Ανάλυση καθυστέρησης και κατανάλωσης ισχύος εναλλακτικών διατάξεων αθροιστών.

Διπλωματική Εργασία :

Δαδαλιάρης Αντώνης

Σεπτέμβριος, 2005

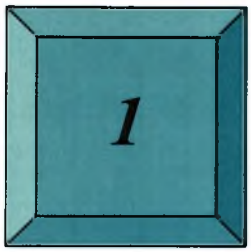
Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή του τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων και βασικό επιβλέποντα της πτυχιακής αυτής εργασίας κ.Γεώργιο Σταμούλη που μου έδωσε την ευκαιρία να πραγματοποιήσω αυτή την μελέτη. Η υποστήριξή του, η αμέριστη συμπαράστασή του, αλλά και οι διαρκείς και εύστοχες υποδείξεις του βοήθησαν στην έγκαιρη ολοκλήρωση αυτής της μελέτης.

Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον έταιρο επιβλέποντα καθηγητή και επισκέπτη καθηγητή στο τμήμα, κ.Νέστορα Ευμορφόπουλο για τις συμβουλές του στο μαθηματικό υπόβαθρο που χρησιμοποιήθηκε στις μετρήσεις που παρουσιάζονται και τον διδακτορικό φοιτητή του τμήματος Δημήτρη Καραμπατζάκη για την προγενέστερη δουλεία του στα εργαλεία που χρησιμοποιήθηκαν και την μεταλαμπάδευση της γνώσης αυτής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που μου συμπαραστάθηκαν σε όλη την διάρκεια της εκπόνησης αυτής της εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ



1 Εισαγωγή

1.1 Περιγραφή του Προβλήματος

Η έρευνα πάνω στον τομέα των ψηφιακών κυκλωμάτων μέσα στο πέρασμα των δεκαετιών, από την κατασκευή του πρώτου ηλεκτρονικού υπολογιστή μέχρι και σήμερα, έχει αναθεωρήσει τους βασικούς στόχους και σκοπούς της ουκ ολίγες φορές.

Αρχικά, βασικό μέλημα των επιστημόνων ήταν η λειτουργικότητα των κυκλωμάτων, μεταγενέστερα τέθηκε ως θέμα το υλικό κατασκευής των κυκλωμάτων και η «έκταση» που αυτά καταλαμβάνουν και μέχρι και σήμερα ένα από τα σημαντικότερα ζητήματα, είναι η ταχύτητα απόκρισής τους.

Την τελευταία εικοσαετία, όμως, με την κατακόρυφη αύξηση της χρήσης φορητών (και δη ασυρμάτων) συσκευών, έχει παρατηρηθεί μια αυξανόμενη απαίτηση για σχεδιάσεις χαμηλής κατανάλωσης ισχύος. Η απαίτηση αυτή πηγάζει από την γενικότερη απαίτηση για φορητότητα που συνεπάγεται μεγαλύτερη διάρκεια ζωής της μπαταρίας που χρησιμοποιεί η εκάστοτε συσκευή.

Μια από τις βασικότερες απαιτήσεις κατά την διαδικασία σχεδίασης ενός ψηφιακού κυκλώματος είναι να γνωρίζουμε πόση ισχύ πρόκειται να καταναλώσει, σε άμεση συνάρτηση πάντα με την εφαρμογή στην οποία θα χρησιμοποιηθεί. Έτσι, αφού ο σχεδιαστής γράψει τον κατάλληλο κώδικα, και λαμβάνοντας υπόψη του όλες τις τεχνικές προδιαγραφές που του έχουν δωθεί,

καλείται να υπολογίσει την ισχύ που καταναλώνεται, προκειμένου να επιβεβαιώσει ότι η εκάστοτε σχεδίαση ανταποκρίνεται στις δεδομένες απαιτήσεις. Αυτό γίνεται έτσι ώστε να μην έχουμε λάθη ή προβλήματα κατά την τελική κατασκευή του ολοκληρωμένου κυκλώματος.

Γίνεται, επομένως, εύκολα αντιληπτό το πόσο μεγάλης σημασίας είναι η ακριβής μέτρηση της ισχύος που καταναλώνεται σε ένα κύκλωμα. Αυτός είναι άλλωστε και ο λογος που οδήγησε πολλές εταιρείες στην ανάπτυξη εργαλείων για την εκπόνηση της εργασίας αυτής. Τα περισσότερα, μάλιστα, από αυτά στοχεύουν τόσο στην μείωση της καταναλισκόμενης ισχύος όσο και στην βελτιστοποίηση των υπολοίπων παραμέτρων που αναφέραμε παραπάνω (χώρος που καταλαμβάνει το κύλωμα, καθυστέρηση απόκρισης).

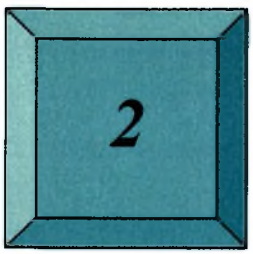
1.2 Στόχος της Εργασίας

Βασικός στόχος αυτής της εργασίας είναι ο υπολογισμός της καθυστέρησης που παρουσιάζει και της ισχύος που καταναλώνει ένα ευρή σύνολο αθροιστών οι οποίοι χρησιμοποιούνται στην πλειοψηφία των ψηφιακών σχεδιάσεων.

Ο επιμέρους υπολογισμός των ποσοτήτων που μολις αναφέραμε γίνεται με χρήση των εργαλείων της Synopsys, και πιο συγκεκριμένα με χρήση του Design Vision και του PrimePower. Η επιλογή των εργαλείων αυτών έγινε λόγω του ότι η Synopsys θεωρείται μια από τις μεγαλύτερες εταιρείες ανάπτυξης λογισμικού για προσομοίωση, σύνθεση και έλεγχο ψηφιακών κυκλωμάτων. Προσφέρει, μάλιστα, ολοκληρωμένες λύσεις στο χώρο αυτό με την παροχή ενός πακέτου που αποτελείται από τον Design Compiler και τον Power Compiler των

οποίων «υποσύνολα» είναι τα εργαλεία που χρησιμοποιήθηκαν σε αυτήν την εργασία.

Τέλος, με την παρουσίαση των συγκριτικών αποτελεσμάτων οδηγούμαστε σε μια πληθώρα συμπερασμάτων που επιβεβαιώνουν ή διαψεύδουν «εδραιομένες» απόψεις που επικρατούν εως και σήμερα.



Προσομοίωση και Σύνθεση Αθροιστών

2.1 Αθροιστές

Η πρόσθεση δυο δυαδικών αριθμών είναι μια από τις πράξεις με την μεγαλύτερη συχνότητα εμφάνισης στις περισσότερες ψηφιακές σχεδιάσεις. Ένας αθροιστής δεν χρησιμοποιείται μόνο όταν πραγματοποιείται η πράξη της πρόσθεσης ή της αφαίρεσης, αλλά και σε περιπτώσεις όπου απαιτούνται πιο πολύπλοκες πράξεις όπως ο πολλαπλασιασμός ή η διαίρεση δυαδικών αριθμών.

Γίνεται, επομένως, αντιληπτό γιατί έγινε η επιλογή να χρησιμοποιηθούν τα συγκεκριμένα κυκλώματα. Οι αθροιστές που χρησιμοποιήθηκαν εμπίπτουν ως επί τω πλείστον σε δυο μεγάλες καηγορίες : τους Carry-Propagate Adders (CPA) και τους Tree Adders.

Στην 1η κατηγορία ανήκουν οι :

- Carry Ripple Adder
- Carry Skip Adder
- Carry Lookahead Adder
- Carry Save Adder
- Carry Select Adder
- Manchester Carry Chain Adder

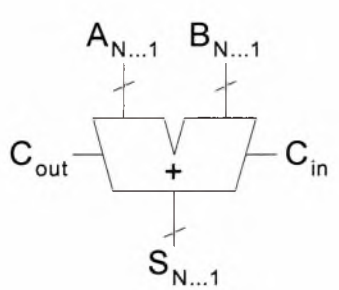
ενώ στην 2η κατηγορία ανήκουν οι ακόλουθοι :

- Han-Carlson Adder
- Elm Adder
- Sklansky Adder

Ακολούθως παρατείθεται η περιγραφή των χαρακτηριστικών των παραπάνω κατηγοριών και των επιμέρους αθροιστών.

Carry-Propagate Adders (CPA):

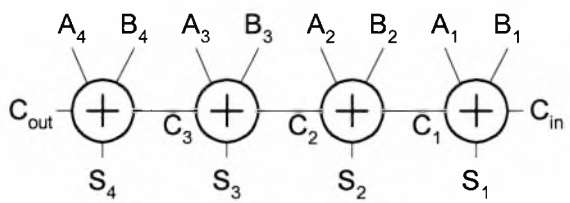
Οι αθροιστές αυτοί παίρνουν ως εισόδους τα δυαδικά διανύσματα-αριθμούς $A_N....A_1$ και $B_N....B_1$ και το κρατούμενο εισόδου C_{in} και υπολογίζουμε το άθροισμα $S_N....S_1$ και το κρατούμενο εξόδου C_{out} όπως φαίνεται και στο ακόλουθο σχήμα.



Σχήμα 1 : Λογική ενός Αθροιστή Διάδοσης Κρατουμένου (Carry Propagate Adder Logic)

Η ονομασία τους προκύπτει από το γεγονός πως το κρατούμενο σε κάθε bit μπορεί να επηρρεάσσει τα κρατούμενα όλων των μεταγενέστερων bits.

1. ***Carry Ripple Adder*** : Ένας carry ripple adder (CRA ή RCA) n-bit αποτελείται από *n* πλήρεις αθροιστές (full adders) παρατεταγμένους σε σειρά όπως φαίνεται και στο παρακάτω σχήμα που απεικονίζει έναν 4-bit RCA.



Σχήμα 2 : Carry Ripple Adder 4-bit

Το κρατούμενο εξόδου της βαθμίδας i, C_i, είναι το κρατούμενο εισόδου της ακόλουθης, i+1 βαθμίδας. Ο τύπος που μας δίνει το τελικό αθροισμα είναι ο ακόλουθος :

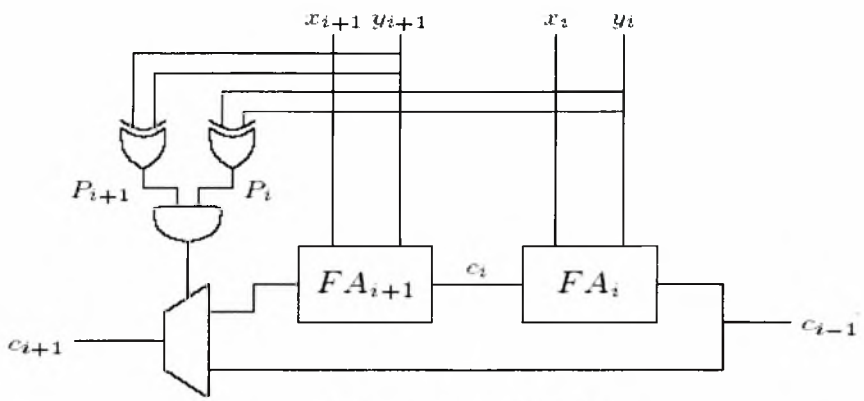
$$\mathbf{S_i = P_i \text{ xor } G_{i-1:0}}$$

Όπου τα σήματα **P** και **G** είναι τα σήματα *Propagate* και *Generate*, τα οποία ορίζονται από τους εξής τύπους :

$$\mathbf{G_i = A_i \text{ and } B_i}$$
$$\mathbf{P_i = A_i \text{ xor } B_i}$$

οι οποίοι «δηλώνουν» πως ένα κρατούμενο «παράγεται» (generates) όταν η εκάστοτε τιμή του C_{out} είναι true (ασχέτως της τιμής του C_{in}), και «διαδίδεται» (propagates) εάν και μόνο εάν ο αθροιστής δεχθεί σε κάποια βαθμίδα του κρατουμενο εισόδου και κάποια από τις εισόδους έχει τιμή true.

2. ***Carry Skip Adder*** : Ο carry skip adder με την λογική που ακολουθεί μειώνει τον χρόνο που απαιτείται για να διαδωθεί το κρατούμενο με το να «αγνοεί» κάποια από τα συνεχόμενα στάδια του αθροιστή. Το σχήμα που ακολουθεί παρουσιάζει τον προκείμενο αθροιστη στην περίπτωση που κάθε ομαδα (η έννοια της ομάδας παρουσιάζεται εκτενώς στην επόμενη παράγραφο) αποτελείται από δυο στάδια.



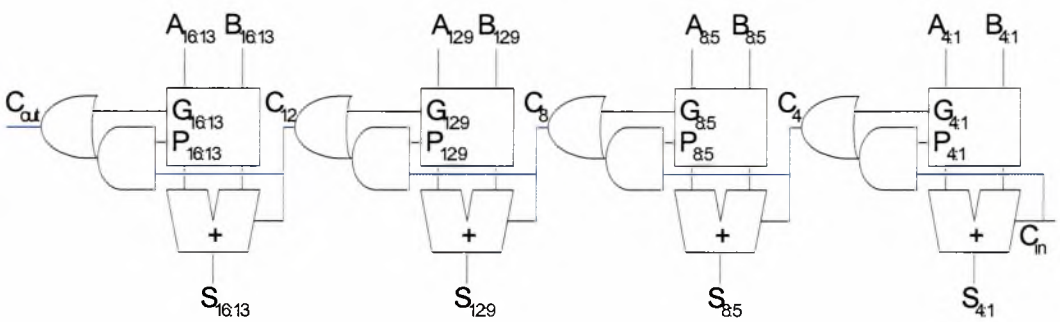
Σχήμα 3 : Carry Skip Adder

Η λογική του αθροιστή βασίζεται στην παρατήρηση πως, η διαδικασία διάδοσης του κρατουμένου μπορεί να παραβλευθεί σε κάποιο στάδιο i εάν και μόνο εάν για το στάδιο αυτό ισχύει $P_i = 0$ (ο τύπος του P_i δίνεται από τους γενικούς τύπους που αναφέρθηκαν στην προηγούμενη παράγραφο). Με βάση την παρατήρηση αυτή πολλά συνεχόμενα στάδια μπορούν να μην υπολογιστούν αν ισχύει η παραπάνω εξίσωση.

Αυτό έχει ως αποτέλεσμα την δημιουργία ομάδων από συνεχόμενα στάδια όπου σε κάθε ομάδα εφαρμόζεται μια λογική παρόμοια με αυτή του carry ripple adder. Κάθε τέτοια ομάδα δημιουργεί ένα ειδικό σήμα το οποίο στην περίπτωση που ικανοποιεί την παραπάνω ισότητα για τα «εσωτερικά του στάδια», οδηγεί τον αθροιστή στο να αγνοήσει τα συγκεκριμένα στάδια και να τροφοδοτήσει το κρατούμενο στο επόμενο γκρουπ. Αν εξετάσουμε, λοιπόν, τον αθροιστή αυτό από την σκοπιά της

καθυστέρησης που παρατηρείται, λόγω της παραπάνω βελτιστοποίησης είναι λογικό να συμπεράνουμε πως είναι «γρηγορότερος» από τον carry ripple adder.

3. ***Carry Lookahead Adder*** : Ο carry lookahead adder είναι παρόμοιος με τον carry skip adder με την διαφορά ότι υπολογίζει ανά ομάδες τα σήματα P και G που αναφέραμε παραπάνω έτσι ώστε να αποφευχθεί η καθυστέρηση που παρουσιάζεται περιμένοντας το ripple που καθορίζει κατά πόσο μια ομάδα δημιουργεί κρατούμενο.



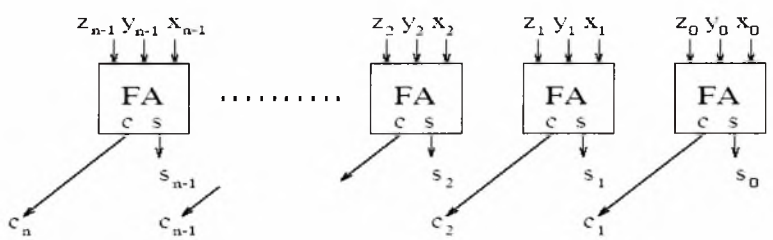
Σχήμα 5 : Carry Lookahead Adder 16-bit

Η παραπάνω εικόνα παρουσιάζει σχηματικά την μορφή ενός 16-bit carry lookahead adder.

4. ***Carry Save Adder*** : Όταν χρειάζεται να προσθέσουμε δυο ή παραπάνω αριθμούς (για παράδειγμα στις περισσότερες υλοποιήσεις πολλαπλασιαστών), η διαδικάσία διάδοσης του κρατουμένου (carry propagation) η οποία μας «κοστίζει» ακριβά σε χρόνο πρέπει να επαναληφθεί αρκετές φορές. Όταν αναφερόμαστε μάλιστα σε σχεδιάσεις με μεγάλο αριθμό τελούμενων καταλαβαίνουμε πως η παραπάνω διαδικασία γίνεται απαγορευτική από άποψη «χρονικού κόστους». Η τεχνική, λοιπόν, που χρησιμοποιούμε στην πλειοψηφία των περιπτώσεων

είναι η carry-save πρόσθεση. Στην τεχνική αυτή αφήνουμε το κρατούμενο να διαδοθεί μόνο στο τελευταίο στάδιο της πρόσθεσης, ενώ σε όλα τα άλλα βήματα δημιουργούμε ένα «μερικό άθροισμα» και μια ακολουθία κρατουμένων ξεχωριστά. Στην ουσία, δηλαδή, ο carry save adder (CSA) δέχεται τρεις αριθμούς n-bit και «δημιουργεί» δυο αποτελέσματα n-bit το κάθε ένα. Αυτά είναι ένα «μερικό άθροισμα» n-bit και ένα κρατούμενο n-bit. Ένας δεύτερος csa δέχεται αυτά τα δυο αποτελέσματα όπως και ακόμη ένα τελούμενο εισόδου και «παράγει» εκ νέου ένα «μερικό άθροισμα» και ένα κρατούμενο. Παρατηρούμε, λοιπόν, πως ένας αθροιστής αυτής της κατηγορίας είναι ικανός να μειώσει τον αριθμό των τελούμενων στην πράξη της πρόσθεσης από τρεις σε δυο.

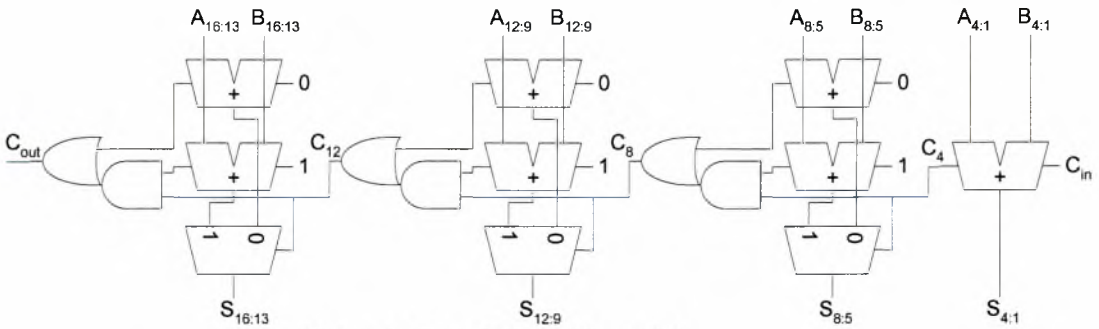
Η παρακάτω εικόνα παρουσιάζει έναν n-bit carry save adder σε μια από τις πολλαπλές παραλλαγες με τις οποίες μπορεί να υλοποιηθεί (εδώ με χρήση υποκυκλωμάτων πλήρους αθροιστή)



Σχήμα 6 : Carry Save Adder

5. ***Carry Select Adder*** : Το critical path τόσο του carry skip adder όσο και του carry lookahead adder εμπεριέχει τον υπολογισμό του κρατουμένου σε κάθε n-bit ομάδα που σχηματίζεται, και του αθροίσματος εν συνεχεία για κάθε ομάδα που δέχεται κρατούμενο εισόδου. Η καθυστέρηση αυτή στο critical path μπορεί να μειωθεί με την χρήση μιας τεχνικής η οποία βασίζεται στον «προ - υπολογισμό» των εξόδων και για τις δυο πιθανές

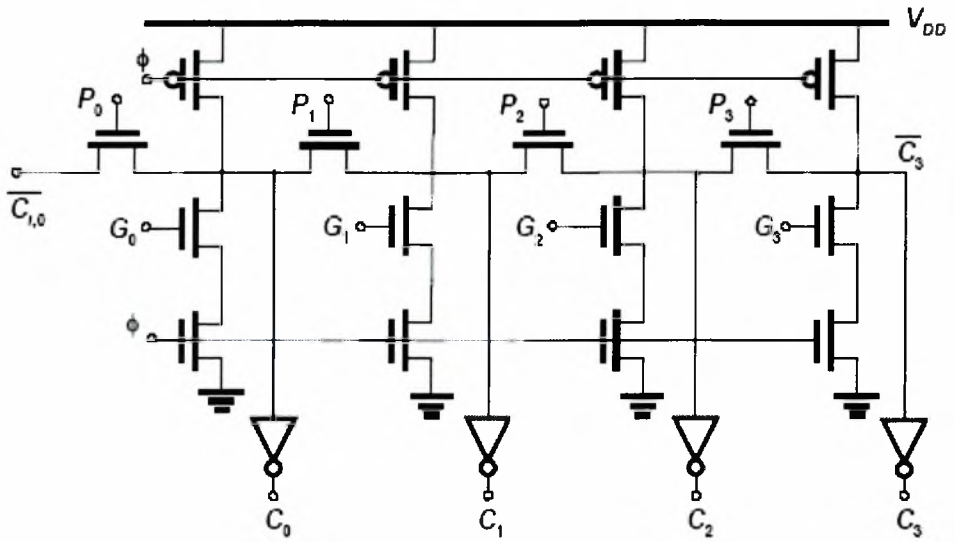
εισόδους και την επιλογή του αποτελέσματος με την χρήση ενός πολυπλέκτη.



Σχήμα 7 : Carry Select Adder 16-bit

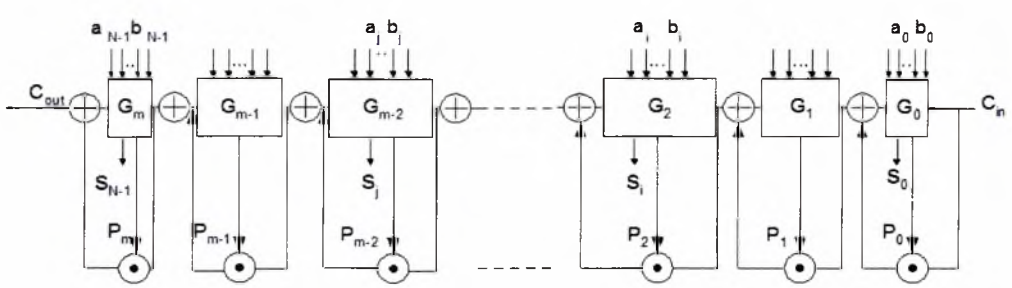
Ο carry select adder υλοποιεί την λειτουργία αυτή με την χρήση δυο αθροιστών n-bit σε κάθε στάδιο, όπως φαίνεται και στο παραπάνω σχήμα. Ο ένας αθροιστής υπολογίζει το άθροισμα υποθέτοντας κρατούμενο εισόδου ίσο με το μηδέν ένω ο έταιρος υπολογίζει το άθροισμα υποθέτοντας ότι το κρατούμενο εισόδου είναι ίσο με την μονάδα. Το πραγματικό κρατούμενο είναι αυτό που ενεργοποιεί, τελικά, τον πολυπλέκτη ο οποίος με την σειρά του μας δίνει το σωστό αποτέλεσμα.

6. ***Manchester Cary Chain Adder*** : Ο αθροιστής αυτός μπορεί να σχεδιαστεί με βάση την λογική διακοπτών χρησιμοποιώντας propagate, generate και kill σήματα. Το «συμπλήρωμα» του κρατούμενου μπορεί να διαδοθεί μέσω της πύλης μετάδοσης, που δημιουργείτα από το nMOS τρανζίστορ ή να «διαγραφει» μέσω του pMOS τρανζίστορ, όπως φαίνεται στο ακόλουθο σχήμα.



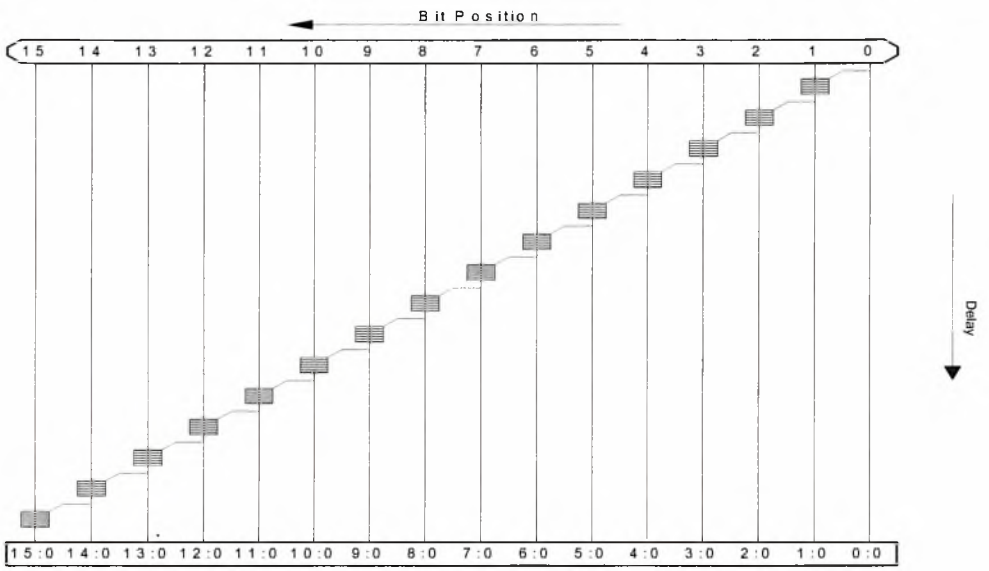
Σχήμα 8 : Manchester Carry Chain Adder 4-bit

Όπως μπορούμε εύκολα να παρατηρήσουμε στο σχήμα που ακολουθεί, για την δημιουργία του Manchester Carry Chain Adder απαιτείται η συνένωση πολλαπλών σταδίων. Η αντίσταση και η χωρητικότητα του αθροιστή αυξάνονται όσο αυξάνεται το μέγεθός του. Επομένως η καθυστέρηση αυξάνεται ανάλογα με το τετράγωνο του μεγέθους. Το γεγονός αυτό καθιστά την συγκεκριμένη τοπολογία ακατάλληλη για μεγάλους αθροιστές. Το κατάλληλο μέγεθος ενός αθροιστή αυτής της κατηγορίας εξαρτάται από την παρασιτική χωρητικότητα και μπορεί να καθοριστεί μέσω προσομοίωσης και υπολογισμών πάνω σε μια συγκεκριμένη τεχνολογία.

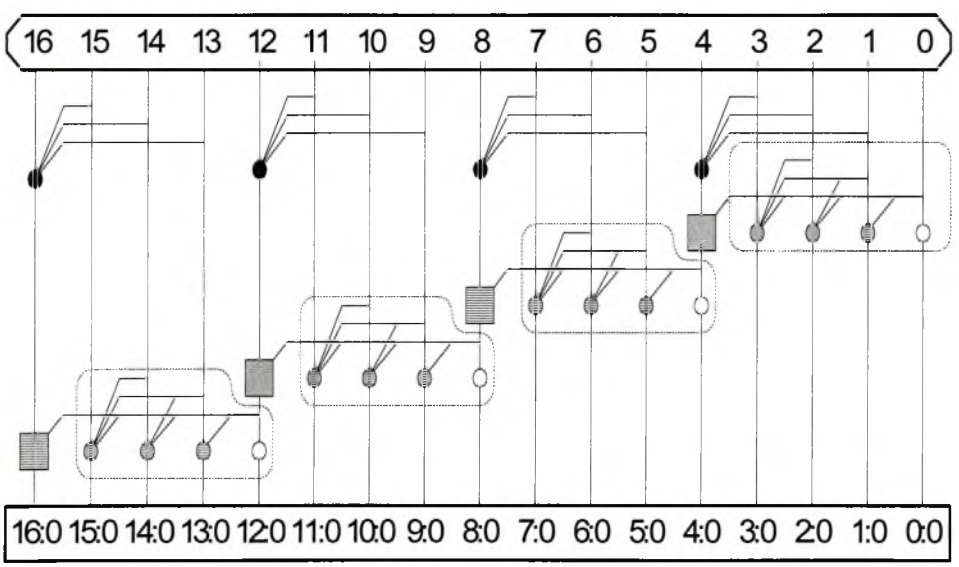


Σχήμα 9 : Manchester Carry Chain Adder

Η περιγραφή των αθροιστών σε θεωρητικό επίπεδο σε συνδυασμό με τις σχηματικές αναπαραστάσεις μας δίνουν μια πολύ καλή περιγραφή της λειτουργίας τους και των ιδιοτήτων τους. Την περιγραφήη αυτή μπορούμε να την επεκτείνουμε παρουσιάζοντας το PG δίκτυο που σχηματίζουν ορισμένοι από αυτούς (όπου P και G είναι τα σήματα propagate και generate αντίστοιχα των οποίων η σημειολογία έχει αναφερθεί αναλυτικά στην αρχή του κεφαλαίου).



Σχήμα 10 : PG Δικτυακό Διάγραμμα του Carry Ripple



Σχήμα 11 : PG Δικτυακό Διάγραμμα του Carry Lookahead

Τα μαύρα κελιά περιέχουν την generate-propagate λογική που διέπει την προκείμενη ομάδα του εκάστοτε αθροιστή (δηλαδή μια AND-OR και μια AND «πύλη» αντίστοιχα όπως περιγράφονται από τις εξισώσεις των σημάτων P και G που παρουσιάστηκαν παραπάνω). Τα γκρι κελιά περιέχουν μόνο την generate λογική της ομάδας. Αυτά χρησιμοποιούνται στην τελική θέση κάθε στήλης, λόγω του ότι το σήμα που παράγει κάθε generate ομάδα ενός αθροιστή, είναι το μόνο που απαιτείται για τον υπολογισμό των αθροισμάτων. Τέλος, παρατηρούμε πως εμφανίζονται και λευκά κελιά τα οποία χρησιμοποιούνται για να αναπαραστήσουν buffers.

Tree Adders :

Για αθροιστές μεγαλύτερους από 16 bits, η καθυστέρηση της carry lookahead ή της carry skip ή της carry select τοπολογίας – διαδικασίας γίνεται κυρίαρχη σε σχέση με την συνολική καθυστέρηση διάδοσης του κρατουμένου διαμέσου του συνόλου των σταδίων της σχεδίασης. Η καθυστέρηση αυτή μπορεί να ελαττωθεί με την χρησιμοποίηση της lookahead λογικής όχι πλέον σε κάθε στάδιο αλλά σε κάθε έναν από ένα σύνολο carry lookahead αθροιστών. Στην γενικότερη περίπτωση μπορούμε να κατασκευάσουμε μια πολυεπίπεδη δομή δένδρου από αθροιστές αυτού του είδους προκειμένου να πετύχουμε λογαριθμική αύξηση της καθυστέρησης.

Οι αθροιστές αυτής της κατηγορίας αναφέρονται στην βιβλιογραφία ως tree adders, logarithmic adders, multilevel lookahead adders ή parallel prefix adders. Οι σημαντικότεροι εκπρόσωποι αυτής της κατηγορίας είναι οι ακόλουθοι:

a. ***Sklansky / "divide & conquer" adder*** : Ο αθροιστής αυτός μειώνει την καθυστέρηση σε $\log_2 N$ στάδια με το να υπολογίζει τα

ενδιάμεσα prefixes παράλληλα με τα prefixes των μεγαλύτερων σταδίων. Το μειωνέκτημα που παρουσιάζεται με την χρήση αυτής της μεθόδου είναι η αύξηση των fanouts τα οποία διπλασιάζονται σε κάθε επίπεδο. Τα υψηλά fanouts προκαλούν κακή απόδοση στους μεγάλους αθροιστές εκτός και αν οι πύλες αυτών έχουν το κατάλληλο «μέγεθος» ή τα σημαντικά για την εξέλιξη της πράξης σήματα (propagate, generate, kill) απομονώνονται προτού χρησιμοποιηθούν με την χρήση κατάλληλων buffers.

b. ***Han - Carlson Adder*** : Τα Han - Carlson δέντρα είναι μια οικογένεια δικτυωμάτων όπου τα «μονά» bits υπολογίζονται με χρήση της μεθόδου που χρησιμοποιεί ο Koggee - Stone και ακολούθως χρησιμοποιεί ένα επιπλέον στάδιο για την μεταφορά των αποτελεσμάτων στα «ζυγά» bits.

c. ***Elm Adder*** : Ο αθροιστής αυτός ονομάζεται και Blocked Lookahead Adder (BLCA). Το σύνολο της σχεδίασής του καταλαμβάνει σχετικά μικρό χώρο (O(NlogN)) και χρησιμοποιεί λιγότερες διασυνδέσεις απ'ότι οι άλλοι αθροιστές της κατηγορίας του. Ο Elm χρησιμοποιεί ένα δυαδικό δέντρο απλών επεξεργαστών για την πραγματοποίηση της πρόσθεση σε λογαριθμικό χρόνο (O(logN)). Τα φύλα του δέντρου υπολογίζουν τα επιμέρους αθροίσματα και τα προωθούν στα επόμενα επίπεδα. Οι κόμβοι στα υψηλότερα σημεία του δέντρου δέχονται τα μερικά αθροίσματα και δημιουργούν και προωθούν την κατάλληλη πληροφορία στα επόμενα επίπεδα ως την κορυφή.

2.2 Προσομοίωση

Για την προσομοίωση των ψηφιακών κυκλωμάτων που παρουσιάζονται στην εργασία αυτή, το εργαλείο που χρησιμοποιήθηκε είναι το Modelsim version 5.7d. Το Modelsim μας παρέχει την δυνατότητα να ελέγξουμε την ορθότητα και χρηστικότητα του κώδικά μας όταν αυτός είναι γραμμένος σε κάποια γλώσσα περιγραφής υλικού (Hardware Description Language (HDL)), όπως η Verilog ή η VHDL.

Στην προκείμενη εργασία το Modelsim χρησιμοποιήθηκε για τον έλεγχο των αρχείων που περιείχαν την περιγραφή των αθροιστών και των αρχείων που αποτελούσαν τα testbenches των τοπολογιών αυτών. Αναλυτικότερα η ροή των εντολών που ακολουθήθηκε για κάθε ένα από τα κυκλώματά μας είναι η ακόλουθη:

i. Με την επιλογή File > Change_Directory μεταφερόμαστε στον φάκελο στον οποίο είναι αποθηκευμένα τα αρχεία VHDL που περιέχουν την περιγραφή του κυκλώματος του εκάστοτε αθροιστή αλλά και του testbench που θα χρησιμοποιήσουμε.

ii. Δημιουργούμε μια νέα βιβλιοθήκη με τις εντολές File > New > Library. Στο πλαίσιο που εμφανίζεται επιλέγουμε "a map to an existing library" και ακολούθως προσδιορίζουμε την τοποθεσία στην οποία βρίσκεται η βιβλιοθήκη με βάση την οποία θέλουμε να γίνει η προσομοίωση της λειτουργίας του κυκλώματος. Στην περίπτωσή μας οι βιβλιοθήκες που χρησιμοποιήσαμε είναι οι βιβλιοθήκες της UMC και πιο

συγκεκριμένα οι UMC13 και UMC 18 και όχι κάποια default επιλογή του προγράμματος.

iii. Ακολούθως, επιλέγουμε Compile και παρουσιάζονται σε παραθυρικό περιβάλλον τα αρχεία που αναφέραμε παραπάνω. Αφού ελέγξουμε την ορθότητα του αρχείου του αθροιστή και στην συνέχεια του αρχείου του testbench αυτού είμαστε έτοιμοι για την κύρια φάση της εργασίας μας, την προσομοίωση.

iv. Επιλέγοντας Simulate προσομοιώνουμε την λειτουργία του κυκλώματος με την βοήθεια του testbench και μας δίνεται η δυνατότητα αναπαράστασης της «λειτουργίας» του κυκλώματός μας με την χρήση κυματομορφών. Τέλος, σε αυτή την φάση της εργασίας μας δίνεται και η δυνατότητα δημιουργίας των VCD αρχείων η οποία θα αναλυθεί εκτενώς στο επόμενο εδάφιο.

VCD (Value Change Dump) Files :

Τα VCD αρχεία, των οποίων η σημαντικότητα θα γίνει περισσότερο κατανοητή κατά το στάδιο του υπολογισμού της καταναλισκώμενης ισχύος των κυκλωματικών στοιχείων που χρησιμοποιούμε, αποτελούν ASCII αρχεία τα οποία περιέχουν πληροφορίες για τη σύνθεση του κυκλώματός μας, ορισμούς για τις μεταβλητές του και τις μεταβολές που υφίστανται οι τιμές των μεταβλητών.

Τα VCD αρχεία χρησιμοποιούνται κατά κύριο λόγο στις σχεδιάσεις που είναι γραμμένες σε Verilog, μια, όμως, από τις σημαντικές καινοτομίες του Modelsim είναι το γεγονός πως μας παρέχει την δυνατότητα χρήσης και υλοποίησής τους με βάση αρχεία τα οποία είναι γραμμένα σε VHDL.

Το Modelsim παρέχει την δυνατότητα επιλογής ανάμεσα σε δυο διαφορετικές ροές δημιουργίας των VCD αρχείων. Η πρώτη, την οποία και ακολουθούμε σε αυτή την εργασία, παράγει ένα four-state VCD αρχείο, χωρίς πληροφορίες που αφορούν την δύναμη οδήγησης των πυλών. Η δεύτερη, παράγει ένα εκτενές αρχείου του τύπου αυτού με πληροφορίες τόσο για την δύναμη οδήγησης των πυλών όσο και για τα δεδομένα και την συμπεριφορά τους σε κάθε port της σχεδίασης.

Παρόλαυτα, η χρησιμοποίηση της δεύτερης ροής δεν κατέστει δυνατή διότι παρουσίασε ορισμένα προβλήματα κατά την χρήση της σε συνδυασμό με τα εργαλεία της Synopsys και αυτό μας οδήγησε στην επιλογή της πρώτης.

Σε πιο πρακτικά ζητήματα τώρα, αφού προσομοιώσουμε και ελέγξουμε την σωστή λειτουργία του testbench, δίνουμε τις ακόλουθες εντολές (στον χρόνο 0 της προσομοίωσης) οι οποίες παράγουν το επιθυμητό αρχείο.

1. vcd file output.vcd
2. run "critical path + 10%critical path"
3. vcd add -r *
4. run 100
5. force a 0000
6. run 100
7. force b 0000
8. run 100
9. force cin 0
10. run 100
11. force a 1111
12. run 100
13. force b 1111
14. run 100

15. force cin 1
16. run 100
17. vcd checkpoint
18. quit -sim

Στο παραπάνω παράδειγμα δημιουργίας ενός VCD αρχείου υποθέτουμε πως διαθέτουμε έναν αθροιστή 4-bit με εισόδους "a" και "b" και κρατούμενο εισόδου "cin". Η διαδικασία ξεκινάει στην χρονική στιγμή 0 και με την πρώτη εντολή δίνουμε το επιθυμητό όνομα στο αρχείο που θα παραχθεί.

Η δεύτερη εντολή χρησιμοποιείται προκειμένου να ληφθούν υπόψη όλες οι μεταβολές που συμβαίνουν στις εισόδους του κυκλώματος. Ακολούθως, τρέχουμε το κύκλωμα για ένα χρονικό διάστημα 100 ns καταγράφοντας τις απαραίτητες πληροφορίες. Η διαδικασία αυτή επαναλαμβάνεται στην συνέχεια αφού όμως δώσουμε τιμές σε ορισμένες ή και σε όλες τις μεταβλητές εισόδου. Στην περίπτωσή μας «αναγκάζουμε» τις μεταβλητές εισόδου να αποκτήσουν ακραίες τιμές (0000 ή 1111) προκειμένου να έχουμε όσο το δυνατόν μεγαλύτερο εύρος πληροφοριών.

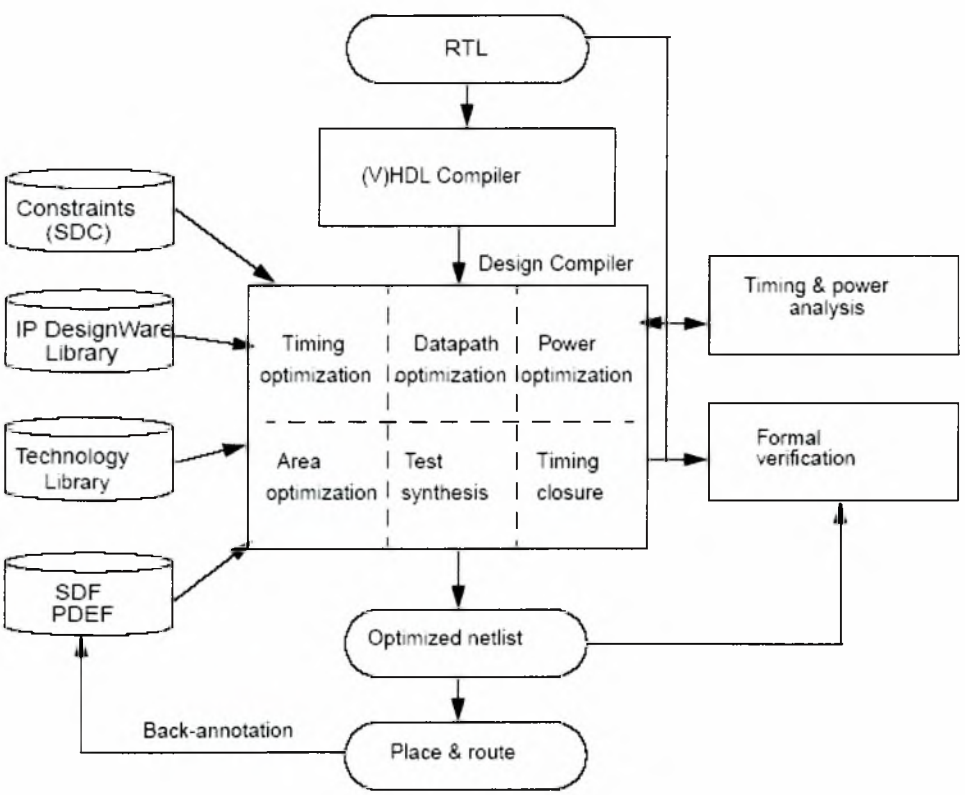
Η διαδικασία σταματάει σε κάποια χρονική στιγμή με την εντολή quit -sim η οποία σταματάει την προσομοίωση.

2.3 Σύνθεση

Όπως έχουμε ήδη αναφέρει, στο μεγαλύτερο τμήμα αυτής της εργασίας χρησιμοποιήθηκαν τα εργαλεία της Synopsys. Ο «πυρήνας» του πακέτου που προσφέρει η Synopsys αποτελείται από τον Design Compiler και τον Power Compiler.

Ο Design Compiler (DC) είναι το λογισμικό που καλείται από το σύνολο των προγραμμάτων που προσφέρονται για την σύνθεση ψηφιακών κυκλωμάτων. Ο DC βελτιστοποιεί τις σχεδιάσεις με απώτερο στόχο την παροχή μικρότερων και γρηγορότερων αναπαραστάσεων μιας λογικής συνάρτησης. Περιέχει υποεργαλεία τα οποία συνθέτουν τις HDL σχεδιάσεις σε τεχνολογικά εξαρτημένες σχεδιάσεις σε επίπεδο πυλών. Υποστηρίζει την δυνατότητα βελτιστοποίησης σχεδιάσεων είτε σε ιεραρχική είτε σε flat μορφή και μπορεί να συνθέσει τόσο ακολουθιακά όσο και συνδιαστικά κυκλώματα βελτιώνοντας την ταχύτητα απόκρισής τους, τον χώρο που καταλαμβάνουν και την ισχύ που καταναλώνουν.

Στο σχήμα της επόμενης σελίδας παρουσιάζεται ο τρόπος με τον οποίο ο DC συμβάλλει στην συνολική ροή σχεδίασης :



Σχήμα 12 : Συμμετοχή του DC στην συνολική ροή σύνθεσης που ακολουθείται

Στο σημείο αυτό θα δώσουμε και τον πρώτο «επίσημο» ορισμό της σύνθεσης ενός ψηφιακού κυκλώματος :

«Σύνθεση ονομάζεται η διαδικασία κατά την οποία μετατρέπουμε μια σχεδίαση, η οποία μας δίνεται σε HDL κώδικα, σε ένα βέλτιστο netlist σε επίπεδο πυλών το οποίο προσδιορίζεται πλήρως από μια τεχνολογική βιβλιοθήκη»

Τα βήματα που ακολουθούμε κατά την διαδικασία της σύνθεσης ενός κυκλώματος, σε άμεση αντιστοιχία με το παραπάνω σχήμα, είναι τα εξής :

1) Δεχόμαστε ως είσοδο, αρχεία τα οποία περιγράφουν το κύκλωμά μας σε επίπεδο καταχωρητών (RTL – Register Transfer Level), δηλαδή αρχεία σε κάποια γλώσσα περιγραφής υλικού (HDL) όπως η Verilog και η VHDL.

2) Κατά την σύνθεση ο Design Compiler μεταφράζει την HDL περιγραφή σε συνθετικά στοιχεία της DesignWare βιβλιοθήκης, όπως είναι για παράδειγμα οι αθροιστές και τα γενικά Boolean συνθετικά στοιχεία (τα οποία στο εξής θα αναφέρονται ως generic Boolean components ή πιο απλά GTECH components). Τα GTECH components δεν έχουν πληροφορίες όσον αφορά τον χρονισμό και την δυνατότητα οδήγησης και δεν προσδιορίζονται – αντιστοιχούν σε κάποια τεχνολογική βιβλιοθήκη. Ο DC χρησιμοποιεί τεχνολογικές (technology), συνθετικές (synthetic) και συμβολικές (symbolic) βιβλιοθήκες από το σημείο αυτό και μετέπειτα για την υλοποίηση της σύνθεσης και την παρουσίαση του αποτελέσματος ως γραφική αναπαράσταση.

3) Αφού γίνει η μετάφραση - μετατροπή του HDL κώδικα σε επίπεδο πυλών, ο DC βελτιστοποιεί την σχεδίαση και αντιστοιχίζει τα στοιχεία που την αποτελούν σε ένα συνδυασμό αποτελούμενο από συγκεκριμένα κελια

βιβλιοθηκών, βασιζόμενος στις επιλογές του χρήστη και τους περιορισμούς που αυτός έχει δώσει. Οι περιορισμοί αποτελούν στην ουσία τις σχεδιαστικές απαιτήσεις του χρήστη για τους στόχους που θέλει να πετύχει με βάση την απόδοση της σχεδίασης, δηλαδή αναφέρονται στους χωρικούς και χρονικούς περιορισμούς κατά κύριο λόγο υπό τους οποίους καλείται να επιτελεστεί η σύνθεση.

4) Αφού ολοκληρωθεί η φάση αυτή, ακολουθώντας την ροή σχεδίασης που παρουσιάστηκε στο παραπάνω σχήμα, «τεστάρουμε» το αποτέλεσμα της σύνθεσης για να δούμε αν «πιάνει» τα standards που θέσαμε στην προηγούμενη φάση έτσι ώστε στην περίπτωση που παρουσιαστεί οποιοδήποτε πρόβλημα να το επιλύσουμε σε όσο το δυνατό πιο πρώιμο στάδιο του κύκλου σχεδίασης.

5) Μετά το πέρας του παραπάνω σταδίου, το κύκλωμά μας, όπως έχει πλέον διαμορφωθεί, είναι έτοιμο για την προσαγωγή του στα place & route εργαλεία του πακέτου που παρέχει η Synopsys, τα οποία σε γενικές γραμμές είναι αρμόδια για την τοποθέτηση και την διασύνδεση των επιμέρους κελιών στην σχεδίαση. Ο σχεδιαστής στο σημείο αυτό, έχει την δυνατότητα να επισημειώσει επιπρόσθετα στοιχεία, όπως οι καθυστερήσεις που παρουσιάζονται στις εσωτερικές διασυνδέσεις (interconnection delays) και να βάλει τον Design Compiler να επανασυνθέσει την σχεδίαση προκειμένου η ανάλυση χρονισμού να είναι πιο ακριβής.

Ο Design Compiler διαβάζει και παράγει αρχεία που προσδιορίζουν σχεδιάσεις σε πολλαπλές μορφές που αποτελούν standards στην ηλεκτρονική σχεδίαση ψηφιακών κυκλωμάτων, συμπεριλαμβανομένων των αρχείων της Synopsys .db και .eqn. Επιπρόσθετα ο DC προσφέρει άμεση σύνδεση με άλλα EDA (Electronic Design Automation) εργαλεία. Αυτό μας δίνει την δυνατότητα

να μεταφέρουμε με ευκολία από εργαλείο σε εργαλείο τους περιορισμούς και τα αποτελέσματα που έχουμε ανακτήσει μέχρι την προκείμενη στιγμή.

Στην συνέχεια θα αναφερθούμε, πιο αναλυτικά, στα εργαλεία που χρησιμοποιήθηκαν για την σύνθεση των κυκλωμάτων της παρούσας εργασίας (Design Vision) και τις βιβλιοθήκες που παρέχει η Synopsys και συνεπικουρούν στην επιτυχημένη σύνθεση ψηφιακών κυκλωμάτων.

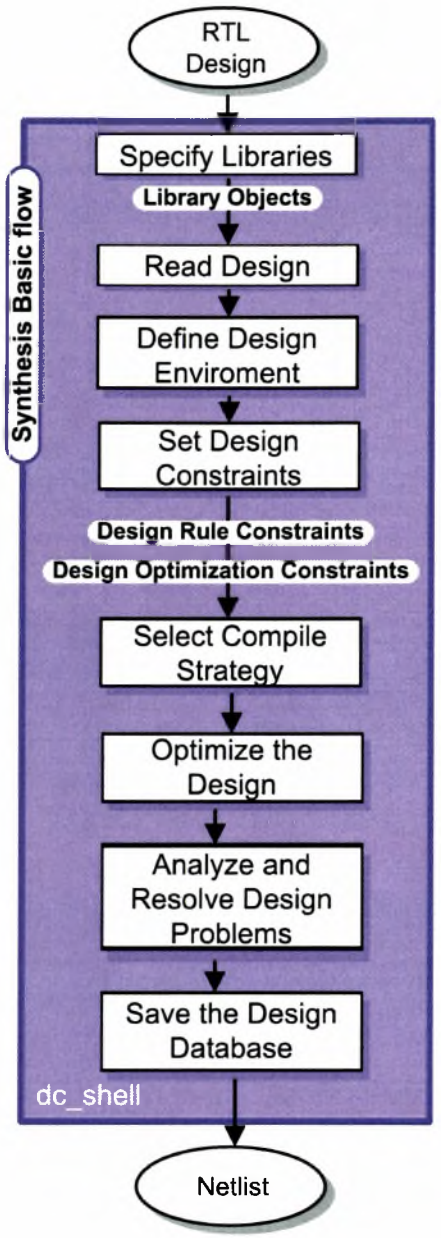
Design Vision :

Το Design Vision είναι μια γραφική διεπαφή (graphical user interface - GUI) για το περιβάλλον σύνθεσης της Synopsys και ένα εργαλείο το οποίο μπορεί να χρησιμοποιηθεί για την ανάλυση της σχεδίασής μας και την «αντιστοίχησή» της στην GTECH τεχνολογική βιβλιοθήκη σε επίπεδο πυλών.

Το γραφικό περιβάλλον του Design Vision περιέχει μενού επιλογών τα οποία μας δίνουν την δυνατότητα να εκτελέσουμε το μεγαλύτερο ποσοστό των εντολών που μπορούν να εκτελεστούν μέσα από το dc_shell. Με άλλα λόγια το εργαλείο αυτό αποτελεί μια μετεξέλιξη του Design Analyzer το οποίο είναι μια ακόμη γραφική διεπαφή που προσφέρεται για την επικοινωνία με τον πυρήνα των εργαλείων της Synopsys. Στο σημείο αυτό πρέπει να αναφέρουμε πως όλες οι δυνατότητες του DC μπορούν να χρησιμοποιηθούν μέσα από την κονσόλα που μας προσφέρει το design vision, και αυτό συμβαίνει διότι τόσο το Design Vision όσο και ο Design Compiler χρησιμοποιούν την ίδια «μηχανή» ανάλυσης στατικού χρονισμού.

Η μεθοδολογία που ακολουθείται είναι όμοια με αυτήν που περιγράψαμε παραπάνω, λόγω του γεγονότος πως το καλούν περιβάλλον είναι αυτό του

Design Compiler. Η μεθοδολογία αυτή μπορεί να περιγραφεί από το ακόλουθο σχήμα :



Σχήμα 13 : Ροή σύνθεσης κυκλώματος με χρήση του Design Compiler

Θα παρουσιάσουμε βήμα – βήμα την πορεία που ακολουθείται βασιζόμενη στο παρακάτω script που αποτελεί μια γενικευμένη εκδοχή αυτών που χρησιμοποιήθηκαν :

1. search_path = {/path}
2. link_library = {/path/library.db}
3. symbol_library = {/path/library.sdb}
4. target_library = {/path/library.db}
5. analyze -format vhdl {/path/adder.vhd}
6. elaborate adder -library WORK
7. set_load 0.003 all_inputs()
8. set_load 0.003 all_outputs()
9. compile -map_effort high -area_effort high -incremental_mapping
10. write -hierarchy -output adder_netlist.db
11. write -hierarchy -output adder_netlist.vhd
12. write_parasitics -output parasitics.reduced.spef
13. report_design
14. report_compile_options
15. report_area
16. report_net

17. report_cell

18. report_area

19. report_timing -from all_inputs() -to all_outputs()

Οι εντολές, με την σειρά που αναγράφονται, αποτελούν ένα τυπικό παράδειγμα ενός script το οποίο οδηγεί το Design Vision στην σύνθεση ενός ψηφιακού κυκλώματος και την αποθήκευση της νέας σχεδίασης στο δίσκο.

Με την πρώτη εντολή καθορίζουμε το σημείο στον δίσκο όπου είναι αποθηκευμένη η βιβλιοθήκη με βάση την οποία θα γίνει η σύνθεση και το σημείο όπου είναι αποθηκευμένη η περιγραφή του κυκλώματός μας σε κάποια γλώσσα περιγραφής υλικού (π.χ. VHDL).

Οι «εντολές» "link library" και "target library" προσδιορίζουν την θέση όπου βρίσκονται οι τεχνολογικές βιβλιοθήκες που προσδιορίζουν τα κελιά και άλλες πληροφορίες ανάλογα με τον «παροχέα» της τεχνολογίας, όπως τα ονόματα των κελιών, τους σχεδιαστικούς κανόνες και τις συνθήκες λειτουργίας. Η παράμετρος "symbol library" καθορίζει τα σύμβολα που είναι διαθέσιμα για την δημιουργία των κατάλληλων schematics.

Για να ξεκινήσουμε να δουλεύουμε πάνω στην σχεδίαση, πρέπει αρχικά να διαβάσουμε την σχεδίασή μας από τον δίσκο στην ενεργή μνήμη του εργαλείου. Από το σημείο αυτό και έπειτα θα πραγματοποιηθούν όλες οι αλλαγές πριν η σχεδίασή μας αποθηκευτεί και πάλι στον δίσκο. Υπάρχουν δυο δυνατές επιλογές στην περίπτωση που θέλουμε να διαβάσουμε μια σχεδίαση :

- *Analyze & Elaborate :* Χρησιμοποιούμε αυτές τις εντολές για να διαβάσουμε HDL σχεδιάσεις και να τις μετατρέψουμε σε αρχεία βάσης δεδομένων στο format της Synopsys (.db).

- *Read* : Η εντολή αυτή χρησιμοποιείται όταν θέλουμε να διαβάσουμε σχεδιάσεις που είναι ήδη σε .db μορφή.

Η εντολή "analyze" εξετάζει το HDL αρχείο προκειμένου να δει εάν βρίσκεται σε σωστή συντακτική λογική και αν είναι δυνατό να υποστεί σύνθεση. Επιπρόσθετα μεταφράζει τα αρχεία μας σε ένα ενδιάμεσο format και τοποθετεί το αρχείο αυτό στον φάκελο που έχουμε προσδιορίσει ως φάκελο εργασίας (working directory).

Η εντολή "elaborate" εξετάζει το «ενδιάμεσο» αρχείο και στην συνέχεια δημιουργεί την σχεδίαση σε μορφή .db αρχείου. Κατά την διάρκει αυτής της διαδικασίας το πρόγραμμα καθορίζει ποια από τα «στοιχεία» της σχεδίασης πρέπει να αντικατασταθούν από συνθετικά «στοιχεία» της βιβλιοθήκης που εχουμε επιλέξει.

Η εντολή "read" αποτελεί ένα συνδυασμό των παραπάνω εντολών, δεν πραγματοποιεί, όμως όσους ελέγχους πραγματοποιούν οι προκείμενες εντολές.

Ακολούθως καλούμαστε να καθορίσουμε το περιβάλλον της σχεδίασης και τους περιορισμούς/παραμέτρους της σχεδίασης. Ο DC απαιτεί την μοντελοποίηση του σχεδιαστικού περιβαλλοντος του κυκλώματος ώστε να προχωρήσει η διαδικασία της σύνθεσης. Με το μοντέλο αυτό καθορίζουμε τις εξωτερικές συνθήκες λειτουργίας (manufacturing process, temperature, voltage), loads, drives, fanouts, και wire load models. Ο καθορισμός όλων αυτών των παραμέτρων επηρεάζει άμεσα την σύνθεση του κυκλώματος και τα αποτελέσματα των βελτιστοποιήσεων που εφαρμόζει ο DC.

Στην συνέχεια με την εντολή "write_parasitics" αποθηκεύουμε πληροφορίες για τα παρασιτικά που παρατηρούνται στο κύκλωμά μας σε μορφή

.spef αρχείου, προκειμένου να χρησιμοποιήσουμε τις πληροφορίες αυτές κατά τον υπολογσμό της ισχύος.

Τέλος, με χρήση της εντολής "report" και μιας σειράς παραμετρων λαμβάνουμε πληροφορίες για την σχεδίαση, τα κελιά, τον χώρο που καταλαμβάνει το κύκλωμα αλλά και το μέγιστο μονοπάτι που υπάρχει στο κύκλωμα (critical path). Ο υπολογισμός του critical path είναι επιθυμητός διότι είθισται ως εμπειρικός κανόνας να τοποθετείται ως περίοδος στα testbenches των κυκλωμάτων η ποσότητα *critical path + 10%critical path.*

DesignWare Libraries :

Οι βιβλιοθήκες του DesignWare προσφέρουν στοιχεία (components) που είναι τεχνολογικά ανεξάρτητα, αποτελούν δηλαδή, στην ουσία, δομικά μπλοκ σε επίπεδο μικροαρχιτεκτονικής τα οποία είναι πλήρως ενοποιημένα με το περιβάλλον της σύνθεσης του πακέτου εργαλείων της Synopsys. Αποτελούν βιβλιοθήκες που αναπτύσσονται από την ίδια την Synopsys. Η Synopsys, λοιπόν, παρέχει τις ακόλουθες σειρές DesignWare βιβλιοθηκών :

- Foundation Library.

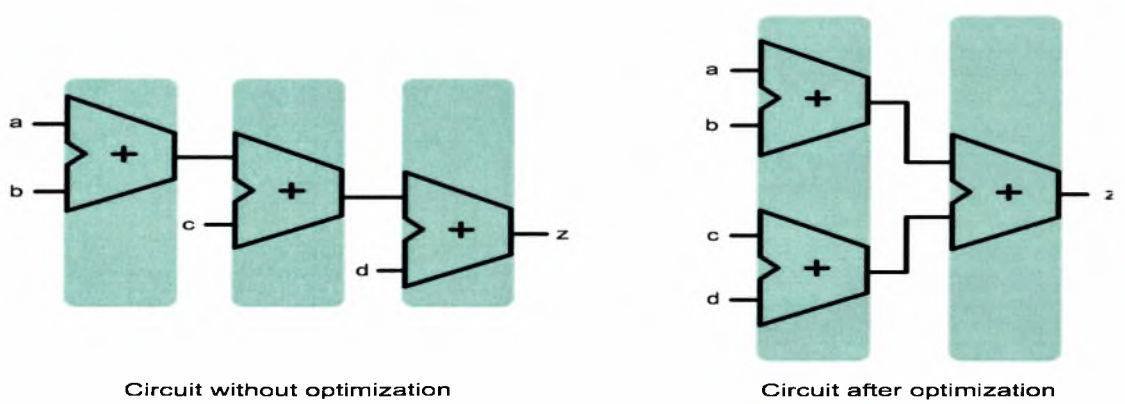
- Digital Signal Processing (DSP) Library.

Στην παρούσα διπλωματική χρησιμοποιήθηκαν πολλά από τα στοιχεία που είναι διαθέσιμα στην Foundation Library.

Η επαναχρησιμοποίηση προηγούμενων κυκλωμάτων είναι ιδιαίτερα επιθυμητή στην διαδικασία της σύνθεσης. Η ροή του DC χρησιμοποιεί τις

Foundation και GTECH βιβλιοθήκες. Η Foundation βιβλιοθήκη είναι μια συλλογή από επαναχρησιμοποιήσιμα, συνθέσιμα δομικά μπλοκ που είναι πλήρως ενοποιημένα με το περιβάλλον των εργαλείων της SYNOPSYS που προσφέρονται για την σύνθεση ψηφιακών κυκλωμάτων.

Η Foundation library μας δίνει την δυνατότητα εκτέλεσης βελτιστοποιήσεων υψηλού βαθμού μέσω της χρήσης κατάλληλων εργαλείων σύνθεσης. Ως παράδειγμα θα αναφέρουμε την περίπτωση στην οποία μέσα σε ένα αρχείο το οποίο περιγράφει μια σχεδίαση, εντοπιστεί ο τελεστής πρόσθεσης «+». Στην περίπτωση, λοιπόν αυτή, ο HDL Compiler (το πακέτο που παρέχει η Synopsys εμπεριέχει τόσο τον VHDL Compiler όσο και τον Verilog Compiler) «αποφασίζει» πως ο τελεστής αυτός περιγράφει στην ουσία έναν αθροιστή. Παραθέτει, επομένως, μια περιληπτική αναπαράσταση της πράξης της πρόσθεσης στο netlist του κυκλώματος. Η προκείμενη αναπαράσταση ονομάζεται synthetic operator και επεξεργάζεται από τους υψηλού επιπέδου αλγόριθμους βελτιστοποίησης τους οποίους εφαρμόζει ο HDL Compiler στην σχεδίαση. Οι βελτιστοποιήσεις αυτές προσφέρουν βελτιστοποιήσεις στην αριθμητική (arithmetic), στον καταμερισμό πόρων (resource sharing), και στην αντιμετάθεση ακίδων (pin permutation).

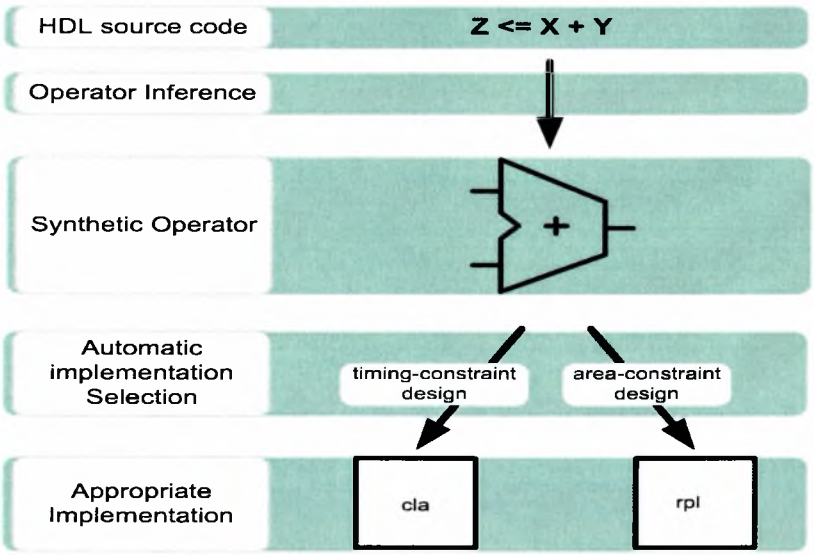


Σχήμα 14 : Βελτιστοποίηση της αριθμητικής

Η βελτιστοποίηση της αριθμητικής χρησιμοποιεί τους κανόνες της άλγεβρας για να βελτιστοποιήσει το μέγεθος και την απόδοση της σχεδίασης με τον επαναπροσδιορισμό της θέσης των πράξεων. Για παράδειγμα, η μαθηματική έκφραση *a+b+c+d* περιγράφει τρία επίπεδα διαδοχικών πράξεων πρόσθεσης, όπου οι μεταβλητές προσθέτονται σε κάθε στάδιο ανά ζεύγη. Με βελτιστοποίηση της αριθμητικής μπορούμε να διατάξουμε τις πράξεις ως εξής: *(a+b) + (c+d)*.Ο νέος αυτός τρόπος «αναπαράστασης» ενδέχεται να προσφέρει μεγαλύτερη ταχύτητα λόγω της μείωσης των επιπέδων λογικής που χρησιμοποιούνται. Στο παραπάνω σχήμα παρουσιάζεται μια αναπαράσταση της αναδιάταξης των πράξεων.

Ο καταμερισμός πόρων επιτρέπει ίδιες λειτουργίες που δεν επικαλύπτονται χρονικά να εκτελούνται από το ίδιο φυσικό H/W. Η αντιμετάθεση ακίδων εκμεταλλεύεται το γεγονός ότι κάποιες πράξεις (όπως η πρόσθεση και ο πολλαπλασιασμός) δεν επηρεάζονται από την εναλλαγή των εισόδων τους.

Με την χρήση της Foundation βιβλιοθήκης, μια πράξη που δίνεται από τον χρήστη μέσω κάποιου αρχείου εισόδου στον DC μπορεί να υλοποιηθεί με πολλαπλούς τρόπους. Στο σημείο αυτό επεμβαίνει ο Design Compiler επιλέγοντας την καταλληλότερη υλοποίηση με βάση την σχεδιασή μας. Στο σημείο αυτό θα αναφέρουμε και πάλι ένα πρακτικό παράδειγμα της λειτουργίας που μόλις περιγράψαμε, το οποίο βασίζεται στην αριθμητική πράξη της πρόσθεσης. Ο DC μπορεί να υλοποιήσει την μη προσημασμένη πρόσθεση είτε χρησιμοποιώντας την τοπολογία ενός carry lookahead adder είτε χρησιμοποιώντας την τοπολογία ενός carry ripple adder. Η επιλογή ανάμεσα στις δυο αυτές δυνατές περιπτώσεις θα γίνει από το εργαλείο της σύνθεσης με βάση τις παραμέτρους που έχει θέσει σε προηγούμεν στάδια της ροής σχεδίασης ο χρήστης. Η διαδικασία αυτή φαίνεται στο ακόλουθο σχήμα:

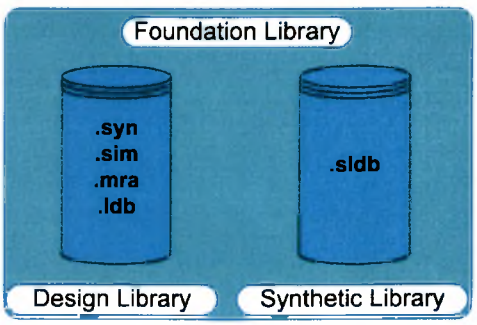


Σχήμα 15 : Διαδικασία επιλογής υλοποίησης

Η Foundation Library αποτελείται από δύο επιμέρους βιβλιοθήκες (όπως παρουσιάζεται και στο ακόλουθο σχήμα), την Design Library και την Synthetic Library:

- Η Design Library είναι ένας Unix φάκελος ο οποίος περιέχει περιγραφές κυκλωμάτων για διάφορες αρχιτεκτονικές.

- Η Synthetic Library είναι ένα δυαδικό αρχείο (με κατάληξη .sldb) το οποίο συνδέει την σχεδίαση με μια Design Library στο εργαλείο σύνθεσης.



Σχήμα 16 : Foundation βιβλιοθήκη

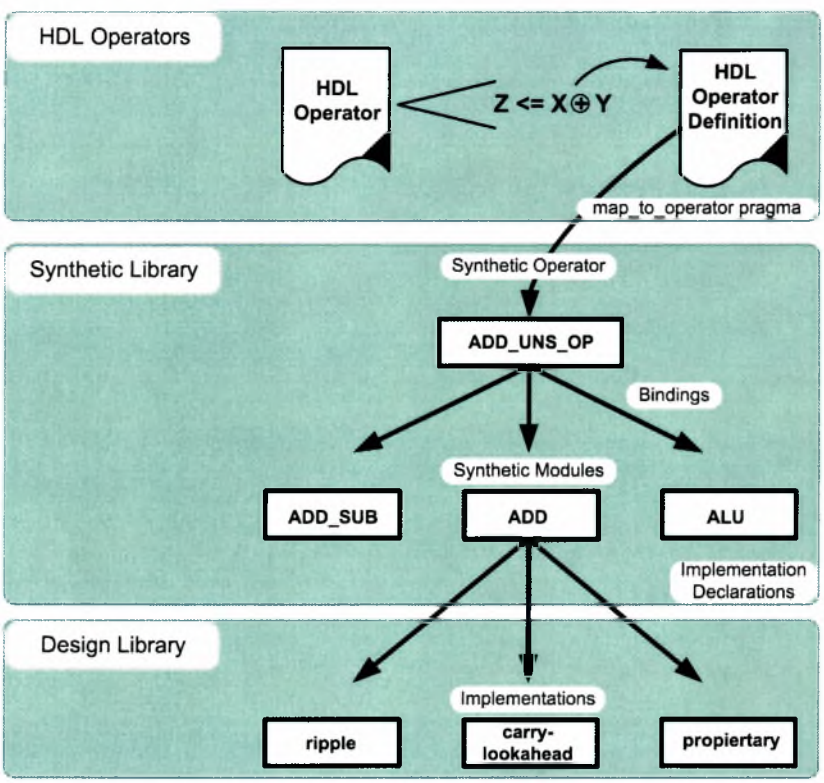
Οι περιγραφές των κυκλωμάτων σε μια Design Library είναι αποθηκευμένες σε δυαδικά αρχεία που είναι άμεσα χρησιμοποιήσιμα από τα εργαλεία της Synopsys. Οι περιγραφές αυτές μπορεί να είναι netlists συγκεκριμένης τεχνολογίας ή hard macros που δε θα επηρεάζονται από την διαδικασία της σύνθεσης, ή ακόμα και πλήρεις ιεραρχικές περιγραφές παραμετροποιημένων και βελτιστοποιημένων σχεδιάσεων.

Η Synthetic Library περιέχει πληροφορία η οποία επιτρέπει στα εργαλεία σύνθεσης να εκτελέσουν βελτιστοποιήσεις υψηλού επιπέδου, όπως αυτές που αναφέραμε παραπάνω, συμπεριλαμβανομένου και της επιλογής υλοποίησης.

Η σύνδεση μεταξύ του πηγαίου κώδικα, της Synthetic Library, και της Design Library γίνεται με τη χρήση μιας ιεραρχίας μοντέλων. Έτσι οι HDL operators συνδέονται με τους synthetic operators, οι οποίοι με την σειρά τους σχετίζονται με τις synthetic modules. Κάθε synthetic module μπορεί να έχει πολλαπλές υλοποιήσεις. Στο σχήμα της επόμενης σελίδας παρουσιάζονται οι μεταβάσεις μέχρι την επιλογή της καταλληλότερης υλοποίησης.

Οι HDL operators είναι δομικά στοιχεία μιας γλώσσας περιγραφής υλικού (VHDL ή Verilog) τα οποία δέχονται τιμές εισόδου και υπολογίζουν τις τιμές εξόδου. Κάποιοι τελεστές υλοποιούνται από την ίδια την γλώσσα (όπως +, -, και

*), όμως και τα ορισμένα από τον χρήστη υποπρογράμματα (functions - procedures) θεωρούνται HDL operators. Οι διαθέσιμοι τελεστές είναι +, -, *, <, >, <=, =>, /, και οι πράξεις που ορίζονται από τις if and case δηλώσεις. Κάθε τελεστής έχει ένα ορισμό γραμμένο σε HDL. Κάθε ορισμός περιέχει την πληροφορία που προσομοιώνει την συμπεριφορά του τελεστή και προαιρετικά και ένα map_to_operator pragma το οποίο συνδέει τον HDL operator με τον κατάλληλο synthetic operator. Πολλοί HDL operators, συμπεριλαμβανομένων και των built-in infix operators, συνδέονται χωρίς κάποιες ειδικές δηλώσεις με την Synopsys Standard Synthetic Library, standard.sldb .



Σχήμα 17 : «Ιεραρχία» των DesignWare βιβλιοθηκών

Η Synthetic Library περιέχει ορισμούς για τους synthetic operators, για τα synthetic modules, και bindings. Επίσης, περιέχει δηλώσεις που συνδέουν τα synthetic modules με τις υλοποιήσεις τους. Οι υλοποιήσεις βρίσκονται στις

ανάλογες Design Libraries. Σε μια Synthetic Library, λοιπόν, μπορούμε να βρούμε πληροφορίες για:

- *Synthetic operator* - Αντιπροσωπεύει την πράξη που έχει κληθεί από τον HDL operator. Τα εργαλεία σύνθεσης εκτελούν βελτιστοποιήσεις υψηλού επιπέδου (αριθμητικής και καταμερισμού πόρων) με την επεξεργασία των synthetic operators.

- *Synthetic module* - Ορίζουν ένα κοινό interface για μια οικογένεια υλοποιήσεων. Όλες οι υλοποιήσεις μιας δεδομένης module έχουν τις ίδιες πόρτες (ports) και την ίδια συμπεριφορά εισόδου - εξόδου.

- *Bindings* - Συνδέουν τους synthetic operators με τις synthetic modules. Για παράδειγμα, ένα binding συνδέει τον synthetic operators της πρόσθεσης με μια adder module (ή μπορούμε να πούμε πως ο synthetic operator της πρόσθεσης είναι bound με την adder module). Ένας ή περισσότεροι synthetic operator μπορούν να συνδεθούν με μια δοσμένη synthetic module, και κάθε τελεστής μπορεί να συνδεθεί με μια ή περισσότερες module.

- *Implementation declarations* - Συνδέει τις synthetic modules με τις υλοποιήσεις σε μια Design Library. Συνεπώς οι implementation declarations συνδέουν την Synthetic Library με την Design Library.

Η Design Library περιέχει τις πραγματικές υλοποιήσεις των σχεδιάσεων. Αυτές οι κυκλωματικές περιγραφές είναι που πραγματοποιούν τις λειτουργικότητες των δομικών στοιχείων της Foundation Library. Οι έννοιες του DesignWare, όπως των synthetic module και implementation είναι παραπλήσιες των εννοιών entity και architecture της VHDL. Μια υλοποίηση μπορούμε να την

δούμε σαν μια αρχιτεκτονική πραγματοποίηση μιας synthetic module. Ένα implementation μπορεί να είναι οτιδήποτε από μια netlist συγκεκριμένης τεχνολογίας έως και μια synthesizable RTL-level περιγραφή σχεδίασης.

Η netlist που παράγεται στο τέλος της διαδικασίας της σύνθεσης με τον DC είναι τεχνολογικά εξαρτημένη. Η υλοποίηση, λοιπόν, της σχεδίασης μας είναι βασισμένη στα διαθέσιμα από την τεχνολογική βιβλιοθήκη δομικά μπλοκ. Στην παρούσα διπλωματική οι τεχνολογικές βιβλιοθήκες που χρησιμοποιήθηκαν είναι της εταιρίας UMC και πιο συγκεκριμένα έγιναν μετρήσεις στις τεχνολογίες 18um, και 13um. Παρακάτω θα παρουσιάσουμε αναλυτικότερα τα βασικά χαρακτηριστικά των τεχνολογικών βιβλιοθηκών της UMC.

Οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι οι παρακάτω:

- *eSi-Route/11™ High Performance 0.18μ Standard Cell Library.*
  Part Number: UMCL18U250
  Rev. 2.1, January, 2001
  Process: L180
  Technology: 0.18um 1.8V/3.3V 1P6M Logic Process

- *eSi-Route/9™ High Density Standard Cell Library.*
  Part Number: UMCL13U210T3
  Revision 2.5, May, 2002
  Process: L130
  Technology: 0.13um 1.2V/3.3V 1P8M Logic Process

Οι βιβλιοθήκες αυτές χρησιμοποιούν ένα νέο, βελτιστοποιημένο για σύνθεση σύνολο από διαθέσιμα κελιά, και έχουν επίσης μια βελτιστοποιημένή σε πυκνότητα αρχιτεκτονική η οποία προσφέρει στους σχεδιαστές συστημάτων

υλοποιήσεις με βελτιστοποιημένο μέγεθος σχεδίασης χωρίς να θυσιάζεται η απόδοση της σχεδίασης. Η βιβλιοθήκη στα 13um παρέχει στα εργαλεία σύνθεσης την δυνατότητα να υλοποιούν σχεδιάσεις με πυκνότητα πυλών μέχρι και 200K gates/mm^2 και αποδόσεις συστημάτων που έχουν συχνότητα από 200 MHz μέχρι 800 MHz.

Στο datasheet της κάθε βιβλιοθήκης παρέχονται πληροφορίες για τα χαρακτηριστικά που διαθέτει η κάθε τεχνολογία αλλά και συγκεκριμένα το κάθε κελί της. Η κάθε βιβλιοθήκη, λοιπόν, έχει κάποια χαρακτηριστικά που είναι κοινά για όλα τα κελιά της . Παρακάτω παρουσιάζονται τα βασικά αυτά χαρακτηριστικά για τις τρεις τεχνολογίες που είχαμε διαθέσιμες.

Κάθε τεχνολογία έχει και κάποιες προτεινόμενες συνθήκες λειτουργίας για τα ολοκληρωμένα κυκλώματα που περιέχει. Οι βιβλιοθήκες της εταιρίας UMC έχουν τα παρακάτω χαρακτηριστικά:

UMCL18U250		
Operating Condition	*Minimum*	*Maximum*
Power Supply	*1.62 V*	*1.98 V*
Junction Temperature	*0 C*	*125 C*

Σχήμα 18 : UMC βιβλιοθήκη στα 0.18 um

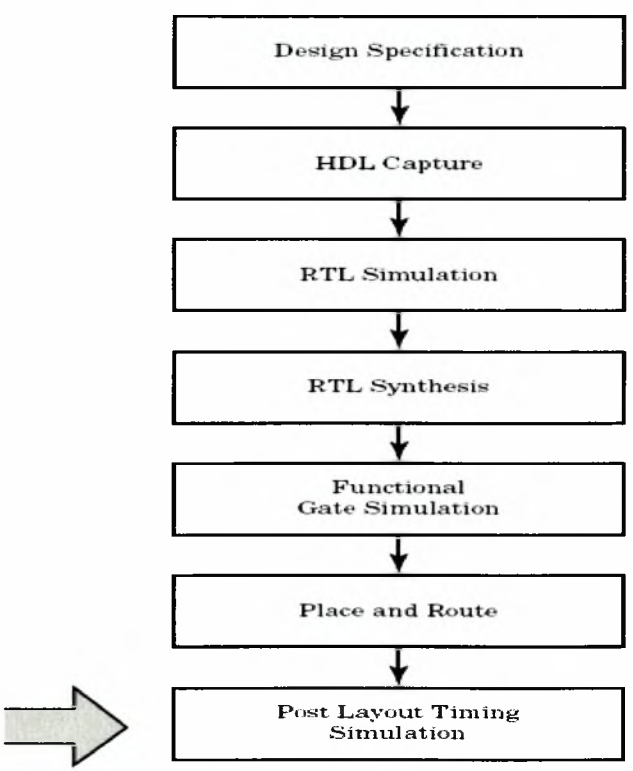
UMCL13U210T3		
Operating Condition	*Minimum*	*Maximum*
Power Supply	*1.08 V*	*1.32 V*
Junction Temperature	*0 C*	*125 C*

Σχήμα 19 : UMC βιβλιοθήκη στα 0.13 um

Κάθε βιβλιοθήκη υποστηρίζει διάφορες επιλογές ως προς το πλήθος των επιπέδων μετάλλου που χρησιμοποιείται για την κατασκευή των κελιών. Η 18um βιβλιοθήκη υποστηρίζει τέσσερα, πέντε και έξι επίπεδα μετάλλου. Τέλος η βιβλιοθήκη στα 13um μπορεί να υλοποιήσει σχεδιάσεις με πέντε, έξι, εφτά και οκτώ επίπεδα.

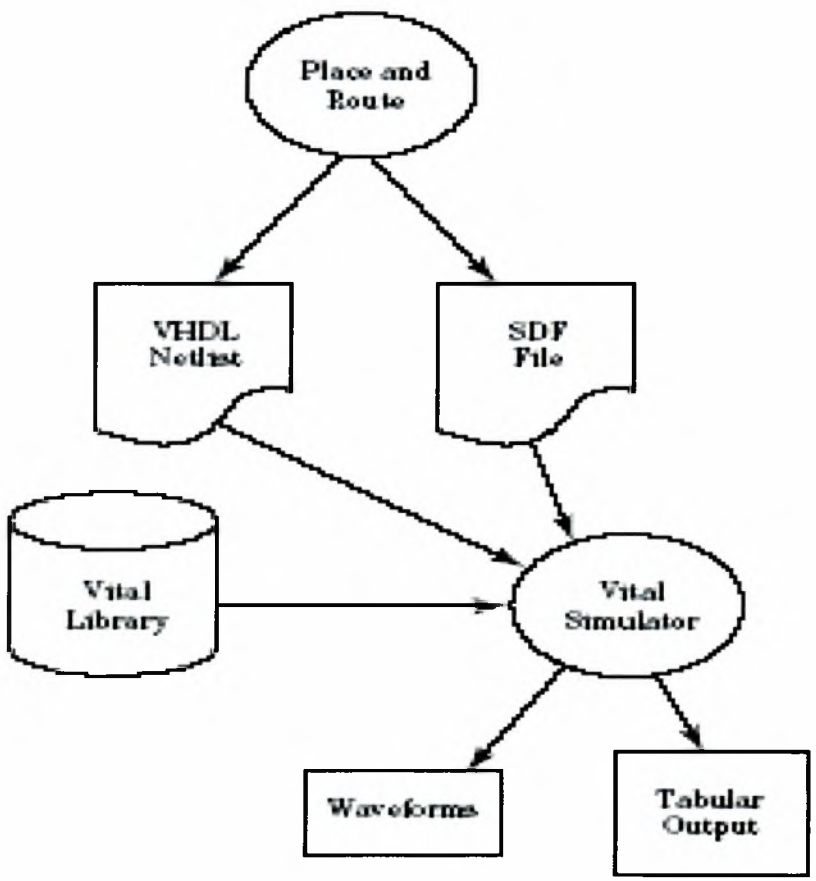
Vital Simulation :

Το τελευταίο «βήμα» στην διαδικασία σχεδίασης που ακολουθήσαμε είναι η πραγματοποίηση της προσομοίωσης του χρονισμού του κυκλώματός μας σε επίπεδο πυλών, όπως φαίνεται και στο ακόλουθο σχήμα :



Σχήμα 20 : Ροή σχεδίασης συμπεριλαμβανομένης και της Vital προοομοίωσης

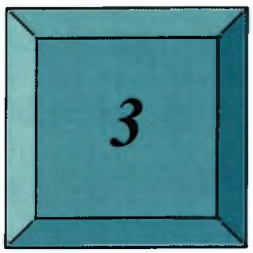
Η post – synthesis αυτή διαδικασία ονομάζεται Vital Simulation. Με την χρήση της VHDL αυτή πραγματοποιείται χρησιμοποιώντας το VITAL (VHDL Initiative Toward ASIC Libraries) standard της IEEE για την μοντελοποίηση του χρονισμού του κυκλώματος σε επίπεδο πυλών. Οι VITAL βιβλιοθήκες χρησιμοποιούνται συναρτήσει ενός VHDL προσομοιωτή (στην περίπτωσή μας το Modelsim).



Σχήμα 21 : Αρχεία εισόδου και αποτελέσματα της VITAL προσομοίωσης.

Κατά την διαδικασία της προκείμενης προσομοίωσης έχουμε ως αρχεία εισόδου ένα VHDL netlist που έχει προκύψει από την σύνθεση του αρχικού κώδικα με χρήση του DC, και μια VITAL βιβλιοθήκη που περιγράφει την «συμπεριφορά»

των διαφόρων οντοτήτων που χρησιμοποιήθηκαν στην σχεδίαση. Το τελικό αποτέλεσμα υπό την μορφή κυματομορφής επιβεβαιώνει ή διαψεύδει την ορθή λειτουργία του κυκλώματός μας βάσει του χρονισμού του.



3 Κατανάλωση Ισχύος

3.1 Βασικές Έννοιες

Η κατανάλωση ισχύος ενός ψηφιακού CMOS κυκλώματος δύναται να περιγραφεί από την ακόλουθη εξίσωση :

$$P_{average} = P_{dynamic} + P_{short\text{-}circuit} + P_{static}$$

Ο πρώτος όρος της εξίσωσής ($P_{average}$) είναι η μέση ισχύς που καταναλώνεται στο κύκλωμα, ο όρος $P_{dynamic}$ περιγράφει την δυναμική κατανάλωση ισχύος που προκαλείται από την αλλαγή κατάστασης (άνοιγμα-κλείσιμο) των διακοπτών, το $P_{short\text{-}circuit}$ είναι το ποσοστό εκείνο της ισχύος που καταναλώνεται όταν υπάρχει άμεσο μονοπάτι από την πηγή στην γείωση, ενώ, τέλος, το P_{static} περιγράφει την στατική κατανάλωση ισχύος. Στα επόμενα εδάφια θα εξηγήσουμε αναλυτικά κάθε έναν από τους όρους της παραπάνω σχέσης.

- ***Static Power :*** Ως στατική ισχύ χαρακτηρίζουμε την ισχύ που καταναλώνεται από μια πύλη όταν αυτή είναι αδρανής ή στατική. Τα CMOS κυκλώματα στην «ιδανική» περίπτωση θεωρείται πως καταναλώνουν μηδενική στατική ισχύ, αφού στην κατάσταση ισορροπίας τους δεν υπάρχει μονοπάτι που να συνδέει την πηγή με την γείωση. Αυτό το σενάριο, βέβαια, δεν υλοποιείται ποτέ στην πραγματικότητα, αφού τα MOS τρανζίστορ δεν αποτελούν τέλειους διακόπτες, και πάντοτε

υπάρχουν, έστω και με πολύ μικρές τιμές ρεύματα διαρροής. Το μεγαλύτερο ποσοστό στατικής ισχύος οφείλεται σε ένα δυναμικό που αναπτύσεται ανάμεσα στην πηγή και την γείωση, το οποίο προκαλείται από ελαττωμένα δυναμικά κατωφλίου τα οποία εμποδίζουν την εκάστοτε πύλη από το να κλείσει εντελώς.

- ***Dynamic Power :*** Η δυναμική ισχύς είναι η ισχύς που καταναλώνεται όταν το κύκλωμα είναι ενεργό. Ένα κύκλωμα είναι ενεργό οποτεδήποτε η τάση σε κάποιο net μεταβάλλεται λόγω της επιβολής στην είσοδο του κάποιου εξωτερικού ερεθίσματος. Με άλλα λόγια η δυναμική ισχύς οφείλεται στην «φόρτιση». Επειδή η τάση στην είσοδο μπορεί να αλλάξει, χωρίς αυτό να συνεπάγεται κάποια λογική μεταβολή στην έξοδο, δυναμική ισχύς καταναλώνεται κι όταν η έξοδος δεν αλλάζει την λογική της κατάσταση. Η δυναμική ισχύς αποτελείται από : α) switching power και β) internal power.

  a. ***Switching Power :*** Switching Power ενός κελιού της σχεδίασης είναι η συνολική ισχύς που καταναλώνεται από την φόρτιση και εκφόρτιση της χωρητικότητας στην έξοδο του κελιού. Η συνολική χωρητικότητα του πυκνωτή στην έξοδο του κελιού είναι το άθροισμα της χωρητικότητας του δικτύου και της πύλης στην έξοδο. Η φόρτιση και η εκφόρτιση είναι αποτέλεσμα των λογικών μεταβάσεων (από 0 σε 1 και αντιστρόφως), επομένως η ισχύς αυξάνεται όσο περισσότερες λογικές μεταβάσεις έχουμε στο κύκλωμά μας. Η σημαντικότητα του switching power γίνεται αντιληπτή από το γεγονός πως αποτελεί το 70% - 90% της συνολικής κατανάλωσης ισχύος σε ένα ενεργό CMOS κύκλωμα.

b. ***Internal Power :*** Ως «εσωτερική» ισχύ χαρακτηρίζουμε οποιαδήποτε ισχύ καταναλώνεται μέσα στα όρια ενός κελιού. Κατά την διάρκεια της λειτουργίας του, ένα ψηφιακό κύκλωμα καταναλώνει εωτερική ισχύ λόγω της συνεχούς φόρτισης και εκφόρτισης των πυκωτών που βρίσκονται στο εσωτερικό των κελιών. Η εσωτερική ισχύς περιλαμβάνει και ένα ποσοστό το οποίο προκαλείται από την στιγμιαία ένωση (δημιουργία μονοπατιού) μεταξύ του P και του N τρανζίστορ μιας πύλης, το ποσοστό αυτό της ισχύος ονομάζεται short-circuit power. Στα περισσότερα κελιά που χρησιμοποιούνται στην εργασία αυτή, το μεγαλύτερο ποσοστό της εσωτερικής ισχύος που καταναλώνεται ωφείλεται στην short-circuit power, γι αυτό και θα δοθεί ιδιαίτερο βάρος σε αυτήν κατά την ανάλυση των πειραματικών μετρήσεων σε επόμενο κεφάλαιο.

c. ***Leakage Power :*** Τα NMOS και PMOS τρανζίστορ που χρησιμοποιούνται στα CMOS κυκλώματα έχουν μη μηδενικά ρεύματα διαρροής και υποκατωφλίου. Τα ρεύματα αυτά μπορούν να «συνεισφέρουν» στην συνολική κατανάλωση ισχύος, ακόμη και όταν το τρανζίστορ δεν παρουσιάζει αλλάγές λογικών τιμών στην λειτουργία του ως διακόπτης. Η υποκατηγορία ισχύος που εξετάζουμε προκαλείται από δυο επιμέρους τύπους ρευμάτων διαρροής :

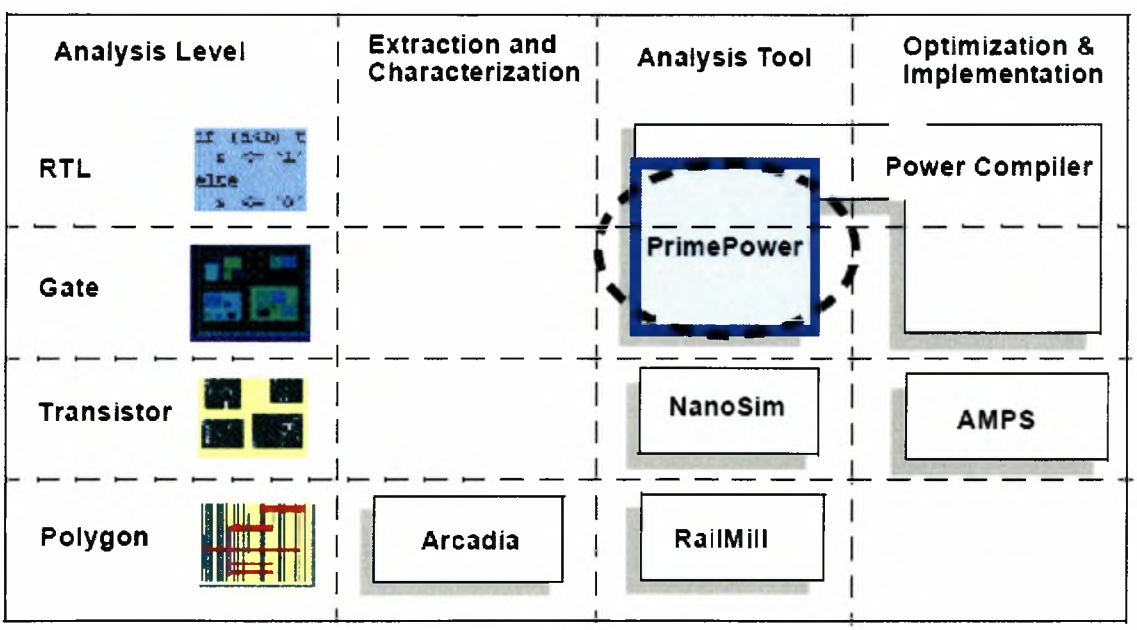
   i. το ρεύμα διαρροής ανάστροφης πόλωσης

   ii. το ρεύμα υποκατωφλίου το οποίο διέρχεται από κάποιο κανάλι του τρανζίστορ, όταν αυτό είναι κλειστό.

3.2 Μεθοδολογία Κατανάλωσης Ισχύος με Χρήση του Prime Power

Το Prime Power είναι ένα εργαλείο ανάλυσης κατανάλωσης ισχύος που λειτουργεί σε επίπεδο πυλών, το οποίο με ακρίβεια υπολογίζει την κατανάλωση ισχύος σε cell – based σχεδιάσεις. Η χρήση του επιβάλλεται σε όσους σχεδιαστές υλοποιούν προϊόντα για power critical εφαρμογές.

Το PP υποστηρίζει τόσο στατική όσο και δυναμική ανάλυση της κατανάλωσης ισχύος σε επίπεδο πυλών. Στην προκείμανη εργασία θα ασχοληθούμε μόνο με την δυναμική ανάλυση. Η εικόνα που ακολουθεί παρουσιάζει τον τρόπο με τον οποίο «εμπλάκεται» το PP στην συνολική ροή της low power σχεδίασης, που προσφέρει το πακέτο της Synopsys που χρησιμοποιούμε.



Σχήμα 22 : Prime Power & Synopsys low power solution

Το πρόγραμμα που εξετάζουμε, δίνει την δυνατότητα στον χρήστη να επιλέξει τον τρόπο λειτουργίας. Έτσι καλούμαστε να επιλέξουμε εάν θα προβούμε σε average analysis, η οποία στηρίζεται σε προεπιλεγμένα

αποτελέσματα όσον αφορά το switching activity και μας δίνει μια εξιδανικευμένη προσέγγιση της συνολικής ισχύος που θα καταναλωθεί, ή σε dynamic (peak power) analysis, όπου λαμβάνουμε μια λεπτομερέστερη εκτίμηση βασιζόμενη σε στοιχεία που εισάγονται από τον χρήστη.

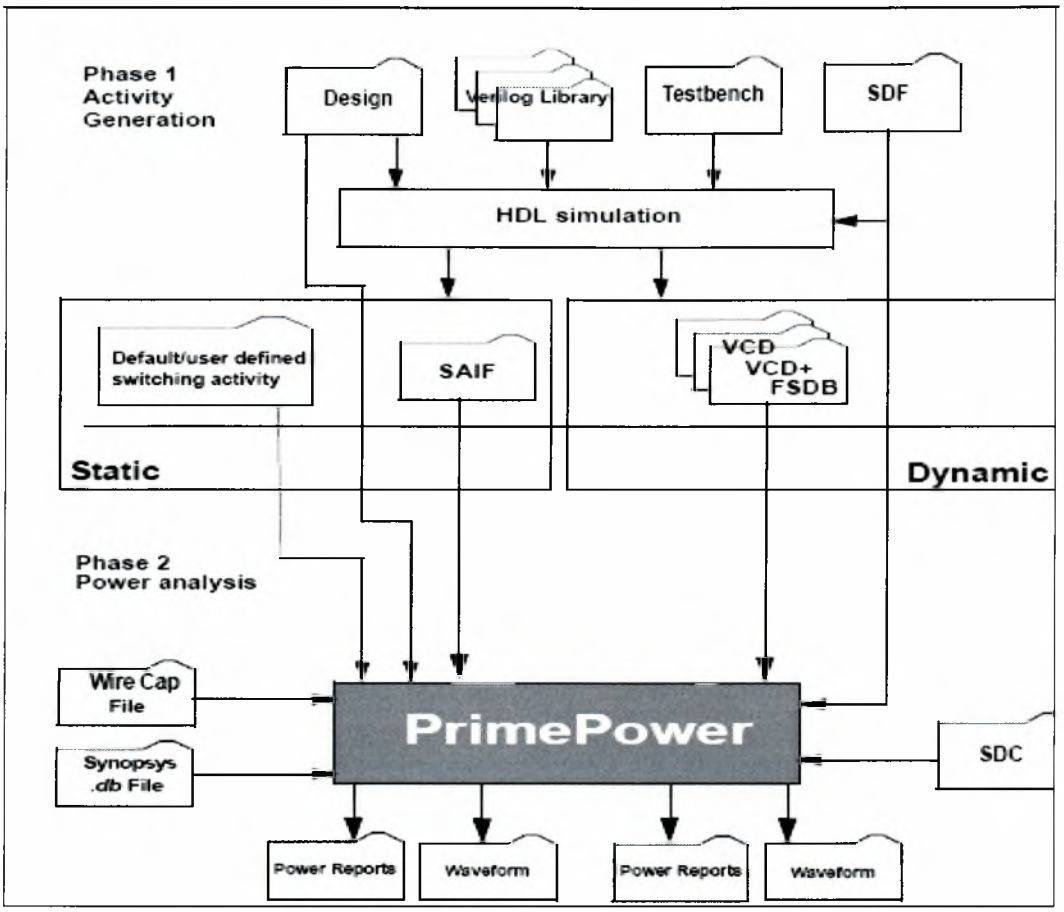
Στο υπόλοιπο μέρος του κεφαλαίου θα περιγράψουμε την ροή προσομοίωσης που ακολουθεί το PP και θα δούμε συνοπτικά τον τρόπο με τον οποίο εκτελεί την ανάλυση καταναλισκόμενης ισχύος.

Prime Power Simulation Flow :

Η ροή που ακολουθεί το PP για την προσομοίωση της λειτουργίας της σχεδίασης πάνω στην οποία θα βασιστεί για την εξαγωγή των ζητούμενων αποτελεσμάτων, αποτελείται από δυο βασικά στάδια :

- την δημιουργία της κατάλληλης πληροφορίας για το switching activity του κυκλώματος, και
- την δημιουργία του power profile της σχεδίασης

Τα στάδια που αναφέραμε παρουσιάζονται στην εικόνα που ακολουθεί, και παρουσιάζονται με περισσότερες λεπτομέριες στο υπόλοιπο κομμάτι του κεφαλαίου.



Σχήμα 23 : Ροή εξομοίωσης της κατανάλωσης ισχύος με χρήση το Prime Power και των VCD αρχείων

Στην πρώτη φάση της εξομοίωσης, στην περίπτωση που θέλουμε να πραγματοποιήσουμε δυναμική ανάλυση για την εξαγωγή αποτελεσμάτων μεγαλύτερης ακρίβειας, καλούμαστε να δώσουμε ως είσοδο στο εργαλείο που χρησιμοποιούμε το gate-level time-based switching activity του κυκλώματος σε μια από τις δυνατές επιλογές που μας δίνονται. Στην παρούσα εργασία δουλέψαμε με βάση τα VCD (Value Change Dump) αρχεία, τα οποία δημιουργήσαμε με χρήση του εργαλείου προσομοίωσης Modelsim. Τα αρχεία αυτά εξάγονται κατά την προσομοίωση του κυκλώματος σε επίπεδο πυλών και περιέχουν μεγάλο κομμάτι από το σύνολο των πληροφοριών για το switching activity που μπορούμε να αντλησουμε για οποιαδήποτε σχεδίαση.

Στην δεύτερη φάση, χρησιμοποιούμε ένα script που περιέχει το σύνολο των εντολών που θέλουμε να εκτελέσει σειριακά ο πυρήνας του Power Compiler ο οποίος καλείται μέσω του Prime Power. Το script που παρουσιάζεται παρακάτω αποτελεί τυπικό παράδειγμα του συνόλου των scripts που χρησιμοποιήθηκαν σε αυτή την εργασία.

```
set search_path                ". . / path1 . . / path2 ."
set link_library               " * library.db"
read_db                        ./ adderi4.db
current_design                 adderi4
link
set_hier_sep /
read_vcd                       -strip_path testbench /dut  ./output.vcd
set_input_transition           0.1 [all_inputs]
read_parasitics                ./ parasitics.reduced.spef
set_waveform_options           -interval 1 -file pwr_vcd -format fsdb
calculate_power                -waveform -reset_neg_power
report_power -file pwr_vcd
report_power -net
report_power -cell
report_power -leaf
```

Ακολούθως θα αναφερθούμε αναλυτικά σε κάθε μια από τις παραπάνω εντολές και την συμβολή της στην εκτέλεση της προσδοκόμενης εργασίας.

Οι μεταβλητές "search_path" και "link_path" περιέχουν ως παραμέτρους τα αρχεία στα οποία το PP θα ψάξει για την εύρεση design data. Πιο συγκεκριμένα τα path που παρουσιάζονται ως παράμετροι του "search_path" περιέχουν όλα εκείνα τα αρχεία που θα χρησιμοποιήσει στην αρχική του φάση το πρόγραμμα, δηλαδή τις σχεδιάσεις σε οποιαδήποτε μορφή και αν βρίσκονται(.vhd, .v, .db) και τις βιβλιοθήκες. Η μεταβλητή "link_path" προσδιορίζει αποκλειστικά και μόνο τις βιβλιοθήκες που θα χρησιμοποιηθούν και θα βοηθήσουν στο να αποφασιστεί ποια elements θα χρησιμοποιηθούν από

την σχεδίασή μας. Το PP ψάχνει για σχεδιαστικά στοιχεία (design elements) στις βιβλιοθήκες με την σειρά που αυτές αναφέρονται στο script.

Στην συνέχεια καλούμαστε να διαβάσουμε στην μνήμη την σχεδίαση πάνω στην οποία θα δουλέψουμε. Επιλέξαμε να διαβάζουμε την σχεδίαση σε μορφή .db αρχείου με την εντολή "read_db". Τα αρχεία .db τα οποία διαβάζει το πρόγραμμα έχουν προεπεξεργαστεί στο Design Vision και παρουσιάζονται υπό την μορφή netlists.

Η εντολή "current_design" θέτει ή ανακτά την τρέχουσα σχεδίαση. Η τρέχουσα σχεδίαση είναι κατ'ουσίαν η σχεδίαση πάνω στην οποία θα εφαρμοστούν όλες οι εντολές που θα επιλέξουμε να διοχετεύσουμε στο Prime Power. Η εντολή αυτή πρέπει να προηγείται πάντα της "link" που ακολουθεί στο παραδειγμά μας.

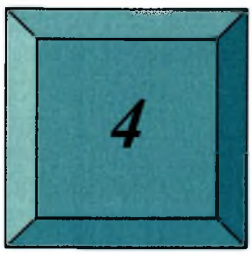
Για να θέσουμε στο προσκήνιο της λειτουργίας του προγράμματος μια σχεδίαση που είναι έτοιμη για power analysis χρησιμοποιούμε την εντολή "link". Με την εντολή αυτή πραγματοποιούνται και οι προσπελάσεις στις βιβλιοθήκες που έχουμε αναφέρει παραπάνω, έτσι ώστε να διαβαστούν και αυτές στην μνήμη του υπολογιστή. Αν δεν έχουμε διαβάσει την σχεδίαση με την εντολή που αναφέραμς στο παραπάνω εδάφιο, τότε το PP με μια διεργασία που ονομάζεται autoload χρησιμοποιεί αυτόματα την εντολή link για τις σχεδιάσεις που έχουν δωθεί ως παράμετρος στην εντολή "search_path".

Συνεχίζοντας, προσδιορίζουμε την θέση του testbench, που περιέχει τις κατάλληλες αναφορές στην σχεδίαση πάνω στην οποία δουλεύουμε, και του VCD αρχείου που έχουμε εξάγει σε προγενέστερη φάση της συνολικής διεργασίας με την εντολή "read_vcd".

Με την εντολή "set_input_transition" προσιορίζουμε έν transition time για κάθε port της σχεδίασης. Το Prime Power χρησιμοποιεί την πληροφορία αυτή κατά τον υπολογισμό της ισχύος της λογικής που οδηγείται από το εκάστοτε port.

Με την επόμενη εντολή διαβάζουμε τα παρασιτικά που έχουμε εξάγει κατά την σύνθεση του κυκλώματος τα οποί επίσης παίζουν σημαντικό ρόλο στην διαδικασία υπολογισμού της συνολικής ισχύος.

Τέλος, αφού δώσουμε την εντολή υπολογισμού της καταναλισκόμενης ισχύος "calculate_power", με χρήση της εντολής "report" και μιας σειράς παραμέτρων παίρνουμε αναλυτικές αναφορές για την ισχύ που καταναλώνεται τόσο στο σύνολο της σχεδιασής μας όσο και σε κάθε net, cell και leaf. Με τον τρόπο αυτό μπορούμε να οπισθοδρομήσουμε στην συνολική σχεδιαστική εργασία τροποποιώντας ανάλογα ορισμένα components της συνολικής σχεδίασης των οποίων η συμπεριφορά δεν μας ικανοποιει.



4 Πειραματικές Μετρήσεις και Αποτελέσματα

4.1 Συγκριτικά Αποτελέσματα και Συμπεράσματα

Στο κεφάλαιο αυτό θα παρουσιάσουμε μια σειρά αποτελεσμάτων τα οποία προέκυψαν μέσω των διεργασιών που αναφέρθηκαν σε προηγούμενες ενότητες της παρούσας εργασίας. Τα προκείμενα αποτελέσματα αναφέρονται κατά κύριο λόγο σε αθροιστές μήκους 32 – bit, αφού αυτή η κατηγορία αθροιστών χρησιμοποιείται στην πλειοψηφία των περιπτώσεων για την υλοποίηση μεγάλων σχεδιάσεων.

Τα «μετρικά» στοιχεία που χρησιμοποιούνται για τις συγκρίσεις είναι : το γινόμενο της καταναλισκώμενης ισχύος επί την συνολική καθυστέρηση επί την περιοχή που καταλαμβάνει η σχεδίαση(**P**ower **D**elay **A**rea product - PDA) και τα επιμέρους στοιχεία του παραπάνω γινομένου που αφορούν την κατανάλωση ισχύος και την καθυστέρηση εμφάνισης του αποτελέσματος στην έξοδο του κυκλώματος.

Στο Παράρτημα Α βρίσκονται συγκεντρωμένα όλα τα αποτελέσματα που ελήφθησαν με χρήση του πακέτου της Synopsys. Μπορούμε εύκολα να παρατηρήσουμε πως όπως ήταν αναμενόμενο σε όλες τις τοπολογίες αθροιστών που εξετάστηκαν, τόσο η καθυστέρηση εμφάνισης του αποτελέσματος, όσο και η ισχύς που καταναλώνεται και η περιοχή που καλείπτεται αυξάνεται σε κάθε αθροιστή όσο αυξάνεται το μήκος των τελεστών του.

Carry Select				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.7	1.31	2.54	5
combinational area	77.760002	1156.032471	2289.600098	4556.730469
total power	3.03E-07	4.55E-04	6.93E-04	3.50E-04
PDA	1.65092E-06	0.689356035	4.031363001	7.96516486

Σχήμα 24 : Πίνακας Αποτελεσμάτων του Carry Select με χρήση της βιβλιοθήκης UMC_13_bc

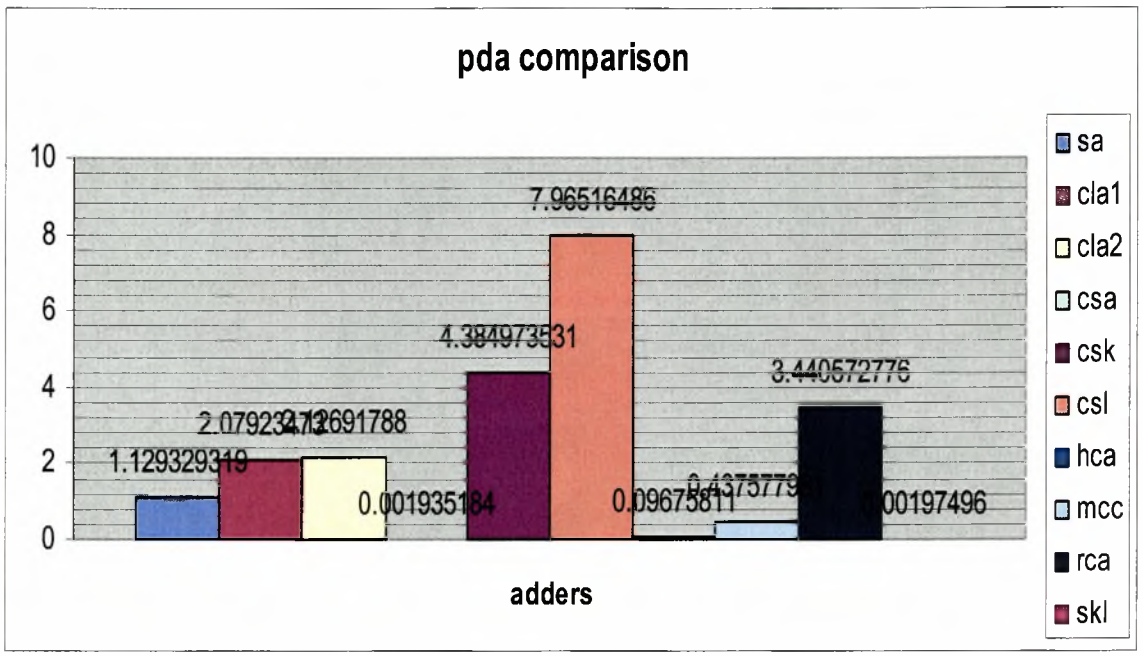
Στην περίπτωση του Carry Save Adder που παρουσιάζεται στον παραπάνω πίνακα, παρατηρούμε πως όσον αφορά την περιοχή που καταλαμβάνει η σχεδίαση, αυτή διπλασιάζεται με τον διπλασιασμό του μεγέθους. Έχουμε, δηλαδή, ανάλογη μεταβολή σε αυτά τα μεγέθη. Αύξηση παρουσιάζεται, επίσης, τόσο στο χρόνο που δαπανάται για τον υπολογισμό του αποτελέσματος, όσο και στο σύνολο της καταναλισκώμενης ισχύος. Στο σημείο αυτό πρέπει να αναφέρουμε πως όλες η μονάδα μέτρησης του χρόνου που χρησιμοποιήθηκε είναι το 1ns και η μονάδα μέτρησης της ισχύος το 1 Watt.

Σύγκριση βάσει του γινομένου PDA :

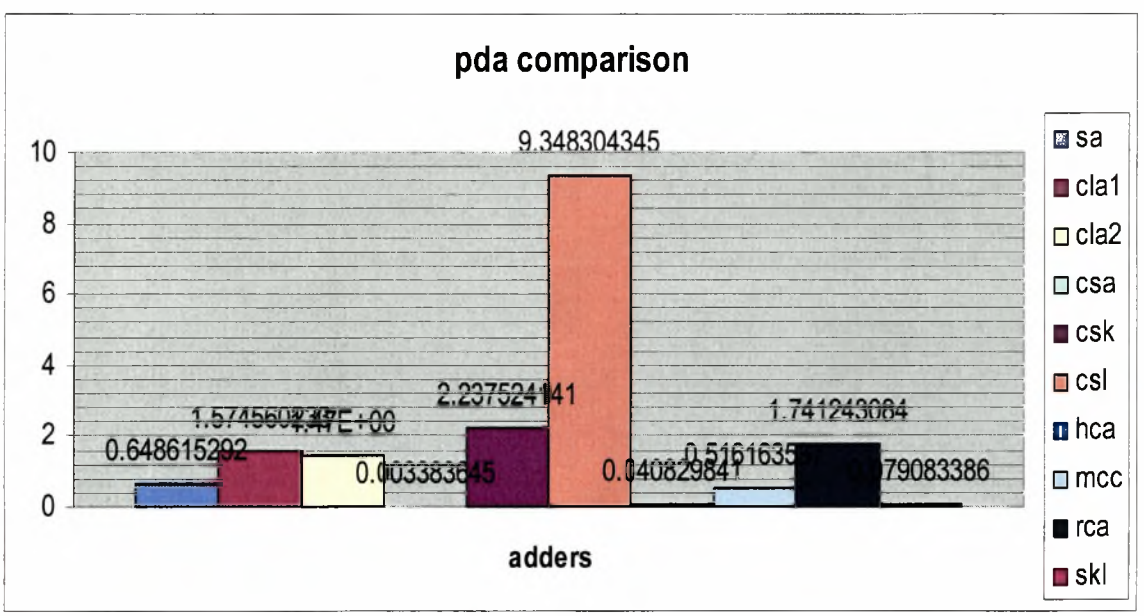
Στην επόμενη σελίδα παρουσιάζονται τα PDA διαγράμματα που δημιουργηθήκαν με βάση τις μετρήσεις του παραρτήματος, για ένα σύνολο τεσσάρων διαφορετικών βιβλιοθηκών.

Θεωρητικά κάθε ένα από τα «συστατικά» στοιχεία του PDA γινομένου πρέπει να έχει την ελάχιστη δυνατή τιμή, έτσι ώστε το συνολικό PDA ποσοστό ενός αθροιστή να υπερτερεί ενός έταιρου αθροιστή και να έχουμε ένα κριτήριο για την σύγκρισή τους. Ένα αφελές συμπέρασμα που μπορούμε να εξάγουμε είναι πως ο αθροιστής με το μικρότερο γινόμενο μπορεί να θεωρηθεί, κατά κάποιο τρόπο, «καλύτερος» ή «καταλληλότερος» των υπολοίπων για την χρήση του στην σχεδίαση ενός μεγαλύτερου κυκλώματος.

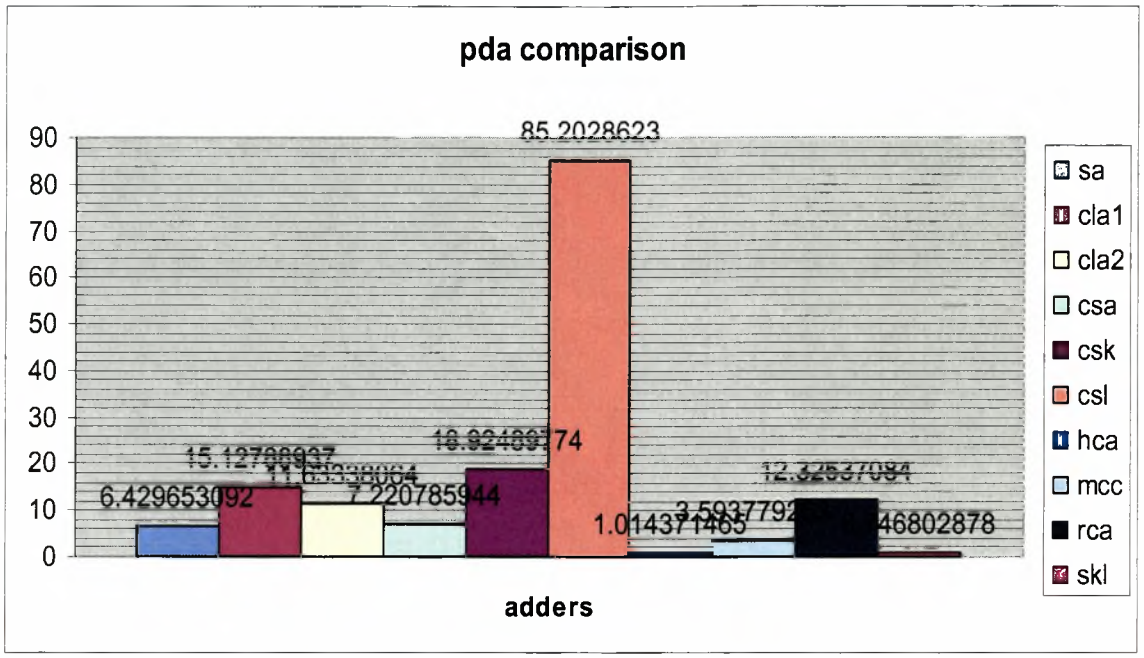
Από αυτή την σκοπιά, μπορούμε να παρατηρήσουμε πως οι tree adders παρουσιάζουν κατά κανόνα μικρότερο PDA γινόμενο από τους carry propagate adders.



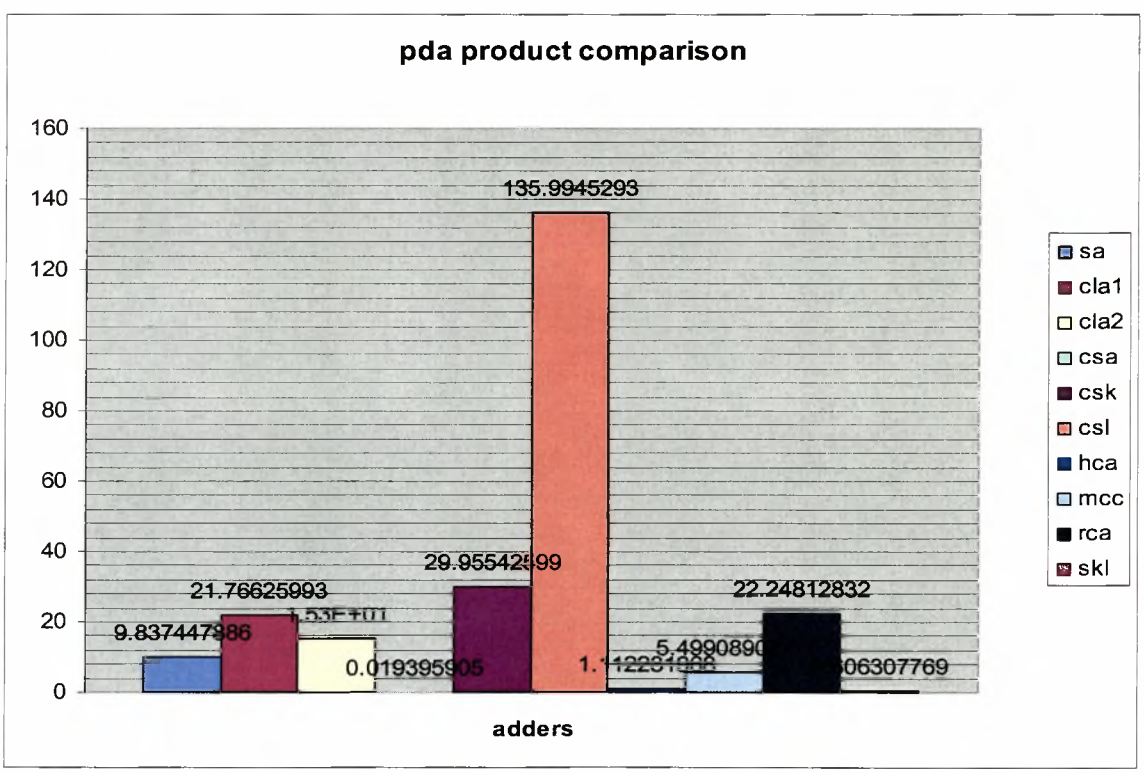
Σχήμα 25 : Σύγκριση γινομένου PDA με χρήση της βιβλιοθήκης UMC_13_bc



Σχήμα 26 : Σύγκριση γινομένου PDA με χρήση της βιβλιοθήκης UMC_13_wc



Σχήμα 27 : Σύγκριση γινομένου PDA με χρήση της βιβλιοθήκης UMC_18_bc



Σχήμα 28 : Σύγκριση γινομένου PDA με χρήση της βιβλιοθήκης UMC_13_wc

Ας εξετάσουμε, όμως, πρακτικά την σημασία των αποτελεσμάτων. Στην πλειοψηφία των σχεδιάσεων, χρειαζόμαστε αθροιστές «ειδικού σκοπού», αθροιστές, δηλαδή, με συγκεκριμένες απαιτήσεις, όσον αφορά ορισμένους μόνο από τους παράγοντες στους οποίους αναφερθήκαμε παραπάνω.

Επι παραδείγματει, ο Carry Skip Adder στα 32 - bit, ο οποίος παρουσιάζεται στον ακόλουθο πίνακα, σύμφωνα με τα αποτελέσματά μας, παρουσιάζει αρκετά μεγάλη PDA τιμή, ανεξαρτήτως της βιβλιοθήκης που χρησιμοποιούμε. Η τιμή αυτή, όμως, οφείλεται κατά το μεγαλύτερο ποσοστό της στο γεγονός πως ο αθροιστής αυτός καταλαμβάνει μεγάλη επιφάνεια. Ο χώρος που καταλαμβάνει, όμως, δεν αποτελεί πλέον πρόβλημα, λόγω του ότι η σύγχρονη σχεδίαση ψηφιακών κυκλωμάτων, κίνειται με άξονα την βελτίωση της κατανάλωσης και της ταχύτητάς του.

Carry Skip				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	()	5.11
combinational area	()	()	()	5334.047852
total power	()	()	()	1.10E-03
PDA	()	()	()	29.95542599

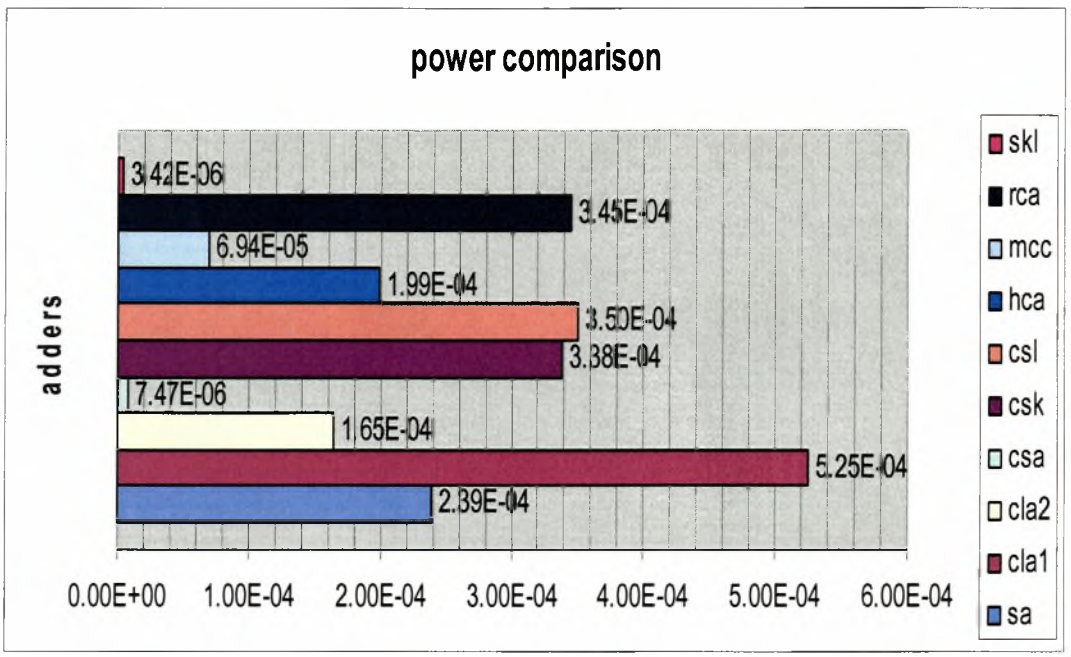
Σχήμα 29 : Πίνακας αποτελεσμάτων του Carry Skip με χρήση της βιβλιοθήκης UMC_18_bc

Καταλήγουμε, επομένως, στο συμπέρασμα πως η PDA τιμή μπορεί να μας δώσει μόνο μια πρόχειρη μόνο εντύπωση της ποιότητας του κυκλώματός μας και όχι μια de facto αναγνώριση της καταλληλότητάς του για την σχεδίασή μας.

Σύγκριση βάσει της καταναλισκώμενης ισχύος :

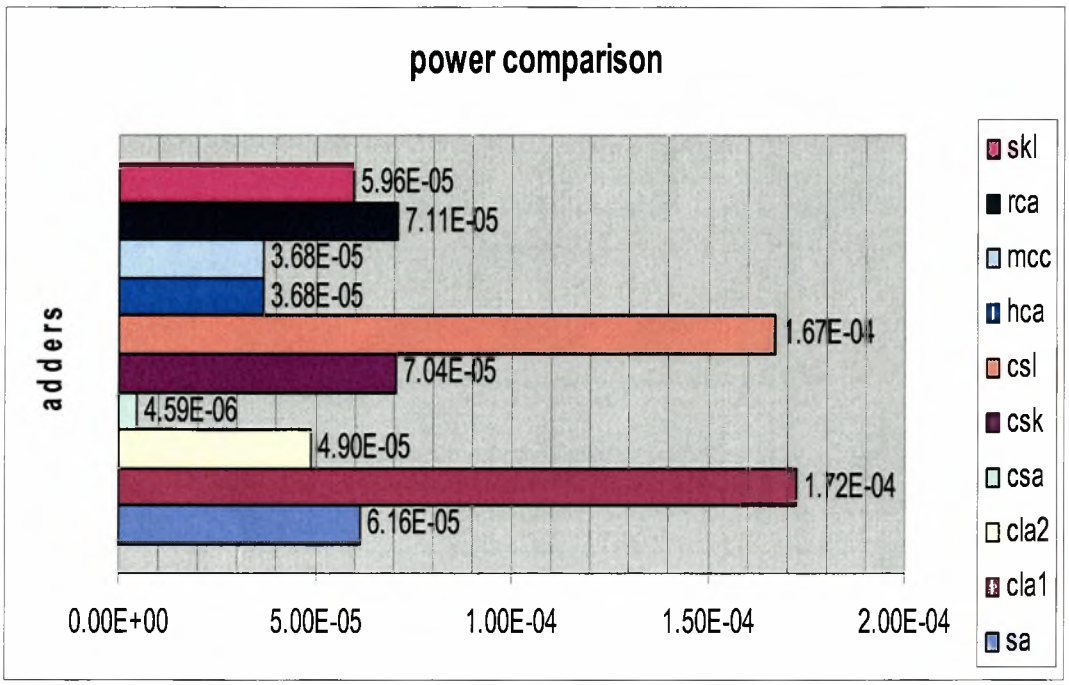
Ας εξετάσουμε, τώρα, τις διαφορές που παρουσιάζουν οι αθροιστές στα 32 - bit σε σχέση με την συνολική κατανάλωση ενέργειας του καθενός. Στα

παρακάτω διαγράμματα παρουσιάζονται τα αποτελέσματα και για τις τέσσερις βιλιοθήκες που παρουσιάστηκαν.



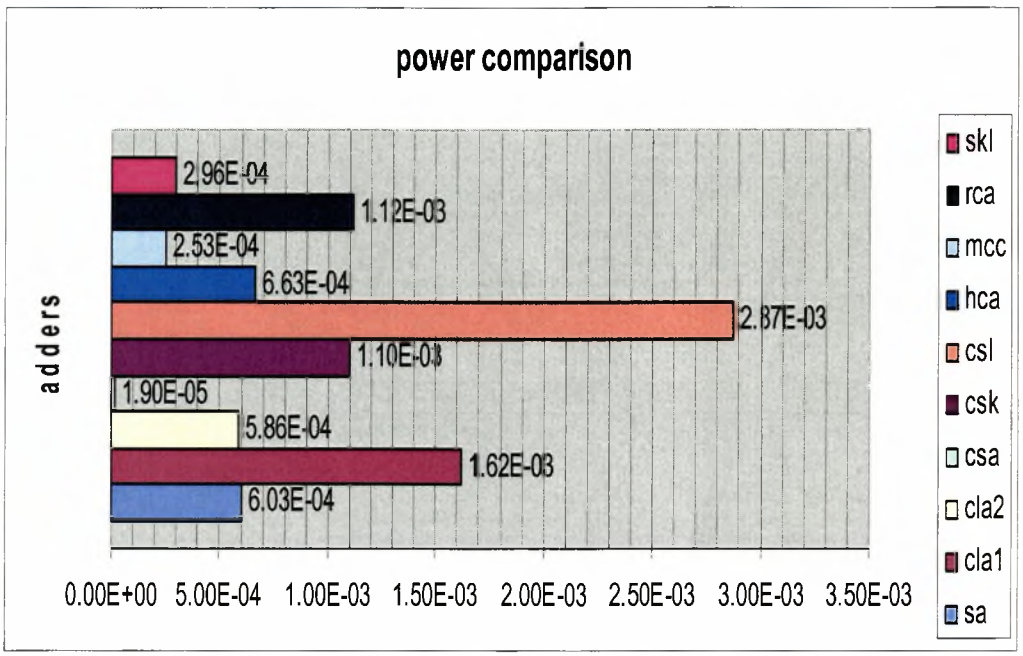
Σχήμα 30 : Κατανάλωση ισχύος με χρήση της βιβλιοθήκης UMC_13_bc

Στο πρώτο σχήμα, παρατηρούμε πως οι tree adders καταναλώνουν λιγότερη ισχύ από τους carry propagate adders με διαφορά που μπορεί να αγγίξει ακόμα και μια τάξη μεγέθους όπως στην περίπτωση του Sklansky Adder. Από εκεί και πέρα όπως αναμένεται ο Carry Lookahead, μια από τις πρώτες παραλλαγές όσον αφορά την τοπολογία ενός αθροιστή, κατανάλωνει την περισσότερη ισχύ σε σχέση με τους υπόλοιπους αθροιστές.



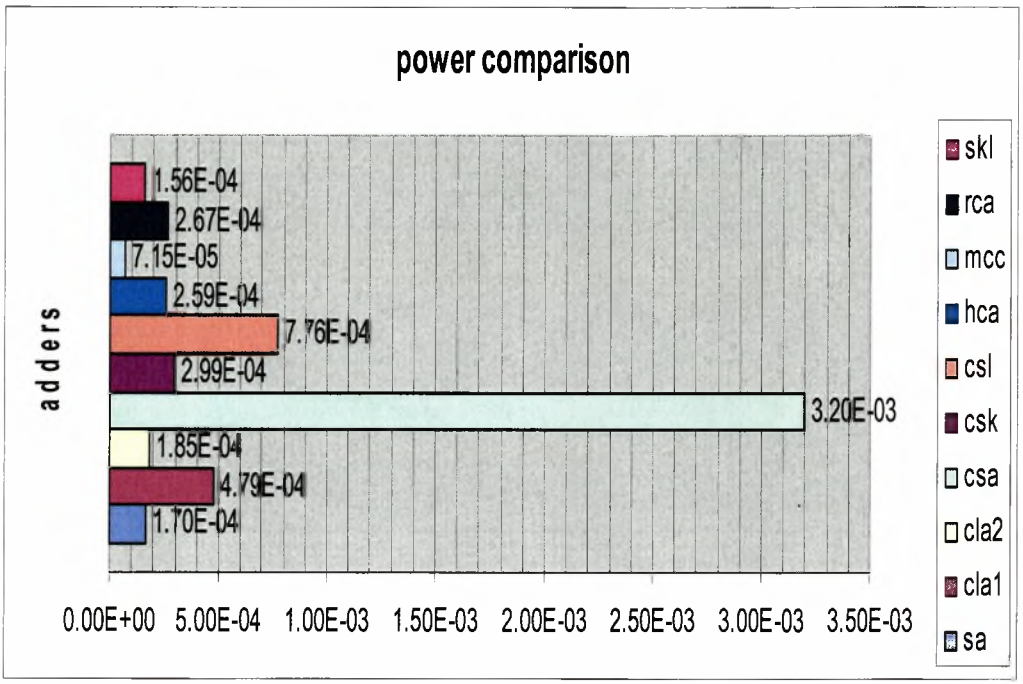
Σχήμα 31 : Κατανάλωση ισχύος με χρήση της βιβλιοθήκης UMC_13_wc

Στην δεύτερη περίπτωση, με χρήση της εκδοχής χειρότερης περίπτωσης της ίδιας βιβλιοθήκης σχεδιαστικών δεδομένων, και πάλι ο Carry Lookahead όπως και ο Carry Select καταναλώνουν ισχύ υπερδιπλάσια από τον επόμενο τι τάξει αθροιστή. Οι tree adders και πάλι εμφανώς είναι πιο συμφέρουσα επιλογή από ενεργειακής πλευράς, η διαφορά τους, όμως από τις κλασσικές τοπολογίες μειώνεται αισθητά σε σχέση με την προηγούμενη περίπτωση, αρκεί να δούμε πως η διαφορά μεταξύ Sklansky και Carry Lookahead μειώθηκε από 0.00521183 W σε 0.00009264 W.



Σχήμα 32 : Κατανάλωση ισχύος με χρήση της βιβλιοθήκης UMC_18_bc

Στις μετρήσεις που πραγματοποιήσαμε με την βοήθει της βιβλιοθήκης UMC_18_bc, παρατηρούμε πως όπως και στις προηγούμενες των περιπτώσεων οι αθροιστές Carry Select και Carry Lookahead είναι αυτοί που έχουν την μεγελύτερη κατανάλωση ενέργειας. Αναλογιζόμενοι τα αποτελέσματα σε πιο αφαιρετικό επίπεδο βλέπουμε πως αν και η αλγεβρική διαφορά των αποτελεσμάτων μειώνεται παρόλαυτα οι tree adders εμφανίζονται να έχουν καλύτερη συμπεριφορά έναντι των carry propagate adders. Τέλος, γίνεται αντιληπτό πως στην κατηγορία των «κλασσικών» αθροιστών οι πιο «προσφατες» υλοποιήσεις και τοπολογίες επιδεικνύουν καλύτερη απόδοση σε σχέση με τις πρώτες παραλλαγές αθροιστών που εμφανίστηκαν, αποδεικνύοντας και πρακτικά την συνολική εξέλιξη στον τομέα αυτό.

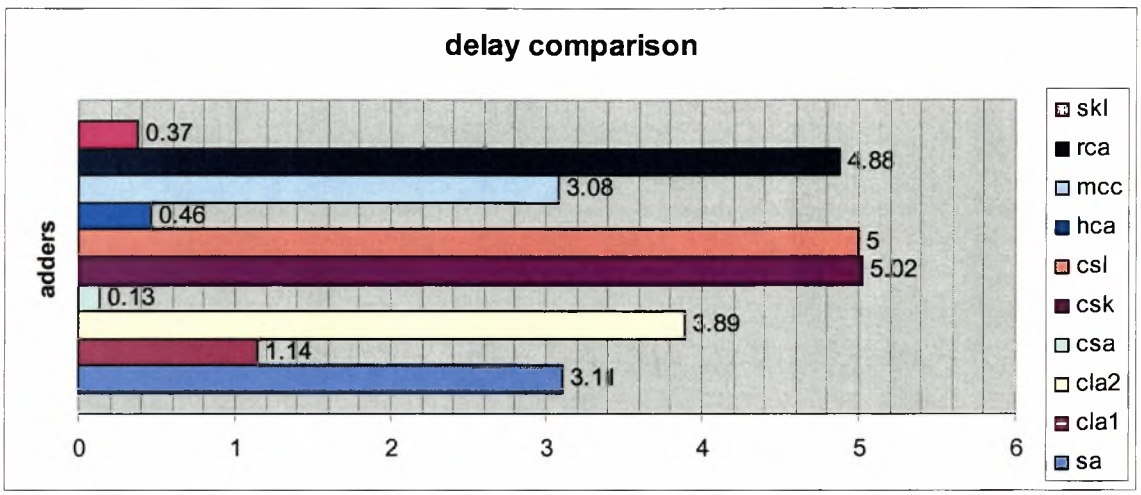


Σχήμα 33 : Κατανάλωση ισχύος με χρήση της βιβλιοθήκης UMC_18_wc

Στην κατηγορία αυτή όπου επιδεικνύεται η χειρότερη συμπεριφορά της βιβλιοθήκης που χρησιμοποιούμε. Όπως γίνεται αντιληπτό από το παραπάνω σχήμα που περιέχει την συνοπτική παρουσίαση των αποτελεσμάτων, οι μεγάλες διαφορές που παρατηρήθηκαν στις προηγούμενες περιπτώσεις εκλείπουν. Εντούτοις και πάλι οι tree adders εμφανίζουν καλύτερα αποτελέσματα. Οι συγκρίσεις, όμως, με ορισμένους από τους «κλασσικούς» αθροιστές μας δείχνουν πως δεν υφίσταται πλέον η έννοια της «καλύτερης επιλογής» (για παράδειγμα η απόδοση ενός πλήρους αθροιστη (full adder) είναι συγκρίσιμη με το σύνολο των αθροιστών). Στην περίπτωση αυτή, εμπλέκονται στην διαδικασία της σύγκρισης δευτερεύοντες παράγοντες που συνεπικουρούν στην λήψη απόφασης για την χρήση ενός εκ των παραπάνω αθροιστών.

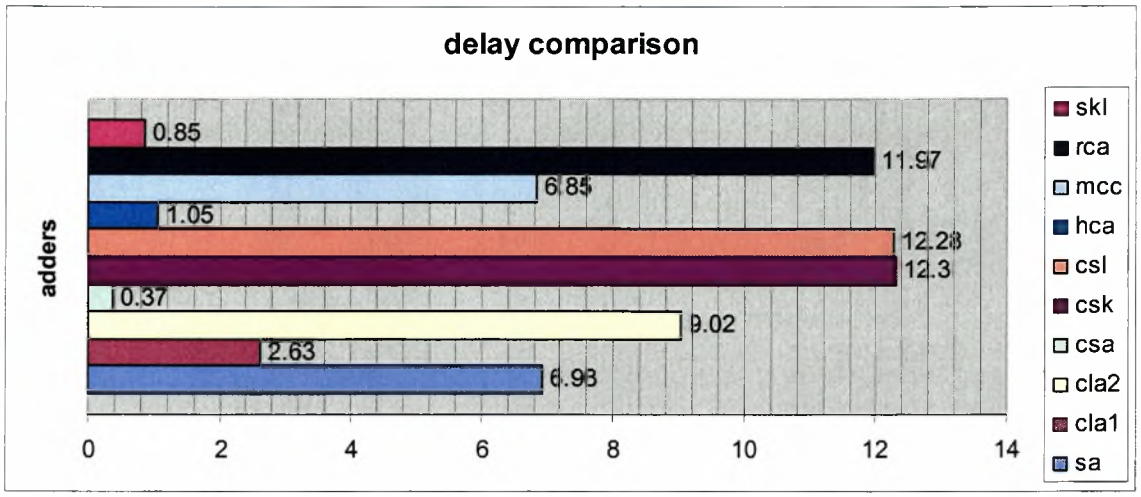
Σύγκριση βάσει της συνολικής καθυστέρησης :

Στις ακόλουθες σελίδες παραθέτουμε τους πίνακες με τα συγκεντρωτικά αποτελέσματα των μετρήσεων που αφορούν την συνολική καθυστέρηση που παρατηρείται στους αθροιστές 32 – bit που εξετάσαμε.



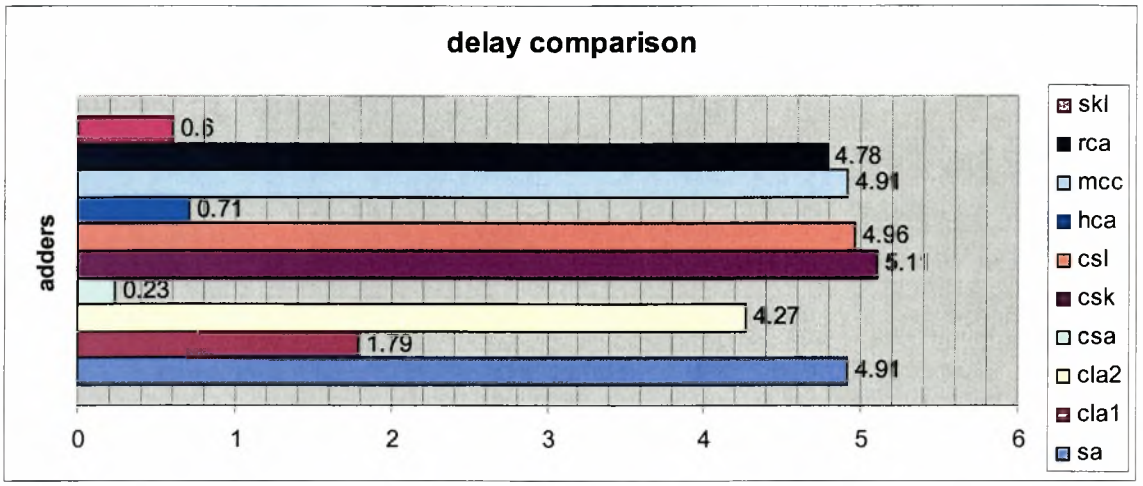
Σχήμα 34 : Συνολική καθυστέρηση με χρήση της βιβλιοθήκης UMC_13_bc

Στο παραπάνω σχήμα, βλέπουμε τα αποτελέσματα που προέκυψαν μετά την επεξεργασία των κυκλωματικών στοιχείων από το Design Vision. Παρατηρούμε πως οι δενδρικές δομές των αθροιστών παρουσιάζουν μικρό critical path και επομένως μικρή καθυστέρηση κατά την εμφάνιση του ζητούμενου αποτελέσματος στην έξοδο του κυκλώματος. Οι csk, csl και rca αθροιστές παρουσιάζουν τις μεγαλύτερες καθυστερήσεις. Ειδικότερα, ο csl και ο csk στις υλοποιήσεις τους εμπεριέχουν ως δομικό στοιχείο και έναν πολυπλέκτη. Σε αυτό το επιπλέον στάδιο οφείλεται κατά κύριο λόγο η επιπλέον καθυστέρηση που παρατηρείται.

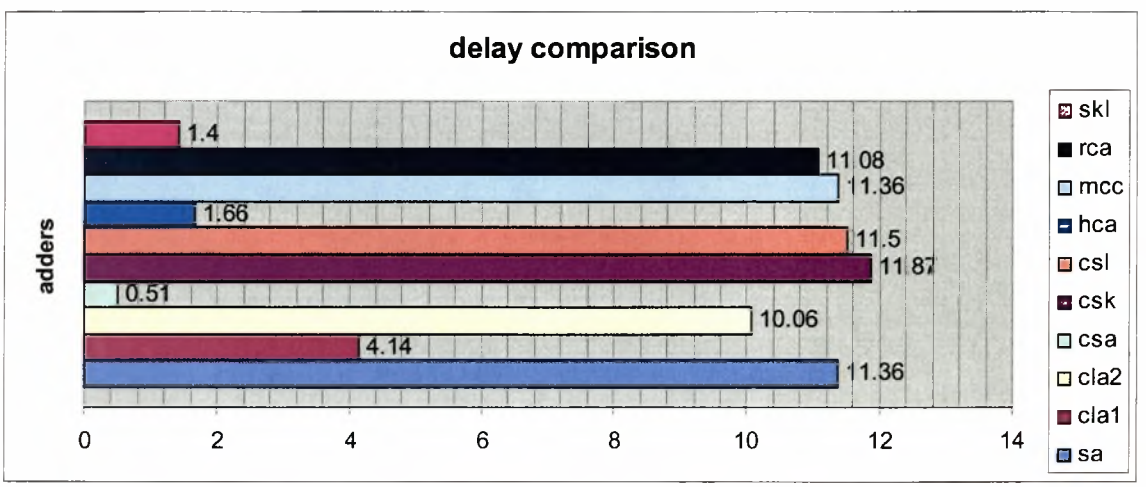


Σχήμα 35 : Συνολική καθυστέρηση με χρήση της βιβλιοθήκης UMC_13_wc

Η χρησιμοποίηση της χειρότερης περίπτωσης της βιβλιοθήκης UMC_13 μας δίνει τα αποτελέσματα που εμφανίζονται στο Σχήμα 35. Και πάλι οι csk και csl παρουσιάζουν την μεγαλύτερη καθυστέρηση σε σχέση με τους υπόλοιπους αθροιστές, ενώ οι αλγεβρικές τους διαφορές με τους tree adders παραμένουν στα ίδια επίπεδα. Στο σημείο αυτό, όμως, πρέπει να αναφερθούμε στην εναλλακτική υλοποίηση του Carry Lookahead η οποία χρησιμοποιήθηκε στις μετρήσεις και η οποία παρόλο που παρουσιάζει καλά αποτελέσματα ως προς την συνολική που καταναλώνεται εντούτοις παρουσιάζει μεγάλες καθυστερήσεις στην απόκρισή της όπως άλλωστε φαίνεται και από τα αποτελέσματα που παρουσιάζονται στους πίνακες αυτής της παραγράφου. Η εναλλακτική αυτή υλοποίηση αποτελεί βασικό παράδειγμα των tradeoffs μεταξύ ισχύος και καθυστέρησης που καλείται να επιλύσει ο σχεδιαστής.



Σχήμα 36 : Συνολική καθυστέρηση με χρήση της βιβλιοθήκης UMC_18_bc



Σχήμα 37 : Συνολική καθυστέρηση με χρήση της βιβλιοθήκης UMC_18_wc

Τα συμπεράσματα, τώρα, που προκύπτουν για τις δυο βιβλιοθήκες της UMC στα 18 um είναι παραπλήσια με αυτά που αναφέρθηκαν στις δυο προηγούμενες παραγράφους. Ένα στοιχείο το οποίο δεν πρέπει να παραλήψουμε είναι το γεγονός πως όσον αφορά τον Manchester Carry Chain Adder η καθυστέρηση που παρατηρείται αυξάνεται όσο «μεγαλώνει» και

«χειροτερεύει» η τεχνολογία που χρησιμοποιούμε, όπως ακριβώς συμβαίνει και στη περίπτωση του πλήρους αθροιστή (full adder).

Στο τελευταίο τμήμα αυτής της ενότητας θα εξετάσουμε την συμπεριφορά δυο αθροιστών που ο καθένας ανήκει σε μια από τις δυο βασικές κατηγορίες που έχουμε αναφέρει. Οι αθροιστές αυτοί (ο Manchester Carry Chain και ο Han Carlson) παρουσιάζονται στους ακόλουθους πίνακες μαζί με τις τιμές των βασικών κριτηρίων σύγκρισης που έχουμε χρησιμοποιήσει εώς τώρα.

Manchester Carry Chain Adder					
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_13_bc
critical path	0.47	0.84	1.59	3.08	
combinational area	255.744	511.4879	1022.975	2045.95	
total power	2.96E-05	3.72E-05	7.92E-05	6.94E-05	
PDA	0.003553	0.015987	0.128789	0.437578	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_13_wc
critical path	1.05	1.88	3.54	6.85	
combinational area	255.744	511.4879	1022.975	2045.95	
total power	1.13E-05	1.73E-05	2.79E-05	3.68E-05	
PDA	0.003037	0.016674	0.100999	0.516164	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_18_bc
critical path	0.75	1.34	2.53	4.91	
combinational area	552.9119	1105.824	2211.647	4423.292	
total power	8.27E-05	7.01E-05	1.42E-04	2.53E-04	
PDA	0.03429	0.103919	0.794556	5.499089	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_18_wc
critical path	1.72	3.1	5.85	11.36	
combinational area	552.9119	1105.824	2211.647	4423.292	
total power	6.15E-05	4.53E-05	7.92E-05	7.15E-05	
PDA	0.058458	0.155394	1.02483	3.593779	

Σχήμα 38 : Συνολικά αποτελέσματα του Manchester Carry Chain Adder

Han Carlson Adder					
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_13_bc
critical path	0.38	0.71	0.98	1.18	
combinational area	255.744003	514.944336	1245.888428	2897.850586	
total power	1.95E-05	1.75E-05	2.05E-05	2.35E-05	
PDA	0.001893119	0.006394527	0.025029899	0.080425786	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_13_wc
critical path	0.98	1.82	2.47	2.83	
combinational area	273.024017	540.864319	1228.608398	2932.410645	
total power	6.26E-06	9.62E-06	1.49E-05	2.49E-05	
PDA	0.001674948	0.009473606	0.045095088	0.206638181	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_18_bc
critical path	0.46	0.86	1.07	1.26	
combinational area	552.916992	1101.767944	2622.285156	6439.853516	
total power	4.54E-05	1.46E-05	5.06E-05	4.34E-05	
PDA	0.011544575	0.013833798	0.142087997	0.352319234	
datawidth	4 bit	8 bit	16 bit	32 bit	UMC_18_wc
critical path	1.07	2.01	2.6	3.02	
combinational area	536.655029	1085.506104	2638.549316	6358.550781	
total power	1.81E-05	9.86E-06	1.39E-05	4.41E-05	
PDA	0.010364687	0.021502302	0.095425775	0.846076397	

Σχήμα 39 : Συνολικά αποτελέσματα του Han Carlson Adder

Όσον αφορά, τώρα, τα συμπεράσματα που μπορούμε να εκμαιεύσουμε από τα δεδομένα αυτά μπορούμε να αναφέρουμε χαρακτηριστικά τα ακόλουθα :

- Σε κάθε μια από τις τέσσερις βιβλιοθήκες, κάθε μια από τις ποσότητες που εξετάζουμε αυξάνεται κατά την αύξηση του μήκους των τελούμενων.

- Οι ποσότητες που εξετάζουμε αυξάνονται όσο «χειροτερεύει» και «μεγαλώνει» η τεχνολογική βιβλιοθήκη που χρησιμοποιούμε.

- Σε κάθε «στατιστική» κατηγορία από αυτές που εξετάζουμε οι tree adders «υπερτερούν» των carry propagate adders.

4.2 Μελλοντικές επεκτάσεις

Όπως είδαμε στο προηγούμενο κεφάλαιο, καταλήξαμε σε ορισμένα χρήσιμα συμπεράσματα για την πλειοψηφία των αθροιστών που χρησιμοποιήθηκαν στις μετρήσεις αυτής της εργασίας. Εντούτοις, οι συνθήκες στις οποίες πραγματοποιήθηκαν οι πειραματικές μετρήσεις ήταν «γενικού σκοπού». Μια σειρά από μετρήσεις σε πιο ακραίες συνθήκες θα οδηγούσε σε ακόμα πιο ακριβή συμπεράσματα για την πραγματική συμπεριφορά των κυκλωμάτων. Ενώ η πραγματοποίηση νέων μετρήσεων σε επίπεδο layout θα οδηγούσε στην ανάκτηση τιμών οι οποίες προσεγγίζουν σε πολύ μεγαλύτερο βαθμό αυτές που παρατηρούνται κατά την φάση της κατασκευής ολοκληρωμένων σχεδιάσεων.

Πίνακες Αποτελεσμάτων

UMC library 13 best case:

Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.37	0.76	1.54	3.11
combinational area	165.888031	359.424072	746.495178	1520.638184
total power	5.60E-07	1.42E-04	2.09E-04	2.39E-04
PDA	3.43659E-05	0.038789046	0.240496859	1.129329319

Carry Lookahead					
datawidth	4 bit	8 bit	16 bit	32 bit	(*)
critical path	0.43	1	()	1.14	3.89
combinational area	172.800034	929.664429	()	3476.725342	3317.7519
total power	4.12E-07	4.37E-04	()	5.25E-04	1.65E-04
PDA	3.06207E-05	0.406170389	()	2.07923473	2.126917

Carry Save				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.13	0.13	0.13	0.13
combinational area	252.28804	501.119598	998.783081	1994.109497
total power	9.53E-07	2.22E-06	2.16E-06	7.47E-06
PDA	3.12396E-05	0.000144558	0.000280978	0.001935184

Carry Skip				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	()	5.02
combinational area	()	()	()	2585.086426
total power	()	()	()	3.38E-04
PDA	()	()	()	4.384973531

Carry Select				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.07	1.31	2.54	5
combinational area	77.760002	1156.032471	2289.600098	4556.730469
total power	3.03E-07	4.55E-04	6.93E-04	3.50E-04
PDA	1.65092E-06	0.689356035	4.031363001	7.96516486

DesignWare Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.72	0.78	()	()
combinational area	165.888016	235.008026	()	()
total power	1.60E-05	8.12E-05	()	()
PDA	0.001905058	0.014878969	()	()

DesignWare Carry Save Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.11	0.13	()	()
combinational area	255.744003	235.008026	()	()
total power	5.67E-07	6.98E-07	()	()
PDA	1.59592E-05	2.13185E-05	()	()

Elm				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	0.66	()
combinational area	()	()	1783.296631	()
total power	()	()	6.37E-04	()
PDA	()	()	0.75020436	()

Han - Carlson				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.07	0.07	0.07	0.46
combinational area	27.648001	55.296001	110.591995	1057.535034
total power	1.64E-07	2.56E-07	5.15E-07	1.99E-04
PDA	3.17206E-07	9.89743E-07	3.98839E-06	0.09675811

Manchester Carry Chain Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.47	0.84	1.59	3.08
combinational area	255.744049	511.487946	1022.97522	2045.950073
total power	2.96E-05	3.72E-05	7.92E-05	6.94E-05
PDA	0.003553103	0.015987272	0.128788693	0.437577981

Ripple Carry Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.57	1.19	2.42	4.88
combinational area	255.744003	511.488159	1022.97644	2045.953003
total power	1.57E-04	2.35E-04	3.14E-04	3.45E-04
PDA	0.022930263	0.143098531	0.777586898	3.440572776

Sklansky				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.37	0.37	0.37	0.37
combinational area	207.360031	400.896027	787.967224	1562.110107
total power	5.71E-07	8.62E-07	1.77E-06	3.42E-06
PDA	4.38013E-05	0.000127802	0.000516623	0.00197496

UMC library 13 worst case:

Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.82	1.7	3.44	6.93
combinational area	165.888031	359.424072	746.495178	1520.638184
total power	2.88E-05	4.28E-05	5.32E-05	6.16E-05
PDA	0.003920332	0.026127255	0.13661459	0.648615292

Carry Lookahead 1					
datawidth	4 bit	8 bit	16 bit	32 bit	(*)
critical path	0.97	2.31	()	2.63	9.02
combinational area	172.800034	929.664429	()	3476.725342	3317.751953
total power	6.49E-06	1.40E-04	()	1.72E-04	4.90E-05
PDA	0.001087158	0.301082981	()	1.574560233	1.47E+00

Carry Save				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.34	0.35	0.36	0.37
combinational area	252.28804	501.119598	998.783081	1994.109497
total power	5.83E-07	1.29E-06	2.57E-06	4.59E-06
PDA	5.00428E-05	0.000226606	0.000922636	0.003383645

Carry Skip				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	()	12.3
combinational area	()	()	()	2585.086426
total power	()	()	()	7.04E-05
PDA	()	()	()	2.237524141

Carry Select				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.67	3.19	6.22	12.28
combinational area	589.248169	1156.032471	2289.600098	4558.458496
total power	7.05E-05	1.42E-04	1.48E-04	1.67E-04
PDA	0.069404655	0.524765912	2.107714266	9.348304345

DesignWare Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.81	1.87	()	()
combinational area	165.888016	235.008026	()	()
total power	6.11E-06	4.40E-05	()	()
PDA	0.001835173	0.019314487	()	()

DesignWare Carry Save Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.26	0.3	()	()
combinational area	255.744003	235.008026	()	()
total power	3.46E-07	3.70E-07	()	()
PDA	2.29934E-05	2.60718E-05	()	()

Elm				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	1.48	()
combinational area	()	()	1783.296631	()
total power	()	()	4.04E-04	()
PDA	()	()	1.066796577	()

Han - Carlson				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.17	0.18	0.19	1.05
combinational area	27.648001	55.296001	110.591995	1057.535034
total power	8.63E-08	1.41E-07	2.29E-07	3.68E-05
PDA	4.05483E-07	1.40142E-06	4.81606E-06	0.040829841

Manchester Carry Chain Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.05	1.88	3.54	6.85
combinational area	255.744049	511.487946	1022.97522	2045.950073
total power	1.13E-05	1.73E-05	2.79E-05	3.68E-05
PDA	0.003037088	0.016674098	0.100998957	0.516163537

Ripple Carry Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.36	2.88	5.91	11.97
combinational area	255.744003	511.488159	1022.97644	2045.953003
total power	3.65E-05	6.39E-05	6.21E-05	7.11E-05
PDA	0.012702089	0.094159651	0.375383148	1.741243084

Sklansky				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.82	0.83	0.84	0.85
combinational area	207.360031	400.896027	787.967224	1562.110107
total power	2.24E-05	2.58E-05	3.80E-05	5.96E-05
PDA	0.00381389	0.00858146	0.025151914	0.079083386

UMC library 18 best case :

Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.57	1.19	2.43	4.91
combinational area	361.832886	784.648865	1630.280884	3321.541016
total power	3.15E-04	4.86E-04	5.59E-04	6.03E-04
PDA	0.065028968	0.453793825	2.212940011	9.837447886

Carry Lookahead 1					
datawidth	4 bit	8 bit	16 bit	32 bit	(*)
critical path	0.69	1.15	()	1.79	4.27
combinational area	374.029907	1796.966309	()	7496.869141	6114.601074
total power	4.29E-05	1.11E-03	()	1.62E-03	5.86E-04
PDA	0.011076821	2.302093538	()	21.76625993	1.53E+01

Carry Save				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.2	0.21	0.22	0.23
combinational area	561.042908	1113.955322	2219.776123	4431.425293
total power	1.50E-06	1.05E-03	6.66E-06	1.90E-05
PDA	0.000168762	0.245861079	0.003253393	0.019395905

Carry Skip				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	()	5.11
combinational area	()	()	()	5334.047852
total power	()	()	()	1.10E-03
PDA	()	()	()	29.95542599

Carry Select				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.78	1.38	2.57	4.96
combinational area	1240.00415	2427.155762	4801.461914	9550.070312
total power	1.12E-03	1.71E-03	2.25E-03	2.87E-03
PDA	1.083267625	5.741000067	27.81381255	135.9945293

DesignWare Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.56	1.13	()	()
combinational area	406.559998	520.399963	()	()
total power	8.20E-05	3.45E-04	()	()
PDA	0.018660128	0.203113146	()	()

DesignWare Carry Save Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.17	0.18	()	()
combinational area	520.395996	520.399963	()	()
total power	6.72E-07	1.40E-06	()	()
PDA	5.94677E-05	0.000130672	()	()

Elm				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	0.88	()
combinational area	()	()	3927.336182	()
total power	()	()	2.87E-03	()
PDA	()	()	9.929248429	()

Han - Carlson				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.11	0.11	0.11	0.71
combinational area	56.917999	113.835999	227.671997	2362.068604
total power	1.87E-07	4.87E-07	9.29E-07	6.63E-04
PDA	1.17206E-06	6.09444E-06	2.32708E-05	1.112231968

Manchester Carry Chain Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.75	1.34	2.53	4.91
combinational area	552.911926	1105.824097	2211.646729	4423.291504
total power	8.27E-05	7.01E-05	1.42E-04	2.53E-04
PDA	0.034290215	0.103918935	0.794556204	5.499089077

Ripple Carry Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.61	1.2	2.39	4.78
combinational area	520.395996	1040.792114	2081.583984	4163.166504
total power	4.96E-04	7.51E-04	9.95E-04	1.12E-03
PDA	0.157324036	0.937961853	4.949613295	22.24812832

Sklansky				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.6	0.6	0.6	0.6
combinational area	455.339905	878.156067	1723.787598	3415.048828
total power	1.73E-04	1.79E-04	2.06E-04	2.96E-04
PDA	0.047236962	0.094155894	0.213163574	0.606307769

UMC library 18 worst case :

Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.33	2.76	5.63	11.36
combinational area	361.832886	784.648865	1630.280884	3321.541016
total power	1.25E-04	1.55E-04	1.61E-04	1.70E-04
PDA	0.059914098	0.335672784	1.474064109	6.429653092

Carry Lookahead 1					
datawidth	4 bit	8 bit	16 bit	32 bit	(*)
critical path	1.61	2.69	()	4.14	10.06
combinational area	374.029907	1796.966309	()	7626.966309	6260.961914
total power	3.26E-05	7.71E-04	()	4.79E-04	1.85E-04
PDA	0.019607246	3.728823691	()	15.12788937	11.63338064

Carry Save				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.51	0.51	0.51	0.51
combinational area	561.042908	1113.955322	2219.776123	4431.425293
total power	3.90E-04	7.98E-04	1.59E-03	3.20E-03
PDA	0.111591434	0.453584784	1.795488115	7.220785944

Carry Skip				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	()	11.87
combinational area	()	()	()	5334.047852
total power	()	()	()	2.99E-04
PDA	()	()	()	18.92489774

Carry Select				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.83	3.21	5.97	11.5
combinational area	1240.00415	2427.155762	4801.461914	9550.070312
total power	1.92E-04	6.37E-04	7.03E-04	7.76E-04
PDA	0.435687858	4.966091755	20.15990294	85.2028623

DesignWare Simple Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.32	2.66	()	()
combinational area	406.559998	520.399963	()	()
total power	3.62E-05	2.21E-04	()	()
PDA	0.019437796	0.305645469	()	()

DesignWare Carry Save Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.4	0.4	()	()
combinational area	520.395996	520.399963	()	()
total power	2.81E-07	7.99E-07	()	()
PDA	5.84301E-05	0.000166361	()	()

Elm				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	()	()	2.04	()
combinational area	()	()	3927.336182	()
total power	()	()	1.36E-03	()
PDA	()	()	10.87196621	()

Han - Carlson				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	0.24	0.24	0.24	1.66
combinational area	56.917999	113.835999	227.671997	2362.068604
total power	1.72E-07	2.58E-07	4.84E-07	2.59E-04
PDA	2.34957E-06	7.04053E-06	2.64191E-05	1.014371465

Manchester Carry Chain Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.72	3.1	5.85	11.36
combinational area	552.911926	1105.824097	2211.646729	4423.291504
total power	6.15E-05	4.53E-05	7.92E-05	7.15E-05
PDA	0.058458493	0.15539372	1.024829544	3.593779263

Ripple Carry Adder				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.41	2.79	5.56	11.08
combinational area	520.395996	1040.792114	2081.583984	4163.166504
total power	1.81E-04	2.79E-04	2.93E-04	2.67E-04
PDA	0.132736886	0.808711084	3.387594755	12.32537084

Sklansky				
datawidth	4 bit	8 bit	16 bit	32 bit
critical path	1.4	1.4	1.4	1.4
combinational area	455.339905	878.156067	1723.787598	3415.048828
total power	8.08E-05	9.12E-05	1.07E-04	1.56E-04
PDA	0.051533549	0.112135261	0.258223382	0.746802878

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία :

[1]. **VHDL Programming by Example**, Douglas L. Perry, McGraw - Hill Fourth Edition 2002.

[2]. **ASIC Design with Synopsys**, Himanshu Bhatnagar.

E-Books :

[3]. **ModelSIM SE**, Tutorial.

[4]. **ModelSIM SE**, Manual.

[5]. **Design Compiler™**, Command - Line Interface Guide, V-2004.06.

[6]. **Design Compiler™**, Reference Manual, V-2004.06.

[7]. **Design Compiler™**, User Guide, V-2004.06.

[8]. **Design Compiler™**, Tutorial Using Design Vision, V-2004.06.

[9]. **Design Analyzer™**, Reference Manual, V-2004.06.

[10]. **Design Vision™**, User Guide, V-2004.06.

[11]. **Power Compiler™**, User Guide, V-2004.06.

[12]. **Power Compiler™**, Quick Reference Guide, V-2004.06.

[13]. **PrimePower™**, Manual, V-2004.06.

[14]. **PrimePower™**, Quick Reference Guide, V-2004.06.

[15]. **Using TCL with Synopsys Tools**, V-2004.06.

[16]. **UMC eSi-Route/9™, High Density Standard Cell Library Datasheet**, Part Number : UMCL13U210T3, Revision 2.5, May 2002.

[17]. **UMC eSi-Route/11™, High Performance 0.18 μ Standard Cell Library Datasheet**, Part Number : UMCL18U250, Revision 2.1, January 2001.

[18]. **A Comparison of Hierarchical Compile Strategies**, Steve Golson, 2001.

[19]. **Resistance is Futile: Building Better Wireload Models**, Steve Golson, 1999.

[20]. **My Favorite dc_shell Tricks**, Steve Golson, 1995.

[21]. **Power Count: Measuring the Power at the VHDL Netlist Level**, A. Th. Schwarzbacker, P. A. Comiskey, J. B. Foley.

[22]. **Designing CMOS Circuits for Low Power**, D. Soudris, C. Piguet, C. Goutis.

Internet Sites :

[23]. **http://solvnet.synopsys.com/**