,...,X_n) = X1\cdot F(1,X_2,...,X_n)+X1'\cdot F(0,X_2,...,X_n)$	(Θεωρήματα επέκτασης του Shannon)
(T15′)	$F(X_1,X_2,...,X_n) = [X_1+F(0,X_2,...,X_n)]\cdot[X_1'+F(1,X_2,...,X_n)$	

τουργία. Αν και η επιλογή του συμβόλου δεν έχει επιπτώσεις στη λειτουργικότητα ενός κυκλώματος, στην Ενότητα 5.1 θα δείξουμε ότι η κατάλληλη επιλογή μπορεί να κάνει πιο κατανοητή τη λειτουργία του κυκλώματος.

Ένα παρόμοιο συμβολικό ισοδύναμο μπορεί να συναχθεί από το θεώρημα T13′. Όπως φαίνεται στην Εικόνα 4-4, μια πύλη NOR μπορεί να υλοποιηθεί ως πύλη OR ακολουθούμενη από έναν αντιστροφέα ή ως αντιστροφείς ακολουθούμενοι από μια πύλη AND.

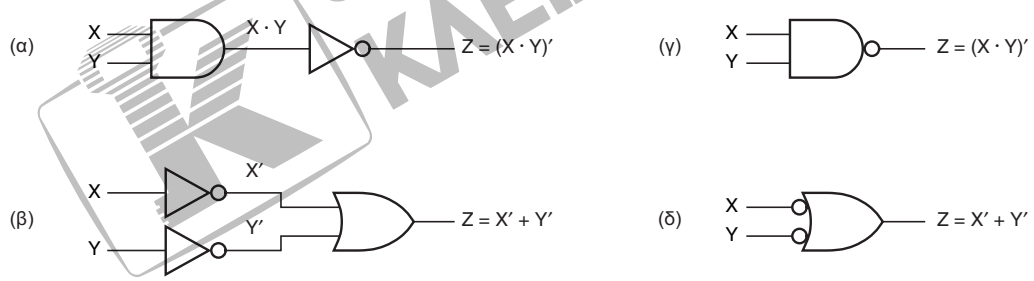
γενικευμένο θεώρημα DeMorgan

συμπλήρωμα λογικής παράστασης

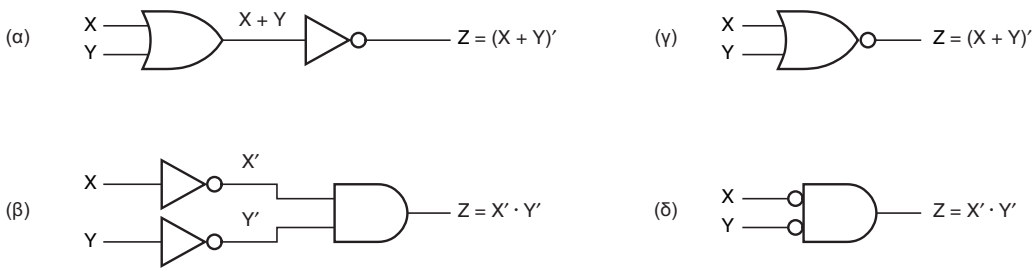
Τα θεωρήματα T13 και T13′ είναι απλώς ειδικές περιπτώσεις του *γενικευμένου θεωρήματος του DeMorgan* T14, το οποίο εφαρμόζεται σε οποιαδήποτε λογική παράσταση F. Εξ ορισμού, *το συμπλήρωμα μιας λογικής παράστασης*, που εκφράζεται ως (F′), είναι μια παράσταση της οποίας η τιμή είναι αντίθετη εκείνης της F για κάθε δυνατό συνδυασμό εισόδων. Το θεώρημα T14 είναι πολύ σημαντικό καθώς μας υποδεικνύει έναν τρόπο χειρισμού και απλούστευσης του συμπληρώματος μιας παράστασης.

Το θεώρημα T14 λέει ότι μπορούμε να πάρουμε το συμπλήρωμα οποιασδήποτε λογικής παράστασης *n* μεταβλητών, εναλλάσσοντας τα + και · μεταξύ τους και υπολογίζοντας τα συμπληρώματα όλων των μεταβλητών. Για παράδειγμα, έστω ότι έχουμε:

$$F(W,X,Y,Z) = (W'\cdot X)+(X\cdot Y)+(W\cdot(X'+Z'))$$
$$= ((W)'\cdot X)+(X\cdot Y)+(W\cdot((X)'+(Z)'))$$



Εικόνα 4-3 Ισοδύναμα κυκλώματα σύμφωνα με το θεώρημα DeMorgan T13: (α) AND-NOT, (β) NOT-OR, (γ) λογικό σύμβολο για πύλη NAND, (δ) ισοδύναμο σύμβολο για πύλη NAND.



Εικόνα 4-4 Ισοδύναμα κυκλώματα σύμφωνα με το θεώρημα του DeMorgan T13′: (α) OR-NOT, (β) NOT-AND, (γ) λογικό σύμβολο μιας πύλης NOR, (δ) ισοδύναμο σύμβολο μιας πύλης NOR.

Στη δεύτερη γραμμή έχουμε τοποθετήσει τα συμπληρώματα των μεταβλητών σε παρενθέσεις για να μας υπενθυμίζουν ότι το ′ είναι ένας τελεστής και όχι μέρος του ονόματος της μεταβλητής. Εφαρμόζοντας το θεώρημα T14, παίρνουμε

$$[F(W,X,Y,Z)]' = ((W')'+X')\cdot(X'+Y')\cdot(W'+((X')'\cdot(Z')'))$$

Χρησιμοποιώντας το θεώρημα T4, η παράσταση μπορεί να απλοποιηθεί ως εξής:

$$[F(W,X,Y,Z)]' = (W+X')\cdot(X'+Y')\cdot(W'+(X\cdot Z))$$

Γενικά, μπορούμε να χρησιμοποιήσουμε το θεώρημα T14 για να πάρουμε το συμπλήρωμα μιας παράστασης με παρενθέσεις, εναλλάσσοντας τα + και - μεταξύ τους και υπολογίζοντας το συμπλήρωμα όλων των μη συμπληρωματικών μεταβλητών, καθώς και αποκαθιστώντας τις συμπληρωματικές μεταβλητές στη μη συμπληρωματική τους μορφή.

Μπορύμε να αποδείξουμε το γενικευμένο θεώρημα του DeMorgan T14 αν δείξουμε ότι όλες οι λογικές συναρτήσεις είναι δυνατόν να γραφούν είτε ως άθροισμα είτε ως γινόμενο επιμέρους συναρτήσεων και κατόπιν αν εφαρμόσουμε αναδρομικά τα θεωρήματα T13 και T13′. Ωστόσο, η απόδειξη που βασίζεται στην αρχή της δυικότητας, η οποία περιγράφεται παρακάτω, είναι πολύ πιο κατατοπιστική και ικανοποιητική.

4.1.5 Δυικότητα

Διατυπώσαμε όλα τα αξιώματα της άλγεβρας μεταγωγής σε ζευγάρια. Η τονισμένη εκδοχή κάθε αξιώματος (π.χ. A5′) βγαίνει από τη μη τονισμένη εκδοχή του (π.χ. A5) με απλή εναλλαγή των 0 και 1 και, αν υπάρχουν, των · και + μεταξύ τους. Επομένως, μπορούμε να διατυπώσουμε το παρακάτω *μεταθεώρημα*, δηλαδή ένα θεώρημα για τα θεωρήματα:

Αρχή της δυικότητας Κάθε θεώρημα ή ταυτότητα της άλγεβρας μεταγωγής παραμένει αληθές αν εναλλάξουμε παντού τα 0 και 1 και τα · και + μεταξύ τους.

μεταθεώρημα

Το μεταθεώρημα είναι αληθές επειδή τα δυικά όλων των αξιωμάτων είναι αληθή, έτσι τα δυικά όλων των θεωρημάτων της άλγεβρας μεταγωγής είναι δυνατόν να αποδειχθούν με χρήση των δυικών των αξιωμάτων.

Τελικά, τι σημασία έχει το ζήτημα αυτό για τα ονόματα ή τα σύμβολα; Αν το λογισμικό που χρησιμοποιήθηκε για τη στοιχειοθεσία αυτού του βιβλίου είχε ένα σφάλμα που θα αντάλλασσε τα 0 - 1 και τα · - + παντού σε αυτό το κεφάλαιο, θα μαθαίνατε τη ίδια ακριβώς άλγεβρα μεταγωγής. Μόνο η ονοματολογία θα ήταν λίγο περίεργη, αν χρησιμοποιούνταν όροι όπως “γινόμενο” για την περιγραφή μιας πράξης που θα χρησιμοποιούσε το σύμβολο “+”.

Η δυικότητα είναι σημαντική καθώς διπλασιάζει τη χρησιμότητα όλων όσων μάθατε για την άλγεβρα μεταγωγής και το χειρισμό των συναρτήσεων μεταγωγής. Πρόκειται για μια πιο πρακτική για τους σπουδαστές διατύπωση, που μειώνει στο μισό την προς εκμάθηση ύλη! Για παράδειγμα, από τη στιγμή που θα μάθει κανείς πώς να συνθέτει λογικά

κυκλώματα AND-OR δύο σταδίων από παραστάσεις με αθροίσματα γινομένων, αυτομάτως γνωρίζει μια δυική τεχνική για τη σύνθεση κυκλωμάτων OR-AND από παραστάσεις με γινόμενα αθροισμάτων.

Υπάρχει μία μόνο σύμβαση στην άλγεβρα μεταγωγής όπου δε θεωρούμε ταυτόσημα τα · και + και συνεπώς η δυικότητα δε διατηρείται απαραίτητα αληθής. Μπορείτε να αντιληφθείτε ποια είναι προτού διαβάσετε την απάντηση που ακολουθεί; Θεωρήστε την ακόλουθη διατύπωση του θεωρήματος T9 και τη σαφώς άτοπη "δυική" της:

$X+X\cdot Y= X$ (θεώρημα T9)
$X\cdot X+Y= X$ (μετά την εφαρμογή της αρχής της δυικότητας)
$X+Y= X$ (μετά την εφαρμογή του θεωρήματος T3′)

Προφανώς η τελευταία γραμμή είναι ψευδής, αλλά πού μπορεί να κάναμε λάθος; Το πρόβλημα εντοπίζεται στην προτεραιότητα των τελεστών. Έχουμε τη δυνατότητα να γράψουμε το αριστερό μέρος της πρώτης γραμμής χωρίς παρενθέσεις λόγω της σύμβασης κατά την οποία το · έχει προτεραιότητα. Ωστόσο, εφαρμόζοντας την αρχή της δυικότητας, θα έπρεπε να έχουμε δώσει προτεραιότητα στο + αντί για το · ή να γράψουμε τη δεύτερη γραμμή ως $X\cdot(X+Y) = X$. Ο καλύτερος τρόπος για να αποφύγουμε προβλήματα όπως αυτό είναι να τοποθετούμε παρενθέσεις στην παράσταση προτού πάρουμε τη δυική της.

δυική λογικής παράστασης

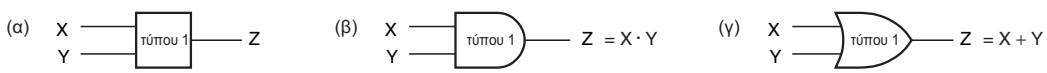
Ας ορίσουμε τυπικά τη *δυική μιας λογικής παράστασης*. Αν $F(X_1,X_2,...,X_n,+,\cdot,')$ είναι μια λογική παράσταση που διατυπώνεται εξ ολοκλήρου με παρενθέσεις και περιλαμβάνει τις μεταβλητές X_1, X_2,... X_n και τους τελεστές +, · και ′, τότε η δυική της παράστασης F, που γράφεται ως F^D, είναι η ίδια παράσταση με εναλλαγή των + και · μεταξύ τους:

$$F^D (X_1,X_2,...,X_n,+,\cdot,') = F(X_1,X_2,...,X_n,\cdot,+,')$$

Φυσικά το γνωρίζετε ήδη αυτό, αλλά γράψαμε τον ορισμό με αυτόν τον τρόπο μόνο και μόνο για να δώσουμε έμφαση στην ομοιότητα ανάμεσα στη δυικότητα και στο γενικευμένο θεώρημα του DeMorgan T14, το οποίο μπορεί να επαναδιατυπωθεί ως εξής:

$$[F(X_1,X_2,...,X_n)]' = F^D (X_1',X_2',...,X_n')$$

Ας εξετάσουμε την πρόταση αυτή στα πλαίσια ενός φυσικού δικτύου.

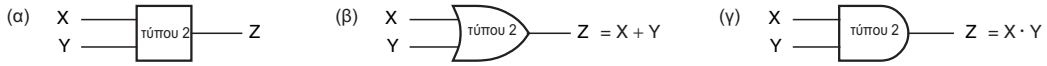


X	Y	Z
LOW	LOW	LOW
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	HIGH

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

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

Εικόνα 4-5 Μια λογική πύλη "τύπου 1": (α) πίνακας ηλεκτρικών λειτουργιών, (β) πίνακας λογικών λειτουργιών και σύμβολο με θετική λογική, (γ) πίνακας λογικών λειτουργιών και σύμβολο με αρνητική λογική.

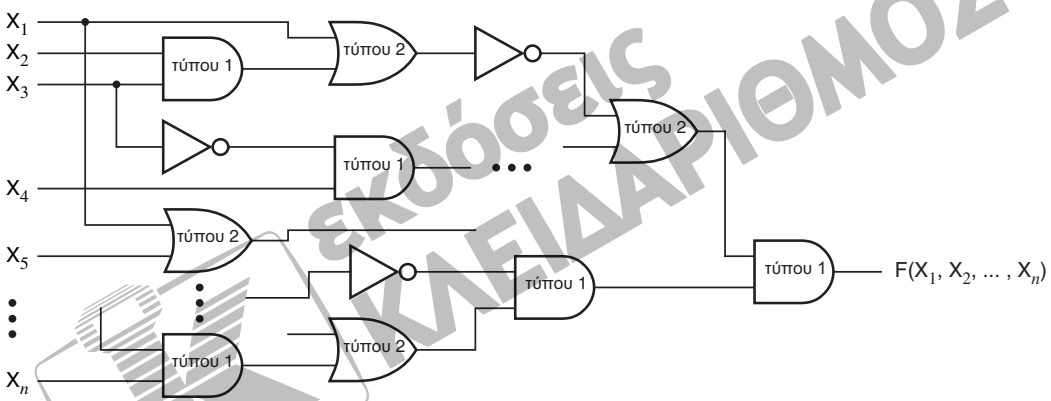


X	Y	Z
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	HIGH

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

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

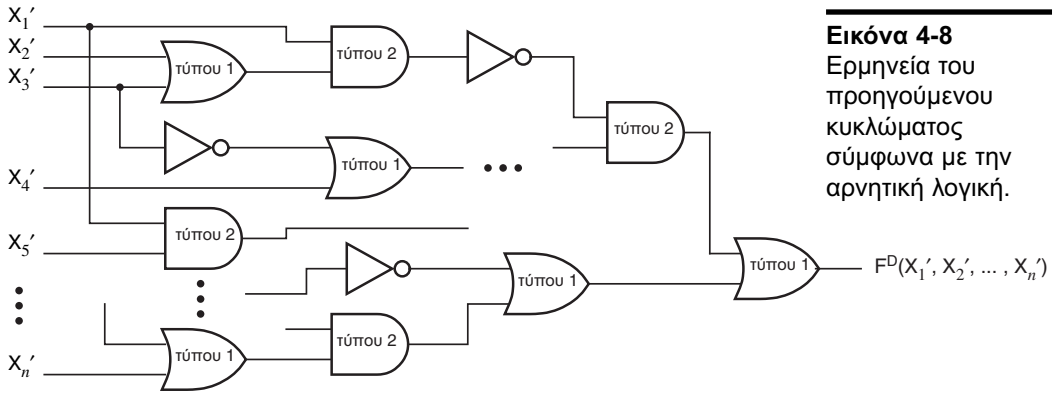
Εικόνα 4-6 Λογική πύλη “τύπου 2”: (α) πίνακας ηλεκτρικών λειτουργιών, (β) πίνακας λογικών λειτουργιών και σύμβολο με θετική λογική, (γ) πίνακας λογικών λειτουργιών και σύμβολο με αρνητική λογική.



Εικόνα 4-7 Κύκλωμα λογικής λειτουργίας που χρησιμοποιεί αντιστροφείς και πύλες τύπου 1 και τύπου 2 με σύμβαση θετικής λογικής.

Η Εικόνα 4-5(α) δείχνει τον πίνακα ηλεκτρικής λειτουργίας ενός λογικού στοιχείου το οποίο θα το αποκαλούμε απλώς πύλη “τύπου 1”. Σύμφωνα με τη σύμβαση της θετικής λογικής (LOW=0 και HIGH=1), πρόκειται για μια πύλη AND ενώ, σύμφωνα με τη σύμβαση της αρνητικής λογικής (LOW=1 και HIGH=0), πρόκειται για πύλη OR, όπως φαίνεται στα (β) και (γ). Μπορούμε επίσης να φανταστούμε μια πύλη “τύπου 2”, όπως φαίνεται στην Εικόνα 4-6, η οποία είναι πύλη OR θετικής λογικής ή πύλη AND αρνητικής λογικής. Παρόμοιοι πίνακες μπορούν να αναπτυχθούν για πύλες με περισσότερες από δύο εισόδους.

Έστω ότι μας δίνεται μια οποιαδήποτε λογική παράσταση $F(X_1,X_2,...,X_n)$. Ακολουθώντας τη σύμβαση θετικής λογικής, μπορούμε να κατασκευάσουμε ένα κύκλωμα που θα αντιστοιχεί στην παράσταση αυτή χρησιμοποιώντας αντιστροφείς για τις πράξεις NOT, πύλες τύπου 1 για τις πράξεις AND, και πύλες τύπου 2 για τις πράξεις OR, όπως φαίνεται στην Εικόνα 4-7. Υποθέτουμε τώρα ότι, χωρίς να αλλάξουμε αυτό το



Εικόνα 4-8
Ερμηνεία του προηγούμενου κυκλώματος σύμφωνα με την αρνητική λογική.

κύκλωμα, αλλάζουμε απλώς τη λογική σύμβαση από θετική σε αρνητική. Μετά μπορούμε να επανασχεδιάσουμε το κύκλωμα όπως φαίνεται στην Εικόνα 4-8. Είναι ξεκάθαρο ότι, για κάθε δυνατό συνδυασμό τάσεων εισόδου (HIGH και LOW), το κύκλωμα εξακολουθεί να παράγει την ίδια τάση εξόδου. Ωστόσο, αν το δούμε από τη σκοπιά της άλγεβρας μεταγωγής, η τιμή εξόδου, 0 ή 1, θα είναι αντίθετη από εκείνη που θα παραγόταν με τη σύμβαση θετικής λογικής. Ομοίως, κάθε τιμή εισόδου είναι αντίθετη από εκείνη που ήταν προηγουμένως. Επομένως, για κάθε δυνατό συνδυασμό εισόδων στο κύκλωμα της Εικόνας 4-7, η έξοδος είναι αντίθετη από εκείνη που παράγεται από τον αντίθετο συνδυασμό που εφαρμόζεται στο κύκλωμα της Εικόνας 4-8:

$$F(X_1,X_2,...,X_n) = [F^D(X_1',X_2',...,X_n')]'$$

Παίρνοντας το συμπλήρωμα και των δύο πλευρών, προκύπτει το γενικευμένο θεώρημα του DeMorgan:

$$[F(X_1,X_2,...,X_n)]' = F^D(X_1',X_2',...,X_n')$$

Καταπληκτικό!

Η ΔΥΙΚΟΤΗΤΑ ΕΙΝΑΙ ΒΟΛΙΚΗ ΚΑΙ ΓΙΑ ΤΟΥΣ ΦΟΙΤΗΤΕΣ ΚΑΙ ΓΙΑ ΤΟΥΣ ΣΥΓΓΡΑΦΕΙΣ

Διαπιστώσατε ότι η δυικότητα αποτελεί τη βάση του γενικευμένου θεωρήματος του DeMorgan. Στην πορεία, η δυικότητα θα περιορίσει στο μισό τον αριθμό των μεθόδων που πρέπει να μάθετε για να χειρίζεστε και να απλοποιείτε λογικές συναρτήσεις. Επίσης, περιόρισε στο μισό την ύλη που είχα να γράψω σε αυτές τις ενότητες!

4.1.6 Καθιερωμένες αναπαραστάσεις λογικών συναρτήσεων

Προτού προχωρήσουμε στην ανάλυση και τη σύνθεση συναρτήσεων συνδυαστικής λογικής, θα παρουσιάσουμε την αναγκαία ονοματολογία και σημειογραφία.

Πίνακας 4-4
Γενική δομή πίνακα αληθείας για λογική συνάρτηση τριών μεταβλητών, F(X,Y,Z).

Γραμμή	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

πίνακας αληθείας

Η πιο βασική αναπαράσταση μιας λογικής συνάρτησης είναι ο *πίνακας αληθείας*. Με φιλοσοφία παρόμοια της μεθόδου απόδειξης της τέλειας επαγωγής, η απλή αυτή αναπαράσταση παραθέτει την έξοδο του κυκλώματος για κάθε δυνατό συνδυασμό εισόδων. Συνήθως οι συνδυασμοί εισόδων διατάσσονται σε γραμμές κατά αύξουσα σειρά δυαδικής αρίθμησης, ενώ οι αντίστοιχες τιμές εξόδου διατάσσονται σε μια στήλη δίπλα στις γραμμές. Η γενική δομή ενός πίνακα αληθείας τριών μεταβλητών παρουσιάζεται στον Πίνακα 4-4.

Οι γραμμές αριθμούνται από το 0 έως το 7, σε αντιστοιχία με τους δυαδικούς συνδυασμούς εισόδων, αυτή η αρίθμηση όμως δεν αποτελεί ουσιαστικό τμήμα του πίνακα αληθείας. Στον Πίνακα 4-5 εμφανίζεται ένας πίνακας αληθείας για μια συγκεκριμένη λογική συνάρτηση τριών μεταβλητών. Κάθε διακριτό υπόδειγμα που αποτελείται από 0 και 1 στη στήλη εξόδου έχει αποτέλεσμα μια διαφορετική λογική συνάρτηση. Υπάρχουν 2^8 τέτοια υποδείγματα. Έτσι, η λογική συνάρτηση του Πίνακα 4-5 είναι μία από τις 2^8 διαφορετικές λογικές συναρτήσεις τριών μεταβλητών.

Ο πίνακας αληθείας μιας λογικής συνάρτησης n μεταβλητών έχει 2^n γραμμές. Προφανώς, οι πίνακες αληθείας είναι πρακτικό να γράφονται μόνο για λογικές συναρτήσεις με μικρό αριθμό μεταβλητών, π.χ. 10 για σπουδαστές και περίπου 4-5 για οποιονδήποτε άλλον.

Οι πληροφορίες που περιέχονται σε έναν πίνακα αληθείας είναι επίσης δυνατόν να εκφραστούν αλγεβρικά. Για να γίνει αυτό, πρέπει να παραθέσουμε πρώτα κάποιους ορισμούς:

όνομα

- *Όνομα* είναι μια μεταβλητή ή το συμπλήρωμα μιας μεταβλητής. Παραδείγματα: X, Y, X′, Y′.

όρος γινομένου

- *Όρος γινομένου* είναι ένα απλό όνομα ή ένα λογικό γινόμενο δύο ή περισσοτέρων ονομάτων. Παραδείγματα: Z′, W·X·Y, X·Y′·Z, W′·Y′·Z.

παράσταση αθροίσματος γινομένων

- Μια *παράσταση αθροίσματος γινομένων* είναι ένα λογικό άθροισμα όρων γινομένου. Παράδειγμα: Z′+W·X·Y+X·Y′·Z+W′·Y′·Z.

Πίνακας 4-5
Πίνακας αληθείας για μια συγκεκριμένη λογική συνάρτηση τριών μεταβλητών, F(X,Y,Z).

Γραμμή	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
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	1

- Ένας *όρος αθροίσματος* είναι ένα απλό όνομα ή ένα λογικό άθροισμα δύο ή περισσοτέρων ονομάτων. Παραδείγματα: Z′, W+X+Y, X+Y′+Z, W′+Y′+Z. *όρος αθροίσματος*
- Μια *παράσταση γινομένου αθροισμάτων* είναι ένα λογικό γινόμενο όρων αθροίσματος. Παράδειγμα: Z′·(W+X+Y)·(X+Y′+Z)·(W′+Y′+Z). *παράσταση γινομένου αθροισμάτων*
- *Κανονικός όρος* είναι ένας όρος γινομένου ή αθροίσματος στον οποίο καμία μεταβλητή δεν εμφανίζεται περισσότερες από μία φορά. Ένας μη κανονικός όρος μπορεί πάντα να απλοποιηθεί σε μια σταθερά ή έναν κανονικό όρο με τη χρήση ενός από τα θεωρήματα T3, T3′, T5, ή T5′. Παραδείγματα μη κανονικών όρων: W·X·X·Y′, W+W+X′+Y, X·X′·Y. Παραδείγματα κανονικών όρων: W·X·Y′, W+X′+Y. *κανονικός όρος*
- *Ελάχιστος όρος n* μεταβλητών είναι ένας κανονικός όρος γινομένου με *n* ονόματα. Υπάρχουν 2^n τέτοιοι όροι γινομένου. Παραδείγματα ελαχίστων όρων 4 μεταβλητών: W′·X′·Y′·Z′, W·X·Y′·Z, W′·X′·Y·Z′. *ελάχιστος όρος*
- *Μέγιστος όρος n* μεταβλητών είναι ένας κανονικός όρος αθροίσματος με *n* ονόματα. Υπάρχουν 2^n τέτοιοι όροι αθροίσματος. Παραδείγματα μεγίστων όρων 4 μεταβλητών: W′+X′+Y′+Z′, W+X′+Y′+Z, W′+X′+Y+Z′. *μέγιστος όρος*

Υπάρχει άμεση αντιστοιχία ανάμεσα στον πίνακα αληθείας, τους ελάχιστους όρους, και τους μέγιστους όρους. Ένας ελάχιστος όρος μπορεί να οριστεί ως όρος γινομένου που ισούται με 1 σε μία ακριβώς γραμμή του πίνακα αληθείας. Ομοίως, ένας μέγιστος όρος μπορεί να οριστεί ως όρος αθροίσματος που ισούται με 0 σε μία ακριβώς γραμμή του πίνακα αληθείας. Ο Πίνακας 4-6 παρουσιάζει αυτή την αντιστοιχία για έναν πίνακα αληθείας 3 μεταβλητών.

Ένας ελάχιστος όρος *n* μεταβλητών μπορεί να αναπαρασταθεί με έναν ακέραιο των *n* bit, που λέγεται *αριθμός ελαχίστου όρου*. Θα χρησιμοποιήσουμε την ονομασία *ελάχιστος όρος i* για να υποδηλώσουμε τον ελάχιστο όρο που αντιστοιχεί στη γραμμή *i* του πίνακα αληθείας. Στον ελάχιστο όρο *i*, μια συγκεκριμένη μεταβλητή εμφανίζεται ως συμπληρωματική αν το αντίστοιχο bit στη δυαδική αναπαράσταση του *i* είναι 0, διαφορετικά είναι μη συμπληρωματική. Για παράδειγμα, η γραμμή 5 έχει δυα- *αριθμός ελαχίστου όρου* *ελάχιστος όρος i*

Πίνακας 4-6
Ελάχιστοι όροι και μέγιστοι όροι για μια λογική συνάρτηση 3 μεταβλητών, F(X,Y,Z).

Γραμμή	X	Y	Z	F	*Ελάχιστος όρος*	*Μέγιστος όρος όρος*
0	0	0	0	F(0,0,0)	X′·Y′·Z′	X+Y+Z
1	0	0	1	F(0,0,1)	X′·Y′·Z	X+Y+Z′
2	0	1	0	F(0,1,0)	X′·Y·Z′	X+Y′+Z
3	0	1	1	F(0,1,1)	X′·Y·Z	X+Y′+Z′
4	1	0	0	F(1,0,0)	X·Y′·Z′	X′+Y+Z
5	1	0	1	F(1,0,1)	X·Y′·Z	X′+Y+Z′
6	1	1	0	F(1,1,0)	X·Y·Z′	X′+Y′+Z
7	1	1	1	F(1,1,1)	X·Y·Z	X′+Y′+Z′

δική αναπαράσταση 101 και ο αντίστοιχος ελάχιστος όρος είναι X·Y′·Z. Όπως μπορείτε να υποθέσετε, η αντιστοιχία για τους μέγιστους όρους είναι ακριβώς η αντίθετη: στο *μέγιστο όρο i*, μια συγκεκριμένη μεταβλητή εμφανίζεται ως συμπληρωματική αν το αντίστοιχο bit στη δυαδική αναπαράσταση του *i* είναι 1. Έτσι, ο μέγιστος όρος 5 (101) είναι X′+Y+Z′. Σημειώστε ότι όλα αυτά έχουν νόημα μόνο αν γνωρίζουμε τον αριθμό των μεταβλητών στον πίνακα αληθείας, δηλαδή 3 στα συγκεκριμένα παραδείγματα.

μέγιστος όρος i

Με βάση την αντιστοιχία ανάμεσα στον πίνακα αληθείας και τους ελάχιστους όρους, μπορούμε εύκολα να δημιουργήσουμε την αλγεβρική αναπαράσταση μιας λογικής συνάρτησης από τον πίνακα αληθείας της. Το *κανονικό άθροισμα* μιας λογικής συνάρτησης είναι ένα άθροισμα ελαχίστων όρων που αντιστοιχούν στις γραμμές του πίνακα αληθείας (συνδυασμοί εισόδων) για τις οποίες η συνάρτηση παράγει έξοδο 1. Για παράδειγμα, το κανονικό άθροισμα για τη λογική συνάρτηση του Πίνακα 4-5 είναι

κανονικό άθροισμα

$$F = \Sigma_{X,Y,Z}(0,3,4,6,7)$$
$$= X'\cdot Y'\cdot Z' + X'\cdot Y\cdot Z + X\cdot Y'\cdot Z' + X\cdot Y\cdot Z' + X\cdot Y\cdot Z$$

Εδώ, η σημειογραφία $\Sigma_{X,Y,Z}(0,3,4,6,7)$ είναι μια *λίστα ελαχίστων όρων* και σημαίνει "το άθροισμα των ελαχίστων όρων 0, 3, 4, 6, και 7 με μεταβλητές X, Y, και Z". Η λίστα ελαχίστων όρων είναι επίσης γνωστή ως το *σύνολο ενεργοποίησης* της λογικής συνάρτησης. Μπορείτε να φανταστείτε ότι κάθε ελάχιστος όρος "ενεργοποιεί" την έξοδο για ακριβώς ένα συνδυασμό εισόδων. Οποιαδήποτε λογική συνάρτηση μπορεί να γραφτεί ως κανονικό άθροισμα.

λίστα ελαχίστων όρων

σύνολο ενεργοποίησης

Το *κανονικό γινόμενο* μιας λογικής συνάρτησης είναι το γινόμενο των μεγίστων όρων που αντιστοιχούν στους συνδυασμούς εισόδων για τους οποίους η συνάρτηση παράγει έξοδο 0. Για παράδειγμα, το κανονικό γινόμενο για τη λογική συνάρτηση του Πίνακα 4-5 είναι

$$F = \Pi_{X,Y,Z}(1,2,5)$$
$$= (X+Y+Z')\cdot(X+Y'+Z)\cdot(X'+Y+Z')$$

Εδώ, η σημειογραφία $\Pi_{X,Y,Z}(1,2,5)$ είναι μια *λίστα μεγίστων όρων* και σημαίνει "το γινόμενο των μεγίστων όρων 1, 2, και 5 με μεταβλητές X, Y, και Z". Η λίστα μεγίστων όρων είναι επίσης γνωστή ως το *σύνολο απενεργοποίησης* της λογικής συνάρτησης. Μπορείτε να φανταστείτε ότι κάθε μέγιστος όρος "απενεργοποιεί" την έξοδο για ακριβώς ένα συνδυασμό εισόδων. Οποιαδήποτε λογική συνάρτηση μπορεί να γραφτεί ως κανονικό γινόμενο.

λίστα μεγίστων όρων

σύνολο απενεργοποίησης

κανονικό γινόμενο

Η μετατροπή μιας λίστας ελαχίστων όρων σε λίστα μεγίστων όρων και το αντίστροφο είναι εύκολη. Για μια συνάρτηση n μεταβλητών, οι δυνατοί αριθμοί ελαχίστου όρου και μεγίστου όρου περιλαμβάνονται στο σύνολο $\{0,1,...,2^n-1\}$; Μια λίστα ελαχίστων όρων και μεγίστων όρων περιέχει ένα υποσύνολο αυτών των αριθμών. Για να εναλλάσσεστε ανάμεσα σε αυτούς τους τύπους λίστας, πάρτε το συμπλήρωμα του συνόλου, για παράδειγμα,

$$\Sigma_{A,B,C}(0,1,2,3) = \Pi_{A,B,C}(4,5,6,7)$$
$$\Sigma_{X,Y}(1) = \Pi_{X,Y}(0,2,3)$$
$$\Sigma_{W,X,Y,Z}(1,2,3,5,7,11,13) = \Pi_{W,X,Y,Z}(4,6,8,9,10,12,14,15)$$

Έχουμε μάθει πλέον πέντε δυνατές αναπαραστάσεις μιας συνδυαστικής λογικής συνάρτησης:

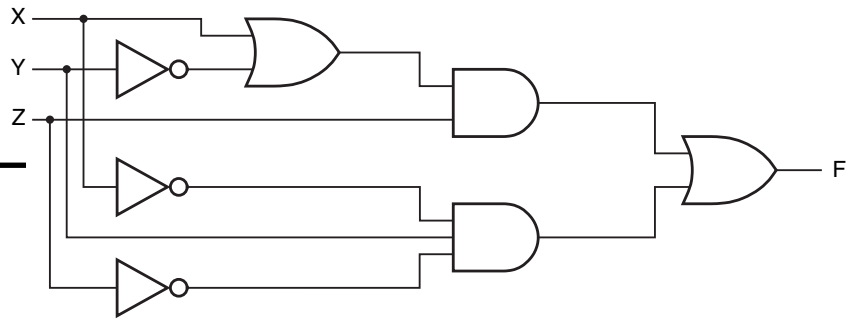
1. Πίνακας αληθείας.
2. Αλγεβρικό άθροισμα ελαχίστων όρων, δηλαδή το κανονικό άθροισμα.
3. Λίστα ελαχίστων όρων με τη χρήση της σημειογραφίας Σ.
4. Αλγεβρικό γινόμενο μεγίστων όρων, δηλαδή το κανονικό γινόμενο.
5. Λίστα μεγίστων όρων με τη χρήση της σημειογραφίας Π.

Κάθε μία από αυτές τις αναπαραστάσεις προσδιορίζει ακριβώς τις ίδιες πληροφορίες. Με δεδομένη οποιαδήποτε από αυτές τις αναπαραστάσεις, μπορούμε να παραγάγουμε τις άλλες τέσσερις χρησιμοποιώντας μια απλή μηχανική μέθοδο.

4.2 Ανάλυση συνδυαστικών κυκλωμάτων

Αναλύουμε ένα συνδυαστικό λογικό κύκλωμα λαμβάνοντας μια τυπική περιγραφή της λογικής του λειτουργίας. Από τη στιγμή που θα έχουμε την περιγραφή της λογικής λειτουργίας, μπορούμε να εκτελέσουμε μια σειρά από εργασίες:

- Μπορούμε να προσδιορίσουμε τη συμπεριφορά του κυκλώματος για διάφορους συνδυασμούς εισόδων.
- Μπορούμε να χειριστούμε μια αλγεβρική περιγραφή για να προτείνουμε διάφορες δομές κυκλωμάτων για τη λογική συνάρτηση.
- Μπορούμε να μετασχηματίσουμε μια αλγεβρική περιγραφή σε μια τυπική μορφή που να αντιστοιχεί σε μια διαθέσιμη δομή κυκλώματος.



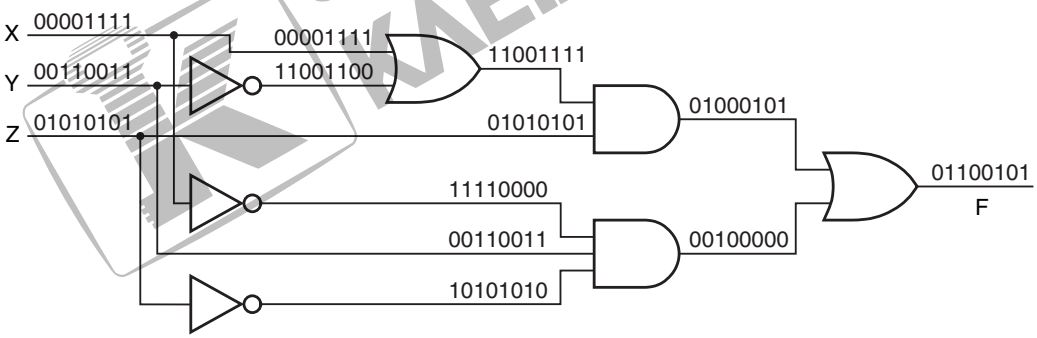
Εικόνα 4-9
Λογικό κύκλωμα τριών εισόδων και μίας εξόδου.

Για παράδειγμα, μια παράσταση αθροίσματος γινομένων αντιστοιχεί άμεσα στη δομή κυκλώματος που χρησιμοποιείται στις προγραμματιζόμενες λογικές διατάξεις (PLD).

- Μπορούμε να χρησιμοποιήσουμε μια αλγεβρική περιγραφή της λειτουργικής συμπεριφοράς του κυκλώματος στην ανάλυση ενός μεγαλύτερου συστήματος το οποίο περιλαμβάνει το κύκλωμα.

Με δεδομένο ένα λογικό διάγραμμα ενός συνδυαστικού κυκλώματος, όπως εκείνο της Εικόνας 4-9, υπάρχουν διάφοροι τρόποι για να πάρετε

Εικόνα 4-10 Έξοδοι πυλών που δημιουργούνται από όλους τους συνδυασμούς εισόδων.



ΜΙΑ ΛΙΓΟΤΕΡΟ ΕΞΑΝΤΛΗΤΙΚΗ ΜΕΘΟΔΟΣ

Μπορείτε εύκολα να πάρετε τα αποτελέσματα της Εικόνας 4-10 με τυπικά εργαλεία λογικής σχεδίασης που να περιλαμβάνουν ένα λογικό προσομοιωτή. Αρχικά σχεδιάζετε το σχηματικό διάγραμμα. Στη συνέχεια εφαρμόζετε τις εξόδους ενός δυαδικού απαριθμητή των 3 bit στις εισόδους X, Y και Z. (Οι περισσότεροι προσομοιωτές έχουν ενσωματωμένους απαριθμητές αυτού του τύπου για παρόμοια προβλήματα.) Ο απαριθμητής περνά επανειλημμένα από τους οκτώ δυνατούς συνδυασμούς εισόδων εκ περιτροπής, με την ίδια σειρά που φαίνεται στην εικόνα. Ο προσομοιωτής σάς επιτρέπει να αναπαραστήσετε γραφικά τις τιμές των σημάτων που προκύπτουν σε οποιοδήποτε σημείο του σχηματικού διαγράμματος, συμπεριλαμβανομένων των ενδιάμεσων σημείων και της εξόδου.

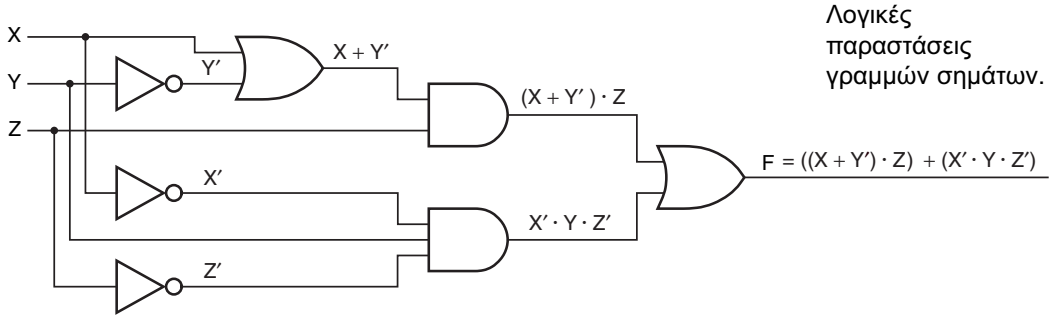
Γραμμή	*X*	*Y*	*Z*	*F*
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Πίνακας 4-7
Πίνακας αληθείας για το λογικό κύκλωμα της Εικόνας 4-9.

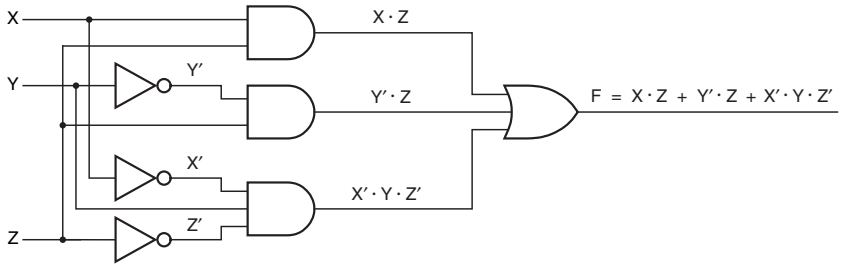
μια τυπική περιγραφή της λειτουργίας του κυκλώματος. Το πιο θεμελιώδες στοιχείο της λειτουργικής περιγραφής είναι ο πίνακας αληθείας.

Χρησιμοποιώντας μόνο τα βασικά αξιώματα της άλγεβρας μεταγωγής, μπορούμε να πάρουμε τον πίνακα αληθείας για ένα κύκλωμα n εισόδων περνώντας από όλους τους 2^n συνδυασμούς εισόδων. Για κάθε συνδυασμό εισόδων, προσδιορίζουμε όλες τις εξόδους των πυλών που παράγονται από το συγκεκριμένο συνδυασμό, μεταδίδοντας πληροφορίες από τις εισόδους του κυκλώματος στις εξόδους του κυκλώματος. Στην Εικόνα 4-10 εφαρμόζεται αυτή η "εξαντλητική" τεχνική στο κύκλωμα του παραδείγματός μας. Σε κάθε γραμμή σήματος του κυκλώματος είναι γραμμένη μια ακολουθία από οκτώ λογικές τιμές οι οποίες είναι παρούσες στη γραμμή αυτή όταν οι είσοδοι XYZ είναι 000, 001,..., 111. Ο πίνακας αληθείας μπορεί να γραφτεί με αντιγραφή της ακολουθίας εξόδου της τελικής πύλης OR, όπως φαίνεται στον Πίνακα 4-7. Μόλις πάρουμε τον πίνακα αληθείας του κυκλώματος, μπορούμε αν θέλουμε να γράψουμε επίσης απευθείας μια λογική παράσταση, δηλαδή το κανονικό άθροισμα ή το κανονικό γινόμενο.

Ο αριθμός των συνδυασμών εισόδων ενός λογικού κυκλώματος αυξάνεται εκθετικά ανάλογα με το πλήθος των εισόδων, έτσι η εξαντλητική προσέγγιση γρήγορα θα σας "εξαντλήσει". Αντί γι΄ αυτό, κανονικά χρησιμοποιούμε μια αλγεβρική προσέγγιση η πολυπλοκότητα



Εικόνα 4-11
Λογικές παραστάσεις γραμμών σημάτων.



Εικόνα 4-12 Κύκλωμα AND-OR δύο επιπέδων.

της οποίας είναι πιο γραμμική αναλογικά με το μέγεθος του κυκλώματος. Η μέθοδος είναι απλή: κατασκευάζουμε μια λογική παράσταση με παρενθέσεις που να αντιστοιχεί στους λογικούς τελεστές και τη δομή του κυκλώματος. Ξεκινάμε από τις εισόδους του κυκλώματος και μεταδίδουμε παραστάσεις μέσω των πυλών προς την έξοδο. Με βάση τα θεωρήματα της άλγεβρας μεταγωγής, μπορούμε να απλοποιήσουμε τις παραστάσεις καθώς προχωράμε ή ακόμα και να αναβάλουμε όλους τους αλγεβρικούς χειρισμούς έως ότου πάρουμε μια παράσταση εξόδου.

Στην Εικόνα 4-11 εφαρμόζεται η αλγεβρική τεχνική στο κύκλωμα του παραδείγματός μας. Η συνάρτηση εξόδου δίνεται στην έξοδο της τελικής πύλης OR:

$$F = ((X+Y')\cdot Z) + (X'\cdot Y\cdot Z')X$$

Για να προκύψει αυτή η παράσταση, δε χρησιμοποιούνται θεωρήματα της άλγεβρας μεταγωγής. Ωστόσο, μπορούμε να χρησιμοποιήσουμε τα θεωρήματα για να μετασχηματίσουμε αυτή την παράσταση σε μια άλλη μορφή. Για παράδειγμα, μπορούμε να πάρουμε ένα άθροισμα γινομένων "με εκτέλεση των επιμέρους πολλαπλασιασμών":

$$F = X\cdot Z+Y'\cdot Z + X'\cdot Y\cdot Z$$

Η νέα παράσταση ανταποκρίνεται σε ένα διαφορετικό κύκλωμα για την ίδια λογική συνάρτηση, όπως φαίνεται στην Εικόνα 4-12.

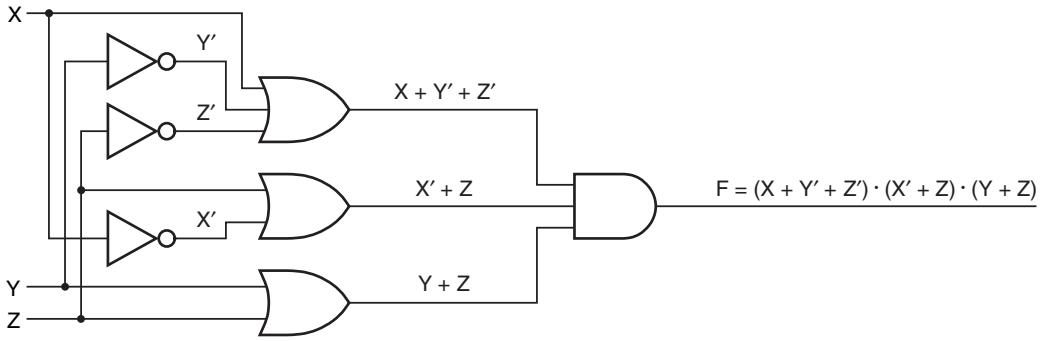
Παρομοίως, μπορούμε να "εκτελέσουμε τις επιμέρους προσθέσεις" στην αρχική παράσταση για να πάρουμε ένα γινόμενο αθροισμάτων:

$$\begin{aligned} F &= ((X+Y')\cdot Z) + (X'\cdot Y\cdot Z') \\ &=(X+Y'+X')\cdot(X+Y'+Y)\cdot(X+Y'+Z')\cdot(Z+X')\cdot(Z+Y)\cdot(Z+Z') \\ &= 1\cdot 1\cdot(X+Y'+Z')\cdot(X'+Z)\cdot(Y+Z)\cdot 1 \\ &= (X+Y'+Z')\cdot(X'+Z)\cdot(Y+Z) \end{aligned}$$

Το αντίστοιχο λογικό κύκλωμα φαίνεται στην Εικόνα 4-13.

Στο επόμενο παράδειγμα αλγεβρικής ανάλυσης χρησιμοποιείται ένα κύκλωμα με πύλες NAND και NOR, το οποίο φαίνεται στην Εικόνα 4-14. Η ανάλυση αυτή είναι ελαφρώς πιο πολύπλοκη σε σχέση με εκείνη του προηγούμενου παραδείγματος, καθώς κάθε πύλη παράγει το συμπλήρω-

Εικόνα 4-13 Κύκλωμα OR-AND δύο επιπέδων.



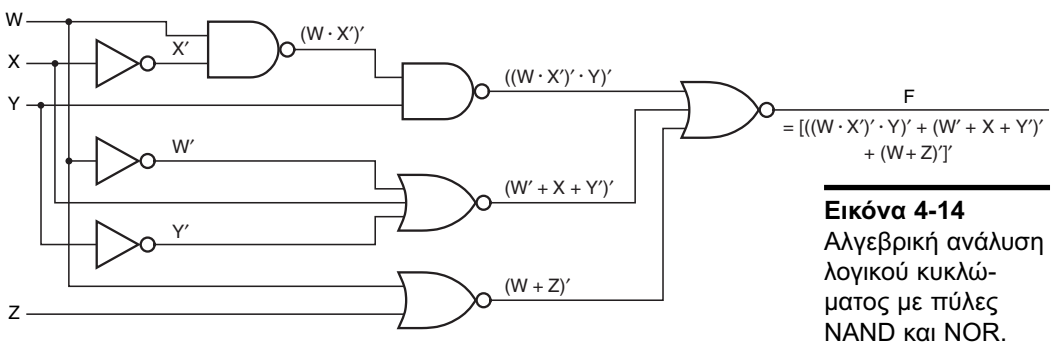
μα μιας επιμέρους παράστασης και όχι ένα απλό άθροισμα ή γινόμενο. Ωστόσο, η παράσταση εξόδου μπορεί να απλοποιηθεί με επαναλαμβανόμενη εφαρμογή του γενικευμένου θεωρήματος του DeMorgan:

$$
\begin{aligned}
F &= [((W\cdot X')'\cdot Y)' + (W' + X+Y')'+(W+Z)']' \\
&= ((W'+X)'+Y')'\cdot(W\cdot X'\cdot Y)'\cdot(W'\cdot Z')' \\
&= ((W\cdot X')'\cdot Y)\cdot(W'+X+Y')\cdot(W+Z) \\
&= ((W'+X)\cdot Y)\cdot(W'+X+Y')\cdot(W+Z)
\end{aligned}
$$

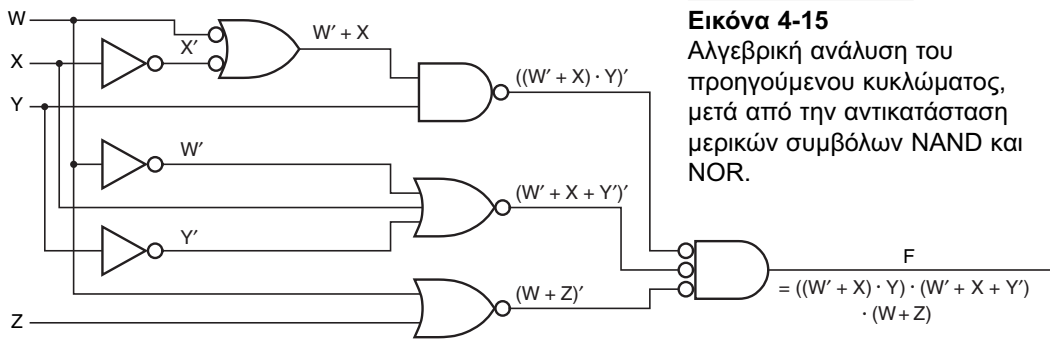
Αρκετά συχνά, το θεώρημα του DeMorgan είναι δυνατόν να εφαρμοστεί *γραφικά* για απλοποίηση της αλγεβρικής ανάλυσης. Από τις Εικόνες 4-3 και 4-4 φαίνεται ότι κάθε μία από τις πύλες NAND και NOR έχει δύο ισοδύναμα σύμβολα. Αν ξανασχεδιάσουμε προσεκτικά την Εικόνα 4-14, μπορούμε να απαλείψουμε μερικές από τις αντιστροφές κατά την ανάλυση χρησιμοποιώντας το θεώρημα T4 [(X′)′=X], όπως φαίνεται στην Εικόνα 4-15. Αυτός ο χειρισμός μάς καθοδηγεί κατευθείαν σε μια απλοποιημένη παράσταση εξόδου:

$$F = ((W' + X)\cdot Y)\cdot(W'+ X+Y')'\cdot(W+Z)$$

Οι Εικόνες 4-14 και 4-15 είναι απλώς δύο διαφορετικοί τρόποι σχεδίασης του ίδιου φυσικού λογικού κυκλώματος. Ωστόσο, όταν απλοποιούμε μια λογική παράσταση χρησιμοποιώντας τα θεωρήματα της άλγε-



Εικόνα 4-14 Αλγεβρική ανάλυση λογικού κυκλώματος με πύλες NAND και NOR.

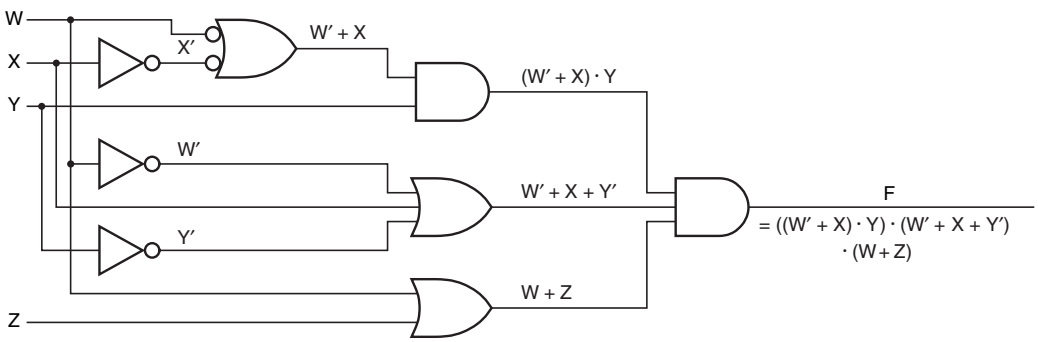


Εικόνα 4-15
Αλγεβρική ανάλυση του προηγούμενου κυκλώματος, μετά από την αντικατάσταση μερικών συμβόλων NAND και NOR.

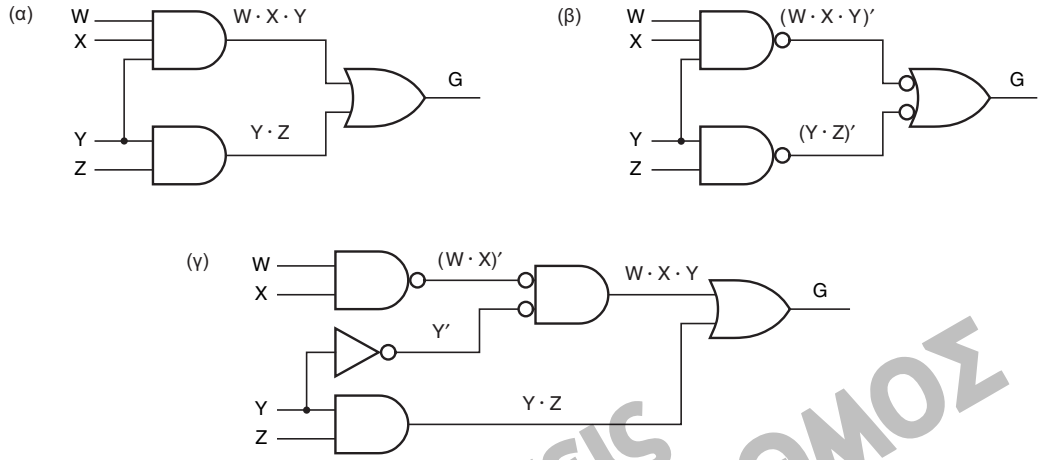
βρας μεταγωγής, παίρνουμε μια παράσταση που αντιστοιχεί σε ένα διαφορετικό φυσικό κύκλωμα. Για παράδειγμα, η απλοποιημένη προηγούμενη παράσταση αντιστοιχεί στο κύκλωμα της Εικόνας 4-16, το οποίο είναι από φυσική άποψη διαφορετικό από εκείνο των δύο προηγούμενων εικόνων. Επιπλέον, θα μπορούσαμε να εκτελέσουμε τους επιμέρους πολλαπλασιασμούς και προσθέσεις στην παράσταση για να πάρουμε παραστάσεις αθροισμάτων γινομένων και γινομένων αθροισμάτων που να αντιστοιχούν σε δύο ή περισσότερα διαφορετικά από φυσικής άποψης κυκλώματα για την ίδια λογική συνάρτηση.

Αν και χρησιμοποιήσαμε τις παραπάνω λογικές παραστάσεις για να μεταδώσουμε πληροφορίες σχετικά με τη φυσική δομή του κυκλώματος, δεν εφαρμόζουμε πάντα τη μέθοδο αυτή. Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε την παράσταση $G(W,X,Y,Z) = W \cdot X \cdot Y + Y \cdot Z$ για να περιγράψουμε οποιοδήποτε από τα κυκλώματα της Εικόνας 4-17. Κανονικά, ο μόνος σίγουρος τρόπος για να προσδιορίσουμε τη δομή ενός κυκλώματος είναι να παρατηρήσουμε το σχηματικό του διάγραμμα. Ωστόσο, για ορισμένες περιορισμένες κατηγορίες κυκλωμάτων, οι δομικές πληροφορίες είναι δυνατόν να προκύψουν από λογικές παραστάσεις. Για παράδειγμα, το κύκλωμα της περίπτωσης (α) θα μπορούσε να περιγραφεί χωρίς αναφορά στη σχεδίαση ως "κύκλωμα AND-OR δύο επιπέδων για την $W \cdot Y + Y \cdot Z$", ενώ το κύκλωμα της περίπτωσης (β) θα μπορού-

Εικόνα 4-16 Ένα διαφορετικό λογικό κύκλωμα για την ίδια λογική συνάρτηση.



Εικόνα 4-17 Τρία κυκλώματα για την G(W,X,Y,Z) = W·X·Y+Y·Z:
(α) AND-OR δύο επιπέδων, (β) NAND-NAND δύο επιπέδων,
(γ) κατά περίπτωση.



σε να περιγραφεί ως "κύκλωμα NAND-NAND δύο επιπέδων για την W·X·Y+Y·Z".

4.3 Σύνθεση συνδυαστικών κυκλωμάτων

4.3.1 Περιγραφές και σχεδιάσεις κυκλωμάτων

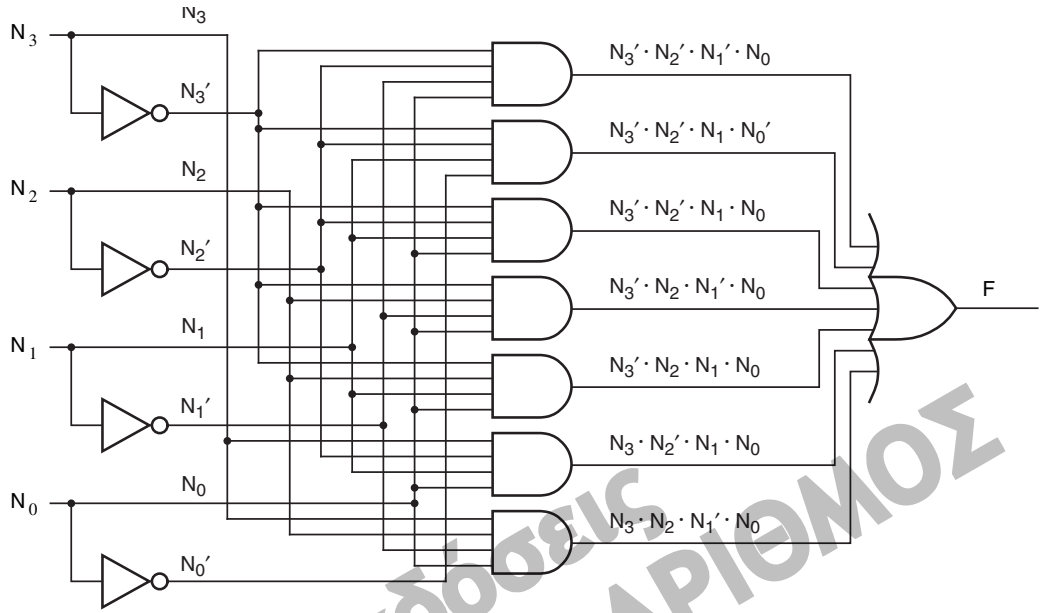
Ποιο είναι το σημείο εκκίνησης για τη σχεδίαση συνδυαστικών λογικών κυκλωμάτων; Συνήθως δίνουμε μια λεκτική περιγραφή ενός προβλήματος ή αναπτύσσουμε ένα τέτοιο πρόβλημα μόνοι μας. Σε κάποιες περιπτώσεις η περιγραφή είναι μια λίστα συνδυασμών εισόδων για τους οποίους ένα σήμα μπορεί να είναι ενεργοποιημένο ή απενεργοποιημένο, δηλαδή το λεκτικό ισοδύναμο ενός πίνακα αληθείας ή της σημειογραφίας Σ ή Π που παρουσιάσαμε πριν. Για παράδειγμα, η περιγραφή ενός ανιχνευτή πρώτων αριθμών των 4 bit θα μπορούσε να είναι η εξής: "Με δεδομένο ένα συνδυασμό εισόδων $N=N_3N_2N_1N_0$ των 4 bit, η συνάρτηση αυτή παράγει έξοδο 1 για N=1, 2, 3, 5, 7, 11, 13, διαφορετικά παράγει έξοδο 0". Μια λογική συνάρτηση που περιγράφεται με τον τρόπο αυτόν μπορεί να σχεδιαστεί απευθείας από την παράσταση του κανονικού αθροίσματος ή γινομένου. Για τον ανιχνευτή πρώτων αριθμών, έχουμε:

$$
\begin{aligned}
F &= \Sigma_{N_3, N_2, N_1, N_0}(1,2,3,5,7,11,13) \\
&= N_3'\cdot N_2'\cdot N_1'\cdot N_0 + N_3'\cdot N_2'\cdot N_1\cdot N_0' + N_3'\cdot N_2'\cdot N_1\cdot N_0 + N_3'\cdot N_2\cdot N_1'\cdot N_0 \\
&\quad + N_3'\cdot N_2\cdot N_1\cdot N_0 + N_3\cdot N_2'\cdot N_1\cdot N_0 + N_3\cdot N_2\cdot N_1'\cdot N_0
\end{aligned}
$$

Το αντίστοιχο κύκλωμα φαίνεται στην Εικόνα 4-18.

Συχνότερα, περιγράφουμε μια λογική συνάρτηση χρησιμοποιώντας τις λέξεις "και", "ή" και "όχι (δεν)" της γλώσσας. Για παράδειγμα, θα

Εικόνα 4-18 Σχεδίαση κανονικού αθροίσματος για ανιχνευτή πρώτων αριθμών των 4 bit.



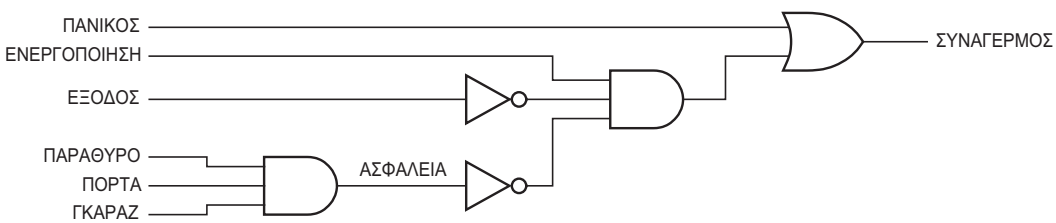
μπορούσαμε να περιγράψουμε ένα κύκλωμα συναγερμού ως εξής: "Η έξοδος ΣΥΝΑΓΕΡΜΟΣ είναι 1 αν η είσοδος ΠΑΝΙΚΟΣ είναι 1 ή αν η είσοδος ΕΝΕΡΓΟΠΟΙΗΣΗ είναι 1, η είσοδος ΕΞΟΔΟΣ είναι 0, και το σπίτι δεν είναι ασφαλές. Το σπίτι είναι ασφαλές αν και οι τρεις είσοδοι ΠΑΡΑΘΥΡΟ, ΠΟΡΤΑ και ΓΚΑΡΑΖ είναι 1". Μια τέτοια περιγραφή μπορεί να μεταφραστεί άμεσα σε αλγεβρικές παραστάσεις:

ΣΥΝΑΓΕΡΜΟΣ = ΠΑΝΙΚΟΣ + ΕΝΕΡΓΟΠΟΙΗΣΗ · ΕΞΟΔΟΣ′ · ΑΣΦΑΛΕΙΑ′

ΑΣΦΑΛΕΙΑ = ΠΑΡΑΘΥΡΟ · ΠΟΡΤΑ · ΓΚΑΡΑΖ

ΣΥΝΑΓΕΡΜΟΣ = ΠΑΝΙΚΟΣ + ΕΝΕΡΓΟΠΟΙΗΣΗ · ΕΞΟΔΟΣ′ · ΠΑΡΑΘΥΡΟ · ΠΟΡΤΑ · ΓΚΑΡΑΖ)′

Σημειώστε ότι χρησιμοποιήσαμε την ίδια μέθοδο στην άλγεβρα μεταγωγής όπως και στην κοινή άλγεβρα για τη διατύπωση μιας σύνθετης πα-



Εικόνα 4-19 Κύκλωμα συναγερμού που παράγεται από λογική παράσταση.

ράστασης: ορίσαμε μια βοηθητική μεταβλητή ΑΣΦΑΛΕΙΑ για να απλοποιήσουμε την πρώτη εξίσωση, αναπτύξαμε μια παράσταση για την ΑΣΦΑΛΕΙΑ, και εφαρμόσαμε αντικατάσταση για να πάρουμε την τελική παράσταση. Μπορούμε εύκολα να σχεδιάσουμε ένα κύκλωμα χρησιμοποιώντας πύλες AND, OR, και NOT οι οποίες υλοποιούν την τελική παράσταση, όπως φαίνεται στην Εικόνα 4-19. Ένα κύκλωμα *υλοποιεί* (“πραγματοποιεί”) μια παράσταση όταν η συνάρτηση εξόδου ισούται με τη συγκεκριμένη παράσταση, οπότε το κύκλωμα ονομάζεται *υλοποίηση* της συνάρτησης.

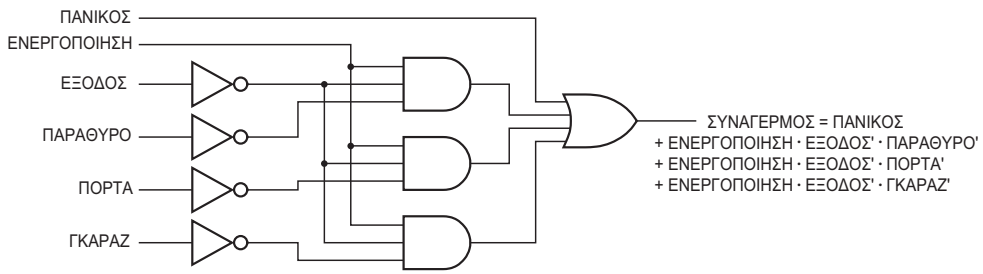
υλοποιώ

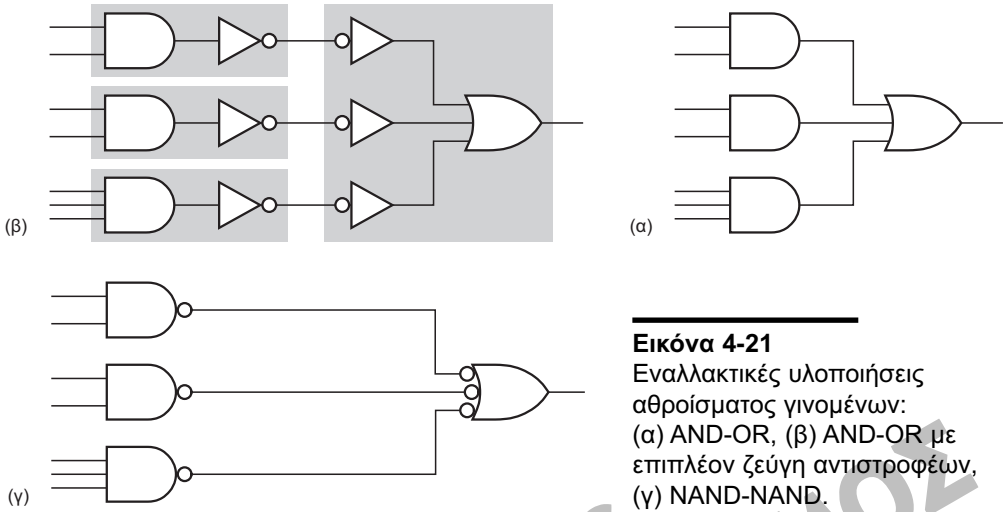
υλοποίηση

Αν έχουμε στη διάθεσή μας μια παράσταση, οποιαδήποτε κι αν είναι αυτή, για μια λογική συνάρτηση, μπορούμε να κάνουμε και άλλα πράγματα εκτός από τη δημιουργία ενός κυκλώματος απευθείας από την παράσταση. Μπορούμε να χειριστούμε την παράσταση για να πάρουμε διαφορετικά κυκλώματα. Για παράδειγμα, μπορούμε να εκτελέσουμε τους επιμέρους πολλαπλασιασμούς στην παράσταση ΣΥΝΑΓΕΡΜΟΣ παραπάνω για να πάρουμε το κύκλωμα αθροίσματος γινομένων της Εικόνας 4-20. Επίσης, αν ο αριθμός των μεταβλητών δεν είναι υπερβολικά μεγάλος, μπορούμε να κατασκευάσουμε τον πίνακα αληθείας της παράστασης και να χρησιμοποιήσουμε κάποια από τις μεθόδους σύνθεσης που εφαρμόζονται στους πίνακες αληθείας, συμπεριλαμβανομένων των μεθόδων κανονικού αθροίσματος ή γινομένου που περιγράφονται παραπάνω και των μεθόδων ελαχιστοποίησης που περιγράφονται παρακάτω.

Γενικά, είναι ευκολότερο να περιγράψουμε ένα κύκλωμα με λέξεις χρησιμοποιώντας λογικούς συνδέσμους και να γράψουμε την αντίστοιχη λογική παράσταση από το να γράψουμε έναν πλήρη πίνακα αληθείας, ειδικά αν ο αριθμός των μεταβλητών είναι μεγάλος. Ωστόσο, μερικές φορές πρέπει να δουλέψουμε με ανακριβείς λεκτικές περιγραφές των λογικών συναρτήσεων, όπως π.χ. “Η έξοδος ΣΦΑΛΜΑ θα είναι 1 αν οι είσοδοι ΑΝΕΒΑΣΜΑ_ΤΑΧΥΤΗΤΑΣ, ΚΑΤΕΒΑΣΜΑ_ΤΑΧΥΤΗΤΑΣ, και ΕΛΕΓΧΟΣ_ΤΑΧΥΤΗΤΑΣ είναι αντιφατικές μεταξύ τους”. Σε αυτή την περίπτωση, η προσέγγιση με τον πίνακα αληθείας είναι καλύτερη επειδή μας επιτρέπει να προσδιορίσουμε την απαιτούμενη έξοδο για κάθε συνδυασμό εισόδων, με βάση τη γνώση μας και την κατανόηση του περι-

Εικόνα 4-20 Έκδοση του κυκλώματος συναγερμού με άθροισμα γινομένων.





Εικόνα 4-21
Εναλλακτικές υλοποιήσεις αθροίσματος γινομένων: (α) AND-OR, (β) AND-OR με επιπλέον ζεύγη αντιστροφέων, (γ) NAND-NAND.

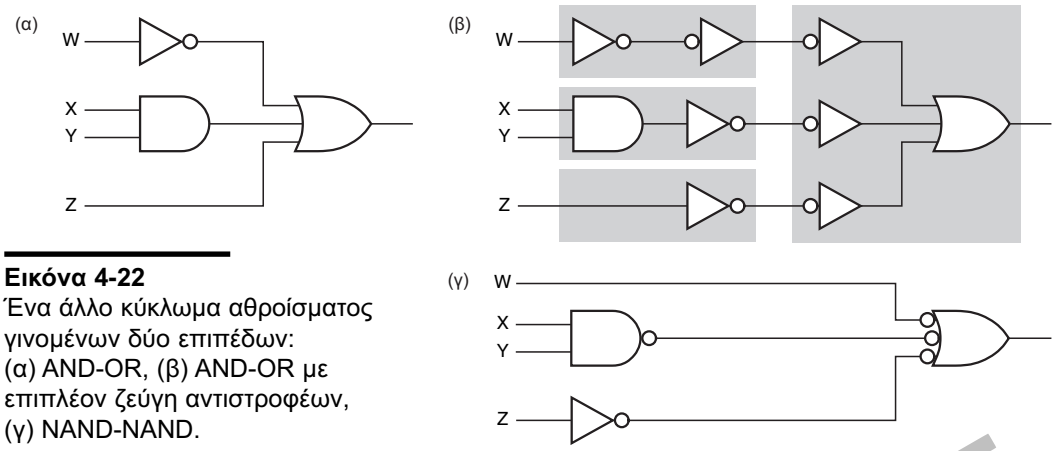
βάλλοντος του προβλήματος (π.χ. στο αυτοκίνητο, τα φρένα δεν είναι δυνατόν να χρησιμοποιηθούν αν δεν έχουμε κατεβάσει ταχύτητα).

4.3.2 Χειρισμοί κυκλωμάτων

Οι μέθοδοι σχεδίασης που έχουμε περιγράψει μέχρι τώρα χρησιμοποιούν πύλες AND, OR, και NOT. Μπορούμε επίσης να θέλουμε να χρησιμοποιούμε τις πύλες NAND και NOR, αφού είναι γρηγορότερες από τις AND και OR στις περισσότερες τεχνολογίες. Ωστόσο, οι περισσότεροι άνθρωποι δεν αναπτύσσουν λογικές προτάσεις χρησιμοποιώντας τα συνδετικά NAND και NOR. Αυτό σημαίνει ότι μάλλον δε θα λέγατε: “Δε θα βγούμε μαζί αν δεν είσαι καθαρός ή πλούσιος και επίσης αν δεν είσαι έξυπνος ή φιλικός”. Θα ήταν πιο φυσικό για εσάς να πείτε: “Θα βγούμε μαζί αν είσαι καθαρός και πλούσιος ή αν είσαι έξυπνος και φιλικός”. Έτσι, με δεδομένη μια πιο “φυσική” λογική έκφραση, χρειαζόμαστε κάποιους τρόπους για να τη μεταφράσουμε σε άλλες μορφές.

Μπορούμε να μεταφράσουμε οποιαδήποτε λογική παράσταση σε μια ισοδύναμη παράσταση αθροίσματος γινομένων απλώς εκτελώντας τους επιμέρους πολλαπλασιασμούς σε αυτήν. Όπως φαίνεται στην Εικόνα 4-21 (α), μια τέτοια παράσταση είναι δυνατόν να πραγματοποιηθεί κατευθείαν με πύλες AND και OR. Οι αντιστροφείς που απαιτούνται για τις συμπληρωματικές εισόδους δεν εμφανίζονται στην εικόνα.

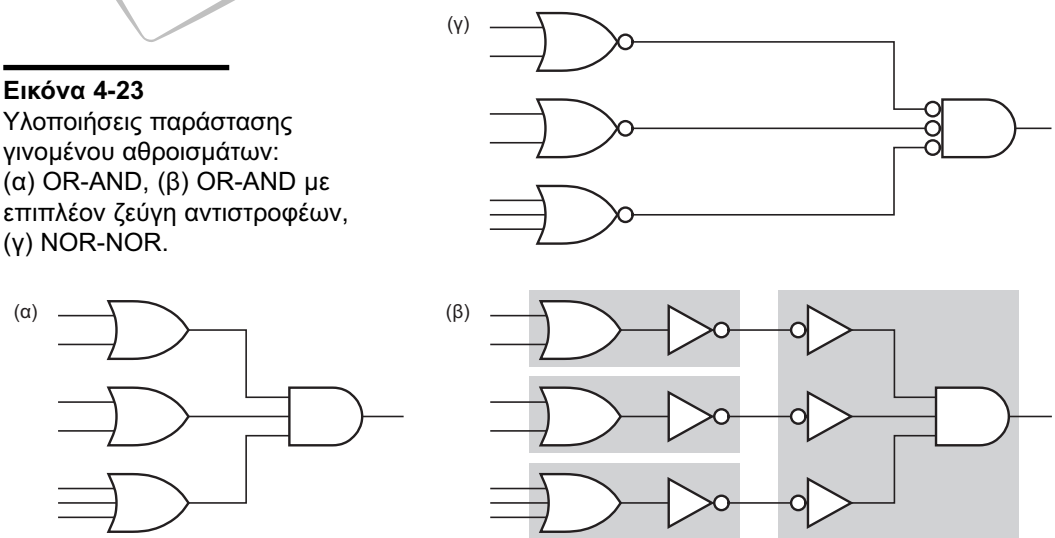
Όπως φαίνεται στην Εικόνα 4-21 (β), μπορούμε να παρεμβάλουμε ένα ζεύγος αντιστροφέων ανάμεσα σε κάθε έξοδο πύλης AND και στην είσοδο της αντίστοιχης πύλης OR σε ένα κύκλωμα AND-OR δύο επιπέδων. Σύμφωνα με το θεώρημα Τ4, αυτοί οι αντιστροφείς δεν έχουν καμία επίδραση στη λειτουργία εξόδου του κυκλώματος. Έχουμε μάλιστα σχεδιάσει το δεύτερο αντιστροφέα κάθε ζεύγους με τη φυσαλίδα αντιστροφής στην είσοδό του για να παρέχει μια γραφική υπενθύμιση ότι οι αντι-



Εικόνα 4-22
Ένα άλλο κύκλωμα αθροίσματος γινομένων δύο επιπέδων:
(α) AND-OR, (β) AND-OR με επιπλέον ζεύγη αντιστροφέων,
(γ) NAND-NAND.

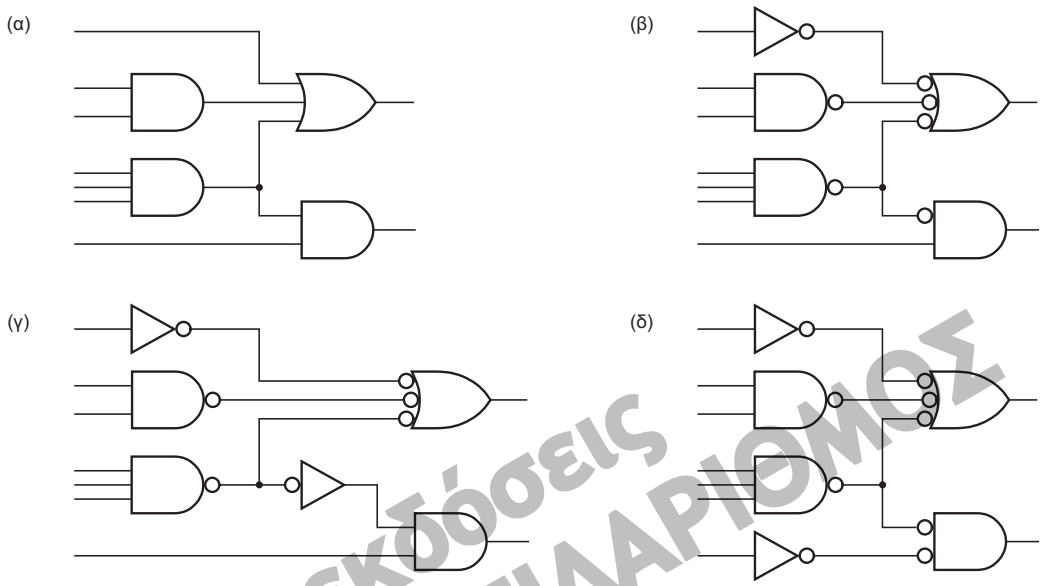
στροφείς αλληλοαναιρούνται. Ωστόσο, αν αυτοί οι αντιστροφείς είναι ενσωματωμένοι στις πύλες AND και OR, καταλήγουμε να έχουμε πύλες AND-NOT στο πρώτο επίπεδο και μια πύλη NOT-OR στο δεύτερο επίπεδο. Αυτά είναι απλώς δύο διαφορετικά σύμβολα για τον ίδιο τύπο πύλης, δηλαδή μιας πύλης NAND. Έτσι λοιπόν, ένα *κύκλωμα* AND-OR δύο επιπέδων μπορεί να μετατραπεί σε ένα *κύκλωμα* NAND-NAND δύο επιπέδων με απλή αντικατάσταση των πυλών.

Αν οποιοσδήποτε όρος γινομένου της παράστασης αθροίσματος γινομένων περιέχει απλώς ένα και μόνο όνομα, τότε μπορούμε να κερδίσουμε ή να χάσουμε αντιστροφείς κατά το μετασχηματισμό από AND-OR σε NAND-NAND. Για παράδειγμα, η Εικόνα 4-22 είναι ένα παράδειγμα όπου δε χρειάζεται πλέον αντιστροφέας στην είσοδο W, αλλά πρέπει να προστεθεί ένας αντιστροφέας στην είσοδο Z.



Εικόνα 4-23
Υλοποιήσεις παράστασης γινομένου αθροισμάτων:
(α) OR-AND, (β) OR-AND με επιπλέον ζεύγη αντιστροφέων,
(γ) NOR-NOR.

Εικόνα 4-24 Χειρισμοί λογικών συμβόλων: (α) αρχικό κύκλωμα, (β) μετασχηματισμός με μια μη τυπική πύλη, (γ) χρήση αντιστροφέα για την απαλοιφή μιας μη τυπικής πύλης, (δ) προτιμώμενη τοποθέτηση αντιστροφέων.



Έχουμε δείξει ότι οποιαδήποτε παράσταση αθροίσματος γινομένων μπορεί να πραγματοποιηθεί με δύο τρόπους: ως κύκλωμα AND-OR ή ως κύκλωμα NAND-NAND. Η δυική αυτής της πρότασης είναι επίσης αληθής: οποιαδήποτε παράσταση γινομένου αθροισμάτων μπορεί να υλοποιηθεί ως *κύκλωμα* OR-AND ή ως *κύκλωμα* NOR-NOR. Στην Εικόνα 4-23 φαίνεται ένα παράδειγμα. Οποιαδήποτε λογική παράσταση μπορεί να μεταφραστεί σε μια ισοδύναμη παράσταση γινομένου αθροισμάτων εκτελώντας τις επιμέρους προσθέσεις σε αυτήν, και γι΄ αυτόν το λόγο έχει δύο υλοποιήσεις κυκλώματος, δηλαδή OR-AND και NOR-NOR.

Το ίδιο είδος χειρισμών μπορεί να εφαρμοστεί σε οποιαδήποτε λογικά κυκλώματα. Για παράδειγμα, στην Εικόνα 4-24(α) φαίνεται ένα κύκλωμα κατασκευασμένο από πύλες AND και OR. Μετά την προσθήκη ζευγών αντιστροφέων, παίρνουμε το κύκλωμα της περίπτωσης (β). Ωστόσο, μία από τις πύλες, δηλαδή μια πύλη AND δύο εισόδων με μια αντεστραμμένη είσοδο, δεν είναι τυπικής μορφής. Μπορούμε να χρησιμοποιήσουμε έναν ξεχωριστό αντιστροφέα όπως φαίνεται στην περίπτωση (γ) για να πάρουμε ένα κύκλωμα το οποίο χρησιμοποιεί μόνο τυπικές πύλες δηλαδή NAND, AND, και αντιστροφείς. Μάλιστα, ένας καλύτερος τρόπος για να χρησιμοποιήσετε τον αντιστροφέα φαίνεται στην περίπτωση (δ), όπου εξαλείφεται ένα επίπεδο καθυστέρησης πύλης και η κάτω πύλη γίνεται NOR αντί για AND. Στις περισσότερες λογικές τεχνολογίες, οι αναστρέφουσες πύλες όπως οι NAND και NOR είναι ταχύτερες από τις μη αναστρέφουσες πύλες όπως οι AND και OR.

ΓΙΑΤΙ ΕΛΑΧΙΣΤΟΠΟΙΗΣΗ; Η ελαχιστοποίηση είναι σημαντικό βήμα τόσο στη σχεδίαση ASIC όσο και στη σχεδίαση σε PLD. Οι επιπλέον πύλες και είσοδοι πυλών απαιτούν περισσότερο χώρο σε ένα ολοκληρωμένο ASIC και επομένως αυξάνουν το κόστος. Ο αριθμός των πυλών μιας διάταξης PLD είναι σταθερός, γι' αυτό ίσως νομίσετε ότι οι επιπλέον θύρες είναι χωρίς κόστος. Αυτό είναι αλήθεια έως ότου σας τελειώσουν και χρειαστεί να κάνετε αναβάθμιση σε μια μεγαλύτερη, πιο αργή, και πιο ακριβή διάταξη PLD. Ευτυχώς, τα περισσότερα εργαλεία λογισμικού για τη σχεδίαση ASIC και PLD έχουν ενσωματωμένο ένα πρόγραμμα ελαχιστοποίησης. Ο στόχος των Ενοτήτων 4.3.3 έως 4.3.8 είναι να σας δώσουν μια αίσθηση για το πώς λειτουργεί η ελαχιστοποίηση.

4.3.3 Ελαχιστοποίηση συνδυαστικών κυκλωμάτων

Συχνά είναι αντιοικονομικό να υλοποιούμε ένα λογικό κύκλωμα κατευθείαν από την πρώτη λογική παράσταση που μας έρχεται στο μυαλό. Οι παραστάσεις κανονικών αθροισμάτων και γινομένων είναι ιδιαίτερα ακριβές, επειδή ο αριθμός των δυνατών ελαχίστων όρων ή μεγίστων όρων (και ως εκ τούτου των πυλών) αυξάνεται εκθετικά ανάλογα με τον αριθμό των μεταβλητών. Η *ελαχιστοποίηση* ενός συνδυαστικού κυκλώματος γίνεται με τον περιορισμό του αριθμού και του μεγέθους των πυλών που χρειάζονται για να το κατασκευάσουμε.

ελαχιστοποίηση

Οι παραδοσιακές μέθοδοι ελαχιστοποίησης συνδυαστικών κυκλωμάτων που θα μελετήσουμε έχουν ως σημείο εκκίνησης έναν πίνακα αληθείας ή, ισοδύναμα, μια λίστα ελαχίστων όρων ή μεγίστων όρων. Αν μας δοθεί μια λογική συνάρτηση που δεν έχει διατυπωθεί με αυτή τη μορφή, τότε πρέπει να τη μετατρέψουμε σε μια κατάλληλη μορφή προτού εφαρμόσουμε αυτές τις μεθόδους. Για παράδειγμα, αν μας δοθεί μια οποιαδήποτε λογική παράσταση, μπορούμε να την αποτιμήσουμε για κάθε συνδυασμό εισόδων για να κατασκευάσουμε τον πίνακα αληθείας. Οι μέθοδοι ελαχιστοποίησης περιορίζουν το κόστος ενός κυκλώματος AND-OR, OR-AND, NAND-NAND, ή NOR-NOR με τρεις τρόπους:

1. Με ελαχιστοποίηση του αριθμού των πυλών πρώτου επιπέδου.
2. Με ελαχιστοποίηση του αριθμού των εισόδων κάθε πύλης πρώτου επιπέδου.
3. Με ελαχιστοποίηση του αριθμού των εισόδων στην πύλη δευτέρου επιπέδου. Αυτό είναι στην πραγματικότητα μια παρενέργεια του πρώτου περιορισμού.

Ωστόσο, οι μέθοδοι ελαχιστοποίησης δε λαμβάνουν υπόψη το κόστος των αντιστροφέων εισόδου, επειδή γίνεται η παραδοχή ότι τόσο οι αληθείς όσο και οι συμπληρωματικές εκδοχές όλων των μεταβλητών εισόδου είναι διαθέσιμες. Αφού αυτό δεν ισχύει πάντα για τη σχεδίαση σε επίπεδο πυλών ή ASIC, είναι ιδιαίτερα κατάλληλες για τη σχεδίαση σε προγραμματιζόμενες λογικές διατάξεις (PLD), αφού οι PLD διαθέτουν "δωρεάν" τόσο αληθείς όσο και συμπληρωματικές εκδοχές όλων των μεταβλητών εισόδου.

Οι περισσότερες μέθοδοι ελαχιστοποίησης βασίζονται στη γενίκευση των συνδυαστικών θεωρημάτων T10 και T10′:

όρος γινομένου · Y + όρος γινομένου · Y′ = όρος γινομένου

(όρος αθροίσματος + Y) · (όρος αθροίσματος + Y′) = όρος αθροίσματος

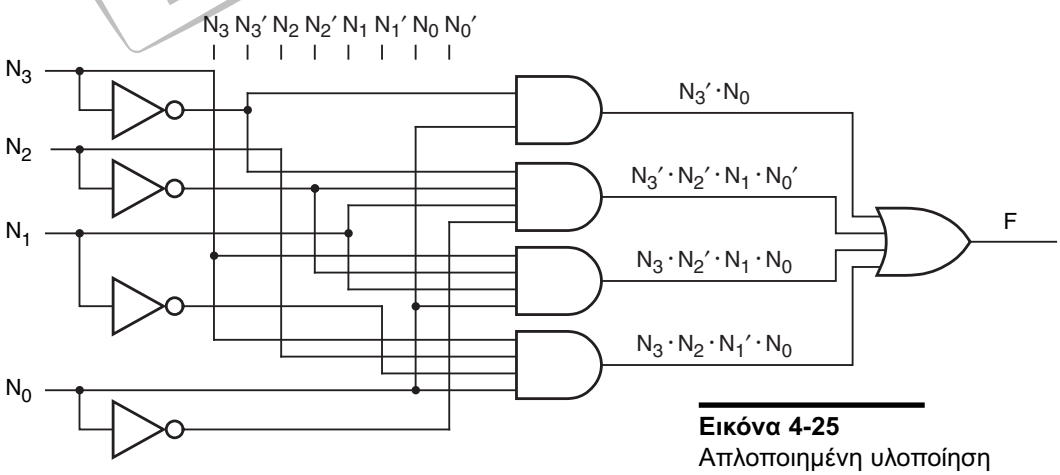
Αυτό σημαίνει ότι, αν δύο όροι γινομένου ή αθροίσματος διαφέρουν μόνο κατά το ότι μια μεταβλητή είναι συμπληρωματική ή όχι, μπορούμε να τους συνδυάσουμε μαζί σε έναν και μοναδικό όρο με μία μεταβλητή λιγότερη. Έτσι εξοικονομούμε μια πύλη, ενώ η πύλη που απομένει έχει μια είσοδο λιγότερη.

Μπορούμε να εφαρμόσουμε αυτή την αλγεβρική μέθοδο όσες φορές χρειάζεται για να συνδυάσουμε τους ελάχιστους όρους 1, 3, 5, και 7 του ανιχνευτή πρώτων αριθμών που παρουσιάζεται στην Εικόνα 4-18 της Ενότητας 4.3.1 μεταξύ τους:

$$
\begin{aligned}
F &= \Sigma_{N_3,\ N_2,\ N_1,\ N_0}\,(1,2,3,5,7,2,11,13)\\
&= N_3'\cdot N_2'\cdot N_1'\cdot N_0 + N_3'\cdot N_2'\cdot N_1\cdot N_0 + N_3'\cdot N_2\cdot N_1'\cdot N_0 + N_3'\cdot N_2\cdot N_1\cdot N_0 + \ldots\\
&= (N_3'\cdot N_2'\cdot N_1'\cdot N_0 + N_3'\cdot N_2'\cdot N_1\cdot N_0) + (N_3'\cdot N_2\cdot N_1'\cdot N_0 + N_3'\cdot N_2\cdot N_1\cdot N_0) + \ldots\\
&= N_3'\cdot N_2'\cdot N_0 + N_3'\cdot N_2\cdot N_0 + \ldots\\
&= N_3'\cdot N_0 + \ldots
\end{aligned}
$$

Το κύκλωμα που προκύπτει εμφανίζεται στην Εικόνα 4-25. Έχει τρεις λιγότερες πύλες, ενώ μία από τις πύλες που απομένουν έχει δύο εισόδους λιγότερες.

Αν είχαμε δουλέψει λίγο περισσότερο στην προηγούμενη παράσταση, θα είχαμε εξοικονομήσει ένα ακόμη ζεύγος εισόδων πύλης πρώτου επιπέδου, αν και δε θα εξοικονομούσαμε άλλες πύλες. Είναι δύσκολο να βρείτε όρους οι οποίοι μπορούν να συνδυαστούν μεταξύ τους μέσα σε έναν κυκεώνα αλγεβρικών συμβόλων. Στην επόμενη ενότητα θα ξεκινή-



Εικόνα 4-25
Απλοποιημένη υλοποίηση αθροίσματος γινομένων για τον ανιχνευτή πρώτων αριθμών των 4 bit.

σουμε την εξερεύνηση μιας μεθόδου ελαχιστοποίησης η οποία ταιριάζει περισσότερο στον ανθρώπινο τρόπο σκέψης. Το σημείο εκκίνησης θα είναι το γραφικό ισοδύναμο ενός πίνακα αληθείας.

4.3.4 Χάρτες Karnaugh

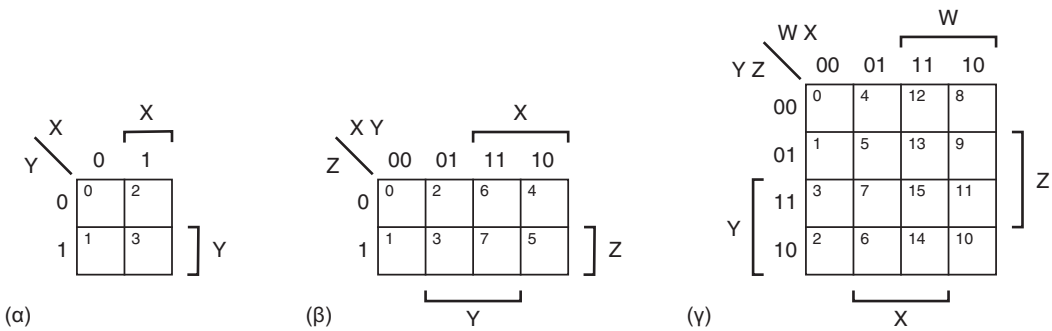
χάρτης Karnaugh

Ο *χάρτης Karnaugh* είναι μια γραφική αναπαράσταση του πίνακα αληθείας μιας λογικής παράστασης. Η Εικόνα 4-26 δείχνει ένα χάρτη Karnaugh για λογικές συναρτήσεις 2, 3, και 4 μεταβλητών. Ο χάρτης για μια λογική συνάρτηση *n* εισόδων είναι ένας πίνακας με 2^n κελιά, ένα για κάθε δυνατό συνδυασμό εισόδων ή ελάχιστο όρο.

Οι γραμμές και οι στήλες ενός χάρτη Karnaugh ονομάζονται έτσι ώστε ο συνδυασμός εισόδων για οποιοδήποτε κελί να προσδιορίζεται εύκολα από τις επικεφαλίδες της γραμμής και της στήλης του συγκεκριμένου κελιού. Ο μικρός αριθμός μέσα σε κάθε κελί είναι ο αντίστοιχος αριθμός ελαχίστου όρου του πίνακα αληθείας, με την παραδοχή ότι οι είσοδοι του πίνακα αληθείας ονομάζονται αλφαβητικά από τα αριστερά προς τα δεξιά (π.χ. X, Y, Z) και οι γραμμές αριθμούνται με δυαδική σειρά αρίθμησης, όπως όλα τα παραδείγματα αυτού του βιβλίου. Για παράδειγμα, το κελί 13 του χάρτη 4 μεταβλητών αντιστοιχεί στη γραμμή του πίνακα αληθείας στην οποία W X Y Z = 1101.

Όταν σχεδιάζουμε το χάρτη Karnaugh μιας δεδομένης συνάρτησης, κάθε κελί του πίνακα περιέχει τις πληροφορίες της αριθμημένης γραμμής του πίνακα αληθείας της συνάρτησης, δηλαδή 0 αν η συνάρτηση είναι 0 για το συγκεκριμένο συνδυασμό εισόδων και 1 σε κάθε άλλη περίπτωση.

Σε αυτό το βιβλίο, χρησιμοποιούμε δύο πλεονάζουσες αριθμήσεις για τις γραμμές και τις στήλες του χάρτη. Για παράδειγμα, ας θεωρήσουμε το χάρτη 4 μεταβλητών της Εικόνας 4-26(γ). Οι στήλες ονομάζονται με τους τέσσερις δυνατούς συνδυασμούς των W και X, W X = 00, 01, 11, και 10. Παρόμοια, οι γραμμές ονομάζονται με τους συνδυασμούς των Y και Z. Αυτές οι ετικέτες μάς δίνουν όλες τις πληροφορίες που χρειαζόμαστε. Ωστόσο, χρησιμοποιούμε επίσης αγκύλες για να συσχετίσουμε τέσσερις περιοχές του χάρτη με τις τέσσερις μεταβλητές. Κάθε περιοχή σε αγκύ-



Εικόνα 4-26 Χάρτες Karnaugh: (α) 2 μεταβλητών, (β) 3 μεταβλητών, (γ) 4 μεταβλητών.

λες είναι εκείνο το μέρος του χάρτη στο οποίο η υποδεικνυόμενη μεταβλητή είναι 1. Προφανώς, οι αγκύλες μεταφέρουν τις ίδιες πληροφορίες που δίνονται από τις ετικέτες των γραμμών και των στηλών.

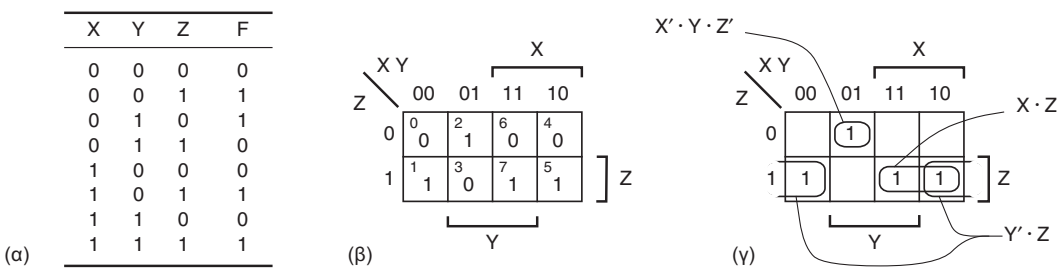
Όταν σχεδιάζουμε ένα χάρτη με το χέρι, είναι πολύ πιο εύκολο να σχεδιάσουμε τις αγκύλες από το να γράψουμε όλες τις ετικέτες. Ωστόσο, διατηρούμε τις ετικέτες στους χάρτες Karnaugh του βιβλίου ως μια επιπλέον βοήθεια κατά την ανάγνωση. Σε κάθε περίπτωση, πρέπει να βεβαιώνεστε ότι έχετε ονομάσει τις γραμμές και τις στήλες με τη σωστή σειρά για να διατηρήσετε την αντιστοιχία ανάμεσα στα κελιά του χάρτη και τους αριθμούς των γραμμών του πίνακα αληθείας που φαίνονται στην Εικόνα 4-26.

Για να αναπαραστήσουμε μια λογική συνάρτηση με ένα χάρτη Karnaugh, αντιγράφουμε απλώς τα 1 και τα 0 από τον πίνακα αληθείας ή την ισοδύναμη αναπαράσταση στα αντίστοιχα κελιά του χάρτη. Στις Εικόνες 4-27(α) και (β) εμφανίζεται ο πίνακας αληθείας και ο χάρτης Karnaugh της λογικής συνάρτησης που αναλύσαμε (σχεδόν εξοντώσαμε) στην Ενότητα 4.2. Από τώρα και στο εξής, θα περιορίσουμε την σύγχιση στους χάρτες αντιγράφοντας μόνο τα 1 ή τα 0, όχι όμως και τα δύο.

4.3.5 Ελαχιστοποίηση αθροισμάτων γινομένων

Τώρα θα σας έχει δημιουργηθεί η απορία σχετικά με την “παράξενη” διευθέτηση των αριθμών των γραμμών και των στηλών στους χάρτες Karnaugh. Υπάρχει ένας σημαντικός λόγος γι′ αυτή τη διευθέτηση: κάθε κελί αντιστοιχεί σε ένα συνδυασμό εισόδων ο οποίος διαφέρει από τους αμέσως γειτονικούς ως προς μία μόνο μεταβλητή. Για παράδειγμα, τα κελιά 5 και 13 του χάρτη 4 μεταβλητών διαφέρουν μόνο ως προς την τιμή του W. Στους χάρτες 3 και 4 μεταβλητών, τα αντίστοιχα κελιά στο αριστερό/δεξί ή επάνω/κάτω περιθώριο είναι λιγότερο προφανείς γείτονες. Για παράδειγμα, τα κελιά 12 και 14 ενός χάρτη 4 μεταβλητών είναι γειτονικά επειδή διαφέρουν μόνο ως προς την τιμή του Y.

Κάθε συνδυασμός εισόδων με “1” στον πίνακα αληθείας αντιστοιχεί σε έναν ελάχιστο όρο στο κανονικό άθροισμα της λογικής συνάρτησης. Εφόσον τα ζεύγη των γειτονικών κελιών “1” στο χάρτη Karnaugh έχουν



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

Εικόνα 4-27 $F = \Sigma_{X,Y,Z}$ (1,2,5,7): (α) πίνακας αληθείας, (β) χάρτης Karnaugh, (γ) συνδυασμός γειτονικών κελιών με τιμή 1.

ελάχιστους όρους που διαφέρουν ως προς μία μόνο μεταβλητή, τα ζεύγη ελαχίστων όρων είναι δυνατόν να συνδυαστούν μεταξύ τους σε έναν απλό όρο γινομένου με εφαρμογή της γενίκευσης του θεωρήματος T10: όρος · Y + όρος · Y′ = όρος. Έτσι, μπορούμε να χρησιμοποιήσουμε ένα χάρτη Karnaugh για να απλοποιήσουμε το κανονικό άθροισμα μιας λογικής συνάρτησης.

Για παράδειγμα, ας θεωρήσουμε τα κελιά 5 και 7 της Εικόνας 4-27(β) και τη συμβολή τους στο κανονικό άθροισμα της συνάρτησης αυτής:

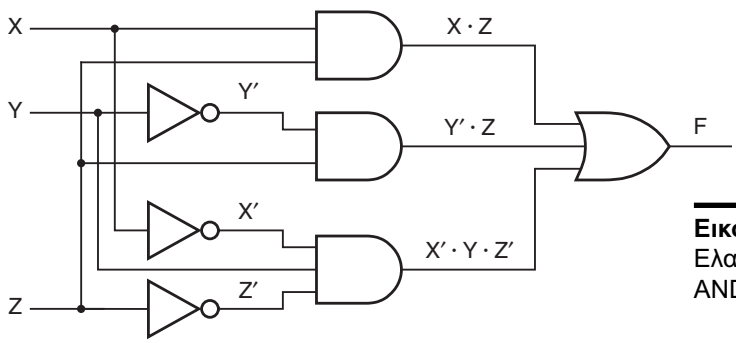
$$\begin{aligned} F &= \ldots + X\cdot Y'\cdot Z + X\cdot Y\cdot Z \\ &= \ldots + (X\cdot Z)\cdot Y' + (X\cdot Z)\cdot Y \\ &= \ldots + X\cdot Z \end{aligned}$$

Αν ξαναθυμηθούμε την αναδίπλωση, βλέπουμε ότι τα κελιά 1 και 5 της Εικόνας 4-27(β) είναι επίσης γειτονικά και είναι δυνατόν να συνδυαστούν μεταξύ τους:

$$\begin{aligned} F &= X'\cdot Y'\cdot Z + X\cdot Y'\cdot Z + \ldots \\ &= X'\cdot(Y'\cdot Z) + X\cdot(Y'\cdot Z) + \ldots \\ &= Y'\cdot Z + \ldots \end{aligned}$$

Γενικά, μπορούμε να απλοποιήσουμε μια λογική συνάρτηση, συνδυάζοντας μεταξύ τους ζεύγη γειτονικών κελιών με τιμή 1 (ελάχιστοι όροι) όπου αυτό είναι δυνατόν και γράφοντας ένα άθροισμα όρων γινομένου που να καλύπτει όλα τα κελιά με τιμή 1. Στην Εικόνα 4-27(γ) εμφανίζεται το αποτέλεσμα της λογικής συνάρτησης του παραδείγματός μας. Βάζουμε σε κύκλο ένα ζεύγος από 1 για να δείξουμε ότι οι αντίστοιχοι ελάχιστοι όροι συνδυάζονται μεταξύ τους σε έναν απλό όρο γινομένου. Το κύκλωμα AND-OR που προκύπτει φαίνεται στην Εικόνα 4-28.

Σε πολλές λογικές συναρτήσεις, η διαδικασία συνδυασμού κελιών μπορεί να επεκταθεί για το συνδυασμό περισσότερων από δύο κελιών με τιμή 1 σε έναν απλό όρο γινομένου. Για παράδειγμα, ας θεωρήσουμε το κανονικό άθροισμα για τη λογική συνάρτηση $F = \Sigma_{X,Y,Z}(0,1,4,5,6)$. Μπορούμε να χρησιμοποιήσουμε τους αλγεβρικούς χειρισμούς των προηγού-



Εικόνα 4-28
Ελαχιστοποιημένο κύκλωμα AND-OR.

μενων παραδειγμάτων όσες φορές χρειάζεται για να συνδυάσουμε μεταξύ τους τέσσερις από τους πέντε ελάχιστους όρους:

$$
\begin{aligned}
F &= X'\cdot Y'\cdot Z' + X'\cdot Y'\cdot Z + X\cdot Y'\cdot Z' + X\cdot Y'\cdot Z + X\cdot Y\cdot Z' \\
&= [(X'\cdot Y')\cdot Z' + (X'\cdot Y')\cdot Z] + [(X\cdot Y')\cdot Z' + (X\cdot Y')\cdot Z] + X\cdot Y\cdot Z' \\
&= X'\cdot Y' + X\cdot Y' + X\cdot Y\cdot Z' \\
&= [X'\cdot (Y') + X\cdot (Y')] + X\cdot Y\cdot Z' \\
&= Y' + X\cdot Y\cdot Z'
\end{aligned}
$$

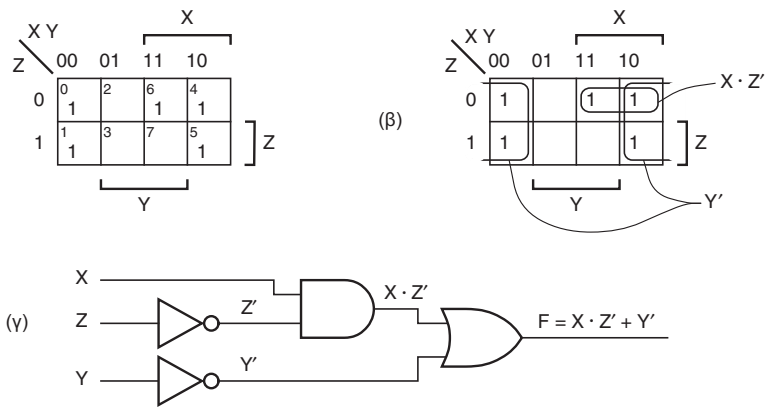
Γενικά, 2^i κελιά με τιμή 1 είναι δυνατόν να συνδυαστούν μεταξύ τους για να σχηματίσουν έναν όρο γινομένου που περιέχει *n-i* ονόματα, όπου *n* ο αριθμός των μεταβλητών της συνάρτησης.

Ένας ακριβής μαθηματικός κανόνας καθορίζει πώς μπορούν να συνδυαστούν μεταξύ τους τα κελιά με τιμή 1, καθώς και τη μορφή του αντίστοιχου όρου γινομένου:

- Ένα σύνολο 2^i κελιών με τιμή 1 είναι δυνατόν να συνδυαστούν μεταξύ τους αν υπάρχουν *i* μεταβλητές της λογικής συνάρτησης οι οποίες παίρνουν όλους τους 2^i δυνατούς συνδυασμούς μέσα στο σύνολο αυτό, ενώ οι *n-i* μεταβλητές που απομένουν έχουν την ίδια τιμή σε ολόκληρο το σύνολο. Ο αντίστοιχος όρος γινομένου έχει *n-i* ονόματα, όπου μια μεταβλητή συμπληρώνεται αν εμφανίζεται ως 0 σε όλα τα κελιά με τιμή 1 και αποσυμπληρώνεται αν εμφανίζεται ως 1.

ορθογώνια σύνολα των 1

Γραφικά, αυτός ο κανόνας σημαίνει ότι μπορούμε να βάλουμε σε κύκλο *ορθογώνια* σύνολα άσσων πλήθους 2^i, τόσο κυριολεκτικά όσο και μεταφορικά, επεκτείνοντας τον ορισμό του ορθογωνίου έτσι ώστε να λαμβάνεται υπόψη η αναδίπλωση στα άκρα του χάρτη. Μπορούμε να προσδιορίσουμε τα ονόματα των αντίστοιχων όρων γινομένου κατευθείαν από το χάρτη, κάνοντας για κάθε μεταβλητή τον παρακάτω προσδιορισμό:



Εικόνα 4-29
$F = \Sigma_{X,Y,Z}$ (0,1,4,5,6): (α) αρχικός χάρτης Karnaugh, (β) χάρτης Karnaugh με όρους γινομένου σε κύκλο, (γ) κύκλωμα AND/OR.

- Αν ο κύκλος καλύπτει μόνο περιοχές του χάρτη όπου η μεταβλητή είναι 0, τότε η μεταβλητή είναι συμπληρωματική στον όρο γινομένου.
- Αν ο κύκλος καλύπτει μόνο περιοχές του χάρτη όπου η μεταβλητή είναι 1, τότε η μεταβλητή είναι μη συμπληρωματική στον όρο γινομένου.
- Αν ο κύκλος καλύπτει περιοχές του χάρτη όπου η μεταβλητή είναι τόσο 0 όσο και 1, τότε η μεταβλητή δεν εμφανίζεται στον όρο γινομένου.

Η παράσταση αθροίσματος γινομένων μιας συνάρτησης είναι δυνατόν να περιέχει όρους γινομένων (σύνολα κελιών με τιμή 1 τοποθετημένα σε κύκλο) οι οποίοι καλύπτουν όλα τα 1 και κανένα από τα 0 του χάρτη.

Ο χάρτης Karnaugh για το πιο πρόσφατο παράδειγμα, $\Sigma_{X,Y,Z}$ (0,1,4,5,6), φαίνεται στην Εικόνα 4-29(α) και (β). Έχουμε βάλει σε κύκλο ένα σύνολο από τέσσερις άσσους που αντιστοιχεί στον όρο γινομένου Y′, καθώς και ένα σύνολο από δύο άσσους που αντιστοιχεί στον όρο γινομένου X·Z′. Παρατηρήστε ότι ο δεύτερος όρος γινομένου έχει ένα όνομα λιγότερο από τον αντίστοιχο όρο γινομένου της αλγεβρικής λύσης (X·Y·Z′). Βάζοντας σε κύκλο το μεγαλύτερο δυνατό σύνολο άσσων που περιλαμβάνει το κελί 6, έχουμε βρει μια λιγότερο δαπανηρή υλοποίηση της λογικής συνάρτησης, αφού μια πύλη AND δύο εισόδων κοστίζει λιγότερο από μία αντίστοιχη πύλη τριών εισόδων. Το γεγονός ότι δύο διαφορετικοί όροι γινομένων καλύπτουν τώρα το ίδιο κελί με τιμή 1 (4) δεν επηρεάζει τη λογική συνάρτηση, αφού για τη λογική πρόσθεση ισχύει ότι 1+1=1 και όχι 2! Το αντίστοιχο κύκλωμα AND/OR δύο επιπέδων φαίνεται στην περίπτωση (γ).

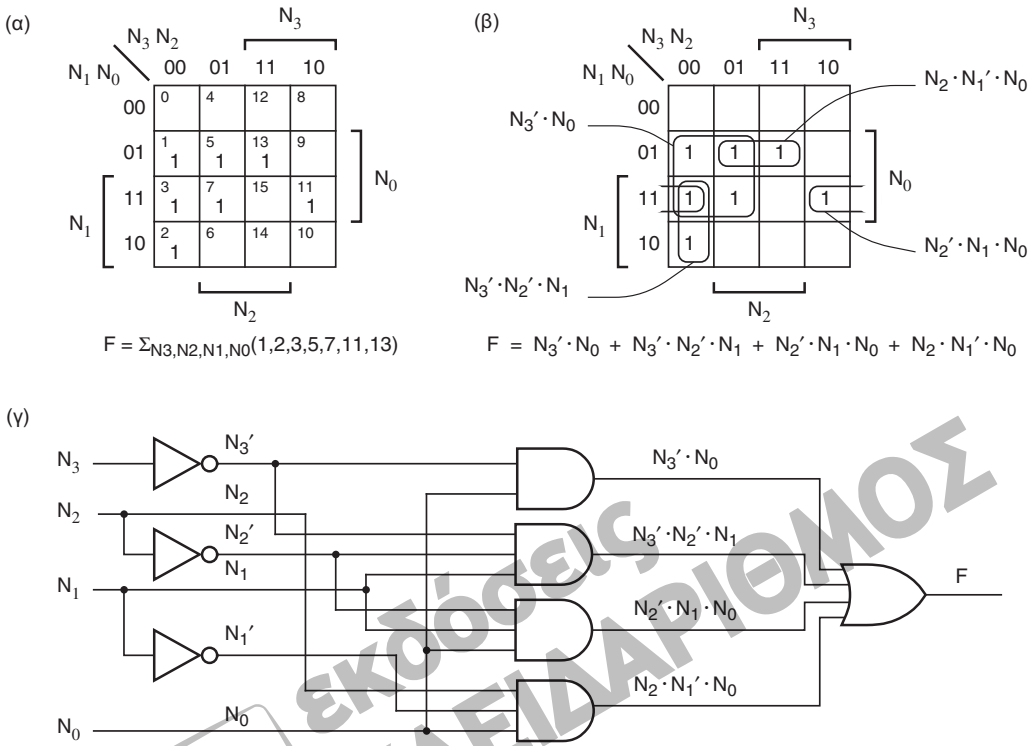
Ένα άλλο παράδειγμα είναι το κύκλωμα του ανιχνευτή πρώτων αριθμών που παρουσιάσαμε στην Εικόνα 4-18, στην Ενότητα 4.3.1, το οποίο είναι δυνατόν να ελαχιστοποιηθεί όπως φαίνεται στην Εικόνα 4-30.

Στο σημείο αυτό, χρειαζόμαστε κάποιους επιπλέον ορισμούς για να ξεκαθαρίσουμε τι κάνουμε:

ελάχιστο άθροισμα

- Το *ελάχιστο άθροισμα* μιας λογικής συνάρτησης $F(X_1,..., X_n)$ είναι μια παράσταση αθροίσματος γινομένων για την F τέτοια ώστε καμία παράσταση αθροίσματος γινομένων για την F να μην έχει λιγότερους όρους γινομένου, ενώ οποιαδήποτε παράσταση αθροίσματος γινομένων με τον ίδιο αριθμό όρων γινομένου να έχει τουλάχιστον τον ίδιο αριθμό ονομάτων.

Αυτό σημαίνει ότι το ελάχιστο άθροισμα έχει τους ελάχιστους δυνατούς όρους γινομένου (πύλες πρώτου επιπέδου και εισόδους πύλης δευτέρου επιπέδου) και, στα πλαίσια αυτού του περιορισμού, τα ελάχιστα δυνατά ονόματα (εισόδους πυλών πρώτου επιπέδου.) Έτσι λοιπόν, ανάμεσα στα τρία κυκλώματα εντοπισμού πρώτων αριθμών, μόνο το ένα της Εικόνας 4-30 που ακολουθεί υλοποιεί ένα ελάχιστο άθροισμα.



Εικόνα 4-30 Ανιχνευτής πρώτων αριθμών: (α) αρχικός χάρτης Karnaugh, (β) όροι γινομένου σε κύκλο, (γ) ελαχιστοποιημένο κύκλωμα.

συνεπάγεται

Ο επόμενος ορισμός δίνει το ακριβές περιεχόμενο της λέξης "συνεπάγεται" όταν μιλάμε για λογικές συναρτήσεις:

- Μια λογική συνάρτηση $P(X_1,...,X_n)$ *συνεπάγεται* μια λογική συνάρτηση $F(X_1,..., X_n)$ αν για κάθε συνδυασμό εισόδων τέτοιο ώστε P=1, ισχύει επίσης ότι F=1

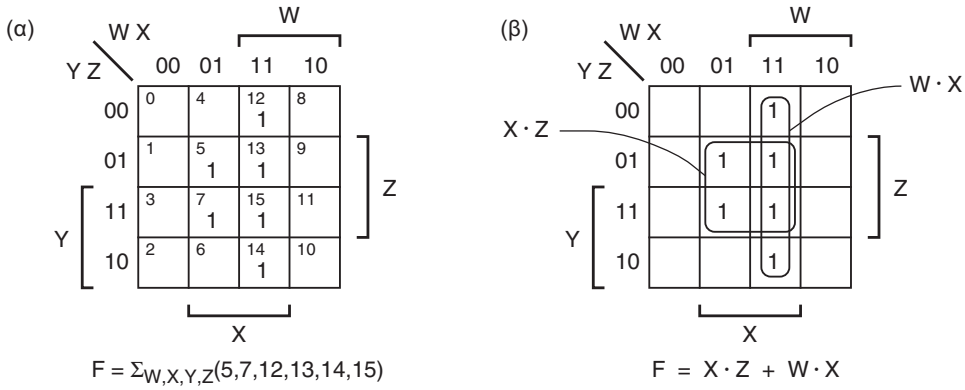
Αυτό σημαίνει ότι αν P συνεπάγεται F, τότε η F είναι 1 για κάθε συνδυασμό εισόδων για τον οποίο η P είναι 1 και πιθανώς και άλλους συνδυασμούς. Μπορούμε να γράψουμε τη συντομογραφία P ==> F. Μπορούμε επίσης να πούμε ότι "η F *συμπεριλαμβάνει* την P" ή ότι "η F *καλύπτει* την P".

συμπεριλαμβάνει
καλύπτει

πρωταρχικός όρος

- *Πρωταρχικός όρος* (*prime implicant*) μιας λογικής συνάρτησης $F(X_1,..., X_n)$ είναι ένας όρος κανονικού γινομένου $P(X_1,..., X_n)$ που συνεπάγεται την F τέτοιος ώστε, αν αφαιρεθεί μια μεταβλητή από την P, τότε ο όρος γινομένου που προκύπτει να μη συνεπάγεται την F.

Στους πίνακες Karnaugh, ένας πρωταρχικός όρος της F είναι ένα σύνολο κελιών με τιμή 1 μέσα σε κύκλο που ικανοποιούν το συνδυαστικό μας



Εικόνα 4-31 F = $\Sigma_{W,X,Y,Z}$ (5,7,12,13,14,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι.

κανόνα έτσι ώστε, αν προσπαθήσουμε να το διευρύνουμε (καλύπτοντας διπλάσιο αριθμό κελιών), να καλύπτει ένα ή περισσότερα 0.

Τώρα έρχεται το πιο σημαντικό μέρος, ένα θεώρημα το οποίο περιορίζει τη δουλειά που χρειάζεται να κάνουμε για να βρούμε το ελάχιστο άθροισμα μιας λογικής συνάρτησης:

Θεώρημα πρωταρχικού όρου Ελάχιστο άθροισμα είναι ένα άθροισμα πρωταρχικών όρων.

Αυτό σημαίνει ότι, για να βρούμε ένα ελάχιστο άθροισμα, δε χρειάζεται να λάβουμε υπόψη μας όρους γινομένου που δεν είναι πρωταρχικοί όροι. Το θεώρημα αυτό αποδεικνύεται εύκολα με τη μέθοδο της απαγωγής σε άτοπο. Ας υποθέσουμε ότι ένας όρος γινομένου P σε ένα "ελάχιστο" άθροισμα *δεν* είναι πρωταρχικός όρος. Τότε, σύμφωνα με τον ορισμό του πρωταρχικού όρου, αν το P δεν είναι πρωταρχικός όρος, μπορούμε να αφαιρέσουμε κάποια ονόματα από το P για να πάρουμε ένα νέο όρο γινομένου P* ο οποίος εξακολουθεί να συνεπάγεται την F. Αν αντικαταστήσουμε το P με το P* στο δεδομένο "ελάχιστο" άθροισμα, το άθροισμα που προκύπτει εξακολουθεί να είναι ίσο με F αλλά έχει ένα όνομα λιγότερο. Επομένως, το "ελάχιστο" άθροισμα που υποθέσαμε στην αρχή δεν ήταν τελικά ελάχιστο.

Ένα άλλο παράδειγμα ελαχιστοποίησης, αυτή τη φορά για μια συνάρτηση 4 μεταβλητών, φαίνεται στην Εικόνα 4-31. Υπάρχουν μόνο δύο πρωταρχικοί όροι και είναι ολοφάνερο ότι και οι δυο τους πρέπει να συμπεριλαμβάνονται στο ελάχιστο άθροισμα για να καλυφθούν όλα τα κελιά με τιμή 1 στο χάρτη. Δε σχεδιάσαμε το λογικό διάγραμμα γι' αυτό το παράδειγμα, επειδή τώρα πια μπορείτε να το κάνετε μόνοι σας.

Το άθροισμα όλων των πρωταρχικών όρων μιας λογικής συνάρτησης λέγεται *πλήρες άθροισμα*. Παρά το γεγονός ότι το πλήρες άθροισμα είναι πάντα ένας εύλογος τρόπος υλοποίησης μιας λογικής συνάρτησης, δεν

πλήρες άθροισμα

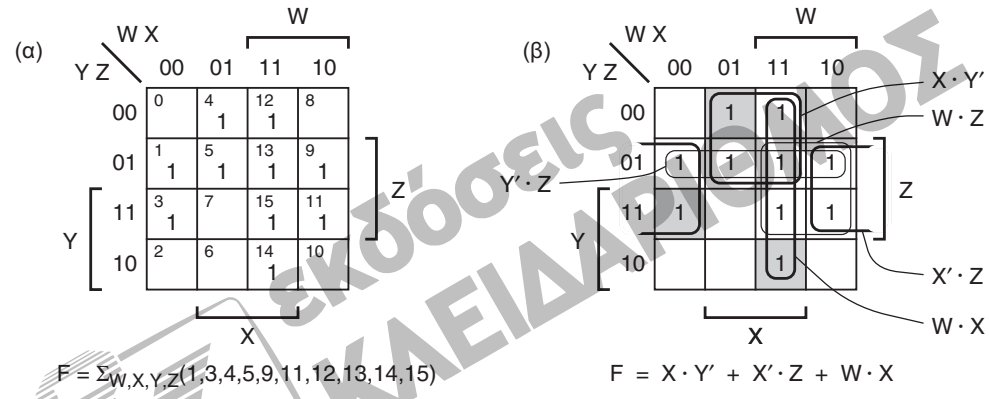
είναι πάντα το ελάχιστο. Για παράδειγμα, ας εξετάσουμε τη λογική συνάρτηση που παρουσιάζεται στην Εικόνα 4-32. Έχει πέντε πρωταρχικούς όρους, αλλά το ελάχιστο άθροισμα περιλαμβάνει μόνο τρεις από αυτούς. Έτσι, πώς μπορούμε να προσδιορίζουμε συστηματικά ποιους πρωταρχικούς όρους πρέπει να συμπεριλαμβάνουμε και ποιους να παραλείπουμε; Χρειαζόμαστε άλλους δύο ορισμούς

διακεκριμένο κελί με τιμή 1

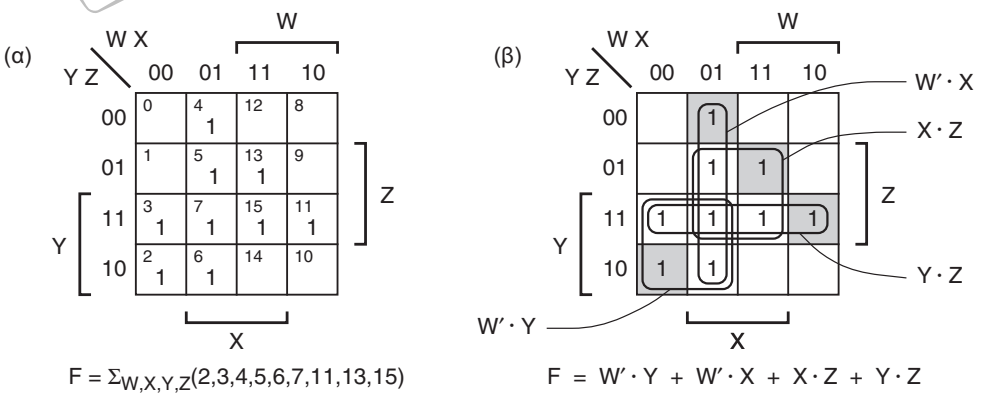
- Ένα *διακεκριμένο κελί με τιμή 1* μιας λογικής συνάρτησης είναι ένας συνδυασμός εισόδων που καλύπτεται από ένα μόνο πρωταρχικό όρο.

ουσιώδης πρωταρχικός όρος

- Ένας *ουσιώδης πρωταρχικός όρος* μιας λογικής συνάρτησης είναι ένας πρωταρχικός όρος που καλύπτει ένα ή περισσότερα διακεκριμένα κελιά με τιμή 1.



Εικόνα 4-32 $F = \Sigma_{W,X,Y,Z}$ (1,3,4,5,9,11,12,13,14,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι και διακεκριμένα κελιά με τιμή 1.



Εικόνα 4-33 $F = \Sigma_{W,X,Y,Z}$ (2,3,4,5,6,7,11,13,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι και διακεκριμένα κελιά με τιμή 1.

Αφού ο ουσιώδης πρωταρχικός όρος είναι ο *μοναδικός* πρωταρχικός όρος που καλύπτει μερικά κελιά με τιμή 1, θα *πρέπει* να συμπεριλαμβάνεται σε κάθε ελάχιστο άθροισμα της λογικής συνάρτησης. Έτσι, το πρώτο βήμα της μεθόδου επιλογής πρωταρχικών όρων είναι απλό: εντοπίζουμε διακεκριμένα κελιά με τιμή 1 και τους αντίστοιχους πρωταρχικούς όρους και κατόπιν συμπεριλαμβάνουμε τους ουσιώδεις πρωταρχικούς όρους στο ελάχιστο άθροισμα. Στη συνέχεια, θα χρειαστεί μόνο να προσδιορίσουμε τον τρόπο κάλυψης των κελιών με τιμή 1, αν υπάρχουν, τα οποία δεν καλύπτονται από τους ουσιώδεις πρωταρχικούς όρους. Στο παράδειγμα της Εικόνας 4-32, τα τρία διακεκριμένα κελιά με τιμή 1 εμφανίζονται σκιασμένα, ενώ οι αντίστοιχοι ουσιώδεις πρωταρχικοί όροι είναι τοποθετημένοι σε κύκλο με πιο έντονες γραμμές. Όλα τα κελιά με τιμή 1 του παραδείγματος αυτού καλύπτονται από ουσιώδεις πρωταρχικούς όρους, συνεπώς δε χρειάζεται να προχωρήσουμε άλλο. Με παρόμοιο τρόπο, στην Εικόνα 4-33 φαίνεται ένα παράδειγμα όπου όλοι οι πρωταρχικοί όροι είναι ουσιώδεις και συνεπώς συμπεριλαμβάνονται όλοι στο ελάχιστο άθροισμα.

Στην Εικόνα 4-34 παρατίθεται μια λογική συνάρτηση στην οποία δεν καλύπτονται όλα τα κελιά με τιμή 1 από ουσιώδεις πρωταρχικούς όρους. Αν αφαιρέσουμε τους ουσιώδεις πρωταρχικούς όρους και τα κελιά με τιμή 1 που καλύπτουν αυτοί, παίρνουμε ένα μικρότερου μεγέθους χάρτη που περιέχει μόνο ένα κελί με τιμή 1 και δύο πρωταρχικούς όρους που το καλύπτουν. Στην περίπτωση αυτή, η επιλογή είναι εύκολη: χρησιμοποιούμε τον όρο γινομένου W′·Z, επειδή διαθέτει λιγότερες εισόδους και κατά συνέπεια έχει μικρότερο κόστος.

Για τις πιο πολύπλοκες περιπτώσεις, χρειαζόμαστε ακόμη έναν ορισμό:

- Με δεδομένους δύο πρωταρχικούς όρους P και Q σε ένα μειωμένου μεγέθους χάρτη, ο P λέγεται ότι επισκιάζει τον Q (συμβολίζεται ως P...Q) αν ο P καλύπτει τουλάχιστον όλα τα κελιά με τιμή 1 που καλύπτει ο Q.

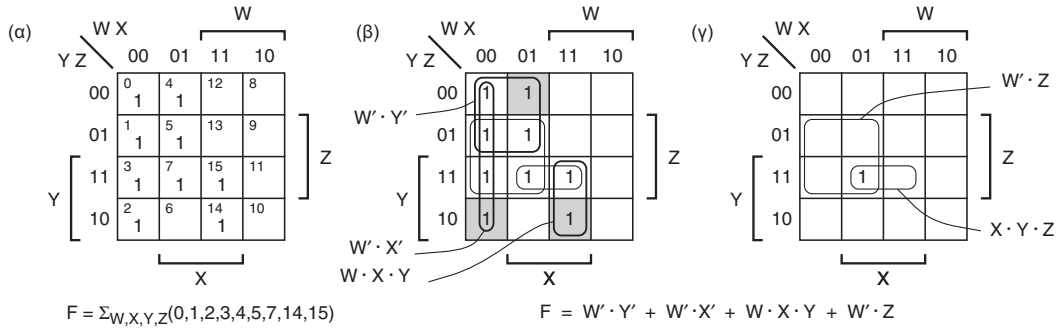
επισκίαση

Αν ο P δεν κοστίζει περισσότερο από τον Q και επισκιάζει τον Q, τότε η τυχόν εξαίρεση του Q από τη θεώρησή μας δε θα μας εμποδίσει να βρούμε ένα ελάχιστο άθροισμα. Με άλλα λόγια, ο P είναι τουλάχιστον το ίδιο καλός με τον Q.

Ένα παράδειγμα επισκίασης φαίνεται στην Εικόνα 4-35. Μετά από την αφαίρεση των ουσιωδών πρωταρχικών όρων, απομένουν δύο κελιά με τιμή 1 κάθε ένα από τα οποία καλύπτεται από δύο πρωταρχικούς όρους. Ωστόσο, ο X·Y·Z επισκιάζει τους άλλους δύο πρωταρχικούς όρους, οι οποίοι για το λόγο αυτόν είναι δυνατόν να εξαιρεθούν από τη θεώρησή μας. Τότε, τα δύο κελιά με τιμή 1 καλύπτονται μόνο από τον X·Y·Z, ο οποίος είναι ένας *δευτερεύων ουσιώδης πρωταρχικός όρος* που πρέπει να συμπεριληφθεί στο ελάχιστο άθροισμα.

δευτερεύων ουσιώδης πρωταρχικός όρος

Στην Εικόνα 4-36 παρουσιάζεται μια πιο δύσκολη περίπτωση — μια λογική συνάρτηση χωρίς ουσιώδεις πρωταρχικούς όρους. Με δοκιμή και

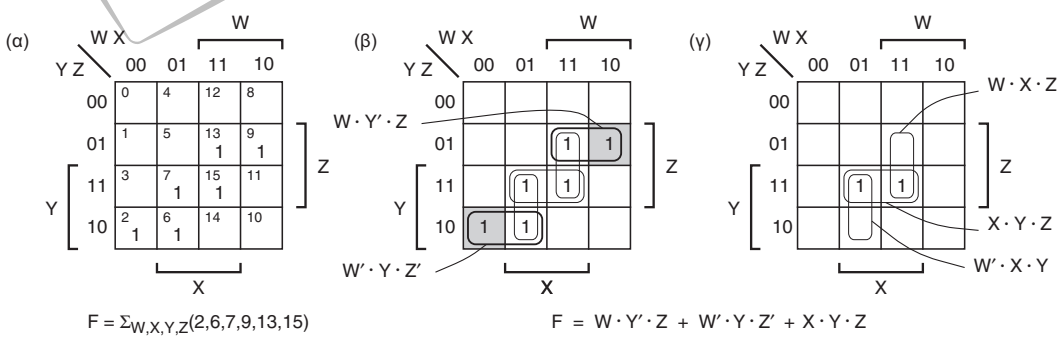


Εικόνα 4-34 F = $\Sigma_{W,X,Y,Z}$ (0,1,2,3,4,5,7,14,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι και διακεκριμένα κελιά με τιμή 1, (γ) μειωμένου μεγέθους χάρτης μετά από την αφαίρεση των ουσιωδών πρωταρχικών όρων και των καλυμμένων κελιών με τιμή 1.

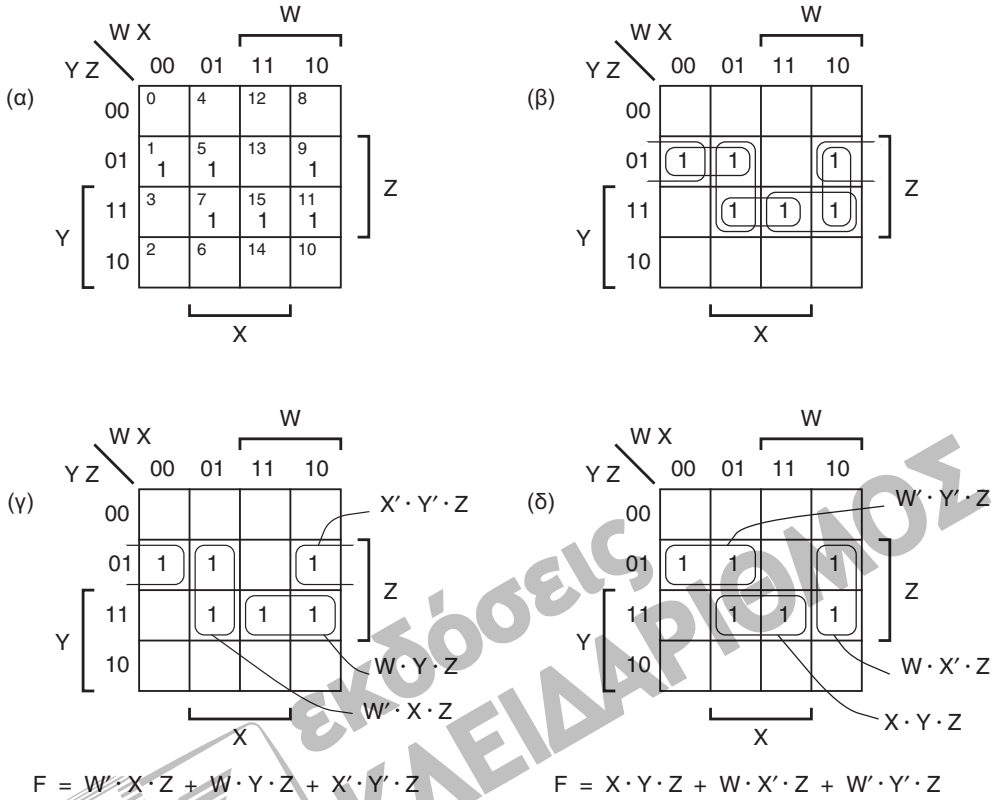
σφάλμα, μπορούμε να βρούμε δύο διαφορετικά ελάχιστα αθροίσματα για τη συνάρτηση αυτή.

μέθοδος διακλάδωσης

Μπορούμε επίσης να προσεγγίσουμε το πρόβλημα συστηματικά, με τη βοήθεια της *μεθόδου διακλάδωσης*. Αρχίζοντας από οποιοδήποτε κελί με τιμή 1, επιλέγουμε αυθαίρετα έναν από τους πρωταρχικούς όρους που το καλύπτουν και τον συμπεριλαμβάνουμε σαν να ήταν ουσιώδης. Αυτό απλοποιεί το απομένον πρόβλημα, το οποίο μπορούμε να ολοκληρώσουμε με το συνήθη τρόπο για να βρούμε ένα προσωρινό ελάχιστο άθροισμα. Επαναλαμβάνουμε τη μέθοδο αυτή, αρχίζοντας με όλους τους άλλους πρωταρχικούς όρους που καλύπτουν το αρχικό κελί με τιμή 1 και δημιουργώντας ένα διαφορετικό προσωρινό ελάχιστο άθροισμα από κάθε σημείο έναρξης. Στην πορεία ενδέχεται να κολλήσουμε και να χρεια-



Εικόνα 4-35 F = $\Sigma_{W,X,Y,Z}$ (2,6,7,9,13,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι και διακεκριμένα κελιά με τιμή 1, (γ) μειωμένου μεγέθους χάρτης μετά από την αφαίρεση των ουσιωδών πρωταρχικών όρων και των καλυμμένων κελιών με τιμή 1.



$F = W' \cdot X \cdot Z + W \cdot Y \cdot Z + X' \cdot Y' \cdot Z$ $\qquad$ $F = X \cdot Y \cdot Z + W \cdot X' \cdot Z + W' \cdot Y' \cdot Z$

Εικόνα 4-36 $F = \Sigma_{W,X,Y,Z}$ (1,5,7,9,11,15): (α) χάρτης Karnaugh, (β) πρωταρχικοί όροι, (γ) ένα ελάχιστο άθροισμα, (δ) άλλο ένα ελάχιστο άθροισμα.

στεί να εφαρμόσουμε τη μέθοδο διακλάδωσης αναδρομικά. Τέλος, εξετάζουμε όλα τα προσωρινά ελάχιστα αθροίσματα που δημιουργήσαμε με τον τρόπο αυτόν και επιλέγουμε εκείνο που είναι πραγματικά ελάχιστο.

4.3.6 Απλοποίηση γινομένων αθροισμάτων

Χρησιμοποιώντας την αρχή της δυικότητας, μπορούμε να ελαχιστοποιούμε παραστάσεις γινομένου αθροισμάτων παρατηρώντας τα 0 σε ένα χάρτη Karnaugh. Κάθε 0 στο χάρτη αντιστοιχεί σε ένα μέγιστο όρο στο κανονικό γινόμενο της λογικής συνάρτησης. Η όλη μέθοδος που περιγράφεται στην προηγούμενη ενότητα είναι δυνατόν να επαναδιατυπωθεί με δυικό τρόπο, συμπεριλαμβανομένων και των κανόνων καταγραφής όρων αθροίσματος που αντιστοιχούν στα σύνολα των 0 μέσα σε κύκλο, για να βρεθεί ένα *ελάχιστο γινόμενο*.

ελάχιστο γινόμενο

Ευτυχώς, εφόσον γνωρίζουμε πώς να βρίσκουμε ελάχιστα αθροίσματα, υπάρχει ένας ευκολότερος τρόπος για να βρούμε το ελάχιστο γινόμενο μιας δεδομένης λογικής συνάρτησης F. Το πρώτο βήμα είναι να υπολογίσουμε το συμπλήρωμα της F για να πάρουμε την F′. Αν υποθέσουμε ότι η F εκφράζεται ως μια λίστα ελαχίστων όρων ή ένας πίνακας αλη-

θείας, ο υπολογισμός του συμπληρώματος είναι πολύ εύκολος: τα 1 της F′ είναι απλώς τα 0 της F. Κατόπιν, βρίσκουμε ένα ελάχιστο άθροισμα για την F′ χρησιμοποιώντας τη μέθοδο της προηγούμενης ενότητας. Τέλος, υπολογίζουμε το συμπλήρωμα του αποτελέσματος χρησιμοποιώντας το γενικευμένο θεώρημα του DeMorgan, το οποίο δίνει ένα ελάχιστο γινόμενο για την (F′)′ = F. (Σημειώστε ότι αν απλώς "εκτελέσετε τις επιμέρους προσθέσεις" στην παράσταση ελαχίστου αθροίσματος για την αρχική συνάρτηση, η παράσταση γινομένου αθροισμάτων που προκύπτει δεν είναι απαραίτητα ελάχιστη. Για παράδειγμα, δείτε την Άσκηση 4.61.)

Γενικά, για να βρούμε την πλέον οικονομική υλοποίηση μιας λογικής συνάρτησης δύο επιπέδων, πρέπει να βρούμε τόσο ένα ελάχιστο άθροισμα όσο και ένα ελάχιστο γινόμενο και να τα συγκρίνουμε μεταξύ τους. Αν ένα ελάχιστο άθροισμα μιας λογικής συνάρτησης έχει πολλούς όρους, τότε ένα ελάχιστο γινόμενο της ίδιας συνάρτησης ενδέχεται να έχει λίγους όρους. Ως στοιχειώδες παράδειγμα γι' αυτή τη ανταλλαγή, θεωρούμε μια συνάρτηση OR 4 εισόδων:

$$F = (W) + (X) + (Y) + (Z) \quad \text{(άθροισμα τεσσάρων τετριμμένων όρων γινομένου)}$$

$$= (W + X + Y + Z) \quad \text{(γινόμενο με ένα μόνο όρο αθροίσματος)}$$

Ένα μη τετριμμένο παράδειγμα είναι να βρείτε το ελάχιστο γινόμενο της συνάρτησης που ελαχιστοποιήσαμε στην Εικόνα 4-33, στην Ενότητα 4.3.5, που διαθέτει δύο μόνο όρους αθροίσματος.

Το αντίστροφο επίσης αληθεύει μερικές φορές, όπως γίνεται φανερό από μια συνάρτηση AND 4 εισόδων:

$$F = (W) \cdot (X) \cdot (Y) \cdot (Z) \quad \text{(γινόμενο τεσσάρων τετριμμένων όρων αθροίσματος)}$$

$$= (W \cdot X \cdot Y \cdot Z) \quad \text{(άθροισμα με ένα μόνο όρο γινομένου)}$$

Ένα μη τετριμμένο παράδειγμα με γινόμενο αθροισμάτων υψηλότερου κόστους είναι η συνάρτηση της Εικόνας 4-29, στην Ενότητα 4.3.5.

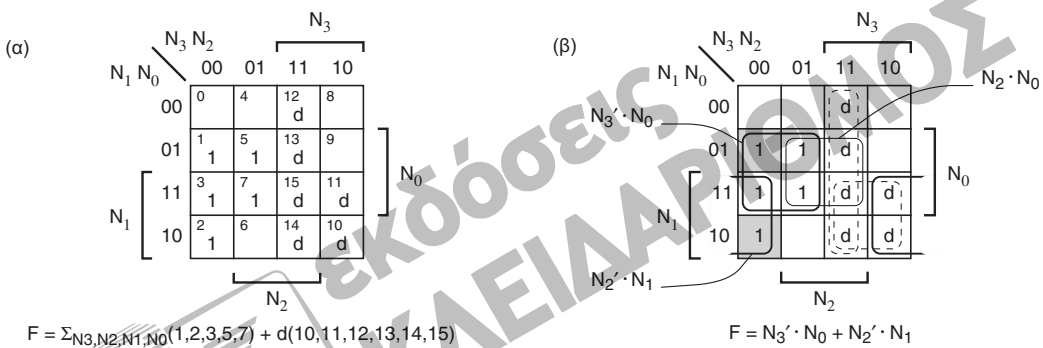
Για μερικές λογικές συναρτήσεις, και οι δύο μορφές ελαχιστοποίησης έχουν ισοδύναμο κόστος. Για παράδειγμα, θεωρούμε μια συνάρτηση "αποκλειστικού OR" 3 εισόδων. Και οι δύο ελάχιστες παραστάσεις έχουν από τέσσερις όρους η κάθε μία, ενώ κάθε όρος έχει τρία κυριολεκτικά:

$$\begin{aligned} F &= \Sigma_{X,Y,Z}\,(1,2,4,7) \\ &= (X'\cdot Y'\cdot Z) + (X'\cdot Y\cdot Z') + (X\cdot Y'\cdot Z') + (X\cdot Y\cdot Z) \\ &= (X + Y + Z)\cdot(X + Y' + Z')\cdot(X'\cdot Y\cdot Z')\cdot(X'\cdot Y'\cdot Z) \end{aligned}$$

Παρόλα αυτά, στις περισσότερες περιπτώσεις κάποια από τις δύο μορφές μάς δίνει καλύτερα αποτελέσματα. Η εξέταση και των δύο μορφών είναι ιδιαίτερα χρήσιμη στις σχεδιάσεις με προγραμματιζόμενες λογικές διατάξεις (PLD).

ΕΛΑΧΙΣΤΟΠΟΙΗΣΗ PLD

Κατά κανόνα, οι διατάξεις PLD έχουν έναν πίνακα AND-OR που αντιστοιχεί σε μια μορφή αθροίσματος γινομένων. Αυτό μπορεί να σας κάνει να πιστεύετε ότι μόνο το ελάχιστο άθροισμα γινομένων έχει σχέση με τη σχεδίαση που βασίζεται σε PLD. Ωστόσο, οι περισσότερες διατάξεις PLD έχουν έναν προγραμματιζόμενο αντιστροφέα/απομονωτή στην έξοδο του πίνακα AND-OR που μπορεί είτε να αναστρέφει είτε όχι. Έτσι, η διάταξη PLD μπορεί να χρησιμοποιεί το ισοδύναμο του ελαχίστου αθροίσματος, με τη χρήση του πίνακα AND-OR για την υλοποίηση του συμπληρώματος της επιθυμητής συνάρτησης, και έπειτα με τον προγραμματισμό του αντιστροφέα/απομονωτή για να εκτελεί αντιστροφή. Τα περισσότερα προγράμματα λογικής ελαχιστοποίησης για PLD βρίσκουν αυτόματα τόσο το ελάχιστο άθροισμα όσο και το ελάχιστο γινόμενο και επιλέγουν εκείνο που απαιτεί λιγότερους όρους.



Εικόνα 4-37 Ανιχνευτής πρώτων αριθμών με ψηφία BCD: (α) αρχικός χάρτης Karnaugh, (β) χάρτης Karnaugh με πρωταρχικούς όρους και διακεκριμένα κελιά με τιμή 1.

*4.3.7 Αδιάφοροι συνδυασμοί εισόδων

Μερικές φορές, οι προδιαγραφές ενός συνδυαστικού κυκλώματος είναι τέτοιες ώστε η έξοδος δεν ενδιαφέρει για ορισμένους συνδυασμούς εισόδων, οι οποίοι λέγονται *αδιάφοροι* (*don't-care*). Αυτό μπορεί να συμβαίνει επειδή οι έξοδοι δεν έχουν σημασία όταν εμφανίζονται οι συγκεκριμένοι συνδυασμοί εισόδων, ή επειδή αυτοί οι συνδυασμοί εισόδων δεν εμφανίζονται ποτέ στην κανονική λειτουργία. Για παράδειγμα, υποθέτουμε ότι θέλουμε να κατασκευάσουμε έναν ανιχνευτή πρώτων αριθμών του οποίου η είσοδος 4 ψηφίων $N = N_3 N_2 N_1 N_0$ είναι πάντα ένα ψηφίο BCD. Οι ελάχιστοι όροι 10-15 δε θα εμφανιστούν ποτέ. Επομένως, η συνάρτηση του ανιχνευτή πρώτων αριθμών με ψηφία BCD μπορεί να γραφεί ως εξής:

αδιάφορο

$$F = \Sigma_{N_3,N_2,N_1,N_0}(1,2,3,5,7) + d(10,11,12,13,14,15)$$

* Σε ολόκληρο το βιβλίο, οι προαιρετικές ενότητες είναι σημειωμένες με αστερίσκο.

σύνολο d

Η λίστα d(...) καθορίζει τους αδιάφορους συνδυασμούς εισόδων για τη συνάρτηση, που είναι επίσης γνωστή και ως το *σύνολο d*. Εδώ, η F πρέπει να είναι 1 για συνδυασμούς εισόδων από το σύνολο (1,2,3,5,7), μπορεί να πάρει οποιεσδήποτε τιμές για εισόδους από το σύνολο d (10,11,12,13,14,15), και πρέπει να είναι 0 για όλους τους άλλους συνδυασμούς εισόδων (στο σύνολο 0).

Η Εικόνα 4-37 δείχνει πώς να βρείτε μια υλοποίηση με ελάχιστο άθροισμα γινομένων για τον ανιχνευτή πρώτων αριθμών με ψηφία BCD, συμπεριλαμβανομένων και των αδιάφορων. Τα d στο χάρτη υποδηλώνουν τους αδιάφορους συνδυασμούς εισόδων. Τροποποιούμε τη διαδικασία τοποθέτησης των άσσων (πρωταρχικοί όροι) σε κύκλο ως εξής:

- Επιτρέπουμε στα d να συμπεριλαμβάνονται όταν τοποθετούμε σύνολα άσσων σε κύκλο, για να κάνουμε τα σύνολα όσο πιο μεγάλα γίνεται. Με τον τρόπο αυτό περιορίζεται ο αριθμός των μεταβλητών στους πρωταρχικούς όρους. Στο παράδειγμα εμφανίζονται δύο τέτοιοι πρωταρχικοί όροι ($N_2 \cdot N_0$ και $N_2' \cdot N_1$).
- Δεν τοποθετούμε σε κύκλο σύνολα τα οποία περιέχουν μόνο d. Αν συμπεριλάβετε τον αντίστοιχο όρο γινομένου στη συνάρτηση, ενδέχεται να αυξήσετε άσκοπα το κόστος. Δύο τέτοιοι όροι γινομένου ($N_3 \cdot N_2$ και $N_3 \cdot N_1$) έχουν τοποθετηθεί σε κύκλο στο παράδειγμα.
- Υπενθύμιση: Όπως συνήθως, δεν τοποθετείτε σε κύκλο τα 0.

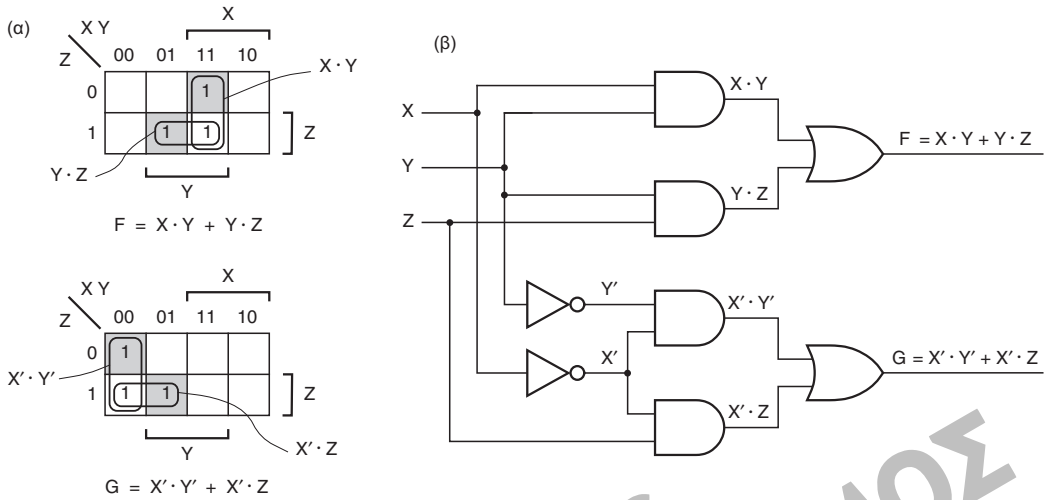
Η υπόλοιπη διαδικασία είναι η ίδια. Ειδικότερα, ψάχνουμε για διακεκριμένα κελιά με τιμή 1 και *μη* διακεκριμένα κελιά με τιμή *d* και συμπεριλαμβάνουμε μόνο τους αντίστοιχους ουσιώδεις πρωταρχικούς όρους και όποιους άλλους χρειάζονται για να καλύψουμε όλα τα 1 στο χάρτη. Στην Εικόνα 4-37, οι δύο ουσιώδεις πρωταρχικοί όροι αρκούν για να καλύψουν όλα τα 1 στο χάρτη. Δύο από τα *d* συμβαίνει επίσης να είναι καλυμμένα, έτσι η F θα είναι 1 για τους αδιάφορους συνδυασμούς εισόδων 10 και 11 και 0 για τις υπόλοιπες αδιάφορες εισόδους.

Κάποιες γλώσσες περιγραφής υλικού (HDL), συμπεριλαμβανομένης της ABEL, παρέχουν στους σχεδιαστές έναν τρόπο καθορισμού των αδιάφορων εισόδων, ενώ το πρόγραμμα λογικής ελαχιστοποίησης τις λαμβάνει υπόψη όταν υπολογίζει ένα ελάχιστο άθροισμα.

*4.3.8 Ελαχιστοποίηση πολλών εξόδων

Τα περισσότερα πραγματικά συνδυαστικά λογικά κυκλώματα απαιτούν περισσότερες από μία εξόδους. Μπορούμε πάντα να χειριστούμε ένα κύκλωμα με *n* εξόδους ως *n* το πλήθος ανεξάρτητα προβλήματα σχεδίασης μίας εξόδου. Ωστόσο, αν κάνουμε κάτι τέτοιο, ενδέχεται να χάσουμε κάποιες ευκαιρίες για βελτιστοποίηση. Για παράδειγμα, ας θεωρήσουμε τις δύο επόμενες λογικές συναρτήσεις:

$$F = \Sigma_{X,Y,Z}\,(3,6,7)$$
$$G = \Sigma_{X,Y,Z}\,(0,1,3)$$



Εικόνα 4-38 Χειρισμός της σχεδίασης 2 εξόδων ως δύο ανεξαρτήτων σχεδιάσεων μίας εξόδου: (α) χάρτες Karnaugh, (β) "ελάχιστο" κύκλωμα.

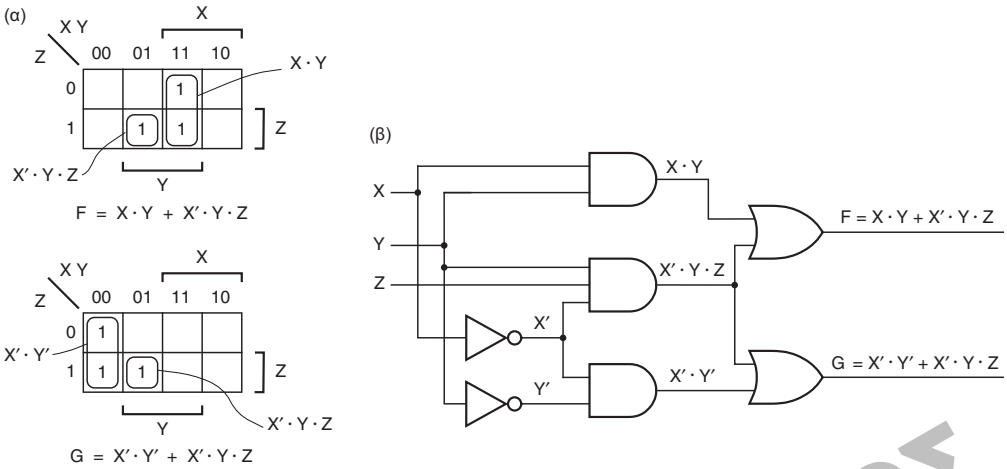
Στην Εικόνα 4-38 παρουσιάζεται η σχεδίαση των F και G ως δύο ανεξάρτητων συναρτήσεων μίας εξόδου. Ωστόσο, όπως φαίνεται στην Εικόνα 4-39, μπορούμε επίσης να βρούμε ένα ζεύγος παραστάσεων αθροίσματος γινομένων που να χρησιμοποιεί από κοινού έναν όρο γινομένου, τέτοιο ώστε το κύκλωμα που προκύπτει να έχει μια πύλη λιγότερη απ΄ ό,τι η αρχική σχεδίαση.

Όταν σχεδιάζουμε συνδυαστικά κυκλώματα πολλών εξόδων με διακριτές πύλες, όπως σε ένα κύκλωμα ASIC, η από κοινού χρήση όρων γινομένου προφανώς περιορίζει το μέγεθος και το κόστος του κυκλώματος. Επιπλέον, οι διατάξεις PLD περιέχουν πολλά αντίγραφα της δομής αθροίσματος γινομένων την οποία έχουμε μάθει πώς να ελαχιστοποιούμε, μία ανά έξοδο, ενώ μερικές PLD επιτρέπουν την από κοινού χρήση όρων γινομένου σε πολλές εξόδους. Έτσι, οι ιδέες που παρουσιάζονται στην ενότητα αυτή εφαρμόζονται σε πολλά προγράμματα λογικής ελαχιστοποίησης.

Πιθανόν να έχετε εντοπίσει τους χάρτες Karnaugh για τις F και G στην Εικόνα 4-39 και να έχετε ανακαλύψει την ελάχιστη λύση. Ωστόσο, τα μεγαλύτερα κυκλώματα είναι δυνατόν να ελαχιστοποιηθούν μόνο με έναν τυπικό αλγόριθμο ελαχιστοποίησης πολλών εξόδων. Εδώ θα παρουσιάσουμε περιληπτικά τις βασικές αρχές ενός τέτοιου αλγόριθμου, ενώ λεπτομέρειες παρατίθενται στις Παραπομπές.

Το κλειδί της επιτυχούς ελαχιστοποίησης πολλών εξόδων ενός συνόλου από n συναρτήσεις είναι να λάβουμε υπόψη όχι μόνο τις n πρωτότυπες συναρτήσεις μίας εξόδου, αλλά και τις "συναρτήσεις γινομένων". Μια *συνάρτηση m γινομένων* ενός συνόλου από n συναρτήσεις είναι το

συνάρτηση m γινομένων



Εικόνα 4-39 Ελαχιστοποίηση πολλών εξόδων για ένα κύκλωμα 2 εξόδων: (α) ελαχιστοποιημένοι χάρτες που περιλαμβάνουν ένα από κοινού χρησιμοποιούμενο όρο (β) ελάχιστο κύκλωμα πολλών εξόδων.

γινόμενο των m συναρτήσεων, όπου $2 \leq m \leq n$. Υπάρχουν 2^n - n -1 τέτοιες συναρτήσεις. Ευτυχώς, στο παράδειγμά μας ισχύει $n = 2$ και έτσι υπάρχει μόνο μία συνάρτηση γινομένου, η F·G, προς μελέτη. Οι χάρτες Karnaugh για τις F, G, και F·G παρουσιάζονται στην Εικόνα 4-40. Γενικά, ο χάρτης για μια συνάρτηση m γινομένων λαμβάνεται με την εκτέλεση της λογικής πράξης AND μεταξύ των χαρτών των m στοιχείων.

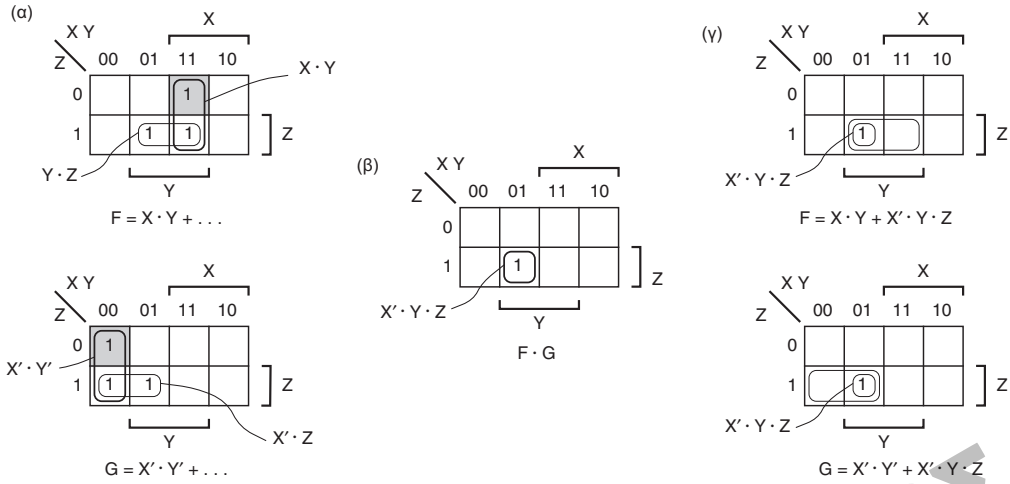
πρωταρχικός όρος πολλών εξόδων

Πρωταρχικός όρος πολλών εξόδων ενός συνόλου n συναρτήσεων είναι ένας πρωταρχικός όρος για μία από τις n συναρτήσεις ή για μία από τις συναρτήσεις γινομένου. Το πρώτο βήμα στην ελαχιστοποίηση πολλών εξόδων είναι να βρούμε όλους τους πρωταρχικούς όρους πολλών εξόδων. Κάθε πρωταρχικός όρος μιας συνάρτησης m γινομένων είναι ένας όρος που είναι δυνατόν να εμπεριέχεται στις αντίστοιχες m εξόδους του κυκλώματος. Αν επιχειρούσαμε να ελαχιστοποιήσουμε ένα σύνολο από 8 συναρτήσεις, θα έπρεπε να βρούμε τους πρωταρχικούς όρους για 2^8 - 8 - 1 = 247 συναρτήσεις γινομένων, καθώς επίσης και για τις 8 δεδομένες συναρτήσεις. Προφανώς, η ελαχιστοποίηση πολλών εξόδων είναι για τους θαρραλέους!

διακεκριμένο κελί με τιμή 1

ουσιώδης πρωταρχικός όρος

Από τη στιγμή που θα βρούμε τους πρωταρχικούς όρους πολλών εξόδων, επιχειρούμε να απλοποιήσουμε το πρόβλημα προσδιορίζοντας τους ουσιώδεις εξ αυτών. Ένα *διακεκριμένο κελί με τιμή 1* μιας συγκεκριμένης συνάρτησης F μίας εξόδου είναι ένα κελί με τιμή 1 το οποίο καλύπτεται από ακριβώς έναν πρωταρχικό όρο της F ή των συναρτήσεων γινομένου όπου συμμετέχει η F. Τα διακεκριμένα κελιά με τιμή 1 στην Εικόνα 4-40 είναι σκιασμένα. Ένας ουσιώδης *πρωταρχικός όρος* μιας συγκεκριμένης συνάρτησης μίας εξόδου είναι αυτός που περιέχει ένα διακεκριμένο κελί με τιμή 1. Όπως και στην ελαχιστοποίηση μίας εξόδου, οι ουσιώδεις



Εικόνα 4-40 Χάρτες Karnaugh για ένα σύνολο από δύο συναρτήσεις: (α) χάρτες για τις F και G, (β) χάρτης 2 γινομένων για την FΉG, (γ) χάρτες μειωμένου μεγέθους για τις F και G μετά από την αφαίρεση βασικών πρωταρχικών όρων και καλυπτόμενων κελιών με τιμή 1.

πρωταρχικοί όροι πρέπει να περιλαμβάνονται στη λύση ελαχίστου κόστους. Μόνο τα κελιά με τιμή 1 τα οποία δεν καλύπτονται από τους ουσιώδεις πρωταρχικούς όρους λαμβάνονται υπόψη στο υπόλοιπο μέρος του αλγόριθμου.

Το τελικό βήμα είναι να επιλέξουμε ένα ελάχιστο σύνολο πρωταρχικών όρων για να καλύψουμε τα υπόλοιπα κελιά με τιμή 1. Στο βήμα αυτόν, πρέπει να εξετάσουμε και τις *n* συναρτήσεις ταυτόχρονα, συμπεριλαμβανομένης και της δυνατότητας από κοινού χρήσης. Λεπτομέρειες για τη διαδικασία αυτή παρατίθενται στις Παραπομπές. Στο παράδειγμα της Εικόνας 4-40(γ), βλέπουμε ότι υπάρχει ένας και μοναδικός όρος γινομένου κοινής χρήσης που καλύπτει τα εναπομένοντα κελιά με τιμή 1 τόσο στην F όσο και στην G.

*4.4 Προγραμματιζόμενες μέθοδοι ελαχιστοποίησης

Προφανώς, η λογική ελαχιστοποίηση μπορεί να είναι πολύ περίπλοκη διαδικασία. Στις πραγματικές εφαρμογές λογικής σχεδίασης, είναι δυνατόν να συναντήσετε δύο είδη προβλημάτων ελαχιστοποίησης: συναρτήσεις με λίγες μεταβλητές τις οποίες μπορείτε να δουλέψετε “με το μάτι” χρησιμοποιώντας τις μεθόδους της προηγούμενης ενότητας, καθώς και πιο πολύπλοκες συναρτήσεις πολλών εξόδων τις οποίες είναι αδύνατο να επεξεργαστείτε χωρίς τη χρήση ενός προγράμματος ελαχιστοποίησης.

Γνωρίζουμε ότι η ελαχιστοποίηση μπορεί να εκτελεστεί οπτικά για συναρτήσεις με λίγες μεταβλητές, με τη βοήθεια της μεθόδου των χαρτών Karnaugh. Σε αυτή την ενότητα θα δείξουμε ότι οι ίδιες πράξεις εί-

αλγόριθμος Quine-McCluskey

ναι δυνατόν να εκτελεστούν σε συναρτήσεις με οσοδήποτε μεγάλο αριθμό μεταβλητών (τουλάχιστον καταρχήν) με τη βοήθεια μιας μεθόδου πινακοποίησης που λέγεται *αλγόριθμος Quine-McCluskey*. Όπως όλοι οι αλγόριθμοι, ο αλγόριθμος Quine-McCluskey μπορεί να μεταφραστεί σε πρόγραμμα υπολογιστή. Επίσης, όπως συμβαίνει και στη μέθοδο του χάρτη, ο αλγόριθμος έχει δύο βήματα: (α) εύρεση όλων των πρωταρχικών όρων της συνάρτησης και (β) επιλογή ενός ελαχίστου συνόλου πρωταρχικών όρων που καλύπτει τη συνάρτηση.

Ο αλγόριθμος Quine-McCluskey περιγράφεται συχνά με τη μορφή χειρόγραφων πινάκων και με χειρωνακτικές διαδικασίες ελέγχου. Ωστόσο, αφού κανείς δεν εφαρμόζει πια αυτές τις διαδικασίες με το χέρι, είναι πιο σωστό να συζητήσουμε τον αλγόριθμο με τη μορφή δομών δεδομένων και συναρτήσεων σε μια γλώσσα προγραμματισμού υψηλού επιπέδου. Ο στόχος αυτής της ενότητας είναι να σας δώσει μια εκτίμηση της υπολογιστικής πολυπλοκότητας που χαρακτηρίζει ένα μεγάλο πρόβλημα ελαχιστοποίησης. Θεωρούμε μόνο πλήρως καθορισμένες συναρτήσεις μίας εξόδου. Ο χειρισμός των αδιάφορων εισόδων και των συναρτήσεων πολλών εξόδων είναι δυνατόν να γίνει με αρκετά απλές τροποποιήσεις των αλγορίθμων μίας εξόδου, όπως αναφέρεται στις Παραπομπές.

*4.4.1 Αναπαράσταση όρων γινομένου

Το σημείο έναρξης του αλγόριθμου ελαχιστοποίησης Quine-McCluskey είναι ο πίνακας αληθείας ή, ισοδύναμα, η λίστα ελαχίστων όρων μιας συνάρτησης. Αν η συνάρτηση καθορίζεται διαφορετικά, θα πρέπει πρώτα να μετατραπεί σε αυτή τη μορφή. Για παράδειγμα, σε μια οποιαδήποτε λογική παράσταση n μεταβλητών, μπορούμε να εκτελέσουμε τους επιμέρους πολλαπλασιασμούς (πιθανώς χρησιμοποιώντας το θεώρημα του DeMorgan) για να πάρουμε μια παράσταση αθροίσματος γινομένων. Από τη στιγμή που έχουμε μια παράσταση αθροίσματος γινομένων, κάθε όρος γινομένου p μεταβλητών παράγει 2^{n-p} ελάχιστους όρους στη λίστα ελαχίστων όρων.

Στην Ενότητα 4.1.6 δείξαμε ότι ένας ελάχιστος όρος μιας λογικής συνάρτησης n μεταβλητών είναι δυνατόν να αναπαρασταθεί με έναν ακέραιο των n bit (δηλαδή τον αριθμό ελαχίστων όρων), όπου κάθε bit δείχνει αν η αντίστοιχη μεταβλητή είναι συμπληρωματική ή μη συμπληρωματική. Ωστόσο, ένας αλγόριθμος ελαχιστοποίησης πρέπει επίσης να χειρίζεται όρους γινομένου οι οποίοι δεν είναι ελάχιστοι όροι, στους οποίους μερικές μεταβλητές δεν εμφανίζονται καθόλου. Έτσι, πρέπει να αναπαραστήσουμε τρεις δυνατότητες για κάθε μεταβλητή σε ένα γενικό όρο γινομένου:

1 Μη συμπληρωματική.
0 Συμπληρωματική.
x Δεν εμφανίζεται.

κυβική αναπαράσταση

Αυτές οι περιπτώσεις αναπαρίστανται με μια ακολουθία χαρακτήρων με n από τα παραπάνω ψηφία στην *κυβική αναπαράσταση* ενός όρου γινο-

μένου. Για παράδειγμα, αν εργαζόμαστε με όρους γινομένου με έως οκτώ μεταβλητές, X7, X6,..., X1, X0, μπορούμε να γράψουμε τους παρακάτω όρους γινομένου και τις κυβικές τους αναπαραστάσεις:

$$X7' \cdot X6 \cdot X5 \cdot X4' \cdot X3 \cdot X2 \cdot X1 \cdot X0' \equiv 01101110$$
$$X3 \cdot X2 \cdot X1 \cdot X0' \equiv xxxx1110$$
$$X7 \cdot X5' \cdot X4 \cdot X3 \cdot X2' \cdot X1 \equiv 1x01101x$$
$$X6 \equiv x1xxxxxx$$

Παρατηρήστε ότι ονομάσαμε για ευκολία τις μεταβλητές ακριβώς όπως τις θέσεις των bit στους δυαδικούς ακεραίους των *n* bit.

Σύμφωνα με την ονοματολογία *n-διάστατων* κύβων και *m-διάστατων* υποκύβων της Ενότητας 2.14, το αλφαριθμητικό 1x01101x αναπαριστά ένα 2-διάστατο υποκύβο ενός 8-διάστατου κύβου, ενώ το αλφαριθμητικό 01101110 αναπαριστά ένα 0-διάστατο υποκύβο ενός 8-διάστατου κύβου. Ωστόσο, στη βιβλιογραφία περί ελαχιστοποίησης, η μέγιστη διάσταση *n* ενός κύβου ή υποκύβου είναι συνήθως υπονοούμενη και έτσι ένας *m*-διάστατος υποκύβος λέγεται απλώς "*m*-διάστατος κύβος" ή για συντομία "κύβος". Σ΄ αυτή την ενότητα, θα ακολουθήσουμε αυτή την πρακτική.

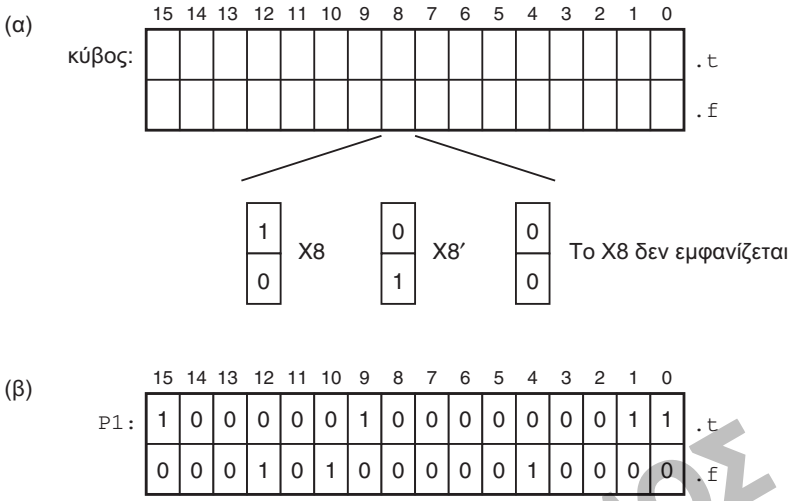
Για να αναπαραστήσουμε έναν όρο γινομένου σε ένα πρόγραμμα υπολογιστή, μπορούμε να χρησιμοποιήσουμε μια δομή δεδομένων με *n* στοιχεία, κάθε ένα από τα οποία έχει τρεις δυνατές τιμές. Στη C, μπορούμε να κάνουμε τις παρακάτω δηλώσεις:

```
typedef enum {complemented, uncomplemented, doesntappear} TRIT;
typedef TRIT[16] CUBE;/ * Αντιπροσωπεύει έναν και μόνο όρο
                          γινομένου με 16 το πολύ μεταβλητές */
```

Ωστόσο, αυτές οι δηλώσεις δεν οδηγούν σε ιδιαίτερα αποδοτική εσωτερική αναπαράσταση των κύβων. Όπως θα δούμε παρακάτω, ο χειρισμός των κύβων είναι πιο εύκολος με τις συνηθισμένες εντολές του υπολογιστή αν ένας όρος γινομένου *n* μεταβλητών αναπαρίσταται με δύο λέξεις υπολογιστή των *n* bit, όπως στις παρακάτω δηλώσεις:

```
#define MAX-VARS 16   /* Μέγιστος αριθ. μεταβλητών όρου γινομένου
*/ typedef unsigned short WORD;   /* Χρήση λέξεων των 16 bit */
struct cube {
  WORD t; /* Bit 1 για μη συμπληρωματικές μεταβλητές.  */
  WORD f; /* Bit 1 για συμπληρωματικές μεταβλητές.    */
};
typedef struct cube CUBE;
CUBE P1, P2, P3; /* Τρεις κύβοι για χρήση από το πρόγραμμα.    */
```

Εδώ, WORD (λέξη) είναι ένα ακέραιος των 16 bit, ενώ ένας όρος γινομένου 16 μεταβλητών απεικονίζεται με μια εγγραφή δύο λέξεων, όπως φαίνεται στην Εικόνα 4-41(α). Η πρώτη λέξη ενός κύβου έχει ένα bit 1 για κάθε μεταβλητή του όρου γινομένου η οποία εμφανίζεται ως μη συμπληρωματική (ή "αληθής", `t`), ενώ η δεύτερη έχει ένα bit 1 για κάθε μεταβλητή η οποία εμφανίζεται ως συμπληρωματική (ή "ψευδής", `f`). Αν μια συγκεκριμένη θέση bit έχει τιμή 0 και στις δύο λέξεις, τότε η αντί-

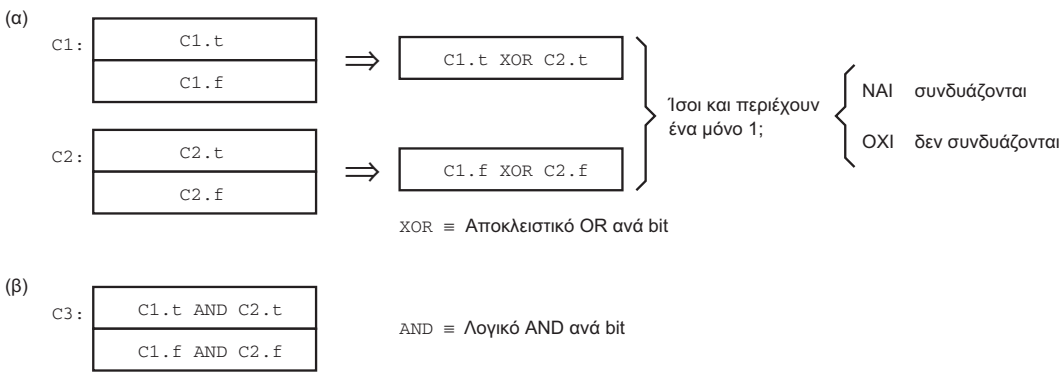


Εικόνα 4-41 Εσωτερική αναπαράσταση όρων γινομένου 16 μεταβλητών σε πρόγραμμα Pascal: (α) γενική μορφή, (β) P1 = X15·X12′·X10′·X9·X4′·X1·X0.

στοιχη μεταβλητή δεν εμφανίζεται, ενώ η περίπτωση κατά την οποία μια συγκεκριμένη θέση bit έχει τιμή 1 και στις δύο λέξεις δε χρησιμοποιείται. Έτσι λοιπόν, η μεταβλητή P1 του προγράμματος στην περίπτωση (β) αναπαριστά τον όρο γινομένου P1 = X15·X12′·X10′·X9·X4′·X1·X0. Αν θέλουμε να αναπαραστήσουμε μια λογική συνάρτηση F με 16 το πολύ μεταβλητές, που περιέχει 100 το πολύ όρους γινομένων, θα πρέπει να δηλώσουμε έναν πίνακα 100 κύβων:

```
CUBE F [100];    /* Χώρος αποθήκευσης λογικής συνάρτησης
                    με 100 το πολύ όρους γινομένου. */
```


Εικόνα 4-42 Χειρισμοί κύβων: (α) προσδιορισμός κατά πόσον οι δύο κύβοι είναι δυνατόν να συνδυαστούν μεταξύ τους με χρήση του θεωρήματος T10: όρος · X + όρος · X′ = όρος, (β) συνδυασμένοι μεταξύ τους κύβοι με χρήση του θεωρήματος T10.



Με τη βοήθεια της παραπάνω αναπαράστασης κύβων, μπορείτε να γράφετε σύντομες και αποδοτικές συναρτήσεις C που χειρίζονται όρους γινομένων με χρήσιμους τρόπους. Ο Πίνακας 4-8 δείχνει αρκετές τέτοιες συναρτήσεις. Σχετικά με τις δύο από τις συναρτήσεις αυτές, η Εικόνα 4-42 δείχνει πώς δύο κύβοι είναι δυνατόν να συγκριθούν και να συνδυαστούν μεταξύ τους, αν αυτό είναι εφικτό, με βάση το θεώρημα T10, σύμφωνα με το οποίο: όρος · X + όρος · X′ = όρος. Αυτό το θεώρημα λέει ότι δύο όροι γινομένου είναι δυνατόν να συνδυαστούν μεταξύ τους αν διαφέρουν ως προς μία μόνο μεταβλητή η οποία εμφανίζεται συμπληρωματική στον έναν όρο και μη συμπληρωματική στον άλλο. Συνδυάζοντας μεταξύ τους δύο m-διάστατους κύβους, προκύπτει ένας (m+1)-διάστατος

Πίνακας 4-8 Συναρτήσεις σύγκρισης και συνδυασμού κύβων, που χρησιμοποιούνται σε πρόγραμμα ελαχιστοποίησης.

```
int EqualCubes(CUBE C1, CUBE C2)  /* Αληθής όταν C1 και C2 ίδια.*/
{
 return ( (C1.t == C2.t) && (Cl.f == C2.f) );
}

int Oneone(WORD w) /* Αληθής όταν το w έχει ακριβώς ένα bit 1. */
                   /* Η βελτιστοποίηση της ταχύτητας αυτής     */
 int ones, b ;     /* της ρουτίνας έχει κρίσιμη σημασία */
 ones = 0;         /* και αφήνεται ως άσκηση για το χάκερ.     */
 for (b=0; b<MAX_VARS; b++) {
  if (W & 1) ones++;
  w = w>1;
}
 return((ones==1));
}

int Combinable(CUBE C1, CUBE C2)
{                         /* Αληθής αν C1 και C2 διαφέρουν μόνο  */
 WORD twordt, twordf;     /* σε μία μεταβλητή, που εμφανίζεται   */
                          /* αληθής στο ένα και ψευδής στο άλλο. */
 twordt = Cl.t ^ C2.t;
 twordf = Cl.f ^ C2.f;
 return( (twordt==twordf) && Oneone(twordt) );
}

void Combine(CUBE C1, CUBE C2, CUBE *C3)
            /* Συνδυάζει τα C1 και C2 με βάση το θεώρημα        */
{           /* T10 και αποθηκεύει το αποτέλεσμα στο C3.          */
            /* Παραδοχή ότι Combinable(Cl,C2) είναι αληθής       */
      C3->t = Ci.t & C2.t;
      C3->f = Ci.f & C2.f;
}
```

κύβος. Χρησιμοποιώντας την αναπαράσταση κύβων, μπορούμε να εφαρμόσουμε το συνδυασμένο θεώρημα σε μερικά παραδείγματα:

$$
\begin{aligned}
010 + 000 &= 0x0 \\
00111001 + 00111000 &= 0011100x \\
101xx0x0 + 101xx1x0 &= 101xxxx0 \\
x111xx00110x000x + x111xx00010x000x &= x111xx00x10x000x
\end{aligned}
$$

*4.4.2 Εύρεση πρωταρχικών όρων με συνδυασμό όρων γινομένου

Το πρώτο βήμα στον αλγόριθμο Quine-McCluskey είναι ο προσδιορισμός όλων των πρωταρχικών όρων της λογικής συνάρτησης. Με το χάρτη Karnaugh, αυτό επιτυγχάνεται οπτικά με τον προσδιορισμό των "μεγαλύτερων δυνατών ορθογωνίων συνόλων από άσσους". Στον αλγόριθμο, αυτό γίνεται με συστηματική επαναληπτική εφαρμογή του θεωρήματος T10 για το συνδυασμό ελαχίστων όρων, κατόπιν 1-διάστατων κύβων, 2-διάστατων κύβων κ.ο.κ. για τη δημιουργία των μεγαλύτερων δυνατών κύβων (μικρότεροι δυνατοί όροι γινομένων) που καλύπτουν μόνο τις τιμές 1 στη συνάρτηση.

Το πρόγραμμα C του Πίνακα 4-9 εφαρμόζει τον αλγόριθμο σε συναρτήσεις με 16 το πολύ μεταβλητές. Χρησιμοποιεί 2-διάσταστους πίνακες,

Πίνακας 4-9 Πρόγραμμα σε C που βρίσκει τους πρωταρχικούς όρους με τη βοήθεια του αλγόριθμου Quine-McCluskey.

```
i#define TRUE 1
#define FALSE 0
#define MAX_CUBES 50

void main()
{
 CUBE cubes[MAX_VARS+1][MAX_CUBES];
 int covered[MAX_VARS+1][MAX_CUBES];
 int numCubes[MAX_VARS+1];
 int m;       /* Τιμή του m σε έναν κύβο-m, π.χ., 'επίπεδο m.' */
 int j, k, p;  /* Ευρετήρια στους κύβους ή καλυπτόμενος πίνακας. */
 CUBE tempCube;
 int found;

 /* Αρχικές τιμές για αριθμό m-κύβων κάθε επιπέδου m. */
 for (m=0; m<MAX_VARS+1; m++) numCubes[m] = 0;

 /* Ανάγνωση λίστας ελάχιστων όρων (0-κύβους) που παρέχει ο χρήστης, */
 /* αποθήκευση στον υποπίνακα του κύβου [0,j], ρύθμιση του covered[0,j] */
 /* σε τιμή false για κάθε ελάχιστο όρο, και ρύθμιση του numCubes[0] */
 /* στο συνολικό αριθμών ελάχιστων όρων που διαβάστηκαν.*/
ReadMinterms;
```

Πίνακας 4-9 (συνέχεια)

```
 for (m=0; m<MAX_VARS; m++)         /* Για όλα τα επίπεδα εκτός του τελευταίου */
  for (j=0; j<numCubes[m]; j++)     /* Για όλους τους κύβους του επιπέδου  */
   for (k=j+1; k<numCubes[m]; k++) /* Για τους υπόλοιπους κύβους */
                                     /* αυτού του επιπέδου  */
    if (Combinable(cubes[m][j], cubes[m][k])) {
      /* Σήμανση κύβων ως καλυμμένων. */
      covered[m][j] = TRUE; covered[m][k] = TRUE;
      /* Συνδυασμός σε έναν κύβο-(m+1), αποθήκευση στο tempCube. */
      Combine(cubes[m][j], cubes[m][k], &tempCube);
      found = FALSE; /* Έλεγχος αν τον ξαναδημιουργήσαμε πριν. */
      for (p=0; p<numCubes[m+1]; p++)
       if (EqualCubes(cubes[m+1][p],tempCube)) found = TRUE;
      if (!found) { /* Προσθήκη του νέου κύβου στο επόμενο επίπεδο. */
       numCubes[m+1] = numCubes[m+1] + 1;
       cubes[m+1][numCubes[m+1]-1] = tempCube;
       covered[m+1][numCubes[m+1]-1] = FALSE;
      }
  }
for (m=0; m<MAX_VARS; m++)         /* Για όλα τα επίπεδα                    */
 for (j=0; j<numCubes[m]; j++)     /* Για όλους τους κύβους του επιπέδου  */
  /* Εκτύπωση των μη καλυπτόμενων κύβων – είναι οι πρωταρχικοί όροι  */
  if (!covered[m][j]) PrintCube(cubes[m][j]);
}
```

τους `cubes[m][j]` και `covered[m][j]`, για την παρακολούθηση των *m*-διάστατων κύβων `MAX_VARS`. Οι 0-διάστατοι κύβοι (ελάχιστοι όροι) δίνονται από το χρήστη. Ξεκινώντας με τους 0-διάστατους κύβους, το πρόγραμμα εξετάζει κάθε ζεύγος κύβων σε κάθε επίπεδο και τους συνδυάζει μεταξύ τους όποτε αυτό είναι δυνατόν σε κύβους στο επόμενο επίπεδο. Οι κύβοι που συνδυάζονται μεταξύ τους σε επόμενο επίπεδο σημειώνονται ως "καλυμμένοι". Οι κύβοι οι οποίοι δεν καλύπτονται είναι πρωταρχικοί όροι.

Παρότι το πρόγραμμα του Πίνακα 4-9 είναι μικρό, ένας έμπειρος προγραμματιστής θα αισθανθεί πολύ απαισιόδοξα με μια πρώτη ματιά στη δομή του. Ο εσωτερικός βρόχος `for` είναι ένθετος με βάθος τέσσερα επίπεδα, ενώ ο αριθμός των φορών που μπορεί να εκτελεστεί είναι $\texttt{MAX\_VARS}\cdot\texttt{MAX\_CUBES}^3$. Σωστά, είναι εκθέτης, δεν είναι δείκτης! Επιλέξαμε την τιμή `maxCube=1000` μάλλον αυθαίρετα (στην πραγματικότητα, είναι υπερβολικά αισιόδοξη επιλογή για πολλές συναρτήσεις) αλλά, αν συμφωνήσετε με αυτόν τον αριθμό, τότε ο εσωτερικός βρόχος είναι δυνατόν να εκτελεστεί *δισεκατομμύρια* φορές.

Φυσικά, ο μέγιστος αριθμός ελαχίστων όρων μιας συνάρτησης n μεταβλητών είναι 2^n, συνεπώς το πρόγραμμα του Πίνακα 4-9 πρέπει να δηλώνει το `maxCubes` ίσο με 2^{16}, τουλάχιστον για να χειρίζεται το μέγιστο δυνατό αριθμό 0-διάστατων κύβων. Μια τέτοια δήλωση δε θα είναι

υπερβολικά απαισιόδοξη. Αν μια συνάρτηση n μεταβλητών περιέχει έναν όρο γινομένου ίσο με μια απλή μεταβλητή, τότε στην πραγματικότητα χρειάζονται 2^{n-1} ελάχιστοι όροι για να καλύψουν αυτόν τον όρο γινομένου.

Για μεγαλύτερους κύβους, η κατάσταση είναι στην πραγματικότητα χειρότερη. Ο αριθμός των δυνατών m-διάστατων υποκύβων ενός n-διστατου κύβου είναι $\binom{n}{m} \times 2^{n-m}$, όπου ο διωνυμικός συντελεστής $\binom{n}{m}$ είναι ο αριθμός των τρόπων επιλογής m μεταβλητών έτσι ώστε να είναι x, ενώ το 2^{n-m} είναι ο αριθμός των τρόπων αντιστοίχισης των 0 και 1 στις υπόλοιπες μεταβλητές. Για συναρτήσεις 16 μεταβλητών, η χείριστη περίπτωση παρουσιάζεται για $m = 5$. Υπάρχουν 8.945.664 δυνατοί 5-διάστατοι υποκύβοι ενός 16-διάστατου κύβου. Ο συνολικός αριθμός των διακριτών m-διάστατων υποκύβων ενός n-διάστατου κύβου για όλες τις τιμές του m είναι 3^n. Έτσι, ένα γενικό πρόγραμμα ελαχιστοποίησης ενδέχεται να απαιτεί πολύ περισσότερη μνήμη απ΄ ό,τι έχουμε κατανείμει στον Πίνακα 4-9.

Υπάρχουν μερικά πράγματα τα οποία μπορούμε να κάνουμε για να βελτιστοποιήσουμε το χώρο αποθήκευσης και το χρόνο εκτέλεσης που απαιτείται στον Πίνακα 4-9 (δείτε τις Ασκήσεις 4.77-4.80), αλλά είναι μηδαμινά σε σύγκριση με τη συντριπτική συνδυαστική πολυπλοκότητα του προβλήματος. Έτσι, ακόμη και στους σημερινούς γρήγορους υπολογιστές με τις τεράστιες μνήμες, η άμεση εφαρμογή του αλγόριθμου Quine-McCluskey για τη δημιουργία πρωταρχικών όρων περιορίζεται γενικά μόνο σε συναρτήσεις με λίγες μεταβλητές (λιγότερες από 15-20).

*4.4.3 Εύρεση ελάχιστης κάλυψης με χρήση του πίνακα πρωταρχικών όρων

Το δεύτερο βήμα για την ελαχιστοποίηση μιας συνδυαστικής λογικής συνάρτησης, από τη στιγμή που έχουμε μια λίστα όλων των πρωταρχικών όρων, είναι η επιλογή ενός ελαχίστου υποσυνόλου αυτών για την κάλυψη όλων των 1 στη συνάρτηση. Ο αλγόριθμος Quine-McCluskey χρησιμοποιεί ένα διδιάστατο πίνακα που λέγεται *πίνακας πρωταρχικών όρων* για το σκοπό αυτό. Στη Εικόνα 4-43(α) εμφανίζεται ένας μικρός αλλά αντιπροσωπευτικός πίνακας πρωταρχικών όρων, που αντιστοιχεί στο πρόβλημα ελαχιστοποίησης με χρήση χάρτη Karnaugh της Εικόνας 4-35. Υπάρχει μια στήλη για κάθε ελάχιστο όρο της συνάρτησης και μια γραμμή για κάθε πρωταρχικό όρο. Κάθε καταχώριση είναι ένα bit το οποίο είναι 1 αν και μόνο αν ο πρωταρχικός όρος της συγκεκριμένης γραμμής καλύπτει τον ελάχιστο όρο της συγκεκριμένης στήλης (στην εικόνα απεικονίζεται ως σύμβολο ελέγχου).

πίνακας πρωταρχικών όρων

Τα βήματα για την επιλογή των πρωταρχικών όρων με τον πίνακα είναι ανάλογα με τα βήματα που χρησιμοποιήσαμε στην Ενότητα 4.3.5 με τους χάρτες Karnaugh:

ελάχιστοι όροι

πρωταρχικοί όροι

(α)

	2	6	7	9	13	15
A	√	√				
B		√	√			
C			√			√
D					√	√
E				√	√	

(β)

	2	6	7	9	13	15
A	√	√				
B		√	√			
C			√			√
D					√	√
E				√	√	

(γ)

	7	15
B	√	
C	√	√
D		√

(δ)

	7	15
B	√	
C	√	√
D		√

(ε)

	7	15
C	√	√

Εικόνα 4-43 Πίνακες πρωταρχικών όρων: (α) αρχικός πίνακας, (β) εμφάνιση διακεκριμένων κελιών με τιμή 1 και ουσιωδών πρωταρχικών όρων, (γ) μετά από την αφαίρεση των ουσιωδών πρωταρχικών όρων, (δ) εμφάνιση των απαλειφθεισών γραμμών, (ε) μετά από την αφαίρεση των απαλειφθεισών γραμμών, εμφάνιση των δευτερευόντων ουσιωδών πρωταρχικών όρων.

1. Εντοπίζουμε τα διακεκριμένα κελιά με τιμή 1. Αυτά εντοπίζονται εύκολα στον πίνακα ως στήλες με έναν άσσο, όπως φαίνεται στην Εικόνα 4-43(β).
2. Συμπεριλαμβάνουμε όλους τους ουσιώδεις πρωταρχικούς όρους στο ελάχιστο άθροισμα. Μια γραμμή η οποία περιέχει ένα σύμβολο ελέγχου σε μία ή περισσότερες στήλες διακεκριμένων κελιών με τιμή 1 αντιστοιχεί σε έναν ουσιώδη πρωταρχικό όρο.
3. Εξαιρούμε από τη θεώρησή μας τους ουσιώδεις πρωταρχικούς όρους και τα κελιά με τιμή 1 (ελάχιστοι όροι) που καλύπτουν αυτοί. Στον πίνακα, αυτό γίνεται με διαγραφή των αντίστοιχων γραμμών και στηλών που σημειώνονται με σκίαση στην Εικόνα 4-43(β). Αν κάποιες γραμμές που απομένουν δεν έχουν σημεία ελέγχου, διαγράφονται κι αυτές. Οι αντίστοιχοι πρωταρχικοί όροι είναι *πλεονάζοντες*, δηλαδή καλύπτονται πλήρως από τους ουσιώδεις πρωταρχικούς όρους. Αυτό το βήμα αφήνει τον μικρότερου μεγέθους πίνακα που φαίνεται στην περίπτωση (γ).

   πλεονάζων πρωταρχικός όρος
4. Εξαιρούμε από τη θεώρησή μας όλους τους πρωταρχικούς όρους που "επισκιάζονται" από άλλους με ίσο ή μικρότερο κόστος. Στον πίνακα, αυτό γίνεται με διαγραφή όλων των γραμμών των οποίων οι στήλες που περιέχουν σύμβολα ελέγχου είναι γνήσιο υποσύνολο άλλης γραμμής, και διαγράφοντας όλες τις γραμμές εκτός από μία όταν ανήκουν σε ένα σύνολο γραμμών με πανομοιότυπες στήλες που περιέχουν σύμβολα ελέγχου. Αυτό απεικονίζεται με σκίαση στην περίπτω-

ση (δ) και το αποτέλεσμα είναι ένας μικρότερου μεγέθους πίνακας της περίπτωσης (ε).

Όταν μια συνάρτηση υλοποιείται με PLD, μπορούμε να υποθέσουμε ότι όλοι οι πρωταρχικοί όροι της έχουν το ίδιο κόστος, επειδή όλες οι πύλες AND μιας διάταξης PLD έχουν όλες τις εισόδους διαθέσιμες. Διαφορετικά, οι πρωταρχικοί όροι πρέπει να ταξινομούνται και να επιλέγονται σύμφωνα με τον αριθμό των εισόδων των πυλών AND.

5. Εντοπίζουμε τα διακεκριμένα κελιά με τιμή 1 και συμπεριλαμβάνουμε όλους τους δευτερεύοντες ουσιώδεις πρωταρχικούς όρους στο ελάχιστο άθροισμα. Όπως και πριν, κάθε γραμμή που περιέχει ένα σημείο ελέγχου σε μία ή περισσότερες στήλες διακεκριμένων κελιών με τιμή 1 αντιστοιχεί σε ένα δευτερεύοντα ουσιώδη πρωταρχικό όρο.
6. Αν όλες οι στήλες που απομένουν καλύπτονται από τους δευτερεύοντες ουσιώδεις πρωταρχικούς όρους, όπως στην περίπτωση (ε), έχουμε τελειώσει. Διαφορετικά, αν είχαν βρεθεί δευτερεύοντες ουσιώδεις πρωταρχικοί όροι στο προηγούμενο βήμα, επιστρέφουμε στο βήμα 3 και επαναλαμβάνουμε. Σε διαφορετική περίπτωση πρέπει να εφαρμοστεί η μέθοδος διακλάδωσης, όπως περιγράφεται στην Ενότητα 4.3.5. Αυτό συνεπάγεται την επιλογή γραμμών, μίας γραμμής κάθε φορά, την επεξεργασία αυτών σαν να ήταν ουσιώδεις, καθώς και αναδρομική εφαρμογή των βημάτων 3-6.

Αν και ο πίνακας των πρωταρχικών όρων επιτρέπει έναν αρκετά απλό αλγόριθμο επιλογής πρωταρχικών όρων, η δομή δεδομένων που απαιτείται σε ένα αντίστοιχο πρόγραμμα υπολογιστή είναι τεράστια, εφόσον απαιτεί $p \cdot 2^n$ bit, όπου p είναι ο αριθμός των πρωταρχικών όρων και n είναι ο αριθμός των bit εισόδου (υποθέτουμε ότι η δεδομένη συνάρτηση δίνει έξοδο 1 για τους περισσότερους συνδυασμούς εισόδων). Ακόμη χειρότερα, η εκτέλεση των βημάτων αυτών που τόσο άνετα περιγράψαμε με λίγες φράσεις απαιτεί τεράστιους υπολογισμούς.

*4.4.4 Άλλες μέθοδοι ελαχιστοποίησης

Αν και οι προηγούμενες ενότητες αποτελούν μια εισαγωγή στους αλγόριθμους λογικής ελαχιστοποίησης, οι μέθοδοι που περιγράφουν δεν είναι με κανέναν τρόπο οι πιο πρόσφατες ούτε οι καλύτερες. Παρακινούμενοι από την ολοένα αυξανόμενη πυκνότητα των ολοκληρωμένων κυκλωμάτων VLSI, πολλοί ερευνητές έχουν ανακαλύψει πιο αποτελεσματικούς τρόπους για την ελαχιστοποίηση των συναρτήσεων των συνδυαστικών λογικών κυκλωμάτων. Τα αποτελέσματά τους κατατάσσονται κατά προσέγγιση σε τρεις κατηγορίες:

1. *Υπολογιστικά περιβάλλοντα*. Οι βελτιωμένοι αλγόριθμοι χρησιμοποιούν τυπικά έξυπνες δομές δεδομένων ή κάνουν αναδιάταξη στη σειρά των βημάτων για να περιοριστούν οι απαιτήσεις μνήμης και ο χρόνος εκτέλεσης των κλασικών αλγορίθμων.

2. *Εμπειρικές μέθοδοι*. Μερικά προβλήματα ελαχιστοποίησης είναι απλώς υπερβολικά μεγάλα για να επιλυθούν με τη χρήση ενός "ακριβούς" αλγόριθμου. Αυτά τα προβλήματα είναι δυνατόν να αντιμετωπιστούν με τη βοήθεια συντομεύσεων και εύστοχων εικασιών για τον περιορισμό του μεγέθους της μνήμης και του χρόνου εκτέλεσης απ' ό,τι θα απαιτείτο από έναν "ακριβή" αλγόριθμο. Ωστόσο, οι εμπειρικές μέθοδοι προσπαθούν να βρουν μια "σχεδόν ελάχιστη" παράσταση μιας λογικής συνάρτησης, και όχι μια αποδεδειγμένα ελάχιστη παράσταση.

   Ακόμη και για προβλήματα τα οποία είναι δυνατόν να επιλυθούν με μια "ακριβή" μέθοδο, μια εμπειρική μέθοδος συνήθως βρίσκει μια καλή λύση δέκα φορές πιο γρήγορα. Μάλιστα, το πιο πετυχημένο τέτοιο πρόγραμμα, το Espresso-II, παράγει ελάχιστα ή σχεδόν ελάχιστα αποτελέσματα στην πλειοψηφία των προβλημάτων (στα πλαίσια ενός ή δύο όρων γινομένων), συμπεριλαμβανομένων και προβλημάτων με δεκάδες εισόδους και εκατοντάδες όρους γινομένων.

3. *Διαφορετική θεώρηση*. Όπως αναφέραμε νωρίτερα, ο χειρισμός της ελαχιστοποίησης πολλών εξόδων είναι δυνατόν να γίνει με απλές και σχεδόν μηχανικές τροποποιήσεις των μεθόδων ελαχιστοποίησης μίας εξόδου. Ωστόσο, θεωρώντας την ελαχιστοποίηση πολλών εξόδων ως πρόβλημα λογικής πολλών τιμών (μη δυαδικής), οι σχεδιαστές του αλγόριθμου Espresso-MV είχαν τη δυνατότητα να κάνουν ουσιαστικές βελτιώσεις στην απόδοση του Espresso-II.

Περισσότερες πληροφορίες γι' αυτές τις μεθόδους μπορείτε να βρείτε στις Παραπομπές.

*4.5 Κίνδυνοι χρονισμού

Οι μέθοδοι ανάλυσης που αναπτύξαμε στην Ενότητα 4.2 δε λαμβάνουν υπόψη την καθυστέρηση του κυκλώματος και προβλέπουν μόνο τη *συμπεριφορά σταθερής κατάστασης* των συνδυαστικών λογικών κυκλωμάτων. Αυτό σημαίνει ότι προβλέπουν την έξοδο ενός κυκλώματος ως συνάρτηση των εισόδων του, υπό την προϋπόθεση ότι οι είσοδοι έχουν παραμείνει σταθερές για μεγάλο χρονικό διάστημα, σε σχέση με τις καθυστερήσεις στα ηλεκτρονικά του κυκλώματος. Ωστόσο, στην Ενότητα 3.6 δείξαμε ότι η πραγματική καθυστέρηση από μια αλλαγή εισόδου στην αντίστοιχη αλλαγή εξόδου σε ένα πραγματικό λογικό κύκλωμα δεν είναι μηδενική και εξαρτάται από πολλούς παράγοντες.

συμπεριφορά σταθερής κατάστασης

Λόγω των καθυστερήσεων του κυκλώματος, η *μεταβατική συμπεριφορά* ενός λογικού κυκλώματος μπορεί να διαφέρει από εκείνη που προβλέπεται από μια ανάλυση σταθερής κατάστασης. Πιο συγκεκριμένα, η έξοδος ενός κυκλώματος μπορεί να παράγει ένα σύντομο παλμό (παροδικό σφάλμα), που συχνά ονομάζεται *glitch*, τη στιγμή κατά την οποία η ανάλυση σταθερής κατάστασης προβλέπει ότι η έξοδος δε θα αλλάξει. Ένας *κίνδυνος* λέγεται ότι υπάρχει όταν ένα κύκλωμα έχει τη δυνατότη-

μεταβατική συμπεριφορά

παροδικό σφάλμα (glitch)

κίνδυνος

τα να παραγάγει ένα τέτοιο παροδικό σφάλμα. Το κατά πόσον το παροδικό αυτό σφάλμα θα συμβεί στην πράξη εξαρτάται από τις ακριβείς καθυστερήσεις και τα λοιπά ηλεκτρικά χαρακτηριστικά του κυκλώματος. Επειδή οι παράμετροι αυτές είναι δύσκολο να ελεγχθούν στα κυκλώματα παραγωγής, οι σχεδιαστές λογικών κυκλωμάτων πρέπει να είναι προετοιμασμένοι να εξαλείψουν τους κινδύνους (την *πιθανότητα εμφάνισης ενός παροδικού σφάλματος*), ακόμη κι αν ένα παροδικό σφάλμα είναι δυνατόν να συμβεί μόνο στη χειρότερη περίπτωση συνδυασμού λογικών και ηλεκτρικών συνθηκών.

*4.5.1 Στατικοί κίνδυνοι

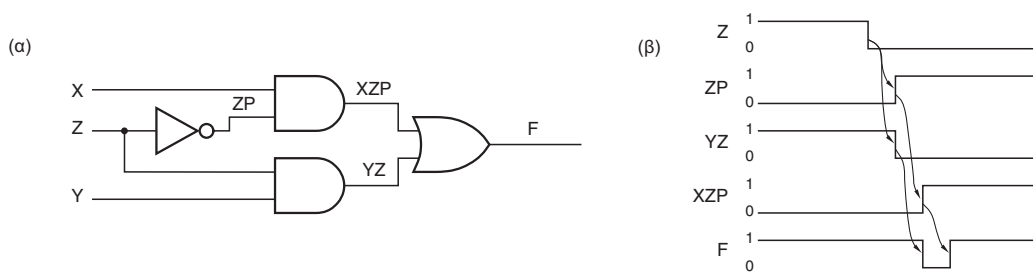
στατικός κίνδυνος 1

Ο *στατικός κίνδυνος 1* είναι η πιθανότητα να παράγει η έξοδος του κυκλώματος ένα παροδικό σφάλμα στάθμης 0 όταν η έξοδος αναμένεται να παραμείνει σταθερή στην τιμή 1 με βάση τη στατική ανάλυση λειτουργίας του κυκλώματος. Ο τυπικός ορισμός δίνεται παρακάτω.

Ορισμός: Ο στατικός κίνδυνος 1 είναι ένα ζεύγος συνδυασμών εισόδων οι οποίοι: (α) διαφέρουν μεταξύ τους κατά μία μόνο μεταβλητή εισόδου και (β) δίνουν και οι δύο έξοδο 1, τέτοιο ώστε να είναι δυνατόν να συμβεί μια στιγμιαία έξοδος 0 κατά τη διάρκεια μετάβασης της μεταβλητής εισόδου κατά την οποία διαφέρουν αυτοί οι συνδυασμοί εισόδου.

Για παράδειγμα, θεωρούμε το λογικό κύκλωμα της Εικόνας 4-44(α). Υποθέτουμε ότι τα X και Y είναι και τα δύο 1, ενώ το Z αλλάζει από 1 σε 0. Τότε, η περίπτωση (β) δείχνει το διάγραμμα χρονισμού, με την παραδοχή ότι η καθυστέρηση διάδοσης μέσα από κάθε πύλη ή αντιστροφέα ισούται με μια μονάδα χρόνου. Ακόμη κι αν η "στατική" ανάλυση προβλέπει ότι η έξοδος θα είναι 1 και για τους δύο συνδυασμούς εισόδων X,Y,Z = 111 και X,X,Z = 110, το διάγραμμα χρονισμού δείχνει ότι η F μεταβαίνει στην τιμή 0 για μια μονάδα χρόνου κατά τη διάρκεια της μετάβασης από 1 σε 0 στο Z, λόγω της καθυστέρησης στον αντιστροφέα που δημιουργεί ένα Z′.

Εικόνα 4-44 Κύκλωμα με στατικούς κινδύνους 1: (α) λογικό διάγραμμα, (β) διάγραμμα χρονισμού.



Ο *στατικός κίνδυνος 0* είναι η πιθανότητα εμφάνισης ενός παροδικού σφάλματος στάθμης 1 όταν η έξοδος του κυκλώματος αναμένεται να παραμείνει σταθερή στην τιμή 0:

Ορισμός: Ο στατικός κίνδυνος 0 είναι ένα ζεύγος συνδυασμών εισόδων οι οποίοι: (α) διαφέρουν μεταξύ τους κατά μία μόνο μεταβλητή εισόδου και (β) δίνουν και οι δύο έξοδο 0, τέτοιο ώστε να είναι δυνατόν να συμβεί μια στιγμιαία έξοδος 1 κατά τη διάρκεια μετάβασης της μεταβλητής εισόδου κατά την οποία διαφέρουν αυτοί οι συνδυασμοί εισόδου. *στατικός κίνδυνος 0*

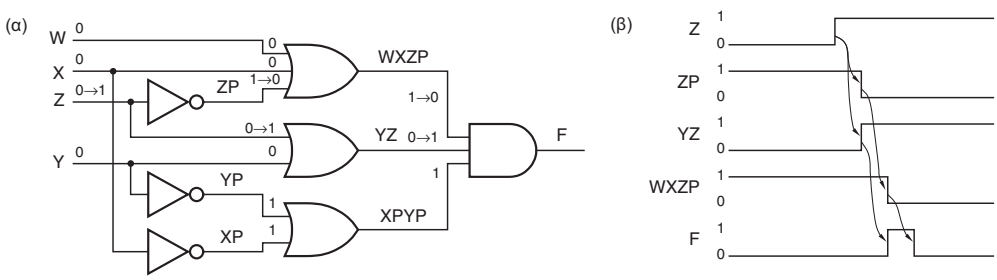
Αφού ο στατικός κίνδυνος 0 είναι απλώς το δυικό του στατικού κινδύνου 1, ένα κύκλωμα OR-AND το οποίο είναι το δυικό της Εικόνας 4-44(α) θα έχει στατικό κίνδυνο 0.

Στην Εικόνα 4-45(α) φαίνεται ένα κύκλωμα OR-AND με τέσσερις στατικούς κινδύνους 0. Ένας από τους κινδύνους συμβαίνει όταν W,X,Y = 000 και το Z αλλάζει, όπως φαίνεται στην περίπτωση (β). Όταν διαβάσετε την επόμενη ενότητα, θα μπορέσετε να βρείτε τους άλλους τρεις κινδύνους και να τους εξαλείψετε όλους.

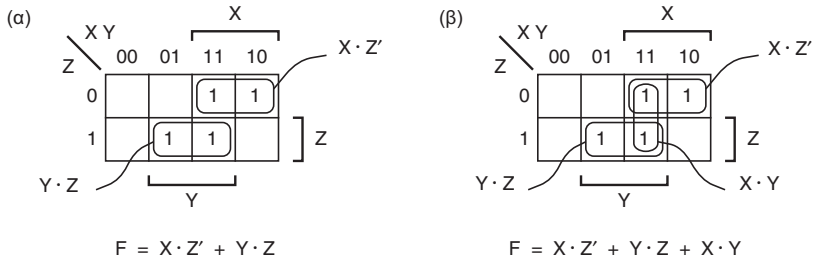
*4.5.2 Εύρεση στατικών κινδύνων με χρήση χαρτών

Για τον εντοπισμό των στατικών κινδύνων σε ένα κύκλωμα αθροίσματος γινομένων ή γινομένου αθροισμάτων δύο επιπέδων, μπορεί να χρησιμοποιηθεί ένας χάρτης Karnaugh. Η ύπαρξη ή η μη ύπαρξη στατικών κινδύνων εξαρτάται από τη σχεδίαση του κυκλώματος για μια λογική συνάρτηση.

Ένα σωστά σχεδιασμένο κύκλωμα αθροίσματος γινομένων (AND-OR) δύο επιπέδων δεν έχει στατικούς κινδύνους 0. Ένας στατικός κίνδυνος 0 θα υπάρχει σε ένα τέτοιο κύκλωμα μόνο αν τόσο μια μεταβλητή όσο και το συμπλήρωμά της συνδεθούν στην ίδια πύλη AND, γεγονός που θα ήταν ανόητο. Ωστόσο, το κύκλωμα *ενδέχεται* να έχει στατικούς κινδύνους 1. Η ύπαρξή τους μπορεί να προβλεφθεί από ένα χάρτη Karnaugh,



Εικόνα 4-45 Κύκλωμα με στατικούς κινδύνους 0: (α) λογικό διάγραμμα, (β) διάγραμμα χρονισμού.



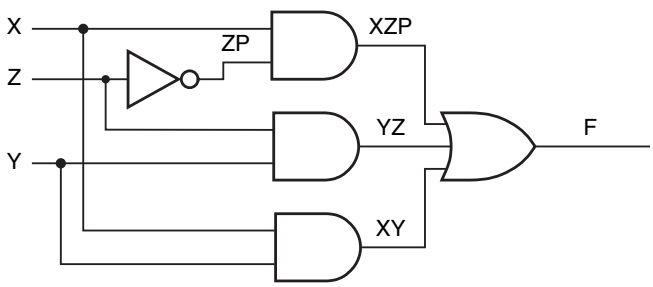
Εικόνα 4-46
Χάρτης Karnaugh για το κύκλωμα της Εικόνας 4-44: (α) όπως σχεδιάστηκε αρχικά, (β) με εξάλειψη των στατικών κινδύνων 1.

όπου οι όροι γινομένου που αντιστοιχούν στις πύλες AND του κυκλώματος είναι τοποθετημένοι σε κύκλο.

Στην Εικόνα 4-46(α) φαίνεται ο χάρτης Karnaugh για το κύκλωμα της Εικόνας 4-44. Στο χάρτη αυτόν, είναι σαφές ότι δεν υπάρχει κανένας όρος γινομένου που να καλύπτει και τους δύο συνδυασμούς εισόδων X,Y,Z = 111 και X,Y,Z = 110. Έτσι, διαισθητικά, είναι δυνατόν να συμβεί στιγμιαίο "παροδικό σφάλμα" στην τιμή 0 για την έξοδο της πύλης αν η έξοδος της πύλης AND η οποία καλύπτει έναν από τους συνδυασμούς μεταβεί στην τιμή 0 προτού η έξοδος της πύλης AND η οποία καλύπτει τον άλλο συνδυασμό εισόδων μεταβεί στην τιμή 1. Ο τρόπος εξάλειψης του κινδύνου είναι επίσης σαφής: απλώς πρέπει να συμπεριληφθεί ένας επιπλέον όρος γινομένου (πύλη AND) που θα καλύψει το επικίνδυνο ζεύγος εισόδων, όπως φαίνεται στην Εικόνα 4-46(β). Αποδεικνύεται ότι ο επιπλέον όρος γινομένου είναι ο *όρος κοινής συναίνεσης* των δύο αρχικών όρων. Γενικά, πρέπει να προσθέτουμε τους όρους κοινής συναίνεσης μεταξύ τους για να εξαλείφουμε τους κινδύνους. Στην Εικόνα 4-47 φαίνεται το αντίστοιχο κύκλωμα χωρίς κινδύνους.

όρος κοινής συναίνεσης

Ένα άλλο παράδειγμα φαίνεται στην Εικόνα 4-48. Σε αυτό το παράδειγμα πρέπει να προστεθούν τρεις όροι γινομένου για τον περιορισμό των στατικών κινδύνων 1.



Εικόνα 4-47
Κύκλωμα από το οποίο έχει εξαλειφθεί ο στατικός κίνδυνος 1.

Ένα σωστά σχεδιασμένο κύκλωμα γινομένου αθροισμάτων (OR-AND) δύο επιπέδων δεν έχει στατικούς κινδύνους 1. Μπορεί ωστόσο να έχει στατικούς κινδύνους 0. Αυτοί οι κίνδυνοι είναι δυνατόν να εντοπιστούν και να εξαλειφθούν με τη μελέτη των γειτονικών 0 στο χάρτη Karnaugh, με τρόπο δυικό των προαναφερθέντων.

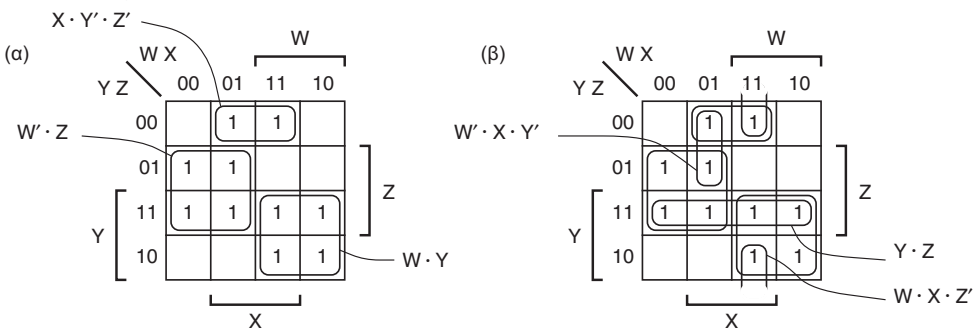
*4.5.3 Δυναμικοί κίνδυνοι

δυναμικός κίνδυνος

Δυναμικός κίνδυνος είναι η πιθανότητα αλλαγής μιας εξόδου περισσότερες από μία φορές ως αποτέλεσμα μίας μόνο μετάβασης εισόδου. Αν υπάρχουν πολλές διαδρομές με διαφορετικές καθυστερήσεις από την αλλαγή της εισόδου στην αλλαγή της εξόδου, είναι δυνατόν να συμβούν πολλές μεταβάσεις εξόδου.

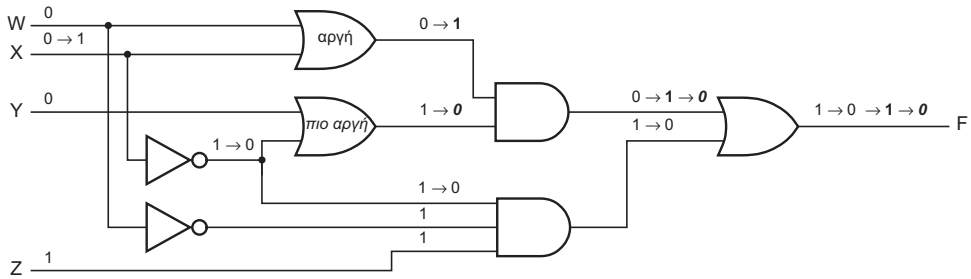
Για παράδειγμα, ας θεωρήσουμε το κύκλωμα της Εικόνας 4-49. Έχει τρεις διαφορετικές διαδρομές από την είσοδο X έως την έξοδο F. Μία από τις διαδρομές διέρχεται μέσω μιας αργής πύλης OR, ενώ μια άλλη διέρχεται μέσω μιας πύλης OR η οποία είναι ακόμη πιο αργή. Αν η είσοδος του κυκλώματος είναι W,X,Y,Z = 0,0,0,1, τότε η έξοδος θα είναι 1, όπως φαίνεται στην εικόνα. Τώρα, ας υποθέσουμε ότι αλλάζουμε την είσοδο X στην τιμή 1. Αν υποθέσουμε ότι όλες οι είσοδοι εκτός από τις δύο που έχουν επισημανθεί ως "αργή" και "πιο αργή" είναι πολύ γρήγορες, οι μεταβάσεις που εμφανίζονται με κανονικά γράμματα συμβαίνουν μετά και η έξοδος γίνεται 0. Κατόπιν, η έξοδος της "αργής" πύλης OR αλλάζει, δημιουργώντας τις μεταβάσεις, που εμφανίζονται με έντονα γράμματα, και η έξοδος γίνεται 1. Τελικά, η έξοδος της "πιο αργής" πύλης OR αλλάζει, δημιουργώντας τις μεταβάσεις, που εμφανίζονται με έντονα και πλάγια γράμματα, και η έξοδος μεταβαίνει στην τελική κατάσταση 0.

Δυναμικοί κίνδυνοι δεν εμφανίζονται σε ένα σωστά σχεδιασμένο κύκλωμα AND-OR ή OR-AND δύο επιπέδων, δηλαδή σε ένα κύκλωμα στο οποίο καμία μεταβλητή μαζί με το συμπλήρωμά της δε συνδέονται με



Εικόνα 4-48 Χάρτης Karnaugh για ένα άλλο κύκλωμα αθροίσματος γινομένων: (α) όπως σχεδιάστηκε αρχικά, (β) με επιπλέον όρους γινομένου για την κάλυψη των στατικών κινδύνων 1.

την ίδια πύλη πρώτου επιπέδου. Σε κυκλώματα πολλών επιπέδων, οι δυναμικοί κίνδυνοι είναι δυνατόν να αποκαλυφθούν με χρήση της μεθόδου που περιγράφεται στις Παραπομπές.



Εικόνα 4-49 Κύκλωμα με δυναμικό κίνδυνο.

*4.5.4 Σχεδίαση κυκλωμάτων χωρίς κινδύνους

Συνδυαστικά κυκλώματα χωρίς κινδύνους απαιτούνται σε λίγες μόνο περιπτώσεις, όπως π.χ. στη σχεδίαση ακολουθιακών κυκλωμάτων ανάδρασης. Οι τεχνικές για την εύρεση των κινδύνων σε τυχαία κυκλώματα, οι οποίες περιγράφονται στις Παραπομπές, είναι αρκετά δύσκολες στη χρήση. Έτσι, όταν απαιτείται σχεδίαση χωρίς κινδύνους, είναι καλύτερα να χρησιμοποιείτε μια δομή κυκλώματος που η ανάλυσή της είναι εύκολη.

Πιο συγκεκριμένα, έχουμε δείξει ότι ένα σωστά σχεδιασμένο κύκλωμα AND-OR δύο επιπέδων δεν έχει στατικούς κινδύνους 0 ή δυναμικούς κινδύνους. Σε ένα τέτοιο κύκλωμα μπορεί να υπάρχουν στατικοί κίνδυνοι 1, αλλά αυτοί είναι δυνατόν να βρεθούν και να απαλειφθούν με χρήση της μεθόδου του χάρτη που περιγράψαμε νωρίτερα. Αν το κόστος δεν

ΟΙ ΠΕΡΙΣΣΟΤΕΡΟΙ ΚΙΝΔΥΝΟΙ ΔΕΝ ΕΙΝΑΙ ΕΠΙΚΙΝΔΥΝΟΙ!

Οποιοδήποτε συνδυαστικό κύκλωμα μπορεί να αναλυθεί ως προς την ύπαρξη κινδύνων. Ωστόσο, ένα καλά σχεδιασμένο *σύγχρονο* (*synchronous*) ψηφιακό σύστημα είναι δομημένο έτσι ώστε η ανάλυση κινδύνων να μην είναι απαραίτητη για τα περισσότερα από τα κυκλώματά του. Σε ένα σύγχρονο σύστημα, όλες οι είσοδοι ενός συνδυαστικού κυκλώματος αλλάζουν σε ένα συγκεκριμένο χρόνο, και οι έξοδοι δε λαμβάνονται υπόψη αν δεν έχει παρέλθει αρκετός χρόνος που να τους επιτρέπει να καταλήξουν σε μια τιμή σταθερής κατάστασης. Η ανάλυση κινδύνων και η απάλειψή τους είναι συνήθως απαραίτητη μόνο στη σχεδίαση ασύγχρονων ακολουθιακών κυκλωμάτων, όπως είναι τα ακολουθιακά κυκλώματα ανάδρασης που περιγράφονται στην Ενότητα 7.9. Σπάνια θα έχετε λόγο να σχεδιάσετε ένα τέτοιο κύκλωμα αλλά, αν το κάνετε, η κατανόηση των κινδύνων θα είναι απολύτως ουσιώδης για ένα αξιόπιστο αποτέλεσμα.

είναι πρόβλημα, τότε μια δυνατή μέθοδος για να πετύχουμε μια υλοποίηση χωρίς κινδύνους είναι να χρησιμοποιήσουμε το πλήρες άθροισμα, δηλαδή το άθροισμα όλων των πρωταρχικών όρων της λογικής συνάρτησης (δείτε την Άσκηση 4.84). Με δυικό τρόπο, μπορεί κανείς να σχεδιάσει ένα λογικό κύκλωμα OR-AND δύο επιπέδων χωρίς κινδύνους για κάθε λογική συνάρτηση. Τελικά, σημειώστε πως ό,τι έχουμε πει σχετικά με τα κυκλώματα AND-OR εφαρμόζεται φυσικά στις αντίστοιχες σχεδιάσεις NAND-NAND, ενώ ό,τι έχουμε πει για τα OR-AND εφαρμόζεται στα NOR-NOR.

4.6 Γλώσσα περιγραφής υλικού ABEL

Η ABEL είναι μια γλώσσα περιγραφής υλικού (HDL) η οποία επινοήθηκε για να επιτρέπει στους σχεδιαστές να ορίζουν λογικές συναρτήσεις που θα υλοποιηθούν σε προγραμματιζόμενες λογικές διατάξεις (PLD). Ένα πρόγραμμα ABEL είναι ένα αρχείο κειμένου που περιέχει διάφορα στοιχεία:

- Τεκμηρίωση, που περιλαμβάνει το όνομα του προγράμματος και σχόλια.
- Δηλώσεις, οι οποίες προσδιορίζουν τις εισόδους και τις εξόδους των λογικών συναρτήσεων που εκτελούνται.
- Εντολές, οι οποίες καθορίζουν τις λογικές συναρτήσεις που εκτελούνται.
- Συνήθως, μια δήλωση του τύπου διάταξης PLD ή άλλης διάταξης-στόχου στην οποία πρόκειται να εκτελούνται οι καθορισμένες λογικές συναρτήσεις.
- Συνήθως, "διανύσματα ελέγχου" τα οποία καθορίζουν τις αναμενόμενες εξόδους των λογικών συναρτήσεων για ορισμένες εισόδους.

επεξεργαστής γλώσσας ABEL
μεταγλωττιστής ABEL

Η ABEL υποστηρίζεται από έναν *επεξεργαστή γλώσσας ABEL* (*ABEL language processor*), τον οποίο θα αποκαλούμε απλώς *μεταγλωττιστή ABEL*. Η δουλειά του μεταγλωττιστή είναι να μεταφράζει το αρχείο κειμένου της γλώσσας ABEL σε ένα "υπόδειγμα ασφαλειών" (fuse pattern) το οποίο μπορούμε να κατεβάσουμε σε μια πραγματική διάταξη PLD. Παρά το γεγονός ότι οι περισσότερες PLD μπορούν να προγραμματίζονται φυσικά μόνο με υποδείγματα που αντιστοιχούν σε παραστάσεις αθροίσματος γινομένων, η ABEL επιτρέπει να αναπαρίστανται συναρτήσεις PLD με πίνακες αληθείας ή με ένθετες προτάσεις "`IF`", καθώς επίσης και με οποιαδήποτε μορφή αλγεβρικής παράστασης. Ο μεταγλωττιστής χειρίζεται αυτές τις μορφές και ελαχιστοποιεί τις συναρτήσεις που προκύπτουν ώστε να ταιριάζουν, αν αυτό είναι δυνατόν, στη διαθέσιμη δομή της διάταξης PLD.

Αργότερα, στην Ενότητα 5.3, θα μιλήσουμε για δομές PLD, υποδείγματα ασφαλειών, και άλλα σχετικά θέματα, και θα δείξουμε πώς μπορούμε να χρησιμοποιήσουμε προγράμματα ABEL σε συγκεκριμένα τσιπ

PLD. Στο μεταξύ, θα δείξουμε πώς μπορεί να χρησιμοποιηθεί η ABEL για τον ορισμό συναρτήσεων συνδυαστικής λογικής χωρίς να πρέπει απαραίτητα να δηλώσουμε τον τύπο της διάταξης προορισμού. Στη συνέχεια, στην Ενότητα 7.11, θα κάνουμε το ίδιο για τις ακολουθιακές λογικές συναρτήσεις.

4.6.1 Δομή προγράμματος ABEL

Ο Πίνακας 4-10 δείχνει την τυπική δομή ενός προγράμματος ABEL και ο Πίνακας 4-11 δείχνει ένα πραγματικό πρόγραμμα το οποίο εμφανίζει τα παρακάτω γλωσσικά χαρακτηριστικά:

αναγνωριστικό

- Τα *αναγνωριστικά* πρέπει να ξεκινούν με γράμμα ή χαρακτήρα υπογράμμισης, επιτρέπεται να περιέχουν 31 το πολύ γράμματα, ψηφία, και χαρακτήρες υπογράμμισης, και έχουν διάκριση πεζών - κεφαλαίων.

`module`

- Ένα αρχείο προγράμματος ξεκινά με μια εντολή `module`, η οποία συσχετίζει ένα αναγνωριστικό (`Alarm_Circuit`) με τη λειτουργική μονάδα του προγράμματος. Τα μεγάλα προγράμματα είναι δυνατόν να έχουν πολλές λειτουργικές μονάδες, κάθε μία με το δικό της τοπικό τίτλο, δηλώσεις, και εξισώσεις. Σημειώστε ότι οι λέξεις-κλειδιά όπως "`module`" δεν έχουν διάκριση πεζών - κεφαλαίων.

`title`

- Η δήλωση `title` καθορίζει ένα αλφαριθμητικό τίτλου το οποίο θα εισαχθεί στα αρχεία τεκμηρίωσης τα οποία δημιουργούνται από το μεταγλωττιστή.

αλφαριθμητικό

- Το *αλφαριθμητικό* είναι μια ακολουθία χαρακτήρων η οποία περικλείεται από μονά εισαγωγικά.

ΝΟΜΙΚΗ ΣΗΜΕΙΩΣΗ Η ονομασία ABEL (Advanced Boolean Equation Language) είναι εμπορικά κατοχυρωμένη από την Data I/O Corporation (Redmond, WA 98073, Η.Π.Α.).

Πίνακας 4-10 Τυπική δομή προγράμματος ABEL.

```
module  όνομα-μονάδας
title αλφαριθμητικό
αναγνωριστικό-συσκευής device τύπος-συσκευής;
δηλώσεις-ακροδεκτών
λοιπές-δηλώσεις
equations
εξισώσεις
test_vectors
διανύσματα-ελέγχου
end όνομα-μονάδας
```

- Η προαιρετική δήλωση `device` περιλαμβάνει ένα αναγνωριστικό συσκευής (ALARMCKT) και ένα αλφαριθμητικό που υποδηλώνει τον τύπο της συσκευής ('P16V8C' για το GAL16V8). Ο μεταγλωττιστής χρησιμοποιεί το αναγνωριστικό συσκευής στα ονόματα των αρχείων τεκμηρίωσης τα οποία δημιουργεί και χρησιμοποιεί, ενώ χρησιμοποιεί τον τύπο της συσκευής για να προσδιορίσει αν μπορεί πράγματι η συσκευή να εκτελέσει τις λογικές συναρτήσεις που καθορίζονται στο πρόγραμμα. *device*
- Τα *σχόλια* ξεκινούν με ένα διπλό εισαγωγικό και τελειώνουν με ένα δεύτερο διπλό εισαγωγικό ή με το τέλος της γραμμής, όποιο από τα δύο εμφανιστεί πρώτο. *σχόλιο*
- Οι *δηλώσεις ακροδεκτών* πληροφορούν το μεταγλωττιστή για τα συμβολικά ονόματα που σχετίζονται με τους εξωτερικούς ακροδέκτες της συσκευής. Αν προηγείται το πρόθεμα NOT (!) πριν από το όνομα του *δηλώσεις ακροδεκτών*

Πίνακας 4-11 Πρόγραμμα ABEL για το κύκλωμα συναγερμού της Εικόνας 4-11.

```
module Alarm_Circuit
title 'Alarm Circuit Example
J. Wakerly, Micro Systems Engineering'
ALARMCKT device 'P16V8C';

" Input pins
PANIC, ENABLEA, EXITING          pin 1, 2, 3;
WINDOW, DOOR, GARAGE             pin 4, 5, 6;
" Output pins
ALARM                            pin 11 istype 'com';

" Constant definition
X = .X.;

" Intermediate equation
SECURE = WINDOW & DOOR & GARAGE;

equations
ALARM = PANIC # ENABLEA & !EXITING & !SECURE;

test_vectors
([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE]-> [ALARM])
[     1,     .X.,     .X.,    .X.,  .X.,  .X.] -> [    1];
[     0,       0,     .X.,     X.,  .X.,  .X.] -> [    0];
[     0,       1,       1,    .X.,  .X.,  .X.] -> [    0];
[     0,       1,       0,      0,   X.,  .X.] -> [    1];
[     0,       1,       0,    .X.,    0,  .X.] -> [    1];
[     0,       1,       0,    .X.,  .X.,    0]  ->[    1];
[     0,       1,       0,      1,    1,    1]  ->[    0];

end Alarm_Circuit
```

σήματος, τότε στον ακροδέκτη θα εμφανίζεται το συμπλήρωμα του κατονομαζόμενου σήματος. Οι δηλώσεις ακροδεκτών είναι δυνατόν να περιέχουν ή να μην περιέχουν αριθμούς ακροδεκτών. Αν δε δίνεται κανένας αριθμός, ο μεταγλωττιστής τούς αποδίδει με βάση τις δυνατότητες της συσκευής-στόχου.

com
istype

- Η λέξη-κλειδί `istype` προηγείται μιας λίστας με μία ή περισσότερες ιδιότητες, που διαχωρίζονται μεταξύ τους με κόμματα, και υποδεικνύει στο μεταγλωττιστή τον τύπο του σήματος εξόδου. Η λέξη-κλειδί "`com`" υποδηλώνει συνδυαστική έξοδο. Αν δε χρησιμοποιηθεί η λέξη-κλειδί "`istype`", ο μεταγλωττιστής γενικά θεωρεί δεδομένο ότι το σήμα είναι είσοδος – εκτός κι αν εμφανίζεται στην αριστερή πλευρά μιας εξίσωσης, οπότε επιχειρεί να κατανοήσει τις ιδιότητες εξόδου από τα συμφραζόμενα. Για την προστασία σας, είναι προτιμότερο να χρησιμοποιείτε απλώς τη λέξη-κλειδί `istype` για όλες τις εξόδους!

λοιπές δηλώσεις

- Οι *λοιπές δηλώσεις* επιτρέπουν στο σχεδιαστή να ορίζει σταθερές και παραστάσεις που βελτιώνουν την αναγνωσιμότητα και απλότητα της λογικής σχεδίασης.

equations

- Η εντολή *equations* δείχνει ότι ακολουθούν λογικές εξισώσεις που ορίζουν τα σήματα εξόδου ως συναρτήσεις των σημάτων εξόδου.

εξισώσεις

- Οι *εξισώσεις* γράφονται ως εντολές ανάθεσης τιμής σε μια συμβατική γλώσσα προγραμματισμού. Κάθε εξίσωση τερματίζεται με ελληνικό ερωτηματικό. Η ABEL χρησιμοποιεί τα παρακάτω σύμβολα για τις λογικές πράξεις:

& (AND) & AND.
 # (OR) # OR.
! (NOT) ! NOT (χρησιμοποιείται ως πρόθεμα).
$ (XOR) $ XOR.
!$ (XNOR) ! $ XNOR.

Όπως συμβαίνει και στις συμβατικές γλώσσες προγραμματισμού, το AND (&) έχει προτεραιότητα έναντι του OR (#) στις παραστάσεις. Η οδηγία *@ALTERNATE* κάνει το μεταγλωττιστή να αναγνωρίζει ένα εναλλακτικό σύνολο συμβόλων για τις εξής πράξεις: *, +, /, :+: και :*: αντίστοιχα. Αυτό το βιβλίο χρησιμοποιεί παντού τα προεπιλεγμένα σύμβολα.

@ALTERNATE

test_vectors

- Η προαιρετική δήλωση *test_vectors* δείχνει ότι ακολουθούν διανύσματα ελέγχου.

διανύσματα ελέγχου

- Τα *διανύσματα ελέγχου* συσχετίζουν συνδυασμούς εισόδων με αναμενόμενες τιμές εξόδου. Χρησιμοποιούνται για προσομοίωση και έλεγχο, όπως εξηγείται στην Ενότητα 4.6.7.

.X.

- Ο μεταγλωττιστής αναγνωρίζει μερικές ειδικές σταθερές, συμπεριλαμβανομένης της *.X.*, δηλαδή ένα απλό bit του οποίου η τιμή είναι "αδιάφορη".

end

- Η εντολή *end* υποδεικνύει το τέλος της λειτουργικής μονάδας.

Οι εξισώσεις συνδυαστικών εξόδων χρησιμοποιούν το *μη χρονισμένο τελεστή ανάθεσης τιμής* =. Η αριστερή πλευρά της εξίσωσης κανονικά περιέχει ένα όνομα σήματος. Η δεξιά πλευρά είναι μια λογική παράσταση, όχι απαραίτητα σε μορφή αθροίσματος γινομένων. Στην αριστερή πλευρά της εξίσωσης, το όνομα του σήματος είναι δυνατόν να έπεται του τελεστή NOT, του !. Αυτό είναι ισοδύναμο με το συμπλήρωμα της δεξιάς πλευράς. Η δουλειά του μεταγλωττιστή είναι να δημιουργήσει ένα υπόδειγμα ασφαλειών τέτοιο ώστε το σήμα που προσδιορίζεται στην αριστερή πλευρά να υλοποιεί τη λογική παράσταση της δεξιάς πλευράς.

μη χρονισμένος τελεστής ανάθεσης τιμής =

4.6.2 Λειτουργία του μεταγλωττιστή ABEL

Το πρόγραμμα του Πίνακα 4-11 υλοποιεί τη λειτουργία συναγερμού που περιγράψαμε στην Ενότητα 4.3.1. Το σήμα με το όνομα ENABLE (έγκριση) έχει κωδικοποιηθεί ως ENABLEA επειδή η λέξη ENABLE είναι δεσμευμένη στην ABEL.

Παρατηρήστε ότι δεν εμφανίζονται όλες οι εξισώσεις κάτω από την εντολή `equations`. Νωρίτερα εμφανίζεται μια ενδιάμεση εξίσωση για το αναγνωριστικό SECURE (ασφάλεια). Αυτή η εξίσωση είναι απλώς ένας ορισμός ο οποίος συσχετίζει μια παράσταση με το αναγνωριστικό SECURE. Ο μεταγλωττιστής ABEL αντικαθιστά την παράσταση του αναγνωριστικού SECURE σε κάθε σημείο όπου εμφανίζεται το SECURE μετά από τον ορισμό του.

ενδιάμεση εξίσωση

Στην Εικόνα 4-19 της Ενότητας 4.3.1 υλοποιήσαμε το κύκλωμα συναγερμού κατευθείαν από τις παραστάσεις SECURE (ασφάλεια) και ALARM (συναγερμός), χρησιμοποιώντας πολλά επίπεδα λογικής. Ο μεταγλωττιστής ABEL δε χρησιμοποιεί παραστάσεις για τη διασύνδεση πυλών με αυτόν τον τρόπο. Αντίθετα, επεξεργάζεται τις παραστάσεις για να προκύψει ένα ελάχιστο αποτέλεσμα αθροίσματος γινομένων δύο επιπέδων, κατάλληλο για υλοποίηση σε μια PLD. Έτσι, με τη μεταγλώττιση του Πίνακα 4-11 θα προκύψει ένα αποτέλεσμα ισοδύναμο με το κύκλωμα AND-OR που φαίνεται στην Εικόνα 4-20 της Ενότητας 4.3.1, τοοποίο συμβαίνει να είναι ελάχιστο.

Και πράγματι, αυτό συμβαίνει. Ο Πίνακας 4-12 δείχνει το αρχείο των εξισώσεων που συνέθεσε ο μεταγλωττιστής ABEL. Παρατηρήστε ότι ο μεταγλωττιστής δημιουργεί εξισώσεις μόνο για το σήμα ALARM, τη μοναδική έξοδο. Το σήμα SECURE δεν εμφανίζεται πουθενά.

Ο μεταγλωττιστής βρίσκει μια ελάχιστη παράσταση αθροίσματος γινομένων για το ALARM και το συμπλήρωμά του, !ALARM. Όπως αναφέραμε προηγουμένως, πολλές διατάξεις PLD έχουν τη δυνατότητα να αναστρέφουν ή να μην αναστρέφουν την έξοδο AND-OR που διαθέτουν. Η "εξίσωση ανάστροφης πολικότητας" του Πίνακα 4-12 είναι μια υλοποίηση αθροίσματος γινομένων του σήματος !ALARM και θα μπορούσε να χρησιμοποιηθεί αν είχε επιλεγεί αντιστροφή εξόδου.

Σε αυτό το παράδειγμα, η εξίσωση ανάστροφης πολικότητας έχει έναν όρο γινομένου λιγότερο από την εξίσωση κανονικής πολικότητας του

ALARM. Έτσι, ο μεταγλωττιστής θα επιλέξει αυτή την εξίσωση αν η συσκευή προορισμού έχει επιλέξιμη αντιστροφή εξόδου. Ο χρήστης μπορεί επίσης να εξαναγκάσει το μεταγλωττιστή να χρησιμοποιεί είτε κανονική είτε ανάστροφη πολικότητα για ένα σήμα συμπεριλαμβάνοντας τη λέξη-κλειδί "buffer" ή "invert", αντίστοιχα, στη λίστα ιδιοτήτων istype του σήματος. (Σε μερικούς μεταγλωττιστές ABEL, είναι δυνατή η χρήση των λέξεων-κλειδιών "pos" και "neg" γι' αυτόν το σκοπό, δείτε όμως και την Ενότητα 4.6.6.)

Πίνακας 4-12 Αρχείο εξισώσεων που συνετέθησαν από την ABEL για το πρόγραμμα του Πίνακα 4-11.

```
ABEL 6.30

Design alarmckt created Tue Nov 24 1998

Title: Alarm Circuit Example
Title: J. Wakerly, Micro Systems Engineering

P-Terms   Fan-in Fan-out Type  Name (attributes)
--------- ------ ------- ---- ----------------
  4/3          6       1  Pin      ALARM
=========
  4/3        Best P-Term Total:  3
                Total Pins:  7
               Total Nodes:  0
        Average P-Term/Output:  3

Equations:

ALARM = (ENABLEA & !EXITING & !DOOR
       # ENABLEA & !EXITING & !WINDOW
       # ENABLEA & !EXITING & !GARAGE
       # PANIC);

Reverse-Polarity Equations:

!ALARM = (!PANIC & WINDOW & DOOR & GARAGE
        # !PANIC & EXITING
        # !PANIC & !ENABLEA);
```

4.6.3 Εντολές WHEN και ομάδες εξισώσεων

εντολή WHEN

Εκτός από εξισώσεις, η ABEL παρέχει επιπλέον την *εντολή* ***WHEN*** ως εναλλακτικό τρόπο για τον καθορισμό συνδυαστικών λογικών συναρτήσεων στην ενότητα *equations* ενός προγράμματος ABEL. Ο Πίνακας 4-13 δείχνει τη γενική δομή της εντολής WHEN, η οποία είναι παρόμοια με την εντολή IF σε μια συμβατική γλώσσα προγραμματισμού. Ο όρος

ELSE είναι προαιρετικός. Στο παράδειγμα, η *λογική-παράσταση* είναι μια παράσταση η οποία έχει αποτέλεσμα μια τιμή αληθές (1) ή ψευδές (0). "Εκτελείται" είτε η *εξίσωση- αληθούς* είτε η *εξίσωση-ψευδούς*, ανάλογα με την τιμή της *λογικής-παράστασης*. Αλλά χρειάζεται να είμαστε λίγο πιο ακριβείς σχετικά με το τι εννοούμε με τον όρο "εκτελείται", όπως περιγράφεται πιο κάτω.

```
WHEN λογική-παράσταση THEN
   εξίσωση-αληθούς;
ELSE
   εξίσωση-ψευδούς;
```

Πίνακας 4-13 Δομή εντολής WHEN στην ABEL.

Στην απλούστερη περίπτωση, η *εξίσωση-αληθούς* και η προαιρετική *εξίσωση-ψευδούς* είναι εντολές ανάθεσης τιμής, όπως στις δύο πρώτες προτάσεις WHEN του Πίνακα 4-14 (για X1 και X2). Στην περίπτωση αυτή, η *λογική-παράσταση* υποβάλλεται σε λογικό AND με τη δεξιά πλευρά της *εξίσωσης-αληθούς*, ενώ το συμπλήρωμα της *λογικής-παράστασης* υποβάλλεται σε λογικό AND με τη δεξιά πλευρά της *εξίσωσης-ψευδούς*. Έτσι, οι εξισώσεις των X1A και X2A παράγουν τα ίδια αποτελέσματα όπως οι αντίστοιχες προτάσεις WHEN χωρίς να χρησιμοποιούν την εντολή WHEN.

Παρατηρήστε στο πρώτο παράδειγμα ότι το X1 εμφανίζεται στην *εξίσωση-αληθούς*, αλλά δεν υπάρχει *εξίσωση-ψευδούς*. Τι συμβαίνει λοιπόν στο X1 όταν η *λογική-παράσταση* (!A#B) είναι ψευδής; Ίσως να σκεφτείτε ότι η τιμή του X1 θα είναι "αδιάφορη", γι' αυτούς τους συνδυασμούς εισόδων – αλλά δεν είναι έτσι, όπως εξηγείται παρακάτω.

Τυπικά, ο μη χρονισμένος τελεστής ανάθεσης τιμής = καθορίζει τους συνδυασμούς εισόδων που πρέπει να προστεθούν στο σύνολο ενεργοποίησης του σήματος εξόδου που εμφανίζεται στην αριστερή πλευρά της εξίσωσης. Το σύνολο ενεργοποίησης μιας εξόδου ξεκινά κενό και αυξάνεται κατά ένα κάθε φορά που εμφανίζεται η έξοδος στην αριστερή πλευρά μιας εξίσωσης. Με άλλα λόγια, οι δεξιές πλευρές όλων των εξισώσεων για την ίδια (μη συμπληρωματική) έξοδο υποβάλλονται σε πράξη OR μεταξύ τους. (Αν η έξοδος εμφανίζεται συμπληρωματική στην αριστερή πλευρά, η δεξιά πλευρά συμπληρώνεται πριν από την πράξη OR.) Έτσι λοιπόν, η τιμή του X1 είναι 1 μόνο για τους συνδυασμούς εισόδων για τους οποίους η *λογική-παράσταση* (!A#B) είναι αληθής και η δεξιά πλευρά της *εξίσωσης-αληθούς* (C&!D) είναι επίσης αληθής.

Στο δεύτερο παράδειγμα το X2 εμφανίζεται στη αριστερή πλευρά δύο εξισώσεων, άρα η ισοδύναμη εξίσωση που εμφανίζεται για το X2A λαμβάνεται εφαρμόζοντας την πράξη OR μεταξύ των δύο δεξιών πλευρών των εξισώσεων, αφού προηγουμένως εφαρμόσουμε την πράξη AND μεταξύ κάθε μίας από τις δεξιές πλευρές και της κατάλληλης συνθήκης.

Πίνακας 4-14 Παραδείγματα εντολών WHEN.

```
module WhenEx
title 'WHEN Statement Examples'

" Input pins
A, B, C, D, E, F                    pin;

" Output pins
X1, X1A, X2, X2A, X3, X3A, X4       pin istype 'com';
X5, X6, X7, X8, X9, X10             pin istype 'com';

equations

WHEN (!A # B) THEN X1 = C & !D;

X1A = (!A # B) & (C & !D);

WHEN (A & B) THEN X2 = C # D;
ELSE X2 = E # F;

X2A = (A & B) & (C # D)
    # !(A & B) & (E # F);

WHEN (A) THEN X3 = D;
ELSE WHEN (B) THEN X3 = E;
ELSE WHEN (C) THEN X3 = F;

X3A = (A) & (D)
    # !(A) & (B) & (E)
    # !(A) & !(B) & (C) & (F);

WHEN (A) THEN
      {WHEN (B) THEN X4 = D;}
ELSE X4 = E;

WHEN (A & B) THEN X5 = D;
ELSE WHEN (A # !C) THEN X6 = E;
ELSE WHEN (B # C) THEN X7 = F;

WHEN (A) THEN {
      X8 = D & E & F;
      WHEN (B) THEN X8 = 1; ELSE {X9 = D; X10 = E;}
} ELSE {
      X8 = !D # !E;
      WHEN (D) THEN X9 = 1;
      {X10 = C & D;}
}

end WhenEx
```

Πίνακας 4-15 Αρχείο εξισώσεων που συνέθεσε η ABEL για το πρόγραμμα του Πίνακα 4-14.

```
ABEL 6.30

Design whenex created Wed Dec `2 1998

Title: WHEN Statement Examples
P-Terms    Fan-in Fan-out Type Name
--------- ------ ------- ---- -----
   2/3     4       1      Pin  X1
   2/3     4       1      Pin  X1A
   6/3     6       1      Pin  X2
   6/3     6       1      Pin  X2A
   3/4     6       1      Pin  X3
   3/4     6       1      Pin  X3A
   2/3     4       1      Pin  X4
   1/3     3       1      Pin  X5
   2/3     4       1      Pin  X6
   1/3     3       1      Pin  X7
   4/4     5       1      Pin  X8
   2/2     3       1      Pin  X9
   2/4     5       1      Pin  X10
=========
  36/42     Best P-Term Total: 30
                  Total Pins: 19
                 Total Nodes: 0
        Average P-Term/Output: 2
```

```
Equations:

X1 = (C & !D & !A
   # C & !D & B);

X1A = (C & !D & !A
    # C & !D & B);

X2 = (D & A & B
   # C & A & B
   # !B & E
   # !A & E
   # !B & F
   # !A & F);

X2A = (D & A & B
    # C & A & B
    # !B & E
    # !A & E
    # !B & F
    # !A & F);

X3 = (C & !A & !B & F
   # !A & B & E
   # D & A);

X3A = (C & !A & !B & F
    # !A & B & E
    # D & A);

X4 = (D & A & B
   # !A & E);

X5 = (D & A & B);

X6 = (A & !B & E
   # !C & !A & E);

X7 = (C & !A & F);

X8 = (D & A & E & F
   # A & B
   # !A & !E
   # !D & !A);

X9 = (D & !A
   # D & !B);

X10 = (C & D & !A
    # A & !B & E);
```

```
Reverse-Polarity Eqns:

!X1 = (A & !B
    # D
    # !C);

!X1A = (A & !B
     # D
     # !C);

!X2 = (!C & !D & A & B
    # !B & !E & !F
    # !A & !E & !F);

!X2A = (!C & !D & A & B
     # !B & !E & !F
     # !A & !E & !F);

!X3 = (!C & !A & !B
    # !A & B & !E
    # !D & A
    # !A & !B & !F);

!X3A = (!C & !A & !B
     # !A & B & !E
     # !D & A
     # !A & !B & !F);

!X4 = (A & !B
    # !D & A
    # !A & !E);

!X5 = (!A
    # !D
    # !B);

!X6 = (A & B
    # C & !A
    # !E);

!X7 = (A
    # !C
    # !F);

!X8 = (A & !B & !F
    # D & !A & E
    # A & !B & !E
    # !D & A & !B);

!X9 = (!D
    # A & B);

!X10 = (A & B
     # !D & !A
     # !C & !A
     # A & !E);
```

Η *εξίσωση-αληθούς* και η προαιρετική *εξίσωση-ψευδούς* σε μια εντολή WHEN μπορεί να είναι οποιεσδήποτε εξισώσεις. Επιπλέον, οι εντολές WHEN μπορούν να είναι "ένθετες" με χρήση μιας άλλης εντολής WHEN ως *εξίσωσης-ψευδούς*. Όταν οι εντολές είναι ένθετες, όλες οι συνθήκες που οδηγούν σε μια "εκτελεσμένη" εντολή υποβάλλονται σε πράξη AND μεταξύ τους. Η εξίσωση για το X3 και η αντίστοιχή της χωρίς εντολές WHEN για το X3A στον Πίνακα 4-14 αποτελούν παραδείγματα της γενικής αρχής.

Η *εξίσωση-αληθούς* μπορεί να είναι μια άλλη εντολή WHEN αν περικλείεται από αγκύλες, όπως φαίνεται στο παράδειγμα X4 στον πίνακα. Αυτή είναι απλώς μια περίπτωση της γενικής χρήσης των αγκυλών που περιγράφεται στη συνέχεια.

Αν και σε όλα τα παραδείγματα WHEN γίνεται ανάθεση τιμών στην ίδια έξοδο στα πλαίσια κάθε τμήματος μιας δεδομένης εντολής WHEN, αυτό δεν είναι υποχρεωτικό. Η προτελευταία εντολή WHEN στον Πίνακα 4-14 αποτελεί ένα τέτοιο παράδειγμα.

Είναι συχνά χρήσιμο να κάνουμε περισσότερες από μία αναθέσεις τιμών στην *εξίσωση-αληθούς* ή στην *εξίσωση-ψευδούς* ή και στις δύο. Γι' αυτόν το σκοπό η ABEL υποστηρίζει ομάδες εξισώσεων οπουδήποτε υποστηρίζει μια απλή εξίσωση. *Ομάδα εξισώσεων* είναι απλώς μια ακολουθία εντολών που περικλείονται από αγκύλες, όπως φαίνεται στην τελευταία εντολή WHEN του πίνακα. Οι επιμέρους εντολές της ακολουθίας είναι δυνατόν να είναι εντολές ανάθεσης τιμής, ή εντολές WHEN, ή ένθετες ομάδες εξισώσεων. Δε χρησιμοποιείται ελληνικό ερωτηματικό μετά την αγκύλη κλεισίματος της ομάδας εξισώσεων. Στον Πίνακα 4-15 παρατίθενται οι εξισώσεις που παράγονται από το μεταγλωττιστή ABEL για ολόκληρο το πρόγραμμα του παραδείγματος.

`truth_table`	(*λίστα-εισόδων* -> *λίστα-εξόδων*) *τιμή-εισόδου* -> *τιμή-εξόδου*; ... *τιμή-εισόδου* -> *τιμή-εξόδου*;

Πίνακας 4-16 Δομή πίνακα αληθείας της ABEL

4.6.4 Πίνακες αληθείας

πίνακας αληθείας
`truth_table`
λίστα-εισόδων
λίστα-εξόδων

Η ABEL παρέχει έναν ακόμη τρόπο για να ορίζονται συνδυαστικές λογικές συναρτήσεις: τον *πίνακα αληθείας*, με τη γενική μορφή που φαίνεται στον Πίνακα 4-16. Η λέξη-κλειδί `truth_table` εισάγει έναν πίνακα αληθείας. Η *λίστα-εισόδων* και η *λίστα-εξόδων* δίνουν τα ονόματα των σημάτων εισόδου και τις εξόδους στις οποίες επιδρούν. Κάθε μία από αυτές τις λίστες είναι είτε ένα μοναδικό όνομα σήματος είτε ένα *σύνολο*. Τα σύνολα περιγράφονται πλήρως στην Ενότητα 4.6.5. Μετά από

την εισαγωγή του πίνακα αληθείας, ακολουθεί μια σειρά προτάσεων κάθε μία από τις οποίες καθορίζει μια τιμή εισόδου και μια απαιτούμενη τιμή εξόδου με τη βοήθεια του τελεστή "`->`". Για παράδειγμα, ο πίνακας αληθείας ενός αντιστροφέα είναι ο εξής:

μη χρονισμένος τελεστής πίνακα αληθείας, `->`

```
truth_table        (X  -> NOTX)
                    0  -> 1
                    1  -> 0
```

Η λίστα των τιμών εισόδου δε χρειάζεται να είναι πλήρης. Χρειάζεται να καθοριστεί μόνο το σύνολο ενεργοποίησης της συνάρτησης, εκτός αν είναι ενεργοποιημένη η επεξεργασία των αδιάφορων εισόδων (δείτε στην Ενότητα 4.6.6). Ο Πίνακας 4-17 δείχνει πώς μπορεί να καθοριστεί η συνάρτηση του ανιχνευτή πρώτων αριθμών που περιγράφεται στην Ενότητα 4.3.1 με τη βοήθεια ενός προγράμματος ABEL. Για ευκολία, το αναγνωριστικό `NUM` ορίζεται ως συνώνυμο του συνόλου των τεσσάρων bit εισόδου `[N3,N2,N1,N0]`, γεγονός που επιτρέπει να γραφτεί σε μια τιμή εισόδου των 4 bit ως δεκαδικός ακέραιος.

Τόσο οι πίνακες αληθείας όσο και οι εξισώσεις μπορούν να χρησιμοποιηθούν μέσα στο ίδιο πρόγραμμα ABEL. Η λέξη-κλειδί `equations` εισάγει μια ακολουθία εξισώσεων, ενώ η λέξη-κλειδί `truth_table` εισάγει έναν και μοναδικό πίνακα αληθείας.

Πίνακας 4-17 Πρόγραμμα ABEL για τον ανιχνευτή πρώτων αριθμών.

```
module PrimeDet
title '4-Bit Prime Number Detector'

" Input and output pins
N0, N1, N2, N3            pin;
F                         pin istype 'com';

" Definition
NUM = [N3,N2,N1,N0];

truth_table  (NUM -> F)
               1 -> 1;
               2 -> 1;
               3 -> 1;
               5 -> 1;
               7 -> 1;
              11 -> 1;
              13 -> 1;
end PrimeDet
```

4.6.5 Πεδία τιμών, σύνολα, και σχέσεις

Τα περισσότερα ψηφιακά συστήματα περιλαμβάνουν διαύλους, καταχωρητές, και άλλα κυκλώματα τα οποία χειρίζονται μια ομάδα από δύο ή περισσότερα σήματα με πανομοιότυπο τρόπο. Η ABEL παρέχει αρκετές

συντομεύσεις για να διευκολύνει τον ορισμό και τη χρήση τέτοιων σημάτων.

Η πρώτη συντόμευση είναι για την ονομασία όμοιων αριθμημένων σημάτων. Όπως φαίνεται στους ορισμούς των ακροδεκτών του Πίνακα 4-18, ένα *πεδίο τιμών* (*range*) ονομάτων σημάτων μπορεί να οριστεί με την αναγραφή του πρώτου και του τελευταίου ονόματος του πεδίου τιμών, χωρισμένα με "`..`". Για παράδειγμα, όταν γράφουμε "`N3..N0`" είναι σαν να γράφουμε "`N3,N2,N1,N0`". Παρατηρήστε στον πίνακα ότι το πεδίο τιμών μπορεί να είναι είτε σε αύξουσα είτε φθίνουσα σειρά.

πεδίο τιμών

Στη συνέχεια, χρειαζόμαστε μια διευκόλυνση για να γράφουμε πιο συμπαγείς εξισώσεις όταν μια ομάδα σημάτων αντιμετωπίζεται με πανομοιότυπο τρόπο, για τον περιορισμό της πιθανότητας σφαλμάτων και ασυνεπειών. Ένα *σύνολο* της ABEL είναι απλώς μια καθορισμένη συλλογή σημάτων τα οποία αντιμετωπίζονται ως ενιαία μονάδα. Όταν εφαρμόζεται μια λογική πράξη, όπως π.χ. AND, OR, ή όταν αποδίδεται τιμή σε ένα σύνολο, αυτό γίνεται σε κάθε στοιχείο του συνόλου.

σύνολο

Κάθε σύνολο ορίζεται στην αρχή του προγράμματος με το συσχετισμό ενός ονόματος συνόλου με μια λίστα των στοιχείων του συνόλου μέσα σε αγκύλες (π.χ. `N=[N3,N2,N1,N0]` στον Πίνακα 4-18). Στη λίστα των στοιχείων του συνόλου είναι δυνατόν να χρησιμοποιείται σημειογραφία συντομεύσεων (`YOUT=[Y1..Y4]`), αλλά τα ονόματα των στοιχείων δε χρειάζεται να είναι παρόμοια ή να έχουν οποιαδήποτε αντιστοιχία με το όνομα του συνόλου (`COMP=[EQ,GE]`). Τα στοιχεία του συνόλου είναι επίσης δυνατόν να είναι σταθερές (`GT=[0,1]`). Σε κάθε περίπτωση, ο αριθμός και η σειρά των στοιχείων σε ένα σύνολο είναι σημαντικά, όπως θα δούμε παρακάτω.

Οι περισσότεροι από τους τελεστές της ABEL μπορούν να εφαρμόζονται σε σύνολα. Όταν μια πράξη εφαρμόζεται σε δύο ή περισσότερα σύνολα, όλα τα σύνολα πρέπει να έχουν τον ίδιο αριθμό στοιχείων και η πράξη εφαρμόζεται ξεχωριστά στα στοιχεία των συνόλων που βρίσκονται σε αντίστοιχες θέσεις, ανεξάρτητα από τα ονόματα ή τους αριθμούς τους. Έτσι, η εξίσωση "`YOUT=N&M`" είναι ισοδύναμη με τις εξής τέσσερις εξισώσεις:

```
Y1 = N3 & M3;
Y2 = N2 & M2;
Y3 = N1 & M1;
Y4 = N0 & M0;
```

Όταν μια πράξη περιλαμβάνει μεταβλητές τόσο εντός όσο και εκτός του συνόλου, οι μεταβλητές εκτός συνόλου συνδυάζονται ξεχωριστά με τα στοιχεία του συνόλου σε κάθε θέση. Έτσι, η εξίσωση "`ZOUT=(SEL&N)#(!SEL&M)`" είναι ισοδύναμη με τέσσερις εξισώσεις της μορφής "`Zi=(SEL&Ni)#(!SEL&Mi)`" για `i` ίσο με 0 έως 3.

Ένα άλλο σημαντικό χαρακτηριστικό είναι η δυνατότητα της ABEL να μετατρέπει τις "σχέσεις" σε λογικές παραστάσεις. *Σχέση* είναι ένα ζεύγος τελεστέων οι οποίοι συνδυάζονται μεταξύ τους με έναν από τους

σχέση

σχεσιακούς τελεστές που παρουσιάζονται στον Πίνακα 4-19. Ο μεταγλωττιστής μετατρέπει μια σχέση σε μια λογική παράσταση η οποία είναι 1 αν και μόνο αν η σχέση είναι αληθής.

σχεσιακός τελεστής

Οι τελεστέοι μιας σχέσης αντιμετωπίζονται ως απρόσημοι ακέραιοι, ενώ οποιοσδήποτε από τους δύο τελεστέους μπορεί να είναι είτε ακέραιος είτε σύνολο. Αν ο τελεστέος είναι ένα σύνολο, θεωρείται ως απρόσημος δυαδικός ακέραιος με την πρώτη από αριστερά μεταβλητή να αναπαριστά το πλέον σημαντικό bit. Από προεπιλογή, στην ABEL οι αριθμοί θεωρούνται ότι έχουν βάση το 10. Οι δεκαεξαδικοί και οι δυαδικοί αριθμοί αναπαρίστανται με το πρόθεμα “^h” ή “^b”, αντίστοιχα, όπως φαίνεται στην τελευταία εξίσωση του Πίνακα 4-18.

Τα σύνολα και οι σχέσεις της ABEL επιτρέπουν να εκφράζονται μεγάλες λειτουργικές δυνατότητες μέσα σε πολύ λίγες γραμμές κώδικα. Για παράδειγμα, οι εξισώσεις του Πίνακα 4-18 δημιουργούν ελαχιστοποιημένες εξισώσεις με 69 όρους γινομένου, όπως φαίνεται στη σύνοψη του Πίνακα 4-20.

Πίνακας 4-18 Παραδείγματα πεδίων τιμών, συνόλων, και σχέσεων στην ABEL.

```
module SetOps
title 'Set Operation Examples'

" Input and output pins
N3..N0, M3..M0, SEL                          pin;
Y1..Y4, Z0..Z3, EQ, GE, GTR, LTH, UNLUCKY    pin istype 'com';

" Definitions
N      = [N3,N2,N1,N0];
M      = [M3,M2,M1,M0];
YOUT   = [Y1..Y4];
ZOUT   = [Z3..Z0];

COMP   = [EQ,GE];
GT     = [ 0, 1];
LT     = [ 0, 0];

equations

YOUT = N & M;
ZOUT = (SEL & N) # (!SEL & M);
EQ = (N == M);
GE = (N >= M);
GTR = (COMP == GT);
LTH = (COMP == LT);
UNLUCKY = (N == 13) # (M == ^hD) # ((N + M) == ^b1101);

end SetOps
```

*4.6.6 Αδιάφορες είσοδοι

Μερικές εκδόσεις του μεταγλωττιστή της ABEL έχουν μια περιορισμένη δυνατότητα να χειρίζονται αδιάφορες εισόδους. Όπως αναφέραμε προηγουμένως, οι εξισώσεις της ABEL καθορίζουν συνδυασμούς εισόδων οι οποίοι ανήκουν στο σύνολο ενεργοποίησης μιας λογικής συνάρτησης. Οι υπόλοιποι συνδυασμοί θεωρείται ότι ανήκουν στο σύνολο απενεργοποίησης. Αν, αντίθετα, κάποιοι συνδυασμοί εισόδων μπορούν να αποδοθούν στο σύνολο *d*, τότε το πρόγραμμα ενδέχεται να έχει τη δυνατότητα να

Πίνακας 4-19 Σχεσιακοί τελεστές της ABEL.

Σύμβολο	*Σχέση*
==	ίσο με
!=	όχι ίσο με
<	μικρότερο από
<=	μικρότερο ή ίσο με
>	μεγαλύτερο από
>=	μεγαλύτερο ή ίσο με

Πίνακας 4-20 Σύνοψη εξισώσεων που συνέθεσε η ABEL για το πρόγραμμα του Πίνακα 4-18.

```
 P-Terms     Fan-in    Fan-out    Type    Name (attributes)
---------    ------    -------    ----    -----------------
  1/2          2         1        Pin     Y1
  1/2          2         1        Pin     Y2
  1/2          2         1        Pin     Y3
  1/2          2         1        Pin     Y4
  2/2          3         1        Pin     Z0
  2/2          3         1        Pin     Z1
  2/2          3         1        Pin     Z2
  2/2          3         1        Pin     Z3
 16/8          8         1        Pin     EQ
23/15          8         1        Pin     GE
  1/2          2         1        Pin     GTR
  1/2          2         1        Pin     LTH
16/19          8         1        Pin     UNLUCKY
=========
 69/62              Best P-Term Total:   53
                           Total Pins:   22
                          Total Nodes:   0
                  Average P-Term/Output:   4
```

χρησιμοποιήσει αυτές τις αδιάφορες εισόδους για να κάνει καλύτερη δουλειά στην ελαχιστοποίηση.

Η γλώσσα ABEL ορίζει δύο μηχανισμούς για την απόδοση συνδυασμών εισόδων στο σύνολο *d*. Για να χρησιμοποιήσετε έναν από τους δύο αυτούς μηχανισμούς, πρέπει να συμπεριλάβετε την οδηγία `@DCSET` του μεταγλωττιστή στο πρόγραμμά σας ή να συμπεριλάβετε το "`dc`" στη λίστα ιδιοτήτων `istype` των εξόδων για τις οποίες θέλετε να λαμβάνονται υπόψη οι αδιάφορες είσοδοι.

@DCSET
dc

Ο πρώτος μηχανισμός είναι ο *μη χρονισμένος τελεστής ανάθεσης τιμής "αδιάφορο"*, `?=`. Αυτός ο τελεστής χρησιμοποιείται στις εξισώσεις αντί του "=", για να δείξει ότι οι συνδυασμοί εισόδου που συμφωνούν με τη δεξιά πλευρά θα πρέπει να μπουν στο σύνολο *d* αντί στο σύνολο ενεργοποίησης. Παρά το γεγονός ότι αυτός ο τελεστής αναφέρεται στην τεκμηρίωση του μεταγλωττιστή ABEL που χρησιμοποιώ προσωπικά, ατυχώς φαίνεται να μη λειτουργεί σωστά, γι' αυτό δεν πρόκειται να ασχοληθούμε περισσότερο μαζί του.

μη χρονισμένος τελεστής ανάθεσης τιμής "αδιάφορο", ?=

Πίνακας 4-21
Πρόγραμμα ABEL με τη χρήση αδιάφορων εισόδων.

```
module DontCare
title 'Dont Care Examples'
@DCSET

" Input and output pins
N3..N0, A, B                pin;
F, Y                        pin istype 'com';

NUM = [N3..N0];
X = .X.;

truth_table   (NUM->F)
                0->0;
                1->1;
                2->1;
                3->1;
                4->0;
                5->1;
                6->0;
                7->1;
                8->0;
                9->0;

truth_table ([A,B]->Y)
              [0,0]->0;
              [0,1]->X;
              [1,0]->X;
              [1,1]->1;

end DontCare
```

Ο δεύτερος μηχανισμός είναι ο πίνακας αληθείας. Όταν είναι ενεργοποιημένη η επεξεργασία των αδιάφορων εισόδων, κάθε συνδυασμός εισόδων που δεν αναφέρεται ρητά στον πίνακα αληθείας μπαίνει στο σύνολο *d*. Έτσι, ο ανιχνευτής πρώτων αριθμών ψηφίου BCD που περιγράφεται στην Ενότητα 4.3.7 μπορεί να οριστεί με την ABEL όπως φαίνεται στον Πίνακα 4-21. Για τους συνδυασμούς εισόδων 10-15 υπονοείται η τιμή "αδιάφορο", επειδή αυτοί οι συνδυασμοί δεν εμφανίζονται στον πίνακα αληθείας και είναι ενεργός η οδηγία `@DCSET`.

Μπορούμε επίσης να περιγράψουμε ρητά αδιάφορους συνδυασμούς, όπως φαίνεται στο δεύτερο πίνακα αληθείας. Όπως παρουσιάσαμε στην αρχή αυτής της ενότητας, η ABEL αναγνωρίζει το `.X.` ως ειδική σταθερά του 1 bit της οποίας η τιμή είναι "αδιάφορο". Στον Πίνακα 4-21, το αναγνωριστικό "X" έχει εξισωθεί με τη σταθερά αυτή απλώς για να κάνει ευκολότερη την πληκτρολόγηση των τιμών "αδιάφορο" στον πίνακα αληθείας. Οι ελαχιστοποιημένες εξισώσεις που προκύπτουν από τον Πίνακα 4-21 εμφανίζονται στον Πίνακα 4-22. Σημειώστε ότι οι δύο εξισώσεις για την F δεν είναι ίσες. Ο μεταγλωττιστής έχει επιλέξει διαφορετικές τιμές για τα "αδιάφορα".


```
Equations:
F = (!N2 & N1
     # !N3 & N0);
Y = (B);

Reverse-Polarity Equations:
!F = (N2 & !N0
      # N3
      # !N1 & !N0);
!Y = (!B);
```

Πίνακας 4-22 Ελαχιστοποιημένες εξισώσεις που προκύπτουν από τον Πίνακα 4-21.

Πίνακας 4-23 Δομή των διανυσμάτων ελέγχου της ABEL.

`test_vectors`	(*λίστα-εισόδων* `->` *λίστα-εξόδων*)
	τιμή-εισόδου `->` *τιμή-εξόδου*;
	`...`
	τιμή-εισόδου `->` *τιμή-εξόδου*;

4.6.7 Διανύσματα έλεγχου

Τα προγράμματα της ABEL μπορούν να περιέχουν προαιρετικά διανύσματα ελέγχου, όπως δείξαμε στον Πίνακα 4-11 στην Ενότητα 4.6.1. Η γενική μορφή των διανυσμάτων ελέγχου μοιάζει πολύ με έναν πίνακα αληθείας, και φαίνεται στον Πίνακα 4-23. Η λέξη-κλειδί `test_vectors` εισάγει έναν πίνακα αληθείας. Η *λίστα-εισόδων* και η

test_vectors
λίστα-εισόδων

λίστα-εξόδων δίνουν τα ονόματα των σημάτων εισόδου και τις εξόδους στις οποίες αυτά επιδρούν. Κάθε μία από αυτές τις λίστες είναι είτε ένα μοναδικό όνομα σήματος, ή ένα σύνολο. Μετά από την εισαγωγή του διανύσματος ελέγχου, ακολουθεί μια σειρά προτάσεων κάθε μία από τις οποίες καθορίζει μια τιμή εισόδου και μια αναμενόμενη τιμή εξόδου με τη βοήθεια του τελεστή "`->`". *λίστα-εξόδων*

Τα διανύσματα ελέγχου της ABEL έχουν δύο κύριες χρήσεις και σκοπούς:

1. Μόλις ο μεταγλωττιστής ABEL μεταφράσει το πρόγραμμα σε "υπόδειγμα ασφαλειών" για μια συγκεκριμένη συσκευή, προσομοιώνει τη λειτουργία της τελικής προγραμματιζόμενης διάταξης εφαρμόζοντας τις εισόδους του διανύσματος ελέγχου σε ένα μοντέλο λογισμικού της συσκευής και συγκρίνοντας τις εξόδους της με τις αντίστοιχες εξόδους του διανύσματος ελέγχου. Ο σχεδιαστής μπορεί να καθορίσει μια ακολουθία διανυσμάτων ελέγχου για να διασφαλίσει ότι η συσκευή θα συμπεριφέρεται κατά τα αναμενόμενα για μερικούς ή όλους τους συνδυασμούς εισόδων.
2. Αφού μια διάταξη PLD προγραμματιστεί φυσικά, η μονάδα προγραμματισμού εφαρμόζει τις εισόδους του διανύσματος ελέγχου στη φυσική συσκευή και συγκρίνει τις εξόδους της συσκευής με τις αντίστοιχες εξόδους του διανύσματος ελέγχου. Αυτό γίνεται για να ελεγχθεί αν ο προγραμματισμός και η λειτουργία της συσκευής είναι σωστά.

Δυστυχώς, τα διανύσματα ελέγχου της ABEL σπάνια κάνουν πολύ καλή δουλειά σε αυτές τις εργασίες, όπως θα εξηγήσουμε παρακάτω.

Τα διανύσματα ελέγχου του Πίνακα 4-11 επαναλαμβάνονται στον Πίνακα 4-24, με τη διαφορά ότι για λόγους αναγνωσιμότητας έχουμε χρησιμοποιήσει το αναγνωριστικό `X` το οποίο έχει εξισωθεί με τη σταθερά αδιάφορης τιμής `.X.` και έχουμε προσθέσει σχόλια για την αρίθμηση των διανυσμάτων ελέγχου.

Πράγματι, ο Πίνακας 4-24 εμφανίζεται να είναι ένα πολύ καλό σύνολο διανυσμάτων ελέγχου. Από την άποψη του σχεδιαστή, αυτά τα διανύσματα καλύπτουν πλήρως την αναμενόμενη λειτουργία του κυκλώματος συναγερμού όπως αναλύεται διάνυσμα-προς-διάνυσμα παρακάτω:

1. Αν το σήμα `PANIC` (πανικός) είναι 1, τότε η έξοδος συναγερμού (`F`) πρέπει να ενεργοποιείται ανεξάρτητα από τις τιμές των λοιπών εισόδων. Τα υπόλοιπα διανύσματα καλύπτουν τις περιπτώσεις όπου το `PANIC` είναι 0.
2. Αν ο συναγερμός δεν είναι ενεργοποιημένος, τότε η έξοδος πρέπει να είναι απενεργοποιημένη.
3. Αν ο συναγερμός είναι ενεργοποιημένος αλλά εκείνη τη στιγμή βγαίνουμε από το σπίτι, τότε η έξοδος πρέπει να είναι απενεργοποιημένη.

4-6. Αν ο συναγερμός είναι ενεργοποιημένος και εκείνη τη στιγμή δε βγαίνουμε από το σπίτι, τότε η έξοδος πρέπει να ενεργοποιείται

όταν οποιοδήποτε από τα σήματα των αισθητήρων WINDOW (παράθυρο), DOOR, (πόρτα) ή GARAGE (γκαράζ) είναι 0.

7. Αν ο συναγερμός είναι ενεργοποιημένος, εκείνη τη στιγμή δε βγαίνουμε από το σπίτι, και όλα τα σήματα των αισθητήρων είναι 1, τότε η έξοδος πρέπει να είναι απενεργοποιημένη.

Το πρόβλημα είναι ότι η ABEL δε χειρίζεται τις αδιάφορες τιμές των εισόδων διανυσμάτων ελέγχου με τον τρόπο που θα έπρεπε. Για παράδειγμα, το διάνυσμα ελέγχου 1 θα έπρεπε να ελέγχει 32 διακριτούς συνδυασμούς εισόδων που αντιστοιχούν και στους 32 δυνατούς συνδυασμούς αδιάφορων εισόδων ENABLEA, EXITING, DOOR, και GARAGE. Αλλά δεν το κάνει. Σε αυτή την περίπτωση, ο μεταγλωττιστής ABEL ερμηνεύει την τιμή "αδιάφορο" ως "ο χρήστης αδιαφορεί για την τιμή εισόδου που χρησιμοποιώ", και απλώς αποδίδει την τιμή 0 σε όλες τις αδιάφορες εισόδους ενός διανύσματος ελέγχου. Σε αυτό το παράδειγμα, αν είχατε γράψει εσφαλμένα την εξίσωση εξόδου ως "F = PANIC & !ENABLEA # ENABLEA &...", τα διανύσματα ελέγχου θα εξακολουθούσαν να δίνουν επιτυχές αποτέλεσμα, παρά το γεγονός ότι το κουμπί πανικού θα λειτουργούσε μόνο όταν το σύστημα ήταν απενεργοποιημένο.

μοντέλο αστοχίας κολλήματος σε μία τιμή

Η δεύτερη χρήση των διανυσμάτων ελέγχου είναι στον έλεγχο της φυσικής συσκευής. Τα περισσότερα φυσικά ελαττώματα στις λογικές διατάξεις είναι δυνατόν να εντοπιστούν με χρήση του *μοντέλου αστοχίας κολλήματος σε μία τιμή* (*single stuck-at fault model*), στο οποίο γίνεται η παραδοχή ότι ένα οποιοδήποτε φυσικό ελάττωμα είναι ισοδύναμο με το να έχουμε την είσοδο ή την έξοδο μίας και μοναδικής πύλης "κολλημένη" σε μια λογική τιμή 0 ή 1. Με την απλή σύνθεση ενός συνόλου διανυσμάτων ελέγχου τα οποία φαίνεται πως εξετάζουν τις λειτουργικές προδιαγραφές ενός κυκλώματος, όπως κάναμε στον Πίνακα 4-24, δε διασφαλίζεται η δυνατότητα εντοπισμού όλων των αστοχιών κολλήματος σε μία τιμή. Τα διανύσματα ελέγχου πρέπει να επιλεγούν έτσι ώστε κάθε δυνατή αστοχία κολλήματος σε μία τιμή να προκαλεί εσφαλμένη τιμή στην έξοδο του κυκλώματος για κάποιους συνδυασμούς εισόδων διανυσμάτων ελέγχου.

Ο Πίνακας 4-25 δείχνει ένα πλήρες σύνολο διανυσμάτων ελέγχου για το κύκλωμα του συναγερμού, όταν αυτό υλοποιείται ως κύκλωμα αθροί-

Πίνακας 4-24 Διανύσματα ελέγχου για το πρόγραμμα του κυκλώματος συναγερμού του Πίνακα 4-11.

```
test_vectors
(PANIC,  ENABLEA, EXITING,  WINDOW, DOOR,  GARAGE]->    [ALARM])
[    1,       X,       X,       X,    X,       X]->    [    1]; "1
[    0,       0,       X,       X,    X,       X]->    [    0]; "2
[    0,       1,       1,       X,    X,       X]->    [    0]; "3
[    0,       1,       0,       0,    X,       X]->    [    1]; "4
[    0,       1,       0,       X,    0,       X]->    [    1]; "5
[    0,       1,       0,       X,    X,       0]->    [    1]; "6
[    0,       1,       0,       1,    1,       1]->    [    0]; "7
```

```
test_vectors
(PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE]  -> [ALARM])
[    1,      0,      1,     1,    1,     1]  -> [    1]; "1
[    0,      1,      0,     0,    1,     1]  -> [    1]; "2
[    0,      1,      0,     1,    0,     1]  -> [    1]; "3
[    0,      1,      0,     1,    1,     0]  -> [    1]; "4
[    0,      0,      0,     0,    0,     0]  -> [    0]; "5
[    0,      1,      1,     0,    0,     0]  -> [    0]; "6
[    0,      1,      0,     1,    1,     1]  -> [    0]; "7
```

Πίνακας 4-25 Απλά διανύσματα ελέγχου αστοχιών κολλήματος σε μία τιμή για την ελάχιστη υλοποίηση αθροίσματος γινομένων του κυκλώματος συναγερμού.

σματος γινομένων δύο επιπέδων. Τα τέσσερα πρώτα διανύσματα ελέγχουν αστοχίες κολλήματος στην τιμή 1 στην πύλη OR, ενώ τα τρία τελευταία ελέγχουν για αστοχίες κολλήματος στην τιμή 0 στις πύλες AND. Αποδεικνύεται ότι επαρκούν για τον εντοπισμό όλων των αστοχιών κολλήματος σε μία τιμή. Αν γνωρίζετε σχετικά με τον έλεγχο αστοχιών, μπορείτε να δημιουργήσετε διανύσματα ελέγχου για μικρά κυκλώματα μόνοι σας (όπως έκανα εγώ σε αυτό το παράδειγμα), αλλά οι περισσότεροι σχεδιαστές χρησιμοποιούν αυτοματοποιημένα εργαλεία τρίτων κατασκευαστών για να δημιουργήσουν διανύσματα ελέγχου υψηλής ποιότητας για τις σχεδιάσεις PLD που πραγματοποιούν.

4.7 Γλώσσα περιγραφής υλικού VHDL

Στα μέσα της δεκαετίας του 1980, το Υπουργείο Άμυνας των Η.Π.Α. και το IEEE επιχορήγησαν την ανάπτυξη μια γλώσσας περιγραφής υλικού μεγάλων δυνατοτήτων που λέγεται *VHDL*. Η γλώσσα ξεκίνησε έχοντας, και εξακολουθεί να έχει, τα παρακάτω χαρακτηριστικά:

VHDL

- Οι σχεδιάσεις είναι δυνατόν να αναλύονται ιεραρχικά.
- Κάθε στοιχείο σχεδίασης έχει τόσο μια καλά ορισμένη διασύνδεση (για τη σύνδεσή του με άλλα στοιχεία), όσο και μια επακριβή προδιαγραφή συμπεριφοράς (για την προσομοίωση της λειτουργίας του).
- Οι προδιαγραφές συμπεριφοράς είναι δυνατόν να χρησιμοποιούν είτε έναν αλγόριθμο είτε μια πραγματική δομή υλικού για τον καθορισμό της λειτουργίας ενός στοιχείου. Για παράδειγμα, ένα στοιχείο είναι δυνατόν να καθοριστεί αρχικά από έναν αλγόριθμο, για να επιτρέψει την επαλήθευση της σχεδίασης των στοιχείων υψηλότερου επιπέδου που το χρησιμοποιούν, ενώ αργότερα ο καθορισμός μέσω αλγόριθμου είναι δυνατόν να αντικατασταθεί από μια δομή υλικού.
- Ο ταυτοχρονισμός και ο χρονισμός είναι δυνατόν να μοντελοποιηθούν. Η VHDL χειρίζεται τόσο ασύγχρονες όσο και σύγχρονες δομές ακολουθιακών κυκλωμάτων.
- Η λογική λειτουργία και η συμπεριφορά του χρονισμού μιας σχεδίασης είναι δυνατόν να προσομοιωθούν.

Έτσι, η VHDL ξεκίνησε ως γλώσσα τεκμηρίωσης και μοντελοποίησης, που επιτρέπει τον επακριβή καθορισμό και προσομοίωση της συμπεριφοράς των σχεδιάσεων ψηφιακών συστημάτων.

εργαλεία σύνθεσης VHDL

Αν και η γλώσσα VHDL και το περιβάλλον προσομοίωσης ήταν από μόνα τους σημαντικές καινοτομίες, η χρησιμότητα και η δημοτικότητα της VHDL έκανε ένα σημαντικό άλμα με την εμπορική ανάπτυξη των *εργαλείων σύνθεσης VHDL*. Αυτά τα προγράμματα μπορούν να δημιουργούν δομές λογικών κυκλωμάτων κατευθείαν από τις περιγραφές συμπεριφοράς σε VHDL. Χρησιμοποιώντας τη VHDL μπορείτε να σχεδιάζετε, να προσομοιώνετε και να συνθέτετε τα πάντα, από ένα απλό συνδυαστικό κύκλωμα μέχρι ένα πλήρες σύστημα μικροεπεξεργαστή σε ένα τσιπ.

VHDL-87
VHDL-93

Η VHDL τυποποιήθηκε από το IEEE το 1987 (*VHDL-87*) και επεκτάθηκε το 1993 (*VHDL-93*). Σε αυτή την ενότητα θα περιγράψουμε ένα υποσύνολο των χαρακτηριστικών της γλώσσας τα οποία είναι έγκυρα και στα δύο πρότυπα. Στην Ενότητα 7.12 θα περιγράψουμε τα πρόσθετα χαρακτηριστικά της ακολουθιακής λογικής σχεδίασης.

4.7.1 Σχεδιαστική ροή

σχεδιαστική ροή

Είναι χρήσιμο να κατανοήσουμε το συνολικό περιβάλλον σχεδίασης της γλώσσας VHDL προτού πάμε στην ίδια τη γλώσσα. Μια διαδικασία σχεδίασης με τη VHDL έχει πολλά βήματα, τα οποία λέγονται συνήθως *σχεδιαστική ροή*. Τα βήματα αυτά έχουν εφαρμογή σε οποιαδήποτε διαδικασία σχεδίασης βασίζεται σε μια γλώσσα περιγραφής υλικού (HDL), και παρουσιάζονται συνοπτικά στην Εικόνα 4-50, σε αυτή την ενότητα.

Η λεγόμενη "εμπροσθοφυλακή" ("front-end") της σχεδίασης ξεκινά με τον προσδιορισμό της βασικής προσέγγισης και των δομικών μονάδων σε επίπεδο δομικού διαγράμματος. Οι μεγάλες λογικές σχεδιάσεις, όπως τα προγράμματα λογισμικού, έχουν συνήθως ιεραρχική δομή, και η VHDL παρέχει ένα καλό πλαίσιο εργασίας για τον καθορισμό των λειτουργικών μονάδων και των διασυνδέσεών τους, ενώ τις λεπτομέρειες μπορείτε να τις συμπληρώσετε αργότερα.

Το επόμενο βήμα είναι η σύνταξη του κώδικα VHDL για τις λειτουργικές μονάδες, τις διασυνδέσεις, και τα εσωτερικά τους στοιχεία. Επειδή η VHDL είναι μια γλώσσα που βασίζεται σε κείμενο, μπορείτε να χρησιμοποιήσετε οποιονδήποτε επεξεργαστή κειμένου γι' αυτή τη δουλειά.

ΤΙ ΣΗΜΑΙΝΕΙ VHDL

Το "VHDL" είναι ακρώνυμο του "VHSIC Hardware Description Language" (Γλώσσα περιγραφής υλικού VHSIC). Το VHSIC, με τη σειρά του, είναι ακρώνυμο του "Very High Speed Integrated Circuit" (Ολοκληρωμένο κύκλωμα πολύ υψηλής ταχύτητας), το οποίο ήταν ένα πρόγραμμα του Υπουργείου Άμυνας των Η.Π.Α. για την ενθάρρυνση της έρευνας στην τεχνολογία ολοκληρωμένων κυκλωμάτων υψηλής απόδοσης (θα μπορούσαμε να αποδώσουμε το ακρώνυμο ως "Very Healthy Sums of Instant Cash", ως λογοπαίγνιο για τα μεγάλα ποσά που δαπανήθηκαν για το σκοπό αυτόν!).

VERILOG ΚΑΙ VHDL

Την ίδια περίπου περίοδο που αναπτύχθηκε η VHDL, εμφανίστηκε στο προσκήνιο μια διαφορετική γλώσσα σχεδίασης υλικού. Η *Verilog HDL*, ή απλώς *Verilog*, παρουσιάστηκε από την Gateway Design Automation το 1984 ως μια αποκλειστικής χρήσης γλώσσα περιγραφής υλικού και προσομοίωσης. Η μεταγενέστερη παρουσίαση των εργαλείων σύνθεσης σε Verilog το 1988 από τη νεότευκτη τότε εταιρία Synopsys, καθώς και η απόκτηση της Gateway από την Cadence Design Systems το 1989, ήταν ένας επιτυχημένος συνδυασμός που οδήγησε στην ευρεία διάδοση της χρήσης της γλώσσας.

Σήμερα η VHDL και η Verilog έχουν και οι δύο ευρύτατη χρήση και η κάθε μία τους καταλαμβάνει κατά προσέγγιση το ήμισυ της αγοράς της λογικής σύνθεσης. Η Verilog έχει τις συντακτικές της ρίζες στη γλώσσα C και είναι, σε μερικά χαρακτηριστικά της, μια γλώσσα πιο εύκολη στη χρήση και στην εκμάθηση, ενώ ή VHDL μοιάζει περισσότερο με την Ada (μια γλώσσα προγραμματισμού λογισμικού που επιχορηγήθηκε από το Υπουργείο Άμυνας των Η.Π.Α.) και έχει περισσότερα χαρακτηριστικά τα οποία υποστηρίζουν την ανάπτυξη μεγάλων έργων.

Συγκρίνοντας τα πλεονεκτήματα και τα μειονεκτήματα του να ξεκινήσει κανείς με τη μια ή την άλλη γλώσσα, οι David Pellerin και Douglas Taylor έθεσαν το ζήτημα με τον καλύτερο τρόπο στο βιβλίο τους *VHDL Made Easy!* (Prentice Hall, 1997):

> Και οι δύο γλώσσες είναι εύκολες στην εκμάθηση και δύσκολες στην τέλεια γνώση τους. Και από τη στιγμή που θα έχετε πλέον μάθει μία από τις γλώσσες αυτές, δε θα έχετε πρόβλημα να περάσετε στην άλλη.

Ωστόσο, τα περισσότερα σχεδιαστικά περιβάλλοντα περιλαμβάνουν έναν εξειδικευμένο *επεξεργαστή κειμένου VHDL* ο οποίος κάνει λίγο πιο εύκολη αυτή τη δουλειά. Τέτοιοι επεξεργαστές κειμένου περιλαμβάνουν χαρακτηριστικά όπως αυτόματη επισήμανση των λέξεων-κλειδιών της VHDL, αυτόματη δημιουργία εσοχών, προκαθορισμένα πρότυπα για δομές προγραμμάτων που χρησιμοποιούνται συχνά, καθώς και ενσωματωμένο συντακτικό έλεγχο και πρόσβαση με ένα πάτημα του ποντικιού σας στο μεταγλωττιστή.

επεξεργαστής κειμένου VHDL

Από τη στιγμή που θα έχετε γράψει αρκετό όγκο κώδικα, θα θέλετε φυσικά να τον μεταγλωττίσετε. Ο *μεταγλωττιστής VHDL* αναλύει τον κώδικά σας για τυχόν συντακτικά σφάλματα και επίσης τον ελέγχει ως προς τη συμβατότητά του με άλλες λειτουργικές μονάδες στις οποίες βασίζεται. Δημιουργεί επίσης τις εσωτερικές πληροφορίες οι οποίες χρειάζονται σε έναν προσομοιωτή που θα επεξεργαστεί τη σχεδίασή σας αργότερα. Όπως και σε άλλα προγραμματιστικά εγχειρήματα, σίγουρα δε θα περιμένετε μέχρι το τέλος της κωδικοποίησης για να μεταγλωττίσετε όλον τον κώδικα. Μεταγλωττίζοντας ένα τμήμα κάθε φορά μπορείτε να αποφύγετε τον πολλαπλασιασμό των συντακτικών σφαλμάτων, τις ασυνέπειες στα ονόματα, και άλλα προβλήματα, και βέβαια μπορείτε να έχετε καλύτερη αίσθηση της προοόδου όταν το έργο απέχει ακόμα πολύ από την ολοκλήρωσή του.

ΤΙ ΣΗΜΑΙΝΕΙ VERILOG

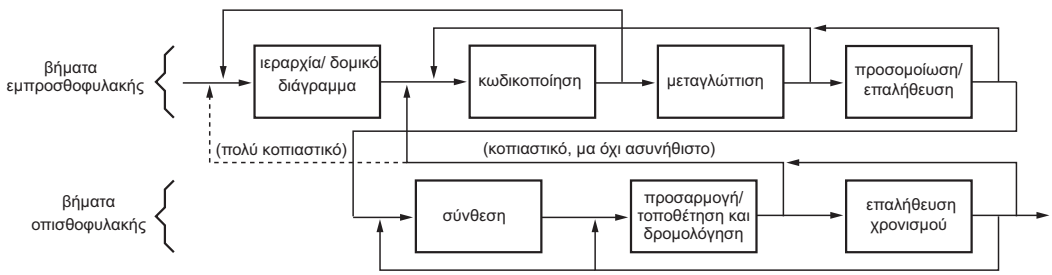
Το “Verilog” δεν είναι ακρώνυμο, αλλά έχει κάποιους ενδιαφέροντες αναγραμματισμούς, όπως π.χ. “I, Clover” (Danny;), “G.I. lover”, “Go, liver!” και “I grovel”. Ας σοβαρευτούμε, όμως! Θα μπορούσε να είναι μια σύντμηση του “VERIfy LOGic” (Επαλήθευση λογικής).

προσομοιωτής VHDL

Ίσως το πλέον ικανοποιητικό βήμα έρχεται μετά: η προσομοίωση. Ο προσομοιωτής VHDL σάς επιτρέπει να καθορίζετε και να εφαρμόζετε εισόδους στη σχεδίασή σας και να παρατηρείτε τις εξόδους της, χωρίς να χρειάζεται να κατασκευάσετε το φυσικό κύκλωμα. Σε μικρά έργα, του είδους που θα μπορούσατε να κάνετε ως εργασία στο σπίτι σε ένα μάθημα ψηφιακής σχεδίασης, θα μπορούσατε να δημιουργείτε εισόδους και να παρατηρείτε τις εξόδους μόνοι σας. Σε μεγαλύτερα έργα, όμως, η VHDL σάς δίνει τη δυνατότητα να δημιουργείτε “περιβάλλοντα δοκιμών” που εφαρμόζουν αυτόματα εισόδους και τις συγκρίνουν με τις αναμενόμενες εξόδους.

επαλήθευση

Στην πραγματικότητα, η προσομοίωση είναι απλώς ένα μέρος ενός μεγαλύτερου βήματος που λέγεται *επαλήθευση*. Βεβαίως, σας ικανοποιεί να βλέπετε το προσομοιωμένο κύκλωμα να παράγει προσομοιωμένες εξόδους, αλλά ο σκοπός της προσομοίωσης είναι μεγαλύτερος: να *επαληθεύει* ότι το κύκλωμα λειτουργεί όπως θέλουμε. Σε ένα συνηθισμένο μεγάλο έργο, ένα σημαντικό μέρος της προσπάθειάς σας στο στάδιο της κωδικοποίησης και μετά από αυτό αναλώνεται στον καθορισμό περιπτώσεων ελέγχου οι οποίες εξετάζουν το κύκλωμα σε ένα μεγάλο εύρος λογικών συνθηκών λειτουργίας. Η εύρεση σφαλμάτων σχεδίασης στο στάδιο αυτό έχει πολύ μεγάλη αξία επειδή, αν τα σφάλματα αυτά βρεθούν αργότερα, θα πρέπει να επαναληφθούν τυπικά όλα τα λεγόμενα βήματα “οπισθοφυλακής” (“back-end”).



Εικόνα 4-50 Βήματα της σχεδίασης ροής σε VHDL ή σε άλλη γλώσσα HDL.

Σημειώστε ότι υπάρχουν τουλάχιστον δύο διαστάσεις προς επαλήθευση. Στην *επαλήθευση λειτουργίας* μελετούμε τη λογική λειτουργία του κυκλώματος ανεξάρτητα από ζητήματα χρονισμού. Οι καθυστερήσεις των πυλών και οι άλλες παράμετροι χρονισμού θεωρούνται ότι είναι μηδενικές. Στην *επαλήθευση χρονισμού* μελετούμε τη λειτουργία του κυκλώματος συμπεριλαμβάνοντας εκτιμώμενες καθυστερήσεις και επαληθεύουμε την ικανοποίηση της διαμόρφωσης, η διατήρησης, και των λοιπών απαιτήσεων χρονισμού ακολουθιακών συσκευών όπως τα δισταθή κυκλώματα (flip-flop). Συνηθίζεται να εκτελούμε λεπτομερή *επαλήθευση λειτουργίας* προτού ξεκινήσουμε τα βήματα "οπισθοφυλακής". Ωστόσο, η δυνατότητα που έχουμε να κάνουμε επαλήθευση *χρονισμού* σε αυτό το στάδιο είναι συχνά περιορισμένη, επειδή ο χρονισμός ενδέχεται να εξαρτάται σε μεγάλο βαθμό από τα αποτελέσματα της σύνθεσης και της προσαρμογής. Μπορούμε να κάνουμε μια προκαταρκτική επαλήθευση χρονισμού για να αποκτήσουμε κάποια άνεση με τη συνολική σχεδιαστική προσέγγιση, αλλά για την τελική επαλήθευση χρονισμού πρέπει να περιμένουμε μέχρι το τέλος.

επαλήθευση λειτουργίας

επαλήθευση χρονισμού

Μετά από την επαλήθευση, είμαστε έτοιμοι να μεταβούμε στο στάδιο της "οπισθοφυλακής". Η φύση και τα εργαλεία του σταδίου αυτού ποικίλλουν κάπως, ανάλογα με την τεχνολογία προορισμού για τη συγκεκριμένη σχεδίαση, αλλά υπάρχουν τρία βασικά βήματα. Το πρώτο είναι η *σύνθεση,* δηλαδή η μετατροπή της περιγραφής VHDL σε ένα σύνολο από βασικά στοιχεία τα οποία μπορούν να συντεθούν στην τεχνολογία προορισμού. Για παράδειγμα, στις διατάξεις PLD ή CPLD, το εργαλείο σύνθεσης μπορεί να δημιουργήσει εξισώσεις αθροίσματος γινομένων δύο επιπέδων. Στα ASIC μπορεί να δημιουργήσει μια λίστα από πύλες και μια *λίστα δικτύου* η οποία περιγράφει τον τρόπο διασύνδεσής τους. Ο σχεδιαστής μπορεί να "βοηθήσει" το εργαλείο σύνθεσης καθορίζοντας ορισμένους *περιορισμούς* που αφορούν τη συγκεκριμένη τεχνολογία, όπως είναι ο μέγιστος αριθμός των λογικών επιπέδων ή η ισχύς των λογικών κυκλωμάτων των απομονωτών που θα χρησιμοποιηθούν.

σύνθεση

λίστα δικτύου

περιορισμοί

Στο βήμα της *προσαρμογής*, το εργαλείο προσαρμογής ή αλλιώς ο *προσαρμοστής* αντιστοιχίζει τα βασικά στοιχεία που έχουν συντεθεί στους διαθέσιμους πόρους συσκευών. Για τις διατάξεις PLD ή CPLD, αυτό μπορεί να σημαίνει αντιστοίχιση εξισώσεων σε διαθέσιμα στοιχεία AND-OR. Για τα ASIC, αυτό μπορεί να σημαίνει τη διάταξη επιμέρους πυλών σε ένα υπόδειγμα και την εύρεση τρόπων για τη σύνδεσή τους με τους φυσικούς περιορισμούς της φόρμας του ASIC. Αυτό λέγεται διαδικασία *τοποθέτησης και δρομολόγησης*. Ο σχεδιαστής συνήθως μπορεί να καθορίσει επιπλέον περιορισμούς σε αυτό το στάδιο, όπως είναι η τοποθέτηση λειτουργικών μονάδων σε ένα ολοκληρωμένο κύκλωμα ή οι αντιστοιχίσεις ακροδεκτών των εξωτερικών ακροδεκτών εισόδου και εξόδου.

προσαρμογή προσαρμοστής

τοποθέτηση και δρομολόγηση

Το "τελικό" βήμα είναι η επαλήθευση του χρονισμού του προσαρμοσμένου κυκλώματος. Μόνο σε αυτό το στάδιο είναι δυνατόν να υπολογιστούν οι πραγματικές καθυστερήσεις του κυκλώματος λόγω του μή-

κους των καλωδίων, του ηλεκτρικού φορτίου και άλλων παραγόντων, με εύλογη ακρίβεια. Κατά τη διάρκεια αυτού του βήματος συνηθίζεται η εφαρμογή των ίδιων περιπτώσεων ελέγχου που είχαν χρησιμοποιηθεί στην επαλήθευση λειτουργίας, αλλά στο βήμα αυτό εκτελούνται πάνω στο κύκλωμα όπως αυτό πρόκειται να κατασκευαστεί στην πραγματικότητα.

Όπως και σε κάθε άλλη δημιουργική διαδικασία, ενδέχεται μερικές φορές να κάνετε δύο βήματα εμπρός και ένα βήμα πίσω (ή ακόμη χειρότερα!). Όπως υπονοείται στην εικόνα, κατά τη διάρκεια της κωδικοποίησης ενδέχεται να αντιμετωπίσετε προβλήματα τα οποία θα σας αναγκάσουν να γυρίσετε πίσω και να αναθεωρήσετε την ιεραρχία σας, ενώ είναι σχεδόν σίγουρο ότι θα συναντήσετε σφάλματα μεταγλώττισης και προσομοίωσης τα οποία θα σας αναγκάσουν να ξαναγράψετε τμήματα του κώδικα.

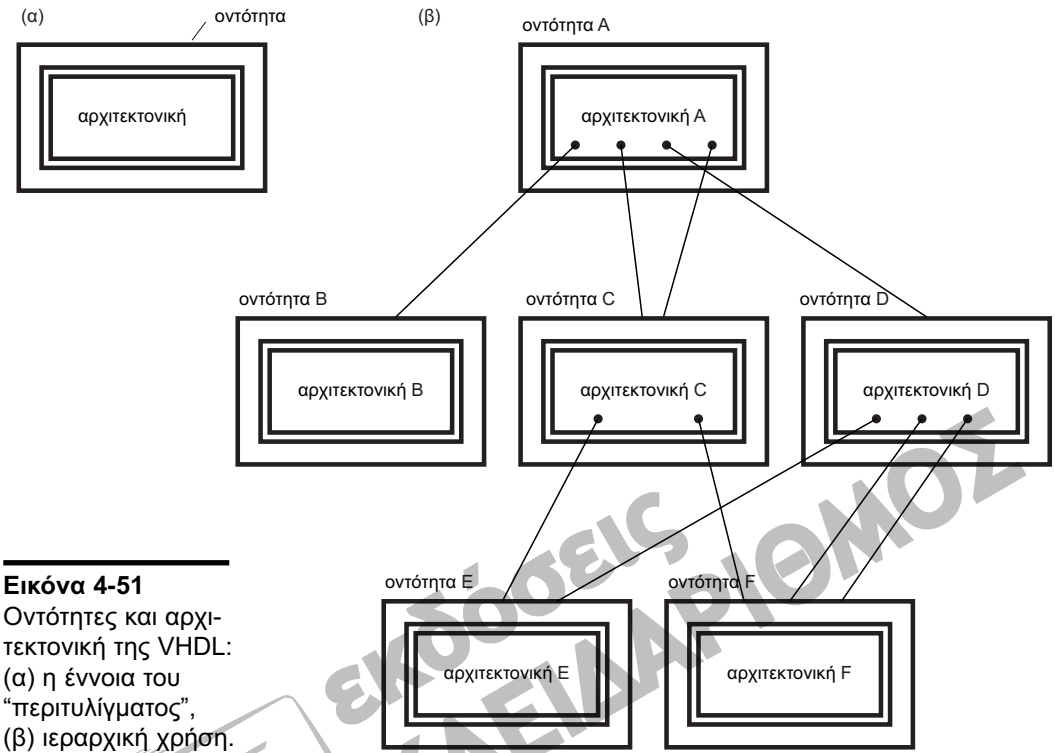
Τα πιο επίπονα προβλήματα είναι αυτά που απαντώνται στην "οπισθοφυλακή" της σχεδιαστικής ροής. Για παράδειγμα, αν η σχεδίαση που έχετε συνθέσει δεν ταιριάζει σε ένα διαθέσιμο FPGA ή δεν καλύπτει τις απαιτήσεις χρονισμού, ενδέχεται να χρειαστεί να γυρίσετε πίσω μέχρι του σημείου να αναθεωρήσετε ολόκληρη τη σχεδιαστική προσέγγιση. Αξίζει να θυμάστε ότι τα εξαιρετικά εργαλεία δεν αντικαθιστούν την προσεκτική σκέψη στο ξεκίνημα μιας σχεδίασης.

ΔΟΥΛΕΥΕΙ!; Ως σχεδιαστής λογικών κυκλωμάτων και κατασκευαστής συστημάτων για πολλά χρόνια, πάντα νόμιζα ότι ήξερα τι σημαίνει να λέει κάποιος για το κύκλωμά του: "δουλεύει!" Σημαίνει ότι μπορείτε να πάτε στο εργαστήριο και να συνδέσετε στην τροφοδοσία ένα πρωτότυπο χωρίς να δείτε καπνούς, να πατήσετε ένα κουμπί επανεκκίνησης, και να χρησιμοποιήσετε έναν παλμογράφο ή λογικό αναλυτή για να δείτε το πρωτότυπο να εκτελεί ένα-προς-ένα τα βήματά του.

Πάντως, με τα χρόνια, η σημασία του "δουλεύει" έχει αλλάξει, τουλάχιστον για κάποιους ανθρώπους. Όταν με προσέλαβαν σε μια νέα δουλειά πριν από λίγα χρόνια, χαιρόμουν πολύ όταν άκουγα ότι πολλά σημαντικά ASIC για ένα σημαντικό νέο προϊόν "δούλευαν" όλα. Αλλά αργότερα (λίγο αργότερα) αντιλήφθηκα ότι τα ASIC λειτουργούσαν μόνο στην προσομοίωση και ότι η σχεδιαστική ομάδα είχε ακόμη πολλούς μήνες σκληρής δουλειάς μπροστά της για τη σύνθεση, την προσαρμογή, την επαλήθευση χρονισμού, και την επανάληψη, προτού μπορέσει να παραγγείλει οποιοδήποτε πρωτότυπο. "Δουλεύει!", σίγουρα. Όπως ακριβώς οι σχολικές εργασίες των παιδιών μου όταν μου λένε "Τελείωσα!".

4.7.2 Δομή του προγράμματος

Η VHDL σχεδιάστηκε με τις αρχές του δομημένου προγραμματισμού, υιοθετώντας ιδέες από τις γλώσσες προγραμματισμού λογισμικού Pascal και Ada. Μία από τις βασικές ιδέες είναι ο καθορισμός της διασύνδεσης μιας λειτουργικής μονάδας υλικού και η ταυτόχρονη απόκρυψη των εσωτερικών στοιχείων της. Έτσι, η *οντότητα* VHDL είναι απλώς μια δήλωση



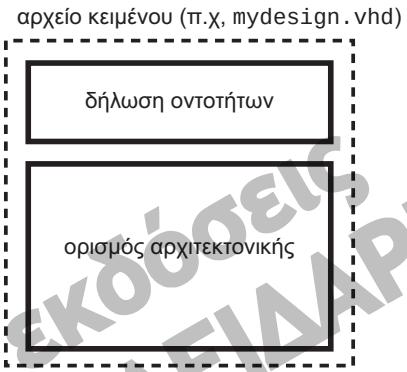
Εικόνα 4-51
Οντότητες και αρχιτεκτονική της VHDL: (α) η έννοια του "περιτυλίγματος", (β) ιεραρχική χρήση.

των εισόδων και των εξόδων μιας λειτουργικής μονάδας, ενώ η *αρχιτεκτονική* VHDL είναι η λεπτομερής περιγραφή της εσωτερικής δομής ή της συμπεριφοράς μιας λειτουργικής μονάδας.

οντότητα
αρχιτεκτονική

Η Εικόνα 4-51(α) απεικονίζει αυτή την έννοια. Πολλοί σχεδιαστές αρέσκονται να θεωρούν τη δήλωση μιας οντότητας VHDL ως "περιτύλιγμα" της αρχιτεκτονικής, το οποίο αποκρύπτει τις λεπτομέρειες που υπάρχουν στο εσωτερικό ενώ παρέχει τα "άγκιστρα" για να τη χρησιμοποιούν οι άλλες λειτουργικές μονάδες. Αυτό διαμορφώνει τη βάση της σχεδίασης ιεραρχικών συστημάτων, σύμφωνα με την οποία η αρχιτεκτονική μιας οντότητας ανωτάτου επιπέδου μπορεί να χρησιμοποιεί (ή να "συγκεκριμενοποιεί") άλλες οντότητες και ταυτόχρονα να αποκρύπτει τις αρχιτεκτονικές λεπτομέρειες των οντοτήτων κατώτερου επιπέδου από τις οντότητες ανωτάτου επιπέδου. Όπως φαίνεται στην περίπτωση (β), μια αρχιτεκτονική ανώτερου επιπέδου είναι δυνατόν να χρησιμοποιεί μια οντότητα κατώτερου επιπέδου περισσότερες από μία φορές, ενώ πολλές αρχιτεκτονικές ανωτάτου επιπέδου είναι δυνατόν να χρησιμοποιούν την ίδια οντότητα κατώτερου επιπέδου. Σε αυτή την εικόνα οι αρχιτεκτονικές B, E, και F είναι αυτόνομες, δηλαδή δε χρησιμοποιούν καμία άλλη οντότητα.

ΑΣ ΔΙΑΜΟΡΦΩΣΟΥΜΕ! Στην πραγματικότητα, η VHDL σάς επιτρέπει να καθορίζετε πολλές αρχιτεκτονικές για μία και μοναδική οντότητα, ενώ παρέχει μια δυνατότητα διαχείρισης της διαμόρφωσης η οποία σας επιτρέπει να καθορίζετε ποια αρχιτεκτονική θα χρησιμοποιείτε κατά τη διάρκεια της εκτέλεσης μιας συγκεκριμένης μεταγλώττισης ή σύνθεσης. Αυτό σας επιτρέπει να δοκιμάζετε στην πράξη μια διαφορετική αρχιτεκτονική προσέγγιση χωρίς να απορρίπτετε ή να αποκρύπτετε τις άλλες προσπάθειές σας. Ωστόσο, δε θα χρησιμοποιήσουμε και δε θα συζητήσουμε περαιτέρω αυτή τη δυνατότητα στο βιβλίο.



Εικόνα 4-52
Δομή αρχείου προγράμματος VHDL.

δήλωση οντοτήτων
ορισμός αρχιτεκτονικής

Στο αρχείο κειμένου ενός προγράμματος VHDL, η *δήλωση οντοτήτων* και ο *ορισμός αρχιτεκτονικής* είναι ξεχωριστά, όπως φαίνεται στην Εικόνα 4-52. Για παράδειγμα, ο Πίνακας 4-26 είναι ένα πολύ απλό πρόγραμμα VHDL για μια πύλη "ανακοπής" δύο εισόδων. Σε μεγάλα έργα οι οντότητες και οι αρχιτεκτονικές μερικές φορές ορίζονται σε ξεχωριστά αρχεία, τα οποία συνδυάζονται μεταξύ τους από το μεταγλωττιστή σύμφωνα με τα δηλωμένα ονόματά τους.

Όπως και οι άλλες γλώσσες προγραμματισμού υψηλού επιπέδου, η VHDL αγνοεί γενικά τα κενά διαστήματα και τις αλλαγές γραμμών που ενδέχεται να υπάρχουν σε ορισμένα σημεία για λόγους αναγνωσιμότητας. Τα *σχόλια* ξεκινούν με δυο παύλες (--) και τελειώνουν στο τέλος της γραμμής.

σχόλια

δεσμευμένες λέξεις
λέξεις-κλειδιά

Η VHDL ορίζει πολλά ειδικά αλφαριθμητικά που λέγονται *δεσμευμένες λέξεις* ή *λέξεις-κλειδιά*. Το παράδειγμά μας περιλαμβάνει μερικές από αυτές: `entity`, `port`, `is`, `in`, `end`, `architecture`, `begin`, `when`, `else`, και `not`. Τα *αναγνωριστικά* που ορίζονται από το χρήστη αρχίζουν με ένα γράμμα και περιέχουν γράμματα, αριθμούς, και χαρακτήρες υπογράμμισης. (Ένας χαρακτήρας υπογράμμισης δεν μπορεί να ακολουθεί έναν άλλο χαρακτήρα υπογράμμισης ή να είναι ο τελευταίος χαρακτήρας σε ένα αναγνωριστικό.) Στο παράδειγμά μας, αναγνωριστικά είναι τα `Inhibit`, `X`, `Y`, `BIT`, `Z`, και `Inhibit_arch`. Το "BIT" εί-

αναγνωριστικά

```
entity Inhibit is          -- γνωστό και ως 'BUT-NOT'
  port (X,Y: in BIT;       -- όπως 'X αλλά όχι Y'
      Z: out BIT);         -- (δείτε [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Πίνακας 4-26 Πρόγραμμα VHDL για μια πύλη "ανακοπής".

```
entity όνομα-οντότητας is
   port (ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος;
         ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος;
         ...
         ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος);
end όνομα-οντότητας;
```

Πίνακας 4-27 Σύνταξη δήλωσης οντότητας στη VHDL.

ναι ένα ενσωματωμένο αναγνωριστικό για έναν προκαθορισμένο τύπο και δε θεωρείται λέξη-κλειδί επειδή μπορεί να οριστεί ξανά. Οι δεσμευμένες λέξεις και τα αναγνωριστικά δεν έχουν διάκριση πεζών-κεφαλαίων.

Μια βασική δήλωση οντότητας έχει τη σύνταξη που φαίνεται στον Πίνακα 4-27. Εκτός από την ονομασία της οντότητας, ο σκοπός της δήλωσης της οντότητας είναι να ορίσει τα σήματα εξωτερικής διασύνδεσης, ή αλλιώς *θύρες*, στο τμήμα *δήλωσης θυρών*. Εκτός από τις λέξεις-κλειδιά `entity`, `is`, `port`, και `and`, μια δήλωση οντότητας έχει τα παρακατω στοιχεία:

θύρα
δήλωση θυρών

όνομα-οντότητας Αναγνωριστικό επιλεγμένο από το χρήστη για την ονομασία της οντότητας

ονόματα-σημάτων Λίστα, με διαχωριστικά κόμματα, ενός ή περισσοτέρων επιλεγμένων από το χρήστη αναγνωριστικών για την ονομασία των σημάτων εξωτερικής διασύνδεσης.

τρόπος-λειτουργίας Μία από τέσσερις δεσμευμένες λέξεις που καθορίζουν την κατεύθυνση του σήματος:

in Το σήμα είναι είσοδος της οντότητας.

out Το σήμα είναι έξοδος της οντότητας. Σημειώστε ότι η τιμή ενός τέτοιου σήματος δεν μπορεί να "αναγνωστεί" στο εσωτερικό της αρχιτεκτονικής της οντότητας, παρά μόνο από άλλες οντότητες που τη χρησιμοποιούν.

buffer Το σήμα είναι έξοδος της οντότητας, και επιπλέον η τιμή του μπορεί να "αναγνωστεί" στο εσωτερικό της αρχιτεκτονικής της οντότητας.

inout Το σήμα μπορεί να χρησιμοποιηθεί ως είσοδος ή έξοδος της οντότητας. Αυτός ο τρόπος λειτουργίας χρησιμοποιείται συνήθως σε ακροδέκτες εισόδου/εξόδου τριών καταστάσεων στις διατάξεις PLD.

τύπος-σήματος Ενσωματωμένος ή οριζόμενος από το χρήστη τύπος σήματος. Στην επόμενη ενότητα θα αναφέρουμε πολλά για τους τύπους των σημάτων.

Σημειώστε ότι δεν υπάρχει ελληνικό ερωτηματικό μετά το τελευταίο *τύπος-σήματος*. Η εναλλαγή της παρένθεσης κλεισίματος με το ελληνικό ερωτηματικό μετά από το τελευταίο *τύπος-σήματος* είναι ένα συνηθισμένο συντακτικό σφάλμα για τους αρχάριους προγραμματιστές VHDL.

ορισμός αρχιτεκτονικής

Οι θύρες μιας οντότητας, οι τρόποι λειτουργίας τους, και οι τύποι είναι όλα όσα φαίνονται από τις άλλες λειτουργικές μονάδες που τη χρησιμοποιούν. Η εσωτερική λειτουργία της οντότητας περιγράφεται στον *ορισμό αρχιτεκτονικής* της, του οποίου η γενική σύνταξη φαίνεται στον Πίνακα 4-28. Το *όνομα-οντότητας* του ορισμού αυτού πρέπει να είναι ίδιο με εκείνο που δόθηκε νωρίτερα στη δήλωση της οντότητας. Το *όνομα-αρχιτεκτονικής* είναι ένα αναγνωριστικό επιλεγόμενο από το χρήστη, που συνήθως σχετίζεται με το όνομα της οντότητας. Μπορεί να είναι το ίδιο με το όνομα της οντότητας, αν θέλετε.

Τα σήματα εξωτερικής διασύνδεσης μιας αρχιτεκτονικής (δηλαδή οι θύρες) κληρονομούνται από το τμήμα δήλωσης θυρών της αντίστοιχης δήλωσης οντότητας. Μια αρχιτεκτονική μπορεί να περιλαμβάνει επίσης σήματα και άλλες δηλώσεις οι οποίες είναι τοπικές σε αυτή την αρχιτεκτονική, όπως συμβαίνει και με τις άλλες γλώσσες υψηλού επιπέδου. Οι δηλώσεις που είναι κοινές για πολλές οντότητες μπορούν να γίνουν σε ένα ξεχωριστό "πακέτο" το οποίο χρησιμοποιείται από όλες τις οντότητες, όπως εξηγείται παρακάτω.

δήλωση σημάτων

Οι δηλώσεις του Πίνακα 4-28 είναι δυνατόν να εμφανίζονται με οποιαδήποτε σειρά. Όταν έρθει η κατάλληλη στιγμή, θα αναλύσουμε πολλά διαφορετικά είδη δηλώσεων και προτάσεων που είναι δυνατόν να εμφανίζονται στη δήλωση αρχιτεκτονικής – αλλά για αρχή, το ευκολότερο είναι η *δήλωση σημάτων*. Αυτή δίνει τις ίδιες πληροφορίες για ένα σήμα που δίνονται και σε μια δήλωση θυρών, με τη διαφορά ότι δεν καθορίζεται τρόπος λειτουργίας:

`signal` *ονόματα-σημάτων* `:` *τύπος-σήματος*;

Μέσα στα πλαίσια μιας αρχιτεκτονικής, είναι δυνατόν να οριστούν μηδέν ή περισσότερα σήματα, που θα αντιστοιχούν κατά προσέγγιση στα κατονομασμένα καλώδια ενός λογικού διαγράμματος. Επίσης, η ανάγνωση και η εγγραφή τους είναι δυνατές μέσα στα πλαίσια της δήλωσης αρχιτεκτονικής ενώ, όπως συμβαίνει και με τα άλλα τοπικά αντικείμενα,

```
architecture όνομα-αρχιτεκτονικής of όνομα-οντότητας is
  δηλώσεις τύπων
  δηλώσεις σημάτων
  δηλώσεις σταθερών
  ορισμοί συναρτήσεων
  ορισμοί διαδικασιών
  δηλώσεις στοιχείων
begin
  ταυτόχρονη-εντολή
  ...
  ταυτόχρονη-εντολή
end όνομα-αρχιτεκτονικής;
```

Πίνακας 4-28 Σύνταξη ενός ορισμού αρχιτεκτονικής σε VHDL.

παραπομπές σε αυτά επιτρέπονται μόνο μέσα στα πλαίσια του περιβάλλοντος ορισμού αρχιτεκτονικής.

Οι *μεταβλητές* της VHDL μοιάζουν με τα σήματα, με τη διαφορά ότι συνήθως δεν έχουν φυσική σημασία σε ένα κύκλωμα. Παρατηρήστε ότι ο Πίνακας 4-28 δεν προβλέπει "δηλώσεις μεταβλητών" σε έναν ορισμό αρχιτεκτονικής. Αντίθετα, μεταβλητές χρησιμοποιούνται στις συναρτήσεις, τις διαδικασίες, και τις διεργασίες της VHDL, τις οποίες θα εξετάσουμε με τη σειρά αργότερα. Μέσα σε αυτά τα στοιχεία προγράμματος, η σύνταξη μιας *δήλωσης μεταβλητών* είναι ακριβώς όπως εκείνη μιας δήλωσης σήματος, με τη διαφορά ότι εδώ χρησιμοποιείται η λέξη-κλειδί `variable`:

μεταβλητή

δήλωση μεταβλητών

```
variable ονόματα-μεταβλητών : τύπος-μεταβλητών;
```

4.7.3 Τύποι και σταθερές

Όλα τα σήματα, οι μεταβλητές, και οι σταθερές ενός προγράμματος VHDL πρέπει να έχουν έναν αντίστοιχο "τύπο". Ο *τύπος* καθορίζει το σύνολο ή το πεδίο των τιμών τις οποίες μπορεί να πάρει το αντικείμενο, ενώ για κάθε δεδομένο τύπο υπάρχει συνήθως ένα σύνολο αντίστοιχων τελεστών (π.χ. πρόσθεση, AND κ.λπ.).

τύπος

Η VHDL έχει μερικούς *προκαθορισμένους τύπους*, που παρατίθενται στον Πίνακα 4-29. Στο υπόλοιπο αυτού του βιβλίου, οι μόνοι προκαθορισμένοι τύποι που θα χρησιμοποιήσουμε είναι οι `integer`, `character`, και `boolean`. Θα περίμενε κανείς ότι οι τύποι που ονομάζονται "`bit`" και "`bit_vector`" θα ήταν ιδιαίτερα σημαντικοί για την ψηφιακή σχεδίαση, αλλά αποδεικνύεται στην πράξη ότι οι εκδοχές αυτών των τύπων που ορίζονται από το χρήστη είναι πιο χρήσιμες, όπως θα δούμε παρακάτω.

προκαθορισμένοι τύποι

Ο τύπος `integer` ορίζεται ως το πεδίο τιμών των ακεραίων που περιλαμβάνει τουλάχιστον την περιοχή από -2.147.483.647 έως και +2.147.483.647 ($-2^{31}+1$ έως και $+2^{31}-1$). Οι υλοποιήσεις της VHDL μπο-

Πίνακας 4-29
Προκαθορισμένοι τύποι της VHDL.

bit	character	severity_level
bit_vector	integer	string
Boo	leanreal	time

Πίνακας 4-30
Προκαθορισμένοι τελεστές για τους τύπους integer και boolean της VHDL.

Τελεστές για τον τύπο `integer`		*Τελεστές για τον τύπο* `boolean`	
+	πρόσθεση	and	AND
-	αφαίρεση	or	OR
*	πολλαπλασιασμός	nand	NAND
/	διαίρεση	nor	NOR
Mod	διαίρεση μόντουλο	xor	Αποκλειστικό OR
Rem	υπόλοιπο μόντουλο	xnor	Αποκλειστικό NOR
Abs	απόλυτη τιμή	not	Συμπλήρωμα
**	εκθετική τιμή		

integer
boolean
true, false
character

ρεί να υπερβαίνουν αυτό το πεδίο τιμών. Ο τύπος *boolean* έχει δύο τιμές, *true* και *false*. Ο τύπος *character* περιέχει όλους τους χαρακτήρες του συνόλου χαρακτήρων ISO των 8 bit. Οι πρώτοι 128 είναι οι χαρακτήρες ASCII. Οι ενσωματωμένοι τελεστές για τους τύπους integer και boolean παρατίθενται στον Πίνακα 4-30.

Οι συνηθέστερα χρησιμοποιούμενοι τύποι στα τυπικά προγράμματα VHDL είναι οι *τύποι που ορίζονται από το χρήστη* και οι πιο συνηθισμένοι από αυτούς είναι οι *απαριθμητοί τύποι*, οι οποίοι ορίζονται με απαρίθμηση των τιμών τους. Οι προκαθορισμένοι τύποι boolean και

ΑΥΣΤΗΡΗ ΔΗΛΩΣΗ ΤΥΠΩΝ

Αντίθετα από τη C, η VHDL είναι μια γλώσσα με αυστηρή δήλωση τύπων. Αυτό σημαίνει ότι ο μεταγλωττιστής δε σας επιτρέπει να αποδίδετε μια τιμή σε ένα σήμα ή μια μεταβλητή, εκτός αν ο τύπος της τιμής συμφωνεί επακριβώς με το δηλωμένο τύπο του σήματος ή της μεταβλητής.

Η αυστηρή δήλωση τύπων είναι ευλογία και κατάρα μαζί. Κάνει το πρόγραμμά σας πιο αξιόπιστο και εύκολο στην αποσφαλμάτωση, επειδή δε σας επιτρέπει να κάνετε “ανόητα λάθη” αποδίδοντας μια τιμή εσφαλμένου τύπου ή μεγέθους. Από την άλλη πλευρά, αυτό μερικές φορές γίνεται εκνευριστικό. Ακόμη και σε απλές πράξεις, όπως είναι η επανερμηνεία ενός σήματος των 2 bit ως ακεραίου (π.χ. για να επιλέξετε ένα από τα τέσσερα αποτελέσματα σε μια εντολή “case”), ενδέχεται να πρέπει να καλέσετε ρητά μια συνάρτηση μετατροπής τύπου.

`character` είναι απαριθμητοί τύποι. Μια δήλωση τύπου για έναν απαριθμητό τύπο έχει τη μορφή που παρουσιάζεται στην πρώτη γραμμή του Πίνακα 4-31. Εδώ, η *λίστα-τιμών* είναι μια λίστα (απαρίθμηση) όλων των δυνατών τιμών του τύπου, με το κόμμα ως διαχωριστικό. Οι τιμές μπορούν είναι αναγνωριστικά καθορισμένα από το χρήστη ή χαρακτήρες (όπου "χαρακτήρας" είναι ένας χαρακτήρας ISO που περικλείεται σε μονά εισαγωγικά). Το πρώτο στιλ χρησιμοποιείται πιο συχνά για τον καθορισμό περιπτώσεων ή καταστάσεων για μια μηχανή καταστάσεων, όπως για παράδειγμα:

```
type traffic_light_state is (reset, stop, wait, go);
```

Το δεύτερο στιλ χρησιμοποιείται στην πολύ σημαντική περίπτωση ενός οριζόμενου από το χρήστη τυπικού λογικού τύπου `std_logic`, που φαίνεται στον Πίνακα 4-32 και αποτελεί μέρος του τυπικού πακέτου IEEE 1164 που περιγράφεται στην Ενότητα 4.7.5. Αυτός ο τύπος περιλαμβάνει όχι μόνο `'0'` και `'1'` αλλά και επτά άλλες τιμές οι οποίες έχουν φανεί χρήσιμες στην προσομοίωση ενός λογικού σήματος (bit) σε ένα πραγματικό λογικό κύκλωμα, όπως εξηγείται με περισσότερες λεπτομέρειες στην Ενότητα 5.6.4.

τύποι που ορίζονται από το χρήστη `std_logic`

Η VHDL επιτρέπει επίσης στους χρήστες να δημιουργούν *υποτύπους* ενός τύπου, χρησιμοποιώντας τη σύνταξη που φαίνεται στον Πίνακα 4-31. Οι τιμές του υποτύπου πρέπει να είναι ένα συνεχές πεδίο τιμών του βασικού τύπου, από την *αρχή* μέχρι το *τέλος*. Για έναν απαριθμητό τύπο,

υποτύποι

απαριθμητός τύπος


```
type ονομα-τύπου is (λίστα-τιμών);

subtype ονομα-υποτύπου is ονομα-τύπου αρχή to τέλος;
subtype ονομα-υποτύπου is ονομα-τύπου αρχή downto τέλος;

constant ονομα-σταθεράς : ονομα-τύπου := τιμή;
```

Πίνακας 4-31 Σύνταξη δηλώσεων τύπων και σταθερών της VHDL.

```
type STD_ULOGIC is ('U',  -- Uninitialized
                    'X',  -- Forcing   Unknown
                    '0',  -- Forcing   0
                    '1',  -- Forcing   1
                    'Z',  -- High Impedance
                    'W',  -- Weak      Unknown
                    'L',  -- Weak      0
                    'H',  -- Weak      1
                    '-'   -- Don't care
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

Πίνακας 4-32 Ορισμός του τύπου std_logic στη VHDL (δείτε την Ενότητα 5.6.4 για επεξήγηση του όρου "resolved").

ΤΙ ΧΑΡΑΚΤΗΡΑΣ! Μπορεί να απορείτε γιατί οι τιμές του τύπου `std_logic` ορίζονται ως χαρακτήρες αντί για αναγνωριστικά του ενός γράμματος. Ασφαλώς, θα ήταν ευκολότερο να πληκτρολογούμε "`U`", "`X`" και τα λοιπά αντί για "`'U'`", "`'X'`" και τα λοιπά. Βέβαια, αυτό θα απαιτούσε ένα αναγνωριστικό του ενός γράμματος διαφορετικό από το "`-`" για να χρησιμοποιείται ως "αδιάφορη" τιμή, αλλά αυτό δεν είναι μεγάλο πρόβλημα. Ο κύριος λόγος που χρησιμοποιούνται χαρακτήρες είναι ότι δεν επιτρέπεται η χρήση των "0" και "1" επειδή αυτά αναγνωρίζονται ήδη ως ακέραιες σταθερές. Αυτό οφείλεται στην αυστηρή δήλωση τύπων της VHDL. Δε θεωρήθηκε σκόπιμο να επιτρέπεται στο μεταγλωττιστή να εκτελεί αυτόματη μετατροπή τύπων ανάλογα με τα συμφραζόμενα.

η έννοια "συνεχόμενο" αναφέρεται στις θέσεις στην πρωτότυπη *λίστα-τιμών* που δίνει τους ορισμούς. Ακολουθούν μερικά παραδείγματα ορισμών υποτύπων:

```
subtype twoval_logic is std_logic range '0' to '1';
subtype fourval_logic is std_logic range 'X' to 'Z';
subtype negint is integer range -2147483647 to -1;
subtype bitnum is integer range 31 downto 0;
```

to
downto

Παρατηρήστε ότι η διάταξη ενός πεδίου τιμών είναι δυνατόν να οριστεί ως αύξουσα ή φθίνουσα, ανάλογα με το αν χρησιμοποιείται το *`to`* ή το *`downto`*. Υπάρχουν ορισμένες ιδιότητες των υποτύπων για τις οποίες αυτή η διάκριση είναι πολύ σημαντική, αλλά δεν τις χρησιμοποιούμε σε αυτό το βιβλίο και δε θα τις συζητήσουμε περισσότερο.

Η VHDL έχει δύο προκαθορισμένους υποτύπους `integer`, που ορίζονται παρακάτω:

σταθερές

δήλωση σταθεράς

Οι *σταθερές* συνεισφέρουν στην αναγνωσιμότητα, συντηρησιμότητα, και φορητότητα των προγραμμάτων σε οποιαδήποτε γλώσσα. Η σύνταξη μιας *δήλωσης σταθεράς* στη VHDL φαίνεται στην τελευταία γραμμή του Πίνακα 4-31. Παρακάτω παρουσιάζονται μερικά παραδείγματα:

Πίνακας 4-33 Σύνταξη δηλώσεων πινάκων στη VHDL.

`type` *όνομα-τύπου* `is array` *(start* `to` *τέλος)* `of` *τύπος-στοιχείου*;
`type` *όνομα-τύπου* `is array` *(αρχή* `downto` *τέλος)* `of` *τύπος-στοιχείου*;
`type` *όνομα-τύπου* `is array` *(τύπος-πεδίου-τιμών)* `of` *τύπος-στοιχείου*;
`type` *όνομα-τύπου* `is array` *(τύπος-πεδίου-τιμών* `range` *αρχή* `to` *τέλος)* `of` *τύπος-στοιχείου*;
`type` *όνομα-τύπου* `is array` *(τύπος-πεδίου-τιμών* `range` *αρχή* `downto` *τέλος)* `of` *τύπος-στοιχείου*;

```
constant BUS_SIZE: integer :=32;     -- πεδίο τιμών στοιχείου
constant MSB: integer := BUS_SIZE-1; -- αριθμός bit του MSB
constant Z: character := 'Z';        -- συνώνυμο της τιμής Hi-Z
```

Προσέξτε ότι οι τιμές μιας σταθεράς μπορούν να είναι μια απλή παράσταση. Οι σταθερές είναι δυνατόν να χρησιμοποιούνται οπουδήποτε μπορεί να χρησιμοποιηθεί η αντίστοιχη τιμή, ενώ μπορούν να έχουν ιδιαίτερα καλή χρήση σε ορισμούς τύπων, όπως θα δείξουμε παρακάτω.

τύποι πινάκων

πίνακας

δείκτης πίνακα

Μια άλλη πολύ σημαντική κατηγορία τύπων που ορίζονται από το χρήστη είναι οι *τύποι πινάκων*. Όπως και σε άλλες γλώσσες, η VHDL ορίζει έναν *πίνακα* ως διατεταγμένο σύνολο στοιχείων του ίδιου τύπου, όπου κάθε στοιχείο επιλέγεται με ένα *δείκτη πίνακα*. Στον Πίνακα 4-33 παρατίθενται αρκετές εκδοχές της σύνταξης για τη δήλωση ενός πίνακα στη VHDL. Στις δύο πρώτες εκδοχές, η *αρχή* και το *τέλος* είναι ακέραιοι οι οποίοι ορίζουν το δυνατό πεδίο τιμών των δεικτών του πίνακα και επομένως το συνολικό αριθμό στοιχείων του πίνακα. Στις τρεις τελευταίες εκδοχές, όλες οι τιμές ή ένα υποσύνολο των τιμών ενός υπάρχοντος τύπου (*τύπος_πεδίου_τιμών*) είναι το πεδίο τιμών των δεικτών του πίνακα.

Στον Πίνακα 4-34 παρατίθενται παραδείγματα δηλώσεων πίνακα. Το πρώτο ζευγάρι παραδειγμάτων είναι πολύ συνηθισμένο και παρουσιάζει αύξοντα και φθίνοντα πεδία τιμών. Το επόμενο παράδειγμα δείχνει πώς μπορεί να χρησιμοποιηθεί μια σταθερά, η `WORD_LEN`, σε μια δήλωση πίνακα, και επίσης ότι μια τιμή που ορίζει περιοχή τιμών μπορεί να είναι μια απλή παράσταση. Το τρίτο παράδειγμα δείχνει ότι ένα στοιχείο πίνακα μπορεί να είναι και το ίδιο πίνακας, δημιουργώντας έτσι έναν πίνακα δύο διαστάσεων. Το τελευταίο παράδειγμα δείχνει ότι ένας απαριθμητός τύπος (ή ένας υποτύπος) μπορεί να καθορίζεται ως πεδίο τιμών των στοιχείων του πίνακα. Στο παράδειγμα αυτό ο πίνακας έχει τέσσερα στοιχεία, με βάση τον προηγούμενο ορισμό του στοιχείου `traffic_light_state`. Τα στοιχεία του πίνακα θεωρούνται ότι διατάσσονται από τα αριστερά προς τα δεξιά, στην ίδια κατεύθυνση με το πεδίο τιμών των δεικτών. Έτσι, τα πιο αριστερά στοιχεία των πινάκων τύπου `monthly_count`, `byte`, `word`, `reg_file`, και `statecount` έχουν δείκτες 1, 7, 31, 1 και `reset` αντίστοιχα.

ΑΦΥΣΙΚΕΣ ΠΡΑΞΕΙΣ

Παρά το γεγονός ότι η VHDL ορίζει τον υποτύπο "`natural`" ως πεδίο τιμών των μη αρνητικών ακεραίων που αρχίζουν από το 0, οι περισσότεροι μαθηματικοί θεωρούν και ορίζουν τους φυσικούς αριθμούς να ξεκινούν από το 1. Στο κάτω-κάτω, στην αρχή της ιστορίας οι άνθρωποι ξεκίνησαν να μετρούν από το 1, ενώ η έννοια του "0" ήρθε πολύ αργότερα. Εξακολουθεί ακόμη να γίνεται πολλή συζήτηση και ενδεχομένως να υπάρχουν διενέξεις πάνω στο θέμα αυτό, ειδικά τώρα που η εποχή των υπολογιστών ώθησε τους περισσότερους από εμάς να θεωρούμε το 0 ως αρχή της αρίθμησης. Για τις πιο πρόσφατες απόψεις, αναζητήστε στον Ιστό τον όρο "natural numbers" (φυσικοί αριθμοί).

Πίνακας 4-34 Παραδείγματα δηλώσεων πινάκων στη VHDL.

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

Μέσα στις προτάσεις ενός προγράμματος VHDL, τα επιμέρους στοιχεία του πίνακα προσπελάζονται με τη βοήθεια του ονόματος του πίνακα και του δείκτη του στοιχείου σε παρενθέσεις. Για παράδειγμα, αν τα M, B, W, R, και S είναι σήματα ή μεταβλητές των πέντε τύπων πινάκων που ορίζονται στον Πίνακα 4-34, τότε τα `M(11)`, `B(5)`, `W(WORD_LEN-5)`, `R(0,0)`, `R(0)`, και `S(reset)` είναι όλα τους έγκυρα στοιχεία.

πίνακας λεκτικών

Οι *πίνακες λεκτικών* περιγράφονται με την αναγραφή των τιμών των στοιχείων σε μία λίστα μέσα σε παρενθέσεις. Για παράδειγμα, η ανάθεση της τιμής "1" σε όλα τα στοιχεία της μεταβλητής B, η οποία είναι τύπου `byte`, θα μπορούσε να γίνει με την εντολή

```
B := ('1','1','1','1','1','1','1','1');
```

Η VHDL έχει επίσης μια βολική σημειογραφία η οποία σας επιτρέπει να καθορίζετε τιμές κατά δείκτη. Για παράδειγμα, για να αποδώσετε στη μεταβλητή W, που είναι τύπου `word`, την τιμή 1 παντού εκτός από τα λιγότερο σημαντικά bit (LSB) του κάθε byte, στα οποία θα αποδώσετε την τιμή 0, μπορείτε να γράψετε

others

```
W := (0=>'0',8=>'0',16=>'0',24=>'0',others=>'1');
```

Οι μέθοδοι που περιγράψαμε παραπάνω μπορούν να χρησιμοποιούνται για πίνακες με οποιονδήποτε *τύπο-στοιχείου*, αλλά ο ευκολότερος τρόπος για να γράψετε ένα λεκτικό τύπου `STD_LOGIC` είναι να χρησιμοποιήσετε ένα "αλφαριθμητικό".

αλφαριθμητικό

Αλφαριθμητικό (*string*) της VHDL είναι μια ακολουθία χαρακτήρων ISO που περικλείονται σε διπλά εισαγωγικά, όπως π.χ. `"Hi there"`. Ένα αλφαριθμητικό είναι απλώς ένας πίνακας χαρακτήρων. Γι' αυτόν το λόγο, στον πίνακα `STD_LOGIC` δεδομένου μήκους είναι δυνατόν να αποδοθεί η τιμή ενός αλφαριθμητικού ίσου μήκους, με την προϋπόθεση ότι οι χαρακτήρες του αλφαριθμητικού λαμβάνονται από το σύνολο των εννιά χαρακτήρων οι οποίοι έχουν οριστεί ως οι δυνατές τιμές των στοιχείων του `STD_LOGIC`, δηλαδή `'0'`, `'1'`, `'U'` κ.λπ. Έτσι, τα δύο προηγούμενα παραδείγματα μπορούν να ξαναγραφούν ως εξής:

```
B := "11111111";
W := "11111110111111101111111011111110";
```

Είναι επίσης δυνατόν να αναφερόμαστε σε ένα συνεχόμενο υποσύνολο ή *τομέα* (*slice*) ενός πίνακα καθορίζοντας τους δείκτες αρχής και τέλους του υποσυνόλου, όπως για παράδειγμα `M(6 to 9)`, `B(3 downto 0)`, `W(15 downto 8)`, `R(0,7 downto 0)`, `R(1 to 2)`, `S(stop to go)`. Παρατηρήστε ότι η κατεύθυνση του τομέα πρέπει να είναι η ίδια με εκείνη του αρχικού πίνακα.

τομέας πίνακα

Τέλος, μπορείτε να συνδυάζετε μεταξύ τους πίνακες ή στοιχεία πινάκων χρησιμοποιώντας τον *τελεστή συνένωσης &*, ο οποίος συνενώνει μεταξύ τους πίνακες και στοιχεία με τη σειρά που γράφονται, από τα αριστερά προς τα δεξιά. Για παράδειγμα, το `'0'&'1'&"1Z"` είναι ισοδύναμο με το `"011Z"`, ενώ η παράσταση `B(6 downto 0)& B(7)` παράγει μια αριστερή κυκλική ολίσθηση ενός bit στον πίνακα `B` που είναι των 8 bit.

τελεστής συνένωσης &

Ο πιο σημαντικός τύπος πίνακα στα τυπικά προγράμματα VHDL είναι ο οριζόμενος από το χρήστη λογικός τύπος `std_logic_vector` κατά το πρότυπο IEEE 1164, ο οποίος ορίζει ένα διατεταγμένο σύνολο από bit `std_logic`. Ο ορισμός αυτού του τύπου έχει την παρακάτω μορφή:

std_logic_vector

```
type STD_LOGIC_VECTOR is array ( natural range <> ) of STD_LOGIC;
```

Αυτό είναι ένα παράδειγμα *τύπου πίνακα χωρίς περιορισμούς*, όπου δηλαδή το πεδίο τιμών του πίνακα δεν καθορίζεται, εκτός από το γεγονός ότι πρέπει να είναι υποσύνολο ενός καθορισμένου τύπου, του `natural` στη συγκεκριμένη περίπτωση. Αυτό το χαρακτηριστικό της VHDL μας επιτρέπει να αναπτύσσουμε αρχιτεκτονικές, συναρτήσεις, και άλλα προγραμματιστικά στοιχεία με πιο γενικό τρόπο, κάπως ανεξάρτητα από το μέγεθος του πίνακα ή το πεδίο τιμών των δεικτών. Ένα πραγματικό πεδίο τιμών καθορίζεται όταν αυτός ο τύπος αποδίδεται σε ένα σήμα ή μια μεταβλητή. Θα δούμε παραδείγματα στην επόμενη ενότητα.

τύπος πίνακα χωρίς περιορισμούς

4.7.4 Συναρτήσεις και διαδικασίες

Όπως συμβαίνει και με τις συναρτήσεις των γλωσσών προγραμματισμού υψηλού επιπέδου, οι *συναρτήσεις* (*functions*) της VHDL δέχονται έναν αριθμό *ορισμάτων* και επιστρέφουν ένα *αποτέλεσμα*. Κάθε ένα από τα ορίσματα, καθώς και το αποτέλεσμα, σε έναν ορισμό ή σε μια κλήση συνάρτησης VHDL έχει έναν προκαθορισμένο τύπο.

συνάρτηση
ορίσματα
αποτέλεσμα

Στον Πίνακα 4-35 φαίνεται η σύνταξη ενός *ορισμού συνάρτησης*. Μετά από το όνομα της συνάρτησης, παρατίθενται μηδέν ή περισσότερες *τυπικές παράμετροι*, οι οποίες χρησιμοποιούνται στα πλαίσια του κορμού της συνάρτησης. Όταν καλείται μια συνάρτηση, οι *πραγματικές παράμετροι* στην κλήση της συνάρτησης αντικαθιστούν τις τυπικές παραμέτρους. Σύμφωνα με την πολιτική αυστηρής δήλωσης τύπων της VHDL, οι πραγματικές παράμετροι πρέπει να είναι του ίδιου τύπου ή υποτύπου με τις τυπικές παραμέτρους. Όταν καλείται η συνάρτηση μέσα από μια

ορισμός συνάρτησης

τυπικές παράμετροι
πραγματικές παράμετροι

αρχιτεκτονική, μια τιμή του *επιστρεφόμενου-τύπου* επιστρέφεται στη θέση της κλήσης της συνάρτησης.

Όπως φαίνεται στον πίνακα, μια συνάρτηση μπορεί να ορίζει τους δικούς της τοπικούς τύπους, σταθερές, μεταβλητές, και ένθετες συναρτήσεις και διαδικασίες. Οι λέξεις-κλειδιά `begin` και `end` περικλείουν μια σειρά από "σειριακές εντολές" οι οποίες εκτελούνται όταν καλείται η συνάρτηση. Αργότερα θα ρίξουμε μια πιο λεπτομερή ματιά στα διαφορετικά είδη σειριακών εντολών και στη σύνταξή τους, αλλά μάλλον μπορείτε να κατανοήσετε τα παραδείγματα αυτά με την υπάρχουσα πείρα σας στον προγραμματισμό.

Η αρχιτεκτονική "πύλης ανακοπής" (inhibit gate) της VHDL που είδαμε στον Πίνακα 4-26, στην Ενότητα 4.7.2, εμφανίζεται στον Πίνακα 4-36 τροποποιημένη έτσι ώστε να χρησιμοποιεί μια συνάρτηση. Στον ορισμό της συνάρτησης, η λέξη-κλειδί `return` υποδηλώνει πότε η συνάρτηση θα πρέπει να επιστρέψει στο τμήμα του κώδικα που την κάλεσε, ενώ ακολουθείται από μια παράσταση με την τιμή που πρέπει να επιστραφεί στο τμήμα του κώδικα που την κάλεσε. Ο τύπος που προκύπτει από αυτή την παράσταση πρέπει να συμφωνεί με τον *επιστρεφόμενο-τύπο* της δήλωσης της συνάρτησης.

return

Το λογικό πακέτο του πρότυπου IEEE 1164 ορίζει πολλές συναρτήσεις οι οποίες έχουν εφαρμογή στους καθιερωμένους τύπους `std_logic` και `std_logic_vector`. Εκτός από τον καθορισμό ενός πλήθους οριζόμενων από το χρήστη τύπων, το πακέτο καθορίζει επίσης τις βασικές λογικές πράξεις που θα εκτελούνται στους τύπους αυτούς, όπως π.χ. `and` και `or`. Έτσι, αξιοποιείται η δυνατότητα της VHDL να *υπερφορτώνει* (*overload*) τελεστές. Αυτή η δυνατότητα επιτρέπει στο χρήστη να καθορίζει μια συνάρτηση η οποία καλείται κάθε φορά που χρησιμοποιείται ένα προκαθορισμένο σύμβολο τελεστή (`and`, `or`, +,

υπερφόρτωση τελεστών

Πίνακας 4-35 Σύνταξη ορισμού συνάρτησης στη VHDL.

```
function όνομα-συνάρτησης (
    ονόματα-σημάτων : τύπος-σήματος;
    ονόματα-σημάτων : τύπος-σήματος;
    ...
     ονόματα-σημάτων : τύπος-σήματος
) return επιστρεφόμενος-τύπος is
  δηλώσεις τύπων
  δηλώσεις σταθερών
  δηλώσεις μεταβλητών
  ορισμοί συναρτήσεων
  ορισμοί διαδικασιών
begin
  σειριακή εντολή
   ...
  σειριακή εντολή
end όνομα-συνάρτησης;
```

κ.λπ.) με ένα αντίστοιχο σύνολο τύπων τελεστέων. Είναι δυνατόν να υπάρχουν πολλοί ορισμοί για ένα δεδομένο σύμβολο τελεστή. Στην περίπτωση αυτή, ο μεταγλωττιστής επιλέγει αυτόματα τον ορισμό που συμφωνεί με τους τύπους των τελεστέων σε κάθε χρήση του τελεστή.

Για παράδειγμα, ο Πίνακας 4-37 παρουσιάζει ένα απόσπασμα κώδικα που προέρχεται από το πακέτο IEEE, το οποίο δείχνει πώς ορίζεται η πράξη "and" για τελεστέους τύπου std_logic. Αυτός ο κώδικας μπορεί να φαίνεται πολύπλοκος, αλλά έχουμε παρουσιάσει ήδη όλα τα βασικά στοιχεία της γλώσσας τα οποία χρησιμοποιεί (εκτός από το "resolved", το οποίο περιγράφεται σε σχέση με τη λογική τριών καταστάσεων στην Ενότητα 5.6.4).

Οι είσοδοι της συνάρτησης μπορεί να είναι τύπου std_ulogic ή του υποτύπου του, std_logic. Ένας άλλος υποτύπος, ο UX01, ορίζεται για να χρησιμοποιείται ως επιστρεφόμενος τύπος της συνάρτησης. Ακόμη και αν μία από τις εισόδους "and" είναι μη λογική τιμή ('Z', 'W', κ.λπ.), η συνάρτηση θα επιστρέφει μόνο μια από τέσσερις δυνατές τιμές. Ο τύπος stdlogic_table ορίζει ένα διδιάστατο πίνακα 9×9 που δεικτοδοτείται με ένα ζεύγος τιμών τύπου std_ulogic. Στον πίνακα and_table, οι τιμές διατάσσονται έτσι ώστε, αν κάποιος δείκτης είναι '0' ή 'L' (ασθενές '0'), η τιμή να είναι '0'. Τιμή '1' υπάρχει μόνο όταν και οι δύο είσοδοι είναι '1' ή 'H' (ασθενές '1'). Διαφορετικά εμφανίζεται μια τιμή 'U' ή 'X'.

Στον ίδιο τον ορισμό της συνάρτησης, τα διπλά εισαγωγικά γύρω από το όνομα της συνάρτησης υποδηλώνουν υπερφόρτωση του τελεστή. Το "εκτελέσιμο" μέρος της συνάρτησης είναι μία και μοναδική παράσταση η οποία επιστρέφει το στοιχείο του πίνακα που δεικτοδοτείται από τις δύο εισόδους, L και R, της συνάρτησης "and".

Λόγω των απαιτήσεων αυστηρής δήλωσης τύπων της VHDL, είναι συχνά απαραίτητο να μετατρέπουμε ένα σήμα από τον έναν τύπο στον άλλο. Για το σκοπό αυτόν, το πακέτο IEEE 1164 περιέχει αρκετές συναρτήσεις μετατροπής τύπων, για παράδειγμα, από BIT σε STD_LOGIC


```
architecture Inhibit_ archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
    if B = '0' then return A;
    else return '0';
    end if;
end ButNot;

begin
    Z <= ButNot(X,Y);
end Inhibit_archf;
```

Πίνακας 4-36 Πρόγραμμα VHDL για μια συνάρτηση "ανακοπής".

Πίνακας 4-37 Ορισμοί που σχετίζονται με την πράξη “and” σε τιμές τύπου STD_LOGIC κατά IEEE 1164.

```
SUBTYPE UX01 IS resolved std_ulogic RANGE 'U' TO '1';
                                                --('U','X','0','1')
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-- truth table for "and" function
CONSTANT and_table : stdlogic_table := (
--      ---------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -     |   |
--      ---------------------------------------------------
        ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | 0 |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ) -- | - |
);
FUNCTION "and" ( L : std_ulogic; R : std_ulogic ) RETURN UX01 IS
BEGIN
       RETURN (and_table(L, R));
END "and";
```


και αντίστροφα. Μια μετατροπή που χρειάζεται συχνά είναι από STD_LOGIC_VECTOR σε μια αντίστοιχη ακέραια τιμή. Το IEEE 1164 δεν περιλαμβάνει τέτοια συνάρτηση μετατροπής, επειδή οι διάφορες σχεδιάσεις ενδέχεται να χρειάζεται να χρησιμοποιούν διαφορετικές ερμηνείες των αριθμών, όπως για παράδειγμα προσημασμένους ή απρόσημους ακεραίους. Ωστόσο, μπορούμε να γράψουμε μια δική μας συνάρτηση, όπως φαίνεται στον Πίνακα 4-38.

Η συνάρτηση CONV_INTEGER χρησιμοποιεί έναν απλό επαναληπτικό αλγόριθμο ισοδύναμο με τον τύπο ανάπτυξης ένθετης παράστασης που είδαμε στην Ενότητα 2.3. Οι εντολές FOR, CASE και WHEN που χρησιμοποιεί η συνάρτηση παρουσιάζονται στην Ενότητα 4.7.8, αλλά εδώ θα πάρετε απλώς μια ιδέα. Η εντολή *null* είναι εύκολη: σημαίνει “μην κάνεις τίποτα”. Το πεδίο τιμών του βρόχου FOR καθορίζεται από το “X'range”, όπου το απλό εισαγωγικό μετά από ένα όνομα σήματος σημαίνει “ιδιότητα”, ενώ το *range* είναι ένα ενσωματωμένο αναγνωριστικό ιδιότητας το οποίο έχει εφαρμογή μόνο στους πίνακες και σημαίνει “πεδίο τιμών των δεικτών αυτού του πίνακα, από τα αριστερά προς τα δεξιά”.

εντολή null

ιδιότητα range

Αντίστροφα, μπορούμε να μετατρέψουμε έναν ακέραιο σε STD_LOGIC_VECTOR όπως φαίνεται στον Πίνακα 4-39. Εδώ πρέπει να καθορίσουμε όχι μόνο την ακέραια τιμή που θα μετατραπεί (ARG), αλλά και τον αριθμό των bit του επιθυμητού αποτελέσματος (SIZE). Παρα-

Πίνακας 4-38 Συνάρτηση VHDL για μετατροπή του τύπου STD_LOGIC _VECTOR σε INTEGER.

```
function CONV_INTEGER (X: STD_LOGIC_VECTOR) return
INTEGER is
   variable RESULT: INTEGER;
begin
   RESULT := 0;
   for i in X'range loop
       RESULT := RESULT * 2;
       case X(i) is
             when '0' | 'L' => null;
             when '1' | 'H' => RESULT := RESULT + 1;
             when others    => null;
       end case;
   end loop;
   return RESULT;
end CONV_INTEGER;
```

τηρήστε ότι η συνάρτηση δηλώνει την τοπική μεταβλητή "result", μια τιμή τύπου STD_LOGIC_VECTOR της οποίας το πεδίο τιμών των δεικτών εξαρτάται από το SIZE. Γι' αυτόν το λόγο, το SIZE πρέπει να είναι μια σταθερά ή άλλη τιμή η οποία είναι γνωστή όταν μεταγλωττίζεται το CONV_STD_LOGIC_VECTOR. Για να εκτελέσει τη μετατροπή, η συνάρτηση χρησιμοποιεί τον αλγόριθμο διαδοχικής διαίρεσης που περιγράψαμε στην Ενότητα 2.3.

διαδικασία

Μια *διαδικασία* (*procedure*) της VHDL μοιάζει με συνάρτηση, με τη διαφορά ότι η διαδικασία δεν επιστρέφει αποτέλεσμα. Ενώ μια κλήση συνάρτησης μπορεί να χρησιμοποιηθεί στη θέση μιας παράστασης, μια κλήση διαδικασίας μπορεί να χρησιμοποιηθεί στη θέση μιας εντολής. Οι διαδικασίες της VHDL επιτρέπουν να προσδιορίζονται τα ορίσματά τους ως τύπου out ή inout, έτσι στην πραγματικότητα οι διαδικασίες μπορούν να "επιστρέφουν" αποτέλεσμα. Ωστόσο, στο υπόλοιπο βιβλίο δε χρησιμοποιούμε διαδικασίες VHDL και έτσι δε θα συζητήσουμε περισσότερο γι' αυτές.

Πίνακας 4-39 Συνάρτηση VHDL για τη μετατροπή από INTEGER σε STD_LOGIC _VECTOR.

```
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
       return STD_LOGIC_VECTOR is
    variable result: STD_LOGIC_VECTOR (SIZE-1 downto 0);
    variable temp: integer;
begin
    temp := ARG;
    for i in 0 to SIZE-1 loop
       if (temp mod 2) = 1 then result(i) := '1';
       else result(i) := '0';
       end if;
       temp := temp / 2;
    end loop;
    return result;
end;
```

4.7.5 Βιβλιοθήκες και πακέτα

βιβλιοθήκη

Η *βιβλιοθήκη* (*library*) της VHDL είναι ένα μέρος όπου ο μεταγλωττιστής VHDL αποθηκεύει πληροφορίες για ένα συγκεκριμένο σχεδιαστικό έργο, συμπεριλαμβανομένων των ενδιάμεσων αρχείων που χρησιμοποιούνται στην ανάλυση, την προσομοίωση, και τη σύνθεση της σχεδίασης. Η θέση της βιβλιοθήκης μέσα στο σύστημα αρχείων του υπολογιστή υπηρεσίας εξαρτάται από την υλοποίηση. Για μια δεδομένη σχεδίαση σε VHDL, ο μεταγλωττιστής δημιουργεί αυτόματα και χρησιμοποιεί μια βιβλιοθήκη που ονομάζεται "`work`".

Ένα πλήρες σχεδιαστικό έργο σε VHDL έχει συνήθως πολλά αρχεία, κάθε ένα από τα οποία περιέχει διαφορετικές σχεδιαστικές μονάδες που περιλαμβάνουν οντότητες και αρχιτεκτονικές. Όταν ο μεταγλωττιστής VHDL αναλύει κάθε αρχείο της σχεδίασης, τοποθετεί τα αποτελέσματα στη βιβλιοθήκη "`work`" στην οποία επίσης αναζητά και τους απαιτούμενους ορισμούς, όπως π.χ. άλλες οντότητες. Λόγω αυτού του χαρακτηριστικού, ένα μεγάλο σχεδιαστικό έργο μπορεί να διαιρεθεί σε πολλά αρχεία και παρόλα αυτά ο μεταγλωττιστής να μπορεί να βρει τις εξωτερικές αναφορές που χρειάζεται.

Δεν είναι δυνατόν να περιλαμβάνονται όλες οι απαιτούμενες πληροφορίες της σχεδίασης στη βιβλιοθήκη "`work`". Για παράδειγμα, ένας σχεδιαστής μπορεί να βασίζεται σε κοινούς ορισμούς ή λειτουργικές μονάδες που περιλαμβάνονται σε μια οικογένεια από διάφορα έργα. Κάθε έργο έχει τη δική του βιβλιοθήκη "`work`" (συνήθως έναν υποκατάλογο μέσα στο συνολικό κατάλογο του εκάστοτε έργου), αλλά πρέπει επίσης να αναφέρεται σε μια κοινή βιβλιοθήκη που περιέχει τους κοινόχρηστους ορισμούς. Ακόμη και μικρά έργα είναι δυνατόν να χρησιμοποιούν μια τυποποιημένη βιβλιοθήκη όπως αυτή που περιέχει τους καθιερωμένους ορισμούς κατά IEEE. Ο σχεδιαστής μπορεί να ορίζει το όνομα μιας τέτοιας βιβλιοθήκης χρησιμοποιώντας μια *φράση* `library` στην αρχή του αρχείου της σχεδίασης. Για παράδειγμα, μπορούμε να καθορίσουμε τη βιβλιοθήκη IEEE ως εξής:

φράση `library`

```
library ieee;
```

Η φράση "`library work;`" περιλαμβάνεται ρητά στην αρχή κάθε αρχείου σχεδίασης VHDL.

Ο καθορισμός ενός ονόματος βιβλιοθήκης σε μια σχεδίαση παρέχει πρόσβαση σε όλες τις οντότητες που έχουν αναλυθεί στο παρελθόν και είναι αποθηκευμένες στη βιβλιοθήκη, αλλά δεν παρέχει πρόσβαση στους ορισμούς τύπων και σε παρόμοιες πληροφορίες. Αυτό είναι δουλειά των "πακέτων" και των "φράσεων `use`", που περιγράφονται παρακάτω.

πακέτο

Ένα *πακέτο* (*package*) της VHDL είναι ένα αρχείο που περιέχει ορισμούς αντικειμένων τα οποία μπορούν να χρησιμοποιούνται σε άλλα προγράμματα. Στα είδη των αντικειμένων τα οποία μπορούν να τοποθετούνται σε ένα πακέτο περιλαμβάνονται οι ορισμοί σημάτων, τύπων, σταθερών, συναρτήσεων, διαδικασιών, και στοιχείων.

Τα σήματα που ορίζονται σε ένα πακέτο είναι "καθολικά" σήματα, δηλαδή είναι διαθέσιμα σε κάθε οντότητα της VHDL που χρησιμοποιεί το πακέτο. Οι τύποι και οι σταθερές που ορίζονται στο πακέτο είναι γνωστά σε κάθε αρχείο που χρησιμοποιεί το πακέτο. Παρόμοια, οι συναρτήσεις και οι διαδικασίες που ορίζονται σε ένα πακέτο είναι δυνατόν να κληθούν μέσα από τα αρχεία που χρησιμοποιούν αυτό το πακέτο, ενώ τα στοιχεία (που περιγράφονται στην επόμενη ενότητα) είναι δυνατόν να "συγκεκριμενοποιηθούν" σε αρχιτεκτονικές που χρησιμοποιούν αυτό το πακέτο.

Μια σχεδίαση μπορεί να "χρησιμοποιήσει" ένα πακέτο συμπεριλαμβάνοντας τη *φράση* `use` στην αρχή του αρχείου του. Για παράδειγμα, για να χρησιμοποιήσουμε όλους τους ορισμούς που περιλαμβάνονται στο πρότυπο πακέτο IEEE 1164, πρέπει να γράψουμε:

φράση `use`

```
use ieee.std_logic_1164.all;
```

Εδώ, "`ieee`" είναι το όνομα μιας βιβλιοθήκης το οποίο έχει αποδοθεί από πριν με μια φράση `library`. Μέσα σε αυτή τη βιβλιοθήκη, το αρχείο που ονομάζεται "`std_logic_1164`" περιέχει τους επιθυμητούς ορισμούς. Η κατάληξη "`all`" υποδεικνύει στο μεταγλωττιστή ότι πρέπει να χρησιμοποιεί όλους τους ορισμούς που περιλαμβάνονται σε αυτό το αρχείο. Αντί για "`all`" μπορείτε να γράψετε το όνομα ενός συγκεκριμένου αντικειμένου, έτσι ώστε ο μεταγλωττιστής να χρησιμοποιεί μόνο τον ορισμό του, όπως για παράδειγμα:

```
use ieee_logic_1164.std_ulogic
```

Αυτή η εντολή καθιστά διαθέσιμο μόνο τον ορισμό του τύπου `std_ulogic` που συναντήσαμε στον Πίνακα 4-32, στην Ενότητα 4.7.3, χωρίς όλους τους σχετικούς τύπους και ορισμούς. Ωστόσο, μπορείτε να γράψετε πολλές φράσεις "`use`", για να χρησιμοποιούνται και άλλοι ορισμοί.

Ο ορισμός πακέτων δε γίνεται αποκλειστικά από φορείς προτυποποίησης. Οποιοσδήποτε μπορεί να γράψει ένα πακέτο, χρησιμοποιώντας τη σύνταξη που παρουσιάζεται στον Πίνακα 4-40. Όλα τα αντικείμενα που ορίζονται ανάμεσα στο "`package`" και στην πρώτη εντολή "`end`" είναι ορατά σε οποιοδήποτε αρχείο σχεδίασης που χρησιμοποιεί το συγκεκριμένο πακέτο. Τα αντικείμενα που ακολουθούν τη λέξη-κλειδί "`package body`" είναι τοπικά. Συγκεκριμένα, παρατηρήστε ότι το πρώτο μέρος περιλαμβάνει "δηλώσεις συναρτήσεων" και όχι ορισμούς. Μια *δήλωση συνάρτησης* αναφέρει μόνο το όνομα της συνάρτησης, τα ορίσματα, και τον τύπο, μέχρι — αλλά χωρίς να περιλαμβάνει — τη λέξη-κλειδί "`is`" του Πίνακα 4-35 που είδαμε στην Ενότητα 4.7.4. Ο πλήρης ορισμός της συνάρτησης δίνεται στον κορμό του πακέτου και δεν είναι ορατός στους χρήστες της συνάρτησης.

δήλωση συνάρτησης

ΠΡΟΤΥΠΑ VHDL ΚΑΤΑ IEEE

Η VHDL παρέχει εξαιρετικές δυνατότητες επέκτασης των τύπων δεδομένων και των συναρτήσεών της. Αυτό είναι σημαντικό, επειδή οι ενσωματωμένοι τύποι `BIT` και `BIT_VECTOR` της γλώσσας είναι στην πραγματικότητα εντελώς ανεπαρκείς για τη μοντελοποίηση πραγματικών κυκλωμάτων τα οποία χειρίζονται επίσης σήματα τριών καταστάσεων, άγνωστα, "αδιάφορα", ή μεταβλητής ισχύος.

Έτσι, λίγο μετά την τυποποίηση της γλώσσας στο πρότυπο IEEE 1076, οι εταιρίες άρχισαν να παρουσιάζουν δικούς τους ενσωματωμένους τύπους δεδομένων για να χειρίζονται λογικές τιμές εκτός των 0 και 1. Φυσικά, κάθε εταιρία παρείχε διαφορετικούς ορισμούς γι' αυτούς τους εκτεταμένους τύπους, με κίνδυνο να δημιουργηθεί ένας "Πύργος της Βαβέλ".

Για να αποφευχθεί αυτή η κατάσταση, το IEEE ανέπτυξε το καθιερωμένο λογικό πακέτο 1164 (`std_logic_1164`) με ένα σύστημα λογικής εννέα τιμών το οποίο ικανοποιεί τις ανάγκες των περισσότερων σχεδιαστών. Αργότερα, ακολούθησε το πρότυπο 1076-3, που περιγράφεται στην Ενότητα 5.9.6, το οποίο περιλαμβάνει πολλά πακέτα με καθιερωμένους τύπους και πράξεις για διανύσματα στοιχείων `STD_LOGIC` τα οποία ερμηνεύονται ως προσημασμένοι ή απρόσημοι ακέραιοι. Στα πακέτα αυτά περιλαμβάνονται τα `std_logic_arith`, `std_logic_signed`, και `std_logic_unsigned`.

Χρησιμοποιώντας τα πρότυπα IEEE, οι σχεδιαστές μπορούν να εξασφαλίσουν υψηλό βαθμό φορητότητας και διαλειτουργικότητας ανάμεσα στις σχεδιάσεις τους. Αυτό γίνεται όλο και πιο σημαντικό καθώς η ανάπτυξη πολύ μεγάλων διατάξεων ASIC απαιτεί συνεργασία όχι μόνο μεταξύ πολλών σχεδιαστών, αλλά και μεταξύ πολλών εταιριών οι οποίες είναι δυνατόν να συνεισφέρουν διαφορετικά μέρη της σχεδίασης ενός "συστήματος σε ένα τσιπ".

Πίνακας 4-40 Σύνταξη του ορισμού ενός πακέτου VHDL.

```
package  όνομα-πακέτου is
  δηλώσεις τύπων
  δηλώσεις σημάτων
  δηλώσεις σταθερών
  δηλώσεις στοιχείων
  δηλώσεις συναρτήσεων
  δηλώσεις διαδικασιών end όνομα-πακέτου;
package body όνομα-πακέτου is
  δηλώσεις τύπων
  δηλώσεις σταθερών
  ορισμοί συναρτήσεων
  ορισμοί διαδικασιών
end όνομα-πακέτου;
```

4.7.6 Στοιχεία δομικής σχεδίασης

Είμαστε τελικά έτοιμοι να δούμε τα ενδότερα μιας σχεδίασης VHDL, δηλαδή το "εκτελέσιμο" τμήμα μιας αρχιτεκτονικής. Ας ξαναθυμηθούμε από τον Πίνακα 4-28 της Ενότητας 4.7.2 ότι ο κορμός μιας αρχιτεκτονικής είναι μια σειρά από ταυτόχρονες εντολές. Στη VHDL, κάθε *ταυτόχρονη εντολή* (*concurrent statement*) εκτελείται συγχρόνως με τις άλλες ταυτόχρονες εντολές που βρίσκονται στον κορμό της ίδιας αρχιτεκτονικής.

ταυτόχρονη εντολή

Αυτή η συμπεριφορά είναι σαφώς διαφορετική από εκείνη των εντολών στις συμβατικές γλώσσες προγραμματισμού λογισμικού, όπου οι εντολές εκτελούνται σειριακά. Οι ταυτόχρονες εντολές είναι απαραίτητες για την προσομοίωση της συμπεριφοράς του υλικού, όπου τα συνδεδεμένα στοιχεία επηρεάζουν τα άλλα διαρκώς, και όχι απλώς σε συγκεκριμένα και διατεταγμένα χρονικά βήματα. Έτσι, στον κορμό μιας αρχιτεκτονικής VHDL, αν η τελευταία εντολή ενημερώνει ένα σήμα το οποίο χρησιμοποιείται από την πρώτη εντολή, τότε ο προσομοιωτής θα επιστρέψει σε αυτή την πρώτη εντολή και θα ενημερώσει τα αποτελέσματά της σύμφωνα με το σήμα που μόλις άλλαξε. Στην πραγματικότητα, ο προσομοιωτής μεταδίδει συνεχώς τις αλλαγές και ενημερώνει τα αποτελέσματα έως ότου σταθεροποιηθεί το προσομοιωμένο κύκλωμα. Αυτό θα το συζητήσουμε με περισσότερες λεπτομέρειες στην Ενότητα 4.7.9.

Η VHDL έχει πολλές διαφορετικές ταυτόχρονες εντολές, καθώς και ένα μηχανισμό για το πακετάρισμα ενός συνόλου σειριακών εντολών οι οποίες θα εκτελούνται ως μία ενιαία ταυτόχρονη εντολή. Αυτές οι εντολές, χρησιμοποιούμενες με διάφορους τρόπους, δημιουργούν τρία κάπως ξεχωριστά μεταξύ τους στιλ περιγραφής και σχεδίασης κυκλωμάτων, τα οποία καλύπτουμε σε αυτή και στις δύο επόμενες ενότητες.

Οι πιο βασικές από τις ταυτόχρονες εντολές της VHDL είναι η εντολή `component`, της οποίας η βασική σύνταξη φαίνεται στον Πίνακα 4-41. Εδώ, το *όνομα-στοιχείου* είναι το όνομα μιας οντότητας που ορίστηκε στο παρελθόν και η οποία πρόκειται να χρησιμοποιηθεί, ή αλλιώς να *συγκεκριμενοποιηθεί* (*instantiated*), στον κορμό της τρέχουσας αρχιτεκτονικής. Δημιουργείται μια παρουσία της κατονομαζόμενης οντότητας για κάθε εντολή `component` που καλεί το όνομά της, ενώ κάθε παρουσία πρέπει να ονομάζεται με μία μοναδική *ετικέτα* (*label*).

συγκεκριμενοποίηση

Οι λέξεις-κλειδιά `port map` (αντιστοιχία θυρών) εισάγουν μια λίστα που συσχετίζει τις θύρες της κατονομαζόμενης οντότητας με σήματα της τρέχουσας αρχιτεκτονικής. Η λίστα μπορεί να γραφεί με δύο διαφορετικά στιλ. Το πρώτο είναι ένα στιλ θέσης: όπως και στις συμβατικές γλώσσες προγραμματισμού, τα σήματα της λίστας συσχετίζονται με τις θύρες της οντότητας με την ίδια σειρά με την οποία εμφανίζονται στον ορισμό της οντότητας. Το δεύτερο είναι ένα ρητό στιλ: κάθε μία από τις θύρες της οντότητας συνδέεται με ένα σήμα με τη βοήθεια του τελεστή "=>", ενώ οι συσχετίσεις είναι δυνατόν να γράφονται με οποιαδήποτε σειρά.

`port map`

Πριν συγκεκριμενοποιηθεί ένα στοιχείο σε μια αρχιτεκτονική, θα πρέπει να δηλωθεί με μια *δήλωση στοιχείου* στον ορισμό της αρχιτεκτονικής

δήλωση στοιχείου

Πίνακας 4-41
Σύνταξη της εντολής component στη VHDL.

ετικέτα: όνομα-στοιχείου `port map` *(σήμα1, σήμα2, ..., σήμαn)*;

ετικέτα: όνομα-στοιχείου `port map` *(θύρα1=>σήμα1, θύρα2=> σήμα2, ..., θύρα=>σήμα)*;

Πίνακας 4-42
Σύνταξη μιας δήλωσης στοιχείου στη VHDL.

```
component  όνομα-στοιχείου

  port  (ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος;
        ονόματα-σημάτων: τρόπος-λειτουργίας τύπος-σήματος;
        ...
        ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος);
end component;
```

(δείτε τον Πίνακα 4-28, στην Ενότητα 4.7.2). Όπως φαίνεται στον Πίνακα 4-42, μια δήλωση στοιχείου είναι ουσιαστικά ίδια με το τμήμα δήλωσης θυρών στην αντίστοιχη δήλωση της οντότητας, δηλαδή αναφέρει το όνομα, τον τρόπο λειτουργίας, και τον τύπο κάθε μίας από τις θύρες της.

Τα στοιχεία που χρησιμοποιούνται σε μια αρχιτεκτονική μπορεί να είναι εκείνα που ορίστηκαν προηγουμένως στα πλαίσια μιας σχεδίασης, ή μπορεί να είναι μέρος μιας βιβλιοθήκης. Ο Πίνακας 4-43 είναι παράδειγμα μιας οντότητας VHDL και της αρχιτεκτονικής της που χρησιμοποιεί στοιχεία, ένας "ανιχνευτής πρώτων αριθμών" που είναι δομικά πανομοιότυπος με το κύκλωμα σε επίπεδο πυλών που είδαμε στην Εικόνα 4-30(γ), στην Ενότητα 4.3.5. Η δήλωση της οντότητας κατονομάζει τις εισόδους και την έξοδο του κυκλώματος. Το τμήμα δηλώσεων της αρχιτεκτονικής ορίζει όλα τα ονόματα σημάτων και τα στοιχεία τα οποία χρησιμοποιούνται εσωτερικά. Τα στοιχεία `INV`, `AND2`, `AND3`, και `AND4` είναι προκαθορισμένα στο περιβάλλον σχεδίασης στο οποίο δημιουργήθηκε και μεταγλωττίστηκε το συγκεκριμένο παράδειγμα (Xilinx Foundation 1.5, δείτε τις Παραπομπές).

Παρατηρήστε ότι οι εντολές στοιχείων του Πίνακα 4-43 εκτελούνται *ταυτόχρονα*. Ακόμη και αν οι εντολές γράφονταν με διαφορετική σειρά, θα συνετίθετο το ίδιο κύκλωμα και η λειτουργία του προσομοιωμένου κυκλώματος θα ήταν η ίδια.

δομική περιγραφή
δομική σχεδίαση

Μια αρχιτεκτονική VHDL η οποία χρησιμοποιεί στοιχεία λέγεται συνήθως *δομική περιγραφή* ή *δομική σχεδίαση*, επειδή ορίζει την ακριβή δομή της αλληλοσύνδεσης των σημάτων και των οντοτήτων που υλοποιούν την οντότητα. Από αυτή την άποψη, μια καθαρή δομική περιγραφή είναι ισοδύναμη με ένα σχηματικό διάγραμμα ή με μια λίστα δικτύου για το κύκλωμα.

Σε μερικές εφαρμογές είναι απαραίτητο να δημιουργήσουμε πολλά αντίγραφα μια συγκεκριμένης δομής μέσα στα πλαίσια μιας αρχιτεκτονικής. Για παράδειγμα, θα δούμε στην Ενότητα 5.10.2 ότι ένας "αθροιστής κυμάτωσης" των *n* bit μπορεί να δημιουργηθεί με αλυσιδωτή σύνδεση *n* "πλήρων αθροιστών". Η VHDL περιλαμβάνει την *εντολή* `generate` που σας επιτρέπει να δημιουργείτε τέτοιες επαναληπτικές δομές χρησιμοποιώντας ένα είδος "βρόχου for", χωρίς να χρειάζεται να γράψετε ξεχωριστά όλες τις συγκεκριμενοποιήσεις των στοιχείων.

Στον Πίνακα 4-44 παρουσιάζεται η σύνταξη ενός απλού επαναληπτικού βρόχου `generate`. Το *αναγνωριστικό* ορίζεται ρητά ως μεταβλητή με τύπο συμβατό με το *πεδίο τιμών*. Η *ταυτόχρονη εντολή* εκτελείται μία φορά για κάθε δυνατή τιμή του *αναγνωριστικού* μέσα στο πεδίο τιμών, ενώ το *αναγνωριστικό* μπορεί να χρησιμοποιηθεί μέσα στην ταυτόχρονη εντολή. Για παράδειγμα, ο Πίνακας 4-45 δείχνει πώς μπορεί να δημιουργηθεί ένας αντιστροφέας των 8 bit.

Η τιμή μιας σταθεράς πρέπει να είναι γνωστή τη στιγμή που μεταγλωττίζεται το πρόγραμμα VHDL. Σε πολλές εφαρμογές, είναι χρήσιμο να σχεδιάζετε και να μεταγλωττίζετε μια οντότητα και την αρχιτεκτονι-

Πίνακας 4-43 Δομικό πρόγραμμα VHDL για ανιχνευτή πρώτων αριθμών.

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

entity prime is
   port ( N: in STD_LOGIC_VECTOR (3 downto 0);
          F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
   U1: INV port map (N(3), N3_L);
   U2: INV port map (N(2), N2_L);
   U3: INV port map (N(1), N1_L);
   U4: AND2 port map (N3_L, N(0), N3L_N0);
   U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
   U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
   U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
   U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

Πίνακας 4-44
Σύνταξη βρόχου for-generate στη VHDL.

```
ετικέτα: for αναγνωριστικό in πεδίο-τιμών generate
            ταυτόχρονη-εντολή
        end generate;
```

Πίνακας 4-45
Οντότητα και αρχιτεκτονική της VHDL για έναν αντιστροφέα των 8 bit.

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

entity inv8 is
   port       (X: in STD_LOGIC_VECTOR (1 to 8);
               Y: out STD_LOGIC_VECTOR (1 to 8) );
end inv8;

architecture inv8_arch of inv8 is
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
   g1: for b in 1 to 8 generate
          U1: INV port map (X(b), Y(b));
         end generate;
end inv8_arch;
```

Πίνακας 4-46
Σύνταξη γενικής δήλωσης VHDL μέσα σε μια δήλωση οντότητας.

```
      entity όνομα-οντότητας is
   generic (ονόματα-σταθερών : τύπος-σταθεράς;
            ονόματα-σταθερών : τύπος-σταθεράς;
            ...
            ονόματα-σταθερών : τύπος-σταθεράς);
   port (ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος;
         ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος;
         ...
         ονόματα-σημάτων : τρόπος-λειτουργίας τύπος-σήματος);
end όνομα-οντότητας;
```

κή της ενώ ταυτόχρονα να αφήνετε απροσδιόριστες μερικές από τις παραμέτρους της, όπως το εύρος του διαύλου. Αυτό μπορείτε να το κάνετε με τη φράση "generic" της VHDL.

γενική σταθερά
γενική δήλωση

Μια η περισσότερες *γενικές σταθερές* είναι δυνατόν να οριστούν σε μια δήλωση οντότητας με μια *γενική δήλωση* πριν από τη δήλωση θυρών, με τη χρήση της σύνταξης που παρατίθεται στον Πίνακα 4-46. Κάθε μία από τις κατονομασμένες σταθερές μπορεί να χρησιμοποιηθεί μέσα στον ορισμό αρχιτεκτονικής της οντότητας, ενώ η τιμή της σταθεράς αναβάλλεται χρονικά έως ότου συγκεκριμενοποιηθεί η οντότητα με μια εντολή στοιχείου (component statement) μέσα σε μια άλλη αρχιτεκτονική. Μέ-

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

entity businv is
   generic (WIDTH: positive);
   port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
          Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end businv;

architecture businv_arch of businv is
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
   g1: for b in WID-1 downto 0 generate
          U1: INV port map (X(b), Y(b));
        end generate;
end businv_arch;
```

Πίνακας 4-47 Οντότητα και αρχιτεκτονική της VHDL για έναν αντιστροφέα διαύλου οποιουδήποτε εύρους.

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

entity businv_example is
   port ( IN8: in STD_LOGIC_VECTOR (7 downto 0);
          OUT8: out STD_LOGIC_VECTOR (7 downto 0);
          IN16: in STD_LOGIC_VECTOR (15 downto 0);
          OUT16: out STD_LOGIC_VECTOR (15 downto 0);
          IN32: in STD_LOGIC_VECTOR (31 downto 0);
          OUT32: out STD_LOGIC_VECTOR (31 downto 0) );
end businv_example;

architecture businv_ex_arch of businv_example is
component businv
   generic (WIDTH: positive);
   port (X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
         Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) ); end component;
begin
U1: businv generic map (WIDTH=>8) port map (IN8, OUT8);
U2: businv generic map (WIDTH=>16) port map (IN16, OUT16);
U3: businv generic map (WIDTH=>32) port map (IN32, OUT32);
end businv_ex_arch;
```

Πίνακας 4-48 Οντότητα και αρχιτεκτονική της VHDL η οποία χρησιμοποιεί τον αντιστροφέα διαύλου οποιουδήποτε εύρους.

σα στη συγκεκριμένη εντολή στοιχείου, αποδίδονται τιμές στις γενικές σταθερές με προτάσεις *generic map* στο ίδιο στιλ με την εντολή `port map`. Ο Πίνακας 4-47 είναι ένα παράδειγμα συνδυασμού των προτάσεων `generic` και `generate` για τον ορισμό ενός "αντιστροφέα διαύλου" με εύρος οριζόμενο από το χρήστη. Πολλά αντίγραφα αυτού του αντιστροφέα, κάθε ένα με διαφορετικό εύρος, συγκεκριμενοποιούνται στο πρόγραμμα του Πίνακα 4-48.

generic map

4.7.7 Στοιχεία σχεδίασης ροής δεδομένων

Αν οι εντολές στοιχείων ήταν οι μόνες ταυτόχρονες εντολές, τότε η VHDL θα ήταν απλώς κάτι παραπάνω από μια ιεραρχική γλώσσα περιγραφής λίστας δικτύου με αυστηρή δήλωση τύπων. Οι αρκετές πρόσθετες ταυτόχρονες εντολές που υπάρχουν επιτρέπουν στη VHDL να περιγράφει ένα κύκλωμα σύμφωνα με τη ροή δεδομένων και τις πράξεις που εκτελούνται σε αυτά μέσα στο κύκλωμα. Αυτό το στιλ λέγεται *περιγραφή ροής δεδομένων* ή *σχεδίαση ροής δεδομένων*.

περιγραφή ροής δεδομένων
σχεδίαση ροής δεδομένων

Στον Πίνακα 4-49 απεικονίζονται δύο ακόμα ταυτόχρονες εντολές που χρησιμοποιούνται στις σχεδιάσεις ροής δεδομένων. Η πρώτη από αυτές είναι η πιο συχνά χρησιμοποιούμενη και λέγεται *εντολή ταυτόχρονης ανάθεσης τιμής σήματος*. Η εντολή αυτή διαβάζεται ως εξής: "το *όνομα-σήματος* λαμβάνει την τιμή *παράσταση*". Λόγω της αυστηρής δήλωσης τύπων της VHDL, ο τύπος της *παράστασης* πρέπει να είναι συμβατός με εκείνον του *ονόματος-σήματος*. Γενικά, αυτό σημαίνει ότι και οι δύο τύποι πρέπει να είναι ταυτόσημοι ή ο τύπος της *παράστασης* πρέπει να είναι υποτύπος του *ονόματος-σήματος*. Στην περίπτωση πινάκων, τόσο ο τύπος όσο και το μήκος των στοιχείων πρέπει να συμφωνούν μεταξύ τους. Ωστόσο, δεν απαιτείται συμφωνία ως προς το πεδίο τιμών των δεικτών και την κατεύθυνση.

εντολή ταυτόχρονης ανάθεσης τιμής σήματος

Ο Πίνακας 4-50 παρουσιάζει μια αρχιτεκτονική για την οντότητα ανίχνευσης πρώτων αριθμών (Πίνακας 4-43, στην Ενότητα 4.7.6), γραμμένη σε στιλ ροής δεδομένων. Σε αυτό το στιλ, δεν εμφανίζουμε ρητά τις πύλες και τις συνδέσεις τους. Αντίθετα, χρησιμοποιούμε τους ενσωματωμένους τελεστές `and`, `or`, και `not` της VHDL. (Στην πραγματικότητα, αυτοί οι τελεστές δεν είναι ενσωματωμένοι τελεστές για τα σήματα τύπου `STD_LOGIC`, αλλά ορίζονται και υπερφορτώνονται από το πακέτο IEEE 1164.) Παρατηρήστε ότι ο τελεστής `not` έχει την υψηλότερη προτεραιότητα, και έτσι δεν απαιτούνται παρενθέσεις γύρω από τις δευτερεύουσες παραστάσεις όπως "`not N(3)`" για να ληφθεί το αναμενόμενο αποτέλεσμα.

Μπορούμε επίσης να χρησιμοποιήσουμε τη δεύτερη, δηλαδή την *υπό συνθήκη*, μορφή της εντολής ταυτόχρονης ανάθεσης τιμής σήματος, χρησιμοποιώντας τις λέξεις-κλειδιά `when` και `else` όπως φαίνεται στον Πίνακα 4-49. Εδώ, μια *λογική-παράσταση* συνδυάζει επιμέρους λογικούς όρους χρησιμοποιώντας τους ενσωματωμένους λογικούς τελεστές της

εντολή υπό συνθήκη ταυτόχρονης ανάθεσης τιμής σήματος
`when`
`else`

Πίνακας 4-49 Σύνταξη εντολών ταυτόχρονης ανάθεσης τιμής σήματος στη VHDL.

όνομα-σήματος <=	*παράσταση*;
όνομα-σήματος <=	*παράσταση* `when` *λογική-παράσταση* `else` *παράσταση* `when` *λογική-παράσταση* `else` `...` *παράσταση* `when` *λογική-παράσταση* `else` *παράσταση*;

Πίνακας 4-50 Αρχιτεκτονική ροής δεδομένων VHDL για τον ανιχνευτή πρώτων αριθμών.

```
architecture prime2_arch of prime is
signal N3L_N0; N3L_N2L_N1; N2L_N1_N0; N2_N1L_N0: STD_LOGIC;
begin
   N3L_N0 <= not N(3) and N(0);
   N3L_N2L_N1 <= not N(3) and not N(2) and N(1) ;
   N2L_N1_N0 <= not N(2) and N(1) and N(0);
   N2_N1L_N0 <= N(2) and not N(1) and N(0);
   F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

VHDL, όπως οι `and`, `or`, και `not`. Οι λογικοί όροι είναι συνήθως λογικές μεταβλητές ή αποτελέσματα συγκρίσεων με τη χρήση των *σχεσιακών τελεστών* =, /= (όχι ίσο), >, >=, < και <=.

σχεσιακοί τελεστές =, /=, >, >=, <, <=

Ο Πίνακας 4-51 είναι ένα παράδειγμα που χρησιμοποιεί εντολές υπό συνθήκη ταυτόχρονης ανάθεσης τιμής. Κάθε μία από τις συγκρίσεις ενός επιμέρους bit `STD_LOGIC`, όπως το `N(3)`, γίνεται ως προς ένα λεκτικό χαρακτήρα `'0'` ή `'1'` και επιστρέφει μια τιμή τύπου `boolean`. Αυτά τα αποτελέσματα σύγκρισης συνδυάζονται μαζί στη λογική παράσταση ανάμεσα στις λέξεις-κλειδιά `when` και `else` κάθε εντολής. Γενικά, οι εντολές `else` είναι υποχρεωτικές, επειδή το συνδυασμένο σύνολο συνθηκών σε μια απλή εντολή πρέπει να καλύπτει όλους τους δυνατούς συνδυασμούς εισόδων.

Ένα άλλο είδος προτάσεων ταυτόχρονης ανάθεσης τιμής είναι η *εντολή επιλεγμένης ανάθεσης τιμής σήματος*, της οποίας η σύνταξη φαίνεται στον Πίνακα 4-52. Αυτή η εντολή υπολογίζει τη δεδομένη *παράσταση* και, όταν η τιμή συμφωνεί με τις *επιλογές*, αποδίδει την αντίστοιχη *τιμή-σήματος* στο *όνομα-σήματος*. Οι *επιλογές* κάθε εντολής είναι δυνατόν να είναι μια απλή τιμή για την *παράσταση* ή μια λίστα τιμών που διαχωρίζονται από κατακόρυφες γραμμές (|). Οι *επιλογές* για ολόκληρη την εντολή πρέπει να είναι αλληλοαποκλειόμενες και να καλύπτουν όλες τις πε-

εντολή επιλεγμένης ανάθεσης τιμής σήματος

Πίνακας 4-51 Αρχιτεκτονική ανιχνευτή πρώτων αριθμών με χρήση υπό συνθήκη ανάθεσης τιμής.

```
architecture prime3_arch of prime is
signal N3L_N0; N3L_N2L_N1; N2L_N1_N0; N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0        <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1    <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0     <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0     <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Πίνακας 4-52 Σύνταξη εντολής επιλεγμένης ανάθεσης τιμής σήματος στη VHDL.

```
with παράσταση select
   όνομα-σήματος <= τιμή-σήματος when επιλογές,
                    τιμή-σήματος when επιλογές,
                    ...
                    τιμή-σήματος when επιλογές;
```

others

ριπτώσεις. Η λέξη-κλειδί `others` μπορεί να χρησιμοποιηθεί στην τελευταία εντολή `when` για να υποδηλώσει όλες τις τιμές για την *παράσταση* που δεν έχουν καλυφθεί μέχρι εκείνη τη στιγμή.

Ο Πίνακας 4-53 είναι μια αρχιτεκτονική για τον ανιχνευτή πρώτων αριθμών που χρησιμοποιεί μια εντολή επιλεγμένης ανάθεσης τιμής σήματος. Όλες οι *επιλογές* για τις οποίες η `F` είναι `'1'` θα μπορούσαν να έχουν γραφεί σε μία και μοναδική εντολή `when`, αλλά εδώ χρησιμοποι-

Πίνακας 4-53 Αρχιτεκτονική ανιχνευτή πρώτων αριθμών με χρήση επιλεγμένης ανάθεσης τιμής σήματος.

```
architecture prime4_arch of prime is
begin
    with N select
       F <= '1' when "0001",
            '1' when "0010",
            '1' when "0011" | "0101" | "0111",
            '1' when "1011" | "1101",
            '0' when others;
end prime4_arch;
```


ΚΑΛΥΨΗ ΟΛΩΝ ΤΩΝ ΠΕΡΙΠΤΩΣΕΩΝ

Τόσο η υπό συνθήκη όσο και η επιλεγμένη ανάθεση τιμής σήματος απαιτούν να καλύπτονται όλες οι δυνατές συνθήκες. Σε μια υπό συνθήκη ανάθεση τιμής σήματος, το τελικό "`else` *παράσταση*" καλύπτει τις απούσες συνθήκες. Στην επιλεγμένη ανάθεση τιμής σήματος, μπορεί να χρησιμοποιηθεί η λέξη "`others`" στην τελική εντολή `when` για να πάρουμε τις συνθήκες που απομένουν.

Στον Πίνακα 4-53, μπορεί να σκεφτείτε ότι αντί να γράψουμε "`others`" στην τελική εντολή `when` θα μπορούσαμε να είχαμε γράψει τους υπόλοιπους συνδυασμούς των 4 bit, δηλαδή "0000", "0100" κ.λπ. Όμως, δεν είναι έτσι! Θυμηθείτε ότι ο τύπος `STD_LOGIC` είναι ένα σύστημα εννέα τιμών, και επομένως μια τιμή `STD_LOGIC_VECTOR` των 4 bit στην πραγματικότητα έχει 9^4 δυνατές τιμές. Έτσι το "`others`" σε αυτό το παράδειγμα καλύπτει στην πραγματικότητα 6.554 περιπτώσεις!

Πίνακας 4-54 Περιγραφή του ανιχνευτή πρώτων αριθμών που βασίζεται περισσότερο στη συμπεριφορά.

```
architecture prime5_arch of prime is
begin
     with CONV_INTEGER(N) select
      F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 |13,
           '0' when others;
end prime5_arch;
```

ούνται πολλές εντολές μόνο και μόνο για εκπαιδευτικούς λόγους. Σε αυτό το παράδειγμα, η εντολή επιλεγμένης ανάθεσης τιμής σήματος έχει τη μορφή λίστας του συνόλου ενεργοποίησης της συνάρτησης F.

Μπορούμε να τροποποιήσουμε ελαφρώς την προηγούμενη αρχιτεκτονική για να έχουμε το πλεονέκτημα της αριθμητικής ερμηνείας του N στον ορισμό της συνάρτησης. Με χρήση της συνάρτησης CONV_INTEGER που ορίσαμε πριν, ο Πίνακας 4-54 παραθέτει τις *επιλογές* με τη μορφή ακεραίων, οι οποίοι φαίνεται αμέσως ότι είναι πρώτοι, όπως απαιτείται. Μπορούμε να θεωρήσουμε αυτή την εκδοχή της αρχιτεκτονικής ως μια περιγραφή "συμπεριφοράς", επειδή περιγράφει την επιθυμητή συνάρτηση με τέτοιον τρόπο ώστε η συμπεριφορά της να είναι εντελώς προφανής.

4.7.8 Στοιχεία σχεδίασης σύμφωνα με τη συμπεριφορά

Όπως είδαμε στο τελευταίο παράδειγμα, μερικές φορές μπορούμε να περιγράψουμε απευθείας μια επιθυμητή συμπεριφορά λογικού κυκλώματος χρησιμοποιώντας μια ταυτόχρονη εντολή. Αυτό είναι καλό, καθώς η δυνατότητα δημιουργίας μιας *σχεδίασης σύμφωνα με τη συμπεριφορά* (*behavioral design*) ή μιας *περιγραφής σύμφωνα με τη συμπεριφορά* (*behavioral description*) είναι ένα από βασικά πλεονεκτήματα των γλωσσών περιγραφής υλικού γενικά και της VHDL ειδικότερα. Ωστόσο, για τις περισσότερες περιγραφές συμπεριφοράς, χρειάζεται να χρησιμοποιήσουμε μερικά πρόσθετα στοιχεία της γλώσσας που περιγράφονται σε αυτή την ενότητα.

σχεδίαση σύμφωνα με τη συμπεριφορά
περιγραφή σύμφωνα με τη συμπεριφορά

Το βασικό στοιχείο περιγραφής της συμπεριφοράς στη VHDL είναι η "διεργασία". *Διεργασία* (*process*) είναι μια συλλογή από "σειριακές" εντολές (που περιγράφονται παρακάτω) οι οποίες εκτελούνται ταυτόχρονα με άλλες ταυτόχρονες εντολές και άλλες διεργασίες. Χρησιμοποιώντας μια διεργασία, μπορείτε να προδιαγράφετε μια σύνθετη αλληλεπίδραση σημάτων και συμβάντων με τρόπο ώστε να εκτελείται ουσιαστικά σε μηδενικό προσομοιωμένο χρόνο κατά την προσομοίωση, και αυτό έχει αποτέλεσμα τη σύνθεση ενός συνδυαστικού ή ακολουθιακού κυκλώματος που εκτελεί άμεσα τη μοντελοποιημένη λειτουργία του.

διεργασία

εντολή διεργασίας

`process`

Μια *εντολή διεργασίας* της VHDL μπορεί να χρησιμοποιηθεί οπουδήποτε μπορεί να χρησιμοποιηθεί μια ταυτόχρονη εντολή. Η εντολή διεργασίας εισάγεται με τη λέξη-κλειδί `process` και έχει τη σύνταξη που φαίνεται στον Πίνακα 4-55. Εφόσον η εντολή διεργασίας χρησιμοποιείται μέσα στα πλαίσια μιας περιβάλλουσας αρχιτεκτονικής, έχει πρόσβαση στους τύπους, τα σήματα, τις σταθερές, τις συναρτήσεις και τις διαδικασίες που έχουν δηλωθεί ή είναι με άλλο τρόπο ορατές στην περιβάλλουσα αρχιτεκτονική. Ωστόσο, μπορείτε επίσης να ορίσετε τύπους, σήματα, σταθερές, συναρτήσεις, και διαδικασίες που να είναι τοπικές στη διεργασία.

μεταβλητή

Παρατηρήστε ότι μια διεργασία μπορεί να μη δηλώνει σήματα, παρά μόνο "μεταβλητές". Μια *μεταβλητή* VHDL παρακολουθεί την κατάσταση μέσα σε μια διεργασία και δεν είναι ορατή έξω από τη διεργασία αυτή. Ανάλογα με τη χρήση, μπορεί ή δεν μπορεί να δώσει ως αποτέλεσμα ένα αντίστοιχο σήμα σε μια φυσική υλοποίηση του μοντελοποιημένου κυκλώματος. Η σύνταξη ενός ορισμού μεταβλητής μέσα σε μια διεργασία μοιάζει με τη σύνταξη μιας δήλωσης σήματος μέσα σε μια αρχιτεκτονική, με τη διαφορά ότι χρησιμοποιείται η λέξη-κλειδί `variable`:

`variable`

`variable` *ονόματα-μεταβλητών* : *τύπος-μεταβλητών*;

εκτελούμενη διεργασία
διεργασία σε αναστολή
λίστα ευαισθησίας

Μια διεργασία της VHDL είναι πάντα είτε *εκτελούμενη* (*running*) είτε *σε αναστολή* (*suspended*). Η λίστα των σημάτων στον ορισμό της διεργασίας λέγεται *λίστα ευαισθησίας* (*sensitivity list*) και καθορίζει πότε εκτελείται η διεργασία. Η διεργασία είναι αρχικά σε αναστολή. Μόλις ένα σήμα της λίστας ευαισθησίας αλλάξει τιμή, η διεργασία συνεχίζει την εκτέλεσή της, ξεκινώντας με την πρώτη σειριακή εντολή της και συνεχίζοντας μέχρι το τέλος. Αν αλλάξει η τιμή σε οποιοδήποτε σήμα της λίστας ευαισθησίας εξαιτίας της εκτέλεσης της διεργασίας, η διεργασία εκτελείται ξανά. Αυτό συνεχίζεται έως ότου η διεργασία να εκτελείται χωρίς να αλλάζει τιμή κάποιο από τα σήματα. Στην προσομοίωση, όλα αυτά συμβαίνουν σε μηδενικό προσομοιωμένο χρόνο.

Αμέσως μετά τη συνέχιση της εκτέλεσης, μία σωστά γραμμένη διεργασία θα τεθεί σε αναστολή μετά από μία ή λίγες εκτελέσεις. Ωστόσο, είναι δυνατόν να γράψετε μια εσφαλμένη διεργασία η οποία δεν θα τίθεται ποτέ σε αναστολή. Για παράδειγμα, υποθέστε ότι μια διεργασία έχει μία μόνο σειριακή εντολή, "`X <= not X`", και μια λίστα ευαισθησίας "`(X)`". Εφόσον το `X` αλλάζει σε κάθε πέρασμα, η διεργασία θα εκτελείται για πάντα σε μηδενικό προσομοιωμένο χρόνο, κάτι όχι πολύ χρήσιμο! Στην πράξη, οι προσομοιωτές έχουν προστασίες που κανονικά μπορούν να εντοπίσουν τέτοια ανεπιθύμητη συμπεριφορά, τερματίζοντας την εκτέλεση της διεργασίας που παρουσιάζει κακή συμπεριφορά μετά από περίπου χίλια περάσματα.

Η λίστα ευαισθησίας είναι προαιρετική. Μια διεργασία χωρίς λίστα ευαισθησίας αρχίζει να εκτελείται σε χρόνο μηδέν σε προσομοίωση. Μια εφαρμογή μιας τέτοιας διεργασίας είναι η παραγωγή κυματομορφών ει-

Πίνακας 4-55 Σύνταξη μιας εντολής διεργασίας της VHDL.

```
process (όνομα-σήματος, όνομα-σήματος, ..., όνομα-σήματος)
   δηλώσεις τύπων
   δηλώσεις μεταβλητών
   δηλώσεις σταθερών
   ορισμοί συναρτήσεων
   ορισμοί διαδικασιών
begin
   ταυτόχρονη-εντολή
   ...
   ταυτόχρονη-εντολή
end process;
```

σόδου σε ένα περιβάλλον δοκιμών, όπως στον Πίνακα 4-65 της Ενότητας 4.7.9.

Η VHDL έχει αρκετά είδη σειριακών εντολών. Το πρώτο είναι η *εντολή σειριακής ανάθεσης τιμής σήματος*. Έχει την ίδια σύνταξη με την ταυτόχρονη έκδοση (*όνομα-σήματος* `<=` *παράσταση*;), αλλά εμφανίζεται στον κορμό μιας διεργασίας και όχι μιας αρχιτεκτονικής. Μια ανάλογη εντολή για μεταβλητές είναι η *εντολή ανάθεσης τιμής μεταβλητής* η οποία έχει τη σύνταξη "*όνομα-μεταβλητής* `:=` *παράσταση*;". Παρατηρήστε ότι χρησιμοποιείται διαφορετικός τελεστής ανάθεσης τιμής, ο `:=`, για τις μεταβλητές.

εντολή σειριακής ανάθεσης τιμής σήματος

εντολή ανάθεσης τιμής μεταβλητής `:=`

Για εκπαιδευτικούς σκοπούς, η αρχιτεκτονική της ροής δεδομένων του ανιχνευτή πρώτων αριθμών στον Πίνακα 4-50 έχει ξαναγραφτεί ως διεργασία στον Πίνακα 4-56. Παρατηρήστε ότι εξακολουθούμε να δουλεύουμε πάνω στην ίδια αρχική οντότητα του `prime` που εμφανίζεται στον Πίνακα 4-43. Μέσα στη νέα αρχιτεκτονική (`prime6_arch`), έχουμε μόνο μία ταυτόχρονη εντολή, η οποία είναι μια διεργασία. Η λίστα ευαισθησίας της διεργασίας περιέχει ακριβώς N κύριες εισόδους της επιθυμητής λειτουργίας συνδυαστικής λογικής. Οι έξοδοι της πύλης AND πρέπει να ορίζονται ως μεταβλητές και όχι ως σήματα, αφού δεν επιτρέπονται ορισμοί σημάτων μέσα σε μια διεργασία. Κατά τα λοιπά, ο κορμός της διεργασίας μοιάζει πολύ με εκείνον της αρχικής αρχιτεκτονικής. Μάλιστα, ένα τυπικό εργαλείο σύνθεσης κατά πάσα πιθανότητα θα δημιουργούσε το ίδιο κύκλωμα για κάθε μία από τις δύο περιγραφές.

Άλλες σειριακές εντολές, πέρα από την απλή ανάθεση τιμής, είναι δυνατόν να μας δώσουν περισσότερο δημιουργικό έλεγχο στην περιγραφή της συμπεριφοράς του κυκλώματος. Η *εντολή* `if`, της οποίας η σύνταξη φαίνεται στον Πίνακα 4-57, είναι πιθανώς η πιο οικεία από αυτές. Στην πρώτη και απλούστερη μορφή της εντολής, ελέγχεται μια *λογική-παράσταση* και εκτελείται μια *σειριακή-εντολή* αν η τιμή της παράστασης είναι αληθής (`true`). Στη δεύτερη μορφή, έχουμε προσθέσει μια εντολή

εντολή `if`

Πίνακας 4-56
Αρχιτεκτονική ροής δεδομένων VHDL σε μορφή διεργασίας για τον ανιχνευτή πρώτων αριθμών.

```
architecture prime6_arch of prime is
begin
   process(N)
      variable N3L_N0, N3L_N2L_N1, N2L_N1_N0,
N2_N1L_N0: STD_LOGIC;
   begin
      N3L_N0     := not N(3)                          and N(0);
      N3L_N2L_N1 := not N(3) and  not N(2)and      N(1)         ;
      N2L_N1_N0  :=               not N(2)and      N(1) and N(0);
      N2_N1L_N0  :=                   N(2)and not  N(1) and N(0);
      F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0 ;
   end process;
end prime6_arch;
```

else

"else" με μια άλλη *σειριακή-εντολή* η οποία εκτελείται αν η τιμή της παράστασης είναι ψευδής (false).

elseif

Για τη δημιουργία ένθετων εντολών if-then-else, η VHDL χρησιμοποιεί την ειδική λέξη-κλειδί *elseif*, η οποία εισάγει τις "ενδιάμεσες" εντολές. Η εντολή elseif για μια *σειριακή-εντολή* εκτελείται αν η *λογική-παράσταση* της είναι αληθής (true) και όλες οι προηγούμενες λογικές παραστάσεις είναι ψευδείς (false). Η προαιρετική *σειριακή-εντολή* της τελικής εντολής else εκτελείται αν όλες οι προηγούμενες λογικές παραστάσεις είναι ψευδείς (false).

Ο Πίνακας 4-58 είναι μια εκδοχή της αρχιτεκτονικής του ανιχνευτή πρώτων αριθμών η οποία χρησιμοποιεί μια εντολή if. Χρησιμοποιείται μια τοπική μεταβλητή NI για να διατηρεί μια μετατραπείσα ακέραια εκδοχή της εισόδου N, έτσι ώστε οι συγκρίσεις στην εντολή if να μπορούν να γραφτούν με χρήση ακεραίων τιμών.

ΠΑΡΑΞΕΝΗ ΣΥΜΠΕΡΙΦΟΡΑ

Θυμηθείτε ότι οι εντολές μέσα σε μια διεργασία εκτελούνται *σειριακά*. Ας υποθέσουμε ότι για κάποιο λόγο γράφαμε την τελευταία εντολή του Πίνακα 4-56 (την ανάθεση τιμής σήματος στην F) ως πρώτη. Τότε θα βλέπαμε μια ιδιαίτερα παράξενη συμπεριφορά από αυτή τη διεργασία.

Την πρώτη φορά που θα εκτελούνταν η διεργασία, ο προσομοιωτής θα διαμαρτυρόταν για το γεγονός ότι οι τιμές των μεταβλητών διαβάζονταν προτού τους αποδοθεί κάποια τιμή. Στις επόμενες επαναλήψεις, θα έπρεπε να αποδοθεί μια τιμή στην F με βάση τις *προηγούμενες* τιμές των μεταβλητών, οι οποίες φυλάσσονται στη μνήμη όσο η διεργασία είναι σε αναστολή. Τότε οι νέες τιμές θα αποδίδονταν στις μεταβλητές και θα φυλάσσονταν στη μνήμη μέχρι την επόμενη επανάληψη. Έτσι η τιμή της εξόδου του κυκλώματος θα υστερούσε πάντα κατά μία αλλαγή εισόδου.

Πίνακας 4-57
Σύνταξη εντολής if της VHDL.

```
if λογική παράσταση then σειριακή-εντολή
end if;

if λογική παράσταση then σειριακή-εντολή
else σειριακή-εντολή
end if;

if λογική παράσταση then σειριακή-εντολή
elsif λογική παράσταση then σειριακή-εντολή
...
elsif λογική παράσταση then σειριακή-εντολή
end if;

if λογική παράσταση then σειριακή-εντολή
elsif λογική παράσταση then σειριακή-εντολή
...
elsif λογική παράσταση then σειριακή-εντολή
else σειριακή-εντολή
end if;
```


Οι λογικές παραστάσεις του Πίνακα 4-58 είναι μη επικαλυπτόμενες, που σημαίνει ότι μόνο μία από αυτές είναι αληθής (`true`) σε μια χρονική στιγμή. Γι' αυτή την εφαρμογή, πραγματικά δε χρειαζόμαστε την πλήρη ισχύ των ένθετων προτάσεων `if`. Μάλιστα, η μηχανή σύνθεσης μπορεί να δημιουργήσει ένα κύκλωμα το οποίο υπολογίζει τις λογικές παραστάσεις εν σειρά, με πιο αργή λειτουργία απ' ό,τι θα μπορούσε να γίνει διαφορετικά. Όταν χρειάζεται να επιλέξουμε ανάμεσα σε πολλές εναλλακτικές λύσεις με βάση την τιμή ενός μόνο σήματος ή παράστασης, η

Πίνακας 4-58
Αρχιτεκτονική του ανιχνευτή πρώτων αριθμών με τη χρήση της εντολής if.

```
architecture prime7_arch of prime is
begin
  process(N)
     variable NI: INTEGER;
  begin
     NI := CONV_INTEGER(N);
     if NI=1 or NI=2 then F <= '1';
     elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <= '1';
     else F <= '0';
     end if;
  end process;
end prime7_arch;
```

εντολή `case`

εντολή `case` είναι συνήθως πιο ευανάγνωστη και μπορεί να δώσει ένα καλύτερο κύκλωμα στο στάδιο σύνθεσης.

Ο Πίνακας 4-59 παρουσιάζει τη σύνταξη της εντολής `case`. Αυτή η εντολή υπολογίζει τη δεδομένη *παράσταση*, βρίσκει μια τιμή που συμφωνεί με μία από τις *επιλογές,* και εκτελεί τις αντίστοιχες *σειριακές-εντολές*. Παρατηρήστε ότι μπορούν να γραφούν μία ή περισσότερες σειριακές εντολές για κάθε σύνολο από *επιλογές.* Οι *επιλογές* είναι δυνατόν να πάρουν τη μορφή μίας απλής τιμής ή πολλών τιμών που διαχωρίζονται από κατακόρυφες γραμμές (|). Οι *επιλογές* πρέπει να είναι αλληλοαποκλειόμενες και να περιλαμβάνουν όλες τις δυνατές τιμές του τύπου για την *παράσταση*. Η λέξη-κλειδί `others` μπορεί να χρησιμοποιηθεί ως η τελευταία από τις *επιλογές* για να καλύψει όλες τις τιμές που δεν έχουν καλυφθεί μέχρι εκείνη τη στιγμή.

Ο Πίνακας 4-60 είναι μια ακόμη αρχιτεκτονική για τον ανιχνευτή πρώτων αριθμών, αυτή τη φορά κωδικοποιημένη με μια εντολή `case`. Όπως και στην ταυτόχρονη εκδοχή, δηλαδή στην εντολή `select` που είδαμε στον Πίνακα 4-54, στην Ενότητα 4.7.7, η εντολή `case` καθιστά ευδιάκριτη την επιθυμητή λειτουργική συμπεριφορά.

εντολή `loop`

Μια άλλη σημαντική κατηγορία σειριακών προτάσεων είναι οι *εντολές* `loop`. Η πιο απλή από αυτές έχει τη σύνταξη που φαίνεται στον Πίνακα 4-61 και δημιουργεί έναν ατέρμονα βρόχο. Αν και οι ατέρμονες βρόχοι είναι ανεπιθύμητοι στις συμβατικές γλώσσες προγραμματισμού, στην Ενότητα 7.12 θα δείξουμε πώς ένας τέτοιος βρόχος μπορεί να είναι πολύ χρήσιμος στη μοντελοποίηση υλικού.

βρόχος `for`

Ένας πιο οικείος βρόχος, που έχουμε ξαναδεί στο παρελθόν, είναι ο βρόχος `for` με τη σύνταξη που φαίνεται στον Πίνακα 4-62. Παρατηρήστε ότι η μεταβλητή *αναγνωριστικό* του βρόχου ορίζεται αποκλειστικά με την εμφάνισή της στο βρόχο `for` και έχει τον ίδιο τύπο με το *πεδίο τιμών*. Αυτή η μεταβλητή μπορεί να χρησιμοποιηθεί μέσα στις σειριακές εντολές του βρόχου και διέρχεται βήμα-προς βήμα από όλες τις τιμές στο *πεδίο τιμών*, από τα αριστερά προς τα δεξιά, με μία τιμή ανά επανάληψη.

εντολή `exit`
εντολή `next`

Δύο πιο χρήσιμες σειριακές εντολές που είναι δυνατόν να εκτελεστούν μέσα σε ένα βρόχο είναι οι "`exit`" και "`next`". Με την εκτέλεσή της, η `exit` δίνει τον έλεγχο στην εντολή που ακολουθεί αμέσως μετά το τέλος του βρόχου. Από την άλλη πλευρά, η εντολή `next` παρακάμπτει τις εντολές που απομένουν σε ένα βρόχο και ξεκινά την επόμενη επανάληψη του βρόχου.

Πίνακας 4-59
Σύνταξη εντολής case της VHDL.

```
case παράσταση is
  when επιλογές => σειριακές-εντολές
  ...
  when επιλογές => σειριακές-εντολές end case;
```

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
     case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
     end case;
  end process;
end prime8_arch;
```

Πίνακας 4-60
Αρχιτεκτονική του ανιχνευτή πρώτων αριθμών με χρήση της εντολής case.

Ο παλιός καλός ανιχνευτής πρώτων αριθμών κωδικοποιείται για μια ακόμη φορά στον Πίνακα 4-63, αυτή τη φορά όμως με χρήση ενός βρόχου `for`. Το εντυπωσιακό με το παράδειγμα αυτό είναι ότι αποτελεί πραγματικά μια περιγραφή συμπεριφοράς: έχουμε χρησιμοποιήσει στην πραγματικότητα τη VHDL για να υπολογίζουμε κατά πόσον η είσοδος `N` είναι πρώτος αριθμός. Έχουμε επίσης αυξήσει το μέγεθος του `N` στα 16 bit, απλώς για να δώσουμε έμφαση στο γεγονός ότι μπορούμε να δημιουργήσουμε ένα συμπαγές μοντέλο κυκλώματος χωρίς να χρειάζεται να καθορίσουμε ρητά χιλιάδες πρώτους αριθμούς.

βρόχος `while`

Το τελευταίο είδος εντολών `loop` είναι ο *βρόχος* `while`, που συντάσσεται όπως στον Πίνακα 4-64. Σε αυτή τη μορφή, η *λογική-παρά-*

```
loop
   σειριακή-εντολή
   ...
   σειριακή-εντολή
end loop;
```

Πίνακας 4-61
Σύνταξη βασικής εντολής `loop` της VHDL.

```
for αναγνωριστικό in πεδίο-τιμών loop
   σειριακή-εντολή
   ...
   σειριακή-εντολή
end loop;
```

Πίνακας 4-62
Σύνταξη βρόχου `for` της VHDL.

Πίνακας 4-63 Αρχιτεκτονική του ανιχνευτή πρώτων αριθμών με χρήση της εντολής `for`.

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

entity prime9 is
   port ( N: in STD_LOGIC_VECTOR (15 downto 0);
          F: out STD_LOGIC );
end prime9;

architecture prime9_arch of prime9 is
begin
   process(N)
   variable NI: INTEGER;
   variable prime: boolean;
   begin
      NI := CONV_INTEGER(N);
      prime := true;
      if NI=1 or NI=2 then null; -- take care of boundary cases
      else for i in 2 to 253 loop
              if (NI mod i = 0) and (NI /= i)
              then
                      prime := false; exit;
              end if;
         end loop;
      end if;
      if prime then F <= '1'; else F <= '0'; end if;
   end process;
end prime9_arch;
```


ΚΑΚΗ ΣΧΕΔΙΑΣΗ

Στον Πίνακα 4-63 υπάρχει ένα καλό παράδειγμα βρόχου `for`, το οποίο όμως είναι παράδειγμα προς αποφυγή για τη σχεδίαση ενός κυκλώματος. Αν και η VHDL είναι μια ισχυρή γλώσσα προγραμματισμού, οι περιγραφές σχεδιάσεων που χρησιμοποιούν την πλήρη ισχύ της ενδέχεται να είναι μη αποδοτικές ή χωρίς δυνατότητα σύνθεσης.

Ο ένοχος στον Πίνακα 4-63 είναι ο τελεστής `mod`. Για την πράξη αυτή απαιτείται διαίρεση ακεραίων, και τα περισσότερα εργαλεία της VHDL δεν έχουν τη δυνατότητα να συνθέσουν κυκλώματα διαίρεσης εκτός από ειδικές περιπτώσεις, όπως είναι η διαίρεση με δύναμη του δύο (η οποία υλοποιείται ως ολίσθηση).

Ακόμη και αν τα εργαλεία μπορούσαν να συνθέσουν ένα διαιρέτη, δε θα θέλαμε να περιγράψουμε έναν ανιχνευτή πρώτων αριθμών με αυτόν τον τρόπο. Η περιγραφή του Πίνακα 4-63 υποδηλώνει ένα συνδυαστικό κύκλωμα και τα εργαλεία θα έπρεπε να δημιουργήσουν 252 συνδυαστικούς διαιρέτες, έναν για κάθε τιμή του *i*, για να "ξεδιπλώσουν" το βρόχο `for` και να υλοποιήσουν το κύκλωμα!

σταση ελέγχεται πριν από κάθε επανάληψη του βρόχου και ο βρόχος εκτελείται μόνο αν η τιμή της παράστασης είναι αληθής (`true`).

Μπορούμε να χρησιμοποιήσουμε διεργασίες για να γράψουμε περιγραφές συμπεριφοράς τόσο των συνδυαστικών όσο και των ακολουθιακών κυκλωμάτων. Πολλά περισσότερα παραδείγματα περιγραφών συνδυαστικών κυκλωμάτων εμφανίζονται στις ενότητες περί VHDL του Κεφαλαίου 5. Για την περιγραφή ακολουθιακών κυκλωμάτων χρειάζονται μερικά επιπλέον χαρακτηριστικά της γλώσσας. Αυτά περιγράφονται στην Ενότητα 7.12, ενώ στις ενότητες περί VHDL του Κεφαλαίου 8 παρουσιάζονται ακολουθιακά παραδείγματα.

Πίνακας 4-64
Σύνταξη βρόχου `while` της VHDL.

```
while λογική παράσταση loop
   σειριακή-εντολή
   ...
   σειριακή-εντολή
end loop;
```


4.7.9 Η χρονική διάσταση και η προσομοίωση

Κανένα από τα παραδείγματα που εξετάσαμε μέχρι τώρα δε μοντελοποιεί τη χρονική διάσταση της λειτουργίας του κυκλώματος. Τα πάντα συμβαίνουν σε μηδενικό προσομοιωμένο χρόνο. Ωστόσο, η VHDL έχει εξαιρετικές δυνατότητες μοντελοποίησης του χρόνου, και αυτό είναι πράγματι μια άλλη σημαντική διάσταση της γλώσσας. Σε αυτό το βιβλίο δε θα μπούμε σε λεπτομέρειες για αυτό το θέμα, αλλά θα παρουσιάσουμε μερικές ιδέες στο σημείο αυτό.

λέξη-κλειδί `after`

Η VHDL σας επιτρέπει να καθορίζετε μια χρονική καθυστέρηση με τη λέξη-κλειδί `after` σε οποιαδήποτε εντολή ανάθεσης τιμής σήματος, συμπεριλαμβανομένων των σειριακών, ταυτόχρονων, υπό συνθήκη, και επιλεγμένων αναθέσεων τιμής. Για παράδειγμα, στην αρχιτεκτονική πύλης ανακοπής του Πίνακα 4-26 στην Ενότητα 4.7.2, μπορείτε να γράψετε:

```
Z <= '1' after 4 ns when X='1' and Y='0' else '0' after 3 ns;
```

Αυτό σας επιτρέπει να μοντελοποιήσετε μια πύλη ανακοπής η οποία έχει καθυστέρηση 4 ns στη μετάβαση από 0 σε 1 και μόνο 3 ns στη μετάβαση από 1 σε 0. Στα καθιερωμένα περιβάλλοντα σχεδίασης ASIC, τα μοντέλα VHDL για όλα τα στοιχεία χαμηλού επιπέδου που περιλαμβάνονται στη βιβλιοθήκη VHDL περιλαμβάνουν τέτοιες παραμέτρους καθυστέρησης. Με τη βοήθεια των εκτιμήσεων αυτών, ο προσομοιωτής μπορεί να προβλέψει κατά προσέγγιση τη συμπεριφορά χρονισμού ενός μεγαλύτερου κυκλώματος το οποίο χρησιμοποιεί αυτά τα στοιχεία.

εντολή `wait`

Ένας άλλος τρόπος χρήσης της χρονικής διάστασης είναι να χρησιμοποιήσετε τη σειριακή εντολή `wait`. Αυτή η εντολή μπορεί να χρησιμοποιηθεί για να θέσει σε αναστολή μια διεργασία για ένα καθορισμένο χρονικό διάστημα. Ο Πίνακας 4-65 είναι ένα παράδειγμα προγράμματος το οποίο χρησιμοποιεί την εντολή `wait` για να δημιουργήσει προσομοιωμένες κυματομορφές εισόδου για τον έλεγχο της πύλης ανακοπής για τέσσερις διαφορετικούς συνδυασμούς εισόδων σε χρονικά βήματα των 10 ns.

Από τη στιγμή που έχετε στα χέρια σας ένα πρόγραμμα VHDL του οποίου η σύνταξη και η σημασιολογία είναι σωστές, μπορείτε να χρησιμοποιήσετε έναν προσομοιωτή VHDL για να παρατηρήσετε τη λειτουργία του. Αν και δε θα μπούμε σε πολλές λεπτομέρειες, είναι χρήσιμο να αποκτήσουμε μια βασική αντίληψη για το πώς λειτουργεί ένας προσομοιωτής.

χρόνος προσομοίωσης

Η λειτουργία του προσομοιωτή αρχίζει στο *χρόνο προσομοίωσης* μηδέν. Εκείνη τη χρονική στιγμή, ο προσομοιωτής θέτει στην αρχική τους κατάσταση όλα τα σήματα αποδίδοντάς τους μια προεπιλεγμένη τιμή (στην οποία δεν πρέπει να βασίζεστε!). Θέτει επίσης στην αρχική τους κατάσταση όλα τα σήματα ή τις μεταβλητές για τα οποία έχουν δηλωθεί ρητά αρχικές τιμές (δεν έχουμε δείξει ακόμη πώς γίνεται αυτό). Μετά ο προσομοιωτής ξεκινά την εκτέλεση όλων των διεργασιών και των ταυτόχρονων εντολών της σχεδίασης.

Πίνακας 4-65 Χρήση της εντολής `wait` της VHDL για τη δημιουργία κυματομορφών εισόδου σε ένα πρόγραμμα περιβάλλοντος δοκιμών.

```
entity InhibitTestBench is
end InhibitTestBench;

architecture InhibitTB_arch of InhibitTestBench is
component Inhibit port (X,Y: in BIT; Z: out BIT); end
component;
signal XT, YT, ZT: BIT;
begin
  U1: Inhibit port map (XT, YT, ZT);
  process
  begin
      XT <= '0'; YT <= '0';
      wait for 10 ns;
      XT <= '0'; YT <= '1';
      wait for 10 ns;
      XT <= '1'; YT <= '0';
      wait for 10 ns;
      XT <= '1'; YT <= '1';
      wait; -- this suspends the process indefinitely
  end process;
end InhibitTB_arch;
```

Φυσικά, ο προσομοιωτής δεν μπορεί να προσομοιώνει πραγματικά όλες τις διεργασίες και τις ταυτόχρονες εντολές την ίδια στιγμή, αλλά μπορεί να παριστάνει ότι το κάνει, με τη βοήθεια μιας χρονικής "λίστας συμβάντων" και μιας "μήτρας ευαισθησίας σήματος". Σημειώστε ότι κάθε ταυτόχρονη εντολή είναι ισοδύναμη με μία διεργασία.

Κατά το χρόνο προσομοίωσης μηδέν, όλες οι διεργασίες είναι προγραμματισμένες προς εκτέλεση και επιλέγεται μία από αυτές. Εκτελούνται όλες οι σειριακές εντολές, συμπεριλαμβανομένων και των τυχόν συμπεριφορών βρόχου που έχουν καθοριστεί. Μόλις η εκτέλεση της συγκεκριμένης διεργασίας ολοκληρωθεί, επιλέγεται μια άλλη κ.ο.κ., έως ότου εκτελεστούν όλες οι διεργασίες. Έτσι ολοκληρώνεται ένας *κύκλος προσομοίωσης*.

κύκλος προσομοίωσης

Κατά τη διάρκεια της εκτέλεσής της, μια διεργασία μπορεί να αποδώσει νέες τιμές σε σήματα. Οι νέες τιμές δεν αποδίδονται αμέσως, αλλά τοποθετούνται στη *λίστα συμβάντων* και προγραμματίζονται να τεθούν σε ισχύ σε συγκεκριμένο χρόνο. Αν η ανάθεση τιμής έχει έναν αντίστοιχο ρητά καθορισμένο χρόνο προσομοίωσης (για παράδειγμα, μετά από μια καθυστέρηση που περιγράφεται από μια φράση `after`), τότε προγραμματίζεται στη λίστα συμβάντων για να τεθεί σε ισχύ την καθορισμένη χρονική στιγμή. Διαφορετικά, υποτίθεται ότι πρέπει να τεθεί σε ισχύ "αμέσως", όμως στην πραγματικότητα προγραμματίζεται να λάβει χώρα κατά τον τρέχοντα χρόνο προσομοίωσης συν μια "καθυστέρηση δέλτα". Η *καθυστέρηση δέλτα* είναι ένας απειροελάχιστα μικρός χρόνος, τέτοιος ώστε ο τρέχων χρόνος προσομοίωσης συν οποιοσδήποτε αριθμός καθυστερήσεων δέλτα να εξακολουθεί να είναι ίσος με τον τρέχοντα χρόνο προσομοίωσης. Αυτή η έννοια επιτρέπει στις διεργασίες να εκτελούνται περισσότερες από μία φορά, αν χρειάζεται, σε μηδενικό χρόνο προσομοίωσης.

λίστα συμβάντων

καθυστέρηση δέλτα

Μόλις ολοκληρωθεί ένας κύκλος προσομοίωσης, η λίστα συμβάντων σαρώνεται για να εντοπιστεί το σήμα ή τα σήματα της λίστας που θα αλλάξουν στο άμεσο μέλλον. Το άμεσο μέλλον μπορεί να είναι τόσο σύντομο όσο μια καθυστέρηση δέλτα αργότερα, ή μπορεί να ισούται με μια πραγματική καθυστέρηση κυκλώματος, στην οποία περίπτωση ο χρόνος προσομοίωσης προχωρά στο συγκεκριμένο χρονικό σημείο. Σε κάθε περίπτωση, γίνονται οι προγραμματισμένες αλλαγές σημάτων. Μερικές διεργασίες ενδέχεται να παρουσιάζουν ευαισθησία στα σήματα που αλλάζουν, όπως φαίνεται από τον πίνακα ευαισθησίας σημάτων. Ο πίνακας αυτός δείχνει, για κάθε σήμα, ποιες διεργασίες περιέχουν το συγκεκριμένο σήμα στη λίστα ευαισθησίας τους. (Η ισοδύναμη διεργασία μιας ταυτόχρονης εντολής περιέχει στη λίστα ευαισθησίας της *όλα* τα σήματα ελέγχου και δεδομένων που χρησιμοποιεί.) Όλες οι διεργασίες που είναι ευαίσθητες στο σήμα που μόλις άλλαξε προγραμματίζονται προς εκτέλεση στον επόμενο κύκλο προσομοίωσης που ξεκινά τώρα.

Η εκτέλεση ενός κύκλου προσομοίωσης σε δύο φάσεις από τον προσομοιωτή, ακολουθούμενη από τη σάρωση της λίστας συμβάντων και την εκτέλεση των επόμενων αναθέσεων τιμής σήματος, συνεχίζεται επα-

όριστον, έως ότου αδειάσει η λίστα. Τότε, η προσομοίωση θα έχει ολοκληρωθεί.

Ο μηχανισμός της λίστας συμβάντων κάνει δυνατή την προσομοίωση της εκτέλεσης των ταυτόχρονων διεργασιών έστω και αν ο προσομοιωτής εκτελείται σε έναν και μόνο υπολογιστή με ένα και μόνο νήμα εκτέλεσης. Ταυτόχρονα, ο μηχανισμός καθυστέρησης δέλτα εξασφαλίζει τη σωστή εκτέλεση ακόμη και αν μια διεργασία ή ένα σύνολο διεργασιών μπορεί να απαιτεί πολλές εκτελέσεις, εκτεινόμενες χρονικά σε αρκετές καθυστερήσεις δέλτα, προτού τα μεταβαλλόμενα σήματα σταθεροποιηθούν σε μια σταθερή τιμή. Αυτός ο μηχανισμός χρησιμοποιείται επίσης για τον εντοπισμό διεργασιών εκτός ελέγχου (όπως είναι η "`X <= not X`"). Αν πραγματοποιηθούν χίλιοι κύκλοι προσομοίωσης σε χρονικό διάστημα χιλίων καθυστερήσεων δέλτα χωρίς να προχωρήσει ο χρόνος προσομοίωσης κατά ένα "πραγματικό" χρονικό διάστημα, το πιθανότερο είναι ότι κάτι δεν πάει καλά.

4.7.10 Σύνθεση

Όπως αναφέραμε στην αρχή αυτής της ενότητας, η VHDL επινοήθηκε αρχικά ως μια γλώσσα περιγραφής λογικών κυκλωμάτων και προσομοίωσης, ενώ αργότερα προσαρμόστηκε και στη σύνθεση. Η γλώσσα διαθέτει πολλά χαρακτηριστικά και δομές η σύνθεση των οποίων είναι αδύνατη. Ωστόσο, το υποσύνολο της γλώσσας και το στιλ των προγραμμάτων που έχουμε παρουσιάσει σε αυτή την ενότητα είναι γενικά συνθέσιμα από τα περισσότερα εργαλεία.

Παρόλα αυτά, ο κώδικας που γράφετε μπορεί να έχει μεγάλη επίδραση στην ποιότητα των κυκλωμάτων που συνθέτετε. Παρακάτω αναφέρονται μερικά παραδείγματα:

- Οι "σειριακές" δομές ελέγχου, όπως π.χ. `if-elseif-elseif-else`, είναι δυνατόν να οδηγήσουν σε μια αντίστοιχη σειριακή αλυσίδα λογικών πυλών σύμφωνα με τις συνθήκες ελέγχου. Αν οι συνθήκες είναι αλληλοαποκλειόμενες, είναι προτιμότερο να χρησιμοποιήσετε μια εντολή `case` ή `select`.
- Οι βρόχοι στις διεργασίες γενικά "ξετυλίγονται" για τη δημιουργία πολλών αντιγράφων συνδυαστικής λογικής για την εκτέλεση των προτάσεων που περιέχονται σε αυτούς. Αν θέλετε να χρησιμοποιήσετε ένα μόνο αντίγραφο της συνδυαστικής λογικής σε μια ακολουθία βημάτων, τότε θα πρέπει να σχεδιάσετε ένα ακολουθιακό κύκλωμα, όπως περιγράφεται σε επόμενα κεφάλαια.
- Όταν χρησιμοποιείτε υπό συνθήκη εντολές σε μια διεργασία χωρίς να ορίζετε αποτέλεσμα για κάποιους συνδυασμούς εισόδων, ο μεταγλωττιστής ενδέχεται να δημιουργήσει μια μανδάλωση που θα διατηρεί την παλιά τιμή ενός σήματος η οποία διαφορετικά θα μπορούσε να αλλάξει. Γενικά, η δημιουργία τέτοιων μανδαλώσεων δεν ενδείκνυται.

Επίσης, μερικά χαρακτηριστικά και δομές της γλώσσας μπορεί απλώς να είναι μη συνθέσιμα, ανάλογα με το εργαλείο που χρησιμοποιείτε. Φυσικά, θα πρέπει να συμβουλευθείτε την τεκμηρίωση για να βρείτε τι επιτρέπεται, τι δεν επιτρέπεται, και τι προτείνεται για ένα συγκεκριμένο εργαλείο.

Για το προβλέψιμο μέλλον, οι σχεδιαστές ψηφιακών κυκλωμάτων που χρησιμοποιούν εργαλεία σύνθεσης θα πρέπει να δίνουν ευλόγως μεγάλη προσοχή στο στιλ κώδικα που χρησιμοποιούν, ώστε να έχουν καλά αποτελέσματα. Προς το παρόν, ο ορισμός του όρου "καλό στιλ κώδικα" εξαρτάται κατά κάποιον τρόπο τόσο από το εργαλείο σύνθεσης όσο και από την τεχνολογία προορισμού. Τα παραδείγματα που παρατίθενται στο υπόλοιπο μέρος του βιβλίου αυτού, παρά το γεγονός ότι είναι συντακτικά και σημασιολογικά σωστά, ελάχιστα αποκαλύπτουν τις πλήρεις δυνατότητες των μεθόδων συγγραφής κώδικα για μεγάλες σχεδιάσεις σε HDL. Η τέχνη και η πρακτική εξάσκηση σε μεγάλες σχεδιάσεις υλικού σε HDL εξακολουθεί σε μεγάλο βαθμό να εξελίσσεται.

Παραπομπές

Μια ιστορική περιγραφή της ανάπτυξης "της επιστήμης της λογικής" από τον Boole παρουσιάζεται στο βιβλίο *The Computer from Pascal to von Neumann* του Herman H. Goldstine (Princeton University Press, 1972). Ο Claude H. Shannon δείχνει πώς η εργασία του Boole μπορεί να εφαρμοστεί στα λογικά κυκλώματα, στο βιβλίο "A Symbolic Analysis of Relay and Switching Circuits" (*Trans. AIEE,* τόμος 57, 1938, σελ. 713-723).

Αν και η άλγεβρα Boole δύο τιμών αποτελεί τη βάση για την άλγεβρα μεταγωγής, μια άλγεβρα Boole δεν είναι υποχρεωτικό να έχει δύο μόνο τιμές. Υπάρχουν άλγεβρες Boole με 2^n τιμές, όπου n οποιοσδήποτε ακέραιος· για παράδειγμα, δείτε το βιβλίο *DiscreteMathematical Structures and Their Applications* του Harold S. Stone (SRA, 1973). Τέτοιες άλγεβρες ορίζονται τυπικά με τη βοήθεια των λεγόμενων *αξιωμάτων Huntington* που επινοήθηκαν από τον E. V. Huntington το 1907· για παράδειγμα, δείτε το βιβλίο *Digital Design* του M. Morris Mano (Prentice Hall, 1984). Μια μαθηματική ανάπτυξη της άλγεβρας Boole με βάση ένα πιο σύγχρονο σύνολο αξιωμάτων περιλαμβάνεται στο *Modern Applied Algebra* των G. Birkhoff και T. C. Bartee (McGraw-Hill, 1970). Η δική μας ανάπτυξη μιας άλγεβρας μεταγωγής με "άμεσο" τεχνικό στιλ ακολουθεί το πρότυπο του Edward J. McCluskey στο *Introduction to the Theory of Switching Circuits* (McGraw-Hill, 1965) και στο *Logic Design Principles* (Prentice Hall, 1986).

αξιώματα Huntington

Το θεώρημα των πρωταρχικών όρων αποδείχτηκε από τον W. V. Quine στο βιβλίο "The Problem of Simplifying Truth Functions" (*Am. Math. Monthly,* τόμος 59, αρ. 8, 1952, σελ. 521-531). Στην πραγματικότητα μπορούμε να αποδείξουμε ένα πιο γενικό θεώρημα πρωταρχικών όρων, δείχνοντας ότι υπάρχει τουλάχιστον ένα ελάχιστο άθροισμα το

οποίο είναι άθροισμα πρωταρχικών όρων, ακόμη και αν καταργήσουμε τον περιορισμό που αφορά τον αριθμό των κυριολεκτικών από τον ορισμό του "ελαχίστου".

Μια γραφική μέθοδος για την απλοποίηση των λογικών συναρτήσεων προτείνεται από τον E. W. Veitch στο βιβλίο "A Chart Method for Simplifying Boolean Functions" (*Proc. ACM,* Μάιος 1952, σελ. 127-133). Το *διάγραμμα Veitch*, που φαίνεται στην Εικόνα 4-53, στην ουσία ανακάλυψε ξανά ένα διάγραμμα που είχε προταθεί από τον Άγγλο αρχαιολόγο A. Marguand ("On Logical Diagrams for *n* Terms", *Philosophical Magazine* XII, 1881, σελ. 266-270). Το διάγραμμα Veitch ή διάγραμμα Marquand χρησιμοποιεί μια "φυσική" σειρά δυαδικής απαρίθμησης των γραμμών και των στηλών του, με αποτέλεσμα μερικές γειτονικές γραμμές και στήλες να διαφέρουν κατά περισσότερες από μία τιμές, ενώ τα γειτονικά κελιά να μην καλύπτονται πάντα από όρους γινομένου. Ο M. Karnaugh έδειξε πώς αντιμετωπίζεται αυτό το πρόβλημα στο "A Map Method for Synthesis of Combinational Circuits" (*Trans. AIEE, Comm. and Electron.,* τόμος 72, μέρος I, Νοέμβριος 1953, σελ. 593-599). Από την άλλη πλευρά, ο George J. Klir, στο βιβλίο του *Introduction to the Methodology of Switching Circuits* (Van Nostrand, 1972), ισχυρίζεται ότι η σειρά της δυαδικής απαρίθμησης είναι το ίδιο καλή με τη σειρά του πίνακα Karnaugh, αν όχι καλύτερη, για την ελαχιστοποίηση των λογικών συναρτήσεων.

Σε αυτό το σημείο η αντιπαράθεση Karnaugh και Veitch είναι φυσικά αδιάφορη, επειδή κανείς δε σχεδιάζει πλέον διαγράμματα για την ελαχιστοποίηση λογικών κυκλωμάτων. Αντίθετα, σήμερα χρησιμοποιούμε προγράμματα υπολογιστών που εκτελούν αλγόριθμους λογικής ελαχιστοποίησης. Ο πρώτος από αυτούς τους αλγόριθμους περιγράφτηκε από τον Quine στο "A Way to Simplify Truth Functions" (*Am. Math, Monthly,* τόμος 62, αρ. 9, 1995, σελ. 627-631) και τροποποιήθηκε από τον E. J. McCluskey στο "Minimization of Boolean Functions" (*Bell Sys. Tech. J.,* τόμος 35, αρ. 5, Νοέμβριος 1956, σελ. 1417-1444). Ο αλγόριθμος Quine-McCluskey περιγράφεται πλήρως στα βιβλία του McCluskey που μνημονεύτηκαν νωρίτερα.

Το βιβλίο του McCluskey που εκδόθηκε το 1965 καλύπτει επίσης τον επαναληπτικό αλγόριθμο κοινής συναίνεσης για την εύρεση πρωταρχικών όρων και αποδεικνύει ότι αυτό αποδίδει. Το σημείο έναρξης του αλγόριθμου αυτού είναι μια παράσταση αθροίσματος γινομένων ή, ισοδύναμα, μια λίστα κύβων. Οι όροι γινομένου *δε χρειάζεται* να είναι ελάχιστοι όροι ή πρωταρχικοί όροι, αλλά *μπορούν* να είναι οποιοδήποτε από τα δυο ή οτιδήποτε ενδιάμεσο. Με άλλα λόγια, οι κύβοι της λίστας είναι δυνατόν να έχουν οποιαδήποτε από τις διαστάσεις, από 0 έως *n* σε μια συνάρτηση *n* μεταβλητών. Ξεκινώντας με τη λίστα των κύβων, ο αλγόριθμος δημιουργεί μια λίστα με όλους τους κύβους πρωταρχικών όρων της συνάρτησης, χωρίς καν να χρειαστεί να δημιουργήσει μια πλήρη λίστα ελαχίστων όρων.

Ο επαναληπτικός αλγόριθμος κοινής συναίνεσης δημοσιεύθηκε πρώτα από τον T. H. Mott, Jr. στο "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of Prime Implicants" (*IRE Trans. Electron. Computers,* τόμος EC-9, αρ. 2, 1960, σελ. 245-252). Ένας γενικευμένος αλγόριθμος κοινής συναίνεσης δημοσιεύθηκε από τον Pierre Tison στο "Generalization of Consensus Theory and Applications to the Minimization of Boolean Functions" (*IEEE Trans. Electron. Computers,* τόμος EC-16, αρ. 4, 1967, σελ. 446-456). Όλοι αυτοί οι αλγόριθμοι περιγράφονται από τον Thomas Downs στο βιβλίο *Logic Design with Pascal* (Van Nostrad Reinhold, 1988).

Όπως εξηγήσαμε στην Ενότητα 4.4.4, ο τεράστιος αριθμός πρωταρχικών όρων σε μερικές λογικές συναρτήσεις καθιστά ανεφάρμοστη ή αδύνατη την ντετερμινιστική εύρεση όλων ή την επιλογή μιας ελάχιστης κάλυψης. Ωστόσο, οι αποτελεσματικές εμπειρικές μέθοδοι είναι δυνατόν να βρουν λύσεις που είναι κοντά στο ελάχιστο. Η μέθοδος Espresso-II περιγράφεται στο *Logic Minimization Algorithms for VLSI Synthesis* των R. K. Brayton, C. McMullen, G. D. Hachtel και A. Sangiovanni-Vincentelli (Kluwer Academic Publishers, 1984). Οι πιο πρόσφατοι αλγόριθμοι Espresso-MV και Espresso-EXACT περιγράφονται στο "Multiple-Valued Minimization for PLA Optimization" των R. L. Rudell και A. Sangiovanni-Vincentelli (*IEEE Trans. CAD,* τόμος CAD-6, αρ. 5, 1987, σελ. 727-750).

Σε αυτό το κεφάλαιο περιγράψαμε μια μέθοδο χαρτογράφησης για την εύρεση στατικών κινδύνων σε κυκλώματα AND-OR και OR-AND δύο επιπέδων, αλλά κάθε συνδυαστικό κύκλωμα είναι δυνατόν να αναλυθεί για τυχόν κινδύνους. Και στα δύο του βιβλία, που εκδόθηκαν το 1965 και το 1986, ο McCluskey ορίζει τα *σύνολα 0* και τα *σύνολα 1* ενός κυκλώματος και δείχνει πώς μπορούν να χρησιμοποιηθούν για την εύρεση των στατικών κινδύνων. Ορίζει επίσης τα *σύνολα P* και τα *σύνολα S* και δείχνει πώς μπορούν να χρησιμοποιηθούν για την εύρεση των δυναμικών κινδύνων.

σύνολο 0
σύνολο 1
σύνολο P
σύνολο S

Πολλές βαθύτερες και διαφορετικές πλευρές της θεωρίας μεταγωγής έχουν παραλειφθεί από αυτό το βιβλίο αλλά περιγράφονται αναλυτικά σε άλλα βιβλία και δημοσιεύσεις. Ένα καλό σημείο έναρξης για την ακαδημαϊκή μελέτη της κλασικής θεωρίας μεταγωγής είναι το βιβλίο του Zvi Kohavi, *Switching and Finite Automata Theory,* δεύτερη έκδοση (McGraw-Hill, 1978), το οποίο περιλαμβάνει ύλη για τη θεωρία των συνόλων, τα συμμετρικά δίκτυα, τη λειτουργική ανάλυση, τη λογική κατωφλίου, τον εντοπισμό αστοχιών, και την ευαισθητοποίηση διαδρομών. Μια άλλη περιοχή με ακαδημαϊκό ενδιαφέρον (αλλά με μικρό εμπορικό ενδιαφέρον) είναι η μη δυαδική *λογική πολλών τιμών*, στην οποία κάθε γραμμή σήματος μπορεί να πάρει περισσότερες από δύο τιμές. Στο βιβλίο του που εκδόθηκε το 1986, ο McCluskey δίνει μια καλή εισαγωγή στη λογική πολλών τιμών, εξηγώντας τα πλεονεκτήματα και τα μειονεκτήματά της, καθώς και το λόγο για τον οποίο δεν έχει τύχει μεγάλης εμπορικής υποστήριξης.

λογική πολλών τιμών

YZ \ WX	00	01	10	11
00	0	4	8	12
01	1	5	9	13
10	2	6	10	14
11	3	7	11	15

Εικόνα 4-53
Διάγραμμα Veitch ή διάγραμμα Marquant τεσσάρων μεταβλητών

Αγωνιζόμουν εδώ και χρόνια να βρω μια προσιτή και έγκυρη πηγή αναφοράς για τη γλώσσα ABEL, και τελικά τη βρήκα: Παράρτημα Α του *Digital Design Using ABEL,* των David Pellerin και Michael Holley (Prentice Hall, 1994). Είναι φυσικό που είναι έγκυρη πηγή, αφού οι Pellerin και Holley είναι εκείνοι που επινόησαν τη γλώσσα και έγραψαν τον αρχικό κώδικα του μεταγλωττιστή! Ο Pellerin έγραψε επίσης, μαζί με το συγγραφέα Douglas Taylor, μια πολύ ωραία αυτόνομη εισαγωγή στη VHDL με τίτλο *VHDL Made Easy!* (Prentice Hall, 1997).

Κάνοντας ένα λογοπαίγνιο με το ακρώνυμο VHDL, θα μπορούσαμε να πούμε ότι η VHDL είναι μια "Very Large Design Language" (πολύ μεγάλη γλώσσα σχεδίασης), και στο βιβλίο αυτό καλύπτουμε μόνο ένα υποσύνολο των χαρακτηριστικών και των δυνατοτήτων της. Υπάρχουν πολλές επιλογές της γλώσσας (όπως π.χ. οι ετικέτες προτάσεων) που δεν τις χρησιμοποιούμε, αλλά καλύπτονται στα ειδικά εγχειρίδια και βιβλία αναφοράς της VHDL. Ένα από τα καλύτερα είναι το *The Designer's Guide to VHDL* (Morgan Kaufman, 1996). Αν και ογκώδες (700 σελίδες), καλύπτει αρκετά συστηματικά ολόκληρη τη γλώσσα και περιλαμβάνει ένα πολύ χρήσιμο και συνοπτικό παράρτημα με την πλήρη σύνταξη της γλώσσας τόσο για την έκδοση VHDL-87 όσο και για την έκδοση VHDL 93. Σε αυτό το βιβλίο, χρησιμοποιούμε ένα υποσύνολο της γλώσσας που είναι συμβατό και με τα δύο πρότυπα.

Μια κάπως πιο σύντομη παρουσίαση της VHDL γίνεται στο βιβλίο *VHDL for Designers* των Stefan Sjoholm και Lennart Lindh (Prentice Hall, 1997). Αυτό το βιβλίο προτείνεται για την έμφαση που δίνει σε πρακτικά θέματα σχεδίασης, μεταξύ των οποίων η σύνθεση και τα περιβάλλοντα δοκιμών. Ένα άλλο πρακτικό βιβλίο είναι το *VHDL for Programmable Logic,* του Kevin Skahill από την εταιρία Cypress Semiconductor (Addison Wesley, 1996).

Όλα τα παραδείγματα ABEL και VHDL σε αυτό το κεφάλαιο και σε όλο το βιβλίο μεταγλωττίστηκαν και, στις περισσότερες περιπτώσεις,

προσομοιώθηκαν με τη βοήθεια του λογισμικού Foundation 1.5 Student Edition της Xilinx, Inc. (San Jose, CA 95124, Η.Π.Α., www.xilinx.com). Το λογισμικό Foundation περιλαμβάνει έναν επεξεργαστή σχηματικών διαγραμμάτων, έναν επεξεργαστή HDL, μεταγλωττιστές για ABEL, VHDL, και Verilog, καθώς και έναν προσομοιωτή της Aldec, Inc. (Henderson, NV 89014, Η.Π.Α., www.aldec.com), μαζί με τα εξειδικευμένα εργαλεία της ίδιας της Xilinx για τη σχεδίαση και τον προγραμματισμό σε CPLD και FPGA. Περιέχει επίσης ένα εξαιρετικό ηλεκτρονικό σύστημα βοήθειας, που περιλαμβάνει εγχειρίδια αναφοράς τόσο για την ABEL όσο και για τη VHDL.

Τα πακέτα VHDL του προτύπου IEEE είναι σημαντικά στοιχεία σε κάθε περιβάλλον σχεδίασης VHDL. Η λίστα με τους ορισμούς τύπων και συναρτήσεων περιλαμβάνεται ως παράρτημα σε μερικά κείμενα VHDL, αλλά αν είστε πραγματικά περίεργοι γι' αυτά, μπορείτε να βρείτε τα πλήρη αρχεία προέλευσης στο υποσύστημα βιβλιοθήκης σε οποιοδήποτε σύστημα σχεδίασης VHDL, συμπεριλαμβανομένου και του λογισμικού Foundation.

Περιγράψαμε συνοπτικά τον έλεγχο συσκευών στο γενικό πλαίσιο των διανυσμάτων ελέγχου της ABEL. Υπάρχουν πολλές και καλά εδραιωμένες δημοσιεύσεις πάνω στον έλεγχο των ψηφιακών συσκευών. Ένα καλό σημείο εκκίνησης για μελέτη είναι το βιβλίο του McCluskey που εκδόθηκε το 1986. Η δημιουργία ενός συνόλου διανυσμάτων ελέγχου που ελέγχουν πλήρως ένα μεγάλο κύκλωμα, όπως μια διάταξη PLD, είναι μια δουλειά που είναι καλύτερο να την αναλαμβάνει ένα πρόγραμμα. Υπάρχει τουλάχιστον μία εταιρία που όλη της η δραστηριότητα εστιάζεται σε προγράμματα που δημιουργούν αυτόματα διανύσματα ελέγχου για τον έλεγχο των PLD (ACUGEN Software, Inc., Nasua, NH 03063, Η.Π.Α., www.acugen.com).

Προβλήματα για εξάσκηση

4.1 Χρησιμοποιώντας τις μεταβλητές ΣΠΑΣΙΚΛΑΣ, ΣΧΕΔΙΑΣΤΗΣ, ΑΠΟΤΥΧΙΑ, και ΜΕΛΕΤΟΥΣΕ, γράψτε μια λογική παράσταση η οποία να είναι 1 για τους επιτυχημένους σχεδιαστές που δε μελετούσαν ποτέ και για τους σπασίκλες που μελετούσαν συνεχώς.

4.2 Αποδείξτε τα θεωρήματα T2-T5 χρησιμοποιώντας την τέλεια επαγωγή.

4.3 Αποδείξτε τα θεωρήματα T1 T3′ και T5′ χρησιμοποιώντας την τέλεια επαγωγή.

4.4 Αποδείξτε τα θεωρήματα T6-T9 χρησιμοποιώντας την τέλεια επαγωγή.

4.5 Σύμφωνα με το θεώρημα του DeMorgan, το συμπλήρωμα του X+Y·Z είναι X′·Y′+Z′. Και όμως, για XYZ = 110, και οι δύο συναρτήσεις έχουν τιμή 1. Πώς μπορεί μια συνάρτηση και το συμπλήρωμά της να έχουν τιμή 1 για τον ίδιο συνδυασμό εισόδων; Πού είναι το λάθος;

4.6 Χρησιμοποιήστε τα θεωρήματα της άλγεβρας μεταγωγής για να απλοποιήσετε τις παρακάτω λογικές συναρτήσεις:

(α) $F = W \cdot X \cdot Y \cdot Z \cdot (W \cdot X \cdot Y \cdot Z' + W \cdot X' \cdot Y \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y' \cdot Z)$

(β) $F = A \cdot B + A \cdot B \cdot C' \cdot D + A \cdot B \cdot D \cdot E' + A \cdot B \cdot C' \cdot E + C' \cdot D \cdot E$

(γ) $F = M \cdot N \cdot O + Q' \cdot P' \cdot N' + P \cdot R \cdot M + Q' \cdot O \cdot M \cdot P' + M \cdot R$

4.7 Γράψτε τον πίνακα αληθείας για κάθε μία από τις παρακάτω λογικές συναρτήσεις:

(α) $F = X' \cdot Y + X' \cdot Y' \cdot Z$ (β) $F = W' \cdot X + Y' \cdot Z' + X' \cdot Z$

(γ) $F = W + X' \cdot (Y' + Z)$ (δ) $F = A \cdot B + B' \cdot C + C' \cdot D + D' \cdot A$

(ε) $F = V \cdot W + X' \cdot Y' \cdot Z$ (στ) $F = (A' + B' + C \cdot D) \cdot (B + C' + D \cdot E')$

(ζ) $F = (W \cdot X)' \cdot (Y' + Z')'$ (η) $F = (((A + B)' + C')' + D)'$

(θ) $F = (A' + B + C) \cdot (A + B' + D') \cdot (B + C' + D') \cdot (A + B + C + D)$

4.8 Γράψτε τον πίνακα αληθείας για κάθε μία από τις παρακάτω λογικές συναρτήσεις:

(α) $F = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z + X \cdot Y' \cdot Z$ (β) $F = M' \cdot N' + M \cdot P + N' \cdot P$

(γ) $F = A \cdot B + A \cdot B' \cdot C' + A' \cdot B \cdot C$ (δ) $F = A' \cdot B \cdot (C \cdot B \cdot A' + B \cdot C')$

(ε) $F = X \cdot Y \cdot (X' \cdot Y \cdot Z + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' + X' \cdot Y' \cdot Z)$ (στ) $F = M \cdot N + M' \cdot N' \cdot P'$

(ζ) $F = (A + A') \cdot B + B \cdot A \cdot C' + C \cdot (A + B') \cdot (A' + B)$ (η) $F = X \cdot Y' + Y \cdot Z + Z' \cdot X$

4.9 Γράψτε το κανονικό άθροισμα και γινόμενο για κάθε μία από τις παρακάτω λογικές συναρτήσεις:

(α) $F = \Sigma_{X,Y}(1, 2)$ (β) $F = \Pi_{A,B}(0, 1, 2)$

(γ) $F = \Sigma_{A,B,C}(2, 4, 6, 7)$ (δ) $F = \Pi_{W,X,Y}(0, 1, 3, 4, 5)$

(ε) $F = X + Y' \cdot Z'$ (στ) $F = V' + (W' \cdot X)'$

4.10 Γράψτε το κανονικό άθροισμα και γινόμενο για κάθε μία από τις παρακάτω λογικές συναρτήσεις:

(α) $F = \Sigma_{X,Y,Z}(0, 3)$ (β) $F = \Pi_{A,B,C}(1, 2, 4)$

(γ) $F = \Sigma_{A,B,C,D}(1, 2, 5, 6)$ (δ) $F = \Pi_{M,N,P}(0, 1, 3, 6, 7)$

(ε) $F = X' + Y \cdot Z' + Y \cdot Z'$ (στ) $F = A'B + B'C + A$

4.11 Αν το κανονικό άθροισμα για μια λογική συνάρτηση *n* εισόδων είναι επίσης ελάχιστο άθροισμα, πόσα κυριολεκτικά υπάρχουν σε κάθε όρο γινομένου του αθροίσματος; Είναι δυνατόν να υπάρχουν άλλα ελάχιστα γινόμενα σε αυτή την περίπτωση;

4.12 Δώστε δύο λόγους που εξηγούν γιατί το κόστος των αντιστροφέων δεν περιλαμβάνεται στον ορισμό του "ελαχίστου" για τη λογική ελαχιστοποίηση.

4.13 Χρησιμοποιώντας χάρτες Karnaugh, βρείτε μια παράσταση ελαχίστου αθροίσματος γινομένων για κάθε μία από τις παρακάτω λογικές συναρτήσεις. Υποδείξτε τα διακεκριμένα κελιά με τιμή 1 σε κάθε χάρτη.

(α) $F = \Sigma_{X,Y,Z}(1, 3, 5, 6, 7)$ (β) $F = \Sigma_{W,X,Y,Z}(1, 4, 5, 6, 7, 9, 14, 15)$

(γ) $F = \Pi_{W,X,Y}(0, 1, 3, 4, 5)$ (δ) $F = \Sigma_{W,X,Y,Z}(0, 2, 5, 7, 8, 10, 13, 15)$

(ε) $F = \Pi_{A,B,C,D}(1, 7, 9, 13, 15)$ (στ) $F = \Sigma_{A,B,C,D}(1, 4, 5, 7, 12, 14, 15)$

4.14 Βρείτε μια παράσταση ελαχίστου γινομένου αθροισμάτων για κάθε μία από τις συναρτήσεις της Άσκησης 4.13 με τη μέθοδο της Ενότητας 4.3.6.

4.15 Βρείτε μια παράσταση ελαχίστου γινομένου αθροισμάτων για τη συνάρτηση για κάθε μία από τις παρακάτω εικόνες και συγκρίνετε το κόστος της

με την παράσταση ελαχίστου αθροίσματος γινομένων που βρήκατε πριν: (α) Εικόνα 4-27, (β) Εικόνα 4-29, (γ) Εικόνα 4-33.

4.16 Χρησιμοποιώντας χάρτες Karnaugh, βρείτε μια παράσταση ελαχίστου αθροίσματος γινομένων για κάθε μία από τις ακόλουθες λογικές συναρτήσεις. Υποδείξτε τα διακεκριμένα κελιά με τιμή 1 σε κάθε χάρτη.

(α) $F = \Sigma_{A,B,C}(0, 1, 2, 4)$ (β) $F = \Sigma_{W,X,Y,Z}(1, 4, 5, 6, 11, 12, 13, 14)$
(γ) $F = \Pi_{A,B,C}(1, 2, 6, 7)$ (δ) $F = \Sigma_{W,X,Y,Z}(0, 1, 2, 3, 7, 8, 10, 11, 15)$
(ε) $F = \Sigma_{W,X,Y,Z}(1, 2, 4, 7, 8, 11, 13, 14)$ (στ) $F = \Pi_{A,B,C,D}(1, 3, 4, 5, 6, 7, 9, 12, 13, 14)$

4.17 Βρείτε μια παράσταση ελαχίστου γινομένου αθροισμάτων για κάθε συνάρτηση της Άσκησης 4.16 με τη μέθοδο της Ενότητας 4.3.6.

4.18 Βρείτε το πλήρες άθροισμα για τις λογικές συναρτήσεις της Άσκησης 4.16(δ) και (ε).

4.19 Χρησιμοποιώντας χάρτες Karnaugh, βρείτε μια παράσταση ελαχίστου αθροίσματος γινομένων για κάθε μία από τις ακόλουθες λογικές συναρτήσεις. Υποδείξτε τα διακεκριμένα κελιά με τιμή 1 σε κάθε χάρτη.

(α) $F = \Sigma_{W,X,Y,Z}(0, 1, 3, 5, 14) + d(8, 15)$ (β) $F = \Sigma_{W,X,Y,Z}(0, 1, 2, 8, 11) + d(3, 9, 15)$
(γ) $F = \Sigma_{A,B,C,D}(1, 5, 9, 14, 15) + d(11)$ (δ) $F = \Sigma_{A,B,C,D}(1, 5, 6, 7, 9, 13) + d(4, 15)$
(ε) $F = \Sigma_{W,X,Y,Z}(3, 5, 6, 7, 13) + d(1, 2, 4, 12, 15)$

4.20 Επαναλάβετε την Άσκηση 4.19, βρίσκοντας μια παράσταση ελαχίστου γινομένου αθροισμάτων για κάθε λογική συνάρτηση.

4.21 Για κάθε λογική συνάρτηση στις δύο προηγούμενες ασκήσεις, προσδιορίστε κατά πόσον η παράσταση ελαχίστου αθροίσματος γινομένων ισούται με την παράσταση ελαχίστου γινομένου αθροισμάτων. Επίσης, συγκρίνετε το κόστος του κυκλώματος για την υλοποίηση κάθε μίας από τις δύο παραστάσεις.

4.22 Για κάθε μία από τις ακόλουθες λογικές παραστάσεις, βρείτε όλους τους στατικούς κινδύνους στο αντίστοιχο κύκλωμα AND-OR και OR-AND δύο επιπέδων και σχεδιάστε ένα κύκλωμα χωρίς κινδύνους που να υλοποιεί την ίδια λογική συνάρτηση.

(α) $F = W \cdot X + W'Y'$ (β) $F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y$
(γ) $F = W' \cdot Y + X' \cdot Y' + W \cdot X \cdot Z$ (δ) $F = W' \cdot X + Y' \cdot Z + W \cdot X \cdot Y \cdot Z + W \cdot X' \cdot Y \cdot Z'$
(ε) $F = (W + X + Y) \cdot (X' + Z')$ (στ) $F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z)$
(ζ) $F = (W + Y + Z') \cdot (W + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z)$

4.23 Γράψτε ένα σύνολο διανυσμάτων ελέγχου σε ABEL για τον ανιχνευτή πρώτων αριθμών του Πίνακα 4-17.

4.24 Γράψτε ένα δομικό πρόγραμμα (οντότητα και αρχιτεκτονική) σε VHDL για το κύκλωμα συναγερμού της Εικόνας 4.19.

4.25 Επαναλάβετε την Άσκηση 4.24 χρησιμοποιώντας το στιλ περιγραφής "ροής δεδομένων".

4.26 Επαναλάβετε την Άσκηση 4.24 χρησιμοποιώντας μια διεργασία και ένα στιλ περιγραφής συμπεριφοράς.

4.27 Γράψτε ένα δομικό πρόγραμμα σε VHDL που να αντιστοιχεί στο λογικό κύκλωμα που βασίζεται σε πύλες NAND το οποίο φαίνεται στην Εικόνα 5.17.

Ασκήσεις

4.28 Σχεδιάστε ένα μη τετριμμένο λογικό κύκλωμα το οποίο να περιέχει ένα βρόχο ανάδρασης, αλλά που θα έχει μια έξοδο η οποία εξαρτάται μόνο από την τρέχουσα είσοδό του.

4.29 Αποδείξτε το συνδυαστικό θεώρημα T10 χωρίς να χρησιμοποιήσετε τέλεια επαγωγή, αλλά θεωρώντας ως αληθή τα θεωρήματα T1-T9 και T1′-T9′.

4.30 Δείξτε ότι το συνδυαστικό θεώρημα T10 είναι απλώς μια ειδική περίπτωση του θεωρήματος κοινής συναίνεσης (T11) που χρησιμοποιείται μαζί με το θεώρημα κάλυψης (T9).

4.31 Αποδείξτε ότι $(X+Y')\cdot Y=X\cdot Y$ *χωρίς* να χρησιμοποιήσετε τέλεια επαγωγή. Μπορείτε να θεωρήσετε ότι τα θεωρήματα T1-T11 και T1′-T11′ είναι αληθή.

4.32 Αποδείξτε ότι $(X+Y)\cdot(X'+Z)=X\cdot Z+X'\cdot Y$ *χωρίς* να χρησιμοποιήσετε τέλεια επαγωγή. Μπορείτε να θεωρήσετε ότι τα θεωρήματα T1-T11 και T1′-T11′ είναι αληθή.

4.33 Δείξτε ότι μια πύλη AND *n* εισόδων μπορεί να αντικατασταθεί από (*n*-1) πύλες AND δύο εισόδων. Ισχύει η ίδια πρόταση για τις πύλες NAND; Αιτιολογήστε την απάντησή σας.

4.34 Πόσοι διαφορετικοί φυσικοί τρόποι υπάρχουν για να υλοποιήσετε τη συνάρτηση $V \cdot W \cdot X \cdot Y \cdot Z$ χρησιμοποιώντας τέσσερις πύλες AND δύο εισόδων (4/4 ενός 74x08); Αιτιολογήστε την απάντησή σας.

4.35 Χρησιμοποιήστε την άλγεβρα μεταγωγής για να αποδείξετε ότι αν συνδέσετε μεταξύ τους δύο εισόδους μιας πύλης AND ή OR των (*n*+1) εισόδων η πύλη συμπεριφέρεται ως πύλη *n* εισόδων.

4.36 Αποδείξτε τα θεωρήματα του DeMorgan (T13 και T13′) χρησιμοποιώντας την πεπερασμένη επαγωγή.

4.37 Ποιο λογικό σύμβολο προσεγγίζει καλύτερα την εσωτερική υλοποίηση μιας πύλης NOR τύπου TTL στην Εικόνα 4-4, εκείνο του (γ) ή εκείνο του (δ); Γιατί;

4.38 Χρησιμοποιήστε τα θεωρήματα της άλγεβρας μεταγωγής για να ξαναγράψετε την παρακάτω παράσταση χρησιμοποιώντας όσο το δυνατόν λιγότερες αντιστροφές (επιτρέπονται τα συμπληρώματα παραστάσεων σε παρενθέσεις):

$$B' \cdot C + A \cdot C \cdot D' + A' \cdot C + E \cdot B' + E \cdot (A + C) \cdot (A' + D')$$

4.39 Αποδείξτε το αληθές ή το ψευδές των παρακάτω προτάσεων:

(α) Έστω A και B *μεταβλητές* της άλγεβρας μεταγωγής. Τότε, $A\cdot B=0$ και $A+B=1$ συνεπάγεται $A=B'$.

(β) Έστω X και Y *παραστάσεις* της άλγεβρας μεταγωγής. Τότε, $X\cdot Y=0$ και $X+Y=1$ συνεπάγεται $X=Y'$.

4.40 Αποδείξτε τα θεωρήματα επέκτασης του Shannon. (*Υπόδειξη:* Μην παρασυρθείτε. Είναι εύκολο.)

γενικευμένα θεωρήματα επέκτασης του Shannon

4.41 Τα θεωρήματα επέκτασης του Shannon είναι δυνατόν να γενικευτούν ώστε να "αφαιρούν" όχι μόνο μία αλλά *i* μεταβλητές, έτσι ώστε μια λογική συνάρτηση να μπορεί να αναπαρασταθεί ως άθροισμα ή γινόμενο 2^i όρων. Διατυπώστε τα γενικευμένα θεωρήματα επέκτασης του Shannon.

4.42 Δείξτε πώς τα γενικευμένα θεωρήματα επέκτασης του Shannon καταλήγουν στις αναπαραστάσεις κανονικού αθροίσματος και κανονικού γινομένου των λογικών συναρτήσεων.

4.43 Μια *πύλη αποκλειστικού OR* (*XOR*) είναι μια πύλη δύο εισόδων της οποίας η έξοδος είναι 1 αν και μόνο αν μόνο μία από τις εισόδους της είναι 1. Γράψτε τον πίνακα αληθείας, την παράσταση αθροίσματος γινομένων, και το αντίστοιχο κύκλωμα AND-OR για τη συνάρτηση αποκλειστικού OR.

πύλη αποκλειστικού OR (*XOR*)

4.44 Από την άποψη της άλγεβρας μεταγωγής, ποια είναι η λειτουργία μιας πύλης XOR δύο εισόδων της οποίας οι είσοδοι είναι συνδεδεμένες μεταξύ τους; Με ποιον τρόπο θα διέφερε η συμπεριφορά εξόδου μιας πραγματικής πύλης XOR;

4.45 Ένας σχεδιαστής, αφού ολοκληρώσει τη σχεδίαση και τη σύνθεση ενός ψηφιακού συστήματος με SSI, διαπιστώνει ότι απαιτείται ένας ή περισσότεροι αντιστροφείς. Ωστόσο, οι μοναδικές διαθέσιμες πύλες στο σύστημα είναι μια OR 3 εισόδων, μια AND 2 εισόδων, και μια XOR 2 εισόδων. Πώς θα πρέπει να υλοποιήσει ο σχεδιαστής τη συνάρτηση αντιστροφέα χωρίς να προσθέσει άλλο ολοκληρωμένο κύκλωμα;

4.46 Κάθε σύνολο τύπων λογικών πυλών που μπορεί να υλοποιήσει οποιαδήποτε λογική συνάρτηση λέγεται *πλήρες σύνολο* λογικών πυλών. Για παράδειγμα, οι πύλες AND δύο εισόδων, οι πύλες OR δύο εισόδων, και οι αντιστροφείς είναι πλήρη σύνολα, επειδή οποιαδήποτε λογική συνάρτηση είναι δυνατόν να διατυπωθεί ως άθροισμα γινομένων μεταβλητών και των συμπληρωμάτων τους, ενώ πύλες AND και OR με οποιονδήποτε αριθμό εισόδων είναι δυνατόν να δημιουργηθούν από πύλες 2 εισόδων. Οι πύλες NAND 2 εισόδων αποτελούν ένα πλήρες σύνολο λογικών πυλών; Αποδείξτε την απάντησή σας.

πλήρες σύνολο

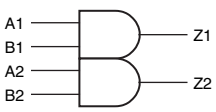
4.47 Οι πύλες NOR 2 εισόδων αποτελούν ένα πλήρες σύνολο λογικών πυλών; Αποδείξτε την απάντησή σας.

4.48 Οι πύλες XOR 2 εισόδων αποτελούν ένα πλήρες σύνολο λογικών πυλών; Αποδείξτε την απάντησή σας.

4.49 Ορίστε μια πύλη δύο εισόδων, που να μην είναι NAND, NOR, ή XOR, η οποία σχηματίζει ένα πλήρες σύνολο λογικών πυλών αν επιτρέπονται οι σταθερές είσοδοι 0 και 1. Αποδείξτε την απάντησή σας.

4.50 Μερικοί θεωρούν ότι υπάρχουν *τέσσερις* βασικές λογικές συναρτήσεις, οι AND, OR, NOT, και *BUT*. Η Εικόνα Χ4.50 είναι ένα πιθανό σύμβολο μιας *πύλης BUT* τεσσάρων εισόδων. Επινοήστε μια χρήσιμη μη τετριμμένη συνάρτηση προς εκτέλεση από την πύλη BUT. Η συνάρτηση θα πρέπει να έχει κάποια σχέση με το όνομα (BUT). Έχετε υπόψη ότι, λόγω της συμμετρίας του συμβόλου, η συνάρτηση θα είναι συμμετρική ως προς τις εισόδους A και B κάθε τμήματος και ως προς τα τμήματα 1 και 2. Περιγράψτε τη συνάρτηση της πύλης BUT που δημιουργήσατε και γράψτε τον πίνακα αληθείας.

BUT
πύλη BUT



Εικόνα Χ4.50

4.51 Γράψτε λογικές παραστάσεις για τις εξόδους Z1 και Z2 της πύλης BUT που σχεδιάσατε στο προηγούμενο πρόβλημα και φτιάξτε το αντίστοιχο λογικό διάγραμμα χρησιμοποιώντας πύλες AND και OR και αντιστροφείς.

4.52 Οι περισσότεροι σπουδαστές δεν έχουν πρόβλημα στη χρήση του θεωρήματος Τ8 για τους επιμέρους πολλαπλασιασμούς στις λογικές παραστά-

σεις, αλλά πολλοί δυσκολεύονται να χρησιμοποιήσουν το θεώρημα T8′ για τις επιμέρους προσθέσεις σε μια λογική παράσταση. Πώς μπορεί να χρησιμοποιηθεί η δυικότητα για να ξεπεραστεί αυτό το πρόβλημα;

4.53 Πόσες διαφορετικές λογικές συναρτήσεις *n* μεταβλητών υπάρχουν;

4.54 Πόσες διαφορετικές λογικές συναρτήσεις F(X,Y) 2 μεταβλητών υπάρχουν; Γράψτε μια απλοποιημένη αλγεβρική παράσταση για κάθε μία από αυτές.

αυτοδυική λογική συνάρτηση ⊕

4.55 Η *αυτοδυική λογική συνάρτηση* είναι μια συνάρτηση F τέτοια ώστε $F = F^D$. Ποια από τις ακόλουθες συναρτήσεις είναι αυτοδυική; (Το σύμβολο ⊕ υποδηλώνει την πράξη αποκλειστικού OR (XOR).)

(α) $F = X$ (β) $F = SX,Y,Z(0, 3, 5, 6)$

(γ) $F = X \cdot Y' + X' \cdot Y$ (δ) $F = W \cdot (X \oplus Y \oplus Z) + W' \cdot (X \oplus Y \oplus Z)'$

(ε) Μια συνάρτηση F των 7 μεταβλητών τέτοια ώστε F = 1 αν και μόνο αν 4 ή περισσότερες από τις μεταβλητές είναι 1.

(στ) Μια συνάρτηση F των 10 μεταβλητών τέτοια ώστε F = 1 αν και μόνο αν 5 ή περισσότερες από τις μεταβλητές είναι 1.

4.56 Πόσες αυτοδυικές λογικές συναρτήσεις *n* μεταβλητών εισόδου υπάρχουν; (*Υπόδειξη:* Λάβετε υπόψη τη δομή του πίνακα αληθείας μιας αυτοδυικής λογικής συνάρτησης.)

4.57 Αποδείξτε ότι οποιαδήποτε λογική συνάρτηση *n* εισόδων $F(X_1,..., X_n)$ η οποία μπορεί να γραφτεί με τη μορφή $F = X_1 \cdot G(X_2,..., X_n) + X_1' \cdot G^D(X_2,..., X_n)$ είναι αυτοδυική.

4.58 Θεωρώντας ότι μια αναστρέφουσα πύλη έχει καθυστέρηση διάδοσης ίση με 5 ns και ότι μια μη αναστρέφουσα πύλη έχει καθυστέρηση διάδοσης ίση με 8 ns, συγκρίνετε τις ταχύτητες των κυκλωμάτων τής Εικόνας 4-24(α), (γ), και (δ).

4.59 Βρείτε τις παραστάσεις ελαχίστου γινομένου αθροισμάτων για τις λογικές συναρτήσεις των Εικόνων 4-27 και 4-29.

4.60 Χρησιμοποιήστε την άλγεβρα μεταγωγής για να δείξετε ότι οι λογικές συναρτήσεις που λαμβάνονται στην Άσκηση 4.59 ισούνται με τις συναρτήσεις AND-OR που λαμβάνονται στις Εικόνες 4-27 και 4-29.

4.61 Προσδιορίστε κατά πόσον οι παραστάσεις γινομένου αθροισμάτων που λαμβάνονται "με την εκτέλεση των επιμέρους προσθέσεων" στα ελάχιστα αθροίσματα των Εικόνων 4-27 και 4-29 είναι ελάχιστες.

4.62 Αποδείξτε ότι ο κανόνας για το συνδυασμό 2^i κελιών με τιμή 1 μεταξύ τους σε ένα χάρτη Karnaugh είναι αληθής, χρησιμοποιώντας τα αξιώματα και τα θεωρήματα της άλγεβρας μεταγωγής.

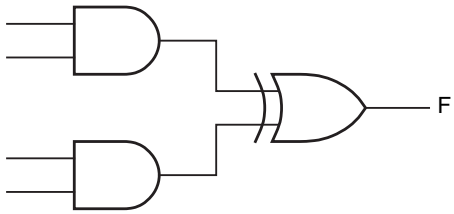
μη πλεονάζον άθροισμα

4.63 Ένα *μη πλεονάζον άθροισμα* μιας λογικής συνάρτησης F είναι ένα άθροισμα πρωταρχικών όρων της F τέτοιο ώστε, αν ένας πρωταρχικός όρος διαγραφεί, το άθροισμα να μην ισούται πλέον με F. Αυτό ακούγεται σαν να πρόκειται για ελάχιστο άθροισμα, αλλά το μη πλεονάζον άθροισμα δεν είναι απαραίτητα ελάχιστο. Για παράδειγμα, το ελάχιστο άθροισμα της συνάρτησης στην Εικόνα 4-35 έχει μόνο τρεις όρους γινομένου, αλλά υπάρχει ένα μη πλεονάζον άθροισμα με τέσσερις όρους γινομένου. Βρείτε το μη πλεονάζον άθροισμα και σχεδιάστε ένα χάρτη της συνάρτησης, τοποθετώντας σε κύκλο μόνο τους πρωταρχικούς όρους του μη πλεονάζοντος αθροίσματος.

4.64 Βρείτε άλλη μια λογική συνάρτηση στην Ενότητα 4.3 που να έχει ένα ή περισσότερα μη ελάχιστα μη πλεονάζοντα αθροίσματα και σχεδιάστε το χάρτη της, τοποθετώντας σε κύκλο μόνο τους πρωταρχικούς όρους του μη πλεονάζοντος αθροίσματος.

4.65 Σχεδιάστε ένα χάρτη Karnaugh και αποδώστε μεταβλητές στις εισόδους του κυκλώματος AND-XOR της Εικόνας Χ4.65, έτσι ώστε η έξοδός του να είναι $F=\Sigma_{W,X,Y,Z}(6,7,12,13)$. Σημειώστε ότι η πύλη εξόδου είναι XOR 2 εισόδων και όχι OR.

Εικόνα Χ4.65



4.66 Ένα κύκλωμα "συγκριτή" των 3 bit λαμβάνει δύο αριθμούς των 3 bit, $P = P_2P_1P_0$ και $Q = Q_2Q_1Q_0$. Σχεδιάστε ένα κύκλωμα ελαχίστου αθροίσματος γινομένων το οποίο να παράγει έξοδο 1 αν και μόνο αν $P > Q$.

4.67 Βρείτε τις ελάχιστες παραστάσεις αθροίσματος γινομένων πολλών εξόδων για $F=\Sigma_{X,Y,Z}(0,1,2)$, $G=\Sigma_{X,Y,Z}(1,4,6)$, και $H=\Sigma_{X,Y,Z}(0,1,2,4,6)$.

4.68 Αποδείξτε κατά πόσον η ακόλουθη παράσταση αποτελεί ελάχιστο άθροισμα. Κάντε το με τον ευκολότερο δυνατό τρόπο (αλγεβρικά, χωρίς να χρησιμοποιήσετε χάρτες).

$$F = T' \cdot U \cdot V \cdot W \cdot X + T' \cdot U \cdot V' \cdot X \cdot Z + T' \cdot U \cdot W \cdot X \cdot Y' \cdot Z$$

4.69 Σύμφωνα με το βιβλίο, ένας πίνακας αληθείας ή το ισοδύναμό του είναι το σημείο εκκίνησης στις παραδοσιακές μεθόδους συνδυαστικής ελαχιστοποίησης. Ο ίδιος ο χάρτης Karnaugh περιέχει τις ίδιες πληροφορίες με τον πίνακα αληθείας. Με δεδομένη μια παράσταση αθροίσματος γινομένων, μπορούμε να γράψουμε τα 1 που αντιστοιχούν σε κάθε όρο γινομένου κατευθείαν στο χάρτη, χωρίς να αναπτύξουμε ένα ρητό πίνακα αληθείας ή μια λίστα ελαχίστων όρων, και μετά να προχωρήσουμε με τη διαδικασία ελαχιστοποίησης του πίνακα. Με αυτόν τον τρόπο, βρείτε μια παράσταση ελαχίστου αθροίσματος γινομένων για κάθε μία από τις ακόλουθες λογικές συναρτήσεις:

(α) $F = X' \cdot Z + X \cdot Y + X \cdot Y' \cdot Z$ (β) $F = A' \cdot C' \cdot D + B' \cdot C \cdot D + A \cdot C' \cdot D + B \cdot C \cdot D$

(γ) $F = W \cdot X \cdot Z' + W \cdot X' \cdot Y \cdot Z + X \cdot Z$ (δ) $F = (X' + Y') \cdot (W' + X' + Y) \cdot (W' + X + Z)$

(ε) $F = A \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C' + A \cdot B \cdot D + A' \cdot C \cdot D + B \cdot C \cdot D'$

4.70 Επαναλάβετε το Πρόβλημα 4.69, βρίσκοντας μια παράσταση ελαχίστου γινομένου αθροισμάτων για κάθε λογική συνάρτηση.

4.71 Κατασκευάστε την παράσταση ελαχίστου γινομένου αθροισμάτων για τη συνάρτηση του ανιχνευτή πρώτων αριθμών ψηφίου BCD της Εικόνας 4.37. Προσδιορίστε κατά πόσον η παράσταση αλγεβρικά ισούται με την

παράσταση ελαχίστου αθροίσματος γινομένων ή όχι, και εξηγήστε το αποτέλεσμα.

χάρτης Karnaugh 5 μεταβλητών

4.72 Ένας χάρτης Karnaugh για μια συνάρτηση 5 μεταβλητών είναι δυνατόν να σχεδιαστεί όπως στην Εικόνα Χ4.72. Σε έναν τέτοιο χάρτη, τα κελιά που καταλαμβάνουν την ίδια σχετική θέση στους υποχάρτες V=0 και V=1 θεωρούνται ότι είναι γειτονικά. (Πολλά παραδείγματα χαρτών Karnaugh 5 μεταβλητών εμφανίζονται στις Ενότητες 7.4.4 και 7.4.5.) Βρείτε μια παράσταση ελαχίστου αθροίσματος γινομένων για κάθε μία από τις ακόλουθες συναρτήσεις χρησιμοποιώντας ένα χάρτη 5 μεταβλητών

(α) $F = \Sigma_{V,W,X,Y,Z}(5, 7, 13, 15, 16, 20, 25, 27, 29, 31)$

(β) $F = \Sigma_{V,W,X,Y,Z}(0, 7, 8, 9, 12, 13, 15, 16, 22, 23, 30, 31)$

(γ) $F = \Sigma_{V,W,X,Y,Z}(0, 1, 2, 3, 4, 5, 10, 11, 14, 20, 21, 24, 25, 26, 27, 28, 29, 30)$

(δ) $F = \Sigma_{V,W,X,Y,Z}(0, 2, 4, 6, 7, 8, 10, 11, 12, 13, 14, 16, 18, 19, 29, 30)$

(ε) $F = \Pi_{V,W,X,Y,Z}(4, 5, 10, 12, 13, 16, 17, 21, 25, 26, 27, 29)$

(στ) $F = \Sigma_{V,W,X,Y,Z}(4, 6, 7, 9, 11, 12, 13, 14, 15, 20, 22, 25, 27, 28, 30) + d(1, 5, 29, 3\ 1)$

4.73 Επαναλάβετε το Πρόβλημα 4.72, βρίσκοντας μια παράσταση ελαχίστου γινομένου αθροισμάτων για κάθε λογική συνάρτηση.

χάρτης Karnaugh 6 μεταβλητών

4.74 Ένας χάρτης Karnaugh για μια συνάρτηση 6 μεταβλητών είναι δυνατόν να σχεδιαστεί όπως φαίνεται στην Εικόνα Χ4.74. Σε έναν τέτοιο χάρτη, τα κελιά που καταλαμβάνουν την ίδια σχετική θέση σε γειτονικούς υποχάρτες θεωρούνται ότι είναι γειτονικά. Ελαχιστοποιήστε τις ακόλουθες συναρτήσεις χρησιμοποιώντας χάρτες 6 μεταβλητών:

(α) $F = \Sigma_{U,V,W,X,Y,Z}(1, 5, 9, 13, 21, 23, 29, 31, 37, 45, 53, 61)$

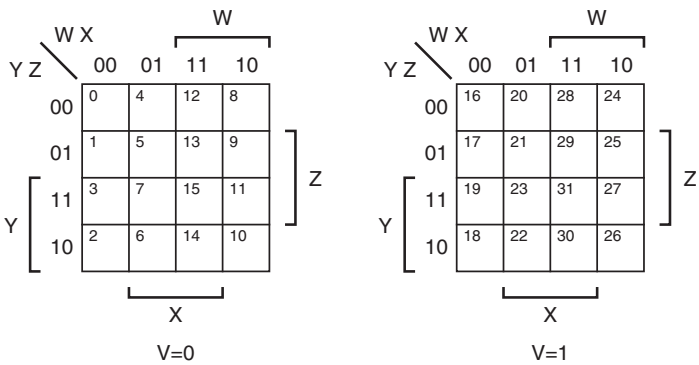
(β) $F = \Sigma_{U,V,W,X,Y,Z}(0, 4, 8, 16, 24, 32, 34, 36, 37, 39, 40, 48, 50, 56)$

(γ) $F = \Sigma_{U,V,W,X,Y,Z}(2, 4, 5, 6, 12–21, 28–31, 34, 38, 50, 51, 60–63)$

4.75 Υπάρχουν $2n$ m-διάστατοι υποκύβοι σε έναν n-διάστατο κύβο για την τιμή $m=n-1$. Δείξτε τις αναπαραστάσεις τους με κείμενο και τους αντίστοιχους όρους γινομένου. (Μπορείτε να χρησιμοποιήσετε αποσιωπητικά όπου απαιτείται, π.χ. 1,2,...,n.)

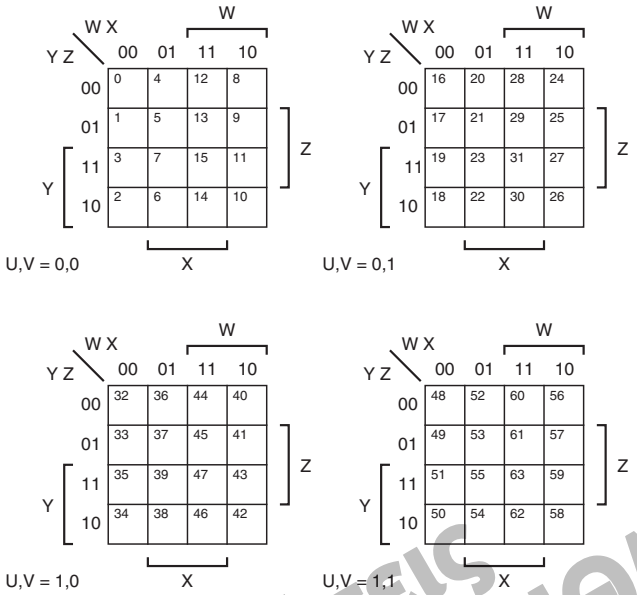
4.76 Υπάρχει ακριβώς ένας m-διάστατος υποκύβος ενός n-διάστατου κύβου για την τιμή $m=n$. Η αναπαράστασή του με κείμενο είναι xx...xx. Γράψτε τον όρο γινομένου που αντιστοιχεί σε αυτόν τον κύβο.

4.77 Το πρόγραμμα C του Πίνακα 4-9 χρησιμοποιεί τη μνήμη με μη αποδοτικό τρόπο, επειδή κατανέμει μνήμη για ένα μέγιστο αριθμό κύβων σε κά-



Εικόνα Χ4.72

Εικόνα Χ4.74



θε επίπεδο, ακόμη και αν αυτό το μέγιστο δε χρησιμοποιείται ποτέ. Επανασχεδιάστε το πρόγραμμα έτσι ώστε οι πίνακες `cubes` και `used` να είναι μονοδιάστατοι πίνακες και κάθε επίπεδο να χρησιμοποιεί μόνο όσες καταχωρίσεις πίνακα χρειάζονται. (*Υπόδειξη:* Μπορείτε και πάλι να αποδίδετε κύβους σειριακά, αλλά θα παρακολουθείτε το σημείο έναρξης στον πίνακα για κάθε επίπεδο.)

4.78 Συναρτήσει του m, πόσες φορές ανακαλύπτεται ξανά κάθε διακριτός m-διάστατος κύβος στον Πίνακα 4.9 μόνο και μόνο για να βρεθεί στον εσωτερικό βρόχο και να απορριφθεί; Προτείνετε μερικούς τρόπους για να εξαλειφθεί αυτή η μειωμένη αποδοτικότητα.

4.79 Ο τρίτος βρόχος `for` στον Πίνακα 4-9 προσπαθεί να συνδυάσει όλους τους m-διάστατους κύβους σε ένα δεδομένο επίπεδο με όλους τους άλλους m-διάστατους κύβους στο συγκεκριμένο επίπεδο. Στην πραγματικότητα, μόνο οι m-διάστατοι κύβοι που έχουν x στις ίδιες θέσεις είναι δυνατόν να συνδυαστούν μεταξύ τους, για να μπορέσετε να περιορίσετε τον αριθμό των επαναλήψεων του βρόχου χρησιμοποιώντας μια πιο εξεζητημένη δομή δεδομένων. Σχεδιάστε μια δομή δεδομένων που να διαχωρίζει τους κύβους σε ένα δεδομένο επίπεδο σύμφωνα με τη θέση των x τους και προσδιορίστε το μέγιστο μέγεθος που απαιτείται για διάφορα στοιχεία της δομής δεδομένων. Ξαναγράψτε αντίστοιχα τον Πίνακα 4-9.

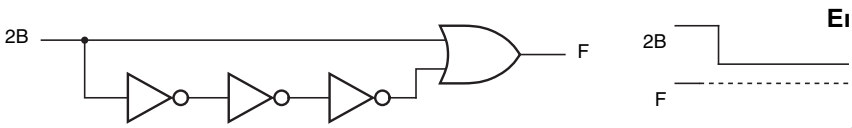
4.80 Εκτιμήστε κατά πόσον η εξοικονόμηση που επιτυγχάνεται στις επαναλήψεις εσωτερικού βρόχου του Προβλήματος 4.80 αντισταθμίζει την επιβάρυνση που συνεπάγεται η συντήρηση μιας πιο σύνθετης δομής δεδομένων. Επιχειρήστε να κάνετε εύλογες υποθέσεις για το πώς κατανέμονται οι κύβοι σε κάθε επίπεδο και δείξτε πώς επηρεάζονται τα αποτελέσματά σας από αυτές τις υποθέσεις.

4.81 Βελτιστοποιήστε τη συνάρτηση `Oneone` του Πίνακα 4-8. Μια προφανής βελτιστοποίηση είναι να εγκαταλείπεται ο βρόχος νωρίς, αλλά υπάρχουν κι άλλες βελτιστοποιήσεις που εξαλείφουν πλήρως το βρόχο `for`.

Μια από αυτές βασίζεται σε αναζήτηση μέσα σε πίνακες, ενώ μια άλλη χρησιμοποιεί έναν έξυπνο υπολογισμό που περιλαμβάνει συμπλήρωση, χρήση του αποκλειστικού OR, και πρόσθεση.

4.82 Επεκτείνετε το πρόγραμμα C του Πίνακα 4-9 ώστε να χειρίζεται τις αδιάφορες συνθήκες. Εφοδιάστε το με άλλη μία δομή δεδομένων, την `dc[MAX_VARS+1][MAX_CUBES]`, που να υποδηλώνει κατά πόσον ένας δεδομένος κύβος περιέχει μόνο "αδιάφορες" τιμές, και ενημερώνετέ τη καθώς διαβάζονται και δημιουργούνται κύβοι.

4.83 (*Κύκλωμα Hamlet.*) Συμπληρώστε το διάγραμμα χρονισμού και εξηγήστε τη λειτουργία του κυκλώματος της Εικόνας Χ4.83. Από πού πήρε το όνομά του το κύκλωμα;



Εικόνα Χ4.83

4.84 Αποδείξτε ότι ένα κύκλωμα AND-OR δύο επιπέδων που αντιστοιχεί στο πλήρες άθροισμα μιας λογικής συνάρτησης είναι πάντα χωρίς κινδύνους.

4.85 Βρείτε μια λογική συνάρτηση τεσσάρων μεταβλητών της οποίας η υλοποίηση ελαχίστου αθροίσματος γινομένων δεν είναι χωρίς κινδύνους, αλλά για την οποία υπάρχει μια υλοποίηση αθροίσματος γινομένων χωρίς κινδύνους με λιγότερους όρους γινομένου από το πλήρες άθροισμα.

4.86 Αρχίζοντας με τις εντολές `WHEN` στο πρόγραμμα ABEL του Πίνακα 4-14, συντάξτε τις λογικές εξισώσεις των μεταβλητών `X4` έως `X10` του προγράμματος. Εξηγήστε τις τυχόν διαφορές ανάμεσα στα αποτελέσματά σας και τις εξισώσεις του Πίνακα 4-15

4.87 Σχεδιάστε ένα διάγραμμα κυκλώματος που να αντιστοιχεί στις ελάχιστες εξισώσεις αθροίσματος γινομένων δύο επιπέδων για το κύκλωμα του συναγερμού, όπως δίνεται στον Πίνακα 4-12. Σε κάθε είσοδο και έξοδο αντιστροφέα, πύλης AND, και πύλης OR, γράψτε ένα ζεύγος αριθμών (*t0, t1*) όπου *t0* είναι ο αριθμός ελέγχου από τον Πίνακα 4-25, που εντοπίζει αστοχία κολλήματος στην τιμή 0 στη συγκεκριμένη γραμμή, και *t1* είναι ο αριθμός ελέγχου που εντοπίζει αστοχία κολλήματος στην τιμή 1.

4.88 Γράψτε ένα πρόγραμμα (οντότητα και αρχιτεκτονική) σε VHDL, σε στιλ ροής δεδομένων, που να αντιστοιχεί στο κύκλωμα πλήρους αθροιστή της Εικόνας 5-86.

4.89 Χρησιμοποιώντας την οντότητα που σχεδιάσατε στο Πρόβλημα 4.88, γράψτε ένα δομικό πρόγραμμα σε VHDL για τον αθροιστή κυμάτωσης των 4 bit χρησιμοποιώντας τη δομή της Εικόνας 5.87.

4.90 Χρησιμοποιώντας την οντότητα που ορίσατε στο Πρόβλημα 4.88, γράψτε ένα δομικό πρόγραμμα σε VHDL για τον αθροιστή κυμάτωσης των 16 bit σύμφωνα με τις γενικές κατευθύνσεις της Εικόνας 5.87. Χρησιμοποιήστε την εντολή `generate` για να δημιουργήσετε τους αθροιστές των 16 bit και τις συνδέσεις των σημάτων τους.

4.91 Ξαναγράψτε την αρχιτεκτονική του ανιχνευτή πρώτων αριθμών του Πίνακα 4.63 χρησιμοποιώντας μια εντολή `while`.