



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

**Σχεδίαση Αριθμητικών Κυκλωμάτων Χαμηλής
Κατανάλωσης Με Χρήση Εργαλείων Αυτοματοποίησης
Σχεδίασης**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Σταθούσης

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2005



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή

Σχεδίαση Αριθμητικών Κυκλωμάτων Χαμηλής Κατανάλωσης Με Χρήση Εργαλείων Αυτοματοποίησης Σχεδίασης

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9^η Μαρτίου 2005.

Αθήνα, Μάρτιος 2005

.....
Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

.....
Τσανάκας Παναγιώτης
Καθηγητής Ε.Μ.Π.

.....
Κουκούτσης Ηλίας
Επίκουρος Καθηγητής Ε.Μ.Π.

.....
Μάριος Σταθούσης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Μάριος Σταθούσης, 2005

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Σε όσους με αγαπούν ,
και σε όσους αγαπώ

Πρόλογος

Ο κόσμος της τεχνολογίας εξελίσσεται ραγδαία. Οι εποχές των πρώτων ηλεκτρονικών υπολογιστών με λυχνίες φαίνονται σχεδόν προϊστορικές. Σήμερα υπάρχουν ασύρματα δίκτυα τεράστιων ταχυτήτων, για φορητούς υπολογιστές με υπολογιστική δύναμη μεγαλύτερη από αυτήν που είχαν οι υπολογιστές της NASA το 1969 όταν και έστειλε τον πρώτο άνθρωπο στο φεγγάρι κινητά τρίτης γενιάς που παίζουν βίντεο και κατεβάζουν μουσική από το Internet. Η τεχνολογία φαίνεται πλέον να κινείται στην κατεύθυνση των πολλών επεξεργασιών πάνω σε ένα και μόνο chip.

Όλα αυτά έχουν σαν αποτέλεσμα τα ολοκληρωμένα κυκλώματα, πάνω στα οποία πέφτει να σηκώσουν το βάρος των εξελίξεων, να γίνονται όλο και μικρότερα ενώ οι απαιτήσεις για ταχύτητα να γίνονται ολοένα και μεγαλύτερες. Με αυτά τα δύο δεδομένα ανακύπτει το ζήτημα της κατανάλωσης ισχύος των ολοκληρωμένων κυκλωμάτων, η οποία πρέπει να είναι μικρή για να βοηθάει από την μια μεριά την ψύξη των επεξεργασιών και από την άλλη για να μεγαλώνει την διάρκεια ζωής της μπαταρίας των φορητών συστημάτων.

Σκοπός της εργασίας είναι να αποτελέσει αυτή καθεαυτή εφαλτήριο για την ανάλυση και την σχεδίαση Low Power κυκλωμάτων και ειδικότερα πολλαπλασιαστών οι οποίοι αποτελούν τον ακρογωνιαίο λίθο της υπολογιστικής δυνατότητας ενός επεξεργαστή.

Προκειμένου να πετύχουμε τον στόχο μας, θα παρουσιάσουμε σε αυτήν την εργασία τα εργαλεία τα οποία μπορούν να χρησιμοποιηθούν για την μελέτη, ανάλυση και προσομοίωση πολλαπλασιαστών, αλλά και γενικότερα ψηφιακών κυκλωμάτων, και τα κυκλώματα που χρησιμοποιούνται για την κατασκευή πολλαπλασιαστών.

Τέλος, στην παρούσα εργασία θα κάνουμε μια ανάλυση τεσσάρων πολύ δημοφιλών πολλαπλασιαστών. Οι πολλαπλασιαστές αυτοί είναι ο carry save, ο carry propagate, ο Booth και ο Wallace. Επίσης θα γίνει η ανάλυση για τον αθροιστή διάδοσης κρατούμενου και τον carry skip adder καθώς και δύο ευρέως διαδεδομένων σειριακών πολλαπλασιαστών.

Η ανάλυση έχει τρία επίπεδα: area, time, power τα οποία θα συγκριθούν για ένα πλήθος αριθμών εισόδου. Στόχος μας είναι να καταλήξουμε σε ένα συμπέρασμα για το ποιος είναι ο καλύτερος πολλαπλασιαστής και ποιος είναι ο καλύτερος αθροιστής με γνώμονα και τις τρεις προαναφερθείσες παραμέτρους.

Λέξεις κλειδιά : Low Power, πολλαπλασιαστές, carry save, carry propagate, Booth, Wallace trees, carry skip adder, serial multiplier.

Abstract

The world of technology is constantly evolving. The times of the first computers with the vacuum tubes seem today almost prehistoric. Nowadays we talk about wireless networks of high speeds, about portable computers with computing power larger than the one that NASA computers had in 1969 when they send the first man on the moon. We talk about 3G cellular phones that can download music files from the Internet and play video. Technology is now moving towards multiple processors in a sole chip. The integrated circuits are called upon to rise to the occasion and become smaller and smaller and at the same time become faster. Considering all these, the matter of power dissipation of integrated circuits rises. Power dissipation owes to be minimum so that it helps the cooling of processors and aim at longer battery life of portable systems. The aim of the present diploma thesis is to establish itself as a foundation of analyzing and designing low power multipliers. In order to achieve this goal, this thesis involves the presentation of the tools that helped us in the analysis and the simulation of not only multipliers but generally of all digital circuits, as well as the presentation of the circuits which construct a multiplier. Finally, we will focus our attention on four very widespread multipliers. Those multipliers are: carry save, carry propagate, Booth multiplier and Wallace multiplier. Moreover we will examine two adders, carry propagate and carry skip, as well as two serial multipliers. The analysis has three levels: area, timing and power. Our intention is to reach a conclusion as to which one is the more efficient of all based on all three parameters.

Key - Words: Low Power, multipliers, carry save, carry propagate, Booth, Wallace, Wallace trees, carry skip adder, serial multiplier.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κύριο Κιαμάλ Πεκμεστζή, Καθηγητή Ε.Μ.Π. Θα ήθελα επίσης να ευχαριστήσω τον υποψήφιο διδάκτορα Κώστα Αναγνωστόπουλο για την αμέριστη βοήθεια και συμπαράσταση στην διεκπεραίωση αυτής της εργασίας. Τέλος θέλω να ευχαριστήσω τους φίλους μου και ειδικότερα την οικογένεια μου που με ανέχτηκαν όλο αυτόν τον καιρό.

Πίνακας Περιεχομένων

1. Εισαγωγή	22
1.1 Μια ματιά στο χθες και στο σήμερα	22
1.2 Σκοπός της εργασίας	23
1.3 Οργάνωση της εργασίας.....	25
2. Background	26
2.1 Low Power	26
2.1.1 Κατανάλωση Ισχύος	27
2.1.2 Τεχνικές Ελαχιστοποίησης	34
2.1.2.1 Design Time Τεχνικές	35
2.1.2.2 Run Time Τεχνικές	40
2.1.2.3 Μείωση της Ισχύος στην Ανενεργή (sleep) Φάση	42
2.2 Εισαγωγή στην VHDL	44
2.2.1 Εισαγωγικά	44
2.2.2 Ιστορικά	44
2.2.3 Γενικά	45
2.2.4 Δομικά	45
2.3 EDA TOOLS	47
2.3.1 Εργαλεία Προσομοίωσης	49
2.3.2 Εργαλεία Σύνθεσης	50
2.3.3 Εργαλεία Ισχύος	51
3. Κυκλώματα	56
3.1 FULL ADDER.....	56
3.2 FULL ADDER NAND	57
3.3 ADDER_N	58
3.4 ADDER_N NAND	59
3.5 CARRY SKIP ADDER	60
3.6 ΣΕΙΡΙΑΚΟΣ ΑΘΡΟΙΣΤΗΣ	61
3.7 ΣΕΙΡΙΑΚΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ	62
3.8 ΓΕΝΙΚΕΣ ΑΡΧΕΣ ΠΑΡΑΛΛΗΛΩΝ ΠΟΛΛΑΠΛΑΣΙΑΣΤΩΝ.....	64
3.9 Ο ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ BOOTH	65
3.10 ΠΑΡΑΛΛΗΛΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΔΙΑΔΟΣΗ ΚΡΑΤΟΥΜΕΝΟΥ	69
3.11 ΠΑΡΑΛΛΗΛΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΣΩΣΙΜΟ ΚΡΑΤΟΥΜΕΝΟΥ	71
3.12 ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΔΕΝΔΡΟΥ WALLACE	74
3.13 ΣΕΙΡΙΑΚΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΚΑΤΕΒΑΣΜΑ ΚΡΑΤΟΥΜΕΝΟΥ	78

4. Μετρήσεις	79
4.1 Παράλληλοι Πολλαπλασιαστές	79
4.1.1 Carry Propagate Multiplier	79
4.1.2 Carry Save Multiplier	82
4.1.3 Booth Multiplier	84
4.1.4 Wallace Multiplier	86
4.2 Παράλληλοι Αθροιστές	88
4.2.1 Carry Propagate Adder	88
4.2.2 Carry Skip Adder	90
4.3 Σειριακοί Πολλαπλασιαστές	92
4.3.1 Serial Multiplier	92
4.3.2 Serialmul3xdown	93
5. Σχόλια και Συμπεράσματα	94
5.1 Σύγκριση Παράλληλων Πολλαπλασιαστών	94
5.2 Σύγκριση Παράλληλων Αθροιστών	102
Παράρτημα Α Κώδικας	110
Παράρτημα Β Αναλυτικότερα τα εργαλεία του συστήματος ...	141
Βιβλιογραφία	154

Πίνακας Σχημάτων

Σχήμα 1.1 Πρόβλεψη 1986	22
Σχήμα 2.1 Η Αγορά των Ηλεκτρονικών Υπολογιστών	26
Σχήμα 2.2 Μοντέλο αντιστροφέα CMOS για τον υπολογισμό της στατικής κατανάλωσης ισχύος	27
Σχήμα 2.3 Μοντέλο για τις παρασιτικές διόδους που παρουσιάζονται σε έναν αντιστροφέα CMOS	28
Σχήμα 2.4 Κυματομορφή μεταγωγής εισόδου και μοντέλο για το ρεύμα βραχυκυκλώματος	31
Σχήμα 2.5 Η κατανάλωση ισχύος ενός αντιστροφέα CMOS σε συνάρτηση με τον λόγο μεταξύ των χρόνων ανόδου/καθόδου εισόδου και εξόδου	32
Σχήμα 2.6 Σχέδιο με αποκλίνοντα μήκη κρίσιμων μονοπατιών	36
Σχήμα 2.7 Τεχνική ομαδοποιημένης μείωσης τάσης	37
Σχήμα 2.8 Η πολύπλεξη αυξάνει το switching activity	39
Σχήμα 2.9 Ένας αντιστροφέας με έλεγχο καταφλιού	41
Σχήμα 2.10 Παράδειγμα για αρχιτεκτονική gate clocking	42
Σχήμα 2.11 Χρησιμοποίηση SLEEP τρανζίστορ τόσο στην τροφοδοσία όσο και στην γείωση	43
Σχήμα 2.12 Πολλαπλή περιγραφή κυκλώματος σε VHDL	46
Σχήμα 2.13 Κύκλωμα και μοντέλο	46
Σχήμα 2.14 Διάγραμμα ροής εργασίας	47
Σχήμα 2.15 Διάγραμμα ροής για την προσομοίωση στο ModelSim	49
Σχήμα 2.16 Διαδικασία σύνθεσης στον Design Compiler	50
Σχήμα 2.17 Ροή ενέργειας σε κάθε επίπεδο	51
Σχήμα 2.18 Ροή ενέργειας από το επίπεδο RTL στο επίπεδο πύλης	52
Σχήμα 2.19 Μεθοδολογία υπολογισμού ενέργειας στον Power Compiler	53
Σχήμα 2.20 Μεθοδολογία υπολογισμού στο Prime Power	54
Σχήμα 3.1 Πλήρης αθροιστής σε επίπεδο πυλών	56
Σχήμα 3.2 Πλήρης αθροιστής με πύλες NAND NOR	57
Σχήμα 3.3 Παράλληλος αθροιστής των τεσσάρων (4) bits με διάδοση κρατουμένου	58
Σχήμα 3.4 Υλοποίηση πλήρους αθροιστή με πύλες NAND	59
Σχήμα 3.5 Carry skip block	60
Σχήμα 3.6 Σειριακός αθροιστής δύο σειριακών εισόδων και παράλληλης εξόδου	61
Σχήμα 3.7 Πολλαπλασιαστής μιας σειριακής και μιας παράλληλης εισόδου, και παράλληλης εξόδου	62
Σχήμα 3.8 Κυκλωματικό διάγραμμα πολλαπλασιαστή	63
Σχήμα 3.9 Διάγραμμα χρονισμού πολλαπλασιαστή	63
Σχήμα 3.10 Κύτταρο ελέγχου Booth	67
Σχήμα 3.11 Κύτταρο πολλαπλασιαστή Booth	67
Σχήμα 3.12 Πολλαπλασιαστής Booth 3x3	68
Σχήμα 3.13 Η βασική δομική μονάδα του πολλαπλασιαστή με διάδοση Κρατουμένου.....	69
Σχήμα 3.14 Παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου	70
Σχήμα 3.15 Δομική μονάδα του πολλαπλασιαστή με σώσιμο κρατουμένου	71
Σχήμα 3.16 Παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου	72
Σχήμα 3.17 Τελικός αθροιστής	73
Σχήμα 3.18 Ομαδοποίηση των μερικών γινομένων δυο αριθμών των 5 bits	74
Σχήμα 3.19 Ο πίνακας των μερικών γινομένων μετά την αντικατάσταση και η νέα ομαδοποίηση	75

Σχήμα 3.20 Πολλαπλασιαστής 6x6 σε δομή δένδρου Wallace	76
Σχήμα 3.21 Σειριακός Πολλαπλασιαστής Κατεβάσματος	78
Σχήμα 4.1 Area για carry propagate	79
Σχήμα 4.2 Area για carry propagate β	80
Σχήμα 4.3 Time για carry propagate	80
Σχήμα 4.4 Time για carry propagate β	81
Σχήμα 4.5 Area για carry save	82
Σχήμα 4.6 Area για carry save β	82
Σχήμα 4.7 Time για carry save	83
Σχήμα 4.8 Time για carry save β	83
Σχήμα 4.9 Area για booth	84
Σχήμα 4.10 Area για booth β	84
Σχήμα 4.11 Time για Booth	85
Σχήμα 4.12 Time για Booth β	85
Σχήμα 4.13 Area για wallace	86
Σχήμα 4.14 Area για wallace β	86
Σχήμα 4.15 Time για wallace	87
Σχήμα 4.16 Time για wallace	87
Σχήμα 4.17 Area για carry propagate adder	88
Σχήμα 4.18 Time για carry propagate adder	88
Σχήμα 4.19 Power για carry propagate adder	89
Σχήμα 4.20 Area για carry skip adder	90
Σχήμα 4.21 Time για carry skip adder	91
Σχήμα 4.22 Power για carry skip adder	91
Σχήμα 5.1 Area για 8 bits	94
Σχήμα 5.2 Area για 16 bits	95
Σχήμα 5.3 Area για 32 bits	95
Σχήμα 5.4 Area για 64 bits	96
Σχήμα 5.5 Time για 8 bits	96
Σχήμα 5.6 Time για 16 bits	97
Σχήμα 5.7 Time για 32 bits	97
Σχήμα 5.8 Time για 64 bits	98
Σχήμα 5.9 Power για 8 bits	98
Σχήμα 5.10 Power για 16 bits	98
Σχήμα 5.11 Power για 32 bits	100
Σχήμα 5.12 Power για 64 bits	101
Σχήμα 5.13 Area για 8 bits	102
Σχήμα 5.14 Time για 8 bits	104
Σχήμα 5.15 Power για 8 bits	103
Σχήμα 5.16 Area για 16 bits	104
Σχήμα 5.17 Time για 16 bits	104
Σχήμα 5.18 Power για 16 bits	105
Σχήμα 5.19 Area για 32 bits	106
Σχήμα 5.20 Time για 32 bits	106
Σχήμα 5.21 Power για 32 bits	107
Σχήμα 5.22 Area για 64 bits	108
Σχήμα 5.23 Time για 64 bits	108
Σχήμα 5.24 Power για 64 bits	109
Σχήμα B.1 Χαρακτηριστικό γραφικό περιβάλλον του ModelSim	147
Σχήμα B.2 Απεικόνιση του Wave Window	147

Σχήμα B.3 Διάγραμμα ροής για προσομοίωση	148
Σχήμα B.4 Περιβάλλον προσομοίωσης	150
Σχήμα B.5 Διάγραμμα ροής της ενεργειακής προσομοίωσης του Prime Power	158

Πίνακας Πινάκων

Πίνακας [2.1] Τεχνικές ελαχιστοποίησης ισχύος	34
Πίνακας [3.1] Πίνακας αληθείας του full adder.....	56
Πίνακας [3.2] Μερικά γινόμενα πολλαπλασιαστή.....	63
Πίνακας [3.3] Μερικά γινόμενα πολλαπλασιαστή 4x4.....	64
Πίνακας [3.4] Κωδικοποίηση Ψηφίων.....	65
Πίνακας [3.5] Λειτουργία P και Q.....	66
Πίνακας [3.6] Κωδικοποίησης του τροποποιημένου αλγορίθμου Booth.....	77
Πίνακας [3.7] Αντιστοίχιση λειτουργιών και κωδικοποιημένων ψηφίων.....	77
Πίνακας [4.1] Area για carry propagate.....	79
Πίνακας [4.2] Time για carry propagate.....	80
Πίνακας [4.3] Power για carry propagate.....	81
Πίνακας[4.4] Area για carry save.....	82
Πίνακας [4.5] Time για carry save.....	83
Πίνακας [4.6] Power για carry save.....	83
Πίνακας [4.7] Area για booth.....	84
Πίνακας [4.8] Time για Booth.....	85
Πίνακας [4.9] Power για booth.....	85
Πίνακας [4.10] Area για wallace.....	86
Πίνακας [4.11] Time για wallace.....	87
Πίνακας [4.12] Power για wallace.....	87
Πίνακας [4.13] Area για carry propagate adder.....	88
Πίνακας [4.14] Time για carry propagate adder.....	88
Πίνακας [4.15] Power για carry propagate adder.....	89
Πίνακας [4.16] Area για carry skip adder.....	90
Πίνακας [4.17] Time για carry skip adder.....	90
Πίνακας [4.18] Power για carry skip adder.....	91
Πίνακας [4.19] Area για serial multiplier.....	92
Πίνακας [4.20] Power για serial multiplier.....	92
Πίνακας [4.21] Area για serialmul3xdwn.....	93
Πίνακας [4.22] Time για serialmul3xdwn.....	93
Πίνακας [4.23] Power για serialmul3xdwn.....	93
Πίνακας [5.1] Area για 8 bits.....	94
Πίνακας [5.2] Area για 16 bits.....	95
Πίνακας [5.3] Area για 32 bits.....	95
Πίνακας [5.4] Area για 64 bits.....	96
Πίνακας [5.5] Time για 8 bits.....	96
Πίνακας [5.6] Time για 16 bits.....	97
Πίνακας [5.7] Time για 32 bits.....	97
Πίνακας [5.8] Time για 64 bits.....	98
Πίνακας [5.9] Power για 8 bits.....	98
Πίνακας [5.10] Power για 16 bits.....	99
Πίνακας [5.11] Power για 32 bits.....	100
Πίνακας [5.12] Power για 64 bits.....	100
Πίνακας [5.13] Area για 8 bits.....	102
Πίνακας [5.14] Time για 8 bits.....	102
Πίνακας [5.15] Power για 8 bits.....	103
Πίνακας [5.16] Area για 16 bits.....	104
Πίνακας [5.17] Time για 16 bits.....	104

Πίνακας [5.18] Power για 16 bits.....	105
Πίνακας [5.19] Area για 32 bits.....	106
Πίνακας [5.20] Time για 32 bits.....	106
Πίνακας [5.21] Power για 32 bits.....	107
Πίνακας [5.22] Area για 64 bits.....	108
Πίνακας [5.23] Time για 64 bits.....	108
Πίνακας [5.24] Power για 64 bits.....	109

List of acronyms

1. CMOS - Complementary Metal Oxide Semiconductor
2. VHSIC - Very High Speed Integrated Circuits
3. VHDL - VHSIC Hardware Description Language
4. ASIC - Application- Specific Integrated Circuit
5. VLSI - Very Large Scale of Integration
6. VCD - Value Change Dump
7. TCL - Tool Command Language
8. FPGA - Field Programmable Gate Arrays
9. EDA - Electronic Design Automation
10. SDF - Standard Delay Format
11. DVS - Dynamic Voltage Scaling
- 12.DTS - Dynamic Threshold Scaling
- 13.LSPP - Least Significant Partial Product
- 14.LSB - Least Significant Bit
- 15.MSB - Most Significant Bit

1.2 Σκοπός της εργασίας.

Σκοπός της εργασίας είναι να αποτελέσει αυτή καθεαυτή εφαλτήριο για την ανάλυση και την σχεδίαση low power κυκλωμάτων και ειδικότερα πολλαπλασιαστών οι οποίοι αποτελούν τον ακρογωνιαίο λίθο της υπολογιστικής δυνατότητας ενός επεξεργαστή.

Προκειμένου να επιτύχουμε αυτόν τον στόχο θα παρουσιάσουμε αρχικά κάποια απαραίτητα στοιχεία για την κατανάλωση ισχύος.

Όπως γνωρίζουμε, η κατανάλωση ισχύος χωρίζεται σε τρία μέρη. Την στατική κατανάλωση που οφείλεται στα ρεύματα διαρροής, την κατανάλωση ισχύος που οφείλεται στα ρεύματα μεταγωγής και στα ρεύματα φόρτισης και εκφόρτισης χωρικών φορτίων και τέλος την κατανάλωση βραχυκυκλώματος που οφείλεται στο ρεύμα που ρέει όταν και τα δύο τρανζίστορ άγουν.

Θα δούμε τεχνικές ελαχιστοποίησης της καταναλισκόμενης ισχύος, οι οποίες ως επί το πλείστον ανταλλάζουν ταχύτητα και επίδοση με ισχύ. Οι τεχνικές ελαχιστοποίησης της καταναλισκόμενης ισχύος χωρίζονται σε κατηγορίες ανάλογα με το ποιο από τα είδη της καταναλισκόμενης ισχύος ελαχιστοποιεί και με το πότε γίνεται αυτή η αντιμετώπιση, την ώρα της λειτουργίας ή την ώρα της σχεδίασης. Οι τεχνικές αυτές περιέχουν εν μέσω άλλων διαφορετικές τάσεις τροφοδοσίας, διαφορετικά κατώφλια και clock gating.

Στην συνέχεια θα δούμε κάποια εισαγωγικά στοιχεία για την γλώσσα περιγραφής υλικού VHDL. Η VHDL αριθμεί πολλά πλεονεκτήματα μεταξύ αυτών και την δυνατότητα περιγραφής από το πιο απλό μέχρι και το πλέον πολύπλοκο κύκλωμα με ένα περιορισμένο σύνολο εντολών. Επίσης, χάρις στην ανεξαρτησία της από την τεχνολογία κατασκευής μπορεί να χρησιμοποιηθεί για πάρα πολλές εφαρμογές από διαφορετικούς κατασκευαστές. Στην VHDL θα παρουσιάσουμε τέλος τους τρεις κυρίαρχους τρόπους με τους οποίους μπορεί ένα κύκλωμα να περιγραφεί και αυτοί είναι ο structural, ο dataflow και ο behavioral.

Θα ακολουθήσουν τα εργαλεία τα οποία μας βοήθησαν στο να συνθέσουμε να αναλύσουμε και να προσομοιώσουμε όλα τα κυκλώματα. Αρχή θα κάνουμε με τον Design Compiler. Ο Design Compiler είναι ένα πολύ χρήσιμο εργαλείο για την ανάλυση, την προσομοίωση και την βελτιστοποίηση ψηφιακών κυκλωμάτων. Στην συγκεκριμένη εργασία εκμεταλλευτήκαμε τις δυνατότητες του προγράμματος του Synopsys για τα source αρχεία τα οποία μας επιτρέπουν την προσομοίωση των κυκλωμάτων και την εξαγωγή αναφορών για τον χώρο που πιάνει το κάθε ένα πάνω στο chip και τον χρόνο τον οποίο χρειάζεται για να υπολογισθεί το αποτέλεσμα από την στιγμή που θα εισέλθουν τα δεδομένα.

Επίσης μας επιτρέπει να σχηματίσουμε synthesizable αρχεία τα οποία θα προσομοιωθούν με το ModelSim για να περάσουν το switching activity στα VCD αρχεία.

Το επόμενο εργαλείο είναι το Prime Power επίσης της Synopsys. Το Prime Power θα πάρει τα αρχεία VCD και τρέχοντας τα θα μας δώσει λεπτομερείς

ενεργειακές αναφορές για τα κυκλώματα χωρίζοντας την καταναλισκόμενη ισχύ σε δυναμική και διαρροής με αναλυτικά στοιχεία για την καθεμία.

Το τρίτο εργαλείο που μας βοήθησε είναι το ModelSim. Το ModelSim είναι και αυτό ένα πολύ χρήσιμο πρόγραμμα όσον αφορά την σχεδίαση, ανάλυση και αποσφαλμάτωση ψηφιακών κυκλωμάτων που περιγράφονται σε VHDL, μέσα από ένα φιλικό προς τον χρήστη γραφικό περιβάλλον. Το ModelSim μας έδωσε την δυνατότητα μέσα από το περιβάλλον προσομοίωσης του να τρέξουμε τα synthesizable αρχεία που φτιάξαμε με τον Design Compiler και μέσω των κατάλληλων εντολών να περάσουμε το switching activity σε κάποια VCD αρχεία, τα οποία θα χρειαστεί το Prime Power για να μας δώσει τις αναφορές για την κατανάλωση ισχύος.

Στην συνέχεια θα παρουσιάσουμε τα κυκλώματα τα οποία χρησιμοποιήθηκαν για την κατασκευή των πολλαπλασιαστών, εστιάζοντας σε τέσσερις πολύ διαδεδομένους πολλαπλασιαστές: τον carry save, τον carry propagate, τον Booth και τον Wallace, στον αθροιστή διάδοσης κρατουμένου και στον carry skip adder καθώς και σε δυο σειριακούς πολλαπλασιαστές.

Ο carry save και ο carry propagate είναι δύο πολλαπλασιαστές της οικογένειας των παράλληλων με πολύ κοινά γνωρίσματα. Στην μία περίπτωση, αυτή του carry propagate το κρατούμενο διαδίδεται από τον ένα αθροιστή στον διπλανό του ίδιου επιπέδου, ενώ στον carry save το κρατούμενο περνάει στην κάτω βαθμίδα.

Ο Booth πολλαπλασιαστής χρησιμοποιεί τον ομώνυμο αλγόριθμο μετατροπής αριθμών από την κανονική μορφή στην μορφή Booth και στην συνέχεια περνάει στο κομμάτι του πολλαπλασιασμού των δύο αριθμών. Η χρησιμότητα του έγκειται στο γεγονός ότι ο αλγόριθμος είναι γενικός και ισχύει είτε οι αριθμοί είναι θετικοί είτε είναι αρνητικοί.

Ο πολλαπλασιαστής Wallace χρησιμοποιεί αρχικά τον τροποποιημένο αλγόριθμο του Booth για να μετατρέψει τους δύο αριθμούς και στην συνέχεια με την ειδική δομή των δένδρων Wallace ομαδοποιεί τα μερικά γινόμενα και δίνοντας έτσι μεγαλύτερη οικονομία υλικού και μεγαλύτερη σαφώς ταχύτητα στους υπολογισμούς μας.

Ο αθροιστής διάδοσης κρατουμένου είναι ο κλασικός παράλληλος αθροιστής ενώ ο carry skip adder είναι ένας παράλληλος αθροιστής που χρησιμοποιεί ένα ειδικό κύκλωμα για να παραλείπει τον υπολογισμό του κρατουμένου όταν αυτό δεν χρειάζεται και έτσι να φτάνει σε αποτέλεσμα πιο γρήγορα.

Τέλος οι δύο σειριακοί πολλαπλασιαστές μοιράζονται την ιδιότητα να έχουνε μια παράλληλη είσοδο και μια σειριακή από την οποία μπαίνουν τα δεδομένα και κάθε στιγμή υπολογίζεται το εκάστοτε μερικό γινόμενο.

Για καθένα από αυτούς τους πολλαπλασιαστές κάναμε μετρήσεις που αφορούν area, timing και power με απώτερο σκοπό να συγκρίνουμε τις τέσσερις αυτές δομές και να καταλήξουμε στο ποιος είναι ο αποτελεσματικότερος. Ας μην ξεχνάμε ότι η καταναλισκόμενη ισχύς είναι πολύ σημαντικό ζήτημα για ένα ολοκληρωμένο κύκλωμα αλλά δεν είναι το μόνο. Πολύ σημαντικά είναι επίσης ο χώρος που πιάνει το κύκλωμα πάνω στο chip αλλά και η ταχύτητα με την οποία υπολογίζεται το αποτέλεσμα. Έτσι με την μελέτη των τριών αυτών συνιστωσών θα οδηγηθούμε στο συμπέρασμα του ποιος είναι ο αποτελεσματικότερος.

Το ίδιο ισχύει και για τους δύο παράλληλους αθροιστές.

1.3 Οργάνωση εργασίας

Η παρούσα εργασία δομείται ως εξής:

Στο **κεφάλαιο 2** γίνεται μια παρουσίαση του θεωρητικού υποβάθρου το οποίο είναι απαραίτητο. Εξετάζουμε τους παράγοντες κατανάλωσης της ισχύς σε ένα κύκλωμα καθώς και κάποιες τεχνικές αντιμετώπισης του προβλήματος. Γίνεται λόγος για τα εργαλεία που χρησιμοποιήθηκαν προκειμένου να πάρουμε τις μετρήσεις και τέλος κάποια εισαγωγικά πράγματα πάνω στην γλώσσα περιγραφής υλικού VHDL.

Στο **κεφάλαιο 3** γίνεται μνεία στα κυκλώματα τα οποία χρησιμοποιούνται για την κατασκευή των πολλαπλασιαστών αλλά και οι ίδιοι πολλαπλασιαστές.

Στο **κεφάλαιο 4** παρουσιάζονται οι μετρήσεις που γίνανε για κάθε ένα παράλληλο αλλά και σειριακό πολλαπλασιαστή και για τους δύο αθροιστές για το μέγεθος , τον χρόνο και την ισχύ.

Στο **κεφάλαιο 5** γίνονται σχόλια για τα αποτελέσματα των μετρήσεων. Επιδεικνύονται συγκριτικοί πίνακες και διαγράμματα για διάφορα μήκη λέξεων εισόδου.

Στο **παράρτημα Α** κατατίθεται ο κώδικας για όλα τα κυκλώματα που παρουσιάζονται στο κεφάλαιο 3 σε γλώσσα περιγραφής υλικού VHDL.

Στο **παράρτημα Β** γίνεται μια περισσότερο αναλυτική παρουσίαση των προγραμμάτων που χρησιμοποιήθηκαν με έμφαση στις γενικότερες δυνατότητες τους αλλά και στις συγκεκριμένες ενέργειες στις οποίες εμείς προβήκαμε.

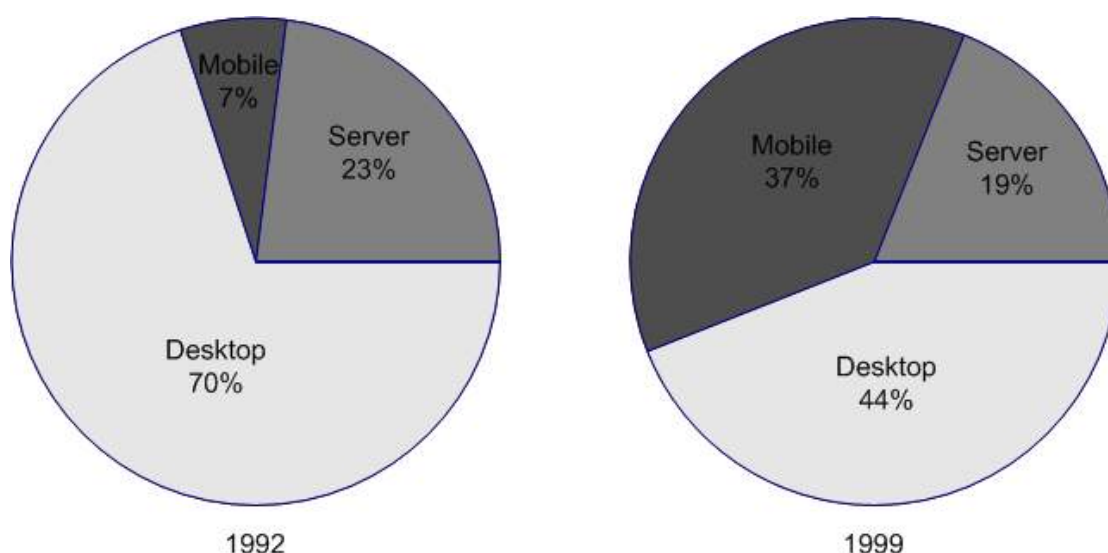
Κεφάλαιο 2

Background

2.1 Low Power

Η συνεχής ανάπτυξη της αγοράς των κινητών ηλεκτρονικών συστημάτων και των συστημάτων που τροφοδοτούνται με μπαταρία (π.χ κινητά τηλέφωνα, laptop) απαιτεί την σχεδίαση συστημάτων χαμηλής ενέργειας (low power design). Όσο συνεχίζεται η αύξηση της πυκνότητας ,της πολυπλοκότητας και του μεγέθους των chip τόσο αυξάνεται η δυσκολία επαρκούς ψύξης αυτών. Η παροχή αυτής επιφέρει είτε αύξηση κόστους είτε περιορισμό της λειτουργικότητας των συστημάτων, γεγονός που οδήγησε στην ανάπτυξη τεχνικών και εργαλείων για την σχεδίαση κυκλωμάτων χαμηλής ενέργειας.

Ενδεικτικά είναι τα δύο διαγράμματα που ακολουθούν και δείχνουν το πόσο μεγάλωσε το ποσοστό των φορητών υπολογιστών από το 1992 μέχρι το 1999



Σχήμα 2.1 Η Αγορά των Ηλεκτρονικών Υπολογιστών

Η ελάττωση της ενέργειας στοχεύει στην ελαχιστοποίηση της μέγιστης στιγμιαίας ενέργειας ή της μέσης ενέργειας. Στο σημείο αυτό καλό είναι να διευκρινίσουμε ότι στις περισσότερες πραγματικές εφαρμογές η ελαχιστοποίηση της ενέργειας δεν είναι ποτέ το μοναδικό ζητούμενο. Η επίδοση του συστήματος είναι πάντα ένας κρίσιμος παράγοντας. Δυστυχώς, η μείωση της ενέργειας δεν έρχεται χωρίς τίμημα. Χρειάζεται να θυσιάσουμε ένα μέρος της επίδοσης προκειμένου να επιτύχουμε την ποθητή ελαχιστοποίηση ενέργειας. Από εκεί και πέρα είναι στην διακριτική ευχέρεια του κάθε σχεδιαστή η επιλογή του όσον αφορά στον συμβιβασμό μεταξύ επίδοσης και ενέργειας. Μια επίσης πολύ μεγάλη ανταλλαγή που καλούμαστε να κάνουμε είναι αυτός ανάμεσα σε ενέργεια και ευελιξία. Πολλοί σχεδιαστές έχουν παρατηρήσει ότι συστήματα σχεδιασμένα για μια συγκεκριμένη λειτουργία είναι πολύ πιο αποδοτικά σε ενέργεια από συστήματα πολλαπλών λειτουργιών όταν τα τελευταία καλούνται να κάνουν το ίδιο πράγμα με τα πρώτα.

2.1.1 Κατανάλωση ισχύος

Η ισχύς που καταναλώνεται σε ένα κύκλωμα CMOS αποτελείται από τρεις συνιστώσες:

- Στατική κατανάλωση ισχύος λόγω ρεύματος διαρροής ή άλλων ρευμάτων που ρέουν συνεχώς από την τροφοδοσία ισχύος
- Δυναμική κατανάλωση λόγω του ρεύματος μεταγωγής και της φόρτισης και εκφόρτισης των χωρητικών φορτίων.
- Κατανάλωση βραχυκυκλώματος λόγω του ρεύματος που ρέει όταν και τα δύο τρανζίστορ άγουν ταυτόχρονα

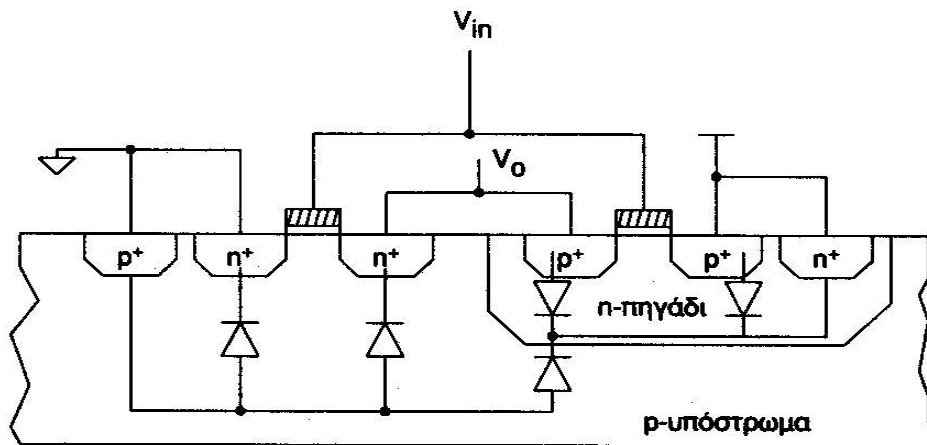
Στατική κατανάλωση

Έστω συμπληρωματική πύλη CMOS (σχήμα 2.2) στην οποία αν η είσοδος είναι '0' το αντίστοιχο n-στοιχείο είναι 'OFF' και το p-στοιχείο είναι 'ON'. Η τάση εξόδου είναι V_{DD} ή λογικό '1'. Παρόμοια, αν η είσοδος είναι '1' το αντίστοιχο n-στοιχείο είναι 'ON' και το p-στοιχείο είναι 'OFF', και συνεπώς η έξοδος είναι σε λογικό '0' (ή V_{SS}). Από την λειτουργία της πύλης CMOS προκύπτει ότι πάντοτε ένα τρανζίστορ είναι 'OFF'. Επίσης, επειδή δεν ρέει καθόλου ρεύμα στην πύλη δεν υπάρχει καθόλου μονοπάτι ρεύματος DC από V_{DD} σε V_{SS} και άρα το ρεύμα ηρεμίας (μόνιμη κατάσταση) είναι μηδέν και προφανώς η στατική ισχύς P_S είναι μηδενική.



Σχήμα 2.2 Μοντέλο αντιστροφέα CMOS για τον υπολογισμό της στατικής κατανάλωσης ισχύος

Στην πραγματικότητα υπάρχει μικρή κατανάλωση λόγω των διαρροών αντίστροφης πόλωσης μεταξύ των περιοχών διάχυσης και υποστρώματος. Επίσης υπάρχει στατική κατανάλωση λόγω της περιοχής υποκατωφλίου. Για την καλύτερη κατανόηση του φαινομένου διαρροής σε ένα στοιχείο θα μελετηθεί ένα απλό μοντέλο του αντιστροφέα CMOS το οποίο περιλαμβάνει τις παρασιτικές διόδους. Οι διόδοι αυτοί δημιουργούνται από τις διαχύσεις πηγής – υποδοχής και τις διαχύσεις που βρίσκονται στο n-πηγάδι. Το μοντέλο αυτό φαίνεται στο παρακάτω σχήμα.



Σχήμα 2.3 Μοντέλο για τις παρασιτικές διόδους που παρουσιάζονται σε έναν αντιστροφέα CMOS

Εύκολα μπορούμε να δούμε ότι υπάρχει μια ακόμα παρασιτική διάδος μεταξύ του n-πηγαδιού και υποστρώματος. Επειδή οι παρασιτικές διόδους είναι αντίστροφα πολωμένες, μόνο το ρεύμα διαρροής συνεισφέρει στην στατική κατανάλωση ισχύος. Το ρεύμα διαρροής περιγράφεται από την εξίσωση της διόδου:

$$i_o = i_s (e^{\frac{qV}{kT}} - 1)$$

όπου i_s = ανάστροφο ρεύμα κόρου

V = τάση διόδου

q = φορτίο ηλεκτρονίου

k = σταθερά Boltzman

T = θερμοκρασία

Η στατική κατανάλωση ισχύος είναι το γινόμενο του ρεύματος διαρροής του στοιχείου και της τάσης τροφοδοσίας. Για κάθε στοιχείο το ρεύμα θα πρέπει να κυμαίνεται μεταξύ 0.1 nA και 0.5nA για θερμοκρασία δωματίου.

Η συνολική στατική κατανάλωση ισχύος είναι:

$$P_s = \sum_1^N \text{ρεύμα διαρροής} * \text{τάση τροφοδοσίας} \quad \text{με } N = \text{αριθμός στοιχείων}$$

Για παράδειγμα, η τυπική στατική κατανάλωση ισχύος λόγω διαρροής ενός αντιστροφέα που λειτουργεί στα 5V είναι μεταξύ 1 και 2 nanowatts.

Στις πύλες ψεύδο-nMOS, λόγω του απευθείας μονοπατιού μεταξύ τροφοδοσίας και γείωσης υπάρχει στατική κατανάλωση ισχύος. Αν χρησιμοποιηθούν τέτοιες πύλες η στατική τους κατανάλωση θα πρέπει να ληφθεί υπόψη για τον υπολογισμό της συνολικής κατανάλωσης.

Δυναμική κατανάλωση

Τα δύο τρανζίστορ του αντιστροφέα CMOS άγουν για πολύ μικρό χρονικό διάστημα, το οποίο και σημαίνει ότι υπάρχει ένας μικρός παλμός ρεύματος από V_{DD} μέχρι V_{SS} .

Επιπλέον απαιτείται ρεύμα για την φόρτιση και εκφόρτιση του χωρικού φορτίου εξόδου. Ο παράγοντας αυτός είναι και ο πιο σπουδαίος όρος στην δυναμική κατανάλωση ισχύος.

Ο παλμός ρεύματος από την τάση V_{DD} μέχρι την V_{SS} καταλήγει σε μια κατανάλωση 'βραχυκυκλώματος' η οποία εξαρτάται από τον χρόνο ανόδου/καθόδου της εισόδου, το χωρητικό φορτίο και την σχεδίαση της πύλης.

Αν δεν υπάρχει φορτίο η ύπαρξη του ρεύματος βραχυκυκλώματος είναι προφανής. Αυξανόμενου του χωρητικού φορτίου, το ρεύμα φόρτισης ή εκφόρτισης αρχίζει να καθορίζει σημαντικά το ρεύμα που ρέει από τις τροφοδοσίες ισχύος. Με τις κατάλληλες εξομοιώσεις αποδεικνύεται ότι οι αργές ακμές ανόδου/καθόδου αυξάνουν το ρεύμα βραχυκυκλώματος.

Θεωρώντας ότι ο χρόνος ανόδου και καθόδου μιας βηματικής εισόδου είναι πολύ μικρότερος από την περίοδο επανάληψης, η δυναμική κατανάλωση ισχύος μπορεί να μοντελοποιηθεί. Η μέση δυναμική ισχύς P_d που καταναλώνεται κατά την διάρκεια της μεταγωγής για είσοδο τετραγωνικού παλμού, V_{in} και συχνότητα $f_p = 1/t_p$ δίνεται η σχέση:

$$P_d = \frac{1}{t_p} \int_0^{t_p/2} i_n(t) V_{out} dt + \frac{1}{t_p} \int_{t_p/2}^{t_p} i_p(t) (V_{DD} - V_{out}) dt$$

όπου i_n = μεταβατικό ρεύμα n-στοιχείου

i_p = μεταβατικό ρεύμα π-στοιχείου.

Για βηματική είσοδο και $i_n = C_L dV_{out}/dt$ (C_L = χωρητικότητα φορτίου)

$$P_d = \frac{C_L V_{DD}^2}{t_p}$$

και με $f_p = 1/t_p$

$$P_d = C_L V_{DD}^2 f_p$$

Συμπερασματικά, για μια επαναληπτική βηματική είσοδο, η μέση ισχύς που καταναλώνεται είναι ανάλογη της ενέργειας που απαιτείται για την φόρτιση και εκφόρτιση της χωρητικότητας του κυκλώματος. Επίσης, από την τελευταία σχέση προκύπτει ότι η ισχύς είναι ανάλογη της συχνότητας μεταγωγής και ανεξάρτητη των παραμέτρων ενός στοιχείου.

Η παραπάνω σχέση μπορεί να γραφεί και $P_d = C_L V_{DD}^2 f_{0 \rightarrow 1}$ όπου το $f_{0 \rightarrow 1}$ αντιπροσωπεύει την συχνότητα των μεταγωγών που καταναλώνουν ενέργεια και ονομάζεται *switching activity*. Καθώς η τεχνολογία εξελίσσεται, πηγαίνουμε σε ακόμη μεγαλύτερες τιμές για το $f_{0 \rightarrow 1}$ (καθώς το t_p μειώνεται).

Συγχρόνως, η συνολική χωρητικότητα στο chip (C_L) αυξάνεται καθώς όλο και περισσότερες πύλες τοποθετούνται πάνω του.

Άξιο αναφοράς είναι το επόμενο παράδειγμα. Ένα CMOS chip 0.25-μm με ένα ρολόι στα 500 MHz και με μια χωρητικότητα φορτίου κατά μέσο όρο στα 15fF/πύλη, για τάση τροφοδοσίας 2.5V, έχει κατανάλωση ισχύος ανά πύλη 50μW. Για ένα σχέδιο με 1 εκατομμύριο πύλες και υποθέτοντας ότι μια μεταγωγή γίνεται ανά ακμή του ρολογιού, έχουμε κατανάλωση ισχύος ίση με 50W!!! Αυτή η εκτίμηση είναι μάλλον απαισιόδοξη καθώς στην πραγματικότητα ο αριθμός των πυλών που αλλάζουν με 500 MHz είναι πολύ μικρότερος από τον συνολικό.

Είναι φανερό λοιπόν ότι με την αύξηση της πολυπλοκότητας των ολοκληρωμένων κυκλωμάτων, το ενεργειακό πρόβλημα θα γίνεται όλο και μεγαλύτερο κάνοντας ελκυστικότερες τις χαμηλές τάσεις τροφοδοσίας. Η μείωση της V_{DD} έχει τετραγωνική επίδραση στην P_d . Από την άλλη πλευρά όταν το κάτω φράγμα της τάσης τροφοδοσίας ορίζεται από εξωτερικούς παράγοντες (όπως συμβαίνει σχεδόν πάντα στην πραγματικότητα) ή όταν η απώλεια επίδοσης, εξαιτίας της μείωσης της τάσης τροφοδοσίας, είναι πλέον πολύ μεγάλη οι μόνοι παράγοντες που μπορούν να μας βοηθήσουν στην μείωση της κατανάλωσης είναι η φυσική χωρητικότητα και το switching activity.

Η μείωση του switching activity μπορεί να επιτευχθεί σε επίπεδο αρχιτεκτονικής και λογικής. Η μείωση της φυσικής χωρητικότητας γίνεται μέσω της μείωσης του μεγέθους των τρανζίστορ, αφού το μεγαλύτερο μέρος της χωρητικότητας ενός κυκλώματος οφείλεται στις χωρητικότητες των τρανζίστορ (πύλης και διάχυσης). Και πάλι όμως επηρεάζεται η επίδοση του κυκλώματος, με δυνατότητα όμως αντιστάθμισης μέσω διάφορων τεχνικών επιτάχυνσης. Η μόνη περίπτωση κατά την οποία πρέπει να αυξήσουμε το μέγεθος του τρανζίστορ, είναι όταν η χωρητικότητα φορτίου κυριαρχείται από εξωτερικούς παράγοντες όπως η χωρητικότητα των καλωδίων.

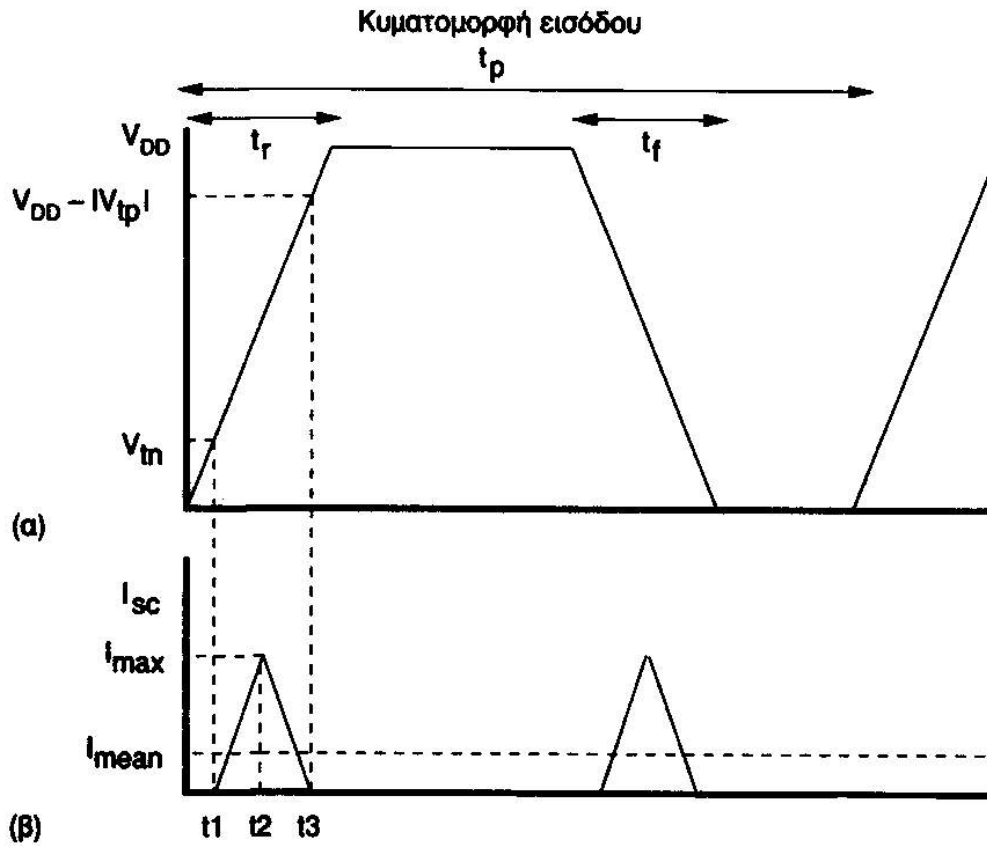
Κατανάλωση βραχυκυκλώματος

Η κατανάλωση ισχύος βραχυκυκλώματος δίνεται από την σχέση

$$P_{sc} = I_{mean} \cdot V_{DD}$$

Για την κυματομορφή που φαίνεται στο σχήμα 2.4 το ρεύμα βραχυκυκλώματος σε έναν αντιστροφέα χωρίς φορτίο είναι

$$I_{mean} = 2 \times \left[\frac{1}{T} \int_{t1}^{t2} I(t) dt + \frac{1}{T} \int_{t2}^{t3} I(t) dt \right]$$



Σχήμα 2.4 Κυματομορφή μεταγωγής εισόδου και μοντέλο για το ρεύμα βραχυκυκλώματος

Θεωρώντας ότι $V_{tn} = -V_{tp}$, $\beta_n = \beta_p = \beta$ και ότι η συμπεριφορά είναι συμμετρική γύρω από το σημείο t_2 το ρεύμα δίνεται από

$$I_{mean} = 2 \times \frac{2}{T} \int_{t_1}^{t_2} \frac{\beta}{2} (V_{in}(t) - V_t)^2 dt$$

$$\text{με } V_{in}(t) = \frac{V_{DD}}{t_r} t, \quad t_1 = \frac{V_t}{V_{DD}} t_r, \quad t_2 = \frac{t_r}{2}$$

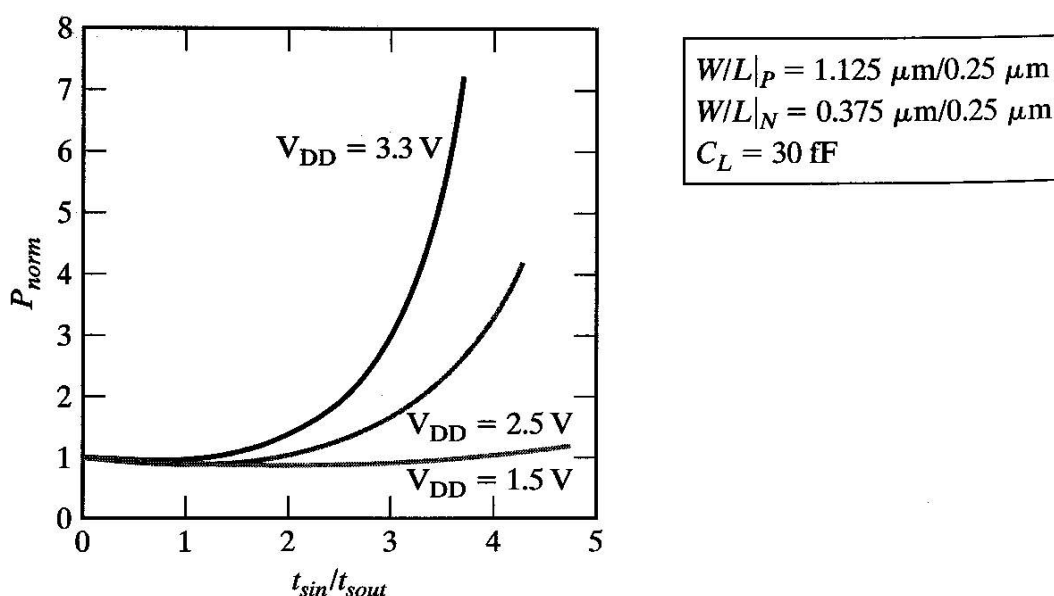
Τελικά, λαμβάνοντας υπόψη ότι $t_r = t_f (= t_{rf})$, η ισχύς βραχυκυκλώματος ενός αντιστροφέα χωρίς φορτίο είναι

$$P_{sc} = \frac{\beta}{12} (V_{DD} - 2V_t)^{\frac{3t_{rf}}{t_p}}$$

όπου t_p είναι η περίοδος της κυματομορφής. Η παραπάνω σχέση δείχνει ότι το ρεύμα βραχυκυκλώματος εξαρτάται από το β και του χρόνους ανόδου και καθόδου της κυματομορφής εισόδου. Αργοί χρόνοι ανόδου σε κόμβους μπορεί να οδηγήσουν σε σημαντική (20%) κατανάλωση βραχυκυκλώματος για αντιστροφείς με φορτία. Όποτε

μια πρακτική λύση για μείωση της κατανάλωσης βραχυκυκλώσεως είναι να διατηρούνται όσο το δυνατό πιο γρήγορες. Τελικά, όσο πιο πολύ αυξάνεται το χωρητικό φορτίο τόσο μειώνεται η σημασία της κατανάλωσης βραχυκυκλώματος λόγω της χωρητικής κατανάλωσης P_d .

Μια δεύτερη πρακτική λύση είναι να προσπαθήσουμε να ταιριάξουμε τους χρόνους ανόδου/καθόδου των σημάτων εισόδου και εξόδου. Σε επίπεδο συνολικού κυκλώματος, αυτό σημαίνει ότι οι χρόνοι ανόδου/καθόδου όλων των σημάτων θα πρέπει να κρατηθούν εντός κάποιων ορίων. Η παραπάνω λύση μπορεί να μην είναι η βέλτιστη για την κάθε πύλη ξεχωριστά αλλά συνολικά δίνει αποτελέσματα. Αυτό μπορεί να φανεί στο σχήμα 2.5 που δείχνει την κατανάλωση βραχυκυκλώσεως ενός αντιστροφέα σε συνάρτηση με τον λόγο r μεταξύ του χρόνου ανόδου και του χρόνου καθόδου.



Σχήμα 2.5 Η κατανάλωση ισχύος ενός αντιστροφέα CMOS σε συνάρτηση με τον λόγο μεταξύ των χρόνων ανόδου/καθόδου εισόδου και εξόδου

Όταν η χωρητικότητα φορτίου είναι πολύ μικρή για ένα δεδομένο μέγεθος αντιστροφέα ($r > 2 \dots 3$ για $V_{DD} = 5V$), η ισχύς κυριαρχείται από το ρεύμα βραχυκυκλώσεως. Για μεγάλες τιμές φορτίου χωρητικότητας, όλη η κατανάλωση ισχύος αφιερώνεται στην φόρτιση και εκφόρτιση του φορτίου χωρητικότητας.

Όταν οι χρόνοι ανόδου/καθόδου εξομοιώνονται, το μεγαλύτερο μέρος της κατανάλωσης ισχύος σχετίζεται με την δυναμική ισχύ και ένα πολύ μικρό ποσοστό ($< 10\%$) είναι αφιερωμένο στα ρεύματα βραχυκύκλωσης.

Μπορούμε επίσης να παρατηρήσουμε ότι το ρεύμα βραχυκύκλωσης μειώνεται όταν χαμηλώνουμε την τάση τροφοδοσίας. Στην ακραία περίπτωση που $V_{DD} < V_{Th} + |V_{Tp}|$, η κατανάλωση βραχυκυκλώσεως μηδενίζεται καθώς ποτέ οι δύο συσκευές δεν άγουν συγχρόνως.

Με τις τάσεις κατωφλίου να μεταβάλλονται με ρυθμό μικρότερο από τις τάσεις τροφοδοσίας, η κατανάλωση βραχυκυκλώσεως γίνεται λιγότερο σημαντική σε submicron τεχνολογίες. Με τάση τροφοδοσίας της τάξης των 2.5V και κατώφλια

γύρω στα 0.5V, μια αναλογία μεταξύ του χρόνου ανόδου και του χρόνου καθόδου της τάξεως το 2 είναι αρκετή για μια 10% μείωση της κατανάλωσης.

Συνολική κατανάλωση ισχύος

Η συνολική κατανάλωση ισχύος προκύπτει από την άθροιση των τριών συνιστωσών κατανάλωσης ισχύος, δηλ.

$$P_{total} = P_s + P_d + P_{sc}$$

Σε τυπικά CMOS κυκλώματα, η χωρητική κατανάλωση είναι ο κυρίαρχος παράγοντας. Η κατανάλωση βραχυκυκλώματος μπορεί να κρατηθεί εντός συγκεκριμένων ορίων δια της προσεκτικής σχεδίασης. Η διαρροή ισχύς αγνοείται προς το παρόν, αλλά παίζει σημαντικό ρόλο στις καινούργιες τεχνολογίες .

2.1.2 Τεχνικές ελαχιστοποίησης

Θα ήταν λάθος να δούμε το θέμα της μείωσης της κατανάλωσης ισχύος χωρίς να λάβουμε υπόψη και τι άλλες παραμέτρους του κυκλώματος. Ας μην ξεχνάμε ότι η κατανάλωση ισχύος είναι μόνο ένα από τα κρίσιμα χαρακτηριστικά ενός σχεδίου και άρα θα πρέπει να μην βελτιωθεί εις βάρος των άλλων

Τα τυπικά ψηφιακά κυκλώματα έχουν συνήθως περιορισμούς που αφορούν τη καθυστέρηση ή το throughput.

Ένα σχέδιο που έχει περιορισμό καθυστέρησης πρέπει να έχει τελειώσει τους υπολογισμούς του μέχρι ενός δοσμένου χρόνου. Η interactive τηλεπικοινωνίες και τα παιχνίδια είναι παραδείγματα αυτής της κατηγορίας.

Από την άλλη πλευρά, τα σχέδια με περιορισμό στο throughput πρέπει να διατηρούν ένα σταθερό throughput δεδομένων. Για παράδειγμα μια σύνδεση 1000BaseT Gigabit Ethernet πρέπει να διατηρεί ένα σταθερό throughput της τάξης του 1 Gbit/s. Οι τεχνικές βελτιστοποίησης που είναι διαθέσιμες, διαφέρουν για κάθε περιορισμό. Η παραλληλία ή το pipelining φέρνουν αποτελέσματα για τα σχέδια με περιορισμό throughput αλλά ενδέχεται να μην είναι εφαρμόσιμα για την άλλη περίπτωση.

Με μια δεδομένη αρχιτεκτονική μπορούμε να κάνουμε ανταλλαγές μεταξύ ταχύτητας, μεγέθους και ενέργειας δια της επιλογής της τάσης τροφοδοσίας, του κατωφλιού των τρανζίστορ και του μεγέθους των συσκευών. Αυτή η στρατηγική μας δίνει μια μεγάλη γκάμα για τεχνικές ελαχιστοποίησης της ισχύος που παρουσιάζονται στο επόμενο πίνακα

	Constant Throughput/Latency	Variable Throughput/Latency	
	Design Time	Sleep Mode	Run Time
Active	Lower V_{DD} , Multi- V_{DD} , Transistor Sizing Logic Optimizations	Clock Gating	Dynamic voltage scaling
Leakage	Multi- V_{Th} + Active Techniques	Sleep transistors Variable V_{Th}	Variable V_{Th} + Active Techniques

Πίνακας [2.1] Τεχνικές ελαχιστοποίησης ισχύος

Στον πίνακα [2.1] οι τεχνικές έχουν κατηγοριοποιηθεί ως εξής:

- **Βάσει χρόνου** – Μερικές τεχνικές πραγματοποιούνται την ώρα της σχεδίασης του κυκλώματος (Design time) και αφορούν συνήθως τις διαστάσεις των τρανζίστορ. Οι τάσεις τροφοδοσίας και τα κατώφλια των τρανζίστορ μπορούν να καθοριστούν στατικά την ώρα της σχεδίασης ή δυναμικά την ώρα της εκτέλεσης (Run time). Υπάρχουν τέλος και οι τεχνικές που αφορούν τον χρόνο για τον οποίο ένα μέρος του κυκλώματος είναι ανενεργό (Sleep mode).
- **Βάσει στόχου** – Κάποιες τεχνικές στοχεύουν στην μείωση της δυναμικής κατανάλωσης (Active) ενώ κάποιες άλλες την στατική κατανάλωση ή κατανάλωση διαρροής (Leakage).

2.1.2.1 Design Time τεχνικές

Μείωση της τάσης τροφοδοσίας

Μια μείωση της τάσης τροφοδοσίας θα έχει τετραγωνική επίδραση στην δυναμική κατανάλωση ισχύος και για αυτό παραμένει μια από τις πιο διάσημες τεχνικές. Από την άλλη μεριά η μείωση της τάσης τροφοδοσίας επιφέρει πολύ μεγάλη καθυστέρηση στις πύλες CMOS. Αυτή η καθυστέρηση μπορεί να αντισταθμιστεί στο επίπεδο μονοπατιού, από μια σειρά λογικών και αρχιτεκτονικών βελτιστοποιήσεων.

Το πιο συνηθισμένο παράδειγμα είναι αυτό της αντικατάστασης ενός αθροιστή διάδοσης από έναν αθροιστή με πρόβλεψη κρατουμένου που είναι πιο γρήγορος. Θα πρέπει να παρατηρήσουμε ότι και αυτή ακόμη η αλλαγή δεν έρχεται χωρίς παράπλευρες συνέπειες, καθώς θα επιφέρει μεγαλύτερη πολυπλοκότητα στο κύκλωμα και συμπερασματικά μεγαλύτερη φυσική χωρητικότητα.

Εκτός του παραλληλισμού μια άλλη τεχνική που εξισορροπεί την απώλεια της απόδοσης είναι το pipelining.

Πολλαπλές τάσεις τροφοδοσίας

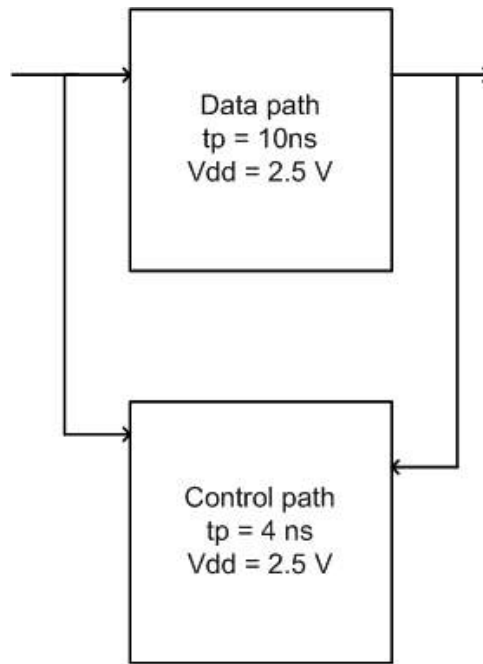
Μια ακόμη καλύτερη προσέγγιση είναι αυτή της επιλεκτικής μείωσης της τροφοδοσίας σε κάποιες πύλες που:

- ο ανταποκρίνονται σε γρήγορα μονοπάτια και τελειώνουν τους υπολογισμούς γρήγορα.
- οδηγούν μεγάλα χωρητικά φορτία. Αυτές οι πύλες θα έχουν και το μεγαλύτερο κέρδος με την ίδια βέβαια αύξηση καθυστέρησης.

Σε αυτές τις πύλες οι μείωση της τάσης τροφοδοσίας θα μειώσει την κατανάλωση ισχύος χωρίς να επιφέρει απαγορευτικές αυξήσεις στην καθυστέρηση.

Η χρήση περισσότερες της μιας τάσης τροφοδοσίας ήδη χρησιμοποιείται στα σημερινά ολοκληρωμένα κυκλώματα. Αρκεί να πούμε ότι μια ξεχωριστή τάση τροφοδοσίας χρησιμοποιείται για το I/O για λόγους συμβατότητας. Τα τρανζίστορ του δακτυλίου I/O είναι σχεδιασμένα με πιο παχιά οξείδια πύλης, για να αντέχουν μεγαλύτερες τάσεις.

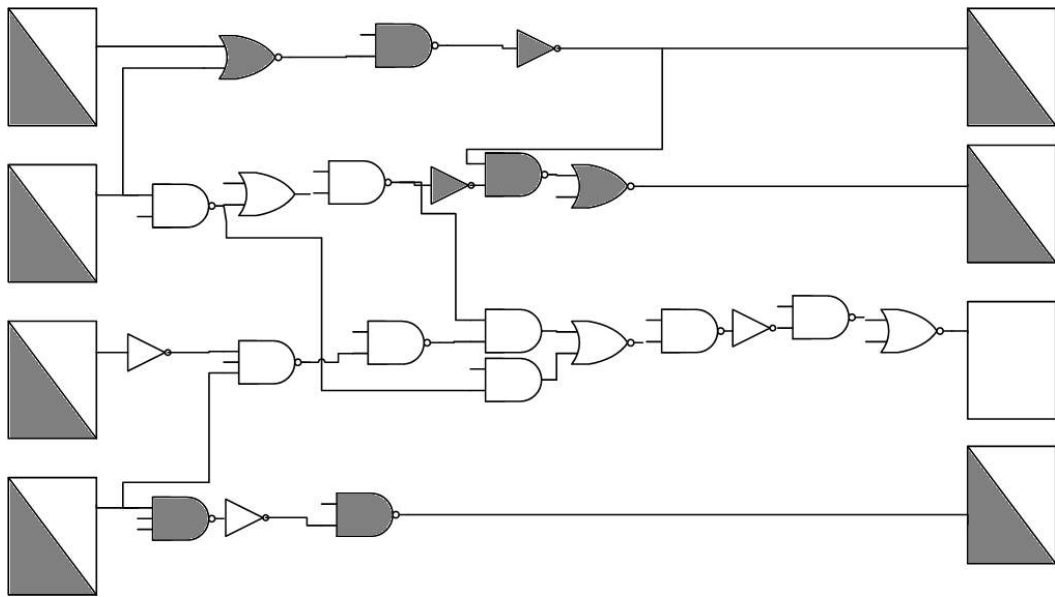
Επιλογή τάσης σε επίπεδο υπομονάδας Σε αυτήν την προσέγγιση, κάθε υπομονάδα θα επιλέγει μεταξύ δύο (ή περισσότερων) τάσεων τροφοδοσίας. Ας πάρουμε για παράδειγμα το κύκλωμα που φαίνεται στο σχήμα 2.6 . Αποτελείται από ένα μονοπάτι δεδομένων με κρίσιμο μονοπάτι των 10ns και ένα μονοπάτι ελέγχου με κρίσιμο μονοπάτι των 4ns, που λειτουργούν σε τάση τροφοδοσίας των 2.5 V. Θεωρώντας ότι το μονοπάτι των δεδομένων έχει μια σταθερή καθυστέρηση και ένα σταθερό throughput και ότι δεν μπορεί να χαμηλώσει την τάση τροφοδοσίας του, και αφού το block ελέγχου τελειώνει νωρίτερα (έχει περιθώριο χρόνου) μπορούμε να χαμηλώσουμε σε αυτό την τάση. Μια μείωση του V_{DDL} σε 1 V θα αυξήσει την καθυστέρηση του κρίσιμου μονοπατιού σε 10ns και θα μειώσει την κατανάλωση περισσότερο από 5 φορές.



Σχήμα 2.6 Σχέδιο με αποκλίνοντα μήκη κρίσιμων μονοπατιών

Πρέπει να γνωρίζουμε ότι όταν συνδυάζουμε πολλαπλές τάσεις τροφοδοσίας πάνω στο ίδιο chip, τότε υπάρχει ανάγκη για μετατροπείς στάθμης (level converters) όποτε μια μονάδα που βρίσκεται σε χαμηλότερη στάθμη τάσης πρέπει να οδηγήσει μια πύλη που βρίσκεται σε υψηλότερη στάθμη. Αυτό γίνεται γιατί εάν μια πύλη που τροφοδοτείται με V_{DDL} οδηγεί μια πύλη που τροφοδοτείται με V_{DDH} τότε το PMOS τρανζίστορ ποτέ δεν βγαίνει εκτός λειτουργίας με αποτέλεσμα ένα συνεχές στατικό ρεύμα.

Πολλαπλές τάσεις μέσα σε ένα block Η ίδια προσέγγιση υιοθετείται και στην περίπτωση που θέτουμε ξεχωριστά την τάση τροφοδοσίας για κάθε κύτταρο μέσα σε ένα block. Μελετώντας το ιστόγραμμα των καθυστερήσεων για τα κρίσιμα μονοπάτια ενός συνηθισμένου ψηφιακού block παρατηρούμε ότι πολύ λίγα μονοπάτια είναι κρίσιμα ή σχεδόν κρίσιμα και ότι τα περισσότερα μονοπάτια έχουν πολύ μικρότερες καθυστερήσεις. Αυτά τα μονοπάτια ουσιαστικά σπαταλάνε ενέργεια καθώς δεν υπάρχει όφελος στο να τελειώνουν νωρίτερα. Για κάθε ένα από αυτά τα μονοπάτια, η τάση τροφοδοσίας μπορεί να μειωθεί στο βέλτιστο επίπεδο. Θα μπορούσαμε να επιτύχουμε ελάχιστη κατανάλωση ενέργειας εάν κάθε μονοπάτι γινόταν κρίσιμο. Αυτή όμως η προσέγγιση δεν είναι ακριβώς πραγματοποιήσιμη. Μια πιο πρακτική υλοποίηση φαίνεται στο σχήμα 2.7. Χρησιμοποιώντας μια τεχνική ομαδοποιημένης μείωσης τάσης (clustered voltage-scaling) κάθε μονοπάτι ξεκινάει με την υψηλή τάση τροφοδοσίας και αλλάζει σε χαμηλότερη τάση όταν υπάρχει χρονικό περιθώριο.



Σχήμα 2.7 Τεχνική ομαδοποιημένης μείωσης τάσης. Κάθε μονοπάτι ξεκινάει με υψηλή τάση και αλλάζει σε χαμηλή (γκρι πύλες) όταν υπάρχει χρονικό περιθώριο. Η αλλαγή στάθμης γίνεται στους καταχωρητές

Η επίδραση όλων αυτών σε ένα κύκλωμα είναι ότι με την προσθήκη του δεύτερου επιπέδου τάσης περισσότερα μονοπάτια γίνονται κρίσιμα με αποτέλεσμα να έχουμε σημαντική οικονομία ενέργειας. Μελέτες έχουν δείξει ότι η προσθήκη ενός τρίτου ή και τέταρτου επιπέδου τάσης δίνουν οριακές βελτιώσεις.

Πολλαπλά Κατώφλια

Η χρησιμοποίηση συσκευών με πολλαπλά κατώφλια είναι άλλος ένας τρόπος να ανταλλάξουμε ταχύτητα με ισχύ. Οι περισσότερες υπο-0.25-μm CMOS τεχνολογίες προσφέρουν δύο τύπους τρανζίστορ τόσο για τα p-τύπου όσο και τα n-τύπου με κατώφλια που διαφέρουν κατά 100mV. Αυτά με το υψηλότερο κατώφλι δίνουν ένα ρεύμα διαρροής περίπου μια τάξη μεγέθους από αυτά με το χαμηλότερο με κόστος όμως ενός δυναμικού ρεύματος μικρότερου κατά περίπου 30%.

Συμπερασματικά τα τρανζίστορ με το χαμηλό κατώφλι χρησιμοποιούνται σε χρονικά κρίσιμα μονοπάτια ενώ τα άλλα χρησιμοποιούνται οπουδήποτε αλλού. Η βασική λογική είναι όσο περισσότερες πύλες πρέπει να μετατραπούν σε υψηλού κατωφλίου έως ότου καταναλωθεί εξ'ολοκλήρου το περιθώριο χρόνου. Μελέτες έχουν δείξει ότι μια προσεκτική ανάθεση κατωφλίων μπορεί να οδηγήσει σε μείωση του ρεύματος διαρροής έως και 90%.

Δύο από τα πλεονεκτήματα αυτής της μεθόδου είναι ότι αφενός δεν χρειάζονται μετατροπείς στάθμης και αφετέρου εκτός από την μείωση του ρεύματος διαρροής έχουμε ως επιπρόσθετο κέρδος και μια μείωση στην δυναμική ισχύ χάρις κυρίως στην μείωση της χωρητικότητας μεταξύ πύλης και καναλιού στην κατάσταση off.

Μείωση της switching χωρητικότητας μέσω της επιλογής μεγέθους του τρανζίστορ

Γνωρίζουμε ότι για μια αλυσίδα αντιστροφών με δεδομένο φορτίο και αριθμό σταδίων, η ελάχιστη καθυστέρηση επιτυγχάνεται όταν το μέγεθος του κάθε αντιστροφέα στην αλυσίδα είναι ο γεωμετρικός μέσος των γειτόνων του.

$$s_i^2 = s_{i+1} s_{i-1}$$

Όταν η ελάχιστη καθυστέρηση που επιτυγχάνεται είναι κάτω από την επιθυμητή καθυστέρηση τότε διαμορφώνεται ένα πρόβλημα βελτιστοποίησης που ελαχιστοποιεί την switching χωρητικότητα κάτω από περιορισμούς καθυστέρησης. Μια καλή επιλογή είναι να προσαρμόσουμε τον παράγοντα μείωσης σύμφωνα με την παρακάτω εξίσωση

$$s_i^2 = \frac{s_{i+1} s_{i-1}}{1 + \mu s_{i-1}}$$

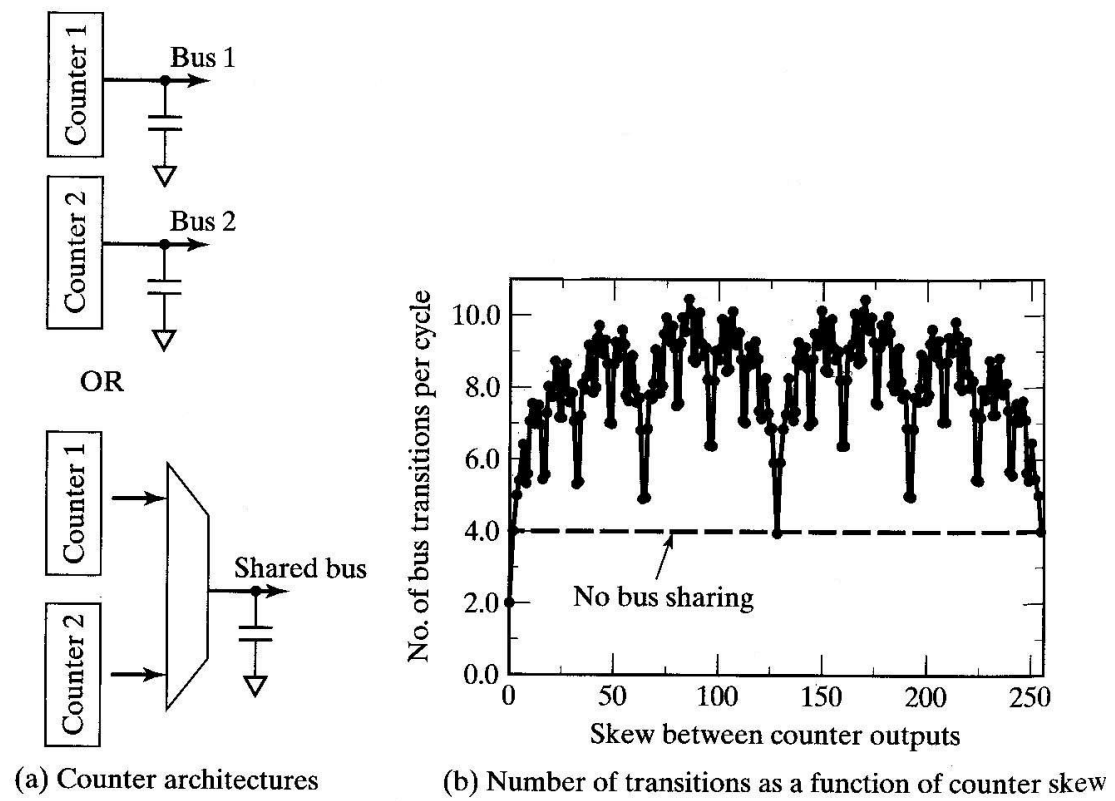
Ο παράγοντας μ είναι ένας θετικός αριθμός που εξαρτάται από το περιεχόμενο χρόνου που είναι διαθέσιμο.

Γνωρίζουμε ότι τα τελευταία στάδια στην αλυσίδα είναι και τα μεγαλύτερα, συνεπώς είναι και τα πρώτα υποψήφια για σμίκρυνση. Από την στιγμή όμως που η σμίκρυνση οποιουδήποτε από τους αντιστροφείς κατά ένα ποσοστό θα επιφέρει την ίδια αύξηση καθυστέρησης, θα ήταν καλύτερο να το κάνουμε στο στάδιο που θα έχουμε τον μεγαλύτερο αντίκτυπο που είναι οι μεγαλύτερη αντιστροφείς

Η ίδια ακριβώς αρχή χρησιμοποιείται και στα λογικά μονοπάτια. Στον αντίποδα πρέπει να παρατηρήσουμε ότι η πρακτική εφαρμογή όλων αυτών δεν είναι τόσο απλή, καθώς η σμίκρυνση ενός μονοπατιού επηρεάζει την καθυστέρηση όλων των μονοπατιών που μοιράζονται τις λογικές πύλες κάνοντας έτσι δύσκολη την απομόνωση ενός συγκεκριμένου μονοπατιού.

Μείωση του switching activity με κατανομή πόρων

Η πολυπλεξία πολλαπλών λειτουργιών σε μια μονάδα hardware έχει καταστροφικά αποτελέσματα από πλευράς κατανάλωσης ισχύος. Εξάλλου η αύξηση της φυσικής χωρητικότητας επιφέρει αύξηση του switching activity. Αυτό φαίνεται στο παράδειγμα του σχήματος 2.8. Στο (α) μέρος βλέπουμε δύο μετρητές που τρέχουν στην μία περίπτωση σε διαφορετικό hardware και στην άλλη πολυπλέκονται στην ίδια μονάδα. Στο μέρος (β) φαίνεται ο αριθμός των μεταβάσεων σαν συνάρτηση της ασυμμετρίας μεταξύ των δύο μετρητών. Όπως βλέπουμε, η πολύπλεξη τείνει να κάνει τυχαία τα σήματα δεδομένων που παρουσιάζονται στην λειτουργική μονάδα αυξάνοντας έτσι το switching activity. Συμπερασματικά, όταν θέλουμε χαμηλή κατανάλωση ισχύος πρέπει να αποφεύγεται η επαναχρησιμοποίηση πόρων και να γίνεται μια πιο προσεκτική κατανομή αυτών.



Σχήμα 2.8 Η πολύπλεξη αυξάνει το switching activity

2.1.2.2 Run Time τεχνικές

Δυναμική μεταβολή τάσης τροφοδοσίας (DVS)

Μια στατική μείωση της τάσης τροφοδοσίας μειώνει την ενέργεια ανά λειτουργία και επεκτείνει τον χρόνο ζωής της μπαταρίας σε βάρος της απόδοσης. Συχνά αυτή μείωση της επίδοσης δεν είναι αποδεκτή, ειδικά σε περιπτώσεις με περιορισμούς καθυστέρησης. Σημαντική οικονομία στην ισχύ μπορεί να γίνει εφόσον συνειδητοποιήσουμε ότι η κορυφαία επίδοση δεν είναι απαραίτητη κάθε χρονική στιγμή. Ας πάρουμε για παράδειγμα έναν επεξεργαστή γενικού σκοπού σε κάποια φορητή συσκευή όπως ένα notebook ή ένα κινητό τηλέφωνο.

Μπορούμε να χωρίσουμε τις λειτουργίες σε τρεις κατηγορίες: υπολογιστικές εργασίες, διεργασίες χαμηλής ταχύτητας και ανενεργή λειτουργία. Οι υπολογιστικές εργασίες μαζί με λειτουργίες που έχουν μικρή καθυστέρηση χρειάζονται ολόκληρο το υπολογιστικό throughput προκειμένου να ικανοποιήσουν τους χρονικούς περιορισμούς. Τα βίντεο MPEG και η αποσυμπίεση ηχητικών αρχείων είναι δυο τέτοια παραδείγματα. Εργασίες με χαμηλό throughput και μεγάλη καθυστέρηση όπως η επεξεργασία κειμένου, η είσοδος δεδομένων και τα memory backups αιτούν μόνο ένα μέρος του μέγιστου throughput του επεξεργαστή. Δεν υπάρχει καμία ανταμοιβή αν οι υπολογισμοί τελειώσουν γρήγορα ενώ εάν η εργασία τελειώσει νωρίτερα, μπορεί να θεωρηθεί σπατάλη ενέργειας. Τέλος πολύ φορητοί επεξεργαστές περνάνε μεγάλο μέρος του χρόνου λειτουργίας τους ανενεργοί.

Συνολικά λοιπόν, το υπολογιστικό throughput και η καθυστέρηση που περιμένουμε από έναν κινητό επεξεργαστή αλλάζει δραματικά στο πεδίο του χρόνου.

Η μείωση της συχνότητας του ρολογιού όταν εκτελούνται οι ‘μικρές’ εργασίες μειώνει την ισχύ αλλά δεν εξοικονομεί ενέργεια αφού κάθε λειτουργία εξακολουθεί να εκτελείται σε υψηλό επίπεδο τάσης. Ωστόσο, εάν παράλληλα με την συχνότητα μειώσουμε και την τάση τροφοδοσίας, τότε θα εξοικονομήσουμε και ενέργεια.

Προκειμένου όμως να έχουμε και υψηλό throughput για τις ‘μεγάλες’ εργασίες και να ελαχιστοποιούμε την ενέργεια για τις ‘μικρές’, τόσο η συχνότητα όσο και η τάση τροφοδοσίας πρέπει να μεταβάλλονται δυναμικά κατά την διάρκεια της εκτέλεσης. Αυτή η τεχνική καλείται dynamic voltage scaling (DVS) και βασική της αρχή είναι ότι κάθε λειτουργία πρέπει να πραγματοποιείται στην χαμηλότερη τάση τροφοδοσίας που ικανοποιεί τους χρονικούς περιορισμούς.

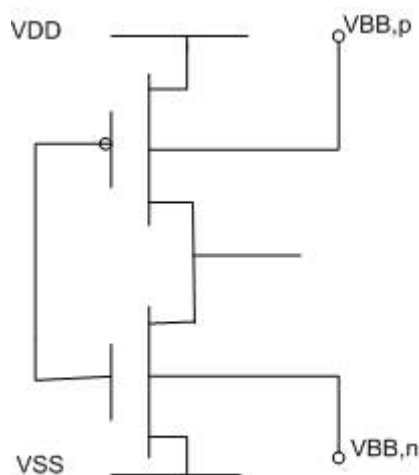
Μια πρακτική υλοποίηση του DVS αποτελείται από τα παρακάτω στοιχεία;

- Ένας επεξεργαστής που μπορεί να δουλεύει κάτω από ένα εύρος τάσεων τροφοδοσίας.
- Ένας ρυθμιστής τροφοδοσίας που θέτει την ελάχιστη τάση που είναι αναγκαία για την λειτουργία στην επιθυμητή συχνότητα
- Ένα λειτουργικό σύστημα (ή scheduler) που υπολογίζει της απαιτούμενες συχνότητες για να εξυπηρετήσουν τα ζητούμενα throughput και τις ζητούμενες διορίες ολοκλήρωσης

Ο ρόλος του λειτουργικού συστήματος είναι να καθορίσει δυναμικά την βέλτιστη συχνότητα (ή τάση) σαν συνάρτηση των συνδυασμένων υπολογιστικών απαιτήσεων όλων των ενεργών εργασιών του συστήματος. Σε μια ακόμη πιο περίπλοκη περίπτωση, κάθε εργασία θα παρέχει μια διορία ολοκλήρωσης ή μια επιθυμητή συχνότητα εκτέλεσης. Σε αυτή την περίπτωση ο scheduler κάνει μια εκτίμηση για το πόσοι κύκλοι του επεξεργαστή απαιτούνται για την ολοκλήρωση κάθε μιας από τις λειτουργίες και υπολογίζει την βέλτιστη συχνότητα για τον επεξεργαστή.

Δυναμική μεταβολή κατωφλίου (DTS)

Κατ' αναλογία με την μεταβολή της τάσης, μπορεί και η δυναμική μεταβολή κατωφλίου να εξοικονομήσει ισχύ. Για υπολογισμούς με μικρή καθυστέρηση, το κατώφλι πρέπει να μειωθεί στην χαμηλότερη τιμή του. Για υπολογισμούς με χαμηλή ταχύτητα μπορεί να αυξηθεί, και στην φάση αναμονής πρέπει να τεθεί στην υψηλότερη δυνατή τιμή ώστε να ελαχιστοποιήσει το ρεύμα διαρροής. Το εργαλείο που μας επιτρέπει να αλλάζουμε τα κατώφλια δυναμικά είναι η πόλωση του υποστρώματος. Για να γίνει αυτό θα πρέπει να υπάρχει ανεξάρτητος έλεγχος όλων των κομματιών τόσο των p- τύπου συσκευών όσο και των n-τύπου, όπως φαίνεται και στο σχήμα 2.9.



Σχήμα 2.9 Ένας αντιστροφέας με έλεγχο κατωφλίου

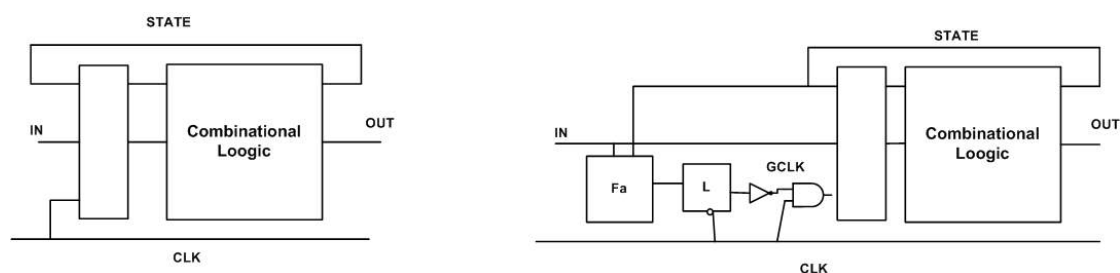
Βάση του σχήματος για το DTS είναι ένας βρόχος ανάδρασης ο οποίος αναλαμβάνει να επιτύχει τους παρακάτω στόχους:

- Να χαμηλώνει την διαρροή στην φάση αναμονής.
- Να μπορεί να αντισταθμίζει τις αποκλίσεις των κατωφλιών σε όλο το chip κατά την διάρκεια της κανονικής λειτουργίας
- Να μπορεί να μειώσει το throughput του κυκλώματος για να μειώσει τόσο την δυναμική ισχύ όσο και την ισχύ διαρροής, βασισμένος σε περιορισμούς επίδοσης

2.1.2.3 Μείωση της Ισχύος στην Ανενεργή (sleep) Φάση

Η τεχνική του clock gating παρέχει έναν τρόπο να σταματάμε επιλεκτικά το ρολόι σε ορισμένες μονάδες που τα αποτελέσματα των υπολογισμών τους είναι άχρηστα, έτσι ώστε να αποτρέψουμε την μετάβαση του κυκλώματος από μια κατάσταση σε μια άλλη. Με άλλα λόγια το σήμα του ρολογιού απενεργοποιείται σύμφωνα με τις ανενεργές φάσεις του σχεδίου.

Ας πάρουμε για παράδειγμα της τεχνικής του clock gating το κύκλωμα στο αριστερό μέρος του σχήματος 2.10 το οποίο αποτελείται από ένα block συνδυαστικής λογικής και από ένα πίνακα με καταχωρητές κατάστασης ενώ έχουμε και ανάδραση. Στο δεξί μέρος του σχήματος 2.10 φαίνεται η μορφή του κυκλώματος μετά την εφαρμογή της τεχνικής του clock gating.



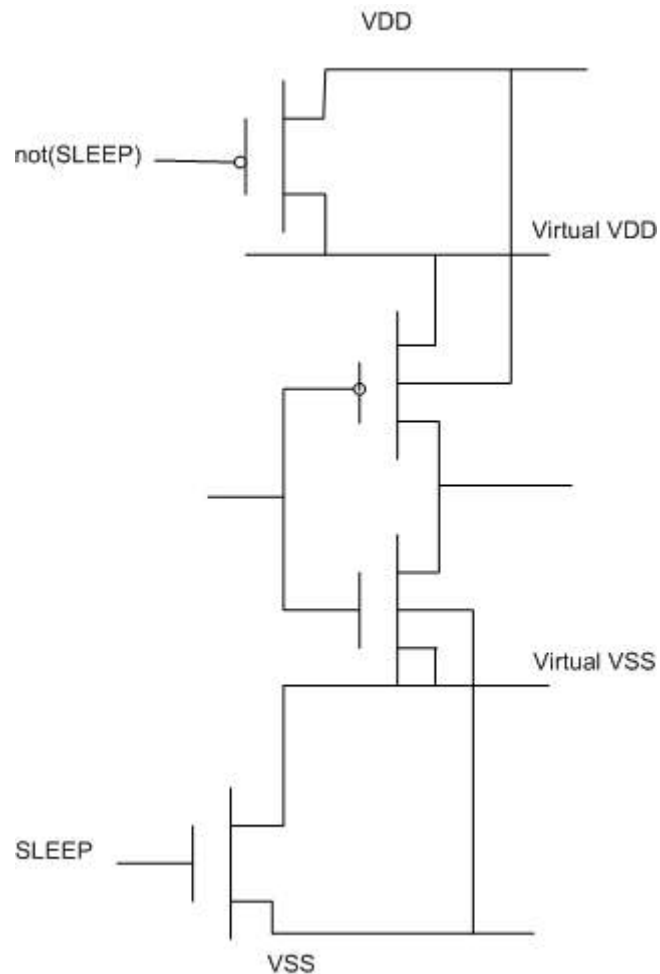
Σχήμα 2.10 Παράδειγμα για αρχιτεκτονική gate clocking

Έχουμε υποθέσει ότι το κύκλωμα έχει ένα απλό ρολόι και ότι οι καταχωρητές κατάστασης είναι flip-flop που ενεργοποιούνται από τις ακμές του ρολογιού. Το συνδυαστικό block Fa ελέγχεται από τις εισόδους του κυκλώματος, την τρέχουσα κατάσταση και τις εξόδους του κυκλώματος και αποτελεί τον μηχανισμό ελέγχου του μηχανισμού του clock gating. Σκοπός του είναι να σταματάει επιλεκτικά το ρολόι όταν δεν συμβαίνει καμία μεταβολή κατάστασης ή εξόδου. Το block L είναι ένας μανδαλωτής ο οποίος είναι 'διάφανος' όταν το ρολόι είναι ανενεργό. Η παρουσία του είναι απαραίτητη καθώς φιλτράρει ενδεχόμενα glitch τα οποία μπορεί να προκύψουν στην έξοδο του Fa.

Από την άλλη πλευρά η σχεδίαση του κυκλώματος ελέγχου μπορεί να φέρει επιπλέον πολυπλοκότητα στο κύκλωμα άρα και επιπλέον μέγεθος και κατανάλωση ισχύος, γι 'αυτό και θα πρέπει να είναι όσο πιο απλή γίνεται ώστε να σταματάει το ρολόι με την μέγιστη αποτελεσματικότητα ξοδεύοντας την ελάχιστη δυνατή ισχύ. Ας αναφέρουμε ότι η τεχνική του clock gating χρησιμοποιείται και στον Power Compiler του Synopsys.

Η ισχύς που εξοικονομείται με το clock gating δεν έρχεται χωρίς τίμημα καθώς έχουμε μια μείωση στην απόδοση αφού η επαναφορά της λειτουργίας ενός προηγούμενως ανενεργού κομματιού εισάγει καθυστέρηση.

Στην φάση της αναμονής καθώς καμία ενεργή μετάβαση δεν συμβαίνει, όλη η κατανάλωση ισχύος οφείλεται στο ρεύμα διαρροής. Για να το αντιμετωπίσουμε αυτό χρησιμοποιούμε ένα σχήμα με sleep τρανζίστορ τα οποία βγάζουν εκτός κυκλώματος τις ράγες τροφοδοσίας όταν το κύκλωμα είναι σε ανενεργή φάση. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας έναν διακόπτη ισχύος στην ράγα τροφοδοσίας ή ακόμη καλύτερα και στην ράγα της γείωσης όπως φαίνεται και στο σχήμα 2.11



Σχήμα 2.11 Χρησιμοποίηση SLEEP τρανζίστορ τόσο στην τροφοδοσία όσο και στην γείωση

Στην φάση κανονικής λειτουργίας το σήμα SLEEP είναι ψηλά και τα sleep τρανζίστορ πρέπει να παρουσιάζουν όσο το δυνατόν μικρότερη αντίσταση. Το μέγεθος καθώς και η επιλογή κατωφλίων για αυτά τρανζίστορ αποτελούν μια διαδικασία ανταλλαγής. Για να ελαχιστοποιήσουμε τις διακυμάνσεις στην τάση τροφοδοσίας, τα sleep τρανζίστορ πρέπει να έχουν πολύ μικρή αντίσταση στην κατάσταση on και γι' αυτό θα πρέπει να είναι μεγάλα. Αυτή όμως η αύξηση μεγέθους επιφέρει μεγάλη ζημιά στο layout. Όταν έχουμε στην διάθεση μας τρανζίστορ με υψηλότερο κατώφλι μπορούμε να επιτύχουμε ακόμη καλύτερο περιορισμό της διαρροής.

Εάν σε κάποιο block σβήσουμε την τροφοδοσία, τότε αυτομάτως σβήνεται από τους καταχωρητές η κατάσταση στην οποία βρίσκονταν. Σε κάποιες πολύ ιδιαίτερες περιπτώσεις αυτό μπορεί να μην μας πειράζει, όπως όταν έχουμε να εκτελέσουμε μια εντελώς καινούργια διεργασία μετά το ξύπνημα του block. Σε περιπτώσεις όμως που μας ενδιαφέρει να κρατήσουμε την προηγούμενη κατάσταση, τότε μπορούμε να αναθέσουμε στο λειτουργικό σύστημα να σώσει τους συγκεκριμένους καταχωρητές σε μια σταθερή μνήμη.

2.2 Εισαγωγή στη VHDL

2.2.1 Εισαγωγικά

Η VHDL είναι μια από τις πιο διάσημες γλώσσες περιγραφής υλικού (HDL) που επιτρέπει την περιγραφή της συμπεριφοράς και της δομής ενός συστήματος. Όπως και οι υψηλού επιπέδου γλώσσες, η VHDL παρέχει όλα τα στοιχεία των σύγχρονων γλωσσών προγραμματισμού (δομή, παραμετροποίηση, υποθετικότητες, βρόχους και ιεραρχία)

VHDL = VHSIC Hardware Description Language

VHSIC = Very High Speed Integrated Circuits

2.2.2 Ιστορικά

Η γλώσσα περιγραφής VHDL αρχικά αναπτύχθηκε κατ'εντολήν του Υπουργείου Άμυνας των ΗΠΑ προκειμένου να είναι δυνατή η τεκμηρίωση της συμπεριφοράς των ASIC που περιλαμβάνονταν στον εξοπλισμό που προμηθεύονταν από τις εταιρείες. Έτσι η VHDL αναπτύχθηκε ως εναλλακτική λύση έναντι των τεράστιων, σε μέγεθος και πολυπλοκότητα, εγχειριδίων χρήσης

Η VHDL έχει σύνταξη παρόμοια με της Pascal και της Ada, όντας απόγονος της Algol.

Η ιδέα της δυνατότητας προσομοίωσης της συμπεριφοράς ήταν τόσο ελκυστική που αμέσως αναπτύχθηκαν λογικοί προσομοιωτές που διάβαζαν VHDL αρχεία. Το επόμενο βήμα ήταν η ανάπτυξη εργαλείων που θα διάβαζαν VHDL αρχεία και θα έδιναν στην έξοδο έναν προσδιορισμό της φυσικής υλοποίησης του κυκλώματος. Κάποια μοντέρνα εργαλεία σύνθεσης μπορούν να εξάγουν από τον κώδικα ολόκληρα block από RAM, μετρητές και αριθμητικά κυκλώματα και να τα υλοποιήσουν ανάλογα με τις ανάγκες του χρήστη. Έτσι ο ίδιος VHDL κώδικας μπορεί να γίνει αντικείμενο διαφορετικής σύνθεσης είτε για την ελαχιστοποίηση του κόστους, της ενέργειας, για την υψηλότερη ταχύτητα είτε για άλλες απαιτήσεις.

Η αρχική έκδοση της VHDL που σχεδιάστηκε με βάση το IEEE standard 1076-1987, περιείχε ένα μεγάλο εύρος από τύπους δεδομένων, συμπεριλαμβανομένων αριθμητικών (integer και real), λογικών (bit και boolean), χαρακτήρων και χρόνου καθώς και πίνακες από bit (bit_vectors) και από χαρακτήρες (string).

Το δεύτερο τεύχος του IEEE 1076 το 1993, έκανε την σύνταξη πιο συνεπή, επέτρεψε μεγαλύτερη ελαστικότητα στην ονομασία, επέκτεινε τον τύπο του χαρακτήρα ώστε να ταιριάζει στον πλήρη 8-bit ASCII και τέλος πρόσθεσε τον τελεστή του xnor.

2.2.3 Γενικά

Η VHDL είναι στην ουσία μια γλώσσα προγραμματισμού γενικού σκοπού. Ένα πρόγραμμα VHDL μπορεί να γραφεί με τέτοιο τρόπο έτσι ώστε να δημιουργεί ένα άλλο πρόγραμμα VHDL το οποίο θα ενσωματωθεί στο σχέδιο το οποίο αναπτύσσεται. Χάρης σε αυτό τον χαρακτήρα γενικού σκοπού, είναι δυνατόν η VHDL να χρησιμοποιηθεί ώστε να γραφεί ένα testbench που θα επιβεβαιώνει την λειτουργικότητα του σχεδίου χρησιμοποιώντας αρχεία που ορίζουν τις διεγέρσεις και την αλληλεπίδραση με τον χρήστη αλλά και θα συγκρίνει τα προκείμενα αποτελέσματα με τα αναμενόμενα.

Κάποια χαρακτηριστικά της VHDL που την κάνουν να διαφέρει από τις άλλες HDL γλώσσες είναι ότι μπορεί να χρησιμοποιηθεί ως μέσω ανταλλαγής της περιγραφής ολοκληρωμένων κυκλωμάτων και των προμηθευτών εργαλείων CAD με σκοπό την αυτοματοποίηση και την επιτάχυνση της διαδικασίας κατασκευής, αλλά και ως μέσο επικοινωνίας μεταξύ σχεδιαστικών προγραμμάτων και προγραμμάτων προσομοίωσης.

Επίσης η VHDL είναι ανεξάρτητη από την τεχνολογία κατασκευής του ολοκληρωμένου κυκλώματος, αλλά μπορεί να υποστηρίξει τεχνολογικά χαρακτηριστικά. Η μοντελοποίηση σε VHDL μπορεί να χρησιμοποιηθεί για την κατασκευή ολοκληρωμένων κυκλωμάτων από διαφορετικούς κατασκευαστές επειδή είναι ανεξάρτητη τεχνολογίας.

Είναι σε θέση να υποστηρίξει διαφορετικά επίπεδα ψηφιακών τεχνικών περιγραφής και μοντελοποίησης όπως, περιγραφής πεπερασμένων μηχανών καταστάσεως, περιγραφή αλγορίθμου, περιγραφή λογικών συναρτήσεων, ενώ μπορεί επίσης να διαβάζει βιβλιοθήκες στοιχείων από διαφορετικούς κατασκευαστές που έχουν γραφεί σε VHDL.

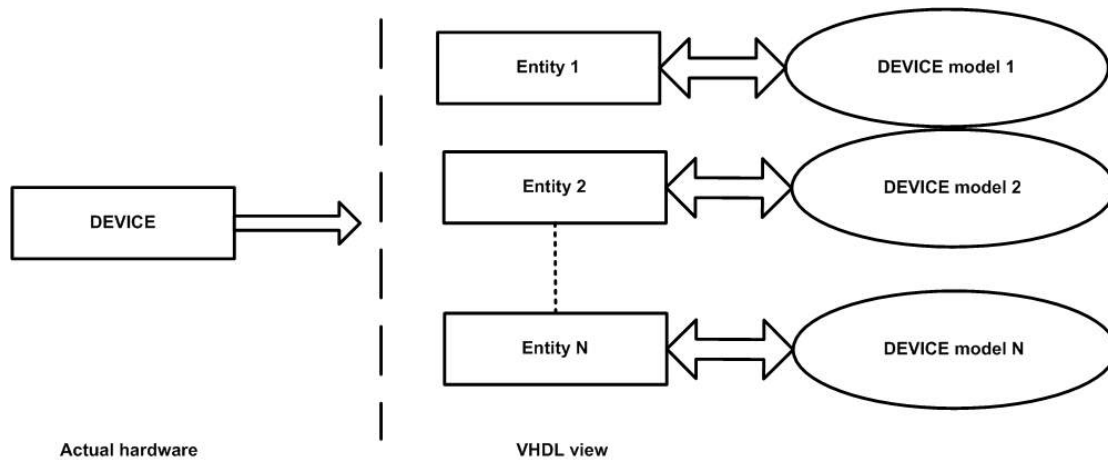
2.2.4 Δομικά

Η VHDL υποστηρίζει τους εξής τρεις βασικούς και διαφορετικούς τρόπους περιγραφής:

- Λίστα των συνδέσεων των components (Structure)
- Λίστα αναθέσεων ταυτόχρονης εκτέλεσης (Dataflow)
- Λίστα αναθέσεων ακολουθιακής εκτέλεσης (Behavior)

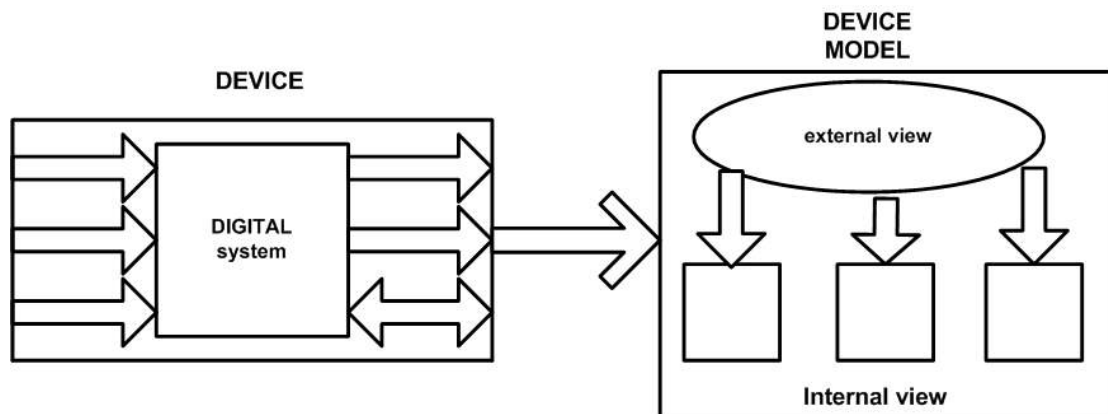
Ένα κύκλωμα μπορεί να περιγραφεί χρησιμοποιώντας είτε έναν από τους παραπάνω τρόπους περιγραφής είτε έναν συνδυασμό αυτών ταυτόχρονα.

Στην VHDL κάθε μοντέλο κυκλώματος ονομάζεται entity και έτσι ένα κύκλωμα μπορεί να περιγράφεται από διαφορετικά entity όπως στο παρακάτω σχήμα:



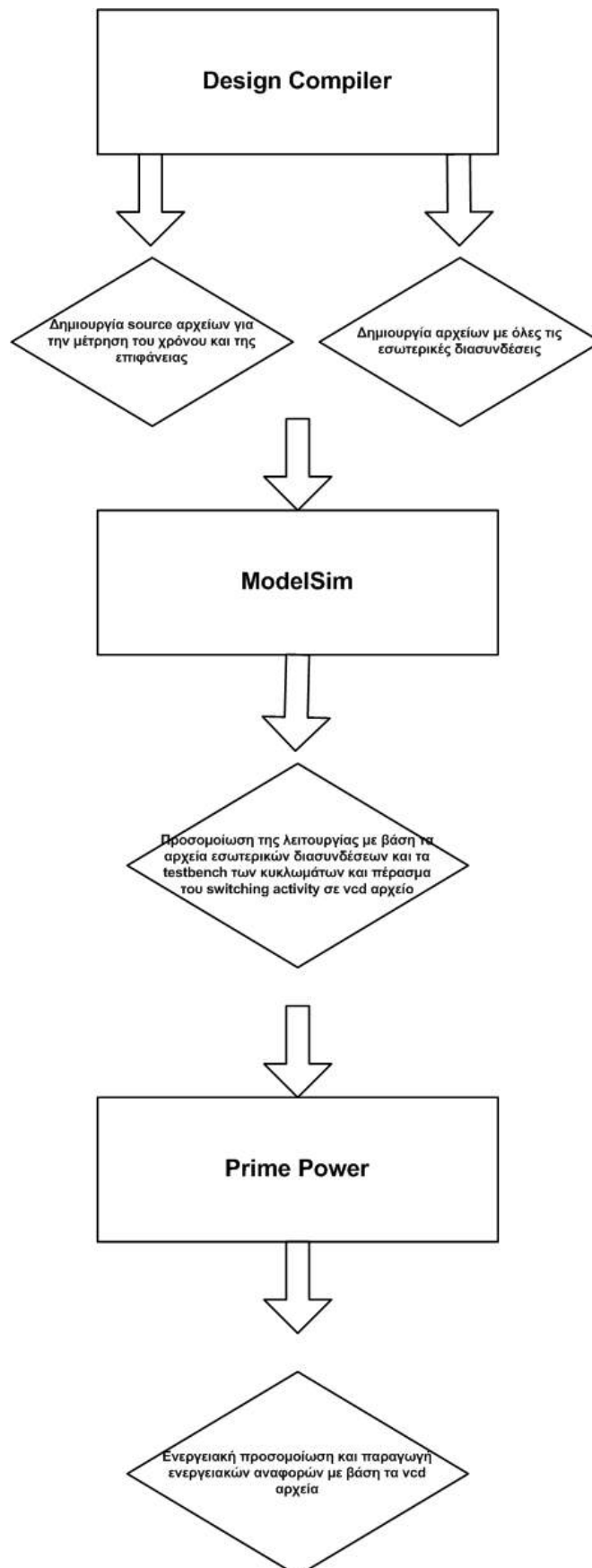
Σχήμα 2.12 Πολλαπλή περιγραφή κυκλώματος σε VHDL

Τέλος η VHDL χρησιμοποιείται για την περιγραφή ενός μοντέλου σε ένα ψηφιακό ολοκληρωμένο κύκλωμα. Το μοντέλο αυτό καθορίζει την εξωτερική συμπεριφορά του κυκλώματος καθώς και διάφορες όψεις της εσωτερικής του σχεδίασης. Η εσωτερική σχεδίαση ορίζει την εσωτερική δομή και την λειτουργία του κυκλώματος, ενώ η εξωτερική συμπεριφορά καθορίζει την σύνδεση του κυκλώματος με τα υπόλοιπα κυκλώματα που το περιβάλλουν. Αντιπροσωπευτικό είναι το επόμενο σχήμα:



Σχήμα 2.13 Κύκλωμα και μοντέλο

2.3 EDA TOOLS



Στο παραπάνω σχήμα φαίνεται ένα διάγραμμα ροής της γενικότερης πορείας μας ώστε να φτάσουμε στις ενεργειακές προσομοιώσεις και στις μετρήσεις για επιφάνεια, χώρο και ενέργεια.

Αρχικά λοιπόν περνάμε τα vhdl αρχεία των κυκλωμάτων στον Design Compiler και μέσω αρχείων source που δημιουργούμε ,τα οποία κάνουνε υλοποίηση των κυκλωμάτων και σύνδεσή τους με τις βιβλιοθήκες, παίρνουμε τις μετρήσεις για την επιφάνεια και των χρόνο των κυκλωμάτων. Επίσης με την βοήθεια του Design Vision παράγουμε αρχεία που περιέχουν όλες τις εσωτερικές διασυνδέσεις των κυκλωμάτων τα οποία θα χρησιμοποιήσει το ModelSim.

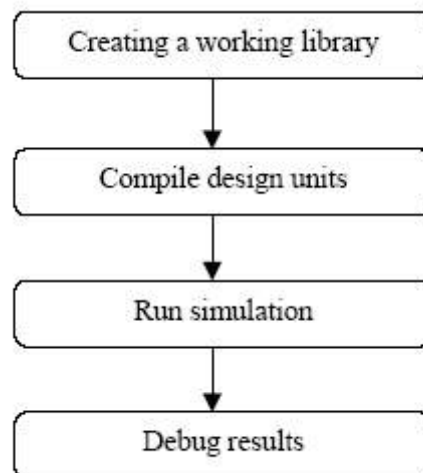
Το ModelSim θα προσομοιώσει την λειτουργία των κυκλωμάτων μέσω των αρχείων εσωτερικών διασυνδέσεων και μέσω των αρχείων testbench με εισόδους έναν μεγάλο πλήθος αριθμών. Όλα τα αποτελέσματα της προσομοίωσης και κυρίως το switching activity θα περαστεί σε αρχεία vcd που θα χρησιμοποιηθούν από το Prime Power.

Τελικά το Prime Power θα πάρει τα αρχεία εσωτερικών διασυνδέσεων μαζί με τα vcd που περιέχουν το switching activity και θα προχωρήσει σε ενεργειακές προσομοιώσεις που θα καταλήξουν σε αναφορές για την συνολική καταναλισκόμενη ισχύ, για την κατανάλωση διαρροής και για την δυναμική κατανάλωση.

2.3.1 Εργαλεία προσομοίωσης

- ModelSim

Αρχικά λοιπόν γράφουμε τον κώδικά σε VHDL και στην συνέχεια τον ελέγχουμε. Η προσομοίωση γίνεται με το ModelSim που είναι ένα εργαλείο προσομοίωσης και αποσφαλμάτωσης για VHDL, Verilog . Παρακάτω φαίνεται το διάγραμμα ροής για την προσομοίωση



Σχήμα 2.15 Διάγραμμα ροής για την προσομοίωση στο ModelSim

Αρχικά δημιουργούμε μια βιβλιοθήκη εργασίας στην οποία δουλεύουμε και ακολούθως κάνουμε compile και προσομοιώνουμε τα σχέδια μας.

Μέσα από την προσομοίωση του ModelSim μπορούμε να περάσουμε το switching activity σε ένα αρχείο VCD το οποίο μετά θα χρησιμοποιήσει το Prime Power για να μας δώσει λεπτομερείς αναφορές ισχύος.

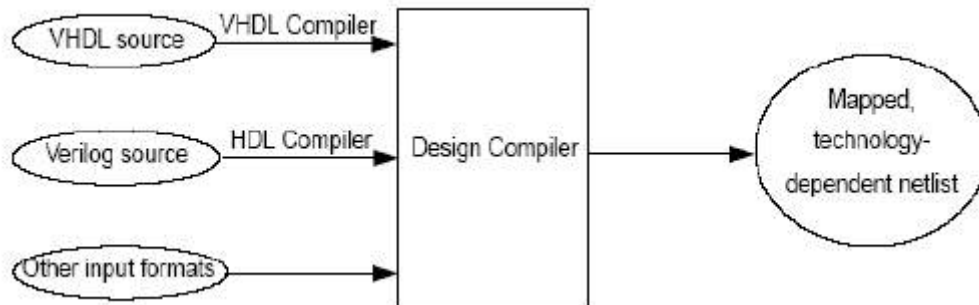
2.3.2 Εργαλεία Σύνθεσης

ο Design Compiler

Ο Design Compiler είναι από τα πολλά εργαλεία του Synopsys. Ο Design Compiler περιέχει εργαλεία τα οποία συνθέτουν HDL σχέδια σε βελτιστοποιημένα σχέδια σε επίπεδο πυλών. Υποστηρίζει μια μεγάλη γκάμα από ιεραρχικά και μη σχεδιαστικά μοντέλα ενώ μπορεί να βελτιστοποιήσει σε σχέση με το μέγεθος την ταχύτητα και την ενέργεια τόσο ακολουθιακά όσο και συνδυαστικά σχέδια.

Ο Design Compiler διαβάζει και γράφει σε EDA format το οποίο περιέχει τα format του Synopsys internal database (.db) και equation (.eqn). Ο Design Compiler περιέχει επίσης και τα ακόλουθα εργαλεία DC Professional , DC Expert , DFT Compiler , DC Ultra και DC Ultra Plus

Στο ακόλουθο σχήμα φαίνεται η διαδικασία σύνθεσης στον Design Compiler



Σχήμα 2.16 Διαδικασία σύνθεσης στον Design Compiler

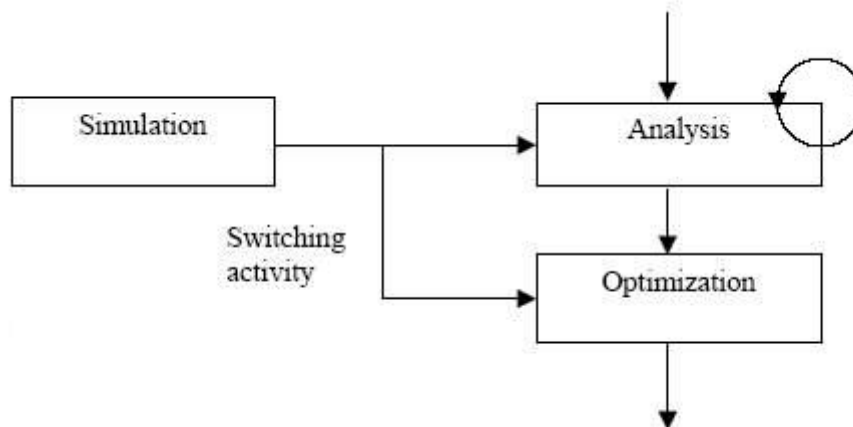
Τα προαναφερθέντα προϊόντα παράγουν αποδοτικά ASIC σχέδια χρησιμοποιώντας FPGA και έτοιμες βιβλιοθήκες. Αυτά τα εργαλεία βοηθούν στην εξερεύνηση σχεδιαστικών ανταλλαγών που αφορούν σχεδιαστικούς περιορισμούς στην ενέργεια ,στο μέγεθος και στην ταχύτητα κάτω υπό διάφορα φορτία και διάφορες συνθήκες τάσης και θερμοκρασίας.

Ο Design Compiler μας επιτρέπει εκτός από την σύνθεση των κυκλωμάτων και την προσομοίωσή τους με απώτερο σκοπό την εξαγωγή των αναφορών για area και timing.

2.3.3 Εργαλεία Ισχύος

- Power compiler

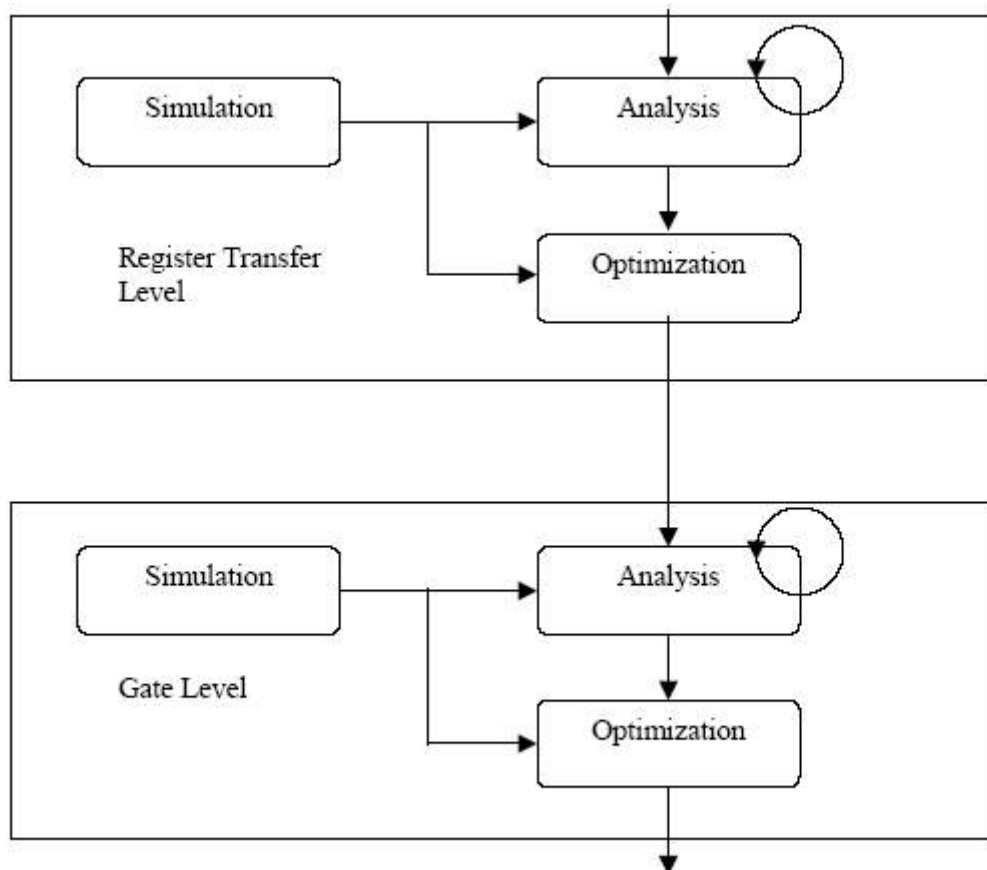
Ο Power compiler χρησιμοποιείται σε επίπεδο RTL και επίπεδο πυλών για να παρουσιάσει τα ενεργειακά στοιχεία ή για να τα βελτιστοποιήσει. Σε κάθε επίπεδο χρησιμοποιείται η ανάλυση και η εξομοίωση ,προκειμένου να βελτιωθεί το σχέδιο πριν περάσουμε στο αμέσως χαμηλότερο επίπεδο. Η εξομοίωση παρέχει το switching activity που δίνει τις απαραίτητες πληροφορίες για την βελτιστοποίηση. Όσο υψηλότερο το επίπεδο τόσο περισσότερη ενέργεια μπορούμε να εξοικονομήσουμε.



Σχήμα 2.17 Ροή ενέργειας σε κάθε επίπεδο.

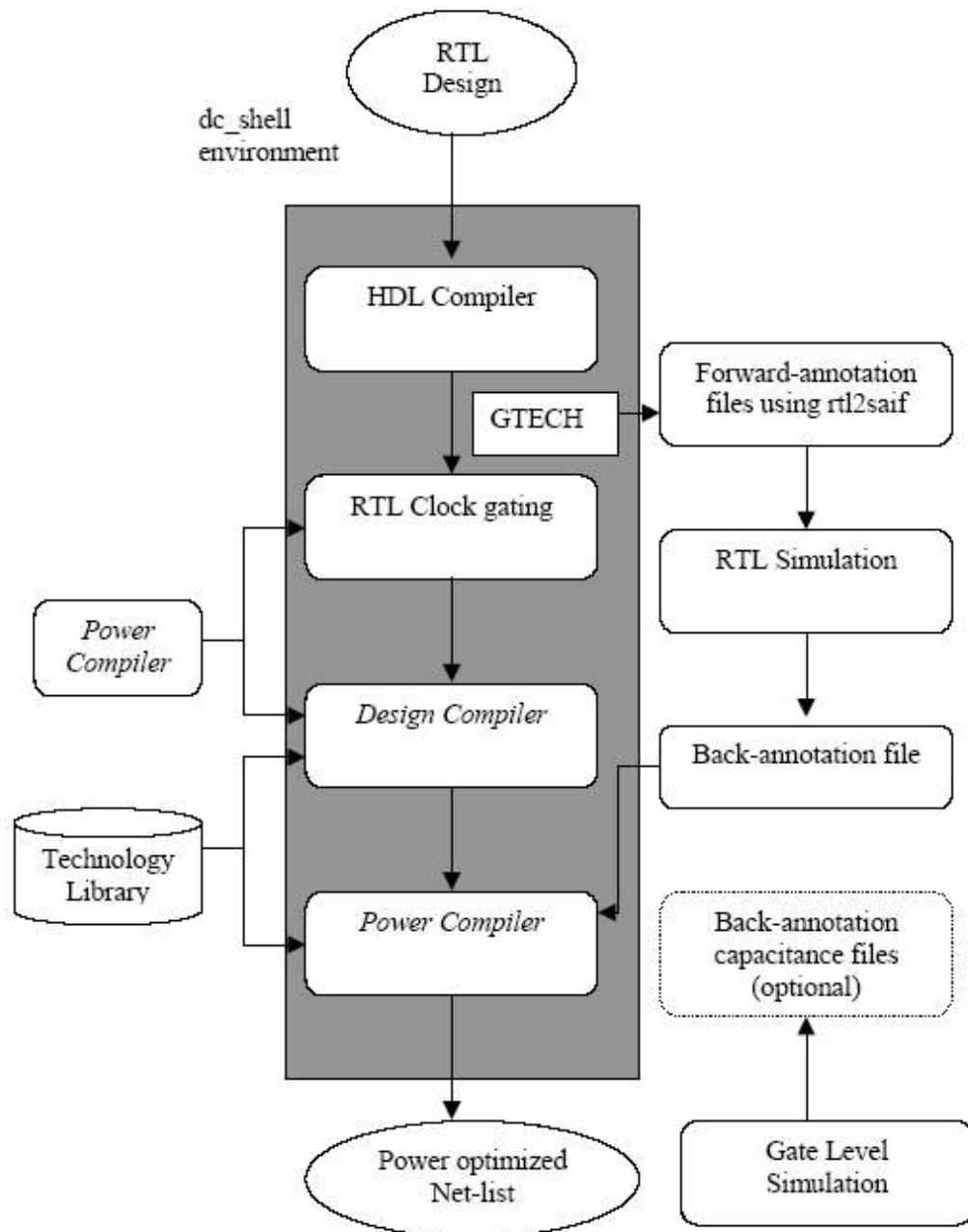
Η εσωτερική ενέργεια του κυττάρου και η μετάβαση του δικτύου επηρεάζουν άμεσα την δυναμική ισχύ του σχεδίου. Για να παρουσιάσει και για να βελτιστοποιήσει την ενέργεια ο Power Compiler χρειάζεται πληροφορία μετάβασης. Αυτή η πληροφορία μετάβασης ονομάζεται switching activity

Ο Power Compiler μοντελοποιεί το switching activity σε όρους στατικής πιθανότητας και ρυθμού μεταβολής. Στατική πιθανότητα είναι η πιθανότητα ένα σήμα να βρίσκεται σε μια συγκεκριμένη λογική κατάσταση και εκφράζεται με ένα νούμερο ανάμεσα στο 0 και στο 1. Ο υπολογισμός γίνεται κατά την διάρκεια της προσομοίωσης του σχεδίου μέσω της σύγκρισης του χρόνου για τον οποίο το σήμα βρίσκεται σε αυτή την κατάσταση και του συνολικού χρόνου της προσομοίωσης. Ρυθμός μεταβολής είναι ο αριθμός των μεταβάσεων από το λογικό1 στο λογικό 0 και ανάποδα ανά μονάδα χρόνου.



Σχήμα 2.18 Ροή ενέργειας από το επίπεδο RTL στο επίπεδο πύλης

Το σχήμα 2.19 δείχνει την μεθοδολογία υπολογισμού ενέργειας με τον συνδυασμό του Power Compiler και του Design Compiler. Πριν γίνει ο υπολογισμός της ενέργειας από τον Power Compiler πρέπει να παραχθεί η net list σε επίπεδο πύλης από τον Design Compiler. Το κύριο μέλημα της προσομοίωσης είναι η πληροφόρηση σχετικά με το switching activity και η δημιουργία ενός αρχείου με την ονομασία Back- annotation το οποίο περιέχει το switching activity από την RTL προσομοίωση και την προσομοίωση σε επίπεδο πυλών.



Σχήμα 2.19 Μεθοδολογία υπολογισμού ενέργειας στον Power Compiler

- Prime Power

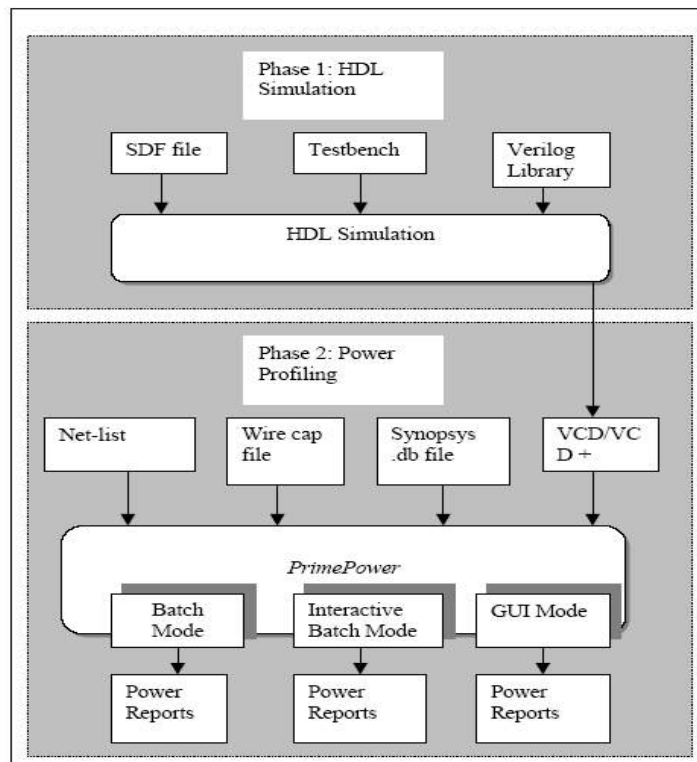
Το άλλο εργαλείο που χρησιμοποιούμε για τον υπολογισμό της ενέργειας είναι το Prime Power. Το Prime Power είναι ένα πολύ χρήσιμο δυναμικό εργαλείο για την ανάλυση και την προσομοίωση σε επίπεδο πυλών, το οποίο υπολογίζει και αναλύει με μεγάλη ακρίβεια την κατανάλωση ισχύος. Επίσης είναι αρκετά φιλικό προς τον χρήστη, με τις ευνόητες και λεπτομερείς ενεργειακές αναφορές του. Χάρης σε αυτά τα χαρακτηριστικά του αποτελεί πολύ χρήσιμο εργαλείο για τους σχεδιαστές κυκλωμάτων που αφορούν τηλεπικοινωνίες και φορητούς υπολογιστές όπου η κατανάλωση ισχύος είναι πολύ κρίσιμο ζήτημα.

Ο κύκλος εργασιών του Prime Power χωρίζεται σε δύο φάσεις

- a) Προσομοίωση HDL
- b) Power profiling

Κατά την πρώτη φάση το Prime Power μεταφράζει το switching activity σε ένα αρχείο VCD.

Στην δεύτερη φάση παράγεται η ενεργειακή αναφορά που δίνει το ενεργειακό προφίλ σε επίπεδο κυττάρου, block και chip. Για να γίνει αυτό είναι απαραίτητο το switching activity, η συνδεσιμότητα (connectivity) του σχεδίου και πληροφορίες σχετικά με την pin-to-pin καθυστέρηση. Το connectivity παρέχεται σε μορφή VHDL σε net list αρχεία, ο pin-to-pin χρονισμός απορρέει από τις βιβλιοθήκες του Synopsys



Σχήμα 2.20 Μεθοδολογία υπολογισμού στο Prime Power

Πιο αναλυτικά, οι είσοδοι της πρώτης φάσης είναι αρχεία SDF , αρχεία testbench και βιβλιοθήκες Verilog. Το VCD αρχείο που παράγεται , περνάει στο Prime Power μαζί με τα db Net-list αρχεία του Synopsys και ανάλογα στην κατάσταση λειτουργίας του Prime Power (Batch, GUI ...) παίρνουμε τις αντίστοιχες αναφορές ενέργειας. Το GUI δίνει την δυνατότητα γραφικής απεικόνισης των αποτελεσμάτων.

Η ενεργειακή ανάλυση με το Prime Power καθορίζει την κατανάλωση ισχύος του chip με βάση το switching activity. Καθορίζει τόσο την δυναμική ισχύ όσο και την στατική. Η Στατική αναφέρεται συχνά ως διαρροή ισχύς (Leakage Power). Τέλος μια ακόμη χρήσιμη λειτουργία του Prime Power είναι ο καθορισμός του λεγόμενου Glitch power. Αν δύο μεταβολές είναι πολύ κοντά η μια στην άλλη τόσο όσο ο χρόνος μεταξύ τους να είναι μικρότερος από τον χρόνο ανόδου και καθόδου του συγκεκριμένου pin τότε έχουμε glitch.

Κεφάλαιο 3

Κυκλώματα

3.1 FULL ADDER

Το βασικό δομικό κύτταρο για κάθε αριθμητικό κύκλωμα γενικότερα , αλλά και ειδικότερα για τα κυκλώματα πολλαπλασιαστών , είναι ο πλήρης αθροιστής (full adder FA). Ο πλήρης αθροιστής θα πρέπει να δέχεται τρεις εισόδους: οι δύο θα είναι από τους αντίστοιχους προστιθέμενους δυαδικούς αριθμούς και η τρίτη ένα κρατούμενο από την προηγούμενη βαθμίδα.. Θα έχει δύο εξόδους: η μια θα είναι το αποτέλεσμα-άθροισμα των δύο bits ων προσθετέων και η δεύτερη το κρατούμενο που πιθανώς να παράγεται.

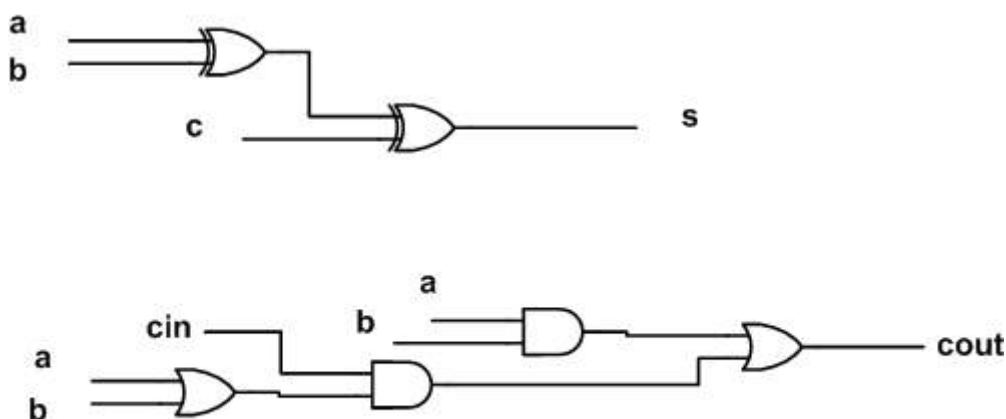
Ο πίνακας αληθείας του πλήρη αθροιστή φαίνεται παρακάτω:

a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Πίνακας [3.1] Πίνακας αληθείας του full adder

Από τον πίνακα βγαίνει ότι οι σχέσεις που ισχύουν για τα σήματα εξόδου είναι: $s = a \oplus b \oplus c$ και $cout = ab + acin + bcin$

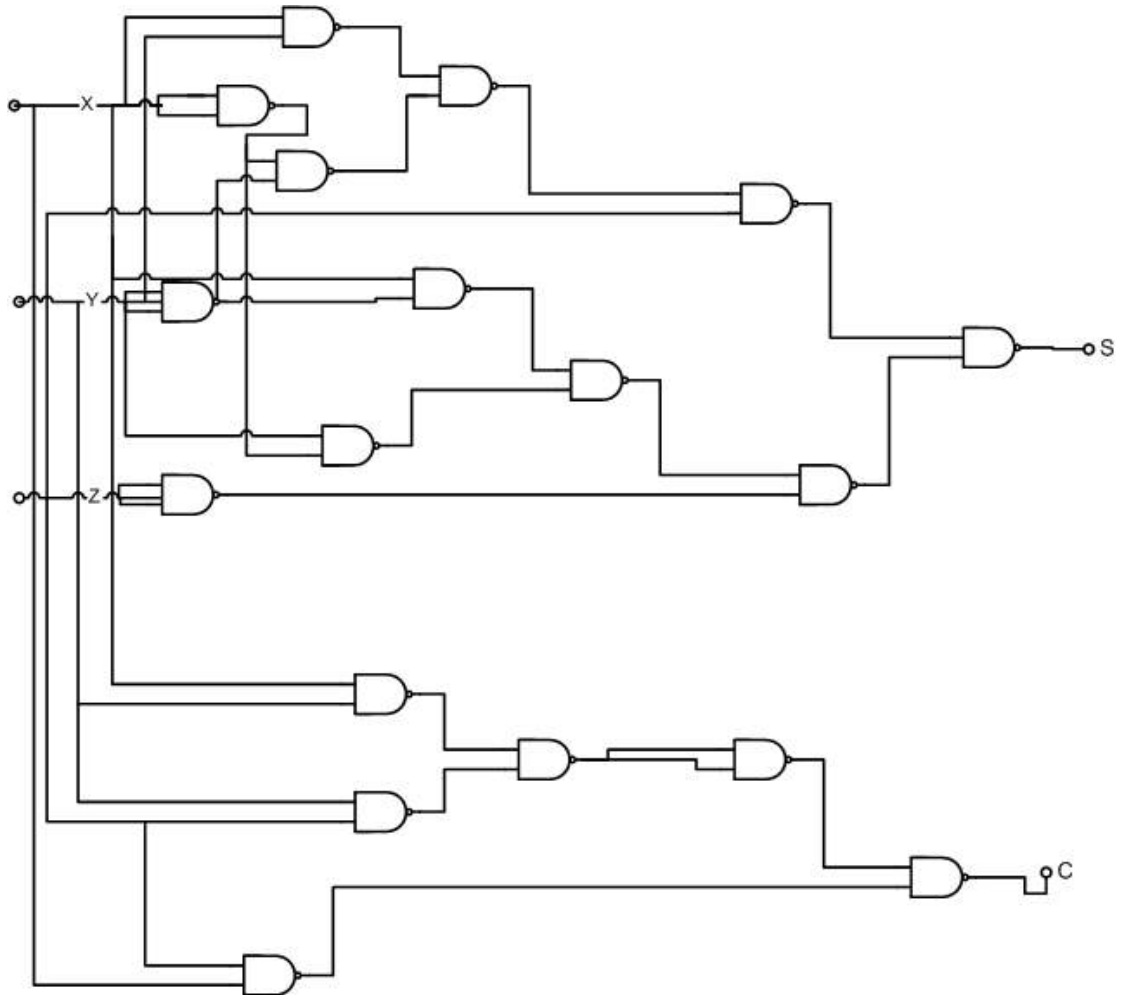
Στο παρακάτω σχήμα φαίνεται ένας πλήρης αθροιστής σε επίπεδο πυλών



Σχήμα 3.1 Πλήρης αθροιστής σε επίπεδο πυλών

3.2 FULL ADDER NAND

Σε αυτή την ενότητα θα μας απασχολήσει ο πλήρης αθροιστής με πύλες NAND και πύλες NOR. Η λειτουργία του είναι ακριβώς η ίδια με του απλού πλήρη αθροιστή με την διαφορά ότι για την υλοποίηση του χρησιμοποιούνται μόνο οι βασικές πύλες NAND και NOR.. Ένα σχήμα του πλήρη αθροιστή με πύλες NAND και πύλες NOR σε επίπεδο πυλών φαίνεται παρακάτω.



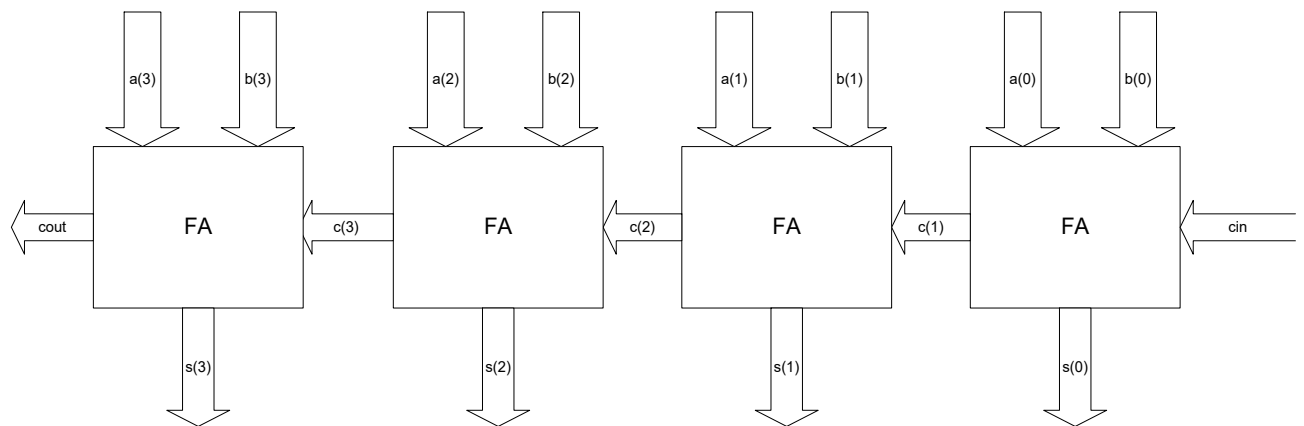
Σχήμα 3.2 Πλήρης αθροιστής με πύλες NAND NOR

3.3 ADDER_N

Ο παράλληλος αθροιστής με διάδοση κρατουμένου είναι ένα κύκλωμα που υλοποιεί την πράξη της πρόσθεσης μεταξύ δύο δυαδικών αριθμών των N bits.

Το επόμενο σχήμα μας δείχνει έναν παράλληλο αθροιστή για $N=4$, αποτελείται από N αθροιστές. Η έξοδος c_{out} της μιας βαθμίδας συνδέεται με την είσοδο c_{in} της επόμενης.

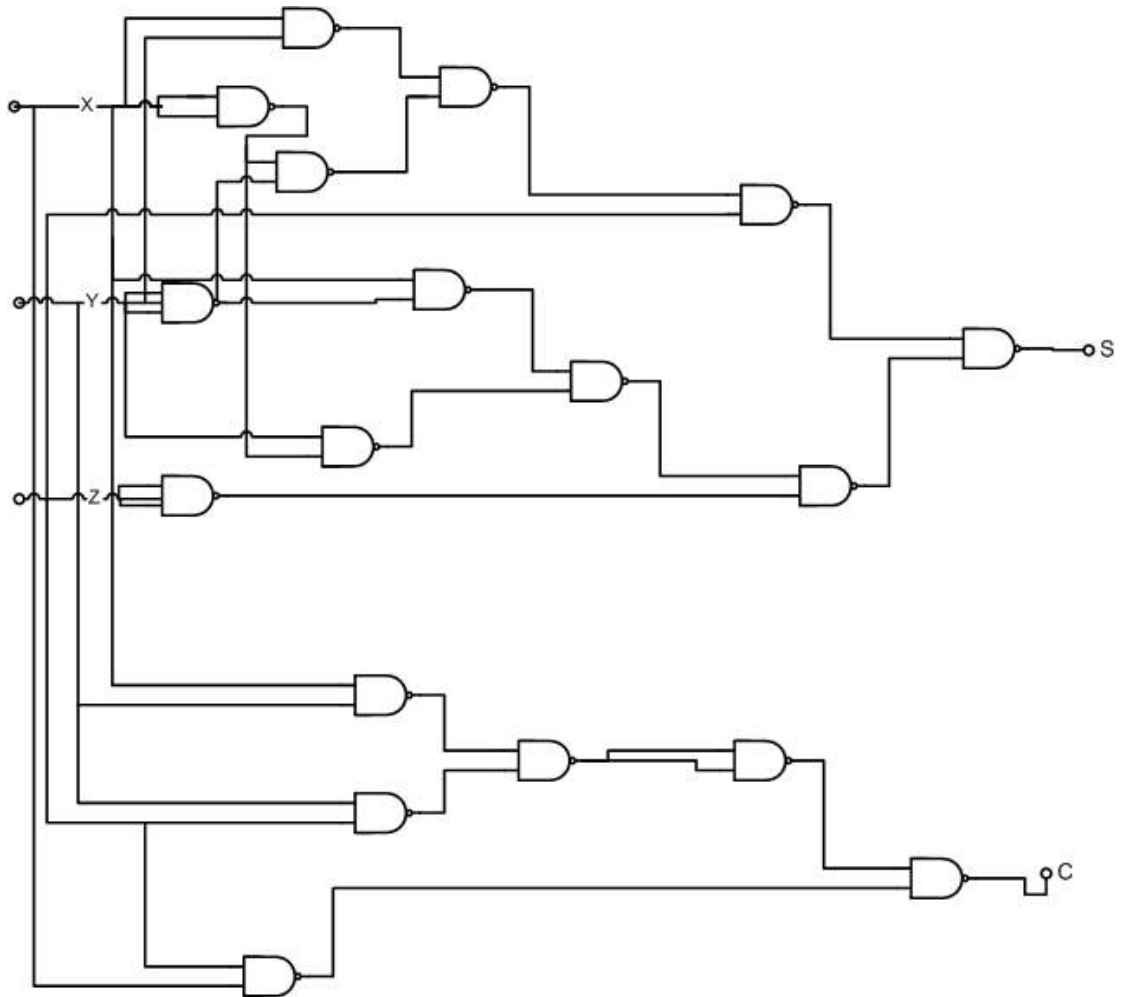
Το μειονέκτημα της συγκεκριμένης αρχιτεκτονικής είναι ότι αν και τα δεδομένα εισόδου είναι ταυτόχρονα διαθέσιμα σε όλες τις βαθμίδες, κάθε μια περιμένει να τελειώσει η προηγούμενη ώστε να είναι διαθέσιμο το c_{in} προκειμένου να κάνει την άθροιση για να δώσει το τελικό αποτέλεσμα.



Σχήμα 3.3 Παράλληλος αθροιστής των τεσσάρων (4) bits με διάδοση κρατουμένου

3.4 ADDER_N NAND

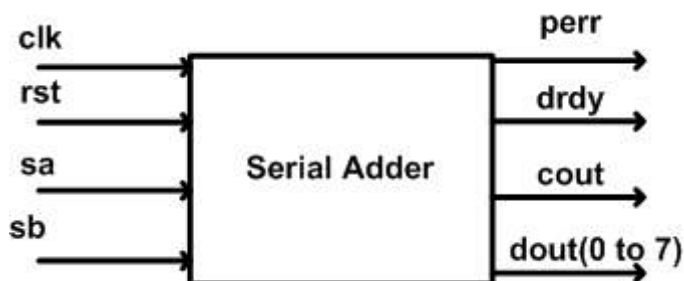
Ο παράλληλος αθροιστής με διάδοση κρατουμένου με χρήση πυλών NAND-NOR είναι το ίδιο κύκλωμα με το προηγούμενο με την διαφορά ότι για την υλοποίηση των επιμέρους πλήρων αθροιστών χρησιμοποιούνται αποκλειστικά πύλες NAND και πύλες NOR.



Σχήμα 3.4 Υλοποίηση πλήρους αθροιστή με πύλες NAND

Ο σειριακός αθροιστής είναι ένα κύκλωμα το οποίο έχει δύο εισόδους δεδομένων σειριακής μορφής, τις προσθέτει μεταξύ τους και δίνει το αποτέλεσμα σε παράλληλη μορφή.

Παρακάτω φαίνεται ένα σχήμα του κυκλώματος:



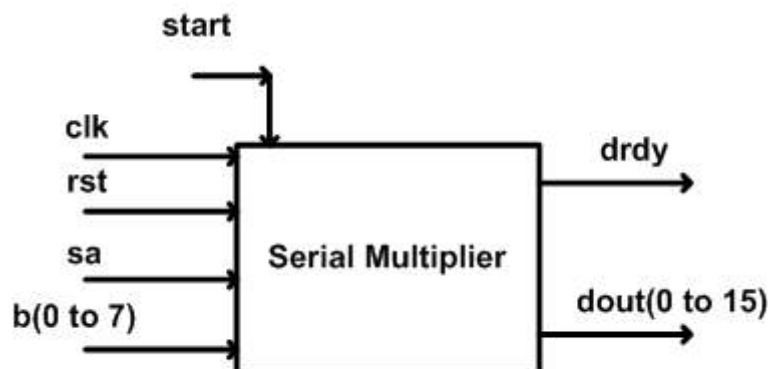
Σχήμα 3.6 Σειριακός αθροιστής δύο σειριακών εισόδων και παράλληλης εξόδου

Για την κατασκευή του κυκλώματος χρησιμοποιούμε έναν σειριακό αθροιστή δηλαδή έναν πλήρη αθροιστή με ανατροφοδότηση κρατούμενου. Το αποτέλεσμα εσωτερικά βρίσκεται σε σειριακή μορφή την οποία εμείς μετατρέπουμε σε παράλληλη.

Και σε αυτό το κύκλωμα χρησιμοποιούνται οι ίδιες αρχές ως προς την λήψη των σειριακών δεδομένων με την μόνη διαφορά ότι σε αυτή την περίπτωση η σειριακή είσοδος μεταφέρει τα δεδομένα ξεκινώντας από το LSB. Η έξοδος του κυκλώματος `drdy` και εδώ γίνεται '1' για να μας ειδοποιήσει πότε τα δεδομένα στην θύρα εξόδου είναι έγκυρα ενώ η έξοδος `perr` γίνεται '1' αν εντοπισθεί λάθος σε κάποιο από τα bit ισοτιμίας των δύο σειριακών εξόδων.

3.7 ΣΕΙΡΙΑΚΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ

Ο σειριακός πολλαπλασιαστής είναι ένα κύκλωμα με μια σειριακή είσοδο και μια παράλληλη και έξοδο σε παράλληλη μορφή. Το κύκλωμα που θα παρουσιαστεί εδώ έχει σχεδιασθεί για πολλαπλασιασμό αριθμών των 8 bit. Ένα διάγραμμα του πολλαπλασιαστή δίνεται παρακάτω:



Σχήμα 3.7 Πολλαπλασιαστής μιας σειριακής και μιας παράλληλης εισόδου, και παράλληλης εξόδου

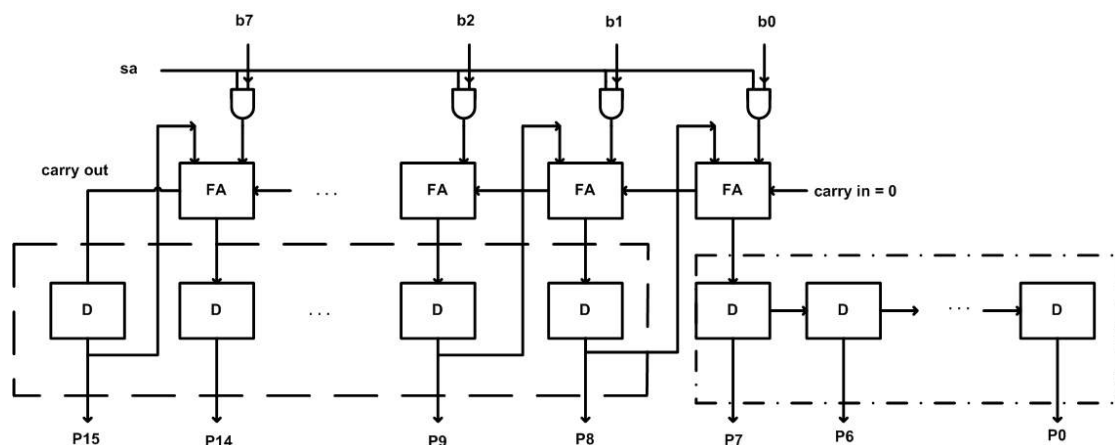
Για την υλοποίηση του κυκλώματος εργασθήκαμε ως κάτωθι:

Αρχικά περνάμε τα αποτελέσματα σε έναν παράλληλο καταχωρητή τα δεδομένα που έρχονται σε παράλληλη μορφή. Εν συνεχεία περνάμε τα μερικά γινόμενα $b \times sa_i$ σε έναν παράλληλο αθροιστή του οποίου το LSPP (Least Significant Partial Product) ολισθαίνει δεξιά κάθε φορά που έρχεται καινούργια είσοδος sa_i έτσι ώστε μετά από 7 περιόδους να περιέχει τους 7 LS όρους του γινομένου ($P_7 - P_0$). Τα υπόλοιπα μερικά γινόμενα του παράλληλου αθροιστή περνάνε σε παράλληλο καταχωρητή το περιεχόμενο του οποίου ανατροφοδοτείται στον παράλληλο αθροιστή του προηγούμενο επιπέδου έτσι ώστε μετά το πέρας 7 περιόδων, ο καταχωρητής να περιέχει τους 7 MS όρους του γινομένου ($P_{15} - P_8$).

Τα δύο επόμενα σχήματα θα μας βοηθήσουν να κατανοήσουμε ακόμη καλύτερα όσα περιγράφηκαν παραπάνω. Το πρώτο είναι ένας πίνακας που δείχνει με τι ισούται ακριβώς ο κάθε όρος του γινομένου. Το δεύτερο είναι κυκλωματικό διάγραμμα του πολλαπλασιαστή.

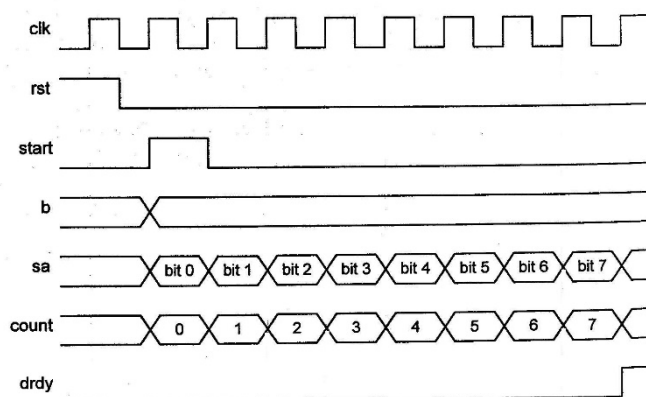
							a ₇	a ₆	a ₅	a ₄	a ₃	a ₂	a ₁	a ₀	Πολ/στέος
							b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	Πολ/στής
							a ₇ b ₀	a ₆ b ₀	a ₅ b ₀	a ₄ b ₀	a ₃ b ₀	a ₂ b ₀	a ₁ b ₀	a ₀ b ₀	
						a ₇ b ₁	a ₆ b ₁	a ₅ b ₁	a ₄ b ₁	a ₃ b ₁	a ₂ b ₁	a ₁ b ₁	a ₀ b ₁		
					a ₇ b ₂	a ₆ b ₂	a ₅ b ₂	a ₄ b ₂	a ₃ b ₂	a ₂ b ₂	a ₁ b ₂	a ₀ b ₂			
			a ₇ b ₃	a ₆ b ₃	a ₅ b ₃	a ₄ b ₃	a ₃ b ₃	a ₂ b ₃	a ₁ b ₃	a ₀ b ₃					
		a ₇ b ₄	a ₆ b ₄	a ₅ b ₄	a ₄ b ₄	a ₃ b ₄	a ₂ b ₄	a ₁ b ₄	a ₀ b ₄						
	a ₇ b ₅	a ₆ b ₅	a ₅ b ₅	a ₄ b ₅	a ₃ b ₅	a ₂ b ₅	a ₁ b ₅	a ₀ b ₅							
a ₇ b ₆	a ₆ b ₆	a ₅ b ₆	a ₄ b ₆	a ₃ b ₆	a ₂ b ₆	a ₁ b ₆	a ₀ b ₆								
a ₇ b ₇	a ₆ b ₇	a ₅ b ₇	a ₄ b ₇	a ₃ b ₇	a ₂ b ₇	a ₁ b ₇	a ₀ b ₇								
p ₁₄	p ₁₃	p ₁₂	p ₁₁	p ₁₀	p ₉	p ₈	p ₇	p ₆	p ₅	p ₄	p ₃	p ₂	p ₁	p ₀	Γινόμενο

Πίνακας [3.2] Μερικά γινόμενα πολλαπλασιαστή



Σχήμα 3.8 Κυκλωματικό διάγραμμα πολλαπλασιαστή

Οι χρονισμοί που θα χρησιμοποιήσουμε σε αυτό το κύκλωμα είναι : Τα δεδομένα που έρχονται σε σειριακή μορφή ξεκινάνε από το LSB την στιγμή που ο μετρητής έχει τιμή '0' και το αποτέλεσμα είναι έτοιμο μετά και το τέλος της 7^{ης} περιόδου του ρολογιού οπότε και η έξοδος drdy γίνεται '1'. Η όλη διαδικασία αρχίζει όταν η είσοδος start γίνει '1'. Παρακάτω δίνεται ένα ενδεικτικό διάγραμμα χρονισμού.



Σχήμα 3.9 Διάγραμμα χρονισμού πολλαπλασιαστή

3.8 ΓΕΝΙΚΕΣ ΑΡΧΕΣ ΠΑΡΑΛΛΗΛΩΝ ΠΟΛΛΑΠΛΑΣΙΑΣΤΩΝ

Όλοι οι παράλληλοι πολλαπλασιαστές βασίζονται στην παρατήρηση ότι τα μερικά γινόμενα μπορούν να υπολογιστούν ανεξάρτητα. Εάν έχουμε δύο απρόσημους δυαδικούς ακέραιους αριθμούς Α και Β που δίνονται από τις σχέσεις $A = \sum_{i=0}^{m-1} a_i \cdot 2^i$

και $B = \sum_{j=0}^{n-1} b_j \cdot 2^j$. Το γινόμενο τους θα δίνεται από την σχέση

$$P = A \times B = \sum_{i=0}^{m-1} a_i \cdot 2^i \cdot \sum_{j=0}^{n-1} b_j \cdot 2^j = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (a_i b_j) \cdot 2^{i+j} = \sum_{k=0}^{m+n-1} p_k \cdot 2^k$$

Η τελευταία σχέση γίνεται πιο κατανοητή μέσα από τον επόμενο πίνακα

				a ₃	a ₂	a ₁	a ₀	Πολ/στέος
				b ₃	b ₂	b ₁	b ₀	Πολ/στής
				a ₃ b ₀	a ₂ b ₀	a ₁ b ₀	A ₀ b ₀	
			a ₃ b ₁	a ₂ b ₁	a ₁ b ₁	a ₀ b ₁		
		a ₃ b ₂	a ₂ b ₂	a ₁ b ₂	a ₀ b ₂			
	a ₃ b ₃	a ₂ b ₃	a ₁ b ₃	a ₀ b ₃				
p ₇	p ₆	p ₅	p ₄	p ₃	p ₂	p ₁	p ₀	Γινόμενο

Πίνακας [3.3] Μερικά γινόμενα πολλαπλασιαστή 4x4

Ο παραπάνω πίνακας δείχνει πως ομαδοποιούνται τα μερικά γινόμενα για να προκύψει το γινόμενο των δύο αριθμών.

Το κοινό σημείο όλων των πολλαπλασιαστών είναι ότι προσπαθούν να δημιουργήσουν και να προσθέσουν κατάλληλα τα μερικά γινόμενα έτσι ώστε να πετύχουν το επιθυμητό αποτέλεσμα.

3.9 Ο ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ BOOTH

Η παράσταση ενός αριθμού X σε συμπλήρωμα ως προς 2 δίνεται από την παρακάτω σχέση:

$$X = -x_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot x_i$$

Ο αριθμός X μπορεί να γραφεί ισοδύναμα $X = 2^n X - X$

$2X =$	$-x_{n-1}$	x_{n-2}	x_{n-3}	$\dots$	x_0	0
$-X =$	0	$-x_{n-1}$	x_{n-2}	$\dots$	x_1	x_0
	Z_{n-1}	Z_{n-2}	$\dots$	Z_1	Z_0	

όπου

$$Z_0 = 0 - x_0$$

$$Z_1 = x_0 - x_1$$

$$Z_2 = x_1 - x_2$$

.

.

.

$$Z_{n-2} = x_{n-3} - x_{n-2}$$

$$Z_{n-1} = x_{n-2} - x_{n-1}$$

Όπως είναι φανερό ισχύει πάντα η σχέση $z_i = x_{i-1} - x_i$. Επομένως, ο αριθμός X δίνεται από την σχέση $X = \sum_{i=0}^{n-1} z_i \cdot 2^i$.

Έστω τώρα ότι έχουμε το γινόμενο $P = X * Y$ όπου $Y = y_{n-1} y_{n-2} \dots y_1 y_0$ και $X = x_{n-1} x_{n-2} \dots x_1 x_0$. Για το γινόμενο P ισχύει, με βάση τον αλγόριθμο του Booth, η

$$\text{σχέση } P = X * Y = \sum_{i=0}^{n-1} (y_{i-1} - y_i) \cdot X \cdot 2^i$$

Από την παραπάνω σχέση αντιλαμβανόμαστε ότι σε κάθε βήμα i , το X πολλαπλασιάζεται με ένα από τα στοιχεία του συνόλου $\{-1, 0, 1\}$ σύμφωνα με το αποτέλεσμα της αφαίρεσης των διαδοχικών ψηφίων του πολλαπλασιαστή Y . Στην ουσία με την κωδικοποίηση Booth ελέγχουμε τις σειρές από μονάδες που παρουσιάζονται μέσα σε κάθε δυαδικό αριθμό.

Y_i	Y_{i-1}	Κωδικοποιημένα Ψηφία ($Y_{i-1} - Y_i$)	
0	0	0	Δεν υπάρχει σειρά από 1
0	1	1	Τέλος σειράς από 1
1	0	-1	Αρχή σειράς από 1
1	1	0	Μέση σειράς από 1

Πίνακας [3.4] Κωδικοποίηση Ψηφίων

Με λίγα λόγια ο πολλαπλασιασμός γίνεται ως εξής:

- Αν $Y_i Y_{i-1} = 00$ ή $Y_i Y_{i-1} = 11$ τότε ολίσθησε το τρέχον άθροισμα των μερικών γινομένων κατά ένα bit προς τα αριστερά
- Αν $Y_i Y_{i-1} = 01$ τότε πρόσθεσε τον πολλαπλασιαστή X στο τρέχον άθροισμα των ενδιάμεσων γινομένων και ολίσθησε το αποτέλεσμα κατά μια θέση προς τα αριστερά
- Αν $Y_i Y_{i-1} = 10$ τότε αφάίρεσε τον πολλαπλασιαστή X από τρέχον άθροισμα των ενδιάμεσων γινομένων και ολίσθησε το αποτέλεσμα κατά μια θέση προς τα αριστερά

Το βασικό πλεονέκτημα ενός κυκλώματος κωδικοποίησης Booth είναι το γεγονός ότι η κωδικοποίηση θετικών και αρνητικών αριθμών είναι η ίδια. Επίσης, καθώς το κάθε ψηφίο του κωδικοποιημένου αριθμού είναι συνάρτηση μόνο των δύο bit $Y_i Y_{i-1}$, μπορεί να παραχθεί αμέσως ανεξάρτητα από τα προηγούμενα μερικά γινόμενα. Το σημαντικότερο όμως στοιχείο της κωδικοποίησης Booth είναι ότι μπορούμε με την μετατροπή του δυαδικού αριθμού να μειώσουμε τον αριθμό των προσθέσεων που απαιτούνται για τον πολλαπλασιασμό. Καθώς υπεύθυνες για τις προσθέσεις είναι οι μονάδες, και κατά την κωδικοποίηση μονάδα έχουμε όταν αρχίζει ή τελειώνει μια σειρά από 1, βλέπουμε ότι η κωδικοποίηση επιτρέπει την μετατροπή δυαδικών αριθμών με μεγάλο αριθμό μονάδων σε αριθμούς με λίγα μη μηδενικά ψηφία .

Κυκλωματική εφαρμογή

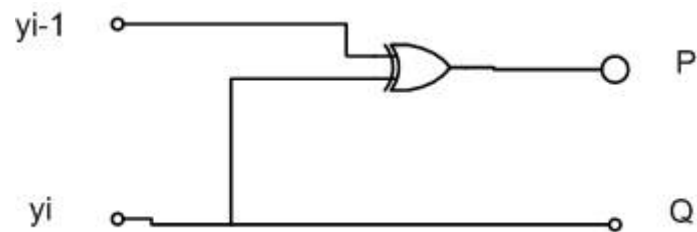
Ο πολλαπλασιασμός με τον αλγόριθμο του Booth θα γίνει με την χρήση κυττάρων που με την βοήθεια σημάτων ελέγχου θα προσθέτουν, θα αφαιρούν ή απλώς θα ολισθαίνουν το μερικό γινόμενο. Τα σήματα που θα χρησιμοποιήσουμε θα είναι τα P και Q η λειτουργία των οποίων εξηγείται στο παρακάτω πίνακα

$Y_i \ Y_{i-1}$	Κωδικοποιημένα ψηφία $Z_i = Y_{i-1} - Y_i$	P	Q	
0 0	0	0	0	Πρόσθεσε 0
0 1	1	1	0	Πρόσθεσε X
1 0	-1	1	1	Αφαίρεσε X
1 1	0	0	1	Αφαίρεσε 0

Πίνακας [3.5] Λειτουργία P και Q

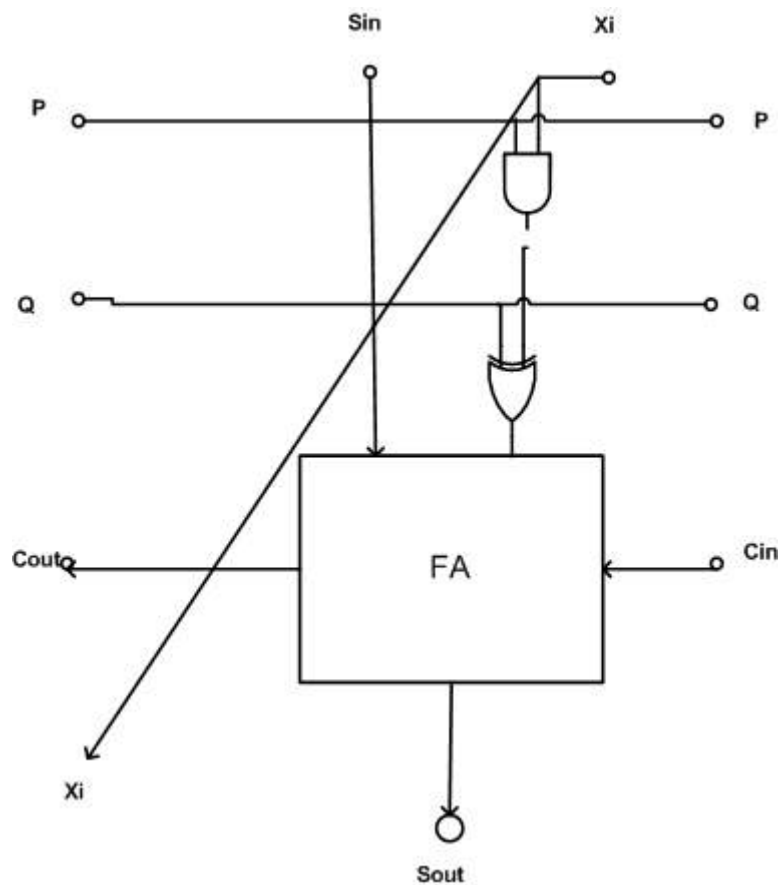
Παρατηρούμε ότι ισχύουν οι σχέσεις $P = Y_i \oplus Y_{i-1}$ και $Q = Y_i$

Το κύτταρο ελέγχου που θα υλοποιεί τα δύο αυτά σήματα φαίνεται παρακάτω



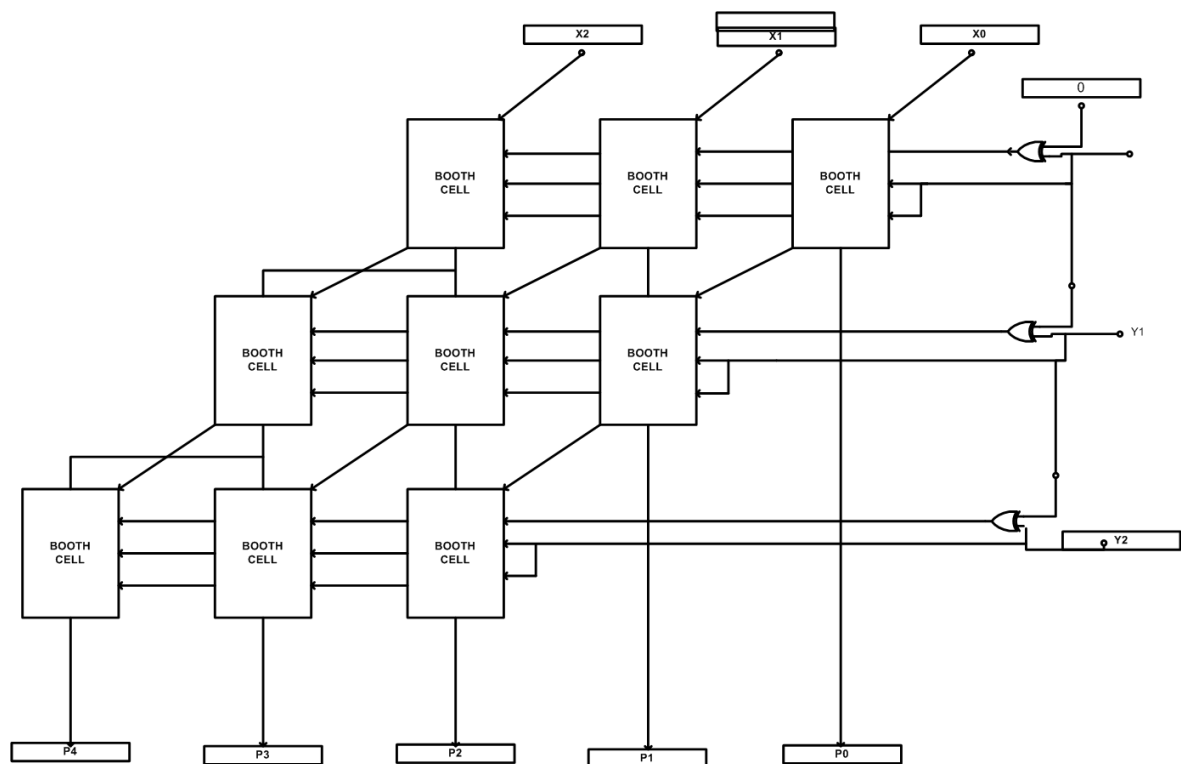
Σχήμα 3.10 Κύτταρο ελέγχου Booth

Συνολικά το κύτταρο του πολλαπλασιαστή Booth μαζί με τα σήματα ελέγχου θα είναι



Σχήμα 3.11 Κύτταρο πολλαπλασιαστή Booth

Στο παρακάτω σχήμα φαίνεται ένας πολλαπλασιαστής Booth 3x3



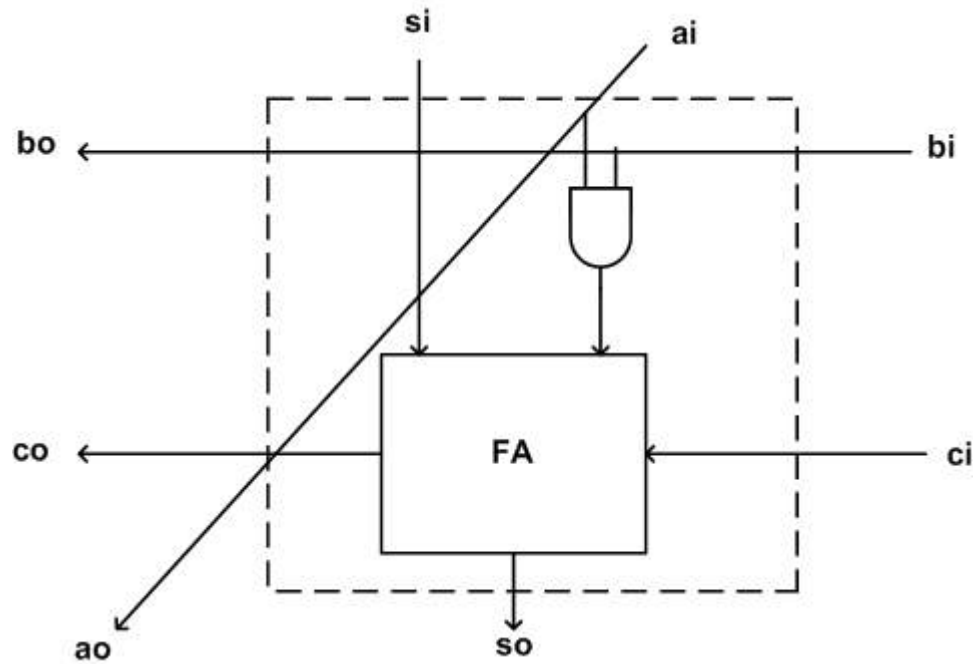
Σχήμα 3.12 Πολλαπλασιαστής Booth 3x3

Παρατηρούμε ότι το σήμα Q που βγαίνει από τα κύτταρα ελέγχου εισέρχεται και στην είσοδο κρατούμενου των κυττάρων του πολλαπλασιαστή. Στην περίπτωση της αφαίρεσης το σήμα είναι 1 όπως και το σήμα P. Εφόσον το P είναι 1, η τιμή των χ_i περνάει από την πύλη AND του κυττάρου του πολλαπλασιαστή. Η πύλη XOR αντιστρέφει τα χ_i ($Q = 1$) δημιουργώντας έτσι το συμπλήρωμα του X ως προς 1 ενώ με την πρόσθεση του Q στο κρατούμενο εισόδου του πλήρη αθροιστή προκύπτει τελικά το συμπλήρωμα του X ως προς 2.

3.10 ΠΑΡΑΛΛΗΛΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΔΙΑΔΟΣΗ ΚΡΑΤΟΥΜΕΝΟΥ

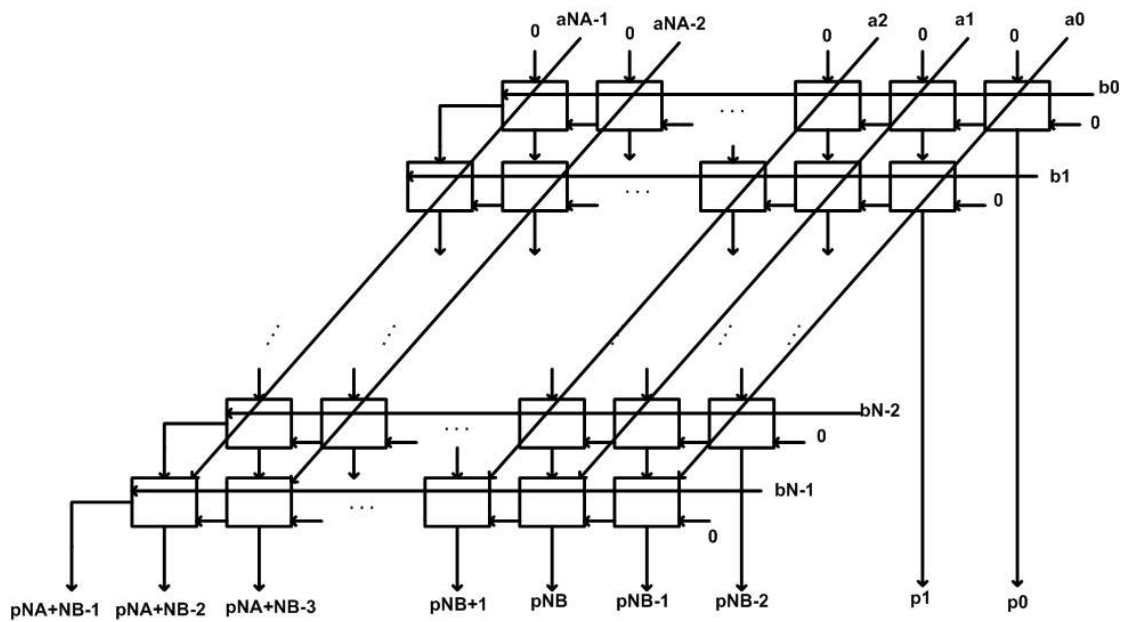
Ο παράλληλος πολλαπλασιαστής με αθροιστή διάδοσης αποτελείται από ένα δίκτυο παραγωγής μερικών γινομένων $a_i b_j$ το οποίο κατασκευάζεται από λογικές πύλες AND και έναν αθροιστή.

Στο παρακάτω σχήμα φαίνεται το κύτταρο του πολλαπλασιαστή.



Σχήμα 3.13 Η βασική δομική μονάδα του πολλαπλασιαστή με διάδοση κρατουμένου

Στο επόμενο σχήμα φαίνεται η δομή ενός παράλληλου πολλαπλασιαστή δύο αριθμών a και b των NA και NB bit αντίστοιχα.



Σχήμα 3.14 Παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου

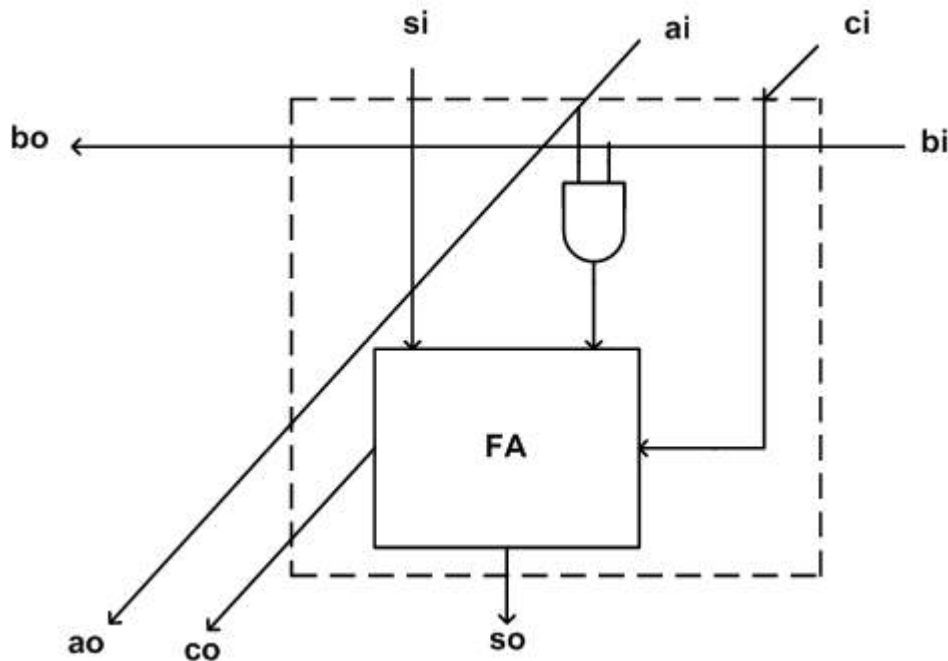
Το κάθε κουτάκι του παραπάνω σχήματος αναπαριστά το κύτταρο του πολλαπλασιαστή, στο οποίο παράγεται το μερικό γινόμενο $a_i b_j$ το οποίο προστίθεται με το αντίστοιχης τάξης ψηφίο του μερικού αθροίσματος (s_i) και το κρατούμενο το οποίο διαδίδεται από το προηγούμενο κύτταρο (c_i).

Η πορεία του κρατουμένου που διαδίδεται στο διπλανό κύτταρο της ίδιας σειράς έχει δώσει το όνομα στον παράλληλο πολλαπλασιαστή.

3.11 ΠΑΡΑΛΛΗΛΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΣΩΣΙΜΟ ΚΡΑΤΟΥΜΕΝΟΥ

Ο παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου ακολουθεί τις γενικές αρχές που διέπουν όλους τους παράλληλους πολλαπλασιαστές. Αποτελείται και αυτός από ένα δίκτυο παραγωγής μερικών γινομένων το οποίο είναι κατάλληλα διατεταγμένο ώστε να προκύπτουν σωστά οι όροι του γινομένου.

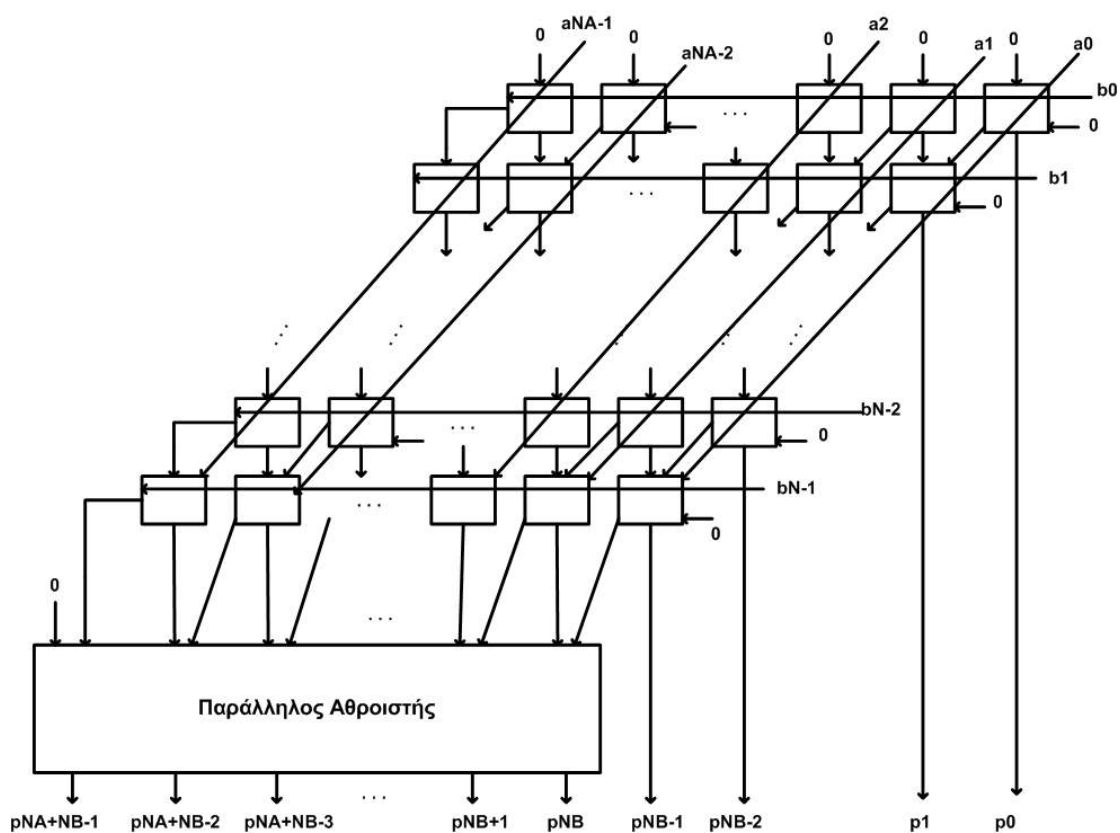
Στο παρακάτω σχήμα μπορούμε να δούμε την βασική μονάδα του πολλαπλασιαστή:



Σχήμα 3.15 Δομική μονάδα του πολλαπλασιαστή με σώσιμο κρατουμένου

Όπως μπορούμε να παρατηρήσουμε το κύτταρο του πολλαπλασιαστή αποτελείται από μια πύλη AND που παράγει το μερικό γινόμενο $a_i b_i$ το οποίο προστίθεται ,μέσω του πλήρη αθροιστή, με την έξοδο του κυττάρου της προηγούμενης βαθμίδας. Η ιδιαιτερότητα του κυττάρου και κατ' επέκταση όλου του πολλαπλασιαστή είναι ότι το κρατούμενο εξόδου του κυττάρου περνάει ως κρατούμενο εισόδου στο κύτταρο της επόμενης βαθμίδας.

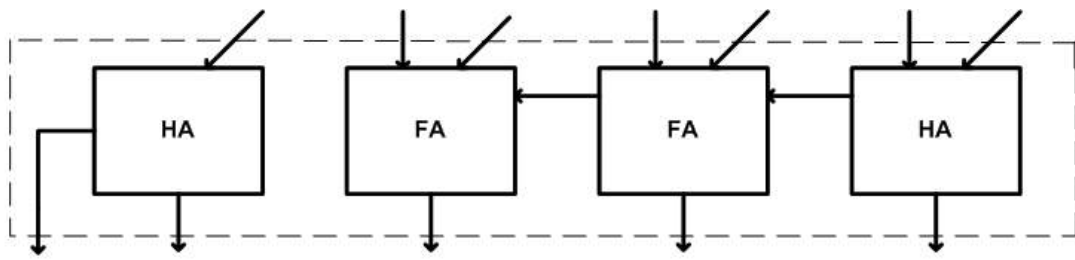
Στο επόμενο σχήμα παρουσιάζεται ένας παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου με εισόδους δύο αριθμούς a και b των NA και NB bit αντίστοιχα.



Σχήμα 3.16 Παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου

Όπως μπορούμε να παρατηρήσουμε στην τελευταία βαθμίδα του πολλαπλασιαστή έχουμε τα NA πιο σημαντικά bit του γινομένου σε μορφή αθροίσματος κρατουμένου. Προκειμένου να τα μεταφέρουμε σε κανονική μορφή χρησιμοποιούμε έναν παράλληλο αθροιστή στον οποίο δεν εισέρχονται τα σήματα a_i και b_i γιατί δεν παίρνουν μέρος στην πρόσθεση των κρατουμένων και των μερικών γινομένων. Για αυτήν την δουλειά μπορούμε να χρησιμοποιήσουμε είτε έναν αθροιστή με διάδοση κρατουμένου είτε έναν αθροιστή πρόβλεψης κρατουμένου.

Στο παρακάτω σχήμα μπορούμε να δούμε μια ενδεικτική μορφή του τελικού αθροιστή



Σχήμα 3.17 Τελικός αθροιστής

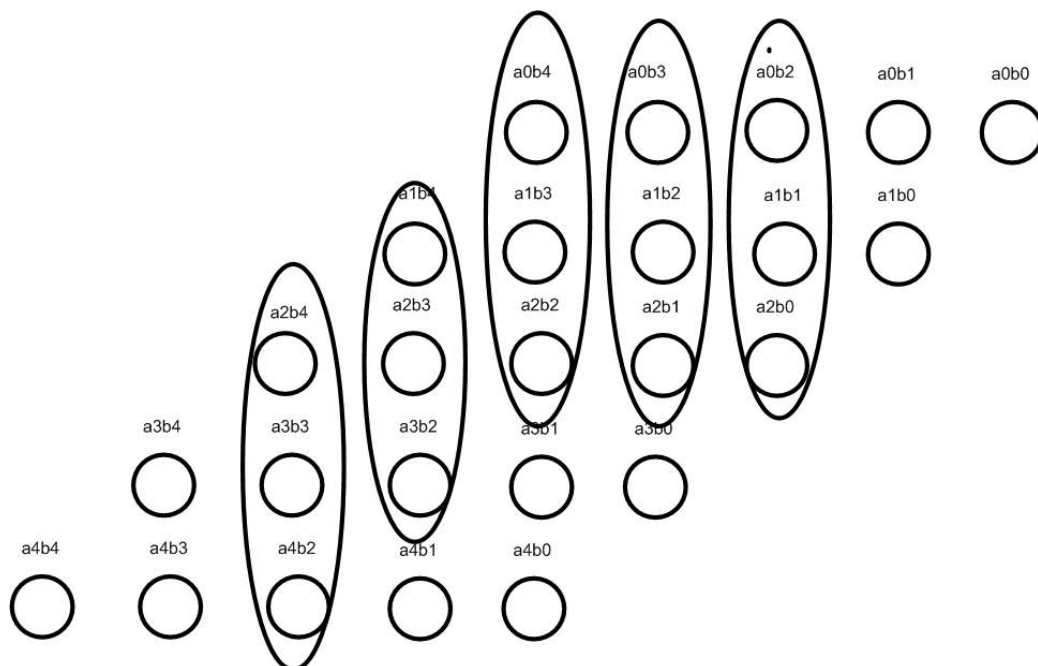
3.12 ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΔΕΝΔΡΟΥ WALLACE

Η φιλοσοφία των δένδρων Wallace στηρίζεται στην ιδιότητα των πλήρων αθροιστών να συμπιέζουν τα τρία bits σε δύο. Έτσι εάν έχουμε π.χ να προσθέσουμε δύο αριθμούς των n bits, τα ομαδοποιούμε σε λέξεις των τριών bits και χρησιμοποιούμε μια σειρά από $n/3$ αθροιστές με σώσιμο κρατουμένου. Μέχρι αυτό το σημείο έχουμε συμπίεση τα n bits σε $n/3$ bits. Στο επόμενο επίπεδο παραθέτουμε μια σειρά από $2n/9$ ($2n/3 * 1/3$) αθροιστές. Συνεχίζουμε αυτή την διαδικασία μέχρις ότου να έχουν προστεθεί όλα τα κρατούμενα των ενδιάμεσων αθροιστών. Στο τελευταίο στάδιο θα έχουν απομείνει προς πρόσθεση μόνο δύο αριθμοί που θα βρίσκονται σε μορφή «αθροίσματος – κρατουμένου». Για την τελική πρόσθεση και προκειμένου να έρθει το αποτέλεσμα σε κανονική μορφή έχουμε να διαλέξουμε μεταξύ ενός αθροιστή διάδοσης κρατουμένου και ενός αθροιστή πρόβλεψης κρατουμένου για μεγαλύτερη ακρίβεια.

Ο πολλαπλασιαστής δένδρου Wallace οφείλει την ταχύτητά του στο γεγονός ότι πουθενά μέσα στην δομή του δεν έχουμε διάδοση κρατουμένου, παρά μόνο στο τελικό στάδιο. Εύκολα παρατηρούμε ότι ένας πολλαπλασιαστής δένδρου Wallace έχει λιγότερα επίπεδα από έναν πολλαπλασιαστή με σώσιμο κρατουμένου. Αυτό σημαίνει μεγαλύτερη ταχύτητα και αυξημένη απόδοση.

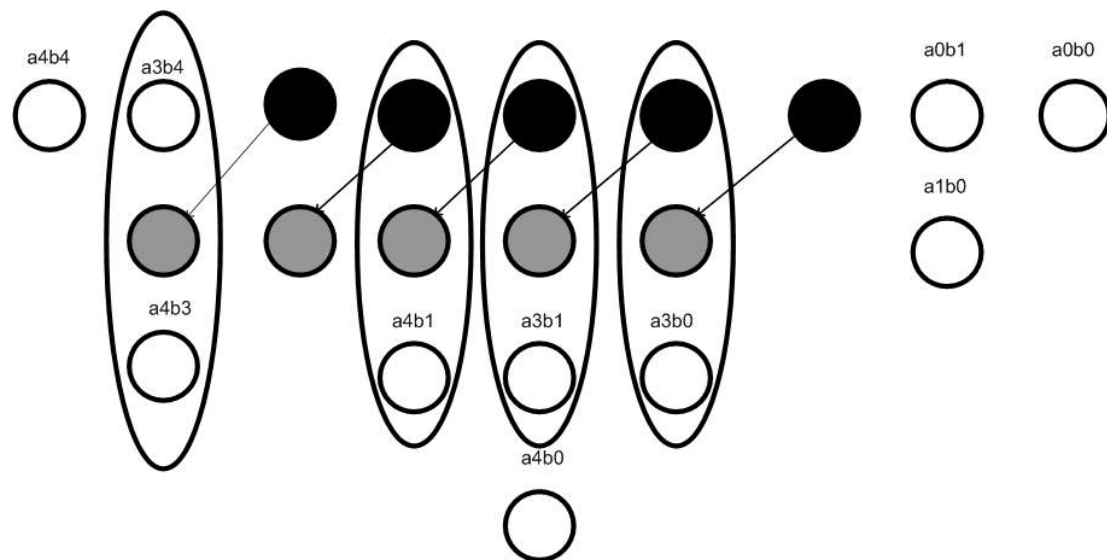
Το τίμημα που έχει αυτή η βελτίωση της απόδοσης είναι η χρήση περισσότερου υλικού καθώς χρειαζόμαστε έναν μεγαλύτερο αθροιστή στο τελικό στάδιο από τον αντίστοιχο στον πολλαπλασιαστή με σώσιμο κρατουμένου. Επίσης οι πολλαπλασιαστές δένδρων Wallace παρουσιάζουν μεγάλη πολυπλοκότητα στις εσωτερικές τους διασυνδέσεις γεγονός που δυσκολεύει αισθητά την παραμετροποίησή τους.

Στο σημείο αυτό θα παρουσιάσουμε τον αλγόριθμο κατασκευής ενός δένδρου Wallace. Στο παρακάτω σχήμα βλέπουμε τα μερικά γινόμενα δύο αριθμών των 5 bits στοιχισμένα έτσι ώστε η κατακόρυφη πρόσθεση τους να μας δίνει το γινόμενο τους. Στο σχήμα παρουσιάζεται και η ομαδοποίηση ανά τρία που γίνεται σε αυτά.



Σχήμα 3.18 Ομαδοποίηση των μερικών γινομένων δυο αριθμών των 5 bits

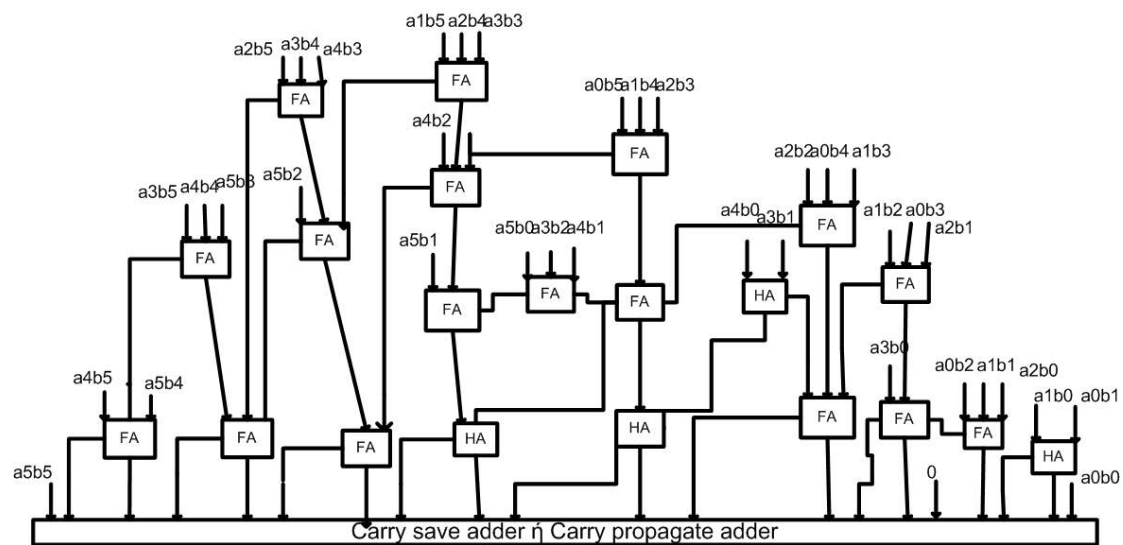
Ξεκινάμε από τα bits χαμηλότερης σημασίας και όπου συναντήσουμε τρία μερικά γινόμενα, τα ομαδοποιούμε. Αυτά τα μερικά γινόμενα που ομαδοποιήθηκαν αποτελούν τις εισόδους των πλήρων αθροιστών. Τώρα πλέον τα αντικαθιστούμε στον πίνακα με τις εξόδους του πλήρη αθροιστή, δηλαδή με το άθροισμα που τοποθετείται στην ίδια κατακόρυφο και το κρατούμενο το οποίο τοποθετείται στην επόμενη στήλη λόγω του μεγαλύτερου βάρους. Στο επόμενο σχήμα φαίνεται πως έχει διαμορφωθεί το σκηνικό στον πίνακα των μερικών γινομένων μετά την πρώτη ομαδοποίηση και την αντικατάσταση με τις εξόδους του πλήρη αθροιστή.



Σχήμα 3.19 Ο πίνακας των μερικών γινομένων μετά την αντικατάσταση και η νέα ομαδοποίηση

Η διαδικασία αυτή συνεχίζεται έως ότου γίνουν όλες οι δυνατές ομαδοποιήσεις, τόσο των μερικών γινομένων όσο και των αθροισμάτων και των κρατούμενων. Ένα πράγμα το οποίο μπορεί να προβληματίσει τον σχεδιαστή είναι το πώς θα κατανεμηθούν τα μερικά γινόμενα, τα μερικά αθροίσματα και τα κρατούμενα της ίδιας στήλης στους πλήρεις αθροιστές. Θα πρέπει να σημειώσουμε ότι η δομή του δένδρου Wallace στηρίζεται στην προσεταιριστικότητα της πρόσθεσης, οπότε δεν μας απασχολεί το πώς θα γίνει η κατανομή αλλά πρέπει να εξασφαλίσουμε την πρόσθεση όλων των παραγόντων που ανήκουν στην ίδια κατακόρυφο.

Στο παρακάτω σχήμα φαίνεται το πλήρες σχέδιο ενός πολλαπλασιαστή



Σχήμα 3.20 Πολλαπλασιαστής 6x6 σε δομή δένδρου Wallace

Εάν τώρα θέλουμε το κύκλωμα μας να είναι ακόμη πιο γρήγορο και αποδοτικό μπορούμε να συνδυάσουμε τον τροποποιημένο αλγόριθμο του Booth με μια δομή δένδρου Wallace.

Ας δούμε πρώτα ορισμένα πράγματα για τον τροποποιημένο αλγόριθμο του Booth (Modified Booth algorithm)

Η κωδικοποίηση ενός δυαδικού αριθμού Y δίνεται από την σχέση $Y = \sum_{i=0}^{n-1} z_i 2^i$

Η οποία μπορεί να αναλυθεί περαιτέρω:

$$Y = \sum_{i=0}^{n-1} z_i 2^i = \sum_{j=0}^{n/2-1} z_{2j} 2^{2j} + z_{2j+1} 2^{2j+1} = \sum_{j=0}^{n/2-1} (z_{2j} + 2z_{2j+1}) 2^{2j} = \sum_{j=0}^{n/2-1} w_j 4^j$$

όπου z_i είναι το ψηφίο του Y σε απλή κωδικοποίηση Booth ενώ το w_i είναι το ψηφίο κωδικοποιημένο σύμφωνα με τον τροποποιημένο αλγόριθμο Booth.

Στον τροποποιημένο αλγόριθμο Booth κωδικοποιούμε τριάδες από bits σε αντίθεση με τα ζευγάρια bits του απλού αλγορίθμου Booth. Ο παρακάτω πίνακας θα μας βοηθήσει να καταλάβουμε καλύτερα την διαδικασία κωδικοποίησης.

			Κωδικοποίηση Booth		Κωδικοποίηση τροποποιημένου Booth
Y_{i+1}	Y_i	Y_{i-1}	Z_{i+1}	Z_i	$W_i = 2Z_{i+1} + Z_i = y_i - 2y_{i+1} + y_{i-1}$
0	0	0	0	0	0
0	0	1	0	+1	+1
0	1	0	+1	-1	+1
0	1	1	+1	0	+2
1	0	0	-1	0	-2
1	0	1	-1	+1	-1
1	1	0	0	-1	-1
1	1	1	0	0	0

Πίνακας [3.6] Κωδικοποίησης του τροποποιημένου αλγορίθμου Booth

Το σύνολο αναφοράς σε αυτόν τον αλγόριθμο είναι $\{-2, -1, 0, +1, +2\}$. Κάθε ένα από τα κωδικοποιημένα ψηφία αντιστοιχεί σε μια διαφορετική λειτουργία. Αναλυτικά εάν έχουμε να πολλαπλασιάσουμε δύο αριθμούς A και B, και ο B κωδικοποιηθεί σύμφωνα με τον αλγόριθμο τότε:

Κωδικοποιημένα ψηφία	Λειτουργία
0	Πρόσθεσε το 0 στο μερικό γινόμενο
+1	Πρόσθεσε το (A) στο μερικό γινόμενο
+2	Πρόσθεσε το (2A) στο μερικό γινόμενο
-2	Αφαίρεσε το (2A) από το μερικό γινόμενο
-1	Αφαίρεσε το (A) από το μερικό γινόμενο

Πίνακας [3.7] Αντιστοίχιση λειτουργιών και κωδικοποιημένων ψηφίων

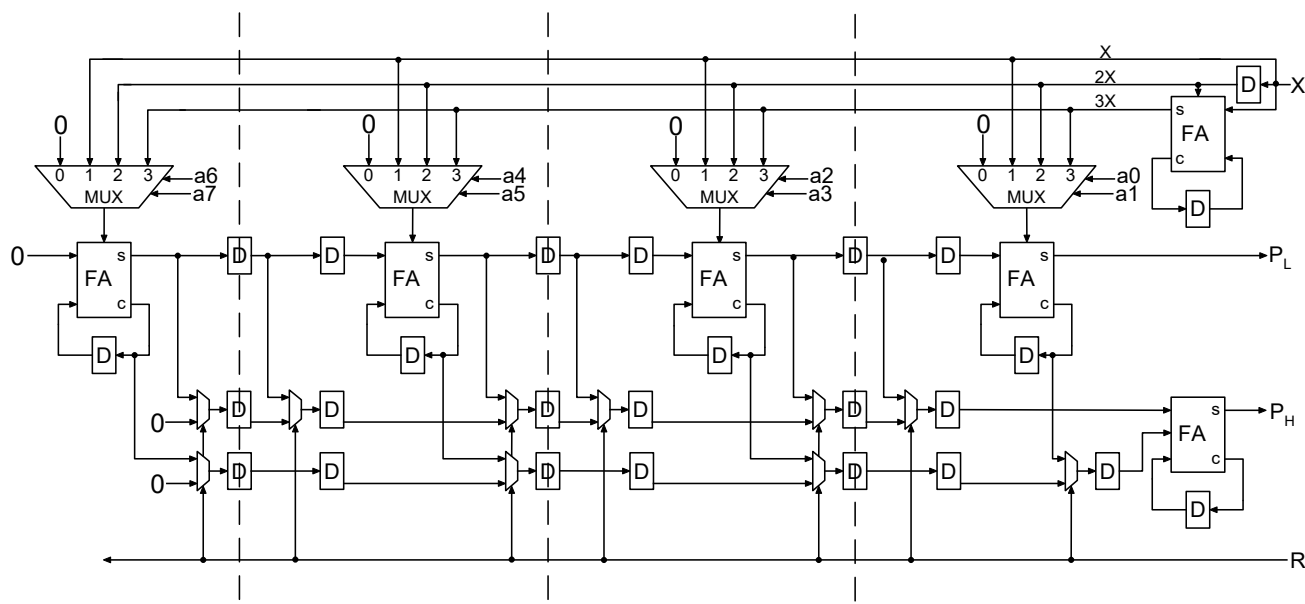
Τα δύο μεγάλα πλεονεκτήματα του τροποποιημένου αλγορίθμου Booth είναι ότι εφαρμόζεται το ίδιο ανεξάρτητα από το αν οι αριθμοί είναι θετικοί ή αρνητικοί και ότι έχουμε μείωση των μερικών γινομένων και άρα στον αριθμό των προσθέσεων που έχουμε να κάνουμε.

Το κύκλωμα επιτελεί, με λίγα λόγια, την εξής λειτουργία: Αρχικά χρησιμοποιείται ο τροποποιημένος αλγόριθμος του Booth για να παραχθούν τα μερικά γινόμενα τα οποία θα προστεθούν με την βοήθεια ενός δένδρου Wallace. Στον κώδικα που ακολουθεί, περιέχονται διαδικασίες που εκτελούν τις δυο παραπάνω λειτουργίες καθώς και άλλες όπως της στοίχισης των μερικών γινομένων, της παραγωγής του συμπληρώματος ως προς 2 μέσω ενός αθροιστή πρόβλεψης κρατουμένου και της τελικής πρόσθεσης που πρέπει να γίνει στο δένδρο Wallace μέσω και πάλι ενός αθροιστή πρόβλεψης κρατουμένου.

3.13 ΣΕΙΡΙΑΚΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΤΗΣ ΜΕ ΚΑΤΕΒΑΣΜΑ ΚΡΑΤΟΥΜΕΝΟΥ

Το πρόβλημα των σειριακών-παράλληλων πολλαπλασιαστών βρίσκεται στο γεγονός ότι απαιτούνται μια σειρά κύκλων για να αδειάσει ο πολλαπλασιαστής από τα μερικά γινόμενα και στην συνέχεια να εισαχθεί το επόμενο δεδομένο.

Για την επίλυση του παραπάνω προβλήματος έχει προταθεί η παρακάτω τοπολογία.



Σχήμα 3.21 Σειριακός Πολλαπλασιαστής Κατεβάσματος

Γνωρίζουμε ότι $A \cdot X = \sum_{j=0}^{n/2-1} (a_{2j} + 2a_{2j+1}) 2^{2j} X$ ο όρος στην παρένθεση μπορεί να

πάρει την τιμή 0,1,2,3 ο οποίος παράγεται από μια ολίσθηση και μια πρόσθεση σε ένα σειριακό αθροιστή. Ένας πολυπλέκτης 4:1 επιλέγει τον κατάλληλο όρο και τον οδηγεί στον κυρίως αθροιστή. Μόλις τελειώσουν τα bits της σειριακής εισόδου τότε ένα σήμα ελέγχου ενεργοποιείται και κατεβαίνουν τα επιμέρους αθροίσματα και κρατούμενα σε δύο καταχωρητές ολίσθησης. Στο τέλος των καταχωρητών υπάρχει ένας σειριακός αθροιστής ο οποίος μετατρέπει την μορφή κρατουμένου αθροίσματος σε κανονική μορφή.

Κεφάλαιο 4

Μετρήσεις

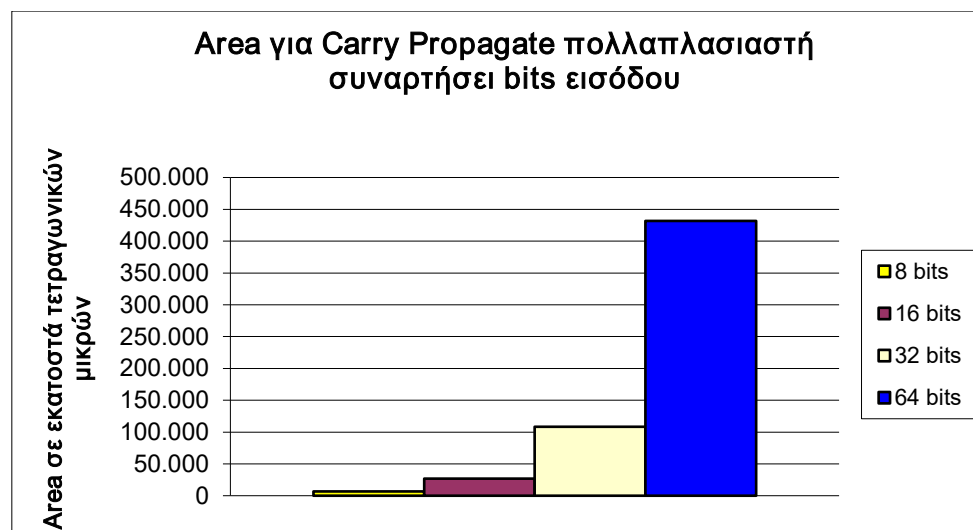
4.1 Παράλληλοι Πολλαπλασιαστές

4.1.1 Carry Propagate Multiplier

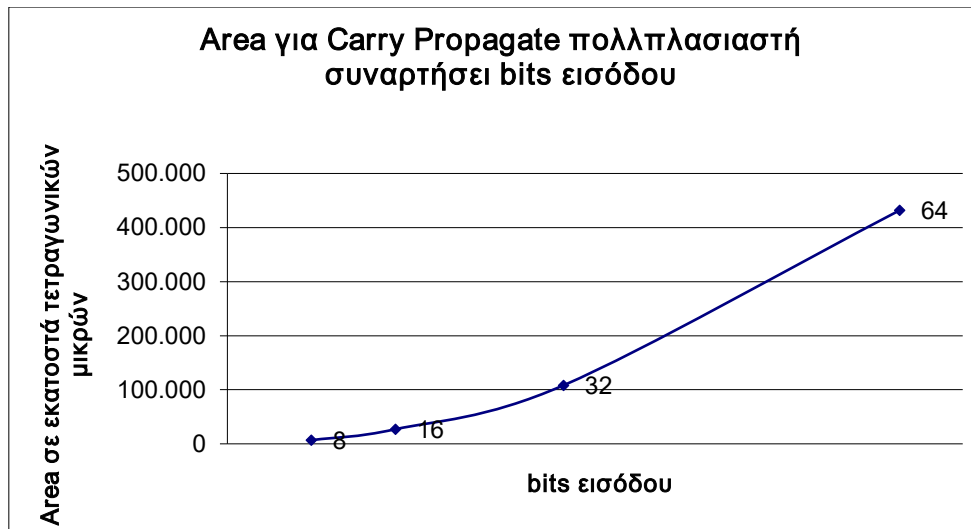
Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον πολλαπλασιαστή carry propagate για 8,16,32,64 bits εισόδου

Number of bits	Area (10^{-14} m^2)
8	6.773
16	27.067
32	108.247
64	431.906

Πίνακας [4.1] Area για carry propagate



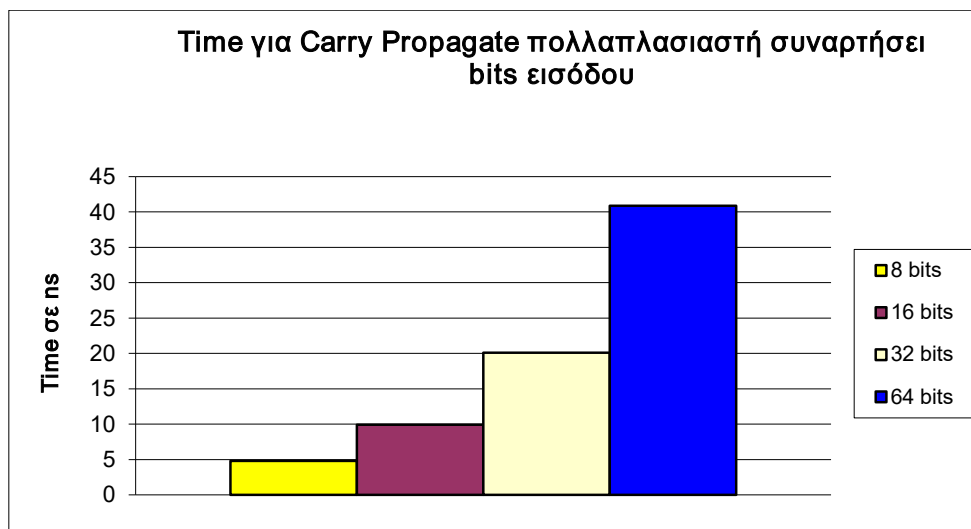
Σχήμα 4.1 Area για carry propagate



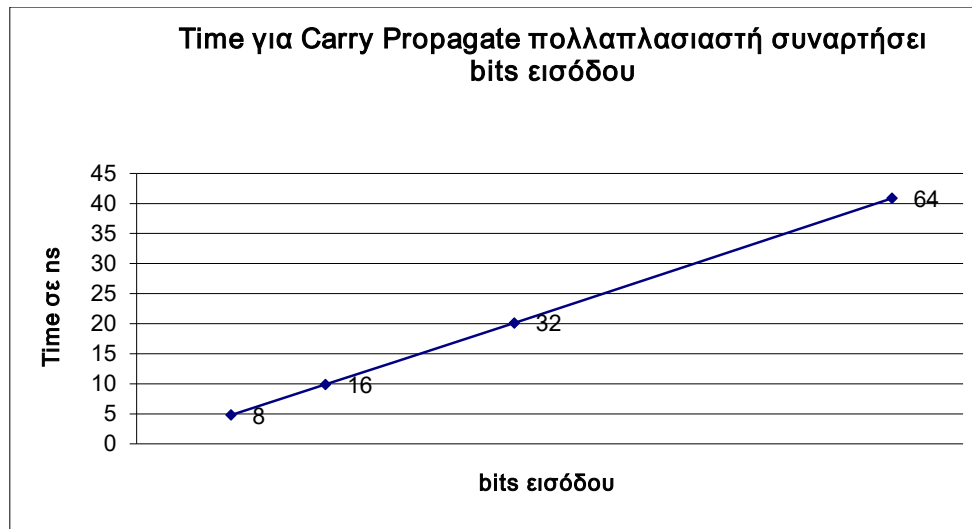
Σχήμα 4.2 Area για carry propagate

Number of bits	Time (nsec)
8	4,79
16	9,9
32	20,12
64	40,88

Πίνακας [4.2] Time για carry propagate



Σχήμα 4.3 Time για carry propagate



Σχήμα 4.4 Time για carry propagate

Number of bits	Power (Watt)
8	6.422e-03
16	3.938e-02
32	3.630e-01
64	3.152

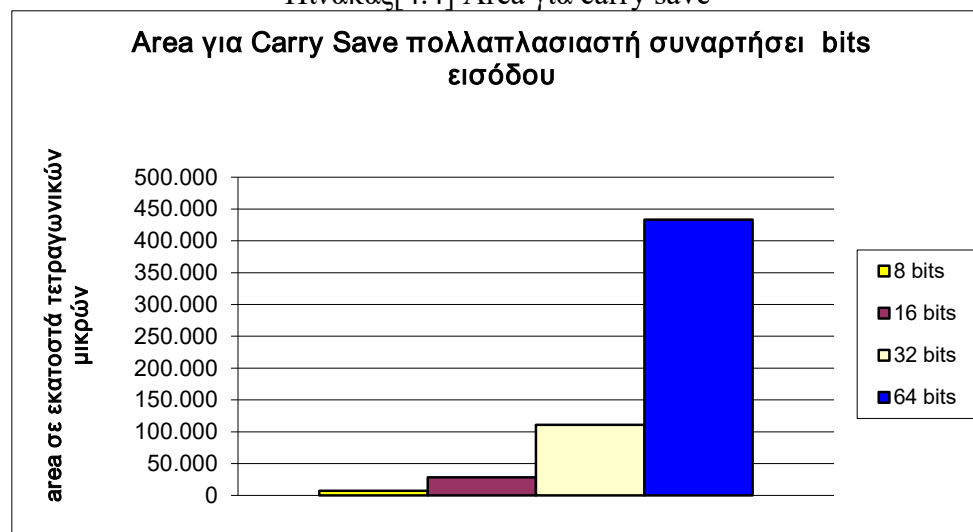
Πίνακας [4.3] Power για carry propagate

4.1.2 Carry Save Multiplier

Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον πολλαπλασιαστή carry save για 8,16,32,64 bits εισόδου

Number of bits	Area (10^{-14} m^2)
8	7.488
16	28.498
32	111.110
64	433.329

Πίνακας[4.4] Area για carry save



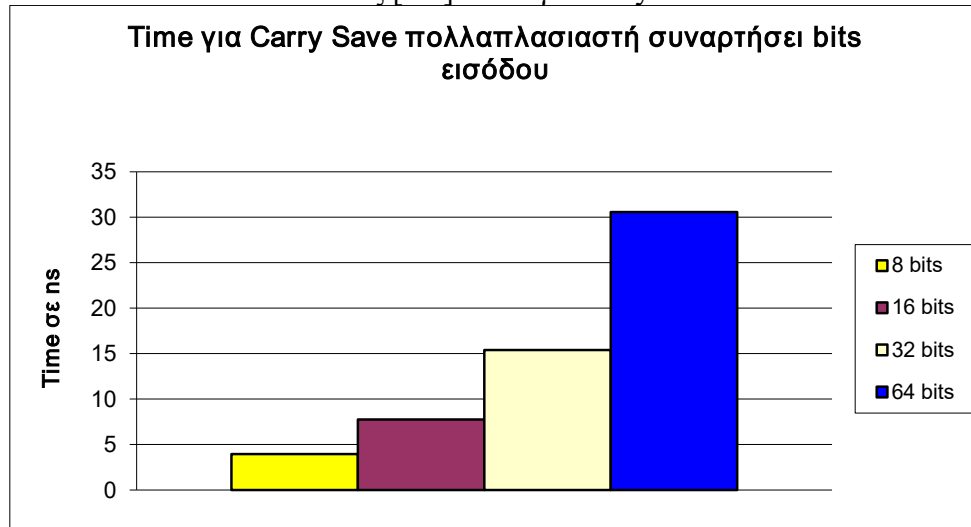
Σχήμα 4.5 Area για carry save



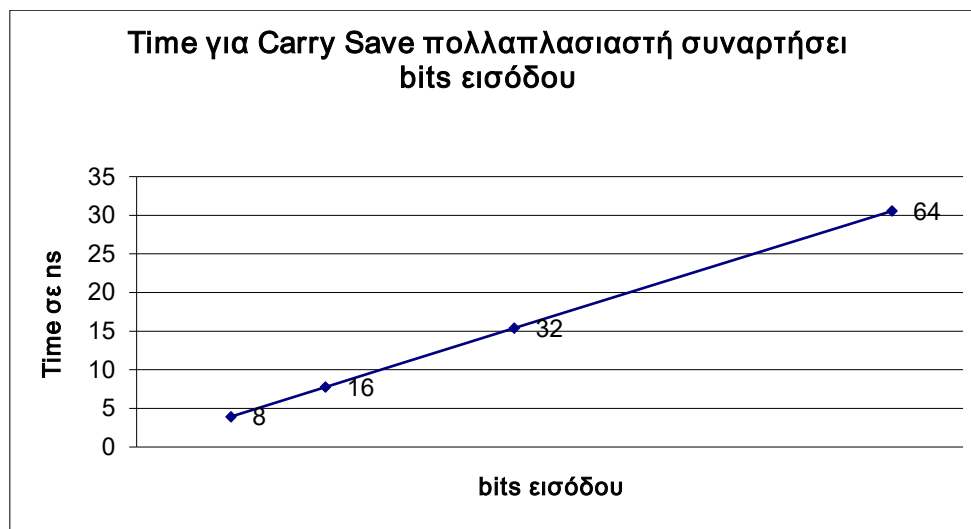
Σχήμα 4.6 Area για carry save

Number of bits	Time (nsec)
8	3,94
16	7,76
32	15,4
64	30,56

Πίνακας [4.5] Time για carry save



Σχήμα 4.7 Time για carry save



Σχήμα 4.8 Time για carry save

Number of bits	Power (Watt)
8	5.386e-03
16	3.170e-02
32	1.536e-01
64	2.328

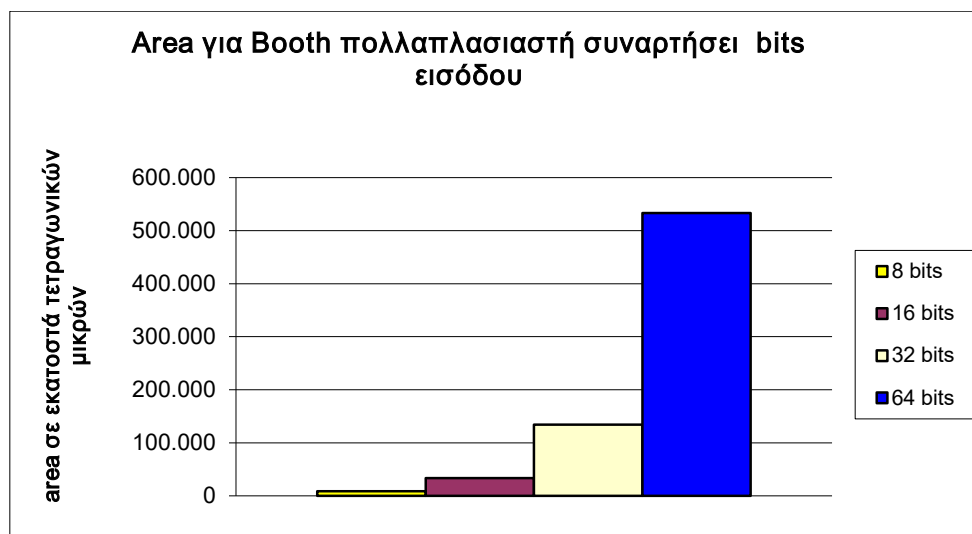
Πίνακας [4.6] Power για carry save

4.1.3 Booth Multiplier

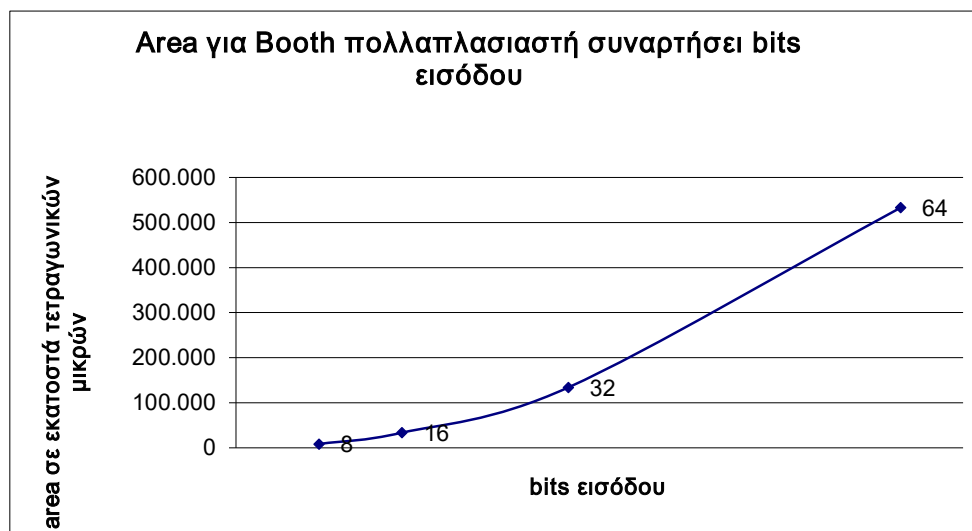
Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον πολλαπλασιαστή booth multiplier για 8,16,32,64 bits εισόδου

Number of bits	Area (10^{-14} m^2)
8	8.534
16	33.741
32	134.118
64	533.109

Πίνακας [4.7] Area για booth



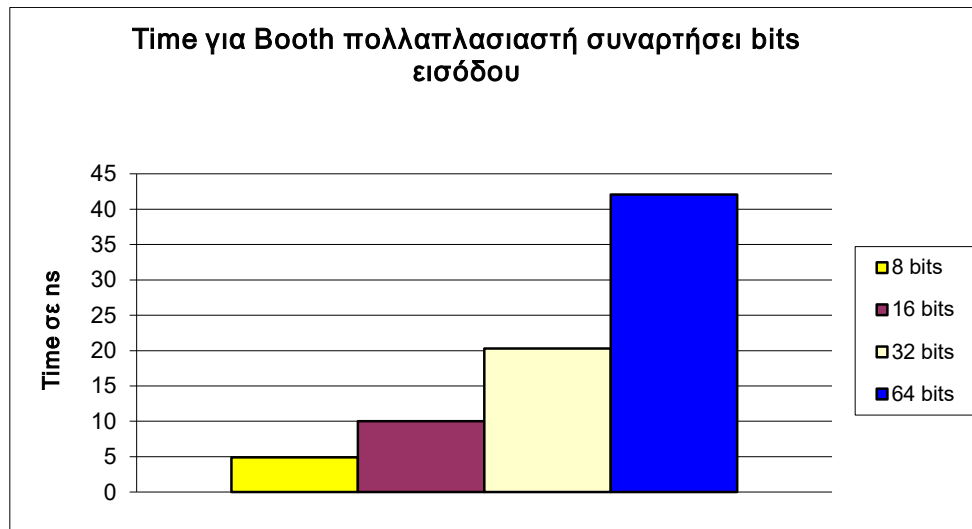
Σχήμα 4.9 Area για booth



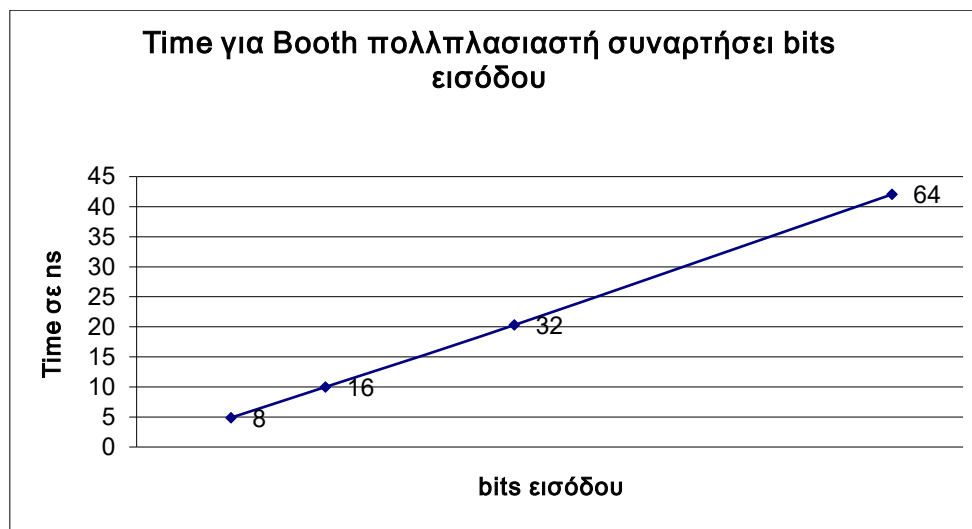
Σχήμα 4.10 Area για booth

Number of bits	Time (nsec)
8	4,89
16	10
32	20,31
64	42,06

Πίνακας [4.8] Time για Booth



Σχήμα 4.11 Time για Booth



Σχήμα 4.12 Time για Booth

Number of bits	Power (watt)
8	9.915e-03
16	3.250e-02
32	1.737e-01
64	1.697

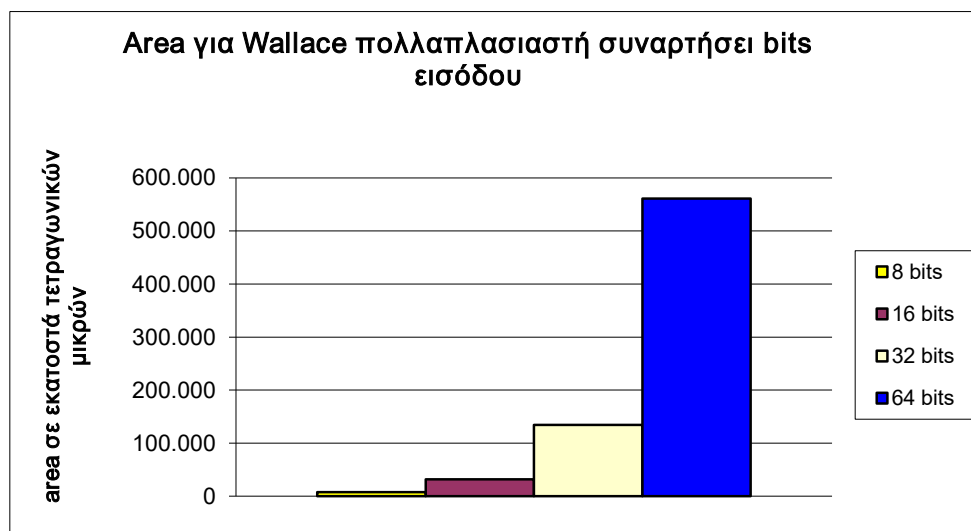
Πίνακας [4.9] Power για booth

4.1.4 Wallace Multiplier

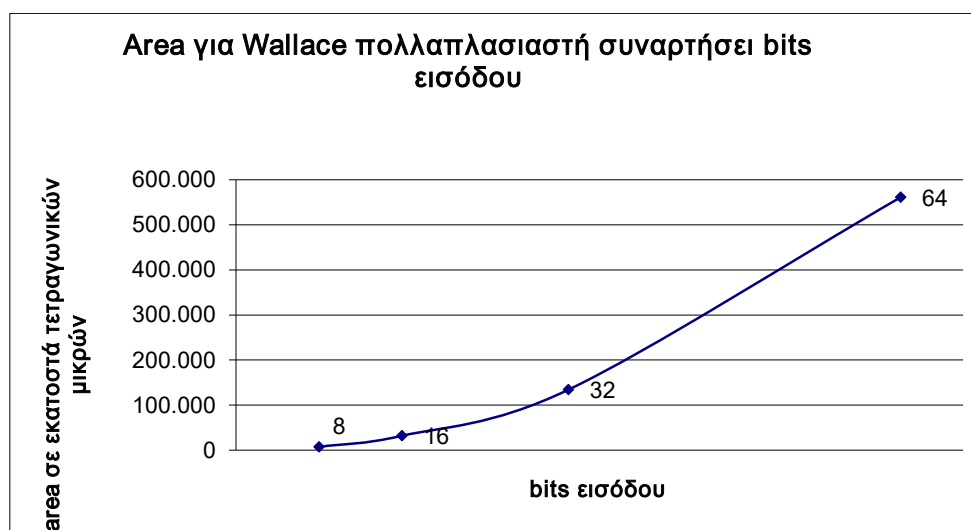
Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον πολλαπλασιαστή booth multiplier για 8,16,32,64 bits εισόδου

Number of bits	Area (10^{-14} m^2)
8	7.314
16	32.043
32	134.118
64	561.359

Πίνακας [4.10] Area για wallace



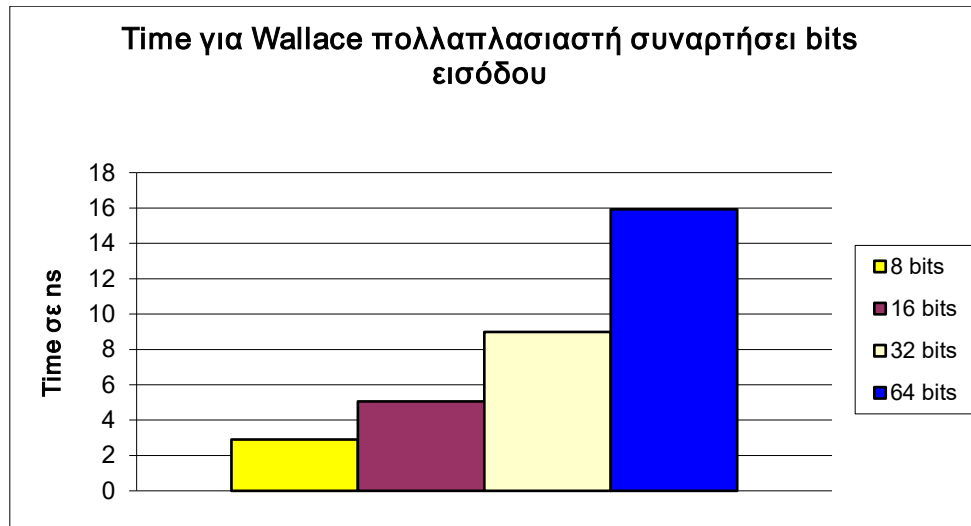
Σχήμα 4.13 Area για wallace



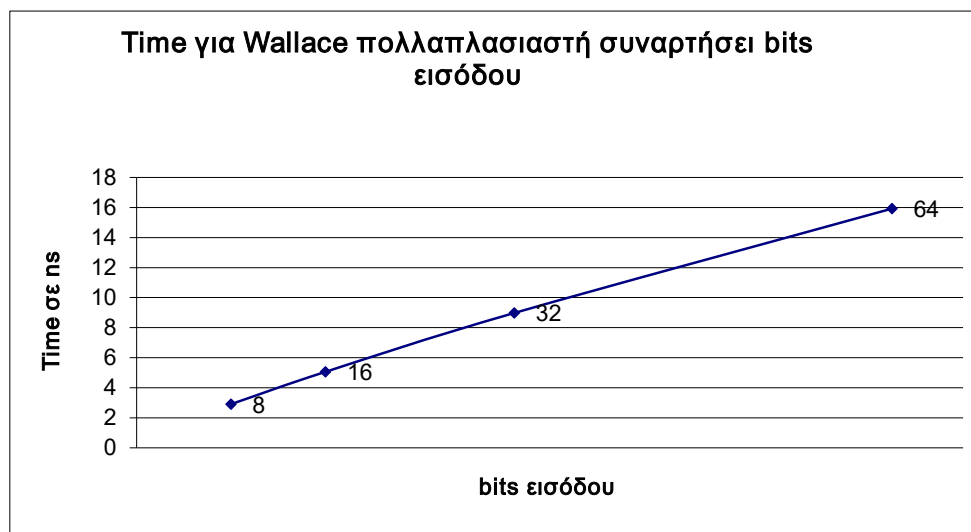
Σχήμα 4.14 Area για wallace

Number of bits	Time (nsec)
8	2,91
16	5,06
32	8,98
64	15,93

Πίνακας [4.11] Time για wallace



Σχήμα 4.15 Time για wallace



Σχήμα 4.16 Time για wallace

Number of bits	Power (Watt)
8	2.736e-03
16	5.079e-03
32	1.134e-02
64	2.232e-01

Πίνακας [4.12] Power για wallace

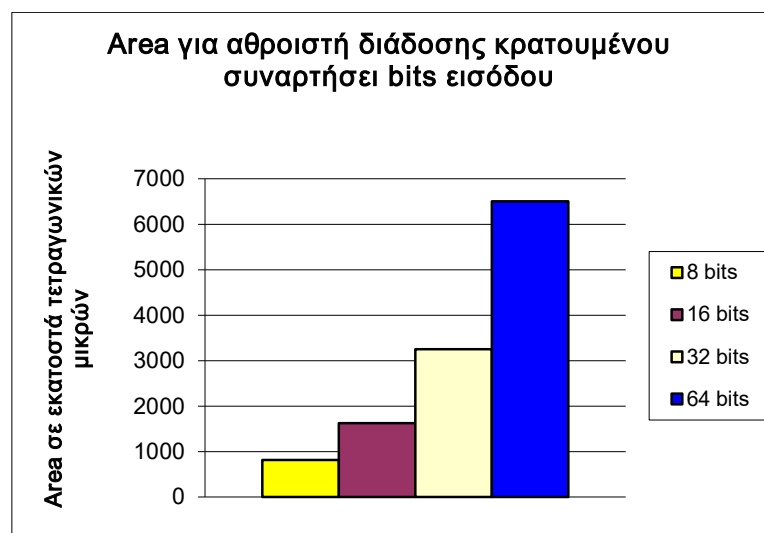
4.2 Παράλληλοι Αθροιστές

4.2.1 Carry Propagate Adder

Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον αθροιστή διάδοσης κρατούμενου για 8,16,32,64 bits εισόδου.

Number of bits	Area (10^{-14} m^2)
8	813,05
16	1626,09
32	3252,17
64	6504,33

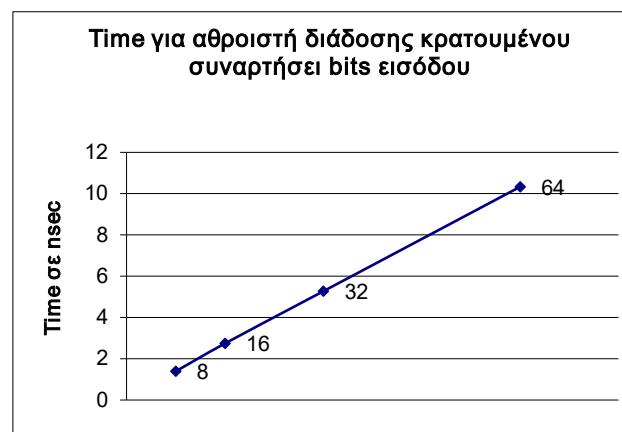
Πίνακας [4.13] Area για carry propagate adder



Σχήμα 4.17 Area για carry propagate adder

Number of bits	Time (nsec)
8	1,39
16	2,74
32	5,27
64	10,33

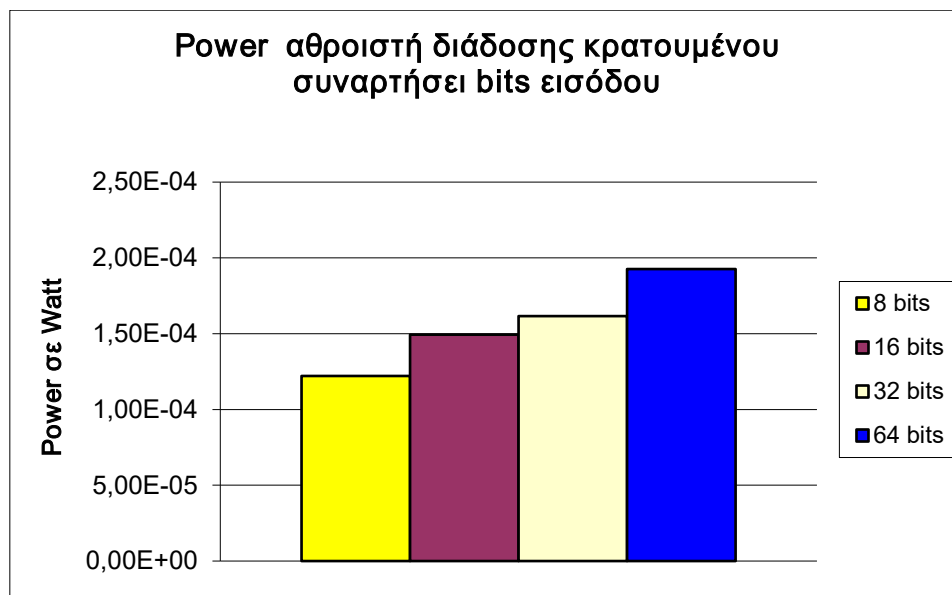
Πίνακας [4.14] Time για carry propagate adder



Σχήμα 4.18 Time για carry propagate adder

Number of bits	Power (Watt)
8	1,20e-04
16	1,493e-04
32	1,615e-04
64	1,926e-04

Πίνακας [4.15] Power για carry propagate adder



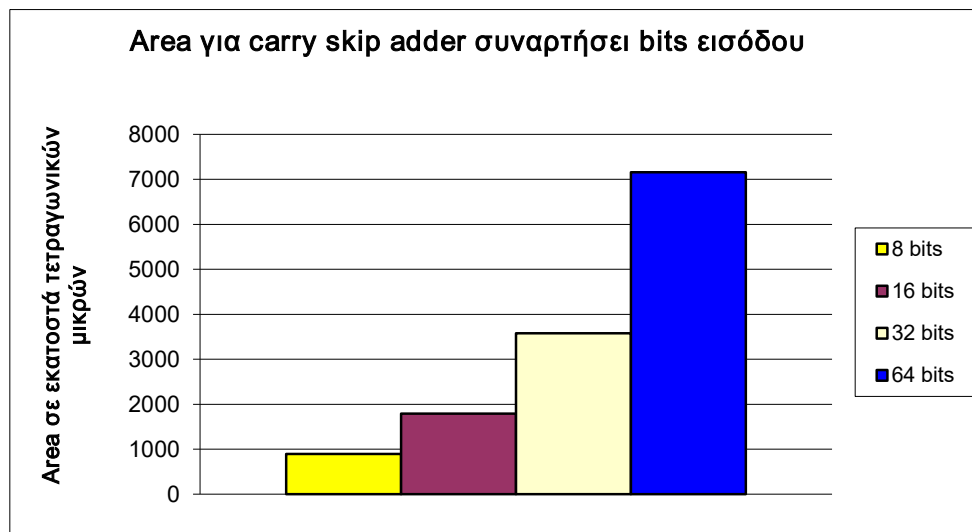
Σχήμα 4.19 Power για carry propagate adder

4.2.2 Carry Skip Adder

Παρακάτω παρουσιάζονται σε πίνακες οι μετρήσεις που γίνανε για area, time και power για τον carry skip adder για 8,16,32,64 bits εισόδου.

Number of bits	Area (10^{-14} m^2)
8	894,39
16	1788,81
32	3577,65
64	7155,33

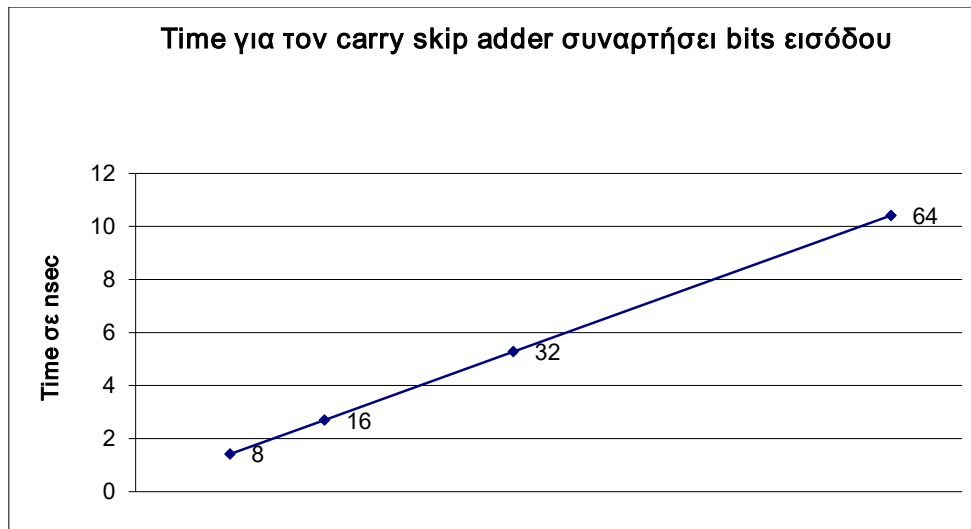
Πίνακας [4.16] Area για carry skip adder



Σχήμα 4.20 Area για carry skip adder

Number of bits	Time (nsec)
8	1,42
16	2,7
32	5,28
64	10,42

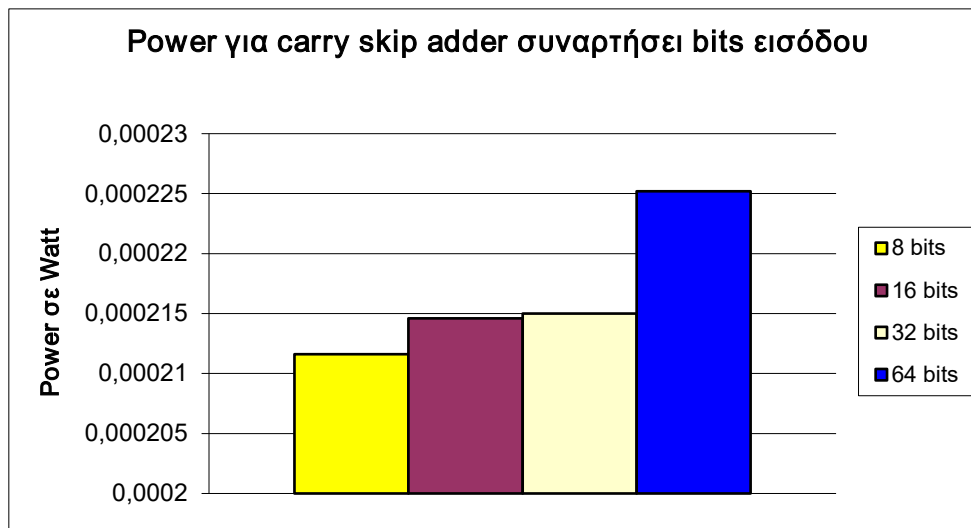
Πίνακας [4.17] Time για carry skip adder



Σχήμα 4.21 Time για carry skip adder

Number of bits	Power (Watt)
8	2.11e-04
16	2.15e-04
32	2.15e-04
64	2.25e-04

Πίνακας [4.18] Power για carry skip adder



Σχήμα 4.22 Power για carry skip adder

4.3 Σειριακοί Πολλαπλασιαστές

4.3.1 Serial Multiplier

Παρακάτω παρουσιάζονται οι μετρήσεις για τον απλό σειριακό πολλαπλασιαστή serial multiplier για 8,16,32 και 64 bits εισόδου.

	Area (10^{-14} m^2)			
	8 bits	16 bits	32 bits	64 bits
Σειριακός πολλαπλασιαστής	3536,97	6334,01	11401,22	20750,18

Πίνακας [4.19] Area για serial multiplier

Time

Το κάθε μερικό γινόμενο (partial product) βγαίνει σε 0.22 nsec

Definitions:

Total Power = Dynamic + Leakage

Dynamic Power = Switching + Internal

Switching Power = load capacitance charge or discharge power

Internal Power = power dissipated within a cell

Αριθμός bits	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
8	1.430e-04	1.429e-04 (99.99%)	1.038e-08 (0.01%)
16	1.430e-04	1.429e-04 (99.99%)	1.038e-08 (0.01%)
32	5.699e-04	5.699e-04 (99.99%)	3.530e-08 (0.01%)
64	1.034e-03	1.034e-03 (99.99%)	6.862e-08 (0.01%)

Πίνακας [4.20] Power για serial multiplier

4.3.2 Serialmul3xdown

Παρακάτω παρουσιάζονται οι μετρήσεις για τον απλό σειριακό πολλαπλασιαστή serialmul3xdown για 8,16,32 και 64 bits εισόδου.

Area (10^{-14} m^2)				
	8 bits	16 bits	32 bits	64 bits
Σειριακός Πολλαπλασιαστής Κατεβάσματος	6854.23	13749.11	27538.87	55118.39

Πίνακας [4.21] Area για serialmul3xdown

Time (nsec)				
	8 bits	16 bits	32 bits	64 bits
Σειριακός Πολλαπλασιαστής Κατεβάσματος	1.17	1.34	1.69	2.34

Πίνακας [4.22] Time για serialmul3xdown

Definitions:

Total Power = Dynamic + Leakage

Dynamic Power = Switching + Internal

Switching Power = load capacitance charge or discharge power

Internal Power = power dissipated within a cell

Αριθμός bits	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
8	1.303e-08	2.884e-09 (22,13%)	1.015e-08 (77,87%)
16	2.245e-08	2.477e-09 (11,03%)	1.998e-08 (88,97%)
32	4.481e-08	5.175e-09 (11,55%)	3.964e-08 (88,45%)
64	8.953e-08	1.058e-08 (11,82%)	7.895e-08 (88,18%)

Πίνακας [4.23] Power για serialmul3xdown

Θα πρέπει να σημειώσουμε σε αυτό το σημείο ότι οι δύο αυτοί σειριακοί πολλαπλασιαστές δεν θα μπουν στην διαδικασία της σύγκρισης εξαιτίας της τόσο διαφορετικής τους φιλοσοφίας. Ο serial multiplier σε αντίθεση με τον serialmul3xdown δεν είναι συστολικός που σημαίνει ότι πρέπει να περιμένει να τελειώσει ο πολλαπλασιασμός που γίνεται εκείνη την στιγμή για να περάσει στον επόμενο. Σε όλη αυτήν την διάρκεια ένα μεγάλο μέρος του πολλαπλασιαστή μένει ανενεργό από πλευράς υπολογισμών αλλά συνεχίζει να τροφοδοτείται με ρεύμα.

Από την άλλη πλευρά ο serialmul3xdown επιτρέπει την συνεχή λειτουργία χάρις στα D flip/flop που χωρίζουν το κάθε στάδιο του πολλαπλασιαστή.

Συμπερασματικά, βλέπουμε ότι μια σύγκριση μεταξύ των δύο πολλαπλασιαστών θα ήταν εκ των πραγμάτων άνευ ουσίας .

Κεφάλαιο 5

Σχόλια και Συμπεράσματα

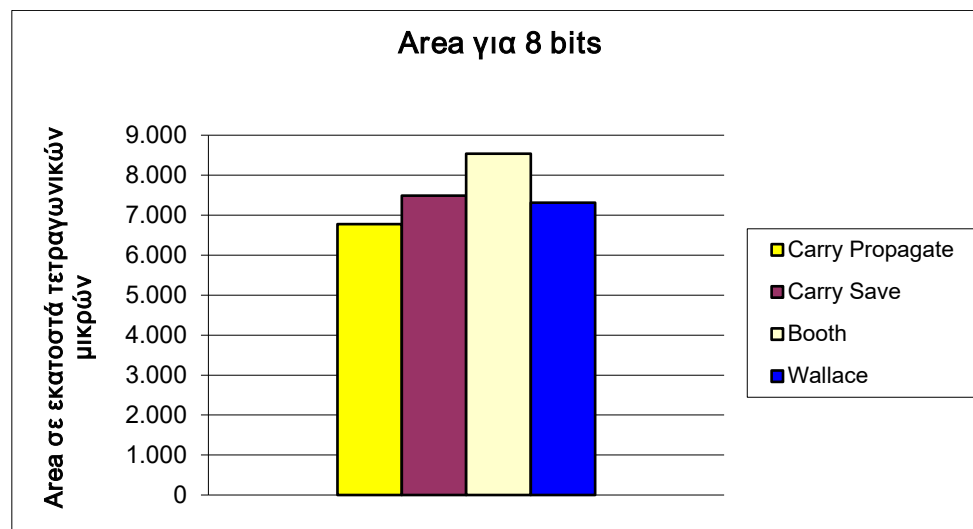
5.1 Σύγκριση Παράλληλων Πολλαπλασιαστών

Σε αυτό το κεφάλαιο θα εξετάσουμε την επίδοση των παράλληλων πολλαπλασιαστών και στα τρία επίπεδα ανάλυσης και θα προβούμε στις απαραίτητες συγκρίσεις προκειμένου να καταλήξουμε στο ποιος είναι ο πιο αποδοτικός πολλαπλασιαστής.

Θα αρχίσουμε με την σύγκριση του area για τα διάφορα μήκη λέξεων.

Είδος Πολλαπλασιαστή	Area σε εκατοστά του τετραγωνικού μικρού(10^{-14} m^2)
Carry Propagate Multiplier	6772.93
Carry Save Multiplier	7488.45
Booth Multiplier	8533.75
Wallace Multiplier	7313.58

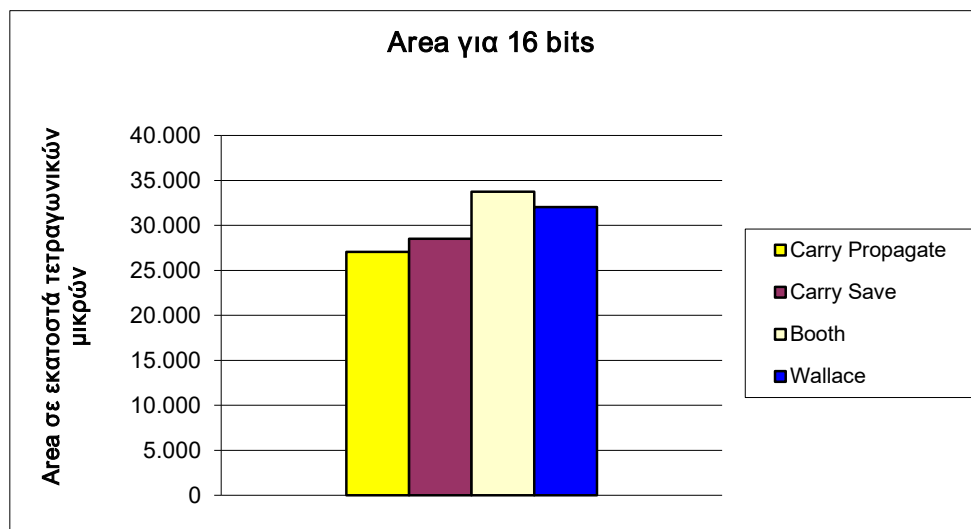
Πίνακας [5.1] Area για 8 bits



Σχήμα 5.1 Area για 8 bits

Είδος Πολλαπλασιαστή	Area σε εκατοστά του τετραγωνικού μικρού(10^{-14}m^2)
Carry Propagate Multiplier	27067.33
Carry Save Multiplier	28498.44
Booth Multiplier	33740.81
Wallace Multiplier	32043.06

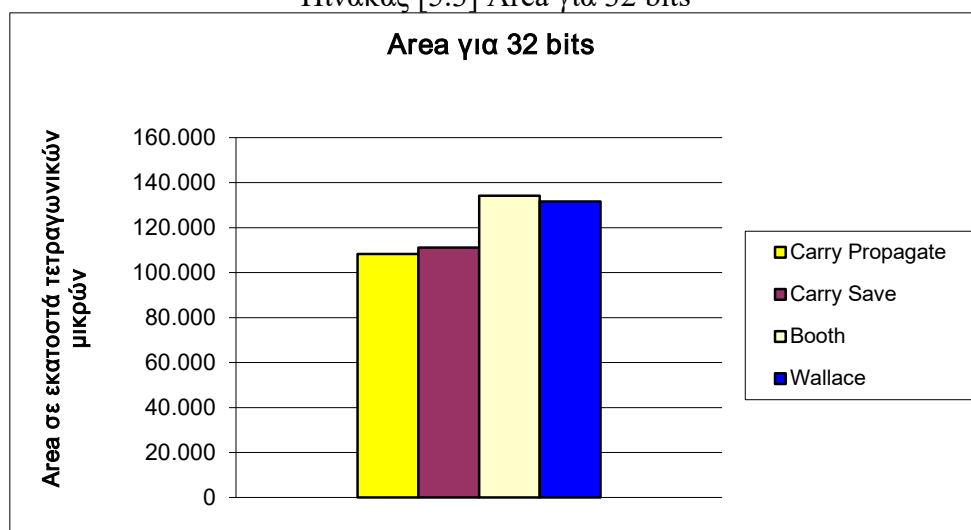
Πίνακας [5.2] Area για 16 bits



Σχήμα 5.2 Area για 16 bits

Είδος Πολλαπλασιαστή	Area σε εκατοστά του τετραγωνικού μικρού(10^{-14}m^2)
Carry Propagate Multiplier	108247.27
Carry Save Multiplier	111109.52
Booth Multiplier	134117.56
Wallace Multiplier	131611.50

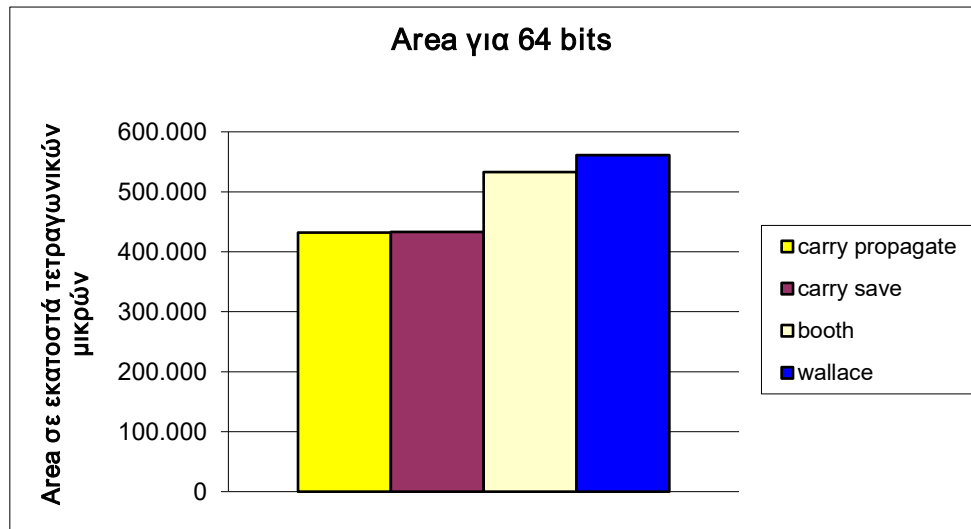
Πίνακας [5.3] Area για 32 bits



Σχήμα 5.3 Area για 32 bits

Είδος Πολλαπλασιαστή	Area σε εκατοστά του τετραγωνικού μικρού(10^{-14} m^2)
Carry Propagate Multiplier	431906.53
Carry Save Multiplier	433329.50
Booth Multiplier	533109.52
Wallace Multiplier	561359.50

Πίνακας [5.4] Area για 64 bits

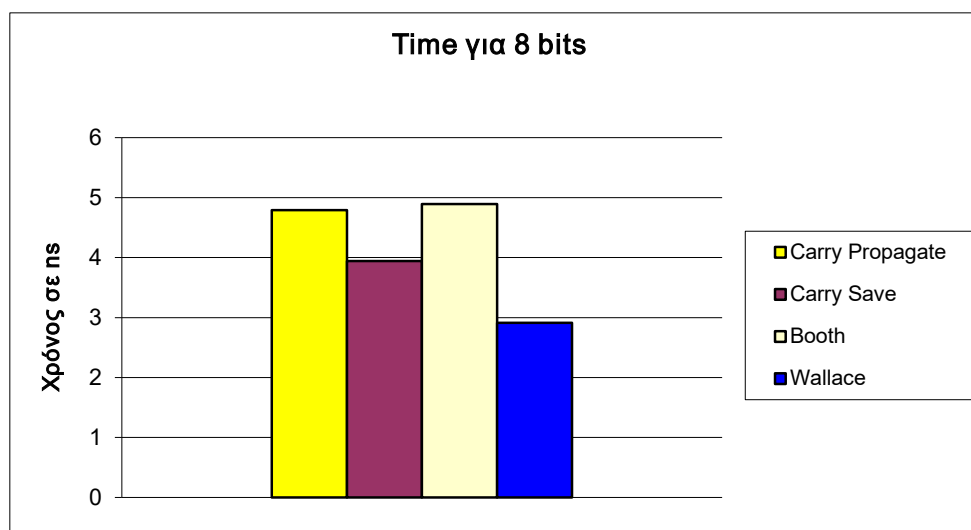


Σχήμα 5.4 Area για 64 bits

Στην συνέχεια θα συγκρίνουμε τις επιδόσεις στον τομέα του χρόνου.

Είδος Πολλαπλασιαστή	Time σε nsec
Carry Propagate Multiplier	4.79
Carry Save Multiplier	3.94
Booth Multiplier	4.89
Wallace Multiplier	2.91

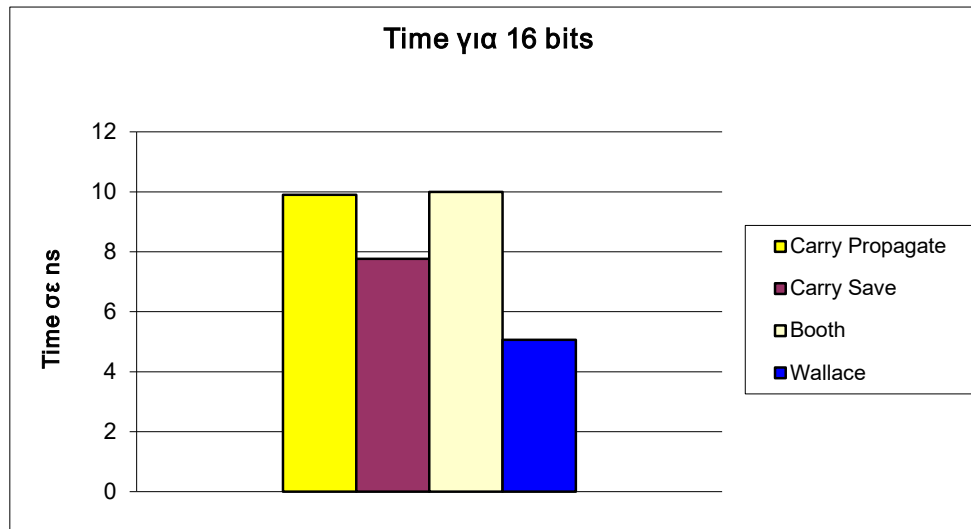
Πίνακας [5.5] Time για 8 bits



Σχήμα 5.5 Time για 8 bits

Είδος Πολλαπλασιαστή	Time σε nsec
Carry Propagate Multiplier	9.90
Carry Save Multiplier	7.76
Booth Multiplier	10.00
Wallace Multiplier	5.06

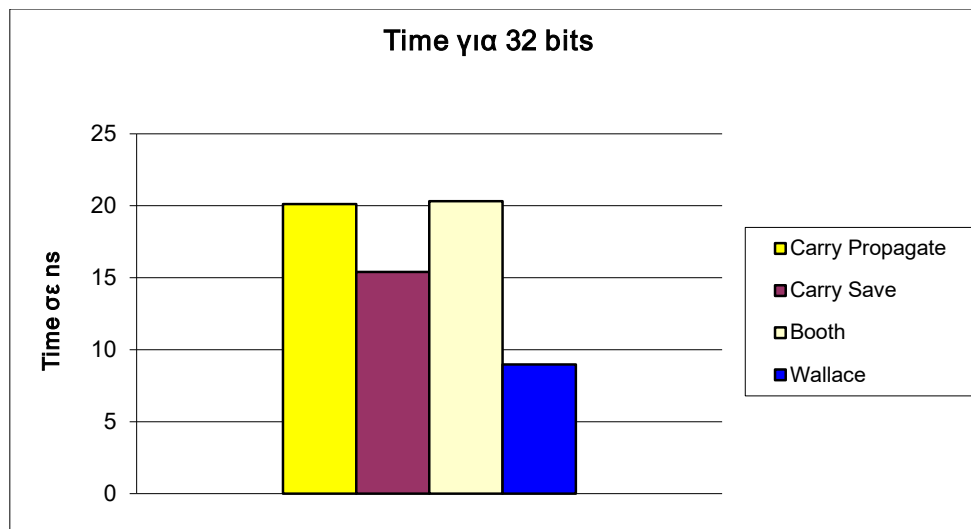
Πίνακας [5.6] Time για 16 bits



Σχήμα 5.6 Time για 16 bits

Είδος Πολλαπλασιαστή	Time σε nsec
Carry Propagate Multiplier	20.12
Carry Save Multiplier	15.40
Booth Multiplier	20.31
Wallace Multiplier	8.98

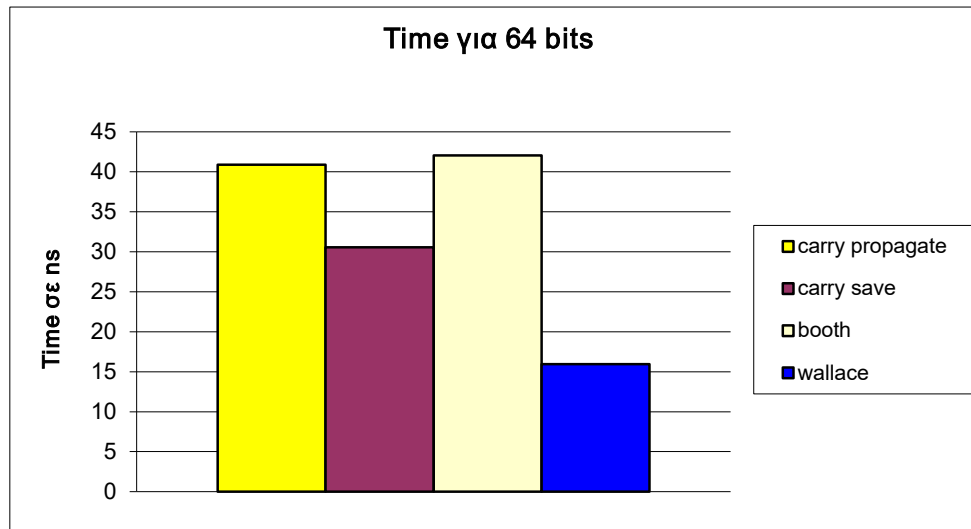
Πίνακας [5.7] Time για 32 bits



Σχήμα 5.7 Time για 32 bits

Είδος Πολλαπλασιαστή	Time σε nsec
Carry Propagate Multiplier	40.88
Carry Save Multiplier	30.56
Booth Multiplier	42.06
Wallace Multiplier	15.93

Πίνακας [5.8] Time για 64 bits



Σχήμα 5.8 Time για 64 bits

Τέλος θα συγκρίνουμε τις επιδόσεις στον τομέα της καταναλισκόμενης ισχύος.

Definitions:

Total Power = Dynamic + Leakage

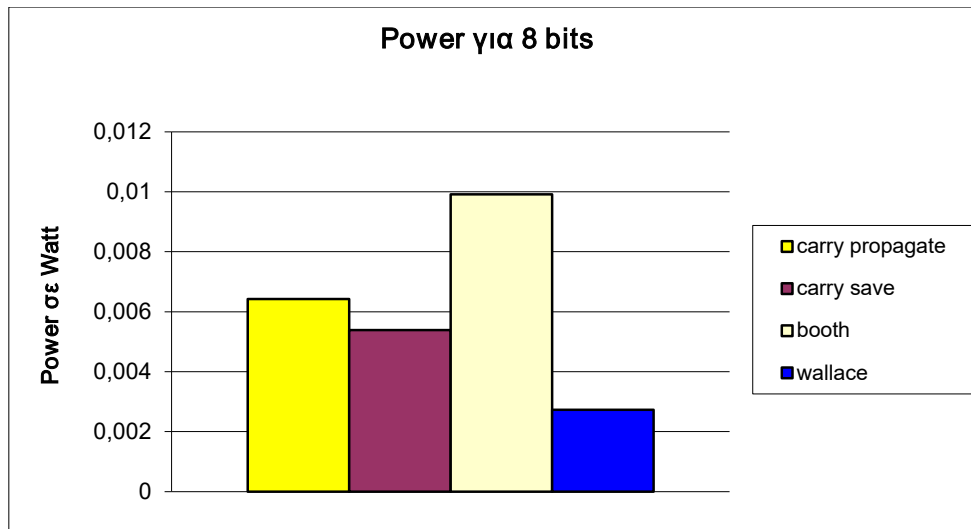
Dynamic Power = Switching + Internal

Switching Power = load capacitance charge or discharge power

Internal Power = power dissipated within a cell

Είδος Πολλαπλασιαστή	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
Carry Propagate Multiplier	6.422e-03	6.422e-03 (100%)	3.452e-08 (0%)
Carry Save Multiplier	5.386e-03	5.386e-03 (100%)	3.830e-08 (0%)
Booth Multiplier	9.915e-03	9.915e-03 (100%)	4.304e-08 (0%)
Wallace Multiplier	2.736e-03	2.736e-03 (100%)	3.275e-08 (0%)

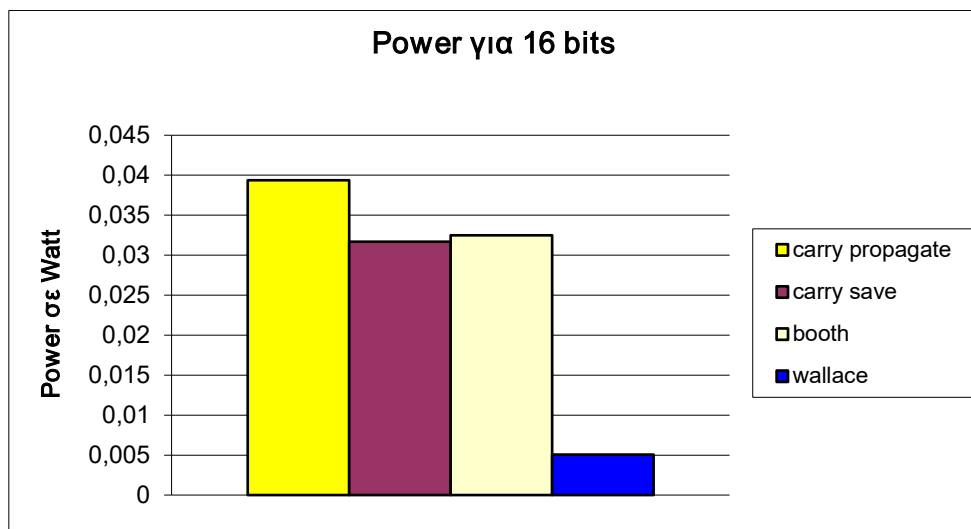
Πίνακας [5.9] Power για 8 bits



Σχήμα 5.9 Power για 8 bits

Είδος Πολλαπλασιαστή	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
Carry Propagate Multiplier	3.938e-02	3.938e-02 (100%)	1.381e-07 (0%)
Carry Save Multiplier	3.170e-02	3.170e-02 (100%)	1.456e-07 (0%)
Booth Multiplier	3.250e-02	3.250e-02 (100%)	1.696e-07 (0%)
Wallace Multiplier	5.079e-03	5.079e-03 (100%)	1.447e-07 (0%)

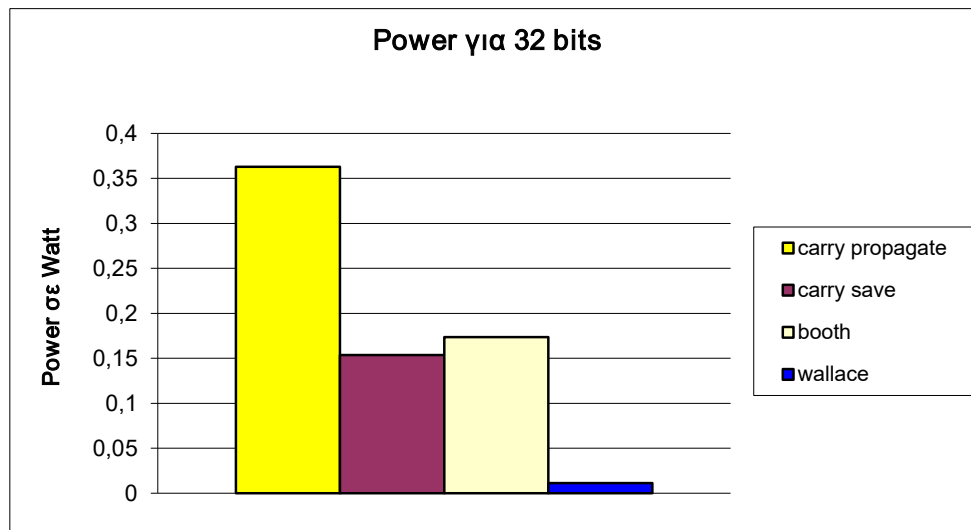
Πίνακας [5.10] Power για 16 bits



Σχήμα 5.10 Power για 16 bits

Είδος Πολλαπλασιαστή	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
Carry Propagate Multiplier	3.630e-01	3.630e-01 (100%)	5.523e-07 (0%)
Carry Save Multiplier	1.536e-01	1.536e-01 (100%)	5.674e-07 (0%)
Booth Multiplier	1.737e-01	1.737e-01 (100%)	6.725e-07 (0%)
Wallace Multiplier	1.134e-02	1.134e-02 (100%)	1.237e-08 (0%)

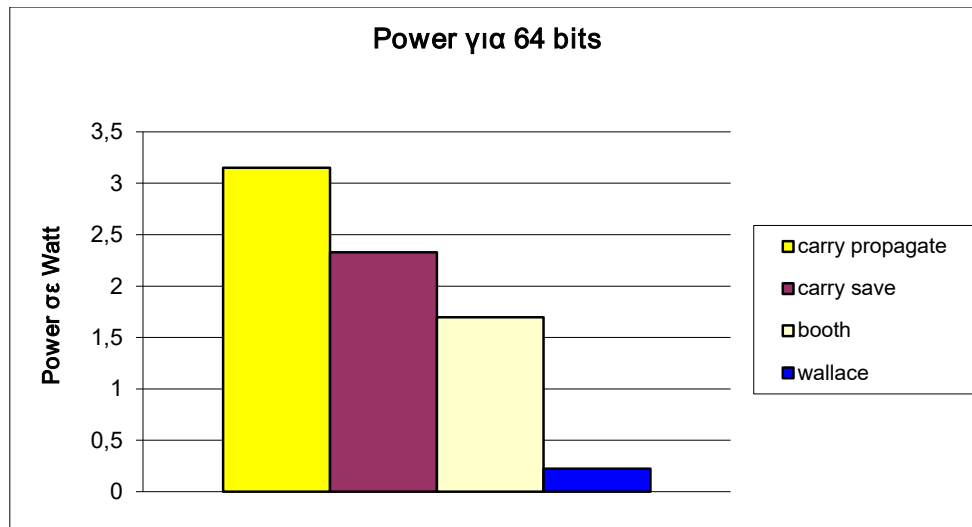
Πίνακας [5.11] Power για 32 bits



Σχήμα 5.11 Power για 32 bits

Είδος Πολλαπλασιαστή	Total Power (Watt)	Dynamic Power (Watt)	Leakage Power (Watt)
Carry Propagate Multiplier	3.152	3.152 (100%)	1.138e-05 (0%)
Carry Save Multiplier	2.328	2.238 (100%)	1.456e-05 (0%)
Booth Multiplier	1.697	1.697 (100%)	1.826e-05 (0%)
Wallace Multiplier	2.232e-01	2.232e-01 (100%)	1.427e-06 (0%)

Πίνακας [5.12] Power για 64 bits



Σχήμα 5.12 Power για 64 bits

Οφείλουμε να σχολιάσουμε στο σημείο αυτό ότι το leakage είναι τόσο μικρό γιατί είμαστε σε τεχνολογία 0.25μm UMC. Εάν ήμασταν σε τεχνολογία 0.13μm τότε θα βλέπαμε να καταλαμβάνει μεγαλύτερο ποσοστό επί της συνολικής καταναλισκόμενης ισχύος.

Με όλα τα δεδομένα στο τραπέζι μπορούμε να προβούμε σε κάποιες παρατηρήσεις και να φτάσουμε σε σίγουρα συμπεράσματα..

Αρχικά παρατηρούμε ότι ο carry propagate πολλαπλασιαστής είναι πιο αργός από τους άλλους με την εξαίρεση του Booth. Ωστόσο χρήσιμο είναι να θυμηθούμε ότι το μεγάλο πλεονέκτημα του πολλαπλασιαστή Booth δεν είναι η ταχύτητα του αλλά το γεγονός ότι μπορεί να λειτουργήσει τόσο για αρνητικούς όσο και για θετικούς αριθμούς.

Ο carry save είναι πιο γρήγορος από τον carry propagate όπως περιμέναμε άλλωστε, καταλαμβάνοντας περίπου την ίδια επιφάνεια με αυτόν και καταναλώνοντας λιγότερη ισχύ.

Είναι φανερό ότι ο καλύτερος πολλαπλασιαστής από πλευράς κατανάλωσης ισχύος αλλά και από πλευράς χρόνου είναι ο wallace, με μοναδικό μειονέκτημα την εκτεταμένη επιφάνεια που καταλαμβάνει επάνω στο chip. Από πλευράς κατανάλωσης ισχύος, ο wallace είναι μια τάξη μεγέθους κάτω από τους υπόλοιπους πολλαπλασιαστές ενώ είναι και πολύ πιο γρήγορος.

Θα πρέπει όμως να είμαστε πολύ προσεκτικοί με τον πολλαπλασιαστή wallace. Στις προσομοιώσεις που γίνανε δεν λήφθηκε υπόψη ούτε το εκτεταμένο wiring ούτε το πολύ μεγάλο lay – out του πολλαπλασιαστή. Η επίδραση των δύο αυτών παραμέτρων στην κατανάλωση ισχύος και στον χρόνο βρίσκεται υπό μελέτη, οπότε θα πρέπει να ξέρουμε ότι τα συνολικά αποτελέσματα ίσως να μην υπερέχουν τόσο πολύ έναντι των άλλων.

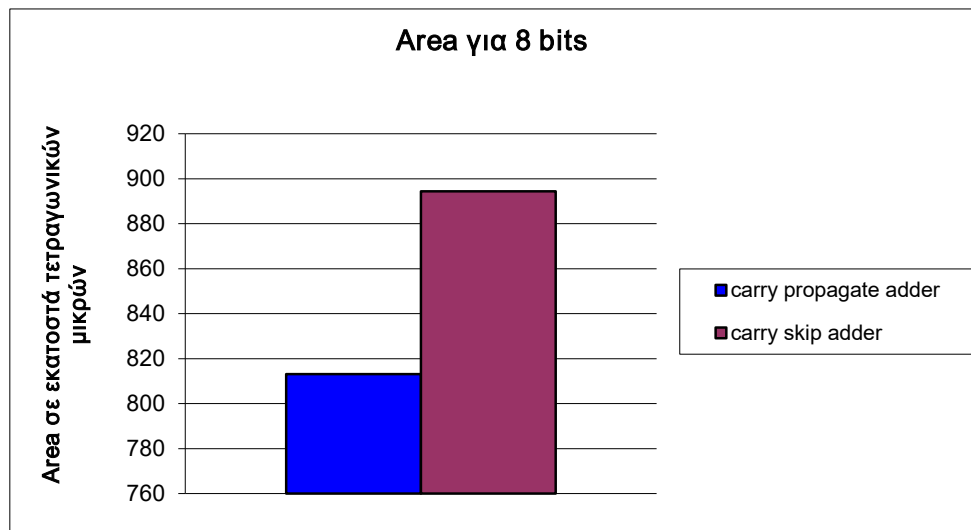
5.2 Σύγκριση Παράλληλων Αθροιστών

Σε αυτό το κεφάλαιο θα παραθέσουμε τα συγκριτικά αποτελέσματα των δύο παράλληλων αθροιστών όσον αφορά area, time και power για 8,16,32 και 64 bits.

8 bits

	Area (10^{-14} m^2)
Carry Propagate Adder	813.05
Carry Skip Adder	894.39

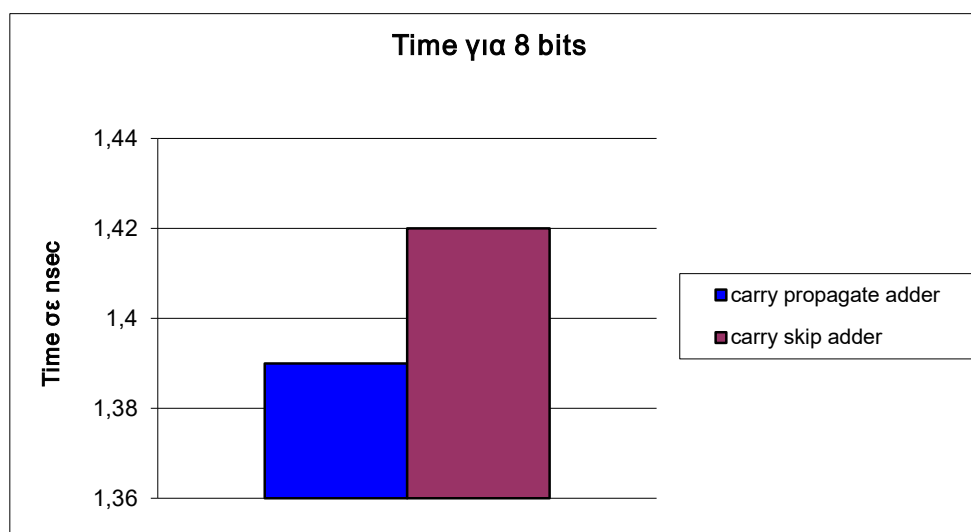
Πίνακας [5.13] Area για 8 bits



Σχήμα 5.13 Area για 8 bits

	Time (nsec)
Carry Propagate Adder	1.39
Carry Skip Adder	1.42

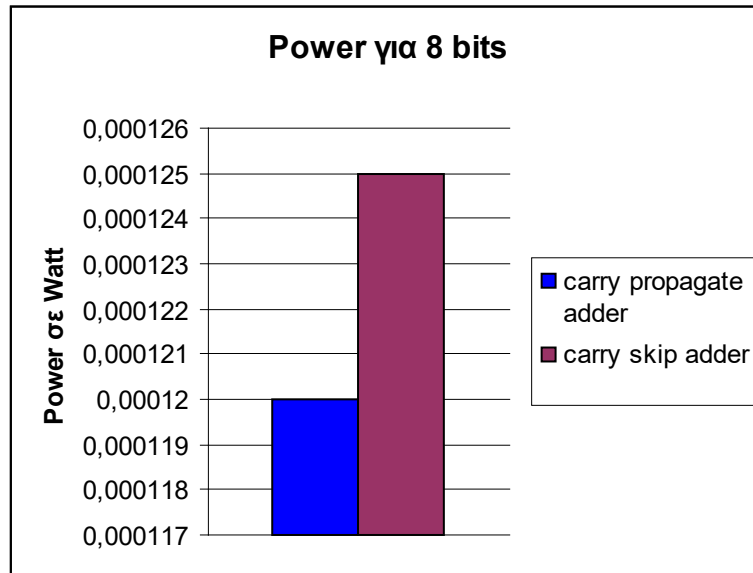
Πίνακας [5.14] Time για 8 bits



Σχήμα 5.14 Time για 8 bits

	Power (Watt)
Carry Propagate Adder	1.20e-04
Carry Skip Adder	1.25e-04

Πίνακας [5.15] Power για 8 bits

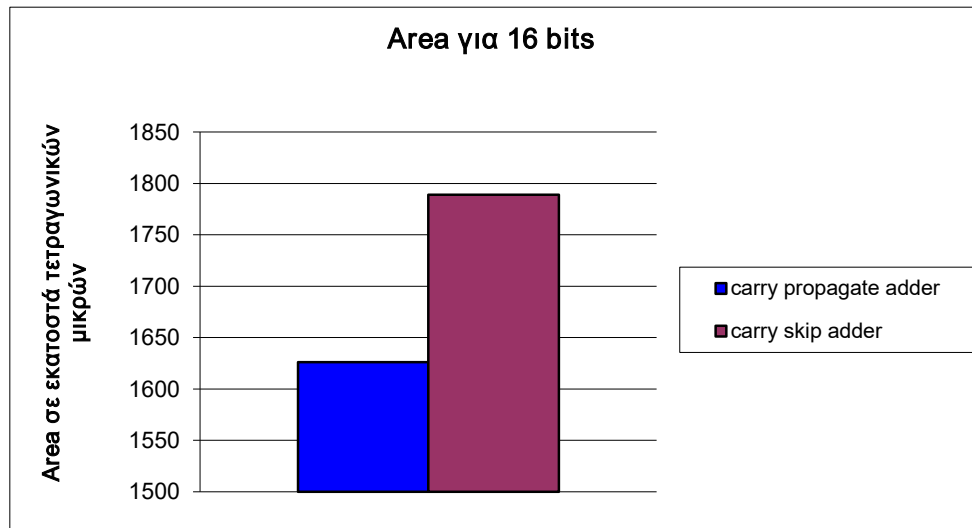


Σχήμα 5.15 Power για 8 bits

16 bits

	Area (10^{-14} m^2)
Carry Propagate Adder	1626.09
Carry Skip Adder	1788.81

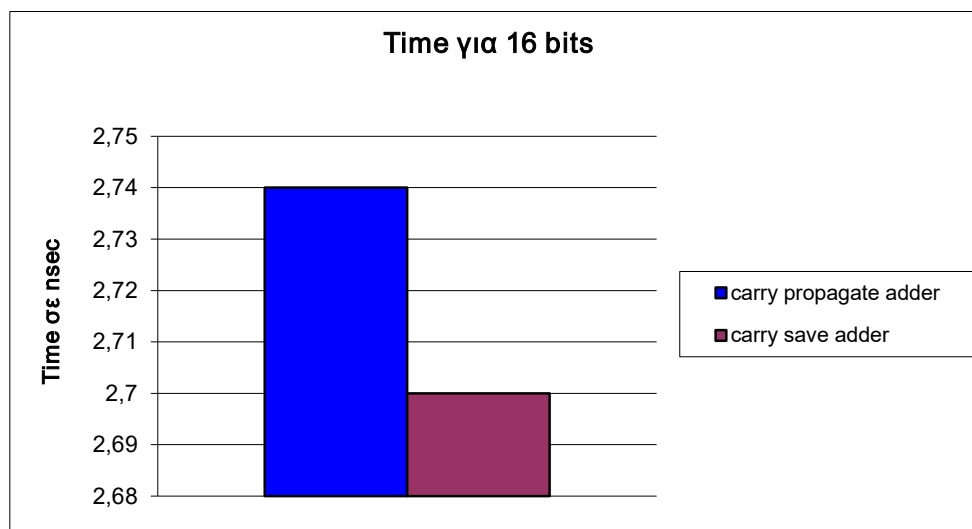
Πίνακας [5.16] Area για 16 bits



Σχήμα 5.16 Area για 16 bits

	Time (nsec)
Carry Propagate Adder	2,74
Carry Skip Adder	2,70

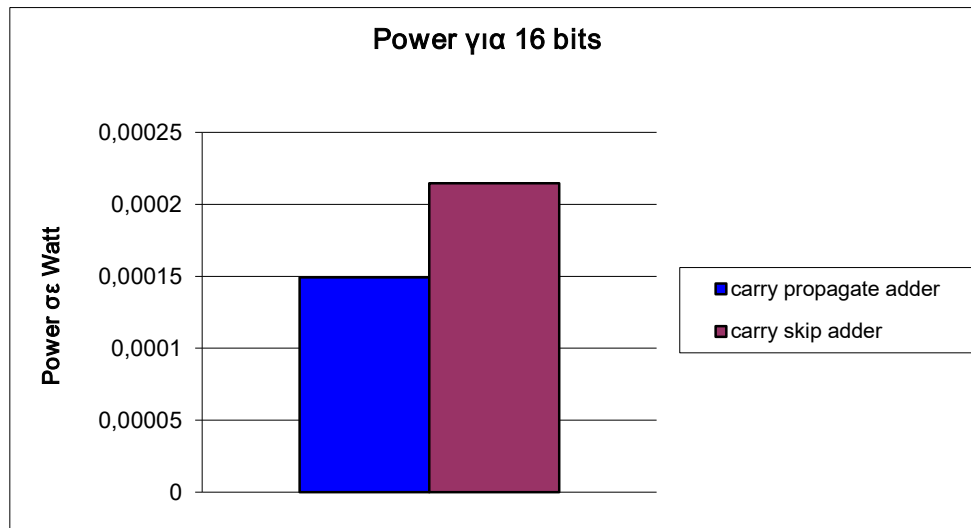
Πίνακας [5.17] Time για 16 bits



Σχήμα 5.17 Time για 16 bits

	Power (Watt)
Carry Propagate Adder	1,49-04
Carry Skip Adder	2,15e-04

Πίνακας [5.18] Power για 16 bits

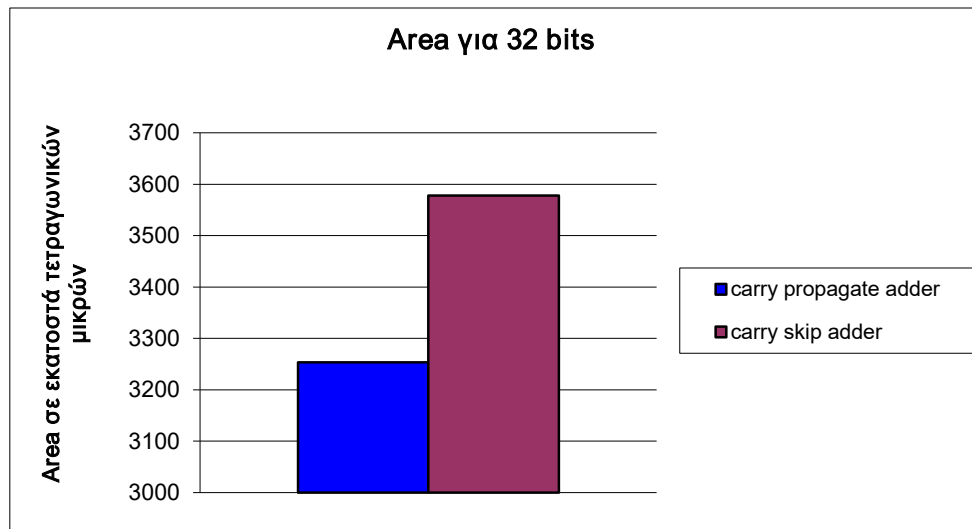


Σχήμα 5.18 Power για 16 bits

32 bits

	Area (10^{-14} m^2)
Carry Propagate Adder	3252.17
Carry Skip Adder	3577.65

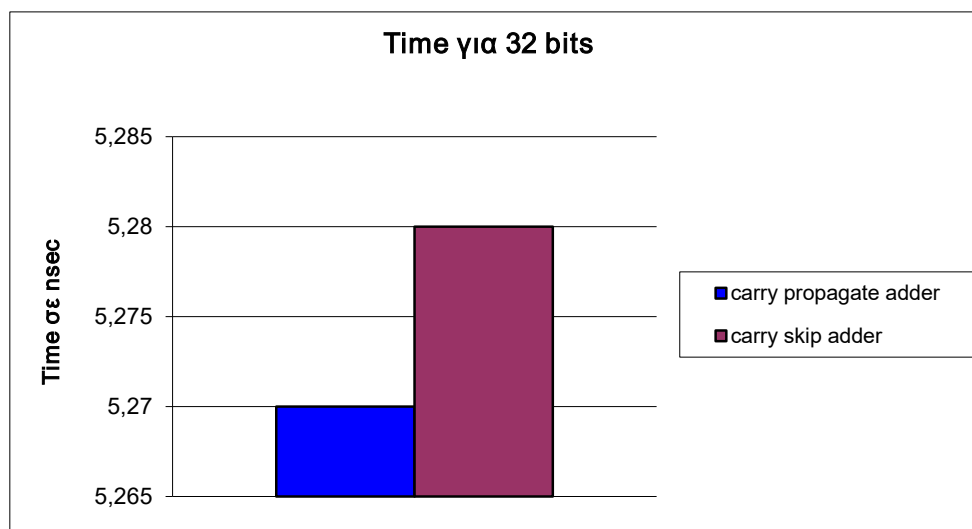
Πίνακας [5.19] Area για 32 bits



Σχήμα 5.19 Area για 32 bits

	Time (nsec)
Carry Propagate Adder	5,27
Carry Skip Adder	5,28

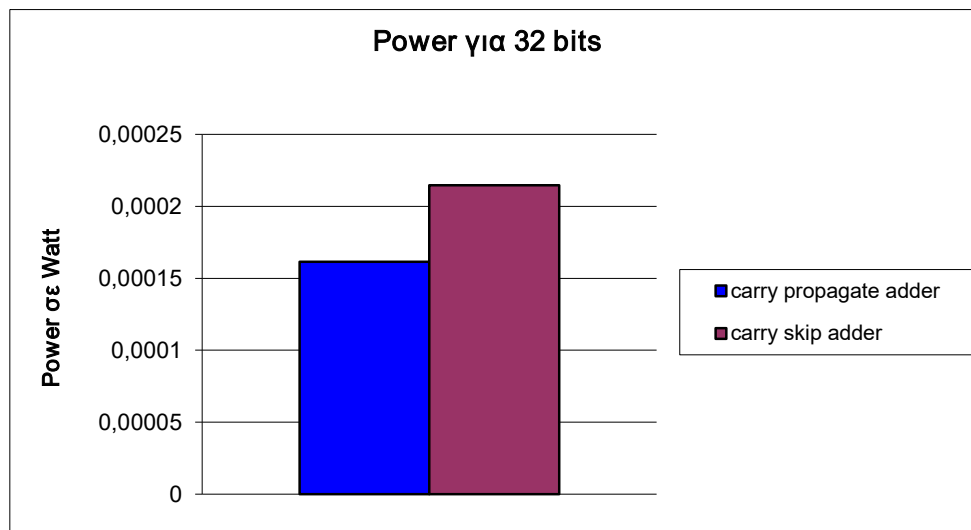
Πίνακας [5.20] Time για 32 bits



Σχήμα 5.20 Time για 32 bits

	Power (Watt)
Carry Propagate Adder	1,62-04
Carry Skip Adder	2,15e-04

Πίνακας [5.21] Power για 32 bits

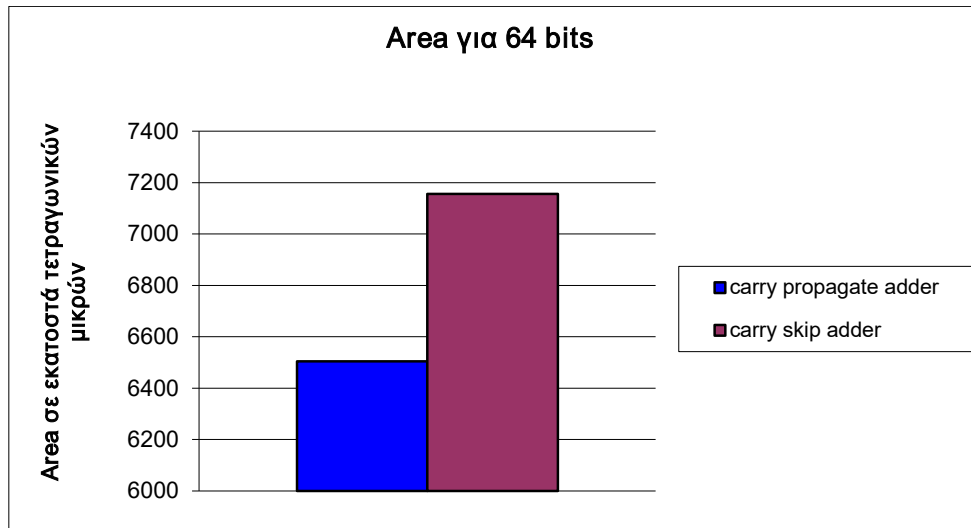


Σχήμα 5.21 Power για 32 bits

64 bits

	Area (10^{-14} m^2)
Carry Propagate Adder	6504,33
Carry Skip Adder	7155,33

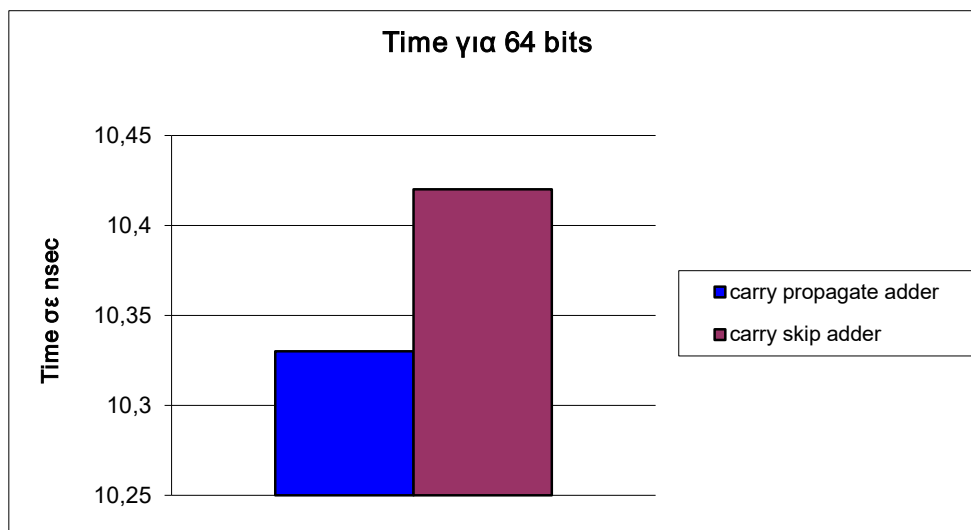
Πίνακας [5.22] Area για 64 bits



Σχήμα 5.22 Area για 64 bits

	Time (nsec)
Carry Propagate Adder	10,33
Carry Skip Adder	10,42

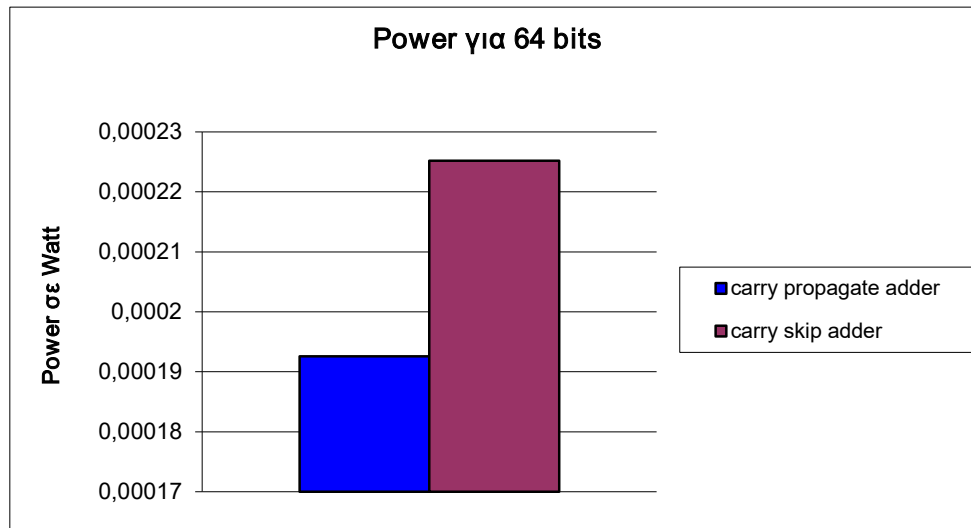
Πίνακας [5.23] Time για 64 bits



Σχήμα 5.23 Time για 64 bits

	Power (Watt)
Carry Propagate Adder	1,93-04
Carry Skip Adder	2,25e-04

Πίνακας [5.24] Power για 64 bits



Σχήμα 5.24 Power για 64 bits

Μετά απο την προσεκτική μελέτη των δεδομένων των δύο παράλληλων αθροιστών, είμαστε πλέον σε θέση να καταλήξουμε στο ποιος είναι ο καλύτερος με βάση και τις τρεις παραμέτρους.

Παίρνοντας υπόψιν όλα τα δεδομένα, βλέπουμε οτι τελικά ο παράλληλος αθροιστής διάδοσης κρατουμένου υπερτερεί με βραχεία κεφαλή έναντι του carry skip adder, κυρίως σε θέμα κατανάλωσης ισχύος. Όσον αφορά τον χρόνο και την επιφάνεια, οι δύο αθροιστές είναι λίγο πολύ στο ίδιο επίπεδο αλλά η σημαντικότητα της μικρότερης κατανάλωσης ισχύος μας ωθεί στο να ανακηρύξουμε τον παράλληλο πολλαπλασιαστή διάδοσης κρατουμένου νικητή σε αυτή την άτυπη μονομαχία.

Ο λόγος για τον οποίο ο carry skip adder είναι σχετικά πιο αργός απο τον carry propagate και καταναλώνει και μεγαλύτερη ισχύ είναι επειδή το κύκλωμα που χρησιμεύει για να αποφασίσει ο αθροιστής αν θα πάρει υπόψιν του το κρατούμενο, αποτελείται από λογικές πύλες οι οποίες εισάγουν καθυστέρηση και επιπλέον κατανάλωση ισχύος. Από την άλλη πλευρά ο αθροιστής διάδοσης κρατουμένου είναι βελτιστοποιημένος, εξού και το προβάδισμά του έναντι του άλλου.

Παράρτημα Α

Κώδικας

Full adder

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

entity f_adder is
    port (
        x: in STD_LOGIC;
        y: in STD_LOGIC;
        z: in STD_LOGIC;
        s: out STD_LOGIC;
        c: out STD_LOGIC
    );
end f_adder;

architecture netlist of f_adder is
begin
    s<= x xor y xor z;
    cout<= ((x or y) and z) or (x and y) ;
end netlist;
```

Full adder nand nor

```

library IEEE;
use IEEE.std_logic_1164.all;

entity f_adder is
    port (
        x: in STD_LOGIC;
        y: in STD_LOGIC;
        z: in STD_LOGIC;
        s: out STD_LOGIC;
        c: out STD_LOGIC
    );
end f_adder;

architecture NAND_NOR of f_adder is
    signal minus_x, minus_y, minus_z
    , a, b, c1, d, e, f, g, h, k, l, m, n, o: std_logic;
begin
    minus_x<= (x NAND x);
    minus_y<= (y NAND y);
    minus_z<= (z NAND z);
    a<= (minus_x NAND minus_y);
    b<= x NAND y;
    c1<= (minus_x NAND y);
    d<= (minus_y NAND x);
    e<= a NAND b;
    f<= c1 NAND d;
    g<= e NAND z;
    h<= (f NAND minus_z);
    s<= g NAND h;

    k<= x NAND y;
    l<= y NAND z;
    m<= k NAND l;
    n<= m NAND m;
    o<= x NAND z;
    c<= n NAND o;

end NAND_NOR;

```

Adder N

```

library      IEEE;

```

```

use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity adder_n is
    generic(N:integer:=4);
    port (
        a,b:in std_logic_vector(N-1 downto 0);
        cin:in std_logic;
        s:out std_logic_vector(N-1 downto 0);
        cout:out std_logic);
end adder_n;

architecture adder_n1 of adder_n is

    signal c: std_logic_vector(N downto 1);

begin
    s<=(a xor b) xor (c(N-1 downto 1)&cin);
    c<=((a or b) and (c(N-1 downto 1)&cin)) or (a and b);
    cout<=c(N);
end adder_n1;

```

```

library      IEEE;

                                Adder_N nand

```

```

use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity adder_n is
    generic(N:integer:=4);
    port (
        a,b:in std_logic_vector(N-1 downto 0);
        cin:in std_logic;
        s:out std_logic_vector(N-1 downto 0);
        cout:out std_logic);
end adder_n;

architecture struct of adder_n is
    component f_adder
        port(
            x: in STD_LOGIC;
            y: in STD_LOGIC;
            z: in STD_LOGIC;
            s: out STD_LOGIC;
            c: out STD_LOGIC
        );
    end component;
    signal c2: std_logic_vector(0 to N);

begin
    c2(0)<=cin;
    cout<=c2(N);
    adders: for k in 0 to N-1 generate
        A1: f_adder port map (a(k),b(k),c2(k),s(k),c2(k+1));
    end generate adders;
end struct;

```

Carry skip adder

```

library IEEE;

```

```

use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity carry_skip_adder is
    generic(N:integer:=4);           -- to N prepei na
    port (
        a,b:in std_logic_vector(N-1 downto 0);
        cin:in std_logic;
        s:out std_logic_vector(N-1 downto 0);
        cout:out std_logic);

end carry_skip_adder;

architecture carry_skip_adder of carry_skip_adder is
    component f_adder
        port(
            x: in STD_LOGIC;
            y: in STD_LOGIC;
            z: in STD_LOGIC;
            s: out STD_LOGIC;
            c: out STD_LOGIC
        );
    end component;
    signal c2: std_logic_vector(0 to N);
    signal s1:std_logic_vector(0 to N-1);
    signal k:std_logic_vector(0 to N/4);
    signal skip:std_logic_vector(0 to N/4);

begin
    c2(0)<=cin;
    cout<=c2(N);
    H1:for i in 0 to (N/4 -1) generate
        s1(0+4*i)<= a(0+4*i) or b(0+4*i);
        s1(1+4*i)<= a(1+4*i) or b(1+4*i);
        s1(2+4*i)<= a(2+4*i) or b(2+4*i);
        s1(3+4*i)<= a(3+4*i) or b(3+4*i);
        skip(i)<= (s1(0+4*i) and s1(1+4*i) and
s1(2+4*i) and s1(3+4*i));
    end generate;

    H2:for i in 0 to (N/4 -1) generate
        k(i)<=skip(i) and c2(4*i);
    end generate;

    H3:for i in 1 to N/4 generate
        c2(4*i)<=(c2(4*i-1) or k(i-1));
    end generate;

    H4:for i in 0 to (N/4 -1) generate
        H5: for j in 0 to 3 generate
            A1: f_adder port map (a(4*i + j),b(4*i +
j),c2(4*i + j),s(4*i + j),c2(4*i +j +1));
        end generate;
    end generate;
end generate;

```

```
end carry_skip_adder;
```

Serial Adder

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

```

use IEEE.std_logic_unsigned.all;

entity seradder2 is
    port (
        clk,rst,sa,sb: in std_logic;
        perr,drdy: out std_logic;
        dout: out std_logic_vector(7 downto 0));
end seradder2;

architecture rtl of seradder2 is
    signal reg: std_logic_vector(7 downto 0);
    signal cnt: std_logic_vector(2 downto 0);
    signal cnt7_ff,cout_ff,sen,normal: std_logic;
    signal perra_ff,perrb_ff,pen,cen: std_logic;
begin
    regs: process
    begin
        wait until (clk'event and clk='1');
        if (rst='0') then
            reg<=(reg'range=>'0');
            cnt<=(cnt'range=>'0');
            normal<='1';
            perra_ff<='0';
            perrb_ff<='0';
            drdy<='0';
            pen<='0';
            cnt7_ff<='0';
            cout_ff<='0';
        else
            if (sen='1') then
                reg<=(cout_ff xor sa xor sb)&reg(7
downto 1);
                cout_ff<=((sa or sb) and cout_ff)
or (sa and sb);
            end if;
            if (perra_ff='1' or perrb_ff='1') then
                normal<='0';
            end if;
            if (cen='1') then
                cnt<=cnt+1;
            end if;
            if (pen='1') then
                perra_ff<=perra_ff xnor sa;
                perrb_ff<=perrb_ff xnor sb;
            end if;
            drdy<=pen;
            cnt7_ff<=cnt(2) and cnt(1) and cnt(0);
            pen<=cnt7_ff;
        end if;
    end process;
    sen<=normal and (cnt7_ff or cnt(2) or cnt(1) or
cnt(0));
    cen<='1' when (normal='1') and (pen='0') and
                (cnt7_ff='0') and (perra_ff='0')
and (perrb_ff='0') and

```

```

                                ((cnt="000" and sa='1' and sb='1')
or
                                (cnt/="000"))
                                else '0';
    perr<=perra_ff or perrb_ff;
    dout<=reg;
end rtl;

```

Serial multiplier

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

use IEEE.std_logic_unsigned.all;

entity sermult is
  generic
  (
    N: positive := 8
  );
  port (
    start,clk,rst: in std_logic;
    sa: in std_logic;
    b: in std_logic_vector(N-1 downto 0);
    drdy: out std_logic;
    dout: out std_logic_vector(2*N -1  downto 0)
  );
end sermult;

architecture rtl of sermult is
  component adder_n is
    generic(N:integer:=8);
    port (
      a,b:in std_logic_vector(N-1 downto 0);
      cin:in std_logic;
      s:out std_logic_vector(N-1 downto 0);
      cout:out std_logic);
  end component;

  -- internal signal declarations
  signal b_ff,and_out: std_logic_vector(N-1 downto 0);
  signal add_out: std_logic_vector(N downto 0);
  signal p_high_ff,p_low_ff: std_logic_vector(N-1 downto 0);
  signal counter: std_logic_vector(2 downto 0);      -- auto
thelei allagi an allaxei to N
  signal count_state: std_logic;
  signal zero: std_logic;
  begin
    and_gates: for k in 0 to N-1 generate
      and_out(k)<=sa and b_ff(k);
    end generate;

    zero<='0';
    fa8: adder_n port map
(a=>p_high_ff,b=>and_out,cin=>zero,s=>add_out(N-1 downto
0),cout=>add_out(N));

    count: process (start,clk,rst)
    begin
      if (rst='1') then
        counter<="000";
        count_state<='0';
      elsif (start='1') then
        count_state<='1';
        drdy<='0';
      elsif (clk'event and clk='1') then
        if (start='1') or (count_state='1') then
          counter<=counter+1;
          if (counter="111") then

```

```

                                count_state<='0';
                                drdy<='1';
                                end if;
                                end if;
                                end if;
end process;

b_reg: process (clk,rst)
begin
    if (rst='1') then
        b_ff<=(others=>'0');
    elsif (clk'event and clk='1') then
        b_ff<=b;
    end if;
end process;

par_reg: process (clk,rst)
begin
    if (rst='1') then
        p_high_ff<=(others=>'0');
    elsif (clk'event and clk='1') then
        p_high_ff<=add_out(N downto 1);
    end if;
end process;

shift_reg: process (clk,rst)
begin
    if (rst='1') then
        p_low_ff<=(others=>'0');
    elsif (clk'event and clk='1') then
        p_low_ff(N-2 downto 0)<=p_low_ff(N-1 downto
1);
        p_low_ff(N-1)<=add_out(0);
    end if;
end process;

dout<=p_high_ff&p_low_ff;
end rtl;

```

Booth conversion

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

entity booth_conv is
    generic (N: integer:=5);
    port (
        y:in std_logic_vector(N-1 downto 0);
        booth:inout std_logic_vector(N downto 0)
    );
end booth_conv;

architecture arch1 of booth_conv is

    function booth1(y_in : in std_logic_vector)
        return std_logic_vector is variable
    booth_out:std_logic_vector(N downto 0);
    begin
        for i in y 'range loop
            if ((y_in(i)='0') and (y_in(i+1)='0'))
                then booth_out(i):='0';

            elsif ((y_in(i)='1') and (y_in(i+1)='0'))
                then booth_out(i):='1';

            elsif ((y_in(i)='0') and (y_in(i+1)='1'))
                then booth_out(i):='Z';

            elsif ((y_in(i)='1') and (y_in(i+1)='1'))
                then booth_out(i):='0';

            end if;
        end loop;
        return booth_out;
    end booth1;

    signal y_in  : std_logic_vector(N downto 0);

begin
    y_in<=y&'0';
    booth<=booth1(y_in);

end arch1;

```

Booth cell

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

entity booth_cell is
    port (
        si: in STD_LOGIC;
        xi: in STD_LOGIC;
        pi: in STD_LOGIC;
        qi: in STD_LOGIC;
        ci: in STD_LOGIC;
        po: out STD_LOGIC;
        qo: out STD_LOGIC;
        so: out STD_LOGIC;
        xo: out STD_LOGIC;
        co: out STD_LOGIC
    );
end booth_cell;

architecture struct of booth_cell is
    component f_adder is
        port (
            x: in STD_LOGIC;
            y: in STD_LOGIC;
            z: in STD_LOGIC;
            s: out STD_LOGIC;
            c: out STD_LOGIC
        );
    end component;

    signal a,b: STD_LOGIC;

begin
    a <= pi and xi;
    b <= a xor qi;
    so <= b xor si xor ci;
    co <= (b and si) or (b and ci) or (si and ci);
    xo <= xi;
    po <= pi;
    qo <= qi;

end struct;

```

Booth multiplier 3x3

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

entity booth_multiplier is
  generic
  (
    NA : positive := 3
  );

  port (
    x: in STD_LOGIC_VECTOR(NA-1 downto 0);
    y: in STD_LOGIC_VECTOR(NA-1 downto 0);
    p: out STD_LOGIC_VECTOR(2*NA-2 downto 0)
  );
end booth_multiplier;

architecture booth_multiplier of booth_multiplier is

  signal yi : std_logic_vector(NA downto 0);

  subtype a_word is std_logic_vector(NA-1 downto 0);
  type a_word_array is array(natural range <>) of a_word;
  signal xi,xo,pi,po,qi,qo,si,so,ci,co : a_word_array(NA-1
downto 0);

  component booth_cell is
  port (
    si: in STD_LOGIC;
    xi: in STD_LOGIC;
    pi: in STD_LOGIC;
    qi: in STD_LOGIC;
    ci: in STD_LOGIC;
    po: out STD_LOGIC;
    qo: out STD_LOGIC;
    so: out STD_LOGIC;
    xo: out STD_LOGIC;
    co: out STD_LOGIC
  );
end component;

begin

gcd: for i in 0 to NA-1 generate
gca: for j in 0 to NA-1 generate
gc:      booth_cell port map
      (
        xi => xi(i)(j),
        pi => pi(i)(j),
        si => si(i)(j),
        ci => ci(i)(j),
        qi => qi(i)(j),
        xo => xo(i)(j),
        po => po(i)(j),
        so => so(i)(j),
        qo => qo(i)(j),
        co => co(i)(j)
      );

```

```

        end generate;
end generate;

yi<= y(NA-1 downto 0) & '0';

a: for i in 0 to NA-1 generate
    pi(i)(0)<=yi(i+1) xor yi(i);
    qi(i)(0)<=yi(i+1);
end generate;

b: for j in 0 to NA-1 generate
    xi(0)(j)<=x(j);
end generate;

c: for i in 1 to NA-1 generate
d:   for j in 0 to NA-1 generate
    xi(i)(j)<=xo(i-1)(j);
    end generate;
end generate;

e:for i in 0 to NA-1 generate
f:   for j in 1 to NA-1 generate
    ci(i)(j)<=co(i)(j-1);
    end generate;
end generate;

g: for i in 0 to NA-1 generate
    ci(i)(0)<=qi(i)(0);
    end generate;

k: for i in 0 to NA-1 generate
l:   for j in 1 to NA-1 generate
    pi(i)(j)<=po(i)(j-1);
    qi(i)(j)<=qo(i)(j-1);
    end generate;
end generate;

si(0)<=(others => '0');

m: for i in 1 to NA-1 generate
    si(i)<=so(i-1)(NA-1)&so(i-1)(NA-1 downto 1);
end generate;

p(2*NA-2 downto NA-1)<= so(NA-1)(NA-1 downto 0);

n: for i in 0 to NA-2 generate
    p(i)<=so(i)(0);
end generate;

end booth_multiplier;

```

Carry Propagate cell

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

entity mul_cell is
    port (
        ai: in STD_LOGIC;
        bi: in STD_LOGIC;
        si: in STD_LOGIC;
        ci: in STD_LOGIC;
        ao: out STD_LOGIC;
        bo: out STD_LOGIC;
        so: out STD_LOGIC;
        co: out STD_LOGIC
    );
end mul_cell;

architecture mul_cell of mul_cell is

    signal ab: STD_LOGIC;

begin
    ab <= ai and bi;
    so <= ab xor si xor ci;
    co <= (ab and si) or (ab and ci) or (si and ci);
    ao <= ai;
    bo <= bi;
end mul_cell;

```

Carry Propagate multiplier

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mul_u_cp is

```

```

generic
(
    NA : positive := 6;
    NB : positive := 4
);

port (
    a: in STD_LOGIC_VECTOR(NA-1 downto 0);
    b: in STD_LOGIC_VECTOR(NB-1 downto 0);
    p: out STD_LOGIC_VECTOR(NA+NB-1 downto 0)
);
end mul_u_cp;

architecture mul_u_cp of mul_u_cp is

    subtype a_word is std_logic_vector(NA-1 downto 0);
    type a_word_array is array(natural range <>) of
a_word;
    signal ai,ao,bi,bo,si,so,ci,co : a_word_array(NB-1
downto 0);

    component mul_cell
        port
        (
            ai : in std_logic;
            bi : in std_logic;
            si : in std_logic;
            ci : in std_logic;
            ao : out std_logic;
            bo : out std_logic;
            so : out std_logic;
            co : out std_logic
        );
    end component;

begin

    --cell generation
gcd: for i in 0 to NB-1 generate
gca: for j in 0 to NA-1 generate
gc:      mul_cell port map
        (
            ai => ai(i)(j),
            bi => bi(i)(j),
            si => si(i)(j),
            ci => ci(i)(j),
            ao => ao(i)(j),
            bo => bo(i)(j),
            so => so(i)(j),
            co => co(i)(j)
        );
        end generate;
    end generate;
end mul_u_cp;

```

```

        --intermediate wires generation
gasw: for i in 1 to NB-1 generate
        ai(i) <= ao(i-1);
        si(i) <= co(i-1)(NA-1) & so(i-1)(NA-1
downto 1);
        end generate;

gbciw:          for i in 0 to NB-1 generate
gbcjw:          for j in 1 to NA-1 generate
                bi(i)(j) <= bo(i)(j-1);
                ci(i)(j) <= co(i)(j-1);
            end generate;
        end generate;

        --input connections
gai: ai(0) <= a;
gsi: si(0) <= (others => '0');
gbi: for i in 0 to NB-1 generate
        bi(i)(0) <= b(i);
    end generate;
gci: for i in 0 to NB-1 generate
        ci(i)(0) <= '0';
    end generate;

        --output connections
gpa: p(NA+NB-1 downto NB) <= co(NB-1)(NA-1) & so(NB-
1)(NA-1 downto 1);
gpb: for i in 0 to NB-1 generate
        p(i) <= so(i)(0);
    end generate;

end mul_u_cp;

```

Carry Save cell

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mul_cell is
    port

```

```

        (
        ai:in std_logic;
        bi:in std_logic;
        si:in std_logic;
        ci:in std_logic;
        ao:out std_logic;
        bo:out std_logic;
        so:out std_logic;
        co:out std_logic
        );
end mul_cell;

architecture dataflow of mul_cell is
    signal ab : std_logic;

begin
    ab<=ai and bi;
    so<=ab xor si xor ci;
    co<=(ab and si) or (ab and ci) or (si and ci);
    ao<=ai;
    bo<=bi;
end dataflow;

```

Carry Save Multiplier

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mul_u_cs is
    generic

```

```

    (
    NA: positive := 6;
    NB: positive := 4
    );
    port
    (
    a: in std_logic_vector(NA-1 downto 0);
    b: in std_logic_vector(NB-1 downto 0);
    p: out std_logic_vector(NA+NB-1 downto 0)
    );
end mul_u_cs;

architecture struct of mul_u_cs is
component mul_cell
    port
    (
    ai, bi, si, ci: in std_logic;
    ao, bo, so, co: out std_logic
    );
end component;
component f_adder
    port (
    a, b, cin: in std_logic;
    s, cout: out std_logic
    );
end component;

subtype a_word is std_logic_vector(NA-1 downto 0);
type a_word_array is array(natural range <>) of a_word;

    signal ai, bi, si, ci, ao, bo, so, co: a_word_array(NB-1
downto 0);

    signal ll_a, ll_b, ll_cin, ll_s, ll_cout: a_word;

begin

11:      for i in 0 to NB-1 generate
12:          for j in 0 to NA-1 generate
13:              mul_cell port map(
                  ai => ai(i)(j),
                  bi => bi(i)(j),
                  si => si(i)(j),
                  ci => ci(i)(j),
                  ao => ao(i)(j),
                  bo => bo(i)(j),
                  so => so(i)(j),
                  co => co(i)(j));
              end generate;
          end generate;
14:      for i in 1 to NB-1 generate
          ai(i) <= ao(i-1);
          si(i) <= '0' & so(i-1)(NA-1 downto 1);
          end generate;

    ai(0) <= a;

```

```

        si(0) <= (others => '0');

15:        for i in 0 to NB-1 generate
16:            for j in 1 to NA-1 generate
                bi(i)(j) <= bo(i)(j-1);
            end generate;
            bi(i)(0) <= b(i);
        end generate;

17:        for i in 1 to NB-1 generate
18:            for j in 0 to NA-1 generate
                ci(i)(j) <= co(i-1)(j);
            end generate;
        end generate;
        ci(0) <= (others => '0');

185:       for i in NB-1 downto 0 generate
            p(i) <= so(i)(0);
        end generate;

19:        for i in NA-1 downto 0 generate
110:       f_adder port map (a => ll_a(i), b => ll_b(i),
cin => ll_cin(i),
                                s => ll_s(i), cout =>
ll_cout(i));
        end generate;

        ll_a <= '0' & so(NB-1)(NA-1 downto 1);
        ll_b <= co(NB-1)(NA-1 downto 0);

111:       for i in 1 to NA-1 generate
            ll_cin(i) <= ll_cout(i-1);
        end generate;
        ll_cin(0) <= '0';

        p(NA+NB-1 downto NB) <= ll_s;

end struct;

```

Wallace Multiplier

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_signed.all;

```

ENTITY wbm IS

GENERIC (n : INTEGER :=4 ; m : INTEGER :=4);

PORT(Input_x : IN STD_LOGIC_VECTOR (n-1 DOWNTO 0);
Input_y : IN STD_LOGIC_VECTOR (m-1 DOWNTO 0);
Result : OUT STD_LOGIC_VECTOR (m+n-1 DOWNTO 0));

END wbm;

ARCHITECTURE Behavioural OF wbm IS

TYPE matrix IS ARRAY (n/2 DOWNTO 0) OF STD_LOGIC_VECTOR (m+n-1
DOWNTO 0);

PROCEDURE Generate_PP_Booth (x : IN STD_LOGIC_VECTOR (n-1
DOWNTO 0);

y : IN STD_LOGIC_VECTOR (m-1 DOWNTO 0);
PP : INOUT matrix) IS

FUNCTION Adder_cla (input1, input2 : STD_LOGIC_VECTOR (m-1 DOWNTO
0);

carry_in : STD_LOGIC)
RETURN STD_LOGIC_VECTOR IS

VARIABLE carry : STD_LOGIC;
VARIABLE generat : STD_LOGIC_VECTOR (m-1 DOWNTO 0) ;
VARIABLE propagate : STD_LOGIC_VECTOR (m-1 DOWNTO 0) ;
VARIABLE sum : STD_LOGIC_VECTOR (m-1 DOWNTO 0) ;

BEGIN

carry := carry_in;
FOR i IN 0 TO (m-1) LOOP
generat (i) := input1(i) AND input2(i);
propagate(i) := input1(i) XOR input2(i);
sum (i) := propagate(i) XOR carry;
carry := generat(i) OR (propagate(i) AND carry);

```

END LOOP;
RETURN sum ;
END;

```

```

VARIABLE i, j, k    : INTEGER;
VARIABLE nb_pp      : INTEGER; -- number of partial products
VARIABLE Intermediate : STD_LOGIC_VECTOR (m-1 DOWNT0 0);
VARIABLE Null_Vector : STD_LOGIC_VECTOR (m-1 DOWNT0 0);
VARIABLE x_coded    : STD_LOGIC_VECTOR (n+1 DOWNT0 0);
VARIABLE Selector    : STD_LOGIC_VECTOR (3 DOWNT0 1);

BEGIN

FOR i IN 0 TO (n-1) LOOP x_coded(i+1) := x(i); END LOOP;

IF (n rem 2 = 0) THEN
    nb_pp := n/2;
ELSE
    nb_pp := n/2 + 1;
    x_coded(n+1) := x_coded(n);
END IF;

x_coded(0):='0';

FOR i IN 0 TO (m-1) LOOP Null_Vector(i) := '0'; END LOOP;

j := 0;
FOR i IN 0 TO (nb_pp-1) LOOP

    selector := x_coded(j+2) & x_coded(j+1) & x_coded(j);

    CASE Selector IS

        WHEN "000" => FOR k IN 0 TO (m+n-1) LOOP
            PP(i)(k) := '0';
        END LOOP;

        WHEN "001" => FOR k IN 0 TO (m-1) LOOP
            PP(i)(k) := y(k);
        END LOOP;

        FOR k IN m TO (m+n-1) LOOP
            PP(i)(k) := PP(i)(m-1);
        END LOOP;

        WHEN "010" => FOR k IN 0 TO (m-1) LOOP

```

```

        PP(i)(k) := y(k);
    END LOOP;

    FOR k IN m TO (m+n-1) LOOP
        PP(i)(k) := PP(i)(m-1);
    END LOOP;

WHEN "011" => PP(i)(0) := '0';
    FOR k IN 0 TO (m-1) LOOP
        PP(i)(k+1) := y(k);
    END LOOP;

    FOR k IN m+1 TO (m+n-1) LOOP
        PP(i)(k) := PP(i)(m);
    END LOOP;

WHEN "100" => FOR k IN 0 TO (m-1) LOOP
    Intermediate(k) := y(k) XOR '1';
END LOOP;

.
    Intermediate := Adder_cla (Intermediate, Null_Vector, '1');

    PP(i)(0) := '0';
    FOR k IN 0 TO (m-1) LOOP
        PP(i)(k+1) := Intermediate(k);
    END LOOP;

    FOR k IN m+1 TO (m+n-1) LOOP
        PP(i)(k) := PP(i)(m);
    END LOOP;

WHEN "101" => FOR k IN 0 TO (m-1) LOOP
    Intermediate(k) := y(k) XOR '1';
END LOOP;

    Intermediate := Adder_cla (Intermediate, Null_Vector, '1');

    FOR k IN 0 TO (m-1) LOOP
        PP(i)(k) := Intermediate(k);
    END LOOP;

    FOR k IN m TO (m+n-1) LOOP
        PP(i)(k) := PP(i)(m-1);
    END LOOP;

```

```

    WHEN "110" => FOR k IN 0 TO (m-1) LOOP
        Intermediate(k) := y(k) XOR '1';
    END LOOP;

    Intermediate := Adder_cla (Intermediate, Null_Vector,'1');

    FOR k IN 0 TO (m-1) LOOP
        PP(i)(k) := Intermediate(k);
    END LOOP;

    FOR k IN m TO (m+n-1) LOOP
        PP(i)(k) := PP(i)(m-1);
    END LOOP;

    WHEN "111" => FOR k IN 0 TO (m+n-1) LOOP
        PP(i)(k) := '0';
    END LOOP;

    WHEN OTHERS => NULL;

END CASE;

j:=j+2;
END LOOP;

END;

-----
-----

PROCEDURE Shiftleft_PP ( p    : IN    INTEGER ; -- numbers of bits to shift
                        Vector : INOUT STD_LOGIC_VECTOR (m+n-1 DOWNT0 0)) IS
    VARIABLE i : INTEGER;
    VARIABLE T : STD_LOGIC_VECTOR (m+n-1 DOWNT0 0);

BEGIN

    FOR i IN 0 TO (m+n-1) LOOP T(i)    := '0';    END LOOP;
    FOR i IN 0 TO (m+n-1-p) LOOP T(i+p) := Vector(i); END LOOP;
    FOR i IN 0 TO (m+n-1) LOOP Vector(i) := T(i);    END LOOP;

END;

```

```
-----
-----

PROCEDURE Wallace_Tree ( PP, Sum, Carry : INOUT matrix) IS
```

```

    PROCEDURE Bits_Full_Adder ( a, b, c  : IN  STD_LOGIC;
    Sum, Carry : OUT STD_LOGIC ) IS
    variable minus_a,minus_b,minus_c ,a1,b1,c1,d,e,f,g,h,k,l,m,n,o: std_logic;
```

```

BEGIN
    minus_a:= (a NAND a);
    minus_b:= (b NAND b);
    minus_c:= (c NAND c);
    a1:=(minus_a NAND minus_b);
    b1:= a NAND b;
    c1:= (minus_a NAND b);
    d:= (minus_b NAND a);
    e:= a1 NAND b1;
    f:= c1 NAND d;
    g:= e NAND c;
    h:= (f NAND minus_c);
    Sum:= g NAND h;

    k:= a NAND b;
    l:= b NAND c;
    m:= k NAND l;
    n:= m NAND m;
    o:= a NAND c;
    Carry:= n NAND o;

END;
```

```

    PROCEDURE Vectors_Full_Adder ( a, b, c : IN  STD_LOGIC_VECTOR (m+n-1
DOWNT0 0);
                                Sum   : OUT STD_LOGIC_VECTOR (m+n-1 DOWNT0 0);
                                Carry  : OUT STD_LOGIC_VECTOR (m+n-1 DOWNT0 0))
IS
```

```

BEGIN
    FOR i IN 2 TO (m+n-1) LOOP
        Bits_Full_Adder (a(i), b(i), c(i), sum(i), carry(i));
    END LOOP;
END;
```

TYPE Int_Vect IS ARRAY (n-1 DOWNT0 0) OF INTEGER;

VARIABLE nb_level : INTEGER;
VARIABLE num_level : INTEGER;
VARIABLE num_col : INTEGER;
VARIABLE nb_Vectors_Full_Adder : Int_Vect;
VARIABLE num_Vectors_Full_Adder : Int_Vect;
VARIABLE nb_output : Int_Vect;
VARIABLE vect : STD_LOGIC_VECTOR (m+n-1 DOWNT0 0);
VARIABLE nb_PP : INTEGER ;

BEGIN

IF (n rem 2 = 0) THEN
 nb_PP := n/2;
ELSE
 nb_PP := n/2 +1;
END IF;

IF nb_PP = 3 THEN nb_level := 1;
ELSIF nb_PP = 4 THEN nb_level := 2;
ELSIF nb_PP <=6 THEN nb_level := 3;
ELSIF nb_PP <=9 THEN nb_level := 4;
ELSIF nb_PP <=13 THEN nb_level := 5;
ELSIF nb_PP <=19 THEN nb_level := 6;
ELSIF nb_PP <=28 THEN nb_level := 7;
ELSIF nb_PP <=42 THEN nb_level := 8;
ELSIF nb_PP <=63 THEN nb_level := 9;
ELSE NULL;
END IF;

FOR num_level IN 1 TO nb_level LOOP

 nb_Vectors_Full_Adder (num_level) := nb_PP / 3 ;

 nb_output (num_level) := (nb_PP rem 3) + 2*nb_Vectors_Full_Adder
 (num_level);

 num_col := 0 ;
 FOR num_Vectors_Full_Adder IN 1 TO nb_Vectors_Full_Adder(num_level)
LOOP
 Vectors_Full_Adder (PP(num_col), PP(num_col+1), PP(num_col+2),
 Sum(num_Vectors_Full_Adder),
Carry(num_Vectors_Full_Adder));
 num_col := num_col + 3 ;
 END LOOP;

```

    FOR num_Vectors_Full_Adder IN 1 TO nb_Vectors_Full_Adder(num_level)
LOOP
    FOR i IN 0 TO (m+n-2) LOOP
        vect(i+1) := Carry(num_Vectors_Full_Adder)(i);
    END LOOP;
    carry (num_Vectors_Full_Adder) := vect;
    carry (num_Vectors_Full_Adder)(0) := '0' ;
END LOOP;

```

```

    num_col:=0 ;
    FOR num_Vectors_Full_Adder IN 1 TO nb_Vectors_Full_Adder(num_level)
LOOP
    PP (num_col) := Sum (num_Vectors_Full_Adder) ;
    PP (num_col+1) := Carry (num_Vectors_Full_Adder) ;
    num_col := num_col + 2 ;
END LOOP;

```

```

CASE (nb_PP rem 3) IS Vectors_Full_Adder
    WHEN 2 => PP(num_col) := PP(nb_PP-2) ;
                PP(num_col+1) := PP(nb_PP-1) ;
    WHEN 1 => PP(num_col) := PP(nb_PP-1) ;
    WHEN OTHERS => NULL ;
END CASE;

```

```

    nb_PP := nb_output(num_level) ;

```

```

END LOOP;

```

```

END;

```

```

-----
-----

```

```

PROCEDURE Addition_cla ( input1, input2 : IN STD_LOGIC_VECTOR (m+n-
1 DOWNT0 2);

```

```

    output      : OUT STD_LOGIC_VECTOR (m+n-1 DOWNT0 2))
IS

```

```

    VARIABLE carry      : STD_LOGIC;
    VARIABLE generat     : STD_LOGIC_VECTOR (m+n-1 DOWNT0 2) ;
    VARIABLE propagate   : STD_LOGIC_VECTOR (m+n-1 DOWNT0 2) ;
    VARIABLE sum         : STD_LOGIC_VECTOR (m+n-1 DOWNT0 2) ;

```

```

BEGIN

```

```

    carry := '0';
    FOR i IN 2 TO (m+n-1) LOOP
        generat(i) := input1(i) AND input2(i);
        propagate(i) := input1(i) XOR input2(i);
        sum (i) := propagate(i) XOR carry;
        carry := generat(i) OR (propagate(i) AND carry);
    END LOOP;
    output := sum ;
END;

```

```

-----
----- Main program -----
-----

```

BEGIN

Multiplication : PROCESS (Input_x, Input_y)

```

    VARIABLE x          : STD_LOGIC_VECTOR (n-1 DOWNT0 0);
    VARIABLE y          : STD_LOGIC_VECTOR (m-1 DOWNT0 0);
    VARIABLE Reduced_PP_1 : STD_LOGIC_VECTOR (m+n-1
DOWNT0 2);
    VARIABLE Reduced_PP_2 : STD_LOGIC_VECTOR (m+n-1
DOWNT0 2);
    VARIABLE Reduced_Final_Sum : STD_LOGIC_VECTOR (m+n-1
DOWNT0 2);
    VARIABLE Reduced_Output_Wallace_1 : STD_LOGIC_VECTOR (m+n-1
DOWNT0 2);
    VARIABLE Reduced_Output_Wallace_2 : STD_LOGIC_VECTOR (m+n-1
DOWNT0 2);
    VARIABLE PP          : matrix ;
    VARIABLE Sum, Carry  : matrix ;
    VARIABLE LSB0, LSB1  : STD_LOGIC ;
    VARIABLE nb_PP       : INTEGER ;

```

BEGIN

```

    x := Input_x ;
    y := Input_y ;

    ASSERT (n <= 126)
    REPORT "n must be inferior or equal to 126"
    SEVERITY Error;

    IF n=1 THEN
        FOR i IN 0 to (m-1) LOOP
            Result(i) <= x(0) AND y(i);
        END LOOP;
    
```

```

FOR i IN m to (m+n-1) LOOP
    Result(i) <= x(0) AND y(m-1);
END LOOP;

ELSE

    Generate_PP_Booth (x, y, PP);

    IF n=2 THEN

        Result <= PP(0);

    ELSE

        IF n rem 2 = 0 THEN nb_PP := n/2;
        ELSE nb_PP := n/2 +1;
        END IF;

        FOR i IN 0 TO (nb_PP-1) LOOP
            Shiftleft_PP (2*i, PP(i));
        END LOOP;

        LSB0 := PP(0)(0);
        LSB1 := PP(0)(1);

        IF n<=4 THEN

            FOR i IN 2 to (m+n-1) LOOP
                Reduced_PP_1 (i) := PP(0)(i);
            END LOOP;
            FOR i IN 2 to (m+n-1) LOOP
                Reduced_PP_2 (i) := PP(1)(i);
            END LOOP;
            Addition_cla (Reduced_PP_1, Reduced_PP_2, Reduced_Final_Sum);

        ELSE

            Wallace_tree (PP, Sum, Carry);

            FOR i IN 2 to (m+n-1) LOOP
                Reduced_Output_Wallace_1 (i) := Sum(1)(i);
            END LOOP;
            FOR i IN 2 to (m+n-1) LOOP
                Reduced_Output_Wallace_2 (i) := Carry (1)(i);
            END LOOP;
            Addition_cla (Reduced_output_wallace_1,
                Reduced_Output_Wallace_2,
                Reduced_Final_Sum);
        END IF;
    END IF;
END IF;

```

```

        END IF;
    Result <= (Reduced_Final_Sum & LSB1 & LSB0);

    END IF;

    END IF;
END PROCESS Multiplication;

END Behavioural;

```

Serialmul3xdown

```

library IEEE;
use IEEE.std_logic_1164.all;

entity serialmul3xd is
    generic(N: integer:=4);
    port(xin,clk,rst: in std_logic;

```

```

        a: in std_logic_vector(2*N-1 downto 0);
        R: in std_logic;
        pl,ph: out std_logic);
end serialmul3xd;

architecture serialmul3xd of serialmul3xd is
component f_adder
    port(a,b,cin: in std_logic;
        s,cout: out std_logic);
end component;
component dff
    port(d,clk,rst: in std_logic;
        q: out std_logic);
end component;
component mux4
    port(mux_in: in std_logic_vector(3 downto 0);
        sel: in std_logic_vector(1 downto 0);
        mux_out: out std_logic);
end component;
component mux2
    port(mux_in0,mux_in1: in std_logic;
        sel: in std_logic;
        mux_out: out std_logic);
end component;

signal fa,fad,mux_out,cout,cin,
        muxs1,muxs2,muxc,ds1,dc1: std_logic_vector(N-1 downto 0);
signal fadd,ds2,dc2: std_logic_vector(N downto 0);
signal x,x2,x3: std_logic;
signal ciout,ciin: std_logic;
signal cphout, cphin: std_logic;
signal mux_in: std_logic_vector(3 downto 0);
begin
f1:for k in 0 to N-1 generate
m1: mux4 port map(mux_in => mux_in,
                 sel=>a(2*k+1 downto 2*k),
                 mux_out=>mux_out(k));

end generate f1;
f2:for k in 1 to N-1 generate
fa1: f_adder port map(a=>mux_out(k), b=>fadd(k+1), cin=>cin(k), cout=>cout(k),
s=>fa(k));

```

Παράρτημα Β

Αναλυτικότερα τα εργαλεία του συστήματος

Στο κεφάλαιο αυτό περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν για την προσομοίωση των κυκλωμάτων και για τον υπολογισμό διαφόρων ενεργειακών και χρονικών παραμέτρων καθώς και παραδείγματα λειτουργίας αυτών των εργαλείων.

Το πρώτο εργαλείο που χρησιμοποιήσαμε είναι η έκδοση 6.0 του ModelSim που δίνει την δυνατότητα για την συγγραφή, αποσφαλμάτωση και προσομοίωση κυκλωμάτων που περιγράφονται σε VHDL. Τα βασικά χαρακτηριστικά του εργαλείου αυτού είναι:

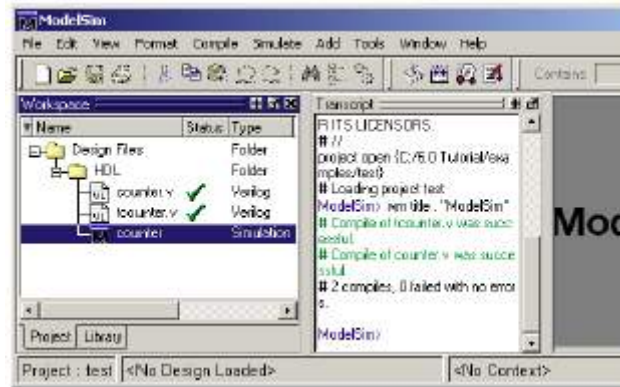
- Ευκολία στην χρήση λόγω του πρωτοποριακού γραφικού περιβάλλοντος.
- Υποστήριξη κώδικα σε VHDL , Verilog και SystemC ταυτόχρονα.
- Εκτεταμένη υποστήριξη αποσφαλμάτωσης
- Προηγμένο code coverage για βελτίωση του κώδικα
- Δυνατότητα βελτιστοποίησης σε επίπεδο πυλών
- Παρακολούθηση της χρησιμοποίησης της μνήμης και της κεντρικής μονάδας επεξεργασίας
- Δυνατότητα βελτιστοποίησης επίδοσης
- Σύγκριση κυματομορφών

Το δεύτερο πρόγραμμα που χρησιμοποιήσαμε είναι το Synopsys V2003.12 της ομώνυμης εταιρείας . Το Synopsys είναι ένα από ισχυρότερα εργαλεία για την μελέτη των ψηφιακών κυκλωμάτων , με τις δυνατότητες του να μην περιορίζονται μόνο σε αυτά. Μεταξύ άλλων περιέχει και τα παρακάτω εργαλεία :

- Design Analyzer
- Behavioral Compiler
- Astro
- FPGA Compiler
- Hercules
- Milkyway
- Scirocco
- System Studio
- SystemC Compiler
- HSPICE
- Power Management
- Design Budgeting
- Saber Designer

ModelSim

Προκειμένου να δούμε την λειτουργία των κυκλωμάτων με πραγματικά δεδομένα χρησιμοποιήσαμε την πλατφόρμα προσομοίωσης του ModelSim version 6.0 . Η συγκεκριμένη πλατφόρμα προσφέρει μια σειρά από εργαλεία που επιτρέπουν την συγγραφή, αποσφαλμάτωση και προσομοίωση κυκλωμάτων που περιγράφονται σε VHDL μέσα από ένα φιλικό προς τον χρήστη γραφικό περιβάλλον.



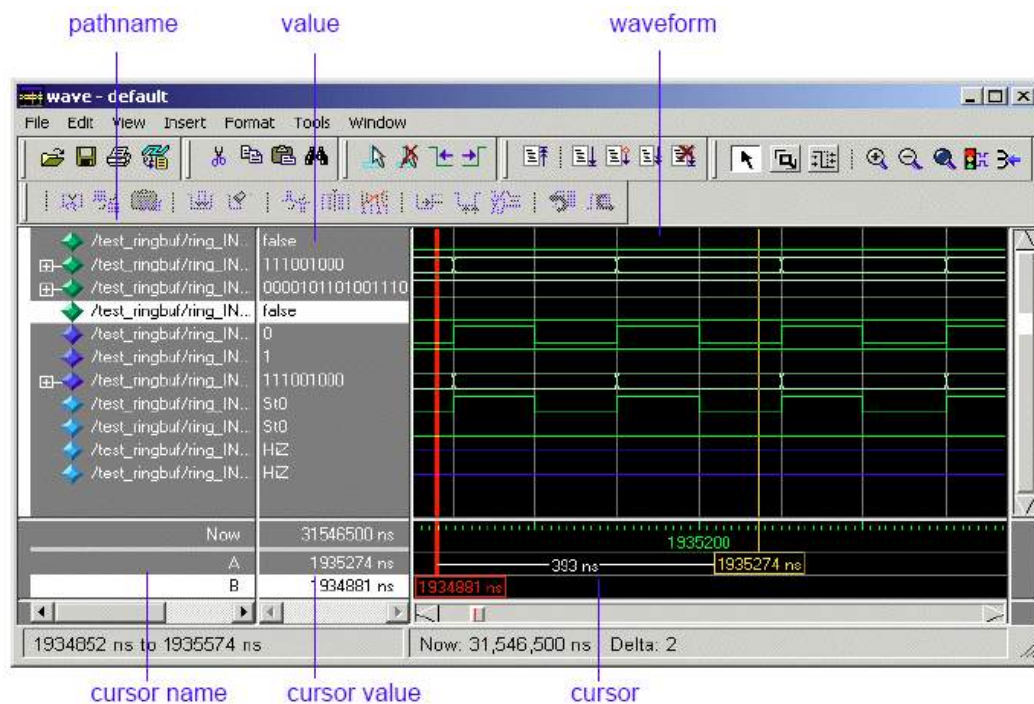
Σχήμα B.1 Χαρακτηριστικό γραφικό περιβάλλον του ModelSim

Οι βασικές εντολές του ModelSim είναι:

Compile: Με την εντολή αυτή ελέγχουμε το σχέδιο για τυχόν λάθη.

Simulate: Με την εντολή αυτή τρέχει μια προσομοίωση του σχεδίου μας. Μέσα στην διαδικασία της προσομοίωσης μας παρέχονται ένα πλήθος επιλογών.

- Εισαγωγή breakpoints για εντοπισμό τυχόν λαθών
- Επιλογή διέγερσης μέσα από διάφορες επιλογές όπως clock, constant , και counter μέσα από τον Waveform editor



Σχήμα B.2 Απεικόνιση του Wave Window

- Παρακολούθηση της διαδικασίας μέσα από το παράθυρο κυματομορφών (wave window) με την δυνατότητα εστίασης σε οποιοδήποτε χρονικό σημείο.

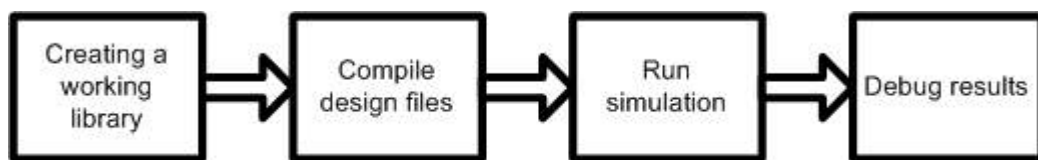
- Διερεύνηση των διασυνδέσεων του κυκλώματος μέσα από το Dataflow window με δυνατότητα παρουσίασης της ιεραρχίας.
- Ανάλυση της επίδοσης μέσα από το Profile window
- Παρουσίαση στατιστικών για την χρησιμοποίηση κομματιών του κώδικα
- Σύγκριση των κυματομορφών αλλά και των δεδομένων δύο κυκλωμάτων μέσω του Comparison wizard
- Παρακολούθηση της προσομοίωσης μέσω διαδραστικού animation που δείχνει τις καταστάσεις και τα διάφορα στάδια της προσομοίωσης μέσω του State Diagram Editor

Χρήση του ModelSim για τον carry save multiplier

Στο κεφάλαιο αυτό παρουσιάζεται η χρήση του ModelSim για τον πολλαπλασιαστή με σώσιμο κρατουμένου. Παρουσιάζονται οι απαραίτητες ρυθμίσεις καθώς και οι ενέργειες στις οποίες προβήκαμε ώστε να παραχθεί μέσω της προσομοίωσης ένα αρχείο VCD το οποίο θα τρέξει στο περιβάλλον του Synopsys μέσω του PrimePower για να μας δώσει τις απαραίτητες πληροφορίες όσον αφορά τα θέματα κατανάλωσης ενέργειας .

Δημιουργία καινούργιου project

Στο παρακάτω σχήμα φαίνεται ένα διάγραμμα ροής που δείχνει τα βήματα που πρέπει να γίνουν προκειμένου έχουμε μια επιτυχή προσομοίωση.



Σχήμα B.3 Διάγραμμα ροής για προσομοίωση

Για να δημιουργήσουμε ένα νέο project στο ModelSim πρέπει να:

- Δημιουργήσουμε μια βιβλιοθήκη στην οποία θα εργαζόμαστε
- Δημιουργήσουμε ένα νέο project
- Προσθέσουμε στο καινούργιο project τα αρχεία του πολλαπλασιαστή

Έτσι η διαδικασία είναι η εξής:

Επιλέγουμε “File >New >Library ” δίνουμε ένα όνομα, εν προκειμένω work, και πατάμε OK .Από εδώ και πέρα όλα τα καινούργια project θα προστίθεται σε αυτήν την βιβλιοθήκη.

Στην συνέχεια επιλέγουμε “File >New >Project ”ώστε να δημιουργήσουμε ένα καινούργιο project. Του δίνουμε το όνομα carry_save και επιλέγουμε ως βιβλιοθήκη εργασίας την work που φτιάξαμε στο προηγούμενο βήμα.

Έπειτα αρχίζουμε να προσθέτουμε αρχεία στο project, μέσω του “Add existing files”.

Τα αρχεία που προσθέτουμε εμείς είναι

- ο Τα synthesizable VHDL αρχεία του full adder, του κυττάρου του πολλαπλασιαστή mul_cell και του πολλαπλασιαστή ολόκληρου mul_u_cs που έχουμε φτιάξει με το Synopsys
- ο Τα κανονικά VHDL αρχεία full_adder, mul_cell και mul_u_cp.
- ο Το testbench αρχείο του πολλαπλασιαστή με σώσιμο κρατουμένου, το οποίο μας επιτρέπει να παίρνουμε τις εισόδους του πολλαπλασιαστή από αρχείο text και να βγάζει τα αποτελέσματα σε αρχείο text.
- ο Τα δύο αρχεία text με τις 200.000 εισόδους για την προσομοίωση, καθώς και το αρχείο στο οποίο μετά την προσομοίωση θα γραφούν τα 200.000 αποτελέσματα
- ο Τα αρχεία με τα οποία περιγράφει το Synopsys τις λογικές πύλες τις οποίες χρησιμοποιούμε στο σχέδιό μας.

Compilation του project.

Στην συνέχεια επιλέγουμε μέσω του “Compile >Compile Order” την σειρά με την οποία θα κάνουμε compile τα αρχεία. Ξεκινάμε από το πιο χαμηλό στην ιεραρχία (αρχεία λογικών πυλών) και καταλήγουμε στο υψηλότερο και πολυπλοκότερο (testbench).

Μετά επιλέγουμε “Compile >Compile” γίνεται το compilation όλων των αρχείων τα οποία συνδέονται στην βιβλιοθήκη εργασίας work.

Simulation

Προκειμένου να γίνει η προσομοίωση χρειάζεται να φορτώσουμε το testbench αρχείο από την βιβλιοθήκη work και να πατήσουμε “Simulation >Start simulation”.

Μέσω του μενού “View” μπορούμε να επιλέξουμε να βλέπουμε στην οθόνη και το Debug Window και το Wave Window με τις κυματομορφές.

Επειδή θέλουμε να περάσουμε την όλη διαδικασία της προσομοίωσης σε ένα αρχείο VCD οι ενέργειες που κάνουμε είναι να πληκτρολογήσουμε στο transcript:

```
vcd file marios.vcd -- δημιουργία του VCD αρχείου marios
vcd add -r * --καθορισμός του τι θα αποθηκεύσουμε στο αρχείο
run -all --έναρξη προσομοίωσης
vcd checkpoint --πέρασμα της διαδικασίας στο VCD αρχείο
quit -sim --έξοδος από την προσομοίωση
```

Στο επόμενο σχήμα φαίνεται μια χαρακτηριστική απεικόνιση του περιβάλλοντος προσομοίωσης:

Synopsys

Τα δύο εργαλεία τα οποία εμείς χρησιμοποιήσαμε είναι ο Design Compiler και το Prime Power.

Design Compiler

Ο Design Compiler είναι ένα εργαλείο το οποίο μας δίνει την δυνατότητα να εισάγουμε ένα αρχείο σε VHDL μορφή και μέσα από ένα σύνολο εντολών να δούμε, μεταξύ άλλων, χρονικές παραμέτρους. Τα κυριότερα χαρακτηριστικά του Design Compiler είναι:

- Χειρισμός μέσα από αρχεία script, δυνατότητα ελέγχου λαθών, δυνατότητα περάσματος των αποτελεσμάτων σε αρχεία εξόδου και έλεγχος ροής με loops και if statements μέσα από το Command-Line
- Παρατήρηση της καθυστέρησης σε οποιοδήποτε μονοπάτι του κυκλώματος και δυνατότητα επιβολής χρονικών περιορισμών σε μονοπάτια.
- Παρατήρηση της καθυστέρησης σε κάθε στοιχείο του κυκλώματος και δυνατότητα επιβολής μέγιστης και ελάχιστης καθυστέρησης.
- Παρατήρηση του χρόνου μετάβασης και δυνατότητα επιβολής μέγιστου χρόνου μετάβασης
- Παρατήρηση της χωρητικότητας και δυνατότητα επιβολής μέγιστης και ελάχιστης χωρητικότητας
- Παρατήρηση της χρησιμοποίησης των πόρων και δυνατότητα επιβολής ελάχιστης χρησιμοποίησης
- Δυνατότητα δημιουργίας ρολογιού με βάση την ανάγκη του σχεδιαστή
- Δυνατότητα Register Retiming καθώς και επιλογή για pipeline
- Δυνατότητα βελτιστοποίησης του σχεδίου

Οι εντολές που χρησιμοποιήσαμε στην ανάλυση του κυκλώματος μαζί με τις λειτουργίες που αυτές επιτελούν είναι:

- set search_path: Ορίζει το μονοπάτι στο οποίο βρίσκονται όλα τα αρχεία vhd, μέσα από το οποίο τρέχουμε το dc_shell
- set link_library: Ορίζει την λίστα των αρχείων του σχεδίου που θα χρησιμοποιηθεί κατά την διαδικασία του link
- set target_library: Ορίζει την λίστα των τεχνολογικών βιβλιοθηκών εξαρτημάτων που θα χρησιμοποιηθούν κατά το compilation του σχεδίου
- set symbol_library: Ορίζει την βιβλιοθήκη συμβόλων που θα χρησιμοποιηθεί κατά την σχηματική δημιουργία
- set: Δημιουργεί καινούργιες μεταβλητές
- analyze -format vhd: Η εντολή analyze αναλύει HDL αρχεία και αποθηκεύει την ενδιάμεση δομή σε μια συγκεκριμένη βιβλιοθήκη
- elaborate: Κατασκευάζει ένα κύκλωμα από το ενδιάμεσο στάδιο μιας VHDL περιγραφής
- compile: Πραγματοποιεί σύνθεση και βελτιστοποίηση ενός σχεδίου σε λογικό επίπεδο και επίπεδο πυλών

- uniquify: Ψάχνει για αναφορές σχεδίων που έχουν το ίδιο όνομα στην ιεραρχία του τρέχοντος σχεδίου και δημιουργεί νέα σχέδια ,με μοναδικό όνομα για το καθένα
- link: Επιλύει σχεδιαστικές παραπομπές. Για ολοκληρωθεί ένα σχέδιο πρέπει να συνδεθούν όλα τα επιμέρους σχέδια και τα συστατικά των βιβλιοθηκών και έτσι το σχέδιο θα είναι λειτουργικό. Η δουλειά αυτής της εντολής είναι να εντοπίσει όλες αυτές τις αναφορές και να τις συνδέσει.
- report_cell: Παρουσιάζει πληροφορίες και στατιστικά για τα κύτταρα του τρέχοντος σχεδίου
- report_area: Παρουσιάζει πληροφορίες και στατιστικά για τον χώρο που καταλαμβάνει το συνδυαστικό μέρος, το μη-συνδυαστικό μέρος και το σχέδιο συνολικά. Επίσης δείχνει τον αριθμό των εισόδων, των εξόδων, των αναφορών και των δικτύων του σχεδίου.
- report_timing: Παρουσιάζει πληροφορίες για τον χρονισμό και τον χρόνο ενός σχεδίου.

Χρήση του Synopsys για κανονικά vhdl αρχεία

Στο κεφάλαιο αυτό παρουσιάζεται η χρήση του Synopsys για τον παράλληλο αθροιστή με πύλες NAND_NOR. Παρουσιάζονται οι απαραίτητες ρυθμίσεις καθώς και οι ενέργειες στις οποίες προβήκαμε ώστε να αποκτήσουμε στατιστικά και πληροφορίες σχετικά με τον χώρο, τα κύτταρα και τον χρόνο.

Αρχικά δημιουργούμε το script αρχείο με το όνομα addern_nand το οποίο φαίνεται παρακατω

```
set search_path /home/kanagno/marios/lab/addern_NAND
set link_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set symbol_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2.s
db
set target_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db

analyze -format vhdl nand_nor.vhd
analyze -format vhdl adder_n_nand.vhd
elaborate adder_n
uniquify
compile
set
report_cell
report_area
report_timing
```

Στο παραπάνω αρχείο επιλέγουμε τις βιβλιοθήκες του Synopsys που θα μας προμηθεύσουν με τα κατάλληλα αρχεία , ενώ έχουμε την δυνατότητα να συμπεριλάβουμε και τις εντολές που χρειάζονται για το σωστό compilation του σχεδίου (analyze, elaborate) αλλά και τις εντολές που δηλώνουν τις ενέργειες που επιθυμούμε να κάνουμε (report_cell,report_area κ.τ.λ)

Αφού το σώσουμε επιλέγουμε από το τρέχων μονοπάτι “ dc_shell –tcl_mode” και μπαίνουμε στον Design Compiler.
Από εκεί επιλέγουμε dc_shell >source addern_nand.scr και ο Design Compiler μας δίνει τις παρακάτω πληροφορίες

```
dc_shell-t> source addern_nand.scr
Loading db file '/usr/synopsys/libraries/syn/standard.sldb'
Loading db file '/usr/synopsys/libraries/syn/gtech.db'
Loading db file
'/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db'
Reading in the Synopsys vhd1 primitives.
/home/kanagno/marios/lab/addern_NAND/nand_nor.vhd:
/home/kanagno/marios/lab/addern_NAND/adder_n_nand.vhd:
Information: Building the design 'f_adder'. (HDL-193)
Current design is now 'adder_n'
  Uniquifying cell 'A1_3' in design 'adder_n'. New design is 'f_adder_0'.
  Uniquifying cell 'A1_2' in design 'adder_n'. New design is 'f_adder_1'.
  Uniquifying cell 'A1_1' in design 'adder_n'. New design is 'f_adder_2'.
  Uniquifying cell 'A1_0' in design 'adder_n'. New design is 'f_adder_3'.
Information: Evaluating DesignWare library utilization. (UISN-27)

=====
=====
| DesignWare Building Block Library | Version | Available |
=====
=====
| Basic DW Building Blocks          | 2003.12-DWF_0312 | * |
| Licensed DW Building Blocks       |                  |   |
=====
=====
Loading target library 'umcl18u250t2_typ'
Loading design 'adder_n'
Information: Design 'adder_n' has no optimization constraints set. (OPT-108)
Beginning Mapping Optimizations (Medium effort)
-----
Structuring 'f_adder_0'
Mapping 'f_adder_0'
Structuring 'f_adder_1'
Mapping 'f_adder_1'
Structuring 'f_adder_2'
Mapping 'f_adder_2'
Structuring 'f_adder_3'
Mapping 'f_adder_3'
```

Beginning Delay Optimization Phase

ELAPSED TIME	AREA	WORST NEG SLACK	TOTAL NEG SLACK	DESIGN RULE COST	ENDPOINT
-----------------	------	--------------------	--------------------	---------------------	----------

Beginning Area-Recovery Phase (cleanup)

ELAPSED TIME	AREA	WORST NEG SLACK	TOTAL NEG SLACK	DESIGN RULE COST	ENDPOINT
-----------------	------	--------------------	--------------------	---------------------	----------

0:00:04	357.8	0.00	0.0	0.0	
---------	-------	------	-----	-----	--

Optimization Complete

Transferring design 'adder_n' to database 'adder_n.db'

Current design is 'adder_n'.

Information: Updating design information... (UID-85)

Report : cell

Design : adder_n

Version: V-2003.12

Date : Thu Dec 9 11:37:54 2004

Attributes:

b - black box (unknown)

h - hierarchical

n - noncombinational

r - removable

u - contains unmapped logic

Cell	Reference	Library	Area	Attributes
A1_0	f_adder_3		89.440002	h
A1_1	f_adder_2		89.440002	h
A1_2	f_adder_1		89.440002	h
A1_3	f_adder_0		89.440002	h

Total 4 cells 357.760010

```

Report : area
Design : adder_n
Version: V-2003.12
Date   : Thu Dec  9 11:37:54 2004
*****

Library(s) Used:

    umcl18u250t2_typ (File:
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db)

Number of ports:      14
Number of nets:       17
Number of cells:      4
Number of references:  4

Combinational area:   357.759979
Noncombinational area: 0.000000
Net Interconnect area: undefined (No wire load specified)

Total cell area:      357.760010
Total area:           undefined

*****

Report : timing
    -path full
    -delay max
    -max_paths 1
Design : adder_n
Version: V-2003.12
Date   : Thu Dec  9 11:37:54 2004
*****

Operating Conditions:
Wire Load Model Mode: top

Startpoint: b[0] (input port)
Endpoint: cout (output port)
Path Group: (none)
Path Type: max

Point              Incr    Path
-----
input external delay 0.00    0.00 r
b[0] (in)           0.00    0.00 r
A1_0/y (f_adder_3)  0.00    0.00 r
A1_0/U9/Z (EXOR2D1) 0.16    0.16 f
A1_0/U8/Z (AO22D1)  0.15    0.31 f
A1_0/c (f_adder_3)  0.00    0.31 f

```

A1_1/z (f_adder_2)	0.00	0.31 f
A1_1/U8/Z (AO22D1)	0.16	0.47 f
A1_1/c (f_adder_2)	0.00	0.47 f
A1_2/z (f_adder_1)	0.00	0.47 f
A1_2/U8/Z (AO22D1)	0.16	0.63 f
A1_2/c (f_adder_1)	0.00	0.63 f
A1_3/z (f_adder_0)	0.00	0.63 f
A1_3/U8/Z (AO22D1)	0.13	0.76 f
A1_3/c (f_adder_0)	0.00	0.76 f
cout (out)	0.00	0.76 f
data arrival time		0.76

(Path is unconstrained)		

Χρήση του Synopsys με synthesizable αρχεία

Στο κεφάλαιο αυτό παρουσιάζεται η χρήση του ModelSim για τον πολλαπλασιαστή με διάδοση κρατούμενου. Παρουσιάζονται οι απαραίτητες ρυθμίσεις καθώς και οι ενέργειες στις οποίες προβήκαμε ώστε να παραχθούν τα απαραίτητα synthesizable αρχεία τα οποία ,μέσω της προσομοίωσης του ModelSim θα δώσουν ένα αρχείο VCD που θα τρέξει αργότερα στο PrimePower για να μας δώσει τις απαραίτητες πληροφορίες όσον αφορά τα θέματα κατανάλωσης ενέργειας . Προκειμένου να παραχθεί το synthesizable αρχείο του πολλαπλασιαστή πρέπει να γίνει βήμα-βήμα παραγωγή αρχικά του synthesizable του f_adder και στην συνέχεια synthesizable του κυττάρου του πολλαπλασιαστή.

Όλα τα προαναφερθέντα θα γίνουν μέσα από το σταδιακό source των παρακάτω αρχείων :

Αρχείο test1.scr

```
set search_path /home/kanagno/Desktop/fresh_start
set link_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set target_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set symbol_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2.s
db
analyze -format vhdl nand_nor.vhd
elaborate f_adder
compile

write -format vhdl -output syntesizednand.vhd
```

Αρχείο test2.scr

```
set search_path /home/kanagno/Desktop/fresh_start
set link_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set target_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set symbol_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2.s
db
analyze -format vhdl syntesizednand.vhd
elaborate f_adder
analyze -format vhdl mul_cell.vhd
elaborate mul_cell
link
uniquify
compile
write -format vhdl -output synthesizedmull_cell.vhd
```

Αρχείο test3.scr

```
set search_path /home/kanagno/Desktop/fresh_start
set link_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set target_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2_t
yp.db
set symbol_library
/usr/synopsys/umc_libraries/UMCL18U250D2_2.4/design_compiler/umcl18u250t2.s
db
analyze -format vhdl synthesizedmull_cell.vhd
elaborate mul_cell
analyze -format vhdl mul_u_cp.vhd
elaborate mul_u_cp
link
uniquify
compile
write -format vhdl -output synthesizedmul_u_cp.vhd
```

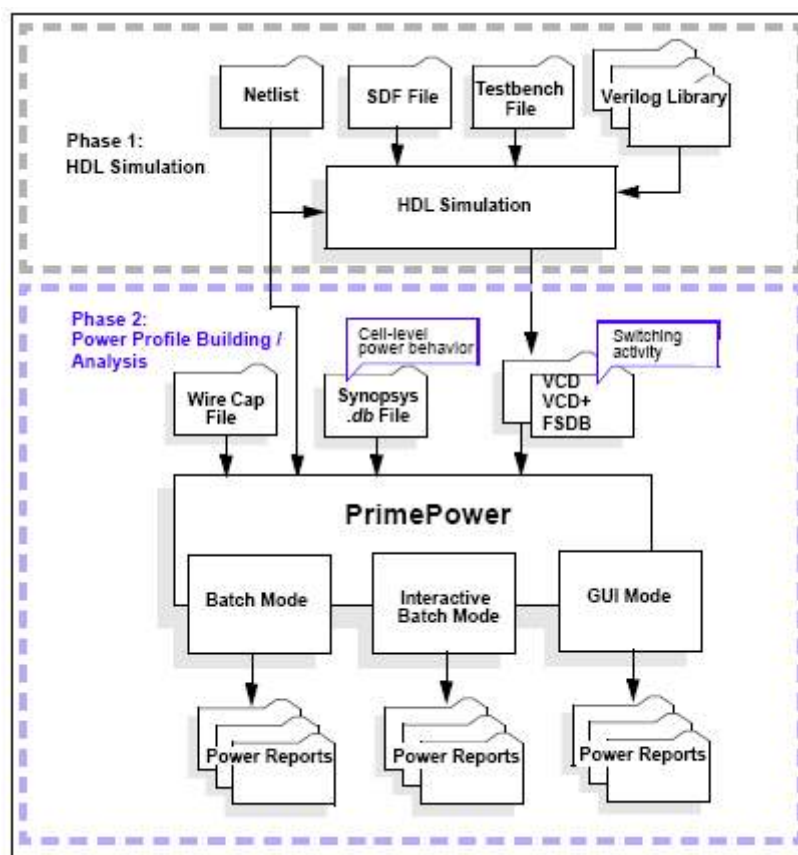
PRIME POWER

Το Prime Power είναι ένα εργαλείο του Synopsys που μας δίνει την δυνατότητα να κάνουμε μια ακριβή δυναμική ενεργειακή ανάλυση σε ένα περιβάλλον υψηλού επιπέδου ASIC (Application specific integrated circuit) σχεδίων.

Το Prime Power μπορεί εν μέσω άλλων να:

- Μοντελοποιήσει εναλλαγές χωρητικότητας, δεδομένα ενέργειας για βραχυκυκλώματα και δεδομένα για στατική ενέργεια.
- Προσομοιώσει πολύπλοκες πύλες και megacells.
- Πάρει υπόψην του στιγμιαίες εξαρτήσεις κυττάρων, βλάβες, πολλαπλά φορτία καθώς και μη- γραμμικά ενεργειακά προφίλ από κάποια κύτταρα σε ορισμένες συνθήκες.
- Παράσχει ιεραρχική πλοήγηση σε επίπεδο κυττάρου, μπλοκ και πλήρους τσιπ.

Στο παρακάτω σχήμα φαίνεται το διάγραμμα ροής της ενεργειακής προσομοίωσης του Prime Power :



Σχήμα Β.5 Διάγραμμα ροής της ενεργειακής προσομοίωσης του Prime Power

Βιβλιογραφία

- [1] LUCA BENINI, GIOVANNI DE MICHELI AND ENRICO MACII 2001, Designing Low –Power Circuits : Practical Recipes
- [2] Κ.Ζ.ΠΕΚΜΕΣΤΖΗ 2003 , Ψηφιακά Συστήματα VLSI
- [3] Ν.Η.ΥΕΣΤΕ ΑΝΔ ΕΣΗΡΑΓΙΑΝ (Μετάφραση- Επιμέλεια Κ.Ζ.Πεκμεστζί , Δ. Σουντρής , Κ. Γκούτης) 1996 , Σχεδίαση Ολοκληρωμένων Κυκλωμάτων CMOS VLSI
- [4] JAN.M.RABAEY , ANATHA CHANDRAKASAN AND BORIVOJE NIKOLIC 1996 , Digital Integrated Circuits
- [5] CHRISTIAN PIGUET 2000 , Low – Power Electronics Design
- [6] Μ.ΜΟΡΡΙΣ ΜΑΝΟ 1992, Ψηφιακή Σχεδίαση