ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΕΛΛΑΣ

Ταχέα Κυκλώματα Παράλληλης Σύγκρισης για Χρονοδρομολογητές

Κώστας Γ. Ε. Χαρτερός

Μεταπτυχιακή Εργασία

Ηράκλειο, Μάρτιος 2002

Ταχέα Κυκλώματα Παράλληλης Σύγκρισης για Χρονοδρομολογητές

Κώστας Γ. Ε. Χαρτερός

Μεταπτυχιακή Εργασία

Επιβλέπων: Μανόλης Κατεβαίνης, *Καθηγητής*

Περίληψη

Η αποθήκευση σε ουρά ανά ροή και οι εξεζητημένοι χρονοδρομολογητές είναι σημαντικοί μηχανισμοί για την παροχή εγγυήσεων Ποιότητας Υπηρεσίας στα δίκτυα. Οι περισσότεροι προηγμένοι αλγόριθμοι χρονοδρομολόγησης βασίζονται σε μία γνωστή υπολογιστική αρχή: την ουρά προτεραιότητας. Ουρές προτεραιότητας φτιάχνονται χρησιμοποιώντας δομές δεδομένων στοίβας, όταν το σύνολο των στοιχείων στην ουρά αλλάζει με αργούς ρυθμούς. Ωστόσο, όταν το πλήθος των στοιχείων της ουράς αλλάζει με αυθαίρετα γρήγορους ρυθμούς, είναι αναγκαία η χρήση νέων δομών σε υλικό για την υποστήριξη ταχέων λειτουργιών. Στην παρούσα εργασία αναπτύσσουμε τέτοια κυκλώματα. Χρησιμοποιούμε ένα διαδικό δένδρο συγκριτών, για τον εντοπισμό του μικρότερου στοιχείου σε ένα αυθαίρετο (μη-διατεταγμένο) σύνολο στοιχείων. Μελετήθηκε η διαδικασία σύγκρισης και σχεδιάστηκε ένας συγκριτής δύο αριθμών, ο οποίος είναι ο δομικός λίθος του δένδρου. Παρουσιάζουμε μία καινοτομία στην οργάνωση του δένδρου, στο οποίο τα σήματα μεταδίδονται κατά μήκος του συγκριτή δύο αριθμών και διαμέσου των επιπέδων του δένδρου, ταυτόχρονα. Με αυτόν τον τρόπο η καθυστέρηση κάθε συγκριτή και η καθυστέρηση κάθε επιπέδου του δέντρου επικαλύπτονται αντί να προστίθενται. Το διαδικό δέντρο συγκριτών είναι η καρδιά ενός χρονοδρομολογητή σταθμισμένης κυκλικής εξυπηρέτησης, που σχεδιάσαμε. Όλα τα κυκλώματα περιγράφηκαν σε συνθέσιμη γλώσσα Verilog (HDL). Επιπλέον, περιγράφηκαν σε κώδικα γλώσσας C, για λόγους ελέγχου και επαλήθευσης. Παρουσιάζουμε αποτελέσματα σύνθεσης για καθυστέρηση, κατανάλωση ισχύος και εμβαδόν κυκλώματος, για τεχνολογία 0.18μm CMOS.

Fast parallel comparison circuits for scheduling

Kostas G. I. Harteros

Master of Science Thesis

Supervisor: Manolis Katevenis, *Professor*

Abstract

Per-flow queueing and sophisticated schedulers are an important mechanism for providing Quality of Service (QoS) guarantees in networks. Most advanced scheduling algorithms rely on a common computational primitive: the priority queue. Priority queues can be built efficiently using heap data structures, when the set of elements in the queue varies slowly. However, if this set of eligible flows changes arbitrarily fast, new hardware structures are needed to support high-speed operation; in this work we develop such circuits. We use a binary tree of comparators, which locates the minimum in an arbitrary set of elements. The comparison operation is studied extensively and a 2-element comparator, which is the main block of the binary tree, is designed. We developed an innovative organization for the tree, where signals are propagated across each 2-element comparator as well as the tree levels, at the same time; in this way, the delays of the individual comparators and the delays of the tree levels are placed in parallel, rather than in series. This binary tree of comparators is the heart of a weighted-round-robin scheduler that we designed. Our designs are described in synthesisable Verilog (HDL); in addition, designs were described in C code for verification purposes. Synthesis results are presented in terms of delay, power, and area, for a 0.18 CMOS process.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους όσους βοήθησαν στην περάτωση αυτής της εργασίας. Πρώτον από όλους θελω να ευχαριστήσω τον καθ. Μανόλη Κατεβαίνη που έδωσε την ιδέα για την εργασία, για τις ανεκτίμητες υποδείξεις και ιδέες καθόλη τη διάρκεια της εργασίας.

Θέλω να ευχαριστήσω τον καθ. Διονύση Πνευματικάτο για τις συζητήσεις περί σύνθεσης. Ευχαριστώ τους καθηγητές Απόστολό Τραγανίτη και Ευάγγελο Μαρκάτο για τις υποδείξεις τους. Ευχαριστώ τον Γιώργο Κορνάρο και τον Χρίστο Σωτηρίου για την διδασκαλία των εργαλείων CAD.

Θελω να ευχαριστήσω τον Νίκο Χρυσό για τις πολύτιμες ώρες συνεργασίας, τις συζητήσεις, τις κρίσιμες υποδείξεις και παρατηρήσεις. Ευχαριστώ τον Παύλο Ρομπογιαννάκη για τις διορθώσεις στο αγγλικό κείμενο και την Τασία Ρισάνου για τη βοήθεια στις διαφάνειες της παρουσίασης.

Θέλω να ευχαριστήσω τους Γρηγόρη Γκίκα, Ξένια Ασιμακοπούλου και Αντώνη Δανάλη για την βοήθειά τους στα πρώτα μου βήματα στην Επιστήμη Υπολογιστών.

Ευχαριστώ την Europractice και το Τμήμα Επιστήμης Υπολογιστών για την παροχή των εργαλείων CAD και των Βιβλιοθηκών Τεχνολογίας. Ευχαριστώ το Τμήμα Επιστήμης Υπολογιστών και το Ίδρυμα Τεχνολογίας και Έρευνας για την υποτροφία που μου παρείχε.

Ευχαριστώ όλους τους Μεταπτυχιακούς Φοιτητές του Τμήματος Επιστήμης Υπολογιστών και του Τμήματος Φυσικής του Πανεπιστημίου Κρήτης για την αμέριστη βοήθεια και στήριξη.

Τέλος ευχαριστώ τον Γιώργο, την Ειρήνη τον Βασίλη και τη Βούλα για την αγάπη, την ηθική και υλική στήριξη που μου παρείχαν καθ' όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Στο Διαδίκτυο συνδέονται χιλιάδες υπολογιστές σε ολόκληρο τον κόσμο. Εκεί μπορούν να ανταλλάσσουν μηνύματα και να μοιράζονται πόρους. Είναι μία συλλογή από ιεραρχικά δίκτυα, όπου χρησιμοποιείται μία ποικιλία τεχνολογιών διασύνδεσης. Σε χαμηλότερο επίπεδο, δεκάδες ή εκατοντάδες υπολογιστές συνδέονται μεταξύ τους και με δρομολογητές (routers) διαμέσου Τοπικών Δικτύων Υπολογιστών (LAN) ή διαμορφωτή-αποδιαμορφωτή (modem). Τα δίκτυα υπολογιστών επιτρέπουν στους χρήστες να μοιράζονται πόρους όπως εκτυπωτές, αρχεία, εύρος συχνότητας και σελίδες στον Παγκόσμιο Ιστό. Ωστόσο, η έννοια του ¨μοιράσματος¨ αυτόματα εγείρει το πρόβλημα του ανταγωνισμού για τους διαμοιραζόμενους πόρους. Για δοσμένο σύνολο αιτήσεων για πόρους σε μία ουρά εξυπηρέτησης, ο εξυπηρετητής ασκεί κανόνες χρονοπρογραμματισμού (scheduling disciplines) για να αποφασίσει την επόμενη για εξυπηρέτηση. Οι κανόνες χρονοπρογραμματισμού είναι σημαντικοί, γιατί μπορούν να επιβάλλουν δικαιοσύνη στο διαμοίρασμα πόρων και εγγυήσεις επίδοσης για εφαρμογές όπως η τηλεφωνία και το Video on Demand.

Οι ειδικοί συμφωνούν ότι τα μελλοντικά δίκτυα θα μεταφέρουν τουλάχιστον δύο τύπους εφαρμογών. Αυτές του πρώτου τύπου δεν θα εξαρτώνται από τις επιδόσεις του δικτύου (π.χ. FTP). Οι ανάγκες για καλές επιδόσεις αυτών των εφαρμογών είναι ελαστικές: προσαρμόζονται στο ποσό των διαθέσιμων πόρων. Τέτοιου είδους εφαρμογές καλούνται best effort, επειδή το δίκτυο υπόσχεται μόνο την μεταφορά των πακέτων, χωρίς να εγγυάται για οποιοδήποτε κάτω όριο επίδοσης. Από την άλλη, υπάρχουν εφαρμογές που πρέπει να εξυπηρετηθούν από το δίκτυο και απαιτούν κάποιο κάτω όριο στην επίδοση του δικτύου (π.χ. μετάδοση φωνής). Τέτοιες εφαρμογές προϋποθέτουν εγγυήσεις ποιότητας υπηρεσίας από το δίκτυο. Οι εφαρμογές εγγυημένης εξυπηρέτησης προϋποθέτουν τη διαμοίραση των πόρων προς όφελός τους από το δίκτυο. Όταν συμβαίνει αυτό σε κάποιο δίκτύο, ονομάζεται παροχή Ποιότητας Υπηρεσίας (QoS).

Οι εφαρμογές εγγυημένης εξυπηρέτησης επηρεάζουν τη χρονοδρομολόγηση διαμέσου των κανόνων που επιβάλλει ο εξυπηρετητής κατά μήκος μιας διαδρομής που ακολουθούν τα πακέτα μιας σύνδεσης. Η παροχή εγγυήσεων Ποιότητας Υπηρεσίας προϋποθέτει την απομόνωση ροών, για να δέχεται η κάθε μία το δικό της επίπεδο εξυπηρέτησης. Προαπαιτούμενο για να συμβεί αυτό είναι η αποθήκευση ροών σε ξεχωριστές ουρές: ουρά-ανα-ροή. Ο χρονοδρομολογητης εφαρμόζει κανόνες για κάθε ουρά στην έξοδο του εξυπηρετητή, για να αποφασίσει ποίο πακέτο έχει σειρά να μεταδοθεί. Ορίζονται διαφορετικοί μέσοι χρόνοι καθυστέρησης, εύρος ζώνης και ρυθμός απώλειας πακέτων για διαφορετικές συνδέσεις.

Σε ένα δίκτυο υψηλών ταχυτήτων, ο χρονοδρομολογητής ενός εξυπηρετητή, πρέπει να επιλέξει το επόμενο προς μετάδοση πακέτο, σε χρόνο ίσο με αυτόν της μετάδοσης ενός πακέτου. Ο τελευταίος είναι της τάξης μερικών νανοδευτερολέπτων, και μέσα σε αυτόν το χρόνο πρέπει να παρθεί η απόφαση. Κατά συνέπεια, η εύκολη και φθηνή υλοποίηση σε υλικό (hardware) αντί σε λογισμικό (software) των κανόνων χρονοδρομολόγησης, είναι αναγκαία. Με τη σημερινή τεχνολογία VLSI η υλοποίηση ενός πολύπλοκου χρονοδρομολογητή είναι εφικτή.

Η διάταξη στοιχείων ενός συνόλου είναι ένα πολυσυζητημένο θέμα στην έρευνα της επιστήμης των υπολογιστών. Από την έλευση της τεχνολογίας VLSI, έχουν προταθεί αρκετά κυκλώματα για διάταξη στοιχείων. Ωστόσο, η πλήρης διάταξη των στοιχείων συχνά δεν είναι επιθυμητή. Πολλές εφαρμογές, όπως η χρονοδρομολόγηση με παροχή Ποιότητας Υπηρεσίας, χρησιμοποιούν Ουρα Προτεραιότητας, όπου μόνο ο καθορισμός του ελάχιστου (ή μέγιστου) στοιχείου είναι αρκετός. Το γενικό πλαίσιο του προβλήματος με το οποίο ασχολείται η παρούσα εργασία είναι η υλοποίηση ενός εξεζητημένου και ταχύ αλγορίθμου χρονοδρομολόγησης χιλιάδων ανταγωνιζόμενων ροών, παρέχοντας Ποιότητα Υπηρεσίας.

Στη βιβλιογραφία έχουν αναφερθεί υλοποιήσεις ουρών προτεραιότητας με δομές δεδομένων. Η καλύτερη από αυτές είναι η στοίβα (heap). Το πρόβλημα με τη στοίβα είναι ότι δεν μπορούν να εισέλθουν και να εξέλθουν πολλά στοιχεία από αυτή, ταυτόχρονα. Η κατάσταση αυτή μπορεί να εμφανιστεί στον χρονοδρομολογητή ενός εξυπηρετητή και έχει να κάνει με έλεγχο ροών και μνήμη του εξυπηρετητή.

1.1 Συνεισφορά της παρούσας εργασίας

Η προσέγγιση που παρουσιάζεται σε αυτή τη δουλειά είναι η εύρεση του μικρότερου (ή μεγαλύτερου) από ένα σύνολο μη-διατεταγμένων αριθμών. Σε αυτή την κατάσταση η ταυτόχρονη εισαγωγή και εξαγωγή πολλών στοιχείων στο σύνολο είναι εύκολη. Η συνεισφορά αυτής της δουλειάς είναι η παρουσίαση μιας καινοτομίας στην οργάνωση του διαδικού δέντρου, όπου τα σήματα διαδίδονται κατά μηκος του συγκριτή δύο αριθμών και κατά μήκος των επιπέδων του δέντρου. παράλληλα. Έτσι οι καθεστέρηση των επιπέδων και των συγκριτών επικαλύπτονται αντί να προστίθενται. Το διαδικό δέντρο συγκριτών είναι η καρδιά ενός χρονοδρομολογητή σταθμισμένης κυκλικής εξυπηρέτησης που σχεδιάσαμε. Το κύκλωμα γράφτηκε σε συνθέσιμη γλώσσα Verilog. Τα αποτελέσματα της σύνθεσης του κυκλώματος δείχνουν περίοδο κύκλου ρολογιού μικρότερη από 10ns για χρονοδρομολογητή με 256 ροές, σε τεχνολογία CMOS ASIC 0.18μm.

Κεφάλαιο 2

Υλοποίηση ουρών προτεραιότητας σε υλικό

Η παροχή Ποιότητας Υπηρεσίας κάνει την αναγκαία την αποθήκευση ροών σε διαφορετικές ουρές, μαζί με τη χρήση ενός καλού χρονοδρομολογητή. Εκεί οι προτεραιότητες χρησιμοποιούνται σαν βασικός μηχανισμός. Η ομαδοποίηση ροών (ιεραρχική χρονοδρομολόγηση) είναι ένας άλλος μηχανισμός: αρχικά επιλέγεται μία ομάδα ροών και στη συνέχεια μία ροή μέσα στην ομάδα. Οι δυσκολότεροι κανόνες χρονοδρομολόγησης είναι αυτοί που ανήκουν στην οικογένεια της σταθμισμένης κυκλικής εξυπηρέτησης (Weighted Round Robin - WRR).

2.0.1 Σταθμισμένη Κυκλική Εξυπηρέτηση

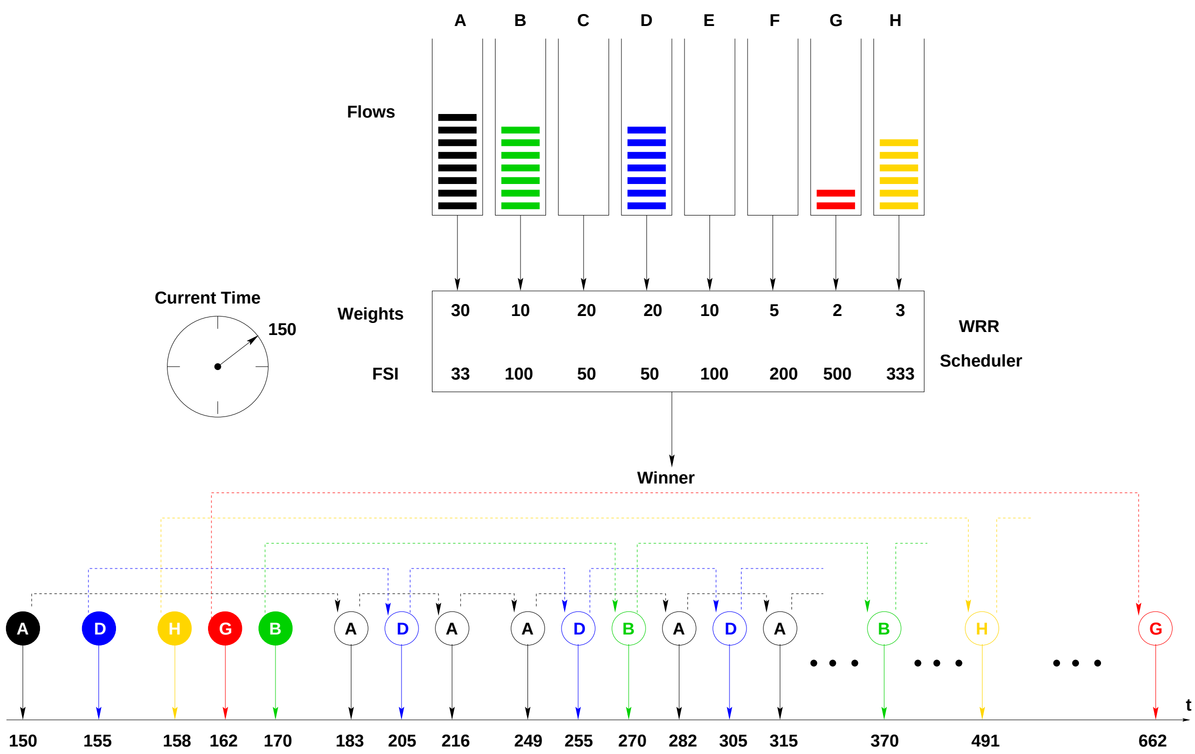
Η εφαρμογή των κανόνων αυτής της οικογένειας επιβάλουν την αντιστοίχηση ενός *βάρους* σε κάθε ροή. Στις ροές με μεγάλο βάρος πρέπει να παρέχονται καλύτερες υπηρεσίες σε σχέση με τις υπόλοιπες. Η ανάγκη για ένα σύστημα μέτρησης οδηγεί στον ορισμό ενός χρόνου εσωτερικό στο σύστημα. Με βάση αυτόν μπορούμε να ορίσουμε χρονικά μεγέθη που έχουν άμεση σχέση με κάθε ροή. Ορίζεται ο χρόνος μεταξύ δύο διαδοχικών εξυπηρετήσεων μιας ροής (Flow Service Interval - FSI) ο οποίος είναι αντιστρόφως ανάλογος με το βάρος της. Επίσης ορίζεται ο χρόνος επόμενης εξυπηρέτησης της ροής (Next Service Time - NST). Οι Ουρές Προτεραιότητας ορίζονται ως ακολούθως:

- Διατηρούμε ένα μεταβλητό σύνολο ενεργών ροών
- Συσχετίζουμε κάθε ροή με ένα χρόνο επόμενης εξυπηρέτησης
- Βρίσκουμε και εξυπηρετούμε τη ροή με το *μικρότερο* χρόνο εξυπηρέτησης
- Προγραμματίζουμε σε μελλοντικό χρόνο την εξυπηρετηθείσα ροή

Στο Σχ. ;; παρουσιάζεται ένα παράδειγμα χρονοπρογραμματισμού Σταθμισμένης Κυκλικής Εξυπηρέτησης, για την κατανόηση της ακολουθίας των γεγονότων. Φαίνονται οκτώ ροές, A εως H. Πέντε από αυτές είναι ενεργές (έχουν μή-άδειες ουρές). Ο χρονοδρομολογητής πρέπει να εξυπηρετήσει τις ενεργές ροές, με τέτοια σειρά ώστε ο ρυθμός εξυπηρέτησης να είναι ανάλογος με το βάρος της ροής. Η ροή A έχει προγραμματιστεί να εξυπηρετηθεί σε χρόνο 150, η D σε χρόνο 155, η H σε χρόνο 158, η G σε χρόνο 162 και η B σε χρόνο 170. Η ροή που θα εξυπηρετηθεί νωρίτερα είναι αυτή με το μικρότερο NST. Σε αυτό το παράδειγμα είναι η A, αφού ο χρόνος του συστήματος έχει φτάσει στο 150. Η A μένει ενεργή μετά την εξυπηρέτηση του πακέτου στην κεφαλή της ουράς της, με συνέπειά να επαναχρονοδρομολογηθεί. Ο χρόνος επόμενης εξυπηρέτησης της A προκύπτει από άθροιση του NST (150) με τον FSI(33): $t_{NST,A}$=150+33=183. Η ροή A δεν θα εξυπηρετηθεί μέχρι ο χρόνος να φτάσει στο 183. Συνεχίζοντας, μετά την εξυπηρέτηση της A ο χρόνος του συστήματος αυξάνεται στο 155 και είναι η σειρά της ροής D για εξυπηρέτηση. Αφού εξυπηρετηθεί και επαναπρογραματιστεί με $t_{NST,D}$=155+50=205, ο χρόνος θα προχωρήσει στο 158 κ.ο.κ.

2.1 Σχετική δουλειά για αλγόριθμους ουράς προτεραιότητας υλοποιημένους σε υλικό

Στη βιβλιογραφία έχουν αναφερθεί διάφορες υλοποιήσεις ουρών προτεραιότητας. Σε όλες χρησιμοποιείται ένα σύνολο αριθμών. Για τον λόγο αυτό χρησιμοποιούνται δομές δεδομένων. Οι απλούστερη από αυτές είναι η συνδεδεμένη λίστα [MR00]. Η αρχιτεκτονική του μεταγωγέα που χρησιμοποιείται είναι κοινού χώρου ενταμίευσης (shared buffer) και αποθήκευσης σε ουρές στις εξόδους (output



Σχήμα 2.1: Παράδειγμα εφαρμόγής αλγορίθμου σταθμισμένης κυκλικής εξυπηρέτησης.

queueing) με διαφορετική ουρά προτεραιότητας για κάθε έξόδο. Η αποθήκευση γίνεται ανά πακέτο. Αυτό είναι μειονέκτημα για μεγάλο αριθμό πακέτων N αφού ο χρονοδρομολογητής χρειάζεται πληροφορία για το σύνολο των πακέτων. Σε αντιδιαστολή, με την αποθήκευση ανά ροή που χρησιμοποιείται σε αυτή την εργασία, ο χρονοδρομολογητής χρειάζεται πληροφορία μόνο για την κεφαλή και όχι για το σύνολο της ουράς. Ένα άλλο μειονέκτημα της εργασίας [MR00] είναι η δυσκολία στην εισαγωγή και εξαγωγή πολλών στοιχείων στην ουρά ταυτοχρόνως, που κληρονομείται από τη φύση της συγκεκριμένης δομής.

Μια άλλη προσέγγιση είναι η χρησιμοποίηση σωρού της οποίας τα στοιχεία είναι αποθηκευμένα σε σωρό (heap) [IK01]. Η διάταξη των στοιχείων διατηρείται με τη βοήθεια ενός Διαχειριστή Σωρού που χρησιμοποιεί Ομοχειρία. Τα στοιχεία της ουράς είναι αποθηκευμένα σε δίπορτες μνήμες SRAM. Η εισαγωγή νέου στοιχείου γίνεται από τη ρίζα του δέντρου. Δύο λειτουργίες ορίζονται για τα στοιχεία του δέντρου: *εισαγωγή* και *διαγραφή*. Συνεχόμενες διαγραφές χωρίζονται από ένα αδρανή κύκλο. Η ρίζα του δέντρου έχει πάντα το μικρότερο στοιχείο. Το μειονέκτημα αυτής της τοπολογίας είναι η δυσκολία εισαγωγής και εξαγωγής πολλών στοιχείων στη στοίβα, ταυτόχρονα. Αντίθετα, στην εργασία που παρουσιάζεται εδώ οι ταυτόχρονες εισαγωγές και εξαγωγές είναι εφικτές.

Η παρούσα εργασία ασχολείται με αλγόριθμους εύρεσης του μικρότερου στοιχείου από ένα αυθαίρετο (μη-διατεταγμένο) πλήθος στοιχείων. Πριν να παρουσιάσουμε την δικιά μας εργασία και έναν ήδη υπάρχον αλγόριθμο θα ορίσουμε το περιβάλλον στο οποίο λειτουργεί ο χρονοδρομολογητής.

2.2 Το περιβάλλον του χρονοδρομολογητή και το πρόβλημα με τη στοίβα

Ο χρονοδρομολογητής είναι κρίσιμο κομμάτι ενός μεταγωγέα. Είναι υπεύθυνος για την σείρα εξυπηρέτησης των ροών εκφράζοντας την πολιτική του διαχειριστή του δικτύου. Δέχεται εντολές από τον Διαχειριστή Ουρών (Queue Manager) του μεταγωγέα και πληροφορία (backpressure) από τον έλεγχο ροής (flow control). Αυτά εκφράζονται με ένα σύνολο ready bits, ένα για κάθε ροή. Καθε ready bit δείχνει τη αν μία ροή είναι ενεργή ή ανενεργή.

Σε πρώτη προσέγγιση μία ροή είναι ανενεργή όταν η ουρά της είναι άδεια. Αυτή γίνεται ενεργή, μόλις φτάσει ένα πακέτο στην ουρά. Σε μία πιο κοντινή ματιά στον μεταγωγέα, παρουσιάζονται και άλλες περιπτώσεις κατά τις οποίες μία ροή κάνει τη μετάβαση από ενεργή σε ανενεργή και το αντίστροφο. Η πρώτη περίπτωση έχει να κάνει με τη Διαφύλλωση στη Μνήμη (Memory Interleaving). Η φυσική μνήμη στην οποία αποθηκεύονται πακέτα είναι συνήθως SDRAM. Όπως είναι γνωστο διαδοχικές προσπελάσεις σε αυτή τη μνήμη δεν γίνονται σε διαδοχικούς κύκλους ρολογιού, για λόγους προφόρτισης (precharging) της μνήμης. Το αποτέλεσμα είναι οι ροές που βρίσκονται στο κομμάτι της μνήμης που γίνεται η προφόρτιση να εμφανιστούν σαν ανενεργές στον χρονοδρομολογητή ταυτόχρονα, με εντολή του διαχειριστή ουρών. Ο τελευταίος δεν μπορεί να εξυπηρετήσει κάποια από αυτές που τυχόν επιλέγει από τον χρονοδρομολογητή.

Η επόμενη περίπτωση που μπορεί κάποιες ροές να εμφανιστούν ανενεργές είναι λόγω backpressure. Σήματα backpressure εγείρονται από εμφάνιση καταστάσεων είτε μέσα, είτε έξω από τον μεταγωγέα. Υποθέστε ότι χρησιμοποιούμε crossbar με ενταμιευτές σε καθε crosspoint [CK02], και χρονοδρομολογητες σε κάθε είσοδο και έξοδο του μεταγωγέα. Εάν οι ενταμιευτές σε κάποιο crosspoint γεμίσουν, τότε σήματα backpressure εγείρονται από αυτό το σημείο προς τον χρονοδρομολογητή στην είσοδο. Όλες οι ροές που έχουν πακέτα για αυτή την είσοδο πρέπει να καθιστούν ανενεργές ταυτόχρονα. Από την άλλη, ένας χρονοδρομολογητής εξόδο "βλέπει" πολλά πακέτα να έρχονται ταυτόχρονα στα crosspoints, με συνέπεια πολλές ροές να γίνονται ενεργές ταυτόχρονα.

Ας υποθέσουμε για τη συνέχεια ότι δεν έχουμε crossbar άλλα στοιχεία μεταγωγής όπως Banyan ή Benes. Υπάρχει περίπτωση σε κάποιο σύνδεσμο του δικτύου μεταγωγής να εμφανίζεται συμφόρηση. Τότε σήματα backpressure ασκούνται από αυτό τον σύνδεσμό προς τα προηγούμενα στοιχεία μεταγωγής. Άρα σε κάθε "προηγούμνο" στοιχείο μεταγωγής, όλες οι ροές που έχουν πακέτα για το συμφορημένο σημείο πρέπει να γίνουν ανενεργές ταυτόχρονα. Επιπλέον, μόλις το προβληματικό σημείο αποσυμφορηθεί, όλες οι προηγούμενες ξαναγίνονται ενεργές ταυτόχρονα. Η κατάσταση αυτή μπορεί να εμφανιστεί όχι μόνο σε στοιχεία μεταγωγής μέσα σε ένα μεταγωγέα, αλλά και μεταξύ μεταγωγέων σε ένα δίκτυο, με παρόμοια αποτελέσματα για τις ροές.

Κατά συνέπεια, υπάρχουν καταστάσεις που εμφανίζονται σε ένα μεταγωγέα και προκαλούν την ταυτόχρονη αλλαγή κατάστασης ενός μεγάλου αριθμού ροών από ενεργές σε μη-ενεργές και το αντίστροφο. Οπότε, το πλήθος των στοιχείων του συνόλου που αποτελουν την ουρά προτεραιότητας αλλάζει ταχέα. Οι υλοποιήσεις ουρών προτεραιότητας με δομές δεδομένων δυσκολεύονται να ανταποκριθούν σε αυτές τις αλλαγές. Η λύση που προτείνεται σε αυτή την εργασία είναι η χρήση ενός αλγορίθμου που υπολογίζει το μικρότερο από ένα σύνολο αυθαίρετων (μη-διατετεγμένων) στοιχείων. Έτσι η ταυτόχρονη εισαγωγή και εξαγωγή στοιχείων είναι μία εύκολη διαδικασία. Στην επόμενη ενότητα θα συζητήσουμε για δύο τέτοιους αλγορίθμους, έναν προϋπάρχον και αυτόν που προτείνεται σε αυτή την εργασία.

2.3 Αλγόριθμοι για την εύρεση του μικρότερου στοιχείου σε ένα σύνολο μη διατεταγμένων στοιχείων

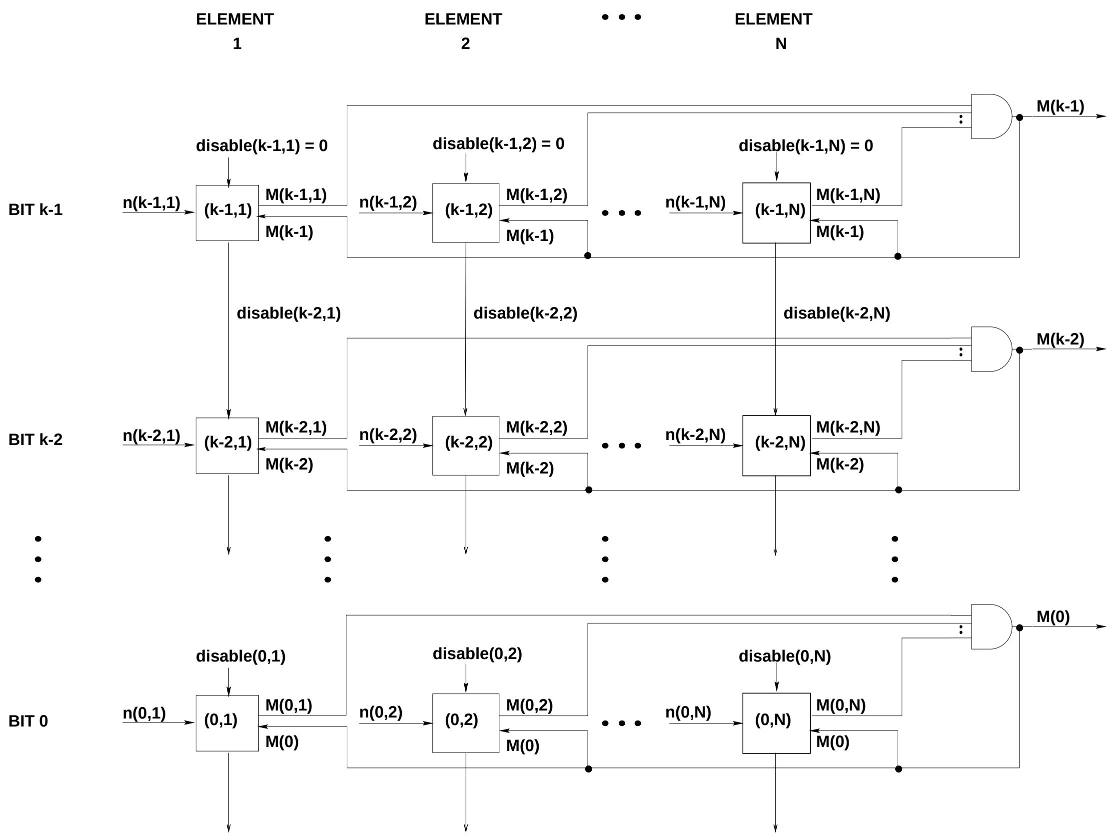
Το πρόβλημα του καθορισμού του ελαχίστου ορίζεται ως εξής. Δοσμένου συνόλου N στοιχείων n_1,n_2,... n_N, είναι επιθυμητή η εύρεση στοιχείου $M \in N$, τέτοιο ώστε $M \geq n_i$ ($M \leq n_i$) ($N \geq i \geq 1$). Στη συζήτηση παρακάτω υποθέτουμε ότι το στοιχείο ν_i αναπαρίσταται ως k-bit μη-προσιμασμένος διαδικός αριθμός ($n_{i,k-1}$,$n_{i,k-2}$,$n_{i,k-3}$,... $n_{i,0}$). Αρχικά στην ενότητα *αρχιτεκτονική πίνακα* θα συζητήσουμε για μια προϋπάρχουσα λύση, ενώ η δική μας λύση παρουσιάζεται στην ενότητα *αρχιτεκτονική διαδικου δέντρου.*

2.3.1 Αρχιτεκτονική Πίνακα

Η αρχιτεκτονική πίνακα προτάθηκε στην εργασία [VM93]. Εκεί υπολογίζεται το μεγαλύτερο στοιχείο ενός συνόλου. Εδώ παρουσιάζεται με μία μικρή αλλαγή, ώστε να υπολογίζει το μικρότερο στοιχείο. Η αρχιτεκτονική φαίνεται στο Σχ. ;; . Ο πίνακας χωρίζεται σε kxN κύτταρα. Οι k είσοδοι σε μία στήλη δέχονται τα bits του στοιχείου. Έτσι κάθε στήλη είναι ένα στοιχείο. Τα bits σε καθε γραμμή είναι ίδιας σημαντικότητας, με το Πιο Σημαντικό Βιτ (Most Significant Bit) όλων των στοιχείων να βρίσκεται στην ανώτερη γραμμή. Υπάρχουν disable σήματα που καθορίζουν την συμμετοχή κάθε στοιχείου στον διαγωνισμό για το μικρότερο. Υπάρχει ένα τέτοιο σήμα ανά στήλη. Αρχικά τίθεται στην τιμή 0, που σημαίνει ότι το στοιχείο παίρνει μέρος στη σύγκριση. Τα σήματα αυτά είναι ριπης (ripple) και ξεκινώντας από την ανώτερη γραμμή καταλήγουν στην κατώτερη. Κατά την κατάβαση και ανάλογα με την τιμή του bit του στοιχείου, το disable παραμένει 0 ή αλλάζει σε 1. Το στοιχείο που θα καταλήξει να έχει 0 το disable είναι το μικρότερο στοιχείο της σύγκρισης. Αλλαγή στο disable συμβαίνει για κάποιο στοιχείο, όταν το bit του διαφέρει από αυτό του μικρότερου. Στην έξοδο κάθε πύλης ΑΝΔ παράγεται ένα bit του μικρότερου στοιχείου.

Αυτό το κύκλωμα έχει δύο σημαντικά μειονεκτήματα. Τα πρώτα δύο αφορούν τις πύλες AND οι οποίες έχουν fan-in και fan-out ισό με το πλήθος των αριθμών.

Αυτά μειώνουν σημαντικά τις επιδόσεις του κυκλώματος, λόγω του μεγάλου φορτίου για αυτές τις πύλες. Μία άλλη συνέπεια είναι η καθυστέρηση της απόδοσης τιμής σε κάθε disable σήμα το οποίο πρέπει να περιμένει την έξοδο κάθε πύλης να σταθεροποιηθεί. Η πολυπλοκότητα του αλγορίθμου είναι $O(k)+O(\log_2 N)$. Εξαρτάται από τον αριθμό των bits γραμμικά και από το πλήθος των στοιχείων λογαριθμικά, λογω των πυλών AND.



Σχήμα 2.2: Η αρχιτεκτονική πίνακα για τη σύγκριση N στοιχείων με k bit το καθένα. Κάθε γραμμή παράγει ένα bit του μικρότερου στοιχείου.

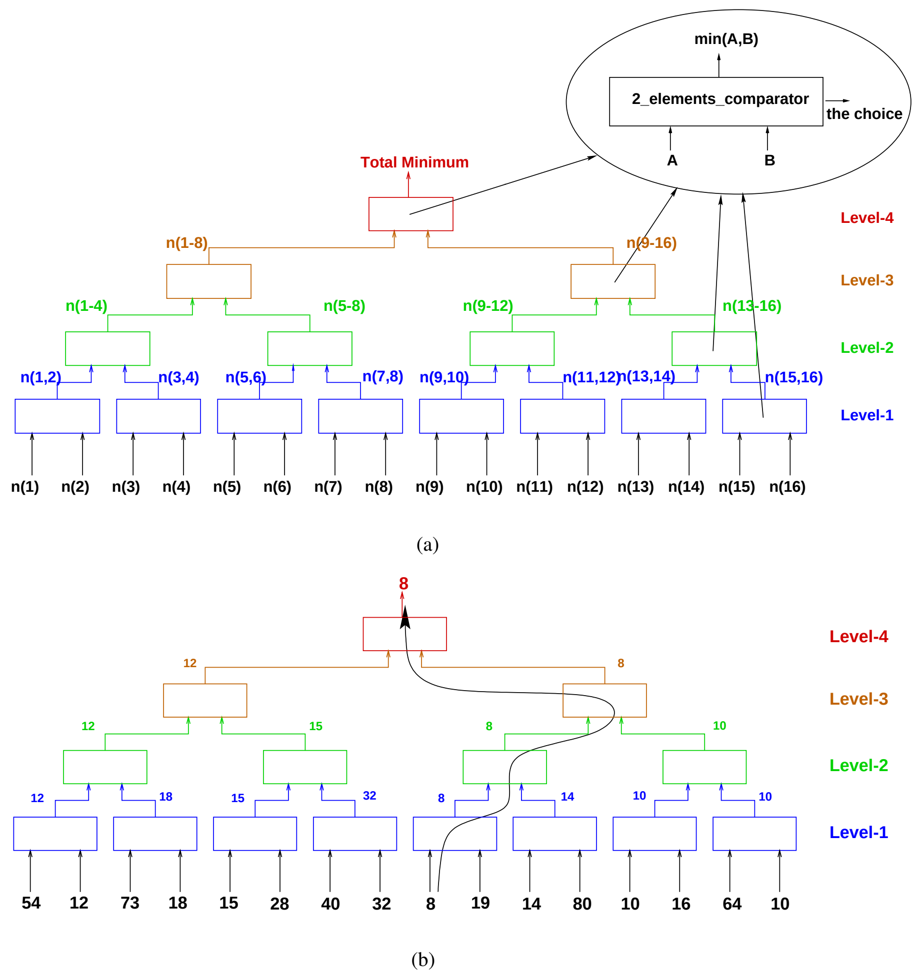
2.3.2 Αρχιτεκτονική Δυαδικού Δεντρου

Δύο ιδέες προτείνονται για την εύρεση του μικρότερου στοιχείου σε ένα μηδιατεταγμένο σύνολο. Η πρώτη αφορά τη χρησιμοποίηση ενός *διαδικού δέντρου συγκριτών*. Κάθε κόμβος του δέντρου είναι ένας συγκριτής δύο στοιχείων, για τον οποίο υποθέτουμε πολυπλοκότητα υπολογισμού O(1). Όλα τα στοιχεία εισάγονται από τα φύλλα του δέντρου, ταυτόχρονα. Η πολυπλοκότητα υπολογισμού του ελαχίστου στοιχείου, που κληρονομείται από τη δομή του δέντρου, είναι $O(\log_2 N)$. Αυτό είναι το ύψος του δέντρου για N στοιχεία. Αντίθετα, η πολυπλοκότητα εμβαδού κυκλώματος και κατανάλωσης ισχύος είναι γραμμική συνάρτηση του πλήθους των στοιχείων O(N).

Η διαδικασία εύρεσης ξεκινάει από το κατώτερο επίπεδο, το οποίο εκτελεί N/2 ταυτόχρονες συγκρίσεις, σε χρόνο O(1). Τη σκυτάλη παίρνει το αμέσως ανώτερο επίπεδο, το οποίο εκτελεί N/4 ταυτόχρονες συγκρίσεις σε χρόνο O(1). Η διαδικασία τελειώνει στον συγκριτή-ρίζα του δέντρου, στου οποίου την έξοδο εμφανίζεται το ελάχιστο στοιχείο του συνόλου. Το γεγονός ότι κάθε συγκριτής προωθεί ένα από τα δύο στοιχεία εισόδου του στην έξοδο, δίνει την ευκαιρία σε ένα στοιχείο να ταξιδέψει από τα φύλλα στη ρίζα, και να γίνει ο νικητής του διαγωνισμού. Η μοίρα των υπόλοιπων στοιχείων είναι να σταματήσουν σε κάποιο συγκριτή-κόμβο, την άνοδό τους προς τη ρίζα.

Μία υλοποίηση για N=16 φαίνεται στο Σχ. ;; . Κάθε κόμβος είναι ένας συγκριτής δύο αριθμών. Η συνάρτηση που υλοποιεί είναι min(A,B) (ή max(A,B) για την εύρεση του μεγαλυτέρου). Επιπλέον του νικητή στην έξοδο, ο συγκριτής παρέχει ένα bit που δείχνει τον νικητή της σύγκρισης. Αυτό χρησιμοποιείται για τον εντοπισμό της εισόδου του νικητή στο δέντρο. Ο πατέρας κόμβος σε κάθε υπόδεντρο έχει τον μικρότερο στοιχείο από αυτά που βρίσκονται στα αντίστοιχα φύλλα του δέντρου. Στο σχήμα (β) φαίνεται ένα αριθμητικό παράδειγμα. Το στοιχείο-νικητής (8) ταξιδεύει, διαμέσου των κόμβων που δείχνει η γραμμή, από το φύλλο στη ρίζα.

Η δεύτερη ιδέα αφορά την παραλληλία των συγκρίσεων στο δέντρο. Με τον όρο παραλληλία εννοείται η όσο το δυνατόν ταυτόχρονη σύγκριση στοιχείων, σε όλα τα επίπεδα του δέντρου. Με τον τρόπο αυτό μειώνεται δραματικά η καθυστέρηση του υπολογισμού. Η εξήγηση της παραλληλίας απαιτεί μια πιο κοντινή ματιά στη διαδικασία της σύγκρισης. Η φύση της πράξης επιβάλει να ξεκινήσουμε από τα MSBs και να καταλήξουμε στα LSBs. Οπότε, τεμαχίζουμε τον συγκριτή δύο αριθμών σε κομμάτια ίσα με το πλήθος των bits ενός στοιχείου. Το κάθε κομμάτι συγκρίνει δύο bits και παράγει το αποτέλεσμα της σύγκρισης. Με αυτή τη διαδικασία τα bits του νικητή στην έξοδο ενός συγκριτή παράγονται διαδοχικά, με τα



Σχήμα 2.3: (a) Το διδαδικό δέντρο συγκριτών, με 8 φύλλα (N=16). Η καθυστέρηση του κυκλώματος είναι O($\log_2$N) = 4. (b) Αριθμητικό παράδειγμα.

MSBs να βγαίνουν πρώτα. Έτσι, η σύγκριση των MSBs του συγκριτή δύο αριθμών στο επίπεδο (Level-1) πάνω από τα φύλλα (Level-0) ξεκινάει με καθυστέρηση μόνο ενός κομματιού που συγκρίνει δύο bits, χωρίς να περιμένει το συνολικό αποτέλεσμα. Παρόμοια κατάσταση προκύπτει και για το επόμενο επίπεδο (Level-2). Το σημαντικό αποτέλεσμα είναι η ταυτόχρονη διάδοση των σημάτων κάτα μήκος των συγκριτών και κατά μήκος του διαδικού δέντρου.

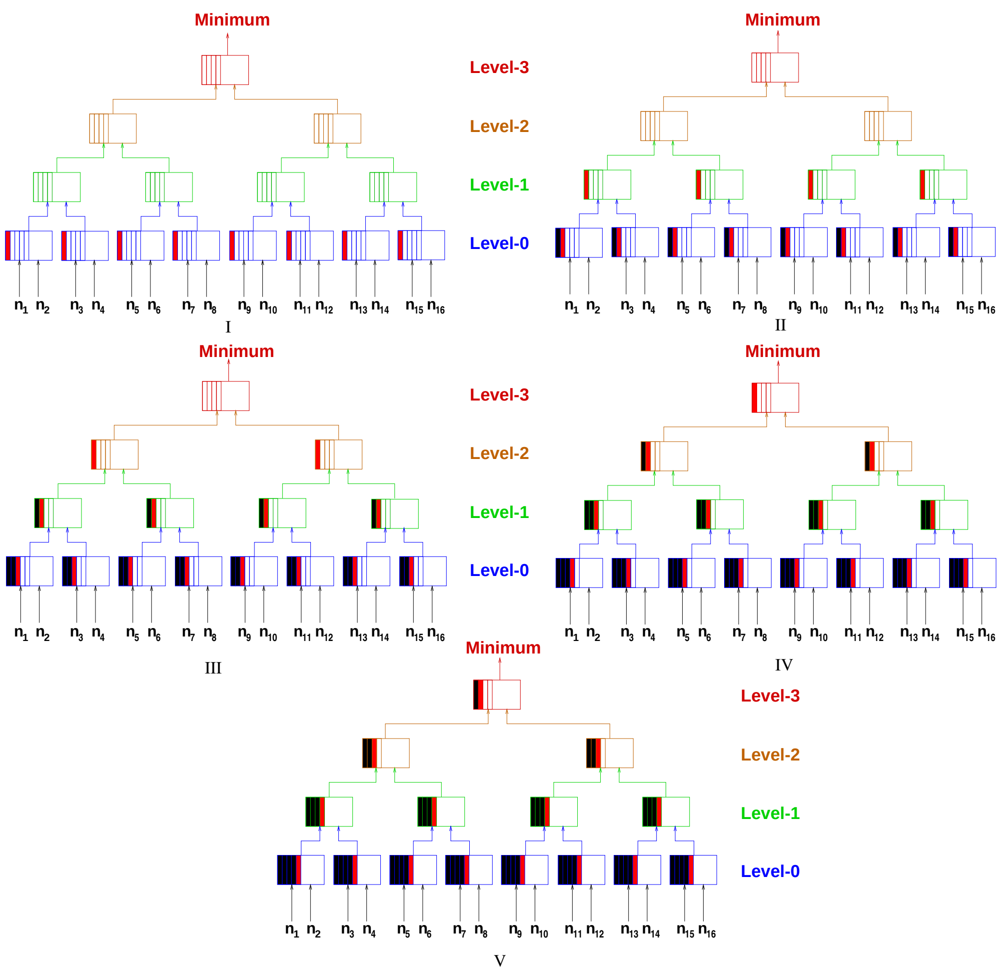
Η ιδέα εξηγείται και με το επόμενο παράδειγμα. Υποθέτουμε ότι έχουμε N στοιχεία, που το καθένα έχει k bits ($n_{k-1}, \ldots n_1, n_0$). Επιπλέον υποθέτουμε ότι όλα τα στοιχεία στα φύλλα είναι διαθέσιμα την ίδια χρονική στιγμή. $R_{i,j}$ είναι το bit i της εξόδου κάθε συγκριτή στο επίπεδο j. Στο Σχ. ;; δείχνονται διαδοχικές χρονικές στιγμές της σύγκρισης και η μεταδοση των σημάτων στο δέντρο. Παρακάτω εξηγείται η ακολουθία των γεγονότων του σχήματος. Σε κάθε bullet περιγράφονται γεγονότα που συμβαίνουν παράλληλα.

- Οι συγκριτές στα φύλλα (Level-0) παράγουν τα $R_{k-1,0}$ bits, πού είναι το αποτέλεσμα της σύγκρισης των n_{k-1} 2-MSBs. Αυτά παραδίδονται στο αμέσως επόμενο επίπεδο συγκριτών.

- Οι συγκριτές στο Level-0 ξεκινάνε τη σύγκριση των n_{k-2} bits. Τα $R_{k-2,0}$ bits παραδίδονται στο Level-1. Οι συγκριτές στο Level-1 συγκρίνουν τα $R_{k-1,0}$ bits, αφού είναι έτοιμα στην έξοδο των συγκριτών του Level-0. Τα παραγόμενα $R_{k-1,1}$ bits παραδίδονται στο Level-2.

- Οι συγκριτές στο Level-0 αρχίζουν τη σύγκριση των n_{k-3} bits. Τα παραγόμενα $R_{k-3,0}$ bits παραδίδονται στο Level-1. Οι συγκριτές στο Level-1 συγκρίνουν τα $R_{k-2,0}$ bits. Τα παραγόμενα $R_{k-2,1}$ bits, αυτής της σύγκρισης παραδίδονται στο Level-2. Οι συγκριτές στο Level-2 συγκρίνουν τα $R_{k-2,1}$ bits. Τα bits $R_{k-1,2}$ του αποτελέσματος παραδίδονται στο Level-3 του δέντρου.

- Η διαδικασία συνεχίζεται έως ότου ο συγκριτής δύο αριθμών στη ρίζα του δέντρου υπολογίσει το $R_{0,n}$bit, το οποίο είναι το LSB του νικητή.



Σχήμα 2.4: Τα διαδοχικά συμβάντα σε μία σύγκριση 16 αριθμών. Τα ενεργά κελιά είναι κόκκινα. Τα μαύρα έχουν τελειώσει τη σύγκριση, ενώ τα άσπρα δεν έχουν αρχίσει ακόμα. Στο σχήμα V το MSBτου νικητή έχει υπολογιστεί μετά από καθυστέρηση 4 κελιών.

Κεφάλαιο 3

Η Διαδικασία της Σύγκρισης

Ο συγκριτής δυο στοιχείων χωρίζεται σε ένα σύνολο από βασικά κελιά, κάθε ένα από τα οποία σύγκρνει 2 bits. Ας υποθέσουμε ότι τα δύο k-bit στοιχεία A ($a_{k-1}a_{k-2}\ldots a_0$) και B ($b_{k-1}b_{k-2}\ldots b_0$) είναι οι είσοδοι και το k-bit στοιχείο M ($m_{k-1}m_{k-2}\ldots m_0$) είναι έξοδος του συγκριτή. Σύμφωνα με την λειτουργία που εκτελείται από τη δομή του συγκριτή ο επιθυμητός αριθμός (ο A ή ο B) θα εμφανιστεί στην έξοδο. Ο Πίν. **;;** καταγράφει τους δυνατούς συνδυασμούς με τις αντίστοιχες εξόδους.

Είσοδοι	Λειτουργία	Έξοδοι
A$\geq$ B	Μέγιστο	M = A
A$\geq$ B	Ελάχιστο	M = B

Πίνακας 3.1: Λειτουργία του συγκριτή δύο αριθμών.

Η απόφαση παίρνεται από εκείνο το κελί συγκριτή 2-bit που πρώτο βρίσκει άνισα bit των στοιχείων A και B. Αυτή η απόφαση δεν μπορεί να αλλάξει από τα επόμενης σημαντικότητας κελιά που ακολουθούν στην αλυσίδα συγκρίσεως. Η φύση της λειτουργίας της σύγκρισης είναι τέτοια, ούτως ώστε οι υπολογισμοί να ξεκινήσουν από τα MSBs. Έτσι το κελί που εκτελεί την σύγκριση των δύο MSBs πρέπει να γνωστοποίησε την εκλογή του στο κελί που θα συγκρίνει τα επόμενα δύο σημαντικότερα bit. Η διαδικασία της *εκλογής* είναι ανάλογη με αυτή του κρατουμένου στη λειτουργία της πρόσθεσης. Τρεις καταστάσεις μπορούν να συμβούν για την *εκλογή*: ¨Το A είναι ίσο του B¨, ¨ Το A είναι το μικρότερο¨, ¨Το B είναι το μικρότερο¨. Οι τρεις καταστάσεις μπορούν να κωδικοποιηθούν με δύο bit. Έτσι η *εκλογή* αποτελείται από δύο σήματα C_1 και C_2.

Μια αναλλοίωτη συνθήκη που πρέπει να ικανοποιείται κατά μήκος του συγκριτή δύο στοιχείων, είναι η ακόλουθη: *Τα κελιά σύγκρισης 2-bit, που ακολουθούν αυτό το κελί που πήρε την απόφαση υπέρ του A ή του B, δεν μπορούν σε καμία περίπτωση να αλλάξουν την τιμή της απόφασης αυτής.*

3.1 Ο Συγκριτής 2 bit

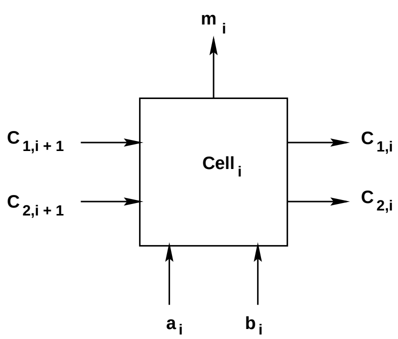
Όπως αναφέρθηκε και νωρίτερα η βασική μονάδα ενός συγκριτή δύο στοιχείων, αποτελεί το κελί που κάνει την σύγκριση 2-bit. Ένας k-bit συγκριτής δύο στοιχείων αποτελείται από k πλήθους τέτοια κελιά. Κάθε κελί εκτελεί την ίδια λειτουργία, η οποία περιλαμβάνει δύο βήματα:

1. Παράγει την *εκλογή*, που γνωστοποιείται στο επόμενο κελί που θα συγκρίνει τα επόμενης σημαντικότητας bit, περιλαμβάνοντας και την πληροφορία από την *εκλογή*, του προηγούμενου κελιού.

2. Παράγει το m_i bit του αποτελέσματος M, το οποίο είναι ένα από τα τρέχοντα bit a_i ή b_i (του A ή του B) και το παρουσιάζει στην έξοδο, χρησιμοποιώντας την τιμή της εκλογής που παράγεται από το τρέχων κελί.

Το κελί που εκτελεί την σύγκριση 2-bit και οι πιθανές τιμές της *εκλογής* φαίνονται στο Σχ. **;;** . Στον πίνακα του Σχ. **;;** , η συνεισφορά της προηγούμενης τιμής της *εκλογής* για τον καθορισμό της τρέχουσας τιμής, δεν λήφθηκε υπόψη. Η αρχική συνθήκη των σημάτων $C_{1,k}$ και $C_{2,k}$ είναι ¨Το A είναι ίσο του B¨. Η συνθήκη αυτή διατηρείται και διαδίδεται σειριακά κατά μήκος των κελιών του συγκριτή, κατά τη διάρκεια ισχύος της συνθήκης a_i=β_i, για κάθε ζευγάρι bits στις εισόδους. Μόλις η αρχική συνθήκη πάψει να ισχύει ($a_i \neq b_i$), αλλάζει και η νέα τιμή της *εκλογής* είναι μια από τις άλλες που παρουσιάζονται στον πίνακα στο Σχ. **;;** . Σύμφωνα με την αναλλοίωτη συνθήκη, η καινούργια αυτή τιμή της εκλογής δεν θα μπορεί να αλλάξει από τα επόμενης σημαντικότητας κελιά. Διατηρείται κατά μήκος του συγκριτή δύο στοιχείων, μέχρι το τέλος της σύγκρισης. Ο Πίν.

;; δείχνει τη σχέση ανάμεσα στις εισόδους και στις εξόδους του ι κελιού, σαν απόδειξη της προηγούμενης ανάλυσης. Οι πρώτες 2 γραμμές στον Πίν. ;; υποδηλώνουν την κατάσταση στην οποία ένα από τα προηγούμενα κελιά έχει πάρει την απόφαση και άλλαξε της αρχικές τιμές των σημάτων C_1 και C_2. Οι τιμές αυτές διατηρούνται και διαδίδονται διαμέσω του κελιού χωρίς να αλλάξουν. Η αρχική συνθήκη "Το Α ισούται του Β" που φθάνει στις εισόδους του τρέχοντος κελιού στις τέσσερις τελευταίες γραμμές στον Πίν. ;; μπορεί να αλλάξει αν και μόνο αν $a_i \neq b_i$.



a_i	b_i	*εκλογή*
0	0	Α ίσο με Β
0	1	το Α μικρότερο/το Β μεγαλύτερο
1	0	το Α μεγαλύτερο/το Β μικρότερο
1	1	Α ίσο με Β

Σχήμα 3.1: Το κελί από το οποίο αποτελείται ο συγκριτής δύο στοιχείων. Ο πίνακας δεξιά δείχνει τις πιθανές τιμές της εκλογής σαν συνάρτηση των bits των στοιχείων.

Είσοδοι			Έξοδοι	
Εκλογή$_{i+1}$	a_{i+1}	b_{i+1}	*Εκλογή*$_i$	m_i
το Α μικρότερο/μεγαλύτερο	ξ	ξ	το Α μικρότερο/μεγαλύτερο	α_i
το Β μικρότερο/μεγαλύτερο	ξ	ξ	το Β μικρότερο/μεγαλύτερο	β_i
Α ίσο με Β	0	0	Α ίσο με Β	0
Α ίσο με Β	0	1	το Α μικρότερο/το Β μεγαλύτερο	0/1
Α ίσο με Β	1	0	το Α μεγαλύτερο/το Β μικρότερο	0/1
Α ίσο με Β	1	1	Α ίσο με Β	1

Πίνακας 3.2: Ο πίνακας αλήθειας για την έξοδο m_i και την *εκλογή*.

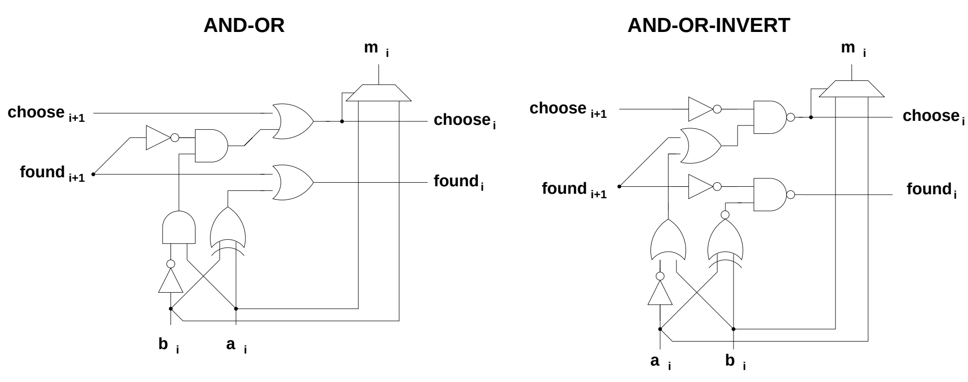
3.1.1 Το κύκλωμα

Τα σήματα C_1 και C_2 που αποτελούν την εκλογή πρέπει να κωδικοποιηθούν κατάλληλα για ελαχιστοποίηση της καθυστέρησης διάδοσης κατά μήκος του συγκριτή. Η επιλεγμένη κωδικοποίηση φαίνεται στον Πίν. ;; . Με τη βοήθεια αυτής της κωδικοποίησης μπορούμε να μετονομάσουμε τα σήματα της *εκλογής*. Παρατηρούμε ότι το C_2 γίνεται 1 όταν ο νικητής έχει καθοριστεί. Οπότε μετονομάζεται σε *found*. Το σήμα C_2 είναι 0 όταν ὸ Α είναι ο μικρότερος‟ και 1 όταν ¨ο Β είναι ο μικρότερος¨, οπότε μετονομάζεται σε *choose*.

2	α_i	β_i	*Εκλογή*
	0	0	Α ίσο με Β
	0	1	το Α μικρότερο/το Β μεγαλύτερο
	1	0	το Α μεγαλύτερο/το Β μικρότερο
	1	1	Α ίσο με Β

Πίνακας 3.3: Η κωδικοποίηση που χρησιμοποιείται για τα σήματα των κελιών.

Η κωδικοποίηση εφαρμόζεται στο *i* κελί, και ο πίνακας αληθείας για τις εισόδους και τις εξόδους παρουσιάζεται στον Πίν. ;; . Το τελευταίο χρησιμοποιείται για απλοποίηση Bool, με την βοήθεια των χαρτών Karnaugh. Είναι συχνά χρήσιμο από την πλευρά της υλοποίησης να ορίσουμε τα $choose_i$ και $found_i$ σαν συναρτήσεις κάποιων ενδιάμεσων σημάτων gc_i και gf_i. Ας σημειωθεί ότι τα gc_i και gf_i είναι συναρτήσεις των a_i και b_i. και δεν εξαρτώνται από την *εκλογή*. Τα κυκλώματα για την υλοποίηση του *i* κελιού, που βασίζονται στις ελαχιστοποιημένες εξισώσεις Bool, φαίνονται στο Σχ. ;; .



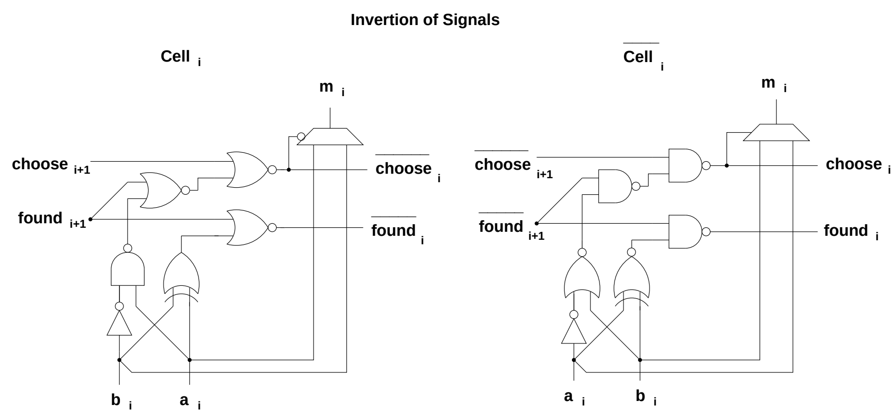
Σχήμα 3.2: α) Υλοποίηση AND-OR. β) Υλοποίηση AND-OR_INVERT

Ο αριθμός των πυλών που χρησιμοποιούνται για το *i* κελί είναι 8 για AND-OR, και 9 για AND-OR-INVERT περιλαμβάνοντας τον πολυπλέκτη που υπολογίζει το

<table>
<tr><th colspan="5">Είσοδοι</th><th colspan="3">Έξοδοι</th></tr>
<tr><th>Εκλογή$_{i+1}$</th><th>choose$_{i+1}$</th><th>found$_{i+1}$</th><th>a$_{i+1}$</th><th>b$_{i+1}$</th><th>choose$_i$</th><th>found$_i$</th><th>The choice$_i$</th></tr>
<tr><td rowspan="4">Α ίσο με Β</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>Α ίσο με Β</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>το Α μικρότερο/το Β μεγαλύτερο</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>το Β μικρότερο/το Α μεγαλύτερο</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>Α ίσο με Β</td></tr>
<tr><td rowspan="4">το Α μικρότερο/
το Β μεγαλύτερο</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td rowspan="4">το Α μικρότερο/
το Β μεγαλύτερο</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td rowspan="4">αδιάφοροι όροι</td><td>1</td><td>0</td><td>0</td><td>0</td><td>ξ</td><td>ξ</td><td rowspan="4">αδιάφοροι όροι</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>ξ</td><td>ξ</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>ξ</td><td>ξ</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>ξ</td><td>ξ</td></tr>
<tr><td rowspan="4">το Β μικρότερο/
το Α μεγαλύτερο</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td rowspan="4">το Β μικρότερο/
το Α μεγαλύτερο</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Πίνακας 3.4: Ο πίνακας αλήθειας εισόδων/εξόδων βασισμένος στην κωδικοποίηση των κρατουμένων του κελιού.

m_i .Η καθυστέρηση από το *found*$_{i+1}$ στο *found*$_i$ είναι ένα επίπεδο πύλης. Η AND-OR υλοποίηση του *i* κελιού κρύβει περισσότερο από ένα επίπεδο λογικής ανά πύλη. Οι πύλες OR και AND υλοποιούνται από πύλες NOR και NAND με αντιστροφείς από μπροστά. Έτσι κάθε μια από αυτές έχει καθυστέρηση δύο επίπεδα πυλών. Μείωση της καθυστέρησης αυτής μπορεί να επιτευχθεί χρησιμοποιώντας αντιστροφή της πόλωσης των σημάτων. Το κελί *i+1* ακολουθείται από το κελί *i* με αντίστροφης πολικότητας σήματα σε σχέση με το κελί *i+1*. Το αποτέλεσμα είναι μια σχεδίαση αποτελούμενη από NAND και NOR, που έχει καθυστέρηση ενός επιπέδου πύλης ανά συγκριτή 2-bit για τα]τλ*ςηοοσε*$_i$ και *found*$_i$. Το Σχ. **;;** αναπαριστά την προαναφερθείσα μέθοδο βελτιστοποίησης. Ο αριθμός των πυλών για κάθε υλοποίηση του *i* κελιού είναι 7. Η καθυστέρηση της *found*$_i$ είναι ένα επίπεδο πύλης. Το ίδιο ισχύει για την *choose*$_i$ για διαδοχικά κελιά. Επιπρόσθετα το fan-out και το fan-in όλων των πυλών σε ένα κελί είναι μικρότερο ή ίσο του δύο.

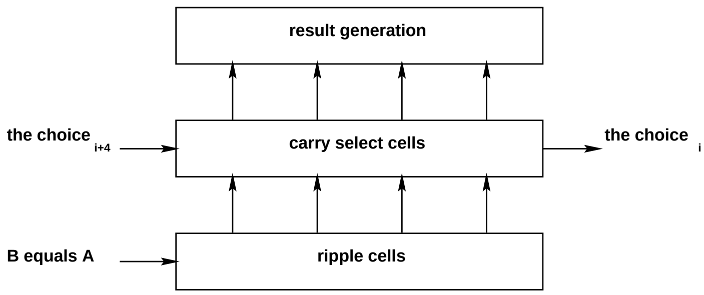


Σχήμα 3.3: Αντιστροφή των σημάτων για το κελί$_i$.

3.2 Ο συγκριτής δύο αριθμών επιλογής κρατουμένου

Στον συγκριτή ripple-carry, κάθε κελί συγκριτή 2-bit πρέπει να περιμένει τα εισερχόμενα κρατούμενα προτού ένα εξέλθων κρατούμενο παραχθεί. Ένας τρόπος να αποφύγουμε επιδέξια αυτή τη γραμμική εξάρτηση είναι να προετοιμάσουμε τα κρατούμενα, υποθέτοντας ότι τα στοιχεία είναι ίσα. Μόλις η πραγματική τιμή του εισερχόμενου κρατουμένου γίνει γνωστή, το σωστό αποτέλεσμα επιλέγεται με ένα απλό κελί επιλογής κρατουμένου carry select cell (cs-cell).

Ας υποθέσουμε μια αλυσίδα από τέσσερα κελιά συγκριτών (Σχ. **;;**)2-bit, υπολογίζοντας τα bits από το *i+3* έως το *i*. Αντί να περιμένουμε την άφιξη του *choice*$_{i+4}$, η σύγκριση ξεκινάει, σαν να ήταν όλα τα προηγούμενα bit κ-1,$\cdots$,ι+4 ίσα. Όταν το *choice*$_{i+4}$ τελικά πάρει τιμή, επιλέγεται είτε αυτό είτε το *choice*$_i$. Η επιλογή γίνεται σύμφωνα με την τιμή του *choice*$_{i+4}$. Εάν η τελευταία δείχνει το νικητή, τα κρατούμενα που παράγονται από το παρόν βλοςκ απορρίπτονται. Διαφορετικά απορρίπτονται τα εισερχόμενα κρατούμενα. Η κατάσταση αυτή αναδεικνύει μια ύπαρξη προτεραιότητας, βασισμένη στην αναλλοίωτη συνθήκη του συγκριτή δύο στοιχείων. Το cs-cell (carry select cell) είναι υπεύθυνο για την κατάλληλη επιλογή των κρατουμένων.



Σχήμα 3.4: Συγκριτής επιλογής κρατουμένου τεσσάρων bit: τοπολογία.

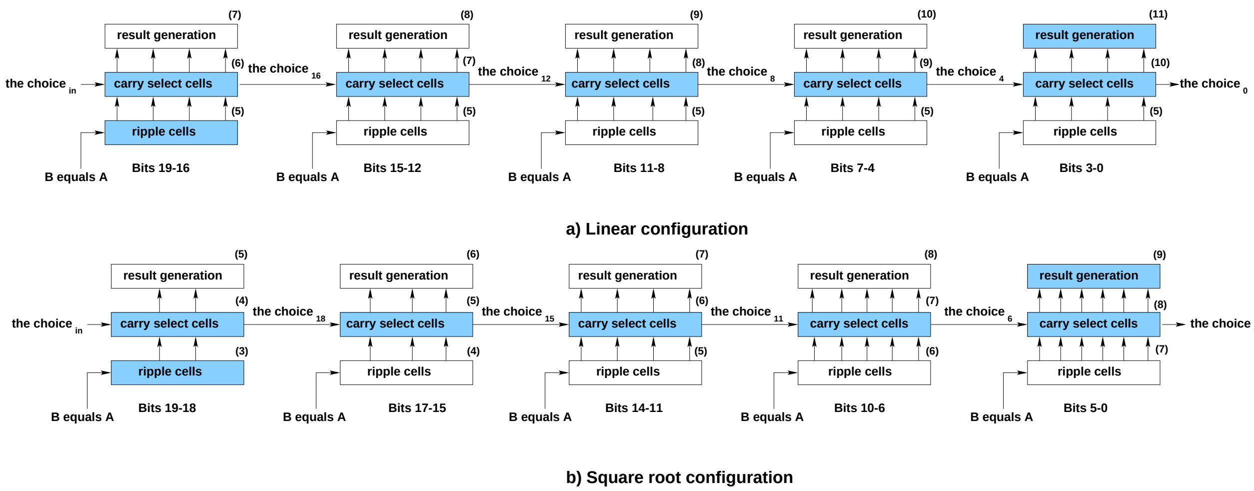
Ας φανταστούμε την περίπτωση ενός 20-bit συγκριτή επιλογής κρατουμένου δύο στοιχείων, ο οποίος κατασκευάζεται από τοπολογίες επιλογής κρατουμένου των 4-bit. Όπως αναφέρθηκε και νωρίτερα η καθυστέρηση διάδοσης του κάθε cell είναι είναι ένα επίπεδο πύλης ανά bit. Οι χειρότερες περιπτώσεις χρόνων άφιξης των σημάτων σε σχέση με την χρονική στιγμή που ενεργοποιείται η είσοδος στους διαφορετικούς κόμβους του δικτύου, σημειώνονται και φαίνονται στο Σχ. ;; . Η ανάλυση αυτή αναδεικνύει το γεγονός ότι το κρίσιμο μονοπάτι του συγκριτή διαδίδεται (ripples) διαμέσου του δικτύου των cs-cell's. Από την πλευρά του κυκλώματος, μπορούμε να υπολογίσουμε μία τάξη μεγέθους της χειρότερης περίπτωσης καθυστέρησης διάδοσης, για την συνδεσμολογία.

$$\mathrm{t_{comparator}} = \mathrm{t_g} + (\mathrm{k}) \cdot \mathrm{t_{carry}} + \left(\frac{\mathrm{k}}{\mathrm{K}}\right) \cdot \mathrm{t_{cs-cell}} + \mathrm{t_{mux}}$$

Στην παραπάνω σχέση τα k και K παριστάνουν το συνολικό αριθμό bits και τον αριθμό των bits ανά στάδιο αντίστοιχα. Η καθυστέρηση του κρατουμένου διαμέσου μιας συνδεσμολογίας 4-bit είναι ανάλογη με το μήκος του σταδίου ή διαφορετικά ισούται με k*t_{carry}.

Η καθυστέρηση διάδοσης του συγκριτή είναι για μια ακόμη φορά γραμμικώς ανάλογη του k. Η γραμμική αυτή συμπεριφορά οφείλεται στο ότι η ¨εκλογή¨ πρέπει να διαδοθεί διαμέσου όλων των σταδίων.

Η επόμενη τοπολογία στο Σχ. ;; (κάτω) αναπαριστά μια μεγάλη βελτίωση στην καθυστέρηση του συγκριτή του Σχ. ;; (πάνω). Απλή επισκόπηση του γραμμικού συγκριτή carry select, αναδεικνύει μια μεγάλη ευκαιρία. Ας σκεφτούμε τα cs-cells του τελευταίου σταδίου του συγκριτή. Οι είσοδοι αυτών των cs-cells είναι οι



Σχήμα 3.5: α) Συγκριτής με γραμμική κσθυστέρηση . β) Συγκριτής με καθυστέρηση τετραγωνικής ρίζας.

εκλογές των κελιών διάδοσης (ripple) της τρέχουσας συνδεσμολογίας και οι ¨εκλογές¨ του προηγούμενου σταδίου σύγκρισης cs-cells. Υπάρχει σημαντική διαφορά ανάμεσα στους χρόνους άφιξης των σημάτων αυτών. (Οι καθυστερήσεις φαίνονται στις παρενθέσεις). Η *εκλογή* του τρέχοντος block συγκρίσεως είναι έτοιμη, πολύ πριν φθάσει η *εκλογή* του προηγούμενου block. Είναι λογικό να εξισώσουμε την καθυστέρηση των σημάτων διαμέσω και των δυο μονοπατιών. Αυτό μπορεί να επιτευχθεί προσθέτοντας σταδιακά περισσότερα bits στα διαδοχικά στάδια στον συγκριτή δύο στοιχείων, απαιτώντας σταδιακά περισσότερο χρόνο για την δημιουργία των σημάτων κρατουμένου. Για παράδειγμα το πρώτο στάδιο μπορεί να συγκρίνει 2 bits, το δεύτερο περιέχει 3, το τρίτο έχει 4 κ.ο.κ. όπως παρουσιάστηκε στο Σχ. **;;** (κάτω). Οι χρόνοι άφιξης δείχνουν ότι αυτή η τοπολογία συγκρίσεως είναι γρηγορότερη από την προηγούμενη. Μπορεί να παρατηρηθεί ότι η ασυμφωνία στους χρόνους άφιξης στους κόμβους των cs-cells έχει εξαφανισθεί. Το απλό κόλπο του να κάνουμε τα στάδια του συγκριτή προοδευτικά μεγαλύτερα, οδηγούν σε μια δομή με υπο-γραμμικά χαρακτηριστικά καθυστέρησης. Κάτι τέτοιο φαίνεται από την ακόλουθη ανάλυση. Ας υποθέσουμε ότι ένας συγκριτής k-bit περιέχει P στάδια και ότι το πρώτο στάδιο συγκρίνει K bits. Ένα επιπρόσθετο bit προστίθεται σε κάθε επακόλουθο στάδιο. Τότε ισχύει η ακόλουθη σχέση:

$$k = K + (K+1) + (K+2) + \ldots + (K+P-1) =$$
$$= K \cdot P + \frac{P \cdot (P-1)}{2} = \frac{P^2}{2} + P \cdot \left(K - \frac{1}{2}\right)$$

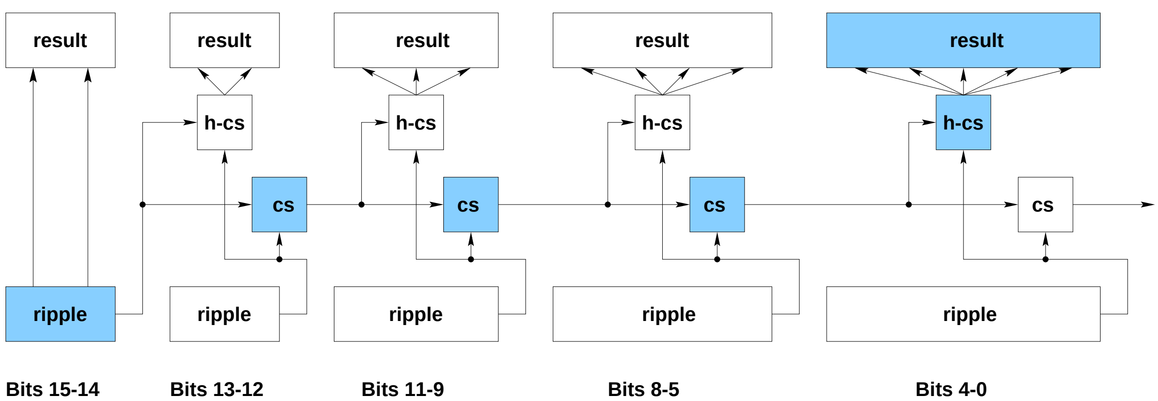
Εάν K≪k, ο πρώτος όρος κυριαρχεί και η παραπάνω εξίσωση απλοποιείται στην:

$$\mathrm{k} \cong \frac{\mathrm{P}^2}{2} \Rightarrow \mathrm{P} = \sqrt{2 \cdot \mathrm{k}}$$

Έτσι η καθυστέρηση μπορεί να εκφραστεί στην:

$$\mathrm{t_{comparator}} = \mathrm{t_g} + \mathrm{K} \cdot \mathrm{t_{carry}} + (\sqrt{2 \cdot \mathrm{k}}) \cdot \mathrm{t_{cs-cell}} + \mathrm{t_{mux}}$$

Η καθυστέρηση είναι ανάλογη της $\sqrt{k}$ για μεγάλους συγκριτές (K≪k), ή $\mathrm{t_{comparator}} = \mathrm{O}(\sqrt{\mathrm{k}})$. Το κύκλωμα συγκριτή επιλογής κρατουμένου τετραγωνικής ρίζας επιδέχεται περαιτέρω βελτιστοποίηση, με μια καλύτερη ματιά. Η αρχική τιμή της eklog'hc συνήθως δεν διατίθεται και τα κελιά επιλογής κρατουμένου του αρχικού block είναι περιττά. Το δεύτερο κατά σειρά cs-cell παίρνει τα κρατούμενα κατευθείαν από τα κελιά
τλριππλε. Ένα άλλο θέμα αποτελεί το γραμμικά αυξανόμενο fan-out στην έξοδο των cs-cells, καθώς περισσότερα blocks προστίθενται στην αλσίδα του συγκριτή. Το ίδιο ισχύει και για τους αθροιστές τύπου carry select. Η ιδέα είναι να μεταφέρουμε το fan-out από το μονοπάτι που ανακοινώνει την *εκλογή* στο επόμενης σημαντικότητας block, στο μονοπάτι που παράγει το αποτέλεσμα. Το τελευταίο δεν είναι στο κρίσιμο μονοπάτι, οπότε μπορεί να καθυστερήσει περισσότερο από το κρατούμενο. Ο σχεδιασμός των ιδεών αυτών αναπαρίσταται στο Σχ. **;;** . ...



Σχήμα 3.6: Ο συγκριτής επιλογής κρατουμένου έπειτα από τη βελτιστοποίηση του κρίσιμου μονοπατιού.

Τα block επιλογής κρατουμένου (h-cs) που φαίνονται στο Σχ. **;;** είναι cs-blocks χωρίς την πύλη που υπολογίζει το κρατούμενο found. Το τελευταίο δεν χρειάζεται για το υπολογισμό των αποτελεσμάτων. Το fan-out των block επιλογής κρατουμένου που ανήκουν στο κρίσιμο μονοπάτι είναι σταθερό και ίσο με 2. Για μεγάλες τιμές του k το fan-out των h-cs κελιών είναι σημαντικό. Μια λύση είναι να χρησιμοποιήσουμε ένα δένδρο από buffers από το κελί οδηγό στα οδηγούμενα κελιά. Η δομή αυτή έχει λογαριθμική καθυστέρηση.

Η εξίσωση για την καθυστέρηση τώρα πρέπει να οριστεί ξανά. Η ακόλουθη λοιπόν σχέση ισχύει: $\mathrm{k} = \mathrm{K} + \mathrm{K} + (\mathrm{K}+1) + (\mathrm{K}+2) + \ldots + (\mathrm{K}+\mathrm{P}-1) =$
$= \mathrm{K}\cdot(\mathrm{P}+1) + \frac{\mathrm{P}\cdot(\mathrm{P}-1)}{2} = \frac{\mathrm{P}^2}{2} + \mathrm{P}\cdot(\mathrm{K}-\frac{1}{2}) + \mathrm{K}$

Λύνοντας την παραπάνω εξίσωση για P, $\mathrm{P}^2 + (2\cdot\mathrm{K}-1)\cdot\mathrm{P} + 2\cdot(\mathrm{K}-\mathrm{k}) = 0$

παίρνουμε το ακόλουθο αποτέλεσμα: $\mathrm{P} = \frac{-(2\cdot\mathrm{K}-1)+\sqrt{(2\cdot\mathrm{K}-1)^2+8\cdot(\mathrm{k}-\mathrm{K})}2}{}$

Έτσι η καθυστέρηση για τον συγκριτή επιλογής κρατουμένου μπορεί να εκφραστεί όπως προηγουμένως με την σχέση: $\mathrm{t_{comparator}} = \mathrm{t_g} + \mathrm{K}\cdot\mathrm{t_{carry}} + \mathrm{P}\cdot\mathrm{t_{cs-cell}} + (\log_2\mathrm{P})\cdot\mathrm{t_{mux}}$

Εάν K≪k το αποτέλεσμα είναι το ίδιο με την πριν-την-βελτιστοποίηση συνδεσμολογία, οπότε: $\mathrm{P} = \sqrt{2\cdot\mathrm{k}}$

3.3 Το Διαδικό Δέντρο Συγκριτών

Το δυαδικό δένδρο σχεδιάζεται με τοπολογία συγκριτή επιλογής κρατουμένου, για τον συγκριτή δύο στοιχείων. Απαιτείται η εξίσωση των καθυστερήσεων ανάμεσα σε συγκριτές δύο στοιχείων που βρίσκονται σε συνεχόμενα επίπεδα του δένδρου. Η καθυστέρηση που παρουσιάζει το κομμάτι της διάδοσης (ripple block) μπορεί να παρθεί από το Σχ. **;;** . Εκεί είναι σχεδιασμένο ένα δυαδικό δένδρο πέντε επιπέδων, με διαφορετικούς συγκριτές δυο στοιχείων σε κάθε επίπεδο του δένδρου. Μόνο ένας συγκριτής ανά επίπεδο φαίνεται, για απλότητα. Το επίπεδο 0 κατασκευάζεται με ένα συγκριτή δυο στοιχείων όπως φαίνεται στο Σχ. **;;** . Ένας τέτοιος σχεδιασμός δεν υπάρχει στα επόμενα επίπεδα, όπου οι συγκριτές είναι υβριδικοί: ripple και επιλογής κρατουμένου. Το ποσοστό του ripple καθορίζεται από ένα απλό κανόνα: Όταν κινούμαστε ένα επίπεδο πάνω μεταφέρουμε το τελευταίο κομμάτι του συγκριτή δυο στοιχείων από το τμήμα επιλογής κρατουμένου σε αυτό του ripple του συγκριτή. Ο κανόνας αυτός εξισώνει της καθυστερήσεις διάδοσης των διαφορετικών τμημάτων του συγκριτή δυο στοιχείων, σε κάθε επίπεδο του δένδρου. Η εξίσωση των καθυστερήσεων είναι απαραίτητη για να διατηρήσουμε το στοιχείο του παραλληλισμού και να είναι ταχεία η διάδοση της *εκλογής* κατά μήκος του συγκριτή δυο στοιχείων, δια μέσω των cs-cells. Έτσι απαιτείται προσεκτικός καταμερισμός των bit του στοιχείου αποτελέσματος για να τροφοδοτηθεί το επόμενο επίπεδο του συγκριτή δυο στοιχείων.

Όλοι οι συγκριτές υπολογίζουν παράλληλα σε όλα τα επίπεδα. Το κρίσιμο μονοπάτι του κυκλώματος πρέπει να οριστεί. Η πολυπλοκότητα θα υπολογιστεί από τα γνωστά χαρακτηριστικά του διαδικού δένδρου και από τον συγκριτή επιλογής κρατουμένου. Ξεκινώντας από το κατώτερο επίπεδο του συγκριτή, το κρίσιμο μονοπάτι φαίνεται στο Σχ. **;;** . Έτσι τα σήματα της eklog'hc πρέπει να διαδοθούν κατά μήκος του συγκριτή των δυο στοιχείων και η πολυπλοκότητα είναι $t_{comparator} = O(\sqrt{k})$ για k-bit στοιχεία. Η υπόλοιπη καθυστέρηση είναι η διάδοση, από τα φύλλα στην ρίζα του δένδρου. Η πολυπλοκότητα της τοπολογίας αυτής προσδιορίζεται με τον ακόλουθο τρόπο: Ας υποθέσουμε ότι l είναι ο αριθμός των bit που υπολογίζονται από το τελευταίο τμήμα του συγκριτή δυο στοιχείων, που βρίσκεται στα φύλλα. Τότε

$$l = K + P - 1$$

Για όλους τους συγκριτές στο δένδρο, συγκρίνοντας N στοιχεία, το άθροισμα S αυτών των τμημάτων είναι $S = l - 1 + l - 2 + \cdots l - \log_2 N \Rightarrow$

$$S = \log_2 N \cdot l - \frac{\log_2 N \cdot (\log_2 N + 1)}{2} \Rightarrow$$

$$S = \log_2 N \cdot [l - \frac{\log_2 N + 1}{2}]$$

Αντικαθιστώντας το l $S = \log_2 N \cdot [K + P - 1 - \frac{\log_2 N + 1}{2}]$

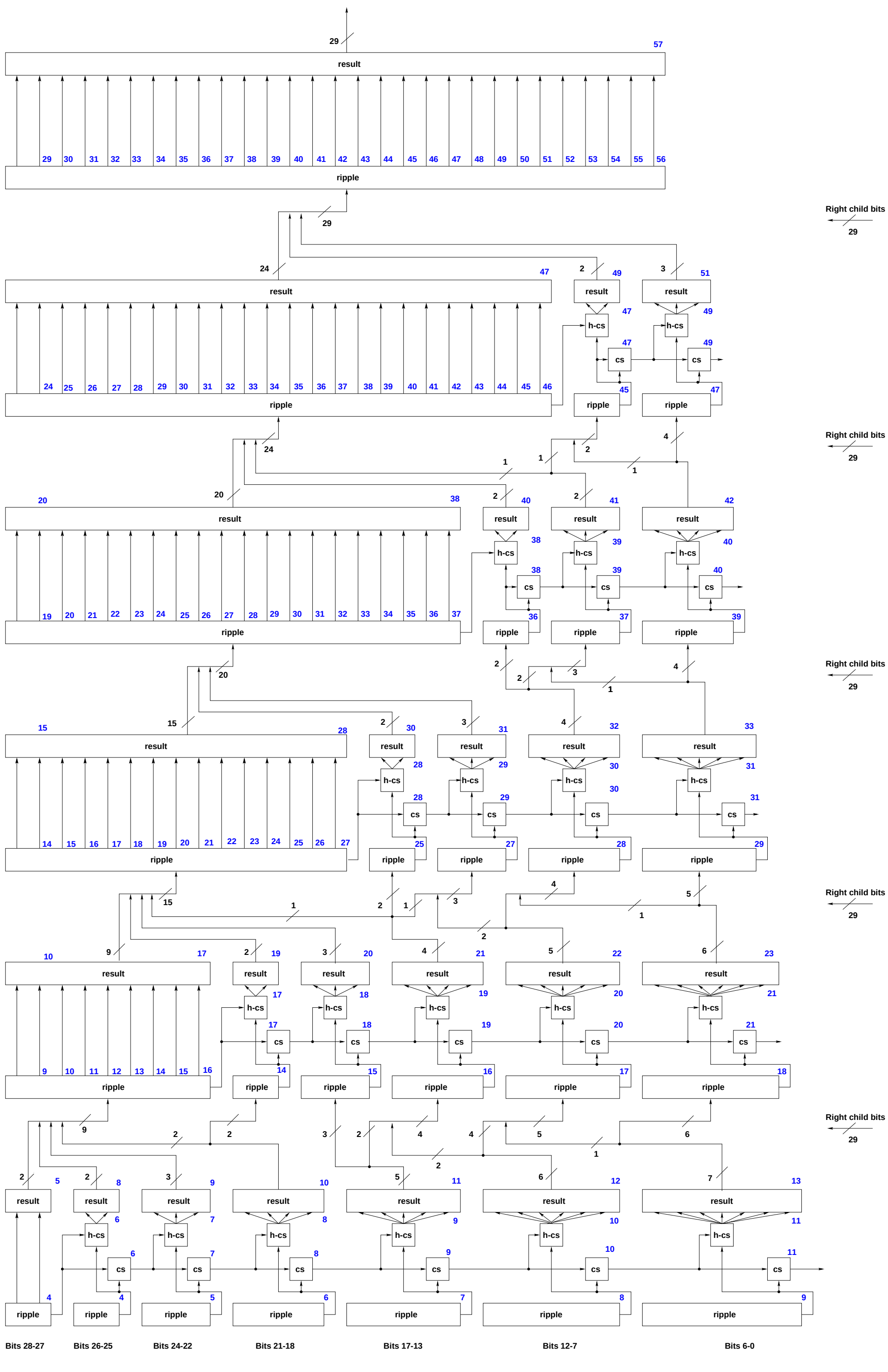
Συνήθως K=2 $S = \log_2 N \cdot [P - \frac{\log_2 N - 1}{2}]$

$$P = \sqrt{2 \cdot k} \Rightarrow$$

και με αντικατάσταση στο Σ προκύπτει $S = \log_2 N \cdot [\sqrt{2 \cdot k} - \frac{\log_2 N - 1}{2}]$

Έτσι καταλήγουμε σε μια εξίσωση του S, έχοντας μόνο τον αριθμό των στοιχείων N

και των αριθμό των bits ανά στοιχείο k. Η εξίσωση αυτή μας δίνει την συνολική πολυπλοκότητα του διαδικού δένδρου $O(\sqrt{k}) + O(\log_2 N \cdot [\sqrt{k} - \log_2 N])$
Αυτή η σχέση είναι έγκυρη για $\sqrt{k} > \log_2 N)$
Δεν λαμβάνεται υπόψη η πολυπλοκότητα των hs-cells με το μεγάλο fan-out. Η πολυπλοκότητα αυτή εξαρτάται λογαριθμικά από το l και απορρίπτεται σε αυτήν την προσέγγιση.



Σχήμα 3.7: Διαδικό δέντρο συγκριτών 5 επιπέδων με υβριδικούς συγκριτές δύο στοιχείων.

Κεφάλαιο 4

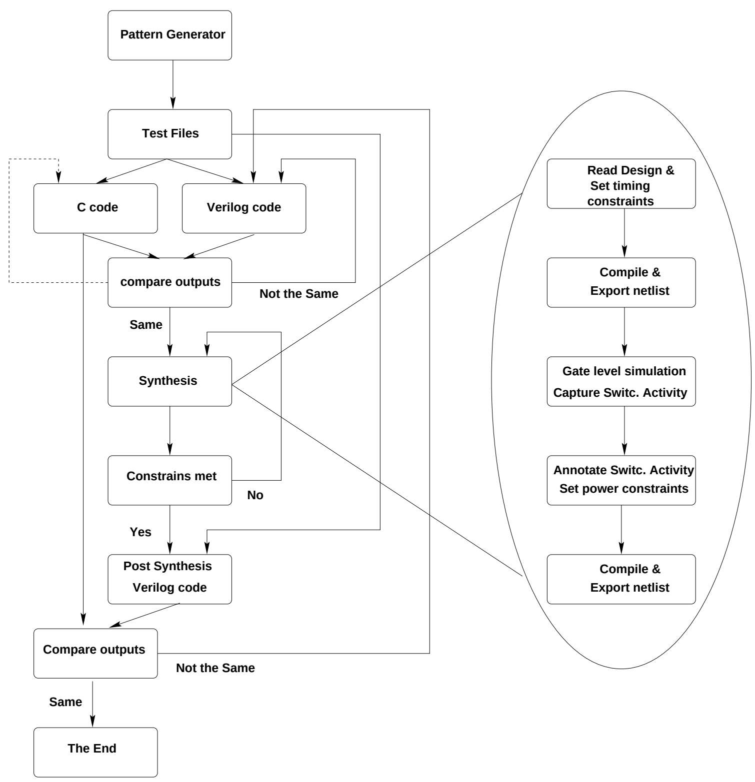
Αποτελέσματα Σύνθεσης

Ροή Σχεδίασης

Η διαδικασία-ροή σχεδίασης design flow ενός κυκλώματος παίζει σημαντικό ρόλο στην τελική του απόδοση. Πολλές ροές σχεδίασης προτείνονται για επαλήθευση verification και σύνθεση. Αυτή που χρησιμοποιείται εδώ περιλαμβάνει αρκετά βήματα για βελτιστοποίηση της ισχύος και της καθυστέρησης. Στο Σχ. **;;** φαίνεται η διαδικασία που χρησιμοποιείται για αυτό το κύκλωμα.

Ο C κώδικας για τον συγκριτή διαδικού δέντρου είναι πολύ απλός. Δημιουργεί τον αριθμό των στοιχείων που θα εισαχθούν στον κώδικα verilog. Αυτά τα στοιχεία γράφονται σε αρχείο το οποίο χρησιμοποιεί ο κώδικας verilog ως είσοδο. Το block της σύνθεσης κρύβει πολύ πληροφορία, όπως φαίνεται στο Σχ. **;;** . Η ροή σύνθεσης που ακολουθείται χρησιμοποιεί προσομοίωση επιπέδου πυλών για να μετρήσει το ανοιγόκλειμα των κόμβων του κυκλώματος. Αυτό βοηθάει το εργαλείο σύνθεσης να πραγματοποιεί πιο ακριβείς υπολογισμούς κατανάλωσης ισχύος.

Στην στρατηγική βοττομ-υπ μεμονωμένα κυκλώματα μεταφράζονται ξεχωριστά (τλςομπιλε). Μετά από επιτυχή μετάφραση τα κυκλώματα δεν ξαναπειράζονται (dont_touch) για να αποφευχθούν περαιτέρω αλλαγές κατά την διάρκεια επόμενων φάσεων μετάφρασης. Έπειτα τα μεταφρασμένα κυκλώματα συλλέγονται για να συνθέσουν υψηλότερου επιπέδου της ιεραρχίας κυκλώματα. Αυτή η διαδικασία σύνθεσης συνεχίζεται για όλη την ιεραρχία μέχρι να συνθέσουμε το κορυφαίο block. Η μέθοδος αυτή επιτρέπει την μετάφραση μεγάλων κυκλωμάτων, διότι ο Δεσιγν ὃμπιλερ δεν χρειάζεται να φορτώσει όλα τα κυκλώματα στην μνήμη την



Σχήμα 4.1: Η ροή σχεδίασης

ίδια χρονική στιγμή.

4.1 Τεχνολογία Σύνθεσης

Τα εργαλεία CAD που χρησιμοποιήθηκαν για επαλήθευση και σύνθεση είναι Cadence Verilog-XL και Synopsys. Η τεχνολογία που χρησιμοποιήθηκε για την υλοποίηση του κυκλώματος είναι 0.18μm CMOS από την Virtual Silicon (δόθηκε από την Europractice).

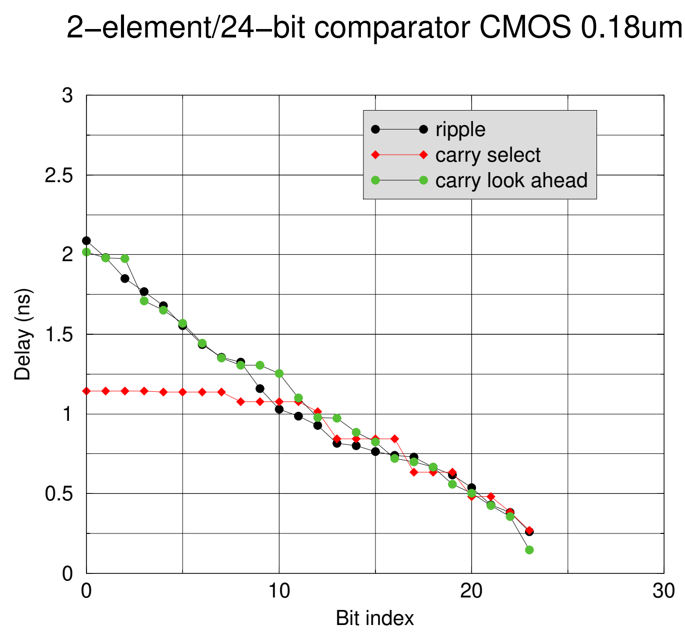
4.2 Δυαδικό Δέντρο Συγκριτών

Το Δυαδικό δέντρο συγκριτών όπως φαίνεται στο Κεφ. **;;** μπορεί να σχεδιαστεί με τρεις τρόπους σύμφωνα με την υλοποίηση του συγκριτή δύο στοιχείων: ripple, carry select και carry look ahead. Οι παράμετροι που χρησιμοποιήθηκαν είναι ο αριθμός των στοιχείων (N: N∈{2, 4, 16, 64, 256}) και ο αριθμός των bits ανά στοιχείο (k: k∈{8, 16, 24}). Οι περιορισμοί καθυστέρησης αρχικά για το δέντρο ήταν 10ns. Πρώτα πέρασε από σύνθεση ο συγκριτής δύο bit και έπειτα ο συγκριτής δύο στοιχείων ο οποίος χρησιμοποιεί το cell του συγκριτή δύο bit. Στο σημείο αυτό είναι απαραίτητο να ξεκινήσει η χρήση του Power Compiler για την βελτιστοποίηση της ισχύος του συγκριτή δύο στοιχείων. Αυτή η συμβολή του στα πρώτα επίπεδα σύνθεσης παράγει καλύτερα αποτελέσματα από ότι στα τελευταία.

4.2.1 Αποτελέσματα Καθυστέρησης

Τα αποτελέσματα για τις τρεις υλοποιήσεις του συγκριτή δύο στοιχείων φαίνονται στο Σχ. **;;** .

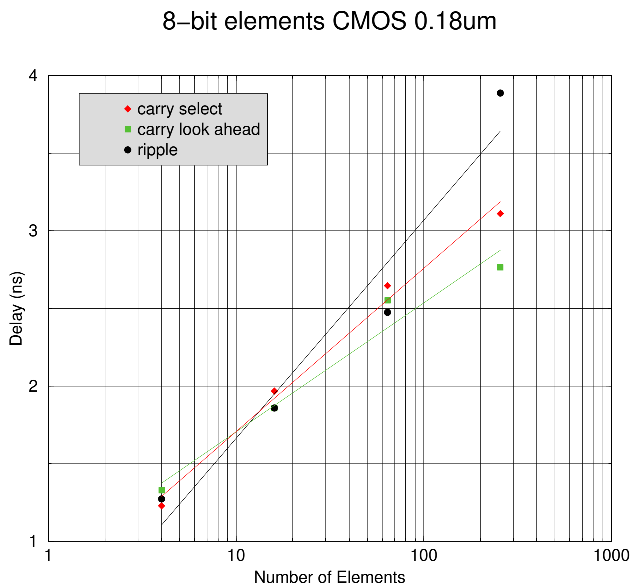
Ο συγκριτής επιλογής κρατουμένου εμφανίζει καλύτερη απόδοση από τους άλλους. Στο Σχ. **;;** χρησιμοποιούνται 24-bit συγκριτές. Όπως αποδείχτηκε νωρίτερα ο συγκριτής επιλογής κρατουμένου είναι γρηγορότερος από τον ripple και τον carry look ahead. Η απόδοση της carry look ahead τοπολογίας είναι μικρή εξαιτίας του μεγάλου fan-out, το οποίο αυξάνει γραμμικά από το MSB στο LSB, έτσι τα αποτελέσματα της καθυστέρησης είναι σχεδόν γραμμικά. Η επόμενη



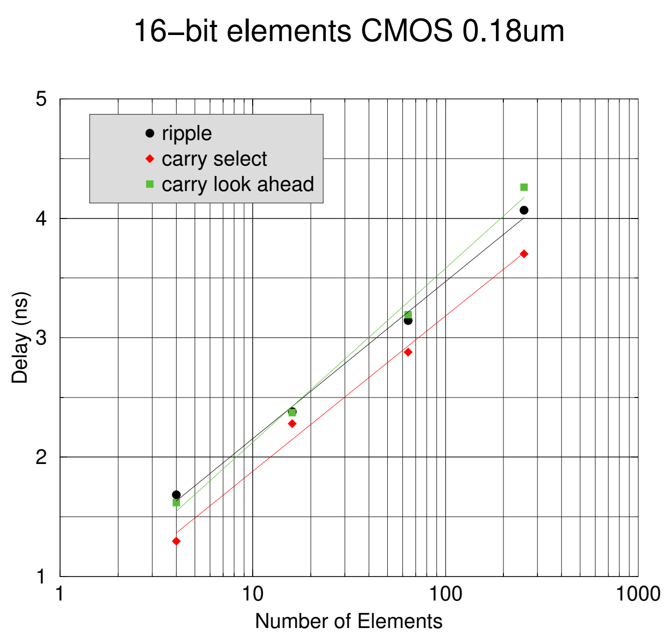
Σχήμα 4.2: Αποτελέσματα καθυστέρησης για τις τρείς τοπολογίες.

σύγκριση που έγινε είναι για το δυαδικό δέντρο. Τα Κεφ. ;; - Σχ. ;; δείχνουν καθυστερήσεις για τις τιμές του πλάτους και τον αριθμό των στοιχείων. Αρχικά χρησιμοποιούνται 8-βιτ στοιχεία και τα αποτελέσματα φαίνονται στο σχήμα 39.

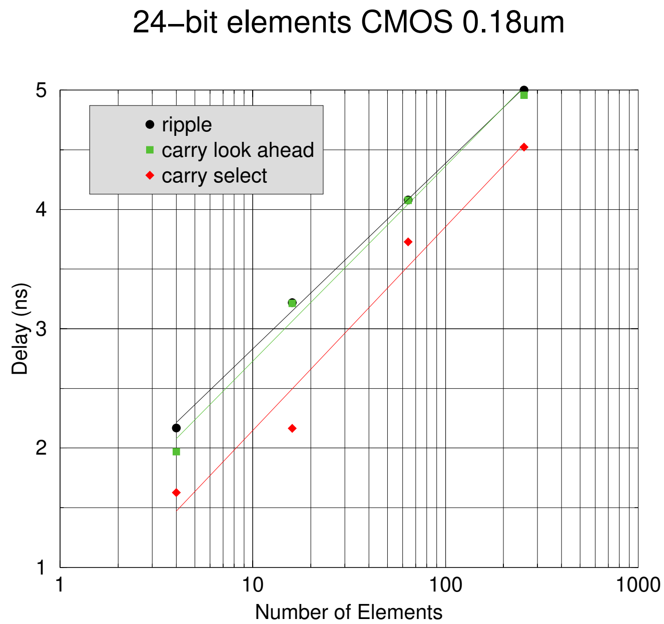
Ο αριθμός των βιτσ είναι μικρός έτσι ο συγκριτής επιλογής κρατουμένου δεν θα έχει μεγαλύτερη απόδοση σε σχέση με τις άλλες υλοποιήσεις. Για μικρότερο αριθμό στοιχείων η απόδοση είναι ίδια για όλες τις τοπολογίες. Όσο ο αριθμός στοιχείων μεγαλώνει το carry look ahead κερδίζει, γεγονός που συμβαίνει διότι ο αριθμός των bits των στοιχείων είναι μικρός και οδηγεί σε μικρό fan-in και fan-out. Στη συνέχεια 16-bit στοιχεία χρησιμοποιούνται για το δυαδικό δέντρο. Τα αποτελέσματα φαίνονται στο Σχ. ;; . Η τοπολογία επιλογής κρατουμένου έχει καλύτερη απόδοση από τις άλλες δύο. Το εύρος του στοιχείου είναι αρκετό για να αναδείξει τα καλά χαρακτηριστικά του συγκριτή. Ο ripple και ο carry look ahead συναγωνίζονται και ο πρώτος είναι λίγο πιο γρήγορος από τον δεύτερο σε πολλούς αριθμούς. Τέλος μετράται η καθυστέρηση για τον υπολογισμό του ελαχίστου μεταξύ 24-bit στοιχείων. Τα αποτελέσματα φαίνονται στο Σχ. ;; . Τοπολογία επιλογής κρατουμένου είναι γρηγορότερη από τις άλλες δύο, ενώ η καθυστέρηση της ripple και της carry look ahead είναι ίδιες.



Σχήμα 4.3: Σύγκριση αποτελεσμάτων καθυστέρησης για το διαδικό δέντρο με 8-bit στοιχεία.



Σχήμα 4.4: Σύγκριση αποτελεσμάτων καθυστέρησης για το διαδικό δέντρο με 16-bit στοιχεία.



Σχήμα 4.5: Σύγκριση αποτελεσμάτων καθυστέρησης για το διαδικό δέντρο με 24-bit στοιχεία.

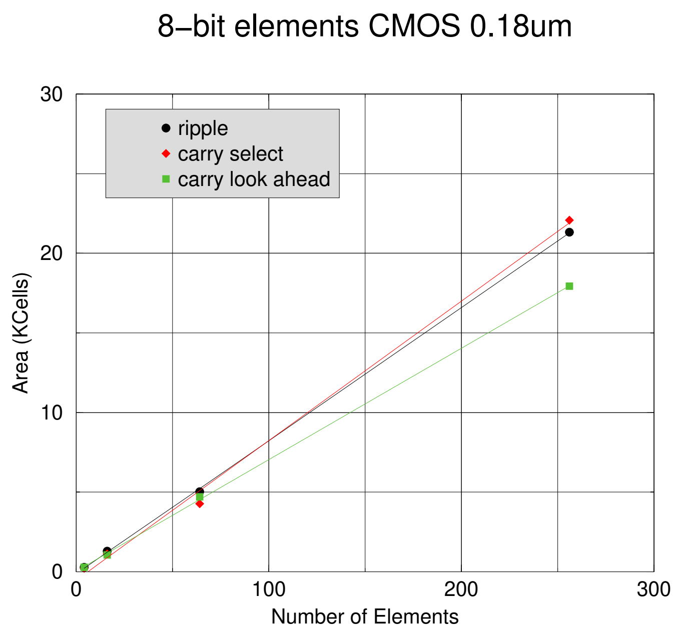
Καταλήγοντας για τις καθυστερήσεις του διαδικού δέντρου, ο συγκριτης επιλογής κρατουμένου έχει καλύτερη απόδοση από τους άλλους δύο. Οι καθυστερήσεις αυξάνουν λογαριθμικά με τον αριθμό των στοιχείων. Η συμπεριφορά αυτή κληρονομείται από το δυαδικό δέντρο.

4.2.2 Αποτελέσματα Εμβαδού Κυκλώματος

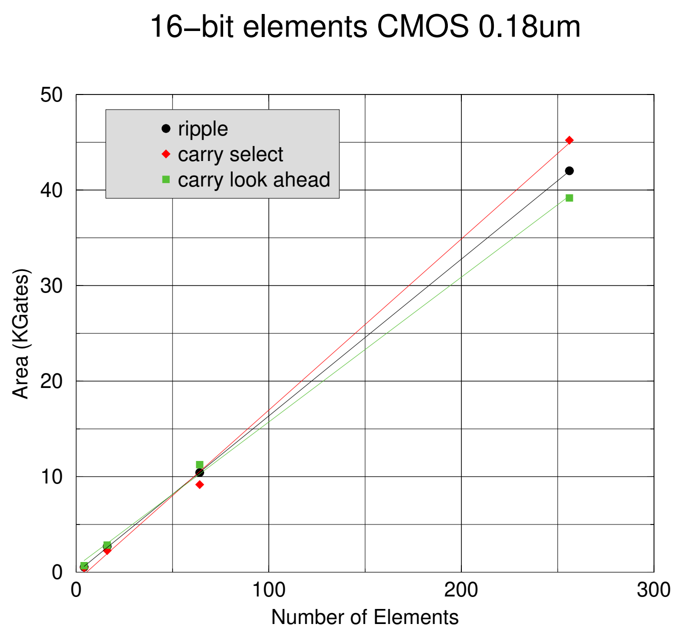
Στα Σχ. **;;** - Σχ. **;;** φαίνονται αποτελέσματα εμβαδού για το δυαδικό δέντρο. Το εμβαδόν αυξάνει γραμμικά με τον αριθμό των στοιχείων και μετράται σε cells, που είναι πιο ακριβές από mm^2 διότι επέρχονται αλλαγές μετά το place and root. Το εμβαδόν για 24-bit, 256-στοιχείων, ripple τοπολογίας είναι περίπου $0.75mm^2$.

4.2.3 Αποτελέσματα Ισχύος

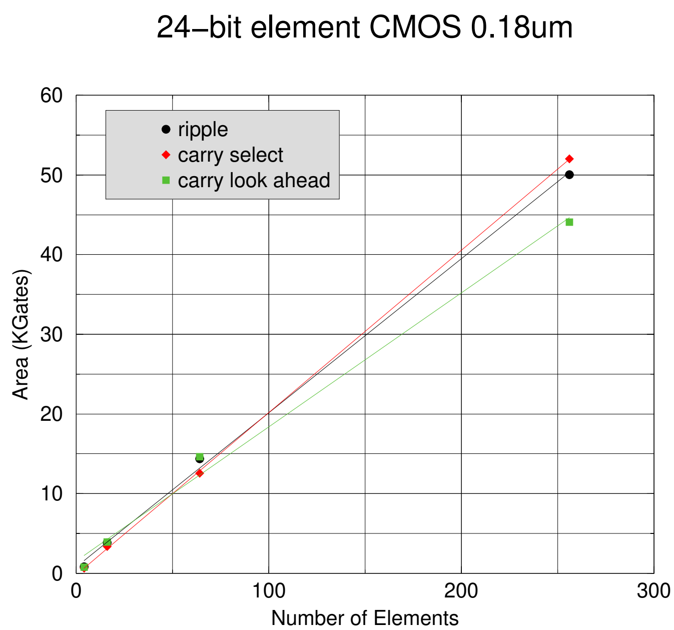
Σε αύτο το μέρος παρουσιάζονται αποτελέσματα σύνθεσης για ισχύ. Οι τιμές που φαίνονται στα σχήματα είναι κοντά στην χειρότερη περίπτωση. Αρχικά εισάγονται μηδενικά στην είσοδο του διαδικού δέντρου συγκριτών. Στον επόμενο κύκλο ένα μη μηδενικό σύνολο στοιχείων εισάγεται στις εισόδους. Αυτές οι τιμές



Σχήμα 4.6: Σύγκριση αποτελεσμάτων εμβαδού για το διαδικό δέντρο με 8-bit στοιχεία.

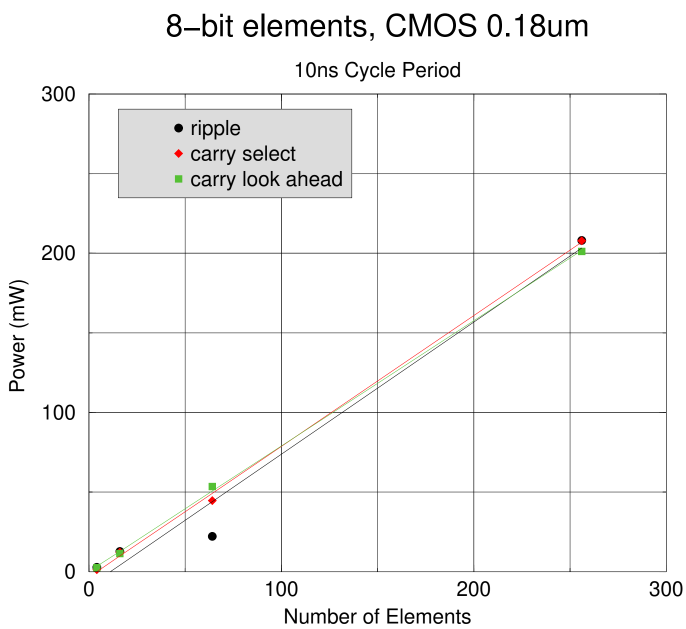


Σχήμα 4.7: Σύγκριση αποτελεσμάτων εμβαδού για το διαδικό δέντρο με 16-bit στοιχεία.

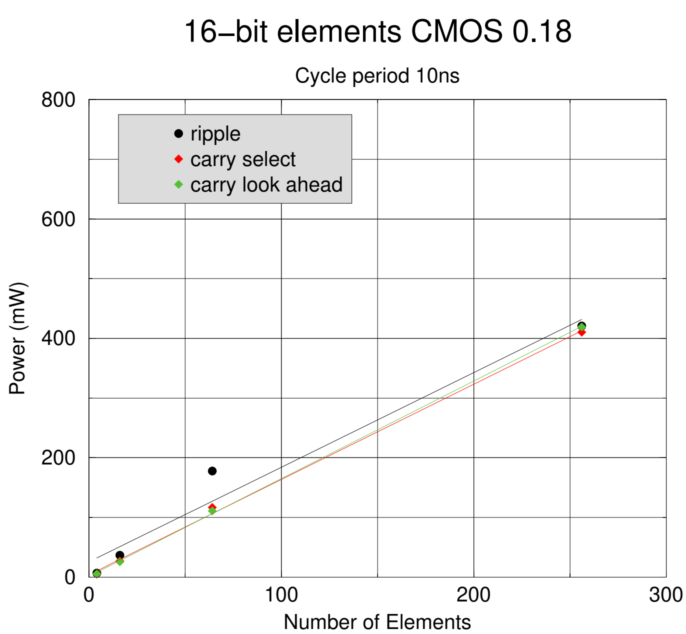


Σχήμα 4.8: Σύγκριση αποτελεσμάτων εμβαδού για το διαδικό δέντρο με 24-bit στοιχεία.

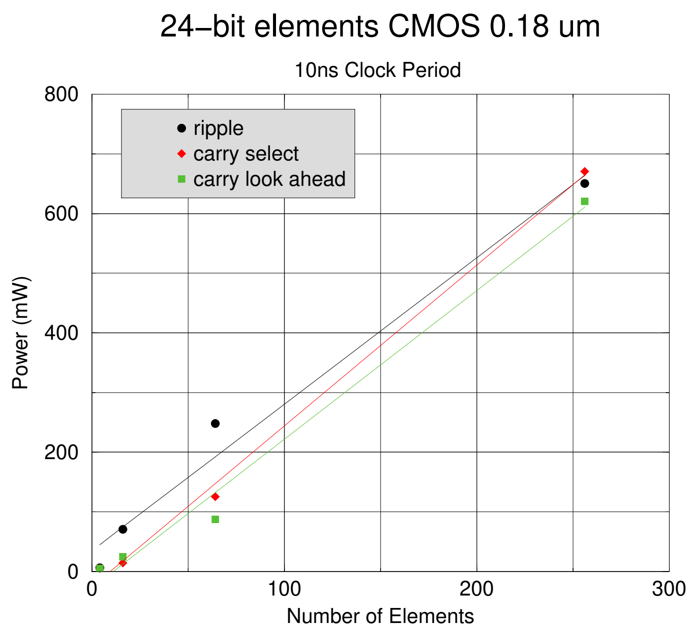
μειώνονται σταδιακά από το στοιχείο μέγιστης τιμής. Για k-bit αριθμούς η μέγιστη τιμή είναι 2^{k-1}, έτσι για N στοιχεία το εύρος των τιμών είναι [2^{k-1},2^{k-1}-N]. Σκοπός μας είναι να ενεργοποιήσουμε όσους περισσότερους κόμβους του δέντρου είναι δυνατό.



Σχήμα 4.9: Σύγκριση αποτελεσμάτων κατανάλωσης ισχύος για 8-bit στοιχεία



Σχήμα 4.10: Σύγκριση αποτελεσμάτων κατανάλωσης ισχύος για 16-bit στοιχεία



Σχήμα 4.11: Σύγκριση αποτελεσμάτων κατανάλωσης ισχύος για 24-bit στοιχεία

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Εργασία

Το κίνητρο για αυτή την εργασία ήταν η σχεδίαση μίας νέας δομής σε υλικό, για την υποστήριξη ταχέων λειτουργιών σε μία ουρά προτεραιότητας, όπου το σύνολο των ροών μεταβάλλεται αυθαίρετα γρήγορα. Η προσέγγιση αυτής της εργασίας, είναι η εύρεση του μικρότερου (ή μεγαλύτερου) στοιχείου σε ένα αυθαίρετο (μη-διατεταγμένο) σύνολο. Για να το πετύχουμε αυτό, προτείνουμε ένα ταχύ αλγόριθμο που βασίζεται στον υπολογισμό μέσω διαδικού δέντρου. Αναπτύξαμε μία νέα οργάνωση για το δέντρο, όπου τα σήματα διαδίδονται ταυτόχρονα κατά μήκος του συγκριτή δύο αριθμών και των επιπέδων του δέντρου. Η πολυπλοκότητα μιας τέτοιας δομής είναι υπο-γραμμική, χαρακτηριστικό που κληρονομείται από τον συγκριτή επιλογής κρατουμένου και το διαδικό δέντρο. Το πλήθος των στοιχείων που παίρνουν μέρος σε κάθε επανάληψη του αλγορίθμου είναι μεταβλητό, χωρίς να επηρεάζει την επίδοση του κυκλώματος.

Τα αποτελέσματα σύνθεσης δείχνουν ότι η καθυστέρηση του διαδικού δένδρου όταν χρησιμοποιούνται συγκριτές επιλογής κρατουμένου είναι μικρότερη των 5ns για 256 αριθμούς, σε τεχνολογία 0.18μm CMOS. Επιπλέον, τα αποτελέσματα σύνθεσης δείχνουν καλές επιδόσεις για την κατανάλωση ισχύος και το εμβαδόν κυκλώματος.

Περαιτέρω μελέτη του της οργάνωσης του δέντρου θα οδηγήσει σε ομοχειρία (pipeline) των επιπέδων του δέντρου. Αυτή μπορεί να εφαρμοστεί με δύο τρόπους:

είτε κύματος wave pipeline, είτε την κλασσική με καταχωτητές να χωρίζουν τα επίπεδα. Με αυτό τον τρόπο το ίδιο δέντρο θα μπορεί να χρησιμοποιηθεί από τουλάχιστον $\log_2 N$ σύνολα. Επίσης πρέπει να σχεδιαστεί σε Full-Custom για δύο λόγους: πρώτον να εκμεταλλευτεί την δυναμική αποθήκευση και δεύτερον να χρησιμοποιήσει την τοπολογία για place and root που προτείνεται στην εργασία [YoSi]

Βιβλιογραφία

[CK02] N. Chryssos, M. Katevenis, *"Weighted Max-Min Fair Scheduling for a Crosspoint-Buffered Crossbar"*, M.Sc. Thesis, Computer Science Departament, University of Crete, April 2002 (under preparation).

[IK01] A. Ioannou, M. Katevenis, *"Pipelined Heap (Priority Queue) Management for Advanced Scheduling in High-Speed Networks"*, Proc. IEEE International Conference on Communications (ICC' 2001), Helsinki, Finland, June 2001.

[GM99] P. Gupta, N. McKeown: *"Designing and Implementing a Fast Crossbar Scheduler"*, IEEE Micro, Jan.-Feb. 1999, pp. 20-28.

[KaSM97] M. Katevenis, D. Serpanos, E. Markatos, *"Multi-Queue Management and Scheduling for Improved QoS in Communication Networks"*, Proceedings of EMMSEC'97 (European Multimedia, Microprocessor Systems, and Electronic Commerce Conference), Florence, Italy, Nov. 1997, pp. 906-913; http://archvlsi.ics.forth.gr/muqpro/classSch.html

[Kat01] M. Katevenis *"CS-534: Packet Switch Architecture"*, lectures, Fall 2001 http://archvlsi.ics.forth.gr/˜kateveni/534/

[Kes97] S.Keshav, *"An Engineering Approach to Computer Networking"*, Addison Wesley, 1997, ISBN 0-201-63442-2.

[KKVK97] G. Kornaros, C. Kozyrakis, P. Vatsolaki, M. Katevenis, *"Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow Control"*, Proc. 17th Conf. on Advanced Research in VLSI (ARVLSI'97), Univ. of Michigan at Ann Arbor, MI USA, Sep. 1997, pp. 127-144; http://archvlsi.ics.forth.gr/atlasI/atlasI_arvlsi97.ps.gz

[MRS00] S. Moon, J. Rexford, K. G. Shin, *"Scalable Hardware Priority Queue Architectures for High-Speed Packet Switches"*, Trans. on Computers, vol. 49, No 11, November 2000

[Rab96] Jan M. Rabaey, *"Digital Integrated MRS Circuits, A Design Perspective"*, Prentice Hall, Inc., 1996.

[Tho83] C. D. Thompson, *"The VLSI complexity of sorting"*, IEEE Trans. Computers, vol. C-32, pp. 1171-1184, 1983.

[VM93] M. Vai and M.M. Moy, *"Real-time maximum value determination on easily testable VLSI architecture"*, IEEE Trans. Circuits System I, vol. 40, pp. 283-285, Apr. 1993.

[YoSi89] H. Y. Young, A. D. Singh, *"On Implementing Large Binary Tree Architectures in VLSI and WSI"*, IEEE Trans. on Computers, vol. 38(4), pp. 526-537, 1989.

[Zha95] H. Zhang, *"Service Disciplines for Guaranteed Performance in Packet Switching Networks"*, Proceedings of the IEEE, vol. 83, no. 10, Oct. 1995, pp. 1374-1396.