Υλοποίηση και αξιολόγηση Testbench για Σχεδίαση Κυκλωμάτων Χαμηλής Ισχύος
Implementation and Validation of Low Power Design Testbench

Διπλωματική Εργασία

Νικολοπούλου Λουίζα

Επιβλέποντες Καθηγητές
Σταμούλης Γεώργιος
Ευμορφόπουλος Νέστωρ
Τσομπανοπούλου Παναγιώτα

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

0 ΠΕΡΙΕΧΟΜΕΝΑ

Περιεχόμενα εικόνων

Ευχαριστίες

Με την περάτωση της παρούσας εργασίας θα ήθελα να ευχαριστήσω θερμά μια σειρά ανθρώπων που με υποστήριξαν και με βοήθησαν καθ' όλη τη διάρκεια της εκπόνησής της. Αρχικά, θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές κ. Σταμούλη Γεώργιο, κ. Ευμορφόπουλο Νέστωρα, κα. Τσομπανοπούλου Παναγιώτα για την εμπιστοσύνη που έδειξαν στο πρόσωπό, για την άριστη συνεργασία και συνεχή καθοδήγηση που μου παρείχαν.

Ιδιαίτερα θα ήθελα να ευχαριστήσω τον διδάκτορα κ. Δαδαλιάρη Αντώνιο για την πολύτιμη βοήθειά του, χωρίς αυτή η ολοκλήρωση της παρούσας εργασίας θα ήταν ανέφικτη. Επίσης, πολλά ευχαριστώ θα ήθελα να δώσω και στους συνεργάτες του Ε5 για την υποστήριξη και τις παρεμβάσεις τους.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και τους φίλους μου για την αμέριστη υποστήριξη και την ανεκτίμητη βοήθεια που μου παρείχαν τόσο κατά τη διάρκεια όσο και κατά την εκπόνηση της διπλωματικής μου εργασίας.

Νικολοπούλου Λουίζα

1ΣΥΝΟΨΗ

Στη σημερινή εποχή η τεχνολογία εξελίσσεται ραγδαία, έχουν περάσει ανεπιστρεπτί οι εποχές των πρώτων υπολογιστών με λυχνίες που φαίνονται σχεδόν προϊστορικές. Η τεχνολογία φαίνεται πλέον να κινείται στην κατεύθυνση των πολλών επεξεργαστών πάνω σε ένα και μόνο chip.

Το αποτέλεσμα αυτής της τάσης είναι τα ολοκληρωμένα κυκλώματα, να γίνονται όλο και μικρότερα ενώ οι απαιτήσεις για ταχύτητα να αυξάνονται συνεχώς. Ανακύπτει λοιπόν το ζήτημα της κατανάλωσης ισχύος, η οποία θα πρέπει να είναι η μικρότερη δυνατή δεδομένου των απαιτήσεων. Όπως είναι φανερό πέφτει μεγάλο βάρος στη βελτιστοποίηση των κυκλωμάτων ώστε να αντεπεξέλθουν στα νέα δεδομένα.

Σκοπός της εργασίας είναι να παρουσιάσει τη προσπάθεια βελτιστοποίησης της front-end σχεδίασης ενός κυκλώματος και συγκεκριμένα ενός μικροεπεξεργαστή του Altor32 αρχιτεκτονικής τύπου OpenRisc.

Προκειμένου να πετύχουμε τον στόχο μας, θα παρουσιάσουμε σε αυτήν την εργασία τα εργαλεία τα οποία μπορούν να χρησιμοποιηθούν για την μελέτη, ανάλυση και προσομοίωση ολοκληρωμένων κυκλωμάτων. Στην προσπάθεια αυτή, θα χρησιμοποιήσουμε διαφορετικές τεχνικές βελτιστοποίησης της σχεδίασης

αναζητώντας εκείνη που θα συνδυάζει καλύτερο χρονισμό με μικρότερη καταναλισκόμενη ισχύ.

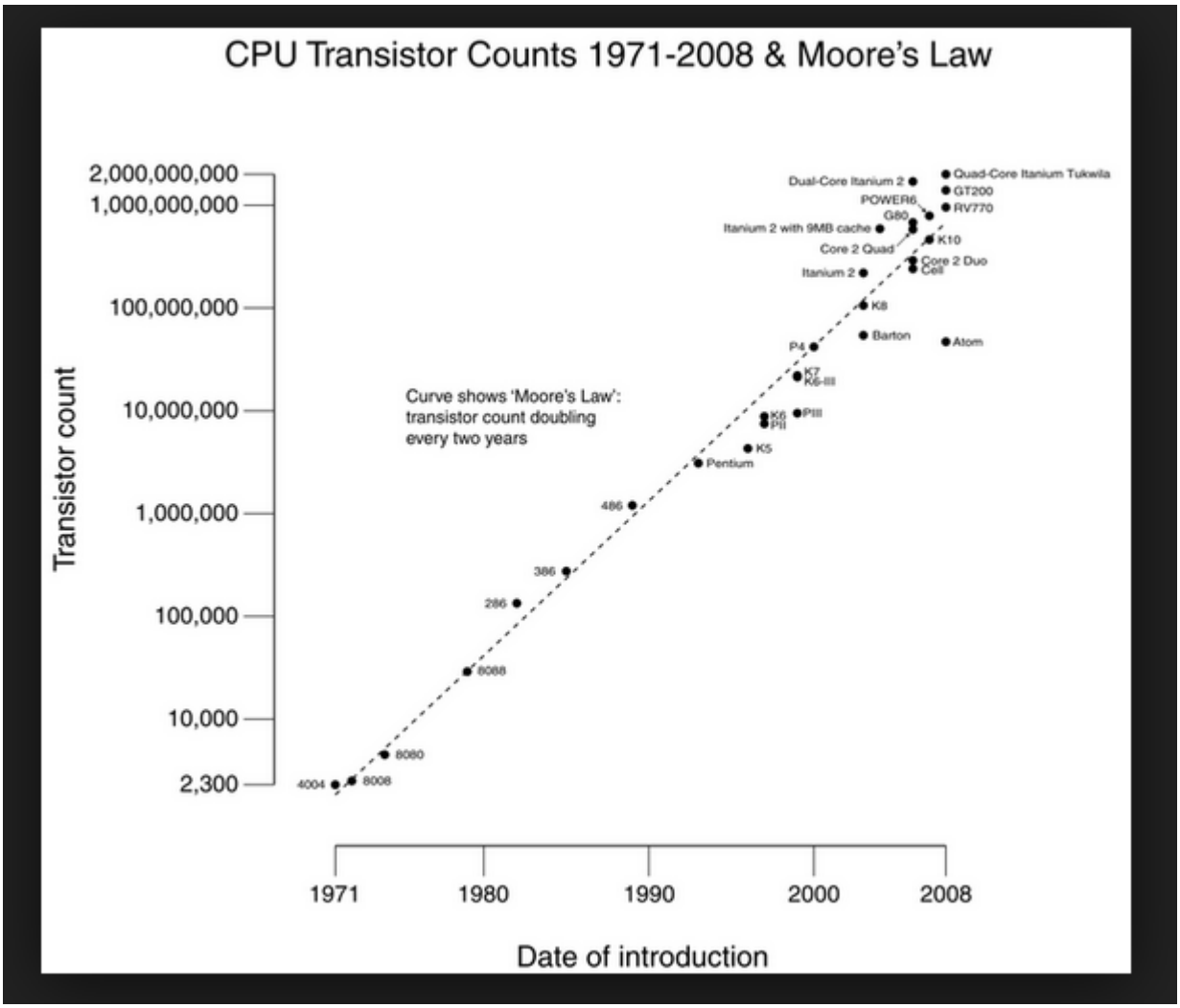
Λέξεις κλειδιά:

Synopsys, Design Compiler, DFT, PrimeTime, Σχεδίαση Ολοκληρωμένων κυκλωμάτων.

2 ΕΙΣΑΓΩΓΗ

Η σχεδίαση ολοκληρωμένων κυκλωμάτων (Integrated Circuit Design –IC Design) αφορά την δημιουργία - υλοποίηση κυκλωματικών στοιχείων όπως μικροεπεξεργαστές (microprocessors), μνήμες (RAM/ROM memories, flash memories), FPGAs (Field Programmable Gate Arrays) και ASICs (Application Specific Integrated Circuits). Τα σύγχρονα ολοκληρωμένα κυκλώματα είναι πολύπλοκα και πολυεπίπεδα, με ένα chip μεγάλου μεγέθους να αποτελείται πλέον από πάνω από ένα δισεκατομμύριο τρανζίστορ. Το γεγονός αυτό, σε συνάρτηση με την συνεχώς αυξανόμενη ανάγκη της αγοράς για παραγωγή ολοκληρωμένων κυκλωμάτων στο μικρότερο δυνατό χρονικό διάστημα έχει οδηγήσει στην ανάπτυξη και χρήση εργαλείων αυτοματοποιημένης σχεδίασης (Automated Design Tools). Η χρήση εργαλείων αυτής της κατηγορίας κρίνεται πλέον απαραίτητη στην πλειοψηφία των βημάτων που ακολουθούνται κατά την σχεδίαση ενός κυκλώματος. Τη ραγδαία αύξηση του αριθμού των τρανζίστορ διατύπωσε πρώτος ο Γκόρντον Μούρ, συνοψίζοντας πως κάθε δύο χρόνια επιτυγχάνεται ο διπλασιασμός του αριθμού των τρανζίστορ που συνθέτουν ένα επεξεργαστή.

2.1 Ο Νομος του Moore



Εικόνα 1 Moore's Law

Ο Γκόρντον Μουρ (Gordon Moore), συνιδρυτής του εταιρείας κατασκευής μικροεπεξεργαστών Intel, περιέγραψε το 1965 σε μια ιστορική μαθηματική μελέτη τους λόγους για τους οποίους η πυκνότητα των τρανζίστορ, δηλαδή ο αριθμός των τρανζίστορ ανά μονάδα επιφάνειας, σε ένα πυκνό ολοκληρωμένο κύκλωμα θα διπλασιάζεται κάθε χρόνο για τουλάχιστον μια δεκαετία από τότε. Το 1975

κοιτάζοντας ξανά τα δεδομένα για την επόμενη δεκαετία αναθεώρησε την «πρόβλεψή» του θέτοντας το διάστημα που απαιτείται για τον διπλασιασμό των τρανζίστορ ενός πυκνού ολοκληρωμένου κυκλώματος στα δύο έτη.

Η επισήμανση του Μουρ μεταμόρφωσε την τεχνολογία των υπολογιστών από μια εξεζητημένη και δαπανηρή δραστηριότητα σε μια καθολική και προσιτή αναγκαιότητα. Άνοιξε δρόμο στη δημιουργία ταχύτερων, μικρότερων, πιο προσιτών οικονομικά τρανζίστορ. Καθώς ολοένα και περισσότερα τρανζίστορ χωρούσαν σε μικρότερο χώρο, η υπολογιστική ισχύς αυξανόταν και η ενεργειακή αποδοτικότητα βελτιωνόταν, με μικρότερο κόστος για τον τελικό χρήστη. Αυτή η εξέλιξη, όχι μόνο προήγαγε τις υφιστάμενες βιομηχανίες και αύξησε την παραγωγικότητα, αλλά έχει αναδείξει εντελώς νέους κλάδους, τροφοδοτούμενους από την ισχυρή και οικονομικά προσιτή τεχνολογία.

Οι επιδόσεις και το κόστος είναι οι δύο κινητήριοι μοχλοί της τεχνολογικής ανάπτυξης. Όλη η μοντέρνα υπολογιστική τεχνολογία που γνωρίζουμε και απολαμβάνουμε σήμερα, ξεπήδησε από τα θεμέλια που έθεσε ο νόμος του Μουρ. Από το Internet, τα social media και τα μοντέρνα data analytics, θεωρείται ότι όλες αυτές οι καινοτομίες συνδέονται με τον Μουρ και τα ευρήματά του. Οι οικονομικά προσιτοί και αναρίθμητοι υπολογιστές που μας περιστοιχίζουν, αλλάζουν τον τρόπο που δουλεύουμε, παίζουμε και επικοινωνούμε. Η θεμελιώδης δύναμη του νόμου του Μουρ έχει οδηγήσει σε ανακαλύψεις στις σύγχρονες πόλεις, στις συγκοινωνίες, την υγεία, την εκπαίδευση, και την παραγωγή ενέργειας.

Καθώς προχωρούμε στο μέλλον, ο νόμος του Μουρ και οι καινοτομίες που απορρέουν από αυτόν, οδηγούν στην ολοένα αυξανόμενη ενσωμάτωση των υπολογιστών στην καθημερινότητά μας.

Η πρόβλεψη επαληθεύθηκε από την πραγματικότητα, καθώς έκτοτε ο αριθμός των τρανζίστορ ενός ολοκληρωμένου κυκλώματος διπλασιάζεται κάθε δύο

χρόνια. Η πρόβλεψη του Μουρ, ύστερα από την πρακτική επαλήθευσή της, ονομάστηκε *«Νόμος του Μουρ»*.

Ο Ν**όμος του Moore**, με την έννοια της φρενήρους προόδου στο χώρο των μικροηλεκτρονικών, συνεχίζει να ισχύει χάρη σε νέες τεχνικές μεθόδους κατασκευής. Με την εμφάνιση των πρώτων επεξεργαστών στα 14nm, ο Νόμος του Moore θα συνεχίσει να υφίσταται τουλάχιστον μέχρι να φτάσουμε το όριο των 7 νανομέτρων, εκεί που τα **κβαντικά φαινόμενα** καθιστούν επιβεβλημένη την ριζική αλλαγή των υλικών και των μεθόδων που χρησιμοποιούνται για την κατασκευή chips.

Ήδη οι εταιρείες που βρίσκονται στην αιχμή της τεχνολογίας κατασκευής chips, δηλαδή Intel, IBM, Samsung και TSMC, βρίσκονται σε συνεννόηση και μοιράζονται πολλά από τα ευρήματα των ερευνών τους.

2.2 Εργαλεια CAD / EDA

Η ραγδαία αύξηση του αριθμού των τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα έκανε επιτακτική την ανάγκη χρησιμοποίησης της τεχνολογίας των υπολογιστών για την διευκόλυνση της διαδικασίας σχεδιασμού ενός ολοκληρωμένου κυκλώματος. Σαν αποτέλεσμα δημιουργήθηκαν ειδικά σχεδιασμένα λογισμικά που υποβοηθούν στο σχεδιασμό και στην υλοποίηση ενός κυκλώματος.

Εικόνα 2 VLSI CAD Tools Vendors

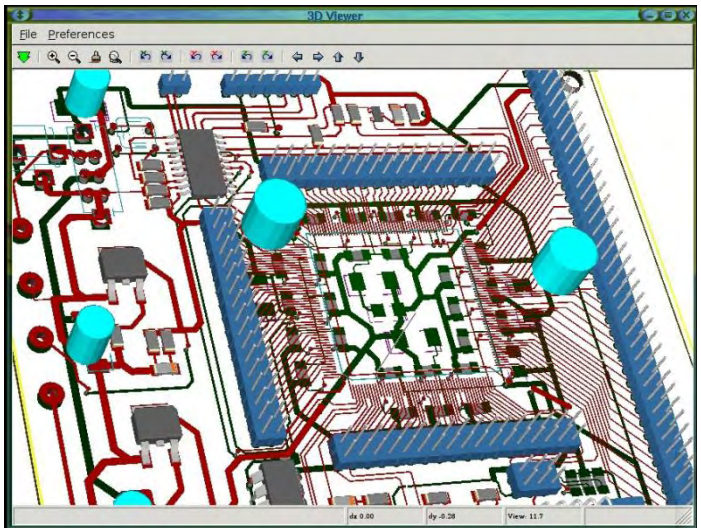
Όταν αναφερόμαστε στο όρο **Computer Aided Design** (CAD), αναφερόμαστε πρακτικά στην χρησιμοποίηση της τεχνολογίας των υπολογιστών για την διευκόλυνση της διαδικασίας σχεδιασμού ενός αντικειμένου.

Η αυτοματοποιημένη ηλεκτρονική σχεδίαση **(Electronic Design Automation) EDA** είναι η υλοποίηση ενός κυκλώματος με παράλληλη χρήση ειδικών λογισμικών τα οποία έχουν δημιουργηθεί κατά περίσταση γα την υποβοήθηση της διαδικασίας.

2.3 Ιστορικη Αναδρομη

Πριν από την αυτοματοποίηση της διαδικασίας, ο σχεδιασμός των ολοκληρωμένων κυκλωμάτων πραγματοποιούνταν χειρωνακτικά. Ο σχεδιασμός

βασιζόταν στην γραφική απεικόνιση του κυκλώματος και πιο συγκεκριμένα στην τροποποίηση της ηλεκτρονικής περιγραφής του κυκλώματος στην γραφική αναπαράστασή του. Το πρόβλημα που είχε η διά χειρός σχεδίαση ενός κυκλώματος ήταν η πιθανότητα το τελικό κύκλωμα να μην είναι λειτουργικό ακόμη και να εκτελεί λανθασμένες εργασίες.



Εικόνα 3 3D PCB Layout

Η δεκαετία του ‘70 σηματοδοτεί την αρχή της ανάπτυξης των πρώτων εργαλείων χωροθέτησης. Οι καινοτομίες που εισήχθησαν κατά την περίοδο αυτή αποτέλεσαν τη βάση όλης της ερευνητικής δραστηριότητας του τομέα τα χρόνια που ακολούθησαν.

Τα πρώτα EDA tools εμφανίστηκαν εντός ακαδημαϊκού περιβάλλοντος. Το VLSI Tools Tarball, ένα από τα δημοφιλέστερα εργαλεία της πρώιμης αυτής εποχής, αναπτύχθηκε στο πανεπιστήμιο του Berkley και ήταν κατ’ ουσίαν μια συλλογή εφαρμογών σε περιβάλλον UNIX για το σχεδιασμό VLSI συστημάτων. Μέχρι τις αρχές της δεκαετίας του ‘80, οι μεγαλύτερες εταιρίες του κλάδου ανέπτυσσαν εργαλεία αυτής της κατηγορίας προς ιδίαν χρήση (in house), χωρίς να παρέχεται η

δυνατότητα χρήσης ή απόκτησής τους από άλλους φορείς. Το πρόβλημα αυτό αναγνωρίστηκε από πολλούς σχεδιαστές οι οποίοι αναλογιζόμενοι το τεχνολογικό και οικονομικό ενδιαφέρον που παρουσίαζε ο τομέας, αποφάσισαν να ασχοληθούν με την παραγωγή EDA tools σε βιομηχανική κλίμακα. Η αλλαγή στάσης απέναντι στο τρόπο χρήσης και διάθεσης των εργαλείων γίνεται αντιληπτή και από το γεγονός πως κατά την περίοδο αυτή ιδρύθηκαν ορισμένες από τις σημαντικότερες εταιρίες του κλάδου, όπως η Mentor Graphics και η Valid Logic Systems. Επιπρόσθετα κατά την δεκαετία του ‘80 έχουμε και την ανάπτυξη δύο υψηλού επιπέδου γλωσσών περιγραφής υλικού της VHDL και της Verilog, η χρήση των οποίων ως προτύπων άνοιξε το δρόμο για την δημιουργία των πρώτων εργαλείων λογικής σύνθεσης (logic synthesis).

Τα προγράμματα αυτά οδήγησαν στο να μειωθεί η άμεση ανάγκη επιπλέον εξειδικευμένου προσωπικού, δίνοντας περαιτέρω δυνατότητες ανάπτυξης σε εταιρείες μεσαίου μεγέθους. Το γεγονός πως, σε σύντομο σχετικά χρονικό διάστημα, από την πρώιμη εμφάνισή τους, οι εταιρείες κατέστησαν οικονομικά προσιτή την απόκτησή τους και την χρήση τους στον προσωπικό υπολογιστή ενός μέσου χρήστη, βοήθησε στο μέγιστο βαθμό τους σχετιζόμενους με το αντικείμενο κλάδους μηχανικής.

Οι σύγχρονες ροές σχεδίασης ολοκληρωμένων κυκλωμάτων αποτελούνται πλέον από πολλαπλά βήματα, σε κάθε ένα από τα οποία γίνεται χρήση του κατάλληλου εργαλείου. Στην αρχή της προαναφερθείσας διαδικασίας έχουμε κατά κανόνα την δημιουργία μιας περιγραφής του κυκλώματος βάσει κάποιας HDL γλώσσας σε επίπεδο κελιών, τα οποία είναι τεχνολογικά ανεξάρτητα. Ακολούθως, ο σχεδιαστής παρέχει τις κατάλληλες τεχνολογικές βιβλιοθήκες οι οποίες συνδράμουν στην επιτυχή και λεπτομερή προσομοίωση της λειτουργίας του κυκλώματος, ενώ κατά το τελευταίο στάδιο παρέχονται στον σχεδιαστή οι τελικές προδιαγραφές για τις συνθήκες λειτουργίας τους.

2.4 ΤΟΜΕΙΣ ΕΦΑΡΜΟΓΗΣ

Τα εργαλεία CAD/EDA αποτελούν την πλέον ενδεδειγμένη λύση κατά την σχεδίαση ολοκληρωμένων κυκλωμάτων και βρίσκουν εφαρμογή στους ακόλουθους τομείς:

- DESIGN

  Αναφέρεται στην περιγραφή σχεδίασης με τη χρήση κάποιας γλώσσας περιγραφής υλικού.

  - ✓ High Level Synthesis
  - ✓ Logic Synthesis
  - ✓ Schematic Capture
  - ✓ Layout

- SIMULATION

  Αναφέρεται στη μοντελοποίηση της συμπεριφοράς του συστήματος και στον έλεγχό της με την παροχή κατάλληλων ερεθισμάτων.

  - ✓ Logic Simulation
  - ✓ Behavioral Simulation
  - ✓ Hardware Emulation

- ANALYSIS & VERIFICATION

  Αναφέρεται στον έλεγχο της ορθότητας της σχεδίασης.

- ✓ Functional Verification
- ✓ Formal Verification
- ✓ Equivalence Checking
- ✓ Timing Analysis
- ✓ Physical Verification

- MANUFACTURING

  Αναφέρεται στην φυσική κατασκευή ενός κυκλώματος

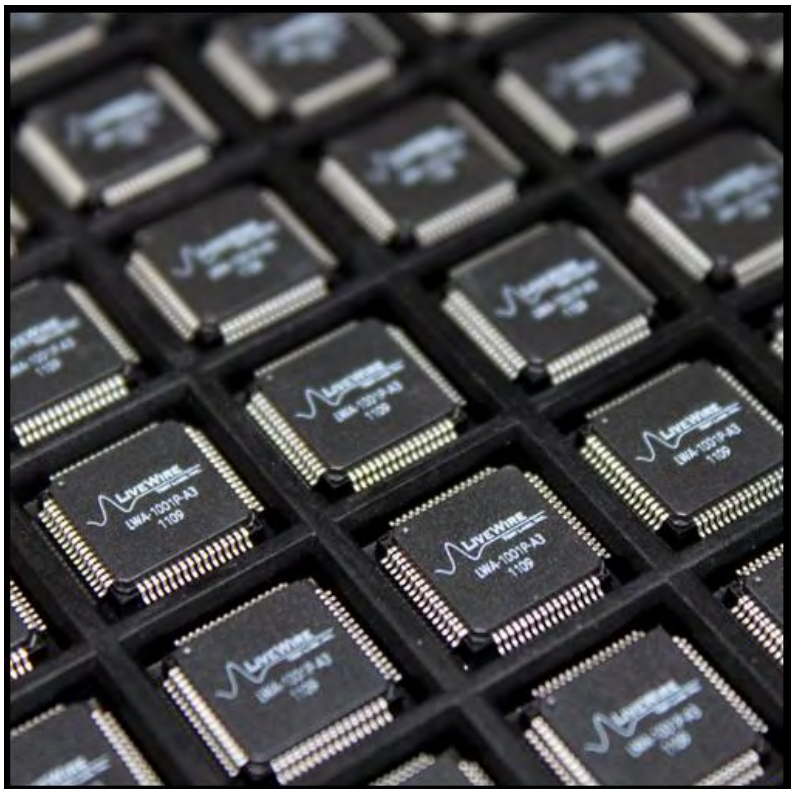
  - ✓ Mask Data Preparation
    - Resolution enhancement techniques
    - Optical proximity correction
    - Mask generation
    - Automatic test pattern generation
    - Built-in self-test

3 DESIGN FLOW

Η Ροή Σχεδίασης προϋποθέτει το συνδυασμό ενός πλήθους από EDA για την ορθή ολοκλήρωση του σχεδιασμού ενός ολοκληρωμένου κυκλώματος.

3.1 ΒΑΣΙΚΕΣ ΈΝΝΟΙΕΣ

Τα ASICs (Application Specific Integrated Circuits) είναι κυκλώματα ειδικού σκοπού, οι σχεδιάσεις των οποίων προορίζονται για συγκεκριμένη χρήση (π.χ μικροεπεξεργαστές, μνήμες κλπ). Τα VLSI (Very Large Scale Integration) χαρακτηρίζονται πρακτικά οποιαδήποτε chip με μεγάλο αριθμό από τρανζίστορ. Τα FPGAs είναι απεικόνιση σε προγραμματιζόμενο πίνακα πυλών, βασίζονται σε ολοκληρωμένα που πέρα από τρανζίστορ, περιέχουν και προ-υλοποιημένο πίνακα πιθανών διασυνδέσεων μεταξύ τους.



Εικόνα 4 A tray of Application-specific integrated circuit (ASIC) chips.

Υπάρχουν διάφορες τεχνοτροπίες σχεδιασμού ενός ολοκληρωμένου κυκλώματος. Ο σχεδιαστής πρέπει να επιλέξει κάποια από αυτές τις τεχνοτροπίες όταν ο στοχευόμενος σχεδιασμός δεν είναι διαθέσιμος σαν ένα εμπορικό ολοκληρωμένο, ή όταν τα διαθέσιμα δεν καλύπτουν κάποιες ή όλες από τις ανάγκες του σχεδιαστή για ταχύτητα, εμβαδόν και κατανάλωση ισχύος. Οι διαθέσιμες τεχνοτροπίες είναι :

- Πλήρως εξειδικευμένος σχεδιασμός (Full – custom design). Στην τεχνοτροπία αυτή ο σχεδιαστής θα πρέπει να σχεδιάσει εξ αρχής, ακόμη και τα βασικά δομικά στοιχεία, δηλαδή τις πύλες και τα στοιχεία μνήμης του σχεδιασμού του. Παρότι αυτός ο τρόπος παρέχει τη μέγιστη ευελιξία στον σχεδιαστή, είναι προφανές ότι είναι και ο πλέον επίπονος χρονικά. Επιπλέον η πιθανότητα για σχεδιαστικά λάθη είναι πολύ μεγάλη και το κόστος

κατασκευής πολύ υψηλό. Η τεχνοτροπία αυτή σήμερα χρησιμοποιείται για μικρούς σχετικά σχεδιασμούς με πολύ αυξημένες απαιτήσεις σε ταχύτητα, εμβαδόν και κατανάλωση ισχύος.

- Μερικά εξειδικευμένος σχεδιασμός (Semi-custom design). Στην τεχνοτροπία αυτή παρότι μερικά σχεδιαστικά κομμάτια παρέχονται έτοιμα σα μια βιβλιοθήκη, ο σχεδιαστής έχει τη δυνατότητα αν δε καλύπτεται να σχεδιάσει τα δικά του και μετά να φτιάξει το σχεδιασμό του σα μίγμα έτοιμων και νέων υποσχεδιασμών. Προφανώς η ευελιξία που δίνεται στο σχεδιαστή είναι αντίστοιχη με αυτήν της προηγούμενης περίπτωσης, αλλά επίσης ο χρόνος σχεδιασμού μπορεί να μικρύνει σημαντικά. Χρησιμοποιώντας κατά πλειοψηφία έτοιμα σχεδιαστικά κομμάτια, η πιθανότητα σχεδιαστικών λαθών μικραίνει.

- Σχεδιασμός με έτοιμα σχεδιαστικά κομμάτια (Standard-cell design). Στην τεχνοτροπία αυτή ο σχεδιαστής εφοδιάζεται με μια σχεδιαστική βιβλιοθήκη και εκφράζει το σχεδιασμό του σαν την διασύνδεση στοιχείων αυτής της βιβλιοθήκης. Προφανώς η βιβλιοθήκη αυτή παρέχεται από τον τελικό κατασκευαστή του ολοκληρωμένου και μπορεί να περιέχει από πολύ λίγα έως πάρα πολλά και πολύ σύνθετα σχεδιαστικά κομμάτια. Η βιβλιοθήκη θα μπορούσε να περιέχει μόνο τη πύλη NAND δύο εισόδων μιας και κάθε λογική συνάρτηση μπορεί να εκφραστεί βάσει αυτής. Ωστόσο για τη διευκόλυνση των σχεδιαστών οι βιβλιοθήκες που παρέχονται σήμερα περιέχουν όλες τις λογικές πύλες (και μάλιστα σε διάφορες εκδόσεις ταχύτητας, εμβαδού και οδηγητικής ικανότητας), στοιχεία μνήμης, μικρά έως μεσαία συνδυαστικά κυκλώματα (αθροιστές, πολλαπλασιαστές, κλπ), μικρά έως μεσαία ακολουθιακά κυκλώματα (καταχωρητές, ολισθητές, μετρητές κλπ). Το μεγάλο πλεονέκτημα χρησιμοποίησης αυτής της τεχνοτροπίας είναι

προφανώς ο χρόνος ολοκλήρωσης του σχεδιασμού. Ωστόσο, ξεφεύγει πλέον από τα χέρια του σχεδιαστή η δυνατότητα καθορισμού των ηλεκτρικών χαρακτηριστικών των στοιχειωδών σχεδιαστικών κομματιών, με αποτέλεσμα τόσο οι μέγιστες ταχύτητες, όσο και το ελάχιστο εμβαδόν και η ελάχιστη κατανάλωση ισχύος που μπορεί να επιτευχθεί να μη μπορούν να καθοριστούν άμεσα από αυτόν.

Η σχεδίαση μας βασίζεται στην τεχνοτροπία των Standard-cell.

3.2 Βασικη Ροη Σχεδιασης Ολοκληρωμενων Κυκλωματων

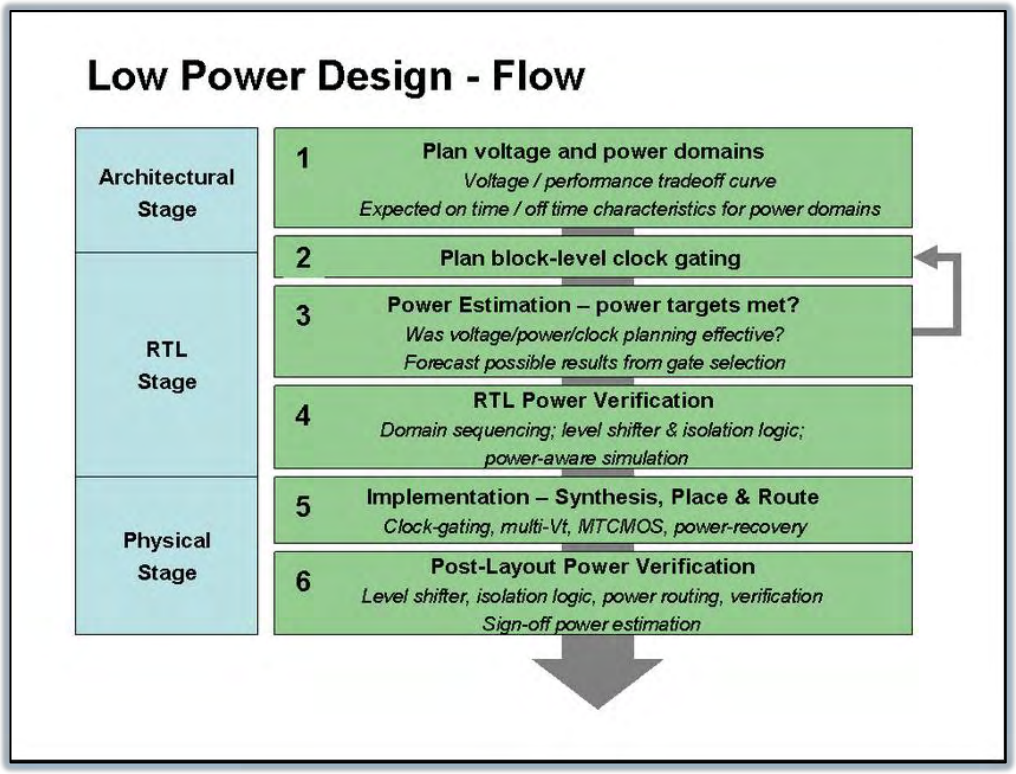
Μια σχεδίαση ολοκληρωμένου κυκλώματος χονδρικά περιλαμβάνει 3 βασικά επίπεδα:

1) **System Level Design –** ορισμός των λειτουργικών προδιαγραφών.

2) **RTL Design (Front-End) –** μετατροπή προδιαγραφών χρήστη σε επίπεδο RTL. Το RTL περιγράφει την ακριβή συμπεριφορά των ψηφιακών κυκλωμάτων, καθώς και τις διασυνδέσεις εισόδων, εξόδων.

3) **Physical Design (Back-End) –** συνδυασμός ενός RTL με μια βιβλιοθήκη που συμπεριλαμβάνει τις διαθέσιμες πύλες για τη σχεδίαση ενός chip. Για την επίτευξη αυτού, πρέπει να καθοριστούν ποιες πύλες θα χρησιμοποιηθούν, σε ποιο σημείο θα τοποθετηθούν πάνω στο chip και ποια θα είναι η καλωδίωση μεταξύ τους. Σε αυτό το στάδιο, οι κυκλωματικές αναπαραστάσεις των στοιχείων (συστήματα και διασυνδέσεις) της σχεδίασης μετατρέπονται σε

γεωμετρικές αναπαραστάσεις σχημάτων, που όταν κατασκευαστούν με τα κατάλληλα στρώματα από υλικά, θα εξασφαλίσουν την απαιτούμενη λειτουργία.



Εικόνα 5 Low Power Design Flow

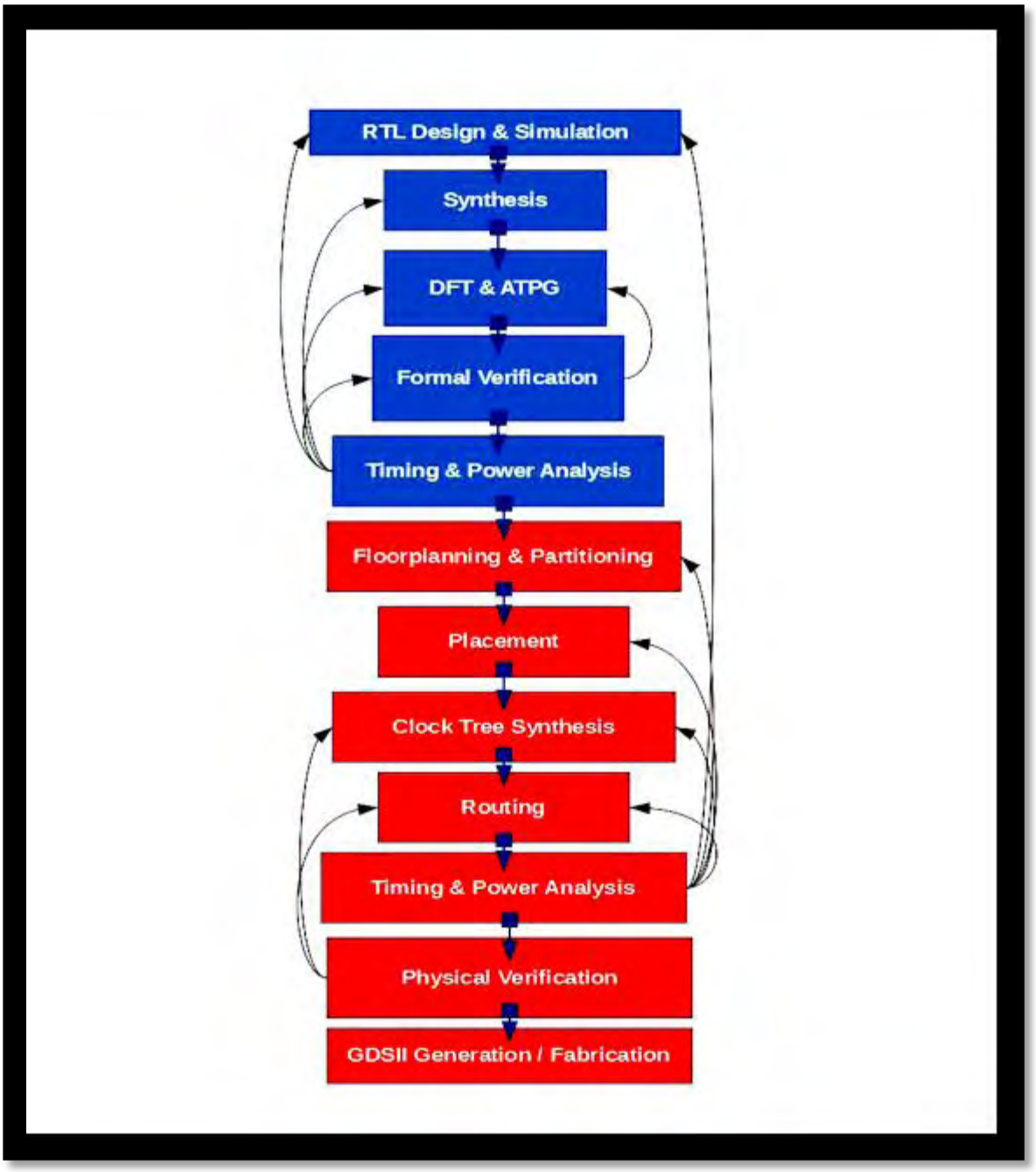
Καθένα από τα παραπάνω επίπεδα περιλαμβάνει μια σειρά από επιμέρους βήματα. Θα εξετάσουμε πιο αναλυτικά τα βήματα από τα οποία αποτελούντα τα επίπεδα της σχεδίασης RTL ή όπως ονομάζεται αλλιώς Front-End σχεδίαση και της Physical ή διαφορετικά Back-End σχεδίασης εκτενέστερα σε επόμενο κεφάλαιο.

3.3 Ροη Σχεδιασης Κυκλωματων Τυπου ASIC

Οι προκλήσεις της αυξανόμενης καθυστέρησης μεταξύ ων διασυνδέσεων (inter connection delay) οδήγησε σε έναν νέο τρόπο σκέψης σχετικά με την σχεδίαση και την υλοποίηση των EDA. Νέες προκλήσεις, όπως η διαρροή ρεύματος (leakage power), η μεταβλητότητα (variability) και η αξιοπιστία (reliability) θα συνεχίσουν να αποτελούν λόγο να επαναπροσδιορίζονται οι τεχνικές που χρησιμοποιούνται στην περάτωση του σχεδιασμού ενός ολοκληρωμένου κυκλώματος. Υπάρχουν δύο διακριτές ροές σχεδίασης, μια για την υλοποίηση ASIC και μια την υλοποίηση VLSI. Η εργασία αυτή βασίζεται στη ροή σχεδίασης που αφορά κυκλώματα τύπου ASICεπικεντρώνεται στη Front-End σχεδίαση ενός τέτοιου τύπου κυκλώματος.

Η ροή σχεδίασης που αφορά κυκλώματα τύπου ASIC χωρίζεται σε δυο επιμέρους ροές:

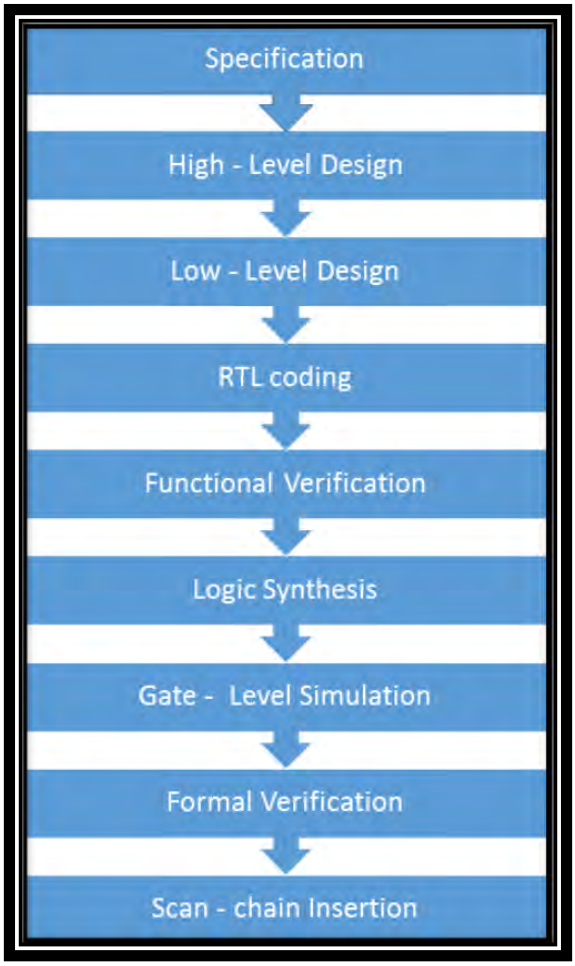
- Front-end flow
- Back-end flow



Εικόνα 6 Simplified ASIC Design Flow. Με μπλε χρώμα απεικονίζεται η front-end σχεδίαση και με κόκκινο η back-end σχεδίαση

3.3.1 Front-End flow

Η Front-end ροή είναι η διαδικασία που οδηγεί από την έννοια του netlist σε αυτή των λογικών πυλών. Περιλαμβάνει στάδια όπως τον αρχιτεκτονικό σχεδιασμό, τη προσομοίωση και τη σύνθεση. Η front-end ροή ολοκληρώνεται με το βήμα της Λογικής Σύνθεσης.



Εικόνα 7 Front-End Design Flow

Αναλυτικότερα :

- Specification: Ανάλυση του απώτερου στόχου, εκτίμηση των προβλημάτων που μπορεί να παρουσιαστούν και των πόρων που έχουμε στη διάθεσή μας για την υλοποίηση του τελικού αποτελέσματος.

- High-level design: Διάφορα blocks ορίζονται και περιγράφεται η επικοινωνία μεταξύ τους. Η περιγραφή δίνεται σε υψηλού επιπέδου γλώσσες (System C, C, C++).

- Low-level design: Περιγράφεται η υλοποίηση κάθε μπλοκ ξεχωριστά. Περιέχει λεπτομέρειες σχετικά με Finite State Machines (FSMs), μετρητές, καταχωρητές κ.λ.π.

- RTL coding: Το στάδιο κατά το οποίο μια σχεδίαση επιπέδου Low-Level εκφράζεται με τη βοήθεια μιας γλώσσας Hardware Description Languages (HDL) χρησιμοποιώντας στοιχεία τα οποία μπορούν να συνθεθούν.

- Functional Verification: Επαληθεύεται ότι η σχεδίαση έχει την αναμενόμενη λειτουργία. Δημιουργούνται Testbenches έτσι ώστε να εφαρμοστούν όλα τα πιθανά “ερεθίσματα” στην είσοδο της σχεδίασης με σκοπό τον έλεγχο όλων των πιθανών αποτελεσμάτων.

- Logic Synthesis: Η διαδικασία κατά την οποία το εργαλείο σύνθεσης δέχεται ως είσοδο τη περιγραφή της λειτουργίας του κυκλώματος σε επίπεδο καταχωρητών, την τεχνολογία που θέλουμε να χρησιμοποιήσουμε και τους περιορισμούς που έχουμε θέσει στην

σχεδίαση και επιστρέφει την σχεδίαση σε επίπεδο λογικών πυλών με την χρήση των πυλών της βιβλιοθήκης που δόθηκε ως είσοδος. Αφού παραχθεί η σχεδίαση σε επίπεδο λογικών πυλών (gate-level netlist) γίνεται ανάλυση χρονισμού για να ελεγχθεί αν η σχεδίαση καλύπτει τις απαιτήσεις σε επίπεδο χρονισμού.

- Gate-level Simulation: Ελέγχει εάν η υπό έλεγχο σχεδίαση Design Under Test (DUT) είναι λειτουργικά σωστή.

Προτού προχωρήσουμε στη back-end ροή της σχεδίασης συνήθως προηγείται το στάδιο της Τυπικής Επαλήθευση (Formal Verification) και η εισαγωγή αλυσίδων σάρωσης (scan-chain).

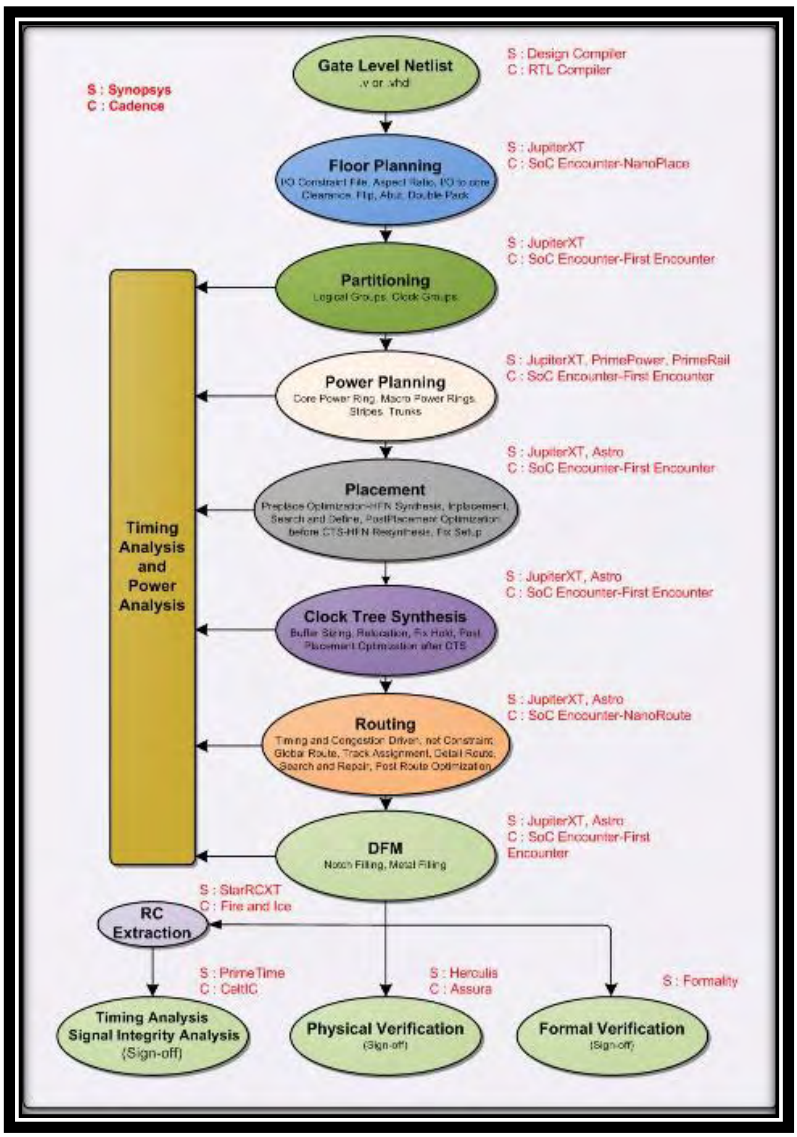
- Formal verification: Ελέγχεται αν η μετατροπή της σχεδίασης από το επίπεδο καταχωρητών στο επίπεδο των λογικών πυλών είναι σωστή.

- Scan-chain insertion: Εισαγωγή αλυσίδων σάρωσης (scan-chain) σε περίπτωση που το ASIC αποτελεί σχεδίαση για έλεγχο Design For Testability (DFT).

Στο κεφάλαιο 5 θα παρουσιάσουμε αναλυτικά τα βήματα που ακολουθήσαμε για τη δική μας σχεδίαση.

3.3.2 Back-End flow

Η Back-end ροή ή η Φυσική Σχεδίαση είναι το στάδιο στη καθιερωμένη ροή σχεδίασης που ακολουθεί την Front-end ροή. Σε αυτό το στάδιο, οι κυκλωματικές αναπαραστάσεις των στοιχείων (συστήματα και διασυνδέσεις) της σχεδίασης μετατρέπονται σε γεωμετρικές αναπαραστάσεις σχημάτων, που όταν κατασκευαστούν με τα κατάλληλα στρώματα από υλικά, θα εξασφαλίσουν την απαιτούμενη λειτουργία. Το επόμενο στάδιο μετά την Φυσική Σχεδίαση είναι η διαδικασία της Κατασκευής του συστήματος που υλοποιείται στα Wafer Fabrication Houses.



Εικόνα 8 Back-End Design Flow

Αναλυτικότερα η Back-end ροή :

- Gate-level netlist: Η σχεδίαση του κυκλώματος μετά την ολοκλήρωση της Front-end ροής.

- Floorplanning: Είναι η διαδικασία προσδιορισμού των δομών που πρέπει να τοποθετηθούν κοντά μεταξύ τους, και τον καθορισμό του χώρου που

απαιτείται κατά τέτοιο τρόπο ώστε να πληρούν τους ενίοτε αλληλοσυγκρουόμενους στόχους του διαθέσιμου χώρου (το κόστος του τσιπ), τις απαιτούμενες επιδόσεις, και την επιθυμία να έχουμε τα πάντα κοντά μεταξύ τους.

- Partitioning: Είναι η διαδικασία της διαίρεσης του τσιπ σε μικρότερα τετράγωνα. Αυτό γίνεται κυρίως για να διαχωριστούν τα διάφορα λειτουργικά τμήματα και επίσης να κάνει την τοποθέτηση και την δρομολόγηση της σχεδίασης ευκολότερη. Ο διαχωρισμός μπορεί να γίνει στη φάση του Register Transfer Level (RTL) σχεδιασμού , όταν οι σχεδιαστές χωρίζουν το σύνολο του σχεδιασμού σε επιμέρους τμήματα και στη συνέχεια προχωρούν στο σχεδιασμό κάθε επιμέρους ενότητας ξεχωριστά. Αυτές οι επιμέρους μονάδες εν τέλει συνδέονται μεταξύ τους και αποτελούν την τελική σχεδίαση.

- Placement: Είναι η διαδικασία της τοποθέτησης των κελιών της σχεδίασης στην περιοχή του πυρήνα.

- Clock Tree Synthesis (CTS): Πριν από την CTS φάση, το σήμα του ρολογιού δεν μεταβιβάζεται και θεωρείται ιδανικό. Το δέντρο του σήματος του ρολογιού ξεκινά από την πηγή που παράγει τον παλμό του ρολογιού και καταλήγει στις εισόδους των Flip-flop.

- Routing: Υπάρχουν δύο τύποι δρομολόγησης της φυσικής σχεδίασης, η Global routing και η Detailed routing. Με την Global routing δεσμεύονται οι πόροι δρομολόγησης που χρησιμοποιούνται για τις συνδέσεις. Η Detailed routing αναθέτει σε συγκεκριμένες διαδρομές μεταλλικά στρώματα και αγωγούς, εντός των συνολικών πόρων δρομολόγησης.

- Signoff: Ελέγχει την ορθότητα του φυσικού σχεδίου, πριν να είναι έτοιμο για παραγωγή. Υπάρχουν διάφορες κατηγορίες Signoff ελέγχων.

  - Design Rule Checking (DRC): Επίσης γνωστός και ως γεωμετρικός έλεγχος. Περιλαμβάνει την εξακρίβωση του αν ο σχεδιασμός μπορεί να κατασκευαστεί αξιόπιστα λαμβάνοντας υπόψη τους σημερινούς περιορισμούς της μεθόδου της φωτολιθογραφίας.

  - Layout Versus Schematic (LVS): Επίσης γνωστή ως Σχηματική Επαλήθευση. Χρησιμοποιείται για να βεβαιωθούμε ότι η τοποθέτηση και δρομολόγηση των κελιών της σχεδίασης δεν έχει αλλάξει τη λειτουργικότητα του ως προς κατασκευή κυκλώματος.

  - Formal Verification: Επαληθεύεται για το αν η λειτουργικότητα της σχεδίασης μετά το τέλος της Φυσικής Σχεδίασης είναι η αναμενόμενη.

  - Voltage-drop analysis: Επίσης γνωστή και ως IR-drop ανάλυση. Σε αυτό το στάδιο επαληθεύεται εάν η ισχύς του πλέγματος είναι αρκετά ισχυρή ώστε να εξασφαλίσει ότι η τάση που εκπροσωπεί το κάθε κελί δεν ‘πέφτει‘ ποτέ κάτω από ένα προκαθορισμένο περιθώριο.

  - Signal-integrity analysis: Ελέγχεται η ποιότητα του σήματος, αναλύεται ο θόρυβος που οφείλεται στη αλληλεπίδραση μεταξύ των σημάτων και ελέγχεται η επίδρασή του στη λειτουργικότητα του κυκλώματος.

- Static Timing Analysis (STA): Χρησιμοποιείται για να εξακριβωθεί εάν όλα τα λογικά μονοπάτια δεδομένων στο σχεδιασμό μπορούν να λειτουργήσουν με τη προβλεπόμενη συχνότητα ρολογιού.

- Electro-migration Lifetime Check (ELC): Ελέγχει το κύκλωμα ώστε να εξασφαλίσει ένα ελάχιστο χρόνο ζωής λειτουργίας του κυκλώματος στη προβλεπόμενη συχνότητα ρολογιού χωρίς το κύκλωμα να υποκύψει στο φαινόμενο της Ηλεκτρομετανάστευσης.

4 CPU DESIGN DESCRIPTION

Πριν ξεκινήσει η σχεδίαση ενός ολοκληρωμένου κυκλώματος, αρχικά πρέπει να περιγραφεί με 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. Ακολουθεί μια συνοπτική περιγραφή των βασικότερων χαρακτηριστικών τους.

4.1 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 transferlevel - RTL), ενώ η χρήση της ενδείκνυται και για την επαλήθευση αναλογικών και υβριδικών σχεδιάσεων.

Η διαφορά της γλώσσας Verilog από τις υπόλοιπες γλώσσες προγραμματισμού είναι 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 επικοινωνήσουν 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, αξίζει να σημειωθεί πως μόνο ένα υποσύνολο από τις δηλώσεις ενός κώδικα Verilog μπορεί να χρησιμοποιηθεί ως είσοδος σε ένα εργαλείο σύνθεσης, γι’ αυτό και πρέπει να είμαστε ιδιαίτερα προσεκτικοί κατά την συγγραφή του κώδικα.

4.2 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 περαιτέρω 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.

4.3 VERILOG VS VHDL

Το κύκλωμα που απεικονίζεται στην παρακάνω εικόνα είναι ένας πλήρης αθροιστής. Ακολουθεί η περιγραφή του σε Verilog και VHDL.

Κάνοντας μια σύγκριση ανάμεσα στις δύο γλώσσες βλέπουμε πως η Verilog έχει πιο μικρό συντακτικό, το οποίο μοιάζει αρκετά με τη γλώσσα προγραμματισμού C. Οι περιγραφές χαμηλού επιπέδου είναι πιο κοντά στο πραγματικό hardware. Μπορούμε να δημιουργήσουμε άμεσα τα στιγμιότυπα

οποιαδήποτε πύλης και να δηλώσουμε πάλι άμεσα τα καλώδια και τους καταχωρητές που αποτελούν το κύκλωμα. Από την άλλη πλευρά δεν έχει τη δυνατότητα να δημιουργήσει νέους τύπους δεδομένων, καθώς και η διαφορά μεταξύ ενός καλωδίου και ενός καταχωρητή δεν είναι πάντοτε εμφανής στον απλό χρήστη.

Verilog Description

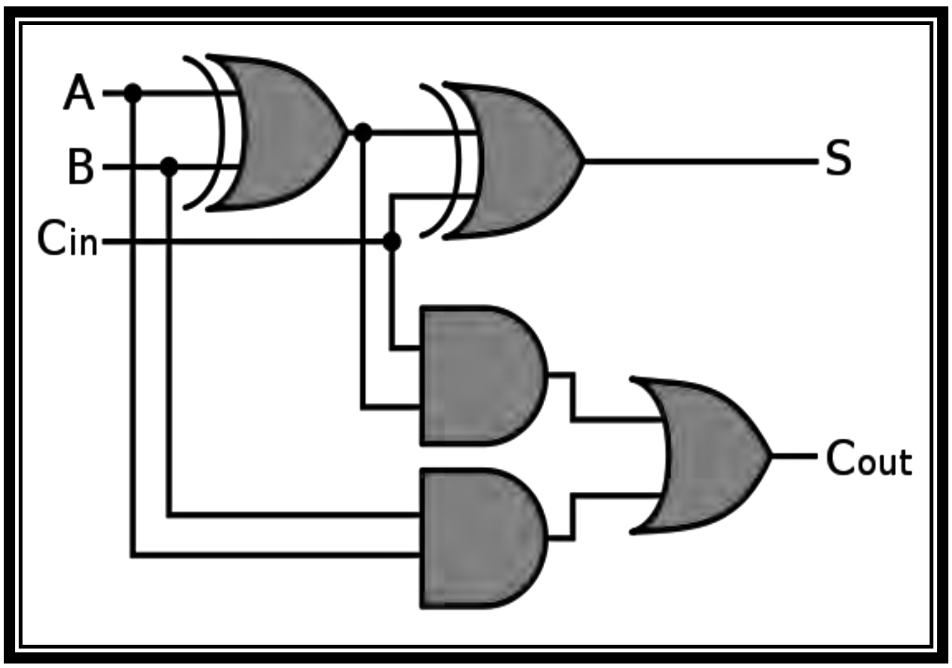
```
module fulladd(A, B, Cin, S, Cout);
        input A;
        input B;
        input Cin;
        output S;
        output Cout;
        wire S, Cout;
        assign sum=a^b^c;
        assign carry=((a&b) | (b&c) | (a&c));
endmodule
```

VHDL Description

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY fulladd IS
PORT (          A:      IN STD_LOGIC;
                B:      IN STD_LOGIC;
                Cin:    IN STD_LOGIC;
                S:      OUT STD_LOGIC;
                Cout:   OUT STD_LOGIC);
    END fulladd ;
    ARCHITECTURE beh OF fulladd IS
    BEGIN
        S <=A XOR B XOR Cin ;
        Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;
    END beh ;
```

Εικόνα 9 VHDL code versus Verilog code

Στον αντίποδα, η VHDL παρέχει εύκολη περιγραφή ασύγχρονων σημάτων και εύκολη διάκριση μεταξύ διαφορετικών κομματιών κώδικα. Ένα ακόμη χαρακτηριστικό της VHDL είναι ότι αποτελεί Strongly typed language, πράγμα που μπορούμε να το ερμηνεύσουμε και ως πλεονέκτημα και ως μειονέκτημα. Το αρνητικό είναι ότι δεν υποστηρίζει μετατροπές μεταξύ διαφορετικών τύπων δεδομένων.



Εικόνα 10 Fulladder

4.4 ALTOR32 CPU

Το κύκλωμα που χρησιμοποιήσαμε είναι η υλοποίηση σε Verilog ενός επεξεργαστή αρχιτεκτονικής OpenRISC 1000.

Altor32_alu.v :

Το αρχείο αυτό περιέχει τον κώδικα για μία αριθμητική και λογική μονάδα. Η μονάδα αυτή είναι υπεύθυνη για την εκτέλεση αριθμητικών πράξεων και λογικών υπολογισμών. Πιο συγκεκριμένα δέχεται σαν εισόδους τα σήματα “op_i” (4 bits) που καθορίζει τον τύπο της πράξης προς εκτέλεση, τα “a_i”(32 bits) και “b_i” (32 bits) στα οποία αντιστοιχούν οι τελεσταίοι της πράξης προς εκτέλεση και το σήμα “c_i”(1 bit) που αντιπροσωπεύει το κρατούμενο εισόδου για την πράξη της πρόσθεσης ή της αφαίρεσης. Αντίστοιχα οι έξοδοι του κυκλώματος αυτού είναι τα

αποτελέσματα των πράξεων που μπορεί να γίνουν στο κύκλωμα. Οι έξοδοι αυτοί καταλήγουν στο σήμα “p_o” (32bits) όπου τελικά επιλέγεται το σωστό αποτέλεσμα προς την έξοδο δια μέσου ενός πολυπλέκτη. Η συνθήκη με την οποία ο πολυπλέκτης αυτός επιλέγει το αντίστοιχο αποτέλεσμα καθορίζεται από το σήμα εισόδου “op_i” στο οποίο είναι κωδικοποιημένη η πράξη που θα γίνει στην μονάδα. Αναλυτικά οι υποστηριζόμενες πράξεις είναι οι εξής :

- Αριστερή λογική ολίσθηση.
- Δεξιά λογική ολίσθηση.
- Δεξιά αριθμητική ολίσθηση με διατήρηση πιο σημαντικού bit (most significant bit, msb)
- Πρόσθεση
- Πρόσθεση με κρατούμενο εισόδου
- Αφαίρεση
- Λογικό “ΚΑΙ”
- Λογικό “Η”
- Λογικό “XOR”

Επιπλέον η αριθμητική και λογική μονάδα διαθέτει και ένα κύκλωμα σύγκρισης το οποίο μπορεί να συγκρίνει τους δύο αριθμούς εισόδου (“a_i” και “b_i”) και να παράγει ένα αποτέλεσμα ανάλογα με την πράξη σύγκρισης που ζητήθηκε. Οι πράξη αυτή μπορεί να είναι οποιαδήποτε από τις παρακάτω:

- Ισότητα (a=b ?)

- Μεγαλύτερο από (a>b ?)
- Μικρότερο από (a<b ?)

Να σημειωθεί ότι σε κάθε κύκλο μηχανής, η αριθμητική και λογική μονάδα (alu) εκτελεί όλες τις προαναφερθείσες πράξεις σύγκρισης και αριθμητικής λογικής αλλά ένα μόνο αποτέλεσμα διοχετεύεται στην έξοδο, το οποίο επιλέγεται από έναν πολυπλέκτη εξόδου. Ακόμα, το κύκλωμα σύγκρισης είναι βασισμένο σε συνδυαστική λογική, όπως φυσικά και τα αριθμητικά και λογικά κυκλώματα και είναι ικανά να ολοκληρώνουνε την πράξη σε έναν κύκλο μηχανής. Τέλος, οι κωδικοί λειτουργίας (operation codes, opcodes) της αριθμητικής και λογικής μονάδας δίνονται στο αρχείο "altor32_defs" το οποίο γίνεται και included κατά την διάρκεια του compilation.

Altor32.v :

Υπό προγραμματιστική οπτική γωνία, το αρχείο αυτό λειτουργεί σαν "main" της σχεδίασης. Είναι το αρχείο που είναι πιο ψηλά στην ιεραρχία και αυτό που καλεί τα υπόλοιπα αρχεία-κυκλώματα για να συντονίσει την λειτουργία του επεξεργαστή. Στο αρχείο διακρίνονται οι φάσεις εκτέλεσης μία εντολής σε διακριτά στάδια που αντιπροσωπεύουν έναν κύκλο μηχανής:

- Ανάκληση εντολής. Κατά την διάρκεια του σταδίου αυτού, η επόμενη εντολή προς εκτέλεση αποθηκεύεται από την Icache (instructioncache) σε κάποιον καταχωρητή για να γίνει εκκίνηση της επεξεργασίας της. Στην περίπτωση που λόγο απλουστευμένης έκδοσης (lite) δεν υπαρχει Icache στον επεξεργαστή, τότε οι εντολές διαβάζονται από ένα αρχείο εισόδου. Επιπλέον στην έκδοση αυτή του επεξεργαστή, μαζί με την ανάκληση εντολής γίνεται και η αποκωδικοποίηση της. Κατά την αποκωδικοποίηση εντολής τα 32 bits της εκάστοτε εντολής "σπάνε" σε

επιμέρους πεδία τα οποία περιέχουν πληροφορία σχετικά με το είδος της εντολής, με τους τελεσταίους εισόδου της αριθμητικής και λογικής μονάδας, με την πράξη που θα γίνει, με το μέρος από το οποίο θα ανακτηθούνε οι τελεσταίοι (στον φάκελο καταχωρητών) και το μέρος που θα γίνει η τελική αποθήκευση του αποτελέσματος (στον φάκελο καταχωρητών ή στη μνήμη δεδομένων).

- Προσπέλαση του φακέλου καταχωρητών. Ο φάκελος καταχωρητων είναι μία δομή αποθήκευσης προσωρινών δεδομένων στον επεξεργαστή. Η δομή αυτή διακρίνεται για την υψηλή ταχύτητα προσπέλασης της και για την σχετικά μικρή αποθηκευτική της χωρητικότητα. Στην σχεδίαση αυτή ο φάκελος καταχωρητών, η δομή και το μέγεθος του επιλέγεται από τον χρήστη κατά την διάρκεια του compilation. Έτσι δίνονται επιλογές για 3 διαφορετικές αρχιτεκτονικές φακέλου καταχωρητών, σύμφωνα με πρότυπα που χρησιμοποιούνται από εταιρίες. Οι επιλογές που έχει ο χρήστης είναι :

Α) Φάκελος καταχωρητών σύμφωνα με τα πρότυπα της XILINX

Β) Φάκελος καταχωρητών σύμφωνα με τα πρότυπα της ALTERA

Γ) Φάκελος καταχωρητών ειδικός για προσομοίωση του κυκλώματος, όχι όμως για την διαδικασία της σύνθεσης.

- Εκτέλεση εντολής. Κατά τη διάρκεια του σταδίου αυτού, οι τελεσταίοι της εντολής που έχουν διαβαστεί από τον φάκελο καταχωρητών ή από τη μνήμη δεδομένων, ανακατευθύνονται στις δύο εισόδους της αριθμητικής και λογικής μονάδας, μαζί με ένα ακόμα σήμα που καθορίζει την επιλογή πράξης. Στη συνέχεια η πράξη λαμβάνει χώρα στην ALU και ολοκληρώνεται σε έναν κύκλο μηχανής όπου και παράγεται το αποτέλεσμα.

- Προσπέλαση μνήμης δεδομένων. Στο στάδιο αυτό γίνεται η προσπέλαση της μνήμης δεδομένων όπου και αποθηκεύονται αποτελέσματα που παράχθηκαν στην

αριθμητική και λογική μονάδα, ή αποθηκεύονται τιμές σε συγκεκριμένες θέσεις μνήμης.

- Εγγραφή αποτελέσματος. Στο στάδιο αυτό, γίνεται εγγραφή στον φάκελο καταχωρητών αποτελέσματα που δημιουργήθηκαν από την ALU ή που ανακτήθηκαν από την μνήμη.

Να σημειωθεί ότι στην σχεδίαση αυτή η ανάκληση και η αποκωδικοποίηση εντολής γίνεται στον ίδιο κύκλο μηχανής και άρα ανήκουν στη ίδια φάση.

Altor32_lite.v :

Το αρχείο αυτό συμβάλει στην λειτουργία του κυκλώματος ελέγχου του επεξεργαστή. Είναι ουσιαστικά ένα πολύ μεγάλο fsm (finite state machine- μηχανή πεπερασμένων καταστάσεων) το οποίο σε κάθε κύκλο μηχανής ελέγχει την κατάσταση στην οποία βρίσκεται, τις εντολές οι οποίες είναι προς εκτέλεση και τελικώς παράγει σήματα ελέγχου απαραίτητα για την σωστή λειτουργία του επεξεργαστή. Τέτοια σήματα είναι μεταξύ άλλων και τα σήματα επιλογής πράξης στην αριθμητική και λογική μονάδα, σήματα προς τη μνήμη για εγγραφή και ανάγνωση δεδομένων, σήματα προς τον φάκελο καταχωρητών για εγγραφή αποτελέσματος ή ανάγνωση τελουμένου εισόδου. Η ακολουθιακή λογική με την οποία φτιάχτηκε το κύκλωμα αυτό, το καθιστά ικανό να παράγει σήματα ελέγχου σε κάθε κύκλο μηχανής τα οποία αντιστοιχούνε σε κάθε εντολή που εκτελείται την συγκεκριμένη χρονική στιγμή, ανεξάρτητα σε ποια φάση εκτέλεσης βρίσκονται.

altor32_regfile_alt.v , altor32_regfile_sim.v , altor32_regfile_xil.v :

Τα αρχεία αυτά περιγράφουν την αρχιτεκτονική και την δομή των φακέλων καταχωρητών ανάλογα με τα κύρια τεχνικά χαρακτηριστικά που διέπουν την κάθε

τεχνολογία. Ανάλογα με τις επιλογές του χρήστη κατά τη διαδικασία της μεταγλώττισης, ένα από τα αρχεία αυτά χρησιμοποιούνται στην τελική σχεδίαση ενώ τα άλλα μένουν εκτός της διαδικασίας της σύνθεσης. Αξίζει να σημειωθεί ότι ο κάθε φάκελος καταχωρητών που περιγράφεται διαφέρει στο μέγεθος, στο εύρος bits των καταχωρητών και στην αρχιτεκτονική σχεδίαση του. Οι φάκελοι καταχωρητών της ALTERA και της XILINIX μπορούνε να χρησιμοποιηθούνε για την παραγωγή τελικού κώδικα με την διαδικασία της σύνθεσης ενώ η σχεδίαση " altor32_regfile_sim.v " μπορεί μόνο να χρησιμοποιηθεί για τις ανάγκες προσομοίωσης του συστήματος.

altor32_funcs.v:

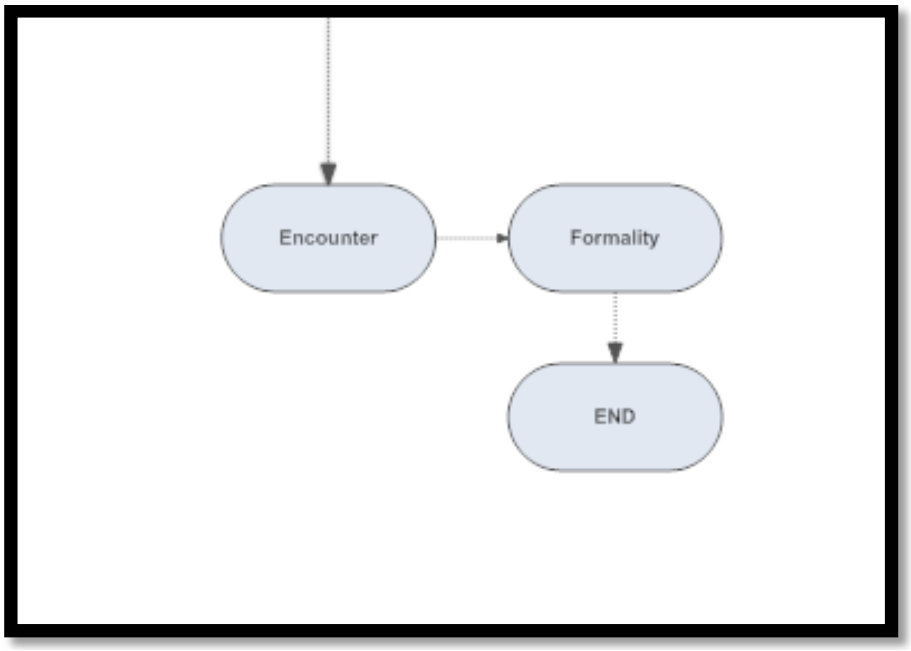
Στο αρχείο αυτό περιγράφονται οι επιμέρους λειτουργίες της αριθμητικής και λογικής μονάδας (ALU) εκτός από τις αριθμητικές πράξεις. Προγραμματιστικά το αρχείο αυτό έχει συναρτήσεις που καλούνται από την ALU όταν η πράξη που είναι προς εκτέλεση είναι οποιαδήποτε άλλη εκτός από αριθμητική. Ιεραρχικά όμως, οι περιγραφές αυτές ανήκουν στην ALU σε ένα επίπεδο ιεραρχίας παρακάτω και βρίσκονται εκεί υλοποιημένες σαν κύκλωμα, ανεξαρτήτως ποιας πράξης θα εκτελεστεί στον κάθε κύκλο μηχανής. Πιο συγκεκριμένα, οι λειτουργίες που περιγράφονται στο αρχείο αυτό είναι λειτουργίες σύγκρισης, επέκτασης πρόσημου και λειτουργίες για τον υπολογισμό της τελικής διεύθυνσης προσπέλασης μνήμης για ανάγνωση ή αποθήκευση δεδομένων.

altor32_defs.v :

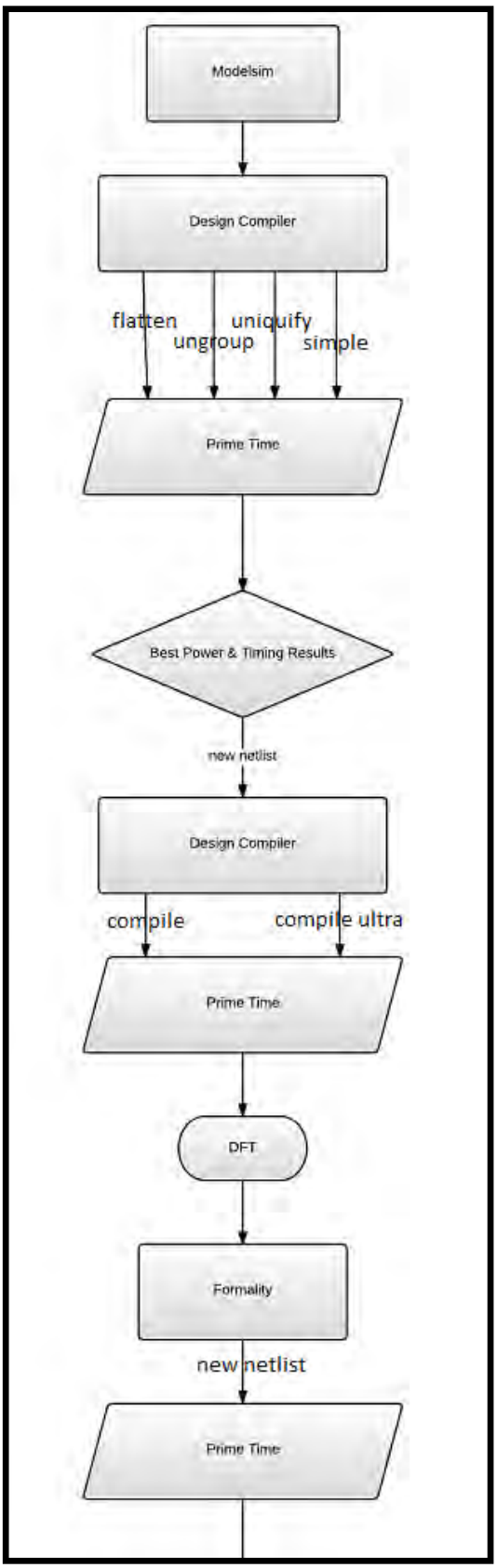
Στο αρχείο αυτό υπάρχουν όλες οι δηλώσεις των σταθερών μεταβλητών που χρησιμοποιούνται στην σχεδίαση. Σε επίπεδο υλικού οι δηλώσεις αυτές αντιστοιχούνε σε hard-wired σήματα τα οποία έχουν διαφορετικό εύρος σε bits και βοηθούνε στην αποκωδικοποίηση και εκτέλεση των εντολών.

5 Design Flow

Στο παρόν κεφάλαιο θα παρουσιάσουμε αναλυτικά τα βήματα της σχεδίασης μας. Το πρώτο βήμα μιας τυπικής ροής σχεδίασης είναι η συγγραφή του κώδικα περιγραφής του κυκλώματος. Ακολουθεί η front-endροή σχεδίασης που χρησιμοποιήσαμε. Παρατηρούμε ότι για την ολοκληρωμένη σχεδίαση επόμενο βήμα είναι το παρακάτω κομμάτι το οποίο αποτελεί μέρος της back-end ροής και δε θα ασχοληθούμε.



Εικόνα 11 Back-End of our flow



Εικόνα 12 Front-End of flow

Πριν προχωρήσουμε στην εξέταση κάθε βήματος ξεχωριστά θα αναφερθούμε σε κάποιες βασικές παραμέτρους της σχεδίασής μας. Πρόκειται για έναν επεξεργαστή ο οποίος λειτουργεί στα 60MHz - 100 MHz. Με βάση αυτή τη πληροφορία ορίσαμε την περίοδο κατά το πρώτο πέρασμα της σύνθεσης στα 12.35ns (80MHz). Αφού επιλέξαμε την τεχνική βελτιστοποίησης της σχεδίασης στην οποία παρατηρήσαμε συνδυασμό από καλύτερα αποτέλεσμα χρονισμού και μικρότερα αποτελέσματα καταναλισκόμενης ισχύος, ξαναπεράσαμε τη σχεδίαση από το βήμα της σύνθεσης ορίζοντας τη περίοδο ρολογιού σε 16.6 ns(80MHz). Αρχικά, καθορίσαμε την περιοχή που ήταν επιτρεπτή να καταλαμβάνει το κύκλωμα σε 0, ώστε να πετύχουμε τη μικρότερη δυνατή τιμή. Στο επόμενο πέρασμα χαλαρώσαμε λίγο τη τιμή αυτή δίνοντας νέα τιμή το αποτέλεσμα της εξίσωσης area + (area * 20%), όπου area είναι η τιμή που προέκυψε από το πρώτο πέρασμα. Επειδή, δεν είχαμε πιο συγκεκριμένα κριτήρια λειτουργίας για τον καθορισμό των παραμέτρων χρησιμοποιήσαμε τις default τιμές ενός απλού αντιστροφέα της βιβλιοθήκης που χρησιμοποιήσαμε.

5.1 MODELSIM

Εφόσον έχει ολοκληρωθεί η συγγραφή του κώδικα περιγραφής του κυκλώματος, αυτό που πρέπει να γίνει είναι να ελεγχθεί η ορθή λειτουργία του. Η διαδικασία αυτή ονομάζεται προσομοίωση. Η διαδικασία αυτή κρίνεται απαραίτητη στο πρώιμο αυτό στάδιο διότι τα επιτυχή αποτελέσματα αυτού του βήματος ελαχιστοποιούν την πιθανότητα παρουσίασης ανεπιθύμητων αποτελεσμάτων κατά τη λειτουργία του κυκλώματος.

Βασικό πλεονέκτημα της διαδικασίας της προσομοίωσης σε αυτό το επίπεδο είναι η ταχύτητα εκτέλεσής της. Το εργαλείο που επιλέξαμε για το βήμα είναι το ModelSim (Mentor Graphics).

Το Modelsim χρησιμοποιείται για την προσομοίωση, επαλήθευση και αποσφαλμάτωση σχεδιάσεων οι οποίες περιγράφονται σε μια από τις ακόλουθες γλώσσες περιγραφής υλικού:

- VHDL
- Verilog
- System Verilog
- System C

Η δική μας σχεδίαση είναι γραμμένη σε Verilog και αποτελείται από επτά αρχεία όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο: altor32.v altor32_lite.v altor32_alu.v altor32_regfile_xil.v altor32_regfile_alt.v altor32_regfile_sim.v altor32_defs.v

- Verilog file/files creation – οι κώδικες που αποτελούν το κύκλωμά μας είναι γραμμένοι σε κώδικα Verilog.

- Library declaration - δημιουργία μιας βιβλιοθήκης για την αποθήκευση της σχεδίασής μας.

- Library mapping - η βιβλιοθήκη που δημιουργήθηκε στο παραπάνω βήμα πρέπει, ακολούθως, να αντιστοιχιστεί σε κάποια ή κάποιες από τις προϋπάρχουσες βιβλιοθήκες οι οποίες περιλαμβάνουν ορισμούς και υλοποιήσεις βασικών συναρτήσεων και πράξεων που ενδέχεται να χρειαστούν.

- Compilation – έλεγχος συντακτικής ορθότητας.

- Simulation - Εκκίνηση της διαδικασίας της προσομοίωσης για ένα συγκεκριμένο σετ τιμών εισόδου και για καθορισμένο χρονικό διάστημα.

- Waveform Generation - Δημιουργία κατάλληλης κυματομορφής στην οποία μπορούμε να παρατηρήσουμε την εναλλαγή τιμών κάθε σήματος της σχεδίασης κατά την πάροδο του χρόνου.

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

5.2 DESIGN COMPILER

Μια από τις μεγαλύτερες εταιρίες στο χώρο των EDA είναι η Synopsys. Η Synopsys προσφέρει ένα ολοκληρωμένο πακέτο σχεδίασης, καθώς περιλαμβάνει εργαλεία τόσο για front-end σχεδίαση όσο και για back-end. Ο πυρήνας του πακέτου αποτελείται από τον Design 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. Ένα από τα πλεονεκτήματα του είναι ότι αποτελείται από υποεργαλεία τα οποία συνθέτουν τις HDL σχεδιάσεις από 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.

Μετά την επιτυχημένη προσομοίωση της σχεδίασης μας και αφού επαληθεύσαμε την ορθή λειτουργία της επόμενο βήμα είναι να την περάσουμε από τον DC. Θα περάσουμε το κύκλωμα μας με 4 διαφορετικά scripts καθένα από τα οποία περιλαμβάνει διαφορετικούς περιορισμούς. Στόχος μας είναι να βρούμε την σχεδίαση εκείνη που συνδυάζει καλύτερα timing (χρονισμού) και power (καταναλισκόμενη ισχύς) αποτελέσματα. Αρχικά, η περίοδος του ρολογιού ορίζεται στα 12.5nsδηλαδή σε 80MHzκαι η παράμετρος area σε 0 (περιορίζουμε την περιοχή που θα καταλάβει η σχεδίαση στην μικρότερη δυνατή).

Το κάθε script μας έχει να κάνει με την ιεραρχία της σχεδίασης. Οι τέσσερις διαφορετικές ιεραρχίες που μπορούμε να έχουμε είναι οι εξής:

- ✓ Flatten–μετατρέπει ουσιαστικά τα μονοπάτια συνδυαστικής λογικής της σχεδίασης σε αναπαράσταση λογικής δύο επιπέδων, αφαιρώντας ταυτόχρονα τις ενδιάμεσες μεταβλητές

- ✓ Ungroup–ισοπεδώνει τελείως την ιεραρχία της σχεδίασης. Ολόκληρο το κύκλωμα θεωρείται ένα επίπεδο.

- ✓ Uniquify – στην ουσία εξατομικεύεται η σχεδίαση κάθε κελιού της σχεδίασης.

- ✓ Nothing – παραμένει η σχεδίαση του κυκλώματος καθεαυτή χωρίς καμία επίδραση πάνω στην ιεραρχία της.

Ακολουθεί αναλυτική περιγραφή των scriptsγια κάθε μια από τις παραπάνω επιλογές. Ο DC καλείται από το περιβάλλον Unix με την εντολή **dc_shell** όπου ανοίγει σε επίπεδο κονσόλας. Αν θέλουμε να δουλέψουμε πάνω στο γραφικό περιβάλλον του DC δεν έχουμε παρά να δώσουμε την εντολή **gui_start**. Όλα τα τρεξίματα έχουν γίνει σε επίπεδο κονσόλας χρησιμοποιώντας scripts τύπου .tcl, που υποστηρίζει το εργαλείο.

Η γλώσσα TCL (Tool Command Language) δημιουργήθηκε από τον John Ousterhout στο UC Berkeley. Είναι μια Scripting γλώσσα, με ευκολία στην αυτοματοποίηση επαναλαμβανόμενων εργασιών. Χρησιμοποιείται σε συνάρτηση με την πλειοψηφία των βιομηχανικών CAD εργαλείων. Γενικά είναι εύκολη και γρήγορη στην εκμάθηση καθώς και εύκολη στη χρήση και την εφαρμογή της σε οποιοδήποτε περιβάλλον. Πρακτικά όλη η διαδικασία υλοποίησης μιας σχεδίασης μέσω της χρήσης CAD εργαλείων, πραγματοποιείται με την αλλεπάλληλη εφαρμογή tcl scripts. Οι πληροφορίες από τα reports που προκύπτουν είναι αυτές που

καθορίζουν κατά πόσο ο σχεδιαστής θα “αναγκαστεί“ να χρησιμοποιήσει την εκάστοτε γραφική διεπαφή.

5.2.1 Flatten

Το πρώτο script που θα αναλύσουμε είναι αυτό που αναφέρεται σε flatten σχεδίαση. Για να τρέξει καλούμε την εντολή “**dc_shell -f dc_script.tcl”**. Αρχικά, θα πρέπει να αποσύρουμε τυχόν προηγούμενη σχεδίαση από το εργαλείο για να αποφύγουμε συγκρούσεις μεταξύ των σχεδιάσεων. Για αυτό το λόγο το script ξεκινάει με την εντολή “**remove_design –all**”. Με τις εντολές

sh rm -rf ./desired_directory

file mkdir ./desired_directory

δημιουργούμε και καθορίζουμε το directory μέσα στο οποίο θα αποθηκευτούν τα αρχεία εξόδου της σχεδίασής μας.

Επόμενο βήμα είναι να σετάρουμε τη σχεδίαση μας. Αυτό επιτυγχάνεται με την εντολή “**set design_name $name**”, στην οποία η τιμή του name πρέπει να αντιστοιχίζεται στην τιμή που έχει οριστεί στον κώδικα της Verilog και συγκεκριμένα στο αρχείο όπου βρίσκεται το κεντρικό module της σχεδίασης. Στην περίπτωση μας έχει το όνομα “cpu”.

Χρησιμοποιώντας την επόμενη εντολή “**set_svf ./desired_directory/${design_name}_syn.svf**” ο DC δημιουργεί ένα svf αρχείο το οποίο θα το χρησιμοποιήσουμε στην πορεία για να ελέγξουμε την αρχικήRTL σχεδίαση με την νέα σχεδίαση που θα προκύψει από την εφαρμογή των

περιορισμών που θα ορίσουμε. Κρατάμε πάντα το .svf αρχείο της αρχικής σχεδίασης καθώς και το .svf της βελτιστοποιημένης σχεδίασης ώστε να επιβεβαιώσουμε πως οι αλλαγές δεν έχουν επηρεάσει τη λειτουργία του κυκλώματος.

Οι παρακάτω εντολές με τη σειρά που τις βλέπουμε είναι να καθορίσουν τη θέση στην οποία βρίσκονται τα αρχεία Verilog της σχεδίασής μας και εκείνη από την οποία θα διαβάσει τη βιβλιοθήκη που θα χρησιμοποιήσουμε.Οι δύο επόμενες εντολές (setlink_library και settarget_library), προσδιορίζουν τη θέση όπου βρίσκονται οι τεχνολογικές βιβλιοθήκες βάσει των οποίων θα δημιουργηθεί το τεχνολογικά εξαρτημένο netlist. Η διαφορά ανάμεσα στις δύο αυτές βιβλιοθήκες είναι ότι η δεύτερη (link) αποτελείται από πιο περίπλοκα κελιά (cells).

set project_root "./SRC"

set lib_root "/home/LIB"

set search_path "$lib_root $project_root"

set target_library "/home/LIB/saed90nm_typ.db"

set synthetic_library "dw_foundation.sldb"

set link_library "$target_library $synthetic_library"

define_design_lib WORK -path ./ WORK_desired_directory

Η βιβλιοθήκη που θα χρησιμοποιήσουμε είναι η "**saed90nm_typ.db**", αποτελεί μέρος της τεχνολογικής βιβλιοθήκης SAED 90nm Educational Design Kit (EDK). Η βιβλιοθήκη έχει δημιουργηθεί με στόχο η χρήση της να βοηθήσει στη βελτιστοποίηση των κύριων χαρακτηριστικών ενός ολοκληρωμένου κυκλώματος.

Περιλαμβάνει μια πληθώρα από ακολουθιακού και συνδυαστικού τύπου κελιών (cells) τα οποία έχουν διαφορετικές δυνατότητες οδήγησης.

Η βασική ροή του DC χρησιμοποιεί τις Foundation και GTECHβιβλιοθήκες. 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. 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.

Η εντολή "**analyze –format verilog $my_verilog_files"** εξετάζει το HDL αρχείο, προκειμένου να διαπιστώσει εάν βρίσκεται σε ορθή συντακτική και λογική μορφή. Επιπρόσθετα, μεταφράζει τα αρχείασε μια ενδιάμεση μορφή και τα τοποθετεί στον φάκελο που έχουμε ορίσει ως φάκελο εργασίας.

Η εντολή "**elaborate design_name**" εξετάζει το ενδιάμεσο αρχείο που έχει δημιουργηθεί από την εκτέλεση της προηγούμενης εντολής και καθορίζει ποια από τα στοιχεία της σχεδίασης πρέπει να αντικατασταθούν από συνθετικά Modulesτης βιβλιοθήκης που έχει επιλεγεί.

Με τις εντολές που εισάγονται, στη συνέχεια, καθορίζονται οι περιορισμοί και οι παράμετροι χρονισμού της σχεδίασης.

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]

set_dont_touch_network [list clk_i rst_i]

set_fix_hold clk_i

set_drive 0.06 [all_inputs]

set_load 5 [all_outputs]

set_input_delay 0.5 -clock clk_i [all_inputs]

set_output_delay 0.5 -clock clk_i [all_outputs]

set_max_area 0

Σύμφωνα με τους παραπάνω περιορισμούς το ρολόι που χρησιμοποιήσαμε έχει περίοδο 12.5ns (να σημειώσουμε η σχεδίαση μας δουλεύει σε 60-100MHz). Η επόμενη εντολή δηλώνει ότι το ρολόι και το reset δε θα επηρεαστούν ούτε θα αντικατασταθούν από τη βελτιστοποίηση. Με την επόμενη ενημερώνουμε την βελτιστοποίηση ότι θα πρέπει να επιλύσει τυχόν παραβάσεις (violators) hold του ρολογιού. Γενικά, ένας τρόπος να ελέγξουμε αν η βελτιστοποίηση βγάζει σωστά αποτελέσματα είναι η απουσία clock hold violations. Έπειτα ορίζουμε τη δυνατότητα οδήγησης κάθε εισόδου σε 0.06 και τον φόρτο κάθε εισόδου σε 5. Γενικά αυτό που κάνουμε σε αυτό το βήμα είναι να επανατροφοδοτήσουμε τον DC με νέα δεδομένα εισόδου, προκειμένου να γίνει μια πιο ακριβής ανάλυση του χρονισμού του κυκλώματος. Τέλος, ορίζουμε ότι το μέγεθος της περιοχής που θα καταλάβει σχεδίαση μας είναι 0. Τη δεύτερη φορά που θα περάσουμε το κύκλωμα θα επιλέξουμε με αυστηρότερα κριτήρια. Ο καθορισμός όλων των παραπάνω δεδομένων επηρεάζει άμεσα την σύνθεση της σχεδίασης και τα αποτελέσματα των βελτιστοποιήσεων που εφαρμόζει ο DC .

Ερχόμαστε στο πιο σημαντικό κομμάτι των εντολών. Η “**set_flatten true**” μετατρέπει ουσιαστικά τα μονοπάτια συνδυαστικής λογικής της σχεδίασης σε αναπαράσταση λογικής δύο επιπέδων, αφαιρώντας ταυτόχρονα τις ενδιάμεσες

μεταβλητές. Μειώνεται ο χρόνος βελτιστοποίησης ωστόσο μεγαλώνει η περιοχή που θα καλύψει η σχεδίαση. Συνήθως η εντολή αυτή χρησιμοποιείται για πολύ μεγάλα κυκλώματα.

Ο DC με την εντολή "**set_structure true**" δημιουργεί λογική δομή στη σχεδίαση με το να προσθέτει ενδιάμεσες μεταβλητές.

Η εντολή "**compile -map_effort high -area_effort high –incremental**" περιγράφει με ποιόν τρόπο βελτιστοποιείται η σχεδίαση. Εμείς την ορίσαμε να πάρει τα πιο αυστηρά κριτήρια.

Οι δύο επόμενες εντολές "**remove_unconnected_ports -blast_buses [find –hierarchy cell {"*"}]**" και "**remove_unconnected_ports [get_cells *]**" αφαιρούν όλα τα ασύνδετα κομμάτια που έχουν δημιουργηθεί νωρίτερα.

Η **"change_names –rules verilog–hierarchy"** αντικαθιστά την ονομασία πυλών, καλωδίων, θυρών ώστε να συμμορφωθούν στον κανόνα ονομασίας που ορίζουμε εμείς.

Όλες οι εντολές που έχουμε χρησιμοποιήσει μέχρι στιγμής έχουν επίδραση πάνω στο κύκλωμα όποτε με την επόμενη εντολή **"check_design"** επιβεβαιώνουμε ότι ακόμη λειτουργεί όπως θα έπρεπε. Στην περίπτωση που σε αυτό το βήμα η σχεδίαση μας δεν είναι συνεπής με τη λειτουργικότητά της πρέπει να ξανά αλλάξουμε τους περιορισμούς που έχουμε θέσει, γιατί η εμφάνιση λαθών στο βήμα αυτό τονίζει την αποτυχία της σύνθεσης της σχεδίασης.

remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]

remove_unconnected_ports [get_cells *]

set_fix_multiple_port_nets -all

check_design

Εδώ έχουμε επανάληψη κάποιων εντολών για να αφαιρεθούν όλα τα περιττά στοιχεία και ξανά ελέγχουμε τη συνέπεια της σχεδίασης.

Καθένα από τα παρακάτω αρχεία εξόδου περιέχει πληροφορίες της νέας σχεδίασης. Με τη σειρά που είναι δηλωμένες οι εντολές παίρνουμε πληροφορίες για την περιοχή που καταλαμβάνει η σχεδίαση μας, για το χρονισμό, ποιότητα αποτελεσμάτων (Quality of results), τη καταναλισκόμενη ισχύ(power), τα χαρακτηριστικά της σχεδίασης, όλες τις παραβάσεις της σχεδίασης, την ιεραρχία, τα κελιά, τα καλώδια και τέλος τις πύλες.

report_area > ./desired_directory/design_name}_syn.area

report_timing > ./desired_directory /${design_name}_syn.timing

report_qor > ./desired_directory /${design_name}_syn.qor

report_power > ./desired_directory /${design_name}_syn.power

report_design > ./desired_directory /${design_name}_syn.design

report_constraint -all_violators >
./desired_directory/${design_name}_syn.violators

report_hierarchy > ./desired_directory /${design_name}_syn.hierarchy

report_resources > ./desired_directory /${design_name}_syn.resources

report_cell > ./desired_directory /${design_name}_syn.cells

report_net > ./desired_directory /${design_name}_syn.nets

report_port > ./desired_directory /${design_name}_syn.ports

Όλες οι write αποθηκεύουν το προκείμενο netlist σε ένα αρχείο μορφοποιημένο σύμφωνα με τη γλώσσα περιγραφής υλικού που έχει οριστεί. Η εντολή write_sdc δημιουργεί ένα αρχείο που εμπεριέχει όλους τους σχεδιαστικούς περιορισμούς του κυκλώματος και δύναται να χρησιμοποιηθεί σε επόμενο στάδιο της σχεδίασης. Στην συνέχεια με την εντολή write_parasitics αποθηκεύονται πληροφορίες για τα παρασιτικά που παρατηρούνται στο κύκλωμα, σε ένα αρχείο .spef, έτσι ώστε να είναι δυνατός σε επόμενα στάδια του σχεδιασμού ο ορθός υπολογισμός της καταναλισκόμενης ισχύος.

write -hierarchy -format verilog -output ./desired_directory /${design_name}_syn.v

write -hierarchy -format vhdl -output ./desired_directory /${design_name}_syn.vhd

write -format vhdl -output ./desired_directory /${design_name}_syn_no_hier.vhd

write_sdc ./desired_directory /${design_name}_syn.sdc

write_sdf -version 2.1 ./desired_directory /${design_name}_syn.sdf

write_parasitics -output ./desired_directory /${design_name}_syn.spef

saif_map -type ptpx -write_map ./desired_directory/${design_name}_syn.SAIF.namemap

Πριν προχωρήσουμε στο επόμενο script θα δώσουμε δύο σημαντικούς ορισμούς απαραίτητους για την κατανόηση των χρονικών παραβάσεων.

- **Hold Time** – ορίζεται το ελάχιστο απαιτούμενο χρονικό διάστημα κατά το οποίο ένα σήμα πρέπει να είναι σταθερό μετά την θετική μετάβαση ενός ρολογιού.
- **Setup Time** – ορίζεται το ελάχιστο απαιτούμενο χρονικό διάστημα κατά το οποίο ένα σήμα πρέπει να είναι σταθερό πριν από την θετική μετάβαση ενός ρολογιού.

5.2.2 Ungroup

Το επόμενο scriptπου χρησιμοποιήσαμε είναι το ungroup. Οι αρχικές εντολές για τον καθορισμό της θέσης αποθήκευσης και των βιβλιοθηκών παραμένει ίδια με το προηγούμενο script. Επίσης, ίδιες παραμένουν και οι εντολές analyze, elaborate, current design. Η μοναδική διαφορά των περιορισμών της σχεδίασης είναι η απουσία της "**set_dont_touch_network [list clk_i rst_i]"**,στη βελτιστοποίηση του ungroup το ρολόι και το reset επηρεάζονται.

Επόμενη εντολή είναι το "**ungroup -all**" η οποία αφαιρεί ολοκληρωτικά την ιεραρχία στης σχεδίασης, η αναπαράσταση της γίνεται σε ένα επίπεδο. Αυτή είναι η διαφορά με την τεχνική flatten.

Εν συνεχεία, προχωρούμε σε compile, αφαιρούμε τυχόν ασύνδετα στοιχεία, ελέγχουμε την συνέπεια του κυκλώματος, επαναλαμβάνουμε τα δυο προηγούμενα βήματα όπως ακριβώς εκτελέσαμε και στο προηγούμενο script.

Τέλος, ορίζουμε τα αρχεία εξόδου και τις εντολές write, όπως προηγουμένως.

5.2.3 Uniquify

Η τρίτη τεχνική που χρησιμοποιήσαμε είναι το uniquify. Πολλαπλασιάζει την ιεραρχία εξατομικεύοντας τη σχεδίαση κάθε στοιχείου του κυκλώματος, δημιουργούνται αντίγραφα της σχεδίασης καθένα με μοναδικό όνομα. Η τεχνική αυτή εκμεταλλεύεται το διαφορετικό τύπο κάθε στοιχείου για να παράξει καλύτερα αποτελέσματα.

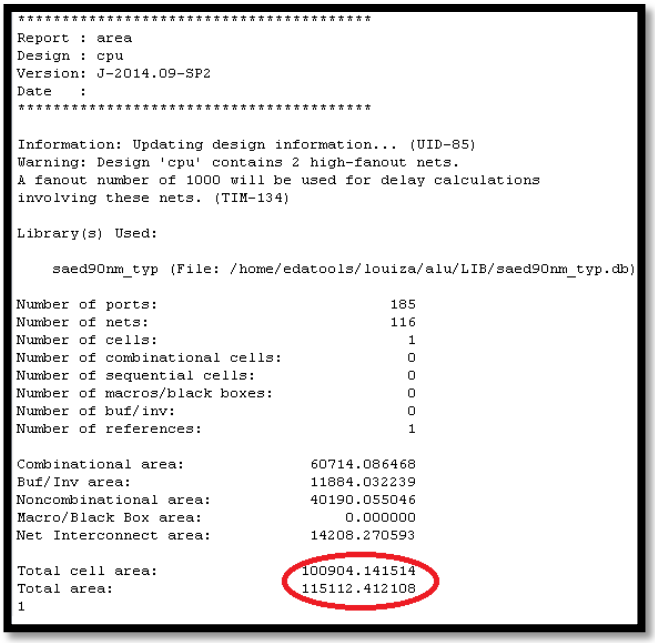
Όπως και με τα προηγούμενα scripts οι αρχικές δηλώσεις παραμένουν ίδιες. Η διαφορά με το ungroup είναι ότι και σε αυτή τη περίπτωση έχουμε την εντολή "**set_dont_touch_network [list clk_i rst_i]**". Στο ungroup δεν είναι απαραίτητη καθώς πρέπει να επέμβουμε σε ολόκληρη τη σχεδίαση ώστε να καταργήσουμε τελείως την ιεραρχία της. Η νέα εντολή που ορίζουμε είναι η "**uniquify**". Η συνέχεια του script είναι πανομοιότυπη με πριν.

5.2.4 Simple

Όπως υποδηλώνει και το όνομα του το script αυτό δεν περιέχει καμία τεχνική βελτιστοποίησης, στην ουσία χρησιμοποιεί μονάχα τους περιορισμούς της σχεδίασης, που ορίζουμε για να προσπαθήσει να τη βελτιστοποιήσει. Έχει όλες τις παραπάνω εντολές που είναι ίδιες σε όλα τα αρχεία, περιέχει επίσης την εντολή "**set_dont_touch_network [list clk_i rst_i]**", που απουσίαζε από το ungroup.

5.2.5 Αρχεία εξόδου / Αναφορές

Πριν προχωρήσουμε σε επόμενο βήμα της σχεδίασης θα παρουσιάσουμε συνοπτικά τα αρχεία εξόδου και τις αναφορές που έχουν προκύψει από το προηγούμενο κεφάλαιο. Εφόσον παράγονται τα ίδια αρχεία, αλλάζουν τα αποτελέσματα σε κάθε τρέξιμο, δε θα τα δούμε για κάθε script ξεχωριστά.



```
****************************************
Report : area
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************

Information: Updating design information... (UID-85)
Warning: Design 'cpu' contains 2 high-fanout nets.
A fanout number of 1000 will be used for delay calculations
involving these nets. (TIM-134)

Library(s) Used:

    saed90nm_typ (File: /home/edatools/louiza/alu/LIB/saed90nm_typ.db)

Number of ports:                          185
Number of nets:                           116
Number of cells:                            1
Number of combinational cells:              0
Number of sequential cells:                 0
Number of macros/black boxes:               0
Number of buf/inv:                          0
Number of references:                       1

Combinational area:              60714.086468
Buf/Inv area:                    11884.032239
Noncombinational area:           40190.055046
Macro/Black Box area:                0.000000
Net Interconnect area:           14208.270593

Total cell area:                100904.141514
Total area:                     115112.412108
1
```

Εικόνα 13 Area Report

Από εδώ μας ενδιαφέρει μόνο το Total Area. Το αποτέλεσμα συν ένα 20% αυτού θα μας καθορίσουν το area που θα δηλώσουμε σε επόμενο βήμα.

```
****************************************
Report : cell
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************

Attributes:
   BO - reference allows boundary optimization
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    p - parameterized
    r - removable
    u - contains unmapped logic

Cell                      Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
U3                        INVX0           saed90nm_typ    5.529600
U7                        AO21X1          saed90nm_typ    10.137600
U9                        AO21X1          saed90nm_typ    10.137600
U10                       AO221X1         saed90nm_typ    12.902400
U11                       AO22X1          saed90nm_typ    11.980800
U16                       AO222X1         saed90nm_typ    14.745600
U18                       AO222X1         saed90nm_typ    14.745600
U20                       AO222X1         saed90nm_typ    14.745600
U22                       AO222X1         saed90nm_typ    14.745600
U24                       AO221X1         saed90nm_typ    12.902400
U26                       AO222X1         saed90nm_typ    14.745600
U38                       AO222X1         saed90nm_typ    14.745600
U43                       AO21X1          saed90nm_typ    10.137600
U44                       AO221X1         saed90nm_typ    12.902400
U45                       AO21X1          saed90nm_typ    10.137600
U46                       AND2X1          saed90nm_typ    7.372800
U48                       AO21X1          saed90nm_typ    10.137600
U50                       AO21X1          saed90nm_typ    10.137600
U52                       AO21X1          saed90nm_typ    10.137600
U54                       AO21X1          saed90nm_typ    10.137600
```

Εικόνα 14 Cells Report

Δίνει πληροφορία για τα cells της σχεδίασης. Δεν πρέπει να υπάρχουν cells που χαρακτηρίζονται ως black box γιατί ο DC δεν μπορεί να τα αναγνωρίσει και κατ' επέκταση να τα βελτιστοποιήσει.

```
****************************************
Report : hierarchy
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************

cpu
    AND2X1                                    saed90nm_typ
    AND2X2                                    saed90nm_typ
    AND2X4                                    saed90nm_typ
    AND3X1                                    saed90nm_typ
    AND3X4                                    saed90nm_typ
    AND4X1                                    saed90nm_typ
    AND4X2                                    saed90nm_typ
    AND4X4                                    saed90nm_typ
    AO21X1                                    saed90nm_typ
    AO21X2                                    saed90nm_typ
    AO22X1                                    saed90nm_typ
    AO22X2                                    saed90nm_typ
    AO221X1                                   saed90nm_typ
    AO221X2                                   saed90nm_typ
    AO222X1                                   saed90nm_typ
    AOI21X1                                   saed90nm_typ
    AOI22X1                                   saed90nm_typ
    AOI22X2                                   saed90nm_typ
    AOI221X2                                  saed90nm_typ
    AOI222X1                                  saed90nm_typ
    AOI222X2                                  saed90nm_typ
    DELLN1X2                                  saed90nm_typ
    DELLN2X2                                  saed90nm_typ
    DFFARX1                                   saed90nm_typ
    DFFASX1                                   saed90nm_typ
    INVX0                                     saed90nm_typ
    INVX1                                     saed90nm_typ
    MUX21X1                                   saed90nm_typ
    MUX21X2                                   saed90nm_typ
    NAND2X0                                   saed90nm_typ
```

Εικόνα 15 Hierarchy Report

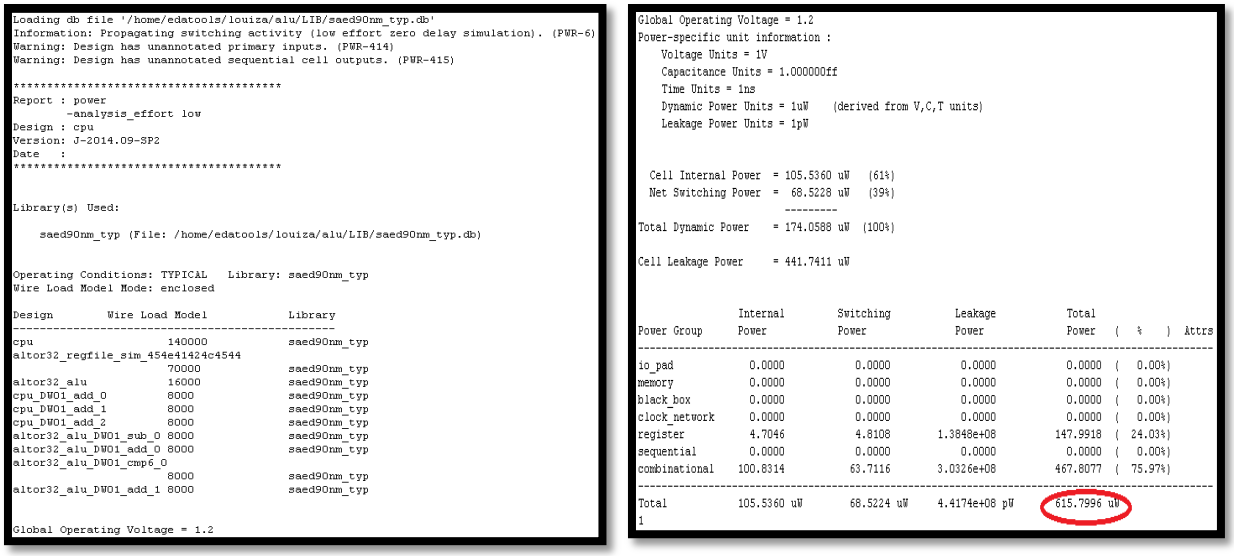
```
****************************************
Report : net
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************


Operating Conditions: TYPICAL   Library: saed90nm_typ
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
cpu                     140000           saed90nm_typ
altor32_regfile_sim_454e41424c4544
                        70000            saed90nm_typ
altor32_alu             16000            saed90nm_typ
cpu_DW01_add_0          8000             saed90nm_typ
cpu_DW01_add_1          8000             saed90nm_typ
cpu_DW01_add_2          8000             saed90nm_typ
altor32_alu_DW01_sub_0 8000              saed90nm_typ
altor32_alu_DW01_add_0 8000              saed90nm_typ
altor32_alu_DW01_cmp6_0
                        8000             saed90nm_typ
altor32_alu_DW01_add_1 8000              saed90nm_typ


Attributes:
   dr - drc disabled
    h - high fanout

Net
 Fanout    Fanin     Load      LoadUR    LoadUF    LoadLR    LoadLF    Resist    Pins      Attr
-----------------------------------------------------------------------------------------------
break_o
      1         1        5.00      5.00      5.00      5.00      5.00      0.20       2
clk_i
   1246         1     1030.88   1030.88   1030.88   1030.88   1030.88   2089.25    1247    dr, h
dmem_ack_i
```

Εικόνα 16 Net Report

```
****************************************
Report : port
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************


Attributes:
    d - dont_touch_network

                          Pin       Wire      Max       Max       Connection
Port              Dir     Load      Load      Trans     Cap       Class       Attrs
-------------------------------------------------------------------------------------
clk_i             in      0.0000    0.0000    --        --        --          d
dmem_ack_i        in      0.0000    0.0000    --        --        --
dmem_dat_i[0]     in      0.0000    0.0000    --        --        --
dmem_dat_i[1]     in      0.0000    0.0000    --        --        --
dmem_dat_i[2]     in      0.0000    0.0000    --        --        --
dmem_dat_i[3]     in      0.0000    0.0000    --        --        --
dmem_dat_i[4]     in      0.0000    0.0000    --        --        --
dmem_dat_i[5]     in      0.0000    0.0000    --        --        --
dmem_dat_i[6]     in      0.0000    0.0000    --        --        --
dmem_dat_i[7]     in      0.0000    0.0000    --        --        --
dmem_dat_i[8]     in      0.0000    0.0000    --        --        --
dmem_dat_i[9]     in      0.0000    0.0000    --        --        --
dmem_dat_i[10]    in      0.0000    0.0000    --        --        --
dmem_dat_i[11]    in      0.0000    0.0000    --        --        --
dmem_dat_i[12]    in      0.0000    0.0000    --        --        --
dmem_dat_i[13]    in      0.0000    0.0000    --        --        --
dmem_dat_i[14]    in      0.0000    0.0000    --        --        --
dmem_dat_i[15]    in      0.0000    0.0000    --        --        --
dmem_dat_i[16]    in      0.0000    0.0000    --        --        --
dmem_dat_i[17]    in      0.0000    0.0000    --        --        --
dmem_dat_i[18]    in      0.0000    0.0000    --        --        --
dmem_dat_i[19]    in      0.0000    0.0000    --        --        --
dmem_dat_i[20]    in      0.0000    0.0000    --        --        --
dmem_dat_i[21]    in      0.0000    0.0000    --        --        --
```

Εικόνα 17 Ports Report

Τα 3 παραπάνω αρχεία δίνουν πληροφορία για τα την ιεραρχία της σχεδίασης, όπως μπορούμε να δούμε σε αυτό το αρχείο το top cellείναι το cpu, για τα καλώδια και τις θύρες του κυκλώματος.

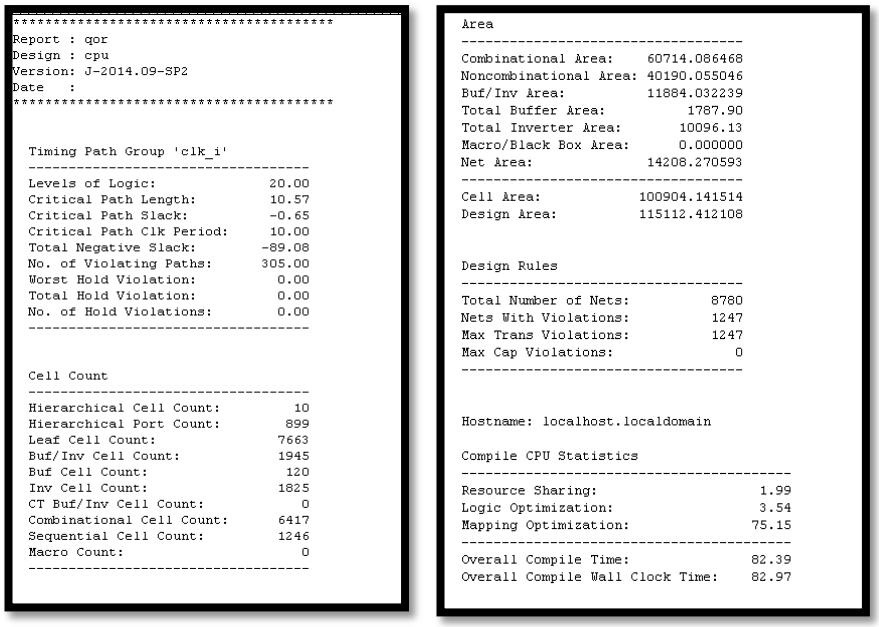


```
Loading db file '/home/edatools/louiza/alu/LIB/saed90nm_typ.db'
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
        -analysis_effort low
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************


Library(s) Used:

    saed90nm_typ (File: /home/edatools/louiza/alu/LIB/saed90nm_typ.db)


Operating Conditions: TYPICAL   Library: saed90nm_typ
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
cpu                     140000           saed90nm_typ
altor32_regfile_sim_454e41424c4544
                        70000            saed90nm_typ
altor32_alu             16000            saed90nm_typ
cpu_DW01_add_0          8000             saed90nm_typ
cpu_DW01_add_1          8000             saed90nm_typ
cpu_DW01_add_2          8000             saed90nm_typ
altor32_alu_DW01_sub_0 8000              saed90nm_typ
altor32_alu_DW01_add_0 8000              saed90nm_typ
altor32_alu_DW01_cmp6_0
                        8000             saed90nm_typ
altor32_alu_DW01_add_1 8000              saed90nm_typ


Global Operating Voltage = 1.2
```

```
Global Operating Voltage = 1.2
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  = 105.5360 uW   (61%)
  Net Switching Power  =  68.5228 uW   (39%)
                         ---------
Total Dynamic Power    = 174.0588 uW  (100%)

Cell Leakage Power     = 441.7411 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
------------------------------------------------------------------------------------------------
io_pad           0.0000           0.0000              0.0000             0.0000  (   0.00%)
memory           0.0000           0.0000              0.0000             0.0000  (   0.00%)
black_box        0.0000           0.0000              0.0000             0.0000  (   0.00%)
clock_network    0.0000           0.0000              0.0000             0.0000  (   0.00%)
register         4.7046           4.8108          1.3848e+08           147.9918  (  24.03%)
sequential       0.0000           0.0000              0.0000             0.0000  (   0.00%)
combinational  100.8314           63.7116         3.0326e+08           467.8077  (  75.97%)
------------------------------------------------------------------------------------------------
Total          105.5360 uW       68.5224 uW      4.4174e+08 pW        615.7996 uW
1
```

Εικόνα 18 Power Report

Από τα πιο σημαντικά αρχεία που δημιουργεί ο DC είναι το αρχείο καταναλισκόμενης ισχύος. Μας ενδιαφέρει το ολικό αποτέλεσμα. Στόχος μας είναι το αποτέλεσμα αυτό και και του χρονισμού να είναι τα καλύτερα δυνατά.



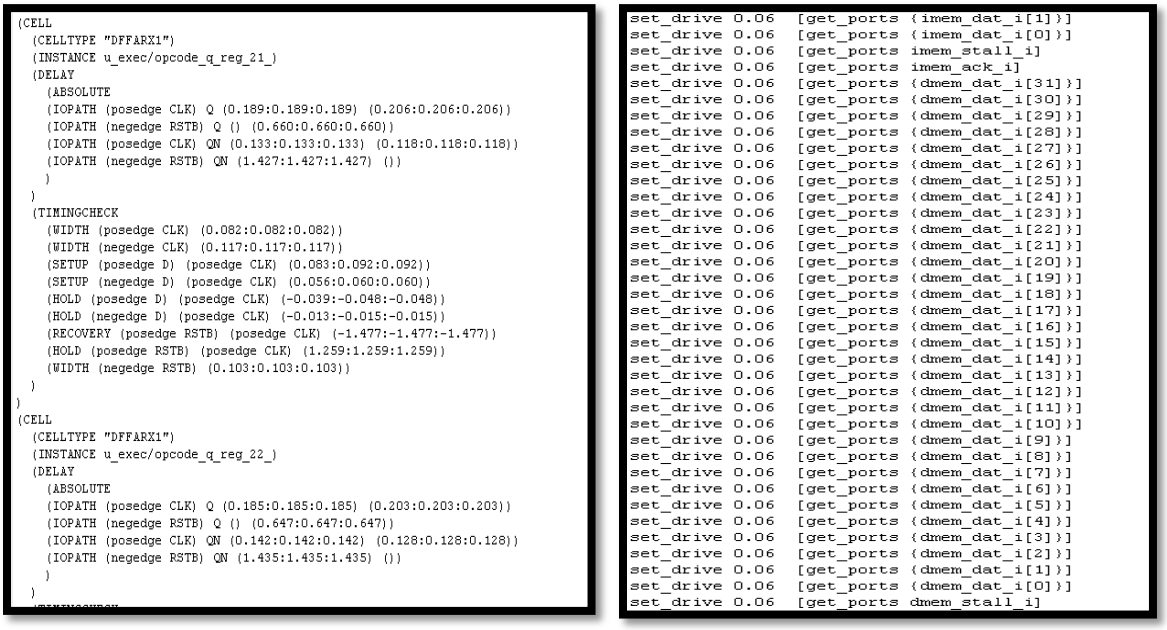
```
****************************************
Report : qor
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************


  Timing Path Group 'clk_i'
  -----------------------------------
  Levels of Logic:               20.00
  Critical Path Length:          10.57
  Critical Path Slack:           -0.65
  Critical Path Clk Period:      10.00
  Total Negative Slack:         -89.08
  No. of Violating Paths:       305.00
  Worst Hold Violation:           0.00
  Total Hold Violation:           0.00
  No. of Hold Violations:         0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:          10
  Hierarchical Port Count:         899
  Leaf Cell Count:                7663
  Buf/Inv Cell Count:             1945
  Buf Cell Count:                  120
  Inv Cell Count:                 1825
  CT Buf/Inv Cell Count:             0
  Combinational Cell Count:       6417
  Sequential Cell Count:          1246
  Macro Count:                       0
  -----------------------------------
```

```
  Area
  -----------------------------------
  Combinational Area:    60714.086468
  Noncombinational Area: 40190.055046
  Buf/Inv Area:          11884.032239
  Total Buffer Area:          1787.90
  Total Inverter Area:       10096.13
  Macro/Black Box Area:      0.000000
  Net Area:              14208.270593
  -----------------------------------
  Cell Area:            100904.141514
  Design Area:          115112.412108


  Design Rules
  -----------------------------------
  Total Number of Nets:          8780
  Nets With Violations:          1247
  Max Trans Violations:          1247
  Max Cap Violations:               0
  -----------------------------------


  Hostname: localhost.localdomain

  Compile CPU Statistics
  ---------------------------------------------
  Resource Sharing:                       1.99
  Logic Optimization:                     3.54
  Mapping Optimization:                  75.15
  ---------------------------------------------
  Overall Compile Time:                  82.39
  Overall Compile Wall Clock Time:       82.97
```

Εικόνα 19 QoR Report

Περιέχει στατιστικές πληροφορίες της σχεδίασης.

Ο DC παράγει επίσης τα δύο αρχεία το SDC (Synopsys Design Constraints) και το SDF (Standard Delay Format). Το πρώτο χρησιμοποιείται ως είσοδος σε άλλα εργαλεία για να επιβεβαιωθεί η ορθή λειτουργία της σχεδίασης. Εξάγεται για να ελέγξει αν το κύκλωμα δουλεύει στην συχνότητα που έχουμε ορίσει χωρίς να παρουσιάζει παραβάσεις setup/hold. Το δεύτερο μεταφέρει πληροφορίες χρονισμού, όπως πόση καθυστέρηση παρατηρείται σε πύλη, θύρα καλώδιο.

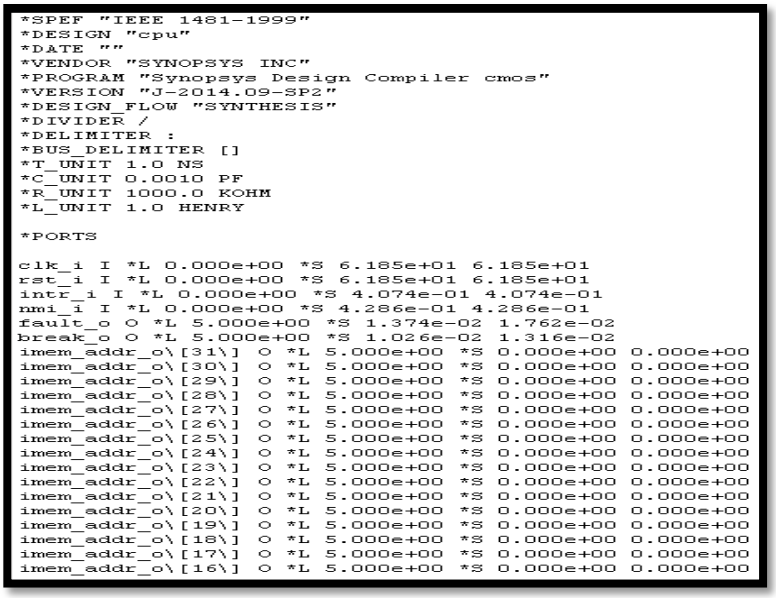


```
(CELL
  (CELLTYPE "DFFARX1")
  (INSTANCE u_exec/opcode_q_reg_21_)
  (DELAY
    (ABSOLUTE
    (IOPATH (posedge CLK) Q (0.189:0.189:0.189) (0.206:0.206:0.206))
    (IOPATH (negedge RSTB) Q () (0.660:0.660:0.660))
    (IOPATH (posedge CLK) QN (0.133:0.133:0.133) (0.118:0.118:0.118))
    (IOPATH (negedge RSTB) QN (1.427:1.427:1.427) ())
    )
  )
  (TIMINGCHECK
    (WIDTH (posedge CLK) (0.082:0.082:0.082))
    (WIDTH (negedge CLK) (0.117:0.117:0.117))
    (SETUP (posedge D) (posedge CLK) (0.083:0.092:0.092))
    (SETUP (negedge D) (posedge CLK) (0.056:0.060:0.060))
    (HOLD (posedge D) (posedge CLK) (-0.039:-0.048:-0.048))
    (HOLD (negedge D) (posedge CLK) (-0.013:-0.015:-0.015))
    (RECOVERY (posedge RSTB) (posedge CLK) (-1.477:-1.477:-1.477))
    (HOLD (posedge RSTB) (posedge CLK) (1.259:1.259:1.259))
    (WIDTH (negedge RSTB) (0.103:0.103:0.103))
  )
)
(CELL
  (CELLTYPE "DFFARX1")
  (INSTANCE u_exec/opcode_q_reg_22_)
  (DELAY
    (ABSOLUTE
    (IOPATH (posedge CLK) Q (0.185:0.185:0.185) (0.203:0.203:0.203))
    (IOPATH (negedge RSTB) Q () (0.647:0.647:0.647))
    (IOPATH (posedge CLK) QN (0.142:0.142:0.142) (0.128:0.128:0.128))
    (IOPATH (negedge RSTB) QN (1.435:1.435:1.435) ())
    )
  )
```

```
set_drive 0.06  [get_ports {imem_dat_i[1]}]
set_drive 0.06  [get_ports {imem_dat_i[0]}]
set_drive 0.06  [get_ports imem_stall_i]
set_drive 0.06  [get_ports imem_ack_i]
set_drive 0.06  [get_ports {dmem_dat_i[31]}]
set_drive 0.06  [get_ports {dmem_dat_i[30]}]
set_drive 0.06  [get_ports {dmem_dat_i[29]}]
set_drive 0.06  [get_ports {dmem_dat_i[28]}]
set_drive 0.06  [get_ports {dmem_dat_i[27]}]
set_drive 0.06  [get_ports {dmem_dat_i[26]}]
set_drive 0.06  [get_ports {dmem_dat_i[25]}]
set_drive 0.06  [get_ports {dmem_dat_i[24]}]
set_drive 0.06  [get_ports {dmem_dat_i[23]}]
set_drive 0.06  [get_ports {dmem_dat_i[22]}]
set_drive 0.06  [get_ports {dmem_dat_i[21]}]
set_drive 0.06  [get_ports {dmem_dat_i[20]}]
set_drive 0.06  [get_ports {dmem_dat_i[19]}]
set_drive 0.06  [get_ports {dmem_dat_i[18]}]
set_drive 0.06  [get_ports {dmem_dat_i[17]}]
set_drive 0.06  [get_ports {dmem_dat_i[16]}]
set_drive 0.06  [get_ports {dmem_dat_i[15]}]
set_drive 0.06  [get_ports {dmem_dat_i[14]}]
set_drive 0.06  [get_ports {dmem_dat_i[13]}]
set_drive 0.06  [get_ports {dmem_dat_i[12]}]
set_drive 0.06  [get_ports {dmem_dat_i[11]}]
set_drive 0.06  [get_ports {dmem_dat_i[10]}]
set_drive 0.06  [get_ports {dmem_dat_i[9]}]
set_drive 0.06  [get_ports {dmem_dat_i[8]}]
set_drive 0.06  [get_ports {dmem_dat_i[7]}]
set_drive 0.06  [get_ports {dmem_dat_i[6]}]
set_drive 0.06  [get_ports {dmem_dat_i[5]}]
set_drive 0.06  [get_ports {dmem_dat_i[4]}]
set_drive 0.06  [get_ports {dmem_dat_i[3]}]
set_drive 0.06  [get_ports {dmem_dat_i[2]}]
set_drive 0.06  [get_ports {dmem_dat_i[1]}]
set_drive 0.06  [get_ports {dmem_dat_i[0]}]
set_drive 0.06  [get_ports dmem_stall_i]
```

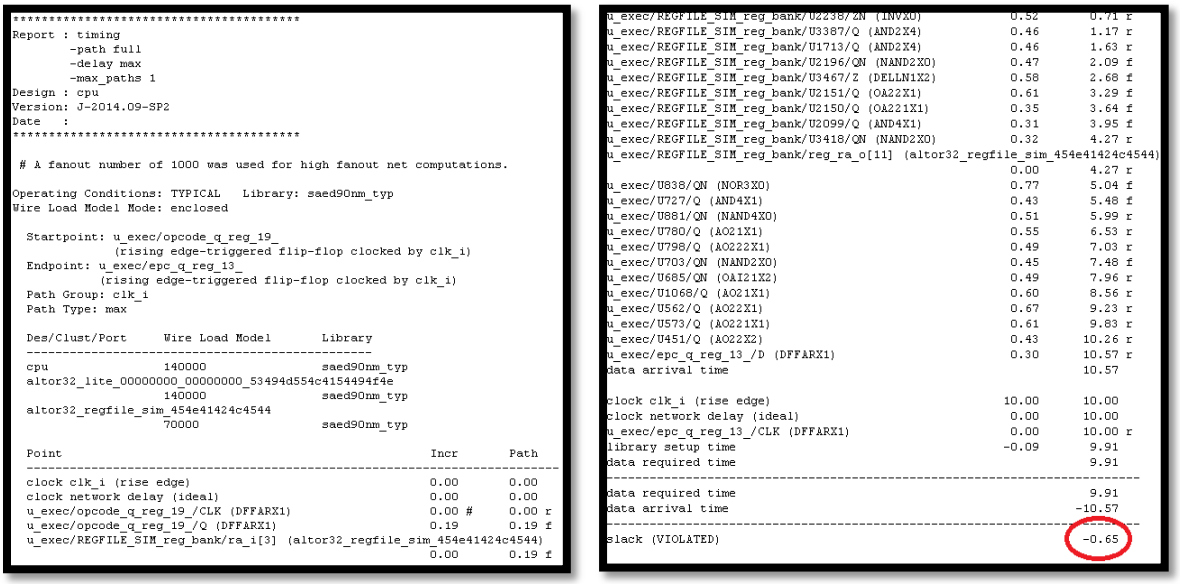
Εικόνα 20 Στιγμιότυπο sdf / Στιγμιότυπο sdc

Το .spef χρησιμοποείται από τον PrimeTime, περιλαμβάνει παρασιτικά αντίστασης και χωρητικότητας.



```
*SPEF "IEEE 1481-1999"
*DESIGN "cpu"
*DATE ""
*VENDOR "SYNOPSYS INC"
*PROGRAM "Synopsys Design Compiler cmos"
*VERSION "J-2014.09-SP2"
*DESIGN_FLOW "SYNTHESIS"
*DIVIDER /
*DELIMITER :
*BUS_DELIMITER []
*T_UNIT 1.0 NS
*C_UNIT 0.0010 PF
*R_UNIT 1000.0 KOHM
*L_UNIT 1.0 HENRY

*PORTS

clk_i I *L 0.000e+00 *S 6.185e+01 6.185e+01
rst_i I *L 0.000e+00 *S 6.185e+01 6.185e+01
intr_i I *L 0.000e+00 *S 4.074e-01 4.074e-01
nmi_i I *L 0.000e+00 *S 4.286e-01 4.286e-01
fault_o O *L 5.000e+00 *S 1.374e-02 1.762e-02
break_o O *L 5.000e+00 *S 1.026e-02 1.316e-02
imem_addr_o\[31\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[30\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[29\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[28\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[27\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[26\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[25\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[24\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[23\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[22\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[21\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[20\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[19\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[18\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[17\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
imem_addr_o\[16\] O *L 5.000e+00 *S 0.000e+00 0.000e+00
```

Εικόνα 21 Spef Report



```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : cpu
Version: J-2014.09-SP2
Date   :
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: TYPICAL   Library: saed90nm_typ
Wire Load Model Mode: enclosed

  Startpoint: u_exec/opcode_q_reg_19_
              (rising edge-triggered flip-flop clocked by clk_i)
  Endpoint: u_exec/epc_q_reg_13_
            (rising edge-triggered flip-flop clocked by clk_i)
  Path Group: clk_i
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  cpu                140000                saed90nm_typ
  altor32_lite_00000000_00000000_53494d554c4154494f4e
                     140000                saed90nm_typ
  altor32_regfile_sim_454e41424c4544
                     70000                 saed90nm_typ

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock clk_i (rise edge)                                 0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  u_exec/opcode_q_reg_19_/CLK (DFFARX1)                   0.00 #     0.00 r
  u_exec/opcode_q_reg_19_/Q (DFFARX1)                     0.19       0.19 f
  u_exec/REGFILE_SIM_reg_bank/ra_i[3] (altor32_regfile_sim_454e41424c4544)
                                                          0.00       0.19 f
```

```
u_exec/REGFILE_SIM_reg_bank/U2238/ZN (INVX0)            0.52       0.71 r
u_exec/REGFILE_SIM_reg_bank/U3387/Q (AND2X4)            0.46       1.17 r
u_exec/REGFILE_SIM_reg_bank/U1713/Q (AND2X4)            0.46       1.63 r
u_exec/REGFILE_SIM_reg_bank/U2196/QN (NAND2X0)          0.47       2.09 f
u_exec/REGFILE_SIM_reg_bank/U3467/Z (DELLN1X2)          0.58       2.68 f
u_exec/REGFILE_SIM_reg_bank/U2151/Q (OA22X1)            0.61       3.29 f
u_exec/REGFILE_SIM_reg_bank/U2150/Q (OA221X1)           0.35       3.64 f
u_exec/REGFILE_SIM_reg_bank/U2099/Q (AND4X1)            0.31       3.95 f
u_exec/REGFILE_SIM_reg_bank/U3418/QN (NAND2X0)          0.32       4.27 r
u_exec/REGFILE_SIM_reg_bank/reg_ra_o[11] (altor32_regfile_sim_454e41424c4544)
                                                        0.00       4.27 r
u_exec/U838/QN (NOR3X0)                                 0.77       5.04 f
u_exec/U727/Q (AND4X1)                                  0.43       5.48 f
u_exec/U881/QN (NAND4X0)                                0.51       5.99 r
u_exec/U780/Q (AO21X1)                                  0.55       6.53 r
u_exec/U798/Q (AO222X1)                                 0.49       7.03 r
u_exec/U703/QN (NAND2X0)                                0.45       7.48 f
u_exec/U685/QN (OAI21X2)                                0.49       7.96 r
u_exec/U1068/Q (AO21X1)                                 0.60       8.56 r
u_exec/U562/Q (AO22X1)                                  0.67       9.23 r
u_exec/U573/Q (AO221X1)                                 0.61       9.83 r
u_exec/U451/Q (AO22X2)                                  0.43      10.26 r
u_exec/epc_q_reg_13_/D (DFFARX1)                        0.30      10.57 r
data arrival time                                                 10.57

clock clk_i (rise edge)                                10.00      10.00
clock network delay (ideal)                             0.00      10.00
u_exec/epc_q_reg_13_/CLK (DFFARX1)                      0.00      10.00 r
library setup time                                     -0.09       9.91
data required time                                                 9.91
--------------------------------------------------------------------------
data required time                                                 9.91
data arrival time                                                -10.57
--------------------------------------------------------------------------
slack (VIOLATED)                                                 -0.65
```

Εικόνα 22 TimingReport

Το timing Report παρέχει πληροφορία για το χειρότερο μονοπάτι της σχεδίασης. Το τελικό αποτέλεσμα είναι η αφαίρεση του χρόνου άφιξης από τον απαιτούμενο χρόνο. Όταν η τιμή αυτή είναι αρνητική σημαίνει ότι ο χρόνος άφιξης είναι πολύ μεγαλύτερος από τον απαιτούμενο χρόνο και πρέπει να προσαρμόσουμε το ρολόι αντίστοιχα.

Το δεύτερο πιο σημαντικό αρχείο είναι το violators. Περιλαμβάνει πληροφορίες για τις παραβάσεις της σχεδίασης. Δύο τύποι παραβίασης όπως τους ορίσαμε στο τέλος του προηγούμενου κεφαλαίου είναι οι setup και hold.Όταν το σήμα του ρολογιού ταξιδεύει πιο αργά από το μονοπάτι από register σε register με αποτέλεσμα να επιτρέπεται να διεισδύσει το σήμα σε δύο καταχωρητές την ίδια χρονική στιγμή, έχουμε παραβίαση αναμονής, επειδή τα προηγούμενα δεδομένα δεν παραμένουν αρκετό χρονικό χρονικό διάστημα ώστε να επιτευχθεί σωστός χρονισμός. Είναι σημαντικό να σημειωθεί ότι δεν πρέπει να έχουμε παραβάσεις hold, σε περίπτωση εμφάνισής του θα πρέπει να αλλάξουμε τη σχεδίαση από την αρχή. Η δεύτερη παραβίαση που μπορεί να υπάρξει είναι η setupπου στην ουσία τα δεδομένα δεν προλαβαίνουν να φτάσουν στον προορισμό τους μέχρι τον επόμενο κτύπο ρολογιού. Αυτό μπορεί να διορθωθεί εύκολα αλλάζοντας την περίοδο του ρολογιού.

Τέλος, έχουμε τα αρχεία .v, .vhdστα οποία μεταφράζεται η καινούργια σχεδίαση σε γλώσσα verilog και vhdlαντίστοιχα.

5.3 PRIMETIME

Αφού ολοκληρώθηκε η διαδικασία του Design Compiler προχωρούμε στο επόμενο εργαλείο που προσφέρει η Synopsysto PrimeTime, που επικεντρώνεται πάνω στην στατική ανάλυση του χρονισμού, παρέχοντας πληροφορίες για τα παρακάτω.

- **Critical Path:**
  - Το μονοπάτι εντός του κυκλώματος το οποίο παρουσιάζει την μεγαλύτερη καθυστέρηση,

- **Arrival Time:**
  - Ο χρόνος που απαιτείται για να φτάσει ένα σήμα σε ένα συγκεκριμένο σημείο (κόμβο) του κυκλώματος.
- **Required Time:**
  - Η “τελευταία“ χρονική στιγμή κατά την οποία μπορεί να φτάσει ένα σήμα σε κάποιο συγκεκριμένο σημείο του κυκλώματος, χωρίς να αλλοιώσει τον προκαθορισμένο, από τον κύκλο ρολογιού, χρονισμό του κυκλώματος.
- **Slack:**
  - Μια μετρική που ισούται με την διαφορά μεταξύ του απαιτούμενου χρόνου (required time) και του χρόνου άφιξης (arrival time).
  - Θετικό slack: υπάρχει περιθώριο περεταίρω βελτιστοποίησης.
  - Αρνητικό slack:επιβάλλεται να γίνουν τροποποιήσεις, σε αντίθετη περίπτωση η καθυστέρηση στον κόμβο που παρατηρούμε ενδέχεται να προκαλέσει λογικά σφάλματα ή/και άυξηση της συνολικής καθυστέρησης του κυκλώματος.

Οι κατηγορίες χρονισμού χωρίζονται ως εξής:

✓ Static Timing Analysis - η πιο διαδεδομένη μέθοδος υπολογισμού του απαιτούμενου χρόνου για την ορθή λειτουργία ενός κυκλώματος.Ο τελικός χρονισμός απέχει από τον χρονισμό που επιδεικνύει η σχεδίαση κατά την λειτουργία της.

✓ Statistical Timing Analysis – αποτελεί υποκατηγορία της στατικής ανάλυσης χρονισμού, αντί τελικού αποτελέσματος, έχουμε παραγωγή μιας τελικής κατανομής πιθανών χρονισμών εξόδου.

- ✓ Dynamic Timing Analysis - ελέγχει τον χρονισμό της σχεδίασης βάσει μιας ομάδας test vectors που καθορίζει τις τιμές στις εισόδους της, γενικά είναι πολύ αργή μέθοδος.

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

Τρέξαμε το εργαλείο αυτό χρησιμοποιώντας όλες τις παραπάνω βελτιστοποιήσεις που χρησιμοποιήσαμε στον DC. Ο PrimeTime καλείται και αυτός σε περιβάλλον Unix με την εντολή **pc_shell,** όπου ανοίγει σε επίπεδο κονσόλας. Αν θέλουμε να δουλέψουμε πάνω στο γραφικό περιβάλλον του PT δεν έχουμε παρά να δώσουμε την εντολή **gui_start**. Όλα τα τρεξίματα έχουν γίνει σε επίπεδο κονσόλας χρησιμοποιώντας scripts τύπου .tcl, που υποστηρίζει το εργαλείο.

Toscriptπου ακολουθεί είναι πανομοιότυπο και για τις τέσσερις τεχνικές: flatten, ungroup, uniquify, simple.

Όπως και στο προηγούμενο βήμα, αρχικά πρέπει να δημιουργήσουμε και να ορίσουμε το φάκελο εργασίας. Αυτό επιτυγχάνεται με τις παρακάτω εντολές:

shrm -rf ./desired_directory

filemkdir ./desired_directory

Στη συνέχεια ορίζουμε και συνδέουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε, με τον ίδιο τρόπο που κάναμε και στον DC.

set project_root "./desired_directory/ SRC"

set lib_root "/ desired_directory /LIB"

set search_path "$lib_root $project_root"

set target_library "/ desired_directory /LIB/saed90nm_typ.db"

setlink_path "* $target_library"

Η επόμενη εντολή είναι η "**read_file –format verilog /desired_directory/cpu_syn.v"** διαβάζει τη σχεδίαση που έχει προκύψει από τον DC, άρα για κάθε τεχνική διαβάζει το αντίστοιχο αρχείο. Η εκτέλεσή της είναι ισοδύναμη με τις εντολέςanalyzeκαι elaborate.

Έπειτα ορίζουμε ποια θα είναι η τρέχουσα σχεδίαση με την εντολή "**current_design cpu** ", δηλώνουμε το top cell με λίγα λόγια.

Επόμενο βήμα είναι να συνδέσουμε τη σχεδίαση ώστε να λυθούν όλες οι σχεδιαστικές αναφορές. Εκτός από αυτό, η εντολή "**link**" που ακολουθεί χτίζει την εσωτερική αναπαράσταση της σχεδίασης για να είναι έτοιμη για ανάλυση. Στόχος είναι να εντοπιστούν όλα τα σχεδιαστικά στοιχεία και τα στοιχεία της βιβλιοθήκης και να συνδεθούν με τη τρέχουσα σχεδίαση.

Η"**read_sdc -echo /desired_directory/cpu_syn.sdc**" διαβάζει το αρχείο που περιλαμβάνει τους χρονικούς περιορισμούς της σχεδίασης και έχει δημιουργηθεί από τον DC.

Βασική προϋπόθεση για τον PrimeTime είναι ο ορισμός της συχνότητας του ρολογιού για αυτό το λόγο ακολουθεί η εντολή "**create_clock -period 12.5 [get_ports clk_i]**".

Το "**set_propagated_clock [get_ports clk_i]"** παίρνει το ρολόι και το προωθεί στη σχεδίαση, οπότε θα υπάρχει σίγουρα skew. Με τον όρο Clock skew εννοούμε τον διαμοιρασμό του ρολογιού στα ακολουθιακά στοιχεία του κυκλώματος, ακόμη κι αν τα στοιχεία βρίσκονται στο ίδιο επίπεδο παρουσιάζονται καθυστερήσεις.

Η εντολή αυτή είναι αλληλένδετη με την από πάνω "**set_clock_uncertainty 0.2 [get_ports clk_i]"**γιατί στην ουσία προσπαθεί να ορίσει την καθυστέρηση που θα προκληθεί εξαιτίας set_propagated_clock.

Πριν προχωρήσουμε πρέπει να ελέγξουμε για τυχόν χρονικές παραβάσεις, δίνοντας την εντολή "**check_timing**", στην περίπτωση ύπαρξης παραβιάσεων εκτυπώνεται κάποιο μήνυμα, για να συνεχίσουμε πρέπει να το επιλύσουμε.

Ακολουθούν οι εντολές για την παραγωγή των αρχείων που περιέχουν πληροφορίες για το χρονισμό ολόκληρης της σχεδίασης

report_timing > /desired_directory /cpu_syn_pt.timing

καιαπόκαταχωρητήσεκαταχωρητή

report_timing -from [all_registers -clock_pins] -to [all_registers -data_pins] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_r2r_max.timing

report_timing -from [all_registers -clock_pins] -to [all_registers -data_pins] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_r2r_min.timing

report_timing -from [all_registers -clock_pins] -to [all_outputs] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_r2o_max.timing

report_timing -from [all_registers -clock_pins] -to [all_outputs] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_r2o_min.timing

report_timing -from [all_inputs] -to [all_registers -data_pins] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_i2r_max.timing

report_timing -from [all_inputs] -to [all_registers -data_pins] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net > / desired_directory /cpu_syn_pt_i2r_min.timing

report_constraints -all_violators > / desired_directory /cpu_syn_pt.violators

report_clock_timing -type skew -verbose > / desired_directory /cpu_syn_pt.skew

Από τα παραπάνω αρχεία θα πρέπει να δώσουμε μεγάλη προσοχή στο .violators. πρέπει για ακόμη μια φορά να απουσιάζουν παραβιάσεις τύπου hold. Με

την επιτυχημένη ολοκλήρωση του PrimeTime για να συνεχίσουμε την ροή της σχεδίασης θα πρέπει να αποφασίσουμε ποια από τις παραπάνω τεχνικές θα κρατήσουμε. Για να πάρουμε αυτή την απόφαση θα πρέπει να συγκρίνουμε τα αποτελέσματα που αφορούν στο χρονισμό και στην καταναλισκόμενη ισχύ, εφόσον καμία δεν παρουσιάζει παραβιάσεις που να καθιστούν τη σχεδίαση αποτυχημένη. Θα επιλέξουμε εκείνη που συνδυάζει καλύτερα αποτελέσματα.

Εξαιτίας του τρόπου σχεδίασης του κυκλώματος επιλέγουμε το simple script. Κρατάμε το απλό, γιατί δεν διαταράσσει την ιεραρχία της σχεδίασης και τα αποτελέσματα του timingκαι του power δεν είναι τα χειρότερα δυνατά σε σύγκριση με τα υπόλοιπα.

Συνεχίζοντας με τη ροή της δικής μας σχεδίασης όπως έχει ήδη αναφερθεί αυτή που επιλέξαμε είναι η απλή. Σειρά έχει να ξαναπεράσουμε τη συγκεκριμένη σχεδίαση από τον Design Compiler πραγματοποιώντας κάποιες αλλαγές στο script, ώστε να προσπαθήσουμε να βελτιστοποιήσουμε τη σχεδίαση όσο γίνεται περισσότερο.

Πάλι θα χρησιμοποιήσουμε δύο τρόπους με δύο διαφορετικά scripts. Η διαφορά θα έγκειται στον τρόπο μεταγλώττισης.

- ✓ Compile- με την εντολή αυτή εκκινείται η διαδικασία της σύνθεσης και της βελτιστοποίησης της τρέχουσας σχεδίασης. Κατά το compilation εμφανίζεται στην κονσόλα του εργαλείου ένα progress report που περιγράφει με ποιόν τρόπο βελτιστοποιείται η σχεδίαση (π.χ. πόσα passes γίνονται κατά την διαδικασία βελτιστοποίησης της καταλαμβανόμενης από το κύκλωμα συνολικής περιοχής κλπ.).

- ✓ Compile Ultra–όπως και με το compile, με την εντολή αυτή εκκινείται η διαδικασία της σύνθεσης. Η διαφορά είναι ότι πραγματοποιεί το high-effort compile, δηλαδή πιο αυστηρά κριτήρια και δεν ανέχεται σχεδιαστικές παρεκκλίσεις.

Θα αλλάξουμε επίσης, την περίοδο ρολογιού από 12.5 σε 16.6 (60MHz)και πλέον η παράμετρος areaθα έχει τιμή. Θα πάρουμε το αποτέλεσμα που παρήγαγε ή πρώτη φορά που περάσαμε το scriptαπό τον DCκαι θα του προσθέσουμε το 20% αυτού η νέα τιμή θα είναι 132487.0.

Area + (area *20%) = max_area_value

Ξεκινώντας την ανάλυση του compile script έχουμε τις ίδιες εντολές που έχουμε δει μέχρι στιγμής. Αφαιρούμε τυχόν προηγούμενες σχεδιάσεις, καθορίζουμε τη θέση αποθήκευσης των αρχείων, ορίζουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε, προσχωρούμε σε analyzeκαι elaborateτης σχεδίασης, δηλώνουμε τη νέα και περίοδο και area, οι υπόλοιποι περιορισμοί που είχαμε παραμένουν ίδιοι. Επόμενο βήμα είναι να εκτελεστεί η εντολή "**compile -map_effort high**". Το υπόλοιπο script παραμένει αμετάβλητο.

Τρέχουμε και το αντίστοιχο για το compile ultra, με μόνη διαφορά στην εντολή "**compile_ultra -timing_high_effort_script".**

Συγκρίνουμε τα αποτελέσματα και κρατάμε το αποτέλεσμα του compile_ultra script, γιατί το STA (Static Timing Analysis) μας δίνει λιγότερα μονοπάτια με setup violations και μικρότερα (χρονικά) setup violations.

5.4 DESIGN FOR TESTABILITY

Μετά την ολοκλήρωση της επιλογής ανάμεσα στις δυο σχεδιάσεις (compile, compile_ultra) σειρά έχει η ανάλυση του κυκλώματος, με χρήση του DFT (Design For Testability) εργαλείου της Synopsys. Αποτελεί μέρος του Design Compiler και μπορεί χρησιμοποιηθεί κατευθείαν από την κονσόλα μέσω ενός τυπικού script. Όπως δηλώνει και το όνομα του είναι εκείνο το σύνολο των σχεδιαστικών τεχνικών που χρησιμοποιούνται στη βιομηχανία προκειμένου να εμπλουτίσουν τις δυνατότητες ελέγχου μιας σχεδίασης. Στόχο έχει να ελεγχθεί η ορθή λειτουργία της σχεδίασης πριν από τη 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” δεδομένων (data patterns) μετακινείται εντός του κυκλώματος έως ότου φτάσει στην έξοδό του, όπου τα τελικά αποτελέσματα συγκρίνονται από τον σχεδιαστή έναντι κάποιων προϋπολογισμένων “καλών” αποτελεσμάτων.

Η παραπάνω διαδικασία ονομάζεται αλυσίδα σάρωσης (scan chain) και αποτελείται από τα επόμενα σήματα που βοηθούν στην παρακολούθηση του συνολικού μηχανισμού ελέγχου:

- ✓ Scan In – σήμα εισόδου του scan chain.

- ✓ Scan Out – σήμα εξόδου 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 – 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.

Οι αλλαγές που μπορούν να γίνουν στο κύκλωμα ώστε να υπάρξει καλύτερη πρόσβαση στα εσωτερικά κυκλωματικά στοιχεία είναι οι εξής:

- ✓ Controllability: Η δυνατότητα να θέσουμε κάποιους κόμβους του κυκλώματος στη λογική τιμή που θέλουμε.

- ✓ Observability: Η δυνατότητα να παρατηρήσουμε τις τιμές των εσωτερικών κόμβων της σχεδίασης.

Στα πλεονεκτήματα χρήσης του DFT και συγκεκριμένα της μεθόδου scanchaineίναι ότι αυξάνει το ποσοστό των λαθών που μπορούν να ανιχνευτούν, μειώνει το συνολικό χρόνο ελέγχου της σχεδίασης και την απαιτούμενη καταναλισκόμενη ισχύ. Τέλος, υποστηρίζει τον παράλληλο έλεγχο πολλαπλών διασυνδέσεων εντός του κυκλώματος.

Το script που χρησιμοποιήσαμε περιλαμβάνει τις εντολές τις ίδιες εντολές με πριν, διαβάζουμε θέτουμε τους κατάλληλους περιορισμούς και οπωσδήποτε περιλαμβάνουμε την εντολή “**set_default_scan_style multiplexed_flip_flop**” όπου ορίζουμε ότι το κύκλωμα θα ελεγχθεί με τον τυπικό τρόπο που δηλώσαμε παραπάνω.

Επόμενη καινούργια εντολή που συναντάμε είναι η “**set_scan_configuration**”, δηλώνει τον αριθμό των συνολικών αλυσίδων που θα σχηματιστούν. Τέλος, προσθέτουμε τις εντολές για να παραχθούν τα reports, με τον ίδιο τρόπο με πριν.

5.5 FORMALITY

Με το που ολοκληρωθεί το στάδιο του DFT πρέπει να ελεγχθεί η αρχική με την παρούσα σχεδίαση ώστε να διαπιστωθούν κατά πόσο τα εργαλεία που χρησιμοποιήθηκαν δεν αλλοίωσαν την λειτουργικότητα του κυκλώματος.

Formal Verification ονομάζεται η διαδικασία απόδειξης της ισοδυναμίας μεταξύ δύο διαφορετικών περιγραφών (σε επίπεδο αφαίρεσης) ενός κυκλώματος. Αξίζει να σημειώσουμε πως η διαδικασία αυτή είναι ιδιαιτέρως χρήσιμη για συνδυαστικά κυκλώματα και ψηφιακά κυκλώματα με εσωτερικές αναπαραστάσεις μνήμης. Το εργαλείο CAD που χρησιμοποιούμε 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.

Δημιουργούμε και θέτουμε τη θέση αποθήκευσης:

shrm -rf /desired_directory

file mkdir / desired_directory

Η επόμενη διαβάζει ένα .svf αρχείο το οποίο παράγεται αυτόματα κατά την διαδικασία της σύνθεσης και περιέχει πληροφορίες για τον τρόπο που αυτή έχει επιτελεστεί.

set_svf -append { / desired_directory /cpu_syn.svf }

Διαβάζει την αρχική σχεδίαση

read_verilog -container i -libname WORK -01 { cpu_syn.v }

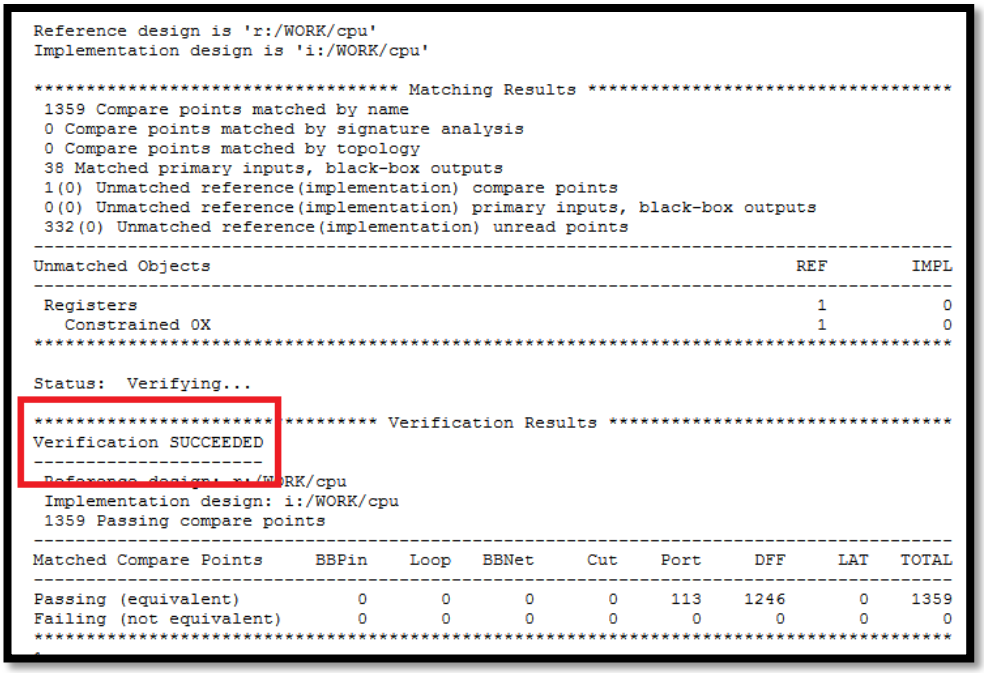
Διαβάζει την τεχνολογική βιβλιοθήκη που έχει χρησιμοποιηθεί κατά την σύνθεση του κυκλώματος

read_db { LIB/saed90nm_typ.db }

verify

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.

Στην περίπτωσή μας το αποτέλεσμα επαληθεύει την ισοδυναμία των περιγραφών όπως φαίνεται και παρακάτω.



Εικόνα 23 Formatily Ouput

Ροή σχεδίασης New_Netlist (συνέχεια)

Ξανά πίσω στη δική μας σχεδίαση το μόνο που έχει απομείνει για να ολοκληρωθεί η front-endσχεδίαση είναι να περάσουμε την περιγραφή κυκλώματος που έχει προκύψει από το DFT, στον PrimeTime.

Χρησιμοποιούμε το ίδιο script με πριν με διαφορετική περίοδο ρολογιού και πλέον διαβάζει το Netlist που έχει προκύψει από το DFT. Συγκρίνοντας τα αρχεία χρονισμού (timing reports) που προέκυψαν τη πρώτη φορά που τρέξαμε τον PT για την αρχική σχεδίαση με την παρούσα σχεδίαση βλέπουμε να υπάρχει βελτίωση στο χρονισμό. Άρα, μπορούμε να παρατηρήσουμε πως όντως οι περιορισμοί που ορίσαμε είχαν θετική επίδραση στη σχεδίαση. Κάπως έτσι τελειώνει σε αυτό το σημείο η front-endροή σχεδίασης.

6 SCRIPTS

Στο παρόν κεφάλαιο θα παρουσιάσουμε τα scripts που χρησιμοποιήθηκαν κατά τη διάρκεια της σχεδίασης, η ανάλυση των εντολών πραγματοποιήθηκε στο προηγούμενο κεφάλαιο. Για τον Design compiler χρησιμοποιήθηκαν 4 scripts.

```
#dc_shell -f dc_script.tcl
#real clock
remove_design -all

sh rm -rf /home/edatools/louiza/alu/SYN_RCLOCK_louiza
file mkdir /home/edatools/louiza/alu/SYN_RCLOCK_louiza

set design_name cpu

set_svf /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.svf

set project_root "/home/edatools/louiza/alu/SRC"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set synthetic_library "dw_foundation.sldb"
set link_library "$target_library $synthetic_library"
set my_verilog_files [list altor32.v altor32_lite.v altor32_alu.v  altor32_regfile_alt.v
                           altor32_regfile_xil.v altor32_defs.v altor32_regfile_sim.v]

define_design_lib WORK -path /home/edatools/louiza/alu/WORK_SYN_RCLOCK_louiza

analyze -format verilog $my_verilog_files
elaborate cpu

current_design cpu

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]
set_dont_touch_network [list clk_i rst_i]
set_fix_hold clk_i

set_drive 0.06 [all_inputs]
set_load 5 [all_outputs]
set_input_delay 0.5 -clock clk_i [all_inputs]
set_output_delay 0.5 -clock clk_i [all_outputs]
set_max_area 0

compile -map_effort high -area_effort high -incremental

remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]
```

```
set_fix_multiple_port_nets -all

check_design

report_area > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.area
report_timing > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.timing
report_qor > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.qor
report_power > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.power
report_design > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.design
report_constraint -all_violators > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.violators
report_hierarchy > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.hierarchy
report_resources > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.resources
report_cell > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.cells
report_net > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.nets
report_port > /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.ports

write -hierarchy -format verilog -output /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.v
write -hierarchy -format vhdl -output /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.vhd
write -format vhdl -output /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn_no_hier.vhd
write_sdc /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.sdc
write_sdf -version 2.1 /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.sdf
write_parasitics -output /home/edatools/louiza/alu/SYN_RCLOCK_louiza/${design_name}_syn.spef
saif map -type ptpx -write map /home/edatools/louiza/alu/SYN RCLOCK louiza/${design name} syn.SAIF.namemap
```

Εικόνα 24 Script simple

```
#dc_shell -f dc_script.tcl
#real clock
#no dont_touch
#ungroup
remove_design -all

sh rm -rf /home/edatools/louiza/alu/SYN_UNGROUP_louiza
file mkdir /home/edatools/louiza/alu/SYN_UNGROUP_louiza

set design_name cpu

set_svf /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.svf

set project_root "/home/edatools/louiza/alu/SRC"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set synthetic_library "dw_foundation.sldb"
set link_library "$target_library $synthetic_library"
set my_verilog_files [list altor32.v altor32_lite.v altor32_alu.v  altor32_regfile_alt.v
                           altor32_regfile_xil.v altor32_defs.v altor32_regfile_sim.v]

define_design_lib WORK -path /home/edatools/louiza/alu/WORK_SYN_UNGROUP_louiza

analyze -format verilog $my_verilog_files
elaborate cpu

current_design cpu

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]
set_fix_hold clk_i

set_drive 0.06 [all_inputs]
set_load 5 [all_outputs]
set_input_delay 0.5 -clock clk_i [all_inputs]
set_output_delay 0.5 -clock clk_i [all_outputs]
set_max_area 0

ungroup -all

compile -map_effort high -area_effort high -incremental
```

```
remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

change_names -rules verilog -hierarchy

check_design

remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

set_fix_multiple_port_nets -all

check_design

report_area > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.area
report_timing > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.timing
report_qor > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.qor
report_power > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.power
report_design > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.design
report_constraint -all_violators > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.violators
report_hierarchy > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.hierarchy
report_resources > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.resources
report_cell > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.cells
report_net > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.nets
report_port > /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.ports

write -hierarchy -format verilog -output /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.v
write -hierarchy -format vhdl -output /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.vhd
write -format vhdl -output /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn_no_hier.vhd
write_sdc /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.sdc
write_sdf -version 2.1 /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.sdf
write_parasitics -output /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.spef
saif_map -type ptpx -write_map /home/edatools/louiza/alu/SYN_UNGROUP_louiza/${design_name}_syn.SAIF.namemap
```

Εικόνα 25 Ungroup script

```
#dc_shell -f dc_script.tcl
#real clock
#uniquify
remove_design -all

sh rm -rf /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza
file mkdir /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza

set design_name cpu

set_svf /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.svf

set project_root "/home/edatools/louiza/alu/SRC"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set synthetic_library "dw_foundation.sldb"
set link_library "$target_library $synthetic_library"
set my_verilog_files [list altor32.v altor32_lite.v altor32_alu.v  altor32_regfile_alt.v
                           altor32_regfile_xil.v altor32_defs.v altor32_regfile_sim.v]

define_design_lib WORK -path /home/edatools/louiza/alu/WORK_SYN_UNIQUIFY_louiza

analyze -format verilog $my_verilog_files
elaborate cpu

current_design cpu

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]
set_dont_touch_network [list clk_i rst_i]
set_fix_hold clk_i

set_drive 0.06 [all_inputs]
set_load 5 [all_outputs]
set_input_delay 0.5 -clock clk_i [all_inputs]
set_output_delay 0.5 -clock clk_i [all_outputs]
set_max_area 0

uniquify

compile -map_effort high -area_effort high -incremental
```

```
remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

change_names -rules verilog -hierarchy

check_design

remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

set_fix_multiple_port_nets -all

check_design

report_area > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.area
report_timing > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.timing
report_qor > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.qor
report_power > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.power
report_design > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.design
report_constraint -all_violators > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.violators
report_hierarchy > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.hierarchy
report_resources > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.resources
report_cell > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.cells
report_net > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.nets
report_port > /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.ports

write -hierarchy -format verilog -output /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.v
write -hierarchy -format vhdl -output /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.vhd
write -format vhdl -output /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn_no_hier.vhd
write_sdc /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.sdc
write_sdf -version 2.1 /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.sdf
write_parasitics -output /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.spef
saif_map -type ptpx -write_map /home/edatools/louiza/alu/SYN_UNIQUIFY_louiza/${design_name}_syn.SAIF.namemap
```

Εικόνα 26 Uniquify script

```
#dc_shell -f dc_script.tcl
#real clock
#flatten
remove_design -all

sh rm -rf /home/edatools/louiza/alu/SYN_FLATTEN_louiza
file mkdir /home/edatools/louiza/alu/SYN_FLATTEN_louiza

set design_name cpu

set_svf /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.svf

set project_root "/home/edatools/louiza/alu/SRC"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set synthetic_library "dw_foundation.sldb"
set link_library "$target_library $synthetic_library"
  set my_verilog_files [list altor32.v altor32_lite.v altor32_alu.v  altor32_regfile_alt.v
                            altor32_regfile_xil.v altor32_defs.v altor32_regfile_sim.v]

define_design_lib WORK -path /home/edatools/louiza/alu/WORK_SYN_FLATTEN_louiza

analyze -format verilog $my_verilog_files
elaborate cpu

current_design cpu

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]
set_dont_touch_network [list clk_i rst_i]
set_fix_hold clk_i

set_drive 0.06 [all_inputs]
set_load 5 [all_outputs]
set_input_delay 0.5 -clock clk_i [all_inputs]
set_output_delay 0.5 -clock clk_i [all_outputs]
set_max_area 0

set_flatten true
set_structure true
compile -map_effort high -area_effort high -incremental
```

```
remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

change_names -rules verilog -hierarchy

check_design

remove_unconnected_ports -blast_buses [find -hierarchy cell {"*"}]
remove_unconnected_ports [get_cells *]

set_fix_multiple_port_nets -all

check_design

report_area > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.area
report_timing > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.timing
report_qor > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.qor
report_power > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.power
report_design > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.design
report_constraint -all_violators > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.violators
report_hierarchy > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.hierarchy
report_resources > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.resources
report_cell > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.cells
report_net > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.nets
report_port > /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.ports

write -hierarchy -format verilog -output /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.v
write -hierarchy -format vhdl -output /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.vhd
write -format vhdl -output /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn_no_hier.vhd
write_sdc /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.sdc
write_sdf -version 2.1 /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.sdf
write_parasitics -output /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.spef
saif_map -type ptpx -write_map /home/edatools/louiza/alu/SYN_FLATTEN_louiza/${design_name}_syn.SAIF.namemap
```

Εικόνα 27 flatten

Ακολουθεί toscriptπου χρησιμοποιήθηκε για τον PrimeTime είναι το ίδιο και για τις 4 τεχνικές, αυτό που διέφερε είναι ο φάκελος απ' όπου δίαβαζε το αντίστοιχο αρχείο sdcκάθε σχεδίασης.

```
#pt_shell -f pt_script.tcl

sh rm -rf /home/edatools/louiza/alu/PT_RCLOCK_louiza
file mkdir /home/edatools/louiza/alu/PT_RCLOCK_louiza

set project_root "/home/edatools/louiza/alu/SRC2"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set link_path "* $target_library"

read_file -format vhdl /home/edatools/louiza/alu/SYN_RCLOCK_louiza/cpu_syn.vhd

current_design cpu
link_design

read_sdc -echo /home/edatools/louiza/alu/SYN_RCLOCK_louiza/cpu_syn.sdc

create_clock -period 12.5 -waveform {0 6.25} [get_ports clk_i]
check_timing
set_propagated_clock [get_ports clk_i]
set_clock_uncertainty 0.2 [get_ports clk_i]

report_timing > /home/edatools/louiza/alu/PT_RCLOCK_louiza/cpu_syn_pt.timing

#Register to register reports.
report_timing -from [all_registers -clock_pins] -to [all_registers -data_pins] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_r2r_max.timing
report_timing -from [all_registers -clock_pins] -to [all_registers -data_pins] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_r2r_min.timing
report_timing -from [all_registers -clock_pins] -to [all_outputs] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_r2o_max.timing
report_timing -from [all_registers -clock_pins] -to [all_outputs] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_r2o_min.timing
report_timing -from [all_inputs] -to [all_registers -data_pins] -delay_type max -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_i2r_max.timing
report_timing -from [all_inputs] -to [all_registers -data_pins] -delay_type min -path_type full_clock -nosplit -max_paths 1 -nworst 1 -trans -cap -net
                    > ./cpu_syn_pt_i2r_min.timing
report_constraints -all_violators
                    > ./cpu_syn_pt.violators
report_clock_timing -type skew -verbose
                    > ./cpu_syn_pt.skew
```

Εικόνα 28 PrimeTime

Το script DFT είναι το επόμενο

```
#dc_shell -f dft_script.tcl

set design_name cpu

sh rm -rf /home/edatools/louiza/alu/SYN_RCLOCK_basic_design_after_DFT
file mkdir /home/edatools/louiza/alu/SYN_RCLOCK_basic_design_after_DFT

set project_root "/home/edatools/louiza/alu/SYN_RCLOCK_basic_design/"
set lib_root "/home/edatools/louiza/alu/LIB"
set search_path "$lib_root $project_root"
set target_library "/home/edatools/louiza/alu/LIB/saed90nm_typ.db"
set synthetic_library "dw_foundation.sldb"
set link_library " * $target_library $synthetic_library"

set myFiles "/home/edatools/louiza/alu/SYN_RCLOCK_basic_design/cpu_syn.vhd"
remove_design -all

define_design_lib WORK -path /home/edatools/louiza/alu/WORK_SYN_DFT_basic_design_after_DFT

analyze -format vhdl $myFiles
elaborate cpu

current_design cpu

link

set test_default_scan_style multiplexed_flip_flop

create_clock -period 16.6 -waveform {0 8.3} [get_ports clk_i]
set_dont_touch_network [list clk_i rst_i]
set_fix_hold clk_i
set_fix_multiple_port_nets -all -buffer_constants

compile -map_effort high -exact_map

check_design
report_constraint -all_violators
```

Εικόνα 29 DFT

Τελός το script Formality

```
#fm_shell -f fm_script.tcl

sh rm -rf /home/edatools/louiza/alu/FM_RCLOCK
file mkdir /home/edatools/louiza/alu/FM_RCLOCK

set_svf -append { /home/edatools/louiza/alu/SYN_RCLOCK_basic_design/cpu_syn.svf }

read_verilog -container r -libname WORK -01 { altor32.v altor32_alu.v altor32_lite.v altor32_regfile_xil.v
altor32_regfile_sim.v altor32_regfile_alt.v altor32_defs.v }

set hdlin_dwroot /home/edatools/Synthesis-2011.09-SP3
set_top r:/WORK/cpu
read_verilog -container i -libname WORK -01 { /home/edatools/louiza/alu/SYN_RCLOCK_basic_design_after_DFT/cpu_syn.v }

read_db { /home/edatools/louiza/alu/LIB/saed90nm_typ.db }

set_top i:/WORK/cpu

match

verify
```

Εικόνα 30 Formality

Τέλος, θα παρουσιάσουμε ένα απλό αρχείο .do που χρησιμοποιήσαμε για την προσομοίωση του κυκλώματος στο Modelsim.

```
#################################
# A very simple modelsim do file #
#################################

# 1) Create a library for working in
vlib work

# 2) Compile the half adder
vcom -93 -explicit -work work altor32.v altor32_lite.v altor32_alu.v
                   altor32_regfile_xil.v altor32_regfile_alt.v  altor32_regfile_sim.v altor32_defs.v

# 3) Load it for simulation
vsim cpu

# 4) Open some selected windows for viewing
view structure
view signals
view wave

# 5) Show some of the signals in the wave window
add wave -noupdate -divider -height 32 Inputs
add wave -noupdate clk_i
add wave -noupdate rst_i


# 6) Set some test patterns

# clk_i = 0, rst_i = 0 at 0 ns
force clk_i 0 0
force rst_i 0 0

# clk_i = 1, rst_i = 0 at 12.5 ns
force clk_i 1 12.5
force rst_i 0 12.5

# clk_i = 0, rst_i = 1 at 25 ns
force clk_i 0 25
force rst_i 1 25

# clk_i = 1, rst_i = 1 at 37.5 ns
force clk_i 1 37.5
force rst_i 1 37.5

# clk_i = 0, rst_i = 0 at 37.5 ns
force clk_i 0 55
force rst_i 0 55

# clk_i = 1, rst_i = 1 at 37.5 ns
force clk_i 1 67.5
force rst_i 1 67.5


# 7) Run the simulation for 400 ns
run 400ns
```

Εικόνα 31 Modelsim .do file

7 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Antonios Dadaliaris. Reliability Driven Placement Algorithms. PhD thesis, Computer Science Dept., University Of Thessaly, June 2012.

[2] Kim, N.S.; Austin, T.; Baauw, D.; Mudge, T.; Flautner, K.; Hu, J.S.; Irwin, M.J.; Kandemir, M.; Narayanan, V., "Leakage current: Moore's law meets static power," Computer , vol.36, no.12, pp.68,75, Dec. 2003.

[3] VHDL Programming by Example, Douglas L. Perry, McGraw – Hill Fourth

[4] ASIC Design with Synopsys, Himanshu Bhatnagar

[5] ModelSIM SE, Manual

[6] Design Compiler™, Command – Line Interface Guide

[7] Design Compiler™, Reference Manual

[8] Design Compiler™, User Guide

[9] Design Compiler™, Tutorial Using Design Vision

[10] Power Compiler™, User Guide

[11] PrimePower™, Manual

[12] PrimePower™, Quick Reference Guide

[13] PrimeTime™, User Guide

[14] PrimeTime™, Quick Reference Guide

[15] Using TCL with Synopsys Tools

[16] A Comparison of Hierarchical Compile Strategies, Steve Golson

[17] 20. My Favorite dc_shell Tricks, Steve Golson

[18] Measuring the Power at the VHDL Netlist Level, A. Th. Schwarzbacker, P. A. Comiskey, J.B. Foley

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

[20] Golshan, K. “Physical Design Essentials: An ASIC Design Implementation Perspective”.New York: Springer (2007). ISBN 0-387-36642-3

[21] Lavagno, Martin and Scheffer. “Electronic Design Automation For Integrated Circuits Handbook”. (2006). ISBN 0-8493-3096-3

[22] Jansen, D. “The Electronic Design Automation Handbook”. Kluwer Academic Publishers. (2003). ISBN 1-4020-7502-2

[23] M. Pedram, “Power minimization in IC design: Principles and applications” ACM Trans. Design Automat. Electron. Syst., vol. 1, no.1, pp. 2-56, 1990.

[24] .Sangiovanni-Vincentelli, A. , "The tides of EDA," Design & Test of Computers, IEEE, Nov.-Dec. 2003

[25] Golshan, K. “Physical Design Essentials: AnASIC Design Implementation Perspective”. New York: Springer (2007), ISBN: 978-0-387-36642-5.

[26] Robert Brayton and Jason Cong. Electronic Design Automation – Past, Present and Future. NFS Workshop, Jul 2009.

[27] Dunlop, A.E. and Kernighan, B.W. , "A Procedure for Placement of Standard-Cell VLSI Circuits," Computer-Aided Design of Integrated Circuits and Systems, IEEE transactions on, January 1985.

[28] D. J. Dean, Thermal Design of Electronic Circuit Boards and Packages. Edinburgh, Scotland: Electrochemical Publications, 1985.

[29] Forum for Electronics, [online] Available:

[30] VLSI Low Power

[31] Physical design (electronics)

[32] Integrated_circuit

[33] Electronic_design_automation

[34] Computer-aided_design

[35] low-power-design-techniques

[36] Integrated_circuit_design