ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

“ΡΟΗ ΣΧΕΔΙΑΣΗΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΑΒΑΡΑΤΖΗΣ ΓΕΩΡΓΙΟΣ

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω των καθηγητή του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών και βασικό επιβλέποντα αυτής της Διπλωματικής εργασίας κ. Γ. Σταμούλη που μου έδωσε την ευκαιρία να πραγματοποιήσω αυτή τη μελέτη. Η υποστήριξή του και η αμέριστη συμπαράστασή του βοήθησαν στην έγκαιρη ολοκλήρωση αυτής της εργασίας.

Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον έτερο επιβλέποντα καθηγητή του τμήματος κ. Ν. Ευμορφόπουλο και τους συναδέλφους του γραφείου Ε5 για την προγενέστερη δουλειά τους πάνω στα ζητήματα με τα οποία καταπιάνεται η προκείμενη εργασία.

Τέλος, ιδιαίτερα θα ήθελα να ευχαριστήσω το διδάκτορα του τμήματος Αντώνιο Δαδαλιάρη, που μου εμπιστεύτηκε το υλικό της διατριβής του και μου έδωσε την ευκαιρία να συμβάλλω στην ερευνητική του μελέτη. Χωρίς την πολύτιμη βοήθειά του , η περάτωση της παρούσας εργασίας θα ήταν σχεδόν αδύνατη.

Περιεχόμενα

Πίνακας Εικόνων

Περίληψη

Η πολυπλοκότητα της σχεδίασης ψηφιακών ολοκληρωμένων κυκλωμάτων (Integrated Circuit Design - IC Design) έχει αυξηθεί κατακόρυφα τα τελευταία χρόνια. Οι σύγχρονες σχεδιάσεις αποτελούνται από δισεκατομμύρια τρανζίστορ και καλούνται να είναι πλήρως λειτουργικές καθ' όλη τη διάρκεια δράσης τους, καταναλώνοντας την μικρότερη δυνατή ενέργεια και επιτελώντας τις διεργασίες τους στο μικρότερο δυνατό χρονικό διάστημα. Καθίσταται σαφές, λοιπόν, πως τα χρόνια κατά τα οποία η σχεδίαση γινόταν εξ ολοκλήρου χειρονακτικά έχουν περάσει ανεπιστρεπτί.

Για την διευκόλυνση της σχεδίασης ψηφιακών κυκλωμάτων παρέχονται, πλέον, ολοκληρωμένες ροές σχεδίασης οι οποίες χρησιμοποιούν σε κάθε ένα από τα επιμέρους στάδιά τους ηλεκτρονικά εργαλεία αυτοματοποίησης (Electronic Design Automation tools - EDA tools) τα οποία είναι ικανά να πραγματοποιήσουν τις κατάλληλες διεργασίες πάνω σε σχεδιάσεις εκατομμυρίων πυλών.

Στην παρούσα εργασία παρουσιάζουμε μια βασική ροή σχεδίασης ολοκληρωμένων κυκλωμάτων, αναλύουμε το θεωρητικό υπόβαθρο πίσω από κάθε βήμα που ακολουθείται και πραγματοποιούμε εκτενείς αναφορές σε ορισμένα από τα σημαντικότερα εργαλεία που χρησιμοποιούνται αυτή την στιγμή στη βιομηχανία.

1.Εισαγωγή

Η σχεδίαση ολοκληρωμένων κυκλωμάτων αποτελεί κατ' ουσίαν ένα υποσύνολο του γνωστικού αντικειμένου των Ηλεκτρολόγων Μηχανικών. Η ψηφιακή σχεδίαση ολοκληρωμένων κυκλωμάτων παράγει, ως τελικό αποτέλεσμα, κυκλωματικά στοιχεία όπως μικροεπεξεργαστές (microprocessors), μνήμες (RAM memories, ROM memories, flash memories), FPGAs και ASICs (Application Specific Integrated Circuits). Τα σύγχρονα ολοκληρωμένα κυκλώματα (όπως οι τρέχοντες επεξεργαστές υπολογιστών) αποτελούνται από πάνω από ένα δισεκατομμύριο τρανζίστορ. Η πίεση της αγοράς για παραγωγή ολοκληρωμένων κυκλωμάτων σε συντομότερο χρονικό διάστημα, με μεγαλύτερες δυνατότητες, σε συνάρτηση με το γεγονός που προαναφέρθηκε έχει καταστήσει επιτακτική την χρήση EDA tools στην πλειοψηφία των βημάτων που ακολουθούνται κατά την σχεδίαση ενός ολοκληρωμένου κυκλώματος.

Η αυτοματοποιημένη ηλεκτρονική σχεδίαση είναι, πρακτικά, η υλοποίηση ενός κυκλώματος με παράλληλη χρήση ειδικών λογισμικών τα οποία έχουν δημιουργηθεί κατά περίσταση για την υποβοήθηση της διαδικασίας. Στην περίπτωση που εξετάζουμε, η εν σειρά χρήση μιας ομάδας εργαλείων για την παραγωγή της τελικής σχεδίασης, στοιχειοθετεί μια ροή σχεδίασης (design flow).

Οι βασικότεροι τομείς σχεδίασης για τους οποίους έχουν αναπτυχθεί κατάλληλα λογισμικά, τα οποία εμπίπτουν στην παραπάνω κατηγορία, είναι οι ακόλουθοι:

- DESIGN
    - High Level Synthesis
    - Logic Synthesis
    - Layout

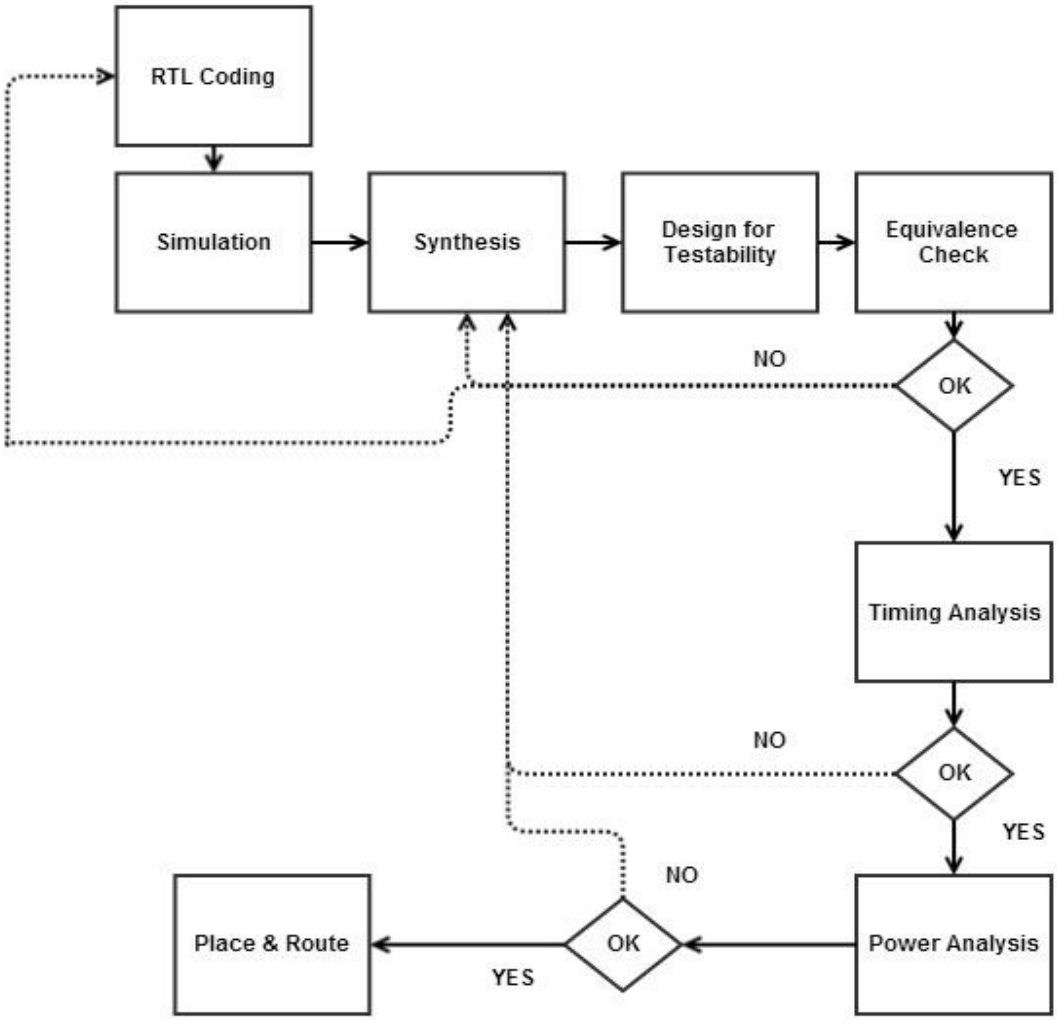
- SIMULATION
    - Logic Simulation
    - Behavioral Simulation
    - Hardware Emulation

- ANALYSIS & VERIFICATION
    - Functional Verification
    - Formal Verification
    - Equivalence Checking
    - Static Timing Analysis
    - Physical Verification

- MANUFACTURING
    - Mask Data Preparation
    - Mask Generation
    - Automatic Test Pattern Generation
    - Build-In Self-Test

2.Ροή Σχεδίασης Ολοκληρωμένων Κυκλωμάτων

Στην εικόνα που ακολουθεί παρουσιάζεται η ροή σχεδίασης που θα αναλύσουμε στην παρούσα διπλωματική εργασία. Όπως γίνεται εύκολα αντιληπτό, η ροή αυτή είναι δυναμική, και αυτό διότι παρόλο που τα ακολουθούμενα βήματα είναι διακριτά και εκτελούνται σειριακά, εντούτοις, υπάρχει η πιθανότητα βάσει των παραγόμενων αποτελεσμάτων να επιστρέψουμε σε προηγούμενες καταστάσεις προκειμένου να διορθώσουμε προβλήματα που έχουν παρουσιαστεί.



Εικόνα 2.1: Ροή Σχεδίασης Ψηφιακών Ολοκληρωμένων Κυκλωμάτων

Στα επόμενα κεφάλαια της παρούσας εργασίας θα γίνει εκτενής αναφορά σε κάθε ένα από τα παραπάνω βήματα και αναλυτική παρουσίαση των πιο διαδεδομένων εργαλείων που επιτελούν την εκάστοτε λειτουργία.

3.Περιγραφή Υλικού

Το πρώτο βήμα στη σχεδίαση ενός ολοκληρωμένου κυκλώματος, είναι η περιγραφή του κατά τέτοιο τρόπο ώστε να καθίσταται δυνατή η χρήση του από τα κατάλληλα εργαλεία. Οι γλώσσες που έχουν κυριαρχήσει στην αγορά τα τελευταία χρόνια και πλέον αποτελούν και πρότυπα περιγραφής υλικού είναι η VHDL και η Verilog. Στα επόμενα εδάφια θα γίνει μια συνοπτική παρουσίαση των βασικότερων χαρακτηριστικών τους.

3.1.Verilog

Η Verilog είναι μια γλώσσα περιγραφής υλικού (hardware description language - HDL) η οποία χρησιμοποιείται για την μοντελοποίηση ηλεκτρονικών συστημάτων. Χρησιμοποιείται, κυρίως, για την σχεδίαση και την επαλήθευση ψηφιακών κυκλωμάτων σε επίπεδο καταχωρητή (register transfer level - RTL), ενώ η χρήση της ενδείκνυται και για την επαλήθευση αναλογικών και υβριδικών σχεδιάσεων.

Η Verilog διαφέρει από τις συνήθεις γλώσσες προγραμματισμού στο ότι περιλαμβάνει τρόπους και δομές περιγραφής του τρόπου μεταβολής τιμών κατά την πάροδο του χρόνου. Στην γλώσσα περιλαμβάνονται δυο τελεστές ανάθεσης, ένας blocking τελεστής (=) και ένας non-blocking τελεστής. Ο non-blocking τελεστής επιτρέπει στους σχεδιαστές να περιγράφουν τις εκάστοτε ενημερώσεις των μηχανές καταστάσεων (state machines) δίχως να επιβάλλεται η δήλωση και η χρήση προσωρινών μεταβλητών. Η ύπαρξη των παραπάνω εννοιών, βοηθά τους σχεδιαστές να γράψουν γρήγορα περιγραφές κυκλωμάτων σε συμπαγή και συνοπτική μορφή.

Μια Verilog σχεδίαση αποτελείται από ένα σύνολο από δομικά στοιχεία (modules). Τα modules εμπεριέχουν την ιεραρχία της σχεδίασης και μπορούν να επικοινωνήσουνε με άλλα modules μέσω ενός συνόλου input, output και bidirectional ports. Εσωτερικά, ένα module μπορεί να περιέχει οποιονδήποτε συνδυασμό των παρακάτω στοιχείων: net declarations, variable declarations, σύγχρονα και ασύγχρονα statement blocks και στιγμιότυπα άλλων modules. Οι ακολουθιακές δηλώσεις τοποθετούνται εντός ενός block το οποίο ξεκινάει με την λέξη begin και τερματίζεται με

την λέξη end και εκτελούνται σειριακά εντός του block. Εν τούτοις, τα blocks εκτελούνται σε παραλληλία μεταξύ τους, χαρακτηριστικό που αποτελεί μεγάλο πλεονέκτημα της γλώσσας.

Τέλος, πρέπει να αναφέρουμε πως μόνο ένα υποσύνολο από τις δηλώσεις ενός κώδικα Verilog μπορεί να δοθεί ως είσοδος σε ένα εργαλείο σύνθεσης, γι αυτό και πρέπει να είμαστε ιδιαίτερα προσεκτική κατά την συγγραφή του κώδικα.

Στην εικόνα που ακολουθεί παρουσιάζεται ο κώδικας για την περιγραφή ενός πλήρους αθροιστή 1 bit.

```
`timescale 1ns / 1ps
module full_adder (
        input a,
        input b,
        input cin,
        output s,
        output cout );

       assign {cout,s} = a + b + cin;

endmodule
```

Εικόνα 3.1: Πλήρης Αθροιστής, περιγραφή στη γλώσσα υλικού Verilog

3.2.VHDL

Η VHDL (VHSIC Hardware Description Language) είναι μια γλώσσα περιγραφής υλικού η οποία χρησιμοποιείται για την περιγραφή ψηφιακών κυκλωμάτων. Η συγκεκριμένη γλώσσα μπορεί επίσης να χρησιμοποιηθεί και ως γλώσσα παράλληλου προγραμματισμού γενικού σκοπού .

Το βασικότερο πλεονέκτημα της VHDL, όταν αυτή χρησιμοποιείται για τον σχεδιασμό ολοκληρωμένων συστημάτων, είναι πως επιτρέπει στον σχεδιαστή να

πιστοποιήσει την ορθή συμπεριφορά του συστήματος πριν αυτό δοθεί για περεταίρω τροποποιήσεις στα εργαλεία σύνθεσης.

Ένα ακόμη πλεονέκτημα της VHDL είναι το γεγονός πως υποβοηθά στην σχεδίαση παράλληλων / σύγχρονων συστημάτων. Σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού, οι εντολές της προκείμενης γλώσσας εκτελούνται παράλληλα, γεγονός που καθιστά ιδιαίτερα εύκολη την περιγραφή ταυτόχρονων συστημάτων.

Επιπρόσθετα, μια ολοκληρωμένη σχεδίαση η οποία έχει δημιουργηθεί με χρήση της VHDL έχει το πλεονέκτημα της μεταφερσιμότητας (portability) καθώς μπορεί να ενσωματωθεί σε οποιαδήποτε VLSI τεχνολογία. Στην εικόνα που ακολουθεί παρουσιάζεται ο VHDL κώδικας για την περιγραφή ενός πλήρους αθροιστή 1 bit.

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FULL_ADDER is
      port( a, b, cin        : in  STD_LOGIC;
            sum, cout        : out STD_LOGIC );
end FULL_ADDER;

architecture BHV of FULL_ADDER is
begin

      sum <=  (not a and not b and cin) or
                  (not a and b and not cin) or
                  (a and not b and not cin) or
                  (a and b and cin);
       cout <= (not a and b and cin) or
                  (a and not b and cin) or
                  (a and b and not cin) or
                  (a and b and cin);
end BHV;
```

Εικόνα 3.2: Πλήρης Αθροιστής, περιγραφή στη γλώσσα υλικού VHDL

4.Προσομοίωση

Το πρώτο βήμα μιας τυπικής ροής σχεδίασης, όπως αυτή έχει παρουσιαστεί σε προηγούμενη ενότητα είναι η συγγραφή του κώδικα περιγραφής του κυκλώματος. Όταν ολοκληρωθεί το βήμα αυτό ο σχεδιαστής καλείται να ελέγξει την ορθή λειτουργία του κυκλώματος. Η διαδικασία αυτή ονομάζεται προσομοίωση και η επιτυχημένη ολοκλήρωσή της ελαχιστοποιεί την πιθανότητα παρουσίασης ανεπιθύμητων αποτελεσμάτων κατά την λειτουργία του κυκλώματος.

Ένα από τα πιο ευρέως διαδεδομένα εργαλεία προσομοίωσης της λειτουργίας ενός κυκλώματος είναι το ModelSim το οποίο και θα παρουσιάσουμε λεπτομερώς στην επόμενη ενότητα.

4.1. ModelSim

Το ModelSim μας παρέχει την δυνατότητα να ελέγξουμε την ορθότητα και χρηστικότητα του κώδικά μας όταν αυτός είναι γραμμένος σε κάποια γλώσσα περιγραφής υλικού όπως η Verilog ή η VHDL. Στο εδάφιο αυτό θα παρουσιάσουμε μια τυπική ροή εντολών η οποία μπορεί να ακολουθηθεί στην περίπτωση που έχουμε διαθέσιμο τόσο τον κώδικα της σχεδίασής μας όσο και το testbench της προκείμενης τοπολογίας, και θέλουμε να πιστοποιήσουμε την ορθή λειτουργία της σχεδίασης και να δημιουργήσουμε τα κατάλληλα αρχεία δραστηριότητας τα οποία δύναται να χρησιμοποιηθούν κατά την ανάλυση καταναλισκώμενης ισχύος:

1. Με την επιλογή File > Change Directory μεταφερόμαστε στον φάκελο στον οποίο είναι αποθηκευμένα τα αρχεία που περιέχουν την περιγραφή του κυκλώματος αλλά και του testbench που θα χρησιμοποιήσουμε.

2. Δημιουργούμε μια νέα βιβλιοθήκη με τις εντολές File > New > Library. Στο πλαίσιο που εμφανίζεται επιλέγουμε “a map to an existing library” και ακολούθως προσδιορίζουμε την τοποθεσία στην οποία βρίσκεται η βιβλιοθήκη βάσει της οποίας θέλουμε να γίνει η προσομοίωση της λειτουργίας του κυκλώματος.

3. Ακολούθως, επιλέγουμε Compile και παρουσιάζονται σε παραθυρικό περιβάλλον τα αρχεία που έχουμε αναφέρει παραπάνω. Αφού ελέγξουμε την εννοιολογική και συντακτική ορθότητα του αρχείου του κυκλώματος αλλά και του testbench αυτού είμαστε έτοιμοι για την κύρια φάση της εργασίας μας στην προσομοίωση.

4. Επιλέγοντας Simulate προσομοιώνουμε την λειτουργία του κυκλώματος με την βοήθεια του testbench και μας δίνεται η δυνατότητα αναπαράστασης της λειτουργίας του κυκλώματός μας με την χρήση κυματομορφών. Τέλος σε αυτή τη φάση της διαδικασίας μας δίνεται και η δυνατότητα δημιουργίας των VCD αρχείων η οποία θα αναλυθεί εκτενώς στην επόμενη παράγραφο.

4.1.1. VCD Files

Τα VCD (Value Change Dump) αρχεία, των οποίων η σημαντικότητα θα γίνει περισσότερο κατανοητή κατά το στάδιο υπολογισμού της καταναλισκώμενης ισχύος, είναι κατ' ουσίαν ASCII αρχεία τα οποία περιέχουν πληροφορίες για την σύνθεση του κυκλώματος, ορισμούς για τις μεταβλητές του και τις μεταβολές που υφίστανται οι τιμές των μεταβλητών.

Τα VCD αρχεία χρησιμοποιούνται κατά κύριο λόγο στις σχεδιάσεις που είναι γραμμένες σε Verilog, μια όμως από τις σημαντικότερες καινοτομίες που έχει εισάγει το ModelSim στις τελευταίες του εκδόσεις είναι η δυνατότητα χρήσης και υλοποίησής τους με βάση αρχεία τα οποία είναι γραμμένα σε VHDL.

Το ModelSim παρέχει την δυνατότητα επιλογής ανάμεσα σε δυο διαφορετικές ροές δημιουργίας VCD αρχείων. Η πιο διαδεδομένη από αυτές παράγει ένα four state VCD αρχείο, χωρίς πληροφορίες που αφορούν την δύναμη οδήγησης των πυλών. Η έτερη, παράγει ένα εκτενές αρχείο του τύπου αυτού με πληροφορίες τόσο για την δύναμη οδήγησης των πυλών όσο και για τα δεδομένα και την συμπεριφορά τους σε κάθε port της σχεδίασης.

Αφού, λοιπόν προσομοιώσουμε και ελέγξουμε την ορθή λειτουργία του testbench, δίνουμε τις ακόλουθες εντολές (στον χρόνο 0 της προσομοίωσης), οι οποίες και παράγουν το επιθυμητό αρχείο:

1. vcd file output.vcd
2. run “critical path + 10% critical path”
3. vcd add -r *
4. run 100
5. force a 0
6. run 100
7. force b 0
8. run 100
9. force cin 0
10. run 100
11. force a 1
12. run 100
13. force b 1
14. run 100
15. force cin 1
16. run 100
17. vcd checkpoint
18. quit -sim

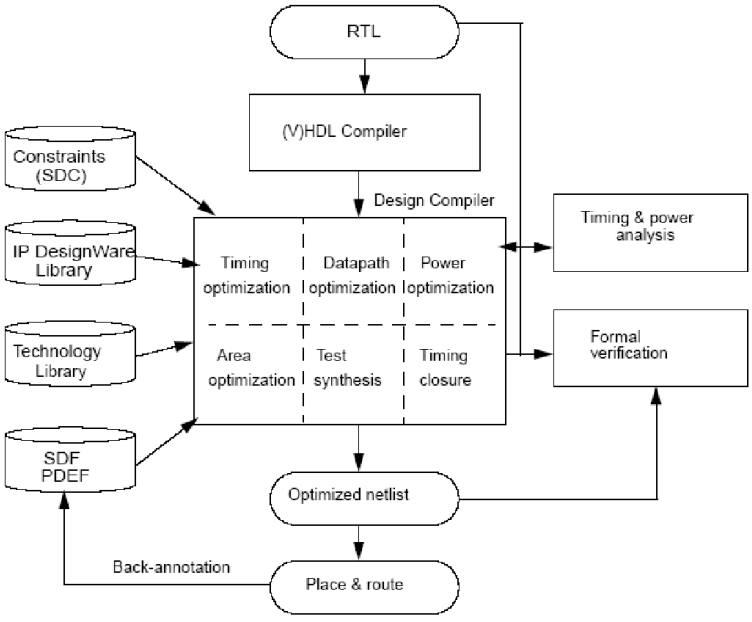
Στο παραπάνω παράδειγμα δημιουργίας VCD αρχείου, υποθέτουμε πως διαθέτουμε έναν πλήρη αθροιστή 1 bit με εισόδους “a” και “b” και κρατούμενο εισόδου “cin”. Η διαδικασία ξεκινάει την χρονική στιγμή 0 και με την πρώτη εντολή δίνουμε το επιθυμητό όνομα στο αρχείο που θα παραχθεί.

Η δεύτερη εντολή χρησιμοποιείται προκειμένου να ληφθούν υπόψη όλες οι μεταβολές που συμβαίνουν στις εισόδους του κυκλώματος. Ακολούθως, θέτουμε σε λειτουργία το κύκλωμα για ένα χρονικό διάστημα 100 ns καταγράφοντας τις απαραίτητες πληροφορίες. Η διαδικασία αυτή επαναλαμβάνεται στην συνέχεια αφού όμως δώσουμε τιμές σε ορισμένες ή και σε όλες τις μεταβλητές εισόδου. Στην περίπτωσή μας “εξαναγκάζουμε” τις μεταβλητές εισόδου να αποκτήσουν “ακραίες” τιμές (0 ή 1) προκειμένου να έχουμε όσο το δυνατόν μεγαλύτερο εύρος πληροφοριών.

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

5.Σύνθεση

Η Synopsys είναι μια από τις μεγαλύτερες EDA εταιρίες του χώρου. Ο πυρήνας του πακέτου σχεδίασης που προσφέρει η Synopsys αποτελείται από τον Design Compiler και τον Power Compiler. Ο Design Compiler (DC) είναι το λογισμικό που καλείται από το σύνολο των προγραμμάτων που προσφέρονται για την σύνθεση ψηφιακών κυκλωμάτων. Ο DC βελτιστοποιεί τις σχεδιάσεις με απώτερο στόχο την παροχή μικρότερων και γρηγορότερων αναπαραστάσεων μιας λογικής συνάρτησης. Περιέχει υποεργαλεία τα οποία συνθέτουν τις HDL σχεδιάσεις σε τεχνολογικά εξαρτημένες σχεδιάσεις σε επίπεδο πυλών. Υποστηρίζει την δυνατότητα βελτιστοποίησης είτε σε ιεραρχική είτε σε flat μορφή και δύναται να συνθέσει τόσο ακολουθιακά όσο και συνδυαστικά κυκλώματα βελτιώνοντας την ταχύτητα απόκρισής τους, τον χώρο που καταλαμβάνουν και την ισχύ που καταναλώνουν. Στην εικόνα που ακολουθεί παρουσιάζεται ο τρόπος με τον οποίο ο DC συμβάλλει στην συνολική ροή σχεδίασης:



Εικόνα 5.1: Συμβολή του Design Compiler στη συνολική ροή σχεδίασης

Στο σημείο αυτό θα δώσουμε και τον πρώτο “επίσημο” ορισμό της σύνθεσης ενός ψηφιακού κυκλώματος: “Σύνθεση ονομάζεται η διαδικασία κατά την οποία μετατρέπουμε μια σχεδίαση, η οποία μας δίνεται σε HDL κώδικα σε ένα βέλτιστο netlist το οποίο προσδιορίζεται πλήρως από μια τεχνολογική βιβλιοθήκη”.

Τα βήματα που ακολουθούμε κατά την διαδικασία της σύνθεσης ενός κυκλώματος σε άμεση αντιστοιχία με την παραπάνω εικόνα είναι τα ακόλουθα:

1. Ορίζουμε ως είσοδο τα αρχεία τα οποία περιγράφουν το κύκλωμά μας σε register transfer level, δηλαδή αρχεία σε κάποια γλώσσα περιγραφής υλικού.

2. Ο DC μεταφράζει την HDL περιγραφή σε συνθετικά στοιχεία της DesignWare βιβλιοθήκης, όπως για παράδειγμα οι αθροιστές και πιο αφαιρετικά boolean συνθετικά στοιχεία (τα οποία στο εξής θα αναφέρονται ως generic boolean components ή πιο απλά GTECH components). Τα GTECH components δεν έχουν πληροφορίες όσον αφορά τον χρονισμό και την δυνατότητα οδήγησης και δεν αντιστοιχούν σε κάποια τεχνολογική βιβλιοθήκη.

3. Αφού ολοκληρωθεί η μετατροπή του HDL κώδικα σε επίπεδο πυλών, ο DC βελτιστοποιεί την σχεδίαση και αντιστοιχίζει τα στοιχεία που την αποτελούν σε ένα συνδυασμό αποτελούμενο από συγκεκριμένα κελιά βιβλιοθηκών, βασιζόμενος στις επιλογές του χρήστη και τους περιορισμούς που αυτός έχει θέσει. Οι περιορισμοί αυτοί αποτελούν, στην ουσία, τις σχεδιαστικές απαιτήσεις του χρήστη για τους στόχους που θέλει να πετύχει με βάση την απόδοση της σχεδίασης, δηλαδή αναφέρονται στους χωρικούς και χρονικούς περιορισμούς υπό τους οποίους καλείται να επιτελεστεί η σύνθεση.

4. Αφού ολοκληρωθεί το παραπάνω στάδιο, ελέγχουμε το αποτέλεσμα της σύνθεσης για να δούμε εάν έχουν επιτευχθεί οι στόχοι που θέσαμε. Σε

περίπτωση που παρουσιαστεί οποιοδήποτε πρόβλημα ο χρήστης καλείται να το λύσει είτε αλλάζοντας τους περιορισμούς που έχει θέσει είτε τροποποιώντας την περιγραφή του κυκλώματος. Είναι σύνηθες πρακτική τα προβλήματα που εμφανίζονται να επιλύονται σε όσο το δυνατό πιο πρώιμο στάδιο του σχεδιαστικού κύκλου.

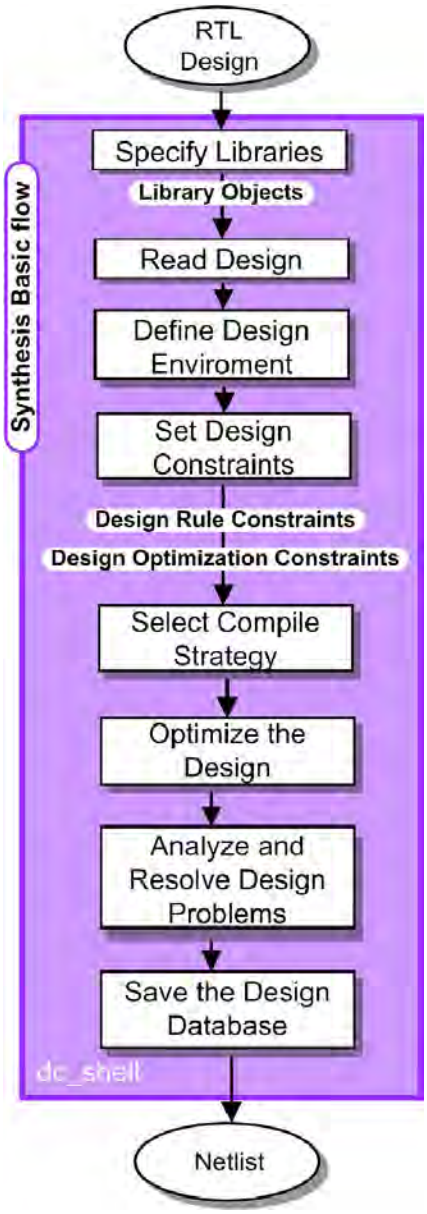
Ο DC διαβάζει και παράγει αρχεία που προσδιορίζουν σχεδιάσεις σε πολλαπλές μορφές που αποτελούν standards στην ηλεκτρονική σχεδίαση ψηφιακών κυκλωμάτων, συμπεριλαμβανομένων των αρχείων της Synopsys .db και .eqn. Επιπρόσθετα, ο DC προσφέρει άμεση σύνδεση με άλλα EDA tools. Αυτό μας δίνει την δυνατότητα να μεταφέρουμε με ευκολία από εργαλείο σε εργαλείο τους περιορισμούς και τα αποτελέσματα που έχουμε ανακτήσει μέχρι την προκείμενη στιγμή.

5.1. Design Vision

Το Design Vision είναι μια γραφική διεπαφή για το περιβάλλον σύνθεσης της Synopsys. Το γραφικό περιβάλλον του Design Vision περιέχει μενού επιλογών τα οποία μας δίνουν την δυνατότητα να εκτελέσουμε το μεγαλύτερο ποσοστό των εντολών που μπορεί να εκτελεστούν μέσα από το dc_shell. Με άλλα λόγια, το εργαλείο αυτό αποτελεί μια μετεξέλιξη του Design Analyzer το οποίο επίσης αποτελεί μια ακόμη γραφική διεπαφή που προσφέρεται για την επικοινωνία με τον πυρήνα των εργαλείων της Synopsys.

Στο σημείο αυτό πρέπει να αναφέρουμε πως όλες οι δυνατότητες του DC μπορούν να χρησιμοποιηθούν μέσα από την κονσόλα που μας προσφέρει το Design Vision, και αυτό συμβαίνει διότι τόσο το Design Vision όσο και ο Design Compiler χρησιμοποιούν την ίδια μηχανή στατικής ανάλυσης χρονισμού.

Η μεθοδολογία που ακολουθείται είναι παρόμοια με αυτή που έχουμε περιγράψει σε προηγούμενο εδάφιο και μπορεί να παρουσιαστεί υπό την μορφή διαγράμματος ως εξής:



Εικόνα 5.2: Ροή σύνθεσης με χρήση του Design Compiler

Στο κομμάτι που ακολουθεί θα παρουσιάσουμε βήμα προς βήμα την πορεία που ακολουθείται, βασιζόμενοι στο script που ακολουθεί και αποτελεί μια γενικευμένη εκδοχή των όσων χρησιμοποιήθηκαν:

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

Οι εντολές που παρουσιάστηκαν, αποτελούν ένα τυπικό παράδειγμα ενός script το οποίο οδηγεί κατά την εκτέλεσή του στην σύνθεση ενός ψηφιακού κυκλώματος και την αποθήκευση της νέας σχεδίασης στο δίσκο.

Με την πρώτη εντολή καθορίζουμε το σημείο στο δίσκο όπου είναι αποθηκευμένη η βιβλιοθήκη με βάση την οποία θα γίνει η σύνθεση και το σημείο όπου είναι αποθηκευμένη η περιγραφή του κυκλώματος σε κάποια γλώσσα περιγραφής υλικού.

Οι εντολές link_library και target_library που ακολουθούν, προσδιορίζουν την θέση όπου βρίσκονται οι τεχνολογικές βιβλιοθήκες που προσδιορίζουν τα κελιά και άλλες πληροφορίες ανάλογα με τον πάροχο της τεχνολογίας, όπως τα ονόματα των κελιών, οι σχεδιαστικοί κανόνες και οι συνθήκες λειτουργίας. Η παράμετρος

symbol_library καθορίζει τα σύμβολα που είναι διαθέσιμα για την δημιουργία των κατάλληλων σχηματικών.

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

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

- Read: Η εντολή αυτή χρησιμοποιείται όταν θέλουμε να ανακαλέσουμε σχεδιάσεις οι οποίες βρίσκονται ήδη σε .db μορφή.

Η εντολή analyze εξετάζει το HDL αρχείο προκειμένου να δει εάν βρίσκεται σε σωστή συντακτική λογική και αν είναι δυνατό να υποστεί την διαδικασία της σύνθεσης. Επιπρόσθετα, μεταφράζει τα αρχεία σε ένα ενδιάμεσο format το οποίο και τοποθετεί στο φάκελο που έχει προσδιοριστεί ως φάκελος εργασίας.

Η εντολή elaborate εξετάζει το “ενδιάμεσο” αρχείο και στην συνέχεια παράγει την .db αναπαράσταση της σχεδίασης. Κατά την διάρκεια αυτής της διαδικασίας το πρόγραμμα καθορίζει ποια από τα στοιχεία της σχεδίασης πρέπει να αντικατασταθούν από συνθετικά στοιχεία της βιβλιοθήκης που έχουμε επιλέξει.

Η εντολή read αποτελεί ένα συνδυασμό των παραπάνω εντολών, δεν πραγματοποιεί όμως όσους συντακτικούς ελέγχους εκτελούν οι δυο προηγούμενες εντολές.

Ακολούθως, καλούμαστε να καθορίσουμε το περιβάλλον της σχεδίασης και τους περιορισμούς/παραμέτρους της σχεδίασης. Ο DC απαιτεί την μοντελοποίηση του σχεδιαστικού περιβάλλοντος του κυκλώματος ώστε να προχωρήσει η διαδικασία της

σύνθεσης. Με το μοντέλο αυτό καθορίζουμε τις εξωτερικές συνθήκες λειτουργίας (manufacturing process, temperature, voltage), loads, drives, fanouts και wire load models. Ο καθορισμός όλων αυτών των παραμέτρων επηρεάζει άμεσα την σύνθεση του κυκλώματος και τα αποτελέσματα των βελτιστοποιήσεων που εφαρμόζει ο DC.

Στην συνέχεια με την εντολή write_parasitics αποθηκεύουμε πληροφορίες για τα παρασιτικά που παρατηρούνται στο κύκλωμα σε μορφή .spef αρχείου, προκειμένου να χρησιμοποιήσουμε τις πληροφορίες αυτές κατά τον υπολογισμό της καταναλισκώμενης ισχύος.

Τέλος, με την χρήση της εντολής report και μιας σειράς παραμέτρων λαμβάνουμε πληροφορίες για την σχεδίαση, τα κελιά τον χώρο που καταλαμβάνει το κύκλωμα αλλά και το μέγιστο, χρονικά, 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.

5.2. DesignWare Libraries

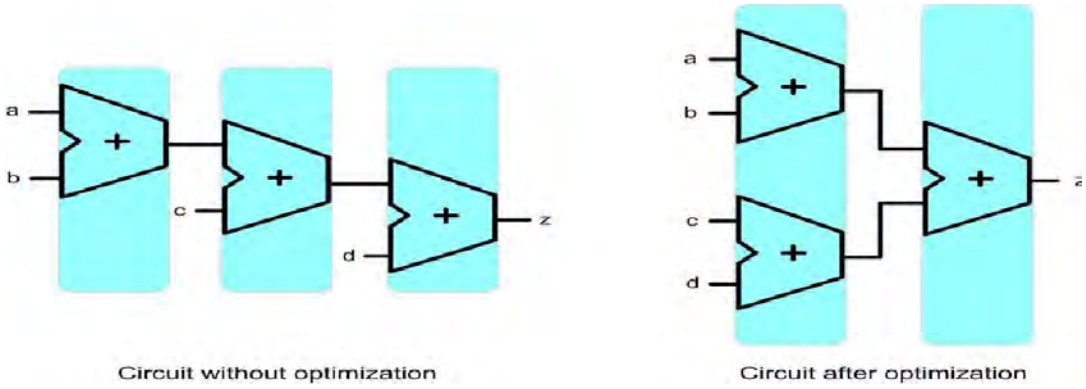
Οι DesignWare βιβλιοθήκες προσφέρουν components που είναι τεχνολογικά ανεξάρτητα, αποτελούν δηλαδή, δομικά block σε επίπεδο μικροαρχιτεκτονικής τα οποία είναι πλήρως ενοποιημένα με το περιβάλλον της σύνθεσης του πακέτου εργαλείων της Synopsys. Είναι βιβλιοθήκες που αναπτύσσονται από την ίδια την εταιρία και χωρίζονται σε δυο υποκατηγορίες:

- Foundation Library

- Digital Signal Processing (DSP) Library

Η επαναχρησιμοποίηση κυκλωμάτων είναι ιδιαίτερα επιθυμητή στην διαδικασία της σύνθεσης. Η βασική ροή του DC χρησιμοποιεί τις Foundation και GTECH βιβλιοθήκες. Η Foundation βιβλιοθήκη είναι μια συλλογή από επαναχρησιμοποιήσιμα, συνθέσιμα δομικά στοιχεία που είναι πλήρως ενοποιημένα με το περιβάλλον της

Synopsys, τα οποία προσφέρονται για την σύνθεση ψηφιακών κυκλωμάτων. Η Foundation βιβλιοθήκη, μας δίνει επιπλέον την δυνατότητα εκτέλεσης βελτιστοποιήσεων υψηλού βαθμού μέσω της χρήσης κατάλληλων εργαλείων σύνθεσης. Ως παράδειγμα θα αναφέρουμε την περίπτωση στην οποία μέσα σε ένα αρχείο το οποίο περιγράφει μια σχεδίαση, εντοπιστεί ο τελεστής πρόσθεσης “+”. Στην περίπτωση 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 αναπαράσταση της συνάρτησης (arithmetic), στον καταμερισμό πόρων (resource sharing) και στην αντιμετάθεση ακίδων (pin permutation).



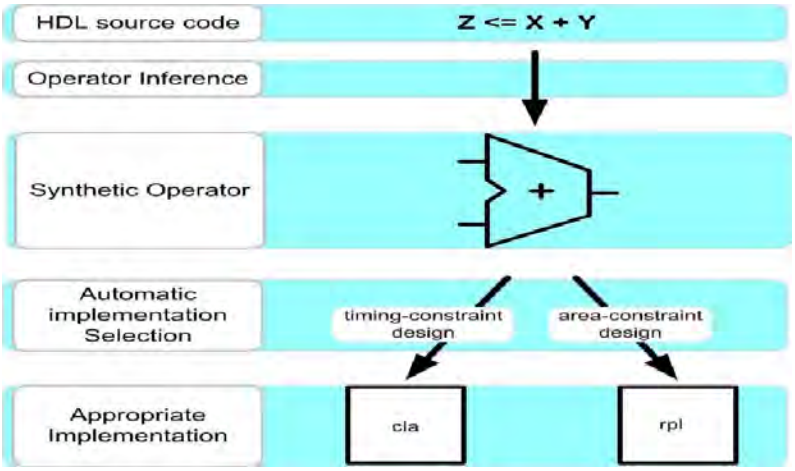
Εικόνα 5.3: Αριθμητικές βελτιστοποίησης του Design Compiler

Η βελτιστοποίηση της αριθμητικής χρησιμοποιεί τους κανόνες της άλγεβρας για να βελτιστοποιήσει το μέγεθος και την απόδοση της σχεδίασης με τον επαναπροσδιορισμό της θέσης των πράξεων. Για παράδειγμα, η μαθηματική έκφραση a+b+c+d περιγράφει τρία επίπεδα διαδοχικών πράξεων πρόσθεσης όπου οι μεταβλητές προσθέτονται σε κάθε στάδιο ανά ζεύγη. Με βελτιστοποίηση της αριθμητικής μπορούμε να διατάξουμε τις πράξεις ως εξής: (a+b)+(c+d). Ο νέος αυτός τρόπος αναπαράστασης

θα προσφέρει μεγαλύτερη ταχύτητα λόγω της μείωσης των επιπέδων λογικής που χρησιμοποιούνται. Στο παραπάνω σχήμα παρουσιάζεται μια αναπαράσταση της αναδιάταξης των πράξεων.

Ο καταμερισμός πόρων επιτρέπει ίδιες λειτουργίες που δεν επικαλύπτονται χρονικά να εκτελούνται από το ίδιο φυσικό hardware.

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 σχεδίασή μας. Ένα απλό και πρακτικό παράδειγμα της λειτουργίας που μόλις περιγράψαμε, το οποίο βασίζεται στην αριθμητική πράξη της πρόσθεσης, είναι η υλοποίηση της μη προσημασμένης πρόσθεσης με την χρήση είτε του carry lookahead adder είτε με χρήση της carry ripple τοπολογίας. Η επιλογή ανάμεσα στις δυο αυτές δυνατές περιπτώσεις θα γίνει από το εργαλείο σύνθεσης βάσει των παραμέτρων που έχει θέσει ο χρήστης σε προηγούμενα στάδια της ροής σχεδίασης. Η διαδικασία αυτή παρουσιάζεται στην επόμενη εικόνα:

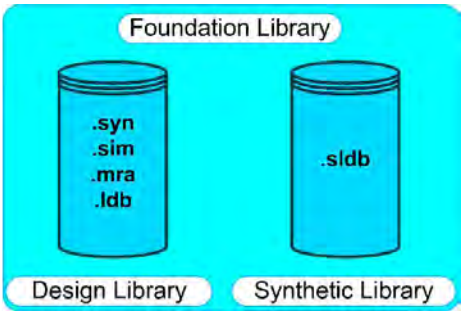


Εικόνα 5.4: Επιλογή υλοποίησης βάσει κριτηρίων βελτιστοποίησης

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

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

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



Εικόνα 5.5: Foundation Library

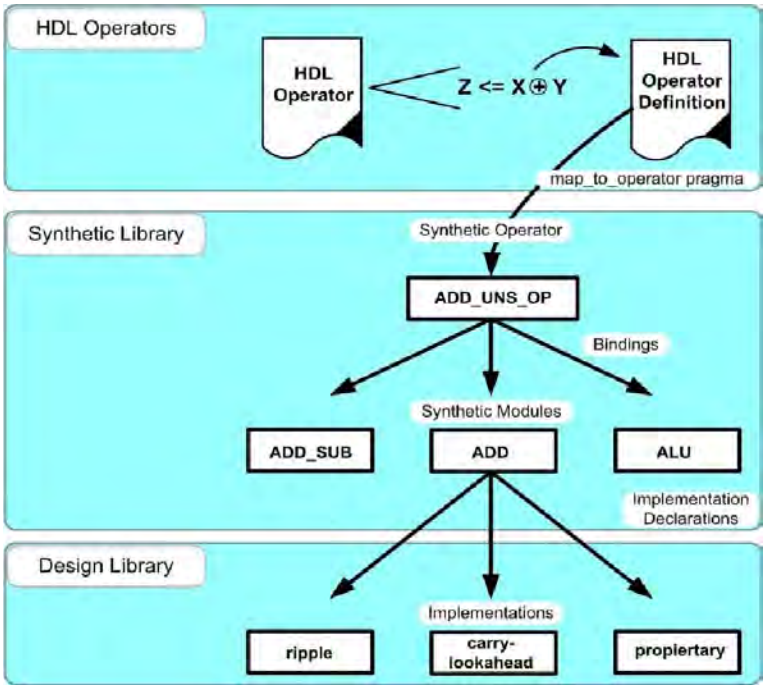
Οι περιγραφές των κυκλωμάτων σε μια Design Library είναι αποθηκευμένες σε δυαδικά αρχεία που είναι άμεσα χρησιμοποιήσιμα από τα εργαλεία της Synopsys. Οι περιγραφές αυτές μπορεί να είναι netlists συγκεκριμένης τεχνολογίας ή hard macros που δεν θα επηρεάζονται από την διαδικασία της σύνθεσης, ή ακόμη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

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

Η σύνδεση μεταξύ του πηγαίου κώδικα, της Synthetic Library και της Design Library γίνεται με τη χρήση μιας ιεραρχίας μοντέλων. Έτσι, οι HDL operators συνδέονται με τους synthetic operators οι οποίοι με την σειρά τους σχετίζονται με τα synthetic

modules. Κάθε synthetic module δύναται να έχει πολλαπλές υλοποιήσεις. Στο σχήμα που ακολουθεί παρουσιάζονται οι μεταβάσεις μέχρι την επιλογή της καταλληλότερης υλοποίησης.

Οι HDL operators είναι δομικά στοιχεία μιας γλώσσας περιγραφής υλικού 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, ένα map_to_operator_pragma το οποίο συνδέει τον HDL operator με τον κατάλληλο synthetic operator. Πολλοί HDL operators, συμπεριλαμβανομένων και των built-in infix operators, συνδέονται χωρίς κάποιες ειδικές δηλώσεις με την Synopsys Standard Synthetic Library standard.sldb.



Εικόνα 5.6: Synthetic Library

Η Synthetic Library περιέχει ορισμούς για τους synthetic operators, για τα synthetic modules και bindings. Επίσης, περιέχει δηλώσεις που συνδέουν τα synthetic modules με τις υλοποιήσεις τους. Οι υλοποιήσεις βρίσκονται στις ανάλογες Design Libraries. Σε μια Synthetic Library, λοιπόν, μπορούμε να βρούμε πληροφορίες για:

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

- Synthetic Module: Ορίζουν ένα κοινό interface για μια οικογένεια υλοποιήσεων. Όλες οι υλοποιήσεις ενός δεδομένου module έχουν τα ίδια ports και την ίδια συμπεριφορά εισόδου-εξόδου.

- Bindings: Συνδέουν τους synthetic operators με τα synthetic modules. Για παράδειγμα, ένα binding συνδέει τον synthetic operator της πρόσθεσης με ένα adder module. Ένας ή περισσότεροι synthetic operators μπορούν να συνδεθούν με ένα συγκεκριμένο synthetic module και κάθε τελεστής μπορεί να συνδεθεί με ένα ή περισσότερα modules.

- Implementation Declarations: Συνδέει τα 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 synthetic module και implementation είναι παραπλήσιες των εννοιών entity και architecture της VHDL. Μια υλοποίηση δύναται να ειδωθεί σαν μια αρχιτεκτονική πραγματοποίηση ενός

synthetic module. Ένα implementation μπορεί να είναι οτιδήποτε, από ένα netlist συγκεκριμένης τεχνολογίας έως και μια synthesizable rtl-level περιγραφή σχεδίασης.

6. Σχεδιασμός με σκοπό την Δοκιμή

Με τον όρο Design for Testability (DFT) εννοούμε εκείνο το σύνολο των σχεδιαστικών τεχνικών που χρησιμοποιούνται στην βιομηχανία προκειμένου να εμπλουτίσουν τις δυνατότητες ελέγχου μιας σχεδίασης. Ο στόχος της εφαρμογής αυτών των τεχνικών σε μια σχεδίαση είναι να καταστεί δυνατός ο έλεγχος της ορθής λειτουργίας του υλικού μιας σχεδίασης πριν από την δημιουργία του τελικού προϊόντος.

Η πιο συνήθης μέθοδος που χρησιμοποιείται για την μεταφορά δεδομένων ελέγχου (test data) από τις εισόδους της σχεδίασης, στο υπό εξέταση κύκλωμα και την εξέταση της ορθότητας του τελικού αποτελέσματος όπως αυτό παρατηρείται στις εξόδους του κυκλώματος, ονομάζεται scan design. Σύμφωνα με την μέθοδο αυτή όλοι οι registers του κυκλώματος ενώνονται σε μια ή περισσότερες αλυσίδες οι οποίες χρησιμοποιούνται για να αποκτήσει πρόσβαση ο σχεδιαστής στους εσωτερικούς κόμβους του κυκλώματος. Αφού δημιουργηθεί η αλυσίδα που μόλις περιγράψαμε μία σειρά “προτύπων” δεδομενων (data patterns) μετακινείται εντός του κυκλώματος εώς ότου φτάσει στην έξοδό του, όπου τα τελικά αποτελέσματα συγκρίνονται από τον σχεδιαστή έναντι κάποιων προϋπολογισμένων “καλών” αποτελεσμάτων.

Η αλυσίδα που μόλις περιγράψαμε ονομάζεται στην σύγχρονη βιβλιογραφία αλυσίδα σάρωσης (scan chain) και η βασική της δομή περιέχει τα ακόλουθα σήματα που υποβοηθούν στην παρακολούθηση του συνολικού μηχανισμού ελέγχου:

- scan_in: το σήμα χρησιμοποιείται για τον ορισμό της εισόδου του scan chain.
- scan_out: το σήμα χρησιμοποιείται για τον καθορισμό της εξόδου του scan chain.
- scan_enable: το συγκεκριμένο σήμα προστίθεται στην σχεδίαση προκειμένου να δημιουργηθεί ένας άτυπος shift register ο οποίος θα ενώνει όλους τους registers της σχεδίασης.

- clock_signal: το προκείμενο σήμα χρησιμοποιείται προκειμένου να ελέγξει όλους τους registers της αλυσίδας κατά την φάση της μετακίνησης των πειραματικών δεδομένων αλλά και κατά τη φάση της αποθήκευσης τους στην έξοδο της αλυσίδας.

Τα σημαντικότερα πλεονεκτήματα που παρουσιάζει η χρήση της προαναφερθείσας τεχνικής κατά τον σχεδιασμό ενός κυκλώματος είναι τα ακόλουθα:

- αυξάνει το ποσοστό των λαθών που μπορεί να ανιχνευθούν.

- μειώνει τον συνολικό χρόνο ελέγχου του κυκλώματος.

- υποστηρίζει τον παράλληλο έλεγχο πολλαπλών διασυνδέσεων εντός του κυκλώματος.

- μειώνει την απαιτούμενη επεξεργαστική ισχύ για τον έλεγχο του κυκλώματος.

Το εργαλείο της Synopsys που δύναται να επιτελέσει την παραπάνω λειτουργία ονομάζεται DFT Compiler, είναι μέρος του Design Compiler και ένα τυπικό script που μπορεί να χρησιμοποιηθεί απευθείας από την κονσόλα του εργαλείου είναι το ακόλουθο:

1. remove_design -all
2. set test_default_scan_style
3. read_verilog full_adder.v
4. source constraints.sdc
5. set_dft_signal -type ScanMasterClock -port clk
6. set_dft_signal -type ScanIn -port a
7. set_dft_signal -type ScanOut -port b
8. create_test_protocol
9. dft_drc
10. write_test_protocol -out full_adder_test.spf

11. create_clock clk -period 100
12. compile -scan
13. link
14. read_test_protocol full_adder_test.spf
15. set_scan_configuration -chain_count 1
16. preview_dft
17. insert_dft
18. compile -scan -incremental
19. dft_drc -coverage_estimate
20. write -format verilog -o full_adder_scan.v
21. write_sdf full_adder.sdf

Με την χρήση της πρώτης εντολής, αφαιρούμε από την μνήμη του εργαλείου όσες σχεδιάσεις έχουν φορτωθεί μέχρι αυτό το σημείο. Ακολουθεί η εντολή set_default_scan_style με την οποία δηλώνουμε πως ο τρόπος με τον οποίος θα ελεγχθεί το κύκλωμα είναι ο τυπικός, όπως αυτός έχει παρουσιαστεί παραπάνω. Υπάρχουν πολλές DFT τεχνικές, η πιο βασική, όμως, και ευρέως χρησιμοποιούμενη είναι αυτή που παραθέσαμε σε προηγούμενο εδάφιο.

Ακολούθως, διαβάζουμε στη μνήμη του εργαλείου δύο αρχεία, το πρώτο από αυτά είναι το αρχείο που προέκυψε από την διαδικασία της σύνθεσης και το δεύτερο περιλαμβάνει όλους τους περιορισμού που έχει θέσει ο σχεδιαστής (τόσο χρονικούς όσο και χωρικούς). Το δεύτερο αρχείο είναι της μορφής .sdc (Synopsys Design Constraint) και αποτελεί πλέον πρότυπο για την μεταφορά των σχεδιαστικών περιορισμών μεταξύ των EDA tools.

Οι τρεις επόμενες εντολές καθορίζουν τα σήματα του scan chain που στην περίπτωσή μας ταυτίζονται με τα σήματα του ρολογιού, της εισόδου και της εξόδου της σχεδίασης αντίστοιχα.

Η εντολή dft_drc ελέγχει το κατά πόσο ακολουθούνται οι σχεδιαστικού κανόνες της τεχνολογικής βιβλιοθήκης που έχουμε χρησιμοποιήσει. Σε περίπτωση που αυτός ο έλεγχος αναφέρει προβλήματα, ο σχεδιαστής καλείται να τα λύσει πριν να συνεχίσει με τα επόμενα βήματα της διαδικασίας. Η ύπαρξη λαθών καθιστά σίγουρη την παρουσία σφαλμάτων κατά την διαδικασία του manufacturing.

Με την εντολή create_clock δημιουργούμε το ρολόι της σχεδίασης και με την εντολή compile -scan λέμε στον DFT compiler να προχωρήσει στην αντικατάσταση όλων των flip-flop της σχεδίασης από flip-flop τα οποία θα περιλαμβάνουν σήματα ελέγχου, τα οποία θα ενωθούν στα επόμενα βήματα προκειμένου να σχηματίσουν την αλυσίδα σάρωσης.

Η εντολή set_scan_configuration -chain_count 1, δηλώνει τον αριθμό των συνολικών αλυσίδων που θα σχηματιστούν, που στην προκειμένη περίπτωση θα είναι μία, και η εντολή insert_dft δημιουργεί αυτή την αλυσίδα.

Ακολουθεί ένα ακόμη incremental compilation, ώστε να προκύψει ένα ενημερωμένο netlist, μετά το πέρας της διαδικασίας του scan insertion και η εντολή dft_drc -coverage_estimate η οποία μας ενημερώνει για το ποσοστό των πιθανών λαθών που μπορούν να εντοπιστούν με βάση το scan chain που έχει δημιουργηθεί (το ποσοστό αυτό πρέπει να είναι αρκούντως υψηλό).

Οι δυο τελευταίες εντολές χρησιμοποιούνται έτσι ώστε να δημιουργηθεί το αναθεωρημένο netlist και να παραχθεί ένα .sdc (Standard Delay Format) αρχείο το οποίο προσδιορίζει τις καθυστερήσεις που παρουσιάζουν όλα τα μονοπάτια της σχεδίασης και θα χρησιμοποιηθεί κατά την ανάλυση χρονισμού.

7.Έλεγχος Ισοδυναμίας

Όταν ολοκληρωθεί η διαδικασία του DFT και πριν ξεκινήσει η ανάλυση χρονισμού και καταναλισκομένης ισχύος, ο σχεδιαστής καλείται να ελέγξει την ισοδυναμία της αρχικής και της παρούσας μορφής του κυκλώματος. Ο έλεγχος αυτός είναι απαραίτητος προκειμένου να διαπιστώσουμε το κατά πόσο τα εργαλεία που χρησιμοποιήθηκαν ακολούθησαν τους περιορισμούς που τέθηκαν χωρίς να αλλάξουν την λειτουργικότητα του κυκλώματος.

Ένα εργαλείο που μπορεί να επιτελέσει τον παραπάνω έλεγχο και το οποίο χρησιμοποιείται ευρέως είναι το Formality της Synopsys. Όπως και στα υπόλοιπα εργαλεία που έχουν παρουσιαστεί μέχρι τώρα, δίνεται η δυνατότητα στον σχεδιαστή να δουλέψει είτε μέσα από μια γραφική διεπαφή είτε μέσω της κονσόλας εντολών με την χρήση του κατάλληλου script. Στο εδάφιο που ακολουθεί παρουσιάζεται ένα τυπικό script για την υποθετική περίπτωση ενός πλήρους αθροιστή 1 bit όπως αυτός έχει παρουσιαστεί σε προηγούμενα κεφάλαια.

1. set synopsys_auto_setup true
2. set hdlin_ignore_parallel_case true
3. set hdlin_ignore_full_case true
4. set verification_verify_directly_undriven_output true
5. set hdlin_ignore_embedded_configuration false
6. set svf_ignore_unqualified_fsm_information true
7. set_svf -append {/path/default.svf}
8. read_db {/path/tech_lib.db}
9. read_verilog -container r -libname WORK 01 {/path/full_adder.v}
10. set_top r:/WORK/full_adder
11. read_verilog -container i -libname WORK -01 {/path/full_adder_scan.v}
12. set_top i:/WORK/full_adder
13. match
14. verify

Οι πρώτες εντολές του script καλούνται προκειμένου να επισημάνουμε ποιες περιπτώσεις μπορεί να αγνοήσει το εργαλείο κατά την σύγκριση των υλοποιήσεων. Ακολουθεί η εντολή set_svf η οποία διαβάζει ένα .svf αρχείο το οποίο παράγεται αυτόματα κατά την διαδικασία της σύνθεσης και περιέχει πληροφορίες για τον τρόπο που αυτή έχει επιτελεστεί.

Η εντολή read_db διαβάζει την τεχνολογική βιβλιοθήκη που έχει χρησιμοποιηθεί κατά την σύνθεση του κυκλώματος, στην .db μορφή της όμως, η οποία αποτελεί εσωτερικό format της Synopsys και θα πρέπει να δημιουργηθεί μέσω του Library Compiler σε περίπτωση που δεν είναι διαθέσιμη από τον πάροχο.

Η εντολή read_verilog -container r διαβάζει την Verilog RTL περιγραφή του κυκλώματος. Με αυτή την εντολή δημιουργείται ένα reference design, δηλαδή μια σχεδίαση η οποία θα αποτελέσει το σημείο αναφοράς κατά τον έλεγχο του netlist που έχει δημιουργηθεί.

Ακολούθως, καλείται η ίδια εντολή με παράμετρο -container i και διαβάζει το netlist που έχει παραχθεί από τον Design Compiler και το οποίο αποτελεί το implementation design. Αφού ολοκληρωθεί και αυτό το βήμα καλείται η εντολή match η οποία ελέγχει κατά πόσο μπορεί να υπάρξει ταύτιση των δομικών στοιχείων των δυο περιγραφών, και παρουσιάζει μια αναλυτική αναφορά της συνολικής διαδικασίας επισημαίνοντας στοιχεία που έχει αγνοήσει και τυχόν προβλήματα που μπορεί να έχουν παρουσιαστεί.

Τέλος, εκτελείται η εντολή verify η οποία και επαληθεύει ή όχι την ισοδυναμία των κυκλωματικών περιγραφών. Σε περίπτωση που το τελικό αποτέλεσμα είναι αρνητικό ο σχεδιαστής καλείται να εξετάσει την δομή του κυκλώματος και την συνολική διαδικασία σύνθεσής του και να προβεί στις κατάλληλες αλλαγές που θα εξαλείψουν τα προβλήματα που έχουν παρουσιαστεί και θα του επιτρέψουν να προχωρήσει στα επόμενα βήματα της ροής.

8.Ανάλυση Χρονισμού

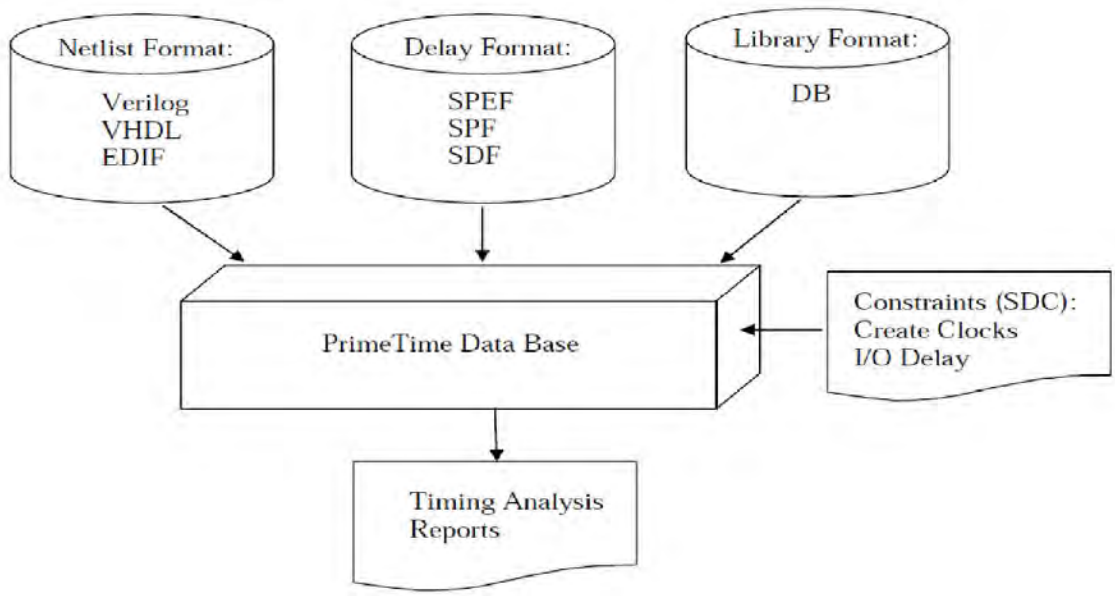
Το επόμενο βήμα στην ροή σχεδίασης που μελετάμε είναι η ανάλυση χρονισμού του κυκλώματος. Το Prime Time της Synopsys είναι ένα εργαλείο το οποίο μπορεί να πραγματοποιήσει στατική ανάλυση χρονισμού πάνω σε σχεδιάσεις πολλών εκατομμυρίων πυλών. Το βασικό πλεονέκτημα του Prime Time είναι το γεγονός πως δεν χρησιμοποιεί test vectors για την προσομοίωση του critical path. Αυτή η προσέγγιση εκμηδενίζει την πιθανότητα να μην αναγνωριστούν και εξεταστούν όλα τα critical paths κατά τον υπολογισμό των delay vectors.

Οι περισσότερες εντολές του εργαλείου είναι όμοιες με αυτές του Design Compiler, με αποτέλεσμα την εύκολη ενσωμάτωση του εργαλείου στην συνολική ροή. Επιπρόσθετα, το Prime Time δίνει την δυνατότητα στον χρήστη να αναλύσει μια σχεδίαση κάτω από διαφορετικές θερμοκρασίες, voltages και process variations.

Όταν το Prime Time αναλύει μια σχεδίαση, ελέγχει κατά πόσο κάθε μονοπάτι καλύπτει τους περιορισμούς για το setup time και το hold time που καθορίζονται από την τεχνολογική βιβλιοθήκη. Επιπρόσθετα, το εργαλείο θα ελέγξει το κατά πόσο τα transition times προς και από τις πύλες του κυκλώματος συμβαδίζουν με τα timing tables της βιβλιοθήκης.

Το setup time παραβιάζεται όταν τα δεδομένα αλλάζουν τιμή γρηγορότερα από την ακμή του ρολογιού, ενώ το hold time παραβιάζεται όταν τα δεδομένα αλλάζουν τιμή μετά την πάροδο της ακμής. Επομένως, υπάρχει ένα χρονικό παράθυρο γύρω από την εκάστοτε ακμή του ρολογιού κατά το οποίο δεν πρέπει να παρουσιάζονται αλλαγές τιμών. Σε περίπτωση που συμβούν, δημιουργείται μια ασταθής (metastable) κατάσταση στην οποία δεν μπορούμε να αναγνωρίσουμε ποιά είναι η τιμή του εκάστοτε σήματος. Για να είμαστε, βέβαια, ακριβής, είθισται να αγνοούμε τα hold violations σε αυτό το στάδιο της σχεδίασης και να ασχολούμαστε εκτενώς με αυτά κατά το στάδιο του clock tree synthesis στο placement tool που χρησιμοποιούμε.

Οι χρονικοί περιορισμοί (timing constraints) αποτελλούν τον τρόπο ενημέρωσης του Prime Time από τον σχεδιαστή για την χρονική συμπεριφορά του κυκλώματος. Υπάρχουν τρεις βασικού περιορισμοί οι οποίοι πρέπει να καθοριστούν προκειμένου να εγγυηθεί το εργαλείο πως το τελικό αποτέλεσμα είναι αυτό που έχει οραματιστεί ο σχεδιαστής. Αυτοί είναι ο προσδιορισμός του ρολογιού της σχεδίασης, της καθυστέρησης εισόδου (input delay) και της καθυστέρησης εξόδου (output delay). Όταν καθοριστούν όλα τα ρολόγια της σχεδίασης το Prime Time υποθέτει πως όλα τα μονοπάτια από register σε register πρέπει να επιτελούν την μετακίνηση δεδομένων εντός ενός κύκλου ρολογιού. Η καθυστέρηση εισόδου καθορίζει τον χρόνο που απαιτείται από την έξοδο ενός άλλου κυκλώματος στην είσοδο του υπό εξέταση κυκλώματος, ενώ η καθυστέρηση εξόδου καθορίζει τον χρόνο που απαιτείται για την μετάδοση δεδομένων από την έξοδο του προκείμενου κυκλώματος στην είσοδο ενός άλλου υποθετικού κυκλώματος. Το συνολικό σύστημα εισόδου - εξόδου του εργαλείου, λοιπόν, αποκτά την ακόλουθη μορφή:



Εικόνα 8.1: PrimeTime I/O

Ένα τυπικό παράδειγμα διενέργειας ανάλυσης χρονισμού με την χρήση ενός tcl script είναι το ακόλουθο:

1. set search_path “/path”
2. set target_library “/path/library.db”
3. set link_path “/path”
4. read_file -format verilog /path/full_adder.v
5. current_design full_adder
6. link_design
7. read_parasitics -format SPEF /path/full_adder.parasitics.spef
8. read_sdc /path/full_adder.sdc
9. create_clock -period [get_ports clk]
10. check_timing
11. set_operating_conditions -min WORST -max WORST
12. report_timing
13. report_timing -from [all_registers] -to [all_registers]
14. report_timing -from [all_registers] -to [all_outputs]
15. report_timing -from [all_inputs] -to [all_outputs]
16. report_constraints -all_violators

Αρχικά, όπως και στα υπόλοιπα εργαλεία της Synopsys που έχουμε παρουσιάσει σε προηγούμενα κεφάλαια της εργασίας, καθορίζουμε τα μονοπάτια στα οποία είναι αποθηκευμένη η σχεδίαση, η τεχνολογική βιβλιοθήκη και διαβάζουμε στην μνήμη του εργαλείου τον κώδικα της σχεδίασης. Ακολούθως, διαβάζουμε το αρχείο που αφορά τα παρασιτικά και το αρχείο που περιγράφει τους χρονικούς περιορισμούς της σχεδίασης, αρχεία τα οποία έχουν παραχθεί από τον Design Compiler κατά την διαδικασία της σύνθεσης.

Στο επόμενο στάδιο εκτελείται ο έλεγχος του χρονισμού και καθορίζονται οι συνθήκες κάτω από τις οποίες θα λειτουργεί το κύκλωμα. Μετά το πέρας αυτού του βήματος το εργαλείο παράγει μια σειρά από αναφορές (reports) οι οποίες περιέχουν

όλες τις απαραίτητες πληροφορίες για τον χρονισμό της σχεδίασης, αλλά και τον χρονισμό κάθε μονοπατιού ξεχωριστά.

Τέλος, η τελευταία εντολή δημιουργεί ένα report στο οποίο παρουσιάζονται όλα εκείνα τα μονοπάτια τα οποία παραβιάζουν μερικούς ή όλους τους περιορισμούς που έχει θέσει ο χρήστης. Στο σημείο αυτό ο σχεδιαστής, ζυγίζοντας τις επιλογές του, αποφασίζει κατά πόσο είναι εφικτό και θεμιτό να τροποποιήσει το κύκλωμα στο στάδιο αυτό ή αν θα συνεχίσει στο επόμενο βήμα της σχεδιαστικής ροής και θα προσπαθήσει να λύσει τα υπάρχοντα προβλήματα σε ακόλουθα στάδια.

9.Ανάλυση Κατανάλωσης Ισχύος

9.1. Βασικές Έννοιες

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

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

Ο πρώτος όρος της εξίσωσης (Paverage) είναι η μέση ισχύς που καταναλώνεται στο κύκλωμα, ο όρος Pdynamic περιγράφει την δυναμική κατανάλωση ισχύος που προκαλείται από την αλλαγή κατάστασης των διακοπτών, το Pshort-circuit είναι το ποσοστό εκείνο της ισχύος που καταναλώνεται όταν υπάρχει άμεσο μονοπάτι από την πηγή στην γείωση, ενώ, τέλος, το Pstatic περιγράφει την στατική κατανάλωση ισχύος. Στα επόμενα εδάφια θα εξηγήσουμε αναλυτικά κάθε έναν από τους όρους της παραπάνω σχέσης.

- Static Power: Ως στατική ισχύ χαρακτηρίζουμε την ισχύ που καταναλώνεται από μια πύλη όταν αυτή είναι στατική ή αδρανής. Τα CMOS κυκλώματα στην ιδανική περίπτωση θεωρείται πως καταναλώνουν μηδενική ισχύ, αφού στην κατάσταση ισορροπίας τους δεν υπάρχει μονοπάτι που να συνδέει την πηγή με την γείωση. Αυτό το σενάριο, βέβαια, δεν υλοποιείται ποτέ στην πραγματικότητα αφού τα MOS τρανζίστορ δεν αποτελούν τέλειους διακόπτες και πάντοτε υπάρχουν, έστω και με πολύ μικρές τιμές ρεύματα διαρροής. Το μεγαλύτερο ποσοστό στατικής ισχύος οφείλεται σε ένα δυναμικό που αναπτύσσεται ανάμεσα στην πηγή και την γείωση, το οποίο προκαλείται από ελαττωμένα δυναμικά κατωφλιού τα οποία εμποδίζουν την εκάστοτε πύλη από το να κλείσει εντελώς.

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

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

    - Internal Power: Ως εσωτερική ισχύ χαρακτηρίζουμε οποιαδήποτε ισχύ καταναλώνεται μέσα στα όρια ενός κελιού. Κατά την διάρκεια της λειτουργίας του, ένα ψηφιακό κύκλωμα καταναλώνει εσωτερική ισχύ λόγω της συνεχούς φόρτισης και εκφόρτισης των πυκνωτών που βρίσκονται στο εσωτερικό των κελιών. Η εσωτερική ισχύς περιλαμβάνει και ένα ποσοστό το οποίο προκαλείται από την στιγμιαία ένωση (δημιουργία μονοπατιού) μεταξύ του P και του N τρανζίστορ μιας πύλης, το ποσοστό αυτό της ισχύος ονομάζεται short-circuit power. Στα περισσότερα κελιά που χρησιμοποιούνται

στη σύγχρονη βιομηχανία, το μεγαλύτερο ποσοστό της εσωτερικής ισχύος οφείλεται στην short-circuit power.

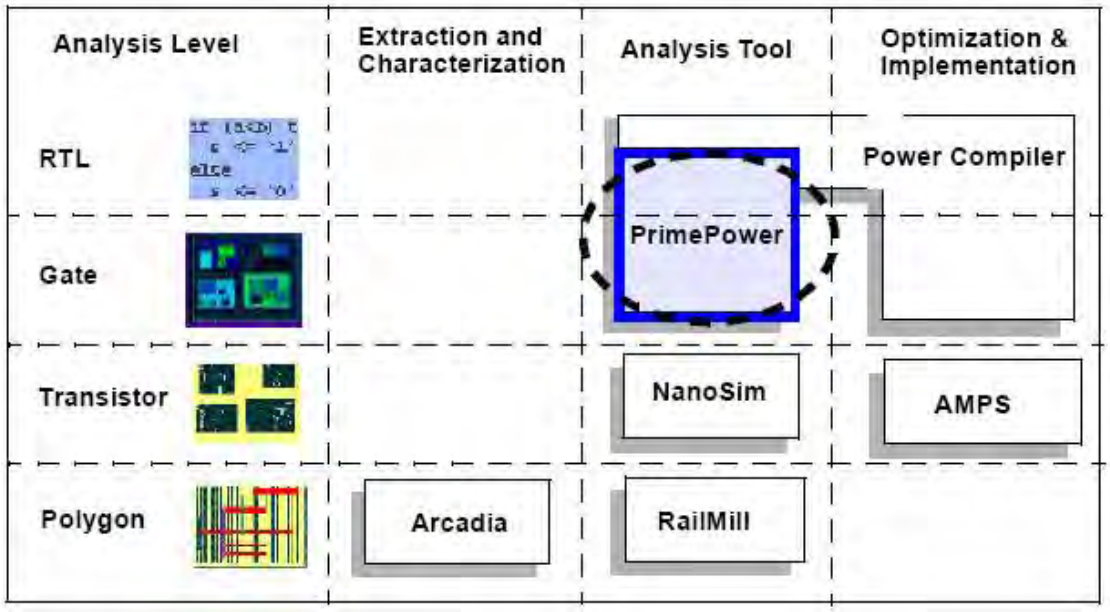
- Leakage Power: Τα NMOS και PMOS τρανζίστορ που χρησιμοποιούνται στα CMOS κυκλώματα έχουν μη μηδενικά ρεύματα διαρροής και υποκατωφλίου. Τα ρεύματα αυτά μπορούν να “συνεισφέρουν” στην συνολική κατανάλωση ισχύος ακόμη και όταν το τρανζίστορ δεν παρουσιάζει αλλαγές λογικών τιμών στην λειτουργία του ως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

  - το ρεύμα διαρροής αντίστροφης πόλωσης
  - το ρεύμα υποκατωφλίου το οποίο διέρχεται από κάποιο κανάλι του τρανζίστορ, όταν αυτό είναι κλειστό.

9.2. Μεθοδολογία Κατανάλωσης Ισχύος με Χρήση του Prime Power

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

Το Prime Power υποστηρίζει τόσο στατική όσο και δυναμική ανάλυση της κατανάλωσης ισχύος σε επίπεδο πυλών. Στην προκείμενη εργασία θα ασχοληθούμε μόνο με την δυναμική ανάλυση. Η εικόνα που ακολουθεί παρουσιάζει τον τρόπο με τον οποίο εμπλέκεται το Prime Power στην ροή σχεδίασης χαμηλής ισχύος που προσφέρει το πακέτο στης Synopsys.



Εικόνα 9.1: Power Analysis στη ροή σχεδίασης της Synopsys

Το πρόγραμμα που εξετάζουμε, δίνει την δυνατότητα στον χρήστη να επιλέξει τον τρόπο λειτουργίας. Έτσι καλούμαστε να επιλέξουμε εάν θα προβούμε σε average analysis, η οποία στηρίζεται σε προεπιλεγμένα αποτελέσματα όσον αφορά το switching activity και μας δίνει μια εξιδανικευμένη προσέγγιση της συνολικής ισχύος που θα καταναλωθεί ή σε dynamic (peak power) analysis, όπου λαμβάνουμε μια λεπτομερέστερη εκτίμηση βασιζόμενη σε στοιχεία που εισάγονται από τον χρήστη.

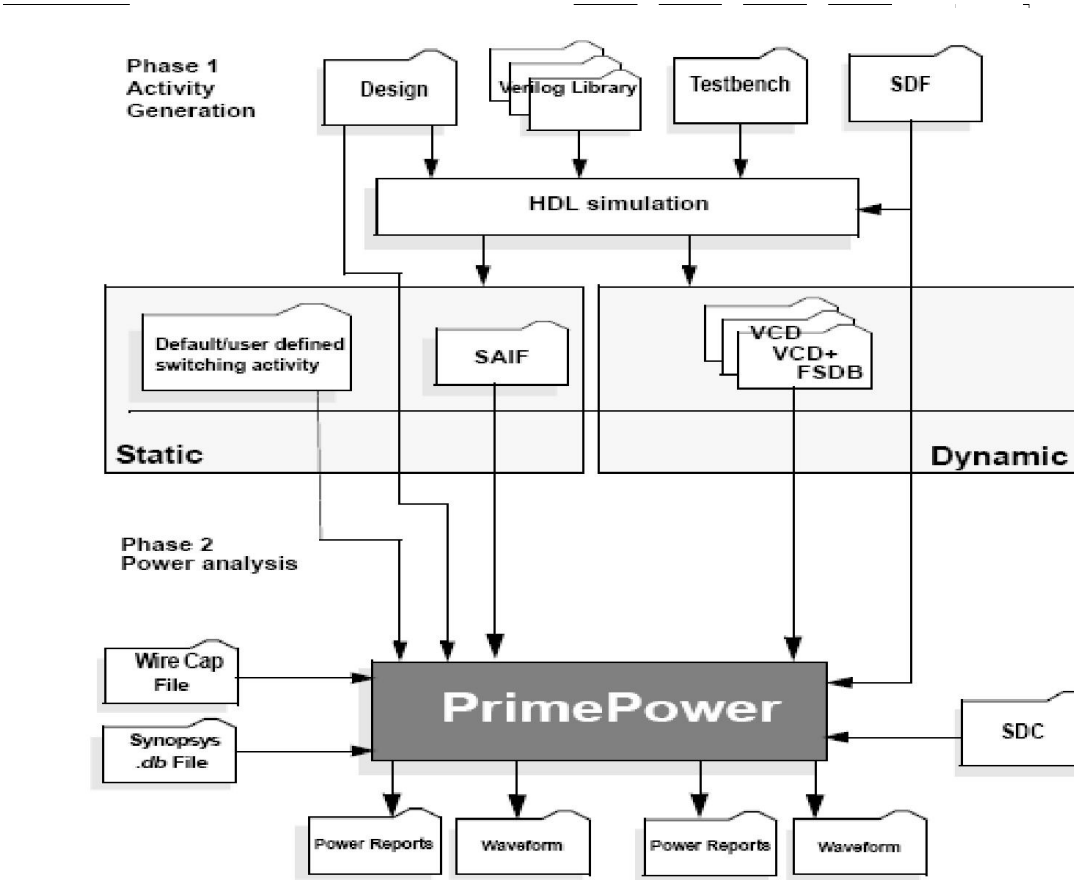
Στο υπόλοιπο μέρος του κεφαλαίου θα περιγράψουμε την ροή προσομοίωσης που ακολουθεί το Prime Power και θα δούμε συνοπτικά τον τρόπο με τον οποίο εκτελεί την ανάλυση καταναλισκώμενης ισχύος.

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

1. την δημιουργία της κατάλληλης πληροφορίας για το switching activity του κυκλώματος και

2. την δημιουργία του power profile της σχεδίασης

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



Εικόνα 9.2: PrimePower Power Analysis Flows

Στην πρώτη φάση της εξομοίωσης, στην περίπτωση που θέλουμε να πραγματοποιήσουμε δυναμική ανάλυση για την εξαγωγή αποτελεσμάτων μεγαλύτερης ακρίβειας, καλούμαστε να δώσουμε ως είσοδο στο εργαλείο που χρησιμοποιούμε το gate level time-based switching activity του κυκλώματος σε μια από τις δυνατές επιλογές που μας δίνονται. Στην παρούσα εργασία θεωρούμε πως έχουμε τα κατάλληλα VCD αρχεία, τα οποία δημιουργήσαμε με χρήση του εργαλείου προσομοίωσης ModelSim. Τα αρχεία αυτά εξάγονται κατά την προσομοίωση του κυκλώματος σε επίπεδο πυλών και περιέχουν μεγάλο κομμάτι από το σύνολο των

πληροφοριών για το switching activity που μπορούμε να αντλήσουμε για οποιαδήποτε σχεδίαση.

Στην δεύτερη φάση, χρησιμοποιούμε ένα script το οποίο περιέχει το σύνολο των εντολών που θέλουμε να εκτελέσει σειριακά ο πυρήνας του Power Compiler, ο οποίος καλείται μέσω του Prime Power. Το script που παρουσιάζεται παρακάτω αποτελεί τυπικό παράδειγμα:

1. set search_path “/path”
2. set link_library “/path/library.db”
3. read_db “/path/full_adder.db”
4. current_design full_adder
5. link
6. read_vcd full_adder.vcd
7. set_input_transition 0.1 [all_inputs]
8. read_parasitics full_adder.spef
9. set_waveform_options -interval 1 -file pwr_vcd -format fsdb
10. calculate_power -waveform -reset_neg_power
11. report_power

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

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

χρησιμοποιηθούν από την σχεδίασή μας. Το Prime Power ψάχνει για τα κατάλληλα design elements στις βιβλιοθήκες, με την σειρά που αυτές αναφέρονται στο script.

Στην συνέχεια καλούμαστε να διαβάσουμε στη μνήμη την σχεδίαση πάνω στην οποία θα δουλέψουμε. Η εντολή current_design ανακτά την τρέχουσα σχεδίαση και πρέπει πάντα να προηγείται της εντολής link, όπως ακριβώς συμβαίνει στο παραπάνω παράδειγμα.

Δίνοντας την εντολή link πραγματοποιούνται οι προσπελάσεις στις βιβλιοθήκες που έχουμε ορίσει, έτσι ώστε να διαβαστούν και αυτές στη μνήμη του εργαλείου

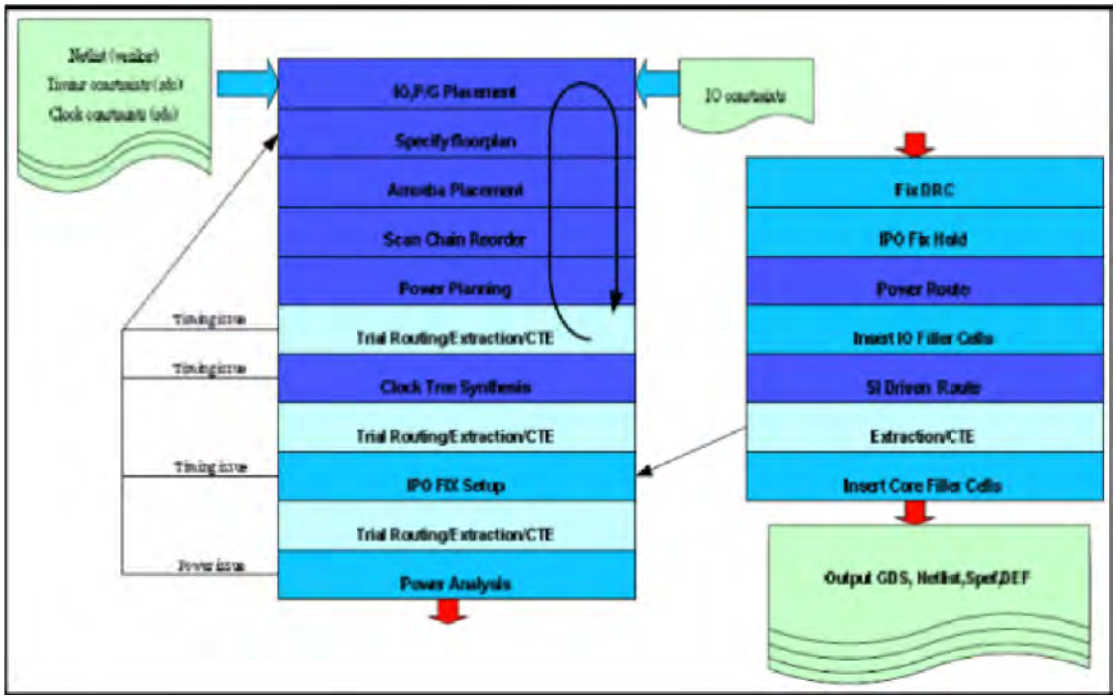
Συνεχίζοντας προσδιορίζουμε την θέση του testbench που περιέχει τις κατάλληλες αναφορές στην σχεδίαση και το VCD αρχείου που έχουμε εξάγει σε προγενέστερη φάση της συνολικής διεργασίας με την εντολή read_vcd.

Με την εντολή set_input_transition προσδιορίζουμε το transition time του κάθε port της σχεδίασης. Το Prime Power χρησιμοποιεί την πληροφορία αυτή κατά τον υπολογισμό της ισχύος της λογικής που οδηγείται από το εκάστοτε port. Με την επόμενη εντολή διαβάζουμε τα παρασιτικά που έχουμε εξάγει κατά την σύνθεση του κυκλώματος τα οποία επίσης παίζουν σημαντικό ρόλο στην διαδικασία υπολογισμού της καταναλισκώμενης ισχύος.

Τέλος, αφού δώσουμε την εντολή υπολογισμού calculate_power, με χρήση της εντολής report και μιας σειράς παραμέτρων παίρνουμε αναλυτικές αναφορές για την ισχύ που καταναλώνεται τόσο στο σύνολο της σχεδίασης όσο και σε κάθε net, leaf, και cell. Βάσει αυτών των αναφορών μπορούμε να οπισθοδρομήσουμε σε προηγούμενα στάδια της συνολικής ροής τροποποιώντας ανάλογα ορισμένα components της σχεδίασης των οποίων η συμπεριφορά δεν μας ικανοποιεί.

10.Χωροθέτηση και Δρομολόγηση

Στην παρακάτω εικόνα παρουσιάζεται μια εκτενής ροή place & route ενός ολοκληρωμένου κυκλώματος, θεμελιώδη βήματα της οποίας αναλύονται στη συνέχεια.

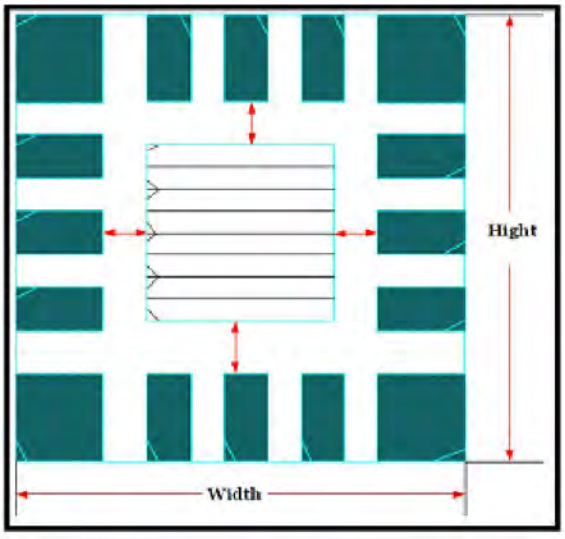


Εικόνα 10.1: Analytical P&R Flow

Τα βιομηχανικά placement tools δέχονται ωε είσοδο ένα netlist σε επίπεδο κελιών και ένα σύνολο αρχείων τα οποία περιγράφουν τους χρονικούς περιορισμούς βάσει των οποίων πρέπει να εκτελεστεί η χωροθέτηση του κυκλώματος. Ακολούθως, ο χρήστης ορίζει τις θέσεις των pads, τα οποία χωρίζονται σε τρεις κατηγορίες: I/O pads, PWR/GND pads και Corner pads, με τα τελευταία να χρησιμοποιούνται αποκλειστικά για την διασύνδεση με τα power rings.

Στο επόμενο βήμα καθορίζεται το floorplan, παρέχεται δηλαδή η χωρική αναλογία μεταξύ του πυρήνα της σχεδίασης και της κενής περιοχής μεταξύ των pads και αυτού. Επιπρόσθετα, καθορίζονται οι “σχετικές” θέσεις τυχόν hard blocks που μπορεί να εμπεριέχονται στη σχεδίαση.

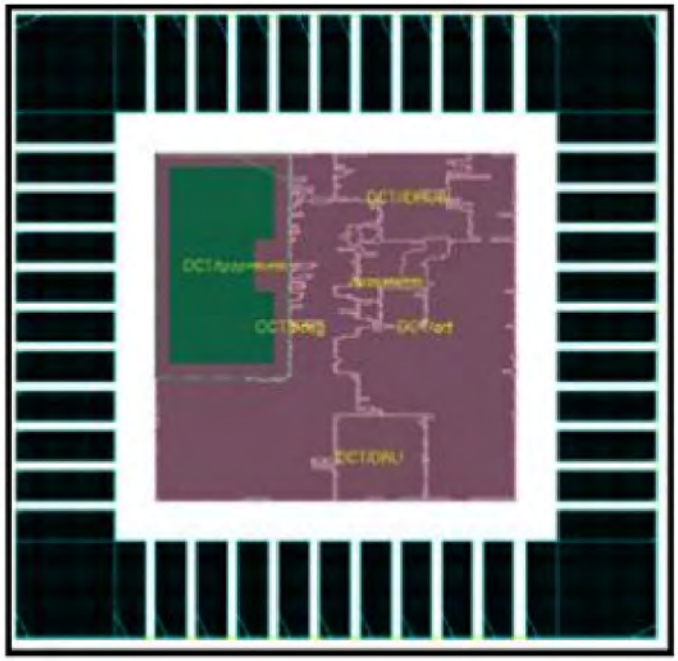
Εικόνα 10.2: I/O, PWR/GND, Corner PAD placement



Εικόνα 10.3: Floorplanning

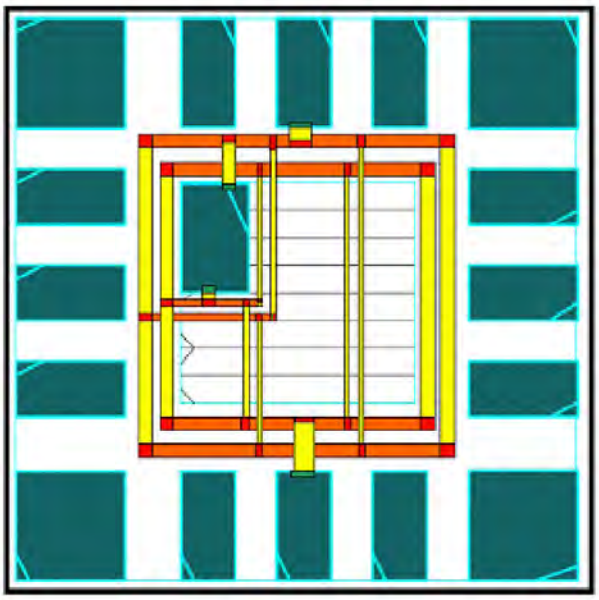
Ακολουθεί το Amoeba placement όπου πραγματοποιείται μια πρώτη χωροθέτηση όλων των module της σχεδίασης.

Το πέρας της παραπάνω διαδικασίας σηματοδοτεί την έναρξη του power planning. Κατά την εκτέλεση αυτού του βήματος, χωροθετούνται τα power rings και τα power stripes λαμβάνοντας υπόψη τα φαινόμενα πτώσης τάσης που θα εμφανιστούν και μπορούν να αλλοιώσουν την απόδοση του κυκλώματος.

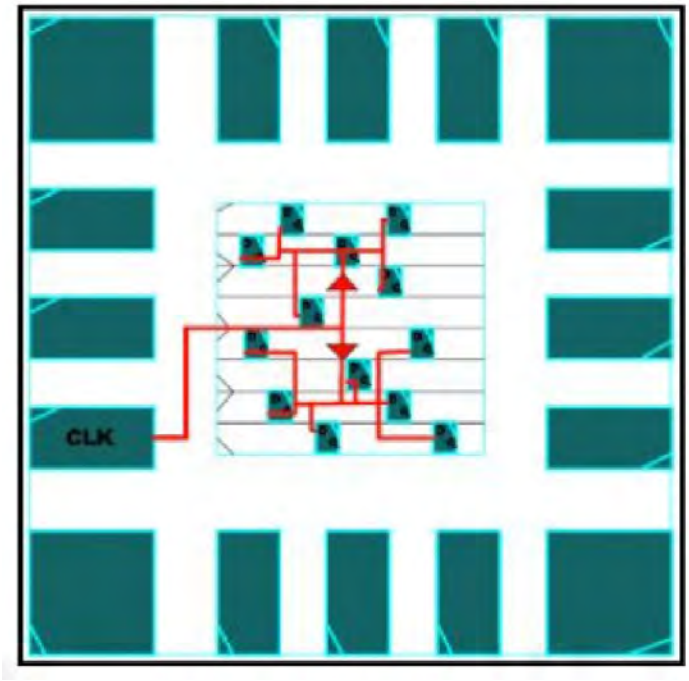


Εικόνα 10.4: Amoeba placement

Το στάδιο του power planning διαδέχεται η διαδικασία του Clock Tree Synthesis (CTS). Στόχος του CTS είναι η ελαχιστοποίηση του skew της σχεδίασης και του insertion delay.

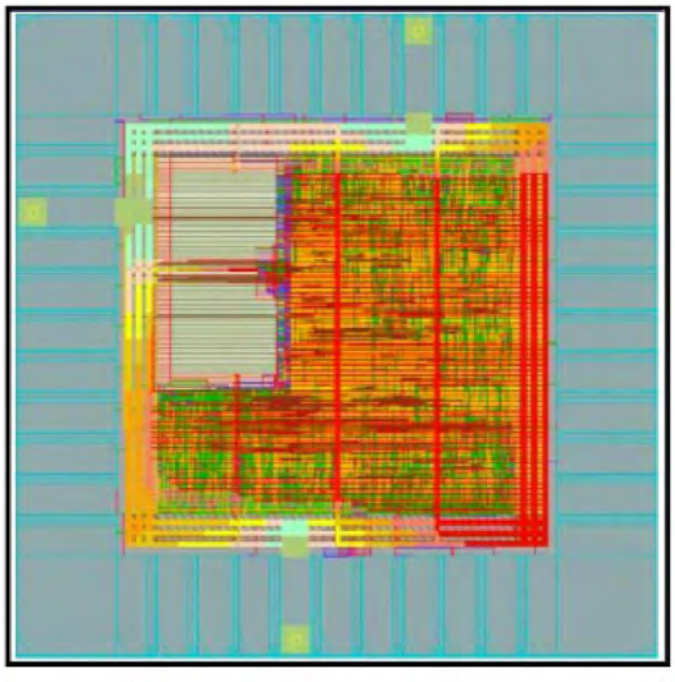


Εικόνα 10.5: Power Planning



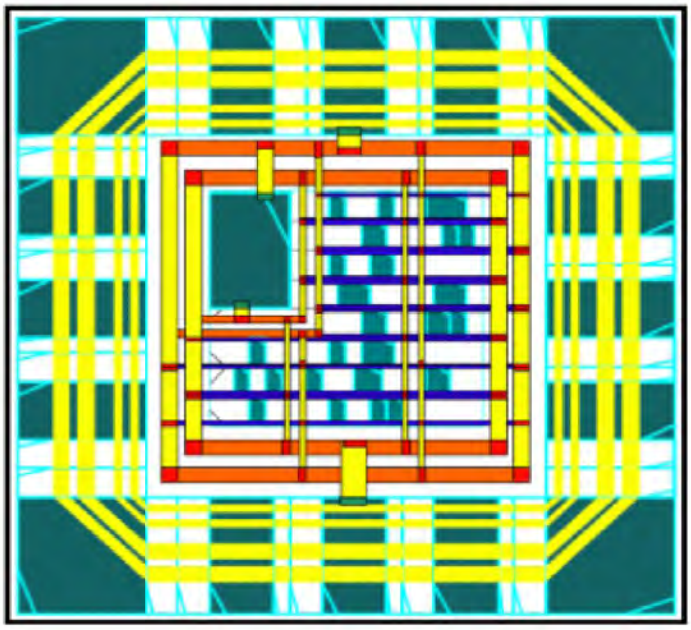
Εικόνα 10.6: Clock Tree Synthesis

Τα δυο επόμενα βήματα της διαδικασίας του place & route, είναι το power analysis και το power routing. Στο πρώτο από αυτά πραγματοποιούνται οι υπολογισμοί για την αριθμητική απεικόνιση των αποτελεσμάτων των φαινομένων πτώσης τάσης και μετανάστευσης ηλεκτρονίων, ενώ στο δεύτερο υλοποιούνται οι συνδέσεις των power pins των κελιών του κυκλώματος με τις global power lines που έχουν καθοριστεί σε προηγούμενο βήμα.

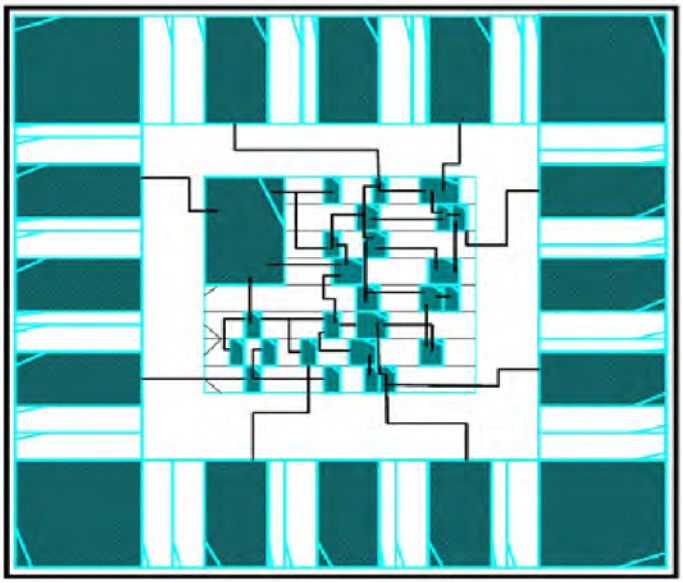


Εικόνα 10.7: Power Analysis

Τέλος, στο στάδιο του routing υλοποιούνται οι τελικές διασυνδέσεις μεταξύ των επιμέρους τμημάτων της σχεδίασης έτσι ώστε να απαιτείται το μικρότερο δυνατό μήκος καλωδίου.



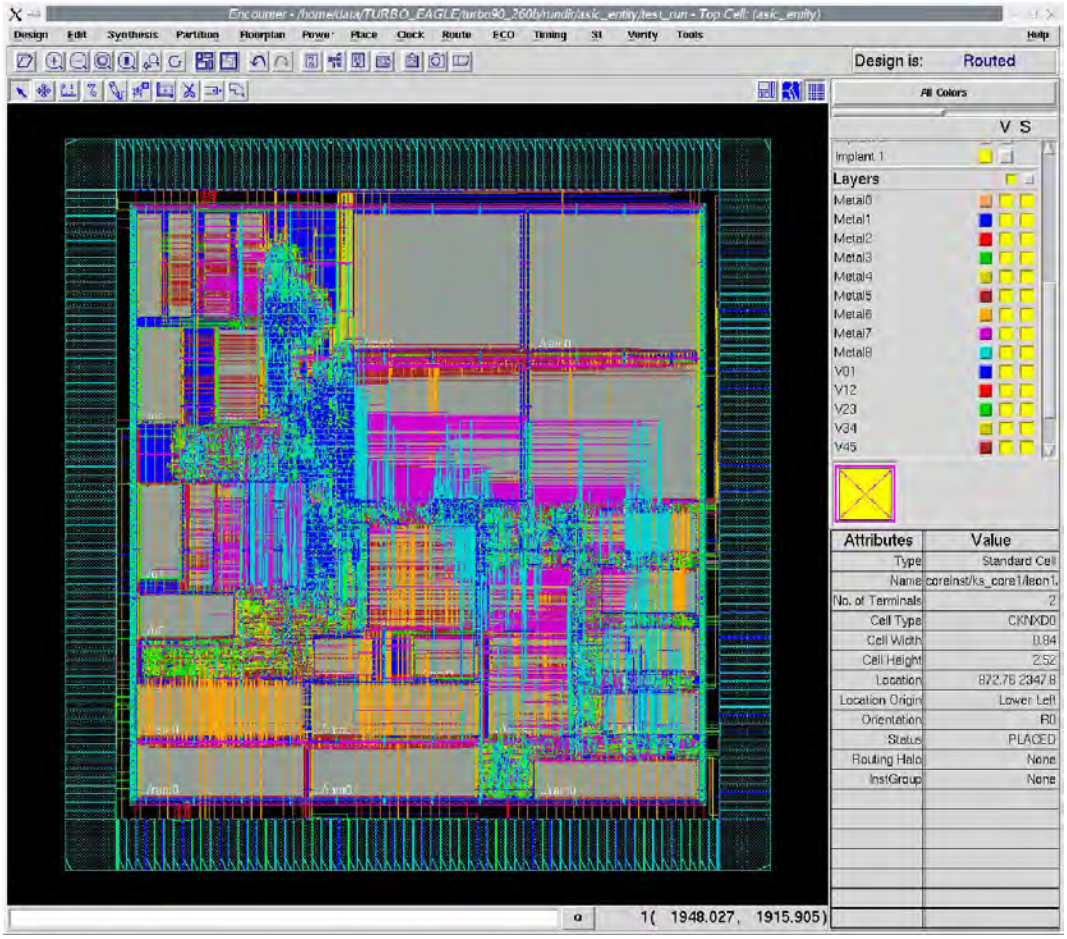
Εικόνα 10.8: Power Routing



Εικόνα 10.9: Final Routing

Το τελικό αποτέλεσμα της διαδικασίας είναι ένα σύνολο αρχείων περιγραφής του κυκλώματος, με σημαντικότερο από αυτά το αρχείο GDSII το οποίο και αποτελεί το πρότυπο που διαβάζεται προκειμένου να εκκινήσει η διαδικασία του manufacturing.

Το σημαντικότερο εργαλείο χωροθέτησης και δρομολόγησης ψηφιακών κυκλωμάτων είναι ο SoC Encounter της Cadence. Το εργαλείο αυτό, ακολουθώντας την λογική όλων των εργαλείων που έχουμε παρουσιάσει σε προηγούμενα κεφάλαια, δύναται να χρησιμοποιηθεί είτε μέσω της γραφικής του διεπαφής είτε μέσω της κονσόλας εντολών. Για λόγους συντομίας στο τελευταίο τμήμα αυτού του κεφαλαίου παρουσιάζουμε ένα τυπικό script για το place & route μιας ψηφιακής σχεδίασης.



Εικόνα 10.10: SoC Encounter GUI

- set fillerCells [list cell1, cell2, ….]

Τα filler cells περιγράφονται μέσα στην τεχνολογική βιβλιοθήκη. Είναι κελιά τα οποία χρησιμοποιούνται για να γεμίσουν τους κενούς χώρους μεταξύ των standard cells έτσι ώστε να αποφύγουμε προβλήματα επιπεδότητας (planarity). Η τοποθέτησή τους κρίνεται απαραίτητη όταν η πυκνότητα ων απαιτούμενων, για την χωροθέτηση, μετάλλων δεν βρίσκεται στα επιθυμητά επίπεδα για fabrication.

Το να γεμίσουμε όλο τον διαθέσιμο χώρο, είναι γενικώς απίθανο. Σε ένα ψηφιακό layout χρειάζονται και κάποιοι ελεύθεροι χώροι προκειμένου να μπορεί να βελτιωθεί το routing. Εάν πάλι δεν χρησιμοποιήσουμε καθόλου filler cells θα καταλήξουμε σε μια σχεδίαση η οποία δεν θα είναι λειτουργική, διότι κατά το fabrication θα εμφανιστούν διαδοχικά διασυνδεδεμένα τρανζίστορ τα οποία θα βρίσκονται σε πολύ μεγάλη απόσταση, γεγονός που έρχεται σε άμεση αντίθεση με τους βασικού κανόνες ορθής σχεδίασης ενός layout.

Τα filler cells δεν προσδίδουν κάποια λειτουργικότητα στο κύκλωμα, διαθέτουν μόνο power και ground rails έτσι ώστε να επιτυγχάνεται η συνεχής τροφοδότηση όλων των στοιχείων της σχεδίασης.

Τέλος, πρέπει να γνωρίζουμε πως όταν ζητάμε από τον Encounter να τοποθετήσει filler cells στην σχεδίασή μας υπάρχει πάντα η πιθανότητα να εμφανιστούν αλληλεπικαλύψεις, οι οποίες πρέπει να διορθωθούν χειρονακτικά. Σε περίπτωση που τα filler cells τοποθετούνται καθ’ ολοκληρία από τον σχεδιαστή, ο βασικός κανόνας που ακολουθείται είναι να τοποθετούμε σε πρώτο στάδιο τα μεγαλύτερα (σε μέγεθος) κελιά και στην συνέχεια να χωροθετούμε τα μικρότερα filler cells.

- set usepct 0.7

Οριοθετούμε το ποσοστό αξιοποίησης του διαθέσιμου χώρου κατά την χωροθέτηση των κελιών. Το ποσοστό αυτό αποτελεί μια εκτίμηση βάσει των μεγεθών

των στοιχείων της σχεδίασης και για τον υπολογισμό του λαμβάνονται υπόψη μόνο τα standard cells και τα hard macros.

- set rowgap 70

Με την μεταβλητή αυτή καθορίζεται το μέγεθος του κενού μεταξύ των σειρών από standard cells του κυκλώματος.

- set aspect 0.9

Η μεταβλητή αυτή καθορίζει την αναλογία των διαστάσεων των πλευρών του chip. Δίνοντας την τιμή 1 ορίζουμε κατ' ουσίαν ένα τετράγωνο, ενώ με τιμές διάφορες του 1 ορίζουμε παραλληλόγραμμα.

- set pwidth 10
- set pspace 2

Οι παραπάνω μεταβλητές ορίζονται προκειμένου να καθορίσουμε το πλάτος των power rails και την απόσταση μεταξύ τους. Ένα power rail είναι στην πραγματικότητα ένα supply voltage το οποίο τροφοδοτεί τα components του κυκλώματος.

- set swidth 10
- set sspace 2

Όπως και παραπάνω καθορίζουμε το πλάτος και την απόσταση μεταξύ των power stripes, τα οποία χρησιμοποιούνται προκειμένου να συνδέσουμε όλα τα components της σχεδίασης με τα power rails.

- set ssdist 100
- set soffset 20
- set coregap 80.0

Οι παραπάνω μεταβλητές ορίζουν τις τιμές που θα χρησιμοποιηθούν για το set-to-set distance, την απόσταση από τα όρια του πυρήνα της πρώτης γραμμής και το κενό μεταξύ του πυρήνα της σχεδίασης και των power rails που θα οριστούν σε επόμενο βήμα.

- loadConfig /path/full_adder.conf
- commitConfig

Στο επόμενο βήμα καλούνται οι προκείμενες εντολές οι οποίες φορτώνουν το configuration file της σχεδίασης το οποίο περιέχει πληροφορίες για την θέση στο σκληρό δίσκο όπου είναι αποθηκευμένες οι τεχνολογικές βιβλιοθήκες που θα χρησιμοποιήσουμε, όπως και οι βιβλιοθήκες χρονισμού και οι περιγραφές των hard macros σε μορφή αρχείων lef. Τα αρχεία lef και def αποτελούν κοινά αποδεκτά standards για την περιγραφή της χρησιμοποιούμενης τεχνολογίας και της διασυνδεσιμότητας της σχεδίασης αντίστοιχα.

Οι εντολές που ακολουθούν υλοποιούν την διαδικασία του floorplanning που έχουμε αναφέρει κατά την ανασκόπηση της συνολικής διαδικασίας. Στο στάδιο του floorplan καθορίζονται οι διαστάσεις της συνολικής σχεδίασης, αλλά και οι επιμέρους γεωμετρικές λεπτομέρειες του συνολικού chip.

- setDrawView fplan
- floorplan -r $aspect $usepct $coregap $coregap $coregap $coregap
- fit

Η πρώτη εντολή αλλάζει τον τρόπο απεικόνισης της σχεδίασης στο γραφικό περιβάλλον του Encounter. Η εντολή που ακολουθεί είναι αυτή που καθορίζει όλες τις επιμέρους λεπτομέρειες του floorplan. Έτσι, στην συγκεκριμένη περίπτωση, ο σχεδιαστής δηλώνει πως επιθυμεί να έχει αναλογία διαστάσεων ίση με $aspect, βελτιστοποίηση στον τρόπο χωροθέτησης ίση με $usepct και απόσταση του πυρήνα της σχεδίασης από το υπόλοιπο chip ίση με $coregap. Η τελευταία εντολή αλλάζει τις διαστάσεις της απεικόνισης της σχεδίασης στην οθόνη του υπολογιστή έτσι ώστε να

είναι δυνατή η θέαση ολόκληρου του chip εντός του γραφικού περιβάλλοντος του εργαλείου.

- clearGlobalNets
- globalNetConnect VSS -type pgpin -pin VSS -inst *
- globalNetConnect VDD -type pgpin -pin VDD -inst *

Στο επόμενο βήμα καθορίζουμε τα global nets, τα δίκτυα τροφοδοσίας, δηλαδή, της σχεδίασης. Αρχικά, με την πρώτη εντολή εξαλείφουμε οποιαδήποτε global nets υπάρχει πιθανότητα να υφίστανται λόγω των αρχείων lef ή def που έχουμε φορτώσει μέσω του configuration file, και στην συνέχεια δημιουργούμε εκ νέου τα global nets για την τροφοδοσία και την γείωση επιλέγοντας να τα ενώσουμε με όλα τα pins του κάθε κελιού της σχεδίασης τα οποία έχουν ορισθεί από την τεχνολογική βιβλιοθήκη για την συγκεκριμένη λειτουργικότητα.

- addRing -spacing_bottom $pspace -spacing_top $pspace -spacing_right $pspace -spacint_left $pspace -width_left $pwidth -width_right $pwidth -width_top $pwidth -width_bottom $pwidth -layer_bottom M5 -layer_top M6 -layer_right M6 -layer_left M6 -jog_distance $pspace -nets {VSS VDD}

Ακολούθως, ορίζουμε το power ring της σχεδίασης, το οποίο περιβάλλει τον πυρήνα της. Η εντολή addRing διαθέτει μια σειρά από παραμέτρους, όπως παρατηρούμε και στο παράδειγμα που παρουσιάζεται, οι οποίες ορίζουν την απόσταση του power ring από το core design, το πλάτος του ring, και το jog_distance το οποίο πρακτικά είναι η μέγιστη απόσταση που μπορεί να διανύσει ο δακτύλιος τροφοδοσίας σε κάποιο άλλο μέταλλο από αυτό στον οποίο το έχουμε ορίσει έτσι ώστε να αποφευχθούν παραβιάσεις των κανόνων σχεδίασης (design rules). Η τελευταία παράμετρος της εντολής καθορίζει ποια είναι τα βασικά power nets για τα οποία θα δημιουργηθούν οι κατάλληλοι δακτύλιοι.

- addStripe -max_same_layer_jog_length 1.0 -width $width -spacing $pspace -set_to_set_distance $ssdist -layer M6 -merge_stripes_value 0.8 -nets {VSS VDD}

Η παραπάνω εντολή χρησιμοποιείται για την δημιουργία των power stripes. Η δημιουργία των stripes δεν είναι απαραίτητη, αποτελεί όμως σύνηθες πρακτική προκειμένου να βελτιώσουμε την διανομή της τροφοδοσίας στο εσωτερικό του κυκλώματος. Όπως και η προηγούμενη εντολή, έτσι και αυτή δίνει στον χρήστη την δυνατότητα μέσω μιας σειράς παραμέτρων να συγκεκριμενοποιήσει τον τρόπο με τον οποίο θα υλοποιηθούν τα power stripes. Η παράμετρος max_same_layer_jog_length ορίζει το μέγιστο μήκος, σε μικρόμετρα, που δύναται να διανύσει ένα stripe στο ίδιο επίπεδο μετάλλου προτού αναγκαστεί να μεταπηδήσει σε επόμενο layer. Η παράμετρος width καθορίζει το πλάτος του εκάστοτε stripe. Με την επιλογή spacing ορίζεται η απόσταση μεταξύ δυο γειτονικών stripes, ενώ, τέλος, με την παράμετρο merge_stripes_value δηλώνουμε πως αν η απόσταση μεταξύ ενός stripe και κάποιου block είναι μικρότερη από την αναγραφόμενη, τότε ο Encounter, καλείται να συγχωνεύσει το προκείμενο power stripe.

- sroute

Στην συνέχεια καλείται ο special router με την εντολή sroute, προκειμένου να δρομολογηθούν τα power και ground nets κατά τέτοιο τρόπο έτσι ώστε να ελαχιστοποιηθεί η πιθανότητα εμφάνισης DRC (Design Rule Check) violations.

Με την εντολή που ακολουθεί, πραγματοποιείται η χωροθέτηση των κελιών της σχεδίασης. Στο σημείο αυτό, πρέπει να τονίσουμε πως παρόλο, που ο Encounter δίνει την δυνατότητα βελτιστοποίησης της σχεδίασης μέσω της εντολής optDesign, εντούτοις η χρήση της κρίνεται σκόπιμη μόνο κατά τα τελευταία στάδια της συνολικής διαδικασίας, αφού οι αλλαγές που δύναται να προκαλέσει στο κύκλωμα μπορεί να στοιχήσουν σε ακρίβεια σε επόμενα στάδια.

- placeDesign -timingDriven

Η χρήση της παραμέτρου -timingDriven οδηγεί τον Encounter στο να τοποθετήσει τα κελιά της σχεδίασης κατά τέτοιο τρόπο ώστε να βελτιώνεται ο συνολικός χρονισμός του κυκλώματος.

- createClockTreeSpec -output clock.ctstch
- clockDesign

Ακολούθως, ο σχεδιαστής καλείται να δημιουργήσει το clock tree της σχεδίασης, μια διαδικασία η οποία ονομάζεται Clock Tree Synthesis (CTS). Κατά το CTS καθορίζεται ο τρόπος διαμοιρασμού του ρολογιού της σχεδίασης σε κάθε ένα από τα επιμέρους κελιά της σχεδίασης. Σημαντικό ρόλο κατά την διαδικασία αυτή παίζουν τα .sdc αρχεία που έχουν παραχθεί κατά την διαδικασία της σύνθεσης και περιγράφουν όλους τους χρονικούς περιορισμούς της σχεδίασης.

- globalDetailRoute

Με την παραπάνω εντολή πραγματοποιείται το routing της σχεδίασης. Ο χρήστης δύναται να χρησιμοποιήσει μια σειρά παραμέτρων για να ορίσει με περισσότερη ακρίβεια την διαδικασία, η επιλογή όμως των κριτηρίων έχει να κάνει με τα χαρακτηριστικά της εκάστοτε σχεδίασης και γι αυτό το λόγο δεν θα γίνει αναλυτική περιγραφή όλων των δυνατών παραμέτρων.

- verifyGeometry
- verifyConnectivity
- streamOut full_adder.gds

Το τελευταίο τμήμα του script που παρουσιάζουμε, πραγματοποιεί την επαλήθευση (verification) της σχεδίασης και δημιουργεί το τελικό GDSII αρχείο περιγραφής.

Η εντολή verifyGeometry ελέγχει τα πλάτη, τις αποστάσεις και γενικότερα την εσωτερική γεωμετρία των αντικειμένων που αποτελούν την σχεδίαση, όπως και τον τρόπο με τον οποίο συνδέονται μέσω των καλωδίων που έχουν δημιουργηθεί κατά την διαδικασία του routing.

Τέλος, η εντολή verifyGeometry εντοπίζει σφάλματα όπως ασύνδετα pins και wires, τα οποία πρέπει να διορθωθούν, συνήθως χειρονακτικά, από τον σχεδιαστή. Η προηγούμενη εντολή σε συνδυασμό με την εντολή που μόλις παρουσιάσαμε, δημιουργούν ένα report file, στο οποίο περιγράφονται αναλυτικά όλα τα σφάλματα που έχουν εντοπιστεί και πρέπει να διορθωθούν.

Βιβλιογραφία

1. *VHDL Programming by Example, Douglas L. Perry, McGraw – Hill Fourth*
2. *ASIC Design with Synopsys, Himanshu Bhatnagar*
3. *ModelSIM SE, Manual*
4. *Design Compiler™, Command – Line Interface Guide*
5. *Design Compiler™, Reference Manual*
6. *Design Compiler™, User Guide*
7. *Design Compiler™, Tutorial Using Design Vision*
8. *Design Analyzer™, Reference Manual*
9. *Design Vision™, User Guide*
10. *Power Compiler™, User Guide*
11. *Power Compiler™, Quick Reference Guide*
12. *PrimePower™, Manual*
13. *PrimePower™, Quick Reference Guide*
14. *PrimeTime™, User Guide*
15. *PrimeTime™, Quick Reference Guide*
16. *Using TCL with Synopsys Tools*
17. *Encounter™, Digital Implementation System Text Command Reference*
18. *Encounter™, Digital Implementation System User Guide*
19. *A Comparison of Hierarchical Compile Strategies, Steve Golson*
20. *My Favorite dc_shell Tricks, Steve Golson*
21. *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. *Golshan, K. "Physical Design Essentials: An ASIC Design Implementation Perspective". New York: Springer (2007). ISBN 0-387-36642-3*
24. *Lavagno, Martin and Scheffer. "Electronic Design Automation For Integrated Circuits Handbook". (2006). ISBN 0-8493-3096-3*
25. *Jansen, D. "The Electronic Design Automation Handbook". Kluwer Academic Publishers. (2003). ISBN 1-4020-7502-2*