ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών & Δικτύων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Ανάλυση Χρονισμού Και Κατανάλωσης Ισχύος Πολλαπλών Διατάξεων Αθροιστών Πρόβλεψης Κρατουμένου"

ΤΖΙΜΑΣ ΑΝΤΩΝΗΣ

Επιβλέποντες καθηγητές:

- Γεώργιος Σταμούλης : Αναπληρωτής Καθηγητής, ΤΜΗΥΤΔ
- Νέστορας Ευμορφόπουλος : Επισκέπτης Καθηγητής, ΤΜΗΥΤΔ

Βόλος 2007

Η εργασία αυτή είναι αφιερωμένη
στον παππού μου Αντώνη.

Ευχαριστίες

Για την επιτυχημένη ολοκλήρωση της διπλωματικής μου εργασίας συνέβαλλαν πολλοί άνθρωποι. Αρχικά, θα ήθελα να ευχαριστήσω τον κ. Γεώργιο Σταμούλη, αναπληρωτή καθηγητή του τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων ο οποίος είναι ο βασικός επιβλέπων καθηγητής αυτής της εργασίας. Ο πολύτιμος χρόνος που αφιέρωσε μαζί μου και οι συμβουλές του αποτέλεσαν το πιο σημαντικό στήριγμα κατά την ολοκλήρωση του συγκεκριμένου έργου .

Επίσης, θα ήθελα να ευχαριστήσω τον έτερο επιβλέποντα καθηγητή και επισκέπτη καθηγητή στο τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, κ. Νέστορα Ευμορφόπουλο για τις συμβουλές και την ουσιαστική αρωγή που μου έδωσε κυρίως πάνω στο μαθηματικό υπόβαθρο που χρησιμοποιήθηκε στις μετρήσεις που παρουσιάζονται.

Στο σημείο αυτό δεν θα μπορούσα να αμελήσω τον διδακτορικό φοιτητή του τμήματος Αντώνιο Δαδαλιάρη για την προγενέστερη δουλειά του στα εργαλεία που χρησιμοποιήθηκαν και για την υποστήριξη που μου παρείχε πάνω σε οποιαδήποτε δυσκολία εμφανίστηκε κατά την εκπόνηση αυτής της διατριβής, καθώς και τους όλους τους υπόλοιπους καθηγητές που καθένας με τον ξεχωριστό του τρόπο επηρέασε το επιστημονικό μου πέρασμα από το τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων.

Τέλος θα ήθελα να πω ένα μεγάλο ευχαριστώ, εκτός από όλους τους φίλους μου, στην οικογένειά μου για την αμέριστη στήριξη που με πρόσφερε όλα αυτά τα χρόνια. Ελπίζω να κριθώ άξιος και να ανταποκριθώ στις απαιτήσεις των ανθρώπων που με βοήθησαν και πίστεψαν στις δυνατότητές μου όλα αυτά τα χρόνια χωρίς να τους απογοητεύσω.

Σας ευχαριστώ θερμά,

Τζίμας Αντώνης

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ 5
1.1) ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ 6
1.2) ΣΤΟΧΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ 8

2.ΑΘΡΟΙΣΤΕΣ ΠΡΟΒΛΕΨΗΣ ΚΡΑΤΟΥΜΕΝΟΥ 9
2.1)ΚΑΤΗΓΟΡΙΕΣ ΑΘΡΟΙΣΤΩΝ 10
2.1.1)ΑΘΡΟΙΣΤΕΣ ΔΙΑΔΟΣΗΣ ΚΡΑΤΟΥΜΕΝΟΥ(CPA) 11
2.1.2)ΗΜΙΑΘΡΟΙΣΤΗΣ(Half Adder) 13
2.1.3)ΠΛΗΡΗΣ ΑΘΡΟΙΣΤΗΣ(Full Adder) 14
2.2)ΑΘΡΟΙΣΤΗΣ ΠΡΟΒΛΕΨΗΣ ΚΡΑΤΟΥΜΕΝΟΥ 16
2.2.1) Default Implementations Of A Carry-Lookahead Adder 18
2.2.2) DesignWare Carry-Lookahead Adder 22

3.ΠΡΟΣΟΜΟΙΩΣΗ 25
3.1) ΕΡΓΑΛΕΙΟ ΠΡΟΣΟΜΟΙΩΣΗΣ 26
3.2) ΔΙΑΔΙΚΑΣΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ 28

4.ΣΥΝΘΕΣΗ 37
4.1) ΕΙΣΑΓΩΓΗ 38
4.2) DESIGN COMPILER 39
4.3) ΟΡΙΣΜΟΙ 46
4.4) DESIGN VISION 47
4.5) DESIGNWARE LIBRARIES 56

5.ΑΝΑΛΥΣΗ ΧΡΟΝΙΣΜΟΥ 65
5.1) ΕΙΣΑΓΩΓΗ 66
5.2)ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ PRIME TIME 67

6.ΑΝΑΛΥΣΗ ΚΑΤΑΝΑΛΙΣΚΩΜΕΝΗΣ ΙΣΧΥΟΣ 90
6.1) ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΚΑΤΑΝΑΛΩΣΗΣ ΙΣΧΥΟΣ 91
6.2) PRIME POWER & ΚΑΤΑΝΑΛΩΣΗ ΙΣΧΥΟΣ 96

7.ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ 106
7.1) ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΓΚΡΙΣΕΙΣ 107

8.ΕΠΙΛΟΓΟΣ – ΣΥΜΠΕΡΑΣΜΑΤΑ 119

ΠΑΡΑΡΤΗΜΑ Α. ΠΙΝΑΚΕΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 121

ΒΙΒΛΙΟΓΡΑΦΙΑ 125

1.ΕΙΣΑΓΩΓΗ

1.1) ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Αλήθεια έχουμε άραγε αναρωτηθεί πόσα χρόνια έχουν περάσει από την λειτουργία του πρώτου ηλεκτρονικού υπολογιστή; Υπάρχουν κάποιοι που θεωρούν πως πέρασαν λίγα χρόνια σε σχέση με την ροή της ιστορίας, αλλά τι συμβαίνει πραγματικά κοιτώντας μέσα από το πρίσμα της τεχνολογίας;

Αν λάβουμε υπ' όψιν μας το νόμο του Moore θα μπορέσουμε να αντιληφθούμε την εκθετική αύξηση της τεχνολογίας και ίσως κατανοήσουμε για ποιο λόγο η επιστήμη αναθεωρεί και επαναπροσανατολίζει τις έρευνες της τόσο ραγδαία. Έτσι, παρατηρούμε πώς από το αρχικό μέλημα των επιστημόνων που ήταν η λειτουργικότητα των διατάξεων των κυκλωμάτων, μεταπηδήσαμε τα τελευταία χρόνια στο κυνήγι των βελτιστοποιήσεων των διατάξεων αυτών ως προς την καθυστέρηση της απόκρισής τους, την επιφάνεια που καταλαμβάνουν και την ισχύ που καταναλώνουν.

Η προαναφερθείσα τάση οφείλεται σε μεγάλο ποσοστό στην αύξηση της κατασκευής και χρήσης φορητών συσκευών, η οποία συνεπάγεται και την χρήση εναλλακτικών πηγών τροφοδοσίας ενέργειας. Επιπλέον, η ανάγκη εξοικονόμησης ενέργειας δεν μένει στα συρτάρια κάποιων επιστημόνων αλλά αποτελεί τον κύριο γνώμονα της έρευνας πάνω σε οποιοδήποτε είδος κυκλωμάτων. Συνεπώς η διαδικασία που ακολουθείται κατά τη σχεδίαση ενός οποιουδήποτε κυκλώματος με οποιεσδήποτε εφαρμογές, περιλαμβάνει από τα πρώτα στάδια την βελτιστοποίηση αλλά και τον ακριβή υπολογισμό των παραμέτρων της σχεδίασης του και περιλαμβάνει και μετρήσεις που αφορούν τον χρονισμό και την κατανάλωση ισχύος σε άμεση συνάρτηση πάντα με την εφαρμογή όπου το κύκλωμα πρόκειται να χρησιμοποιηθεί.

Ο λόγος που πραγματοποιείται αυτή η ενέργεια από τα πρώτα κιόλας στάδια είναι ώστε να εξαλειφθούν ή να αποφευχθούν , όσο είναι αυτό εφικτό, λάθη, προβλήματα ή τυχόν δυσλειτουργίες κατά την κατασκευή του ολοκληρωμένου.

Οι εταιρίες σχεδίασης κυκλωμάτων, λοιπόν, αφού παρατήρησαν αυτήν σχεδιαστική και όχι μόνον ανάγκη προέβησαν στην ανάπτυξη εργαλείων που θα βοηθούσαν τους εκάστοτε σχεδιαστές ώστε να μειώσουν την κατανάλωση της ισχύος και να βελτιστοποιήσουν τις προαναφερθείσες κυκλωματικές παραμέτρους (συνολική επιφάνεια κυκλώματος, χρονισμός σχεδίασης).

1.2) ΣΤΟΧΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Ο βασικός στόχος αυτής της εργασίας είναι η διενέργεια μετρήσεων και υπολογισμών που αφορούν τον χρονισμό σε ένα ψηφιακό κύκλωμα (μείωση της καθυστέρησης απόκρισης) και την ισχύ που καταναλώνεται σε αυτό. Για να διενεργηθούν αυτές οι μετρήσεις έχουμε επιλέξει κυκλώματα που χρησιμοποιούνται στην πλειοψηφία των ψηφιακών σχεδιάσεων (αθροιστές) και πιο συγκεκριμένα τους αθροιστές πρόβλεψης κρατουμένου (**carry lookahead adders**) που αποτελούν μια κλασσική συνδεσμολογία η οποία χρησιμοποιείται.

Σε πρώτη φάση θα χρησιμοποιήσουμε ένα εργαλείο προσομοίωσης για να ελέγχουμε την ορθή λειτουργία των ψηφιακών κυκλωμάτων, το οποίο είναι το **ModelSim SE** της εταιρείας **Model Technology**. Σε δεύτερη φάση, για την διενέργεια των υπολογισμών μας θα χειριστούμε κάποια εργαλεία της πλατφόρμας της εταιρείας **Synopsys** και πιο συγκεκριμένα θα δουλέψουμε με το **Design Vision**, το **PrimePower** και το **Prime Time**.

Η επιλογή αυτών των εργαλείων βασίστηκε στο γεγονός πως η **Synopsys** αποτελεί μια από τις μεγαλύτερες εταιρείες που αναπτύσσουν λογισμικό για προσομοίωση, σύνθεση και έλεγχο ψηφιακών κυκλωμάτων στην παγκόσμια αγορά. 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.

Όπως γίνεται εύκολα αντιληπτό μετά το πέρας των υπολογισμών – μετρήσεων ακολουθεί συγκεντρωτική ανάλυση και επεξεργασία των αποτελεσμάτων από τα οποία ενδεχομένως εξαχθούν κάποια χρήσιμα συμπεράσματα.

2.ΑΘΡΟΙΣΤΕΣ ΠΡΟΒΛΕΨΗΣ ΚΡΑΤΟΥΜΕΝΟΥ

2.1) ΚΑΤΗΓΟΡΙΕΣ ΑΘΡΟΙΣΤΩΝ

Ένας σχεδιαστής ψηφιακών κυκλωμάτων κατά την διάρκεια της καριέρας του θα συναντήσει ουκ ολίγες φορές διατάξεις αθροιστών σαν δομικά στοιχεία άλλων κυκλωμάτων. Αυτό γίνεται γιατί η πρόσθεση δυο δυαδικών αριθμών είναι μια από τις πράξεις με την μεγαλύτερη συχνότητα εμφάνισης. Δεν θα πρέπει επίσης να αμελήσουμε το γεγονός πως πολλές άλλες πράξεις μεταξύ δυαδικών αριθμών έχουν σαν βάση τους την πρόσθεση. Για αυτό το λόγο ένας αθροιστής μπορεί να χρησιμοποιηθεί , όχι μόνο για πρόσθεση , αλλά και για την πράξη της αφαίρεσης , του πολλαπλασιασμού και της διαίρεσης. Επιπλέον υπάρχουν διατάξεις – σχεδιάσεις – κυκλώματα που κάνουν πιο πολύπλοκες ενέργειες με βάση τις βασικές πράξεις που προαναφέραμε.

Συνεπώς είναι εύκολα αντιληπτό πως η επιλογή μελέτης διατάξεων αθροιστών αποτελεί μελέτη στη δομική βάση ενός ευρύτερου φάσματος σχεδιάσεων. Επιπρόσθετα μία κατηγορία τόσο χρήσιμων και βασικών κυκλωμάτων όπως οι αθροιστές δεν θα μπορούσε να είναι μικρή και ενταία.

Λόγω των διαφορετικών ιδιοτήτων που παρουσιάζουν οι αθροιστές ομαδοποιούνται σε δυο βασικές κατηγορίες:

A. Carry-Propagate Adders (CPA).

B. Tree Adders.

Ο αθροιστής που θα μελετήσουμε ανήκει στην πρώτη από τις δύο κατηγορίες, στους αθροιστές δηλαδή, διάδοσης κρατουμένου. Εντούτοις, όμως αποτελεί βασικό δομικό στοιχείο στην πλειοψηφία των αθροιστών της δεύτερης κατηγορίας.

2.1.1) Αθροιστές Διάδοσης Κρατουμένου (CPA)

Οι αθροιστές αυτής της κατηγορίας παίρνουν σαν είσοδο δύο τελεστέους ίσου μήκους bit και ένα κρατούμενο εισόδου (το οποίο πιθανότατα όταν αναφερόμασταε σε πολυεπίπεδες διατάξεις να είναι η έξοδος κρατουμένου της προηγούμενης βαθμίδας) και βγάζουν σαν έξοδο ένα δυαδικό διάνυσμα ίσου μήκους με το μήκος του ενός τελεστή πού αποτελεί το αποτέλεσμα και ένα κρατούμενο εξόδου. Αναλυτικά :

A : ο πρώτος τελεστέος μήκους N bits

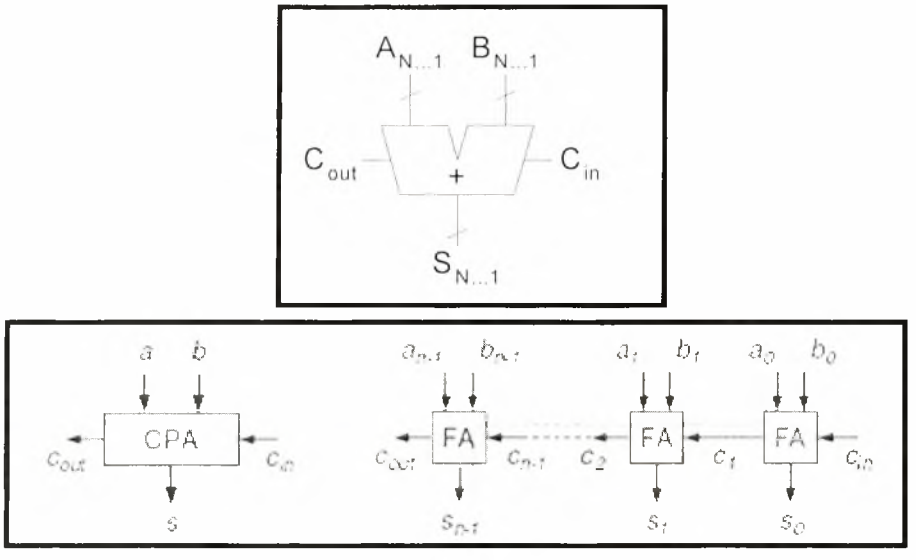
B : ο δεύτερος τελεστέος μήκους N bits

C_{in} : το κρατούμενο εισόδου

S : το αποτέλεσμα της πρόσθεσης A, B μήκους N bits

C_{out} : το κρατούμενο εξόδου

Σχηματικά έχουμε για ένα και για **n** επίπεδα αντίστοιχα :



ΣΧΗΜΑ 2.1 : Σχηματική αναπαράσταση της ακολουθούμενης λογικής για έναν γενικού τύπου αθροιστή της κατηγορίας ενός και πολλαπλών επιπέδων

Η ονομασία τους προκύπτει από το γεγονός πως το κρατούμενο σε κάθε bit μπορεί να επηρεάσει τα κρατούμενα όλων των μεταγενέστερων bits το οποίο διαφαίνεται καθαρά από το τρίτο σχήμα της παραπάνω εικόνας.

Η μαθηματική και λογική υπόσταση τέτοιων αθροιστών διάδοσης κρατουμένου περιγράφονται αναλυτικά παρακάτω.

$$
\begin{aligned}
2^n c_{out} + S &= A + B + c_{in} \\
2^n c_{out} + \sum_{i=0}^{n-1} 2^i s_i &= \sum_{i=0}^{n-1} 2^i a_i + \sum_{i=0}^{n-1} 2^i b_i + c_{in} \\
&= \sum_{i=0}^{n-1} 2^i (a_i + b_i) + c_{in} \\
2c_{i+1} + s_i &= a_i + b_i + c_i \;:\; i = 0, 1, \ldots, n-1 \\
&\quad \text{where } c_0 = c_{in} \text{ and } c_{out} = c_n
\end{aligned}
$$

$$
\begin{aligned}
g_i &= a_i\, b_i \\
p_i &= a_i \oplus b_i \\
s_i &= p_i \oplus c_i \\
c_{i+1} &= g_i + p_i\, c_i \;:\; i = 0, 1, \ldots, n-1 \\
&\quad \text{where } c_0 = c_{in} \text{ and } c_{out} = c_n
\end{aligned}
$$

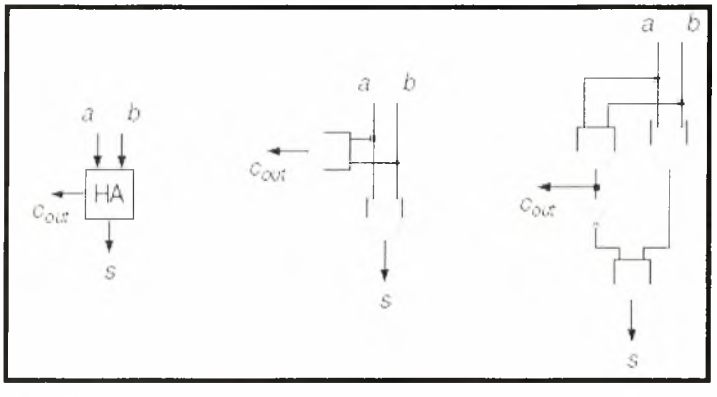
ΣΧΗΜΑ 2.2 : Αριθμητικές & λογικές εξισώσεις ενός γενικού τύπου αθροιστή της κατηγορίας

Όπως διαφαίνεται από τα σχήματα και από τις εξισώσεις, λόγω της σειριακής διάδοσης κρατουμένου ο χρόνος υπολογισμού της εξόδου αυτού του τύπου αθροιστή αυξάνει γραμμικά με το μήκος τελεστέου.

Ξεκινώντας λοιπόν την περιπλάνηση μας στον κόσμο των αθροιστών είμαστε υποχρεωμένοι να παρουσιάσουμε το δομικό στοιχείο ενός πλήρους αθροιστή (**full adder**) το οποίο είναι ένας ημιαθροιστής (**half adder**).

2.1.2) Ημιαθροιστής (Half-Adder)

Είναι το συνδυαστικό κύκλωμα που έχει ως εισόδους δύο δυαδικά ψηφία και έξοδο το άθροισμά τους και το πιθανό κρατούμενο. Το πιο σημαντικό ψηφίο ονομάζεται **carry-out**(c_{out}) επειδή μεταφέρει το γεγονός της υπερχείλισης στο αμέσως υψηλότερο σε θέση **bit**. Αν **A**, **B** οι μονοψήφιοι αριθμοί προς άθροιση και **S**, C_{out} το άθροισμα και το κρατούμενό τους αντίστοιχα, το σύμβολο του και δύο ενδεικτικές υλοποιήσεις φαίνονται παρακάτω στο σχήμα :



ΣΧΗΜΑ 2.3 : Σύμβολο του ημιαθροιστή και δύο ενδεικτικές υλοποιήσεις του

Για την πλήρη κατανόηση της λειτουργίας του ημιαθροιστή παραθέτουμε τον πίνακα αληθείας του, τις λογικές εξισώσεις και τις αριθμητικές εξισώσεις του :

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

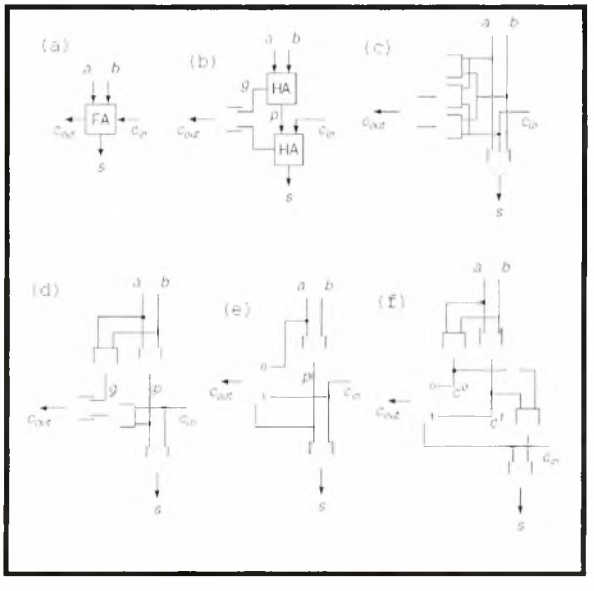
$$s = a \oplus b$$
$$c_{out} = ab$$

$$2c_{out} + s = a + b$$
$$s = (a + b) \bmod 2$$
$$c_{out} = (a + b) \operatorname{div} 2 = \frac{1}{2}(a + b - s)$$

ΣΧΗΜΑ 2.4 : Πίνακας αληθείας και λογικές & αριθμητικές εξισώσεις του ημιαθροιστή

2.1.3) Πλήρης αθροιστής(Full Adder)

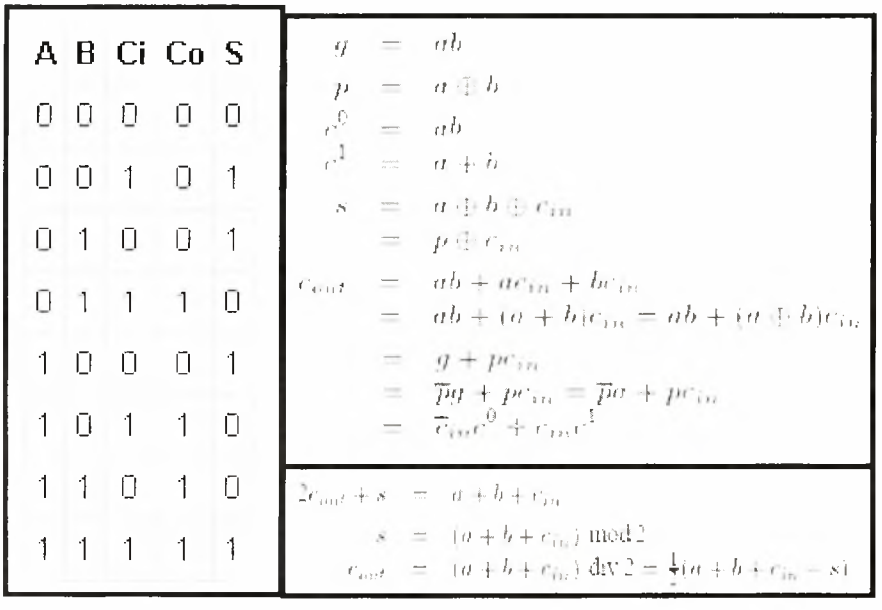
Ο πλήρης αθροιστής ενός ψηφίου διαφέρει από τον ημιαθροιστή στο ότι έχει μία παραπάνω είσοδο, αυτή του κρατούμενου εισόδου. Στην ουσία μπορούμε να πούμε πως είναι ο συνδυασμός δύο ημιαθροιστών άλλωστε αυτό μπορεί να φανεί και από τα σχήματα που παρατίθονται παρακάτω.



ΣΧΗΜΑ 2.5 : Σχηματική αναπαράσταση του πλήρους αθροιστή συναρτήσει ημιαθροιστών και λογικών πυλών

Σημαντικά εσωτερικά σήματα του πλήρους αθροιστή είναι τα **generate (g)** και το **propagate (p)**. Το πρώτο υποδεικνύει αν ένα σήμα κρατουμένου δημιουργείται εντός του πλήρους αθροιστή. Το σήμα διάδοσης υποδεικνύει αν ένα κρατούμενο στην είσοδο διαδίδεται στο κρατούμενο εξόδου χωρίς να αλλαχτεί διαμέσου του πλήρους αθροιστή. Έτσι το κρατούμενο εξόδου μπορεί να υπολογιστεί συναρτήσει των g, p και του κρατουμένου εισόδου και μπορεί να υλοποιηθεί με ποικίλους τρόπους. (π.χ. είτε με **AND-OR** είτε με πολυπλέκτη).

Για την πλήρη κατανόηση της λειτουργίας του αθροιστή παραθέτουμε τον πίνακα αληθείας του, όπως και τις λογικές και αριθμητικές εξισώσεις του.



A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

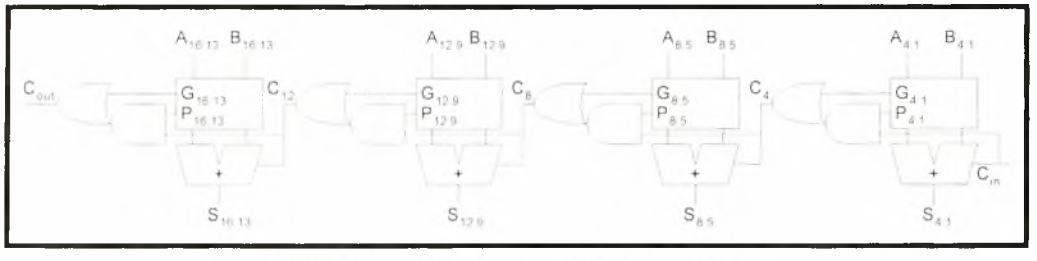
$$
\begin{aligned}
g &= ab \\
p &= a \oplus b \\
c^0 &= ab \\
c^1 &= a + b \\
s &= a \oplus b \oplus c_{in} \\
&= p \oplus c_{in} \\
c_{out} &= ab + ac_{in} + bc_{in} \\
&= ab + (a + b)c_{in} = ab + (a \oplus b)c_{in} \\
&= g + pc_{in} \\
&= \overline{p}g + pc_{in} = \overline{p}a + pc_{in} \\
&= \overline{c}_{in}c^0 + c_{in}c^1
\end{aligned}
$$

$$
\begin{aligned}
2c_{out} + s &= a + b + c_{in} \\
s &= (a + b + c_{in}) \bmod 2 \\
c_{out} &= (a + b + c_{in}) \operatorname{div} 2 = \tfrac{1}{2}(a + b + c_{in} - s)
\end{aligned}
$$

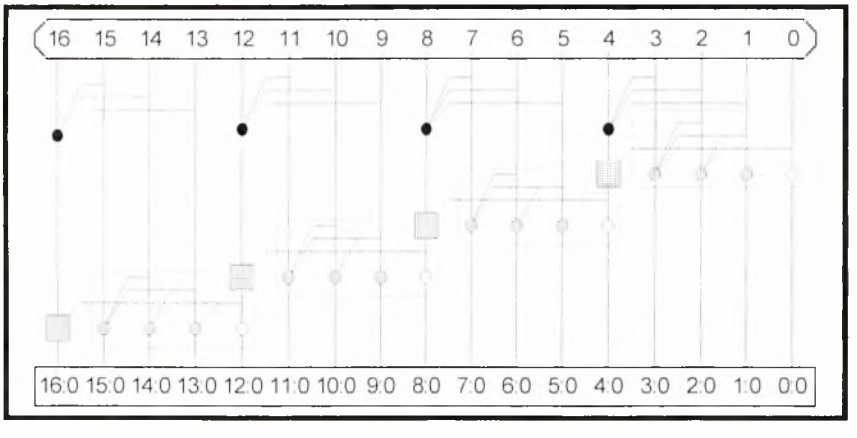
ΣΧΗΜΑ 2.6 : : Πίνακας αληθείας και λογικές & αριθμητικές εξισώσεις του πλήρους αθροιστή

2.2) ΑΘΡΟΙΣΤΗΣ ΠΡΟΒΛΕΨΗΣ ΚΡΑΤΟΥΜΕΝΟΥ (CARRY LOOKAHEAD ADDER)

Ο **Carry Lookahead Adder** (**CLA**) έχει παρόμοια / παραπλήσια λογική με τον **Carry Skip Adder**. Κινούμαστε, λοιπόν, με βάση την λογική της ομαδοποίησης των σημάτων διάδοσης και δημιουργίας κρατουμένων. Η διαφορά με τον **Carry Skip Adder** είναι πως υπολογίζονται ανά ομάδες τα σήματα **propagate** (**P**) και **generate** (**G**) εξαλείφοντας την αναμονή της σειριακής απόφασης για το αν κάποια ομάδα δημιουργεί κρατούμενα ή όχι.



ΣΧΗΜΑ 2.7 : 16-bit Carry Lookahead Adder



ΣΧΗΜΑ 2.8 : PG διάγραμμα του Carry Lookahead Adder

Στην παραπάνω εικόνα παρουσιάζεται το PG διάγραμμα του αθροιστή πρόβλεψης κρατουμένου. Το **PG** Δικτυακό Διάγραμμα οποιουδήποτε αθροιστή προκύπτει μέσω της απεικόνισης των **Propagate** και **Generate**

σημάτων που έχουν προαναφερθεί. Το χρώμα των κελιών μπορεί να πάρει τις ακόλουθες τιμές :

- **Μαύρη** : περιέχουν την **generate –propagate** λογική που διέπει την ομάδα του κάθε αθροιστή.
- **Γκρι** : περιέχουν μόνο την **generate** λογική της ομάδας που ανήκουν και χρησιμοποιούνται στην τελική θέση κάθε στήλης γιατί περιέχουν όλη την απαραίτητη πληροφορία για τον εκάστοτε υπολογισμό.
- **Λευκό** : απεικονίζει τους **buffers**.

2.2.1) Default Implementations Of A Carry-Lookahead Adder :

Στην παρούσα πτυχιακή χρησιμοποιήσαμε τρεις διαφορετικές υλοποιήσεις του συγκεκριμένου αθροιστή (για τελεστέους μεγέθους 4, 8, 16, 32 και 64 bit). Η μία από αυτές παρουσιάζεται στην επόμενη παράγραφο και προέρχεται από τις **DesignWare** βιβλιοθήκες που χρησιμοποιήθηκαν. Οι άλλες δύο υλοποιήσεις σε **structural** και **behavioral VHDL** προέρχονται από παλαιότερη πτυχιακή εργασία συναδέλφου.

Ακολουθούν ενδεικτικά οι κώδικες ενός **structural** και ενός **behavioral carry lookahead** αθροιστή όπως και κάποιες κυματομορφές επιβεβαίωσης της ορθής λειτουργίας τους.

Στην περίπτωση περιγραφής της συμπεριφοράς του αθροιστή, θα έχουμε, λοιπόν :

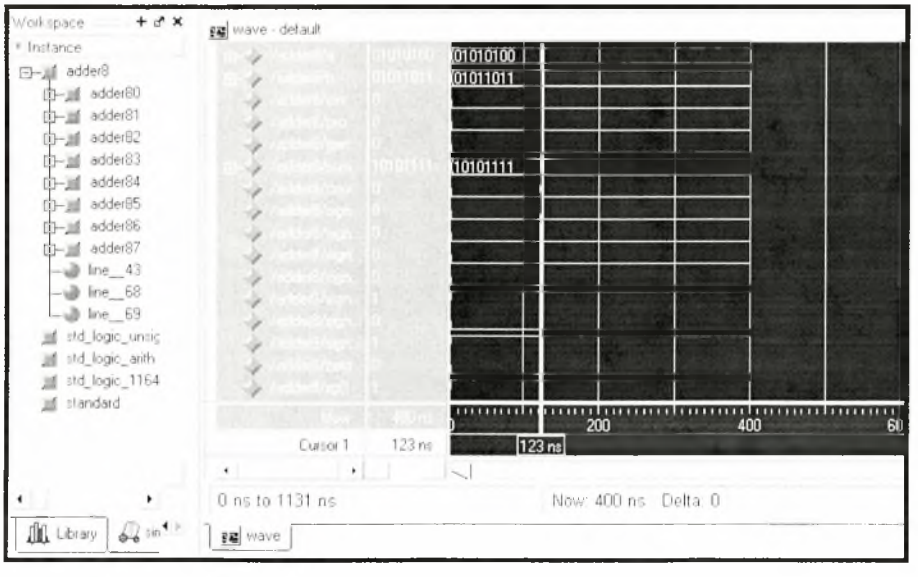
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY c_l_addr_8 IS
  PORT
    (
     x_in    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
     y_in    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
     carry_in : IN  STD_LOGIC;
     sum     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
     carry_out : OUT STD_LOGIC
    );
END c_l_addr_8;

ARCHITECTURE behavioral OF c_l_addr_8 IS

SIGNAL  h_sum          : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  carry_generate  : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  carry_propagate : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL  carry_in_internal: STD_LOGIC_VECTOR(7 DOWNTO 1);

BEGIN
  h_sum <= x_in XOR y_in;
  carry_generate <= x_in AND y_in;
  carry_propagate <= x_in OR y_in;
  PROCESS (carry_generate,carry_propagate,carry_in_internal)
  BEGIN
  carry_in_internal(1) <= carry_generate(0) OR (carry_propagate(0) AND carry_in);
    inst: FOR i IN 1 TO 6 LOOP
        carry_in_internal(i+1)  <=  carry_generate(i)  OR  (carry_propagate(i)  AND
carry_in_internal(i));
        END LOOP;
```

```
  carry_out <= carry_generate(7) OR (carry_propagate(7) AND carry_in_internal(7));
  END PROCESS;

  sum(0) <= h_sum(0) XOR carry_in;
  sum(7 DOWNTO 1) <= h_sum(7 DOWNTO 1) XOR carry_in_internal(7 DOWNTO 1);
END behavioral;
-------------------------------------------------------------------------------------------------
```

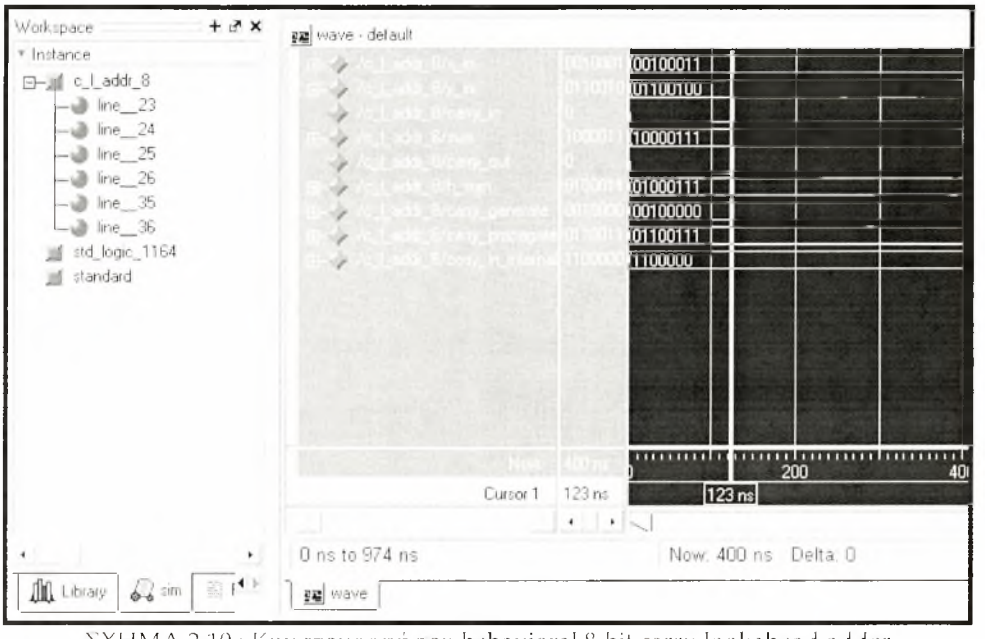
Ενώ κατά την περιγραφή της δομής του αθροιστή βάσει ενός προϋπάρχοντος πλήρους αθροιστή θα έχουμε :

```
-------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY adder8 IS
        PORT
        (
         A       : IN   STD_LOGIC_VECTOR(7 downto 0);
         B       : IN     STD_LOGIC_VECTOR(7 downto 0);
         CIN     : IN     STD_LOGIC;
         Pro   : OUT STD_LOGIC;
         Gen   : OUT STD_LOGIC;
         SUM   : OUT STD_LOGIC_VECTOR(7 downto 0);
         COUT : OUT STD_LOGIC
        );
END adder8;


ARCHITECTURE ADD_struct  OF adder8 IS

COMPONENT addone
        PORT
        (
                a       : in  STD_LOGIC;
                b       : in  STD_LOGIC;
                cin     : in  STD_LOGIC;
                pin     : in STD_LOGIC;
                gin     : in STD_LOGIC;
                pnext  : out STD_LOGIC;
                gnext  : out STD_LOGIC;
                sum    : out STD_LOGIC;
                cout    : out STD_LOGIC
        );
END COMPONENT ;

SIGNAL signal0,signal1,signal2,signal3,signal4,signal5,signal6,signal7: std_logic;
SIGNAL      zero,SP0,SG0,SP1,SG1,SP2,SG2,SP3,SG3,SP4,SG4,SP5,SG5,SP6,SG6,SP7,SG7:
std_logic;

BEGIN
zero<='0';
```

```
adder80 : addone port map(a => A(0),b => B(0), cin => CIN,
                pin => zero,gin=>zero,pnext => SP0,gnext=>SG0,sum =>SUM(0),cout =>
signal0);
adder81 : addone port map(a => A(1),b => B(1), cin => signal0,
                pin => SP0,gin=>SG0,pnext => SP1,gnext=>SG1,sum =>SUM(1),cout =>
signal1);
adder82 : addone port map(a => A(2),b => B(2), cin => signal1,
                pin => SP1,gin=>SG1,pnext => SP2,gnext=>SG2,sum =>SUM(2),cout =>
signal2);
adder83 : addone port map(a => A(3),b => B(3), cin => signal2,
                pin => SP2,gin=>SG2,pnext => SP3,gnext=>SG3,sum =>SUM(3),cout =>
signal3);
adder84 : addone port map(a => A(4),b => B(4), cin => signal3,
                pin => SP3,gin=>SG3,pnext => SP4,gnext=>SG4,sum =>SUM(4),cout =>
signal4);
adder85 : addone port map(a => A(5),b => B(5), cin => signal4,
                pin => SP4,gin=>SG4,pnext => SP5,gnext=>SG5,sum =>SUM(5),cout =>
signal5);
adder86 : addone port map(a => A(6),b => B(6), cin => signal5,
                pin => SP5,gin=>SG5,pnext => SP6,gnext=>SG6,sum =>SUM(6),cout =>
signal6);
adder87 : addone port map(a => A(7),b => B(7), cin => signal6,
                pin => SP6,gin=>SG6,pnext => SP7,gnext=>SG7,sum =>SUM(7),cout =>
COUT);

Pro <= (SP0 and  SP1 and SP2 and SP3 and SP4 and SP5 and SP6 and SP7);
Gen <= (SG7 or (SG6 and SP7) or (SG5 and SP7 and SP6)  or (SG4 and SP7 and SP6 and
SP5) or
    (SG3 and SP7 and SP6 and SP5 and SP4) or
    (SG2 and SP7 and SP6 and SP5 and SP4 and SP3) or (SG1 and SP7 and SP6 and SP5
and SP4 and SP3 and SP2));

END ADD_struct;
```



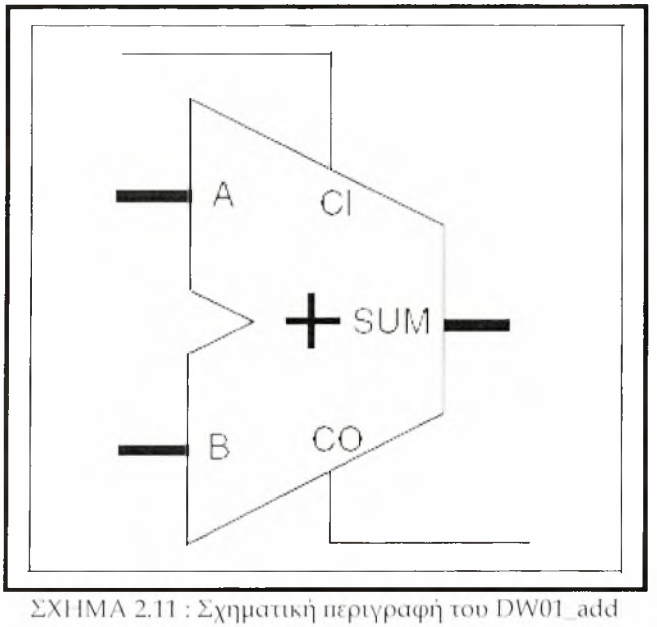
ΣΧΗΜΑ 2.9 : Κυματομορφή του structural 8-bit carry lookahead adder



ΣΧΗΜΑ 2.10 : Κυματομορφή του behavioral 8-bit carry lookahead adder

2.2.2) DesignWare Carry-Lookahead Adder :

Το αρχείο **DW01_add** παρουσιάζει μια παραμετρική περιγραφή ενός αθροιστή. Στο παρακάτω σχήμα παρουσιάζεται η σχηματική περιγραφή αυτού του αθροιστή.



ΣΧΗΜΑ 2.11 : Σχηματική περιγραφή του DW01_add

Όπως παρατηρούμε ο προκείμενος αθροιστής προσθέτει δυο τελεστέους A και B μαζί με ένα κρατούμενο εισόδου **CI** και παράγει το αποτέλεσμα **SUM** και ένα κρατούμενο εξόδου **CO**. Στον ακόλουθο πίνακα παρουσιάζεται η περιγραφή των pins του designware αθροιστή που θα εξετάσουμε.

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
CI	1 bit	Input	Carry-in
SUM	*width* bit(s)	Output	Sum of (A + B + CI)
CO	1 bit	Output	Carry-out

ΣΧΗΜΑ 2.12 : Περιγραφή των pins του DW01_add

Ενώ στον επόμενο πίνακα παρουσιάζονται οι διάφοροι τρόποι με τους οποίους μπορεί να γίνει η σύνθεση του αθροιστή.

Implementation Name	Function	License Feature Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
clf	Fast carry-look-ahead synthesis model	DesignWare
bk	Brent-Kung architecture synthesis model	DesignWare
csm[b]	Conditional-sum synthesis model	DesignWare
rpcs	Ripple-carry-select architecture	DesignWare
pparch[c]	Delay-optimized flexible parallel-prefix	DesignWare

ΣΧΗΜΑ 2.13 : Αναφορά όλων των πιθανών μετεξελίξεων του DW01_add

Κατά την διάρκεια της σύνθεσης ο **Design Compiler** θα επιλέξει την κατάλληλη αρχιτεκτονική ανάλογα με τους περιορισμούς που έχουμε θέσει. Παρόλ' αυτα, μπορούμε να εξαναγκάσουμε το εργαλείο σύνθεσης να επιλέξει μια από τις συγκεκριμένες αρχιτεκτονικές του παραπάνω πίνακα.

Οι τρόποι με τους οποίους μπορούμε να προσπελάσουμε τα **designware components** είναι οι ακόλουθοι :

- **Inferencing** και
- **Instantiation**

Σε ελεύθερη μετάφραση οι παραπάνω όροι μπορούν να προσδιοριστούν ως συμπερασμός και συγκεκριμενοποίηση, αντίστοιχα. Στην παρούσα εργασία θα χρησιμοποιήσουμε τον δεύτερο τρόπο. Σύμφωνα, λοιπόν, με το **datasheet** του **designware adder** η χρήση της γλώσσας περιγραφής υλικού **VHDL** μέσω συγκεκριμενοποίησης ενός δομικού στοιχείου της σχεδίασης **(component)** μπορεί να γίνει με κατάλληλη τροποποίηση του παρακείμενου κομματιού κώδικα :

```
--------------------------------------------------------------------------------
LIBRARY IEEE, DWARE, DW01;
USE IEEE.std_logic_1164.all;
USE DWARE.DWpackages.all;
USE DW01.DW01_components.all;

ENTITY DW01_add_inst IS
        GENERIC (inst_width : NATURAL :=8);
        PORT
        (
                inst_A : IN STD_LOGIC_VECTOR(inst_width-1 DOWNTO 0);
                inst_B : IN STD_LOGIC_VECTOR(inst_width-1 DOWNTO 0);
                inst_CI : IN STD_LOGIC;
                SUM_inst : OUT STD_LOGIC_VECTOR (inst_width-1 DOWNTO 0);
                CO_inst : OUT STD_LOGIC
        );
END DW01_add_inst;

ARCHITECTURE inst OF DW01_add_inst IS
```

```
BEGIN
        U1:DW01_add
        GENERIC MAP (width=>inst_width)
        PORT MAP (A=>inst_A, B=>inst_B, CI=>inst_CI, SUM=>SUM_inst,
CO=>CO_inst);
END inst;

CONFIGURATION DW01_add_inst_cfg_inst OF DW01_add_inst IS
        FOR inst
        END FOR;
END DW01_add_inst_cfg_inst;
```

Στην περίπτωσή μας θα χρησιμοποιήσουμε την **Carry-Lookahead** αρχιτεκτονική για να προβούμε στις κατάλληλες μετρήσεις και συγκρίσεις. Επομένως, ο **VHDL** κώδικας που θα χρησιμοποιήσουμε είναι ο ακόλουθος :

```
LIBRARY IEEE, DW01, SYNOPSYS;
USE IEEE.std_logic_1164.all;
USE DW01.DW01_components.all;
USE synopsys.attributes.all;

ENTITY CLADes64 IS
        GENERIC (wordlength : INTEGER := 64);
        PORT
        (
                A, B : IN STD_LOGIC_VECTOR(wordlength-1 DOWNTO 0);
                CIN : IN STD_LOGIC;
                SUM : OUT STD_LOGIC_VECTOR(wordlength-1 DOWNTO 0);
                COUT : IN STD_LOGIC
        );
END CLADes64;

ARCHITECTURE inst OF CLADes64 IS
        attribute implementation: STRING;
        attribute implementation OF U1 : label is "CLA";

BEGIN
        U1: DW01_add
        GENERIC MAP(width=>wordlength)
        PORT MAP(A=>A, B=>B, CI=>CIN, SUM=>SUM,CO=>COUT);
END inst;
```

Αυτός, λοιπόν, είναι ο δεύτερος αθροιστής που θα χρησιμοποιήσουμε στις μετρήσεις μας. Το μήκος των τελούμενων είναι προφανές πως μπορεί να αλλάξει κατά βούληση. Η λειτουργικότητά του είναι όμοια με αυτή του αθροιστή που παρουσιάστηκε παραπάνω τα αποτελέσματα του όμως, όπως θα δούμε και στην συνέχεια παρουσιάζουν κάποιες διαφοροποιήσεις.

3.ΠΡΟΣΟΜΟΙΩΣΗ ΑΘΡΟΙΣΤΩΝ

3.1) ΕΡΓΑΛΕΙΟ ΠΡΟΣΟΜΟΙΩΣΗΣ

Στα πλαίσια της εκπόνησης του πρώτου μέρους της διπλωματικής εργασίας θα χρησιμοποιήσουμε ένα εργαλείο προσομοίωσης για να ελέγχουμε την ορθή λειτουργία των ψηφιακών κυκλωμάτων (αθροιστών) που θα χρειαστούμε στην συνέχεια. Το εργαλείο προσομοίωσης που θα χρησιμοποιήσουμε είναι το **ModelSim SE** της εταιρείας **Model Technology** του ομίλου **Mentor Graphics Corporation**. Το περιβάλλον λειτουργίας του εργαλείου είναι το λειτουργικό σύστημα **Windows** και οι βασικές του λειτουργίες θα παρουσιαστούν παρακάτω. Το **ModelSim** είναι συμβατό με τις γλώσσες περιγραφής υλικού **VHDL (IEEE 1076-1987 & IEEE 1076-1993)** και **VERILOG**.

Το **ModelSim** είναι ένα εργαλείο το οποίο έχει πάρα πολλές δυνατότητες οι οποίες ξεκινούν από την συγγραφή απλού κώδικα **VHDL** και **VERILOG** και φτάνουν ως την αποσφαλμάτωση μέσω πολύπλοκων διαδικασιών. Πιο συγκεκριμένα στο **ModelSim** υποστηρίζονται οι παρακάτω δυνατότητες :

- Προσομοίωση σχεδίασης γραμμένης σε **VHDL**
- Προσομοίωση σχεδίασης γραμμένης σε **VERILOG**
- Προσομοίωση σχεδίασης γραμμένης σε μίξη των παραπάνω γλωσσών
- Αποσφαλμάτωση μιας σχεδίασης **VHDL**
- Προσομοίωση με χρήση του υποεργαλείου Performance Analyzer
- Προσομοίωση με Code Coverage

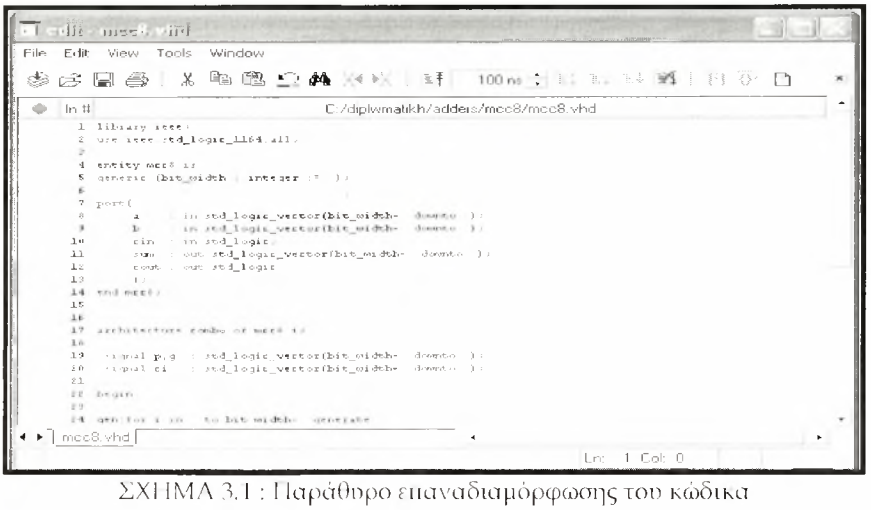
- Σύγκριση κυματομορφών

- Αποσφαλμάτωση με χρήση του παράθυρου ροής δεδομένων (Dataflow Window)

- Προσομοίωση σε Batch - Mode

3.2) ΔΙΑΔΙΚΑΣΙΑ ΠΡΟΣΟΜΟΙΩΣΗΣ :

Αρχικά, θα κάνουμε χρήση του παραθύρου **Source** για την **σύνταξη του πηγαίου κώδικα VHDL** της σχεδίασης του κάθε αθροιστή. Το παράθυρο αυτό παρουσιάζεται παρακάτω και σε αυτό μπορούμε να κάνουμε τις προγραμματιστικές μας προσθήκες και αλλαγές καθώς περιέχει στην ουσία την σχεδίασης μας.



ΣΧΗΜΑ 3.1 : Παράθυρο επαναδιαμόρφωσης του κώδικα

Έπειτα από την επιτυχή ολοκλήρωση της συγγραφής του κώδικα θα πρέπει να τον περάσουμε από τον μεταγλωττιστή **(compiler)** της γλώσσας **VHDL** , ο οποίος είναι ενσωματωμένος στο εργαλείο **ModelSim**. Αυτό μπορεί να γίνει είτε μέσω εντολών στο **main window** του **ModelSim**, είτε μέσω επιλογών με το ποντίκι των κατάλληλων περιεχομένων από τα μενού του εργαλείου (αναλυτικότερη περιγραφή του τρόπου ακολουθεί στην επόμενη παράγραφο υλοποιώντας ένα παράδειγμα).

Αφού περάσουμε επιτυχώς από το στάδιο της μεταγλώττισης του κώδικα του εκάστοτε αθροιστή σημαίνει πως η σχεδίαση μας είναι συντακτικά σωστή. Σε περίπτωση σφάλματος μπορούμε να κάνουμε **debugging** στα συντακτικά λάθη βασιζόμενοι στις παραπάνω τεχνικές που έχουμε

περιγράψει. Τελειώνοντας από αυτό το στάδιο θα πρέπει να κάνουμε φόρτωση **(loading)** της σχεδίασης μας ώστε να μπορέσουμε να κάνουμε την προσομοίωση που ζητάμε.

Στο επόμενο στάδιο, αφού έχουμε φορτώσει επιτυχώς την σχεδίαση μας προχωράμε στην διαδικασία της προσομοίωσης ούτως ώστε να βεβαιωθούμε πως η σχεδίαση μας ανταποκρίνεται επιτυχώς στις προσδοκίες μας και πραγματώνει εν τέλει αυτό που θέλουμε να κάνει. Εδώ, μέσω του παραθύρου των σημάτων **Signals Window** θέτουμε τιμές στα σήματα που μας ενδιαφέρουν ώστε να πάρουμε τις κατάλληλες αποκρίσεις στο **Wave Window** (παράθυρο των κυματομορφών).

Στην ουσία, για να ανακεφαλαιώσουμε, τα στάδια που θα διεκπεραιώσουμε είναι η σύνταξη του κώδικα της σχεδίασης, η μεταγλώττιση του κώδικα **VHDL** του κάθε αθροιστή, η φόρτωση του και τελικά η προσομοίωση του ώστε να επαληθεύσουμε και την σωστή λειτουργία χωρίς να ξεχνάμε σε κάθε στάδιο την εκσφαλμάτωση οποιοδήποτε λαθών εμφανιστούν.

Μεταγλώττιση και φόρτωση της σχεδίασης(Compiling and Loading).

• Ξεκινάμε με τη δημιουργία ενός νέου φακέλου εργασίας. Στον φάκελο αυτό αντιγράφουμε το αρχείο **adderi4.vhd** το οποίο είναι ο κώδικας σε **VHDL** του πιο απλού αθροιστή τεσσάρων **bits**.

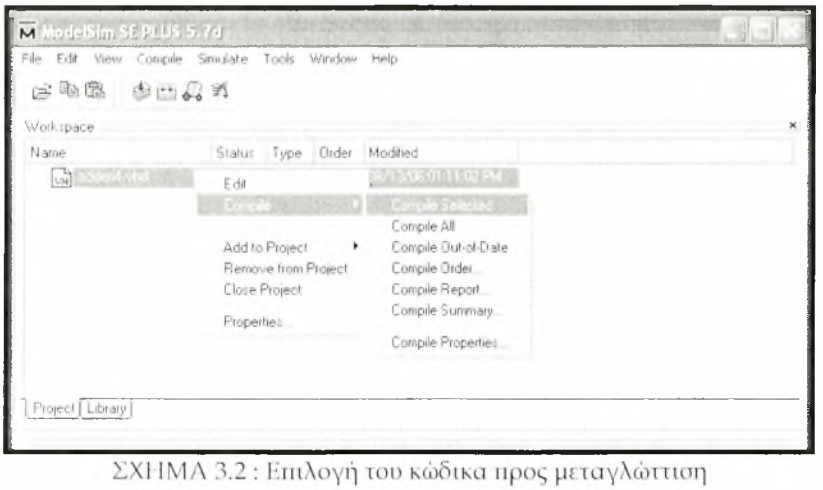
• Στην συνέχεια ξεκινάμε το εργαλείο προσομοίωσης **ModelSim SE**. Αυτό το επιτυγχάνουμε με διπλό κλικ στην συντόμευση στο **Desktop** του υπολογιστή μας ή με την χρήση της επιλογής από το **Start Menu > Programs**.

• Αφού έχει ανοίξει το εργαλείο προσομοίωσης με την επιλογή **File > Change Directory** (**Main Window**) επιλέγουμε ως φάκελο εργασίας τον φάκελο στον οποίο έχουμε αποθηκεύσει το αρχείο μας.

• Πριν μεταγλωττίσουμε ένα αρχείο κώδικα **HDL**, θα χρειαστεί να δημιουργήσουμε μια βιβλιοθήκη για την σχεδίαση, η οποία θα κρατήσει τα αποτελέσματα της μεταγλώττισης μας. Για να δημιουργήσουμε μια νέα βιβλιοθήκη σχεδίασης επιλέγουμε : **File > New > Library**.
Ή εναλλακτικά στο **command window** του **ModelSim** την ακόλουθη εντολή (PROMPT : **vlib library**)

• Στην συνέχεια για να αντιστοιχίσουμε την πραγματική μας βιβλιοθήκη με την λογική βιβλιοθήκη εργασίας **work** πρέπει να γράψουμε στην γραμμή εντολών την εντολή **vmap** ως εξής:
(PROMPT : **vmap work library**)

• Για να μεταγλωττίσουμε το αρχείο **VHDL** μέσα στην νέα βιβλιοθήκη θα πρέπει να επιλέξουμε **File > Open > File** και να επιλέξουμε το αρχείο μας προς μεταγλώττιση. Αφού γίνει αυτό έχουμε πολλούς τρόπους για να δώσουμε στο εργαλείο την εντολή προς μεταγλώττιση. Παρατηρούμε πως στην περιοχή του **workspace** εμφανίζεται το αρχείο μας οπότε εάν κάνουμε δεξί κλικ πάνω στην ονομασία του και επιλέγουμε **Compile>Compile Selected** έχουμε την μεταγλώττιση με το αμέσως επόμενο μήνυμα που εμφανίζεται στις παρακάτω εικόνες.



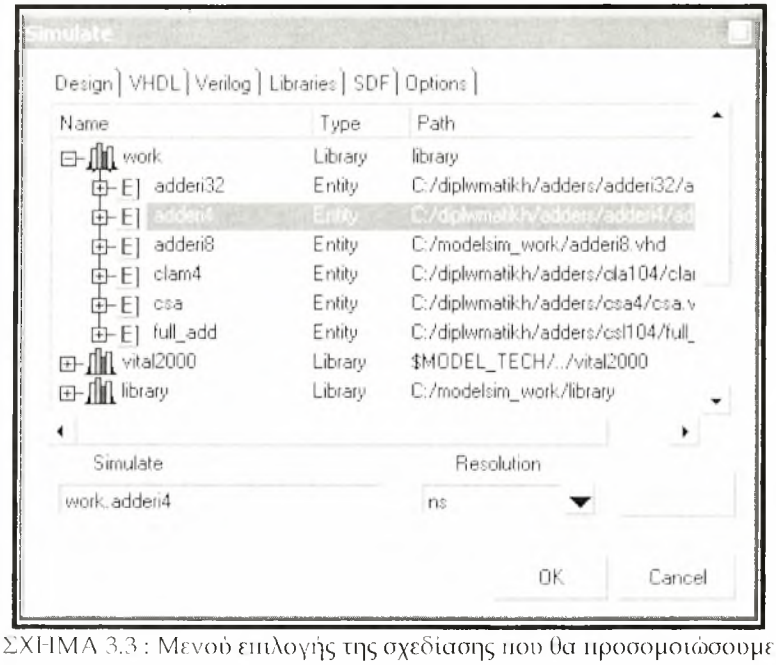
ΣΧΗΜΑ 3.2 : Επιλογή του κώδικα προς μεταγλώττιση

Επίσης ένας άλλος τρόπος να επιτύχουμε το παραπάνω είναι να γράψουμε στην γραμμή εντολών την παρακάτω εντολή.
(PROMPT : **vcom -work library adderi4.vhd**)

Προσομοίωση της σχεδίασης(Simulation - Running).

Αφού έχουμε τελειώσει επιτυχώς με το κομμάτι της μεταγλώττισης προχωράμε στην προσομοίωση του αθροιστή μας ακολουθώντας τα παρακάτω βήματα.

• Ανοίγουμε το παράθυρο προσομοίωσης επιλέγοντας από το μενού **Simulate > Simulate** και εν συνεχεία ανοίγουμε το φάκελο **work**.
• Επιλέγουμε **adderi4** και σιγουρευόμαστε ότι η ανάλυση του εργαλείου προσομοίωσης είναι σε **ns** και πατάμε **OK** όπως φαίνεται στην εικόνα.



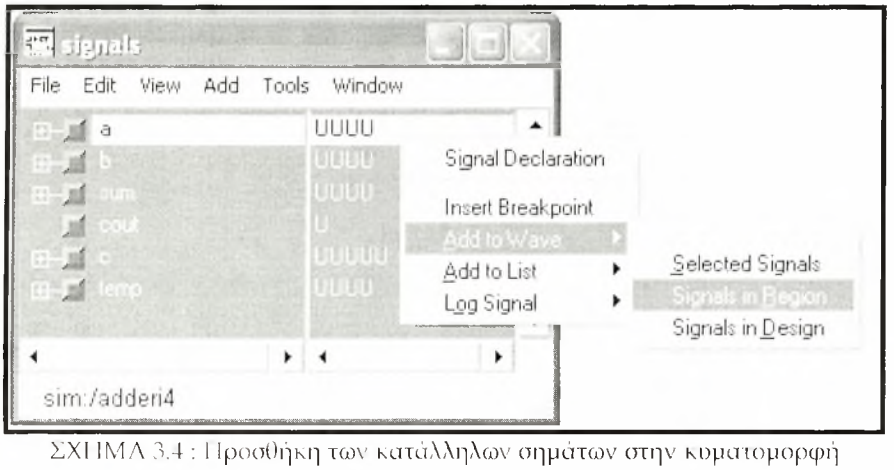
ΣΧΗΜΑ 3.3 : Μενού επιλογής της σχεδίασης που θα προσομοιώσουμε

Στην γραμμή εντολών έχουμε την παρακάτω εντολή.

(PROMPT: **vsim -t ns work. adderi4**)

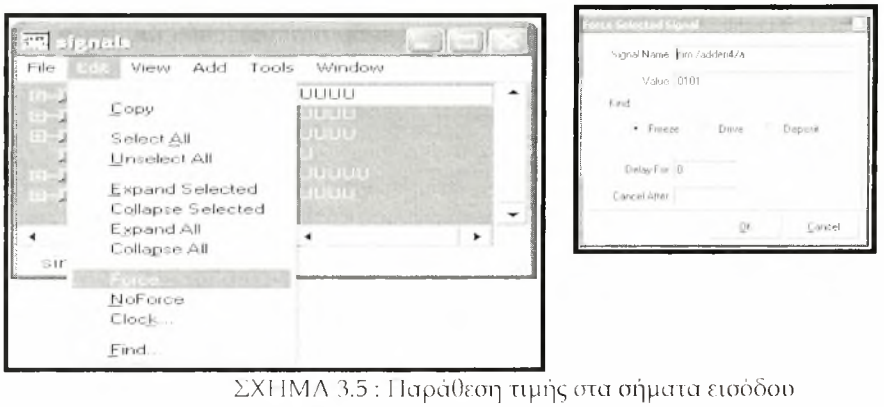
•Ανοίγουμε τα παράθυρα **Signals** και **Wave** με την επιλογή **View>Signals** και **View> Wave** αντίστοιχα.
(Main MENU: **View > <window name>)**

•Στο παράθυρο των σημάτων επιλέγω με δεξί κλικ τα παρακάτω ώστε να μπουν τα σήματα στο παράθυρο των κυματομορφών και το μόνο που απομένει είναι να δώσουμε τιμές στα σήματα ώστε να λάβουμε την κατάλληλη απόκριση κατά την διάρκεια του τρεξίματος.



ΣΧΗΜΑ 3.4 : Προσθήκη των κατάλληλων σημάτων στην κυματομορφή

Αυτό γίνεται με την εντολή **Force** η οποία βρίσκεται στο μενού **Edit>Force** που μας εμφανίζει το παράθυρο που εικονίζεται παρακάτω.



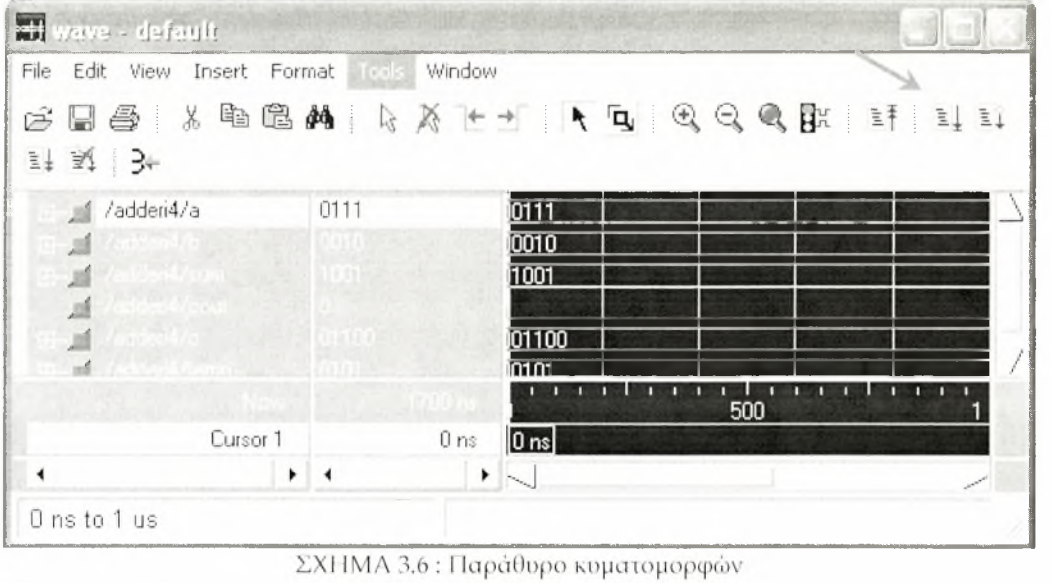
ΣΧΗΜΑ 3.5 : Παράθεση τιμής στα σήματα εισόδου

Στην γραμμή εντολών έχουμε τα παρακάτω (για τις εισόδους **a** και **b** αντίστοιχα) :

(PROMPT: **force –freeze sim:/adderi4/a 0111 0**)

(PROMPT: **force –freeze sim:/adderi4/b 0010 0**)

• Εκκινούμε την προσομοίωση για χρονικό διάστημα 1000 **ns** με την εντολή: (PROMPT: **run 1000**). Ή κλικάροντας το κατάλληλο κουμπί που υποδεικνύεται από το κόκκινο βέλος στο παρακάτω σχήμα.



ΣΧΗΜΑ 3.6 : Παράθυρο κυματομορφών

Επίσης υπάρχει η δυνατότητα να ελέγξουμε τα κυκλώματα με την εισαγωγή **testbenches** ώστε να μην χρειάζεται να βάζουμε χειρονακτικά τα δεδομένα εισόδου.

VCD (Value Change Dump) Files :

Ο λόγος που γίνεται αναφορά σε αυτό το κομμάτι των **VCD** αρχείων είναι διπλός. Σε πρώτη φάση, αναφέρεται στο συγκεκριμένο εδάφιο διότι η εξαγωγή του γίνεται μέσω του εργαλείου προσομοίωσης **Modelsim**. Σε δεύτερη φάση η χρησιμότητα των αρχείων αυτών θα διαφανεί παρακάτω και πιο συγκεκριμένα στο κομμάτι υπολογισμού της καταναλισκομένης ενέργειας κυκλωμάτων με τα εργαλεία της **Synopsys**.

Τα **VCD** αρχεία αποτελούν **ASCII** αρχεία τα οποία περιέχουν πληροφορίες για τη σύνθεση του κυκλώματός, ορισμούς για τις μεταβλητές του και τις μεταβολές που υφίστανται οι τιμές των μεταβλητών. Κατά κύριο λόγο έχουν εφαρμογή σε σχεδιάσεις που είναι υλοποιημένες σε **Verilog**, όμως στην περίπτωση μας θα εκμεταλλευτούμε την υποστηριζόμενη δυνατότητα του εργαλείου προσομοίωσης που χρησιμοποιείται ώστε να γίνει εφαρμογή σε αρχεία **VHDL**.

Η ροή που ακολουθείται σε αυτό το κομμάτι διακλαδίζεται , καθώς υπάρχει η δυνατότητα επιλογής ανάμεσα σε δυο διαφορετικές ροές δημιουργίας **VCD** αρχείων. Η πρώτη από της δυο , και αυτή που ακολουθείται στην παρούσα εργασία παράγει ένα **four-state VCD** αρχείο, χωρίς πληροφορίες που αφορούν την δύναμη οδήγησης των πυλών. Η δεύτερη, παράγει ένα εκτενέστερο αρχείο με πληροφορίες τόσο για την δύναμη οδήγησης των πυλών όσο και για τα δεδομένα και την συμπεριφορά τους σε κάθε **port** της σχεδίασης. Δυστυχώς δεν κατέστη δυνατόν η υπερπήδηση των εμποδίων συμβατότητας που εμφανιστήκαν με τα εργαλεία

της **Synopsys** και του δεύτερου τρόπου παραγωγής αρχείων **VCD** πράγμα που μας οδήγησε στην επιλογή της πρώτου.

Στο καθαρά τεχνικό – διαδικαστικό κομμάτι της ροής που ακολουθείται η διαδικασία υπαγορεύει την σύνταξη ενός αρχείου **testbench** σε γλώσσα **VHDL**. Το αρχείο αυτό συντάσσεται με βάση την έξοδο της σύνθεσης του κυκλώματος (η ακριβής διαδικασία αναφέρεται και αναλύεται ενδελεχώς σε επόμενο ξεχωριστό κεφάλαιο) η οποία είναι μορφής **.vhdl**. Στον χρόνο 0 της προσομοίωσης της εξόδου δίνουμε τις ακόλουθες εντολές σε μορφή **script** στην κονσόλα του **Modelsim** :

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

Οι παραδοχές που έχουν γίνει είναι πως υπάρχει στη διάθεση του σχεδιαστή ένα κύκλωμα αθροιστή **4-bit** με εισόδους "**a**" και "**b**" και κρατούμενο εισόδου "**cin**". Η διαδικασία ξεκινάει στην χρονική στιγμή 0 και με την πρώτη εντολή εισάγεται 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, το κύκλωμα λειτουργεί 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.

Σαρώνοντας έτσι όλο δυνατό εύρος των τιμών εισόδου γίνεται η προσπάθεια μεγιστοποίησης του εύρους ζώνης της πληροφορίας που θα συλλεχθεί. Η διαδικασία σταματάει σε κάποια χρονική στιγμή με την εντολή **quit –sim** η οποία σταματάει την προσομοίωση.

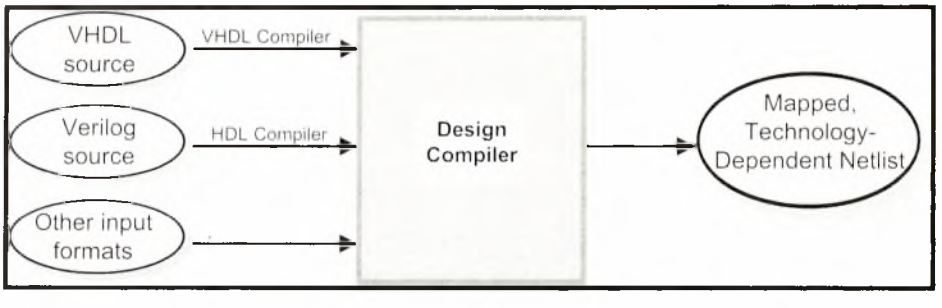
4.ΣΥΝΘΕΣΗ

4.1) ΕΙΣΑΓΩΓΗ :

Όπως έχει αναφερθεί και σε προηγούμενη ενότητα η ροή σχεδίασης (**design flow**) που θα ακολουθηθεί περιλαμβάνει την χρήση τριών εργαλείων της πλατφόρμας της **Synopsys**: του **Design Compiler**, του **Power Compiler** και του **PrimeTime**. Στο προκείμενο εδάφιο θα γίνει αναφορά στο πρώτο από αυτά, το οποίο χρησιμοποιείται για την διαδικασία της σύνθεσης, για την οποία ακριβής ορισμός θα δωθεί στην συνέχεια.

4.2) DESIGN COMPILER :

Ο **Design Compiler** (DC) χρησιμοποιείται για την πραγματοποίηση της σύνθεσης ψηφιακών κυκλωμάτων. Προσφέρει την δυνατότητα βελτιστοποίησης της σχεδίασης που εισάγεται για σύνθεση με στόχο την παροχή μικρότερων και γρηγορότερων αναπαραστάσεων μιας λογικής συνάρτησης. Ο DC είναι μια πλατφόρμα πάνω στην οποία «τρέχουν» κάποια υποεργαλεία με σκοπό την πραγμάτωση του παραπάνω στόχου. Τα υποεργαλεία αυτά έχουν την δυνατότητα σύνθεσης **HDL** σχεδιάσεων σε εξαρτημένες σχεδιάσεις σε επίπεδο πυλών. Επιπρόσθετα, ο DC υποστηρίζει την δυνατότητα σύνθεσης τόσο για ακολουθιακά όσο και για συνδιαστικά κυκλώματα βελτιώνοντας την ταχύτητα απόκρισής τους, τον χώρο που καταλαμβάνουν και τελικά και την ισχύ που καταναλώνουν. Στην εικόνα που ακολουθεί παρουσιάζεται η διαδικασία της σύνθεσης σε σχέση με τις εισόδους και τις εξόδους που δέχεται και παράγει ο DC.



ΣΧΗΜΑ 4.1 : Είσοδος και έξοδος του Design Compiler

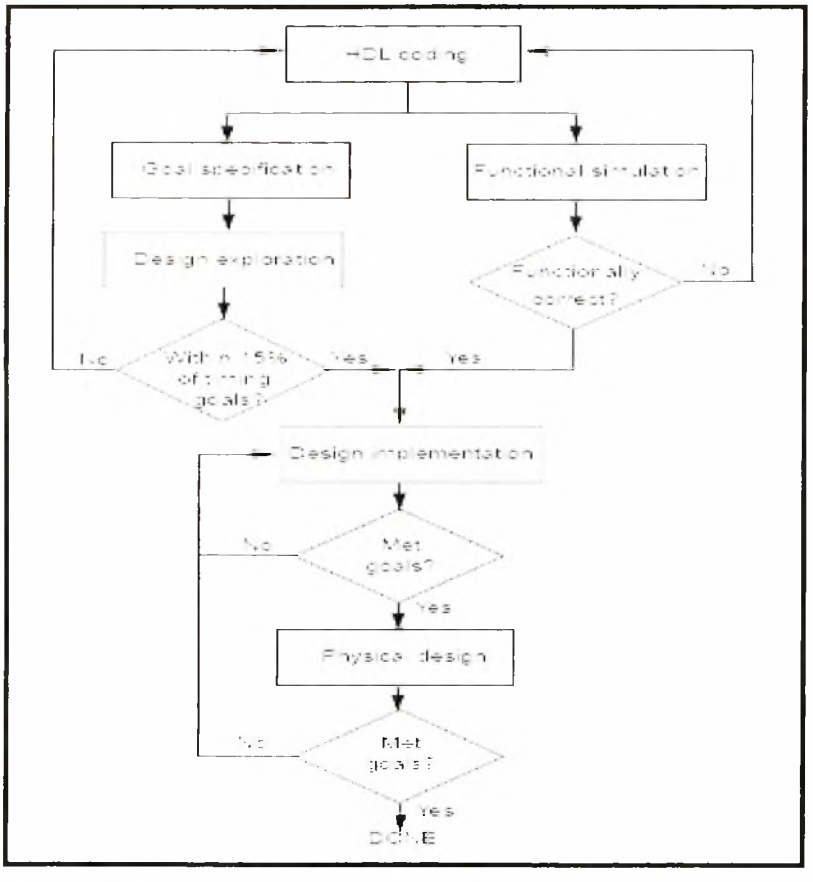
Ο DC μας δίνει την δυνατότητα να επιλέξουμε ένα από τα δύο περιβάλλοντα εργασίας που διαθέτει, τα οποία είναι τα ακόλουθα :

- Το κέλυφος του DC, δηλαδή στην ουσία η γραμμή εντολών του, που είναι γνωστό ως **dc_shell**. Το **shell** αυτό υποστηρίζει δύο γλώσσες, την

dcsh (design compiler shell language) και την γλώσσα πρότυπο για **EDA** εργαλεία **tcl (tool command language)**.

- Η γραφική διεπαφή (**GUI**) του **Design Compiler** η οποία απαντάται σε δύο «εκδόσεις» : τον **Design Analyzer** και το **Design Vision**. Για την παρούσα εργασία αποφασίσαμε να μην ασχοληθούμε με τον **Design Analyzer**, αλλά με το **Design Vision** λόγω κάποιων επιπλέον δυνατοτήτων που μας παρέχει.

Όταν μας ενδιαφέρει τόσο μια τελική υλοποίηση ενός ψηφιακού κυκλώματος, όσο και η αναζήτηση με δοκιμές της καταλληλότερης αρχιτεκτονικής του, καλούμαστε να ακολουθήσουμε μια βασική ροή σύνθεσης η οποία περιγράφεται επαρκώς από το ακόλουθο σχήμα :



ΣΧΗΜΑ 4.2 : Η βασική ροή σύνθεσης και υλοποίησης ενός ψηφιακού κυκλώματος

Τα βήματα που περιγράφονται στο παραπάνω σχήμα περιγράφονται αναλυτικά στο εδάφιο που ακολουθεί :

- ✓ Σε πρώτη φάση το κύκλωμα υλοποιείται με την χρήση μιας **HDL** γλώσσας (**VHDL** ή **Verilog**). Βέβαια για να μπορέσουμε να πάρουμε πιο εποικοδομητικά τελικά αποτελέσματα φροντίζουμε ο κώδικας εισαγωγής της σχεδίασης να ακολουθεί κάποιες ενδεδειγμένες πρακτικές ανάπτυξης.

- ✓ Ακολουθεί η αναζήτηση της κατάλληλης αρχιτεκτονικής και η προσομοίωση του κυκλώματος. Η διαδικασία αυτή μπορεί να μας εξοικονομήσει χρόνο στην συνέχεια, οπότε είναι σχεδόν πάντα τακτική να κάνουμε τα εξής :

  - Μέσω του **DC** υλοποιούμε δομικές σχεδιαστικές λεπτομέρειες, οι οποίες συχνά είναι η αποσαφήνιση των κανόνων σχεδίασης και η βελτιστοποίηση των περιορισμών. Μετά από αυτό το στάδιο ακολουθεί μια αρχική υλοποιημένη μορφή της σχεδίασης μας.

  - Σε αυτό το σημείο υπεισέρχεται ένας περιορισμός της τάξης του 15% σε θέματα χρονισμού. Εάν δεν ικανοποιούν αυτό το κριτήριο τα αποτελέσματα της σχεδίασης μας, τότε αναπροσαρμόζουμε τον κώδικα μας. Και ξεκινάμε πάλι από την αρχή.

  - Κατόπιν ακολουθεί η προσομοίωση με τα κατάλληλα εργαλεία ώστε να επαληθεύσουμε εάν αν το κύκλωμα εκτελεί σωστά όλες τις λειτουργικότητες του

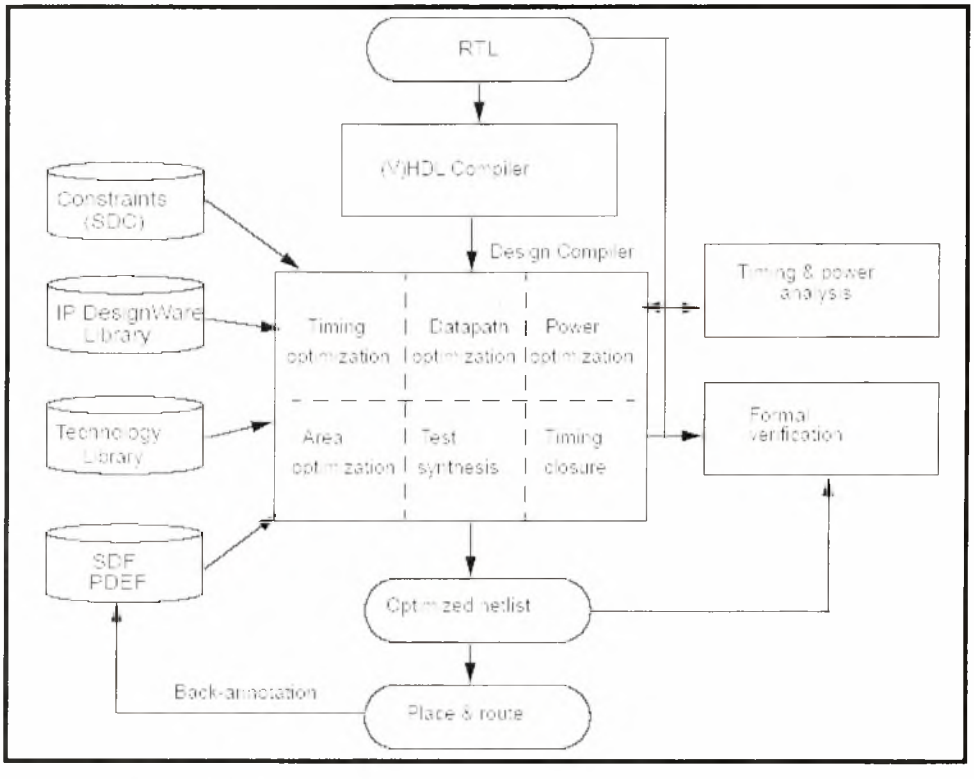
- Για οποιοδήποτε λάθος σχεδιαστικό ή μη διορθώνουμε τον **HDL** κώδικα και επαναλαμβάνουμε από την αρχή.
- Συνεπώς έχουμε τελειώσει όταν οι χρονικές απαιτήσεις δεν υπερβαίνουν την σχεδιαστική σταθερά του 15%.

✓ Κατόπιν εκτελούμε με τη χρήση όλων των δυνατοτήτων του **DC** την τελική υλοποίηση του κυκλώματος με σκοπό να καλυφθούν οι χρονικές απαιτήσεις που έχουμε εισάγει πριν. Σε αυτό το στάδιο έχουμε περάσει από τον κώδικα σε απεικόνιση της σχεδίασης μας σε επίπεδο πυλών και θα πρέπει να επαληθεύσουμε εάν έχουν καλυφθεί πλήρως οι προδιαγραφές που έχουμε θέσει από την αρχή. Αυτό είναι εφικτό κατόπιν της ανάλυσης των αναφορών σε κάθε επίπεδο που μας επιστρέφει ο **DC**, καθώς μπορούμε να ορίσουμε ποια τεχνική θα μας οδηγήσει στο ζητούμενο.

✓ Τέλος, αφού έχουμε ένα παραμετροποιημένο κύκλωμα σύμφωνα με τις αρχικές σχεδιαστικές απαιτήσεις μας περνάμε στο τελευταίο επίπεδο της μεταφοράς στο φυσικό επίπεδο. Ο έλεγχος της αποδοτικότητας γίνεται και εδώ ο οποίος έχει τις εξής δυο εξόδους. Σε περίπτωση που όλα είναι σύμφωνα με τα **constraints** που έχουμε θέσει στην αρχή τότε έχουμε επιτυχία και έξοδο από τη ροή σχεδίασης ενώ σε αντίθετη περίπτωση επιστρέφουμε στο προηγούμενο βήμα για να επαναπροσδιορίσουμε την σχεδίαση μας.

Στο σχήμα που ακολουθεί παρουσιάζεται μια πιο αναλυτική θεώρηση της παραπάνω ροής με την διαφορά πως είναι πια σαφώς ορισμένες οι ενέργειες και οι δυνατότητες του **DC**. Κατόπιν της εισαγωγής αρχείων τα οποία περιγράφουν το κύκλωμά μας σε επίπεδο καταχωρητών (**RTL – Register Transfer Level**), ακολουθεί η σύνθεση, η τεχνολογική αναπαράσταση κατά κάποιο τρόπο, του κυκλώματός μας.

Κατά τη σύνθεση ο DC μεταφράζει την HDL περιγραφή σε συνθετικά στοιχεία της **DesignWare** βιβλιοθήκης, όπως είναι για παράδειγμα οι αθροιστές που θα χρησιμοποιήσουμε σαν βάση σε αυτή την εργασία. Ακολούθως ο DC χρησιμοποιεί τεχνολογικές (**technology**), συνθετικές (**synthetic**) και συμβολικές (**symbolic**) βιβλιοθήκες προκειμένου να γίνει η αντιστοίχηση των κυκλωματικών στοιχείων (αναλυτική περιγραφή των βιβλιοθηκών ακολουθεί σε επόμενη παράγραφο).



ΣΧΗΜΑ 4.3 : Αναλυτική περιγραφή βάση των δυνατοτήτων του DC της ακολουθούμενης ροής σχεδίασης

Είναι φανερό πως οι περιορισμοί (**constraints**) που έχουμε ορίσει από την αρχή επηρεάζουν την σύνθεση και σε συνδυασμό με τις βιβλιοθήκες που έχουμε ορίσει, στην ουσία κρίνουν το αποτέλεσμα αυτού του σταδίου. Είναι σημαντικό να τονίσουμε το γεγονός πως τα **constraints** που θέτει ο σχεδιαστής ή που υπάρχουν εξορισμού στις βιβλιοθήκες που χρησιμοποιούνται απευθύνονται στους τομείς της καταλαμβανόμενης

περιοχής, της καταναλισκόμενης ισχύος και του χρονισμού. Οι δυνατότητες βελτιστοποίησης λοιπόν, του **DC** είναι οι ακόλουθες :

1. ***Timing optimization*** σε ότι έχει σχέση με τον χρονισμό του κυκλώματος

2. ***Area optimization*** σε ότι αφορά την εκτιμώμενη επιφάνεια που θα καλύπτει το κύκλωμα.

3. ***Datapath optimization*** σε ότι αφορά την βελτιστοποίηση των μονοπατιών διάδοσης της πληροφορίας.

4. ***Power optimization*** σε ότι έχει να κάνει με την καταναλισκόμενη ενέργεια του κυκλώματος.

Η έξοδος αυτού του σταδίου είναι ένα **optimized netlist** το οποίο αποτελεί την είσοδο των **place & route** εργαλείων της προκείμενης πλατφόρμας. Το κύκλωμά, όπως έχει πλέον διαμορφωθεί, αποτελεί είσοδο στα 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 σε φυσικό επίπεδο.

Πλέον διαφαίνονται στο κατώτατο επίπεδο επιπρόσθετα στοιχεία, όπως οι καθυστερήσεις που παρουσιάζονται στις εσωτερικές διασυνδέσεις (**interconnection delays**) με πιο ρεαλιστικές τιμές. Η δυνατότητα που παρέχεται σε αυτό το σημείο είναι η ανατροφοδότηση των εκτιμώμενων δεδομένων των προηγουμένων σταδίων που πλέον χαρακτηρίζονται από πραγματικά δεδομένα. Οι παρασιτικές χωρητικότητες καθώς και τα φορτία των διασυνδέσεων ανατροφοδοτούνται σε ανώτερο επίπεδο της ροής ,για παράδειγμα με την μορφή **spef** ή **pdef** αρχείων, και αναπροσαρμόζουν μια

νέα σύνθεση, επιτρέποντας στο σχεδιαστή να δει μια πιο βελτιωμένη έξοδο χρονισμού του αρχικού κυκλώματος.

Ο **Design Compiler** μπορεί να χειριστεί αρχεία που προσδιορίζουν σχεδιάσεις σε πολλαπλές μορφές που αποτελούν **standards** στην ηλεκτρονική σχεδίαση ψηφιακών κυκλωμάτων. Η συμβατότητα του τύπου των αρχείων που διαχειρίζεται ο **DC** (**.db** και **.eqn**) με άλλα **EDA** (**Electronic Design Automation**) εργαλεία αποτέλεσε και σημαντικό παράγοντα επιλογής του εργαλείου της **Synopsys** σε αυτό τον τομέα.

Στα παραπάνω διαφαίνεται η συνολική ροή υλοποίησης μιας σχεδίαση από την στιγμή συγγραφής κώδικα **HDL** μέχρι την εξαγωγή **layout**. Για να γίνει αντιληπτή η συμβολή του **DC** σε όλη αυτή την ακολουθία θα πρέπει πρώτα να ξεκαθαριστούν κάποιοι ορισμοί οι οποίοι είναι απαραίτητη για την συνέχεια. Αυτό είναι απαραίτητο διότι οι εννοιολογικές διαφορές τους είναι μικρές και υπάρχει το ενδεχόμενο σύγχυσης.

4.3) ΟΡΙΣΜΟΙ :

*«Σύνθεση (**Synthesis**) 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. Περιλαμβάνει την ανάγνωση του πηγαίου κώδικα **HDL** και την βελτιστοποίηση της σχεδίασης από αυτή την απεικόνιση».*

*« Βελτιστοποίηση (**Optimization**) είναι το βήμα κατά την διαδικασία της σύνθεσης το οποίο προσπαθεί να ενθυλακώσει ένα συνδυασμό από **library cells** τα οποία ικανοποιούν όσο το δυνατόν περισσότερο τους λειτουργικούς, χρονικούς και χωρικούς περιορισμούς – απαιτήσεις που έχουν ζητηθεί».*

*«Μεταγλώττιση (**Compile**) είναι η ενέργεια του **DC** η οποία εκτελεί το βήμα της βελτιστοποίησης. Μετά την ανάγνωση της σχεδίασης και την διενέργεια άλλων καταλλήλων πράξεων , εκτελείται αυτή η εντολή ώστε να δημιουργηθεί ένα **gate-level netlist** της σχεδίασης».*

4.4) DESIGN VISION :

Το **Design Vision** είναι κατ'ουσία μια γραφική διεπαφή για την χρήση του **Design Compiler**. Δύναται, επομένως να χρησιμοποιηθεί για την ανάλυση και την σύνθεση ψηφιακών κυκλωμάτων.

Σαν μετεξέλιξη του **Design Analyzer** υπάρχει μία 1 – 1 αναλογία του μενού επιλογών και εντολών σύνθεσης με αυτό του κελύφους (**dc_shell**) του DC. Άλλωστε αυτό είναι και το πραγματικό νόημα ανάπτυξης ενός τέτοιου είδους εργαλείου. Λόγω του γεγονότος πως τόσο το **Design Vision** όσο και ο **Design Compiler** χρησιμοποιούν την ίδια «μηχανή» ανάλυσης στατικού χρονισμού συνάγουν στο κοινώς αποδεκτό και από την πράξη πως όλες οι δυνατότητες του DC μπορούν να χρησιμοποιηθούν επαρκώς μέσα από την κονσόλα που μας προσφέρει το **Design Vision** χωρίς να έχουμε απώλεια κάποιων εντολών ή κάποιου μέρους τους.

Λόγω του όγκου των κυκλωμάτων που θα αναλύσουμε όπως είναι εύκολα κατανοητό θα ήταν απαγορευτική η χρήση χειροκίνητης ένθεσης των εντολών από άποψη χρόνου. Σε αυτό το κομμάτι έγινε χρήση της δυνατότητας χρήσης ενός **script**, γλώσσας **tcl** που υποστηρίζεται από την πλατφόρμα που χρησιμοποιούμε και του οποίου η γενικευμένη μορφή παρουσιάζεται και αναλύεται παρακάτω :

---

1. *search_path = {/path}*
2. *link_library = {/path/umce13h210t3_wc_108V_125C.db}*
3. *symbol_library = /path/umce13h210t3.sdb}*
4. *target_library = {/path/umce13h210t3_wc_108V_125C.db}*

5. *analyze -library WORK -format vhdl {/path/adder_name.vhd}*
6. *elaborate adder_name -library WORK*
7. *set_load 0.03 all_inputs()*
8. *set_load 0.03 all_outputs()*
9. *set_max_fanout 3 adder_name*
10. *set_max_transition 0.1 adder_name*
11. *set_max_delay 0.5 -from all_inputs() -to all_outputs()*
12. *set_wire_load_model -name suggested_10K -library umce13h210t3_wc_108V_125C*
13. *compile -map_effort high -area_effort high -ungroup_all*
14. *write -hierarchy -output adder_name.db*
15. *write -hierarchy -format vhdl -output adder_name.vhdl*
16. *write_parasitics -output parasitics.reduced.txt*
17. *write_parasitics -output parasitics.reduced.spef*
18. *echo "antzimas> Building Reports..."*
19. *sh mkdir ./adder_name/Reports*
20. *report_compile_options >> ./adder_name/Reports/ReportCompile_Options.txt*
21. *report_net >> ./adder_name/Reports/ReportNet.txt*
22. *report_cell >> ./adder_name/Reports/ReportCell.txt*

23. *report_constraint -significant_digits 2 -verbose >> ./ adder_name /Reports/ReportConstraints.txt*

24. *report_design >> ./adder_name/Reports/ReportDesign.txt*

25. *report_area >> ./adder_name/Reports/ReportArea.txt*

26. *report_timing -from all_inputs() -to all_outputs() >> ./ adder_name /Reports/ReportTiming.txt*

Οι παραπάνω εντολές σαν μια ενιαία οντότητα αποτελούν ένα script το οποίο μπορεί να εισαχθεί στην κονσόλα του **dc_shell** με την εντολή

- dc_shell> include / path/ *adder_name*.script

Πιο συγκεκριμένα με την εντολή ***search_path*** καθορίζεται το σημείο στο οποίο 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. Στην περίπτωση της εκπόνησης αυτής της εργασίας τα αρχεία είναι γραμμένα σε **VHDL** και τα αρχεία είναι της μορφής ***adder_name.vhd***.

Λόγω του γεγονότος ότι υπάρχει το υπόβαθρο κάποιας βιβλιοθήκης, εύλογο είναι σε κάποιο σημείο να οριστεί η τοποθεσία που βρίσκεται αυτή ή αυτές, αν πρόκειται για περισσότερες από μια, ώστε ο **DC** να μπορέσει να επικοινωνήσει με αυτές λαμβάνοντας και αποθηκεύοντας όποια στοιχεία χρειαστεί κατά τη διάρκεια των διεργασιών του. Η σύνδεση με τις τεχνολογικές βιβλιοθήκες που προσδιορίζουν τα κελιά και άλλες πληροφορίες ανάλογα με τον «πάροχο» της τεχνολογίας, όπως τα ονόματα των κελιών τους σχεδιαστικούς κανόνες και τις συνθήκες λειτουργίας γίνεται μέσω των

εντολών που ακολουθούν, όποιες είναι οι εντολές **link_library** και **target_library**. Η σύνδεση , όπως είναι εύκολα αντιληπτό , με αυτές τις βιβλιοθήκες γίνεται με την πρώτη εντολή ενώ η δεύτερη αποτελεί τον δείκτη στην βιβλιοθήκη αποθήκευσης. Τα δεδομένα που είναι υπεύθυνα για την δημιουργία των συμβολικών απεικονίσεων των πυλών, για παράδειγμα, και κατ' επέκταση την δημιουργία των κατάλληλων **schematics** βρίσκονται και προσπελαύνονται μέσω της εντολής βιβλιοθήκης **symbol_library**.

Το επόμενο βήμα είναι να διαβάσουμε την σχεδίασή μας από τον δίσκο στην ενεργή μνήμη του εργαλείου. Στην ενεργή του μνήμη ο σχεδιαστής μπορεί να επέμβει και να κάνει όλες τις απαραίτητες αλλαγές για την σχεδίαση του πριν αποθηκεύσει πάλι την σχεδίαση στην βιβλιοθήκη από όπου την κάλεσε.

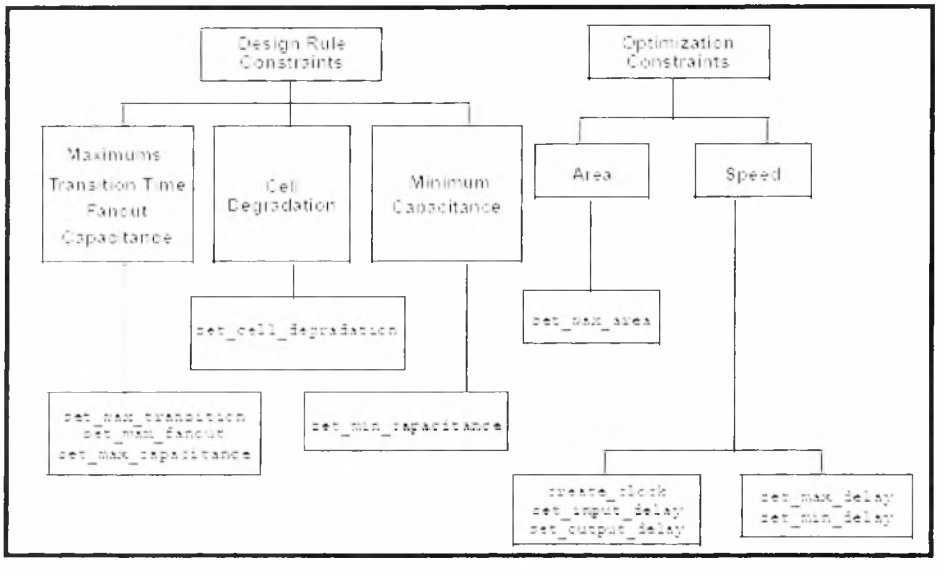
Αυτό γίνεται με το συνδυασμό των δυο επόμενων εντολών **analyze & elaborate** η κλίση των οποίων ισοδυναμεί και με την μετατροπή της **HDL** σχεδίασης σε αρχεία βάσης δεδομένων στο **format** της **Synopsys** (**.db**). Τα ορίσματα **-library WORK -format vhdl** δείχνουν στο **DC** το χώρο της ενεργής μνήμης, δηλαδή το φάκελο εργασίας, και το **format** του αρχείου εισόδου. Ισοδύναμη εντολή με τον συνδυασμό και των δυο προαναφερθέντων είναι η **read** η οποία χρησιμοποιείται όταν θέλουμε να διαβάσουμε σχεδιάσεις που είναι ήδη σε **.db** μορφή.

Η **analyze** εξετάζει το αρχείο εισόδου της και ελέγχει την συντακτική λογική και την δυνατότητα σύνθεσης του. Για την σωστή μετάβαση στο επόμενο στάδιο της ροής μετατρέπει το **vhd** αρχείο σε ένα ενδιάμεσο **format** και τοποθετεί το αρχείο αυτό στον φάκελο που έχουμε προσδιορίσει ως φάκελο εργασίας (**working directory**).

Εν συνεχεία η εισαγωγή της εντολής **elaborate** είναι υπεύθυνη της εξέτασης του ενδιάμεσου αρχείου και την δημιουργία της σχεδίασης σε **.db**. Σε

αυτό το σημείο καθορίζονται ποια από τα «στοιχεία» της σχεδίασης πρέπει να αντικατασταθούν από συνθετικά «στοιχεία» της βιβλιοθήκης που έχει επιλεχθεί.

Ακολούθως θα πρέπει να καθοριστεί το περιβάλλον της σχεδίασης και οι περιορισμοί/παράμετροι της σχεδίασης. Ο DC απαιτεί την μοντελοποίηση του σχεδιαστικού περιβάλλοντος του κυκλώματος ώστε να προχωρήσει η διαδικασία της σύνθεσης. Ένα πολύ απλό μοντέλο με κάποιες εντολές που θα χρησιμοποιήσουμε φαίνεται παρακάτω περιγράφοντας τον τομέα περιορισμού/παραμέτρων και δίνοντας και την κατάλληλη εντολή για **constraint**.



ΣΧΗΜΑ 4.4 : Σχεδιαστικοί περιορισμοί και περιορισμοί βελτιστοποίησης

Σύμφωνα με τα βιομηχανικά πρότυπα σχεδίασης και υλοποίησης κυκλωμάτων και λαμβάνοντας υπ' όψη την κατηγορία των κυκλωμάτων με τα οποία διεξάγεται η παρούσα μελέτη θα πρέπει να τεθεί ένα φορτίο σε όλες τις εισόδους και εξόδους. Αυτό είναι εξ' ορισμού για τους αθροιστές τάξης των 0.03 και επέρχεται με τις εντολές:

- set_load 0.03 all_inputs()

- set_load 0.03 all_outputs()

Για τους ίδιους λόγους που περιγράψαμε πιο πάνω θα πρέπει να φραχτεί το μέγιστο **fanout** στα **ports** εισόδου, οπότε η τιμή 3 στην ουσία θα φράξει το μέγιστο **fanout** που θα παρατηρηθεί στο κύκλωμα – αθροιστή με την εντολή :

- set_max_fanout 3 *adder_name*

Ο μέγιστος χρόνος μεταφοράς (**maximum transition time**) για ένα δίκτυο είναι ο μεγαλύτερος χρόνος που απαιτείται για ένα **pin** οδήγησης (**driving pin**) να αλλάξει λογική τιμή. Πολλές τεχνολογικές βιβλιοθήκες περιέχουν περιορισμούς για το μέγιστο χρόνο μεταφοράς, οι οποίοι όμως ποικίλουν ανά βιβλιοθήκη. Ο **Design Compiler** προσπαθεί να καθορίσει το **maximum transition time** για κάθε **net** ξεχωριστά. Αυτό το πετυχαίνει κάνοντας **buffer** στην έξοδο της πύλη που οδηγεί (**driving gate**). Ο μέγιστος χρόνος μεταφοράς μπορεί να ποικίλει, λόγω της εξάρτησης του από την συχνότητα λειτουργίας του κελιού. Η τιμή που έχει οριστεί για την σύνθεση που διεξάγεται είναι το 0.1 με την εντολή :

- set_max_transition 0.1 *adder_name*

Οι χρόνοι μεταφοράς σε δίκτυα είναι υπολογισμένοι με την χρήση δεδομένων χρονισμού από τις τεχνολογικές βιβλιοθήκες. Σαφέστατα όμως η ποσότητα της μέγιστης μεταφοράς δεν παρέχει έναν απευθείας τρόπο ώστε να περιοριστεί η πραγματική χωρητικότητα των **nets**. Για να γίνει αυτό θα πρέπει να χρησιμοποιηθεί εντολή περιορισμού της χωρητικότητας.

Λόγω σχεδιαστικής απαίτησης θεωρήθηκε απαραίτητο να περιοριστούν οι είσοδοι και οι έξοδοι του κυκλώματος – αθροιστή ώστε να παρθούν χρήσιμα συμπεράσματα για τον χρονισμό των κυκλωμάτων της

μελετημένης κατηγορίας. Στη φάση της έρευνας που διεξάγεται έχει παρθεί η απόφαση της μελέτης του χρονισμού σε πρώτη φάση, η οποία αποτελεί έναν από τους κυρίους παράγοντες επηρεασμού της καταναλισκομένης ισχύος. Οπότε αυτό το πετυχαίνουμε με την παρακάτω εντολή :

- set_max_delay 0.5 -from all_inputs() -to all_outputs()

Στο περιβάλλον της σχεδίασης είναι κρίσιμο για το χρονισμό να τεθεί ένα **constraint** για να προσεγγιστεί το φορτίο και κατά συνέπεια τα παρασιτικά των διασυνδέσεων μέσα στο κύκλωμα. Έχει επιλεχθεί το μοντέλο της **umce13h210t3_wc_108V_125C** βιβλιοθήκης με την χαρακτηριστική ονομασία **suggested_10K** με τον παρακάτω τρόπο :

- set_wire_load_model -name suggested_10K -library umce13h210t3_wc_108V_125C

Στην συνέχεια η αποθήκευση πληροφοριών στο **format** της **Synopsys** γίνεται με την εντολή :

- write -hierarchy -output *adder_name*.db

Ενώ, οι ίδιες πληροφορίες αλλά σε **format vhdl** αποθηκεύονται με την:

- write -hierarchy -format vhdl -output *adder_name*.vhdl

Για τα παρασιτικά που παρατηρούνται στο κύκλωμα γίνεται χρήση της εντολής καταγραφής τους

- write_parasitics -output parasitics.reduced.spef
- write_parasitics -output parasitics.reduced.txt

σε μορφή **.spef** αρχείου, προκειμένου να μπορούν να χρησιμοποιηθούν οι πληροφορίες αυτές κατά τον υπολογισμό της ισχύος.

Για λόγους βελτίωσης της αναφοράς της κονσόλας του κελύφους του DC θεωρήθηκε καλό η χρησιμοποίηση της παρακάτω εντολής που εμφανίζει το μήνυμα στην οθόνη της κονσόλας.

- echo "antzimas> Building Reports..."

Τέλος, με χρήση της εντολής **report** και μιας σειράς παραμέτρων λαμβάνουμε πληροφορίες για τις επιλογές της μεταγλώττισης, για την σχεδίαση, τα κελιά, τα δίκτυα, τους περιορισμούς που έχουν τεθεί, τον χρονισμό, 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**. Για λόγους καλύτερης διαχείρισης της πληροφορίας προς ανάλυση δημιουργείται ένας φάκελος **Reports** που εμπεριέχει όλες τις αναφορές.

- sh mkdir ./ *adder_name* /Reports

- report_compile_options
  >> ./ *adder_name* /Reports/ReportCompile_Options.txt

- report_net >> ./ *adder_name* /Reports/ReportNet.txt

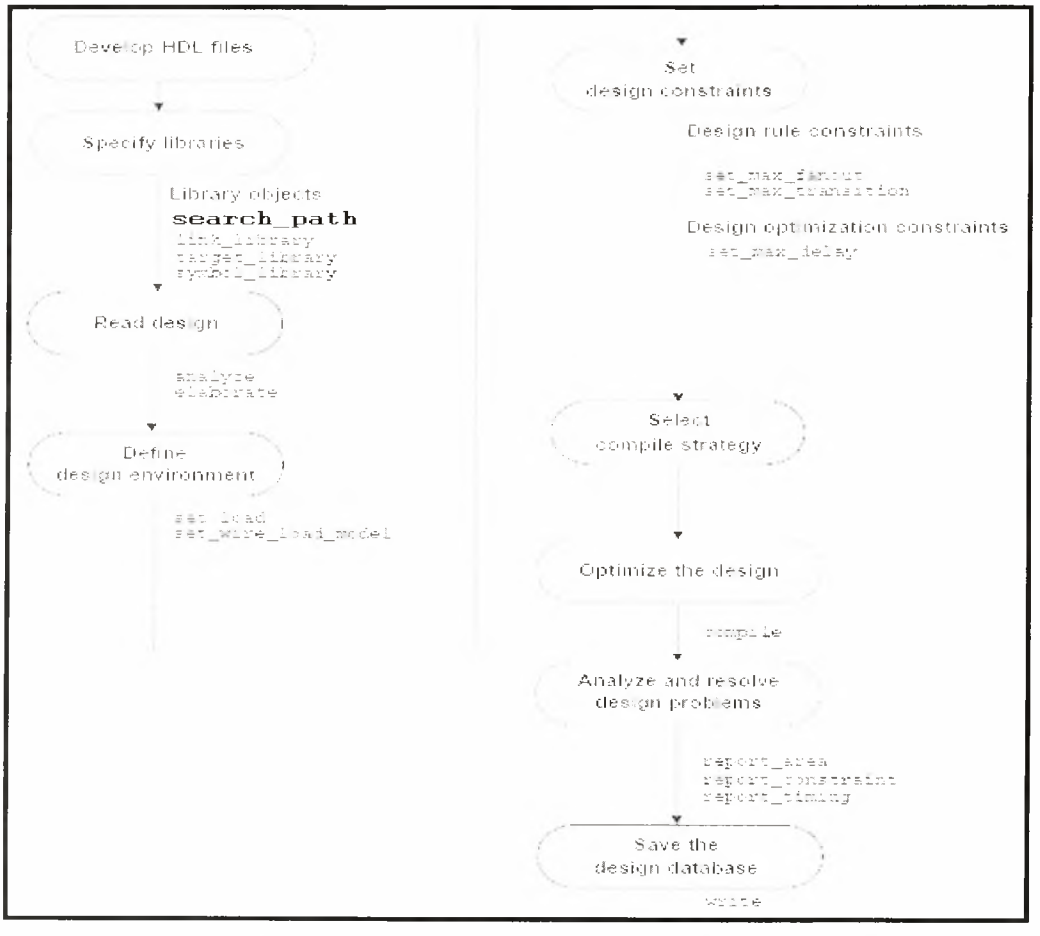
- report_cell >> ./ *adder_name* /Reports/ReportCell.txt

- report_constraint -significant_digits 2 -verbose
  >> ./ *adder_name* /Reports/ReportConstraints.txt

- report_design >> ./ *adder_name* /Reports/ReportDesign.txt
- report_area >> ./ *adder_name* /Reports/ReportArea.txt
- report_timing -from all_inputs() -to all_outputs() >> ./ *adder_name* /Reports/ReportTiming.txt

---

Συμπερασματικά, στο παρακάτω σχήμα διαφαίνονται κατά την διάρκεια της ροής σε ποια κατηγορία ανήκει η κάθε εντολή και για ποιο λόγο χρησιμοποιείται στην επιλεγμένη αλληλουχία και χρονική στιγμή.



ΣΧΗΜΑ 4.5 : Σχηματική αναπαράσταση των χρησιμοποιούμενων εντολών

4.5) DESIGNWARE LIBRARIES :

Οι **DesignWare Libraries** αποτελούν βιβλιοθήκες που αναπτύσσονται από την ίδια την **Synopsys**. Η χρησιμότητα αυτών των βιβλιοθηκών διαφαίνεται στο γεγονός πως παρέχουν στο σχεδιαστή κάποια **components** (δομικά στοιχεία), τα οποία είναι αφηρημένα (**abstract**). Είναι, δηλαδή, ανεξάρτητα τεχνολογικά και αποτελούν στην ουσία δομικά μπλοκ σε επίπεδο μικροαρχιτεκτονικής τα οποία είναι πλήρως ενοποιημένα με το περιβάλλον της σύνθεσης του πακέτου εργαλείων της εταιρίας που τα έχει συντάξει και ενσωματώσει μέσα στις πλατφόρμες τις. Πιο συγκεκριμένα η **Synopsys**, παρέχει τις ακόλουθες δυο σειρές **DesignWare** βιβλιοθηκών :

- **Foundation Library.**
- **Digital Signal Processing (DSP) Library.**

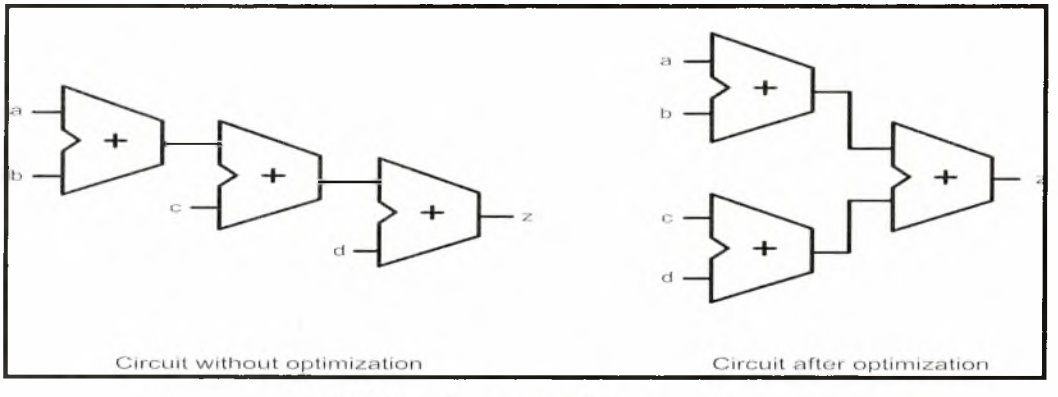
Κατά τη διαδικασία της σύνθεσης είναι πάγια σχεδιαστική τακτική η επαναχρησιμοποίηση προηγούμενων κυκλωμάτων. Το **flow** του DC χρησιμοποιεί τις **Foundation** και **GTECH** βιβλιοθήκες. Η **Foundation** βιβλιοθήκη είναι μια συλλογή από επαναχρησιμοποιήσιμα, συνθέσιμα δομικά μπλοκ που είναι πλήρως ενοποιημένα με το περιβάλλον των εργαλείων της **Synopsys** που προσφέρονται για την σύνθεση ψηφιακών κυκλωμάτων.

Αυτή η βιβλιοθήκη λόγω του χαρακτηριστικού που προαναφέρθηκε παρέχει την δυνατότητα εκτέλεσης βελτιστοποιήσεων υψηλού βαθμού μέσω της χρήσης κατάλληλων εργαλείων σύνθεσης. Ένας τρόπος βελτιστοποίησης μπορεί να διαφανεί με την χρησιμοποίηση κάποιου παραδείγματος. Το σενάριο είναι το εξής :

Λαμβάνεται η περίπτωση στην οποία μέσα σε ένα αρχείο το οποίο περιγράφει μια σχεδίαση, έχει 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**).

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)*.



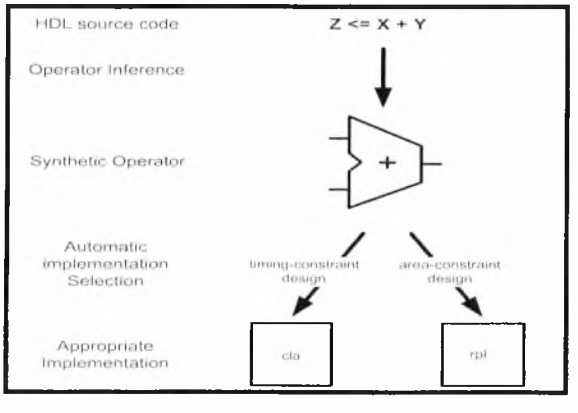
ΣΧΗΜΑ 4.6 : Αριθμητική βελτιστοποίηση

Ο νέος αυτός τρόπος «αναπαράστασης» ενδέχεται να προσφέρει μεγαλύτερη ταχύτητα λόγω της μείωσης των επιπέδων λογικής που

χρησιμοποιούνται. Ο καταμερισμός πόρων, τώρα, επιτρέπει σε 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.

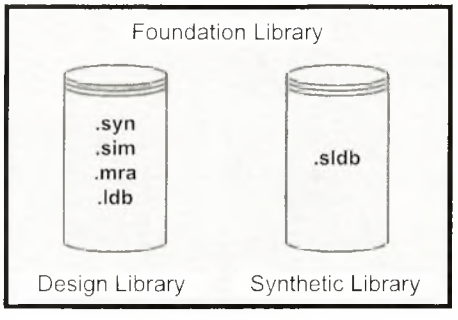
Εκτός από τη θεωρητικό υπόβαθρο μιας τέτοιας βελτιστοποίησης, στην πράξη με την χρήση της **Foundation** βιβλιοθήκης, μια πράξη που δίνεται από τον χρήστη μέσω κάποιου αρχείου εισόδου στον **DC** μπορεί να υλοποιηθεί με πολλαπλούς τρόπους.

Στο σημείο αυτό επεμβαίνει ο **Design Compiler** επιλέγοντας την καταλληλότερη υλοποίηση με βάση την σχεδίαση που έχει εισαχθεί. Ένα πρακτικό παράδειγμα της λειτουργίας που μόλις έχει περιγραφεί, το οποίο βασίζεται στην αριθμητική πράξη της πρόσθεσης είναι απαραίτητο για την περεταίρω κατανόηση του χειρισμού του **DC**. 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 προηγούμενα στάδια της ροής σχεδίασης ο χρήστης. Δηλαδή η διαδικασία επιλογής υλοποίησης θα είναι :



ΣΧΗΜΑ 4.7 : Επιλογή της κατάλληλης αρχιτεκτονικής βάσει των περιορισμών

Επειδή γίνεται εκτενέστερος λόγος για την **Foundation** βιβλιοθήκη θα πρέπει να αναλυθεί πιο διεξοδικά. Αυτό σημαίνει πως θα πρέπει να εξεταστούν τα δομικά της στοιχεία και ο τρόπος αλληλεπίδρασης τους με τον **Design Compiler**. Το σχήμα που ακολουθεί μας δίνει μια πρώτη περιγραφή :



ΣΧΗΜΑ 4.8 : Δομικά στοιχεία της Foundation Library

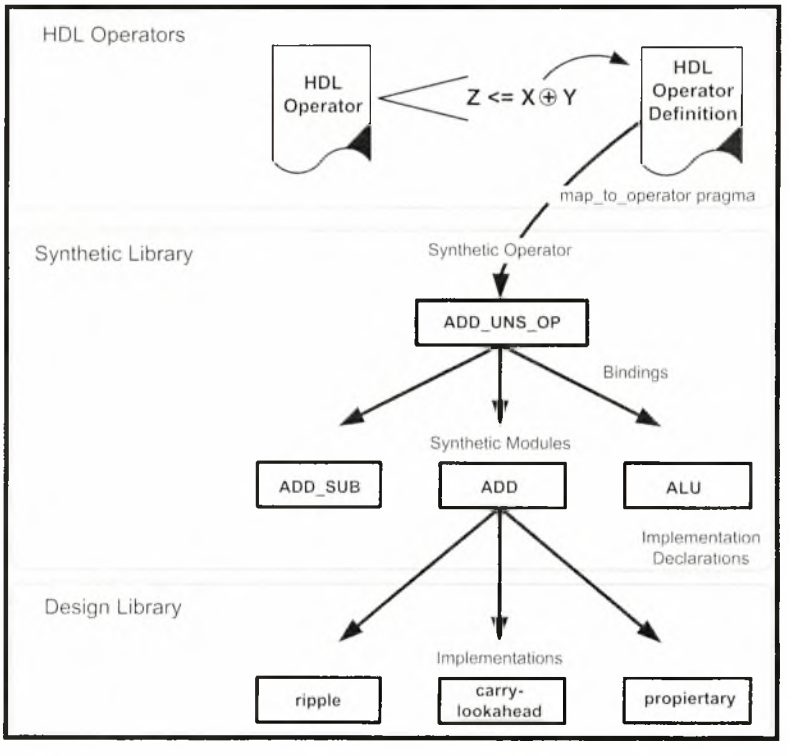
Η **Foundation Library** αποτελείται από δύο επιμέρους βιβλιοθήκες, την **Design Library** και την **Synthetic Library** :

- Η **Design Library** είναι ένας **Unix** φάκελος ο οποίος περιέχει περιγραφές κυκλωμάτων για διάφορες αρχιτεκτονικές. Τα αρχεία τα οποία περιέχει είναι **.syn**, **.sim**, **.mra**, **.ldb format**. 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** από την άλλη μεριά είναι ένα δυαδικό αρχείο (με κατάληξη **.sldb**) το οποίο συνδέει την σχεδίαση με μια **Design Library** στο εργαλείο σύνθεσης. Η **Synthetic Library** περιέχει πληροφορία η οποία επιτρέπει στα εργαλεία σύνθεσης να εκτελέσουν βελτιστοποιήσεις υψηλού επιπέδου, όπως αυτές που αναφέρθηκαν

παραπάνω, συμπεριλαμβανομένου και της επιλογής υλοποίησης. Συνεπώς σε αυτό το σημείο δημιουργείται η εξής ερώτηση: «Πως γίνεται η υλοποίηση της σύνδεσης μεταξύ των δυο δομικών στοιχείων της **Foundation Library**;»

Η απάντηση είναι σχετικά απλή. 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 τα **synthetic modules**. Κάθε **synthetic module** μπορεί να έχει πολλαπλές υλοποιήσεις. Στο παρακάτω σχήμα παρουσιάζονται οι μεταβάσεις μέχρι την επιλογή της καταλληλότερης υλοποίησης.



ΣΧΗΜΑ 4.9 : Σύνδεση μεταξύ δομικών στοιχείων της Foundation Library

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** και **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** πράγμα 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** .

Η **Synthetic Library** περιέχει ορισμούς για τους **synthetic operators**, για τα **synthetic modules**, και **bindings**. Επίσης, περιέχει δηλώσεις που συνδέουν τις **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 πολύ κοντά στις έννοιες του **entity** και **architecture** της **VHDL**. Ένα **implementation** μπορούμε να το δούμε σαν μια αρχιτεκτονική πραγματοποίηση μιας **synthetic module**. Ένα **implementation** μπορεί να είναι οτιδήποτε από μια **netlist** συγκεκριμένης τεχνολογίας έως και ένα **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 είναι βασισμένη στα διαθέσιμα από την τεχνολογική βιβλιοθήκη δομικά μπλοκ. Στην παρούσα εργασία η τεχνολογική βιβλιοθήκη που χρησιμοποιήθηκε είναι της εταιρίας **UMC** και πιο συγκεκριμένα έγιναν μετρήσεις στην τεχνολογία **13um**. Παρακάτω υπάρχουν τα βασικά χαρακτηριστικά της τεχνολογίας βιβλιοθηκών της **UMC**.

Η τεχνολογική βιβλιοθήκη που χρησιμοποιήθηκε είναι η :

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 τέτοιου τύπου χρησιμοποιούν ένα νέο, βελτιστοποιημένο για σύνθεση σύνολο από διαθέσιμα κελιά, και έχουν επίσης μια βελτιστοποιημένη 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** που χρησιμοποιήθηκε παρέχει στα εργαλεία σύνθεσης την δυνατότητα να υλοποιούν σχεδιάσεις με πυκνότητα πυλών μέχρι και **200K gates/mm²** 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. Κάθε τεχνολογία έχει και κάποιες προτεινόμενες συνθήκες λειτουργίας για τα ολοκληρωμένα κυκλώματα που περιέχει. Οι βιβλιοθήκες της εταιρίας **UMC** έχουν τα παρακάτω χαρακτηριστικά:

UMCL13U210T3			
Operating Condition	*Minimum*	*Typical*	*Maximum*
Power Supply	*1.08V*	*1.20V*	*1.32V*
Junction Temperature	*0°C*	*25°C*	*125°C*

ΣΧΗΜΑ 4.9 : Χαρακτηριστικά της χρησιμοποιούμενης βιβλιοθήκης

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

5.ΑΝΑΛΥΣΗ ΧΡΟΝΙΣΜΟΥ

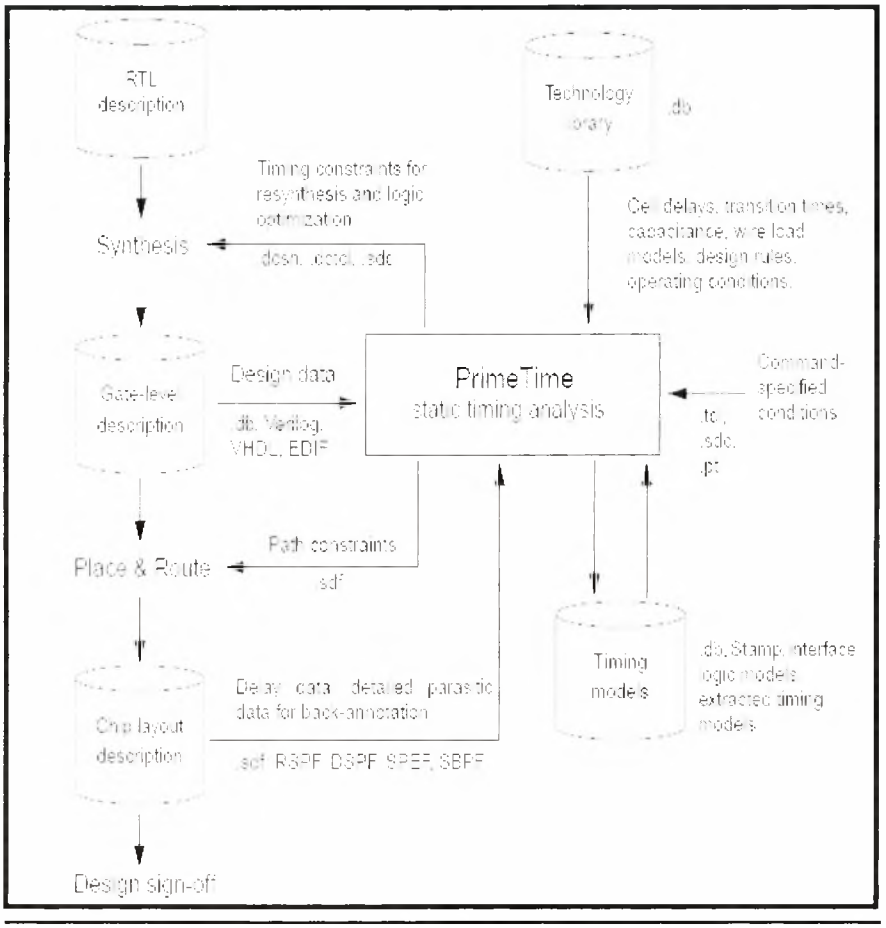
5.1) ΕΙΣΑΓΩΓΗ :

Η ανάλυση χρονισμού θα γίνει με την χρήση του βιομηχανικού εργαλείου της εταιρίας **Synopsys** με την ονομασία **PrimeTime** .Το **PrimeTime** είναι ένα εργαλείο στατικής ανάλυσης χρονισμού σε επίπεδο πυλών το οποίο αποτελεί ένα βασικό κομμάτι της σχεδιαση μεγάλου μεγέθους τρανζίστορ. Το **PrimeTime** εκτιμά την απόδοση χρονισμού μιας σχεδίασης ελέγχοντας όλα τα πιθανά μονοπάτια για παραβιάσεις χρονισμού χωρίς να χρησιμοποιεί λογικούς προσομοιωτές η **test vectors**.

5.2)ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ PRIME TIME :

Το **PrimeTime** είναι ένα εργαλείο στατικής ανάλυσης χρονισμού, το οποίο εστιάζεται σε πολύπλοκες πολλών εκατομμυρίων πυλών σχεδιάσεις. Προσφέρει ένα πραγματικά αξιόλογο συνδυασμό ταχύτητας, χωρητικότητας, ευκολίας στη χρήση και συμβατότητα με τα βιομηχανικά **standards** σε ότι αφορά τα **format** δεδομένων και ροών εργασίας (**workflows**). Το **PrimeTime** ταιριάζει ιδανικά στο μοντέλο της φυσικής ροής σύνθεσης της **Synopsys** (**Synopsys physical synthesis flow**). Αυτό συμβαίνει διότι γίνεται χρήση πολλών ίδιων βιβλιοθηκών, δομών δεδομένων και εντολών με τα υπόλοιπα εργαλεία της **Synopsys**, όπως ο **Design Compiler**. Επιπροσθέτως, μπορεί να χειριστεί σαν ανεξάρτητος και αναλυτής στατικού χρονισμού σε άλλες σχεδιαστικές ροές.

Οι σχεδιαστικές πληροφορίες που είναι δυνατόν να εισαχθούν στο εργαλείο της **Synopsys** μπορούν να απεικονιστούν με ποικίλες, αποδεκτές από το **PrimeTime**, μορφές. Τα βιομηχανικά **standard** εμπεριέχουν **gate-level netlists** σε **.db, Verilog, VHDL** και **EDIF** μορφές. Όποιες πληροφορίες σχετίζονται με καθυστέρηση απεικονίζονται και εισάγονται σε **SDF format** ενώ τα παρασιτικά σε **RSPF, DSPF, SPEF, SBPF formats** και οι χρονικοί περιορισμοί σε **Synopsys Design Constraints (SDC) format**. Ο τρόπος αλληλεπίδρασης διαφαίνεται καθαρότερα στο σχήμα που ακολουθεί και αναλύεται περεταίρω εν συνεχεία.



ΣΧΗΜΑ 5.1: Αλληλεπίδραση του PrimeTime στην συνολική διαδικασία της σύνθεσης

Όπως γίνεται εύκολα αντιληπτό από την παραπάνω εικόνα, η ανάλυση μίας τυπικής σχεδιαστικής ροής έχει κάποια απλά και κατανοητά βήματα. Ξεκινώντας από μια περιγραφή κάποιου ψηφιακού κυκλώματος σε **RTL** επίπεδο (**register transfer level**), ένα εργαλείο σύνθεσης όπως ο **Design Compiler** παράγει μια περιγραφή σχεδίασης σε επίπεδο πυλών. Το **PrimeTime** διαβάζει αυτή την περιγραφή και πιστοποιεί τον χρονισμό κάνοντας χρήση των πληροφοριών που παρέχονται από την τεχνολογική βιβλιοθήκη που χρησιμοποιείται. Εάν διαπιστωθούν από το **PrimeTime** παραβιάσεις χρονισμού η σχεδίαση θα πρέπει να επανασχεδιαστεί με νέα δεδομένα και **timing constraints** (χρονικούς περιορισμούς, οι οποίοι παράγονται από το ίδιο το **PrimeTime**) τα οποία θα επιδιορθώσουν τις καταστάσεις και τις συνθήκες που προκάλεσαν το λάθος για τον χρονισμό.

Όταν μια σχεδίαση σε επίπεδο πυλών είναι ανεξάρτητη και ελεύθερη από χρονικές παραβιάσεις, ο σχεδιαστής μπορεί να προβεί στο κομμάτι του **placement & routing**, δηλαδή την μεταφορά του ψηφιακού κυκλώματος σε φυσικό επίπεδο. Αυτό παράγει μια βάση δεδομένων **chip** φυσικού επιπέδου από την οποία είναι δυνατόν να υπολογιστεί όλη η πληροφορία σε σχέση με την καθυστέρηση και με τα παρασιτικά. Σε αυτό ακριβώς το σημείο υπάρχει η δυνατότητα της ανατροφοδότησης των εξαγόμενων και μη δεδομένων τα οποία επιτρέπουν στο **PrimeTime** να πραγματοποιήσει μια πιο ακριβή και ρεαλιστική ανάλυση χρονισμού για το ίδιο αρχικό κύκλωμα. Ένας επιτυχής προσδιορισμός του χρονισμού του κυκλώματος σε αυτό ακριβώς το σημείο οδηγεί σε επιτυχή ολοκλήρωση της σχεδίασης.

Σημείωση: το διάγραμμα που έχει πρότερα αναλυθεί υποδεικνύει μόνο τα στάδια που σχετίζονται με τον χρονισμό και την ανάλυση του στη ροή της φυσικής σύνθεσης (**physical synthesis flow**). Δεν περιλαμβάνει μη συσχετιζόμενα με το χρονισμό βήματα όπως το **formal verification**, την ανίχνευση της σύνθεσης (**scan synthesis**) και την λογική προσομοίωση (**logic simulation**).

Τύποι ελέγχου

Το **PrimeTime** διενεργεί τους ακολούθους τύπους ελέγχου σχεδίασης:

- **Setup**, αναμονής ανάκτησης και αφαίρεσης
- **User-specified data-to-data** περιορισμοί χρονισμού
- **Clock-gating setup** και **hold** περιορισμοί.
- Ελάχιστης περιόδου και ελάχιστου πλάτος παλμού για ρολόγια
- Σχεδιαστικοί κανόνες (ελάχιστου/μέγιστου χρόνου μετάβασης, χωρητικότητας, και **fanout**)

Χαρακτηριστικά ανάλυσης χρονισμού

Το **PrimeTime** υποστηρίζει μια πληθώρα από εξειδικευμένα και πολύπλοκα χαρακτηριστικά και δυνατότητες ανάλυσης χρονισμού , με σημαντικότερα τα ακόλουθα :

- Πολλαπλά ρολόγια και συχνότητες
- Χρονικές εξαιρέσεις για **multicycle path**
- **False path** εξαιρέσεις χρονισμού και αυτόματος εντοπισμός τους
- **Transparent latch** ανάλυση και **time borrowing**
- Ταυτόχρονη ελάχιστη / μέγιστη ανάλυση καθυστέρησης για **setup** και **hold** περιορισμούς.
- Ανάλυση με **on-chip variation** διεργασίας, **voltage** και συνθήκες θερμοκρασίας (**PVT**).
- **Case** ανάλυση (ανάλυση με περιορισμούς ή ειδικά **transitions** τα οποία εφαρμόζονται σε συγκεκριμένες εισόδους).
- **Mode** ανάλυσης (ανάλυση με **module-specific** μοντέλα λειτουργίας, όπως **mode** ανάγνωσης ή **mode** εγγραφής για **RAM module**).
- **Bottleneck** ανάλυση (παροχή αναφοράς των **cell** που προκαλούν τις περισσότερες χρονικές παραβιάσεις).
- **ECO** ανάλυση χωρίς μετατροπή του πρωτοτύπου **netlist**, κάνοντας χρήση εσωτερικά **buffers**, ανασχεδιασμένων κελιών και τροποποιημένων **nets**
- Ανάλυση των **crosstalk** φαινομένων μεταξύ φυσικών **nets** χρησιμοποιώντας την **PrimeTime *SI* (signal integrity)** επιλογή.

Χρονικά μοντέλα

Το **PrimeTime** υποστηρίζει την χρήση χρονικών μοντέλων ώστε να εκπροσωπήσει τα **chip submodules**. Ένα χρονικό μοντέλο περιέχει πληροφορία σχετικά με τα χαρακτηριστικά χρονισμού, αλλά όχι με την λογική λειτουργία ενός **submodule**. Το **PrimeTime** μπορεί να δημιουργήσει ένα χρονικό μοντέλο από ένα **submodule netlist** και έπειτα να χρησιμοποιήσει αυτό το μοντέλο στη θέση του πραγματικού **netlist** για ανάλυση χρονισμού σε υψηλοτέρα επίπεδα της ιεραρχίας. Αυτή η τεχνική επιταχύνει την ανάλυση.

Μια άλλη χρήση των χρονικών μοντέλων είναι στο να προστατέψει την πνευματική ιδιοκτησία. Σε περίπτωση που γίνει η παροχή από κάποιον σχεδιαστή κάποιας υπομονάδας (**submodule**) σε κάποιον πελάτη ώστε να γίνει μέρος κάποιου μεγαλύτερου **chip** του πελάτη υπάρχει η δυνατότητα αντικατάστασης του πρωτοτύπου **netlist** από το αντίστοιχο χρονικό μοντέλο. Αυτό επιτρέπει στον πελάτη να διενεργήσει ακριβή ανάλυση χρονισμού στο κύκλωμα του χωρίς στην ουσία να έχει πρόσβαση στο υπό ανάπτυξη **netlist**.

Τα χρονικά μοντέλα που υποστηρίζονται από το **PrimeTime** είναι:

- *Quick timing*. Δημιουργείται στο **PrimeTime** κάνοντας χρήση μίας αλληλουχίας εντολών . Αυτός ο τύπος είναι χρήσιμος στα πρώτα στάδια της σχεδίασης , όταν ένα **netlist** δεν είναι ακόμα διαθέσιμο για ένα **submodule**.

- *Extracted model*. Είναι ένα μοντέλο που εμπεριέχει μόνο χρονικές πληροφορίες. Αυτός ο τύπος δεν λαμβάνει υπ' όψιν όλη τη λογική του πρωτοτύπου και το αντικαθιστά με ένα σετ από χρονικά τόξα (**arcs**) μεταξύ ρολογιών, εισόδων και εξόδων

- *Interface logic.* Είναι ένα δομικό μοντέλο χρονισμού που βγαίνει από το **PrimeTime** από ένα **netlist** επιπέδου πυλών. Αυτός ο τύπος μοντέλου διατηρεί το **interface** της λογικής του πρωτοτύπου netlist και αναιρεί την εσωτερική λογική **register-to-register** η οποία έχει ήδη επιβεβαιωθεί στο επίπεδο του module

- *Stamp model.* Σε αυτήν την περίπτωση το χρονικό μοντέλο καθορίζεται σε περιγραφική γλώσσα, είτε γραμμένη στο χέρι είτε μεταφρασμένη από μια άλλη περιγραφή χρονισμού.

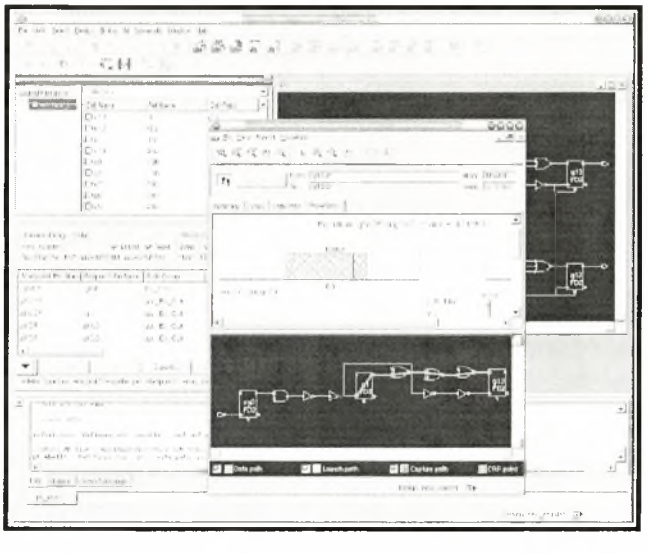
Interface εντολών

Το **PrimeTime** προσφέρει δύο περιβάλλοντα εντολών για ανάλυση χρονισμού.

Α) Το κέλυφος της πλατφόρμας, **pt_shell.** Το κέλυφος των εντολών είναι ένα **command-line** το οποίο υποστηρίζει επιπρόσθετα και την δυνατότητα εκτέλεσης **script** γραμμένων σε **Tcl**. Υποστηρίζει μόνο **text** περιβάλλον στο οποίο εισάγονται εντολές σε ένα **prompt** (**pt_shell>**) και οι αλληλεπίδραση των αποτελεσμάτων – δεδομένων γίνεται υπό την μορφή κειμένου. Το **pt_shell interface** βασίζεται στην **Tcl scripting** γλώσσα, το οποίο σημαίνει πως υποστηρίζονται χαρακτηριστικά όπως διεργασίες, λίστες, και λειτουργίες επεξεργασίας πινάκων.

ΣΧΗΜΑ 5.2: Εικόνα του τερματικού παραθύρου του PrimeTime

Β)Το γραφικό περιβάλλον (**GUI**), είναι βασισμένο σε παράθυρα και κατάλληλο για **interactive** ανάλυση και για γραφική απεικόνιση των δεδομένων και των αποτελεσμάτων. Το **GUI** προσφέρει κάποιες δυνατότητες οπτικής ανάλυσης οι οποίες δεν παρέχονται στο **pt_shell**. Για παράδειγμα, υπάρχει η δυνατότητα προβολής των **schematics** (σχηματικών) της σχεδίασης, των κυματομορφών των ρολογιών και η δημιουργία ιστογραμμάτων και γραφικών από τα αποτελέσματα της ανάλυσης χρονισμού όπως **path slack**, χωρητικότητα δικτύου και κόστος **bottleneck**. Το παράθυρο της κονσόλας μέσα στο ανωτέρου επιπέδου παράθυρο επιτρέπει την εισαγωγή εντολών , όπως ακριβώς και το **pt_shell**.



ΣΧΗΜΑ 5.3: Παρουσίαση στιγμιοτύπου του γραφικού περιβάλλοντος του PrimeTime

Στατική ανάλυση χρονισμού

Η στατική ανάλυση χρονισμού είναι μια μέθοδος επαλήθευσης της απόδοσης του χρονισμού μιας σχεδίασης ελέγχοντας όλες τις πιθανές χρονικές παραβάσεις σε όλους τους συνδυασμούς μονοπατιών. Το **PrimeTime** ελέγχει για παραβάσεις με τον ίδιο τρόπο με τον οποίο κάποιος σχεδιαστής θα το έκανε χειροκίνητα, αλλά με μεγαλύτερη ταχύτητα και ακρίβεια όπως είναι φυσικό. Για να πραγματοποιηθεί κάποιος έλεγχος για παραβάσεις το **PrimeTime** αναλύει σε κομμάτια μονοπατιών χρονισμού την σχεδίαση, υπολογίζει την καθυστέρηση διάδοσης του σήματος κατά μήκος του κάθε μονοπατιού και ελέγχει για παραβιάσεις των περιορισμών χρονισμού μέσα στην σχεδίαση και στο **interface** εισόδου / εξόδου.

Ένας άλλος τρόπος για να πραγματοποιηθεί μια ανάλυση χρονισμού είναι να γίνει χρήση δυναμικής προσομοίωσης, η οποία καθορίζει την πλήρη συμπεριφορά του κυκλώματος για ένα σετ εισαγόμενων στιγμιαίων πινάκων. Σε σύγκριση βέβαια με την στατική ανάλυση είναι πιο αργή γιατί χρειάζεται να προσομοιωθεί η λογική συμπεριφορά του ψηφιακού κυκλώματος. Είναι επίσης πιο χρονοβόρα διότι ελέγχει όλα τα μονοπάτια χρονισμού και όχι μόνο τις λογικές συνθήκες που ενεργοποιούνται από κάποιο σετ ενός πίνακα εισόδου δοκιμής. Από την άλλη πλευρά η στατική ανάλυση χρονισμού μπορεί μόνο να δοκιμάσει – ελέγξει το χρονισμό και όχι τη λειτουργικότητα μιας σχεδίασης.

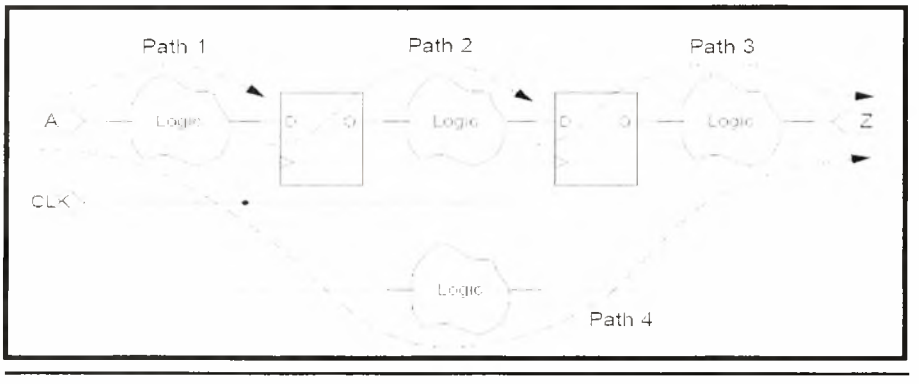
Timing Paths (Μονοπάτια χρονισμού)

Το πρώτο βήμα το οποίο πραγματοποιείται από το **PrimeTime** για ανάλυση χρονισμού είναι η τμηματοποίηση της σχεδίασης σε επιμέρους ομάδες μονοπατιών χρονισμού ή χρονικά μονοπάτια (**Timing Paths**). Κάθε

μονοπάτι έχει ένα σημείο αρχής και τέλους. Το σημείο αρχής είναι ένα μέρος το οποίο τα δεδομένα φθάνουν με την άκρη του κάθε παλμού του ρολογιού (**clock edge**). Τα δεδομένα προωθούνται διαμέσου συνδυαστικής λογικής στο μονοπάτι και στη συνέχεια λαμβάνονται και αιχμαλωτίζονται στο σημείο τέλους από την ακμή κάποιας άλλης ακμής παλμού ρολογιού.

Το σημείο αρχής (**startpoint**) ενός μονοπατιού είναι μια άκρη (**pin**) ρολογιού από ένα ακολουθιακό στοιχείο ή πιθανότατα ένα **port** εισόδου της σχεδίασης. (εξ' αιτίας του γεγονότος πως τα δεδομένα εισόδου μπορούν να εισαχθούν από κάποια εξωτερική πηγή). Το σημείο λήξης (**endpoint**) ενός μονοπατιού μπορεί να είναι μια άκρη δεδομένων εισόδου ενός ακολουθιακού στοιχείου ή πιθανότατα ένα **port** εξόδου της σχεδίασης (εξ' αιτίας του γεγονότος πως τα δεδομένα εξόδου μπορούν να αιχμαλωτιστούν από κάποια εξωτερική καταβόθρα).

Σχηματικά η αναπαράσταση των παραπάνω διαφαίνεται παρακάτω και αναλύεται στη συνέχεια:



ΣΧΗΜΑ 5.4: Σχηματισμός και ανάλυση μονοπατιού στο PrimeTime

Κάθε «λογικό συννεφάκι», στο παραπάνω σχήμα αντιπροσωπεύει ένα δίκτυο συνδυαστικής λογικής. Κάθε μονοπάτι ξεκινά σε ένα σημείο εισαγωγής δεδομένων , διέρχεται διαμέσου κάποιας συνδυαστικής λογικής και τελειώνει

σε κάποιο σημείο αιχμαλωσίας δεδομένων. Οι περιπτώσεις που μπορεί να καταγραφούν είναι οι ακόλουθες :

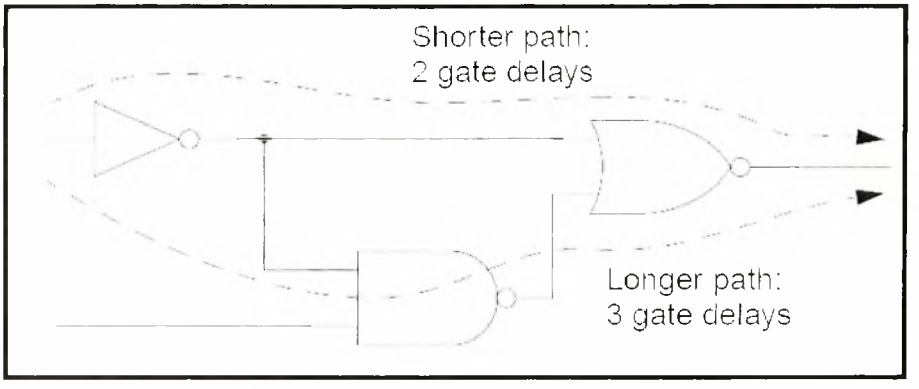
- Το 1ο είδος μονοπατιού ξεκινάει από ένα **port** εισόδου και τελειώνει στο σημείο εισόδου δεδομένων κάποιου ακολουθιακού στοιχείου.

- Το 2ο είδος μονοπατιού ξεκινάει από ένα **pin** ρολογιού κάποιου ακολουθιακού στοιχείου και τελειώνει στο σημείο εισόδου δεδομένων κάποιου ακολουθιακού στοιχειού.

- Το 3ο είδος μονοπατιού ξεκινάει από ένα **pin** ρολογιού κάποιου ακολουθιακού στοιχείου και τελειώνει στο σημείο εξόδου δεδομένων.

- Το 4ο είδος μονοπατιού ξεκινάει από ένα σημείο εισόδου δεδομένων και τελειώνει στο σημείο εξόδου δεδομένων.

Ένα συνδυαστικό «λογικό συννεφάκι» επίσης θα μπορούσε να περιέχει πολλαπλά μονοπάτια όπως διαφαίνεται παρακάτω. Το **PrimeTime** χρησιμοποιεί για τους υπολογισμούς και την ευρύτερη ανάλυση χρονισμού του το μεγαλύτερου μήκους μονοπάτι για να υπολογίσει την μέγιστη καθυστέρηση ή εναλλακτικά το ελάχιστο μονοπάτι για την ελάχιστη καθυστέρηση.



ΣΧΗΜΑ 5.5: Ανάλυση πολλαπλών μονοπατιών στο PrimeTime

Υπολογισμός καθυστέρησης

Μετά το πρώτο βήμα το οποίο πραγματοποιείται από το **PrimeTime** για ανάλυση χρονισμού, την τμηματοποίηση δηλαδή της σχεδίασης σε επιμέρους ομάδες μονοπατιών χρονισμού ή χρονικά μονοπάτια (**Timing Paths**), διεξάγεται ο υπολογισμός της καθυστέρησης κατά μήκος κάθε μονοπατιού. Η συνολική καθυστέρηση του μονοπατιού είναι το αλγεβρικό άθροισμα όλων των επιμέρους καθυστερήσεων όλων των κελιών και των δικτύων στο μονοπάτι. Η μέθοδος του υπολογισμού της καθυστέρησης εξαρτάται από την ύπαρξη του **chip** σε φυσικό επίπεδο. Πριν το **layout**, το **chip**, τυπολογικά είναι άγνωστο, έτσι το εργαλείο ανάλυσης χρονισμού επιφορτίζεται με τον υπολογισμό των καθυστερήσεων των δικτύων λαμβάνοντας υπ' όψιν και κάνοντας χρήση μοντέλων φορτίων καλωδίων (**wire load models**).

Μετά την διεκπεραίωση και μεταφορά σε φυσικό επίπεδο, ένα εξωτερικό εργαλείο μπορεί να καθορίσει με αρκετά ικανοποιητική ακρίβεια τις καθυστερήσεις και να τις καταγράψει σε ένα αρχείο Καθορισμένης Μορφής Καθυστέρησης (**Standard Delay Format - SDF**). Το **PrimeTime** είναι ικανό να διαβάσει το **SDF** αρχείο και να ανατροφοδοτήσει την σχεδίαση με τη απαραίτητη πληροφορία καθυστέρησης για μια πιο ακριβή ανάλυση χρονισμού βασισμένη στο φυσικό επίπεδο πλέον. Επιπρόσθετα παρέχεται η

δυνατότητα αποδοχής μιας λεπτομερέστατης περιγραφής παρασιτικών χωρητικοτήτων και αντιστάσεων στο δίκτυο διασύνδεσης και κατά συνέπεια ο υπολογισμός της καθυστέρησης δικτύου βασισμένης σε αυτήν την πληροφορία θα είναι πιο ακριβής.

Καθυστέρηση κελιών (Cell Delay)

Η καθυστέρηση κελιών είναι το ποσό της καθυστέρησης από την είσοδο έως την έξοδο μιας λογικής πύλης σε ένα μονοπάτι. Στην απουσία κάποιου είδους ανατροφοδότησης πληροφορίας καθυστέρησης από **SDF** αρχείο, το **PrimeTime** υπολογίζει την καθυστέρηση κελιού από πίνακες καθυστέρησης οι οποίοι παρέχονται από τον κατασκευαστή της τεχνολογικής βιβλιοθήκης που χρησιμοποιείται.

Τυπικά, ένας πίνακας καθυστέρησης περιέχει το ποσό της καθυστέρησης σαν μια συνάρτηση μιας ή περισσοτέρων μεταβλητών , όπως για παραδείγματος χρόνου **transition** και χωρητικότητας φορτίου εξόδου. Βασισμένο σε στοιχεία τέτοιων πινάκων , το εργαλείο ανάλυσης χρονισμού υπολογίζει κάθε καθυστέρηση κελιού. Όταν κριθεί απαραίτητο γίνεται χρήση παρεμβολής στις τιμές των πινάκων ώστε να διατηρηθεί μια τιμή καθυστέρησης για κάποιες συγκεκριμένες συνθήκες ορισμένες από την ίδια την σχεδίαση.

Καθυστέρηση δικτύου (Net Delay)

Η καθυστέρηση δικτύου είναι το ποσό της καθυστέρησης από την έξοδο ενός κελιού στην είσοδο κάποιου άλλου κελιού κατά μήκος του μονοπατιού που εξετάζεται. Αυτή η καθυστέρηση υφίσταται λόγω της παρασιτικής χωρητικότητας των διασυνδέσεων μεταξύ των δυο κελιών, γεγονός το οποίο είναι άμεσα συνδεδεμένο με την αντίσταση του δικτύου και την περιορισμένη δύναμη οδήγησης του κελιού που οδηγεί το δίκτυο.

Ο υπολογισμός αυτής της καθυστέρηση διεξάγεται με τους ακόλουθους τρόπους:

- Κάνοντας χρήση ειδικών τιμών χρονισμού από **SDF** αρχείο
- Κάνοντας χρήση λεπτομερών δεδομένων παρασιτικής αντίστασης και χωρητικότητας από **RSPF**, **DSPF**, **SPEF**, ή **SBPF** τύπους αρχείων.
- Εκτιμώντας καθυστερήσεις από μοντέλα φορτίων καλωδίων (**wire load models**).

Ένα μοντέλο φορτίων καλωδίων προσπαθεί να προβλέψει την χωρητικότητα και την αντίσταση δικτύων χωρίς την ύπαρξη ανατροφοδοτούμενων δεδομένων καθυστέρησης ή παρασιτικής πληροφορίας. Η τεχνολογική βιβλιοθήκη παρέχει στατιστικά μοντέλα φορτίων καλωδίων για τον υπολογισμό δεδομένων παρασιτικής αντίστασης και χωρητικότητας βασισμένων στον αριθμό των **fanout pins** σε κάθε **net**. Χρησιμοποιώντας μοντέλα φορτίων καλωδίων μειώνεται η ακρίβεια υπολογισμού καθυστέρησης, διότι αναφερόμαστε σε εκτιμήσεις μεγεθών και όχι σε πραγματικές μετρήσεις που απορρέουν από την ανατροφοδότηση

καθυστερήσεων και παρασιτικών δεδομένων . όμως η συγκεκριμένη μέθοδοι είναι η μόνη διαθέσιμη εν απουσία φυσικής υπόσταση της σχεδίασης.

Ροή ανάλυσης χρονισμού με το PrimeTime

Για την επιτυχή ολοκλήρωση μιας ανάλυσης χρονισμού σε μια σχεδίαση , είναι απαραίτητο να πραγματοποιηθούν κάποια στάδια με την κατάλληλη σειρά. Ένα τυπικό μοντέλο μιας τέτοιας ανάλυσης ηχεί ως εξής:

1. Ανάγνωση των δεδομένων της σχεδίασης (ένα **netlist** επιπέδου πυλών και εξαρτημένες τεχνολογικές βιβλιοθήκες).
2. Περιορισμός της σχεδίασης προσδιορίζοντας τα χαρακτηριστικά του ρολογιού, εισαγομένων συνθηκών χρονισμού και εξαγόμενων χρονικών απαιτήσεων.

3. Προσδιορισμός του περιβάλλοντος και των συνθηκών ανάλυσης όπως συνθήκες λειτουργίας, παραμετροποίηση της ανάλυσης κατά περίπτωση , μοντέλα καθυστέρησης δικτύων και εξαιρέσεις χρονισμού.

4. Έλεγχος των δεδομένων σχεδίασης και των παραμέτρων κατά την προετοιμασία της πλήρους ανάλυσης χρονισμού.

5. Διενέργεια πλήρους ανάλυσης χρονισμού και μελέτη των αποτελεσμάτων.

Η ακόλουθη παράγραφος παρέχει μια περιγραφή των βασικών βημάτων ανάλυσης και εντολών που χρησιμοποιούνται σε κάθε στάδιο, σύμφωνα πάντα και με το **script** που έχουμε κατασκευάσει.

Ανάγνωση δεδομένων σχεδίασης

Το πρώτο και σημαντικότερο βήμα είναι να διεκπεραιωθεί η ανάγνωση και φόρτωση της περιγραφής σε επίπεδο πυλών της σχεδίασης με τις τεχνολογικά εξαρτημένες βιβλιοθήκες που έχουν χρησιμοποιηθεί. Το **PrimeTime** δέχεται περιγραφές σχεδιάσεων και πληροφορίες βιβλιοθηκών σε **.db** (**Synopsys** βάση δεδομένων) μορφή, **.ddc** (**Design Compiler** βάση δεδομένων) και επιπέδου πυλών **netlists** σε **Verilog, VHDL**, και **EDIF** μορφές. Οι κατάλληλες εντολές για την παραπάνω ενέργεια είναι *read_db, read_ddc, read_verilog, read_vhdl* και *read_edif.*

Μετά το πέρας της ανάγνωσης του σετ των αρχείων της ιεραρχικής σχεδίασης η εντολή *link_design* λύνει όλες τις διαφορές και αναφορές μεταξύ διαφορετικών modules στην ιεραρχία χτίζοντας μια εσωτερική εκπροσώπηση της σχεδίασης για την ανάλυσης χρονισμού.

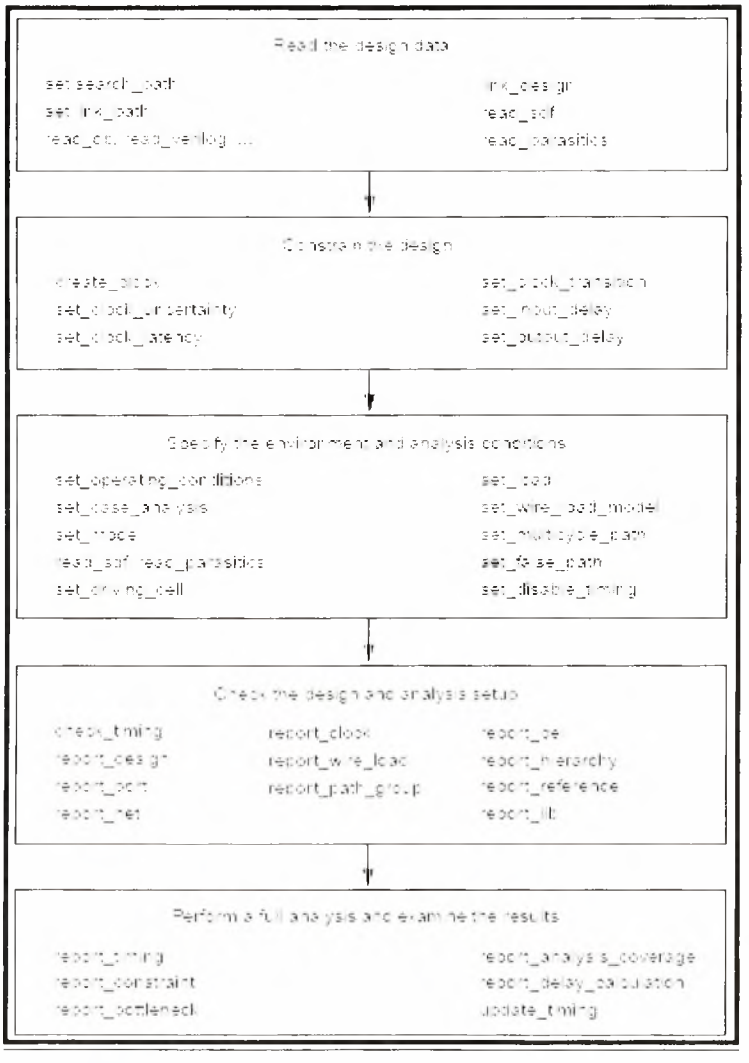
Η *search_path* εξειδικεύει μια λίστα από **directories** στα οποία θα γίνει η αναζήτηση της ανάγνωσης της σχεδίασης και των καταλλήλων συνοδευτικών βιβλιοθηκών

Παραμετροποιώντας την σχεδίαση

Για την ομαλή και ορθή διεξαγωγή της ανάλυσης το **PrimeTime** χρειάζεται πληροφορίες για τις παραμέτρους χρονισμού σε επίπεδο σχεδίασης στα εξής σημεία:

- *Χαρακτηριστικά ρολογιού*
- *Χρόνοι άφιξης και μεταγωγή σήματος σε κάθε είσοδο*

- *Απαιτούμενοι χρόνοι μεταγωγής σήματος σε κάθε έξοδο*



ΣΧΗΜΑ 5.6: Συνοπτική παρουσίαση των βασικών εντολών του PrimeTime

Εξειδικεύοντας το περιβάλλον και τις συνθήκες ανάλυσης

Το **PrimeTime** επιτρέπει την εξιδανίκευση του περιβάλλοντος και των συνθηκών ανάλυσης χρονισμού. Οι τομείς που μπορούν να παραμετροποιηθούν είναι:

- Προσδιορισμός της διαδικασίας, θερμοκρασίας και **voltage** λειτουργίας όπως περιγράφονται στις τεχνολογικές βιβλιοθήκες.
- Προσδιορισμός ανάλυσης περίπτωσης και **mode** ανάλυσης ώστε να περιοριστεί – απαγορευτεί **mode** λειτουργίας της συσκευής κατά τη διάρκεια της ανάλυσης.
- Προσδιορισμός κελιών οδήγησης σε εισόδους και φορτία σε εξόδους.
- Προσδιορισμός χρονικών εξαιρέσεων για μονοπάτια που δεν επηρεάζουν την εξορισμού συμπεριφορά και τρόπο λειτουργίας του **PrimeTime**.
- Προσδιορισμός μοντέλων φορτίων καλωδίων (wire load models) ή αρχείων ανατροφοδότησης πληροφορίας σχετικής με τον υπολογισμό της καθυστέρησης.

Έλεγχος της σχεδίασης και διενέργεια ανάλυσης

Λίγο πριν την πραγματοποίηση της πλήρους ανάλυσης χρονισμού, είναι μια καλή πρόταση ο έλεγχος των χαρακτηριστικών της σχεδίασης όπως η ιεραρχία, τα στοιχεία της βιβλιοθήκης, τα **ports**, τα κελιά, τα δίκτυα, και

των παραμέτρων της ανάλυσης όπως τα ρολόγια, τα μοντέλα φορτίων καλωδίων , οι περιορισμοί της καθυστέρησης εισόδου και οι περιορισμοί της καθυστέρησης εξόδου.

Φαινόμενο Bottleneck

Όταν μια σχεδίαση έχει έναν μεγάλο αριθμό από χρονικές παραβιάσεις, η ανάλυση των bottlenecks μπορεί να βοηθήσει το σχεδιαστή να εντοπίσει τα μέρη στην σχεδίαση που δημιουργούν το πρόβλημα και τα οποία με κάποια αλλαγή (όπως αλλαγή του μεγέθους κάποιας πύλης ή επανασχεδιασμούς του καρκινικού σημείου υπό νέους περιορισμούς) θα μπορέσουν να συνεισφέρουν θετικά στην λειτουργία του κυκλώματος.

Ένα **bottleneck** είναι ένα σημείο στην σχεδίαση το οποίο συνεισφέρει στον πολλαπλασιασμό χρονικών παραβιάσεων. Εξ ορισμού, η εντολή **report_bottleneck** εμφανίζει τα 20 χειρότερα κελιά-φύλλα της ιεραρχικής σχεδίασης με σεβασμό στο **κόστος bottleneck**. Το **κόστος bottleneck** ενός κελιού είναι ο αριθμός των μονοπατιών στα οποία ανήκει και παρατηρείται ή προκαλείται παραβίαση του χρονισμού. Ένα σημείο **bottleneck** είναι ένα σημείο στο κυκλώματος οποίο προκαλεί πολλαπλές χρονικές παραβιάσεις. Η ανάλυση τέτοιων σημείων αναγνωρίζει τέτοια σημεία και καθορίζει την πιθανότητα βελτίωσης του κυκλώματος μέσω της «θεραπείας »ενός τέτοιου σημείου.

Υπάρχουν 4 διαφορετικοί τύποι αναζήτησης σημείων **bottleneck** τα οποία εκφράζονται με αυτές τις επιλογές.

• **delta_delay** – εμφανίζει τα δίκτυα «θύματα» της καθυστέρησης στα οποία παρατηρείται η μεγαλύτερη απόκλιση καθυστέρησης από ένα κατώφλι.

• **delta_delay_ratio** – εμφανίζει τα δίκτυα «θύματα» της καθυστέρησης στα οποία παρατηρείται η μεγαλύτερη καθυστέρηση σε σχέση με την καθυστέρηση επιπέδου (stage delay.

•**total_victim_delay_bump** – εμφανίζει τα δίκτυα «θύματα» της καθυστέρησης στα οποία παρατηρείται η μεγαλύτερη ποσότητα μη φιλτραρισμένων bump υψών(bump heights).

•**delay_bump_per_aggressor** – εμφανίζει τα δίκτυα «θύτες» τα οποία προκαλούν το φαινόμενο **bottleneck** και τα οποία κατατάσσονται ανάλογα με τον αριθμό των μονοπατιών που επηρεάζουν και με το ποσό της καθυστέρησης που προκαλούν.

Εξ' ορισμού το επίπεδο του κατωφλίου είναι μηδέν, το οποίο σημαίνει πως τα κόστη συνδέονται με χρονικές παραβιάσεις μονό. Εάν δεν υπάρχουν παραβιάσεις, δεν υπάρχουν κόστη και δεν επιστρέφεται κανένα δίκτυο.

Πραγματοποιώντας μια πλήρη ανάλυση χρονισμού

Ως συνέχεια της φόρτωσης, ορισμού των περιορισμών – παραμέτρων της σχεδίασης και εξιδανίκευσης των διάφορων συνθηκών ακολουθεί η πλήρης ανάλυση χρονισμού και μελέτη των αποτελεσμάτων. Οι ακόλουθες εντολές είναι συχνά χρησιμοποιούμενες σε αυτό το κομμάτι (οι λειτουργίες τους έχουν αναλυθεί σε προηγούμενο κεφάλαιο και θεωρείται (λόγω συμβατότητας με τα εργαλεία της **Synopsys**) περιττή η περεταίρω ανάλυση τους από τον γράφοντα):

- report_timing
- report_constraint
- report_bottleneck

- report_analysis_coverage
- report_delay_calculation

Συνοψίζοντας όλα τα παραπάνω στοιχεία παρατίθεται το Script που χρησιμοποιήθηκε.

```
echo "antzimas> set search_path..."
set search_path "/usr/synopsys/PrimeTime_Y-2006.06/libraries/syn
/home/antzimas/Desktop/antzimas/carry_lookahead_adders/cla/adder4/
/home/antzimas/Desktop/antzimas/carry_lookahead_adders/cla/adder4/
/usr/synopsys/PrimeTime_Y-2006.06/dw/syn_ver"
echo "antzimas> set link_path..."
set link_path "/usr/synopsys/PrimeTime_Y-
2006.06/libraries/syn/umce13h210t3_wc_108V_125C.db"
echo "antzimas> set symbol_library..."
set symbol_library "/usr/synopsys/syn_vY-
2006.06/libraries/syn/umce13h210t3.sdb"

remove_design -all
echo "antzimas> read_db..."
read_db adder4.db

echo "antzimas> set_operating_conditions..."
set_operating_conditions -analysis_type on_chip_variation

echo "antzimas> set si_enable_analysis..."
set si_enable_analysis TRUE

echo "antzimas> timing_slew_propagation_mode..."
set timing_slew_propagation_mode worst_slew
```

```
echo "antzimas> current_design..."
current_design adder4
echo "antzimas> link_design..."
link_design
echo "antzimas> read_parasitics..."
read_parasitics -keep_capacitive_coupling parasitics_adder4.reduced.spef

set_load 0.03 [all_inputs]
set_load 0.03 [all_outputs]
set_wire_load_model -name suggested_10K -library umce13h210t3_wc_108V_125C
set_max_fanout 3 adder4
set_max_transition 0.1 adder4
set_max_delay 0.5 -from [all_inputs] -to [all_outputs]

check_timing -include { no_driving_cell ideal_clocks  partial_input_delay
unexpandable_clocks } -verbose

echo "antzimas> update_timing..."
echo "update_timing -full"

echo "antzimas> write_parasitics..."
write_parasitics -format SBPF parasitics_adder4.reduced.sbpf

echo "antzimas> write_sdc..."
write_sdc adder4_after_pt.sdc

echo "antzimas> Building Reports..."

echo "antzimas> report_net..."
report_net >> ./adder4/PrimeTime_Reports/ReportNet.txt

echo "antzimas> report_cell..."
report_cell >> ./adder4/PrimeTime_Reports/ReportCell.txt

echo "antzimas> report_constraint..."
```

```
        report_constraint -significant_digits 2 -verbose >>
./adder4/PrimeTime_Reports/ReportConstraints.txt

        echo "antzimas> report_design..."
        report_design >> ./adder4/PrimeTime_Reports/ReportDesign.txt

        echo "antzimas> report_timing..."
        report_timing -from [all_inputs] -to [all_outputs] -path full -input_pins -nets -
transition_time -capacitance -crosstalk_delta -trace_latch_borrow -
report_ignored_register_feedback -significant_digits 2 >>
./adder4/PrimeTime_Reports/ReportTiming.txt

        echo "antzimas> report_wire_load..."
        report_wire_load >> ./adder4/PrimeTime_Reports/ReportWire_load.txt

        echo "antzimas> report_noise..."
        report_noise -verbose >> ./adder4/PrimeTime_Reports/ReportNoise.txt

        echo "antzimas> report_noise_parameters..."
        report_noise_parameters >>
./adder4/PrimeTime_Reports/ReportNoise_Parameters.txt

        echo "antzimas> report_bottleneck..."
        report_bottleneck -max_cells 2000 >>
./adder4/PrimeTime_Reports/ReportBottleneck.txt

        echo "antzimas> report_si_bottleneck..."
        report_si_bottleneck -slack_lesser_than 0.1 >>
./adder4/PrimeTime_Reports/ReportSI_Bottleneck.txt

        echo "antzimas> report_si_noise_analysis..."
        report_si_noise_analysis >>
./adder4/PrimeTime_Reports/ReportSI_Noise_Analysis.txt
```

6.ΑΝΑΛΥΣΗ ΚΑΤΑΝΑΛΙΣΚΩΜΕΝΗΣ ΙΣΧΥΟΣ

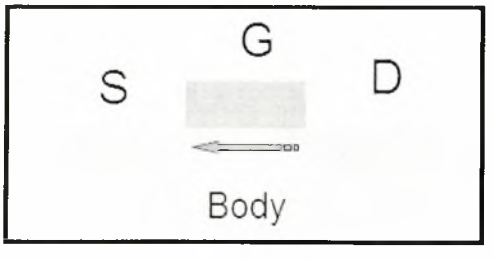
6.1) ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΚΑΤΑΝΑΛΩΣΗΣ ΙΣΧΥΟΣ :

Στο κεφάλαιο αυτό θα γίνει μια εισαγωγή σε βασικές έννοιες κατανάλωσης ισχύος, οι οποίες θα εφαρμοστούν στα κυκλώματα που διεξάγεται η μελέτη. Αυτό γίνεται διότι πάνω στο θεωρητικό υπόβαθρο που θα περιγράφει βασίζεται όλη η μεθοδολογία που ακολουθείται κατά το κομμάτι της ανάλυσης της καταναλισκόμενης ισχύος με τα εργαλεία της **Synopsys**. Παρουσιάζονται, επίσης, ορισμοί και τύποι για στατική και δυναμική ισχύ.

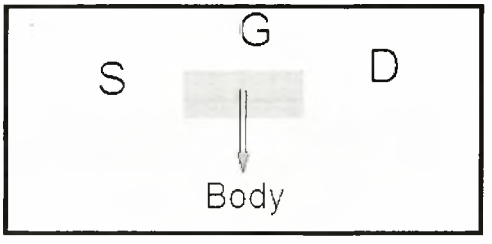
Η ισχύς που καταναλώνεται σε ένα κύκλωμα μπορεί να χωριστεί σε δύο ευρύτερες κατηγορίες:

- **Στατική ισχύς (static power).**
- **Δυναμική ισχύς (dynamic power).**

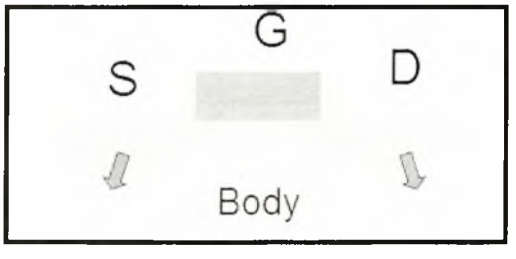
Στατική ισχύς είναι η ισχύς που καταναλώνει μια πύλη όταν δεν αλλάζει την λογική τιμή της εξόδου της, είναι δηλαδή μη ενεργή ή αλλιώς στατική. Στατική ισχύς καταναλώνεται για πολλούς λόγους. Το μεγαλύτερο ποσοστό κατανάλωσης στατικής ισχύος στις τεχνολογίες που αναλύθηκαν, όπως φαίνεται και στο σχήμα προκύπτει από την αγωγή υποκατωφλίου μεταξύ πηγής και υποδοχής (**source-to-drain subthreshold leakage**)



και το ρεύμα διαρροής πύλης καναλιού που οφείλεται σε φαινόμενα διόδευσης από το οξείδιο (**tunneling effects**).



Κατανάλωση στατικής ισχύος, παρουσιάζεται επειδή άγει η ανάστροφα πολωμένη δίοδος μεταξύ των στρωμάτων διαχύσεως και του υποστρώματος. Για το λόγο αυτό, η στατική ισχύς συχνά αναφέρεται ως ισχύς διαρροής (**leakage power**).



Δυναμική ισχύς , σε αντίθεση με την προηγούμενη κατάσταση, είναι η ισχύς που καταναλώνεται όταν μια πύλη είναι ενεργή. Ένα κύκλωμα ονομάζεται ενεργό κάθε φορά που οι τάσεις των δικτύων του εναλλάσσονται σύμφωνα πάντα με τις εισόδους που εφαρμόζονται στο κύκλωμα. Επειδή λοιπόν, η τιμή της τάσης σε ένα δίκτυο εισόδου μπορεί να αλλάξει χωρίς αυτό να σημαίνει ότι θα έχουμε και μια εναλλαγή λογικής τιμής στο δίκτυο εξόδου, δυναμική κατανάλωση ισχύος μπορεί να παρατηρηθεί και σε περιπτώσεις που ένα δίκτυο εξόδου δεν αλλάξει λογική τιμή.

Η δυναμική κατανάλωση ισχύος διαιρείται σε:

- **Switching power** (ισχύς λόγω μεταγωγής λογικής τιμής). Όταν αναφερόμαστε σε καταναλισκόμενη ισχύ λόγω μεταγωγής λογικής τιμής

θεωρούμε τη συνολική ισχύ που καταναλώνεται από την φόρτιση και εκφόρτιση της χωρητικότητας στην έξοδο ενός κελιού της σχεδίασης μας. Στην έξοδο του κελιού η συνολική χωρητικότητα του πυκνωτή αποτελείται από το άθροισμα της χωρητικότητας του δικτύου και της πύλης στην έξοδο. Γνωρίζουμε πως σε κάθε μετάβαση σε αντίθετη λογική κατάσταση έχουμε την φόρτιση ή εκφόρτιση του συγκεκριμένου πυκνωτή.

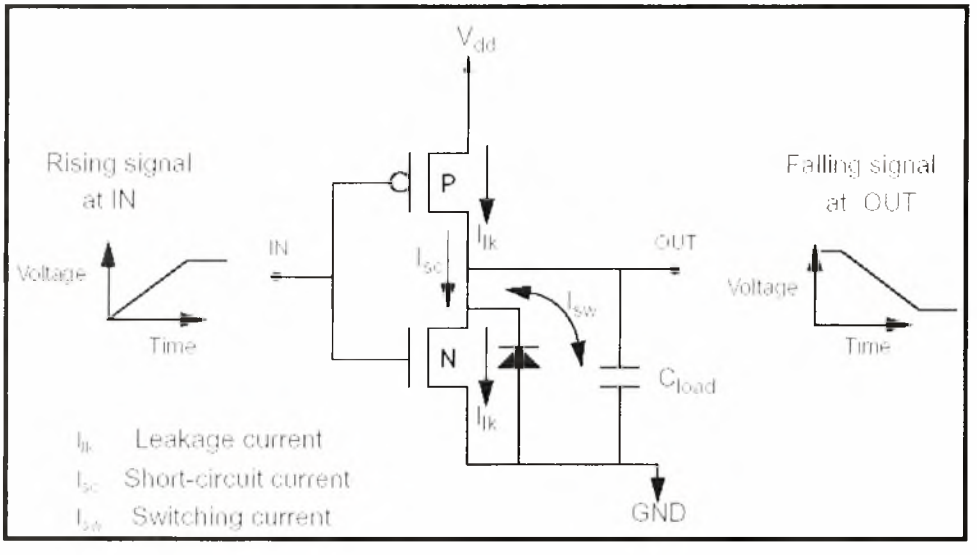
Συνεπώς αυτή η διαδικασία της αλλαγής της κατάστασης του πυκνωτή αυξάνει την ισχύ στο κύκλωμα που μελετάμε. Τελικά, η ισχύς λόγω μεταγωγής λογικής τιμής είναι μια συνάρτηση δύο παραγόντων, της συνολικής χωρητικότητας φορτίου στην έξοδο του κελιού και της συχνότητας των εναλλαγών λογικών τιμών. Για αυτόν ακριβώς το λόγο η **switching power** αποτελεί ένα πολύ κρίσιμο παράγοντα αποτελώντας το 70% - 90% της συνολικής κατανάλωσης ισχύος σε ένα ενεργό **CMOS** κύκλωμα.

- **Internal power** (εσωτερική ισχύς). Ως «εσωτερική ισχύ» χαρακτηρίζουμε οποιαδήποτε ισχύ καταναλώνεται μέσα στα όρια ενός κελιού. Η εναλλαγή των τιμών των πυκνωτών μέσω αλλεπάλληλων φορτίσεων - εκφορτίσεων είναι μια αιτία αύξησης της ισχύς. Οπότε ένα κύκλωμα καταναλώνει εσωτερική ισχύ λόγω της φόρτισης και εκφόρτισης των οποιοδήποτε εσωτερικών χωρητικοτήτων που διαθέτει το κελί. Επιπροσθέτως, σε κάποια πύλη ενδεχόμενα να υπάρξει στιγμιαία εμφάνιση βραχυκυκλώματος μεταξύ του **P** και **N** τρανζίστορ της. Η ισχύς που παράγεται από αυτή τη μεταβολή ονομάζεται και ισχύς βραχυκυκλώσεως (**short-circuit power**).

Για τη πιο εύκολη κατανόηση της καταναλισκόμενης ισχύος παρατίθεται το ακόλουθο παράδειγμα:

Ένα ανοδικό σήμα εφαρμόζεται στην είσοδο **IN** με το σήμα να εναλλάσσεται από χαμηλή τιμή σε υψηλή. Συνεπώς το τρανζίστορ τύπου **N**

ανοίγει και το τρανζίστορ τύπου **P** κλείνει. Ωστόσο, για λίγο χρονικό διάστημα όσο έχουμε την εναλλαγή του σήματος, τόσο το **P** και το **N** τύπου τρανζίστορ μπορούν να είναι ανοιχτά ταυτόχρονα. Στο αναφερθέν αυτό χρονικό διάστημα το ρεύμα I_{sc} ρέει από την τάση V_{dd} στην γείωση **GND**, προκαλώντας κατανάλωση ισχύος βραχυκυκλώματος P_{SC}.



ΣΧΗΜΑ 6.1: Λειτουργεία ενός τρανζίστορ

Για κυκλώματα με γρήγορους χρόνους μεταγωγής, η κατανάλωση ισχύος βραχυκυκλώματος μπορεί να είναι μικρή. Ωστόσο, για κυκλώματα με αργούς χρόνους μεταγωγής, η κατανάλωση ισχύος βραχυκυκλώματος μπορεί να προκαλεί περίπου το 30% επί της συνολικής κατανάλωσης της πύλης. Η κατανάλωση ισχύος βραχυκυκλώματος επηρεάζεται από το μέγεθος του τρανζίστορ και την χωρητικότητα φορτίου στην έξοδο της πύλης. Όταν όμως αναφερόμαστε σε απλά κελιά που παρέχονται από μια βιβλιοθήκη, η εσωτερική ισχύς οφείλεται συνήθως στην ισχύ βραχυκυκλώματος. Για το λόγο αυτό σε αυτή την περίπτωση οι δύο αυτοί όροι θεωρούνται συνώνυμοι.

- **Leakage Power** (ισχύς διαρροής). Στην τεχνολογία **CMOS** τα τρανζίστορ που χρησιμοποιούνται (**NMOS**, **PMOS**) παρουσιάζουν μη μηδενικά ρεύματα διαρροής και υποκατωφλίου. Η παρουσία αυτού του

φαινομένου συνεπάγεται τη συμμετοχή των ρευμάτων αυτών στην συνολική κατανάλωση ισχύος, χωρίς να απαιτείται απαραίτητα η αλλαγή λογικών τιμών στην λειτουργία του κελιού σαν διακόπτη. Τα ρεύματα που είναι υπεύθυνα για αυτή τη λειτουργία είναι το *ρεύμα διαρροής ανάστροφης πόλωσης* και το *ρεύμα υποκατωφλίου* το οποίο διέρχεται από κάποιο κανάλι του τρανζίστορ, όταν αυτό είναι κλειστό.

Συνοψίζοντας όλα τα παραπάνω η κατανάλωση ισχύος ενός ψηφιακού CMOS κυκλώματος (και στην περίπτωση της εκπόνησης αυτής της εργασίας) δύναται να περιγραφεί από την ακόλουθη εξίσωση :

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

- $P_{average}$: μέση ισχύς που καταναλώνεται στο κύκλωμα.
- $P_{dynamic}$: δυναμική κατανάλωση ισχύος εξαιτίας της αλλαγή κατάστασης των διακοπτών.
- $P_{short\text{-}circuit}$: ποσοστό καταναλισκόμενης ισχύος όταν υπάρχει άμεσο μονοπάτι από την πηγή στην γείωση.
- P_{static} : στατική κατανάλωση ισχύος.

6.2) PRIME POWER & ΚΑΤΑΝΑΛΩΣΗ ΙΣΧΥΟΣ :

Συνεχίζοντας το **toolflow** που έχουμε ορίσει από την αρχή σε επίπεδο κατανάλωσης ισχύος παρατηρήθηκε πως τα αποτελέσματα του **DC** δεν ήταν ικανοποιητικά. Για αυτόν ακριβώς το λόγο και για να πάρουμε μια πιο σαφή εικόνα της πραγματικότητας καταφεύγουμε στην χρήση της πλατφόρμας του **Power Compiler (PC)** και του υποεργαλείου **Prime Power (PP)**. Το **Prime Power** είναι ένα εργαλείο ανάλυσης που λειτουργεί σε επίπεδο πυλών, το οποίο με ακρίβεια υπολογίζει την κατανάλωση ισχύος σε **cell – based** σχεδιάσεις. Κατά τα βιομηχανικά σχεδιαστικά πρότυπα για **power critical** εφαρμογές η μη χρησιμοποίηση αυτού του εργαλείου θεωρείται τραγικό λάθος. Τα χαρακτηριστικά που διαθέτει ο **Power Compiler** και που μας οδήγησαν στην επιλογή του ως εργαλείο ανάλυσης της καταναλισκόμενης ισχύος ψηφιακών κυκλωμάτων διαφαίνονται παρακάτω:

- ☑ Ανάλυση ισχύος σε **RTL** επίπεδο και επίπεδο πυλών.
- ☑ **RTL power estimation** το οποίο είναι τεχνολογικά εξαρτημένο.
- ☑ Ανάλυση των σχεδιάσεων για :
  - i. ισχύ μεταγωγής,
  - ii. εσωτερική ισχύ,
  - iii. ισχύ διαρροής.
- ☑ Εισαγωγή πληροφοριών για την δραστηριότητα μεταγωγής από :
  - i. τον χρήστη,
  - ii. **RTL** προσομοίωση,
  - iii. προσομοίωση σε επίπεδο πυλών,
  - iv. συνδυασμό αυτών.

- ☑ Υποστήριξη ακολουθιακών, ιεραρχικών, **gated clock**, **multiple clock** σχεδιάσεων.

- ☑ Υποστήριξη **RAM** και **I/O** μοντελοποίησης χρησιμοποιώντας λεπτομερή μοντέλα ισχύος εξαρτημένα από το μονοπάτι και την κατάσταση.

- ☑ Εξαγωγή αναφορών ισχύος σε όλα τα επίπεδα της διαδικασίας της σχεδίασης.

- ☑ Εξαγωγή αναφορών ισχύος για οποιοδήποτε επίπεδο της ιεραρχίας ώστε να έχουμε γρήγορο **debugging**.

- ☑ Υποστήριξη **interfaces** για τα εργαλεία προσομοίωσης όπως ο **VSS**, ο **MTI**, **VCS**, και **Verilog-XL** για καταγραφή της πληροφορίας της δραστηριότητας μεταγωγής.

Επίσης κατά την βελτιστοποίηση υπάρχουν και οι εξής δυνατότητες:

- **Push-button** μείωση της κατανάλωσης ισχύος σε επίπεδο πυλών.

- Ταυτόχρονη βελτιστοποίηση του χρονισμού, της ισχύος και του μεγέθους του κυκλώματος.

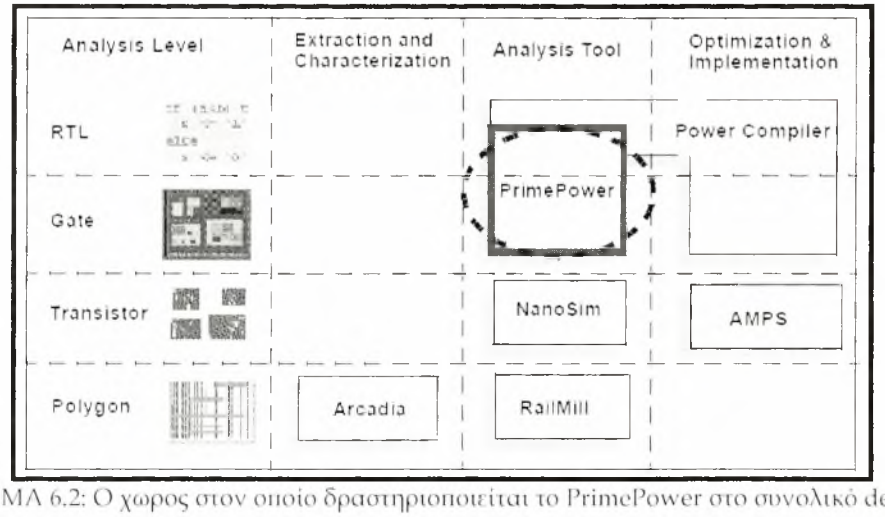
- Βελτιστοποίηση με κριτήρια βασισμένα
  - στην δραστηριότητα μεταγωγής (**switching activity**),
  - την χωρητικότητα (**capacitance**),
  - τους χρόνους μετάβασης (**transition times**).

- Δυνατότητα ανάλυσης της ισχύος και βελτιστοποίηση σύμφωνα με το ίδιο λεπτομερές μοντέλο ισχύος που προέκυψε από την ανάλυση.

- Συμβατότητα με όλα τα εργαλεία της **SYNOPSYS** (**Design Compiler**, **IC Compiler**, **Floorplan Manager**, **Physical Compiler**, και **Module Compiler**, κ.α.).

- Υποστήριξη **RAM** και **I/O** μοντελοποίησης χρησιμοποιώντας λεπτομερή μοντέλα ισχύος εξαρτημένα από την διαδρομή και την κατάσταση.

Σε επίπεδο πυλών το **PP** υποστηρίζει τόσο **στατική** όσο και **δυναμική** ανάλυση της κατανάλωσης ισχύος. Κατά την εκπόνηση της παρούσης διπλωματικής εργασίας θα γίνει **χρήση της δυναμικής** προσέγγισης του υπολογισμού της καταναλισκόμενης ισχύος η οποία θα πραγματοποιηθεί με την κατάλληλη χρήση των **VCD** αρχείων που έχουν ήδη εξαχθεί σε προηγούμενο στάδιο. Ενδεικτικά, για την αποσαφήνιση του χώρου που δραστηριοποιούμαστε την παρούσα χρονική στιγμή της μελέτης παρατίθεται το παρακάτω σχήμα:



ΣΧΗΜΑ 6.2: Ο χωρος στον οποίο δραστηριοποιείται το PrimePower στο συνολικό design flow

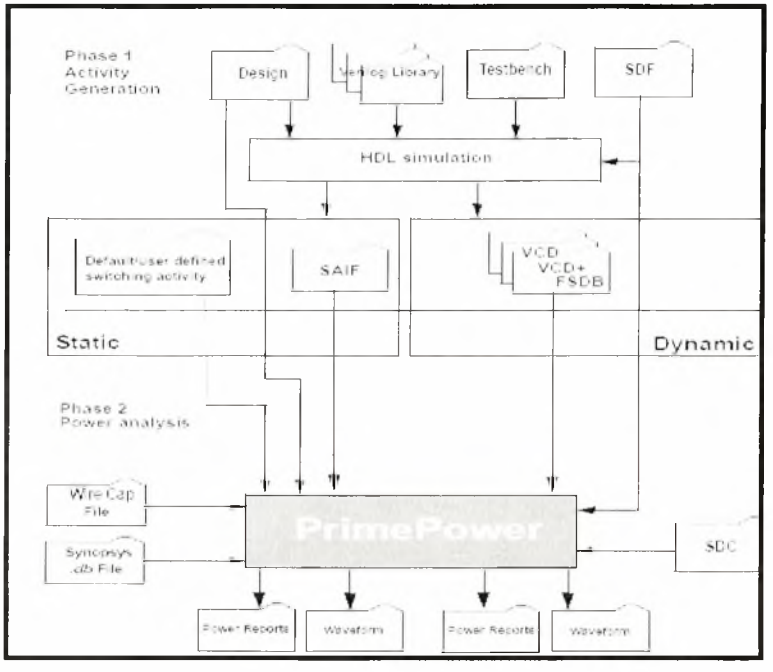
Στην παρούσα φάση υπάρχει η δυνατότητα επιλογής από το χρήστη του τρόπου λειτουργίας ανάμεσα σε **event-based** και **statistical activity based** ανάλυση καταναλισκόμενης ισχύος σε επίπεδο πυλών.

Η ***Statistical activity based power analysis*** χρησιμοποιείται για μέση ανάλυση ισχύος. Το **PrimePower** υποστηρίζει **propagation** από λειτουργία εναλλαγής βασισμένη σε **defaults**, καθορισμένα από το χρήστη. Τα **switching** ή **switching derived** δεδομένα προέρχονται από **HDL simulation** (είτε **RTL** ή **gate-level**), δηλαδή βασίζονται σε προεπιλεγμένα αποτελέσματα που δίνουν μια εξιδανικευμένη προσέγγιση της συνολικής ισχύος που θα καταναλωθεί.

Η ***Event-based power analysis*** χρησιμοποιείται για εξαιρετικά ακριβή ανάλυση ενέργειας με ιδιαίτερη προσοχή στις παραβιάσεις του χρονισμού. Το **PrimePower** υποστηρίζει ανάλυση βασισμένη σε **gate-level simulation activity** over **time**, δηλαδή σε **dynamic analysis** (**peak power**), όπου λαμβάνουμε μια λεπτομερέστερη εκτίμηση βασιζόμενη σε στοιχεία που εισάγονται από τον χρήστη σε μορφή αρχείων.

Για την προσομοίωση της λειτουργίας της σχεδίασης πάνω στην οποία θα βασιστούμε για την εξαγωγή των ζητούμενων αποτελεσμάτων θα πρέπει να περάσουμε δυο φάσεις – στάδια όπως φαίνονται παρακάτω:

- **Phase 1**: δημιουργία κατάλληλης πληροφορίας για το **switching activity (δραστηριότητα μεταγωγής)** του κυκλώματος
- **Phase 2**: δημιουργία του **power profile** της σχεδίασης και διενέργεια της ανάλυσης της καταναλισκόμενης ισχύος.



ΣΧΗΜΑ 6.3: Ροή υπολογισμού καταναλισκώμενης ισχύος με το PrimePower

Κατά την πρώτη φάση της εξομοίωσης και εφόσον έχουμε δεχθεί πως θα διενεργηθεί δυναμική ανάλυση για τους λόγους που έχουν προαναφερθεί, γίνεται εισαγωγή του **gate-level time-based switching activity** του κυκλώματος. Στο κομμάτι της **HDL simulation**, όπως έχει ήδη παρουσιαστεί έχει γίνει χρήση του **Modelsim** για την εξαγωγή των **VCD (Value Change Dump)** αρχείων που θα αποτελέσουν την είσοδο για το **Prime Power**.

Λειτουργώντας πάντα σε επίπεδο πυλών τα εν λόγω αρχεία περιέχουν μεγάλο κομμάτι από το σύνολο των πληροφοριών για το **switching activity (δραστηριότητα μεταγωγής)** που εξάγεται για οποιαδήποτε σχεδίαση.

Τέλος για την συνέχεια και την κατεξοχήν λειτουργία ακολουθήθηκε η τεχνοτροπία που ακολουθήθηκε και στο κομμάτι της σύνθεσης. Πιο συγκεκριμένα ο καθορισμός των συνθηκών και περιορισμών σύμφωνα με τους οποίους θα διεξαχθεί η βελτιστοποίηση της ανάλυσης της καταλισκώμενης ισχύος του «υπό μελέτη» ψηφιακού κυκλώματος με την μορφή ενός **script** το οποίο παρατίθεται στη συνέχεια. Το **script** περιέχει το σύνολο των εντολών θα εκτελέσει σειριακά ο πυρήνας του **Power Compiler** ο

οποίος καλείται μέσω του **PrimePower**. Σαν σύνοψη των **script** που χρησιμοποιηθήκαν αναλύεται το παρακάτω υπόδειγμα :

```
1.  set search_path          ". . / path1 . . / path2 ."
2.  set link_library         " * library.db"
3.  read_db          . / adderi4.db
4.  current_design          adderi4
5.  link
6.  set_hier_sep /
7.  read_vcd         -strip_path testbench / dut  . / output.vcd
8.  set_input_transition   0.1 [all_inputs]
9.  read_parasitics           . / parasitics.reduced.spef
10. set_waveform_options -interval 1 -file pwr_vcd -format fsdb
11. calculate_power          -waveform -reset_neg_power
12. report_power -file pwr_vcd
13. report_power -net
14. report_power -cell
15. report_power -leaf
```

Οι πρώτες δύο εντολές είναι γνωστές από την ανάλυση του **script** που χρησιμοποιήθηκε κατά την διαδικασία της σύνθεση σε προηγούμενο κεφάλαιο. Οι παράμετροι λοιπόν που περιέχονται δείχνουν την τοποθεσία στην οποία θα «ψάξει» ο **PP** για την εύρεση των **design data**.

Τα **path** που παρουσιάζονται ως παράμετροι του "**search_path**" περιέχουν όλα τα αρχεία που θα ενεργοποιηθούν στην αρχική φάση , δηλαδή τις σχεδιάσεις σε οποιαδήποτε μορφή και αν βρίσκονται(**.vhd, .v, .db**) και τις βιβλιοθήκες.

Η "**link_path**" προσδιορίζει αποκλειστικά και μόνο τις βιβλιοθήκες που χρησιμοποιούνται και περιέχουν τα κατάλληλα **elements** της υπό

ανάλυση σχεδίασης. Το **PP** ψάχνει για σχεδιαστικά στοιχεία (**design elements**) στις βιβλιοθήκες σειριακά.

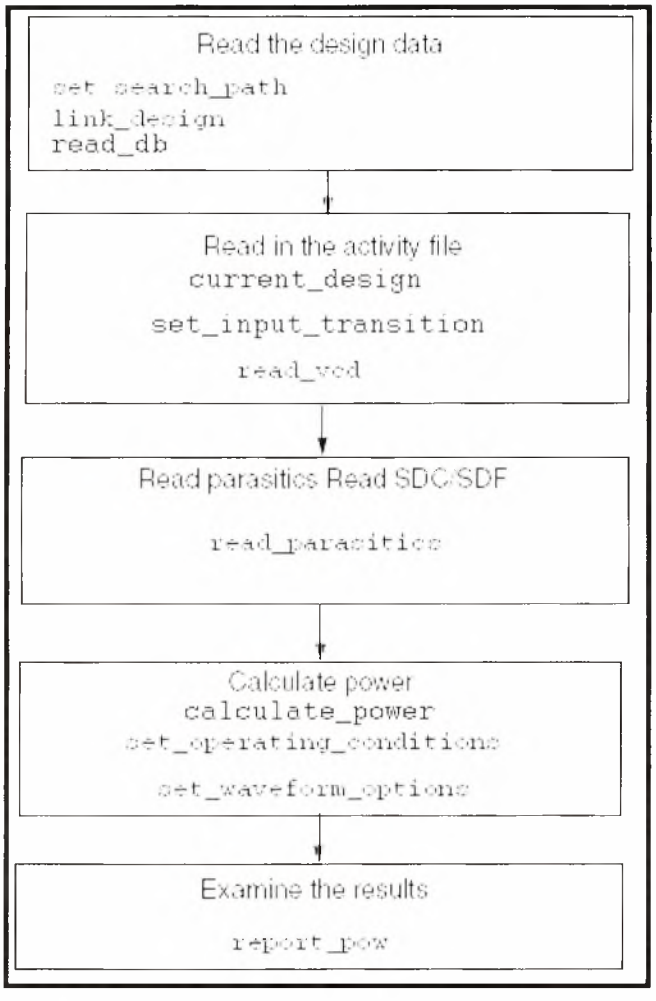
Στην ίδια φιλοσοφία κινείται και η προσπέλαση της μνήμης στην οποία είναι αποθηκευμένη η σχεδίαση που θα χρησιμοποιηθεί. Λόγω προηγούμενης σύμβασης (η οποία έχει ήδη αναλυθεί σε προηγούμενο κεφάλαιο) διαβάζουμε την σχεδίαση σε μορφή **.db** αρχείου με την εντολή "**read_db**". Τα **.db** αρχεία είναι έξοδος του **Design Vision** στο στάδιο της σύνθεσης και παρουσιάζονται υπό την μορφή **netlists**.

Στην τέταρτη εντολή που ακολουθεί ("**current_design**") τίθεται ή ανακτάται η τρέχουσα σχεδίαση. Απαραίτητη προϋπόθεση είναι η ένθεση της εντολής αυτής πριν από την "**link**".

Έχοντας συντάξει ήδη την εντολή "**current_design**" το μόνο που απομένει ώστε να θέσουμε στο προσκήνιο της λειτουργίας του προγράμματος μια σχεδίαση έτοιμη για ανάλυση ενέργειας. Αυτό γίνεται με την προαναφερθείσα εντολή "**link**". Όλες οι προσπελάσεις στις βιβλιοθήκες οι οποίες είναι ορισμένες να πραγματοποιηθούν σε αυτό το στάδιο διενεργούνται και φέρονται στην μνήμη του υπολογιστή. Σε περίπτωση απώλειας η παράληψης της εντολής "**link**" το **PP** χρησιμοποιεί μια διεργασία **autoload** η οποία διεξάγει τις ενέργειες για τις οποίες είναι υπεύθυνη η παραλειφθείσα εντολή.

Σε αυτό το σημείο θα πρέπει να προσδιοριστεί η θέση του **testbench** (περιέχει τις κατάλληλες αναφορές στην σχεδίαση) και του **VCD** αρχείου που έχει εξαχθεί σε προγενέστερη φάση της συνολικής διεργασίας με την εντολή "**read_vcd**".

Ενδεικτικά παρατίθεται το παρακάτω σχήμα στο οποίο διαφαίνεται η ροή των εντολών που χρησιμοποιήθηκαν.



ΣΧΗΜΑ 6.4: Συνοπτική παρουσίαση των βασικών εντολών στο PrimePower

Στη συνέχεια συντάσσεται η "**set_input_transition**", η οποία προσδιορίζει ένα **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**.

Για να μπορέσει το **PP** να υπολογίσει την καταναλισκόμενη ισχύ στο κύκλωμα θα πρέπει να εισαχθούν μερικές ακόμη πληροφορίες. Αυτές έχουν σχέση με τα παρασιτικά του κυκλώματος που αναλύεται. Με την μορφή των **.spef** αρχείων διαβάζεται με την εντολή "**read_parasitics**". Ότι πληροφορία χρειάζεται ώστε να έχουμε πιο αποδοτικό υπολογισμό της συνολικής ισχύος.

Ο υπολογισμός της τελικής ισχύος γίνεται με την "**calculate_power**" και ακολουθώντας την φιλοσοφία των **reports** λαμβάνουμε αναλυτικές αναφορές για την ισχύ που καταναλώνεται τόσο στο σύνολο της σχεδίασής μας όσο και σε κάθε **net**, **cell** και **leaf**. Η ανάλυση της χρήσιμης πληροφορίας που εξάγεται από αυτές τις αναφορές επιτρέπει στον εκάστοτε σχεδιαστή να οπισθοδρομήσει στην συνολική σχεδιαστική εργασία τροποποιώντας ανάλογα ορισμένα **components** της συνολικής σχεδίασης των οποίων η συμπεριφορά δεν ικανοποιεί τα βιομηχανικά πρότυπα και τις σχεδιαστικές απαιτήσεις και περιορισμοί οι οποίοι έχουν υποβληθεί στην αρχή της εργασίας.

7.ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

7.1 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΓΚΡΙΣΕΙΣ :

Στο κεφάλαιο αυτό θα παρουσιάσουμε μια σειρά αποτελεσμάτων τα οποία προέκυψαν μέσω των διεργασιών που αναφέρθηκαν σε προηγούμενες ενότητες της παρούσας εργασίας.

Τα «μετρικά» στοιχεία που χρησιμοποιούνται για τις συγκρίσεις είναι η κατανάλωση ισχύος και η καθυστέρηση εμφάνισης του αποτελέσματος στην έξοδο του κυκλώματος. Κατά την παρούσα χρονική στιγμή εκπόνησης της διπλωματικής θεωρήσαμε επαρκή την εξέταση αυτών των δύο παραγόντων σε βάθος κάνοντας μόνο μια απλή αναφορά στην επιφάνεια που καταλαμβάνουν τα κυκλώματα. Αυτό γίνεται στα πλαίσια της παγκόσμιας τάσης μιας και τα κατασκευαστικά μεγέθη είναι πολύ μικρά για να δημιουργούν κάποιο πρόβλημα χώρου.

Η πρώτη παρατήρηση που μπορεί να γίνει από ένα αναγνώστη είναι, όπως ήταν αναμενόμενο, το γεγονός πως σε όλες τις τοπολογίες αθροιστών που εξετάστηκαν, τόσο η καθυστέρηση εμφάνισης του αποτελέσματος, όσο και η ισχύς που καταναλώνεται και η περιοχή που καλύπτεται αυξάνεται σε κάθε αθροιστή όσο αυξάνεται το μήκος των τελεστών του. Αυτό είναι πολύ εύκολα αντιληπτό εάν λάβουμε και παρατηρήσουμε προσεκτικά και τον αριθμό των κελιών που χρησιμοποιήθηκαν κατά την επεξεργασία. Για παράδειγμα παρατίθεται ο παρακάτω πίνακας.

Carry Lookahead Structural					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,38	2,19	3,52	7,44	14,95
	1,88E+05	3,91E+08	8,10E+08	1,64E+09	3,32E+09
	0	0	0	0	0
	19	36	79	149	297

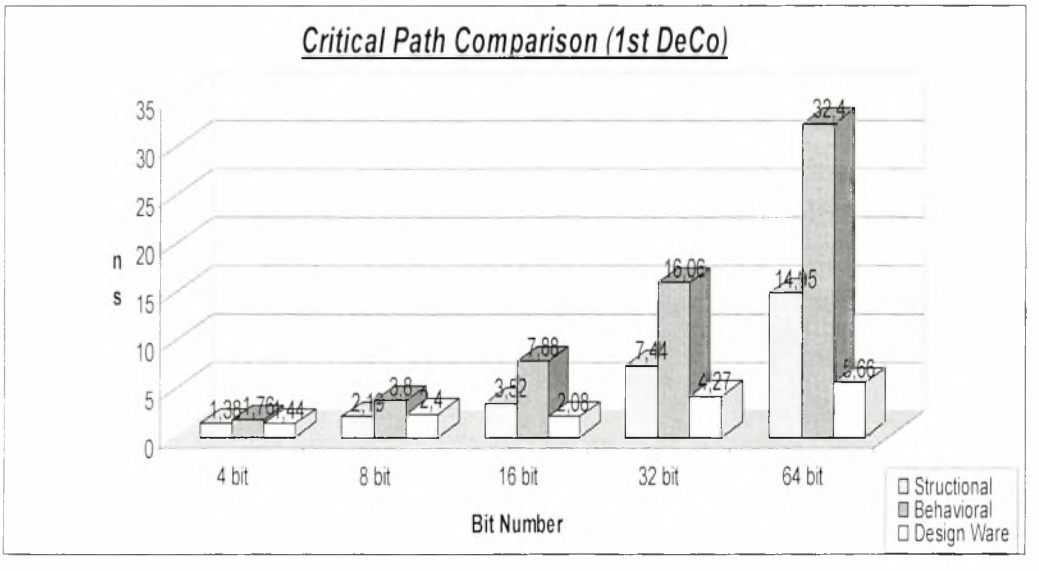
Στο Παράρτημα Α βρίσκονται συγκεντρωμένα όλα τα αποτελέσματα που ελήφθησαν καθώς επίσης και πίνακες αποτελεσμάτων παλαιότερης διπλωματικής που χρησίμευσε σαν μέτρο σύγκρισης των μεθόδων λήψης των

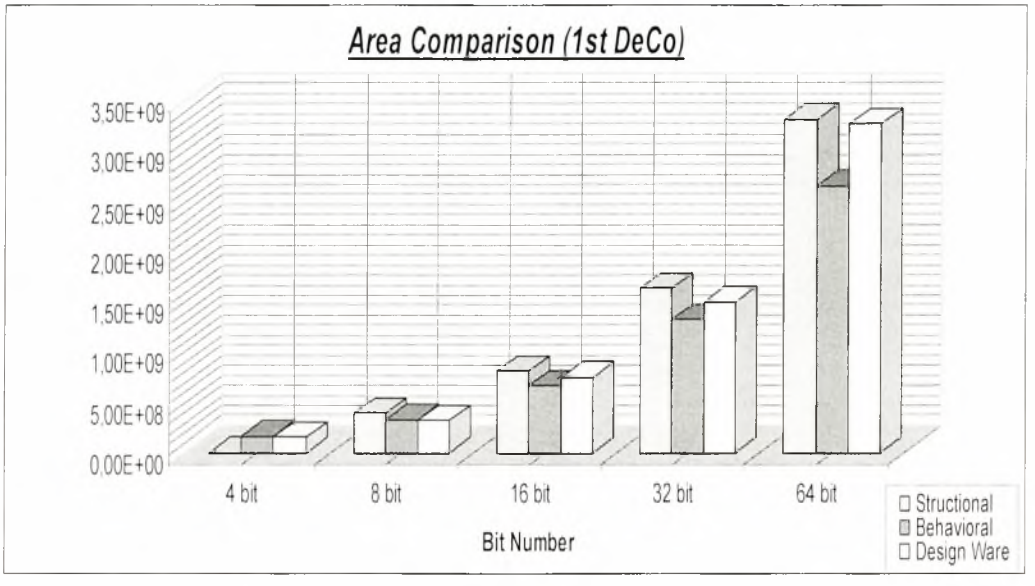
μετρήσεων. Τα επίπεδα που θα πραγματοποιηθούν οι συγκρίσεις θα κινούνται στο εξής άξονα:

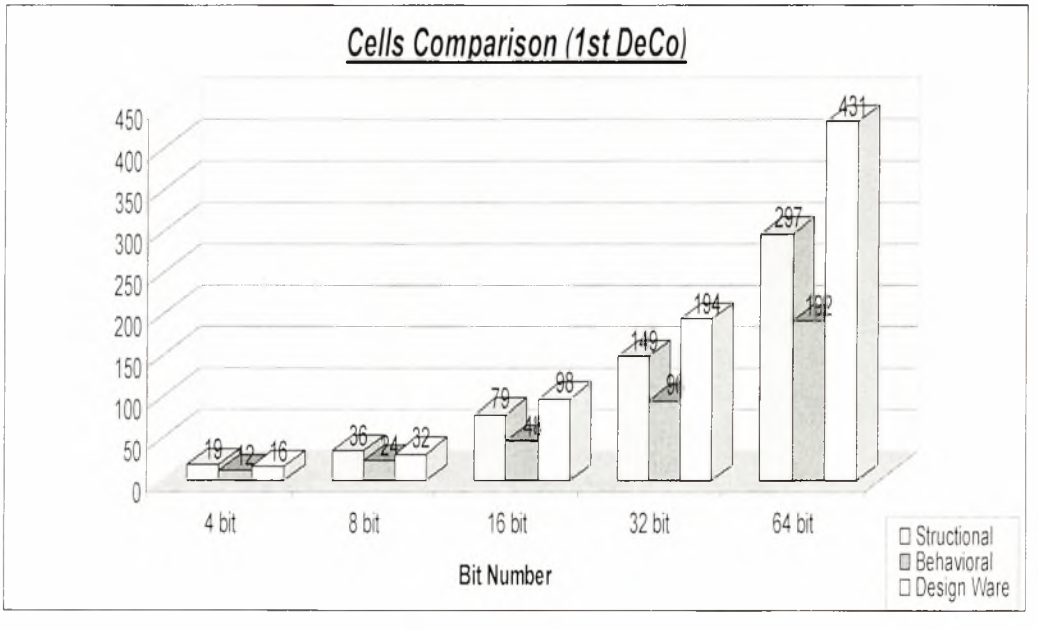
Κατά μήκος της ροής των εργαλείων εξετάζονται όλοι οι αθροιστές μεταξύ τους σε κάθε επίπεδο και έρχονται και σε σύγκριση με παλαιότερες μετρήσεις. Στη συνέχεια ακολουθεί περαιτέρω εξέταση με βάση το προηγούμενο επίπεδο ώστε να διαπιστωθεί η ορθότητα της μεθόδου που ακολουθήθηκε μέσω της επαλήθευσης των υποθέσεων και των προσδοκιών που τεθήκαν κατά την αρχή της εκπόνησης της διπλωματικής αυτής εργασίας. Υπενθυμίζουμε πως οι μονάδες μέτρησης χρόνου είναι το **ns**, της ενέργειας είναι το **W**.

Σύγκριση Αθροιστών κατά το 1ο πέρασμα από DeCo (Design Compiler)

Επειδή θέλαμε να δώσουμε έμφαση στη μέθοδο που χρησιμοποιήσαμε πήραμε τρεις εκδοχές σχεδίασης του αθροιστή που εξετάζουμε. Υπενθυμίζουμε πώς η σχεδίαση μας δεν έχει κανένα περιορισμό.

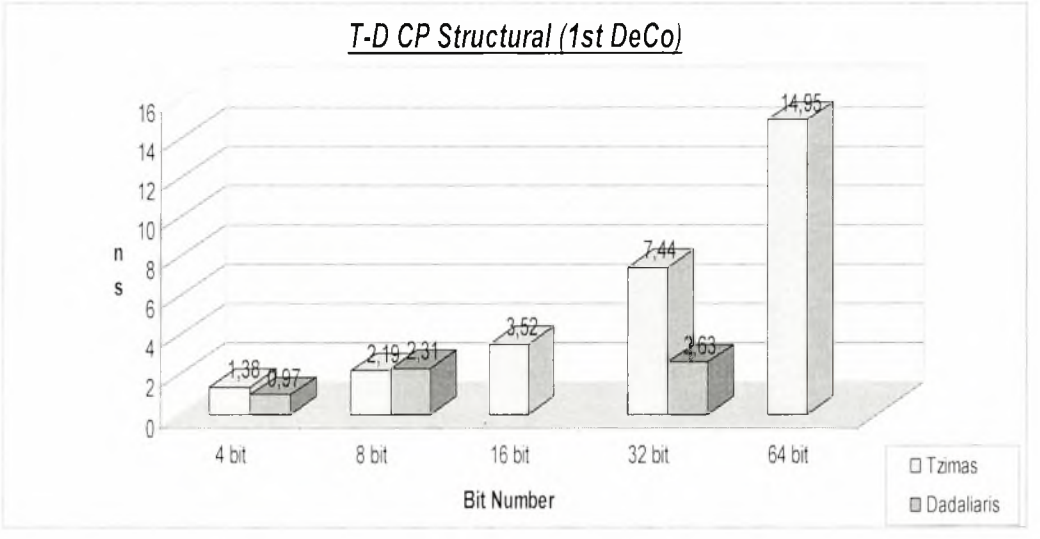


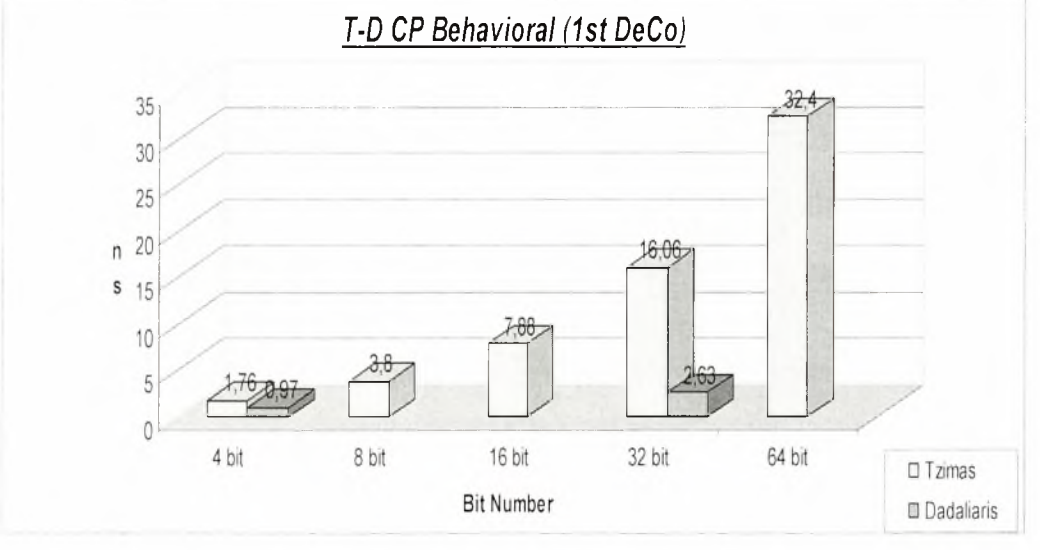




Είναι απολύτως αναμενόμενο το αποτέλεσμα που λαμβάνουμε στην σύγκριση χρονισμού που διεξήγαμε με την πρώτη σύνθεση καθώς ο αθροιστής της Synopsys είναι πιο γρήγορος από τους επίπεδες σχετικά όχι και τόσο βελτιστοποιημένους άλλους δυο αθροιστές.

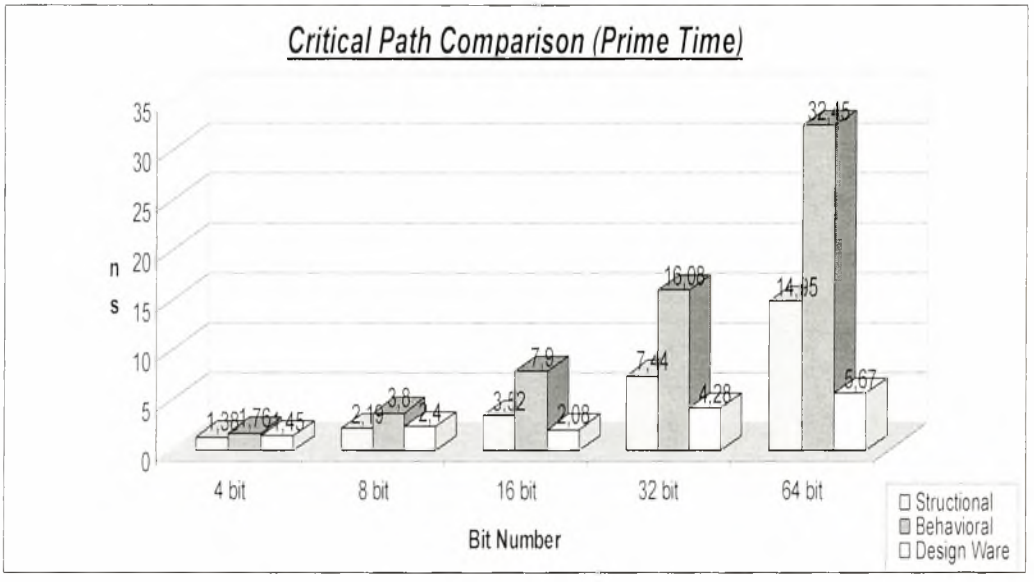
Ακολουθεί η σύγκριση σε αυτό το επίπεδο σε επίπεδο χρονισμού με τον αντίστοιχο αθροιστή προηγούμενης διπλωματικής.

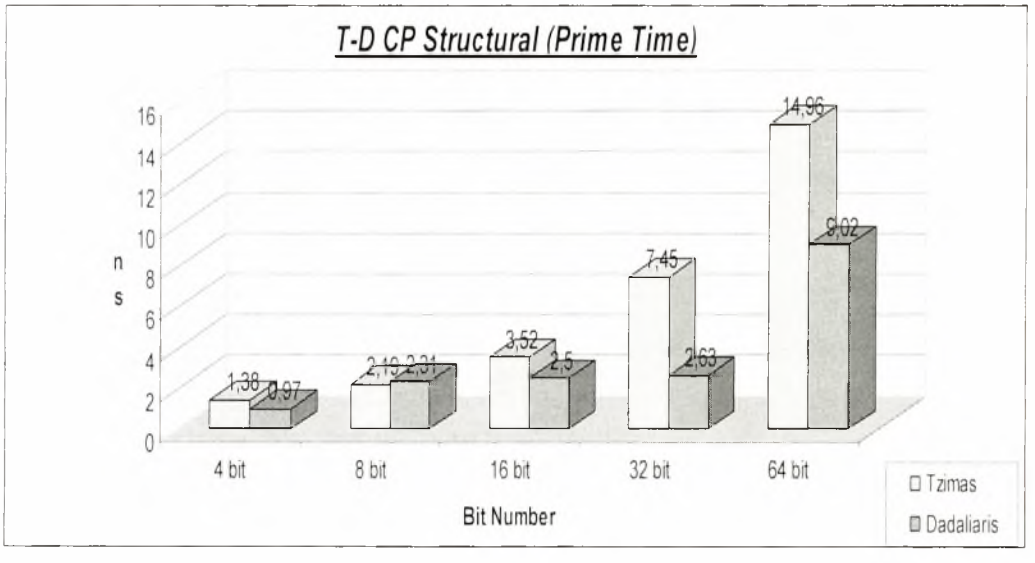


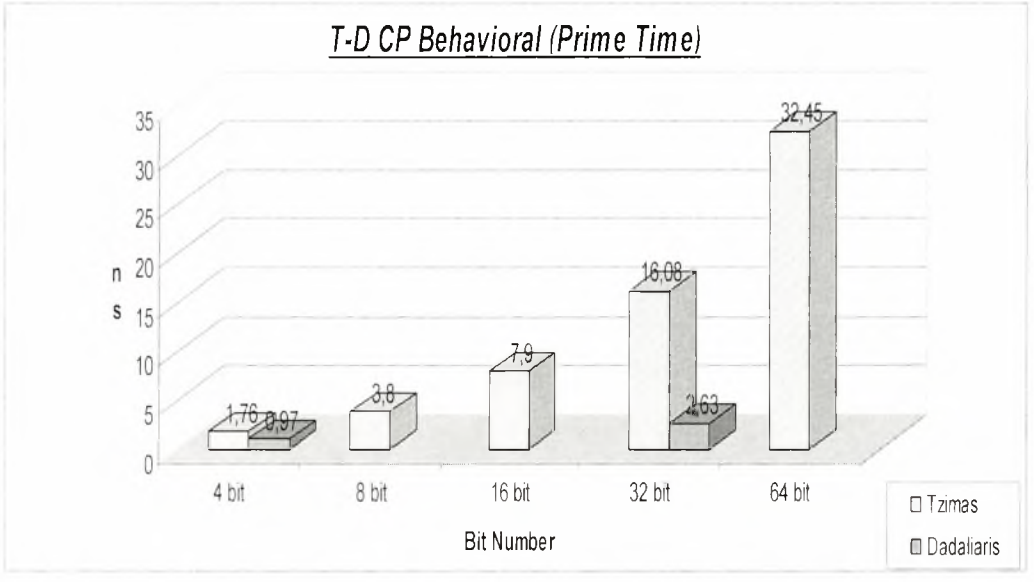


Σύγκριση Αθροιστών μετά το πέρασμα από το Prime Time

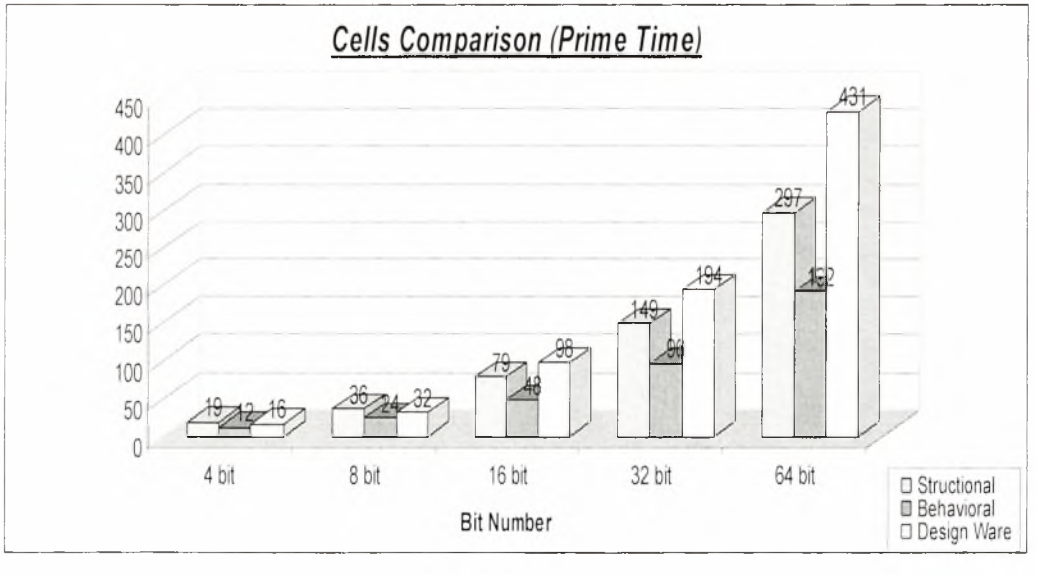
Σε αυτό το σημείο είναι πολύ λογικό να προξενήσει απορία στον αναγνώστη η μη ύπαρξη αλλαγής στα αποτελέσματα σε σχέση με το προηγούμενο στάδιο. Αυτό είναι σαφώς αναμενόμενο διότι με την στατική ανάλυση χρονισμού που διεξάγουμε στην ουσία λαμβάνουμε τους περιορισμούς με τους οποίους θα βελτιστοποιήσουμε τη σχεδίασή μας κατά το 2ο πέρασμα από τον DeCo.


Critical Path Comparison (Prime Time)
35
30
25
20
15
10
5
0
n
s
1,38
1,76
1,45
2,19
3,0
2,4
3,52
7,9
2,08
7,44
16,06
4,28
14,95
32,15
8,67
4 bit
8 bit
16 bit
32 bit
64 bit
Bit Number
Structional
Behavioral
Design Ware


T-D CP Structural (Prime Time)
16
14
12
10
8
6
4
2
0
n
s
1,38
0,97
2,19
2,31
3,52
2,5
7,45
2,63
14,96
9,02
4 bit
8 bit
16 bit
32 bit
64 bit
Bit Number
Tzimas
Dadaliaris

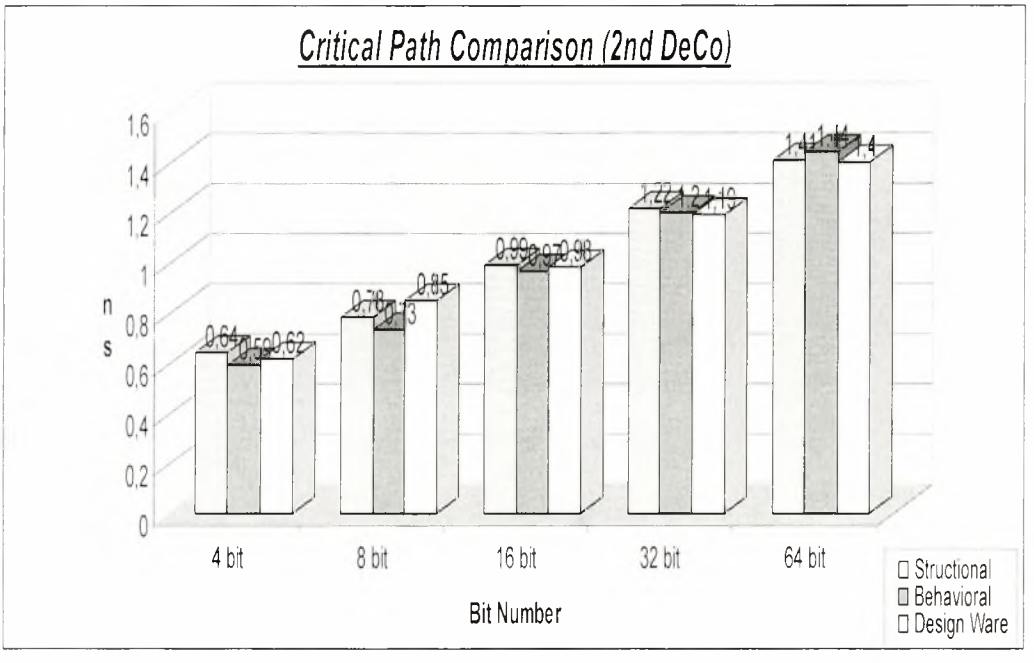


Η μηδενική τιμή που φαίνεται στα σχήματα παρατηρείται διότι σε παλαιότερη διπλωματική από την οποία έχουν παρθεί τα αποτελέσματα οι αθροιστές αυτού του μεγέθους δεν ενδιέφεραν την έρευνα που είχε διεξαχθεί και δεν πραγματοποιήθηκαν μετρήσεις για αυτούς.

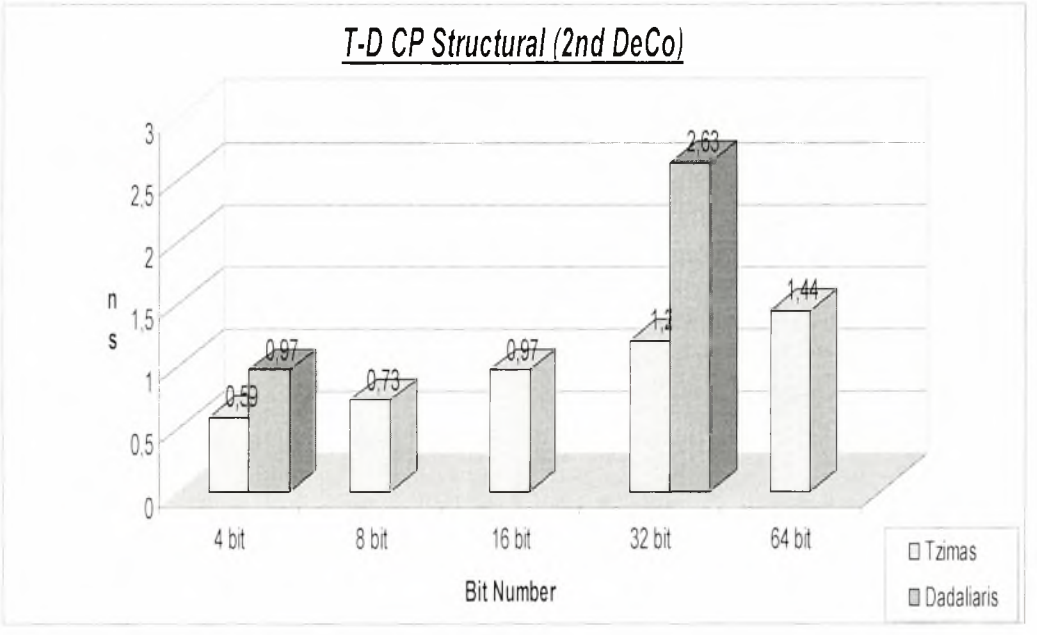


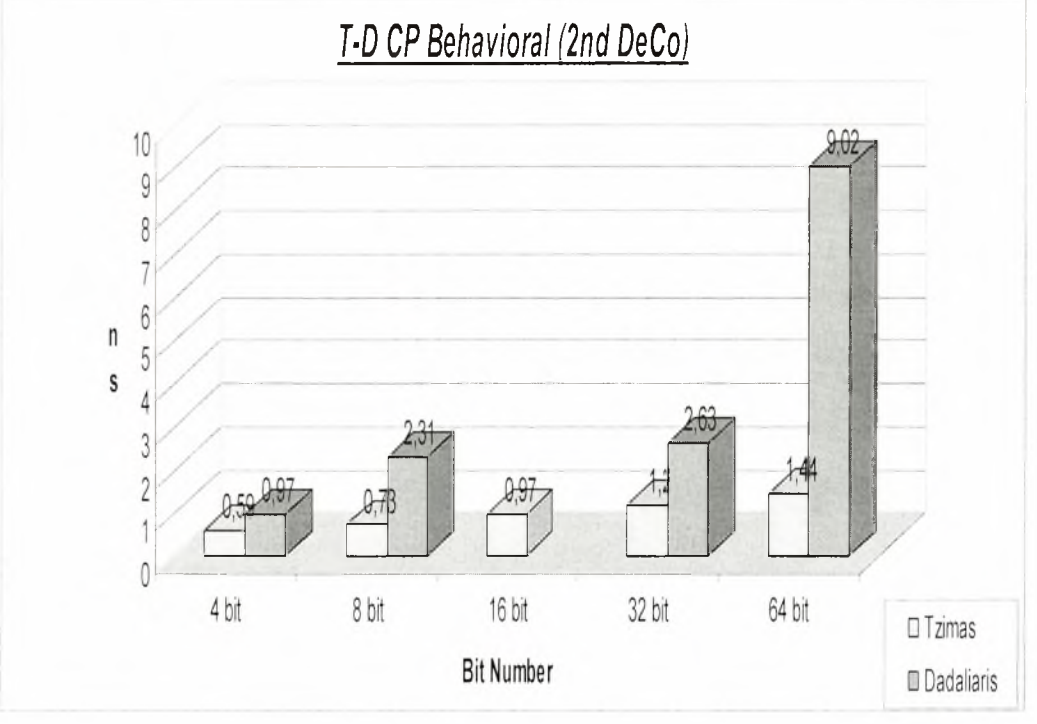
Σύγκριση Αθροιστών κατά το 2ο πέρασμα από DeCo (Design Compiler)

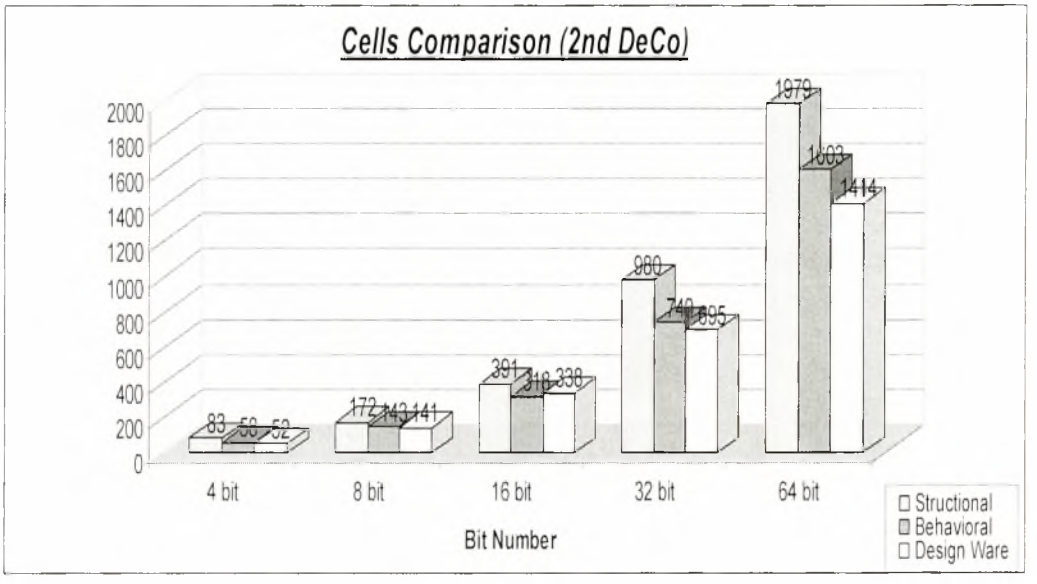
Μετά την σύνθεση παρατηρούμε την δραματική αλλαγή των αποτελεσμάτων που είχαμε στα προηγούμενα επίπεδα. Τώρα οι τιμές των μεγεθών χρονισμού είναι εφάμιλλες με αυτές της **Synopsys** και σε πολλές περιπτώσεις καλύτερες. Πιο συγκεκριμένα παρατίθονται τα παρακάτω γραφήματα.



Οι συγκριτικές μετρήσεις με τις μετρήσεις του παρελθόντος δείχνουν μια βελτίωση χρονισμού πάρα πολύ μεγάλη που φαίνεται παρακάτω.

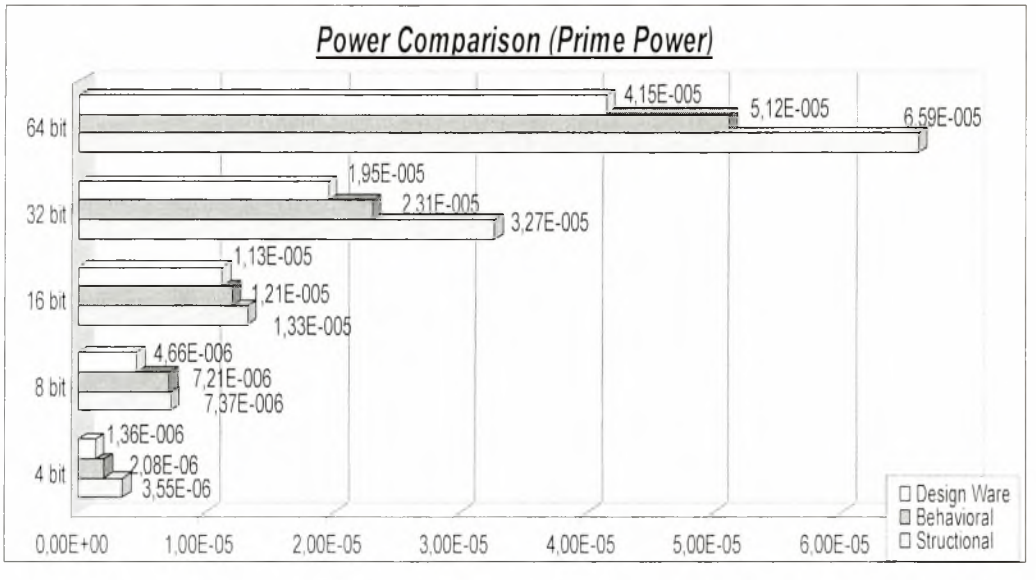

T-D CP Structural (2nd DeCo)
3
2,5
2
1,5
1
0,5
0
n
s
0,59
0,97
0,73
0,97
2,63
1,44
4 bit
8 bit
16 bit
32 bit
64 bit
Bit Number
Tzimas
Dadaliaris

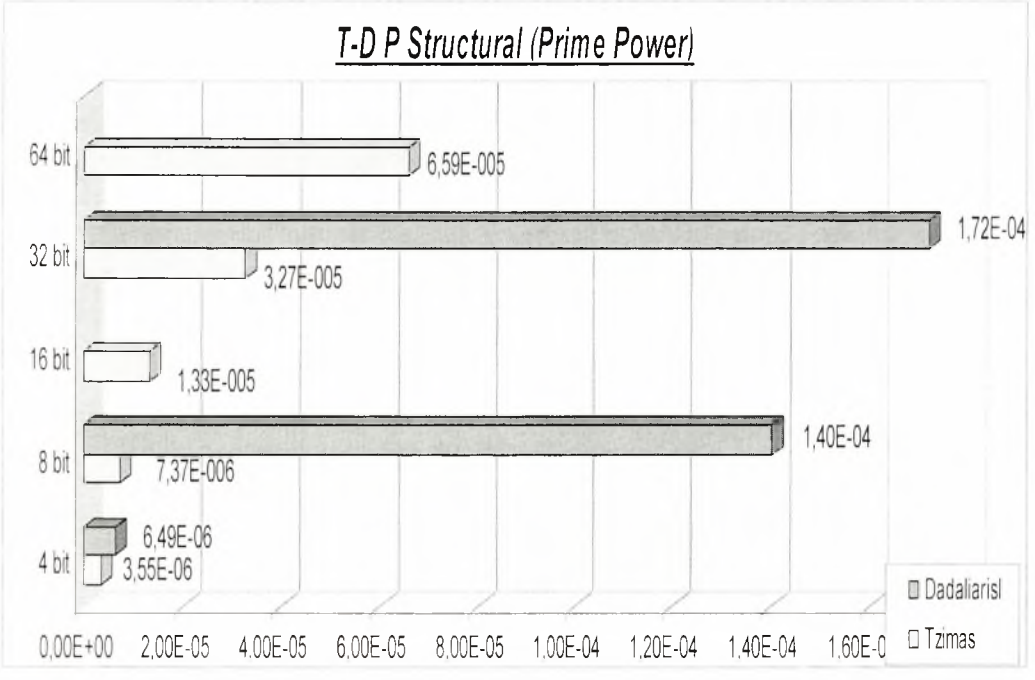

T-D CP Behavioral (2nd DeCo)
10
9
8
7
6
5
4
3
2
1
0
n
s
0,59
0,97
0,73
2,31
0,97
2,63
9,02
4 bit
8 bit
16 bit
32 bit
64 bit
Bit Number
Tzimas
Dadaliaris

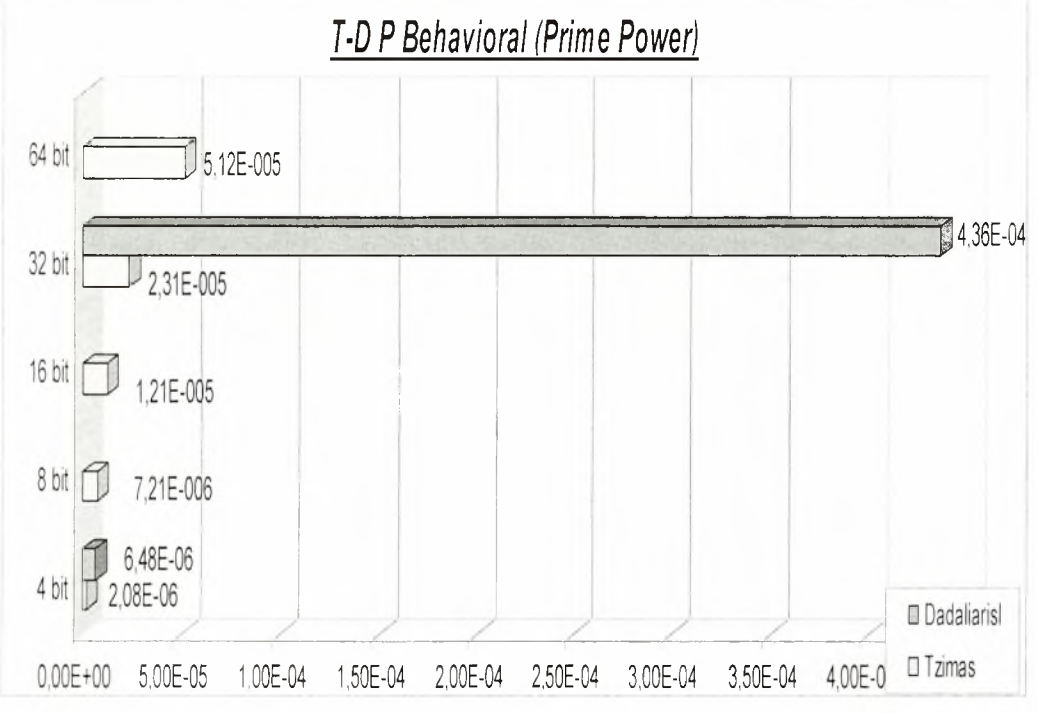


Σύγκριση βάσει της καταναλισκομένης ισχύος :

Ας εξετάσουμε, τώρα, τις διαφορές που παρουσιάζουν οι αθροιστές σχέση με την συνολική κατανάλωση ενέργειας του καθενός. Στα παρακάτω διαγράμματα παρουσιάζονται τα αποτελέσματα για το επίπεδο της καταναλισκόμενης ισχύος καθώς και συγκριτικά των μεθόδων.

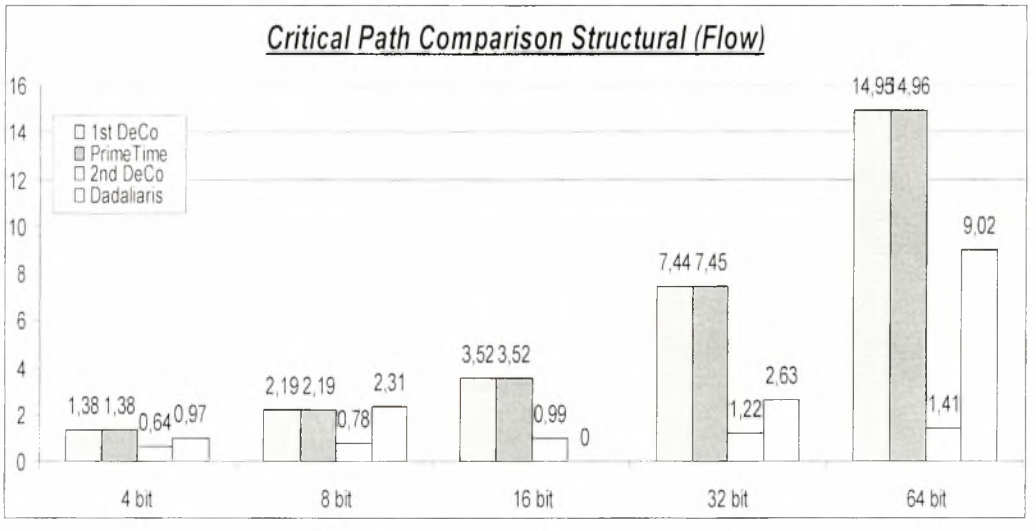


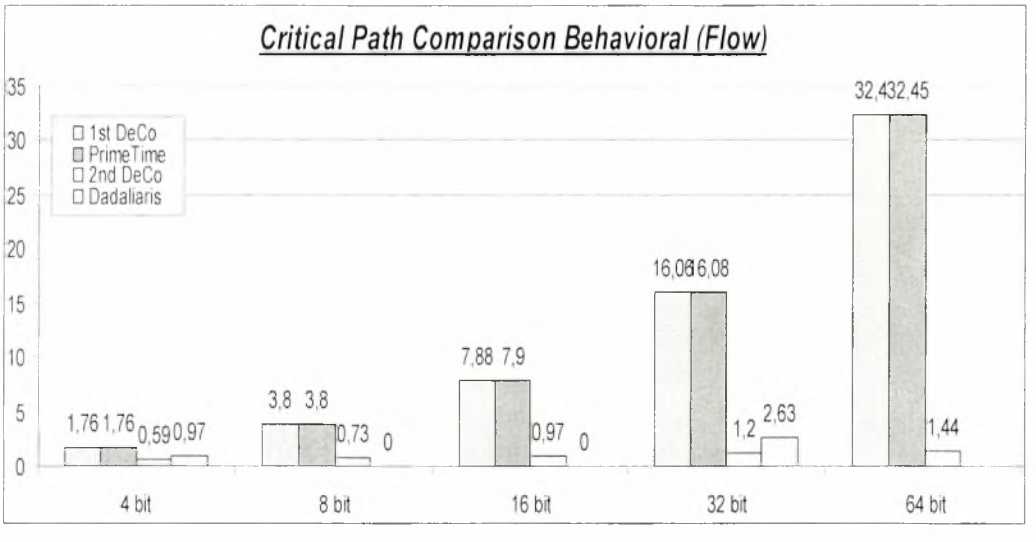

T-D P Structural (Prime Power)
64 bit
32 bit
16 bit
8 bit
4 bit
6,59E-005
1,72E-04
3,27E-005
1,33E-005
1,40E-04
7,37E-006
6,49E-06
3,55E-06
Dadaliarisl
Tzimas
0,00E+00
2,00E-05
4,00E-05
6,00E-05
8,00E-05
1,00E-04
1,20E-04
1,40E-04
1,60E-0

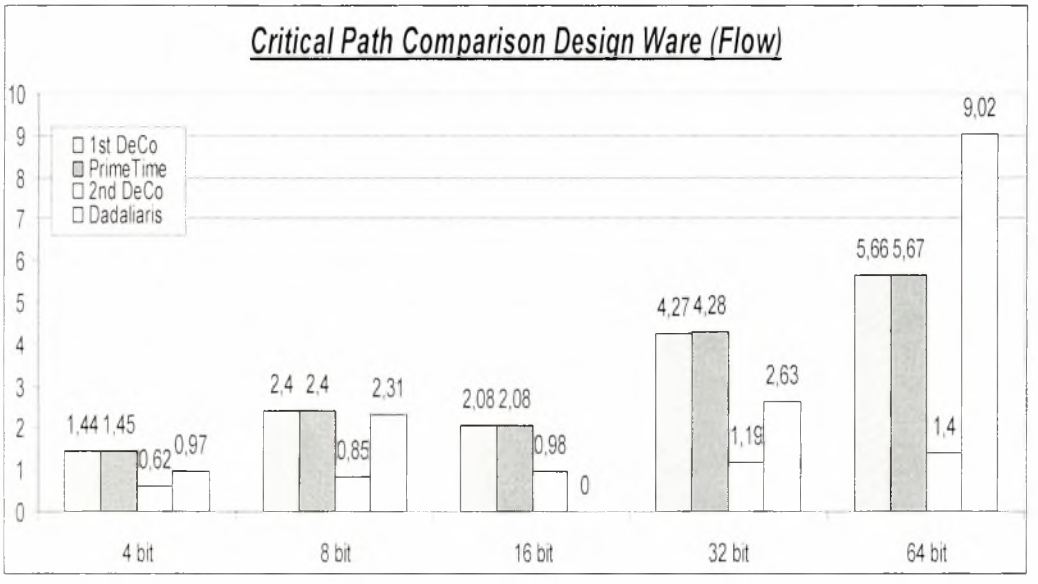

T-D P Behavioral (Prime Power)
64 bit
32 bit
16 bit
8 bit
4 bit
5,12E-005
4,36E-04
2,31E-005
1,21E-005
7,21E-006
6,48E-06
2,08E-06
Dadaliarisl
Tzimas
0,00E+00
5,00E-05
1,00E-04
1,50E-04
2,00E-04
2,50E-04
3,00E-04
3,50E-04
4,00E-0

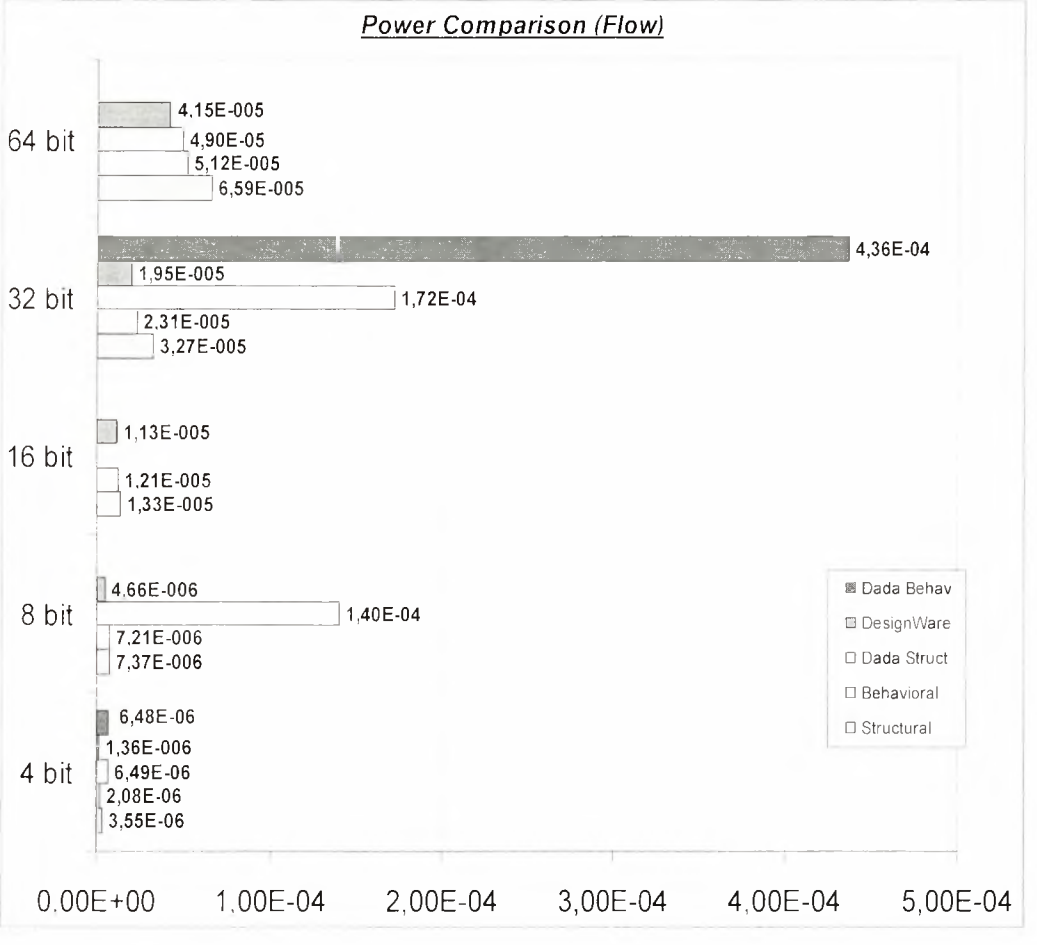
Σύγκριση κατά μήκος της ροής σχεδίασης:

Σε αυτό το σημείο αφού έχουμε εξετάσει ανά επίπεδο την κατανάλωση ισχύος των αθροιστών όσο και τον χρονισμό τους τόσο μεταξύ τους όσο και με παλαιότερες μεθόδους μέτρησης και βελτιστοποίησης ακολουθεί αναλυτική σύγκριση κατά την ροή σχεδίασης.






Critical Path Comparison Design Ware (Flow)
1st DeCo
PrimeTime
2nd DeCo
Dadaliaris
1,44 1,45
0,62
0,97
2,4 2,4
0,85
2,31
2,08 2,08
0,98
0
4,27 4,28
1,19
2,63
5,66 5,67
1,4
9,02
4 bit
8 bit
16 bit
32 bit
64 bit

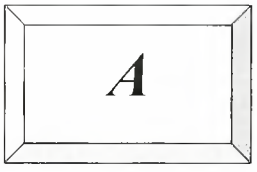

Power Comparison (Flow)
64 bit
4,15E-005
4,90E-05
5,12E-005
6,59E-005
32 bit
4,36E-04
1,95E-005
1,72E-04
2,31E-005
3,27E-005
16 bit
1,13E-005
1,21E-005
1,33E-005
8 bit
4,66E-006
1,40E-04
7,21E-006
7,37E-006
4 bit
6,48E-06
1,36E-006
6,49E-06
2,08E-06
3,55E-06
Dada Behav
DesignWare
Dada Struct
Behavioral
Structural
0,00E+00
1,00E-04
2,00E-04
3,00E-04
4,00E-04
5,00E-04

8.ΕΠΙΛΟΓΟΣ ΣΥΜΠΕΡΑΣΜΑΤΑ

Όπως είδαμε και στο προηγούμενο κεφάλαιο, καταλήξαμε σε ορισμένα χρήσιμα συμπεράσματα για την πλειοψηφία των αθροιστών που χρησιμοποιήθηκαν στις μετρήσεις αυτής της εργασίας. Γενικά παρατηρήσαμε πως η εξαγωγή των περιορισμών με την χρήση του **Prime Time** μας οδήγησε σε μεγάλες βελτιώσεις του σχεδιασμού σε επίπεδο χρονισμού και κατανάλωσης ισχύος, σε σύγκριση πάντα με την πάγια τεχνική αυθαίρετης ένθεσης περιορισμών που χρησιμοποιούνταν πιο πριν.

Σαφέστατα όμως μια σειρά από μετρήσεις σε ακόμα πιο ακραίες συνθήκες θα οδηγούσε σε ακόμα πιο ακριβή συμπεράσματα για την πραγματική συμπεριφορά των κυκλωμάτων.

Επίσης 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. Η προσθήκη νέων εργαλείων κατά την ροή των όπως τα **RailMill, PowerMill, PathMill, Arcadia, NanoSim, IC platform** ενδεχομένως οδηγήσουν σε μια πιο αναλυτική θεώρηση και εξαγωγή πρόσθετων συμπερασμάτων. Ο σκοπός της εργασίας, η ανάλυση και βελτιστοποίηση ψηφιακών κυκλωμάτων με τη χρήση ειδικού λογισμικού, καλύφθηκε σε ένα μεγάλο ποσοστό. Σαν περαιτέρω ενασχόληση θα ήταν ωφέλιμο να γίνουν πειραματικές μετρήσεις με άλλα εργαλεία για να συγκριθούν τα αποτελέσματα.



ΠΙΝΑΚΕΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

1ο Πέρασμα από Design Compiler

Carry Lookahead Structural					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,38	2,19	3,52	7,44	14,95
	1,88E+05	3,91E+08	8,10E+08	1,64E+09	3,32E+09
	()	()	()	()	()
	19	36	79	149	297

Carry Lookahead Behavioral					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,76	3,8	7,88	16,06	32,4
	1,66E+08	3,32E+08	6,64E+08	1,33E+09	2,65E+09
	()	()	()	()	()
	12	24	48	96	192

Carry Lookahead Design Ware					
datawidth	4 bit	8 bit	16 bit	32 bit	64 bit
	1,44	2,4	2,08	4,27	5,66
	1,59E+08	3,18E+08	7,50E+08	1,50E+09	3,27E+09
	()	()	()	()	()
	16	32	98	194	431

Πέρασμα από Prime Time

Carry Lookahead Structural					
	4 bit	8 bit	16 bit	32 bit	64 bit

	1,38	2,19	3,52	7,45	14,96
	1,88E+02	3,91E+02	8,10E+02	1,64E+03	3,32E+03
	()	()	()	()	()
	19	36	79	149	297

Carry Lookahead Behavioral					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,76	3,8	7,9	16,08	32,45
	1,66E+02	3,32E+02	6,64E+02	1,33E+03	2,65E+03
	()	()	()	()	()
	12	24	48	96	192

Carry Lookahead Design Ware					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,45	2,4	2,08	4,28	5,67
	158,98	317,95	749,95	1499,9	3272,84
	()	()	()	()	()
	16	32	98	194	431

2ο Πέρασμα από Design Compiler

Carry Lookahead Structural					
	4 bit	8 bit	16 bit	32 bit	64 bit
	0,64	0,78	0,99	1,22	1,41
	1,46E+09	3,05E+09	5,97E+09	1,50E+10	3,03E+10
	()	()	()	()	()
	83	172	391	980	1979

Carry Lookahead Behavioral					
	4 bit	8 bit	16 bit	32 bit	64 bit
	0,59	0,73	0,97	1,2	1,44
	8,93E+08	2,87E+09	5,29E+09	1,07E+10	2,37E+10
	()	()	()	()	()
	58	143	318	740	1603

Carry Lookahead Design Ware					
	4 bit	8 bit	16 bit	32 bit	64 bit
	0,62	0,85	0,98	1,19	1,4
	6,43E+08	2,09E+09	5,02E+09	9,49E+09	1,95E+10
	()	()	()	()	()
	52	141	338	695	1414

Πέρασμα από Prime Power

Carry Lookahead Structural					
	4 bit	8 bit	16 bit	32 bit	64 bit
	3,55E-06	7,37E-006	1,33E-005	3,27E-005	6,59E-005

Carry Lookahead Behavioral					
	4 bit	8 bit	16 bit	32 bit	64 bit
	2,08E-06	7,21E-006	1,21E-005	2,31E-005	5,12E-005

Carry Lookahead Design Ware					
	4 bit	8 bit	16 bit	32 bit	64 bit
	1,36E-006	4,66E-006	1,13E-005	1,95E-005	4,15E-005

Αντίστοιχα αποτελέσματα από Λαδαλιάρη Αντώνη

Carry Lookahead 1					
	4 bit	8 bit	16 bit	32 bit	(*)
	0,97	2,31	()	2,63	9,02
	172,8	929,6644	()	3476,725	3317,752
	6,49E-06	1,40E-04	()	1,72E-04	4,90E-05
	0,001087	0,301083	()	1,57456	1,47E+00

Carry Lookahead 2				
	4 bit	8 bit	16 bit	32 bit
	0,97	()	()	2,63
	172,8	()	()	3476,725
	6,48E-06	()	()	4,36E-04
	0,001086	()	()	3,990349

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, Y-2006.06.

[6]. **Design Compiler™**, Reference Manual, Y-2006.06.

[7]. **Design Compiler™**, User Guide, Y-2006.06.

[8]. **Design Compiler™**, Tutorial Using Design Vision, Y-2006.06.

[9]. **Design Analyzer™**, Reference Manual, Y-2006.06.

[10]. **Design Vision™**, User Guide, Y-2006.06.

[11]. **Power Compiler™**, User Guide, Y-2006.06.

[12]. **Power Compiler™**, Quick Reference Guide, Y-2006.06.

[13]. **PrimePower™**, Manual, Y-2006.06.

[14]. **PrimePower™**, Quick Reference Guide, Y-2006.06.

[15]. **Using TCL with Synopsys Tools**, Y-2006.06.

[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.

[23]. **PrimeTime™**, Manual, Y-2006.06.

[24]. **PrimeTime™**, Quick Reference Guide, Y-2006.06.

[25]. **PrimeTime™**, User Guide, Y-2006.06.

[26]. **PrimeTime™**, Reference Manual, Y-2006.06.

[27]. **PrimeTime™**, Tutorial Using Design Vision, Y-2006.06.

[28]. **PrimeTime™**, Tutorial, Y-2006.06.

[30].**PrimeTime™** PX User Guide, Y-2006.06.

[31]. **PrimeTime™** SI User Guide, Y-2006.06.

[32]. **PrimeTime™** User Guide (Advanced Timing Analysis) SI User Guide, Y-2006.06.

[33]. **PrimeTime™**User Guide (Fundamentals) , Y-2006.06.

Internet Sites :

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

[35]. http://www.sun.com
[36]. http://www.google.com