Ελένη Αικατερίνη Λελίγκου
Σταμάτης Βολιώτης
Αθανάσιος Κακαρούντας

Η ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ ΣΤΟ ΕΡΓΑΣΤΗΡΙΟ

Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα
www.kallipos.gr

HEALLINK
Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΕΛΕΝΗ ΑΙΚΑΤΕΡΙΝΗ ΛΕΛΙΓΚΟΥ
Επ. Καθηγήτρια ΤΕΙ Στερεάς Ελλάδας

ΣΤΑΜΑΤΗΣ ΒΟΛΙΩΤΗΣ
Καθηγητής ΤΕΙ Στερεάς Ελλάδας

ΑΘΑΝΑΣΙΟΣ ΚΑΚΑΡΟΥΝΤΑΣ
Επ. Καθηγητής Πανεπιστήμιο Θεσσαλίας

Η ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ ΣΤΟ ΕΡΓΑΣΤΗΡΙΟ

Η Λογική Σχεδίαση στο Εργαστήριο

Συγγραφή

Ελένη Αικατερίνη Λελίγκου

Σταμάτης Βολιώτης

Αθανάσιος Κακαρούντας

Κριτικός αναγνώστης

Νικόλαος Ασημάκης

Συντελεστές έκδοσης

Τεχνική Επεξεργασία:

Ευμορφία Κρήτου, Ευδοξία Κοκκίνου

ISBN: 978-960-603-256-1

ΣΥΝΔΕΣΜΟΣ ΕΛΛΗΝΙΚΩΝ ΑΚΑΔΗΜΑΪΚΩΝ ΒΙΒΛΙΟΘΗΚΩΝ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

ΠΕΡΙΕΧΟΜΕΝΑ

Λίστα πινάκων

Πίνακας Ακρωνυμίων

ASIC	application specific integrated circuits
CLB	Configurable Logic Blocks
CPLD	Complex Programmable Logic Devices
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
HDL	hardware description languages
IC	INTEGRATED CIRCUIT
LUT	Look-Up Table
MUX	Multiplexer
PCB	Printed Circuit Board
PLD	Programmable Logic Devices
PSM	PROGRAMMABLE SWITCHING MATRIX
SSI/MSI	small/medium-scale integration
VHDL	V_{HSIC} -Very High Speed Integrated Circuit - Hardware Description Language
VLSI	Very Large-Scale Integration

ΠΡΟΛΟΓΟΣ

Τα ψηφιακά συστήματα αποτελούν τη βάση της σύγχρονης τεχνολογίας. Σχεδόν κάθε συσκευή που τροφοδοτείται με ηλεκτρικό ρεύμα, και όχι μόνο, περιέχει ηλεκτρονικά εξαρτήματα όπου κυριαρχούν τα ψηφιακά κυκλώματα και συστήματα. Έχοντας ως δομικό στοιχείο το τρανζίστορ, τα ψηφιακά συστήματα υλοποιούν λειτουργίες επεξεργασίας και ελέγχου που ανταποκρίνονται σε όλο το φάσμα των δραστηριοτήτων μας. Η "ηλεκτρονική" εποχή που ζούμε η οποία χαρακτηρίζεται από τα κινητά τηλέφωνα, τις τηλεοράσεις υψηλής ευκρίνειας, τους φορητούς υπολογιστές, τα tablets, το internet, το facebook και πολλά άλλα, θα ήταν αδιανόητη χωρίς την τεράστια πρόοδο που έχει επιτευχθεί στην τεχνολογία των ψηφιακών συστημάτων.

Τί είναι όμως αυτό που κάνει τα ψηφιακά συστήματα τόσο σημαντικά για την πρόοδο της τεχνολογίας και κατ' επέκταση τη πρόοδο του πολιτισμού μας; Είναι απλά η δυνατότητα ταχύτατης υλοποίησης πολύπλοκων αλγορίθμων. Η άλγεβρα Boole, το μαθηματικό υπόβαθρο της θεωρίας των ψηφιακών συστημάτων, μας έδωσε τη δυνατότητα περιγραφής, σε μορφή απλών μαθηματικών συναρτήσεων, πολύπλοκων λογικών προτάσεων. Οποιαδήποτε μαθηματική πράξη, οποιαδήποτε διαδικασία επεξεργασίας δεδομένων, οποιαδήποτε διαδικασία ελέγχου, οποιαδήποτε λειτουργία εν γένει αποτελεί έναν αλγόριθμο ο οποίος μπορεί να περιγραφεί με λογικές προτάσεις και άρα να μαθηματικοποιηθεί. Παράλληλα η τεχνολογία του τρανζίστορ μας έδωσε τη δυνατότητα υλοποίησης κυκλωμάτων που εκτελούν τις βασικές πράξεις (τελεστές) της άλγεβρας Boole. Με αυτό τον τρόπο είναι δυνατή η υλοποίηση με κυκλώματα των μαθηματικών εκφράσεων που περιγράφουν πολύπλοκες λογικές προτάσεις και άρα αλγορίθμους. Η μεγάλη πρόοδος της τεχνολογίας των ολοκληρωμένων κυκλωμάτων τα τελευταία χρόνια οδήγησε στην υλοποίηση σε ένα μόνο τσιπ δεκάδων δισεκατομμυρίων τρανζίστορ με δυνατότητα λειτουργίας σε υψηλές ταχύτητες όπως αυτή εκφράζεται με συχνότητες λειτουργίας GHz, πράγμα που συνολικά αντιστοιχεί σε τεράστια υπολογιστική ισχύ. Η εξέλιξη αυτή έχει διαμορφώσει το σήμερα και θα συνεχίσει να διαμορφώνει το μέλλον.

Η υλοποίηση των ψηφιακών συστημάτων είναι μια εξαιρετικά πολύπλοκη διαδικασία. Περιλαμβάνει τη διαδικασία περιγραφής των κυκλωμάτων που αποτελούν το σύστημα, την προσομοίωση της λειτουργίας, τη σύνθεση των κυκλωμάτων, την τοποθέτηση των κυκλωμάτων στην επιφάνεια πυριτίου και τη διασύνδεση αυτών, τη διαδικασία προσομοίωσης και επαλήθευσης μετά την τοποθέτηση, την παραγωγή μασκών, την κατασκευή, τη διαδικασία ελέγχου ορθής λειτουργίας, την τοποθέτηση σε περίβλημα καθώς και πολλές επιμέρους, έως ότου ένα τσιπ να είναι έτοιμο να ενταχθεί σε κάποιο προϊόν. Η πολυπλοκότητα αυτή μαζί με το γεγονός του μεγάλου αριθμού τρανζίστορ που μπορεί να περιέχει ένα ολοκληρωμένο κύκλωμα κάνει απαραίτητη την ανάγκη αυτοματοποίησης της κάθε επιμέρους διαδικασίας. Για το λόγο αυτό έχει αναπτυχθεί ειδικό λογισμικό που διευκολύνει τους μηχανικούς, αυξάνοντας την παραγωγικότητά τους, να ανταποκριθούν στις δυνατότητες της τεχνολογίας. Στο πλαίσιο αυτό έχουν προταθεί γλώσσες περιγραφής υλικού, όπως η VHDL, και πλέον ο σχεδιασμός ψηφιακών συστημάτων αντιστοιχεί στη δημιουργία αντίστοιχου κώδικα. Όμως ο σχεδιαστής/μηχανικός θα πρέπει να κατανοεί τα κυκλώματα που καλείται να περιγράψει σε κώδικα. Επίσης, οικονομικοί λόγοι αλλά και η ανάγκη για μείωση του χρόνου παραγωγής λειτουργικών ψηφιακών ολοκληρωμένων κυκλωμάτων οδήγησε στην ανάπτυξη εναλλακτικών μεθοδολογιών υλοποίησης ανάλογα με το βαθμό προεργασίας στη διαδικασία της υλοποίησης. Στο πλαίσιο αυτό εντάσσεται η τεχνολογία των FPGAs, που αντιστοιχεί σε τσιπ τα οποία μπορούν να προγραμματιστούν ώστε να φιλοξενήσουν ψηφιακά κυκλώματα και συστήματα. Η χρήση αυτών επιταχύνει σημαντικά την υλοποίηση ψηφιακών συστημάτων και τείνει να καθιερωθεί ως βασική επιλογή.

Καθώς τα ψηφιακά συστήματα παίζουν ουσιαστικό ρόλο στη σύγχρονη τεχνολογία πολλά εκπαιδευτικά συγγράμματα έχουν προταθεί για να καλύψουν την απαιτούμενη θεωρία και να κάνουν κατανοητή τη χρήση τους σε φοιτητές και επαγγελματίες. Κάθε νέο σύγγραμμα προσπαθεί να καλύψει κάτι επιπλέον από την ύλη των ψηφιακών συστημάτων. Στο παρόν σύγγραμμα "*Η λογική σχεδίαση στο εργαστήριο*" γίνεται προσπάθεια να παρουσιαστεί όλη η διαδρομή του ψηφιακού ή λογικού σχεδιασμού με έμφαση στην εξάσκηση των φοιτητών στο σχεδιασμό και στην υλοποίηση πραγματικών κυκλωμάτων. Από τη βασική θεωρία του δυαδικού συστήματος, την άλγεβρα Boole, τα βασικά κυκλώματα, συνδυαστικά και ακολουθιακά, αλλά και τη γλώσσα περιγραφής VHDL μέχρι και τον τρόπο υλοποίησης με τον προγραμματισμό του FPGA. Η παρουσίαση της σχετικής ύλης γίνεται παράλληλα με μια ομάδα εργαστηριακών ασκήσεων για περαιτέρω κατανόηση και εμπέδωση της θεωρίας αλλά και της χρήσης των ψηφιακών κυκλωμάτων και της υλοποίησής τους στο εργαστήριο. Ως αποτέλεσμα, το παρόν σύγγραμμα αποτελεί ένα χρήσιμο εκπαιδευτικό και εργαστηριακό βοήθημα για τη διδασκαλία της λογικής σχεδίασης.

Σπύρος Νικολαΐδης

Αν. καθηγητής ΑΠ

Κεφάλαιο 1ο
Τεχνολογίες και εργαλεία σχεδίασης και υλοποίησης ψηφιακών κυκλωμάτων

1.1 Εισαγωγή

Η σύγχρονη εποχή χαρακτηρίζεται από την μετάβαση από την αναλογική-μηχανική ηλεκτρική τεχνολογία στην ψηφιακή τεχνολογία, μετάβαση η οποία πραγματοποιείται με ραγδαίους ρυθμούς και γι' αυτό χαρακτηρίζεται συχνά και ως Ψηφιακή Επανάσταση σε αντιπαραβολή με την Αγροτική και μετέπειτα τη Βιομηχανική Επανάσταση που χαρακτήρισαν τους περασμένους αιώνες. Εμμέσως, ο όρος αυτός αναφέρεται επίσης στις σαρωτικές αλλαγές τις οποίες επέφερε η πληροφορική και η τεχνολογία των επικοινωνιών κατά τη διάρκεια του δεύτερου μισού του 20ου αιώνα. Η πρόοδος της τεχνολογίας στον τομέα των ηλεκτρονικών έχει οδηγήσει στην ανάπτυξη μιας σειράς ψηφιακών συστημάτων που χρησιμοποιούνται σε διάφορα είδη συσκευών με εφαρμογή σε διάφορους τομείς όπως οι επιτραπέζιοι ή φορητοί υπολογιστές, οι τηλεπικοινωνίες, οι ηλεκτρονικές υπηρεσίες, η διασκέδαση κ.α. Οι τεχνολογικές αυτές εξελίξεις αντίστοιχα έχουν τεράστιο κοινωνικό και οικονομικό αντίκτυπο οδηγώντας στη σημερινή εποχή στην ανάδυση της αποκαλούμενης Κοινωνίας της Πληροφορίας, η οποία χαρακτηρίζεται από την δυνατότητα των ανθρώπων να ανταλλάσσουν και να μεταφέρουν πληροφορίες ελεύθερα, με ολοένα αυξανόμενη ταχύτητα και πληθώρα μέσων και να έχουν άμεση πρόσβαση σε γνώσεις που θα ήταν δύσκολο ή αδύνατο να βρεθούν στο παρελθόν. Η κύρια κινητήρια δύναμη πίσω από αυτές τις εξελίξεις ήταν η μαζική παραγωγή και η ευρεία χρήση των ψηφιακών λογικών κυκλωμάτων και οι τεχνολογίες που πηγάζουν από αυτήν και βρίσκουν εφαρμογή

Τα ψηφιακά ηλεκτρονικά συστήματα είναι αυτά που χειρίζονται ή επεξεργάζονται μεγέθη ή δεδομένα που παριστάνονται με ψηφιακό τρόπο (δηλαδή, λαμβάνουν διακριτές τιμές σε μία ορισμένη περιοχή τιμών), σε αντίθεση με τα αναλογικά συστήματα τα οποία χειρίζονται μεγέθη ή δεδομένα που παριστάνονται με αναλογικό τρόπο (δηλαδή, λαμβάνουν συνεχείς τιμές). Η ψηφιακή επεξεργασία αφορά μετασχηματισμό των σημάτων μεταφοράς δεδομένων που το ψηφιακό σύστημα λαμβάνει στις εισόδους του, ενώ το αποτέλεσμα παρέχεται στις εξόδους του συστήματος. Με τον όρο επεξεργασία συνήθως αναφερόμαστε σε ενέργειες όπως αποθήκευση δεδομένων (προσωρινή ή μόνιμη), εκτέλεση πράξεων (λογικών και αριθμητικών), επιλογή (πχ. μιας θέσης μνήμης), απαρίθμηση κλπ..

Βέβαια, στην γενικότερη περίπτωση εφαρμογών η επικοινωνία με το πραγματικό περιβάλλον γίνεται μέσω αναλογικών σημάτων. Ένα αναλογικό σήμα είναι συνεχής συνάρτηση του χρόνου με συνεχές πλάτος επίσης. Αναλογικά σήματα δημιουργούνται όταν μία φυσική κυματομορφή, όπως ένα ακουστικό ή οπτικό κύμα μετατρέπεται σε ηλεκτρικό σήμα. Για το λόγο αυτό η αξιοποίηση των ψηφιακών ηλεκτρονικών βασίζεται στην μετατροπή των αναλογικών σημάτων αρχικά σε διακριτά μέσω της ονομαζόμενης διαδικασίας του κβαντισμού και μετά σε ψηφιακά μέσω της διαδικασίας της κωδικοποίησης. Στα παραδείγματα περιλαμβάνονται το μικρόφωνο, που μετατρέπει ηχητικές μεταβολές πίεσης σε αντίστοιχες μεταβολές τάσης ή ρεύματος και το φωτοηλεκτρικό κύτταρο, που κάνει το ίδιο για μεταβολές έντασης του φωτός. Όταν κάθε δείγμα ενός σήματος διακριτού χρόνου είναι κβαντισμένο (δηλαδή το πλάτος του επιτρέπει να λάβει μόνο ένα πεπερασμένο σύνολο διακριτών τιμών) και στη συνέχεια κωδικοποιημένο, το τελικό σήμα αναφέρεται σαν ψηφιακό σήμα. Η έξοδος από ένα ψηφιακό υπολογιστή είναι ένα παράδειγμα ψηφιακού σήματος. Φυσικά, ένα αναλογικό σήμα μπορεί να μετατραπεί σε ψηφιακή μορφή με δειγματοληψία στο χρόνο, κβαντισμό και κωδικοποίησή του. Τα ψηφιακά σήματα στα περισσότερα σύγχρονα ψηφιακά συστήματα είναι δυαδικά σήματα ηλεκτρικής τάσης, χρησιμοποιούν, δηλαδή, δύο διακριτές τιμές ή καταστάσεις, οι οποίες αναφέρονται και ως στάθμες. Η υψηλή στάθμη και η χαμηλή στάθμη ενός δυαδικού σήματος αντιστοιχούν σε δύο διακριτές (συγκεκριμένες) τιμές τάσης (π.χ. 0 και 3.3 Volts, αντίστοιχα), με τις ενδιάμεσες τιμές να είναι πρακτικά μη υπαρκτές. Οι δύο καταστάσεις ή στάθμες εκφράζονται με δύο λογικές τιμές, τη λογική τιμή 0 και τη λογική τιμή 1, αντίστοιχα.

Στο παρόν βιβλίο θα εστιάσουμε στις βασικές αρχές σχεδίασης και ανάλυσης ψηφιακών συστημάτων χρησιμοποιώντας διαφορετικές εργαστηριακές διατάξεις και τεχνολογίες. Στην παρούσα εισαγωγή θα γίνει μία πολύ σύντομη περίληψη των βασικών αρχών σχεδίασης και λειτουργίας των ψηφιακών ηλεκτρονικών συστημάτων που χρησιμοποιούνται στις εργαστηριακές ασκήσεις που παρουσιάζονται στα επόμενα κεφάλαια. Η αρχική θεμελιώδης θεωρία αφορά τις ιδιότητες του δυαδικού αριθμητικού συστήματος στο οποίο βασίζεται όλη η σχεδίαση των ψηφιακών ηλεκτρονικών κυκλωμάτων. Επέκταση της χρήσης του δυαδικού συστήματος

για την εκτέλεση αριθμητικών πράξεων αποτελεί η άλγεβρα Boole, που συσχετίζει τον κόσμο της πληροφορίας με τον κόσμο των δυαδικών αριθμών εκφράζοντας την αλήθεια λογικών εκφράσεων και τις σχέσεις που τις εκφράζουν και οδηγεί στην ανάπτυξη λογικών συναρτήσεων μεταξύ των μεταβλητών ενός προβλήματος λογικής (και κατ' επέκταση μιας εφαρμογής πληροφορικής). Στη συνέχεια γίνεται αναφορά στα δομικά συστατικά στοιχεία της μικροηλεκτρονικής τεχνολογίας, που βασίζονται στις ιδιότητες των ημιαγωγών και τη σχεδίαση της διάταξης του τρανζίστορ και των συνδυασμό αυτών για την σχεδίαση των ονομαζόμενων λογικών πυλών, που χρησιμοποιούνται για την υλοποίηση λογικών συναρτήσεων μέσω ηλεκτρονικών κυκλωμάτων. Με κατάλληλη συνδεσμολογία και αξιοποιώντας κατάλληλες τεχνικές σχεδίασης και απλοποίησης παρουσιάζονται οι συνθετότερες μονάδες που μπορεί να αναπτυχθούν με βάση τις λογικές πύλες και οδηγούν στα αποκαλούμενα συνδυαστικά και ακολουθιακά κυκλώματα. Συνεχίζοντας με ιεραρχικά βήματα παρουσιάζουμε τη σύνθεση πιο πολύπλοκων κυκλωμάτων για εκτέλεση συνθετότερων λειτουργιών, όπως οι αριθμητικές πράξεις και οι λειτουργίες της κωδικοποίησης ή αποκωδικοποίησης συνδυασμών, αλλά και η διαχείριση, αποθήκευση και μεταφορά συνθετότερων μονάδων πληροφορίας, όπως τα Bytes οι ψηφιακές λέξεις κλπ. Κατάλληλος συνδυασμός των τελευταίων οδηγεί τελικά στις πλέον σύνθετες επεξεργαστικές μονάδες, που χρησιμοποιούνται στις σύγχρονες εφαρμογές ψηφιακών ηλεκτρονικών συστημάτων όπως των ψηφιακών μικροελεγκτών και μικροεπεξεργαστών. Τέλος γίνεται μια συνολική επισκόπηση της εξέλιξης των σχετικών τεχνολογιών και των εργαλείων σχεδίασης (EDA, Electronic Design Automation) που είναι διαθέσιμα σήμερα για τη σχεδίαση και ανάλυση σύνθετων ψηφιακών κυκλωμάτων. Έτσι παρουσιάζονται οι τεχνολογίες μικρής κλίμακας ολοκλήρωσης (small scale integration, SSI) στην οποία βασίζονται διαθέσιμες ψηφιακές ηλεκτρονικές διατάξεις που μπορούμε να χρησιμοποιήσουμε στο εργαστήριο έως πολύ μεγάλης κλίμακας ολοκλήρωσης (very large scale integration, VLSI) τις οποίες μπορούμε να αξιοποιήσουμε για σχεδίαση πολύπλοκων ψηφιακών επεξεργαστών, αλλά και των προγραμματιζόμενων ηλεκτρονικών διατάξεων με την τεχνολογία Field Programmable Gate Array. Η εισαγωγή ολοκληρώνεται με την παρουσίαση βασικών σχεδιαστικών εργαλείων για την αξιοποίηση των παραπάνω τεχνολογιών και τα οποία χρησιμοποιούνται στις εργαστηριακές ασκήσεις που περιλαμβάνονται στα επόμενα κεφάλαια.

Το δεύτερο κεφάλαιο περιλαμβάνει εργαστηριακές ασκήσεις που μπορούν να εκτελεστούν με χρήση διαθέσιμων στο εμπόριο διατάξεων μικρής κλίμακας ολοκλήρωσης (της αποκαλούμενης τεχνολογίας TTL) για τον πειραματισμό με τη σχεδίαση και ανάλυση ψηφιακών συνδυαστικών κυκλωμάτων. Για την εκτέλεση του εργαστηριακού μέρους απαιτείται κατάλληλη εφαρμογή της θεωρίας, για να εφαρμοστούν οι κατάλληλες αρχές σχεδίασης και ανάλυσης ψηφιακών κυκλωμάτων με τις οποίες προσεγγίζονται και τα αναμενόμενα αποτελέσματα από την εκτέλεση της άσκησης.

Το τρίτο κεφάλαιο περιλαμβάνει εργαστηριακές ασκήσεις που επίσης μπορούν να εκτελεστούν με χρήση διαθέσιμων στο εμπόριο διατάξεων μικρής κλίμακας ολοκλήρωσης (της αποκαλούμενης τεχνολογίας TTL) για τον πειραματισμό με τη σχεδίαση και ανάλυση ψηφιακών ακολουθιακών κυκλωμάτων αντίστοιχα. Αντίστοιχα απαιτείται εφαρμογή των κατάλληλων τεχνικών σχεδιασμού και ανάλυσης από τη θεωρία.

Στο τρίτο κεφάλαιο γίνεται αξιοποίηση της γλώσσας VHDL για την περιγραφή ψηφιακών συνδυαστικών και ακολουθιακών κυκλωμάτων. Στόχος είναι η εμπέδωση των σχετικών αρχών ανάλυσης και σύνθεσης και ο συνδυασμός τους με τα διαθέσιμα αυτοματοποιημένα εργαλεία σύνθεσης και προσομοίωσης λειτουργίας για τον σχεδιασμό και επαλήθευση ορθής λειτουργίας ψηφιακών συστημάτων.

Τέλος οι τεχνικές σχεδίασης που παρουσιάστηκαν παραπάνω συνδυάζονται για την σχεδίαση ψηφιακών συστημάτων με χρήση της τεχνολογίας των προγραμματιζόμενων ηλεκτρονικών διατάξεων με την τεχνολογία Field Programmable Gate Array στο τελευταίο κεφάλαιο του βιβλίου. Με χρήση εμπορικά διαθέσιμων ηλεκτρονικών διατάξεων και των σχετικών εργαλείων ανάπτυξης γίνεται περιγραφή πειραματικών διατάξεων που μπορούν να χρησιμοποιηθούν στο εργαστήριο για την εμπέδωση των παραπάνω αρχών σχεδίασης μέσω επιλεγμένων ασκήσεων και παραδειγμάτων.

1.2 Βασικές αρχές σχεδίασης ψηφιακών συστημάτων

Δυαδικό Σύστημα

Καθώς τα δυαδικά σήματα, όπως αναφέρθηκε, λαμβάνουν δύο διακριτές τιμές, τα μεγέθη ή δεδομένα που συμμετέχουν στην επεξεργασία, που επιτελείται στα ψηφιακά συστήματα, μπορούν να εκφραστούν με το δυαδικό αριθμητικό σύστημα, αντί του κλασσικού δεκαδικού συστήματος. Το δυαδικό αριθμητικό σύστημα χρησιμοποιεί δυαδικά ψηφία που μπορούν να λάβουν τις τιμές 0 και 1. Πιο συγκεκριμένα, το δυαδικό είναι ένα

θεσιακό σύστημα με βάση το δύο. Κάθε ψηφίο ανήκει σε μία τάξη μεγέθους μεγαλύτερη κατά ένα από αυτήν του ψηφίου στα δεξιά του. Έτσι, κάθε ψηφίο ενός δυαδικού αριθμού από δεξιά προς τ' αριστερά δηλώνει μονάδες, δυάδες, τετράδες, οκτάδες κ.ο.κ.

Για παράδειγμα ο δυαδικός αριθμός:

$(1101)_2$

αναπαριστά ποσότητα ίση με:

1 μονάδα ($1x2^0$), 0 δυάδες ($0x2^1$), 1 τετράδα ($1x2^2$) και 1 οκτάδα ($1x2^3$).

Ισούται δηλαδή με τον αριθμό 13 του δεκαδικού συστήματος, εφόσον:

$(1x2^0) + (0x2^1) + (1x2^2) + (1x2^3) = 13$

Λόγω της προφανούς συσχέτισης της δυαδικής αναπαράστασης με τις ιδιότητες των ψηφιακών ηλεκτρονικών κυκλωμάτων όπως αναφέρθηκε προηγουμένως, το δυαδικό σύστημα χρησιμοποιείται εκτεταμένα στους ηλεκτρονικούς υπολογιστές για την αναπαράσταση αριθμητικών δεδομένων. Στο δυαδικό σύστημα ισχύουν όλες οι αριθμητικές πράξεις ακριβώς με την ίδια μεθοδολογία εκτέλεσης, όπως τις γνωρίζουμε στο δεκαδικό σύστημα. Για την αξιοποίηση της ψηφιακής τεχνολογίας όμως απαιτείται κατάλληλος τρόπος αναπαράστασης των αρνητικών καθώς και των μη ακεραίων μεγεθών. Για το λόγο αυτό χρησιμοποιείται η αναπαράσταση προσημασμένων ακεραίων αριθμών στο δυαδικό σύστημα σε μορφή προσημασμένου συμπληρώματος ως προς τη βάση του συστήματος (δηλαδή το δύο) ή τη βάση του συστήματος μειωμένη κατά ένα (δηλαδή το ένα). Άλλα χρησιμοποιούμενα συστήματα είναι το σύστημα κινητής υποδιαστολής, το σύστημα σταθερής υποδιαστολής, η δυαδική κωδικοποίηση δεκαδικού, και άλλα.

Άλγεβρα Boole

Πέραν της αξίας των ψηφιακών συστημάτων για την υλοποίηση αριθμητικών πράξεων στο δυαδικό σύστημα αυτά προσλαμβάνουν ακόμα μεγαλύτερη με την δυνατότητα διαχείρισης της πληροφορίας για την μετεξέλιξή τους από απλές αριθμητικές μηχανές σε επεξεργαστικές μονάδες για το πλήρες εύρος των εφαρμογών πληροφορικής. Σε αυτή την κατεύθυνση οι θεωρητικές βάσεις τέθηκαν από τον μαθηματικό George Boole, (1854) (με την συνδρομή επίσης των Edward Huntington και Claude Elwood Shannon -1938) με την εισαγωγή ενός αλγεβρικού συστήματος (άλγεβρα Boole) για συστηματική αντιμετώπιση προβλημάτων λογικής. Στα Μαθηματικά και την Μαθηματική λογική η Άλγεβρα Μπουλ είναι η υποπεριοχή της άλγεβρας όπου οι τιμές των μεταβλητών είναι οι τιμές αληθείας αληθές και ψευδές, που συνήθως αναπαρίστανται με 1 και 0 αντίστοιχα. Η δίτιμη άλγεβρα Boole είναι ένα αλγεβρικό σύστημα το οποίο περιλαμβάνει δύο στοιχεία (τα οποία συμβολίζουμε συνήθως με '0' και '1') και τρεις τελεστές, τους οποίους συμβολίζουμε με + (OR/H), · (AND/KAI), ~ (NOT/OXI). Συχνά, αντί για το σύμβολο ~ χρησιμοποιούμε το σύμβολο ΄ (~x = x΄). Οι κανόνες για τους τελεστές αυτούς ορίζονται σύμφωνα με τους ακόλουθους πίνακες.

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

x	y	x·y
0	0	0
0	1	0
1	0	0
1	1	1

x	~x=x΄
0	1
1	0

Πίνακας 1.1: Τελεστές της δίτιμης άλγεβρας Boole

Η άλγεβρα Boole χαρακτηρίζεται από μια σειρά αξιώματα, θεωρήματα και ιδιότητες (πχ. αντιμεταθετική, προσεταιριστική κ.λπ.). Στο δεύτερο κεφάλαιο θα παρουσιαστούν οι βασικές αρχές της άλγεβρας Boole και οι τεχνικές αξιοποίησής της στη σχεδίαση και ανάλυση ψηφιακών συστημάτων.

Λογικές Συναρτήσεις

Η εφαρμογή των ιδιοτήτων της Άλγεβρας Boole επεκτείνεται στην περιγραφή των σχέσεων με τη μορφή λογικών συναρτήσεων, που συνδέουν φυσικά σήματα που εμφανίζονται σε πρακτικές εφαρμογές πληροφορικής και στα αντίστοιχα συστήματα. Μια λογική συνάρτηση (ή συνάρτηση Boole) εκφράζει μέσω μιας αλγεβρικής έκφρασης τη λογική σχέση μεταξύ δυαδικών μεταβλητών και αποτελείται από δυαδικές μεταβλητές, τις σταθερές τιμές 0 και 1 και τα σύμβολα τα τριών λογικών πράξεων

π.χ.: F(x,y,z) = x΄y + xz.
Για μια δεδομένη τιμή των δυαδικών μεταβλητών, η συνάρτηση μπορεί να είναι ίση με 0 ή 1. Οπότε, οι πιθανές τιμές της συνάρτησης προκύπτουν από τον προσδιορισμό της δυαδικής τιμής της αλγεβρικής έκφρασης για όλους τους συνδυασμούς των εμπλεκομένων μεταβλητών.Μια λογική συνάρτηση μπορεί να περιγραφεί μέσω πίνακα αληθείας που περιλαμβάνει στο αριστερό μέρος όλους τους δυνατούς συνδυασμούς των δυαδικών μεταβλητών (2^n για n μεταβλητές) και στο δεξί μέρος μια στήλη που περιέχει την τιμή της συνάρτησης για καθένα από τους συνδυασμούς του αριστερού μέρους.
Π.χ. ο πίνακας αλήθειας της συνάρτησης $F(x,y,z) = x'y + xz$, περιλαμβάνει οκτώ (2^3) γραμμές και τέσσερις στήλες (μία στήλη για κάθε μεταβλητή και μία στήλη για τη συνάρτηση):

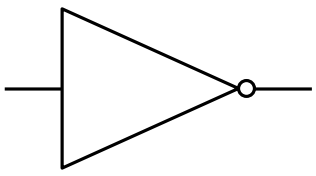
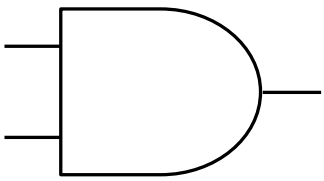
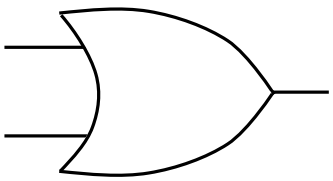
x	*y*	*z*	*F*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

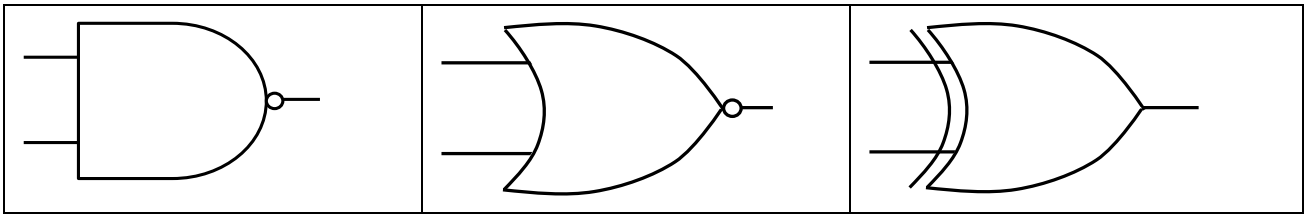
***Πίνακας 1.2: Ο πίνακας αλήθειας της συνάρτησης* F(x,y,z) = x′y + xz**

Στο δεύτερο κεφάλαιο θα παρουσιάσουμε τεχνικές απλοποίησης λογικών συναρτήσεων, οι οποίες τελικά αποδεικνύεται ότι μπορούν να έχουν σημαντική επίπτωση στην πολυπλοκότητα (και άρα το τελικό κόστος) υλοποίησης αυτών μέσω ψηφιακών ηλεκτρονικών κυκλωμάτων (όπως η χρήση του ονομαζόμενου χάρτη Karnaugh).

Λογικές πύλες

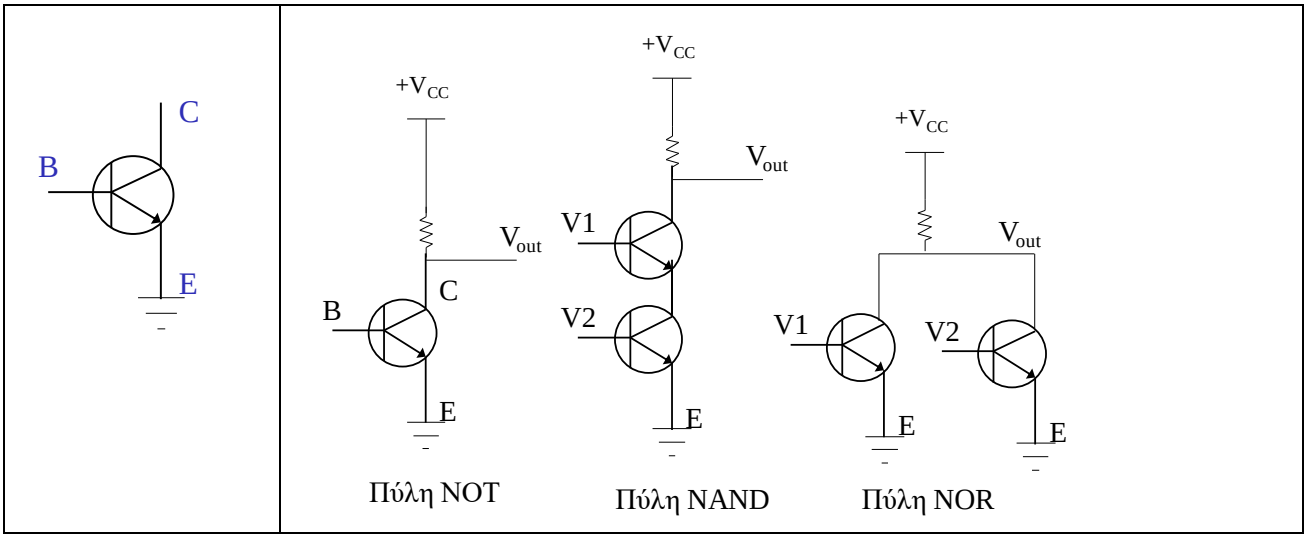
Η συσχέτιση των λογικών συναρτήσεων με τα ψηφιακά ηλεκτρονικά συστήματα βασίζεται στο γεγονός ότι οι λογικές συναρτήσεις είναι δυνατόν να υλοποιηθούν με ηλεκτρονικά λογικά κυκλώματα. Τα ηλεκτρονικά κυκλώματα που μπορούν να εκτελέσουν τις βασικές πράξεις της άλγεβρας Boole ονομάζονται πύλες (gates)
Οι δύο τιμές της άλγεβρας Boole αντιστοιχούν συνήθως σε δύο επίπεδα τάσης (π.χ. το λογικό 1 στην τάση +5V, ενώ το λογικό 0 σε τάση 0 V). Οι λογικές πύλες αντιστοιχούν σε ηλεκτρονικά κυκλώματα τα οποία δέχονται σήματα εισόδου και παράγουν ένα σήμα εξόδου κατ'αντιστοιχία με τη λογική πράξη ή λογική συνάρτηση μεταξύ των σημάτων εισόδου που αυτές υλοποιούν. Οι συνηθέστερες πύλες που συναντούμε στα ψηφιακά ηλεκτρονικά συστήματα παρουσιάζονται στο παρακάτω πίνακα και θα συζητηθούν εκτενέστερα στο 2ο κεφάλαιο. Υπάρχει η δυνατότητα σχεδιασμού και χρησιμοποίησης πυλών πολλαπλών εισόδων.

Αντιστροφέας ή NOT	AND	OR
		
NAND	NOR	XOR



Εικόνα 1.1:Σχεδιασμός Λογικών πυλών

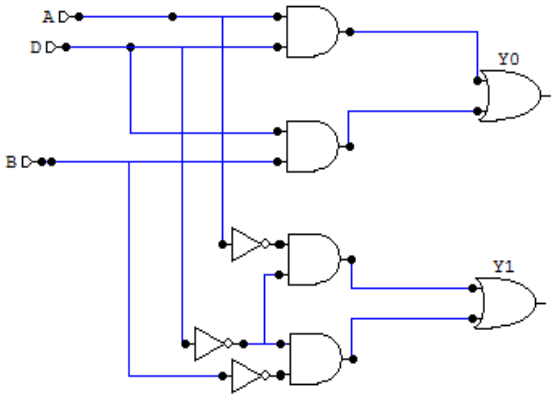
Οι λογικές πύλες υλοποιούνται με διασυνδεδεμένα ηλεκτρονικά στοιχεία που ονομάζονται τρανζίστορ (transistor). Το σημαντικότερο πλεονέκτημα των στοιχείων αυτών, σε σχέση με το χειρισμό δυαδικών σημάτων, αποτελεί το γεγονός ότι έχουν τη δυνατότητα να λειτουργούν ως διακόπτες, παρουσιάζοντας δύο επιτρεπτές καταστάσεις λειτουργίας αντίστοιχες με τις λογικές τιμές 0 και 1. Η τάση στην «πύλη» ελέγχει να περνά ρεύμα από την πηγή (source) στην καταβόθρα (drain). Δεν πρέπει να συγχέεται η «πύλη» του τρανζίστορ με μια λογική πύλη.



Εικόνα 1.2:Σχεδιασμός τρανζίστορ

Συνδυαστικά & ακολουθιακά κυκλώματα

Ένα λογικό κύκλωμα αποτελείται από γραμμές διασύνδεσης λογικών πυλών, που αντιστοιχούν στις διαδρομές των δυαδικών σημάτων και από λογικές πύλες που επιτελούν την επεξεργασία μεταξύ των σημάτων, η οποία δηλώνεται στη λογική συνάρτηση που επιτελείται από κάθε πύλη. Τα ψηφιακά κυκλώματα διακρίνονται σε δύο κατηγορίες: τα συνδυαστικά και τα ακολουθιακά.
Τα λογικά κυκλώματα των οποίων η λογική τιμή της εξόδου ή των εξόδων τους, κάθε χρονική στιγμή, εξαρτάται μόνο από τη λογική τιμή των εισόδων που εφαρμόζεται σε αυτά την ίδια χρονική στιγμή ονομάζονται συνδυαστικά κυκλώματα (combinational circuits).



Εικόνα 1.3: Παράδειγμα συνδυαστικού κυκλώματος

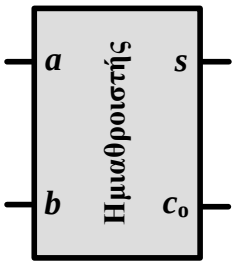
Ένα συνδυαστικό κύκλωμα αποτελείται από μεταβλητές εισόδου, λογικές πύλες και μεταβλητές εξόδου. Για καθένα από τους 2^n πιθανούς συνδυασμούς δυαδικών τιμών στις εισόδους του, υπάρχει μόνο ένας συνδυασμός δυαδικών τιμών στις εξόδους του. Ένα συνδυαστικό κύκλωμα περιγράφεται με m λογικές συναρτήσεις n μεταβλητών (μία συνάρτηση για κάθε μεταβλητή εξόδου) ή με ένα πίνακα αληθείας που δίνει τις τιμές των εξόδων για κάθε δυνατό συνδυασμό των μεταβλητών εισόδου.
Τα συνδυαστικά κυκλώματα μπορούν να χρησιμοποιηθούν για την υλοποίηση οποιουδήποτε σχεδιαστικού προβλήματος με τις προϋποθέσεις που αναφέρθηκαν, αλλά επιπλέον μπορούν να συνδυαστούν σε ιεραρχικές δομές πιο πολύπλοκων δομών, που οδηγούν στη σχεδίαση πιο πολύπλοκων επεξεργαστικών μονάδων, όπως οι παρακάτω:

- Αριθμητικά συνδυαστικά κυκλώματα
  - Πλήρης αθροιστής
  - Ημιαθροιστής
  - Αθροιστής διάδοσης κρατουμένου
  - Αφαιρέτης
- Κωδικοποιητές και αποκωδικοποιητές
- Πολυπλέκτες και αποπολυπλέκτες

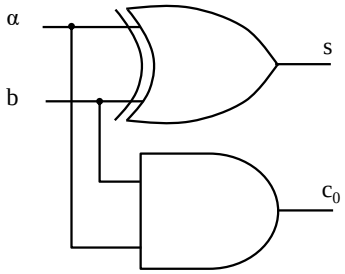
Το βασικό παράδειγμα του δυαδικού αθροιστή παρουσιάζεται παρακάτω και αποτελεί την πρώτη βασική δομική μονάδα που οδηγεί στην εξέλιξη των ψηφιακών υπολογιστών. Οι ψηφιακοί υπολογιστές εκτελούν πολλές διεργασίες επεξεργασίας πληροφορίας, όπως οι αριθμητικές πράξεις. Βασικότερη αριθμητική πράξη για τον υπολογιστή είναι η πρόσθεση δύο δυαδικών ψηφίων (0+0=0, 0+1=1, 1+0=1, 1+1=10). Οι πρώτες τρεις πράξεις παράγουν άθροισμα ενός ψηφίου, αλλά όταν και οι δύο προσθετέοι είναι ίσοι με 1, το δυαδικό άθροισμα αποτελείται από δύο ψηφία. Το πιο σημαντικό ψηφίο του αποτελέσματος αυτού λέγεται κρατούμενο (carry). Το συνδυαστικό κύκλωμα, το οποίο εκτελεί την πρόσθεση δύο ψηφίων ονομάζεται ημιαθροιστής (half-adder) και παρουσιάζεται στη συνέχεια.

a	b	s	c_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Πίνακας 1.3: Αλήθειας ημιαθροιστή

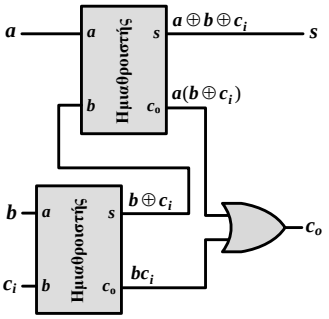


Εικόνα1.4: Συνοπτικό σύμβολο ημιαθροιστή

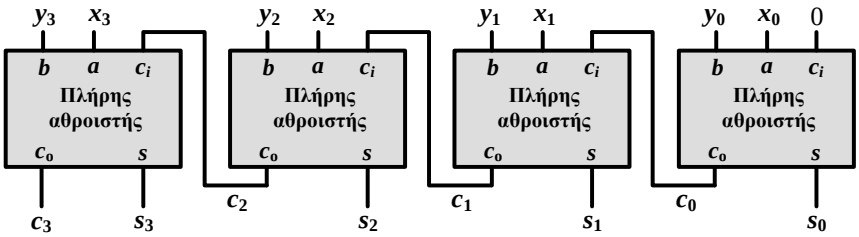


Εικόνα1.5: Συνδυαστικό κύκλωμα ημιαθροιστή

Το κύκλωμα που εκτελεί την πρόσθεση τριών ψηφίων ονομάζεται πλήρης αθροιστής (full-adder) και είναι απαραίτητο ώστε να υπάρχει δυνατότητα για ταυτόχρονη πρόσθεση και του κρατούμενου της προηγούμενης πρόσθεσης. Δυαδικός αφαιρέτης είναι το συνδυαστικό κύκλωμα που εκτελεί τις πράξεις της πρόσθεσης και της αφαίρεσης δυαδικών αριθμών. Με σχεδίαση ιεραρχικού τύπου μπορούμε να σχεδιάσουμε πρώτα τον ημιαθροιστή και τον χρησιμοποιούμε για το σχεδιασμό του αθροιστή. Επέκταση της μεθοδολογίας μπορεί να οδηγήσει στο σχεδιασμό πιο πολύπλοκων κυκλωμάτων για την εκτέλεση σύνθετων αριθμητικών πράξεων μεταξύ πολυψήφιων αριθμών, όπως παρουσιάζεται στο επόμενο σχήμα.



Εικόνα 1.6: Συνδυαστικό κύκλωμα πλήρους αθροιστή

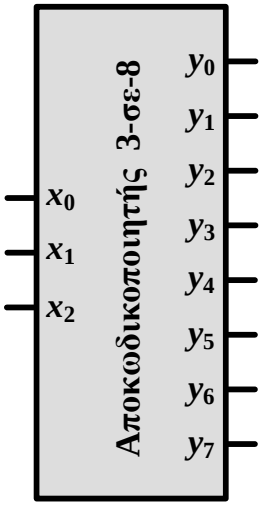


Εικόνα 1.7: Αθροιστής δύο μη προσημασμένων δυαδικών αριθμών τεσσάρων ψηφίων

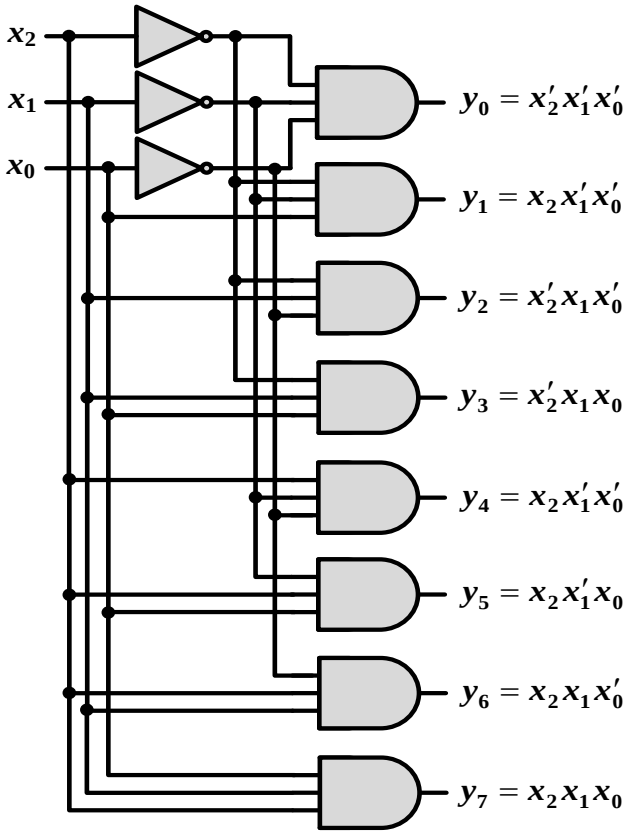
Άλλη ευρεία κατηγορία δομικών μονάδων αποτελούν όπως αναφέρθηκε οι αποκωδικοποιητές δυαδικών συνδυασμών. Οι διακριτές ποσότητες πληροφορίας παριστάνονται στα ψηφιακά συστήματα με χρήση δυαδικών κωδίκων. Ένας δυαδικός κώδικας με n ψηφία μπορεί να παραστήσει μέχρι και 2^n διακριτά στοιχεία κωδικοποιημένης πληροφορίας (ελαχιστόροι). Ο αποκωδικοποιητής (decoder) είναι ένα συνδυαστικό κύκλωμα, που μετατρέπει δυαδικές πληροφορίες n γραμμών εισόδου σε πληροφορίες που λαμβάνονται σε γραμμές εξόδου μέγιστου πλήθους 2^n. Σε έναν αποκωδικοποιητή n-σε-m (δηλαδή με n γραμμές εισόδου και m γραμμές εξόδου, με $m \leq 2^n$), ουσιαστικά ο στόχος είναι η παραγωγή 2^n ή λιγότερων ελαχιστόρων. Στη συνέχεια παρουσιάζεται το παράδειγμα του αποκωδικοποιητή 3-σε-8 που μπορεί να χρησιμοποιηθεί σε εφαρμογές όπου απαιτείται αποκωδικοποίηση οποιουδήποτε κώδικα με τρία ψηφία σε οκτώ εξόδους (μία για κάθε στοιχείο του κώδικα).

x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Πίνακας 1.4: Πίνακας αλήθειας ημιαθροιστή



Εικόνα1.8: Συνοπτικό σύμβολο ημιαθροιστή

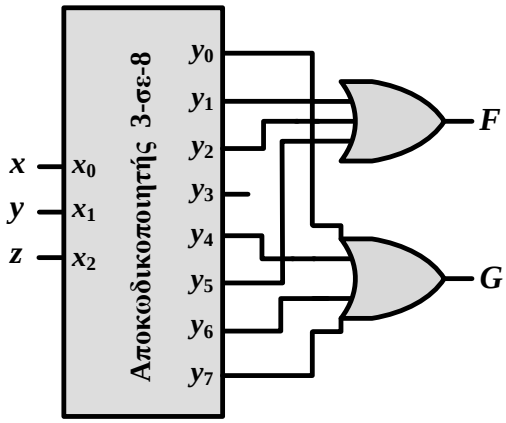


Εικόνα1.9: Συνδυαστικό κύκλωμα ημιαθροιστή

Μία σημαντική εφαρμογή του αποκωδικοποιητή βασίζεται στο γεγονός ότι οποιοδήποτε συνδυαστικό κύκλωμα με n εισόδους και m εξόδους μπορεί να υλοποιηθεί με έναν αποκωδικοποιητή n-σε-2^n και m πύλες Η (OR). Για την υλοποίηση αυτή αρκεί η λογική συνάρτηση του κυκλώματος να είναι εκφρασμένη ως άθροισμα ελαχιστόρων. Για το παράδειγμα πλήρους αθροιστή, που χρησιμοποιήθηκε παραπάνω με την χρήση κωδικοποιητή αρκεί να εκφράσουμε την συνάρτηση του αθροιστή ως ακολούθως:

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

Η παραπάνω έκφραση μπορεί να υλοποιηθεί με χρήση κωδικοποιητή 3-σε-8 όπως φαίνεται παρακάτω.

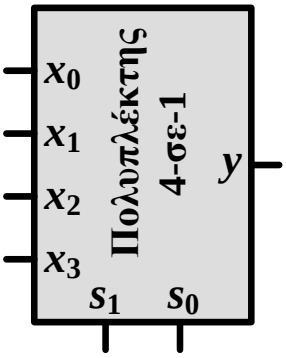


Εικόνα 1.10: Υλοποίηση αθροιστή με χρήση κωδικοποιητή 3-σε-8

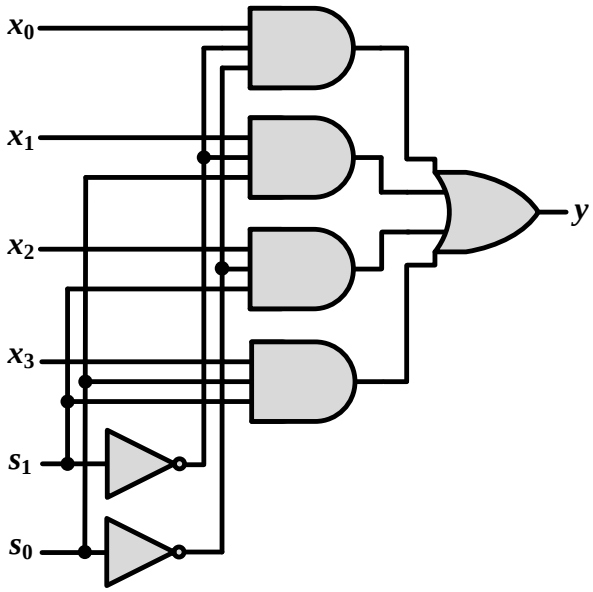
Αντίστοιχα ο πολυπλέκτης (multiplexer – MUX) είναι το συνδυαστικό κύκλωμα, που επιλέγει μία από πολλές εισόδους και κατευθύνει την πληροφορία της σε μία έξοδο. Η επιλογή της εισόδου ελέγχεται από ειδικές εισόδους (σήματα) επιλογής. Σε ένα πολυπλέκτη συμμετέχουν 2^n γραμμές εισόδου και n γραμμές επιλογής και ο συνδυασμός των ψηφίων των γραμμών επιλογής καθορίζουν ποια είσοδος επιλέγεται.

s_1	s_0	y
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

Πίνακας 1.5: Πίνακας αλήθειας πολυπλέκτη 4-σε-1

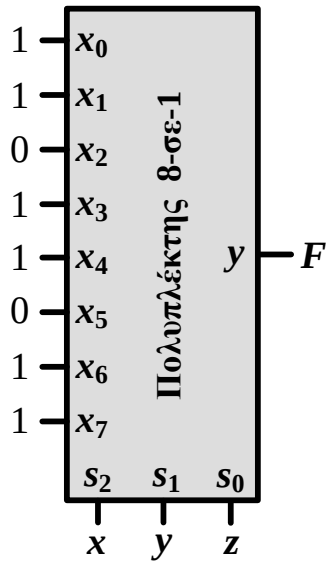


Εικόνα 1.11: συνοπτικό σύμβολο πολυπλέκτη 4-σε-1



Εικόνα1.12: Συνδυαστικό κύκλωμα πολυπλέκτη 4-σε-1

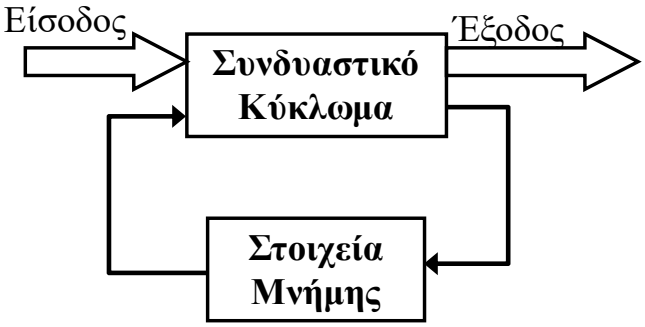
Όπως ένας αποκωδικοποιητής, έτσι και ένας πολυπλέκτης μπορεί να υλοποιήσει οποιαδήποτε συνάρτηση εκφρασμένη σε άθροισμα ελαχιστόρων, αφού διαθέτει τη δομή ενός αποκωδικοποιητή και περιλαμβάνει και μια επιπλέον πύλη Η. Για λογική συνάρτηση n μεταβλητών απαιτείται πολυπλέκτης με (n-1) εισόδους επιλογής και 2^{n-1} εισόδους δεδομένων. Οι περισσότερο σημαντικές (n-1) μεταβλητές συνδέονται στις εισόδους επιλογής και για κάθε συνδυασμό των μεταβλητών επιλογής υπολογίζουμε την έξοδο ως συνάρτηση της τελευταίας μεταβλητής. Η συνάρτηση αυτή μπορεί να ισούται με 0, 1, με την τιμή της τελευταίας μεταβλητής ή με το συμπλήρωμα αυτής.
Π.χ. η υλοποίηση της συνάρτησης $F(x,y,z) = x'y' + xz' + yz$ με έναν πολυπλέκτη 2^n-σε-1, δηλαδή, έναν πολυπλέκτη 8-σε-1 μπορεί να επιτευχθεί όπως φαίνεται παρακάτω.



Εικόνα1.13: Υλοποίηση συνάρτησης με χρήση πολυπλέκτη 8-σε-1

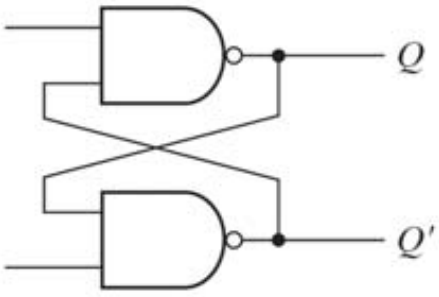
Στα ακολουθιακά κυκλώματα η έξοδος εξαρτάται από την τιμή των εισόδων όχι μόνο εκείνη τη χρονική στιγμή, αλλά και από την τιμή που είχαν σε προηγούμενες χρονικές στιγμές (από μια ακολουθία τιμών εισόδου). Τα ακολουθιακά κυκλώματα μπορεί να συμπεριλαμβάνουν ένα συνδυαστικό τμήμα, αλλά επιπλέον περιλαμβάνουν στοιχεία (ή κυκλώματα) μνήμης (με δυνατότητα να αποθηκεύει πληροφορία ενός δυαδικού ψηφίου), που αναφέρονται ως μανταλωτές (latches) και φλιπ-φλοπ (flip-flops). Τα στοιχεία μνήμης υλοποιούνται με τέτοιο τρόπο ώστε να δημιουργείται ανατροφοδότηση, δηλαδή, η κατάσταση όπου η έξοδος

μίας πύλης τους οδηγείται στην είσοδο μιας δεύτερης πύλης, η έξοδος της οποίας τροφοδοτεί, απευθείας ή μέσω άλλων πυλών, μία από τις εισόδους της πρώτης πύλης. Η πληροφορία που είναι αποθηκευμένη στα στοιχεία μνήμης ενός ακολουθιακού κυκλώματος, αποτελεί την κατάσταση (state) του κυκλώματος. Η παρούσα κατάσταση του κυκλώματος ανατροφοδοτείται στην είσοδο του συνδυαστικού τμήματός του, όπως παρουσιάζεται στην Εικόνα 1.14. Έτσι, οι τιμές των εισόδων και η παρούσα κατάσταση του κυκλώματος, καθορίζουν τις τιμές των εξόδων του και την επόμενη κατάστασή του.

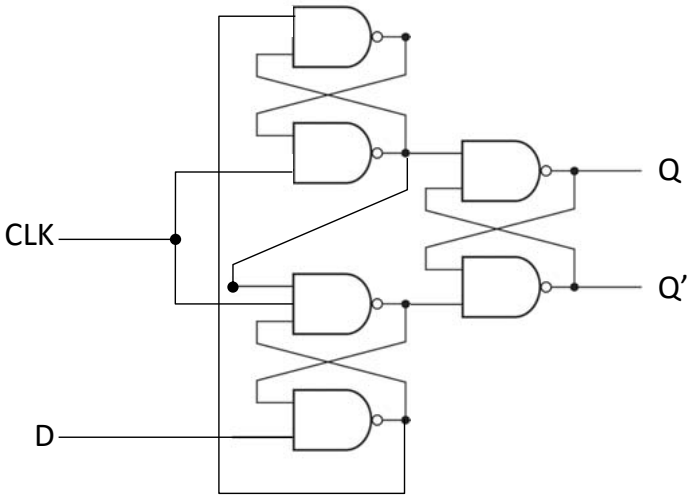


Εικόνα 1.14: Ακολουθιακό κύκλωμα

Στην Εικόνα 1.15 φαίνεται η σύνθεση του μανδαλωτή S-R βασισμένη σε πύλες NAND. Η λειτουργία του βασίζεται στην υπόθεση ότι σε κανονική λειτουργία κρατάμε τις εισόδους στο 1, εκτός κι αν θέλουμε να αλλάξουμε κατάσταση. Με την εφαρμογή ενός στιγμιαίου 0 στην είσοδο θέσης (S) οι έξοδοι μεταβαίνουν στις τιμές Q=1 και Q′=0 (μανδαλωτής σε κατάσταση θέσης). Με την εφαρμογή ενός στιγμιαίου 0 στην είσοδο επαναφοράς (R) οι έξοδοι μεταβαίνουν αντίστοιχα στις τιμές Q=0, Q′= 1 (μανδαλωτής σε κατάσταση επαναφοράς). Η κατάσταση ενός μανδαλωτή μπορεί να αλλάξει μόνο κατά τη διάρκεια του θετικού επιπέδου (λογικό 1) του σήματος ελέγχου (παλμός ρολογιού). Σε έναν μανδαλωτή τύπου D, επιτρέπεται η αλλαγή της εισόδου D όσο ο παλμός ρολογιού παραμένει στο επίπεδο του λογικού 1 και ο μανδαλωτής μπορεί να αποκριθεί στη νέα τιμή εισόδου με αποτέλεσμα πιθανή νέα κατάσταση εξόδου. Αυτό οδηγεί σε απρόβλεπτη λειτουργία, που δημιουργεί προβλήματα στο σχεδιασμό ακολουθιακών κυκλωμάτων με κοινή πηγή ρολογιού για όλα τα στοιχεία μνήμης. Έτσι, τα flip-flop σχεδιάζονται ώστε να πυροδοτούνται μόνο κατά τη διάρκεια της μετάβασης (αλλαγής επιπέδου) ενός σήματος χρονισμού του αποκαλούμενου ρολογιού, όπως φαίνεται στην Εικόνα 1.16. Ένα ακολουθιακό κύκλωμα του οποίου η κατάσταση μπορεί να αλλάξει μόνο σε διακριτές χρονικές στιγμές, αναφέρεται ως σύγχρονο ακολουθιακό κύκλωμα (synchronous sequential circuit). Οι διακριτές χρονικές στιγμές καθορίζονται από μια περιοδική σειρά παλμών, η οποία συνιστά ένα σήμα που αναφέρεται ως ρολόι (clock) και παράγεται από κατάλληλη διάταξη χρονισμού (γεννήτρια ρολογιού, clock generator). Η αλλαγή κατάστασης ενός σύγχρονου ακολουθιακού κυκλώματος είναι δυνατή μόνο κατά την αλλαγή της λογικής τιμής του σήματος ρολογιού. Επειδή, τα σύγχρονα ακολουθιακά κυκλώματα διατρέχουν μια ακολουθία καταστάσεων, αναφέρονται και ως μηχανές πεπερασμένων καταστάσεων (finite-state machines).



Εικόνα 1.15: Μανδαλωτής S-R

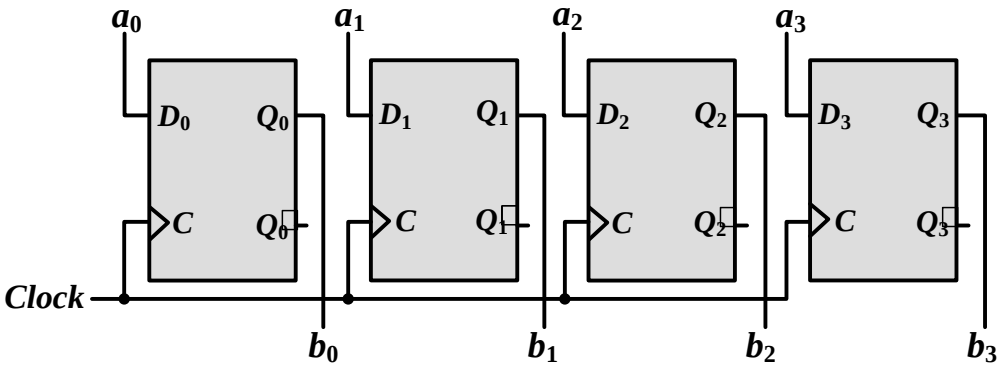


Εικόνα 1.16: DFlip-flop

Όπως και με τη σχεδίαση συνθετότερων ψηφιακών κυκλωμάτων με ιεραρχική διασύνδεση βασικών συνδυαστικών κυκλωμάτων, έτσι και με την χρήση των διαφόρων τύπων flip-flop ως βασική μονάδα μπορούν να σχεδιαστούν συνθετότερα βασικά ακολουθιακά κυκλώματα. Συνήθη τέτοια ακολουθιακά κυκλώματα (καταχωρητές και μετρητές) που αποτελούν χρήσιμα δομικά στοιχεία για την υλοποίηση ψηφιακών συστημάτων μπορούμε να καταγράψουμε τα ακόλουθα:

- Παράλληλοι καταχωρητές
- Καταχωρητές ολίσθησης
- Σύγχρονοι δυαδικοί μετρητές
- Σύγχρονοι μετρητές επαναλαμβανόμενης ακολουθίας αριθμών
- Σύγχρονοι μετρητές με δυνατότητα παράλληλης φόρτωσης
- Ασύγχρονοι μετρητές

Για παράδειγμα στην Εικόνα 1.17 παρουσιάζεται η σύνθεση ενός καταχωρητή τεσσάρων δυαδικών ψηφίων με δυνατότητα παράλληλης φόρτωσης βασισμένου στο D flip-flop που παρουσιάστηκε παραπάνω.



Εικόνα 1.17: Σύνθετο ακολουθιακό κύκλωμα με χρήση D Flip-flop: Καταχωρητής με παράλληλη φόρτωση

1.3 Βασικές τεχνολογίες σχεδίασης και κατασκευής ψηφιακών συστημάτων

Σχεδίαση και κατασκευή ψηφιακών κυκλωμάτων

Η υλοποίηση των ψηφιακών κυκλωμάτων όπως αυτά που παρουσιάσαμε παραπάνω γίνεται με τη βοήθεια ολοκληρωμένων κυκλωμάτων (Integrated Circuit – IC), τα οποία τυποποιούνται σε συσκευασίες γνωστές και ως *chip* και μπορούν να γίνουν εμπορικά διαθέσιμα. Έτσι τα κυκλώματα λογικών πυλών καθώς και σύνθετων συνδυαστικών και ακολουθιακών κυκλωμάτων ή πιο πολύπλοκων επεξεργαστικών μονάδων γίνονται διαθέσιμα προς ολοκλήρωση σε συστήματα εφαρμογών μέσω μεταλλικών διεπαφών που επιτρέπουν την ηλεκτρική τους διασύνδεση με άλλα ηλεκτρονικά υποσυστήματα. Στη συνέχεια κάνουμε μια σύντομη παρουσίαση των διαθέσιμων τεχνολογιών για την κατασκευή ολοκληρωμένων κυκλωμάτων καθώς και των μεθοδολογιών και εργαλείων σχεδίασης και ανάπτυξης εφαρμογών ηλεκτρονικών συστημάτων. Αντίστοιχα ολοκληρωμένα θα χρησιμοποιηθούν στις ασκήσεις των κεφαλαίων 2 και 3 επάνω σε συνδυαστικά και ακολουθιακά κυκλώματα, ενώ προχωρημένα εργαλεία σχεδίασης και επαναδιατάξιμα ηλεκτρονικά θα χρησιμοποιηθούν στις ασκήσεις που παρουσιάζονται στα κεφάλαια 4 και 5.

Διατάξεις απλών λογικών πυλών και ψηφιακών κυκλωμάτων (small/medium-scaleintegration, SSI/MSI)

Η τεχνολογία μικρής κλίμακας ολοκλήρωσης (SSI) παρέχει την δυνατότητα υλοποίησης δομών της τάξης των λίγων δεκάδων τρανζίστορ σε μία μονή ημιαγώγιμη διάταξη (συνήθως πυριτίου) ενώ η μεσαίας κλίμακας ολοκλήρωση (MSI) παρέχει την δυνατότητα υλοποίησης δομές της τάξης των 100 τρανζίστορ περίπου. Οι παραπάνω διατάξεις οδηγούν σε ολοκληρωμένα κυκλώματα που περιέχουν έτοιμες μερικές λογικές πύλες ή στοιχειώδη ψηφιακά κυκλώματα. Τα ολοκληρωμένα κυκλώματα βρίσκονται σε κατάλληλες θήκες με περίβλημα από κεραμικό υλικό, από το οποίο βγαίνουν κατάλληλοι *ακροδέκτες* (pins). Μέσω αυτών των ακροδεκτών το ολοκληρωμένο κύκλωμα μπορεί να συνδεθεί με άλλα ολοκληρωμένα κυκλώματα ώστε να σχηματιστεί το επιθυμητό κύκλωμα Ο αριθμός των ακροδεκτών διαφέρει και μπορεί να είναι από δεκατέσσερις έως και εκατοντάδες, ανάλογα με το ολοκληρωμένο.

Τα ολοκληρωμένα κυκλώματα ταξινομούνται σε διάφορες οικογένειες ανάλογα με την τεχνολογία με την οποία είναι κατασκευασμένα. Οι πιο διαδεδομένες οικογένειες είναι:

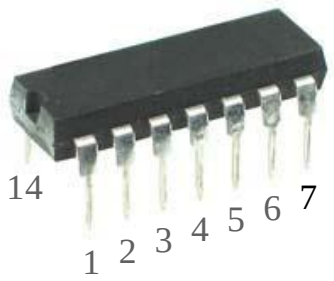
- TTL Transistor – Transistor Logic
- MOS Metal – Oxide – Semiconductor
- CMOS Complementary MOS

Η οικογένεια TTL αποτελείται από αρκετές υποοικογένειες ή σειρές. Το όνομα της κάθε σειράς και το χαρακτηριστικό της πρόθημα φαίνονται στον ακόλουθο πίνακα.

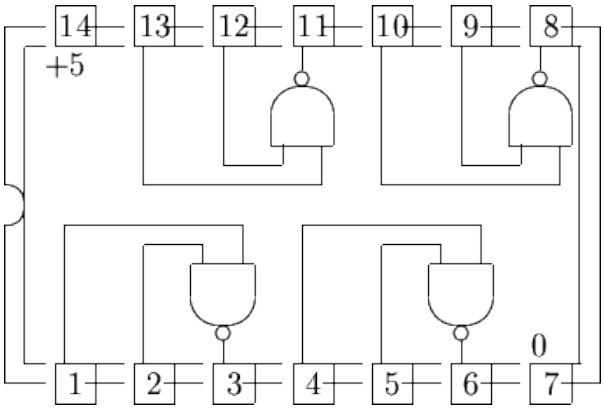
Σειρές TTL	**Πρόθημα**	**Παράδειγμα**
Βασική (Standard) TTL	74	7486
Υψηλής ταχύτητας TTL	74H	74H86
Χαμηλής ισχύος TTL	74L	74L86
Schottky TTL	74S	74S86
Χαμηλής ισχύος Schottky TTL	74LS	74LS86
Προηγμένη Schottky TTL	74AS	74AS86
Προηγμένη χαμηλής ισχύος Schottky TTL	74ALS	74ALS86

Πίνακας1.6: Σειρές TTL

Παράδειγμα ολοκληρωμένου κυκλώματος λογικών πυλών, αποτελεί αυτό που παρουσιάζεται στην Εικόνα σε φυσική μορφή και ως διάγραμμα στην Εικόνα 1.18 και περιλαμβάνει τέσσερις πύλες NAND δύο εισόδων. Τα κυκλώματα μικρής κλίμακας ολοκλήρωσης παρουσιάζουν λειτουργικότητά περιορισμένη σε σχέση με την επιφάνεια που καταλαμβάνουν.



Εικόνα 1.18: Ολοκληρωμένο κύκλωμα 7400



Εικόνα 1.19: Λογικό διάγραμμα 7400

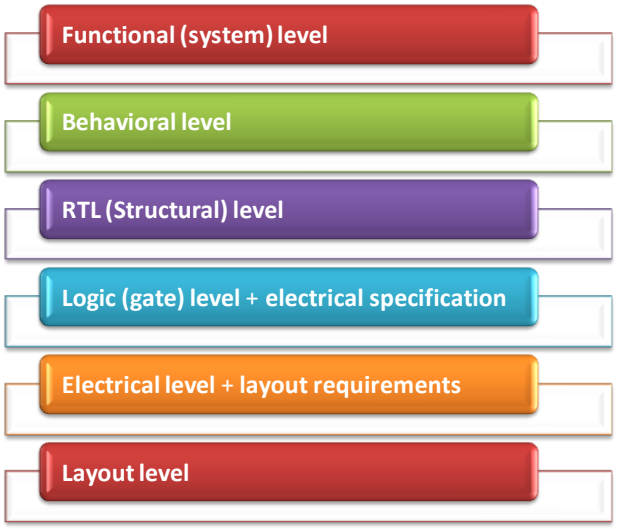
Σχεδίαση και κατασκευή σύνθετων ψηφιακών κυκλωμάτων (verylarge-scaleintegration, VLSI)

Τα σύγχρονα ολοκληρωμένα κυκλώματα τεχνολογίας CMOS πολύ μεγάλης κλίμακας ολοκλήρωσης (very large scale integration, VLSI), διαθέτουν το πλεονέκτημα της ολοκλήρωσης πολύ μεγάλου πλήθους από τρανζίστορ MOSFET, τα οποία κατασκευάζονται σε πολύ μικρά μεγέθη. Η ταχύτητα εξέλιξης της τεχνολογίας κατασκευής ολοκληρωμένων κυκλωμάτων (βασισμένη στην τεχνική της φωτολιθογραφίας) έχει οδηγήσει σε τεράστια πυκνότητα ολοκλήρωσης σε κάθε μονάδα πυριτίου. Χαρακτηριστικά μπορούμε να αναφέρουμε την εξέλιξη των μικροεπεξεργαστών της εταιρείας Intel που παρουσιάζουν τις παρακάτω επιδόσεις:

- 2003:
- IntelPentium 4 mprocessor (55 εκατομμύρια τρανζίστορ)
- 512 MbitDRAM (> 0.5 δις τρανζίστορ)
- 2010:
- Six-Core Core i7 (Gulftown) (1,17 δισ τρανζίστορ)

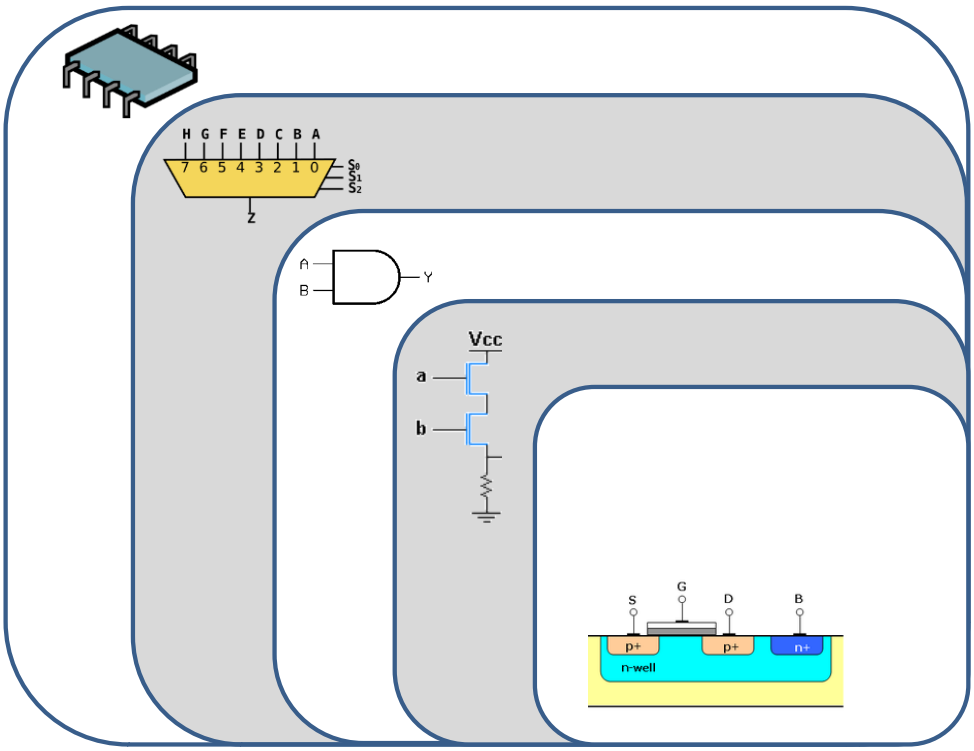
Προφανώς οι δυνατότητες αυτές που προσφέρουν οι νέες τεχνολογίες ολοκλήρωσης απαιτούν και νέες σχεδιαστικές μεθόδους, οι οποίες θα μπορούν να δώσουν με αυτοματοποιημένο τρόπο λύσεις για την περιγραφή, σχεδίαση, βελτιστοποίηση και κατασκευή πολύπλοκων ψηφιακών ηλεκτρονικών καθώς καθίσταται δύσκολη, αν όχι ανέφικτη η επέμβαση του σχεδιαστή σε χαμηλό επίπεδο για ολόκληρη την έκταση του υπό ανάπτυξη κυκλώματος (που μπορεί να είναι της κλίμακας των εκατομμυρίων πυλών όπως είδαμε παραπάνω). Για το λόγο αυτό έχουν αναπτυχθεί σύγχρονες ιεραρχικές μεθοδολογίες βασισμένες σε υπολογιστικές εφαρμογές, οι οποίες επιτρέπουν την σχεδίαση και ανάλυση με βάση τη μοντελοποίηση υποσυστημάτων, τα οποία συντίθενται σε υψηλότερου επιπέδου ιεραρχίες. Με επανάληψη της διαδικασίας μπορεί να γίνει απλοποίηση των επιμέρους βημάτων, επικεντρώνοντας στη σχεδίαση και επαλήθευση επιμέρους λογικών μονάδων, οι οποίες να διασυνδεθούν για την υλοποίηση πιο σύνθετων συστημάτων.
Οι επιμέρους μεθοδολογίες σχεδίασης καταρχήν εστιάζουν στα διαφορετικά επίπεδα σχεδίασης όπως φαίνεται στην Εικόνα . Έτσι οι απαιτήσεις σχεδίασης ξεκινούν σε υψηλό επίπεδο από την περιγραφή της αρχιτεκτονικής και των προδιαγραφών λειτουργίας (Functional/system level), την λειτουργική περιγραφή/μοντελοποίηση με ανάθεση δομών και πόρων επί του ολοκληρωμένου (Behavioral level), την περιγραφή της λογικής σχεδίασης

σε επίπεδο λειτουργικών μονάδων (RTL/Structural level), την σχεδίαση του λογικού κυκλώματος με επιλογή συγκεκριμένης τεχνολογίας και ηλεκτρικών προδιαγραφών (Logic/gatelevel + electrica lspecification), την σχεδίαση στο ηλεκτρικό επίπεδο με την συνδεσμολογία των επιμέρους μονάδων, (Electrical level + layout requirements) και τέλος την σχεδίαση και κατασκευή των δομών επί των ημιαγώγιμων διατάξεων (Layoutlevel) που θα χρησιμοποιηθούν για την κατασκευή του τελικού ολοκληρωμένου (chip).



Εικόνα 1.20: Ιεραρχίες σχεδίασης

Όπως ήδη αναφέρθηκε οι σύγχρονες ιεραρχικές μεθοδολογίες βασίζονται σε σχεδιαστικές εφαρμογές, οι οποίες επιτρέπουν την σχεδίαση και ανάλυση με βάση τη μοντελοποίηση υποσυστημάτων, τα οποία συντίθενται σε υψηλότερου επιπέδου ιεραρχίες. Η σχεδίαση με βάση τη μοντελοποίηση και σύνθεση απλούστερων μονάδων επίσης μπορεί να εφαρμοστεί αντίστοιχα σε κάθε επίπεδο της παραπάνω ιεραρχίας, όπως απεικονίζεται στην Εικόνα 1.21.
Έτσι, για κάθε επιμέρους επίπεδο και στάδιο της κατασκευής του ολοκληρωμένου διατίθενται εξειδικευμένα σχεδιαστικά εργαλεία και μεθοδολογίες, οι οποίες όμως μοιράζονται τα κοινά στοιχεία της μοντελοποίησης, ανάπτυξης βιβλιοθηκών για επαναχρησιμοποίηση έτοιμων δομικών μονάδων και της ιεραρχικής σχεδίασης.



Εικόνα 1.21: Ιεραρχίες σχεδίασης με βάση τη μοντελοποίηση υποσυστημάτων

Σε άμεση συνάφεια με τα επίπεδα αφαίρεσης και σχεδίασης μπορούμε γενικά να διακρίνουμε τα ακόλουθα επίπεδα περιγραφής και τεχνικών περιγραφής και επαλήθευσης λειτουργίας σύνθετων ολοκληρωμένων όπως απεικονίζεται στην Εικόνα 1.22

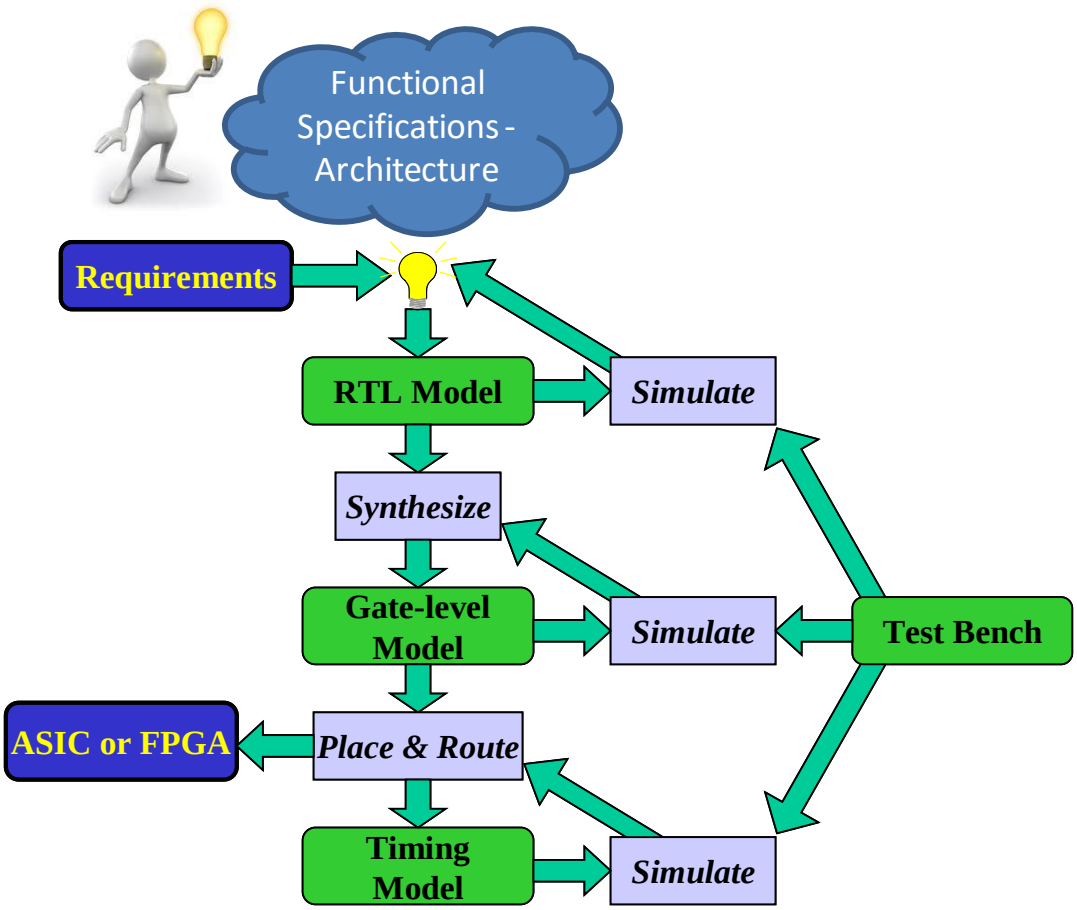
- Υψηλού επιπέδου περιγραφές (με χρήση γραφικών σχεδιαστικών εργαλείων ή/και γλωσσών HDL)
  - Λειτουργικές περιγραφές (Behavioral)
  - Περιγραφές λειτουργικότητας σε επίπεδο μονάδων (RTL)
  - Περιγραφές λειτουργικότητας σε επίπεδο πυλών (Gate-Level ή structural περιγραφές)

- Χαμηλού επίπεδου περιγραφές
  - Circuit-Level περιγραφές(π.χ. SPICE)
  - Layout

Επιπλέον στην Εικόνα παρουσιάζεται και η μεθοδολογία σχεδίασης και επαλήθευσης ορθής λειτουργίας του κυκλώματος που σχεδιάζεται, η οποία και αυτή γίνεται με ιεραρχικό και επαναληπτικό τρόπο. Έτσι σε κάθε επίπεδο πρέπει να εφαρμόζεται και η κατάλληλη τεχνική επαλήθευσης πριν την εκτέλεση του επόμενου βήματος. Για το σκοπό αυτό τα σύγχρονα σχεδιαστικά εργαλεία παρέχουν τη δυνατότητα εκτέλεσης σεναρίων προσομοίωσης είτε μέσω γραφικού τρόπου, είτε μέσω των αποκαλούμενων γλωσσών περιγραφής υλικού υψηλού επιπέδου για την μοντελοποίηση των υποσυστημάτων του κυκλώματος καθώς και των εξωτερικών διεγέρσεων αυτού.
Οι γλώσσες περιγραφής υλικού υψηλού επιπέδου (hardware description languages, HDLs) κερδίζουν έδαφος, ως μεθοδολογία σχεδίασης και ανάλυσης ψηφιακών συστημάτων καθώς παρουσιάζουν τα παρακάτω προτερήματα:

- Εύκολη περιγραφή υλικού και εξίσου εύκολες διορθώσεις
- Συμπαγή περιγραφή
- Δεν χρειάζεται να προσπαθεί κανείς να ακολουθήσει πολύπλοκες γραμμές διασυνδέσεων
- Εύκολη ανάλυση (πληθώρα εργαλείων)

Από την άλλη δεν ενδείκνυται κάποιος να χρησιμοποιεί HDLs, όταν το κύκλωμα που θέλει να σχεδιάσει έχει κάποιο πολύ κρίσιμο χαρακτηριστικό (ταχύτητα, μέγεθος, κατανάλωση ισχύος) Π.χ. data paths σε επεξεργαστές, καθώς η περιγραφή με HDL απαιτεί την αυτόματη μετατροπή σε τελικό μοντέλο κυκλώματος, που δεν είναι εύκολο για πολύπλοκα σχέδια να επιτύχει βέλτιστα αποτελέσματα.



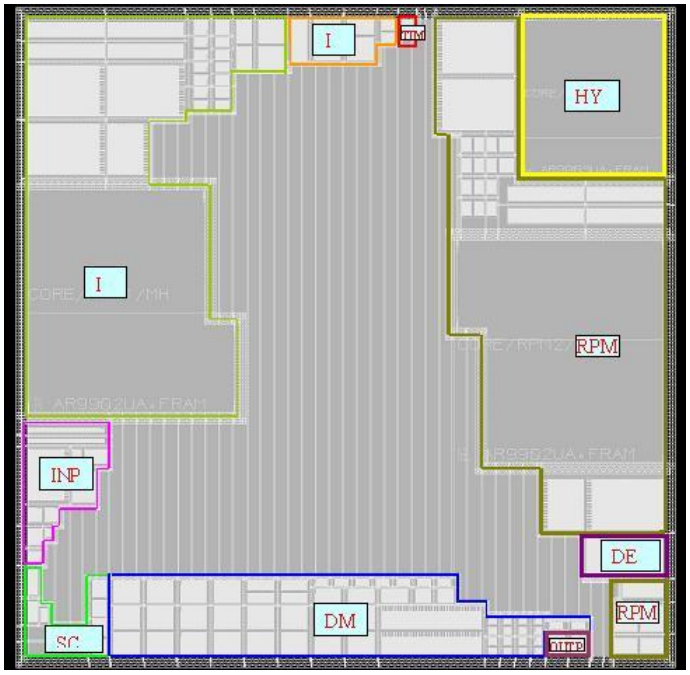
Εικόνα 1.22: Μεθοδολογία και ροή ιεραρχικής σχεδίασης

Για παράδειγμα τα στάδια σχεδίασης που περιγράφονται στην Εικόνα μπορούν να εφαρμοστούν με χρήση γλωσσών HDL, ως ακολούθως: Οι απαιτήσεις προδιαγράφουν την επιθυμητή λειτουργία του κυκλώματος. Στη συνέχεια αναπτύσσεται το RTL(register-transfer level) μοντέλο χρησιμοποιώντας γλώσσα HDL. Η λειτουργία του μοντέλου επαληθεύεται με χρήση προσομοιωτών (προγράμματα λογισμικού). Στη συνέχεια το μοντέλο μετατρέπεται σε κύκλωμα με πύλες και flip-flop (είναι στη διακριτική ευχέρεια του σχεδιαστή αν θέλει να επαληθεύσει τη λειτουργία σε αυτό το επίπεδο). Στη συνέχεια σχεδιάζεται το κύκλωμα με χρονική ακρίβεια (υπολογίζονται οι διαδρομές και οι καθυστερήσεις των σημάτων) και παράγονται τόσο ένα αρχείο προγραμματισμού του chip όσο και ένα μοντέλο για προσομοίωση. Η χρήση της γλώσσας ενδείκνυται για την ανάπτυξη πολύπλοκων κυκλωμάτων που αποτελούνται από πολλές μικρές οντότητες.

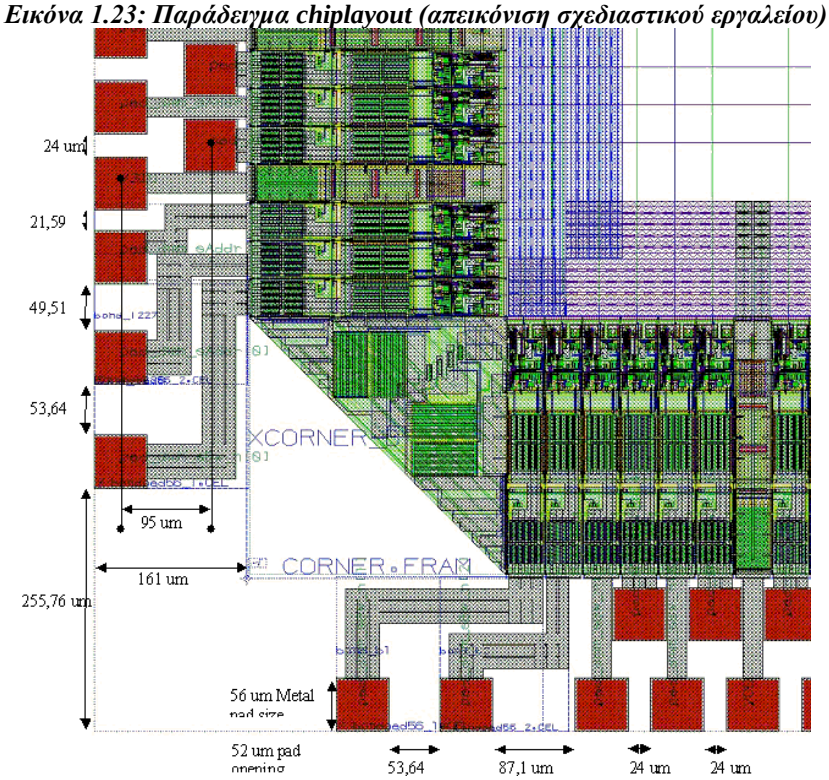
Στο παρόν βιβλίο και συγκεκριμένα στο κεφάλαιο 4 θα περιγραφεί η αντίστοιχη διαδικασία με χρήση της γλώσσας VHDL. Η VHDL (V_{HSIC} -Very High Speed Integrated Circuit – Hardware Description Language) είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή και μοντελοποίηση ψηφιακών κυκλωμάτων. Αρχικοποιήθηκε από το Υπουργείο Αμύνης των ΗΠΑ (Department of Defense-USA) στις αρχές του 1980. Η VHDL διαφέρει από τις συμβατικές γλώσσες κατά το ότι δεν προορίζεται να περιγράψει λειτουργίες που εκτελούνται σειριακά, η μία μετά την άλλη. Κάθε πρόταση ή τμήμα κώδικα περιγράφει λειτουργίες, οι οποίες παράγουν αποτελέσματα χρονική εξαρτημένες με τις υπόλοιπες λειτουργίες, που περιγράφονται στις υπόλοιπα τμήματα του κώδικα. Η χρονική εξάρτηση μπορεί να είναι είτε ταυτόχρονα, είτε κατόπιν μεταβολής συγκεκριμένων σημάτων που η σύνταξη της γλώσσας προβλέπει να δηλώνονται με συγκεκριμένο και μοναδικό τρόπο. Η VHDL μπορεί να περιγράψει τόσο σύγχρονες όσο και ακολουθιακές λογικές λειτουργίες. Με τη χρήση λογισμικού (compiler) ο υπολογιστής μπορεί να συνθέσει τα απαιτούμενα ηλεκτρονικά κυκλώματα (βασισμένος σε συγκεκριμένες βιβλιοθήκες και κανόνες υλοποίησης) που θα υλοποιούν στην πράξη τις σύγχρονες λειτουργίες που περιγράφει ο κώδικας.

Στο τελικά στάδια της ιεραρχικής σχεδίασης, όπως παρουσιάζονται στην Εικόνα 1.20 και
Εικόνα 1.21 πρέπει να επιλεγεί η τεχνολογία κατασκευής του ολοκληρωμένου κυκλώματος, όπως φαίνεται στην Εικόνα 1.22. Στη γενική περίπτωση η σχεδίαση γίνεται με στόχο την υλοποίηση με σύγχρονα

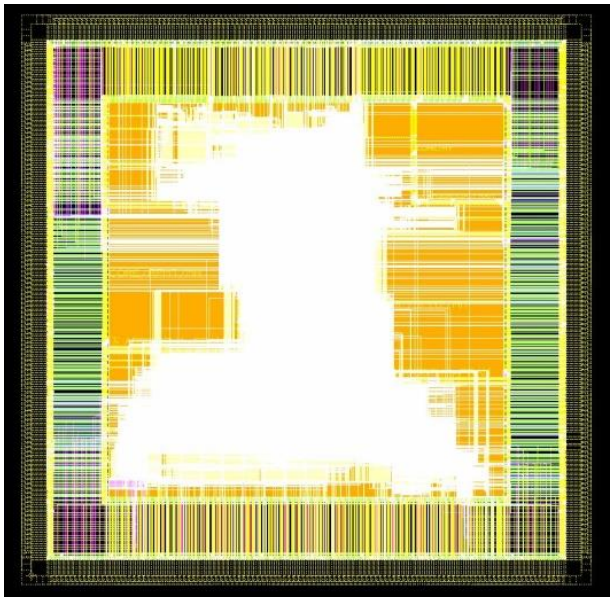
ολοκληρωμένα κυκλώματα ειδικού σκοπού (application specific integrated circuits, ASICs). Στην Εικόνα 1.23 εμφανίζεται ένα παράδειγμα απεικόνισης του τελικού σταδίου κατασκευής ενός σύνθετου επεξεργαστή μετά την περιγραφή όλων των λειτουργικών υπομονάδων του και την υλοποίηση αυτών επί της επιφάνειας της ημιαγώγιμης διάταξης (chiplayout). Στην Εικόνα 1.24 αντίστοιχα απεικονίζεται μια σχεδιαστική λεπτομέρεια που εμφανίζει της λεπτομέρειες τοποθέτησης των αγωγών διασύνδεσης με τις εξωτερικές διεπαφές και τον προσδιορισμό της γεωμετρίας των διατάξεων. Στην Εικόνα 1.25 απεικονίζεται η τελική εικόνα του ολοκληρωμένου μετά την αποτύπωση επί της διάταξης και τέλος στην Εικόνα 1.26 απεικονίζεται ένα λειτουργικό πρωτότυπο μετά την κατασκευή στο εργοστάσιο (fabrication, packaging).



Εικόνα 1.23: Παράδειγμα chiplayout (απεικόνιση σχεδιαστικού εργαλείου)



Εικόνα 1.24: Λεπτομέρεια chiplayout σε μεγέθυνση (απεικόνιση σχεδιαστικού εργαλείου)



Εικόνα 1.25: Παράδειγμα chiplayout (απεικόνιση ενός επιπέδου-layer)

Εικόνα 1.26: Παράδειγμα chip μετά την κατασκευή στο εργοστάσιο (fabrication, packaging)

Εναλλακτική υλοποίηση αυτής των ASIC είναι τα FPGA (Field Programmable Gate Array) ή συστοιχίες επιτόπια προγραμματιζόμενων πυλών. Τα FPGA είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL κα. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία. Η χρήση των FPGA έχει κυρίως το πλεονέκτημα της ευελιξίας και τη δυνατότητα εύκολης τροποποίησης, διόρθωσης και επαναπρογραμματισμού του σχεδίου και περιγράφεται με μεγαλύτερη λεπτομέρεια στη συνέχεια. Στο τελευταίο κεφάλαιο παρουσιάζονται ασκήσεις βασισμένες στη χρήση αναπτυξιακών καρτών επανδρωμένων με FPGA και αντίστοιχων σχεδιαστικών εργαλείων.

Σχεδίαση σύνθετων ψηφιακών κυκλωμάτων με την βοήθεια προγραμματιζόμενων ολοκληρωμένων κυκλωμάτων

- PLD – Programmable Logic Devices

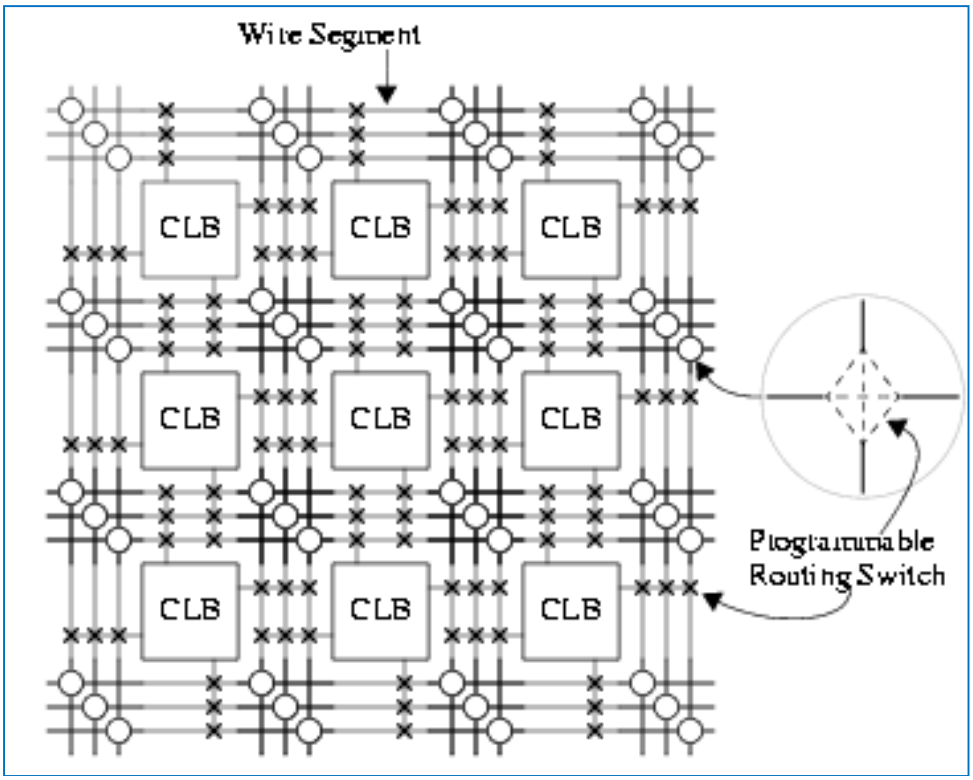
Τα πρώτα προγραμματιζόμενα ολοκληρωμένα κυκλώματα ήταν τα PLD, τα οποία διαδόθηκαν ιδιαίτερα τη δεκαετία του 1980, αν και πρωτοεμφανίστηκαν το 1969. Παράλληλα με την εξέλιξη των ηλεκτρονικών υπολογιστών, δημιουργήθηκαν και πολύπλοκα εργαλεία software για τον προγραμματισμό τους. Τα PLD είναι ουσιαστικά ένα πλέγμα από πύλες AND και OR που μπορούν να προσομοιώσουν οποιοδήποτε ψηφιακό κύκλωμα, αρκεί να προγραμματιστούν οι κατάλληλες συνδέσεις. Αυτό μπορούσε να γίνει χειροκίνητα ή αυτοματοποιημένα με τη χρήση των εργαλείων τύπου EDA (ElectronicDesignAutomation), μέσω μιας γλώσσας προγραμματισμού HDL.

- CPLD – ComplexProgrammable Logic Devices

Τα CPLD έγιναν δημοφιλή τη δεκαετία του 1990 και αποτελούν την εξέλιξη των PLD. Αντί για πύλες περιέχουν Λογικά Μπλοκ συνδεδεμένα μεταξύ τους σε διάταξη πλέγματος. Τα Λογικά Μπλοκ είναι μικρού μεγέθους PLD που μπορούν να προσομοιώσουν οποιαδήποτε πύλη. Τα CPLD επίσης περιέχουν και πολλά επιμέρους τυποποιημένα ψηφιακά κυκλώματα όπως flip-flops, registers, RAM κλπ. Ο προγραμματισμός τους γίνεται επίσης μέσω προγραμματισμού συνδέσεων. Βασικά τους μειονεκτήματα είναι το ότι προγραμματίζονται μόνο μια φορά, αλλά και η περιορισμένη χωρητικότητα τους.
Τα ASIC (ApplicationSpecificIntegratedCircuit) που χρησιμοποιούνται ευρέως σήμερα αποτελούν εξέλιξη των CPLD. Πρόκειται για chip στα οποία ενσωματώνουμε μικρότερα έτοιμα κυκλώματα χρησιμοποιώντας μόνο μια ή δύο λιθογραφικές μάσκες, μειώνοντας έτσι σημαντικά το κόστος παραγωγής.

- FPGA – Field Programmable Gate Array

Τα πρώτα FPGA παρουσιάστηκαν το 1985 από την Αμερικανική εταιρεία Xilinx. Η λειτουργία τους βασίζεται επίσης σε Λογικά Μπλοκ, που εδώ ονομάζονται CLB (Combinational Logic Block). Τα CLB ενώνονται μέσω ενός μεγάλου μεγέθους δικτυώματος (matrix), όπου οι μεταξύ τους συνδέσεις καθορίζονται από προγραμματιζόμενες διασυνδετικές διατάξεις (PSM – ProgrammableSwitching MATRIX). Θέσεις RAM αντικαθιστούν τις μόνιμα αποκαθιστάμενες συνδέσεις, κάνοντας τα FPGA αναπρογραμματιζόμενα.

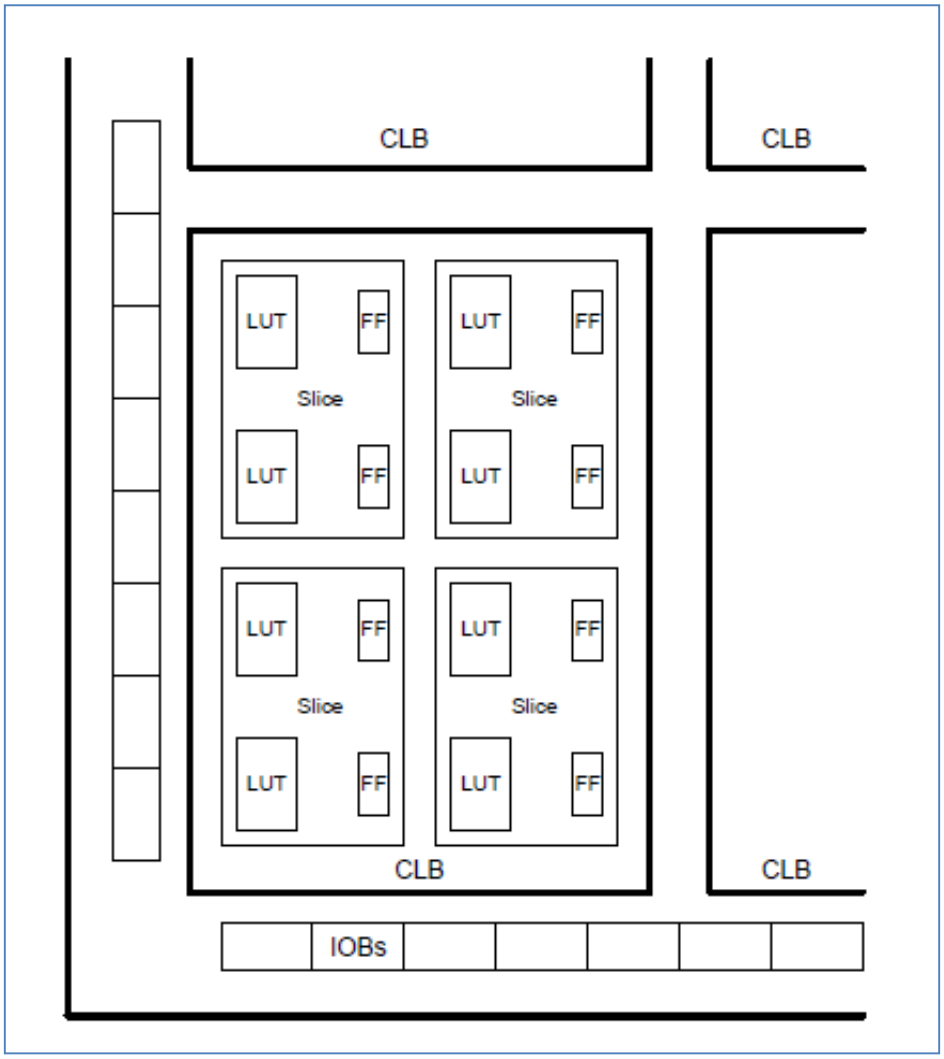


Εικόνα 1.27: Η βασική αρχιτεκτονική ενός FPGA.

Τα σύγχρονα FPGA chip μπορεί να περιέχουν μέχρι και ενσωματωμένους μικροεπεξεργαστές εκτός από flip-flop, μνήμη, κλπ, αποτελώντας έτσι ένα ολοκληρωμένο προγραμματιζόμενο σύστημα. Κάποια περιέχουν ακόμη και αναλογικές διατάξεις όπως ADC, DAC, κλπ, επιτρέποντας μας να δημιουργήσουμε μέχρι και ένα ολοκληρωμένο ψηφιακό σύστημα επεξεργασίας σήματος (DSP). Η χωρητικότητα τους μπορεί να φτάνει μερικά εκατομμύρια πύλες ενώ ο προγραμματισμός τους μπορεί να γίνει και καθώς το κύκλωμα βρίσκεται σε λειτουργία. Εν ολίγοις, τα FPGA είναι πολύ ισχυρά και ευέλικτα. Το κόστος τους συνεχώς μειώνεται ενώ οι δυνατότητες αυξάνονται. Μοναδικό τους μειονέκτημα είναι ότι λόγω της πολύπλοκης αρχιτεκτονικής τους λειτουργούν σε χαμηλότερες ταχύτητες από τα ASIC ενώ καταναλώνουν περισσότερη ενέργεια.
Ένα FPGA είναι μία ημιαγώγιμη συσκευή η οποία περιλαμβάνει στοιχεία προγραμματιζόμενης λογικής καθώς και προγραμματιζόμενες διασυνδέσεις. Τα στοιχεία προγραμματιζόμενης λογικής μπορούν να προγραμματιστούν, έτσι ώστε να παρουσιάζουν τη λειτουργικότητα απλών λογικών πυλών (“and”, “or”, “xor”, “not”) ή τη λειτουργικότητα περισσότερο σύνθετων συνδυαστικών συναρτήσεων, όπως είναι η αποκωδικοποίηση (αποκωδικοποιητές), η άθροιση (αθροιστές) και άλλες μαθηματικές συναρτήσεις. Επίσης, στα στοιχεία αυτά περιλαμβάνονται πολλές φορές και στοιχεία μνήμης τα οποία είτε είναι απλά “flip-flops”, είτε είναι πλήρη τμήματα (blocks) μνήμης. Όλα αυτά τα στοιχεία μπορούν να διασυνδεθούν σύμφωνα με τις απαιτήσεις της εφαρμογής την οποία καλούνται να υλοποιήσουν με τη βοήθεια της υπάρχουσας ιεραρχίας των προγραμματιζόμενων διασυνδέσεων. Αυτή η λειτουργικότητα θα μπορούσε να παρομοιασθεί με εκείνη ενός προγραμματιζόμενου “breadboard”.
Πιο συγκεκριμένα, η βασική αρχιτεκτονική που απαντάται σε ένα FPGA (όπως βλέπουμε στην Εικόνα) περιλαμβάνει μία διάταξη από λογικές δομικές μονάδες (“CLBs” –Configurable Logic Blocks) καθώς και κανάλια διασύνδεσης. Μία τυπική λογική δομική μονάδα ενός FPGA αποτελείται από ένα στοιχείο αντιστοίχισης “LUT” (Look-UpTable) τεσσάρων εισόδων και από ένα “flip-flop”.
Επίσης, πρέπει να αναφερθεί ότι ένα από τα κύρια πλεονεκτήματα των FPGAs είναι το ότι υποστηρίζουν την πλήρη ή μερική επαναδιάταξη (reconfiguration) τους, παρέχοντας έτσι τη δυνατότητα αλλαγής της σχεδίασης «on-the-fly», κατά τη διάρκεια δηλαδή της εκτέλεσης. Κάτι τέτοιο είναι χρήσιμο, είτε γιατί μία τέτοιου είδους δυναμική επαναδιάταξη των στοιχείων μπορεί να αποτελεί αναπόσπαστο τμήμα της εφαρμογής, είτε γιατί μπορεί να απαιτείται για κάποια αναβάθμιση του συστήματος.

- CLB – Combinational Logic Block

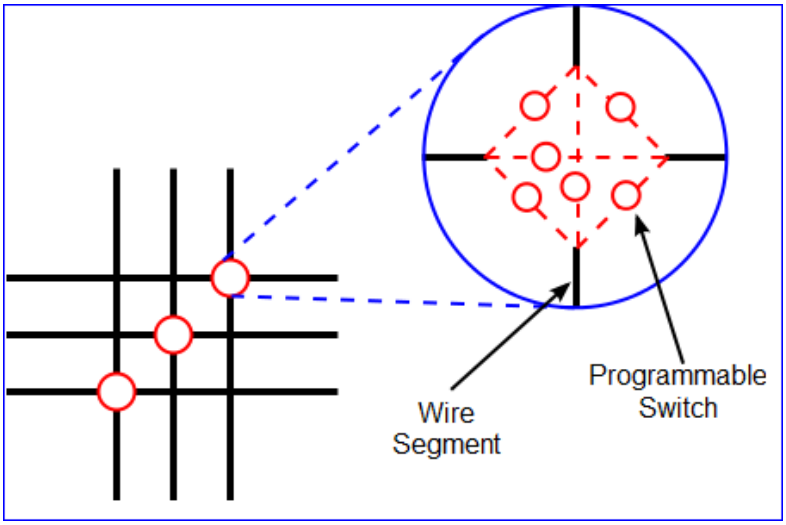
Το CLB είναι το “κύτταρο” ενός FPGA. Αποτελείται από LUTs (Look - UpTables), συνήθως δύο, που εκτελούν τις λογικές πράξεις (Εικόνα 1.28). Τα LUTs είναι στην πραγματικότητα μικρές RAM στις οποίες φορτώνουμε τον πίνακα αληθείας κάποιας πύλης ή κάποιας πιο πολύπλοκης λογικής πράξης. Φορτώνοντας το σωστό περιεχόμενο σε αυτές τις RAM μπορούμε να προσομοιώσουμε οποιαδήποτε λογική πράξη, ενώ αν οδηγήσουμε την έξοδο του LUT σε ένα άλλο, μπορούμε να αυξήσουμε σημαντικά το βαθμό πολυπλοκότητας . Το κάθε LUT είναι 4 – 6 εισόδων και ακολουθείται από flip-flop που συγκρατεί τα αποτελέσματα της επεξεργασίας του. Έτσι, το κάθε LUT μπορεί να προσομοιώσει π.χ. δύο πύλες AND ή μια XOR.



Εικόνα 1.28: Η δομή ενός CLB.

- PSM – Programmable Switching Matrix

Το PSM είναι το δεύτερο δομικό στοιχείο του FPGA. Η λειτουργία του είναι να καθορίζει τις συνδέσεις των οριζόντιων και κάθετων γραμμών στο τεράστιο δικτύωμα που ενώνει μεταξύ τους όλα τα CLB και τις εισόδους/εξόδους του FPGA. Όπως φαίνεται και στην Εικόνα 1.29, η δρομολόγηση του σήματος προς κάθε κατεύθυνση μπορεί να γίνει μέσω έξι μικροσκοπικών διακοπτών, στους οποίους πρέπει να φορτωθεί ένα ‘1’.



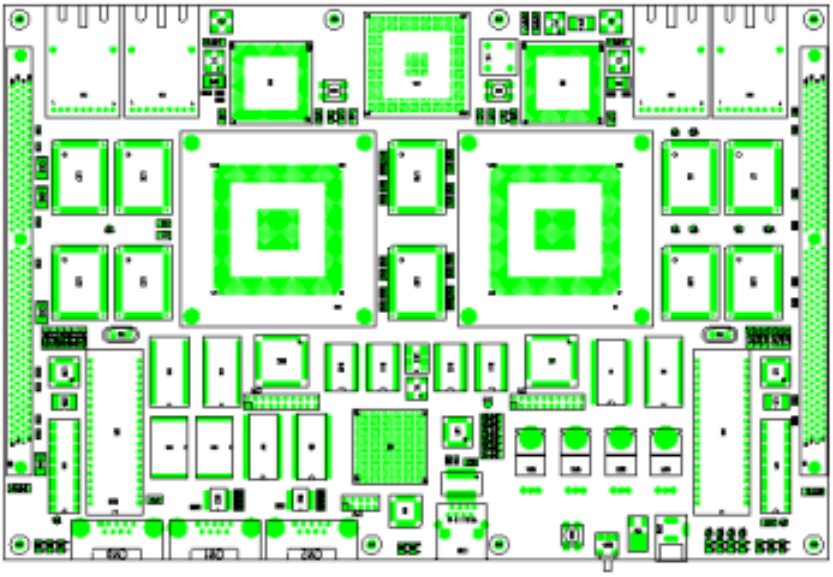
Εικόνα 1.29: Τρόπος διασύνδεσης ενός PSM

- Ο Προγραμματισμός του FPGA

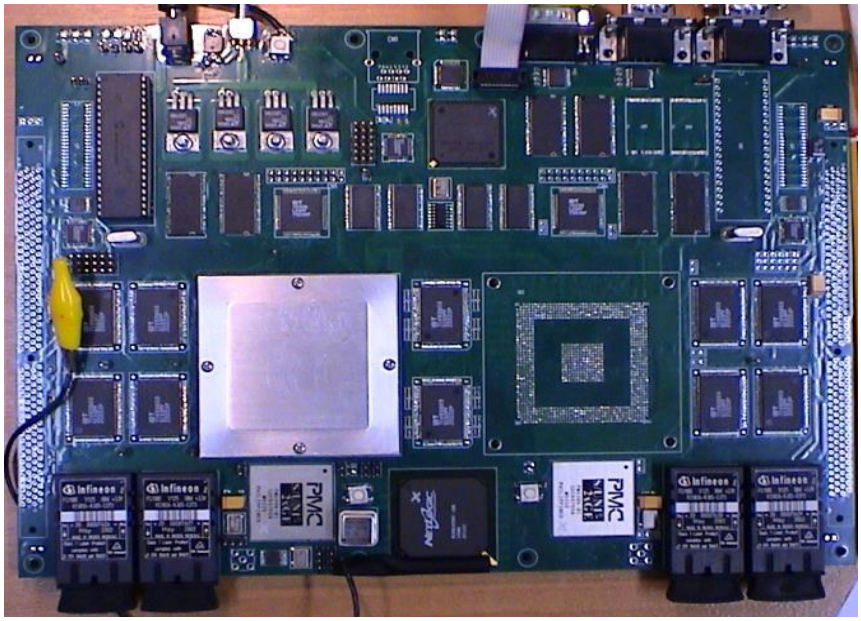
Επειδή όλα τα στοιχεία ενός FPGA (CLB, PSM, IOBs-μπλοκ εισόδου/εξόδου) αποτελούνται από διατάξεις τύπου RAM, τα περισσότερα FPGA board διαθέτουν και μια μνήμη PROM όπου αποθηκεύεται το κύκλωμα μας. Έτσι λοιπόν, κάθε φορά που δίνουμε τάση στην πλακέτα το FPGA αναπρογραμματίζεται και μια αλυσίδα από ‘0’ και ‘1’ αποθηκεύεται διαδοχικά στα διάφορα στοιχεία του, μέσω των pin εισόδου. Προφανώς η σχεδίαση και ο προγραμματισμός απαιτούν τη χρήση κατάλληλης εφαρμογής που αναλαμβάνει να επιτύχει την «αποτύπωση» του ψηφιακού κυκλώματος που σχεδιάστηκε ακολουθώντας τις παραπάνω μεθοδολογίες στους διαθέσιμους πόρους του FPGA που έχει επιλεγεί.

Σχεδίαση και κατασκευή ψηφιακών συστημάτων

Το τελικό στάδιο για την ολοκλήρωση των εφαρμογών ψηφιακών συστημάτων είναι η διασύνδεση όλων των απαραίτητων ηλεκτρονικών συστημάτων επί μίας πλακέτας, η οποία θα παρέχει και τις αναγκαίες ηλεκτρικές διασυνδέσεις και διεπαφές σύνδεσης και επικοινωνίας με εξωτερικά συστήματα. Η σχεδίαση αντίστοιχων τυπωμένων πλακετών (PrintedCircuit Board, PCB) χρησιμοποιεί τα αντίστοιχα εργαλεία σχεδίασης και κατασκευαστικές μεθοδολογίες, οι οποίες προσαρμόζονται στη ροή σχεδίασης των ολοκληρωμένων κυκλωμάτων καθώς απαιτείται διαλειτουγικότητα σε επίπεδο ηλεκτρικών και μηχανολογικών προδιαγραφών. Παράδειγμα τέτοιας σχεδίασης απεικονίζεται στην Εικόνα 1.30 όπου προβλέπεται η τοποθέτηση δύο chip αυτών που απεικονίζονται στην , ενώ στην Εικόνα 1.31 απεικονίζεται το τελικό λειτουργικό σύστημα μετά την τοποθέτηση όλων των ολοκληρωμένων και των ηλεκτρονικών υποσυστημάτων του. Στο τελευταίο κεφάλαιο του βιβλίου παρουσιάζονται ασκήσεις με χρήση αντίστοιχων αναπτυξιακών καρτών, που χρησιμοποιούν ως κεντρική επεξεργαστική μονάδα ολοκληρωμένα FPGA, για πειραματισμό με βασικά ψηφιακά κυκλώματα.



Εικόνα 1.30: Παράδειγμα PCBlayout (απεικόνιση σχεδιαστικού εργαλείου)



Εικόνα 1.31: Κατασκευασμένη και εξοπλισμένη κάρτα επεξεργασίας δεδομένων -ψηφιακό σύστημα

Με την διασύνδεση της κατάλληλης τροφοδοσίας στην πλακέτα και τη διασύνδεση με τα επιλεγμένα ηλεκτρονικά περιφερειακά συστήματα γίνεται δυνατή η εκτέλεση της εφαρμογής (εφόσον προηγηθεί ο προγραμματισμών των ολοκληρωμένων όπου απαιτείται). Με το τελικό αυτό στάδιο περνάμε στην πειραματική διαδικασία επί των εργαστηριακών διατάξεων με την οποία ολοκληρώνετε το παρόν βιβλίο.

1.4 Βιβλιογραφία

Ασημάκης Ν., Ψηφιακά Ηλεκτρονικά, Εκδόσεις GUTENBERG, 2008.
Ασημάκης Ν., Βουρβουλάκης Ι., Κακαρούντας Α., Λελίγκου Ε., Ηλεκτρονικό Βιβλίο Λογική Σχεδίαση, Τ.Ε.Ι. Λαμίας, 2011.
P. Ashenden, "*The Designer's Guide to VHDL*", Morgan Kaufman Publishers, 1996
S. Sjoholm and L. Lennart, "*VHDL for Designers*", Prentice Hall, 1997
B. Cohen, "*VHDL Coding Styles and Methodologies*", Kluwer Academic Publishers, 1999
Z. Navabi, "*VHDL-Analysis and Modeling of Digital Systems*", Mc Graw-Hill, 1993
ΨΗΦΙΑΚΗ ΣΧΕΔΙΑΣΗ (5η έκδοση) Morris MANO, Michael Ciletti ΠΑΠΑΣΩΤΗΡΙΟΥ 1, 2013 Αθήνα

Ψηφιακή Σχεδίαση με τη Γλώσσα VHDL Αρχές και Πρακτικές Δ. Πογαρίδης Β. ΓΚΙΟΥΡΔΑΣ, 2007 Αθήνα
Σχεδίαση ψηφιακών συστημάτων με τη γλώσσα VHDL S. Brown, Z. Vranesic ΤΖΙΟΛΑΣ, Θεσ/νίκη

Κεφάλαιο 2ο Συνδυαστικά κυκλώματα

2.1 Το δυαδικό σύστημα μέτρησης και η δυαδική λογική

2.1.1 Θεωρητικό Υπόβαθρο

Οποιοσδήποτε αριθμός μπορεί να εκφραστεί σε σύστημα μέτρησης με βάση τον αριθμό β, με μια σειρά $\alpha_v\alpha_{v-1}...\alpha_1\alpha_0$ από (ν+1) ψηφία τα οποία έχουν τιμές από 0 ως β-1. Η αναπαράσταση αυτή ουσιαστικά εκφράζει την αλγεβρική παράσταση

$\alpha_v\cdot\beta^v+ \alpha_{v-1}\cdot\beta^{v-1}+....+\alpha_1\cdot\beta^1+\alpha_0\beta^0$ (1)

Παράδειγμα:
Στο δεκαδικό σύστημα, η βάση μέτρησης είναι β=10 και τα ψηφία που χρησιμοποιούμε για να παραστήσουμε έναν αριθμό είναι 0, 1, ...9. Ο αριθμός 128 (α_0=8, α_1=2, α_2=1) ισούται με $1\cdot10^2+2\cdot10^1+8\cdot10^0$. Με άλλα λόγια, το 8 παριστάνει τις μονάδες, το 2 τις δεκάδες και το 1 τις εκατοντάδες.
Στο δυαδικό σύστημα, η βάση μέτρησης είναι το 2 δηλαδή β=2, τα ψηφία μπορούν να πάρουν **δύο μόνο τιμές το 0 και το 1 και κάθε συντελεστής α_i πολλαπλασιάζετε επί 2^i**.Τα δυαδικά ψηφία αλλιώς ονομάζονται bit.Έτσι το πλέον δεξιά ψηφία είναι ο συντελεστής των μονάδων, το επόμενο ο συντελεστής των δυάδων, το επόμενο των τετράδων κ.ο.κ.

Παράδειγμα:
Ο δυαδικός αριθμός 11010 παριστάνει τον δεκαδικό αριθμό
$1\cdot2^4+ 1\cdot2^3+ 0\cdot2^2+ 1\cdot2^1+ 0\cdot2^0= 1\cdot16+1\cdot8+0\cdot4+1\cdot2+0\cdot1= 26$
Στο δεκαδικό σύστημα, με κ ψηφία μπορούμε να παραστήσουμε 10^κ αριθμούς και να μετρήσουμε από το 0 ως το $10^{\kappa-1}$.
Στο δυαδικό σύστημα, με κ ψηφία μπορούμε να παραστήσουμε μέχρι 2^κ αριθμούς και να μετρήσουμε από το 0 ως το $2^{\kappa-1}$.
Με άλλα λόγια, για να παρασταθεί μια ομάδα 2^nδιακεκριμένων στοιχείων με κάποιον δυαδικό κώδικα απαιτούνται τουλάχιστον *η* bit, διότι n bit μπορούν να μπουν στη σειρά με 2^nδιαφορετικούς τρόπους.

Παράδειγμα:
Στο δεκαδικό σύστημα, με 2 ψηφία μπορούμε να γράψουμε 100 αριθμούς οι οποίοι είναι 0, 1, 2, ... 99.
Στο δυαδικό σύστημα, με 3 ψηφία μπορούμε να παραστήσουμε 2^3 = 8 αριθμούς οι οποίοι είναι 000, 001, 010, 011, 100, 101, 110, 111.

Μετατροπή δυαδικών αριθμών

Για να μετατρέψουμε έναν δυαδικό αριθμό σε δεκαδικό δεν έχουμε παρά να εφαρμόσουμε την παράσταση 1 για β=2, όπως στο παραπάνω παράδειγμα.
Για να μετατρέψουμε έναν δεκαδικό αριθμό σε δυαδικό πρέπει να τον εκφράσουμε ως άθροισμα δυνάμεων του 2. Δηλαδή να τον φέρουμε στη μορφή
$\alpha_v\cdot2^v+ \alpha_{v-1}\cdot2^{v-1}+....+\alpha_1\cdot2^1+\alpha_0 2^0$
Για να το πετύχουμε αυτό διαιρούμε επανειλημμένα με το 2 μέχρι το πηλίκο να είναι 1 και επιλέγουμε το 1 και τα υπόλοιπα των διαιρέσεων όπως φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα
Έστω ότι θέλουμε να μετατρέψουμε τον δεκαδικό αριθμό 35 στο δυαδικό σύστημα
35:2=17 και υπόλοιπο 1 35=**1**00011 **(=** $1\cdot32+0\cdot16+0\cdot8+0\cdot4+1\cdot2+1\cdot1$)
17:2=8 και υπόλοιπο 1
8:2 =4 και υπόλοιπο 0
4:2 =2 και υπόλοιπο 0
2:2=(**1**) και υπόλοιπο 0

Παράδειγμα:
Έστω ότι θέλουμε να μετατρέψουμε τον δεκαδικό αριθμό 27 στο δυαδικό σύστημα
27:2=13 και υπόλοιπο 1
13:2=6 και υπόλοιπο 1
6:2=3 και υπόλοιπο 0
3:2=**1** και υπόλοιπο 1
Άρα 27=**1**1011=1·16+1·8+0·4+1·2+1·1

Τα ηλεκτρονικά ψηφιακά συστήματα χρησιμοποιούν σήματα που έχουν δύο διακριτές τιμές και ηλεκτρονικά κυκλώματα που έχουν δύο σταθερές καταστάσεις. Υπάρχει μία άμεση αναλογία μεταξύ των δυαδικών σημάτων, των δυαδικών ηλεκτρονικών κυκλωμάτων και των δυαδικών ψηφίων. Τα ψηφιακά συστήματα, όμως, αναπαριστούν και χειρίζονται όχι μόνο δυαδικούς αριθμούς, αλλά και πολλά άλλα διακριτά στοιχεία πληροφορίας. Κάθε διακριτό στοιχείο πληροφορίας ξεχωριστό μέλος μιας πεπερασμένης ομάδας μπορεί να παρασταθεί με έναν δυαδικό κώδικα. Οι κώδικες πρέπει να είναι δυαδικοί γιατί οι υπολογιστές μπορούν να χειριστούν τα ψηφία 1 και 0 μόνον. Όταν χρησιμοποιείται σε σχέση με κάποιον δυαδικό κώδικα είναι καλύτερα να σκεφτόμαστε τα bit ως μια δυαδική ποσότητα ίση με 0 ή 1.

Άσκηση

1. Να μετατρέψετε στο δυαδικό τους αριθμούς 6, 3, 8, 16, 32, 128.
2. Να μετατρέψετε σε δεκαδικό τους αριθμούς 10000000, 1000, 100, 111011, 1010.
3. Πόσα bit χρειαζόμαστε για να περιγράψουμε 8 αριθμούς στο δυαδικό σύστημα;

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 2.1

Δυαδική λογική

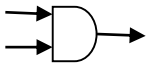
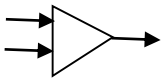
Η **δυαδική λογική** ασχολείται με μεταβλητές που μπορούν να έχουν δύο μόνο διακριτές τιμές και με "λογικές" (δυαδικές) πράξεις. Η δυαδική λογική αποτελείται από δυαδικές μεταβλητές και λογικές πράξεις. Οι μεταβλητές συμβολίζονται με γράμματα του αλφαβήτου, όπως A, B, C, X, Y, Z, κλπ. και κάθε μεταβλητή έχει δύο και μόνο δύο διακριτές δυνατές τιμές: ένα και μηδέν.
Υπάρχουν **τρεις βασικές λογικές πράξεις**; "ΚΑΙ" (AND), "Η" (OR) και "ΟΧΙ" (NOT).

➢ ΚΑΙ (AND): Αυτή η πράξη παριστάνεται με μία τελεία ή και χωρίς . Πρόκειται για τη λογική **ΣΥΖΕΥΞΗ**. Το αποτέλεσμα της πράξης είναι 1 μόνο όταν **και** οι δύο όροι (είσοδοι) είναι 1.
Δηλαδή στη λογική πράξη ΚΑΙ (AND) που συμβολίζεται με z=xy, το z = 1 όταν και μόνον όταν το x = 1 και το y = 1, διαφορετικά το z = 0. (Υπενθυμίζουμε ότι τα x,y και z είναι δυαδικές μεταβλητές και μπορούν να είναι ίσες είτε με το 1 είτε με το 0, και τίποτα άλλο).

➢ Ή (OR)· Αυτή η πράξη παριστάνεται με το σύμβολο συν (σταυρό). Πρόκειται για τη λογική **ΔΙΑΖΕΥΞΗ**. Για παράδειγμα: ***x + y = z* διαβάζεται *"x Ή y* ίσον *z"***, που σημαίνει ότι το *z* =1 αν το x = 1 ή αν το y = 1 ή αν και τα δύο είναι x = 1 και y =*1*. Αν και τα δύο είναι *x* = 0 και y = 0, τότε *z = 0*.

➢ ΟΧΙ (NOT): Αυτή η πράξη συμβολίζεται με μία οξεία μετά το γράμμα ή με μία παύλα πάνω από το γράμμα. Πρόκειται για τη λογική άρνηση. Για παράδειγμα, x′ =*z* ή *x* =*z*, διαβάζεται "όχι *x* ίσον *z"*, που σημαίνει ότι το *z* είναι το "αντίθετο" ή "αντίστροφο" του x*:* αν το x= 1, τότε το *z*= 0 και αν το x=0,τότε το z=1

Για κάθε συνδυασμό τιμών των x και y υπάρχει μια τιμή του z που προσδιορίζεται από τον ορισμό της λογικής πράξης. Αυτοί οι ορισμοί μπορούν να συγκεντρωθούν στον **"πίνακας αληθείας"**. Ο πίνακας αληθείας είναι ένας πίνακας όλων των δυνατών συνδυασμών των τιμών μεταβλητών οπού φαίνεται η σχέση μεταξύ των τιμών που μπορούν να πάρουν οι μεταβλητές και του αποτελέσματος της πράξης. Οι παραπάνω λογικές πράξεις υλοποιούνται από κυκλώματα που ονομάζονται **πύλες.**

ΟΝΟΜΑ	*ΓΡΑΦΙΚΟ ΣΥΜΒΟΛΟ*	*ΑΛΓΕΒΡΙΚΗ ΣΥΝΑΡΤΗΣΗ*	*ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΑΣ*
AND *ΚΑΙ*	*AND* 	*F=XY* *ΈΞΟΔΟΣ 0 (ΔΕΝ ΑΝΑΒΕΙ ΤΟ LED)* *ΈΞΟΔΟΣ 1 (ΑΝΑΒΕΙ ΤΟ LED)* *ΠΡΕΠΕΙ ΚΑΙ ΟΙ ΔΥΟ ΕΙΣΟΔΟΙ ΝΑ ΕΧΟΥΝ ΤΑΣΗ ΓΙΑ ΝΑ ΈΧΟΥΜΕ ΈΞΟΔΟ 1*	*X Y F* *0 0 0* *0 1 0* *1 0 0* *1 1 1*
OR *Η*	*OR* 	*F=X+Y* *ΈΣΤΩ ΜΙΑ ΕΙΣΟΔΟΣ ΝΑ ΕΙΝΑΙ 1 ΤΟΤΕ ΤΟ LED ΑΝΑΒΕΙ ΚΑΙ ΈΧΟΥΜΕ ΕΞΟΔΟ 1*	*X Y F* *0 0 0* *0 1 1* *1 0 1* *1 1 1*
NOT *ΟΧΙ*	*NOT* 	*F=X΄* *ΌΤΑΝ Η ΕΙΣΟΔΟΣ ΕΧΕΙ ΤΑΣΗ ΤΟΤΕ ΤΟ LED ΔΕΝ ΑΝΑΒΕΙ ΔΗΛΑΔΗ ΈΧΟΥΜΕ ΕΞΟΔΟ 0.* *ΌΤΑΝ Η ΕΙΣΟΔΟΣ ΔΕΝ ΈΧΕΙ ΤΑΣΗ ΤΟΤΕ ΤΟ LED ΑΝΑΒΕΙ ΚΑΙ ΈΧΟΥΜΕ ΈΞΟΔΟ 1*	*X F* *0 1* *1 0*

Εικόνα2.1:Λογικές Πύλες

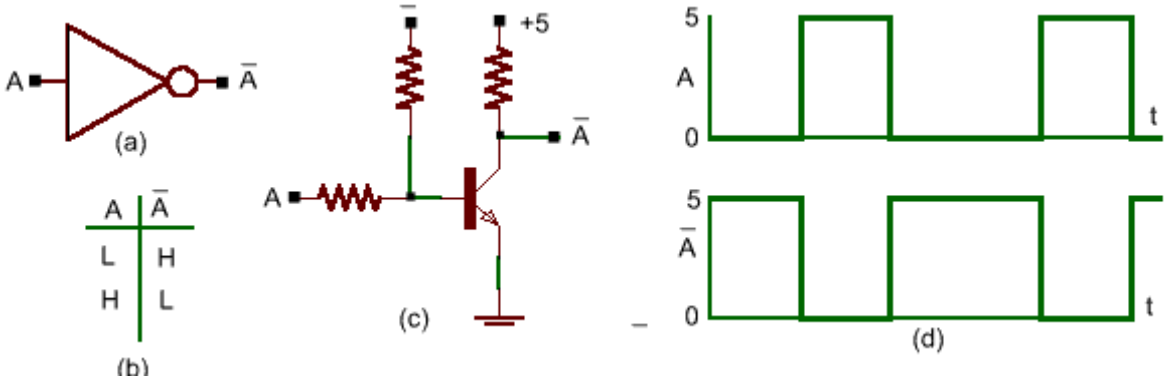
Οι λογικές πύλες IC (Integrated Circuit) ταξινομούνται όχι μονό με τη λογική τους λειτουργία αλλά επίσης και με την οικογένεια στην οποία ανήκουν. Οι πιο διαδεδομένες είναι οι εξής: TTL, ECL, MOS, CMOS. Η TTL είναι μια πολύ διαδεδομένη λογική οικογένεια που διατίθεται σε όλα τα καταστήματα ηλεκτρονικών ειδών σε ιδιαίτερα χαμηλές τιμές γεγονός που την καθιστά κατάλληλη για πειραματισμό.

Τα χαρακτηριστικά των οικογενειών ψηφιακών κυκλωμάτων συγκρίνονται με τις εξής παραμέτρους:

1. η ικανότητα οδήγησης: είναι ο αριθμός των τυπικών φορτιών που μπορεί να οδηγήσει η έξοδος μιας πύλης χωρίς να κινδυνέψει η κανονική της λειτουργίας.
2. η κατανάλωση ισχύος: είναι η ισχύς τροφοδοσίας που απαιτείται για να λειτουργήσει η πύλη.
3. η καθυστέρηση διάδοσης: είναι ο μέσος χρόνος που χρειάζεται για να διαδοθεί η αλλαγή ενός σήματος από την είσοδο στην έξοδο μιας πύλης.
4. το περιθώριο θορύβου: είναι η ελάχιστη τάση εξωτερικού θορύβου που προκαλεί ανεπιθύμητη αλλαγή στην έξοδο.

Κάθε IC περικλείεται σε περίβλημα με 14 ή 16 ακροδέκτες. Μια εγκοπή στην αριστερή πλευρά του πακέτου χρησιμοποιείται για αναφορά στην αρίθμηση των εξωτερικών ακροδεκτών. Οι ακροδέκτες αριθμούνται κατά μήκος των δυο πλευρών, ξεκινώντας από την εγκοπή και συνεχίζοντας αντίθετα με τη φορά των δεικτών του ρολογιού. Οι είσοδοι και οι έξοδοι των πυλών συνδέονται στους εξωτερικούς ακροδέκτες του περιβλήματος.
Το δυαδικό σήμα στις εισόδους και εξόδους κάθε πύλης έχει μια από δυο τιμές. Κατά τη διάρκεια της μετάβασης από τη μια τιμή στην άλλη, μια τιμή του σήματος αντιπροσωπεύει το λογικό 1 και η άλλη το λογικό

0. Έχουμε δυο αντιστοιχίσεις τιμής σήματος. Η υψηλότερη τιμή σήματος συμβολίζεται με Η και η χαμηλότερη με L. Επιλέγοντας την Η ορίζουμε ένα σύστημα θετικής λογικής. Επιλέγοντας L ορίζουμε ένα σύστημα αρνητικής λογικής. Ο τύπος της λογικής καθορίζεται από την αντιστοίχηση των λογικών τιμών στις δυο τιμές του σήματος. Τα τεχνικά εγχειρίδια για ολοκληρωμένα κυκλώματα ορίζουν τις ψηφιακές πύλες όχι με βάση τις λογικές τιμές, αλλά με βάση τις τιμές σήματος, όπως Η και L. Στην Εικόνα 2.2 φαίνεται το σύμβολο, ο πίνακας αληθείας, το κύκλωμα και πιθανό στιγμιότυπο εισόδου και αντίστοιχο στιγμιότυπο εξόδου της πύλης NOT.



Εικόνα 2.2: Η πύλη NOT- α) το σύμβολο μιας πύλης NOT, β) ο πίνακας αληθείας της, c) το κύκλωμα που την υλοποιεί και d) ένα πιθανό στιγμιότυπο εισόδου και εξόδου αυτής.

2.1.2 Πειραματικό Μέρος

Οι πύλες NOT βρίσκονται στο ολοκληρωμένο 7404.
Οι πύλες OR βρίσκονται στο ολοκληρωμένο 7432.
Οι πύλες AND βρίσκονται στο ολοκληρωμένο 7408.

7404
Eightinverters

```
   +--()---+
 A1 |1   14| VCC
!Y1 |2   13| A6
 A2 |3   12| !Y6
!Y2 |4   11| A5
A3 |5   10| !Y5
!Y3 |6    9| A4
GND |7    8| !Y4
 +-------+
```

7432
Quad 2-input OR gates

```
    +---\/---+
 1 -|1A   Vcc|- 14
 2 -|1B    4A|- 13
 3 -|1Y    4B|- 12
 4 -|2A    4Y|- 11
5 -|2B    3A|- 10
 6 -|2Y    3B|-  9
 7 -|gnd   3Y|-  8
    +--------+
```

7408
Quad 2-input AND gates

```
    +---\/---+
 1 -|1A   Vcc|- 14
 2 -|1B    4A|- 13
 3 -|1Y    4B|- 12
 4 -|2A    4Y|- 11
5 -|2B    3A|- 10
 6 -|2Y    3B|-  9
 7 -|gnd   3Y|-  8
    +--------+
```

Εικόνα 2.3: Διάταξη με τα pin των ολοκληρωμένων

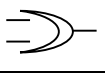
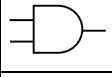
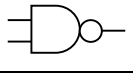
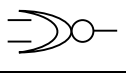
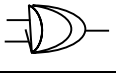
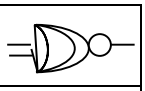
Ακολουθήστε τα παρακάτω βήματα

1. Συνδέστε τα pin τροφοδοσίας του ολοκληρωμένου 7404.
2. Συνδέστε μια είσοδο σε διακόπτη και την αντίστοιχη έξοδο σε ένα LED.
3. Επαληθεύστε τον πίνακα αληθείας της πύλης που περιέχεται στο ολοκληρωμένο.
4. Επαναλάβετε τα παραπάνω για τα ολοκληρωμένα 7432 και 7408. Οι πύλες που περιέχουν αυτά τα ολοκληρωμένα έχουν 2 εισόδους οπότε θα χρειαστεί να συνδέσετε 2 pin σε διακόπτες.

2.2 Ανάλυση κυκλωμάτων

2.2.1 Θεωρητικό Υπόβαθρο

Ένα κύκλωμα μπορεί εκτός από τις απλές πύλες που μάθαμε παραπάνω να περιέχει και αυτές που φαίνονται στον Εικόνα 2.4.

							
x	y	OR	AND	NAND	NOR	XOR	XNOR
0	0	0	0	1	1	0	1
0	1	1	0	1	0	1	0
1	1	1	0	1	0	1	0
1	1	1	1	0	0	0	1

Εικόνα2.4: Πίνακας αληθείας για βασικές πύλες

Η άλγεβρα Boole είναι μία αλγεβρική δομή ορισμένη πάνω σ' ένα σύνολο στοιχείων B, μαζί με δύο δυαδικούς τελεστές + και · , αρκεί να ικανοποιούνται τα παρακάτω αξιώματα :

1. (α) Κλειστή ως προς τον τελεστή + .
   (β) Κλειστή ως προς τον τελεστή · .
2. (α) Ένα ουδέτερο στοιχείο ως προς +, που συμβολίζεται με 0:
   $x + 0 = 0 + x = x$.
   (β) Ένα ουδέτερο στοιχείο ως προς · , που συμβολίζεται με 1:
   $x \cdot 1 = 1 \cdot x = x$.
3. (α) Αντιμεταθετική ως προς +:
   $x + y = y + x$.
   (β) Αντιμεταθετική ως προς · :
   $x \cdot y = y \cdot x$.
4. (α) Ο · είναι επιμεριστικός ως προς +:
   $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (β) Ο + είναι επιμεριστικός ως προς ·:
   $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. Για κάθε στοιχείο $x \in B$, υπάρχει ένα στοιχείο $x' \in B$ (που ονομάζεται "συμπλήρωμα" του x) τέτοιο, ώστε:
   (α) $x + x' = 1$ και
   (β) $x \cdot x' = 0$
6. Υπάρχουν τουλάχιστον δύο στοιχεία $x, y \in B$ που να είναι $x \neq y$.

Συγκρίνοντας την άλγεβρα Boole με την αριθμητική και τη συνηθισμένη άλγεβρα , παρατηρούμε τις παρακάτω διαφορές:

1. Τα αξιώματα του Huntington δεν περιλαμβάνουν τον προσεταιριστικό νόμο.
2. Ο επιμεριστικός νόμος του + ως προς τον · , δηλαδή $x + (y \cdot z) = (x + y) \cdot (x + z)$, ισχύει για την άλγεβρα του Boole, αλλά όχι και για τη συνηθισμένη άλγεβρα.
3. Η άλγεβρα Boole δεν έχει προσθετικά ή πολλαπλασιαστικά αντίστροφα επομένως δεν υπάρχουν πράξεις αφαίρεσης ή διαίρεσης.
4. Το αξίωμα 5 ορίζει έναν τελεστή, που καλείται συμπλήρωμα, ο οποίος δεν υπάρχει στη συνηθισμένη άλγεβρα.
5. Η συνηθισμένη άλγεβρα ασχολείται με τους πραγματικούς αριθμούς, που αποτελούν ένα άπειρο σύνολο. Η άλγεβρα Boole ασχολείται με το σύνολο στοιχείων *B,* που ακόμα δεν το προσδιορίσαμε.

Στην άλγεβρα Boole δύο τιμών που ορίζουμε παρακάτω το *B* έχει μόνο δύο στοιχεία, τα 0 και 1.

Αξίωμα 2	$\mathbf{x + 0 = x}$	$\mathbf{x \cdot 1 = x}$
Αξίωμα 5	$\mathbf{x + x' = 1}$	$\mathbf{x \cdot x' = 1}$
Θεώρημα 1	$\mathbf{x + x = x}$	$\mathbf{x \cdot x = x}$

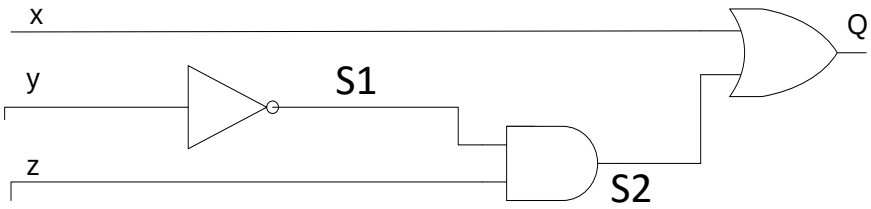
Θεώρημα 2	**x + 1 = 1**	**x · 0 = 0**
Θεώρημα 3	**(x')' =x**	
Αξίωμα 3	**x + y =y + x**	**x y = y x**
Θεώρημα 4	**x + (y+ z) = (x + y) + z**	**x(y z)=(x y) z**
Αξίωμα 4	**x (y + z) = x y+ x z**	**x+ y z = (x + y) (x + z)**
Θεώρημα 5	**(x + y)´ = x´ y´**	**(x y)' =x´ + y´**
Θεώρημα 6	**x + x y = x**	**x (x + y)=x**

Πίνακας 2.1: Θεωρήματα και Αξιώματα στην Άλγεβρα Boole

Η **προτεραιότητα των τελεστών** για τον υπολογισμό των εκφράσεων Boole ακολουθεί τη σειρά:
(1) παρενθέσεις,
(2) ΟΧΙ,
(3) ΚΑΙ και
(4) Ή.
Με άλλα λόγια, κάθε έκφραση μέσα σε παρενθέσεις πρέπει να υπολογίζεται πριν γίνουν άλλες πράξεις. Η επόμενη πράξη που έχει προτεραιότητα είναι το συμπλήρωμα, μετά ακολουθεί η ΚΑΙ, και τέλος η Ή. Το αριστερό μέλος της έκφρασης είναι *(x + y)'*. Το δεξιό μέλος της έκφρασης είναι *x'y'*. Άρα, πρώτα υπολογίζονται τα συμπληρώματα των *x* και *y*, και μετά το «ΚΑΙ» τους.

2.2.2 Πειραματικό Μέρος

1. Υπολογίστε **θεωρητικά** τον πίνακα αληθείας του παρακάτω κυκλώματος:



Υπόδειξη:

Ονομάζουμε S1 το σήμα εξόδου της πύλης NOT και S2 το σήμα εξόδου της πύλης AND.
Υπολογίζουμε πρώτα την έξοδο S1 για κάθε συνδυασμό των εισόδων του κυκλώματος, μετά την έξοδο S2 (πάλι για κάθε συνδυασμό των εισόδων του κυκλώματος) και τέλος την έξοδο Q.

x	y	z	S1	S2	Q
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

2. Περιγράψτε την έξοδο του κυκλώματος συναρτήσει των εισόδων του.

Διαδραστικό πρόγραμμα 2.2

3. Επαληθεύστε τον πίνακα αληθείας που υπολογίσατε, χρησιμοποιώντας τα κατάλληλα ολοκληρωμένα κυκλώματα.

Ακολουθήστε τα παρακάτω βήματα

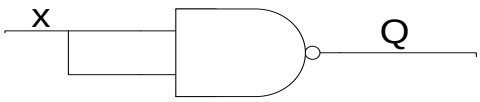
1. Κάνουμε τις συνδέσεις χωρίς να συνδέσουμε το τροφοδοτικό.
2. Ελέγχουμε τις συνδέσεις.
3. Ρυθμίζουμε το τροφοδοτικό και το συνδέουμε στο κύκλωμα.
4. Καταγραφή των αποτελεσμάτων.

Υπενθυμίζεται ότι:
Οι πύλες NOT βρίσκονται στο ολοκληρωμένο 7404.
Οι πύλες OR βρίσκονται στο ολοκληρωμένο 7432.
Οι πύλες AND βρίσκονται στο ολοκληρωμένο 7408.

Άσκηση

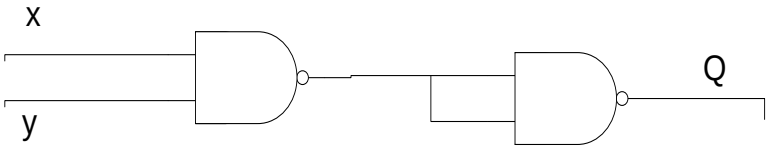
Να βρείτε τον πίνακα αληθείας των παρακάτω κυκλωμάτων και να τους συγκρίνετε με τους πίνακες αληθείας των πυλών AND, OR, NOT.

α



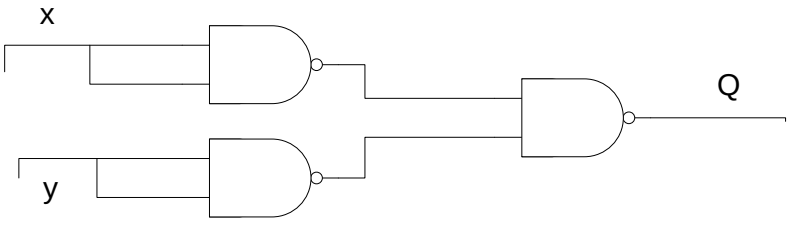
1 AND

β



2 OR

γ



3 NOT

Αντιστοιχίστε τα παραπάνω κυκλώματα στις πύλες.

Διαδραστικό πρόγραμμα 2.3

2.3 Ισοδύναμο με ΠΥΛΕΣ NAND

2.3.1 Θεωρητικό Υπόβαθρο

Με βάση τα αποτελέσματα της άσκησης της προηγούμενης παραγράφου, σε ένα κύκλωμα που αποτελείται από πύλες AND, OR, NOT, αυτές μπορούν να αντικατασταθούν από πύλες NAND.

2.3.2 Πειραματικό Μέρος

1. Υλοποιήστε τη συνάρτηση $F_1 = (y'+x) \cdot z$ χρησιμοποιώντας τις κατάλληλες πύλες και βρείτε θεωρητικά και πειραματικά τον πίνακα αληθείας.
2. Υλοποιήστε την παραπάνω συνάρτηση χρησιμοποιώντας **μόνο** πύλες NAND και βρείτε θεωρητικά και πειραματικά τον πίνακα αληθείας.
3. Υλοποιείστε με πύλες AND, OR και NOT, τη συνάρτηση $F(x, y, z)= \Sigma(0, 1, 3, 6)$

ΠΟΡΕΙΑ:

1. Σχεδιάζουμε το κύκλωμα με πύλες AND, OR και NOT.
2. Κάνουμε τις συνδέσεις χωρίς να συνδέσουμε το τροφοδοτικό.
3. Ελέγχουμε τις συνδέσεις.
4. Ρυθμίζουμε το τροφοδοτικό και το συνδέουμε στο κύκλωμα.
5. Καταγραφή των αποτελεσμάτων.
6. Σχεδιάζουμε το ισοδύναμο του παραπάνω κυκλώματος με NAND.
7. Εκτελούμε τα βήματα 2 ως 5 για το νέο κύκλωμα που σχεδιάσαμε.

Υπενθυμίζεται ότι:
Οι πύλες NAND βρίσκονται στο ολοκληρωμένο 7400.
Οι πύλες XOR βρίσκονται στο ολοκληρωμένο 7486.

Ασκήσεις

1. Να σχεδιάσετε το κύκλωμα που υλοποιεί τη συνάρτηση $F(x,y,z,w)=\Sigma(\alpha, \beta, \gamma)$ όπου α, β, γ τα τρία τελευταία ψηφία του μητρώου σας, με πύλες AND, OR, NOT.
2. Να σχεδιάσετε το ισοδύναμο του παραπάνω κυκλώματος με πύλες NAND.

2.4 Απλοποίηση συναρτήσεων με χάρτη Καρνώ

2.4.1 Θεωρητικό υπόβαθρο

Ελαχιστόροι και Μεγιστόροι

Θεωρείστε δύο μεταβλητές x και y που ενώνονται με την πράξη ΚΑΙ. Αφού κάθε μεταβλητή μπορεί να εμφανιστεί με μια από τις δύο μορφές, υπάρχουν τέσσερις πιθανοί συνδυασμοί: x'y', x'y, xy' και xy. Καθένας απ' αυτούς τους τέσσερις όρους ΚΑΙ αντιστοιχεί σε μία από τις 4 διαφορετικές περιοχές στο διάγραμμα Venn και λέγεται "ελαχιστόρος". Με παρόμοιο τρόπο, οι μεταβλητές μπορούν να συνδυαστούν και να σχηματίσουν 2^nελαχιστόρους. Οι δυαδικοί αριθμοί από το 0 μέχρι το 2^{n-1} γράφονται κάτω από τις η μεταβλητές. Κάθε ελαχιστόρος βγαίνει παίρνοντας το "ΚΑΙ" των n μεταβλητών, όπου κάθε μεταβλητή εμφανίζεται με το συμπλήρωμα της, αν το αντίστοιχο bit του δυαδικού αριθμού είναι 0 ή κανονικά, αν αυτό είναι 1. Ο πίνακας ορίζει επίσης ένα σύμβολο της μορφής m_j για κάθε ελαχιστόρο, όπου το j είναι το δεκαδικό ισοδύναμο του δυαδικού αριθμού που αντιστοιχεί στον ελαχιστόρο.
Με παρόμοιο τρόπο, αν φτιάξουμε το "άθροισμα" (Ή) *n* μεταβλητών, όπου κάθε μεταβλητή μπορεί προαιρετικά να είναι "συμπληρωμένη", παίρνουμε 2^nδιαφορετικούς συνδυασμούς, που ονομάζονται "μεγιστόροι". Οι οκτώ μεγιστόροι των τριών μεταβλητών, μαζί με τη συμβολική τους ονομασία, φαίνονται στην Εικόνα 2.5.

Ελαχιστόροι					Μεγιστόροι	
X	Y	Z	Όρος	Ονομασία	Όρος	Ονομασία
0	0	0	x΄y΄z΄	m0	x+y+z	M0
0	0	1	x΄y΄z	m1	x+y+z΄	M1
0	1	0	x΄yz΄	m2	x+y΄+z	M2
0	1	1	x΄yz	m3	x+y΄+z΄	M3
1	0	0	xy΄z΄	m4	x΄+y+z	M4
1	0	1	xy΄z	m5	x΄+y+z΄	M5
1	1	0	xyz΄	m6	x΄+y΄+z	M6
1	1	1	xyz	m7	x΄+y΄+z΄	M7

Εικόνα1.5: Ελαχιστόροι και μεγιστόροι για 3 μεταβλητές

Οι 2^nμεγιστόροι για ημεταβλητές μπορούν να υπολογιστούν παρόμοια. Κάθε μεγιστόρος είναι το άθροισμα των *n*μεταβλητών όπου κάθε μεταβλητή εμφανίζεται με το συμπλήρωμα της, αν το αντίστοιχο M_i είναι 1 ή "σκέτη", αν αυτό είναι 0. Παρατηρείστε ότι κάθε μεγιστόρος είναι το συμπλήρωμα του αντιστοίχου του ελαχιστόρου, και αντίστροφα.

Ο χάρτης Καρνώ

Ο "χάρτης" για τον οποίο μιλάμε είναι ένα διάγραμμα αποτελούμενο από τετράγωνα. Κάθε τετράγωνο παριστάνει έναν ελαχιστόρο. **Επειδή κάθε συνάρτηση Boole μπορεί να εκφραστεί ως άθροισμα ελαχιστόρων, έπεται ότι μια συνάρτηση Boole αναγνωρίζεται γραφικά στο χάρτη από την περιοχή που καλύπτουν τα τετράγωνα των ελαχιστόρων που περιέχονται στη συνάρτηση.** Δηλαδή, ο χάρτης είναι ένα σχηματικό διάγραμμα όλων των δυνατών τρόπων, με τους οποίους η συνάρτηση μπορεί να εκφραστεί σε πρότυπη μορφή.
Για δύομεταβλητές υπάρχουντέσσεριςελαχιστόροι, κιέτσι ο χάρτης αποτελείται από τέσσερα τετράγωνα, ένα για κάθεελαχιστόρο.

m_0	m_1
m_2	m_3

(α)

x\y	0	1
0	x'y'	x'y
1	xy'	xy

(β)

Εικόνα2.6: Χάρτης Καρνώ για δύο μεταβλητές

Ο πίνακας έχει ξανασχεδιαστεί στην Εικόνα 2.6 (β) έτσι που να φαίνεται η σχέση ανάμεσα στα τετράγωνα και τις δύο μεταβλητές. Τα 0 και 1 που σημειώνονται για κάθε γραμμή και στήλη καθορίζουν τις τιμές των μεταβλητών xκαι *y*.
Για **τρεις δυαδικές μεταβλητές**, υπάρχουν οκτώ ελαχιστόροι. Έτσι, ο χάρτης αυτός αποτελείται από οκτώ τετράγωνα. Παρατηρείστε ότι οι ελαχιστόροι έχουν τοποθετηθεί σε σειρά όχι δυαδικών αριθμών, αλλά σε σειρά όμοια με τον κώδικα *GRAY*. Το χαρακτηριστικό αυτής της σειράς είναι ότι μονάχα ένα bit αλλάζει από 1 σε 0 ή από 0 σε 1 σε κάθε βήμα καθώς ακολουθούμε τη σειρά.

x \ yz	00	01	11	10
0				
1				

Εικόνα 2.7: Χάρτης Καρνώ για τρεις μεταβλητές

Η μεταβλητή εμφανίζεται ως έχει σ' εκείνα τα τετράγωνα, όπου ισούται με 1, και με το συμπλήρωμα της στα τετράγωνα, όπου ισούται με 0. Για ευκολία γράφουμε τ' όνομα κάθε μεταβλητής κάτω από τα τέσσερα τετράγωνα, όπου αυτή είναι με την πραγματική της τιμή.

<u>Οποιαδήποτε δύο γειτονικά τετράγωνα στο χάρτη διαφέρουν κατά μια μόνο μεταβλητή</u>, η οποία εμφανίζεται σαν το συμπλήρωμα της στο ένα τετράγωνο και με την πραγματική της τιμή στο άλλο. Για παράδειγμα τα m_5και m_7βρίσκονται σε δύο γειτονικά τετράγωνα.

Η μεταβλητή y είναι με το συμπλήρωμα της στο m_5 και με την πραγματική της τιμή στο m_7, ενώ οι δύο άλλες μεταβλητές είναι ίδιες και στα δύο τετράγωνα. Από τα αξιώματα της άλγεβρας Boole έπεται ότι το άθροισμα των δύο ελαχιστόρων σε γειτονικά τετράγωνα μπορεί να απλοποιηθεί σε έναν όρο «ΚΑΙ» (AND) με δύο μόνο παράγοντες. Για να ξεκαθαριστεί αυτό, θεωρείστε το άθροισμα δύο γειτονικών τετραγώνων, π.χ. των m_5 και m_7

$m_5 + m_7 = xy'z+xyz=xz(y'+y)=xz$

Παράδειγμα

Απλοποιείστε τη συνάρτηση Boole: $F(x,y,z)$ = X(2,3,4,5)

Βήμα 1

Τοποθετούμε στο χάρτη τις τιμές της συνάρτησης

Αρχικά, ένα (1) σημειώνεται σε κάθε ελαχιστόρο που αντιπροσωπεύει την συνάρτηση. Αυτό φαίνεται στο σχήμα όπου τα τετράγωνα για τους ελαχιστόρους 010, 011, 100 και 101 έχουν σημειωθεί με 1.

x \ yz	00	01	11	10
0			1	1
1	1	1		

Βήμα 2

Το επόμενο βήμα είναι να βρεθούν τυχόν γειτονικά τετράγωνα. Αυτά, σημειώνονται στο χάρτη με δύο ορθογώνια, το καθένα από τα οποία περικλείει δύο άσσους.

Το πάνω δεξιά ορθογώνιο παριστάνει την περιοχή *x'y*. Αυτό φαίνεται, παρατηρώντας ότι η περιοχή των δύο τετραγώνων βρίσκεται στην γραμμή 0,που αντιστοιχεί στο x', και στις δύο τελευταίες στήλες, που αντιστοιχούν στο y.

Όμοια, το κάτω αριστερά ορθογώνιο παριστάνει τον όρο *xy''*. (Η δεύτερη γραμμή αντιστοιχεί στο *x'* και οι δύο αριστερές στήλες στο *y'*). Το λογικό άθροισμα των δύο αυτών γινομένων δίνει την απλοποιημένη έκφραση:

$F = x'y+xy'$

Υπάρχουν περιπτώσεις, όπου δύο τετράγωνα στο χάρτη θεωρούνται γειτονικά αν και δεν "ακουμπούν" μεταξύ τους. Στην Εικόνα 2.8, το m_0 είναι γειτονικό του m_2 και το m_4 είναι γειτονικό του m_6, διότι οι ελαχιστόροι διαφέρουν κατά μία μεταβλητή. Αυτό μπορεί εύκολα να επαληθευτεί αλγεβρικά.

Επομένως, πρέπει να τροποποιήσουμε τον ορισμό των γειτονικών τετραγώνων, ώστε να περιλαμβάνει την παραπάνω, αλλά και άλλες παρόμοιες περιπτώσεις. Αυτό γίνεται θεωρώντας ότι ο χάρτης σχεδιάζεται σε μία επιφάνεια, όπου οι δεξιές και αριστερές ακμές αγγίζουν η μία την άλλη, ώστε να δίνουν γειτονικά τετράγωνα.

Η τοποθέτηση των ελαχιστόρων στο χάρτη Καρνώ για τρεις και τέσσερις μεταβλητές φαίνεται στην Εικόνα 2.8.

x \yz	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

xy\ zw	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6

11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{10}	m_{11}

Εικόνα2.8: Τοποθέτηση ελαχιστόρων στον χάρτη Καρνώ για τρεις (αριστερά) και για τέσσερις (δεξιά) μεταβλητές

Προσοχή
1. Πρέπει να τοποθετήσουμε '1' σε όλα τα κελιά που αντιστοιχούν σε ελαχιστόρους στους οποίους αληθεύει η συνάρτηση. **2. Προσπαθούμε να βρούμε κατά το δυνατόν τις λιγότερες (αριθμητικά) και μεγαλύτερες ομάδες από γειτονικούς '1'.** 3. Οι ομάδες αποτελούνται από 1, 2, 4, 8, 16 γειτονικά '1'. 4. Γειτονικά θεωρούνται οι θέσεις στο χάρτη που αντιστοιχούν σε ελαχιστόρους που διαφέρουν σε μία και μόνο μεταβλητή. 5. **Δεν** είναι γειτονικές θέσεις «διαγώνιοι» ελαχιστόροι καθώς διαφέρουν σε περισσότερες από μια μεταβλητές. **6.** Ένα κελί με τιμή '1' αποτελεί μόνο του μια ομάδα όταν δεν μπορεί να συνδυασθεί με καμία άλλη θέση. **7. Ένα** κελί με τιμή **'1' μπορεί να συμπεριληφθεί σε περισσότερες από μια ομάδες.** **8. Σημειώνεται ότι ο χάρτης Καρνώ είναι «σφαρικός».**

Με τη μέθοδο που αναπτύχθηκε παραπάνω, σχεδιάζουμε το απλούστερο δυνατό κύκλωμα με πύλες AND και OR για μια συνάρτηση F. Χρησιμοποιώντας τον χάρτη Καρνώ, αλλά για την F' **μπορούμε να σχεδιάσουμε τη συνάρτηση F με πύλες OR, AND** κάνοντας χρήση του θεωρήματος DeMorgan.
Παράδειγμα:
Σχεδιάστε το κύκλωμα που υλοποιεί τη συνάρτηση Boole: $F(x,y,z)$ = X(2,3,4,5) με πύλες OR, AND.

Λύση
Τοποθετούμε στον χάρτη Καρνώτην αντίστροφη συνάρτηση της F.

x/yz	00	01	11	10
0	1	1		
1			1	1

Παρατηρούμε ότι σχηματίζονται δύο ομάδες, οπότε η συνάρτηση εκφράζεται ως εξής:
F'=xy'+xy => F=(xy'+xy)'=(xy')'(xy)'=(x'+y)(x'+y')

2.4.2 Πειραματικό Μέρος

1. Να σχεδιαστεί κύκλωμα που να υλοποιεί τη συνάρτηση
F(x, y, z, w)=Σ(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15) με πύλες
A. AND, OR
B. OR, AND

Ασκήσεις

1. Να σχεδιάσετε το κύκλωμα που υλοποιεί τη συνάρτηση F(x,y,z,w)=Σ(α, β, γ) όπου α, β, γ τα τρία τελευταία ψηφία του μητρώου σας, με πύλες α) AND, OR β) OR, AND.
2. Να σχεδιάσετε το ισοδύναμο του παραπάνω κυκλώματος με πύλες NAND.
3. Να εκφράσετε τους παρακάτω πίνακες Καρνώ.

Πίνακας Α

xy\zw	**00**	**01**	**11**	**10**
00	0	0	1	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

Πίνακας Β

xy\zw	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	0	0	1	1

Πίνακας Γ

x\zw	**00**	**01**	**11**	**10**
0	1	1	0	0
1	1	1	0	0

Πίνακας Δ

xy\zw	**00**	**01**	**11**	**10**
00	0	1	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	0	0

Πίνακας Ε

xy\zw	**00**	**01**	**11**	**10**
00	0	0	1	0
01	0	0	1	0
11	1	1	0	0
10	0	0	0	0

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 2.4

Πίνακας Ζ

x\zw	**00**	**01**	**11**	**10**
0	0	0	0	0
1	1	1	0	0

Πίνακας Η

x\zw	**00**	**01**	**11**	**10**
0	1	1	0	1
1	1	1	0	1

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 2.5

2.5 Υλοποίηση κωδίκων και σχεδίαση με αδιάφορους όρους

2.5.1 Θεωρητικό υπόβαθρο

Κώδικας BCD

Ένας δυαδικός κώδικας που διακρίνει 10 στοιχεία πρέπει να έχει τουλάχιστον τέσσερα bit. Αλλά 4 bit έχουν 16 δυνατούς συνδυασμούς οπότε 6 από αυτούς θα παραμείνουν αχρησιμοποίητοι. Μπορούμε να έχουμε πολλούς διαφορετικούς δυαδικούς κώδικες ανάλογα με την αντιστοίχηση που θα κάνουμε στους 10 διαφορετικούς συνδυασμούς των τεσσάρων μπιτ. Ο κώδικας που χρησιμοποιείται πιο συχνά για την κωδικοποίηση των δεκαδικών ψηφίων είναι η απλή δυαδική αντιστοίχηση που φαίνεται στον παρακάτω πίνακα.

Δεκαδικό ψηφίο	Ψηφίο BCD
0	*0000*
1	0001
2	0010
3	0011
4	0100
5	0101
6	*0110*
7	0111
8	1000
9	1001

Πίνακας 2.2: Ο κώδικας BCD

Οι αριθμοί που κωδικοποιούνται με αυτόν τον τρόπο ονομάζονται δυαδικά κωδικοποιημένοι δεκαδικοί και για συντομία BCD. Υπάρχουν και άλλοι δυνατοί δεκαδικοί κώδικες. Ο Πίνακας 2. δίνει τον τετράμπιτο αυτόν κώδικα για το κάθε δεκαδικό ψηφίο.
Ένας αριθμός με *κ* δεκαδικά ψηφία απαιτεί 4κ μπιτ σε μορφή BCD. Το δεκαδικό 396 παρίσταται σε μορφή BCD με 12 μπιτ ως 0011 1001 0110, με την κάθε ομάδα τεσσάρων μπιτ να παριστά ένα δεκαδικό ψηφίο. Ένας δεκαδικός αριθμός σε μορφή BCD έχει ίδια παράσταση με αυτή του ισοδυνάμου δυαδικού αριθμού του, μόνον όταν είναι μεταξύ του 0 και του 9. Ένας αριθμός BCD μεγαλύτερος από το 10 έχει διαφορετική δυαδική παράσταση απ' ό,τι ο ισοδύναμος δυαδικός αριθμός του, παρόλο που και οι δύο περιέχουν 1 και 0.
Επίσης, οι δυαδικοί συνδυασμοί από το 1010 ως το 1111 δεν χρησιμοποιούνται και δεν έχουν νόημα στον κώδικα BCD. Παρατηρείστε το δεκαδικό 185 και την αντίστοιχη τιμή του σε BCD και δυαδική μορφή:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

Η παράσταση του αριθμού σε BCD έχει 12 bit, ενώ ο ισοδύναμος δυαδικός αριθμός χρειάζεται μόνο 8 bit. Είναι προφανές ότι ένας αριθμός BCD χρειάζεται περισσότερα bit από ότι η ισοδύναμη δυαδική τιμή του. Όμως, υπάρχει ένα πλεονέκτημα στη χρήση των δεκαδικών αριθμών, καθώς τα δεδομένα εισόδου και εξόδου παράγονται από ανθρώπους που χρησιμοποιούν το δεκαδικό σύστημα.

2.5.2 Πειραματικό Μέρος

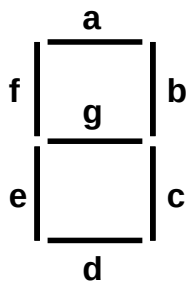
1. Να σχεδιάσετε και υλοποιήσετε κύκλωμα κωδικοποιητή που η λειτουργία του περιγράφεται από τον παρακάτω πίνακα αληθείας.

A	B	C	D	x	y
1	0	0	0	0	0
1	1	0	0	0	1
1	1	1	0	1	0
1	1	1	1	1	1

2. Να σχεδιασθεί το απλούστερο δυνατό κύκλωμα που υλοποιεί τη συνάρτηση F(x, y, z, w)=Σ(0, 1, 4, 10, 14) με συνθήκη αδιαφορίας D(x, y, z, w)=Σ(2, 6, 8)

Άσκηση

1. Ένα δεκαδικό ψηφίο βρίσκεται σε μορφή BCD. Η δεκαδική μορφή του αριθμού αυτού θέλουμε να εμφανιστεί σε ένα 7-segmentdisplay, το οποίο φαίνεται παρακάτω.



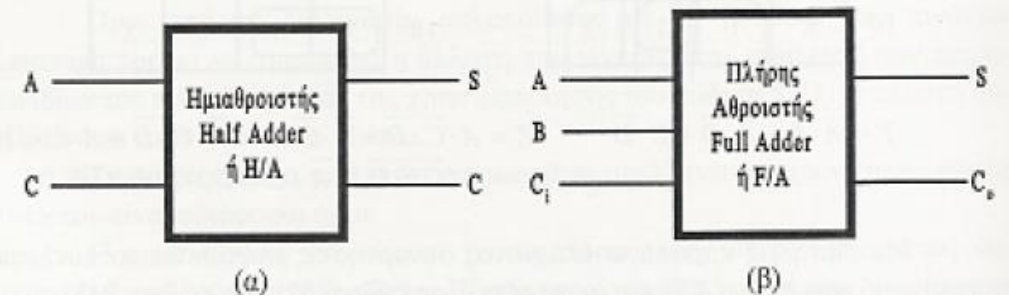
Η μορφή που εμφανίζεται στο display εξαρτάται από το αν κάθε segment (a, b, c, d, e, f, g) έχει τιμή ‘1’ ή όχι. Σχεδιάστε κύκλωμα που δέχεται ως είσοδο τα 4 ψηφία που παριστούν τον δεκαδικό αριθμό σε BCD μορφή και παράγει ως έξοδο τα a, b, c, d, e, f, g. (υπόδειξη συμπληρώστε τον παρακάτω πίνακα και μετά υπολογίστε καθεμιά από τις 7 συναρτήσεις εξόδου με χρήση χάρτη Καρνώ.)

x3	**x2**	**x1**	**x0**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

2.6 Ημιαθροιστής-Πλήρης Αθροιστής

2.6.1 Θεωρητικό Υπόβαθρο

Ο ημιαθροιστής είναι το λογικό συνδυαστικό κύκλωμα που προσθέτει τα δυο δυαδικά ψηφία A και B που εφαρμόζονται στις εισόδους του και δίνει σαν εξόδους το άθροισμά τους και το κρατούμενο. Στην Εικόνα 2.9φαίνεται σε διάγραμμα βαθμίδας ένας ημιαθροιστής όπου A και B είναι οι δύο είσοδοι και S και C οι έξοδοι.



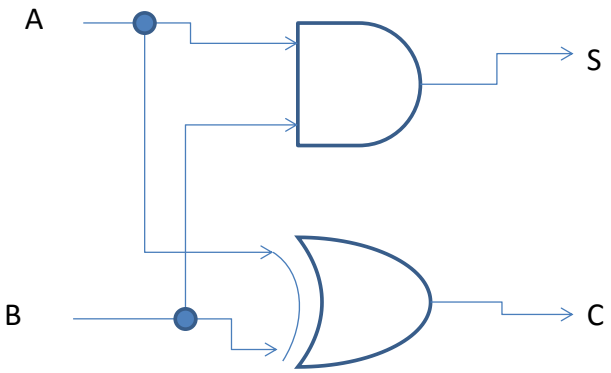
Εικόνα 2.9: διαγράμματα βαθμίδας α) ημιαθροιστή και β) Πλήρους Αθροιστή

<u>Διαδικασία σχεδίασης</u>

Παίρνοντας υπόψη όσα αναφέρθηκαν παραπάνω και τον τρόπο με τον οποίο γίνεται η πρόσθεση δύο δυαδικών αριθμών, συντάσσεται ο πίνακας αληθείας του ημιαθροιστή.

	Είσοδοι		Έξοδοι	
Δεκαδικός	A	B	S	C
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1

Από τον πίνακα αληθείας προκύπτει ότι το άθροισμα S θα είναι :
$S = A \cdot B + A \cdot B = (A \text{ xor } B)$
και το κρατούμενο C θα είναι :
$C = A \cdot B$
Επομένως, η S είναι η έξοδος μιας πύλης XOR της οποίας οι είσοδοι είναι A και B, και η C είναι η έξοδος μιας πύλης AND της οποίας οι είσοδοι είναι A και B. Με βάση τις δύο αυτές λογικές συναρτήσεις συντίθεται το λογικό κύκλωμα που φαίνεται στην Εικόνα 2.10και είναι ένας ημιαθροιστής (H/A).



Εικόνα 2.10: Κύκλωμα ημιαθροιστή

Επέκταση

Στην πράξη σπανίως θα χρησιμοποιηθεί ένα κύκλωμα που θα προσθέτει μόνο δύο δυαδικά ψηφία. Συνήθως το κύκλωμα που θα κάνει πρόσθεση θα πρέπει να προσθέτει δύο ψηφιολέξεις που αποτελούνται από πολλά ψηφία. Το κύκλωμα αυτό θα πρέπει να μπορεί να προσθέτει τα δυο ισοβαρή ψηφία των ψηφιολέξεων μαζί και το κρατούμενο από την προηγούμενη πρόσθεση. Ένα τέτοιο λογικό κύκλωμα που δέχεται στην είσοδό του 2 δυαδικά ψηφία της ίδιας βαρύτητας και το κρατούμενο προηγούμενης τάξης και παράγει το άθροισμα και κρατούμενο της επόμενης τάξης λέγεται πλήρης αθροιστής και φαίνεται στο διάγραμμα βαθμίδας του σχήματος και η διαδικασία σχεδίασης του αναπτύσσεται παρακάτω.

Διαδικασία σχεδίασης

Το ζητούμενο αυτού του προβλήματος είναι να σχεδιαστεί ένα συνδυαστικό κύκλωμα που θα προσθέτει δύο ψηφία A και B και το κρατούμενο C_i από την προηγούμενη πρόσθεση και θα δίνει ένα άθροισμα s και ένα νέο κρατούμενο C_0. Με βάση αυτές τις προδιαγραφές συντάσσεται ο πίνακας αληθείας του πλήρους αθροιστή που δείχνει ο Πίνακας.

	Είσοδοι			Έξοδοι	
Δεκαδικός	A	B	C_i	S	C_0
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Πίνακας 2.3: Πίνακας αληθείας του πλήρους αθροιστή

C_i \ AB	00	01	11	10	C_i \ AB	00	01	11	10
0	0	1	0	1	0	0	0	1	0
1	1	0	1	0	1	0	1	1	1

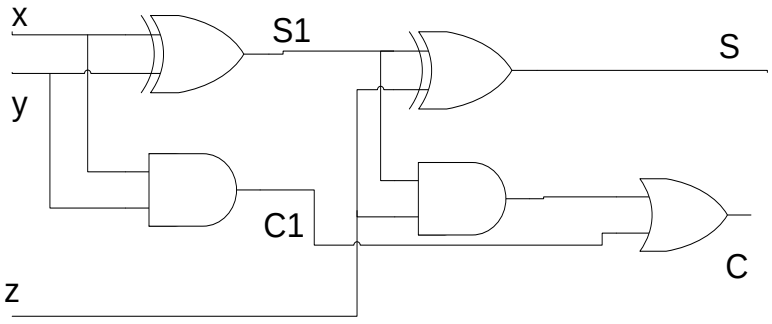
Εικόνα 2.11: Ο χάρτης Καρνώ για τις εξόδους S και C

$$S=A'\cdot B\cdot C_i'+A\cdot B'\cdot C_i'+A'\cdot B'\cdot C_i+A\cdot B\cdot C_i=$$
$$=(A'\cdot B+A\cdot B')\cdot C_i'+(A'\cdot B'+A\cdot B)\cdot C_i=$$
$$=(A'\cdot B+A\cdot B')\cdot C_i'+(A'\cdot B+A\cdot B')'\cdot C_i=$$
$$=(A\oplus B)\cdot C_i'+(A\oplus B)'\cdot C_i=$$
$$=(A\oplus B)\oplus C_i$$

Και κρατούμενο θα δίνεται από :

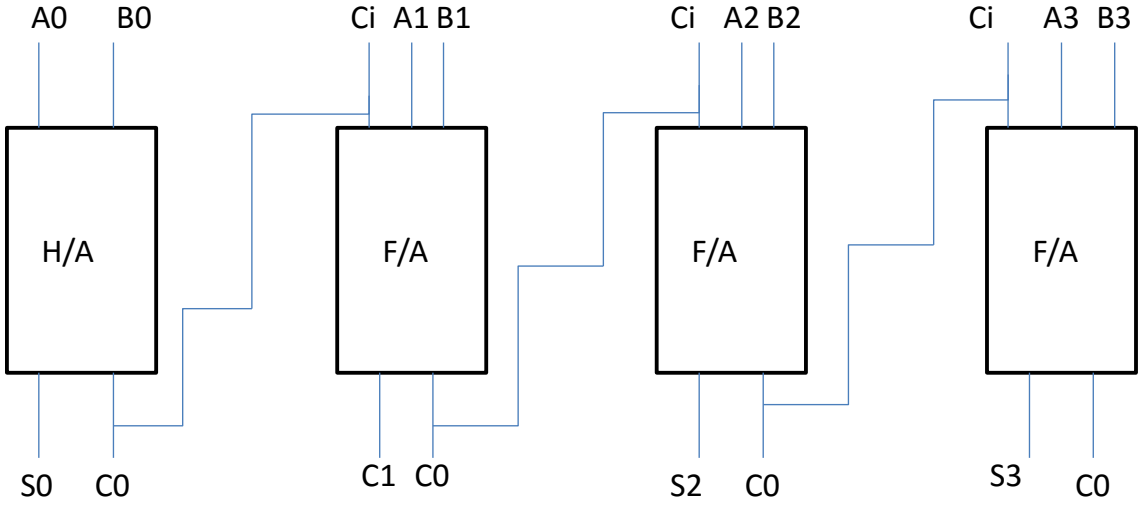
$$C_0=B\cdot C_i+A\cdot C_i+A\cdot B$$

Το κύκλωμα που υλοποιεί τον πλήρη αθροιστή φαίνεται παρακάτω:



Εικόνα 2.12: Πλήρης αθροιστής

Προκειμένου να προστεθούν δύο δυαδικές ψηφιολέξεις κ ψηφίων χρησιμοποιούνται τα παραπάνω λογικά κυκλώματα συνδεδεμένα σε μορφή τέτοια που να προσθέτουν τις δυο ψηφιολέξεις σειριακά ή παράλληλα. Η Εικόνα 2.13 δείχνει έναν παράλληλο πλήρη αθροιστή που προσθέτει δύο ψηφιολέξεις των τεσσάρων ψηφίων. Ο αθροιστής αυτός αποτελείται από τρεις πλήρεις αθροιστές και έναν ημιαθροιστή.



Εικόνα 2.13: Πλήρης Αθροιστής δύο ψηφιολέξεων των 4 bit

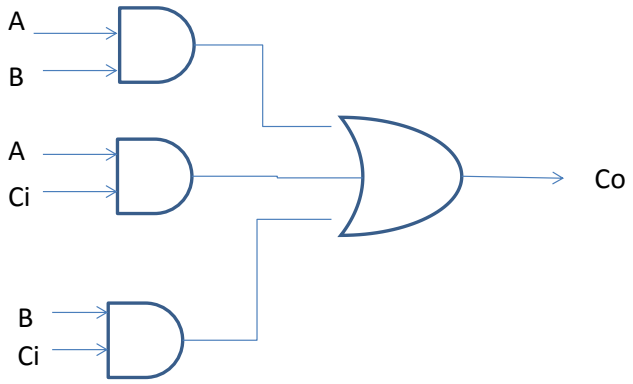
Στην πρόσθεση των λιγότερο σημαντικών ψηφίων χρησιμοποιείται ημιαθροιστής αντί για πλήρη αθροιστή γιατί δεν υπάρχει κρατούμενο από προηγούμενη πρόσθεση.
Προκειμένου να γίνει η πρόσθεση, το κρατούμενο πρέπει να περάσει από όλους τους αθροιστές πράγμα που προκαλεί μεγάλη καθυστέρηση. Αυτό είναι ένα πρόβλημα που στην ουσία αποδυναμώνει το πλεονέκτημα του παράλληλου αθροιστή, σε σχέση με τον αθροιστή σειράς, που είναι η μεγάλη ταχύτητα. Το πρόβλημα αυτό μπορεί να αποφευχθεί αν σχεδιαστεί ένα κύκλωμα πρόβλεψης του κρατούμενου και τροφοδότησής του στην είσοδο κρατουμένου του κάθε επιμέρους αθροιστή.
Είναι φανερό ότι ο ημιαθροιστής θα δώσει κρατούμενο μόνο όταν και οι δύο είσοδοι είναι σε λογικό ‘1’. Αυτή η λογική υλοποιείται με μια πύλη AND στις εισόδους της οποίας οδηγούνται τα δύο ψηφία Ao και Bo και η έξοδος της, που θα είναι λογικό ‘1’μόνο όταν και οι δύο είσοδοι είναι λογικό ‘1’, συνδέεται στην είσοδο κρατουμένου του επόμενου αθροιστή. Για τις επόμενες βαθμίδες το πρόβλημα είναι λίγο πιο πολύπλοκο. Εδώ για να σχεδιαστεί το κύκλωμα πρόβλεψης κρατουμένου θα χρησιμοποιηθεί ο πίνακας αληθείας του πλήρους αθροιστή. Από τον πίνακα αυτό προκύπτει το κρατούμενο θα δίνεται από την συνάρτηση αθροίσματος ελάχιστων όρων:
$C_0=\Sigma(3,5,6,7)$
Με τη βοήθεια του πίνακα Karnaugh τριών μεταβλητών, μπορούμε να εκφράσουμε την ελάχιστη μορφή της F, F_{min}.

C_i \ AB	00	01	11	10

0	***0***	***0***	***1***	***0***
1	***0***	***1***	***1***	***1***

$F_{min} = \boldsymbol{B \cdot C_i + A \cdot C_i + A \cdot B}$

Το κύκλωμα που υλοποιεί αυτή την συνάρτηση είναι το κύκλωμα πρόβλεψης κρατούμενου που φαίνεται στο σχήμα:



Εικόνα2.14: Κύκλωμα πρόβλεψης κρατουμένου

2.6.2 Πειραματικό Μέρος

1. Να σχεδιαστεί ένα κύκλωμα που να προσθέτει δυο μονοψήφιους αριθμούς (ημιαθροιστής). Να επανασχεδιαστεί χρησιμοποιώντας πύλη XOR.
2. Να σχεδιαστεί ένα κύκλωμα που να προσθέτει τρεις μονοψήφιους αριθμούς (πλήρης αθροιστής).
3. Να σχεδιαστεί ένα κύκλωμα που να προσθέτει δυο διψήφιους αριθμούς (δυαδικός αθροιστής 2-bits) με χρήση των παραπάνω κυκλωμάτων.

Υπενθύμιση

Ο δυαδικός αθροιστής 2-bits μπορεί να σχεδιαστεί με 2 πλήρεις αθροιστές.
Με παρόμοιο σχεδιασμό μπορούμε να υλοποιήσουμε δυαδικούς αθροιστές των 3-bits, 4-bits, ….n-bits.
Ο παράλληλος δυαδικός αθροιστής 4-bits περιέχεται στο ολοκληρωμένο 7483.

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 2.6

1. Πόσοι πλήρεις αθροιστές απαιτούνται για να φτιάξουμε έναν αθροιστή δύο 6-ψήφιων δυαδικών αριθμών;

2. Στο κύκλωμα του αθροιστή δύο εξαψήφιων δυαδικών αριθμών,
A. οι δύο έξοδοι κάθε πλήρους αθροιστή συνδέονται στην είσοδο του πλήρους αθροιστή επόμενης βαθμίδας
B. η έξοδος c (κρατουμένου) συνδέεται στην είσοδο του πλήρους αθροιστή επόμενης βαθμίδας
C. η έξοδος S (αθροίσματος) συνδέεται στην είσοδο του πλήρους αθροιστή επόμενης βαθμίδας
D. οι δύο έξοδοι κάθε πλήρους αθροιστή αποτελούν εξόδους του κυκλώματος του αθροιστή εξαψήφιων δυαδικών αριθμών

2.7 Αθροιστές – Αφαιρέτες - Συγκριτές

2.7.1 Θεωρητικό Υπόβαθρο

Για να εκτελέσουμε την αφαίρεση δύο δυαδικών αριθμών Α, Β πρέπει να πάρουμε το συμπλήρωμα ως προς 2 του αφαιρετέου και να το προσθέσουμε στο μειωτέο. Το συμπλήρωμα ως προς 2 προκύπτει αν πάρουμε το συμπλήρωμα ως προς 1 και προσθέσουμε 1.

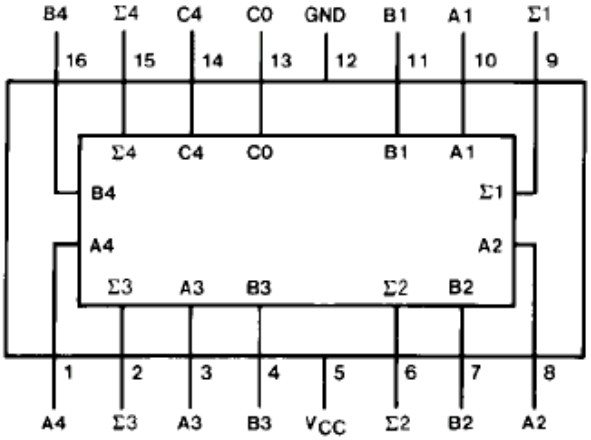
2.7.2 Πειραματικό Μέρος

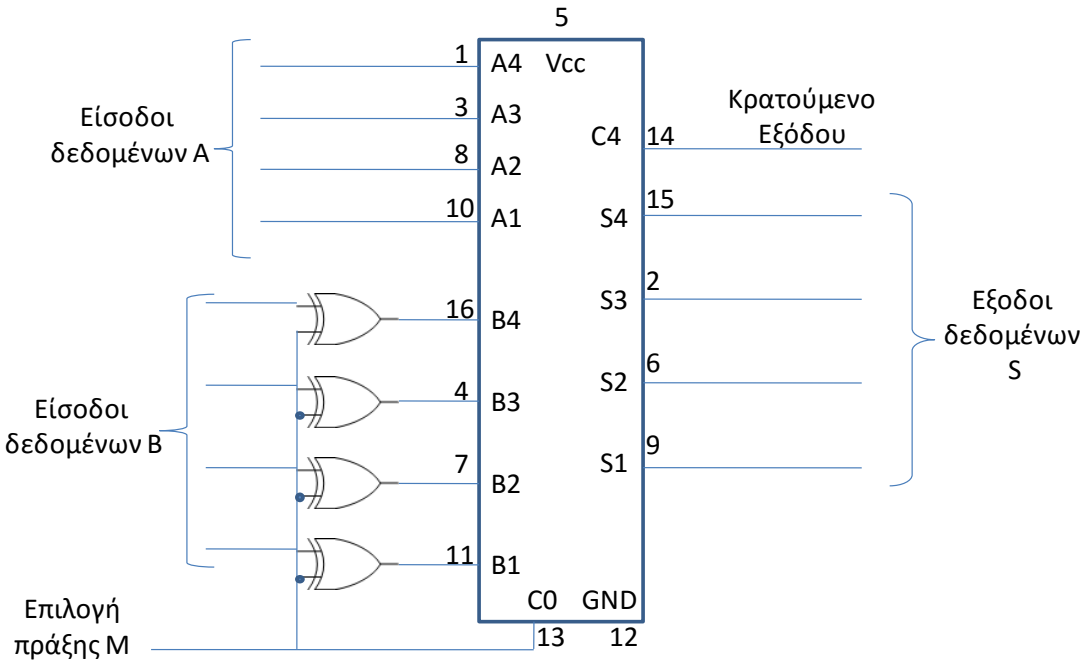
1. Να υλοποιήσετε έναν τετραψήφιο δυαδικό αθροιστή-αφαιρέτη, δηλαδή κύκλωμα που για M=0 εκτελεί πρόσθεση ενώ για M=1 εκτελεί αφαίρεση.

Υπόδειξη

Χρησιμοποιείστε το ολοκληρωμένο κύκλωμα του τετράμπιτου δυαδικού αθροιστή 7483 συνδέοντας στις εισόδους B1 ως B4 το αποτέλεσμα της αποκλειστικής διάξευξης των ψηφίων του αφαιρετέου με το M. Υπενθυμίζεται ότι (xXOR 1)=x΄.

Pinout 7483





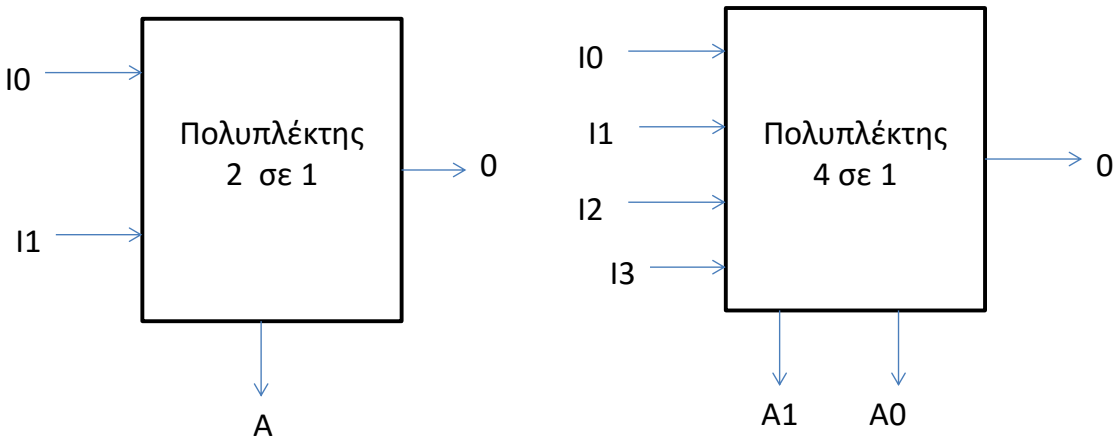
2. Υπολογίστε θεωρητικά και πειραματικά τα αποτελέσματα στις παρακάτω περιπτώσεις:

A	**B**	**A+B**	**A- B**	**σύγκριση**
3	5			
6	8			
5	2			
7	3			
6	0			
5	5			
4	1			
7	3			
5	4			

2.8 Σχεδίαση με πολυπλέκτες

2.8.1 Θεωρητικό Υπόβαθρο

Ο ψηφιακός πολυπλέκτης είναι ένα συνδυαστικό λογικό κύκλωμα με πολλές εισόδους και μια έξοδο και κάθε φορά συνδέει μια μόνο είσοδο στην έξοδο του. Για να μπορεί να γίνεται επιλογή της εισόδου που θα συνδέεται κάθε χρονική στιγμή στην έξοδο, στις γραμμές εισόδων δίνονται συγκεκριμένες και μοναδικές για κάθε γραμμή διευθύνσεις δια μέσου κάποιων γραμμών διευθύνσεων. Στο σχήμα φαίνεται σε διαγράμματα βαθμίδας ο ψηφιακός πολυπλέκτης δυο και τεσσάρων εισόδων I (input) σε μια έξοδο 0 (output). Οι είσοδοι Α είναι είσοδοι-διευθύνσεις. Ανάλογα με τον δυαδικό αριθμό που εφαρμόζεται στις εισόδους αυτές η αντίστοιχη είσοδος I εμφανίζεται στην έξοδο 0. Επομένως, αν ο πολυπλέκτης έχει N εισόδους σε μία έξοδοι τότε χρειάζονται ν γραμμές διεύθυνσης, όπου $N=2^{v}$.



Εικόνα 2.15: Διαγράμματα βαθμίδας πολυπλέκτη (α) 2 εισόδων και (β) 4 εισόδων

Όταν δίνεται ένας πίνακας αληθείας και ζητείται να σχεδιαστεί το συνδυαστικό κύκλωμα που τον υλοποιεί, μπορεί να χρησιμοποιηθεί ένας πολυπλέκτης.

Μεθοδολογία σχεδίασης

1. Για να σχεδιάσω κύκλωμα κ εισόδων με πολυπλέκτη θα χρησιμοποιήσω έναν πολυπλέκτη που να έχει κ-1 εισόδους διευθύνσεων (και επομένως $2^{κ-1}$ εισόδους).
2. Καταγράφω τον πίνακα αληθείας του κυκλώματος.
3. Επιλέγω τις κ-1 εισόδους ως εισόδους διευθύνσεων. Αυτό σημαίνει ότι οι κ-1 είσοδοι θα συνδεθούν στις εισόδους διεθύνσεων του πολυπλέκτη.
4. Το επόμενο βήμα είναι να προσδιορίσω τις τιμές που πρέπει να πάρουν οι $2^{κ-1}$ είσοδοι του πολυπλέκτη I_j. και για κάθε συνδυασμό αυτών (που εμφανίζεται σε 2 γραμμές του πίνακα αληθείας) συγκρίνω την έξοδο του κυκλώματος F με την άλλη είσοδο και τις σταθερές τιμές ‘0’ ή ‘1’. Η παρατήρηση στην οποία βασίζομαι είναι ότι όταν ο συνδυασμός των κ-1 εισόδων που εξετάζω τεθεί στις εισόδους διευθύνσεων, τότε η αντίστοιχη είσοδος I_j πρέπει να πάρει τέτοια τιμή ώστε να υλοποιεί την έξοδο του κυκλώματος F. Έτσι προσδιορίζω τις τιμές που πρέπει να πάρουν οι είσοδοι διευθύνσεων.

2.8.2 Πειραματικό Μέρος

1. Να υλοποιήσετε την παρακάτω συνάρτηση χρησιμοποιώντας τον κατάλληλο πολυπλέκτη.
$F(x,y,z,w) = Σ(0,1,3,4,8,9,15)$.

Υπόδειξη

- Θα χρησιμοποιήσω πολυπλέκτη με 3 εισόδους διευθύνσεων αφού το κύκλωμά μου έχει 4 εισόδους.
- Επιλέγω οι είσοδοι διευθύνσεων να είναι οι είσοδοι x, y, z, οπότε για να προσδιορίσω τι πρέπει να συνδεθεί στις εισόδους I_0-I_7, συγκρίνω την F με την είσοδο w, w', και τις τιμές ‘1’ και ‘0’.

x	y	z	w	F	
0	0	0	0	1	$F=I_0=1$
0	0	0	1	1	
0	0	1	0	0	$F=I_1=w$
0	0	1	1	1	
0	1	0	0	1	$F=I_2=w'$
0	1	0	1	0	
0	1	1	0	0	$F=I_3=0$
0	1	1	1	0	

1	0	0	0	1	$F=I_4=1$
1	0	0	1	1	
1	0	1	0	0	$F=I_5=0$
1	0	1	1	0	
1	1	0	0	0	$F=I_6=0$
1	1	0	1	0	
1	1	1	0	0	$F=I_7=w$
1	1	1	1	1	

Υπόδειξη

Χρησιμοποιείται πολυπλέκτης 8 εισόδων (74151).

Άσκηση

Να σχεδιασθεί κύκλωμα που υλοποιεί την $F(x,y,z,w)=\Sigma(1,3,5,7, 10,11,14,15)$

1. Με πύλες AND, OR, NOT
2. Με πύλες NAND μόνο
3. με πολυπλέκτη

2.9 Βιβλιογραφία

- *ΨΗΦΙΑΚΗ ΣΧΕΔΙΑΣΗ MORRISMANO,* ISBN13: 9789607182661, Εκδόσεις Παπασωτηρίου. Κεφάλαια 2, 3 και 4.
- Ασημάκης Ν., Ψηφιακά Ηλεκτρονικά, Εκδόσεις GUTENBERG, 2008.

Κεφάλαιο 3ο Ακολουθιακά Κυκλώματα με ολοκληρωμένα TTL

3.1 Εισαγωγή στα FLIP – FLOP

3.1.1 Θεωρητικό Υπόβαθρο

Τα σύγχρονα ακολουθιακά κυκλώματα με τα οποία θα ασχοληθούμε στο εργαστήριο των Ψηφιακών συστημάτων 2, αφορούν ψηφιακά συστήματα τα οποία περιέχουν συνήθως συνδυαστικά κυκλώματα. Επιπλέον αυτά τα κυκλώματα, περιέχουν στοιχεία μνήμης, τα οποία και κάνουν όλο το σύστημα να είναι ακολουθιακό. Μια σχηματική διάταξη ενός τέτοιου ακολουθιακού κυκλώματος φαίνεται στην παρακάτω εικόνα (Εικόνα).

Εικόνα 3.1: Σχηματική διάταξη ακολουθιακού κυκλώματος

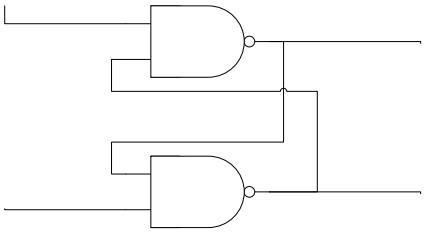
Παρατηρώντας την παραπάνω διάταξη, εξάγουμε το συμπέρασμα ότι αποτελείται από, ένα συνδυαστικό κύκλωμα συνδεδεμένο με στοιχεία μνήμης σε ένα σχηματισμό βρόγχου ανάδρασης. Με το όρο **στοιχεία μνήμης**, εννοούμε συσκευές που μπορούν να αποθηκεύουν δυαδικές πληροφορίες μέσα τους. Η αποθηκευμένη πληροφορία ονομάζεται κατάσταση του κυκλώματος (συστήματος) Ένα ακολουθιακό κύκλωμα δέχεται πληροφορίες από τις εξωτερικές του εισόδους. Οι είσοδοι, σε συνδυασμό με την παρούσα κατάσταση των στοιχείων μνήμης, καθορίζουν τις τιμές των εξόδων. Επίσης αυτές καθορίζουν το πώς θα αλλάξει η κατάσταση των στοιχείων μνήμης.
Συμπεράσματα: σε ένα σύγχρονο ακολουθιακό κύκλωμα, **η έξοδος είναι συνάρτηση όχι μόνο των εισόδων του, αλλά και της παρούσας κατάστασης των στοιχείων μνήμης του**. Η επόμενη κατάσταση αυτών των στοιχείων μνήμης είναι κι αυτή συνάρτηση τόσο των εισόδων όσο και της παρούσας κατάστασης. Καταλήγουμε δηλαδή ότι, για την περιγραφή ενός των ακολουθιακών κυκλωμάτων δεν φτάνει να κοιτάξει κανείς μόνο τις παρούσες τιμές των εισόδων και εξόδων, αλλά πρέπει να μελετήσουμε μια ολόκληρη χρονική ακολουθία εισόδων, εξόδων και καταστάσεων.
Ένα σύγχρονο ακολουθιακό κύκλωμα πρέπει εξ΄ ορισμού, να χρησιμοποιεί σήματα τα οποία επηρεάζουν τα στοιχεία μνήμης του **σε διακριτές στιγμές του χρόνου** μόνο. Πρακτικά αυτό επιτυγχάνεται μέσω μιας «γεννήτριας κύριου – ρολογιού», η οποία τροφοδοτεί το σύστημα με μια περιοδική σειρά «παλμών ρολογιού». Αυτά τα κυκλώματα που χρησιμοποιούν παλμούς συγχρονισμού που εφαρμόζονται σε στοιχεία μνήμης ονομάζονται «ακολουθιακά κυκλώματα με ρολόι ».
Τα στοιχεία μνήμης που χρησιμοποιούνται στα ακολουθιακά κυκλώματα με ρολόι λέγονται FLIP – FLOP. Πρόκειται δηλαδή για δυαδικά κύτταρα που μπορούν να αποθηκεύσουν ένα bit πληροφορίες. Τα FLIP – FLOP έχουν συνήθως δύο εξόδους, μία για την τιμή του bit που είναι αποθηκευμένο μέσα τους και μια για το συμπλήρωμά της. Δυαδικές πληροφορίες μπορούν να καταχωρηθούν στο flip – flop με διάφορους τρόπους, και έτσι έχουμε διάφορους τύπους flip – flop.
Ειδοποιός διαφορά ανάμεσα στους διάφορους τύπους flip – flop είναι ο αριθμός των εισόδων που έχουν και ο τρόπος με το οποίο αυτές οι είσοδοι επηρεάζουν την δυαδική τους κατάσταση. Οι συνηθέστεροι τύποι flip – flop με τους οποίους θα ασχοληθούμε και στο εργαστήριο είναι οι παρακάτω:

- J – K flip – flop
- D flip – flop
- T flip – flop

Είναι προφανές ότι οι παραπάνω τύποι των flip – flop υπάρχουν στο εργαστήριο με την μορφή ολοκληρωμένου (chip), και ονομασία ανάλογη με τον τύπο.

3.1.2 Πειραματικό Μέρος

1. Υπολογίστε θεωρητικά και πειραματικά τον πίνακα αληθείας του παρακάτω κυκλώματος



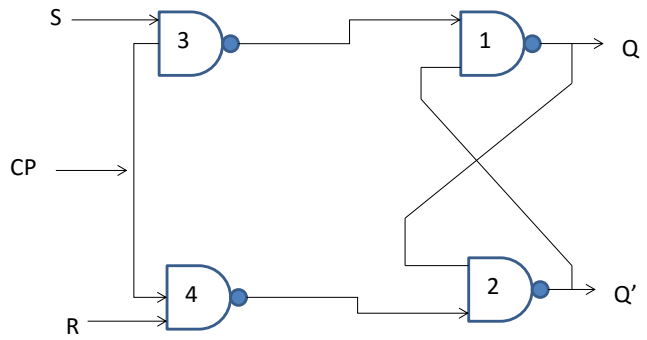
Οι πύλες NAND βρίσκονται στο ολοκληρωμένο 7400.

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.1

Άσκηση για το σπίτι

Υπολογίστε τον πίνακα αληθείας του παρακάτω κυκλώματος.



Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.2

3.2 ΚΥΚΛΩΜΑΤΑ J-K , D, T, FLIP FLOP

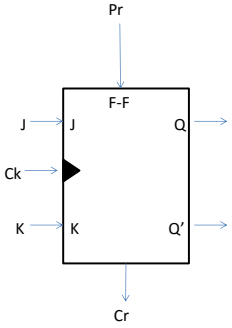
3.2.1 Θεωρητικό Υπόβαθρο

Τα flip-flop τύπου J – K είναι μια πιο εξελιγμένη μορφή του τύπου R – S, εξελιγμένη κατά το ότι η απροσδιόριστη κατάσταση που έχει τον τύπο R – S προσδιορίζεται στον τύπο J – K. Οι είσοδοι J και K συμπεριφέρονται σαν τις εισόδους R – S – η J θέτει (set) το flipflop και το K μηδενίζει (clear). Εάν όμως διεγείρουμε και την J και την K ταυτόχρονα , τότε το flipflop αλλάζει κατάσταση και πάει στην συμπληρωματική αυτής στην οποία ήταν – αν ήταν Q=1 γίνεται Q=0 και αντίστροφα.

Χαρακτηριστικός πίνακας J-Kflip-flop

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Το γραφικό σύμβολο του J-KFlip-Flop



Η λειτουργία τωνpreset και clear

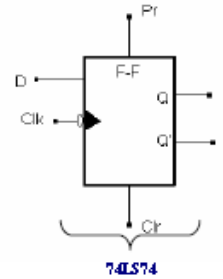
Το γραφικό σύμβολο του J-KFlip-Flop φαίνεται παραπάνω και περιέχεται στα ολοκληρωμένα 74LS76.
Τα σήματα Preset και Clear είναι αυτά που καθορίζουν τις εισόδους σύμφωνα με τον παραπάνω πίνακα. Οι είσοδοι αυτές λέγονται και **άμεσες είσοδοι** καθώς επενεργούν στην έξοδο ασύγχρονα με το ρολόι. Το μεν preset ή directset θέτει την έξοδο (στο '1') το δε clear ή directreset το επαναφέρει στο μηδέν.

Αλληλεπίδραση
Με link στο 74LS76 να δίνουμε το datasheet του ολοκληρωμένουD FLIP – FLOP
Ένας τρόπος εξάλειψης της ανεπιθύμητης συμπεριφοράς στην απροσδιόριστη κατάσταση είναι να εξασφαλιστεί ότι ποτέ δεν θα γίνουν και οι δύο είσοδοι ίσες με '1'. Αυτό συμβαίνει στο Dflip-flop το οποίο έχει μία είσοδο D και το ρολόι.

Χαρακτηριστικός πίνακας Dflip-flop

D	Q(t+1)
0	0
1	1

Το γραφικό σύμβολο του DFlip-Flop



Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.3

3.2.2 Πειραματικό Μέρος

1. Να επαληθευτεί πειραματικά ο χαρακτηριστικός πίνακας του FLIP FLOP τύπου J-K με παλμό ρολογιού με χρήση του ολοκληρωμένου κυκλώματος 7476.
2. Να επαληθευτεί πειραματικά ο χαρακτηριστικός πίνακας του FLIP FLOP τύπου D με παλμό ρολογιού με χρήση του ολοκληρωμένου κυκλώματος 7474.

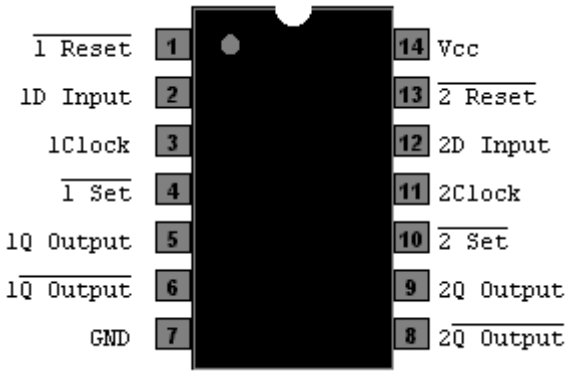
Άσκηση 1

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.3

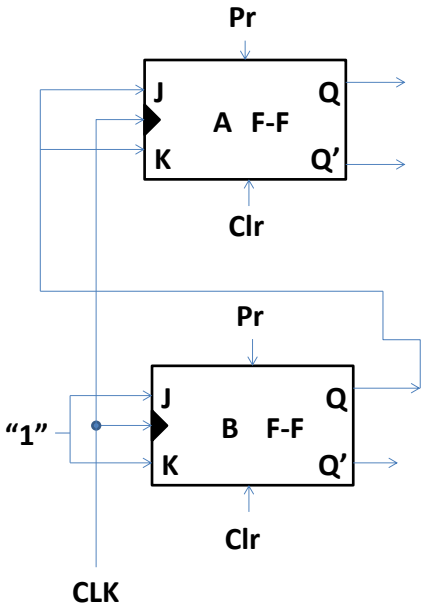
Έχοντας υπόψιν το pinout του ολοκληρωμένου κυκλώματος 74LS74 που φαίνεται παρακάτω, προσδιορίστε τις κατάλληλες τιμές που πρέπει να τεθούν στα pin προκειμένου να λειτουργεί το flipflop 1:

a) 7
b) 14
c) 4
d) 1
e) Ποια λογική τιμή θα πάρουμε στο pin 5 εάν το pin 2 συνδεθεί στο ground?
f) Ποια λογική τιμή θα πάρουμε στο pin 6 εάν το pin 2 συνδέσουμε 5V?



Άσκηση 2

Υπολογίστε τον πίνακα καταστάσεων του παρακάτω κυκλώματος. Υπόδειξη: οι έξοδοι Q_A, Q_B εξαρτώνται από την παρούσα κατάσταση και τις τιμές των εισόδων Ja, Ka, Jb, Kb. Συμπληρώστε τον παρακάτω πίνακα.



Εικόνα 3.2: Κύκλωμα πειράματος με βασικά Flip-Flop

$Q_A(t)$	$Q_B(t)$	J_A	K_A	J_B	K_B	$Q_A(t+1)$	$Q_B(t+1)$
0	0						
0	1						
1	0						
1	1						

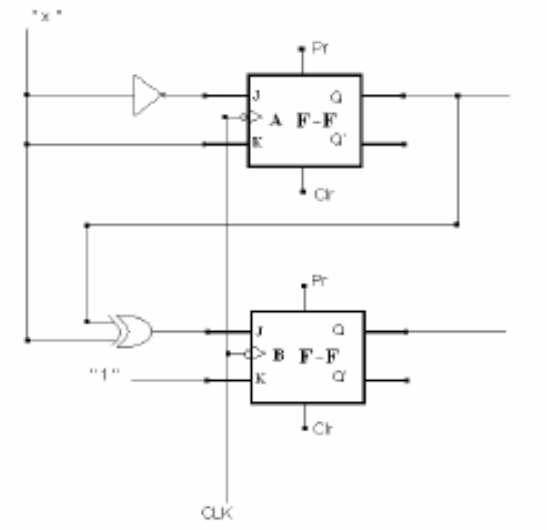
Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.4

3.3 Ανάλυση σύγχρονου ακολουθιακού κυκλώματος

3.3.1 Θεωρητικό Υπόβαθρο

Θα εξετάσουμε τη **μεθοδολογία** που ακολουθούμε για την ανάλυση ενός σύγχρονου ακολουθιακού κυκλώματος με βάση το ενδεικτικό κύκλωμα που ακολουθεί:
Προσδιορίστε το διάγραμμα ροής του παρακάτω κυκλώματος. Για να φτάσετε στο στόχο σας, συμπληρώστε τον πίνακα καταστάσεων ο οποίος περιλαμβάνει την παρούσα και επόμενη κατάσταση των εξόδων των flip-flop, τις εισόδους του κυκλώματος (για παράδειγμα x, y, z) και τις εισόδους των flip-flop. Συνήθως οι τιμές των εισόδων των flip-flop εξαρτώνται από την παρούσα κατάσταση και τις τιμές των εισόδων.



Εικόνα 3.3: Κύκλωμα παραδείγματος ανάλυσης σύγχρονων συνδυαστικών κυκλωμάτων

ΒΗΜΑ 1.

Βρίσκουμε τις εξισώσεις των σημάτων εισόδων των flip-flop (εδώ JA , KA , JB , KB) από το κύκλωμα.

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.5

ΒΗΜΑ 2.

Συμπληρώνουμε τον πίνακα καταστάσεων κατάλληλα και υπολογίζουμε την επόμενη κατάσταση με βάση τις παρούσες καταστάσεις και τις διεγέρσεις. Πιο αναλυτικά για κάθε πιθανό συνδυασμό τιμών των εισόδων και της παρούσας κατάστασης, υπολογίζουμε τις τιμές των εισόδων των flip-flop και εν συνεχεία την επόμενη κατάσταση των flip-flop.

Παρούσα κατάσταση			**Επόμενη Κατάσταση**		**Διέγερση**			
x	$Q_A(t)$	$Q_B(t)$	$Q_A(t+1)$	$Q_B(t+1)$	J_A	K_A	J_B	K_B
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

Πίνακας 3.4: Πίνακας καταστάσεων του κυκλώματος που φαίνεται στην Εικόνα 3.3.3

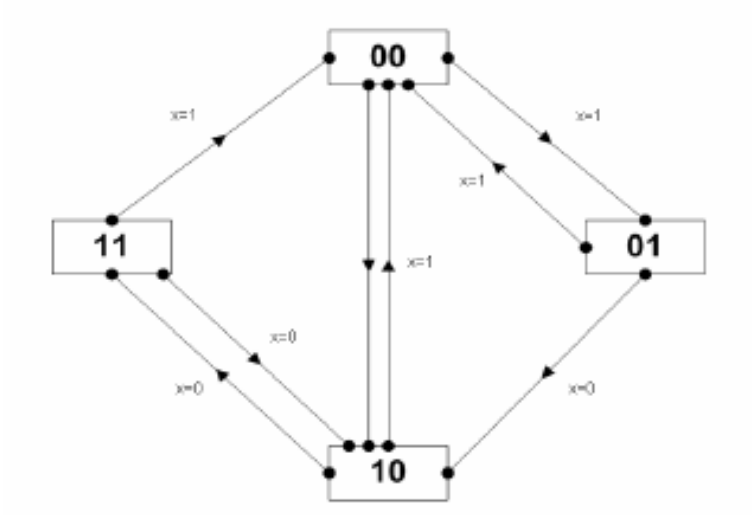
Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 3.6

Αφού συμπληρώστε τις στήλες JA, KA, JB, KB , ελέγξτε την ορθότητα των εξόδων QA(t+1), QB(t+1)

ΒΗΜΑ 3.

Τέλος το διάγραμμα ροής φαίνεται παρακάτω:

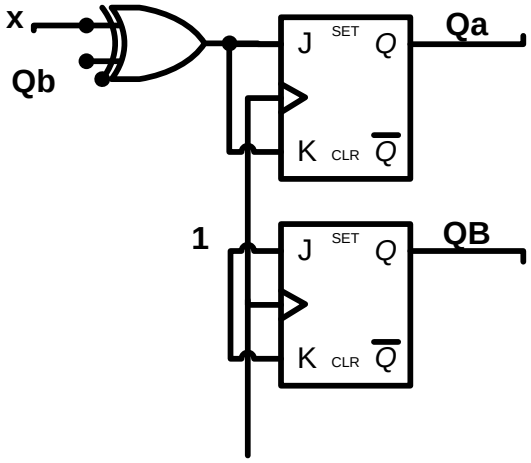


3.3.2 Πειραματικό Μέρος

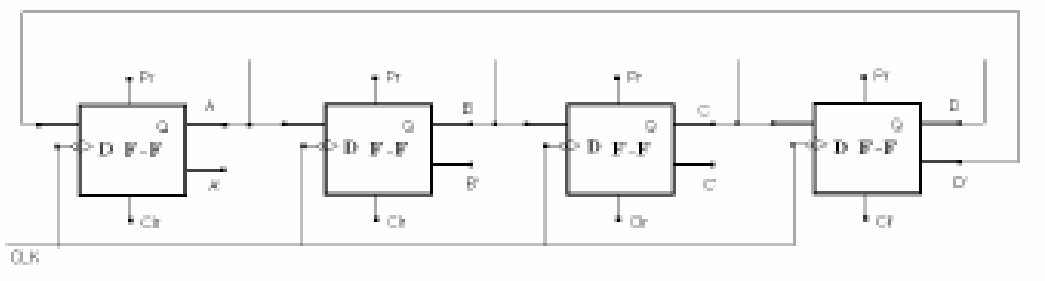
Να επαληθευτεί πειραματικά το παραπάνω διάγραμμα ροής.

Άσκηση

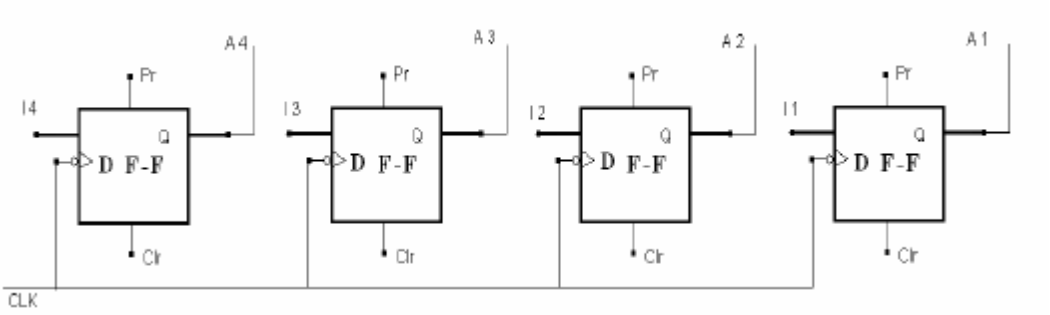
1. Υπολογίστε το διάγραμμα ροής του παρακάτω κυκλώματος.



2. Υπολογίστε το διάγραμμα ροής του παρακάτω κυκλώματος.



3. Υπολογίστε το διάγραμμα ροής και τον πίνακα καταστάσεων του παρακάτω κυκλώματος. Ποια η χρησιμότητα του κυκλώματος αυτού?



3.4 Σχεδίαση σύγχρονου ακολουθιακού κυκλώματος – σχεδίαση μετρητή

3.4.1 Θεωρητικό Υπόβαθρο

Ένα ακολουθιακό κύκλωμα που περνάει από μια προδιαγραμμένη ακολουθία καταστάσεων, όταν του εφαρμόσουμε παλμούς στην είσοδο, λέγεται «μετρητής» (counter). Οι παλμοί εισόδου, που τους λέμε «παλμούς μέτρησης» μπορεί να είναι παλμοί ρολογιού ή μπορεί να προέρχονται από μία εξωτερική πηγή και μπορεί να έρχονται σε κανονικά ή ακανόνιστα χρονικά διαστήματα. Σε ένα μετρητή η ακολουθία των καταστάσεων μπορεί να είναι η δυαδική σειρά μέτρησης ή μια οποιαδήποτε άλλη σειρά. Μετρητές βρίσκουμε σχεδόν σε όλα τα ψηφιακά μηχανήματα. Τους χρησιμοποιούμε για να μετράμε πόσες φορές συμβαίνει κάποιο γεγονός ή για την δημιουργία ακολουθιών χρονισμού για τον έλεγχο των λειτουργιών ενός ψηφιακού συστήματος.

Από τις διάφορες ακολουθίες μέτρησης που μπορεί να έχει ένας μετρητής, η απλούστερη και η ευρύτερα διαδεδομένη είναι η απλή δυαδική σειρά μέτρησης. Ένας τέτοιος μετρητής λέγεται **«δυαδικός μετρητής»**. Ένας «δυαδικός μετρητής» των n-bits αποτελείται από n flip-flops και μπορεί να μετράει στο δυαδικό από το 0 εώς το 2^n-1.

Μεθοδολογία σχεδίασης

Στη συνέχεια θα διατυπώσουμε τη μεθοδολογία σχεδίασης και θα χρησιμοποιήσουμε για παράδειγμα τη σχεδίαση ενός δυαδικού μετρητή των 2-bits ο οποίος εκτελεί φθίνουσα κυκλική μέτρηση για κύκλωμα με J-Kflip-flop.

ΒΗΜΑ 1.

Προσδιορίζουμε το πλήθος των flip-flop που θα χρησιμοποιήσουμε. Στο παράδειγμά μας θα χρησιμοποιήσουμε 2 flip-flop καθώς θέλουμε να σχεδιάσουμε μετρητή των 2-bit.

ΒΗΜΑ 2.

Με βάση τη ζητούμενη λειτουργία, συμπληρώνουμε τις στήλες του πίνακα καταστάσεων που περιγράφουν την παρούσα και επόμενη κατάσταση για όλες τις πιθανές τιμές της παρούσας κατάστασης. Σε περίπτωση που για κάποια τιμή της παρούσας κατάστασης δεν προσδιορίζει η εκφώνηση την επιθυμητή τιμή της επόμενης κατάστασης, συμπληρώνουμε X, δηλαδή αδιάφορους όρους.

Για το παράδειγμά μας συμπληρώνουμε τις 4 πρώτες στήλες, όπως φαίνεται παρακάτω.

Πίνακας Καταστάσεων							
Παρούσα κατάσταση		Επόμενη κατάσταση		Διέγερση			
$Q_A(t)$	$Q_B(t)$	$Q_A(t+1)$	$Q_B(t+1)$	JA	KA	JB	KB
0	0	1	1	1	X	1	X
0	1	0	0	0	X	X	1
1	0	0	1	X	1	1	X
1	1	1	0	X	0	X	1

ΒΗΜΑ 3.

Με βάση τον πίνακα διέγερσης των flip-flop που χρησιμοποιούμε, συμπληρώνουμε τις στήλες που αντιστοιχούν στις εισόδους των flip-flop.

Οι πίνακες διέγερσης για τα J-K kai D flip-flop φαίνονται στον ακόλουθο πίνακα.

Q(t+1)	D
0	0
1	1

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	X
1	0	x	1
1	1	x	0

ΒΗΜΑ 4.

Βρίσκουμε την εξίσωση που εκφράζει τις εισόδους των flip-flop σαν συνάρτηση της παρούσας κατάστασης και των εξωτερικών εισόδων του κυκλώματος (αν υπάρχουν). Επισημαίνεται ότι για να βρούμε την απλούστερη δυνατή εξίσωση, χρησιμοποιούμε χάρτη Καρνώ.

Στο παράδειγμά μας:

JA	B	
A	0	1
0	1	0
1	X	X

JA=B’

KA	B	
A	0	1
0	1	0
1	X	X

KA=B’

JB	B	
A	0	1
0	1	X
1	1	X

JB=1

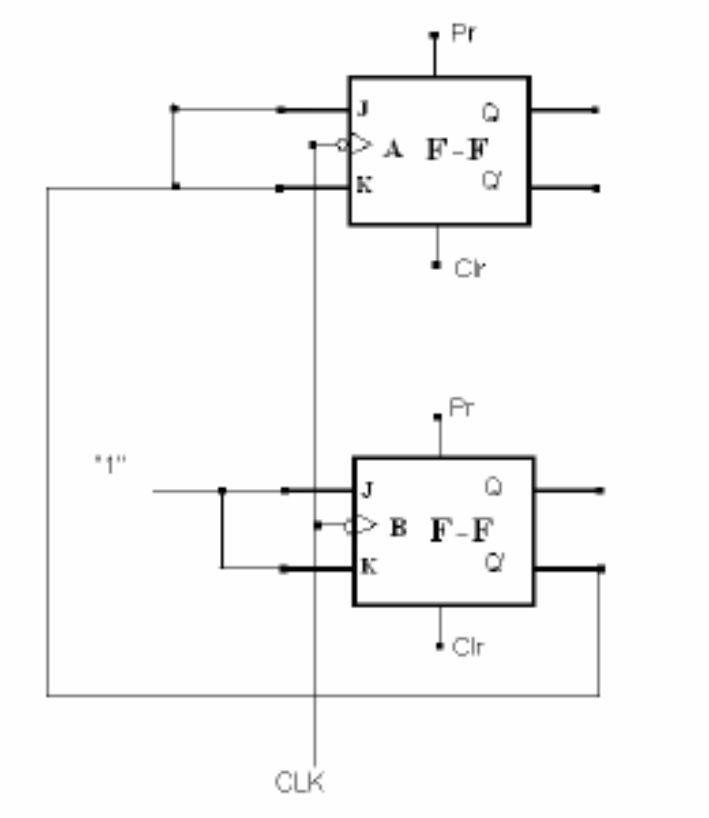
KB	B	
A	0	1
0	1	X
1	1	X

KB=1

<u>ΒΗΜΑ 5</u>

Σχεδιάζουμε το κύκλωμα.

Το κύκλωμα του παραδείγματός μας φαίνεται παρακάτω.



3.4.2 Πειραματικό Μέρος

Να επαληθευτεί πειραματικά ότι το παραπάνω κύκλωμα εκτελεί την επιθυμητή λειτουργία.

Άσκηση

1. Να σχεδιασθεί μετρητής 3-bit ο οποίος εκτελεί φθίνουσα μέτρηση
   a. με J-Kflip-flop
   b. με Dflip-flop.
2. Να σχεδιασθεί μετρητής BCD ο οποίος εκτελεί αύξουσα μέτρηση.

3.5 Αυτοδιόρθωση και σχεδίαση μετρητή με είσοδο

3.5.1 Θεωρητικό Υπόβαθρο

Οι απαιτήσεις σχεδίασης ενός μετρητή περιγράφουν μια ακολουθία καταστάσεων η οποία αλλιώς ονομάζεται κύκλος καταστάσεων. Όταν οι απαιτήσεις δεν περιγράφουν την επόμενη κατάσταση για κάθε πιθανή τιμή της παρούσας κατάστασης, προκύπτει το ζήτημα των μεταβάσεων για τις τιμές κατάστασης εκτός κύκλου (που δεν έχουν προδιαγεγραμμένη επόμενη κατάσταση).

Για παράδειγμα, αν μας ζητηθεί να σχεδιασθεί ένας μετρητής που εκτελεί τις μεταβάσεις 0-3-5-7-0, οι απαιτήσεις σχεδίασης δεν ορίζουν ποια πρέπει να είναι η επόμενη κατάσταση όταν το κύκλωμα βρεθεί (πιθανότατα με τη χρήση των ασύγχρονων σημάτων presetkaiclear) στην κατάσταση 2.

Αν το κύκλωμα βρεθεί σε κατάσταση εκτός κύκλου και η επόμενη κατάσταση είναι κατάσταση του κύκλου, τότε λέμε ότι το κύκλωμα αυτοδιορθώνεται. Αν το κύκλωμα για (έστω και μια) κατάσταση εκτός κύκλου, δεν οδηγείται σε κατάσταση του κύκλου, λέμε ότι το κύκλωμα δεν αυτοδιορθώνεται.

Σε περίπτωση που μας ζητηθεί να σχεδιάσουμε ένα κύκλωμα και δεν ορίζεται τι γίνεται στις καταστάσεις εκτός κύκλου, οφείλουμε να ελέγξουμε αν το κύκλωμα αυτοδιορθώνεται, αφού πρώτα σχεδιάσουμε το κύκλωμα, όπως θα γίνει στο ακόλουθο παράδειγμα.

Παράδειγμα

Να σχεδιαστεί ένα σύγχρονο κύκλωμα με χρήση J-K flipflop , που να μετράει κυκλικά προς τα πάνω τους αριθμούς: 0-3-5-7-0

Λύση

ΒΗΜΑ 1.

Στο παράδειγμά μας θα χρησιμοποιήσουμε 3 flip-flop για να μπορούν να κωδικοποιηθούν οι αριθμοί 0 ως και 7.

ΒΗΜΑ 2 και 3.

Συμπληρώνουμε τον πίνακα καταστάσεων, όπως φαίνεται παρακάτω.

Παρούσα κατάσταση			Επόμενη κατάσταση			Διέγερση					
$Q_A(t)$	**$Q_B(t)$**	**$Q_C(t)$**	**$Q_A(t+1)$**	**$Q_B(t+1)$**	**$Q_C(t+1)$**	**J_A**	**K_A**	**J_B**	**K_B**	**J_C**	**K_C**
0	0	0	0	1	1	0	X	1	X	1	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	0	X	X	X	X	X	X	X	X	X
1	0	1	1	1	1	X	0	1	X	X	0
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	0	0	0	X	1	X	1	X	1

ΒΗΜΑ 4.

Στο παράδειγμά μας:

JA	QB	QC		
QA	00	01	11	10
0	0	X	1	X
1	X	X	X	X

JA=QB

KA	QB	QC		
QA	00	01	11	10
0	X	X	1	X
1	X	0	X	X

KA=QB

JB	QB	QC		
QA	00	01	11	10
0	1	X	X	X
1	X	1	X	X

JB=1

KB	QB	QC		
QA	00	01	11	10
0	X	X	1	X
1	X	X	1	X

KB=1

JC	QB	QC		
QA	00	01	11	10
0	1	X	X	X
1	X	X	X	X

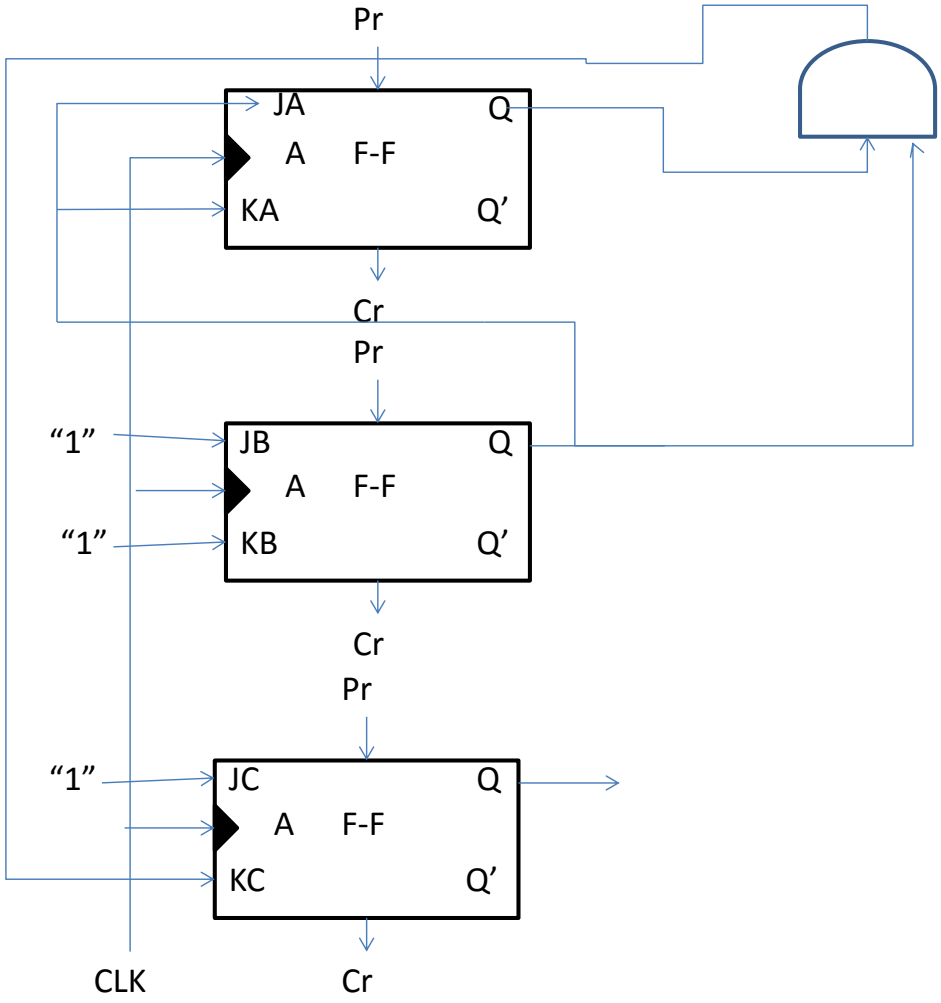
JC=1

KC	QB	QC		
QA	00	01	11	10
0	X	X	0	X
1	X	0	1	X

KC=QA*QB

ΒΗΜΑ 5

Σχεδιάζουμε το κύκλωμα.



Βήμα 6: έλεγχος αυτοδιόρθωσης

Συμπληρώνω τις γραμμές του πίνακα καταστάσεων για τις οποίες υπήρχαν αδιάφοροι όροι δηλαδή για τις τιμές της παρούσας κατάστασης που δεν οριζόταν η επόμενη. Με βάση το κύκλωμα είμαστε σε θέση να συμπληρώσουμε τις τιμές των εισόδων των flip-flop και στη συνέχεια την επόμενη κατάσταση.

$Q_A(t)$	$Q_B(t)$	$Q_C(t)$	$Q_A(t+1)$	$Q_B(t+1)$	$Q_C(t+1)$	J_A	K_A	J_B	K_B	J_C	K_C
0	0	1	0	1	1	0	0	1	1	1	0
0	1	0	1	0	1	1	1	1	1	1	0
1	0	0	1	1	1	0	0	1	1	1	0
1	1	0	0	0	1	1	1	1	1	1	1

Παρατηρούμε ότι ενώ για τις τιμές 1, 2 και 4 της παρούσας κατάστασης, η επόμενη κατάσταση είναι 3, 5, και 7 αντίστοιχα (δηλαδή τιμές του κύκλου), αυτό δεν ισχύει για την κατάσταση 6. Όταν το κύκλωμα βρεθεί στην

κατάσταση 6, η επόμενη κατάσταση είναι η κατάσταση 1, η οποία είναι εκτός κύκλου. Άρα το κύκλωμα δεν αυτοδιορθώνεται σε ένα παλμό ρολογιού. Στον επόμενο όμως παλμό ρολογιού το κύκλωμαθα μεταβεί στην κατάσταση 3, η οποία είναι κατάσταση του κύκλου. Δηλαδή η αλληλουχία είναι 6-1-3. Στην περίπτωση αυτή λέμε ότι το κύκλωμα αυτοδιορθώνεται μετά από 2 παλμούς ρολογιού.

3.5.2 Πειραματικό Μέρος

1. Να σχεδιαστεί και να υλοποιηθεί σύγχρονο κύκλωμα 2- bit με D FlipFlop που να μετρά μικρούς ή μεγάλους αριθμούς ανάλογα με την τιμή κατάλληλου εξωτερικού σήματος ελέγχου. Για παράδειγμα:
    - Για x=0 να μετρά τους αριθμούς : 00 – 01
    - Για x=1 να μετρά τους αριθμούς :10 – 11

Ασκήσεις

1. Να σχεδιαστεί και να υλοποιηθεί σύγχρονο κύκλωμα 3- bit με D FlipFlop που να μετρά άρτιους και περιττούς αριθμούς ανάλογα με την τιμή κατάλληλου εξωτερικού σήματος ελέγχου. Για παράδειγμα:
    - Για x=0 να μετρά τους άρτιους αριθμούς
    - Για x=1 να μετρά τους περιττούς αριθμούς
2. Να σχεδιαστεί και να υλοποιηθεί σύγχρονο κύκλωμα 3- bit με J-K FlipFlop που να μετρά άρτιους και περιττούς αριθμούς ανάλογα με την τιμή κατάλληλου εξωτερικού σήματος ελέγχου.

3.6 Βιβλιογραφία

- Ασημάκης Ν., Ψηφιακά Ηλεκτρονικά, Εκδόσεις GUTENBERG, 2008.
- *«ΨΗΦΙΑΚΗ ΣΧΕΔΙΑΣΗ» MORRISMANO,* ISBN13: 9789607182661, Εκδόσεις Παπασωτηρίου, Κεφάλαια 5, 6, και 7

Κεφάλαιο 4ο Σχεδίαση Κυκλωμάτων με χρήση της γλώσσας VHDL

4.1 Εισαγωγή στη VHDL

4.1.1 Θεωρητικό υπόβαθρο

Η VHDL είναι μια γλώσσα που χρησιμοποιείται για την **περιγραφή και μοντελοποίηση** ψηφιακών κυκλωμάτων. Αρχικοποιήθηκε από το DoD (Department of Defense-USA) στις αρχές του 1980.
Πεδία Εφαρμογής της VHDL

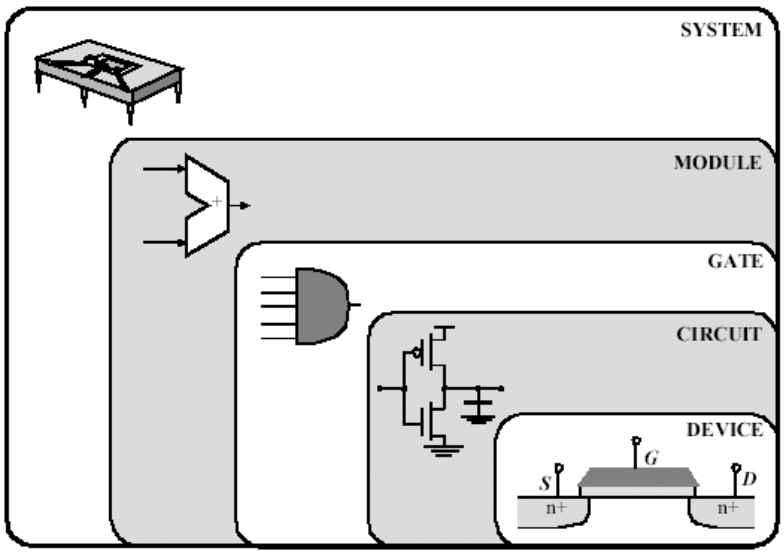
- Εξομοίωση ορθής λειτουργίας (Simulation)
- Σύνθεση ψηφιακών κυκλωμάτων (Synthesis)
- Επιβεβαίωση ορθού σχεδιασμού (DesignVerification)
- Μοντέλα προδιαγραφών (Specification Models)

Τα πλεονεκτήματα της γλώσσας VHDL:

- Παγκόσμιο πρότυπο (IEEE 1076-1987, 1076-1993)
- Υποστήριξη από πληθώρα αναπτυξιακών εμπορικών εργαλείων σχεδιασμού (CAD tools)
- Εύκολη μεταφορά κυκλωματικών περιγραφών σε διαφορετικά αναπτυξιακά περιβάλλοντα
- Δυνατότητα περιγραφής κυκλώματος / συστήματος σε διαφορετικά ιεραρχικά επίπεδα (από επίπεδο πύλης μέχρι επίπεδο συστήματος)
- Υποστήριξη εναλλακτικών σχεδιαστικών μεθοδολογιών (Top-down, Bottom-up, Mixed)
- Ιεραρχικήσχεδίαση (Block Diagrams, Components)
- Επαναχρησιμοποίηση σχεδιασθέντων υπομονάδων (reusable components)
- Χρήση βιβλιοθηκών με σχεδιασθέντα κυκλώματα
- Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος
- Περιγραφή κυκλώματος/συστήματος ανεξάρτητα από την τεχνολογία υλοποίησης
- Επαναχρησιμοποίηση υπάρχουσας κυκλωματικής περιγραφής σε διαφορετικές τεχνολογίες (FPGA xilinx, altera, ASICs)
- Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος
- Υποστήριξη συντρεχουσών και ακολουθιακών δομών (concurrent and sequential constructions). Οι περισσότερες γλώσσες (π.χ. C) υποστηρίζουν μόνο ακολουθιακές δομές. Οι συντρέχουσες δομές είναι απαραίτητες για την περιγραφή της λειτουργίας του υλικού.
- Εύκολη διαχείριση λαθών και επιβεβαίωση ορθής λειτουργίας (Simulation, Error Management, Design Verification)

Στη VHDL διακρίνονται τα εξής επίπεδα σχεδίασης:

- **Functional (system) level**(αρχιτεκτονική)
- **Behaviorallevel** (ανάθεση δομών και πόρων)
- **RTL (Structural) level**(επιλογή τεχνολογίας)
- **Logic (gate) level** + electrical specification
- **Electrical level** + layout requirements
- **Layout level**



Εικόνα 4.1: Τα επίπεδα σχεδίασης

Στο επίπεδο συστήματος προβλέπεται:

- Περιγραφή των προδιαγραφών του συστήματος
- Δεν απαιτείται πληροφορία χρονισμών
- Δεν απαιτείται ακριβής καθορισμός της αρχιτεκτονικής του κυκλώματος / συστήματος
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας

Στο επίπεδο συμπεριφοράς προβλέπεται:

- Αναλυτικότερη περιγραφή της συμπεριφοράς / λειτουργίας του κυκλώματος
- Καθορισμός των απαιτούμενων αλγορίθμων για την ικανοποίηση των προδιαγραφών του συστήματος
- Εμπεριέχει πληροφορίες χρονισμού
- Μη αναλυτικός καθορισμός της αρχιτεκτονικής του κυκλώματος (καταχωρητές, μνήμες, συνδυαστικά κυκλώματα κ.λ.π.)
- Ένα μοντέλο περιγραφής σε επίπεδο συμπεριφοράς αποτελείται από τα λειτουργικά στοιχεία και τη διασύνδεση αυτών
- Κάθε λειτουργικό στοιχείο μπορεί να εμπεριέχει περισσότερα του ενός στοιχεία και πληροφορία χρονισμού

Στα επίπεδα καταχωρητή και πύλης προβλέπεται:

- Επίπεδοκαταχωρητή (Register Transfer Level-RTL)
- Περιγραφή του κυκλώματος με χρήση συνδυαστικών κυκλωμάτων, καταχωρητών, μνημών, σύγχρονων και ασύγχρονων μηχανών πεπερασμένων καταστάσεων
- ΕπίπεδοΠύλης (Gate/Logic-Level)
- Περιγραφή του κυκλώματος σε επίπεδο πύλης
- Χρήση λογικών εξισώσεων (Boolean functions)
- Χρησιμοποιείται κυρίως για το σχεδιασμό βασικών συνδυαστικών κυκλωμάτων (αθροιστές, πολ/στές κ.λ.π.)
- Υψηλοί χρόνοι σύνθεσης και εξομοίωσης

Στη VHDL, οι υποστηριζόμενοι τρόποι περιγραφής κυκλωμάτων είναι:

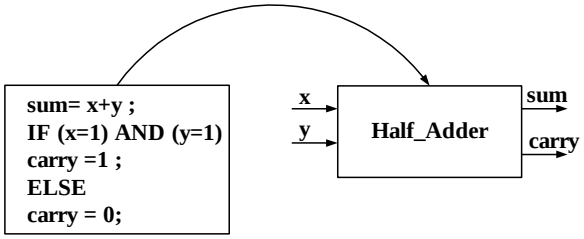
- Συμπεριφοράς (Behavioral VHDL)
- Ροής Δεδομένων (Dataflow VHDL)
- Δομής (Structural VHDL)

Και οι τρεις παραπάνω μέθοδοι περιγραφής μπορούν να χρησιμοποιηθούν σε κάθε επίπεδο της ροής σχεδιασμού. Καθώς μετακινούμαστε από το επίπεδο συμπεριφοράς στο επίπεδο δομής έχουμε:

- Αναλυτικότερη κυκλωματική περιγραφή
- Καλύτερο έλεγχο της σύνθεσης του κυκλώματος
- Μεγαλύτερος κώδικας

- Υψηλότεροι χρόνοι εξομοίωσης

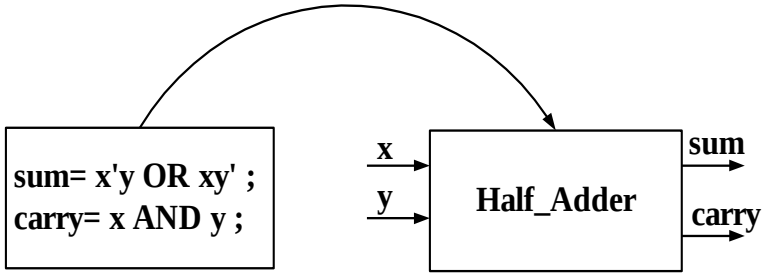
Behavioral VHDL: Χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο. Υποστηρίζει:
Αλγοριθμική περιγραφή της λειτουργίας του κυκλώματος
Δεν περιγράφεται αναλυτικά η κυκλωματική δομή του κυκλώματος
Δεν απαιτούνται αναλυτικές λογικές εξισώσεις
Δυνατότητα εξομοίωσης για επιβεβαίωση ορθής λειτουργίας και κατανόησης (καταρχήν) της λειτουργίας του κυκλώματος
Επαλήθευση μέσω back-annotated πληροφορίας, επιτρέπει επιπλέον να γίνει επιβεβαίωση ορθής λειτουργίας με στοιχεία της τεχνολογίας ολοκλήρωσης



Εικόνα 4.2: Η περιγραφή ενός half_adder με behaviouralVHDL

Data – Flow VHDL: Σε αυτό τον τρόπο περιγραφής έχουμε:

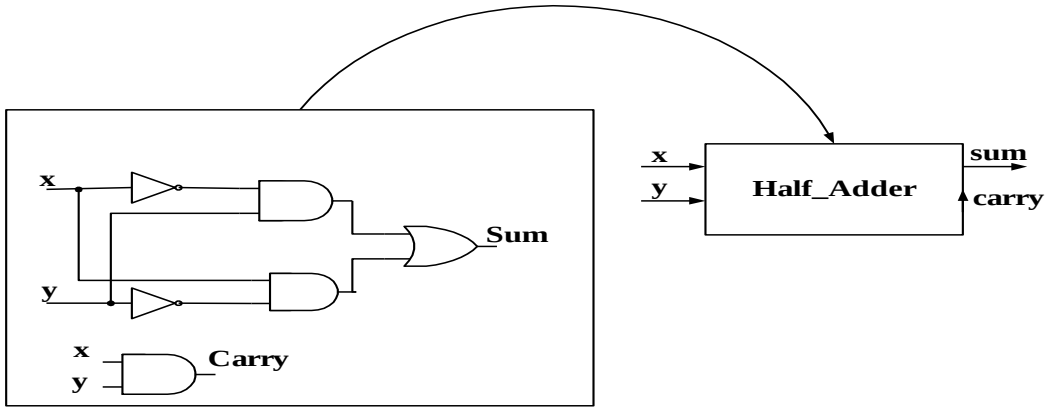
- Αναλυτικότερη περιγραφή της λειτουργίας του κυκλώματος
- Χρήση λογικών εξισώσεων για την μοντελοποίηση της ροής δεδομένων
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας



Εικόνα 4.3: Η περιγραφή ενός half_adder με DataFlowVHDL

Structural VHDL: Χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος.

- Οι δομικές μονάδες ποικίλουν από απλές πύλες μέχρι σύνθετα κυκλώματα
- Προϋποθέτει την ύπαρξη βιβλιοθήκης από σχεδιασθέντα δομικά στοιχεία και την δυνατότητα χρήσης αυτών



Εικόνα 4.4: Η περιγραφή ενός half_adder με StructuralVHDL

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 4.1

Επιλέξτε τη σωστή απάντηση:

1. Η VHDL είναι γλώσσα
   A. προγραμματισμού όπως η C και η Java
   B. περιγραφής ιστοσελίδων
   C. περιγραφής ψηφιακών κυκλωμάτων
   D. αντικειμενοστραφούς προγραμματισμού

Επιλέξτε τη σωστή απάντηση:

2. Η VHDL ΔΕΝ χρησιμοποιείται για
   A. Μοντέλα προδιαγραφών
   B. Σύνθεση ψηφιακών κυκλωμάτων
   C. Επιβεβαίωση ορθού σχεδιασμού
   D. Προγραμματισμό διαδικτυακών εφαρμογών

3. Αντιστοιχίστε τα στοιχεία των δύο στηλών

1. Επαναχρησιμοποίηση σχεδιασθέντων υπομονάδων	A. Επαναχρησιμοποίηση υπάρχουσας κυκλωματικής περιγραφής σε διαφορετικές τεχνολογίες
2. Υποστήριξη συντρεχουσών δομών	B. απαραίτητες για την περιγραφή της λειτουργίας του υλικού
3. Περιγραφή κυκλώματος/συστήματος ανεξάρτητα από την τεχνολογία υλοποίησης	C. Υποστήριξη από πληθώρα αναπτυξιακών εμπορικών εργαλείων σχεδιασμού
4. Παγκόσμιο πρότυπο	D. Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος

4. Βάλτε τα παρακάτω βήματα ανάπτυξης ενός κυκλώματος στη σωστή σειρά
   A. Σύνθεση
   B. Διατύπωση προδιαγραφών
   C. Εξαγωγή αρχείου netlist για προγραμματισμό του ολοκληρωμένου
   D. Ανάπτυξη κώδικα VHDL

5. Αντιστοιχίστε τα στοιχεία των δύο στηλών

1. Ο Behavioral τρόπος περιγραφής στη VHDL	A. Χρησιμοποιεί λογικές εξισώσεις για την μοντελοποίηση της ροής δεδομένων
2. Ο τρόπος περιγραφής dataflow (ροής δεδομένων) στη VHDL	B. Χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο
3. Ο structural τρόπος περιγραφής (περιγραφή δομής) στη VHDL	C. Χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος
4. Η περιγραφή σε επίπεδο συστήματος	D. Παρέχει δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας

4.2 Σχεδίαση απλών συνδυαστικών κυκλωμάτων

4.2.1 Θεωρητικό υπόβαθρο

Για να περιγράψουμε ένα κύκλωμα, φτιάχνουμε ένα αρχείο (στο οποίο δίνουμε ένα όνομα με κατάληξη .vhd). Το αρχείο αυτό οργανώνεται συνήθως σε τρία τμήματα:

1. Το τμήμα δήλωσης των βιβλιοθηκών (library).
2. Το τμήμα δήλωσης της εξωτερικής μορφής του κυκλώματος - οντότητας (entity) όπου ορίζονται το όνομά του και οι διαπεφές του (σήματα εισόδων και εξόδων) του κυκλώματος
3. η περιγραφή της αρχιτεκτονικής της οντότητας (architecture) όπου περιγράφεται η λειτουργία και συμπεριφορά του

Η περιγραφή της **οντότητας** περιλαμβάνει το όνομα αυτής και τα σήματα εισόδου και εξόδου. Η γενικημορφη της δηλωσης μιας οντοτητας είναι:

Περιοχήδήλωσηςοντότητας **(entity)**	
Σύνταξη	**Παράδειγμα**
ENTITYentity_name**IS** **PORT** ([**SIGNAL**] signal_name {,signal,name}:[mode] type_name {; **SIGNAL**] signal_name {,signal_name¨}: [mode] type_name}); **END** entity_name	**ENTITY** example1 **IS** **port** (x1,x2,x3: **IN BIT**; f :**OUT BIT**); **END** example1;

Στον παραπάνω πίνακα οι τονισμένες λέξεις αποτελούν δεσμευμένες λέξεις της γλώσσας.
Η αρχιτεκτονική (architecture) παρέχει τις λεπτομέρειες του κυκλώματος. Η περιγραφή της **αρχιτεκτονικής** μπορεί να γίνει με τους εξής τρόπους:

- behavioral ή μοντέλο συμπεριφοράς το οποίο είναι πιιο κοντά στην ανθρώπινη λογική
- structural/gatelevel, ή δομικό που είναι πιο κοντά στο hardware

Αποτελείται από 2 κύρια μέρη:

- την περιοχή δήλωσης των σημάτων η οποία εμφανίζεται πριν τη λέξη κλειδί begin και
- το σώμα της αρχιτεκτονικής (architecture body)

Για τα σήματα εισόδου – εξόδου υποστηρίζονται οι ακόλουθες καταστάσεις:

- **in** : Σήμα εισόδου – Ανάγνωση σήματος
- **out** : Σήμα εξόδου – Εγγραφή σήματος
- **inout** : Σήμα εισόδου/εξόδου – Ανάγνωση και εγγραφή σήματος
  - Σήμα διπλής κατεύθυνσης(bi-directional)
- **buffer** : Διάβασμα, επανεγγραφή και αποθήκευση τιμής
  - Είναι πάντοτε σήμα εξόδου και όχι διπλής κατεύθυνσης
  - Επιτρέπεται μόνο μία ανάθεση εντός της αρχιτεκτονικής

Οι δεσμευμένες λέξεις της γλώσσας φαίνονται στον ακόλουθο πίνακα:

abs	entity	nor	select
access	exit	not	severity
after	file	null	shared
alias	for	of	signal
all	function		sla
and	generate	open	sll
architecture	generic	or	sra
array	guarded	others	srl
assert	if	out	subtype

attribute	impure	package	then
begin	in	port	to
block	inertial	postponed	transport
body	inout	procedure	type
buffer	is	process	unaffected
bus	label	pure	units
case	library	range	until
component	linkage	record	use
configuration	literal	register	variable
constant	loop	reject	wait
disconnect	map	rem	when
downto	mod	report	while
else	nand	return	with
elsif	new	rol	xnor
end	next	ror	xor

Πίνακας 4.1: Δεσμευμένες λέξεις της γλώσσας VHDL

Η γενική μορφή μιας αρχιτεκτονικής είναι:

Περιοχή δήλωσης αρχιτεκτονικής **(architecture)**	
Σύνταξη	**Παράδειγμα**
ARCHITECTUREarchitecture_name**OF**entity_name**IS** [**SIGNAL** declarations] [**CONSTANT** declarations] [**TYPE** declarations] [**ATTRIBUTE** specifications] **BEGIN** {**COMPONENT** instantiation statement ;} {**CONCURENT ASSINGMENT** statement ;} {**PROCESS** statement ;} {**GENERATE** statement ; } **END** [architecture_name] ;	**ARCHITECTURE**logicFunc**OF** example1 **IS** **BEGIN** f <= (x1 AND x2) OR (NOT x2 AND x3); **END**logicFunc

Επαλήθευση ορθής λειτουργίας κυκλώματος

Για να επαληθεύσουμε τη λειτουργία ενός κυκλώματος χρησιμοποιούμε ένα αρχείο δοκιμών που ονομάζεται test-bench. (Το αρχείο δοκιμών συνήθως ονομάζεται όπως και το αρχείο που δηλώνει το κύκλωμα προσθέτοντας την ένδειξη _TB στο τέλος. Παράδειγμα: comb1_TB.vhd).

Το αρχείο δοκιμών δίνει τιμή στις εισόδους του κυκλώματος (όπως κάνατε στον πάγκο με τους διακόπτες) προκειμένου με την προσομοίωση να ελεγχθεί η ορθή λειτουργία του κυκλώματος (όπως στον πάγκο με τα led). Η γενική μορφή ενός αρχείου δοκιμών είναι:

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

library work;
use work.all;
```

```
entity example1_tb is
end example1_tb;
```

```
architecture example1_tb_a of example1_tb is
```

```
  component example1 is
     port (
              x1,x2,x3: IN  BIT;
                 f             :OUT BIT);
                     );

  end component;

   signal x1,x2,x3, f: std_logic;

begin

example1_inst: example1
     port map (
x1,x2,x3, f );
```

```
stimulus_proc : process is
begin
x1<='0';
x2<='0';
x3<='0';
wait for 12ns;

x1<='0';
x2<='0';
x3<='1';
wait for 2ns;

x1<='0';
x2<='1';
x3<='0';
wait for 3ns;

x1<='0';
x2<='1';
x3<='1';
wait for 7ns;

x1<='1';
x2<='0';
x3<='0';
wait for 2ns;
  wait;

end process stimulus_proc;
end architecture example1_tb_a;
```

4.2.2 Πειραματικό μέρος

Το παρακάτω πρόγραμμα περιγράφει μια πύλη AND δύο εισόδων:

```
library IEEE;
  use IEEE.std_logic_1164.all;
  USE ieee.std_logic_arith.all ;
  USE ieee.std_logic_unsigned.all ;
  use ieee.std_logic_textio.all;   -- in order to use hread( )
library work;
use work.all;
```

```
entity AND_gate is
      port (
            x: in std_logic;
```

```
            y: in std_logic;
            z: out std_logic
                  );
end AND_gate ;
```

```
architecture rtl of AND_gate is

begin
process (x, y)
  begin
    z<=x AND y;
endprocess;

endrtl ;
```

Τα προγράμματα της VHDL αποτελούνται από 3 τμήματα:

- Το τμήμα δήλωσης των βιβλιοθηκών (library).
- Το τμήμα δήλωσης του διεπαφών (εισόδων και εξόδων) του κυκλώματος (entity)
- Το τμήμα δήλωσης της συμπεριφοράς - αρχιτεκτονικής του κυκλώματος.

Επαλήθευση ορθής λειτουργίας κυκλώματος

Για να επαληθεύσουμε τη λειτουργία ενός κυκλώματος χρησιμοποιούμε ένα αρχείο δοκιμών που ονομάζεται test-bench. (Το αρχείο δοκιμών συνήθως ονομάζεται όπως και το αρχείο που δηλώνει το κύκλωμα προσθέτοντας την ένδειξη _TB στο τέλος. Παράδειγμα: AND_gate_TB.vhd).

Το αρχείο δοκιμών έχει την ακόλουθη μορφή:

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

library work;
use work.all;
```

```
entity AND_gate_tb is
end AND_gate_tb;
```

```
architecture AND_gate_tb_a of AND_gate_tb is
component AND_gate is
      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );
  end component;
   signal X, y, z: std_logic;
begin
AND_gate_inst: AND_gate
     port map (
            x, y, z       );
```

```
stimulus_proc : process is
begin
x<='0';y<='0';
wait for 12ns;
x<='1';  y<='0';
wait for 2ns;
x<='1'; y<='1';
wait for 3ns;
```

```
x<='0'; y<='1';
wait for 7ns;
x<='0'; y<='0';
wait for 2ns;
  wait;
end process stimulus_proc;
end architecture AND_gate_tb_a;
```

Το testbench αποτελείται από 4 μέρη:

- Δήλωση βιβλιοθηκών
- Δήλωση οντότητας (παρατηρείστε ότι δεν υπάρχουν είσοδοι και έξοδοι αφού δεν πρόκειται για κύκλωμα με εισόδους και εξόδους αλλά για ένα περιβάλλον δοκιμής)
- Την αρχιτεκτονική η οποία αποτελείται από 2 τμήματα
    - Τμήμα δήλωσης του κυκλώματος που θέλουμε να ελέγξουμε και τον ονομάτων που δίνουμε στα σήματα αυτού
    - Τους συνδυασμούς εισόδων που θέλουμε να δοκιμάσουμε.

Όταν θέλουμε να ελέγξουμε ένα συνδυαστικό κύκλωμα 2 εισόδων δοκιμάζουμε και τους 4 δυνατούς συνδυασμούς των εισόδων, δηλαδή τον πίνακα αληθείας.
Ερώτηση: Ελέγχει όλους τους συνδυασμούς τιμών εισόδων το παραπάνω test-bench; (Απάντηση: ΟΧΙ)
Ο έλεγχος ορθής λειτουργίας γίνεται με τη βοήθεια εργαλείων προσομοίωσης. Ένα τέτοιο εργαλείο είναι το Modelsim.
Οδηγίες συντακτικού ελέγχου κώδικα και προσομοίωσης

1. Κάνουμε συντακτικό έλεγχο του αρχείου που περιγράφει το κύκλωμα. (επιλέγουμε το αρχείο και με δεξί click επιλέγουμε “compile”).
2. Στο κάτω τμήμα της οθόνης (παράθυρο transcript) με πράσινα γράμματα φαίνεται το επιτυχές αποτέλεσμα του συντακτικού ελέγχου ενώ με κόκκινα αν η προσπάθεια δεν ήταν επιτυχής. Στη δεύτερη περίπτωση πρέπει να κάνουμε διπλό click στα κόκκινα γράμματα για να μας δώσει περισσότερες λεπτομέρειες για τα λάθη μας. Οπότε:
    a. Ανοίγουμε το αρχείο περιγραφής του κώδικα
    b. Πηγαίνουμε στη γραμμή που μας υποδεικνύει το Modelsim και διορθώνουμε το λάθος
    c. Αποθηκεύουμε το αρχείο και
    d. Επαναλαμβάνουμε το συντακτικό έλεγχο
3. Αποτο menu τουmodelsimεπιλέγουμε simulate/start simulation/design και επιλέγουμε work/and_gate_tb
4. Απο το παράθυρο objects επιλέγουμε όλα τα σήματα και με δεξί click επιλέγουμε addtowave.
5. Μεγιστοποιούμε το παράθυρο wave και επιλέγουμε run. Τώρα μπορούμε να δούμε στην οθόνη πως αλλάζει η έξοδος z για διαφορετικές τιμές των εισόδων, όπως θα βλέπαμε και στο led στον πάγκο.

4.2.3 Τεστ αυτοαξιολόγησης

Άσκηση 1
Αναπτύξτε το παραπάνω πρόγραμμα με το όνομα AND_gate.vhd. Αναπτύξτε κύκλωμα πύλης OR και πύλης NOT. Τα αντίστοιχα αρχεία να αποθηκευτούν με τις ονομασίες OR_gate.vhd και NOT_gate.vhd αντίστοιχα.
Λύση

Διαδραστικό πρόγραμμα 4.2

Άσκηση 2

Χρησιμοποιώντας το testbench της πύλης AND (αρχείο AND_gate_TB.vhd), δοκιμάστε 40ns μετά την τελευταία αλλαγή τιμών εισόδου το συνδυασμό x=1, y=1 και αποθηκεύστε το νέο testbench με το όνομα AND_gate_TB_v2.vhd
Λύση

Διαδραστικό πρόγραμμα 4.3

Άσκηση 3
Αναπτύξτε test-bench για την πύλη OR που φτιάξατε με τη γλώσσα VHDL. Επαληθεύστε συντακτικά και λειτουργικά την πύλη σας.
(*Υπόδειξη*:

1. προσθέστε τα νέα αρχεία
2. Ελέγξτε τα συντακτικά
3. Επαναλάβετε τα βήματα 3, 4, 5 που φαίνονται παραπάνω για το νέο κύκλωμα.)

Άσκηση 4
Σας δίνεται το αρχείο example1.vhd και το σχετικό testbendh (example1_TB.vhd).

Διαδραστικό πρόγραμμα 4.4

Προσομοιώστε τη λειτουργία του κυκλώματος με το δοσμένο test-bench και απαντήστε:
Τη χρονική στιγμή 20ns, τι τιμή έχει:

- Η είσοδος x1 ☐ Σωστό 0
- Η είσοδος x2 ☐ Σωστό 1
- Η είσοδος x3 ☐ Σωστό 1
- Η έξοδος f ☐ Σωστό 0

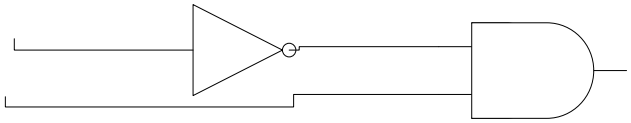
Έλεγχος απαντήσεων

Διαδραστικό πρόγραμμα 4.5

Άσκηση 5
Επαυξήστε το παραπάνω test-bench, προκειμένου να ελέγχει όλες τις πιθανές περιπτώσεις και απαντήστε:
Τι τιμή έχει η έξοδος όταν:
Οι είσοδοι έχουν τιμή 101 ☐ Σωστό 1
Οι είσοδοι έχουν τιμή 111 ☐ Σωστό 1
Λύση

Διαδραστικό πρόγραμμα 4.6

Άσκηση 6
Στο παρακάτω σχήμα φαίνεται ένα συνδυαστικό κύκλωμα.



a) Αναπτύξτε το αρχείο περιγραφής του κυκλώματος αυτού με χρήση της γλώσσας VHDL.
b) Υπολογίστε τον πίνακα αληθείας
c) Αναπτύξτε το αρχείο δοκιμής του κυκλώματος (testbench) με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο δοκιμών με το όνομα comb_v3_TB.vhd

Άσκηση 7
Δίνεται η έξοδος F ενός κυκλώματος η οποία εκφράζεται από τη συνάρτηση F=xy'+z.

a) Συμπληρώστε τον πίνακα αληθείας με βάση θεωρητικούς υπολογισμούς

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

έλεγχος
b) Αναπτύξτε το αρχείο περιγραφής του κυκλώματος αυτού με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο με το όνομα comb.vhd. Λύση
c) Αναπτύξτε το αρχείο δοκιμής του κυκλώματος (testbench) με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο δοκιμών με το όνομα comb_TB.vhd Λύση

Λύση

Διαδραστικό πρόγραμμα 4.7

4.3 Οι δομές IF – THEN – ELSE και CASE στη VHDL

4.3.1 Θεωρητικό υπόβαθρο

Η σύνταξη της ροής **IF** είναι:

[if label :] **IF** boolean expression **THEN**
{sequential statement}

ELSIFboolean expression **THEN**
{sequential statement}
ELSE
{sequential statement}
END IF [if label];

ΠΡΟΣΟΧΗ!!! Είναι ακολουθιακή δήλωση => Η σειρά εμφάνισης των συνθηκών είναι σημαντική
Το ισοδύναμο κύκλωμα έχει προτεραιότητες ανάλογα με τη σειρά εμφάνισης των συνθηκών στον κώδικα.

Παράδειγμα:

```
IFsel = '1' THEN
        c <= d;
ELSE
        c<=a;
endif ;
```

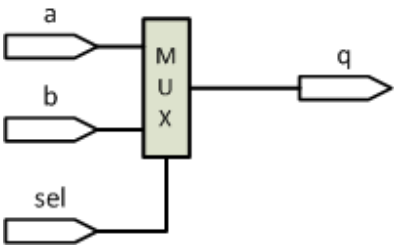
Η σύνταξη της ροής **CASE** είναι:

[case label :] **CASE** expression **IS**
WHEN choise1 =>
{sequential statement}
WHEN choise2 =>
{sequential statement}
. . .
. . .
END CASE;

Παράδειγμα:

```
CASE sel IS
        WHEN '0' => c<=a;
        WHEN '1' => c<=b;
ENDCASE;
```

Δείτε το κυκλωματικό ισοδύναμο



Σημείωση: Η παραπάνω case είναι ισοδύναμη με το παράδειγμα για την IF.
Παρατηρήσεις για την CASE:

- Η έκφραση πρέπει να είναι διακριτού τύπου
- Όλες οι δυνατές τιμές πρέπει να απαριθμούνται
- Χρήση others στην περίπτωση που αυτές είναι πολλές
- Όταν δεν πρέπει να αλλάξει κάποια τρέχουσα τιμή, δηλαδή καμία ενέργεια να μη λάβει χώρα (donothing), χρησιμοποιείται η δήλωση null.

  Παράδειγμα:

```
CASE a IS
WHEN   "00" => q1<= '1';
    "01" =>q1 <='0';
WHENOTHERS  =>null;
```

4.3.2 Πειραματικό μέρος

Άσκηση 1

Περιγράψτε έναν πολυπλέκτη 4 εισόδων δεδομένων (2 σήματα επιλογής/διεύθυνσης) σε VHDL χρησιμοποιώντας τη δομή IF-ELSE

Άσκηση 2
Αναπτύξτε το σχετικό αρχείο δοκιμών και επαληθεύστε τη λειτουργία του πολυπλέκτη.

4.3.3 Τεστ αυτοαξιολόγησης

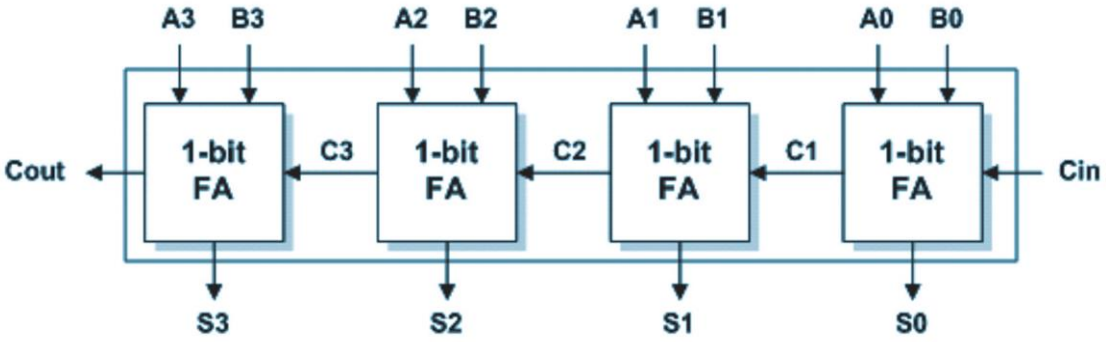
Περιγράψτε έναν πολυπλέκτη 4 εισόδων δεδομένων (2 σήματα επιλογής/διεύθυνσης) σε VHDL χρησιμοποιώντας τη δομή CASE
Λύση

Διαδραστικό πρόγραμμα 4.8

4.4 Διεπίπεδη λογική σε συνδυαστικά κυκλώματα

4.4.1 Θεωρητικό υπόβαθρο

Όπως γνωρίζουμε από την άσκηση σχεδίασης αθροιστών, ένας αθροιστής 4 bit υλοποείται με 4 πλήρης αθροιστές ή τρεις πλήρης αθροιστές και έναν ημιαθροιστή. Το σχετικό κύκλωμα φαίνεται στο ακόλουθο σχήμα, όπου είναι φανερό ότι το δομικό στοιχείο για την κατασκευή του κυκλώματος αθροιστή 4-μπιτων αριθμών είναι ο πλήρης αθροιστής. Το μοντέλο αυτό ονομάζεται δομικό.



Εικόνα 3: Αθροιστής τετραψήφιων δυαδικών αριθμών που αποτελείται από τέσσερις πλήρεις αθροιστές

Θα πρέπει λοιπόν πρώτα να σχεδιαστεί σε VHDL ένας πλήρης αθροιστής του ενός bit (1-bitFA) και στη συνέχεια να κληθεί αυτός 4 φορές ως υποκύκλωμα. Κάθε στοιχειώδες κύκλωμα (ή υποκύκλωμα) που αποτελεί τμήμα ενός πιο σύνθετου κυκλώματος καλείται *στοιχείο* (component). Κάθε στοιχείο σχεδιάζεται σε VHDL ξεχωριστά ως αυτόνομο κύκλωμα.
Στο πρόγραμμα σχεδίασης του συνολικού κυκλώματος (και συγκεκριμένα στο τμήμα της αρχιτεκτονικής) το δομικό στοιχείο – υποκύκλωμα δηλώνεται ως COMPONENT, ενώ καλείται μέσω της δήλωσης PORTMAP.
Στον ακόλουθο πίνακα φαίνεται η σύνταξη της δήλωσης στοιχείου και το παράδειγμα για έναν πλήρη αθροιστή.

δήλωσης στοιχείου **(component)**	
Σύνταξη	**Παράδειγμα**
COMPONENTόνομα_στοιχείου**IS** **PORT(** όνομα_σήματος {,όνομα_σήματος} : mode_σήματος τύπος_δεδομένων; … όνομα_σήματος {,όνομα_σήματος} : mode_σήματος τύπος_δεδομένων**);** **END COMPONENT** ;	**COMPONENT** FA **PORT** (A, B, Cin : IN std_logic; S, Cout : OUT std_logic); **END COMPONENT**;

Η κλήση ενός *στοιχείου* (component) γίνεται με τη δήλωση PORTMAP, της οποίας η γενική σύνταξη φαίνεται στον ακόλουθο πίνακα.

Δήλωση PORT MAP	
Σύνταξη	**Παράδειγμα**
ετικέτα:όνομα_στοιχείου **PORTMAP** (αντιστοίχιση σημάτων του port);	**FA_1:** FA **PORT MAP** (A(0), B(0), C(0), Sum(0), C(1));

Κάθε δήλωση PORTMAP ξεκινά υποχρεωτικά με ένα όνομα ή *ετικέτα* (label), ακολουθούμενο από το όνομα του στοιχείου το οποίο καλείται και τη δήλωση PORT MAP αμέσως μετά. Ακολουθεί μέσα σε παρένθεση η αντιστοίχιση των σημάτων του PORT της δήλωσης COMPONENT με τα σήματα που χρησιμοποιούνται πλέον στο τελικό πρόγραμμα. Με τον τρόπο αυτό γίνεται ουσιαστικά η σύνδεση των δομικών στοιχείων και του τελικού κυκλώματος.

4.4.2 Πειραματικό μέρος

Χρησιμοποιώντας το πλήρη αθροιστή που σας δίνεται εδώ και την ακόλουθη περιγραφή ενός αθροιστή 4-bit, προσομοιώστε έναν αθροιστή 4-bit.

```
entity Adder_4bit is
        port(a,b: in std_logic_vector(3 downto 0);
        enable: in std_logic;
        sum: out std_logic_vector(3 downto 0);
        cout: out std_logic);
end Adder_4bit;
architecturestruct of Adder_4bit is
signal c : std_logic_vector(2 downto 0);
signal s : std_logic_vector(3 downto 0);
component And_2
port( i1, i2 : in std_logic;
        o1: out std_logic);
end component;
component FA
port(a,b,cin: in std_logic;
        s, cout: out std_logic);
end component;
begin
        FA0: FA port map(a(0), b(0), 0, s(0), c(0));
        FA1: FA port map(a(1), b(1), c(0), s(1), c(1));
        FA2: FA port map(a(2), b(2), c(1), s(2), c(2));
        FA3: FA port map(a(3), b(3), c(2), s(3), c(3));
        g0: And_2 port map (s(0), enable, sum(0));
        g1: And_2 port map (s(1), enable, sum(1));
        g2: And_2 port map (s(2), enable, sum(2));
        g3: And_2 port map (s(3), enable, sum(3));
endstruct;
```

4.4.3 Τεστ αυτοαξιολόγησης

Έχοντας αναπτύξει στην προηγούμενη άσκηση τον πολυπλέκτη 4 σε 1, αναπτύξτε πολυπλέκτη 16 σε 1 χρησιμοποιώντας πολλαπλούς πολυπλέκτες 4 σε 1.

Λύση

Διαδραστικό πρόγραμμα 4.9

Δίνεται το ακόλουθο τμήμα κώδικα VHDL

```
ARCHITECTURE structural OF xor3 IS
SIGNAL U1_OUT: STD_LOGIC;

COMPONENT xor2 IS
    PORT(
            I1 : IN STD_LOGIC;
            I2 : IN STD_LOGIC;
            Y : OUT STD_LOGIC
        );
END COMPONENT;

BEGIN
    U1: xor2 PORT MAP (I1 => A,
                       I2 => B,
                       Y => U1_OUT);

    U2: xor2 PORT MAP (I1 => U1_OUT,
                       I2 => C,
                       Y => Result);

END structural;
```

labels

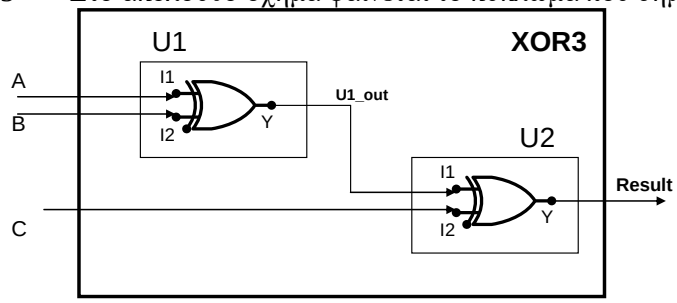
1 Ποιο είναι το δομικό στοιχείο που χρησιμοποιείται στο παράδειγμα? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

2 Πόσες φορές χρησιμοποιείται το δομικό στοιχείο? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

3 Ποιο είναι το label την πρώτη φορά? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

4 Ποιο είναι το όνομα του κυκλώματος που δημιουργείται με την παραπάνω αρχιτεκτονική; Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

5 Στο ακόλουθο σχήμα φαίνεται το κύκλωμα που δημιουργείται. Παρατηρείστε ότι:



- Συμπληρώστε τα κενά.
- Η είσοδος I1 του U1 συνδέεται στην είσοδο .....του κυκλώματος XOR3 ενώ η είσοδος I1 του U2 συνδέεται στην ..... του .....
- Το σήμα U1_out αποτελεί ταυτόχρονα..... του U1, ... του U2 και εσωτερικό σήμα για το XOR3.

Διαδραστικό πρόγραμμα 4.10

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

ENTITY dec4to16 IS
PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
En : IN STD_LOGIC ;
y : OUT STD_LOGIC_VECTOR(0 TO 15) ) ;
END dec4to16 ;

ARCHITECTURE Structure OF dec4to16 IS
      COMPONENT dec2to4
      PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
      En : IN STD_LOGIC ;
      y : OUT STD_LOGIC_VECTOR(0 TO 3) ) ;
      END COMPONENT ;
      SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
G1: FOR i IN 0 TO 3 GENERATE
      Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
     G2: IF i=3 GENERATE
Dec_left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
     END GENERATE ;
END GENERATE ;
END Structure;
```

- Ποιο είναι το δομικό στοιχείο που χρησιμοποιείται στο παράδειγμα? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής
- Πόσες φορές χρησιμοποιείται το δομικό στοιχείο? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

Διαδραστικό πρόγραμμα 4.11

4.5 Σχεδίαση απλών ακολουθιακών κυκλωμάτων

4.5.1 Θεωρητικό υπόβαθρο

Τα σύγχρονα ακολουθιακά κυκλώματα μοντελοποιούνται/περιγράφονται με clockedprocesses. Για τη διέγερση τους απαιτείται αλλαγή της τιμής του σήματος ρολογιού. Ένα ακολουθιακό κύκλωμα αλλάζει κατάσταση μόνο στις χρονικές στιγμές αλλαγής του ρολογιού.
Οι εναλλακτικές περιγραφές του ρολογιού φαίνονται στον ακόλουθο πίνακα:

1	Alt 1: **PROCESS**(CLK) **BEGIN** **IF**clk'event**AND**clk='1' **THEN** q<=d; **ENDIF**; **ENDPROCESS**;
2	Alt 2: **PROCESS**(CLK) **BEGIN** **IF**clk='1' **THEN** q<=d; **ENDIF**; **ENDPROCESS**;
3	Alt 3: **PROCESS**(CLK) **BEGIN** **IF**clk'event**AND**clk='1' **AND**clk'last_value='0' **THEN** q<=d;

	ENDIF; **ENDPROCESS**;
4	Alt 4: **PROCESS** **BEGIN** **WAITUNTIL**clk='1'; q<=d; **ENDPROCESS**;
5	Alt 5: **PROCESS** **BEGIN** **WAITUNTIL**rising_edge(clk); q<=d; **ENDPROCESS**;

Πίνακας 4.2: Οι εναλλακτικές περιγραφές του ρολογιού

4.5.2 Πειραματικό μέρος

Δίνεται ο ακόλουθος μετρητής

```
architecturertlofcounteris
signalz_int: integer;
begin
z<=z_int;
process(rst, clk)
begin
ifrst='0' then
z_int<=0;
elsifclk='1' and clk'event then
z_int<=z_int+1;
end if;
endprocess;
endrtl;
```

Και το σχετικό αρχείο δοκιμών

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

library work;
use work.all;
entity counter_tb is
end counter_tb;
architecture counter_tb_a of counter_tb is

component counter is

      port (
rst: in std_logic;
clk: in std_logic;
            z  : out integer
                  );
end component;

constant clk_hp : time := 3000 ps;
  signal rst      :  std_logic;
```

```
  signal clk      :  std_logic;
  signal Z        :  INTEGER   ;

begin

counterinst: counter
port map
          (rst ,
clk,
           z    );
--------------------------------------------------------------------
clock_gen_proc : process is
begin
clk<= '1';
  wait for clk_hp;

clk<= '0';
  wait for clk_hp;

end process clock_gen_proc;
--------------------------------------------------------------------
stimulus_proc : process is
begin
rst<='0';


wait for 2* clk_hp;
wait for 0.5 ns ;
rst<='1';
wait for 60* clk_hp;
rst<='0';
wait for 2* clk_hp;
rst<='1';
wait for 2* clk_hp;
wait for 20* clk_hp;
  wait;

endprocessstimulus_proc;
```

Άσκηση 1

1. Ακολουθείστε τις οδηγίες προσομοίωσης, και ελέγξτε τη λειτουργία του παραπάνω κυκλώματος counter.
2. Αναπτύξτε κώδικα που να περιγράφει κύκλωμα μετρητή που μετράει από 1 ως 9 κυκλικά και επαληθεύστε τη λειτουργία του.

4.5.3 Τεστ αυτοαξιολόγησης

1. Αναπτύξτε κώδικα που να περιγράφει μετρητή με είσοδο x ο οποίος για x=0 μετράει από 1 έως 7 ανά δύο ενώ x=1 από 6 έως 12 ανά 1. (Υπόδειξη: ο μετρητής να μηδενίζεται όταν ενεργοποιείται το reset και να ελέγχεται πρώτα η περίπτωση x=0 και μετά η τιμή x=1.)

Λύση

Διαδραστικό πρόγραμμα 4.12

2. Χρησιμοποιώντας το ακόλουθο test-bench απαντήστε:

Ποια τιμή έχει ο μετρητής, τη χρονική στιγμή:

a. 3ns
b. 30ns
c. 70ns
d. 240ns

Έλεγχος

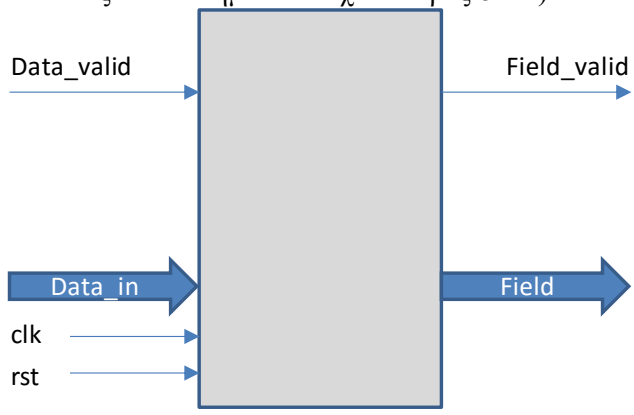
Διαδραστικό πρόγραμμα 4.13

3. Αναπτύξτε κώδικα που να περιγράφει κύκλωμα μετρητή που μετράει από 3 ως 53 κυκλικά και επαληθεύστε τη λειτουργία του.

4.6 Διεπίπεδη λογική σε ακολουθιακά κυκλώματα

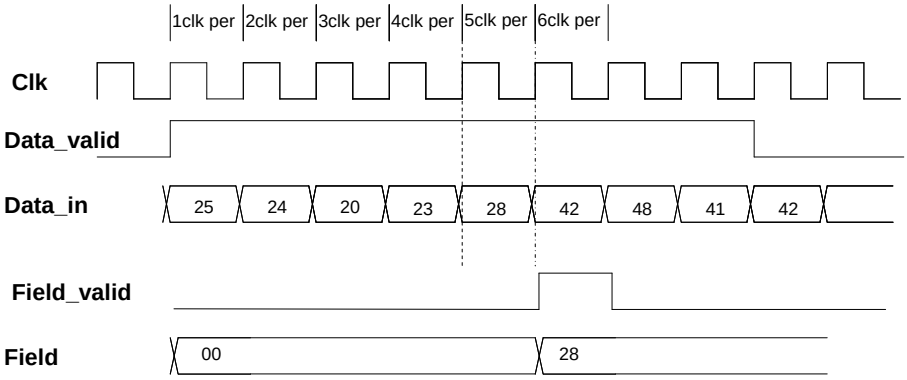
Θα εξετάσουμε την περιγραφή με VHDL διεπίπεδης λογικής με βάση ένα παράδειγμα: ένα κύκλωμα που υλοποιείται συχνά στο δέκτη ενός τηλεπικοινωνιακού συστήματος. Σε ένα τηλεπικοινωνιακό σύστημα, στην πλευρά του δέκτη, όταν λαμβάνεται ένα πακέτο πληροφορίας, αυτό προωθείται για επεξεργασία. Για παράδειγμα, πρέπει να βρεθεί η διεύθυνση του παραλήπτη προκειμένου αυτό να προωθηθεί κατάλληλα. Η θέση στην οποία βρίσκεται κάθε πληροφορία (π.χ. διεύθυνση παραλήπτη) είναι προκαθορισμένη από τα διεθνή πρότυπα (standards). Έτσι στη συσκευή του δέκτη ένα σύνολο ψηφιακών κυκλωμάτων αναλαμβάνουν την επεξεργασία του πακέτου. Υποθέτουμε ότι έχει φτάσει στο δέκτη ένα πακέτο πληροφορίας, και **πρέπει να βρούμε το περιεχόμενο (την πληροφορία) που βρίσκεται στο 5ºbyte του πακέτου** και να το προωθήσουμε σε ένα άλλο κύκλωμα για επεξεργασία. Ζητείται η σχεδίαση του κυκλώματος field_filtering το οποίο θα υλοποιεί την παραπάνω λειτουργία, σύμφωνα με τις παρακάτω προδιαγραφές:

Η εξωτερική μορφή του κυκλώματος field_filtering φαίνεται στο ακόλουθο σχήμα. (Σημειώνεται ότι το σήμα data όπως και το σήμα field έχουν εύρος 8 bit.)

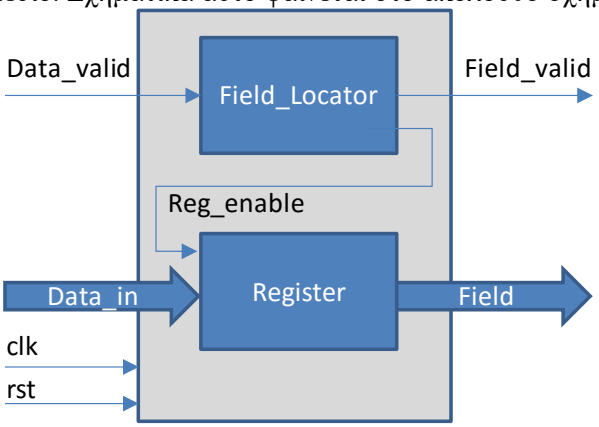


Εικόνα 4.6: Διεπαφές μονάδας field_filtering

Η επιθυμητή (εσωτερική) λειτουργία του κυκλώματος φαίνεται στο ακόλουθο σχήμα.

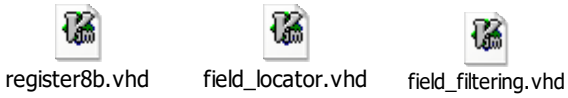


Το κύκλωμα field_filtering να δημιουργηθεί με δομικά στοιχεία (components) έναν καταχωρητή (register) και ένα κύκλωμα field_locator που εντοπίζει τον παλμό ρολογιού στον οποίο το κύκλωμα δέχεται το επιθυμητό πεδίο. Σχηματικά αυτό φαίνεται στο ακόλουθο σχήμα.



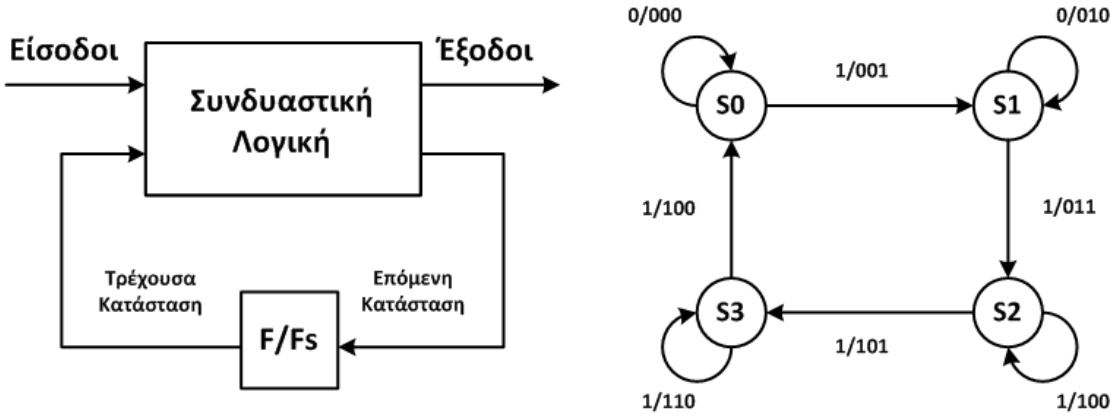
Εικόνα 4.7: Οργάνωση μονάδας field_filtering

Αναπτύξτε το σχετικό αρχείο δοκιμών και προσομοιώστε τη λειτουργία.



4.7 Σχεδίαση μηχανών καταστάσεων

Οι Μηχανές Πεπερασμένων Καταστάσεων (FiniteStateMachines, FSMs) χρησιμοποιούνται, για την υλοποίηση μιας λογικής ελέγχου (controllogic), η οποία είναι πάντοτε γρηγορότερη από την υλοποίηση αυτής σε επεξεργαστή μέσω προγράμματος. Αποτελούνται από συνδυαστικό κύκλωμα και F/Fs και μπορούν να περιγραφούν από ένα διάγραμμα εναλλαγής καταστάσεων (StateTransitionDiagram – STD).



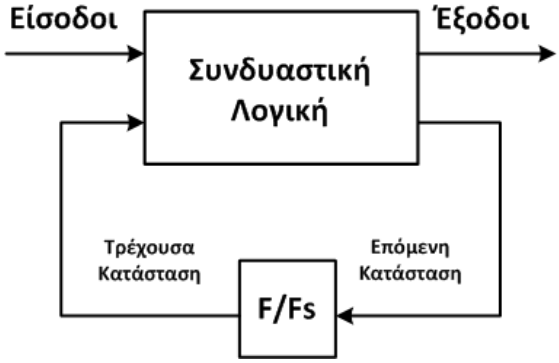
Εικόνα 4.8: Παράδειγμα οργάνωσης μηχανής πεπερασμένων καταστάσεων και διαγράμματος εναλλαγής καταστάσεων (StateTransitionDiagram – STD)

Για τη σωστή λειτουργία μιας μηχανής απαιτείται οι είσοδοι να έχουν σταθεροποιηθεί πριν την άφιξη του ρολογιού. Η καθυστέρηση καθορίζεται από το μέγιστο χρόνο υπολογισμού της νέας κατάστασης και κατά συνέπεια από το μέγιστο χρόνος απόκρισης του συνδυαστικού κυκλώματος.
Οι Μηχανές Πεπερασμένων Καταστάσεων ακολουθούν μια λειτουργία δύο φάσεων:

- Φάση 1: Υπολογισμός της νεάς κατάστασης
- Φάση 2: Δειγματοληψία της νέας κατάστασης

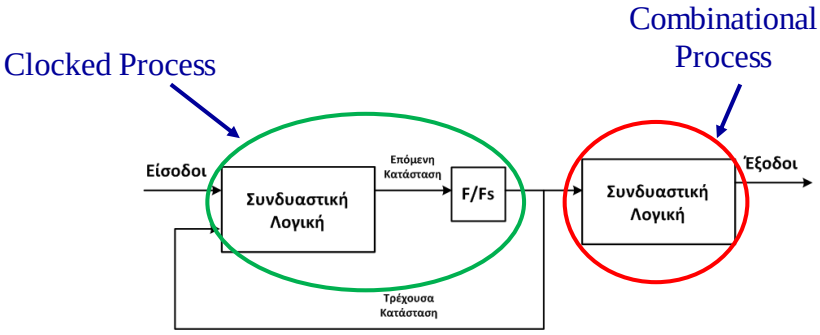
Υπάρχουν δύο μέθοδοι υλοποίησης:

- MealyFSMs : Οι έξοδοι είναι συνάρτηση των εισόδων και της τρέχουσας κατάστασης
-



Εικόνα 4.9: Οργάνωση μηχανής πεπερασμένων καταστάσεωνMealy

- MooreFSMs : Οι έξοδοι είναι συνάρτηση **μόνο** της τρέχουσας κατάστασης



Εικόνα 4.10: Οργάνωση μηχανής πεπερασμένων καταστάσεωνMoore

Οι MooreFSMs απαιτούν ένα κύκλο ρολογιού παραπάνω: ένα κύκλο υπολογισμού της κατάστασης και ένα κύκλο για τον υπολογισμό της εξόδου.
Στην VHDL, **<u>ο σχεδιαστής μπορεί να δώσει ονόματα στις καταστάσεις</u>** αφήνοντας την κωδικοποίηση αυτών (στο δυαδικό σύστημα) στα εργαλεία. Τα ονόματα των καταστάσεων δηλώνονται στην αρχή της αρχιτεκτονικής.
Παράδειγμα:

```
architecturemy_arch of flag_detector is
    type state is (zero,one,two,three,four,five,six,seven,eight);
signalpr_state, nx_state: state;
```

Με αυτό τον τρόπο έχουμε ορίσει 9 καταστάσεις.
Η αρχιτεκτονική ενός τέτοιου κυκλώματος περιγράφεται από δύο process:

- μια που υλοποιεί τις μεταβάσεις στην ακμή του ρολογιού.

Παράδειγμα:

```
PROCESS (rst,clk)
begin
if (rst='0') then
```

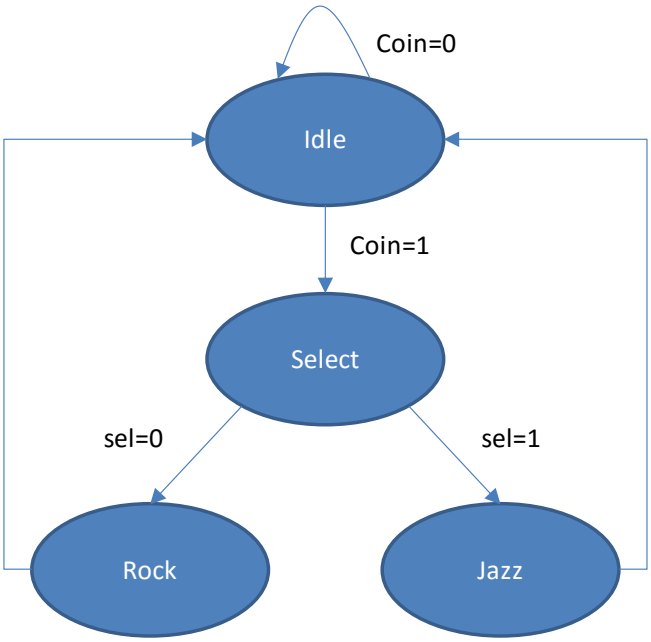
```
pr_state<= zero;
elsif (clk'event and clk='1') then
        pr_state<= nx_state;
endif;
endprocess;
```

- μια που υλοποιεί το συνδυαστικό κύκλωμα και συνήθως περιγράφεται με την εντολή CASE. Η σύνταξη της εντολής case περιγράφεται παρακάτω ενώ επίσης ακολουθεί και ένα παράδειγμα. Στη θέση της «έκφραση» στην περίπτωση των ακολουθιακών κυκλωμάτων συνήθως μπαίνει η παρούσα κατάσταση.
-

Σύνταξη εντολής CASE	**Παράδειγμα**

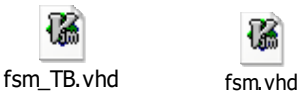
CASE έκφραση IS
WHEN , σταθερη_τιμή =>
εντολή ;
[εντολή;]
WHEN , σταθερη_τιμή =>
εντολή ;
[εντολή;]
WHENOTHERS =>
εντολή ;
[εντολή;]
END CASE
	process (d, pr_state)
	begin
	case pr_state is
	when zero =>
	if (d='0') then
	nx_state<= one;
	else
	nx_state<= zero;
	end if;
	
	when one =>
	if (d='1') then
	nx_state<= two;
	else
	nx_state<= one;
	end if;
	.....
	when eight =>
	if (d='1') then
	nx_state<= two;
	else
	nx_state<= one;
	end if;
	end case;
	end process;
	end my_arch;

Υλοποιείστε με χρήση της γλώσσας VHDL την ακόλουθη μηχανή καταστάσεων. Πρόκειται για μια μηχανή, η οποία ελέγχει την αναπαραγωγή διαφορετικών ειδών μουσικής (Rock ή Jazz) σε μία συσκευή αναπαραγωγής μουσικής κατόπιν πληρωμής (τύπου jukebox).



Εικόνα 4.11: Παράδειγμα μηχανής πεπερασμένων καταστάσεων

Η μηχανή βρίσκεται στην κατάσταση αναμονής (idle) και ενεργοποιείται με την εισαγωγή του κατάλληλου κέρματος (το οποίο σηματοδοτείται με την μετάβαση του σήματος Coin σε λογικό 1). Στην κατάσταση Select ο πελάτης επιλέγει το είδος μουσικής που θέλει με το σήμα sel. Στις καταστάσεις Rock και Jazz εκτελείται το αντίστοιχο μουσικό πρόγραμμα και η συσκευή επιστρέφει στην κατάσταση αναμονής idle.



4.8 Βιβλιογραφία

P. Ashenden, "*The Designer's Guide to VHDL*", Morgan Kaufman Publishers, 1996
S. Sjoholm and L. Lennart, "*VHDL for Designers*", Prentice Hall, 1997
B. Cohen, "*VHDL Coding Styles and Methodologies*", Kluwer Academic Publishers, 1999
Z. Navabi, "*VHDL-Analysis and Modeling of Digital Systems*", Mc Graw-Hill, 1993
On-line μαθήματα σε ASIC/VHDL design:
http://www.dacafe.com/ASICs.htm
http://www.ecs.umass.edu/ece/vspgroup/burleson/courses/558/
http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.vhd/html/vhdl-all-e.html
http://www.eas.asu.edu/~yong598d/VHDL_links.html
Tutorials/Papers:
http://www.vhdl.org/fmf/wwwpages/FMF_ECL_models_paper.html
http://www.bluepc.com/download.html#downloadtop
http://www.symphonyeda.com/products.htm
http://www.angelfire.com/electronic/in/vlsi/vhdl.html#vhdl
Άλλες πηγές:
http://www.ieee.org
http://www.acm.org
http://www.vhdl.org
http://tech-www.informatik.uni-hamburg.de/vhdl/
http://www.eeglossary.com/vhdl.htm
http://www.ent.ohiou.edu/~starzyk/network/Class/ee514/intro.html

Κεφάλαιο 5. Προγραμματισμός Ολοκληρωμένων Κυκλωμάτων

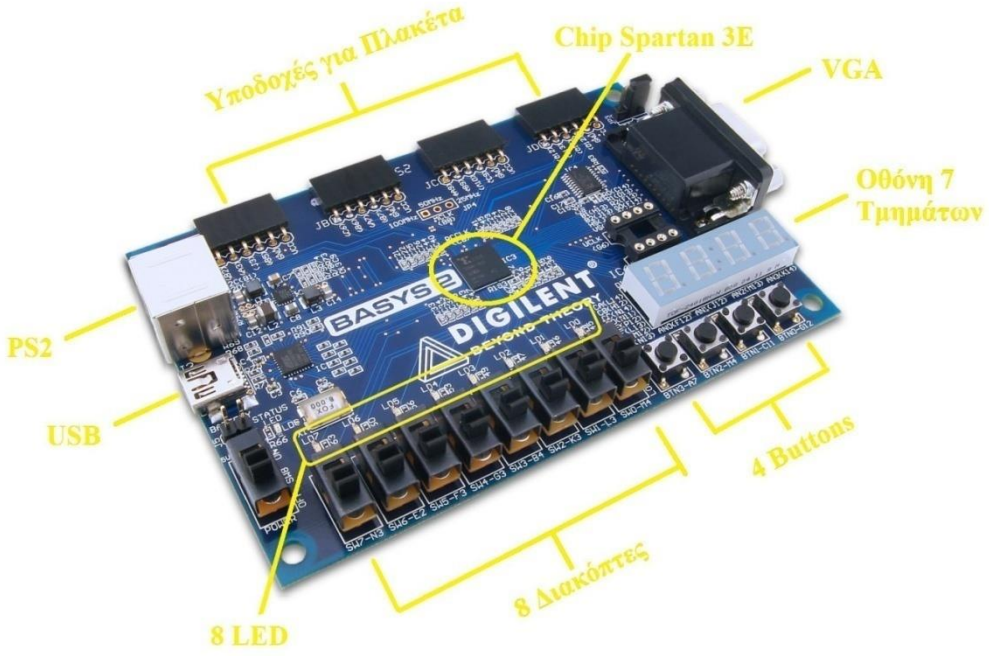
Στο κεφάλαιο αυτό επικεντρωνόμαστε στον προγραμματισμό συσκευών FieldProgrammableGateArrays – FPGA με απλά κυκλώματα. Οι FPGAs συνήθως τοποθετούνται σε κατάλληλες πλακέτες προγραμματισμού που υπάρχουν συχνά στα εργαστήρια. Πρώτα μελετάμε την αρχιτεκτονική μιας τέτοιας πλακέτας και τη διαδικασία προγραμματισμού και εν συνεχεία παραθέτουμε απλά παραδείγματα προγραμματισμού με σύγχρονα αλλά και ασύγχρονα κυκλώματα.

5.1 Συσκευή για πειράματα με προγραμματιζόμενα ολοκληρωμένα DIGILENT BASYS2

Τον τελευταίο καιρό παρατηρείται αξιοσημείωτη αύξηση της πυκνότητας των FPGAs λόγω της αντίστοιχης αύξησης στην πολυπλοκότητα των συστημάτων που σχεδιάζονται. Το FPGAboardBasys2 που φαίνεται και στην **Εικόνα 5.1: Η πλακέτα Basys2 της Digilent**

περιλαμβάνει:

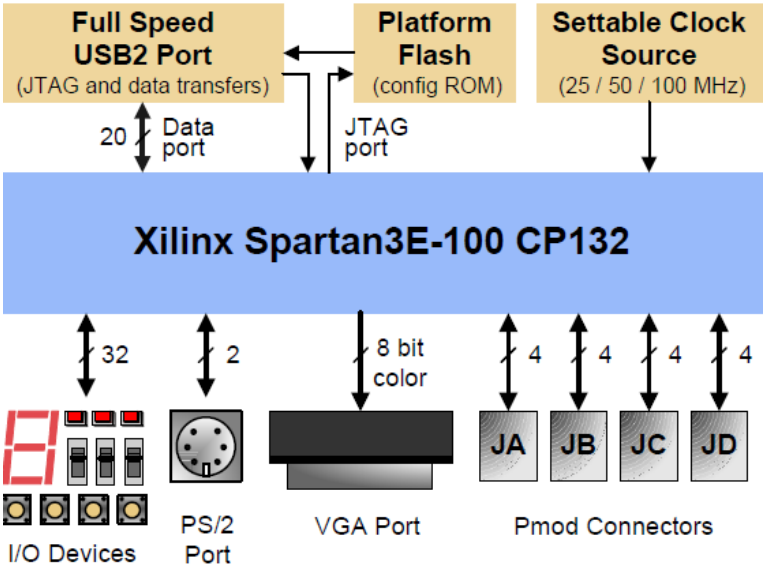
- Το FPGAchipSpartan 3E της Xilinx, με χωρητικότητα 100.000 πύλες.
- 8 διακόπτες
- 4 buttons σαν εισόδους
- 8 led
- ένα 7- segmentdisplay τεσσάρων ψηφίων για εξόδους.



Εικόνα 5.1: Η πλακέτα Basys2 της Digilent

Η τροφοδοσία της πλακέτας και ο προγραμματισμός γίνονται μέσω θύρας USB, ενώ το πρόγραμμα μπορεί να αποθηκευτεί στη μνήμη RAM του FPGA ή στη μνήμη flash της πλακέτας για να μη χάνεται χωρίς την τροφοδοσία.

Επίσης το Basys2 παρέχει υποδοχή πληκτρολογίου PS2, έξοδο για οθόνη VGA, 25/50/100 MHz ρολόι και τέσσερις υποδοχές για πλακέτες επέκτασης. Επίσης, το συγκεκριμένο FPGA περιέχει δύο ενσωματωμένους μικροεπεξεργαστές, έναν microBlaze και έναν PicoBlaze. Το συγκεκριμένο FPGA έχει χαμηλό κόστος και αρκετά μεγάλες δυνατότητες που ικανοποιούν τις ανάγκες ενός αρχάριου προγραμματιστή.



Εικόνα 5.2: Διάγραμμα εισόδων-εξόδων της συσκευής

Για τον προγραμματισμό της FPGA απαιτείται

1. η παραγωγή ενός αρχείου bit (bitfile) το οποίο θα χρησιμοποιηθεί για τον προγραμματισμό της FPGA. Αυτό γίνεται με λογισμικό που παρέχουν οι εταιρείες που διαθέτουν και τις FPGAs και στη συγκεκριμένη περίπτωση πρόκειται για το software**Xilinx ISE Design Suite 12.3**.
2. να περάσουμε το bitfile στο FPGAboard. Για το σκοπό αυτό χρησιμοποιούμε το software**DIGILENT ADEPT SYSTEM V2.6.1,** το οποίο συνοδεύει την πλακέτα μας.

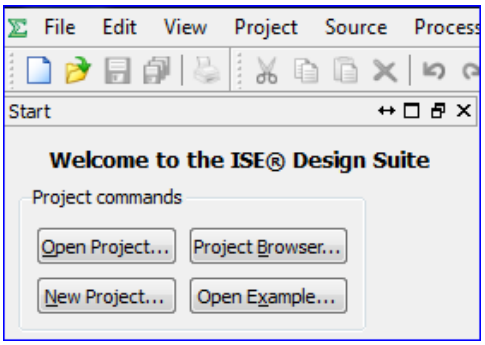
5.1.1 Διαδικασία παραγωγής bitfile προγραμματισμού από VHDL

Για να παράξουμε το αρχείο bit ακολουθούμε τα παρακάτω βήματα:

1. Ανοίγουμε το πρόγραμμα Xilinx ISE Design Suite 12.3
2. Δημιουργούμε New project

Άσκηση Αυτοαξιολόγησης

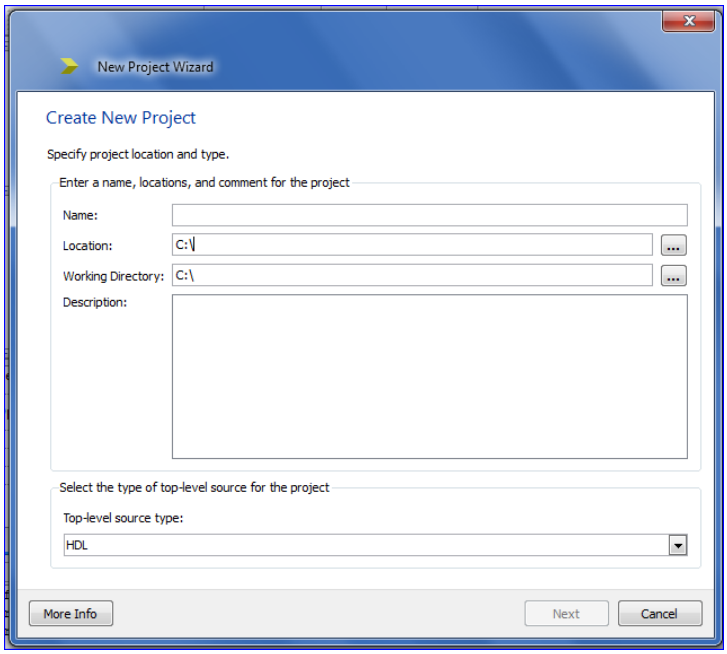
Διαδραστικό πρόγραμμα 5.1



Ανοίγει ο New Project Wizard όπου επιλέγουμε το όνομα και το φάκελο του project, καθώς και τον τρόπο εισαγωγής της περιγραφής του κυκλώματος.

Άσκηση Αυτοαξιολόγησης

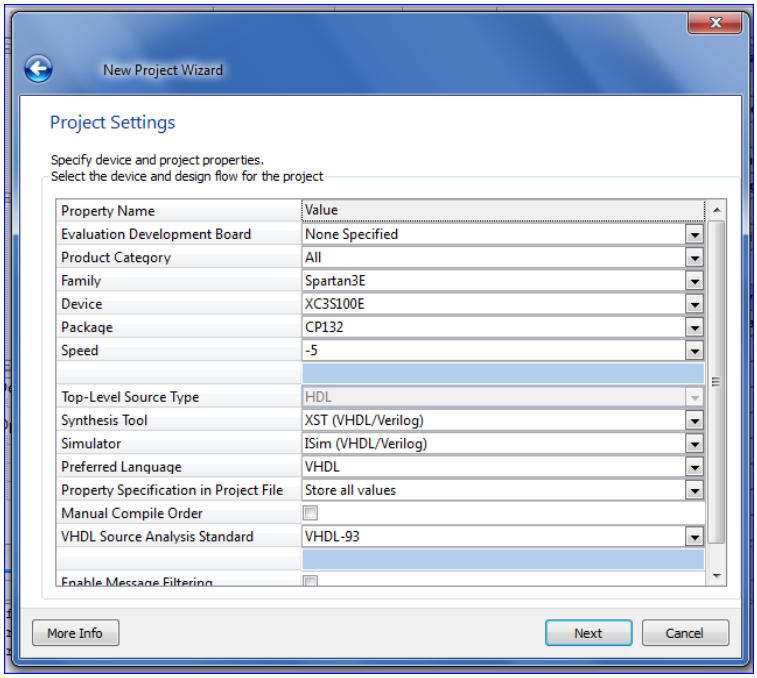
Διαδραστικό πρόγραμμα 5.2



Επιλέγοντας HDL μπορούμε να διαλέξουμε ανάμεσα στις γλώσσες προγραμματισμού VHDL και Verilog στο επόμενο παράθυρο, όπου επίσης επιλέγουμε και το FPGA chip που θα προγραμματίσουμε (Εδώ πρόκειται για το Spartan 3E–100 CP132).

Άσκηση Αυτοαξιολόγησης

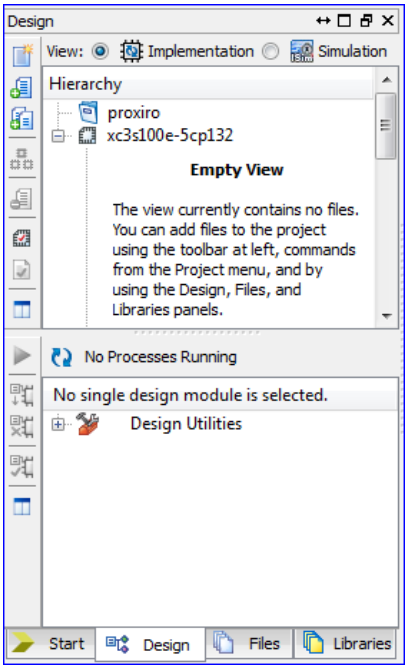
Διαδραστικό πρόγραμμα 5.3



Προσθέτουμε καινούργια αρχεία στο project επιλέγοντας New Source στη καρτέλα Design (ή ανοίγουμε έτοιμα

Άσκηση Αυτοαξιολόγησης

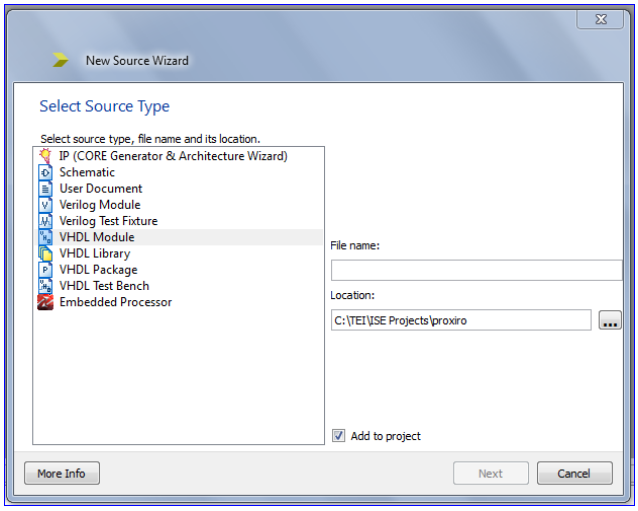
Διαδραστικό πρόγραμμα 5.4



Επιλέγοντας New Source ανοίγει ο New Source Wizard όπου διαλέγουμε τον τύπο του αρχείου που θα προσθέσουμε. Επιλέγουμε VHDL Module.

Άσκηση Αυτοαξιολόγησης

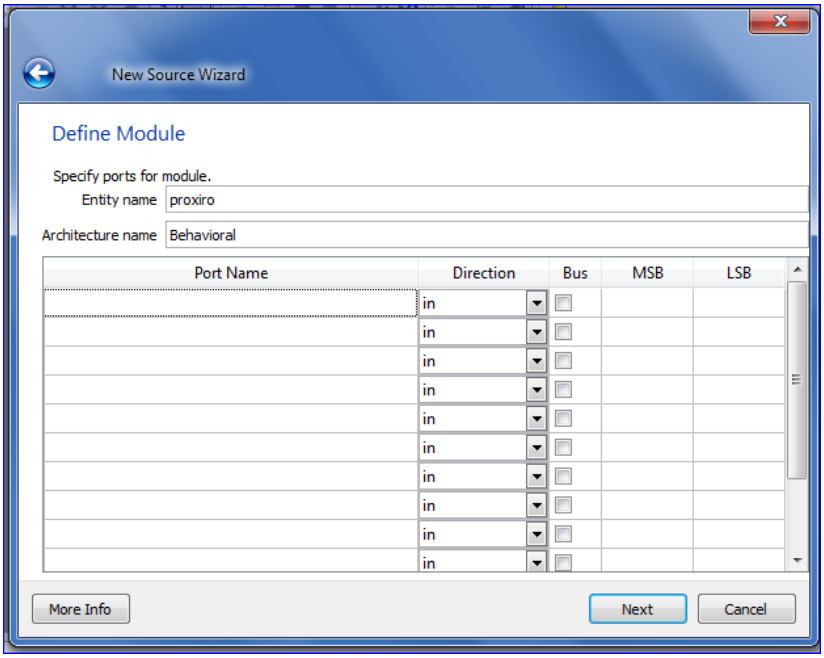
Διαδραστικό πρόγραμμα 5.5



Στη συνέχεια προσθέτουμε τις εισόδους και εξόδους του κυκλώματος και το πρόγραμμα τις δηλώνει αυτόματα στο entity, αλλά μπορούμε να το κάνουμε και μόνοι μας γράφοντας τον κώδικα.

Άσκηση Αυτοαξιολόγησης

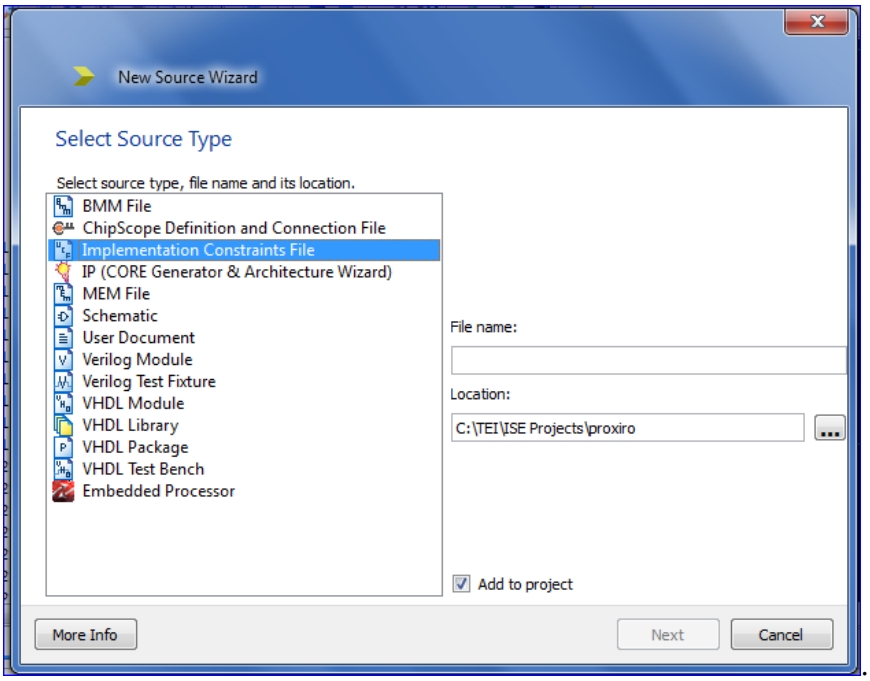
Διαδραστικό πρόγραμμα 5.6



Επίσης, πρέπει να προσθέσουμε και το αρχείο .ucf όπου θα δηλώσουμε τα pins που αντιστοιχούν στις εισόδους και εξόδους μας.

Άσκηση Αυτοαξιολόγησης

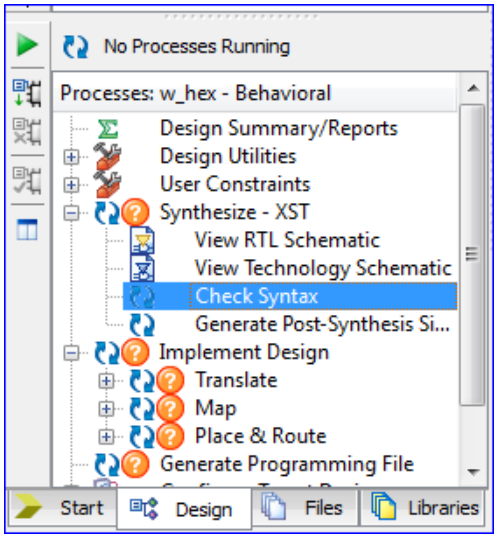
Διαδραστικό πρόγραμμα 5.7



Καθώς αναπτύσσουμε τον κώδικα του προγράμματος πρέπει ανά τακτά διαστήματα να πατάμε Check Syntax για να εντοπίζουμε τυχών λάθη στη σύνταξη.

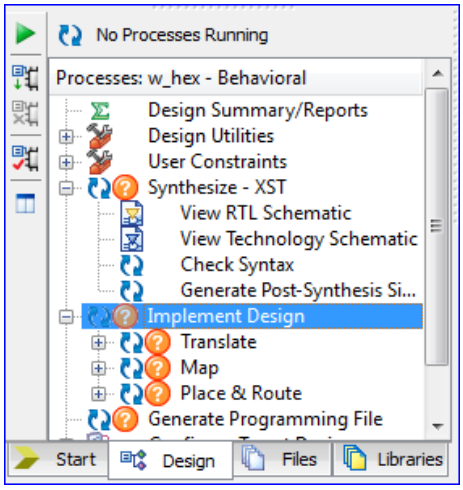
Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 5.8



Άσκηση Αυτοαξιολόγησης

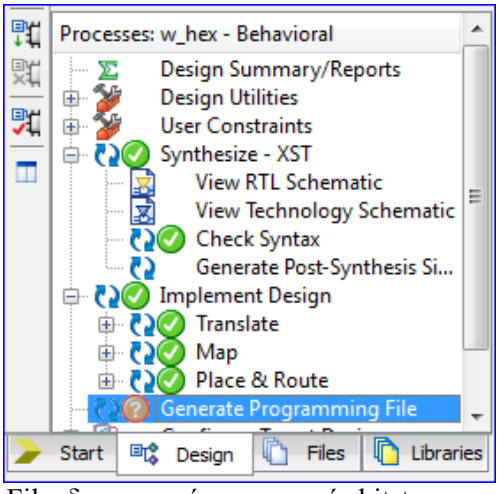
Διαδραστικό πρόγραμμα 5.9



Όταν έχουμε ολοκληρώσει το πρόγραμμα μας πατάμε Implement Design και το ISE ‘υλοποιεί’ το κύκλωμα.

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 5.10



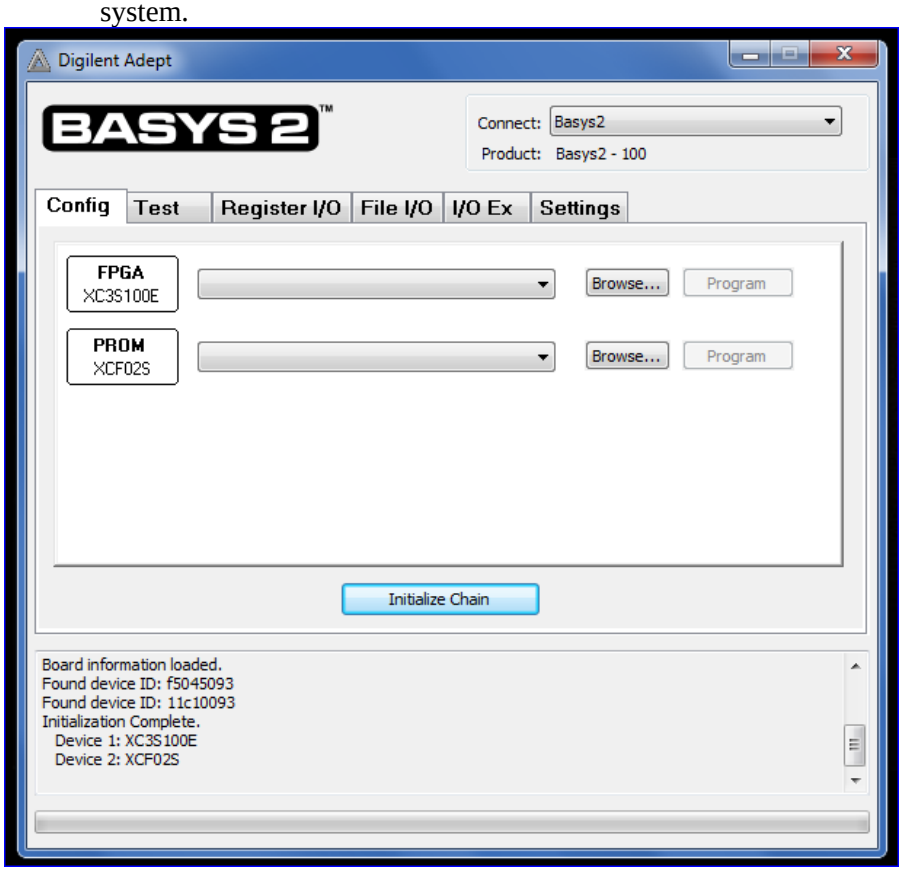
Πατώντας Generate Programming File, δημιουργείται το αρχείο bitstream που θα φορτωθεί στο chip.

5.1.2 Προγραμματισμός της FPGA

Για να προγραμματίσουμε την FPGA, συνδέουμε το FPGA Board στη θύρα USB, αφού έχουμε τοποθετήσει το Mode Jumper (JP3) στη θέση ‘PC’, και ανοίγουμε το πρόγραμμα Adept της Digilent. Το πρόγραμμα εντοπίζει μόνο του τη συσκευή. Κάθε φορά που θέλουμε να προγραμματίσουμε το chip πρέπει να πατάμε Initialize Chain.

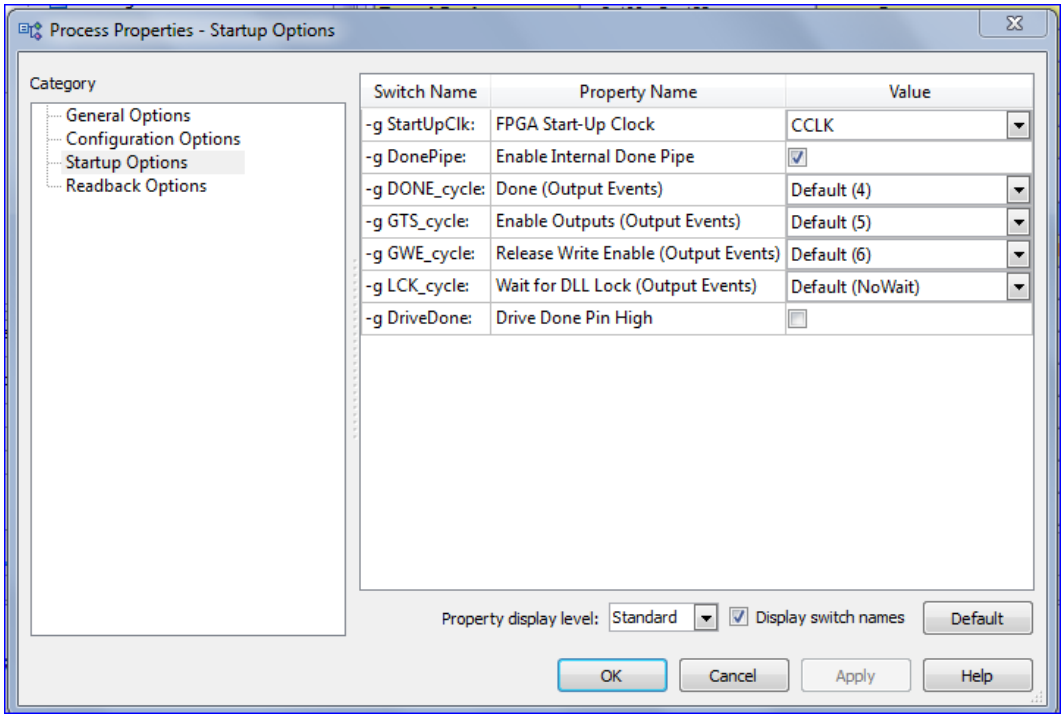
Για να αποθηκεύσουμε το κύκλωμα στη μνήμη flash της συσκευής πατάμε browse στη δεύτερη σειρά (PROM) και βρίσκουμε το αρχείο .bit που δημιουργήσαμε με το ISE και πατάμε ‘Program’.
Για να κατεβάσουμε το bitfile στην FPGA, εκτελούμε τα ακόλουθα βήματα:

1. Συνδέουμε στον υπολογιστή το fpga board και ανοίγουμε την εφαρμογή digilent adept system.



Στο παράθυρο βλέπουμε δυο επιλογές FPGA και PROM. Στην επιλογή FPGA μπορούμε να τρέξουμε εφαρμογές και όταν θα κλείσουμε την συσκευή οι πληροφορίες θα έχουν σβηστεί από τη μνήμη. Στην PROM η εφαρμογή που θα εκτελούμε θα αποθηκεύεται στη μνήμη της συσκευής.
Στην επιλογή FPGA, πατάμε browse και επιλέγουμε το bit file που δημιουργήσαμε προηγουμένως .
Τέλος πατάμε Program και έχουμε προγραμματίσει το Fpga με το ψηφιακό κύκλωμα που είχαμε σχεδιάσει σε VHDL (πύλη AND).
Χρησιμοποιούμε τους διακόπτες και τα LED όπως τα είχαμε επιλέξει στον προγραμματισμό, κάνουμε την επαλήθευση του κυκλώματος πάνω στο FPGA board.

- Όταν ολοκληρωθεί ο προγραμματισμός του chip, κλείνουμε τη συσκευή και τοποθετούμε το Mode Jumper στη θέση ‘ROM’. Μόλις ξανανοίξουμε τη συσκευή το κύκλωμα μας λειτουργεί.
- Μπορούμε επίσης να φορτώσουμε το πρόγραμμα απευθείας στο FPGA χωρίς να αλλάζουμε θέση στο Jumper, αλλά το πρόγραμμα χάνεται μόλις κλείσουμε την τροφοδοσία. Για να το κάνουμε αυτό, κάνουμε δεξί κλικ στο Generate Programming File, στο ISE.



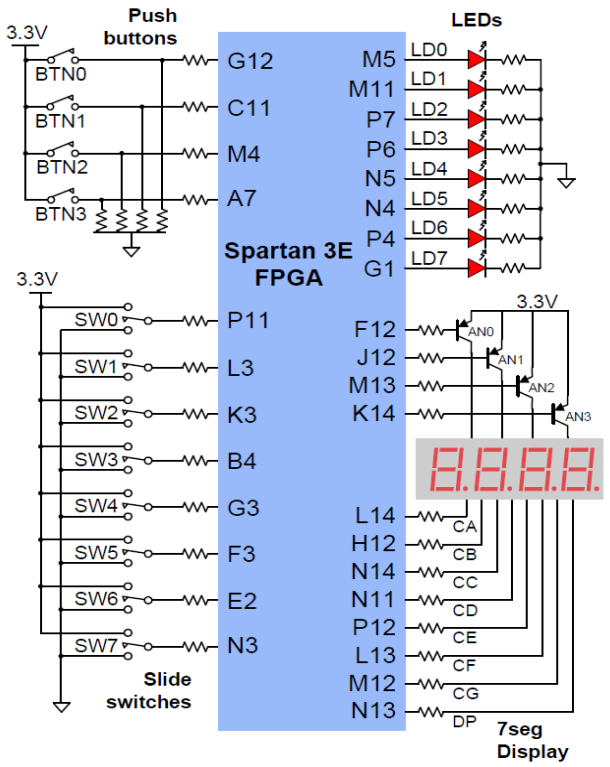
Εικόνα 5.3 : Αλλάζουμε τις ρυθμίσεις του ρολογιού.

Στην κατηγορία Startup Options, αλλάζουμε το Start-up clock από CCLK, που είναι το ρολόι της πλακέτας, σε JTAG που είναι το πρωτόκολλο που χρησιμοποιείται από τη RAM εντός του FPGA. Τώρα μπορούμε να ξαναδημιουργήσουμε το αρχείο .bit και να το φορτώσουμε στο FPGA.

Σημειώνεται ότι για λεπτομέρειες για τα Pin της συγκεκριμένης συσκευής δείτε εδώ.

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 5.11



FPGA pin definition table color key		
Grey		Not available to user
Green		User I/O devices
Yellow		Data ports
Tan		Pmod connector signals
Blue		USB signals

Basys2 Spartan-3E pin definitions

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
C12	JD1	P11	SW0	N14	CC	B2	JA1	P8	MODE0	M7	GND
A13	JD2	M2	USB-DB1	N13	DP	C2	USB-WRITE	N7	MODE1	P5	GND
A12	NC	N2	USB-DB0	M13	AN2	C3	PS2D	N6	MODE2	P10	GND
B12	NC	M9	NC	M12	CG	D1	NC	N12	CCLK	P14	GND
B11	NC	N9	NC	L14	CA	D2	USB-WAIT	P13	DONE	A6	VDDO-3
C11	BTN1	M10	NC	L13	CF	L2	USB-DB4	A1	PROG	B10	VDDO-3
C6	JB1	N10	NC	F13	RED2	L1	USB-DB3	N8	DIN	E13	VDDO-3
B6	JB2	M11	LD1	F14	GRN0	M1	USB-DB2	N1	INIT	M14	VDDO-3
C5	JB3	N11	CD	D12	JD4	L3	SW1	P1	NC	P3	VDDO-3
B5	JA4	P12	CE	D13	RED1	E2	SW6	B3	GND	M8	VDDO-3
C4	NC	N3	SW7	C13	JD3	F3	SW5	A4	GND	E1	VDDO-3
B4	SW3	M6	UCLK	C14	RED0	F2	USB-ASTB	A8	GND	J2	VDDO-3
A3	JA2	P6	LD3	G12	BTN0	F1	USB-DSTB	C1	GND	A5	VDDO-2
A10	JC3	P7	LD2	K14	AN2	G1	LD7	C7	GND	E12	VDDO-2
C9	JC4	M4	BTN2	J12	AN1	G3	SW4	C10	GND	K1	VDDO-2
B9	JC2	N4	LD5	J13	BLU2	H1	USB-DB6	E3	GND	P9	VDDO-2
A9	JC1	M5	LD0	J14	HSYNC	H2	USB-DB5	E14	GND	A11	VDDO-1
B8	MCLK	N5	LD4	H13	BLU1	H3	USB-DB7	G2	GND	D3	VDDO-1
C8	RCCLK	G14	GRN2	H12	CB	B14	TMS	H14	GND	D14	VDDO-1
A7	BTN3	G13	GRN1	J3	JA3	B13	TCK-FPGA	J1	GND	K2	VDDO-1
B7	JB4	F12	AN0	K3	SW2	A2	TDO-USB	K12	GND	L12	VDDO-1
P4	LD6	K13	VSYNC	B1	PS2C	A14	TDO-S3	M3	GND	P2	VDDO-1

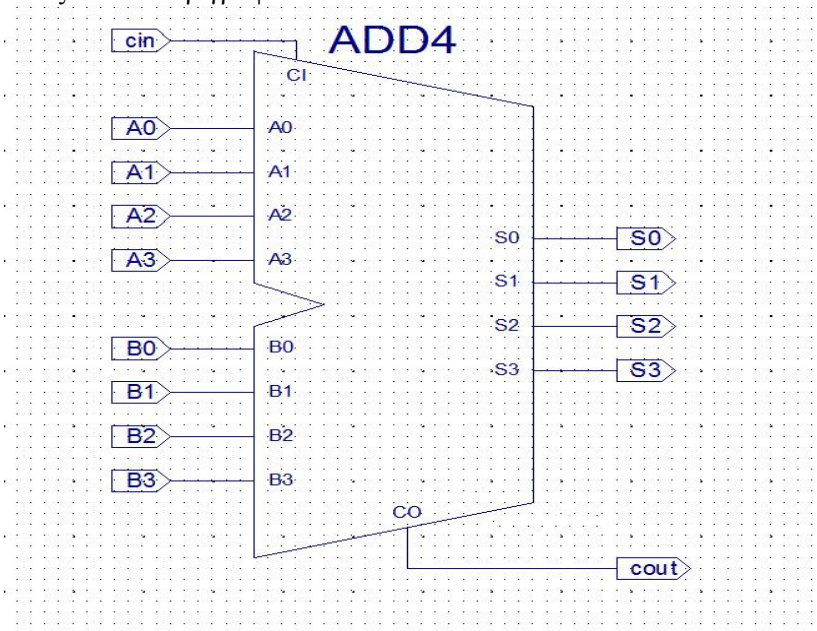
5.1.3 Οδηγίες για την εκτέλεση των ασκήσεων

Για την εκτέλεση καθεμιάς από τις παρακάτω ασκήσεις, ουσιαστικά απαιτούνται:

- Το αρχείο VHDL που περιγράφει το κύκλωμα που θέλουμε να υλοποιήσουμε και
- Το αρχείο ucf που ορίζει τη σύνδεση των pin της FPGA.

5.2 Αθροιστής τετραψήφιων δυαδικών αριθμών

Έστω ότι θέλουμε να υλοποιήσουμε το κύκλωμα άθροισης δύο τετραψήφιων δυαδικών αριθμών. Ονομάζουμε A και B τους δύο τετραψήφιους αριθμούς, S το αποτέλεσμα και Cin, Cout τα κρατούμενα κατώτερης και ανώτερης τάξης αντίστοιχα. Στην Εικόνα φαίνεται το σχηματικό διάγραμμα του κυκλώματος ενώ στη συνέχεια δίνεται ο VHDL κώδικας που το περιγράφει.



Εικόνα 5.4: Σχηματικό διάγραμμα αθροιστή τετραψήφιων δυαδικών αριθμών

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

  entity Full_adder is
    generic (width: integer:=4);
    port (
      A: in std_logic_vector((width-1) downto 0);
      B: in std_logic_vector((width-1) downto 0);
      cin: in std_logic;
      S: out std_logic_vector((width - 1) downto 0);
      cout: out std_logic
    );
  end Full_adder;

  architecture rtl of Full_adder is
    signal i_sum: std_logic_vector(width downto 0);
    constant i_cin_ext: std_logic_vector(width downto 1) := (others => '0');
  begin
    i_sum <= ('0' & A) + ('0' & B) + (i_cin_ext & cin);
    S<= i_sum((width-1) downto 0);
    cout <= i_sum(width);
  end rtl;
```

Μια πιθανή αντιστοιχία εισόδων-εξόδων είναι η ακόλουθη:

Είσοδοι	**Έξοδοι**
NET "A<0>" LOC = P11; NET "A<1>" LOC = L3;	NET "S<0>" LOC = M5; NET "S<1>" LOC = M11;

NET "A<2>" LOC = K3; NET "A<3>" LOC = B4; NET "B<0>" LOC = G3; NET "B<1>" LOC = F3; NET "B<2>" LOC = E2; NET "B<3>" LOC = N3; NET "cin" LOC = A7;	NET "S<2>" LOC = P7; NET "S<3>" LOC = P6; NET "cout" LOC = N5;

5.2.1 Πειραματικό μέρος:

1. Ανοίγωτηνεφαρμογή Xilinx ISE Design Suite
2. Ονομάζωτο Project ADDER_4
3. Επιλέγω Project>New Source>VHDL module με File name **Full_adder**
4. Γράφω τον κώδικα που δόθηκε παραπάνω σε γλώσσα VHDL
5. Στη συνέχεια πατάμε project>new source στο source type επιλέγουμε Implementation Constraints File και βάζουμε όνομα pin και πατάμε next
6. Σε αυτό το παράθυρο επιλέγουμε τις εισόδους και εξόδους του fpga board σύμφωνα με τους κωδικούς που έχει το Fpga που χρησιμοποιούμε στους διακόπτες και τα led
7. Πατάω διπλό κλικ στην επιλογή Generate Programming File και το πρόγραμμα μας δημιουργεί bit file.
8. Περνώ το Bit File στο FPGA με την εφαρμογή DIGILENT ADEPT SYSTEM .
9. Τέλος κάνουμε την επαλήθευση με το FpgaBoard για το Fulladder χρησιμοποιώντας για παράδειγμα τους παρακάτω συνδυασμούς.

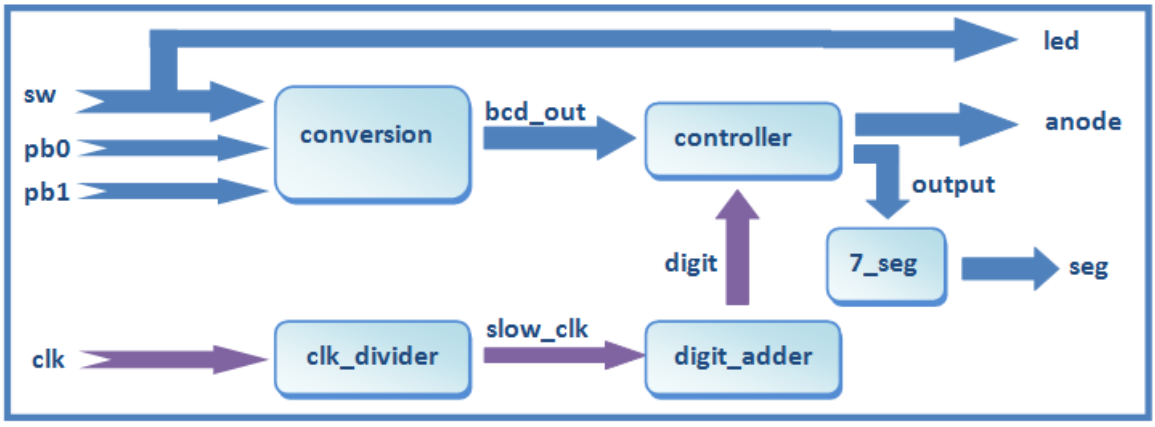
A3	A2	A1	A0	B3	B2	B1	B0	CI	S3	S2	S1	S0	CO
0	1	0	0	0	0	0	1	0	1	0	1	0	0
0	1	0	1	1	0	0	1	1	1	1	1	1	0
0	0	1	0	0	1	1	0	0	1	0	0	0	0
1	0	0	1	0	0	1	1	1	1	1	0	1	0

5.3 Μετατροπέας δυαδικών αριθμών σε δεκαδικούς, οκταδικούς και δεκαεξαδικούς

Θα δημιουργήσουμε ένα μετατροπέα δυαδικών αριθμών σε δεκαδικούς, οκταδικούς και δεκαεξαδικούς. Θα χρησιμοποιήσουμε τα οκτώ switches της πλακέτας σαν παράλληλη είσοδο ενός 8-bit δυαδικού αριθμού και την οθόνη LCD για την εμφάνιση του αριθμού. Με δύο από τα μπουτόν της πλακέτας θα επιλέγουμε τη μετατροπή που επιθυμούμε.

Το συγκεκριμένο κύκλωμα επιλέχθηκε γιατί τέτοιου είδους μετατροπείς χρησιμοποιούνται σε πάρα πολλές διαφορετικές εφαρμογές (τηλεπικοινωνίες, οικιακές συσκευές, αυτοκίνητα, βιομηχανικοί αυτοματισμοί κ.α.). Επίσης μας δίνει τη δυνατότητα να χρησιμοποιήσουμε την οθόνη LCD της πλακέτας και να συμπεριλάβουμε ένα τυπικό κύκλωμα χρονισμού για seven-segmentdisplay.

Η δημιουργία ενός λογικού διαγράμματος κρίνεται απαραίτητη σαν πρώτο βήμα στο σχεδιασμό ενός πολύπλοκου κυκλώματος, αλλά και για απλά κυκλώματα μας βοηθά να μειώσουμε τα σφάλματα της σχεδίασης. Πρώτα πρέπει να ορίσουμε τις εισόδους και εξόδους του κυκλώματος. Σε αυτό το κύκλωμα λοιπόν, οι είσοδοι είναι : τα οκτώ switches, τα δύο push-button για την επιλογή μετατροπέα (pb0, pb1) και ένας παλμός ρολογιού για το σύστημα (clk). Οι έξοδοι μας θα είναι : τα οκτώ led που θα ανάβουν πάνω από κάθε switch διευκολύνοντας την ανάγνωση του δυαδικού αριθμού, οι τέσσερεις κοινές άνοδοι των τεσσάρων ψηφίων (anode) και οι οκτώ κοινές κάθοδοι των τομέων των ψηφίων (seg).



Εικόνα 5.5: Το Λογικό Διάγραμμα του κυκλώματος μας

Όπως βλέπουμε στο διάγραμμα, η κατάσταση των οκτώ switches μεταφέρεται στα οκτώ αντίστοιχα leds, και παράλληλα και στη διεργασία ‘conversion’ όπου γίνεται η μετατροπή, ενώ με τα μπουτόν pb0 και pb1 επιλέγουμε το είδος της μετατροπής (δεκαδικό/δεκαεξαδικό/οκταδικό). Το αποτέλεσμα της μετατροπής φορτώνεται στο σήμα ‘bcd_out’ και μεταφέρεται στον ‘controller’.

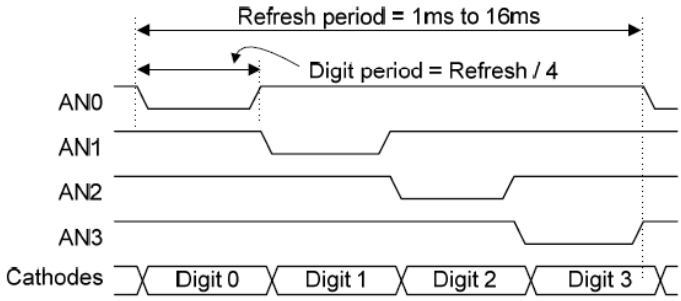
Για τη μετατροπή στο δεκαδικό σύστημα, θα χρησιμοποιήσουμε τον μαθηματικό αλγόριθμο “double-dabble”, (του οποίου η λειτουργία φαίνεται στην Εικόνα 5.6) τον οποίο θα περιγράψουμε στη γλώσσα VHDL για να πάρει ύστερα τη μορφή λογικού κυκλώματος.

```
0000 0000 0000 11110011      Initialization
0000 0000 0001 11100110      Shift
0000 0000 0011 11001100      Shift
0000 0000 0111 10011000      Shift
0000 0000 1010 10011000      Add 3 to ONES, since it was 7
0000 0001 0101 00110000      Shift
0000 0001 1000 00110000      Add 3 to ONES, since it was 5
0000 0011 0000 01100000      Shift
0000 0110 0000 11000000      Shift
0000 1001 0000 11000000      Add 3 to TENS, since it was 6
0001 0010 0001 10000000      Shift
0010 0100 0011 00000000      Shift
```

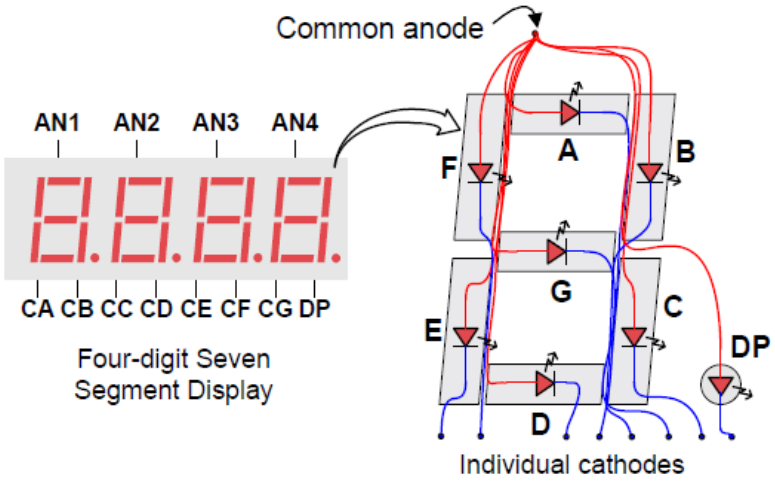
Εικόνα 5.6: Ένα παράδειγμα μετατροπής αριθμού με τον αλγόριθμο 'doubledouble'.

Για να περιορίσουμε τα pin της οθόνης LCD, το Basys2 χρησιμοποιεί πολύπλεξη (από 32 pin σε 12). Υπάρχουν τέσσερεις κοινοί άνοδοι για τα τέσσερα ψηφία και επτά κοινοί κάθοδοι για τους επτά τομείς των ψηφίων (η υποδιαστολή είναι ο όγδοος τομέας). Η διαδικασία ‘controller’ είναι ο πολυπλέκτης του κυκλώματος μας και η διαδικασία ‘7_seg’ ανάβει τους σωστούς τομείς του display ώστε να σχηματιστεί ο αριθμός που βλέπουμε.

Για το ρολόι του κυκλώματος, η πλακέτα Basys2 διαθέτει μια παλμογεννήτρια 50 MHz την οποία μπορούμε να χρησιμοποιήσουμε. Πρώτα όμως πρέπει να υποβιβάσουμε τη συχνότητα αυτή περίπου στα 60 Hz, που είναι μια καλή συχνότητα σάρωσης για την οθόνη LCD. Δηλαδή, τα τέσσερα ψηφία της οθόνης θα ανάβουν διαδοχικά με ταχύτητα σάρωσης εξήντα φορές το δευτερόλεπτο (με τα οκτώ bit μπορούμε να εκφράσουμε τους αριθμούς από 0 έως το 255, αλλά θα χρησιμοποιήσουμε και τα τέσσερα ψηφία της οθόνης προσθέτοντας στο κύκλωμα μας επεκτασιμότητα). Τη διαδοχή αυτή θα αναθέσουμε στο σήμα ‘digit’, ενώ στο σήμα ‘slow_clk’ τον υποβιβασμένο παλμό του ρολογιού.



Εικόνα 5.7: Η πολύπλεξη των ανόδων του display.



Εικόνα 5.8: Οι συνδέσεις του 7-segmentdisplay

```
libraryIEEE;                                        -- ΟΙ ΒΙΒΛΙΟΘΗΚΕΣ ΠΟΥ
useIEEE.STD_LOGIC_1164.ALL;                         -- ΘΑ ΧΡΗΣΙΜΟΠΟΙΗΣΟΥΜΕ
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entityCONVERTORis                                   --ΟΙ ΕΙΣΟΔΟΙ ΚΑΙ ΕΞΟΔΟΙ ΤΟΥ ΚΥΚΛΩΜΑΤΟΣ
Port ( sw : in  std_logic_vector(7 DOWNTO 0) ;
          clk : in  std_logic;
          btn0, btn1 : in  std_logic;
          led : out  std_logic_vector(7 DOWNTO 0) ;
          seg : out  std_logic_vector(7 DOWNTO 0) ;
          anode : out  std_logic_vector(3 DOWNTO 0));
end CONVERTOR;

architecture Behavioral of CONVERTOR is             --ΤΑΣΗΜΑΤΑΕΝΤΟΣΤΟΥΚΥΚΛΩΜΑΤΟΣ
signal bcd_out : std_logic_vector(11 downto 0);
signal slow_clk : std_logic;
signal digit : std_logic_vector(1 downto 0);
signal output : std_logic_vector(3 downto 0);
signal count : std_logic_vector(17 downto 0);
BEGIN

led<= sw;                                           -- ΤΑ LED ΑΝΑΒΟΥΝ ΜΕ ΤΑ SWITCHES
seg(0) <= '1';                                      -- ΗΥΠΟΔΙΑΣΤΟΛΗΣΒΗΣΤΗ

CONVERSION : process (sw, btn0) is
          variable bcd : std_logic_vector(11 downto 0) := (others => '0');
          variable bin : std_logic_vector (7 DOWNTO 0) ;
          variable i : integer range 0 to 7 :=0 ;
begin
```

```
if (btn0 = '1' and btn1 = '0')' then                    -- ΣΕΔΕΚΑΕΞΑΔΙΚΟ
        bcd_out(7 downto 0) <= sw;
        bcd_out(11 downto 8) <= "0000";
elsif (btn0 = '0' and btn1 = '1') then                  -- ΣΕΟΚΤΑΔΙΚΟ
        bcd_out(2 downto 0) <= sw(2 downto 0);
        bcd_out(3) <= '0';
        bcd_out(6 downto 4) <= sw(5 downto 3);
        bcd_out(7) <= '0';
        bcd_out(9 downto 8) <= sw(7 downto 6);
        bcd_out(11 downto 10) <= "00";
else
        bin := sw;
        bcd := x"000";
        for i in 0 to 7 loop                            -- ΟΑΛΓΟΡΙΘΜΟΣ 'DOUBLE-DABBLE'
        bcd(11 downto 1) := bcd(10 downto 0);
         bcd(0) := bin(7);
        bin(7 downto 1) := bin(6 downto 0);
        bin(0) :='0';
        if(i < 7 and bcd(3 downto 0) > "0100") then
                bcd(3 downto 0) := bcd(3 downto 0) + "0011";
        end if;
        if(i < 7 and bcd(7 downto 4) > "0100") then
                bcd(7 downto 4) := bcd(7 downto 4) + "0011";
        end if;
        f(i < 7 and bcd(11 downto 8) > "0100") then
                bcd(11 downto 8) := bcd(11 downto 8) + "0011";
        end if;
        end loop;
        bcd_out <= bcd;
end if;
end process;

clk_divider : process (clk, count)
begin
        if (clk'event and clk = '1') then
          count <= count + 1;
        end if;
        slow_clk <= count(17);                          -- ΔΙΑΙΡΕΣΗΚΑΤΑ 2^17
end process;

digit_adder : process (slow_clk)
begin
        if (slow_clk'event and slow_clk = '1') then
                digit <= digit + 1;                     -- ΕΜΦΑΝΙΖΕΤΑΙ ΤΟ ΕΠΟΜΕΝΟ ΨΗΦΙΟ
        end if;
end process;

controller : process (digit, bcd_out, btn0)
begin
case digit is
        when "00" =>
                output <= bcd_out(3 downto 0);
                anode <= "1110";
        when "01" =>
                output <= bcd_out(7 downto 4);
                if (btn0 = '0' and bcd_out < 10)
                or (btn0 = '1' and bcd_out < 16)
                then anode <= "1111";
                else anode <= "1101";
```

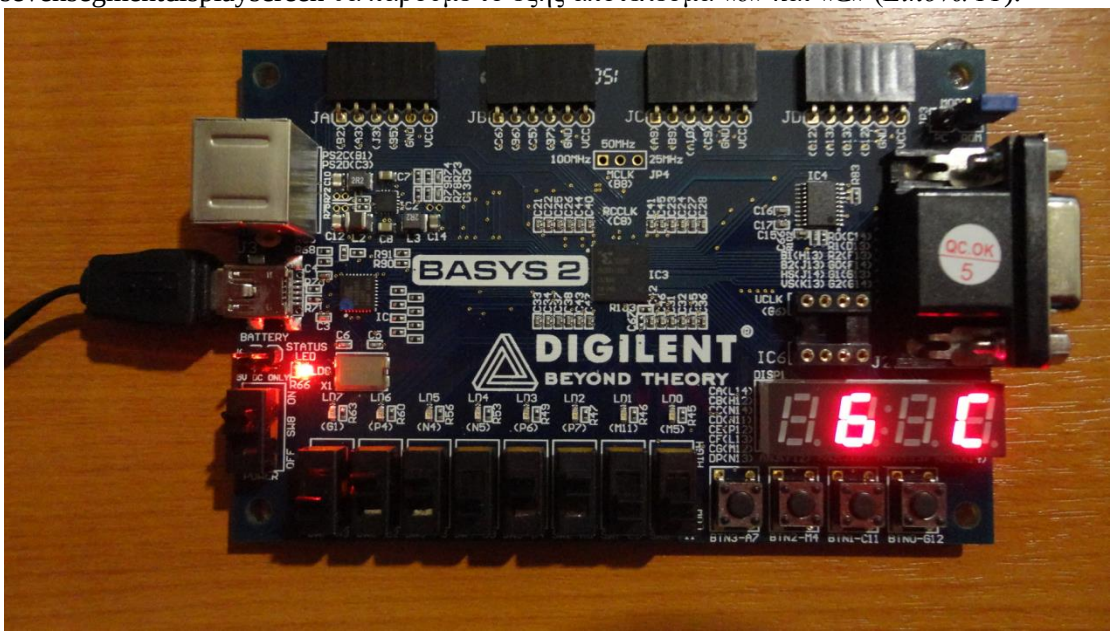
```
                end if;
        when "10" =>
                output <= bcd_out(11 downto 8);
                if output = 0 then
                anode <= "1111";
                else anode <= "1011";
                end if;
        when others =>
                anode <= "1111";
                output <= "0000";
        end case;
end process;

with output select
seg(7 downto 1) <= "0000001" when "0000",       --  0
"1001111" when "0001",                          -- 1
"0010010" when "0010",                          -- 2
"0000110" when "0011",                          -- 3
"1001100" when "0100",                          -- 4
"0100100" when "0101",                          -- 5
"0100000" when "0110",                          -- 6
"0001111" when "0111",                          -- 7
"0000000" when "1000",                          -- 8
"0000100" when "1001",                          -- 9
"0001000" when "1010",                          -- A
"1100000" when "1011",                          -- b
"0110001" when "1100",                          -- C
"1000010" when "1101",                          -- d
"0110000" when "1110",                          -- E
"0111000" when "1111",                          -- F
"1111110" when others;                          -- -  (error)

ENDBEHAVIORAL;
```

Στο αρχείο .ucf (userconstraintsfile) δηλώνουμε σε ποια φυσικά pin του FPGA αντιστοιχούν οι είσοδοι και έξοδοι του κυκλώματος μας. Οι διευθύνσεις των pin δίνονται στα συνοδευτικά manual των FPGA και είναι της μορφής “G11”, “K13”, κλπ.

Είσοδοι	**Έξοδοι**
NET "sw<0>" LOC= "P11"; NET "sw<1>" LOC= "L3"; NET "sw<2>" LOC= "K3"; NET "sw<3>" LOC= "B4"; NET "sw<4>" LOC= "G3"; NET "sw<5>" LOC= "F3"; NET "sw<6>" LOC= "E2"; NET "sw<7>" LOC= "N3"; NET "btn0" LOC= "A7"; NET "btn1" LOC= "M4"; NET "clk" LOC= "B8";	NET "led<0>" LOC= "M5"; NET "led<1>" LOC= "M11"; NET "led<2>" LOC= "P7"; NET "led<3>" LOC= "P6"; NET "led<4>" LOC= "N5"; NET "led<5>" LOC= "N4"; NET "led<6>" LOC= "P4"; NET "led<7>" LOC= "G1"; NET "seg<0>" LOC= "N13"; NET "seg<1>" LOC= "M12"; NET "seg<2>" LOC= "L13"; NET "seg<3>" LOC= "P12"; NET "seg<4>" LOC= "N11"; NET "seg<5>" LOC= "N14"; NET "seg<6>" LOC= "H12"; NET "seg<7>" LOC= "L14"; NET "anode<0>" LOC= "F12"; NET "anode<1>" LOC= "J12"; NET "anode<2>" LOC= "M13";

	NET "anode<3>" LOC= "K14";

Για την επαλήθευση, μπορούμε να δοκιμάσουμε συνδυασμούς των switches. Παραδείγματος χάριν όταν τα 8 switches είναι στη θέση «0» έχουμε τους δυαδικούς αριθμούς των 4 bit «0000» και «0000», οι οποίοι μεταφράζονται στο δεκαεξαδικό σύστημα (*sevensegmentdisplayscreen*) ως εξής «0» και «0». Ένα άλλο παράδειγμα είναι, όταν τα switches έχουν την εξής δυαδική μορφή «0110» «1100» στο sevensegmentdisplayscreen θα πάρουμε το εξής αποτέλεσμα «6» και «C» (*Εικόνα 11*).

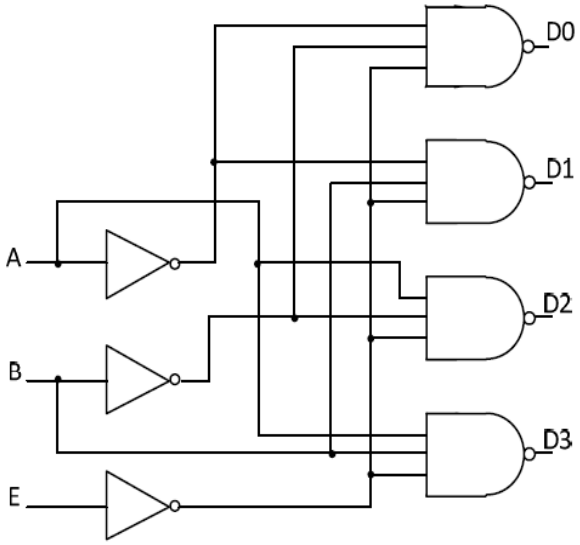


Εικόνα 5.9: Παράδειγμα επαλήθευσης του μετατροπέα

5.4 Αποκωδικοποιητής 2 σε 4

Οι διακριτές ποσότητες πληροφορίας παριστάνονται στα ψηφιακά συστήματα με τη χρήση των δυαδικών κωδικών. Ένας δυαδικός κώδικάς με n μπιτ μπορεί να παραστήσει μέχρι και 2^n διακριτά στοιχεία κωδικοποιημένης πληροφορίας. Ο αποκωδικοποιητής είναι ένα συνδυαστικό κύκλωμα, το οποίο μετατρέπει δυαδικές πληροφορίες που βρίσκονται σε n γραμμές εξόδου μέγιστου πλήθους 2^n. Εάν ο αντίστοιχος κώδικας n μπιτ της πληροφορίας που κωδικοποιούμε έχει αχρησιμοποίητους συνδυασμούς, ο αποκωδικοποιητής μπορεί να έχει λιγότερες από 2^n εξόδους. Όταν όμως η είσοδος επίτρεψης είναι ανενεργή, τότε όλες οι έξοδοι είναι ανενεργές. Για κάθε συνδυασμό των εισόδων, ο αποκωδικοποιητής επιλέγει μία από τις 2^n εξόδους και τη φέρνει σε λογικό 1, ενώ οι υπόλοιπες παραμένουν σε λογικό μηδέν. Επίσης οι αποκωδικοποιητές φέρνουν συχνά και είσοδο «επίτρεψης» (enable), ώστε όταν η είσοδος επίτρεψης είναι ενεργή τότε ο αποκωδικοποιητής επιλέγει όλες οι έξοδοι είναι ανενεργές.

Ένας αποκωδικοποιητής 2 σε 4 φαίνεται στο παρακάτω **Λογικό Διάγραμμα 4**. Οι δύο είσοδοι αποκωδικοποιούνται σε τέσσερις εξόδους, ώστε για καθέναν από τους τέσσερις συνδυασμούς των εισόδων, μόνο μία έξοδος είναι ενεργή. Η κάθε έξοδος αντιπροσωπεύει έναν από τους ελάχιστους όρους (minterms) των n μεταβλητών εισόδου. Για παράδειγμα όταν οι είσοδοι x,y είναι 00, τότε ο ελάχιστος όρος που παράγεται είναι ο x',y', στην έξοδο D0. Ο αντίστοιχος Πίνακας Αληθείας φαίνεται παρακάτω.



Εικόνα 5.10: Το κύκλωμα του αποκωδικοποιητή

Ο πίνακας αληθείας του αποκωδικοποιητή είναι:

E	B	A	D0	D1	D2	D3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Γενικά ένας αποκωδικοποιητής μπορεί να λειτουργήσει είτε με συμπληρωμένες εξόδους είτε με ασυμπλήρωτες εξόδους. Η είσοδος επίτρεψης μπορεί να ενεργοποιηθεί με ένα σήμα 0 ή 1. Μερικοί αποκωδικοποιητές έχουν δύο ή περισσότερες εισόδους επίτρεψης, οι οποίες πρέπει να ικανοποιούν μία λογική συνθήκη, προκειμένου να επιτραπεί η λειτουργία του κυκλώματος.

Οι αποκωδικοποιητές με εισόδους επίτρεψης μπορούν να συνδεθούν μεταξύ τους, έτσι ώστε να δημιουργήσουν ένα μεγαλύτερο κύκλωμα αποκωδικοποιητή.

Ο κώδικας VHDL που περιγράφει τον αποκωδικοποιητή ακολουθεί:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entitydec24dis
PORT(A,B,EN_L:IN BIT;
Q0,Q1,Q2,Q3:OUT BIT);
enddec24d;

ARCHITECTURE dataflow OF dec24d IS
Begin
Q0<=NOT((NOT A)AND (NOT B) AND (NOT EN_L));
Q1<=NOT ((NOT A)AND (B) AND (NOT EN_L));
Q2<=NOT (( A)AND (NOT B) AND (NOT EN_L));
Q3<=NOT((A)AND (B) AND (NOT EN_L));
ENDdataflow;
```

Και σε αυτό το σημείο επιλέγουμε τις εισόδους και εξόδους του fpga board σύμφωνα με τους κωδικούς που έχει το fpga που χρησιμοποιούμε στους διακόπτες και τα led.

Είσοδοι	Έξοδοι
NET "A" **loc**= "E2"; **NET** "B" **loc**= "F3"; **NET** "EN_L" **loc**= "N3";	**NET** "Q0" **loc**= "P6"; **NET** "Q1" **loc**= "P7"; **NET** "Q2" **loc**= "M11"; **NET** "Q3" **loc**= "M5";

5.5 Συγκριτής αριθμών 4-bit

Έστω ότι θέλουμε να συγκρίνουμε δύο τετραψήφιους δυαδικούς αριθμούς A (=$A_3A_2A_1A_0$) και B(=$B_3B_2B_1B_0$). Τα ενδεχόμενα είναι τρία:

- **Να είναι ίσοι**, A=B δηλαδή A_3= B_3 και A_2= B_2καιA_1= B_1 και A_0= B_0. Επειδή αναφερόμαστε σε δυαδικά κυκλώματα, ο αριθμός που μπορεί να πάρει το κάθε ψηφίο είναι 0 ή 1. Άρα η σχέση ισότητας για το καθένα εκφράζεται λογικά με την συνάρτηση XNOR (αποκλειστικού ΟΥΤΕ). Έστω η σχέση X_n για κάθε ζευγάρι ψηφίων A_n, B_n, $X_n=A_nB_n+A'_nB'_n$. Xn=1 μόνο αν τα αντίστοιχα bit της θέσης n είναι ίσα, άρα καταλαβαίνουμε ότι για να ισχύει η συνθήκη ισότητας πρέπει όλες οι μεταβλητές Xn να είναι ίσες με 1. Ονομάζουμε την συνάρτηση που προκύπτει από το παραπάνω συμπέρασμα **F_equal** και αποτελείται από την λογική συνθήκη AND (ΚΑΙ) για όλες τις μεταβλητές, $F\_equal=X_3X_2X_1X_0$
- **Α Μεγαλύτερος από το Β** (A>B). Όταν οι αριθμοί δεν είναι ίσοι, ελέγχουμε το κάθε ψηφίο ξεκινώντας από την πιο σημαντική θέση. Εάν τα ψηφία είναι ίσα, τότε ελέγχουμε το αμέσως επόμενο ζευγάρι. Αν το αντίστοιχο ψηφίο του Α είναι ίσο με 1 και του Β με 0, τότε καταλαβαίνουμε ότι A>B. Η λογική συνάρτηση την οποία ονομάζουμε F_largest είναι: $F\_largest=A_3B_3+A_2B'_2X_3+A_1B'_1X_3X_2+A_0B'_0X_3X_2X_1$
- **Α Μικρότερος από το Β** (A<B). Ισχύει η ίδια μεθοδολογία για το συμπέρασμα αυτό, με την διαφορά ότι το ζεύγος ψηφίων με την μεγαλύτερη σημαντική θέση που θα παρουσιάσει ανισότητα να έχει το A=0 και το B=1. Η λογική συνάρτηση την οποία ονομάζουμε F_lower είναι: $F\_lower=A'_3B_3+A'_2B_2X_3+A'_1B_1X_3X_2+A'_0B_0X_3X_2X_1$

Συνοψίζοντας , η υλοποίηση των Λογικών Διαγραμμάτων του συγκριτή μεγέθους με πύλες για τις συναρτήσεις **F_equal**, **F_largest** και **F_lower** που ανάλογα το αποτέλεσμά τους (0 ή 1) υποδηλώνουν και το αποτέλεσμα, φαίνονται ξεχωριστά για την κάθε μια στα παρακάτω σχήματα.
Ο κώδικας για την υλοποίηση του Συγκριτή Μεγέθους φαίνεται στη συνέχεια:

```
library IEEE;
use ieee.std_logic_1164.all;

entity comparator is
port (A,B: in std_logic_vector(3 downto 0);
F_equal,F_largest,F_lower: out bit);
endcomparator;

architecture Behavioral of comparator is
signal x_n:std_logic_vector(3 downto 0);
begin
```

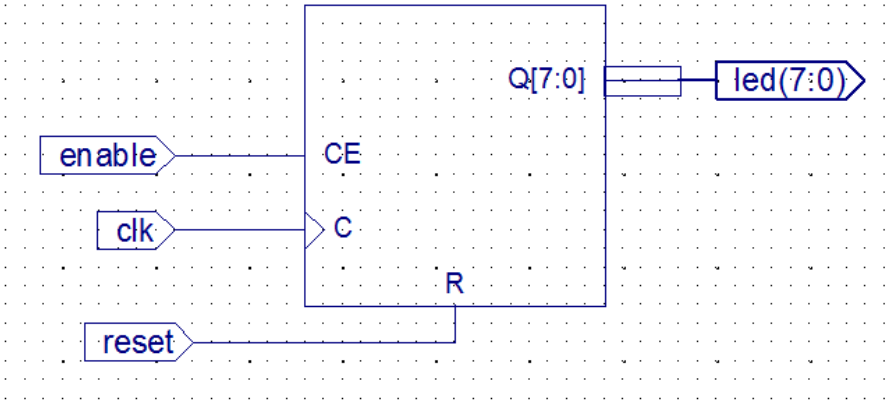
```
process(A,B)
begin

x_n<=A xnor B;
If (x_n(0)and x_n(1)and x_n(2)and x_n(3))='1' then
F_equal<='1';
else
F_equal<='0';
end if;
If ((not B(3) and A(3)) or (not B(2) and A(2) and x_n(3)) or (not B(1) and A(1) and X_n(3) and X_n(2)) or
(not B(0) and A(0) and X_n(1)and X_n(3) and X_n(2)))='1' then
F_largest<='1';
else
F_largest<='0';
endif;
If ((not A(3) and B(3)) or (not A(2) and B(2) and x_n(3)) or (not A(1) and B(1) and X_n(3) and X_n(2)) or
(not A(0) and B(0) and X_n(1)and X_n(2) and X_n(3)))='1' then
F_lower<='1';
else
F_lower<='0';
end if;

end process;
end Behavioral;
```

Ο ορισμός των εισόδων και εξόδων της κάρτας FPGA θα γίνει χρησιμοποιώντας τις ακόλουθες αντιστοιχίσεις στο αρχείο ucf.

Είσοδοι	Έξοδοι
NET "A<0>" **LOC**= "G3"; **NET** "A<1>" **LOC**= "F3"; **NET** "A<2>" **LOC**= "E2"; **NET** "A<3>" **LOC**= "N3"; **NET** "B<0>" **LOC**= "P11"; **NET** "B<1>" **LOC**= "L3"; **NET** "B<2>" **LOC**= "K3"; **NET** "B<3>" **LOC**= "B4";	**NET** "F_EQUAL" **LOC**= "G1"; **NET** "F_LARGEST" **LOC**= "P4"; **NET** "F_LOWER" **LOC**= "N4";

5.6 Μετρητής 8 δυαδικών ψηφίων με επίτρεψη

Η άσκηση περιλαμβάνει δυαδικό μετρητή 8 bit που μετράει κυκλικά από την τιμή <<00000000>> έως την τιμή <<11111111>>. Ο μετρητής αρχίζει να μετράει όταν ο διακόπτης από την θέση 0 βρεθεί στη θέση 1 και θα συνεχίσει να μετράει έως τον πάμε στη θέση 0. Το κύκλωμα του μετρητή απεικονίζεται στην Εικόνα, ενώ ο κώδικας VHDL που τον περιγράφει δίνεται στη συνέχεια.



Εικόνα 5.11: Σχηματικό διάγραμμα μετρητή με enable

```
library ieee;
     use ieee.std_logic_1164.all;
   use ieee.std_logic_unsigned.all;

 entity Counter_8bit  is
 port (
     led   :out std_logic_vector (7 downto 0); -- Output of the counter
     enable :in  std_logic;                    -- Enable counting
     clk    :in  std_logic;                   -- Input clock
     reset  :in  std_logic                    -- Input reset
 );
 end entity;

 architecture rtl of Counter_8bit  is
     signal count :std_logic_vector (7 downto 0);
 begin
     process (clk, reset) begin
         if (reset = '1') then
             count <= (others=>'0');
 elsif (rising_edge(clk)) then
             if (enable = '1') then
                 count <= count + 1;
             end if;
end if;
end process;
   led<= count;
endarchitecture;
```

Για την υλοποίηση, χρησιμοποιούμε τις αντιστοιχίες εισόδων εξόδων του Fpga που φαίνονται παρακάτω:

Είσοδοι	Εξοδοι
NET "Clk" LOC = C8; NET "reset" LOC = A7; NET "enable" LOC = P11;	NET "Led<0>" LOC = M5; NET "Led<1>" LOC = M11; NET "Led<2>" LOC = P7; NET "Led<3>" LOC = P6; NET "Led<4>" LOC = N5; NET "Led<5>" LOC = N4; NET "Led<6>" LOC = P4; NET "Led<7>" LOC = G1;

5.7 Μηχανή καταστάσεων

Για να πειραματιστείτε με τις μηχανές καταστάσεων σε FPGA, χρησιμοποιήστε τον κώδικα VHDL που δόθηκε στην παράγραφο 4.7 και τροποποιήστε τον ώστε να βγαίνει στην έξοδο του κυκλώματος η παρούσα κατάσταση curr_st. Υπόδειξη: ονομάστε το σήμα εξόδου curr_st_ext. Μπορείτε να χρησιμοποιείσετε την ακόλουθη αντιστοίχιση pin.

Είσοδοι	**Έξοδοι**
NET "Clk" LOC = C8; NET "rst" LOC = A7; NET "sel" LOC = P11;	NET "curr_st_ext<0>" LOC = M5; NET " curr_st_ext <1>" LOC = M11;

5.8 Κύκλωμα εξαγωγής πεδίων από πακέτο πληροφορίας

Στην παράγραφο 4.6 έχει περιγραφεί με χρήση της γλώσσας VHDL, ένα κύκλωμα εξαγωγής πεδίων από πακέτο πληροφορίας. Έστω ότι θέλουμε να χρησιμοποιήσουμε τον κώδικα που δίνεται σε εκείνη την παράγραφο για να προγραμματίσουμε το FPGA του Basys2.

Χρειάζεται επιπλέον να προσδιορίσουμε τα pin που θα χρησιμοποιήσουμε. Αν λάβουμε υπόψιν μας ότι τα σήματα εισόδου του κυκλώματος είναι τα ακόλουθα:

- clk, rst, data_valid: in std_logic;
- data_in: in std_logic_vector(7 downto 0);

να απαντήσετε στις ερωτήσεις:

1. αρκούν τα διαθέσιμα pin εισόδου του basys2 για να χειριστούμε το συγκεκριμένο κύκλωμα; Αν ναι, δώστε τις αντιστοιχίσεις.
2. Αν τα pin εισόδου δεν αρκούν, δηλαδή τα pin εισόδου είναι λιγότερα από τα σήματα εισόδου του κυκλώματος, ποιες είναι οι πιθανές λύσεις;

5.9 Τεστ αυτοαξιολόγησης

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 5.12

1. Για να προγραμματίσουμε ένα FPGA ώστε να υλοποιεί ένα συγκεκριμένο κύκλωμα, χρειαζόμαστε:
    a. Ένα αρχείο VHDL μόνο
    b. Ένα αρχείο ucf μόνο
    c. Ένα αρχείο pdf μόνο
    d. Ένα αρχείο vhdl και ένα αρχείο ucf
2. Η διαδικασία της σύνθεσης προηγείται του placementandrouting.
3. Για να επαληθεύσουμε την ορθή λειτουργία ενός FPGA, ο μόνος τρόπος είναι να παρακολουθούμε τα sevensegmentdisplay.
4. Για να προγραμματίσουμε ένα FPGA, αρκεί να έχουμε εγκαταστήσει στον υπολογιστή μας ένα πρόγραμμα προσομοίωσης όπως το ModelSim.
5. Ο μόνος τρόπος να σχεδιάσουμε ένα κύκλωμα και να προγραμματίσουμε ένα FPGA, είναι να το περιγράψουμε χρησιμοποιώντας τη γλώσσα VHDL.
6. Το κύκλωμα που υλοποιείται σε FPGA δεν έχει ακριβώς πάντα την ίδια συμπεριφορά με αυτή που παρουσιάζει το ModelSim όταν χρησιμοποιούμε RTL κώδικα γιατί η προσομοίωση του RTL κώδικα δεν λαμβάνει υπόψιν την καθυστέρηση που παρατηρείται στη διάδοση των λογικών τιμών μέσα στο ολοκληρωμένο και στην αλλαγή της εξόδου των πυλών και flip-flop η οποία μπορεί να είναι καθοριστική για υψηλές τιμές συχνότητας ρολογιών.

5.10 Βιβλιογραφία

On-line μαθήματα σε ASIC/VHDL design:

- http://www.dacafe.com/ASICs.htm
- http://www.ecs.umass.edu/ece/vspgroup/burleson/courses/558/
- http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.vhd/html/vhdl-all-e.html
- http://www.eas.asu.edu/~yong598d/VHDL_links.html

Tutorials/Papers:

- http://www.vhdl.org/fmf/wwwpages/FMF_ECL_models_paper.html
- http://www.bluepc.com/download.html#downloadtop
- http://www.symphonyeda.com/products.htm
- http://www.angelfire.com/electronic/in/vlsi/vhdl.html#vhdl

Άλλες πηγές:

- http://www.ieee.org
- http://www.acm.org
- http://www.vhdl.org
- http://tech-www.informatik.uni-hamburg.de/vhdl/
- http://www.eeglossary.com/vhdl.htm
- http://www.ent.ohiou.edu/~starzyk/network/Class/ee514/intro.html

Παράρτημα - Παραδείγματα κωδίκων VHDL

Field_Locator.vhd

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

entity field_locator is
   port(clk, rst, data_valid: in std_logic; field_valid: out std_logic);
end field_locator;

architecture bhv of field_locator is
       signal counter : std_logic_vector(7 downto 0) ;
       signal fval: std_logic;
begin
   process(clk, rst)
   begin
           if (rst='1') then
           fval<= '0';
             counter <= (others=>'1');
elsif (clk'EVENT AND clk = '1') then
           if (data_valid = '1') then
           counter <= counter + 1;
           end if;
           if (counter = 4) then
                   fval<= '1';
                           counter <= (others=>'0');
           else
                   fval<= '0';
           end if;
       end if;

   end process;
field_valid<= fval;

end architecture bhv;
```

Fsm.vhd

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


library work;
  use work.all;


entity fsm is

      port (
clk: in std_logic;
rst: in std_logic;
            x: in std_logic;
sel: in std_logic;
            outp1: out std_logic_vector(5 downto 0);
            outp2: out std_logic_vector(5 downto 0)
                  );
end fsm ;


architecture rtl of fsm is

signal clki: std_logic;
signal reset: std_logic;
signal xi: std_logic;
signal seli: std_logic;
signal cnt1: std_logic_vector(5 downto 0):="000000";
signal cnt2: std_logic_vector(5 downto 0):="000000";
signal cur_state, nxt_state: std_logic_vector(1 downto 0):="00";

begin
clki<=clk;
xi<=x;
seli<=sel;
reset<=rst;

process (clki, reset)
 begin
   if reset='1' then
          cnt1<=(others=>'0');
          cnt2<=(others=>'0');
          cur_state<=(others=>'0');
   else
          if (clki'event and clki = '1') then
                cur_state<=nxt_state;
                case cur_state is
                when "10" =>
                      cnt1<=cnt1+1;
                      cnt2<=cnt2;
```

```
                    when "11" =>
                            cnt1<=cnt1;
cnt2<=cnt2+1;
                    when others =>
                            cnt1<=cnt1;
                            cnt2<=cnt2;
                    end case;
            end if;
    end if;
 end process;


ctrl_fsm: process (xi, seli, cur_state)
 begin
            case cur_state is
            when "00" =>
                    if (xi='0') then
                            nxt_state<= "00";
                    else
                            nxt_state<= "01";
                    end if;
            when "01" =>
                    if (seli='1') then
                            nxt_state<= "10";
                    else
                            nxt_state<= "11";
                    end if;
            when "10" =>
                    nxt_state<= "00";
            when "11" =>
                    nxt_state<= "00";
            when others =>
                    nxt_state<= "00";
        end case;

 end process;

outp1 <= cnt1;
outp2 <= cnt2;
end rtl ;
```

Fsm_TB/vhd

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

library work;
   use work.all;

entity field_filtering is
   port(clk, rst, data_valid: in std_logic;
          field_valid_out: out std_logic;
data_in: in std_logic_vector(7 downto 0);
       field: out std_logic_vector(7 downto 0));
end field_filtering;

architecture structural of field_filtering is
   component field_locator is
      port(clk, rst, data_valid: in std_logic; field_valid: out std_logic);
   end component;

   component register8b is
      port(
          clk, rst, field_valid: in std_logic;
data_in: in std_logic_vector(7 downto 0);
          field: out std_logic_vector(7 downto 0)
          );
   end component;

   signal field_valid_i, field_valid_d: std_logic;

begin
field_locator_inst: field_locator port map(clk, rst, data_valid, field_valid_i);
   register8b_inst: register8b port map(clk, rst, field_valid_i, data_in, field);

   process(clk)
   begin
      if (clk'EVENT AND clk = '1') then
          field_valid_d<= field_valid_i;
      end if;
   end process;
field_valid_out<= field_valid_d;

end architecture structural;
```

Register8b.vhd

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

entity register8b is
   port(clk, rst, field_valid: in std_logic;
data_in: in std_logic_vector(7 downto 0);
        field: out std_logic_vector(7 downto 0));
end register8b;

architecture bhv of register8b is
signal temp_data : std_logic_vector(7 downto 0);
begin
   process(clk,rst)
   begin
      if (rst = '1') then
         field <= (others=>'0');
temp_data<= (others=>'0');
elsif (clk'EVENT AND clk = '1') then
temp_data<= data_in;

         if (field_valid = '1') then
          field <= temp_data;
         end if;
end if;
   end process;
end architecture bhv;
```

AND_gate.vhd

```
library IEEE;
  use IEEE.std_logic_1164.all;
  USE ieee.std_logic_arith.all ;
  USE ieee.std_logic_unsigned.all ;
  use ieee.std_logic_textio.all;   -- in order to use hread( )
library work;
use work.all;


entity AND_gate is

      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );
end AND_gate ;



architecture rtl of AND_gate is

begin
process (x, y)
 begin
   z<=x AND y;
end process;

end rtl ;
```

AND_gate_TB.vhd

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

library work;
use work.all;


--------------------------------
entity AND_gate_tb is
end AND_gate_tb;
--------------------------------

architecture AND_gate_tb_a of AND_gate_tb is


component AND_gate is
      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );

end component;


signal X, y, z: std_logic;



begin

AND_gate_inst: AND_gate

       port map (
             x, y, z        );




stimulus_proc : process is
begin
x<='0';
y<='0';
wait for 12ns;

x<='1';
y<='0';
wait for 2ns;
```

```
x<='1';
y<='1';
wait for 3ns;

x<='0';
y<='1';
wait for 7ns;

x<='0';
y<='0';
wait for 2ns;



  wait;


end process stimulus_proc;
end architecture AND_gate_tb_a;
```

OR_gate.vhd

```
library IEEE;
  use IEEE.std_logic_1164.all;
  USE ieee.std_logic_arith.all ;
  USE ieee.std_logic_unsigned.all ;
  use ieee.std_logic_textio.all;   -- in order to use hread( )
library work;
use work.all;


entity OR_gate is

      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );
end OR_gate ;



architecture rtl of OR_gate is

begin
process (x, y)
 begin
   z<=x OR y;
end process;

end rtl ;
```

Comb.vhd

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

library work;
use work.all;


---------------------------------
ENTITY comb IS
        port (x,y,z     : IN  std_logic;
              f         :OUT std_logic);
END comb;



---------------------------------

ARCHITECTURE logicFunc OF comb IS
BEGIN
        f <= (x AND (not y)) OR z;
END logicFunc;
```

CounterIF.vhd

```
library IEEE;
  use IEEE.std_logic_1164.all;
  USE ieee.std_logic_arith.all ;
  USE ieee.std_logic_unsigned.all ;
  use ieee.std_logic_textio.all;   -- in order to use hread( )
library work;
use work.all;


entity counterIF is

      port (
rst: in std_logic;
clk: in std_logic;
            x  : in std_logic;
            z  : out integer
                  );
end counterIF ;



architecture rtl of counterIF is

signal z_int: integer;
begin

z<=z_int;
process (rst, clk)
 begin
   if rst='0' then
z_int<=0;
elsifclk='1' and clk'event then
     if x='0' then
       if z_int<7 then
z_int<=z_int+2;
       else
z_int<=1;
       end if;
     else
       if z_int<12 and z_int>5 then
z_int<=z_int+1;
       else
z_int<=6;
       end if;
     end if;
   end if;
end process;

end rtl ;
```

counterIF_TB.vhd

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

library work;
use work.all;


--------------------------------
entity counterIF_tb is
end counterIF_tb;
--------------------------------

architecture counterIF_tb_a of counterIF_tb is

component counterIF is

      port (
rst: in std_logic;
clk: in std_logic;
            z  : out integer
                  );
end component;

constant clk_hp : time := 3000 ps;

  signal rst      :  std_logic;
  signal clk      :  std_logic;
  signal Z        :  INTEGER   ;
  signal x        :  std_logic;

begin

counterinst: counterIF
       port map
           (rst ,
clk,
            x,
            z
                  );

-------------------------------------------------------------------
clock_gen_proc : process is
begin
clk<= '1';
  wait for clk_hp;

clk<= '0';
  wait for clk_hp;
```

```
end process clock_gen_proc;
-------------------------------------------------------------------------


stimulus_proc : process is
   begin
rst<='0';
   x<='0';

   wait for 2* clk_hp;
   wait for 0.5 ns ;

rst<='1';

   wait for 60* clk_hp;
   x<='1';
   wait for 60* clk_hp;
   x<='0';

  wait;


end process stimulus_proc;


end architecture counter_tb_a;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use IEEE.std_logic_textio.all;
use ieee.std_logic_arith.all;

library work;
use work.all;


---------------------------------
entity counterIF_tb is
end counterIF_tb;
---------------------------------

architecture counterIF_tb_a of counterIF_tb is

component counterIF is

      port (
rst: in std_logic;
clk: in std_logic;
            z  : out integer
                  );
end component;
```

```
constant clk_hp : time := 3000 ps;

   signal rst       :  std_logic;
   signal clk       :  std_logic;
   signal Z         :  INTEGER   ;
   signal x         :  std_logic;

begin

counterinst: counterIF
        port map
             (rst ,
clk,
              x,
              z
                        );

-------------------------------------------------------------------
clock_gen_proc : process is
begin
clk<= '1';
   wait for clk_hp;

clk<= '0';
   wait for clk_hp;

end process clock_gen_proc;
-------------------------------------------------------------------


stimulus_proc : process is
    begin
rst<='0';
    x<='0';

    wait for 2* clk_hp;
    wait for 0.5 ns ;

rst<='1';

    wait for 60* clk_hp;
    x<='1';
    wait for 60* clk_hp;
    x<='0';

   wait;


end process stimulus_proc;


end architecture counter_tb_a;
```

Full_adder.vhd

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

library work;
use work.all;


---------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY FA IS
PORT ( A, B, Cin : IN std_logic;
      S, Cout : OUT std_logic );
END FA;


---------------------------------

ARCHITECTURE arc OF FA IS
SIGNAL x1, x2, x3 : std_logic ;
BEGIN
x1 <= A xor B ;
x2 <= A nand B ;
x3 <= Cinnand x1 ;
S <= x1 xorCin ;
Cout<= x2 nand x3 ;
END arc;
```

Example1.vhd

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

library work;
use work.all;


---------------------------------
ENTITY example1 IS
        port (x1,x2,x3: IN  std_logic;
              f         :OUT std_logic);
END example1;
---------------------------------

ARCHITECTURE logicFunc OF example1 IS
BEGIN
        f <= (x1 AND x2) OR (NOT x2 AND x3);
END logicFunc;
```

Example1_TB.vhd

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

library work;
use work.all;

---------------------------------
entity example1_tb is
end example1_tb;
---------------------------------

architecture example1_tb_a of example1_tb is

  component example1 is
     port (
            x1,x2,x3: IN  std_logic;
                f       :OUT std_logic);


  end component;

   signal x1,x2,x3, f: std_logic;

begin

example1_inst: example1
     port map ( x1, x2, x3, f );

stimulus_proc : process is
begin
x1<='0';
x2<='0';
x3<='0';
wait for 12ns;

x1<='0';
x2<='0';
x3<='1';
wait for 2ns;

x1<='0';
x2<='1';
x3<='0';
wait for 3ns;

x1<='0';
x2<='1';
x3<='1';
```

```
wait for 7ns;

x1<='1';
x2<='0';
x3<='0';
wait for 15ns;

wait;
end process stimulus_proc;
end architecture example1_tb_a;
```

Ευρετήριο όρων

Λίστα μαθησιακών στόχων

Το παρόν σύγγραμμα στοχεύει στην επίτευξη των παρακάτω μαθησιακών στόχων για το φοιτητή που το χρησιμοποιεί:

- να αναλύει και να σχεδιάζει συνδυαστικά ψηφιακά κυκλώματα με χρήση τεχνικών αναπαράστασης λογικών συναρτήσεων και άλγεβρας Boole
- να αναλύει και να σχεδιάζει ακολουθιακά ψηφιακά κυκλώματα με χρήση τεχνικών κατάστρωσης διαγραμμάτων καταστάσεων, πινάκων καταστάσεων και πινάκων διέγερσης
- να καταλαβαίνουν τη δομή και λειτουργία ενός κυκλώματος που έχει περιγραφεί με χρήση της γλώσσας VHDL
- να προσομοιώνουν τη λειτουργία ενός κυκλώματος που έχει περιγραφεί στη VHDL χρησιμοποιώντας κατάλληλο λογισμικό
- να προγραμματίζουν FPGA με στόχο την υλοποίηση συνδυαστικών αλλά και ακολουθιακών κυκλωμάτων

Διευκρινίζεται ότι οι μαθησιακοί στόχοι έχουν μόνο έμμεση σχέση με τα μαθησιακά αντικείμενα καθώς οι μαθησιακοί στόχοι σχετίζονται με τις δεξιότητες που επιδιώκεται να αποκτήσει κάποιος (π.χ. ικανότητα σύνθεσης, ανάλυσης, σχεδίασης) ενώ τα μαθησιακά αντικείμενα σχετίζονται με το αντικείμενο της μάθησης (π.χ. ψηφιακά κυκλώματα, προγραμματισμός FPGA).

Χρήση του συγγράμματος με βάση τους μαθησιακούς στόχους

Στον ακόλουθο πίνακα φαίνεται το κεφάλαιο στο οποίο καλύπτεται το κάθε μαθησιακό στόχο προκειμένου οι διδάσκοντες να διαρθρώνουν με ευέλικτο τρόπο την ύλη διδασκαλίας.

Μαθησιακός στόχος	Κεφάλαιο συγγράμματος
Να αναλύει και να σχεδιάζει συνδυαστικά ψηφιακά κυκλώματα με χρήση τεχνικών αναπαράστασης λογικών συναρτήσεων και άλγεβρας Boole	2
Να αναλύει και να σχεδιάζει ακολουθιακά ψηφιακά κυκλώματα με χρήση τεχνικών κατάστρωσης διαγραμμάτων καταστάσεων, πινάκων καταστάσεων και πινάκων διέγερσης	3
Να καταλαβαίνουν τη δομή και λειτουργία ενός κυκλώματος που έχει περιγραφεί με χρήση της γλώσσας VHDL	4
Να προσομοιώνουν τη λειτουργία ενός κυκλώματος που έχει περιγραφεί στη VHDL χρησιμοποιώντας κατάλληλο λογισμικό	4
Να προγραμματίζουν FPGA με στόχο την υλοποίηση συνδυαστικών αλλά και ακολουθιακών κυκλωμάτων	5

Πίνακας 1: Μαθησιακοί στόχοι και αντίστοιχα κεφάλαια

Παραδείγματα:

Ίδρυμα / τμήμα	**Μάθημα**	**Ύλη**
ΤΕΙ Στερεάς Ελλάδας Τμήμα Ηλεκτρολόγων Μηχανικών Τ.Ε	Ψηφιακά συστήματα Ι	Κεφάλαια 1, 2, 4.1 ως και 4.2
ΤΕΙ Στερεάς Ελλάδας Τμήμα Ηλεκτρολόγων Μηχανικών Τ.Ε	Ψηφιακά συστήματα ΙΙ	Κεφάλαια 1, 3, 4.3ως και 4.7
ΤΕΙ Στερεάς Ελλάδας Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε	ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ	Κεφάλαια 1, 2, 4.1 ως και 4.2
ΤΕΙ Στερεάς Ελλάδας Τμήμα Μηχανικών Τεχνολογίας Αεροσκαφών Τ.Ε	ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ	Κεφάλαια 1, 2.1 ως και 2.4, 3.1 ως και 3.4, 4.1 ως και 4.2
ΤΕΙ Πειραιά Τμήμα Μηχανικών Αυτοματισμού Τ.Ε.	Ψηφιακά συστήματα Ι	Κεφάλαια 1, 2, 4.1 ως και 4.2
ΤΕΙ Πειραιά Τμήμα Μηχανικών Αυτοματισμού Τ.Ε.	Ψηφιακά συστήματα ΙΙ	Κεφάλαια 1, 3, 4.3ως και 4.7
ΤΕΙ Πειραιά Τμήμα Ηλεκτρολόγων Μηχανικών Τ.Ε	Λογικήσχεδίαση	Κεφάλαια 1, 2.1 ως και 2.4, 3.1 ως και 3.4, 4.1 ως και 4.2

Βιβλιογραφία

Ασημάκης Ν., Ψηφιακά Ηλεκτρονικά, Εκδόσεις GUTENBERG, 2008.
Ασημάκης Ν., Βουρβουλάκης Ι., Κακαρούντας Α., Λελίγκου Ε., Ηλεκτρονικό Βιβλίο Λογική Σχεδίαση, Τ.Ε.Ι. Λαμίας, 2011.
P. Ashenden, "*The Designer's Guide to VHDL*", Morgan Kaufman Publishers, 1996
S. Sjoholm and L. Lennart, "*VHDL for Designers*", Prentice Hall, 1997
B. Cohen, "*VHDL Coding Styles and Methodologies*", Kluwer Academic Publishers, 1999
Z. Navabi, "*VHDL-Analysis and Modeling of Digital Systems*", Mc Graw-Hill, 1993
Ψηφιακή Σχεδίαση (5η έκδοση) Morris MANO, Michael Ciletti Παπασωτηρίου 1, 2013 Αθήνα
Ψηφιακή Σχεδίαση με τη Γλώσσα VHDL Αρχές και Πρακτικές Δ. Πογαρίδης Β. Γκιούρδας, 2007 Αθήνα
Σχεδίαση ψηφιακών συστημάτων με τη γλώσσα VHDL S. Brown, Z. Vranesic Τζιόλας,Θεσ/νίκη

On-line μαθήματα σε ASIC/VHDL design:

http://www.dacafe.com/ASICs.htm
http://www.ecs.umass.edu/ece/vspgroup/burleson/courses/558/
http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.vhd/html/vhdl-all-e.html
http://www.eas.asu.edu/~yong598d/VHDL_links.html

Tutorials/Papers:

http://www.vhdl.org/fmf/wwwpages/FMF_ECL_models_paper.html
http://www.bluepc.com/download.html#downloadtop
http://www.symphonyeda.com/products.htm
http://www.angelfire.com/electronic/in/vlsi/vhdl.html#vhdl

Άλλες πηγές:

http://www.ieee.org
http://www.acm.org
http://www.vhdl.org
http://tech-www.informatik.uni-hamburg.de/vhdl/
http://www.eeglossary.com/vhdl.htm
http://www.ent.ohiou.edu/~starzyk/network/Class/ee514/intro.html