ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ανάπτυξη Μεθοδολογίας Δυναμικής Διαχείρισης Συχνότητας σε FPGAs μέσω Ελέγχου των Μονοπατιών Δεδομένων σε Πραγματικό Χρόνο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πέτρος Α. Σουσούρης

Επιβλέπων: Δημήτριος Ι. Σούντρης

Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ανάπτυξη Μεθοδολογίας Δυναμικής Διαχείρισης Συχνότητας σε FPGAs μέσω Ελέγχου των Μονοπατιών Δεδομένων σε Πραγματικό Χρόνο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πέτρος Α. Σουσούρης

Επιβλέπων: Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Φεβρουαρίου 2016

.....	.....	.....
Δημήτριος Ι. Σούντρης	Κιαμάλ Πεκμεστζή	Γεώργιος Οικονομάκος
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

……

Πέτρος Α. Σουσούρης

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

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

Σύντομη περίληψη

Τα δεδομένα μέσα σε ένα κύκλωμα σχεδόν πάντοτε απαιτείται να μετακινηθούν από το σημείο δημιουργίας τους μέσα στο κύκλωμα σε ένα άλλο σημείο, ώστε να συνεχιστεί η επεξεργασία τους ή να αποθηκευτούν για μελλοντική χρήση. Η μετακίνηση αυτή δεν γίνεται ακαριαία, αλλά απαιτεί ένα χρονικό διάστημα το οποίο εισάγει πολλούς περιορισμούς που έχουν να κάνουν με τον χρονισμό και την απόδοση του κυκλώματος. Συγκεκριμένα, η μέγιστη επιτρεπόμενη συχνότητα λειτουργίας ενός κυκλώματος εξαρτάται άμεσα από την μέγιστη καθυστέρηση που συναντάται κατά την μετακίνηση των δεδομένων.

Στην συγκεκριμένη διπλωματική εργασία γίνεται προσπάθεια για την ανάπτυξη εργαλείων λογισμικού που είναι απαραίτητα για την μελέτη και ανάλυση των μονοπατιών δεδομένων ενός κυκλώματος. Σκοπός είναι μέσα από την συγκεκριμένη προσέγγιση η ανάπτυξη ενός αυτοματοποιημένου πλαισίου (framework) που θα υποστηρίζει ανάπτυξη τεχνικών για την βελτίωση της απόδοσης των υποεξέταση κυκλωμάτων. Σε αυτήν την κατεύθυνση η αξιολόγηση των κρίσιμων καθυστερήσεων διάδοσης των μονοπατιών δεδομένων και των αλληλεπιδράσεων μεταξύ τους αποτελούν κομβικό σημείο.

Κατά συνέπεια, η παρούσα διπλωματική κινείται πάνω σε δύο άξονες. Πρώτον, η συστηματική διατύπωση των σχεδιαστικών κατευθύνσεων που χρειάζεται να ακολουθήσει ένας σχεδιαστής συστημάτων προκειμένου να βελτιώσει την απόδοση του κυκλώματος που κατασκευάζει. Η αναλυτική περιγραφή θα βοηθήσει τον σχεδιαστή να αποφύγει τεχνικά και σχεδιαστικά προβλήματα που συναντώνται κατά τη διάρκεια μιας διαδικασίας ανάπτυξης ενός ολοκληρωμένου. Δεύτερον, η ανάπτυξη εργαλείων λογισμικού, που σε συνεργασία με εμπορικά εργαλεία της εταιρείας Xilinx, θα βοηθήσουν τον σχεδιαστή να επιτύχει καλύτερη απόδοση στο σύστημα και να επιταχύνουν την διαδικασία της ανάπτυξης και της επαλήθευσης του κυκλώματος. Η βελτίωση που επιτυγχάνεται στην απόδοση συγκρίνεται με το αρχικό κύκλωμα.

Λέξεις κλειδιά: κρίσιμο μονοπάτι, μονοπάτια δεδομένων, μέγιστη καθυστέρηση, συχνότητα λειτουργίας, διαχείριση ρολογιών, Planahead

Abstract

Data created in a circuit is often needed to be transmitted to a part of the design other than their creation place in order to be further processed or stored for future utilization. These transmissions cannot take place in zero time, but a short amount of time is required, which results in many timing constrains regarding the timing and the performance of the circuit. In particular, the maximum allowed frequency for operating a circuit highly depends on the maximum delay met during data transmissions.

In the context of the current diploma thesis it has been developed software tools required for studying and analyzing the data paths of a circuit. This methodology aims to improve the performance of the under examination circuit. That is achieved by analyzing the data path propagation delays as well as their interactions. All the above, help the circuit to apply design improvements that optimize its performance

To conclude, this diploma thesis has two main goals. First, explicitly define design instructions which will guide a system engineer to improve the performance of the targeted integrated circuit. The detailed directions will help the designer to avoid some technical issues that occur during the design process. Second, the development of software framework, in collaboration with commercial EDA tools by Xilinx, will contribute to accomplish even better system performance and accelerate the development and verification of the circuit. The performance improvement is compared to the original circuit.

Key words: data paths, propagation delay, operation frequency, clock management, Planahead.

Ευχαριστίες

Η παρούσα διπλωματική εργασία είναι αποτέλεσμα της συνεργασίας μου με το Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων από τον Απρίλιο του 2014 μέχρι και το Φεβρουάριο του 2016.

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κ. Δημήτριο Σούντρη για την εμπιστοσύνη που μου έδειξε στην ανάθεση της συγκεκριμένης εργασίας, τον διδάκτορα κ. Νικόλαο Ζομπάκη για την καθοδήγηση, τις εύστοχες παρατηρήσεις καθώς και την άψογη συνεργασία που είχαμε καθ' όλη τη διάρκεια της εκπόνησης της παρούσας εργασίας. Επίσης, ευχαριστώ θερμά τους υποψήφιους διδάκτορες και συνεργάτες του εργαστηρίου και ιδιαιτέρως τους κ. Κωνσταντίνο Μαραγκό και Γεώργιο Λεντάρη για την πολύτιμη βοήθεια τους στα προβλήματα που ανέκυψαν κατά τη διάρκεια της έρευνας.

Τέλος, θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου για την υποστήριξη που μου προσέφερε και ιδιαίτερα τις τελευταίες μέρες καθώς και τους φίλους μου, Γιάννη, Αλέξανδρο, Σπύρο, Λεωνίδα, Γιάννη και Γιώργο, για το ενδιαφέρον και την υποστήριξη που έδειχναν κατά τη διάρκεια της εργασίας. Χωρίς εκείνους η παρούσα εργασία δεν θα είχε ολοκληρωθεί επιτυχώς.

1. Εισαγωγή

1.1 Ενσωματωμένα Συστήματα

Ένας απλός ορισμός του ενσωματωμένου συστήματος είναι οποιαδήποτε συσκευή η οποία περιλαμβάνει έναν υπολογιστή που είναι αφοσιωμένος σε μία συγκεκριμένη λειτουργία και δεν είναι γενικού σκοπού. Συχνά, το σύστημα πρέπει να ανταποκρίνεται σε υπολογιστικούς περιορισμούς πραγματικού χρόνου (real time computing constrains) και είναι *ενσωματωμένο* ως μέρος μιας ολοκληρωμένης συσκευής που πολλές φορές περιλαμβάνει υλικό (hardware) και μηχανικά μέρη.

Τα ενσωματωμένα συστήματα ποικίλουν από φορητές συσκευές όπως ψηφιακά ρολόγια και συσκευές μουσικής, έως εφαρμογές μεγάλης κλίμακας, όπως φανάρια και ελεγκτές εργοστασίων, και συστήματα ιδιαίτερα μεγάλης πολυπλοκότητας όπως είναι τα αυτοκίνητα. Η πολυπλοκότητα των ενσωματωμένων συστημάτων μπορεί να είναι μικρή, όπως σε έναν απλό μικροελεγκτή (micro controller), έως υψηλή σε συστήματα με πολλές μονάδες, περιφερειακές συσκευές ή συσκευές διαχείρισης δικτύων. Στην εικόνα 1 είναι ορατά διάφορα σύγχρονα ενσωματωμένα συστήματα διαφορετικής πολυπλοκότητας σχεδίασης από πολλά και διαφορετικά πεδία εφαρμογών.

1: Σύγχρονα Ενσωματωμένα Συστήματα.
Πηγή: es.informatik.uni-kl.de

Τα σύγχρονα συστήματα συχνά βασίζονται σε μικροελεγκτές, δηλαδή επεξεργαστές με ενσωματωμένη μνήμη ή άλλες περιφερειακές συσκευές, αλλά κανονικοί μικροεπεξεργαστές συναντώνται ακόμα ειδικά σε πολύπλοκα συστήματα. Οι επεξεργαστές μπορούν να είναι γενικού σκοπού είτε ειδικά σχεδιασμένοι (custom designed) για μία πολύ συγκεκριμένη εφαρμογή. Ένα χαρακτηριστικό παράδειγμα εξειδικευμένου επεξεργαστή είναι ο επεξεργαστής ψηφιακού σήματος (digital signal processor - DSP).

Για ποιον όμως λόγο χρησιμοποιούνται μικροεπεξεργαστές; Στην ερώτηση αυτή, υπάρχουν δύο απαντήσεις:

- Οι μικροεπεξεργαστές είναι ένας πολύ αποδοτικός τρόπος υλοποίησης ψηφιακών συστημάτων, καθώς προσφέρουν την δυνατότητα επαναχρησιμοποίησης της σχεδίασης του υλικού απλά με μία αλλαγή λογισμικού. Αυτό είναι ιδιαίτερα σημαντικό, καθώς η σχεδίαση ολοκληρωμένων κυκλωμάτων παραμένει μία ακριβή και χρονοβόρα διαδικασία.

- Οι μικροεπεξεργαστές καθιστούν ευκολότερη την σχεδίαση οικογενειών προϊόντων τα οποία μπορούν να κατασκευαστούν για να παρέχουν διαφορετικά σύνολα χαρακτηριστικών σε διαφορετικές τιμές και μπορούν να επεκταθούν για να παρέχουν νέα χαρακτηριστικά, ώστε να συμβαδίζουν με τις ραγδαία μεταβαλλόμενες αγορές.

1.2 Ενσωματωμένη υπολογιστική – προκλήσεις

Η ενσωματωμένη υπολογιστική (embedded computing) είναι από πολλές απόψεις περισσότερο απαιτητική από τα προγράμματα που γράφονται για προσωπικούς υπολογιστές. Η λειτουργικότητα είναι σημαντική τόσο στην υπολογιστική γενικού σκοπού όσο και την ενσωματωμένη υπολογιστική, αλλά οι ενσωματωμένες εφαρμογές πρέπει να ικανοποιούν πολλούς επιπλέον περιορισμούς.

- *Πολύπλοκοι αλγόριθμοι:* Οι λειτουργίες που εκτελούνται από τον μικροεπεξεργαστή μπορεί να είναι ιδιαίτερα σύνθετες (για παράδειγμα, έλεγχος της ροής καύσιμου στο αυτοκίνητο)
- *Διασύνδεση με τον χρήστη:* Οι μικροεπεξεργαστές χρησιμοποιούνται συχνά για τον έλεγχο πολύπλοκων διασυνδέσεων με τον χρήστη οι οποίες μπορούν να περιλαμβάνουν πολλά μενού και επιλογές (για παράδειγμα, σε ένα σύστημα εντοπισμού θέσης (global positioning system - GPS)

Για να γίνουν τα πράγματα ακόμα δυσκολότερα, πολλές λειτουργίες των ενσωματωμένων συστημάτων πρέπει να πραγματοποιούνται μέσα σε συγκεκριμένες προθεσμίες (deadlines).

- *Πραγματικός χρόνος (real time):* Πολλά ενσωματωμένα υπολογιστικά συστήματα πρέπει να λειτουργούν σε πραγματικό χρόνο. Αν τα δεδομένα δεν είναι έτοιμα μέχρι μια συγκεκριμένη προθεσμία, το σύστημα κινδυνεύει με κατάρρευση. Η μη τήρηση των περιορισμών χρόνου μπορεί να δημιουργήσει δυσαρεστημένους πελάτες ή να κοστίσει ακόμα και ανθρώπινες ζωές.
- *Λειτουργίες πολλαπλών ρυθμών (multirate):* Οι λειτουργίες των ενσωματωμένων συστημάτων όχι μόνο πρέπει να ανταποκρίνονται σε συγκεκριμένες προθεσμίες, αλλά πιθανόν πολλές λειτουργίες πραγματικού χρόνου μπορεί να εξελίσσονται παράλληλα. Είναι πιθανό κάποιες λειτουργίες να εκτελούνται με αργό ρυθμό και άλλες με γρήγορο. Οι εφαρμογές πολυμέσων (multimedia) είναι το κύριο παράδειγμα συμπεριφοράς πολλαπλών ρυθμών, καθώς τα τμήματα ήχου και εικόνας εκτελούνται με πολύ διαφορετικούς ρυθμούς αλλά πρέπει να παραμένουν συγχρονισμένα.
- *Κόστος κατασκευής:* Το συνολικό κόστος κατασκευής ενός συστήματος είναι πολύ σημαντικό σε πολλές εφαρμογές και προσδιορίζεται από πολλούς παράγοντες, όπως ο τύπος του επεξεργαστή, η ποσότητα της μνήμης και το πλήθος των εξωτερικών συσκευών.
- *Ισχύς (power):* Η κατανάλωση ισχύος επηρεάζει την διάρκεια ζωής της μπαταρίας των φορητών συστημάτων, που σε πολλές εφαρμογές είναι κρίσιμη, αλλά και την παραγωγή θερμότητας που μπορεί να οδηγήσει σε προσωρινή αδυναμία χρήσης του συστήματος.

- *Περιορισμένοι πόροι συστήματος:* Σε αντίθεση με τους προσωπικούς υπολογιστές, τα περισσότερα ενσωματωμένα συστήματα διαθέτουν περιορισμένους πόρους προς αξιοποίηση (για παράδειγμα, τροφοδοσία από μπαταρία, περιορισμένη ποσότητα κύριας μνήμης, λίγες ή καθόλου συσκευές εισόδου/εξόδου). Επομένως, είναι απαραίτητη η προσεκτική αξιοποίηση τους, ώστε η εφαρμογή που θα τρέξει στο συγκεκριμένο σύστημα να μπορεί να λειτουργεί σωστά.

Οι εξωτερικοί περιορισμοί είναι μια σημαντική πηγή δυσκολίας στην σχεδίαση ενσωματωμένων συστημάτων. Κατά την σχεδίαση, τους πρέπει να ληφθούν υπόψη τα παρακάτω σημαντικά προβλήματα.

Πόσο υλικό χρειάζεται; Υπάρχει τρόπος για σημαντικό έλεγχο της ποσότητας της υπολογιστικής ισχύος που εφαρμόζεται στο πρόβλημα μέσω της επιλογής του τύπου του μικροεπεξεργαστή, την ποσότητα της μνήμης, τις συσκευές εισόδου και εξόδου και πολλά άλλα. Εφόσον πρέπει συχνά να ικανοποιούνται τέτοιοι περιορισμοί απόδοσης και κόστους κατασκευής, η επιλογή του υλικού είναι σημαντική. Αν το υλικό είναι λίγο, το σύστημα δεν θα μπορεί να ανταποκριθεί στις προθεσμίες. Αν το υλικό είναι υπερβολικό, το κόστος του συστήματος αυξάνεται χωρίς ανάλογη βελτίωση της απόδοσης.

Πως ικανοποιούνται οι προθεσμίες; Ο ωμός τρόπος ικανοποίησης μίας προθεσμίας είναι η επιτάχυνση του υλικού, ώστε το πρόγραμμα να εκτελείται γρηγορότερα. Η επιλογή αυτή όμως αυξάνει το κόστος του συστήματος, όπως αναφέρθηκε. Είναι επίσης πιθανό η αύξηση του χρονισμού του επεξεργαστή να μην βελτιώσει τον χρόνο εκτέλεσης, εφόσον η ταχύτητα του προγράμματος μπορεί να περιορίζεται από το σύστημα μνήμης.

Πως ελαχιστοποιείται η κατανάλωση ισχύος; Σε όλα τα συστήματα, η κατανάλωση ισχύος είναι κρίσιμο ζήτημα. Απαιτείται προσεκτική σχεδίαση για την επιβράδυνση μη κρίσιμων τμημάτων του συστήματος για τον περιορισμό της κατανάλωσης ενώ ικανοποιούνται ακόμη οι απαραίτητοι στόχοι απόδοσης.

Σχεδίαση με δυνατότητα αναβάθμισης. Η πλατφόρμα υλικού μπορεί να χρησιμοποιηθεί για αρκετές γενιές προϊόντων με ελάχιστες ή καθόλου αλλαγές. Ωστόσο, είναι επιθυμητή η προσθήκη δυνατοτήτων μέσω του λογισμικού. Είναι επομένως σημαντική η σωστή σχεδίαση του υλικού ώστε να προβλεφθεί η απόδοση λογισμικού που ακόμα δεν έχει σχεδιαστεί.

Αξιοπιστία. Η αξιοπιστία είναι σημαντική κατά την δημιουργία προϊόντων αλλά και σε ορισμένες εφαρμογές, όπως τα κρίσιμα από πλευράς ασφάλειας συστήματα (safety critical systems).

1.3 FPGA - ιστορική εξέλιξη

Το FPGA (Field Programmable Gate Array - συστοιχία επιτόπια προγραμματιζόμενων πυλών) είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL και πολλά άλλα. Μερικά FPGA ενσωματώνουν επίσης αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο πάνω σε ένα τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές

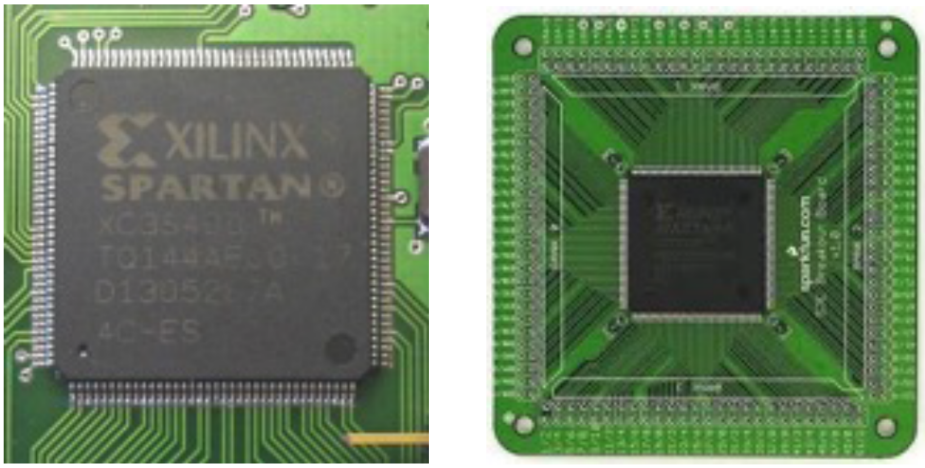
λειτουργίες και διασυνδέονται μεταξύ τους ώστε να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία.

Ο κώδικας με τον οποίο προγραμματίζεται το FPGA γράφεται σε κάποια γλώσσα περιγραφής υλικού όπως η VHDL και η Verilog. Έχει παρόμοιο πεδίο εφαρμογών με άλλα προγραμματιζόμενα ολοκληρωμένα ψηφιακά συστήματα όπως τα PLD και τα ASIC. Όμως το FPGA διαθέτει κάποια ιδιαίτερα χαρακτηριστικά που περιγράφονται παρακάτω:

- Το FPGA χάνει τον προγραμματισμό του κάθε φορά που χάνει την τάση τροφοδοσίας του. Επομένως, απαιτεί εξωτερικό μικροεπεξεργαστή ή μνήμη με μόνιμη συγκράτηση δεδομένων (non volatile memory) από τα οποία θα προγραμματίζεται, κάθε φορά που επανέρχεται η τάση τροφοδοσίας.
- Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του μικροεπεξεργαστή ή τα δεδομένα της μνήμης που το ελέγχει.
- Δεν υπάρχει κάποιο όριο στον αριθμό των φορών που μπορεί να προγραμματιστεί.
- Η κατανάλωση ισχύος είναι σημαντικά αυξημένη σε σχέση με τα ASIC.

Το FPGA είναι ιδιαίτερα κατάλληλο εκεί που οι παράμετροι λειτουργίας πρέπει να αλλάζουν συχνά ή σε μικρές ποσότητες παραγωγής, ενώ το ASIC, λόγω μαζικής παραγωγής, είναι φθηνότερο εκεί που απαιτούνται μεγάλες ποσότητες και η επιθυμητή λειτουργία είναι αυστηρά προκαθορισμένη, χωρίς σφάλματα (τα ASIC δεν μπορούν να προγραμματιστούν ξανά).

Βασική δομική μονάδα του FPGA είναι το λογικό μπλοκ, με την χρήση του οποίου υλοποιούνται οι λογικές συναρτήσεις που εκφράζουν τις λειτουργίες ενός ψηφιακού κυκλώματος. Ανάλογα με το μέγεθος του κυκλώματος, πολλά λογικά μπλοκ συνδέονται για να υλοποιήσουν το πλήθος των απαραίτητων λογικών συναρτήσεων. Στις εικόνες 2 και 3 φαίνονται μερικά FPGAs.



2: FPGA πάνω σε τυπωμένα κυκλώματα
Πηγή: en.wikipedia.com, codehackcreate.com

Η βιομηχανία των FPGA προέκυψε από τις προγραμματιζόμενες μνήμες μόνο ανάγνωσης (programmable read only memory - PROM) και τις προγραμματιζόμενες λογικές συσκευές (programmable logic device - PLD). Και οι δύο προηγούμενες

συσκευές έχουν την δυνατότητα προγραμματισμού σε ομάδες (banches). Ωστόσο, ο προγραμματισμός τους στηρίζονταν σε καλωδιωμένη λογική ανάμεσα στις πύλες.

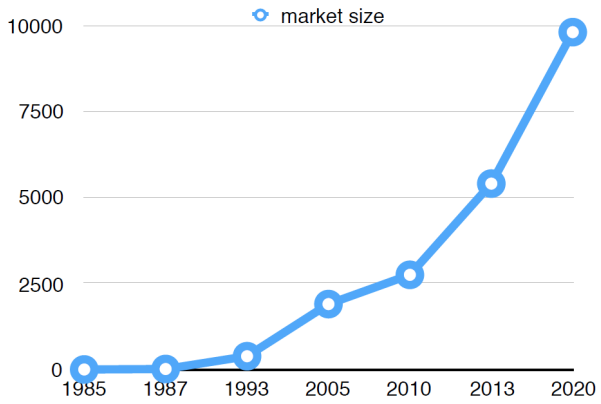
Η δεκαετία του 1990 ήταν εκρηκτική για τα FPGA τόσο από άποψη πολυπλοκότητας της σχεδίασης τους όσο και όγκου παραγωγής. Την ίδια περίοδο τα FPGA χρησιμοποιούνταν σε τηλεπικοινωνιακές εφαρμογές και στα δίκτυα. Προς το τέλος της δεκαετίας, τα FPGA βρήκαν τον δρόμο τους σε πιο εμπορικές εφαρμογές, την αυτοκινητοβιομηχανία και βιομηχανικές εφαρμογές.

Μία πρόσφατη τάση είναι η υβριδική αρχιτεκτονική, δηλαδή ο συνδυασμός των λογικών μπλοκ των παραδοσιακών FPGA με ενσωματωμένους επεξεργαστές και τα σχετικά περιφερειακά για να σχηματίσουν ένα ολοκληρωμένο *σύστημα πάνω σε ένα προγραμματιζόμενο chip* (system on a programmable chip). Η συγκεκριμένη αρχιτεκτονική περιορίζει την κατανάλωση ισχύος, δημιουργεί ένα μικρότερο σε μέγεθος σύστημα και σε μεγαλύτερη αξιοπιστία των συνδέσεων των δύο ξεχωριστών συστημάτων. (Wolf, 2008)

Στον πίνακα και το γράφημα που ακολουθούν υπάρχουν συγκεντρωμένα διάφορα στοιχεία που φανερώνουν την ιστορική εξέλιξη των FPGA (πηγή: en.wikipedia.com).

Year	1982	1987	1992	2000
Number of Gates	8.192	9.000	600.000	millions

1: Εξέλιξη του αριθμού πυλών στα FPGA



3: Εξέλιξη της αγοράς σε εκατομμύρια

1.4 Πλεονεκτήματα χρήσης των FPGA

Η ικανότητα των FPGAs να συνδυάζουν τα καλύτερα στοιχεία από τους δυο κόσμους (ASICs και συστήματα βασισμένα σε επεξεργαστή) οδήγησε στην ευρεία υιοθέτηση τους από όλες τις βιομηχανίες. Τα FPGA παρέχουν ταχύτητα και αξιοπιστία, ενώ δεν απαιτούν το μεγάλο προκαταβολικό κόστος που παρουσιάζει μια σχεδίαση βασισμένη στα ASIC. Το επαναπρογραμματιζόμενο πυρίτιο έχει τα ίδια πλεονεκτήματα και ευελιξία με ένα λογισμικό που τρέχει σε ένα σύστημα βασισμένο σε επεξεργαστή, αλλά δεν περιορίζεται από τον αριθμό των διαθέσιμων υπολογιστικών πυρήνων. Σε αντίθεση με τους επεξεργαστές, τα FPGA είναι παράλληλα από την κατασκευή τους κι έτσι διαφορετικές διεργασίες δεν χρειάζεται να ανταγωνίζονται για τους ίδιους πόρους. Κάθε ανεξάρτητη υπολογιστική διεργασία ανατίθεται σε ένα

διαφορετικό τμήμα του chip και μπορεί να λειτουργεί αυτόνομα χωρίς επιρροή από άλλες διεργασίες. Συνεπώς, η απόδοση ενός τμήματος μιας εφαρμογής δεν επηρεάζεται όταν περισσότερες διεργασίες προστεθούν στο σύστημα. (The Linley Group, 2009)

Τα κύρια πλεονεκτήματα της χρήσης των FPGA συνοψίζονται παρακάτω:

- **Απόδοση**. Εκμεταλλευόμενα την παραλληλία στο υλικό, τα FPGA υπερέχουν ως προς την υπολογιστική ισχύ των ψηφιακών επεξεργαστών σήματος (digital signal processors - DSP) εγκαταλείποντας την λογική της ακολουθιακής εκτέλεσης και επιτυγχάνοντας περισσότερα ανά κύκλο ρολογιού. Η BDTI, μια εταιρεία benchmarking, σε μελέτη της έδειξε ότι τα FPGA μπορούν να παραδώσουν πολλές φορές παραπάνω απόδοση ανά δολάριο σε μερικές εφαρμογές σε σχέση με ένα DSP (BDTI Industry Report, 2006). Ο έλεγχος εισόδων και εξόδων στο επίπεδο του υλικού παρέχει καλύτερους χρόνους απόκρισης και εξειδικευμένη λειτουργικότητα για να ικανοποιήσει τις ανάγκες μιας εφαρμογής.
- **Χρόνος στην αγορά** (time to market). Η τεχνολογία των FPGA προσφέρει ευελιξία και ικανότητες ταχείας προτυποποίησης. Μια ιδέα και μια σχεδίαση μπορούν να δοκιμαστούν στο υλικό χωρίς να μεσολαβήσει η χρονοβόρα διαδικασία της κατασκευής ενός ASIC (Thompson, 2004). Οι συνέχειες αλλαγές και βελτιώσεις της σχεδίασης μπορούν να επιτευχθούν σε ώρες αντί για εβδομάδες. Στο εμπόριο υπάρχουν πολλές επιλογές υλικού με διαφορετικούς τύπους I/O ήδη συνδεδεμένες πάνω σε ένα επαναπρογραμματιζόμενο chip. Επίσης, η συνεχώς αυξανόμενη διαθεσιμότητα εργαλείων λογισμικού υψηλού επιπέδου μειώνουν τον χρόνο εκμάθησης εισάγοντας πολλαπλά επίπεδα αφαίρεσης και προσφέροντας έτοιμες υλοποιήσεις για προηγμένο έλεγχο και επεξεργασία σήματος.
- **Κόστος**. Το προκαταβολικό κόστος σχεδίασης ενός ASIC ξεπερνά κατά πολύ τις αντίστοιχες λύσεις βασισμένες σε FPGA. Η μεγάλη αρχική επένδυση των ASIC μπορεί να δικαιολογηθεί από για κατασκευαστές που παράγουν και πουλούν μαζικά chip. Η φύση του επαναπρογραμματιζόμενου πυριτίου ελαττώνει το κόστος ανάπτυξης και την χρονοβόρα διαδικασία κατασκευής. Επειδή συχνά στην πράξη οι προδιαγραφές ενός συστήματος αλλάζουν με τον χρόνο, το κόστος των συνεχών αλλαγών σε σχεδιάσεις FPGA είναι αμεληταίο όταν συγκριθεί με το μεγάλο κόστος επανασχεδίασης ενός ASIC.
- **Αξιοπιστία**. Ενώ τα εργαλεία λογισμικού παρέχουν το προγραμματιστικό περιβάλλον, τα κυκλώματα των FPGA είναι μια υλοποίηση της εκτέλεσης του προγράμματος στο υλικό. Τα συστήματα βασισμένα σε επεξεργαστή συχνά παρέχουν πολλά επίπεδα αφαίρεσης για βοήθεια στην δρομολόγηση διεργασιών και τον διαμοιρασμό πόρων ανάμεσα σε διεργασίες. Το επίπεδο "οδηγός" ελέγχει τους πόρους του υλικού και το λειτουργικό σύστημα διαχειρίζεται την μνήμη και τον επεξεργαστή. Σε κάθε διαθέσιμο υπολογιστικό πυρήνα μόνο μία εντολή μπορεί να εκτελεστεί κάθε χρονική στιγμή και τα συστήματα βασισμένα σε επεξεργαστή κινδυνεύουν χρονικά κρίσιμες διεργασίες συνεχώς να διακόπτουν η μία την άλλη. Τα FPGA, τα οποία δεν διαθέτουν λειτουργικό σύστημα, ελαχιστοποιούν τους κινδύνους αξιοπιστίας με πραγματικά παράλληλη εκτέλεση εντολών και ντετερμινιστικό υλικό αφιερωμένο σε κάθε διεργασία που υπάρχει στο σύστημα.

- **Μακροχρόνια συντήρηση**. Όπως αναφέρθηκε προηγουμένως, τα FPGA είναι αναβαθμίσιμα και δεν απαιτούν το κόστος και τον χρόνο επαναδιαμόρφωσης όπως ένα ASIC. Τα ψηφιακά πρωτόκολλα επικοινωνιών, για παράδειγμα, έχουν προδιαγραφές που μπορεί να αλλάξουν με τον καιρό και οι διεπαφές που βασίζονται σε ASIC ενδεχομένως να προκαλέσουν προβλήματα συντήρησης και συμβατότητας. Με τις δυνατότητες επαναδιαμόρφωσης τα FPGA μπορούν να ανταπεξέλθουν σε μελλοντικές τροποποιήσεις που θα χρειαστούν. Καθώς ένα προιόν ή ένα σύστημα ωριμάζει, λειτουργικές ενισχύσεις μπορούν να γίνουν σε αυτό χωρίς τον χρόνο που απαιτείται για σχεδίαση υλικού από την αρχή. (National Instruments, 2012)

1.5 Δομή του FPGA

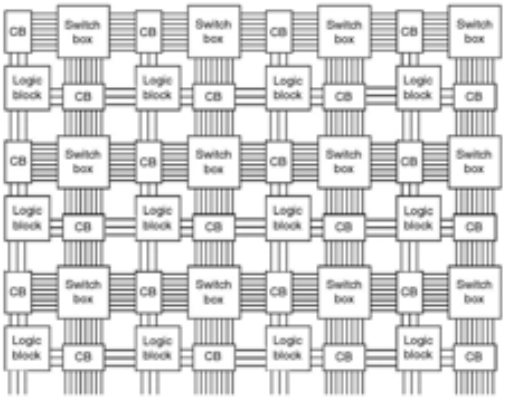
Τα FPGA αποτελούνται από τρία βασικά στοιχεία: logic boards, θύρες εισόδου και εξόδου και προγραμματιζόμενη δρομολόγηση. Ο τύπος της logic board που χρησιμοποιείται επηρεάζει την ταχύτητα και την επιφάνεια του FPGA. Ένας κοινός τύπος logic board που χρησιμοποιείται στα σύγχρονα FPGA βασίζεται στα lookup tables (LUT), τα οποία αποτελούνται από έναν N:1 πολυπλέκτη και μια μνήμη N bit. Όσον αφορά την ψηφιακή λογική, ένα LUT απλά απαριθμεί έναν πίνακα αλήθειας, δίνοντας την δυνατότητα στο FPGA να υλοποιεί περίπλοκη ψηφιακή λογική. (Brown & Rose)

LUT

Ένα LUT είναι ένας πίνακας που αντικαθιστά υπολογισμούς την ώρα της εκτέλεσης με μία πιο απλή και γρήγορη λειτουργία indexing. Αν και τα LUT έχουν επιλεγεί ως η κύρια υπολογιστική μονάδα στα εμπορικά FPGA, το μέγεθος τους σε κάθε logic board πρέπει να προσδιοριστεί προσεκτικά. Τα μεγάλα LUT μπορούν να χειριστούν πιο πολύπλοκους υπολογισμούς και συνεπώς να μειώσουν τις καθυστερήσεις πάνω στην καλωδίωση ανάμεσα στις διάφορες μονάδες. Ωστόσο, αυτό οδηγεί σε πιο αργές υλοποιήσεις των LUT εξαιτίας της χρήσης μεγαλύτερων πολυπλεκτών. Από την άλλη πλευρά, μικρότερα LUT έχουν ως αποτέλεσμα την χρησιμοποίηση μεγαλύτερου αριθμού logic blocks, κάτι που αυξάνει τις καθυστερήσεις καλωδίωσης στην σχεδίαση. Επιπλέον, υπάρχει μια μονάδα αποθήκευσης ενός bit που είναι ένα D flip flop. Ο πολυπλέκτης εξόδου επιλέγει ένα αποτέλεσμα είτε από την συνάρτηση που είναι υλοποιημένη μέσα στο LUT είτε από το bit που είναι αποθηκευμένο στο flip flop.

Διασύνδεση

Τα σύγχρονα FPGA είναι σχεδιασμένα χρησιμοποιώντας την αρχιτεκτονική "νησίδων". Σύμφωνα με αυτήν, οι δομικές μονάδες τοποθετούνται σε ένα δισδιάστατο πλέγμα και διασυνδέονται με ένα συγκεκριμένο μοτίβο. Αυτές οι δομικές μονάδες σχηματίζουν τις νησίδες οι οποίες επιπλέουν στον ωκεανό των διασυνδέσεων. Αυτή η αρχιτεκτονική επιτρέπει στους υπολογισμούς να πραγματοποιηθούν τοπικά στο FPGA, ενώ μεγαλύτεροι υπολογισμοί σπάνε σε κομμάτια και αντιστοιχίζονται σε φυσικά logic blocks μέσα στο πλέγμα.



4: Αρχιτεκτονική τύπου νησίδων με διασυνδέσεις block και switch boxes

Το κάθε block έχει πρόσβαση στους γείτονες του μέσω του block διασύνδεσης, το οποίο συνδέει τις εισόδους και εξόδους του λογικού block στους πόρους δρομολόγησης μέσω προγραμματιζόμενων διακοπτών ή πολυπλεκτών. Το block διασύνδεσης επιτρέπει στην είσοδο και την έξοδο του λογικού block να αποδοθούν σε οριζόντιες και κάθετες διαδρομές, βελτιώνοντας κατά πολύ την ευελεξία δρομολόγησης.

Κάθε διαμορφώμενο στοιχείο του FPGA απαιτεί ένα bit πληροφορίας για να διατηρήσει μία διαμόρφωση καθορισμένη από τον χρήστη. Για ένα FPGA βασιζόμενο σε LUT, αυτές οι προγραμματιζόμενες τοποθεσίες περιλαμβάνουν τα περιεχόμενα του λογικού block και την συνδεσιμότητα. Η διαμόρφωση επιτυγχάνεται μέσω προγραμματισμού των bits που συνδέονται με αυτές τις προγραμματιζόμενες τοποθεσίες, σύμφωνα με την είσοδο του χρήστη. Υπάρχουν πολλοί τρόποι για την αποθήκευση ενός bit δυαδικής πληροφορίας με την πιο δημοφιλή να είναι η SRAM, η antifuse και η flash μνήμη. (Kuon, Tessier, & Rose, 2008)

Η πιο ευρεία χρησιμοποιούμενη μέθοδος για την αποθήκευση πληροφορίας διαμόρφωσης στα εμπορικά FPGA είναι η πτητική στατική RAM, περισσότερο γνωστή ως SRAM. Αυτή η τεχνική έγινε δημοφιλής επειδή παρέχει δυνατότητες γρήγορης και απεριόριστης επαναδιαμόρφωσης σε μια ήδη γνωστή τεχνολογία. Μεινεκτήματα της SRAM είναι η υψηλή κατανάλωση ενέργειας και η πτητικότητα των δεδομένων. Συγκρινόμενη με άλλες τεχνολογίες μνήμης, ένα στοιχείο της SRAM είναι μεγαλύτερο (απαιτεί 6 έως 12 transistor) και παρουσιάζει σημαντική στατική κατανάλωση εξαιτίας ρευμάτων διαρροής. Ένα ακόμα σημαντικό μειονέκτημα είναι ότι η SRAM δεν διατηρεί τα δεδομένα της χωρίς ενέργεια, που σημαίνει ότι κατά την εκκίνηση το FPGA δεν έχει διαμόρφωση και πρέπει να προγραμματιστεί χρησιμοποιώντας λογική και αποθήκευση εκτός chip. Αυτό επιτυγχάνεται χρησιμοποιώντας μη πτητική μνήμη για διατήρηση της διαμόρφωσης και έναν μικροελεγκτή για να πραγματοποιήσει την διαδικασία του προγραμματισμού κατά την εκκίνηση του FPGA.

Αν και λιγότερο δημοφιλής, πολλές οικογένειες συσκευών χρησιμοποιούν μνήμη flash για να αποθηκεύσουν την πληροφορία διαμόρφωσης. Η μνήμη flash είναι διαφορετική από την SRAM επειδή είναι μη πτητική και μπορεί να εγγραφεί περιορισμένο αριθμό φορών. Η μη πτητικότητα της μνήμης flash σημαίνει ότι τα δεδομένα μπορούν να εγγραφούν και να παραμείνουν αποθηκευμένα ακόμα και χωρίς την παροχή ρεύματος. Σε αντίθεση με τα FPGA που βασίζονται σε SRAM, αυτά που βασίζονται σε μνήμη flash παραμένουν διαμορφωμένα από τον χρήστη και δεν χρειάζονται επιπλέον υλικό για να προγραμματιστούν κατά την εκκίνηση, που σημαίνει

ότι είναι έτοιμα να λειτουργήσουν αμέσως. Επιπλέον, ένα κύτταρο flash μνήμης κατασκευάζεται από λιγότερα transistors και συνεπώς έχει μικρότερες απώλειες λόγω ρευμάτων διαρροής. Ωστόσο, οι συγκεκριμένες μνήμες έχουν περιορισμένο κύκλο αναγνώσεων/εγγραφών και συχνά χαμηλότερες ταχύτητες εγγραφής συγκριτικά με τις SRAM. Ο αριθμός των κύκλων εγγραφής εξαρτάται από την τεχνολογία αλλά τυπικά κυμαίνεται σε μερικά εκατομμύρια φορές. Επιπρόσθετα, οι περισσότερες τεχνικές εγγραφής σε flash απαιτούν υψηλότερη τάση συγκριτικά με τα άλλα κυκλώματα. Επομένως, χρειάζονται βοηθητικά κυκλώματα εκτός chip ή δομές όπως αντλίες τάσης για να πραγματοποιήσουν εγγραφές.

Μια τρίτη προσέγγιση για προγραμματισμό είναι η τεχνολογία μνήμης antifuse. Όπως υποδηλώνει και το όνομα, πρόκειται για έναν μεταλλικό σύνδεσμο που συμπεριφέρεται το αντίθετο από μία ασφάλεια. Ο σύνδεσμος antifuse είναι κανονικά ανοιχτός (μη συνδεδεμένος). Μια προγραμματιστική διαδικασία που περιλαμβάνει είτε έναν προγραμματιστή υψηλού ρεύματος είτε μία ακτίνα laser λιώνουν τον σύνδεσμο για να σχηματιστεί μία ηλεκτρική σύνδεση σαν να υπήρχε καλώδιο ανάμεσα στις άκρες του antifuse. Παρουσιάζει αρκετά πλεονεκτήματα αλλά δεν είναι επαναπρογραμματίσιμο. Μόλις ένας σύνδεσμος λιώσει, έχει υποστεί έναν μη αντιστρεπτό μετασχηματισμό. Τα FPGA που βασίζονται σε αυτή την τεχνολογία θεωρούνται προγραμματιζόμενα μόνο μία φορά. Το γεγονός αυτό περιορίζει την ευελιξία και καθιστά ακατάλληλη την τεχνολογία για προτυποποίηση. Ωστόσο, η χρήση της τεχνολογίας συνοδεύεται από μερικά πλεονεκτήματα. Ο σύνδεσμος έχει πολύ μικρό μέγεθος συγκριτικά με τα κύτταρα των άλλων τεχνολογιών που αποτελούνται από αρκετά transistor. Αυτό οδηγεί σε μικρές καθυστερήσεις διάδοσης και μηδενική στατική κατανάλωση ενέργειας επειδή δεν υπάρχουν πλέον ρεύματα διαφυγής. Επίσης οι σύνδεσμοι είναι ιδιαίτερα ανθεκτικοί στην ακτινοβολία, γεγονός που καθιστά την τεχνολογία κατάλληλη για στρατιωτικές και διαστημικές εφαρμογές.

1.6 Έτοιμες βιβλιοθήκες

Πολλά εμπορικά εργαλεία παρέχουν ένα γενικό σετ από τμήματα FPGA, δηλαδή συμβολικές αναπαραστάσεις έτοιμων blocks λειτουργιών που ο χρήστης επιθυμεί να ενσωματώσει στο δικό του FPGA design. Αυτά τα τμήματα παρουσιάζονται στον χρήστη των εργαλείων ως σύμβολα έτοιμα προς χρήση πάνω σε μια πλακέτα.

Τα τμήματα πριν την σύνθεση (pre synthesized components) παρέχονται ως ενότητες κώδικα αντικειμένων (object code) χωρίς να είναι απαραίτητο να αποκαλύψουν τον πηγαίο κώδικα επιπέδου RTL ή netlist. Το σύστημα περιλαμβάνει πολλαπλές βιβλιοθήκες παρέχοντας ένα ολοκληρωμένο σετ τμημάτων προ σύνθεσης, με εύρος από απλές λογικές πύλες μέχρι λειτουργίες υλικού υψηλού επιπέδου, όπως πολλαπλασιαστές και διαμορφωτές παλμών ή ακόμα και επεξεργαστές και περιφερειακά επικοινωνίας.

Αυτά τα έτοιμα τμήματα μπορούν να εισαχθούν σε σχέδια από τον χρήστη του εργαλείου και έπειτα ολόκληρο το design να μεταφερθεί σε μία κατάλληλη φυσική συσκευή. Τα πλεονεκτήματα χρήσης έτοιμων τμημάτων είναι πολλά. Μερικά αναφέρονται ενδεικτικά παρακάτω:

- Μείωση του χρόνου που απαιτείται για την ολοκλήρωση του design καθώς πολλά συχνά χρησιμοποιούμενα τμήματα παρέχονται έτοιμα.

- Ευκολότερος έλεγχος της σωστής λειτουργίας του σχεδίου, αφού τα έτοιμα τμήματα παρέχουν εγγυημένα σωστή λειτουργία.
- Δυνατότητα επαναχρησιμοποίησης τμημάτων πολλές φορές.
- Αποδοτικότερα κυκλώματα, καθώς τα προσφερόμενα τμήματα είναι βελτιστοποιημένα για την λειτουργία που προορίζονται.

1.7 Περιγραφή Προβλήματος και Πρόταση Λύσης

Η καθυστέρηση του κρίσιμου μονοπατιού αναγκάζει το κύκλωμα να λειτουργεί σε μία συγκεκριμένη συχνότητα η οποία δεν παραβιάζει τους περιορισμούς του. Αυτή η συχνότητα υπολογίζεται ως ο αντίστροφος αριθμός της καθυστέρησης που έχει το κρίσιμο μονοπάτι. Αρκετά συχνά η τελική συχνότητα λειτουργίας είναι ακόμα χαμηλότερη για να διασφαλιστεί ότι το κρίσιμο μονοπάτι δεν παραβιάζεται και ότι όλες οι είσοδοι και οι έξοδοι των επιμέρους τμημάτων είναι έγκυρες και σταθερές. Επειδή το κύκλωμα λειτουργεί σε ένα σενάριο "worst case", είναι βέβαιο ότι όλα τα υπόλοιπα μονοπάτια λειτουργούν ομαλά και οι χρονικοί περιορισμοί τους ικανοποιούνται. Αυτή η συχνότητα είναι η υψηλότερη δυνατή που το κύκλωμα μπορεί να λειτουργήσει χωρίς πρόβλημα στα μονοπάτια δεδομένων του.

Ωστόσο, το κύκλωμα συνήθως έχει τη δυνατότητα να λειτουργήσει ακόμα πιο γρήγορα. Αυτό επιτυγχάνεται μέσα από την ανάλυση των μονοπατιών του και της λειτουργικότητας τους. Ανάλογα με την λειτουργικότητα τους, είναι δυνατόν να αυξηθεί η συχνότητα λειτουργίας του κυκλώματος χωρίς να υπάρχουν εμφανείς επιπτώσεις είτε λόγω εκμετάλλευσης της διαφοράς θεωρητικής και πραγματικής καθυστέρησης σε μία πλατφόρμα είτε λόγω χαμηλής επίπτωσης στο τελικό λειτουργικό αποτέλεσμα. Αυτή είναι η κύρια (και απλοποιημένη) ιδέα στην οποία στηρίζεται η δυναμική κλιμάκωση συχνότητας.

Για κάθε σταυροδρόμι που συναντάται, η λογική συνάρτηση που υλοποιείται στο σημείο της διασταύρωσης (το σημείο αυτό είναι συνήθως ένα LUT) μελετάται περαιτέρω. Μέσω της ανάλυσης των διασταυρώσεων, μπορούν να καθοριστεί η λειτουργικότητα τους και κατ' επέκταση η συχνότητα λειτουργίας ολόκληρου του κυκλώματος. Ο τρόπος με τον οποίο αυτός ο έλεγχος επιτυγχάνεται είναι αμετάβλητος καθώς είναι αποθηκευμένος σε ένα lookup table που δεν μεταβάλλεται κατά τον χρόνο εκτέλεσης. Με τον τρόπο αυτόν επιτυγχάνεται η διαχείριση της διασταύρωσης χωρίς να αλλάξει η λογική του κυκλώματος.

1.8 Εργαλεία που χρησιμοποιήθηκαν

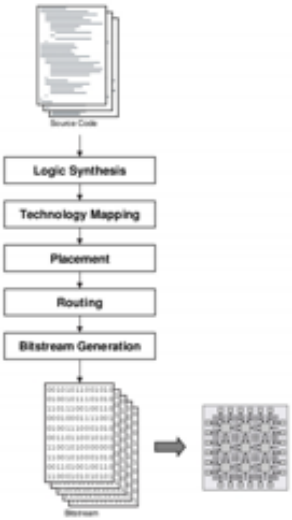
Στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το εργαλείο Planahead της εταιρείας Xilinx. Επιτρέπει στον χρήστη να συνθέσει το σχέδιο του, να πραγματοποιήσει ανάλυση του χρονισμού, να ελέγξει την απόκριση του κυκλώματος σε διάφορες εισόδους και καταστάσεις και να προγραμματίσει το design του πάνω σε μία πλακέτα για πραγματική λειτουργία. Με το εργαλείο αυτό, είναι δυνατή η μελέτη των αποτελεσμάτων υλοποίησης πάνω στο FPGA (implementation) και του χρονισμού (timing) με στόχο την ανάλυση της κρίσιμης λογικής. Επιπλέον, βοηθά στην βελτίωση της απόδοσης του design του χρήστη μέσω floor planning, τροποποίησης των περιορισμών και πολλών διαφορετικών ρυθμίσεων σε επίπεδο σύνθεσης και υλοποίησης.

Κάθε design που υλοποιείται με την βοήθεια του Planahead περνάει από τα στάδια του placement, του mapping και του routing. Ακολουθεί μία σύντομη ανάλυση για κάθε ένα από αυτά τα στάδια:

- **Placement**: είναι ένα απαραίτητο βήμα στην ηλεκτρονική σχεδίαση και αναφέρεται στην ανάθεση τοποθεσιών με ακρίβεια διαφόρων τμημάτων του κυκλώματος μέσα στην περιοχή του chip. Ένα κατώτερης ποιότητας placement όχι μόνο επηρεάζει την απόδοση του chip, αλλά πιθανόν να οδηγήσει στην αδυναμία κατασκευής του παράγοντας καλωδιώσεις μεγάλου μήκους που ξεπερνούν τους διαθέσιμους πόρους για routing. Κατά συνέπεια, η διαδικασία αυτή πρέπει να κάνει τις αναθέσεις ενώπαράλληλα βελτιστοποιεί ένα πλήθος στόχων, ώστε να επιτευχθούν οι προδιαγραφές απόδοσης. Χαρακτηριστικοί στόχοι του placement περιλαμβάνουν:
  - *Συνολικό μήκος καλωδιώσεων:* η ελαχιστοποίηση του μήκους των καλωδιώσεων αποτελεί τον πρωταρχικό στόχο του λογισμικού που υλοποιεί το placement. Αυτό όχι μόνο συντελεί στην μείωση του μεγέθους του chip, αλλά ταυτόχρονα μειώνει την κατανάλωση ισχύος και την καθυστέρηση διάδοσης των σημάτων που είναι ανάλογη του μήκους καλωδίωσης.
  - *Χρονισμός:* ο κύκλος ρολογιού ενός chip καθορίζεται από την καθυστέρηση του μακρύτερου μονοπατιού του, που συχνά αναφέρεται ως κρίσιμο μονοπάτι. Έχοντας καθορισμένες προδιαγραφές απόδοσης, το λογισμικό πρέπει να είναι σίγουρο ότι δεν υπάρχει μονοπάτι που να ξεπερνά την μέγιστη καθορισμένη καθυστέρηση.
  - *Συμφόρηση:* Ενώ είναι απαραίτητο να μειωθεί το μήκος των καλωδιώσεων ώστε να επαρκούν οι πόροι του routing, είναι επίσης αναγκαίο οι πόροι αυτοί να ικανοποιούν προδιαγραφές τοπικότητας πάνω στο chip. Μια περιοχή με συμφόρηση ίσως οδηγήσει σε παρακάμψεις διαδρομών αυξάνοντας τις καλωδιώσεις.
  - *Ισχύς:* η μείωση της ισχύος συνήθως περιλαμβάνει την σωστή κατανομή των τμημάτων για την μείωση της κατανάλωσης και την εξομάλυνση της θερμοκρασίας του chip.
  - Ένας δευτερεύων στόχος του λογισμικού είναι η μείωση του χρόνου που απαιτεί για την ολοκλήρωση του το placement.
- **Mapping:** είναι μία μέθοδος με την οποία το design μπορεί να αντιστοιχιστεί στα φυσικά pins του FPGA στο οποίο θα προγραμματιστεί, δηλαδή πύλες ή διάφορα στοιχεία επιλέγονται από τις βιβλιοθήκες για να υλοποιήσουν τα κυκλώματα του design. Διαφορετικά, είναι το μέσο με το οποίο το design μπορεί να αλληλεπιδράσει με τον "έξω κόσμο". Χαρτογραφώντας εσωτερικά ψηφιακά σήματα σε pins κάποιας συσκευής, η λογική του design μπορεί να επικοινωνήσει με άλλα τμήματα του chip. Ως μέρος του mapping, καθορίζονται και αναλογικά χαρακτηριστικά στα pins, όπως IO standards, δυνάμεις οδήγησης (drive strengths) και slew rates. Σε επίπεδο λογισμικού, το mapping επιτυγχάνεται με χρήση διαμορφώσεων και αρχείων περιορισμών. Ένα FPGA design μπορεί να έχει πολλαπλές καθορισμένες διαμορφώσεις, με κάθε μία να περιέχει το αρχείο περιορισμών (χαρτογράφηση pins, περιορισμοί τοποθέτησης και δρομολόγησης, περιορισμοί ρολογιού και χρονισμού) που απαιτείται για να στοχεύσει σε υλοποίηση πάνω σε διαφορετικές φυσικές συσκευές.

- **Routing:** είναι μία διαδικασία που στηρίζεται στο placement, που καθορίζει την τοποθεσία κάθε ενεργού στοιχείου που χρησιμοποιείται από το κύκλωμα. Μετά το placement, το routing τοποθετεί καλώδια που απαιτούνται για την σύνδεση των τοποθετημένων εξαρτημάτων ενώ διατηρεί όλους τους κανόνες του design. Στο λογισμικό δίνονται κάποια προϋπάρχοντα πολύγωνα που αποτελούνται από pins και προαιρετικά κάποιες προϋπάρχουσες καλωδιώσεις. Κάθε ένα από αυτά τα πολύγωνα συσχετίζεται με ένα net, βάσει ονόματος ή ενός αριθμού. Η κύρια εργασία του router είναι να δημιουργήσει γεωμετρίες ώστε όλα τα pins του ίδιου net να είναι συνδεδεμένα, κανένα pin συσχετισμένο με άλλο net να μην συνδέεται και όλοι οι κανόνες του design να ισχύουν. Ένας router μπορεί να αποτύχει μην συνδέοντας δύο pins που έπρεπε να συνδεθούν (open), συνδέοντας δύο pins που δεν έπρεπε (short) ή παραβιάζοντας κάποιον κανόνα. Επιπλέον, για να συνδεθούν σωστά τα nets, οι routers πρέπει να τηρήσουν τον χρονισμό, να μην δημιουργήσουν προβλήματα crosstalk, να τηρήσουν τις απαιτήσεις πυκνότητας και πολλά άλλα. Από τα παραπάνω είναι εμφανές ότι το routing είναι μία ιδιαίτερα δύσκολη διαδικασία.

  Σχεδόν κάθε πρόβλημα που σχετίζεται με το routing είναι δυσεπίλυτο. Το απλούστερο πρόβλημα δρομολόγησης, γνωστό ως δέντρο του Steiner, εύρεσης του συντομότερου δρόμου για ένα net χωρίς εμπόδια και κανόνες του design είναι NP-δύσκολο αν όλες οι γωνίες επιτρέπονται και NP-πλήρες αν μόνο οριζόντια και κάθετα καλώδια επιτρέπονται. Κατά συνέπεια, οι routers σπάνια προσπαθούν να βρουν μία βέλτιστη λύση. Αντίθετα, σχεδόν ολόκληρη η δρομολόγηση βασίζεται σε ευριστικές λύσεις που προσπαθούν να βρουν απλά μία ικανοποιητική λύση.



5: Διαδικασία κατασκευής κυκλώματος πάνω σε FPGA

Το **Planahead** θα χρησιμοποιηθεί επίσης για την σύνδεση των τριών μερών (κύριο κύκλωμα, κύκλωμα Selector που παρακολουθεί τα σήματα ελέγχου και το digital clock manager που επιλέγει την κατάλληλη συχνότητα λειτουργίας) που συνθέτουν το τελικό κύκλωμα.

Το δεύτερο εργαλείο που θα χρησιμοποιηθεί είναι μία εφαρμογή που αναπτύχθηκε στο πλαίσιο της διπλωματικής και ονομάζεται **Planahead Expander**. Αναλύει τα αποτελέσματα της χρονικής ανάλυσης του κύριου κυκλώματος (τα αποτελέσματα αυτά παρέχονται από το Planahead) και βγάζει σαν έξοδο τα σήματα

ελέγχου καθώς και τις αντίστοιχες συχνότητες λειτουργίας. Η εφαρμογή είναι γραμμένη σε Java για να μπορεί να εκτελεστεί σε οποιοδήποτε λειτουργικό σύστημα. Μαζί με τον Expander έρχεται μία ακόμα εφαρμογή γραμμένη σε Java που ονομάζεται **Generator**. Η εφαρμογή αυτή διαβάζει το αρχείο που δημιουργήθηκε από τον Expander και δημιουργεί ένα αρχείο με κώδικα VHDL που υλοποιεί το κύκλωμα που παρακολουθεί τα σήματα ελέγχου όπως προσδιορίστηκαν από το προηγούμενο εργαλείο.

Το **ISE** της XIlinx χρησιμοποιείται για την κατασκευή της μονάδας digital clock manager. Η μονάδα αυτή μετατρέπει μία συχνότητα εισόδου σε μέχρι έξι (εξαρτάται από τον τύπο του FPGA που χρησιμοποιείται) συχνότητες εξόδου καθορισμένες από τον χρήστη. Στην παρούσα εργασία, μόνο το IP Core Generator του ISE χρησιμοποιείται. Όλες οι άλλες λειτουργίες επιτυγχάνονται μέσα από το Planahead. Η κατασκευή του digital clock manager είναι εύκολη και πραγματοποιείται μέσα από γραφικό περιβάλλον.

Το **FPGA Editor** από την Xilinx χρησιμοποιείται (εφόσον χρειάζεται) για να συνδέσει τα εσωτερικά σήματα του κύριου κυκλώματος με τις εισόδους του κυκλώματος Selector. Αν δεν υπαρχουν εσωτερικά σήματα, σύνδεση μπορεί να πραγματοποιηθεί μέσω κώδικα VHDL και το FPGA Editor δεν χρειάζεται. Επιπλέον, αυτή η εφαρμογή μπορεί να χρησιμοποιηθεί για πληροφορίες σχετικές με την καθυστέρηση σε συγκεκριμένα δρομολογημένα καλώδια που θα βοηθήσουν τον χρήστη να επιταχύνει ακόμα περισσότερο το κύριο κύκλωμα.

Το **Isim** από την Xilinx είναι ένας προσομοιωτής που χρησιμοποιείται για να ελεγχεί η λειτουργία του νέου και βελτιωμένου κυκλώματος. Ο προσομοιωτής δίνει την δυνατότητα ελέγχου των κυματομορφών εισόδου και εξόδου του κυκλώματος.

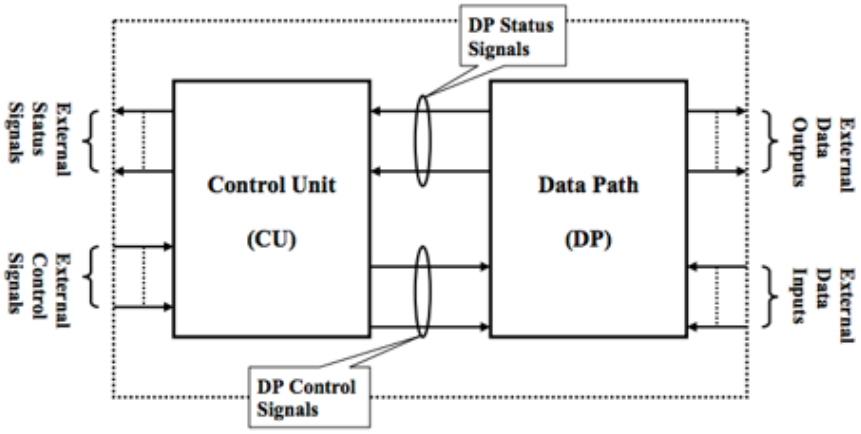
2. Μονοπάτι δεδομένων, μονοπάτι ελέγχου, σύγχρονη και ασύγχρονη σχεδίαση

3.1 Γενικά

Οι περισσότεροι επεξεργαστές και άλλα πολύπλοκα κυκλώματα είναι οργανωμένα σε δύο κύρια τμήματα: το μονοπάτι δεδομένων και το μονοπάτι ελέγχου. Το μονοπάτι δεδομένων περιλαμβάνει όλα τα κυκλώματα που υλοποιούν τους υπολογισμούς που υποστηρίζονται από το σύστημα και αποθηκεύει δεδομένα στη μνήμη. Σ ε πολλές περιπτώσεις, αυτά τα κυκλώματα είναι παράλληλα μεταξύ τους και το τελικό αποτέλεσμα καθορίζεται από την πολυπλεξία των επιμέρους αποτελεσμάτων.

Η μονάδα ελέγχου καθορίζει την λειτουργία του μονοπατιού δεδομένων, ενεργοποιώντας διακόπτες και καθορίζοντας σήματα ελέγχου σε πολυπλέκτες σύμφωνα με τις εντολές της μνήμης. Με αυτόν τον τρόπο, η μονάδα ελέγχου μπορεί να καθορίσει πως τα δεδομένα ρέουν μέσα στο μονοπάτι. (Digital System Design Using Data Path and Control Unit, 2013)

Η γενική δομή ενός σύγχρονου ψηφιακού κυκλώματος που υλοποιεί μία συγκεκριμένη λειτουργία φαίνεται παρακάτω:



6: Control and Data Path

Εξωτερικά σήματα ελέγχου: καθορίζουν την λειτουργία του κυκλώματος.

Εξωτερικά σήματα κατάστασης: υποδεικνύουν την κατάσταση του κυκλώματος.

Εξωτερική είσοδος/έξοδος δεδομένων: δεδομένα στο κύκλωμα.

Σήματα ελέγχου δεδομένων: σήματα που παράγονται από την μονάδα ελέγχου για έλεγχο των δεδομένων.

Σήματα κατάστασης δεδομένων: υποδεικνύουν την κατάσταση του μονοπατιού δεδομένων.

3.2 Μονοπάτι δεδομένων

Το μονοπάτι δεδομένων περιλαμβάνει μπλοκ που διαχειρίζονται δεδομένα. Δεν παρέχουν έλεγχο σε κανένα μπλοκ. Το μονοπάτι δεδομένων μπορεί να παρομοιαστεί με εργάτες που υλοποιούν συγκεκριμένες εργασίες στα δεδομένα και χρειάζονται επίβλεψη από κάποιον (στην συγκεκριμένη περίπτωση η επίβλεψη γίνεται από το μονοπάτι ελέγχου). Μερικά παραδείγματα μονοπατιού δεδομένων είναι:

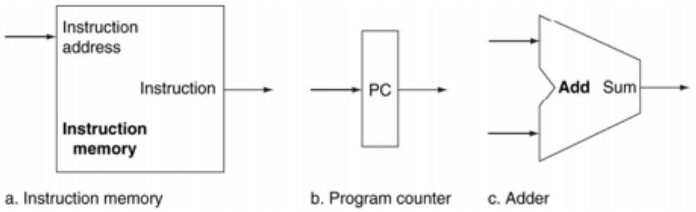
Registers: παράλληλοι καταχωρητές φόρτωσης για ανάγνωση δεδομένων παράλληλα, καταχωρητές ολίσθησης για σειριακή ανάγνωση.

Αριθμητικά κυκλώματα: αθροιστές, αφαιρέτες.

Πολυπλέκτες: δρομολογούν τα δεδομένα σε έναν ή περισσότερους προορισμούς.

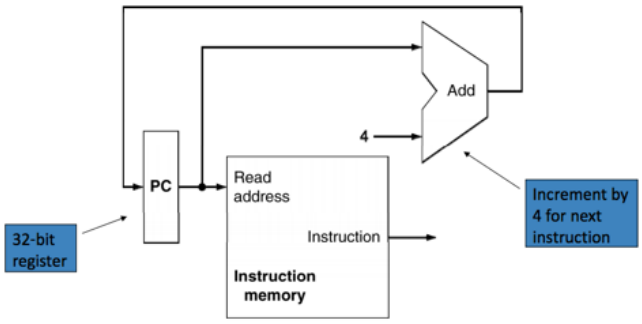
Μετρητές.

Για παράδειγμα, θα κατασκευάσουμε σταδιακά ένα απλό μονοπάτι δεδομένων MIPS θεωρώντας ένα υποσύνολο των λειτουργιών. Για να κατασκευάσουμε την προσκόμιση εντολής, χρειάζονται τα επόμενα τρία κομμάτια.



7: Basic components of Control and Data Path

ένας αθροιστής χρειάζεται για να αυξάνει τον μετρητή προγράμματος στην διεύθυνση της επόμενης εντολής. Μπορεί να υλοποιηθεί ως μια ALU μόνιμα συνδεδεμένη ώστε να υλοποιεί μόνο προσθέσεις. Ως εκ τούτου, δεν απαιτείται ξεχωριστό σήμα ελέγχου. Μια μονάδα μνήμης απαιτείται για την αποθήκευση των εντολών του προγράμματος που παρέχει μόνο λειτουργία ανάγνωσης και συνεπώς δεν απαιτεί σήμα ελέγχου. Τέλος, ο μετρητής προγράμματος είναι ένας καταχωρητής που αποθηκεύει την διεύθυνση της τρέχουσας εντολής. Μια νέα τιμή γράφεται σε κάθε κύκλο. Για την ενεργοποίηση της εγγραφής δεν απαιτείται σήμα ελέγχου. Συνδυάζοντας αυτά τα τρία κομμάτια, δημιουργούμε το μονοπάτι δεδομένων για την προσκόμιση της εντολής.



8: MIPS Data Path

Ένα ακόμα παράδειγμα μονοπατιού δεδομένων είναι το ακόλουθο. Για την πράξη της πρόσθεσης, τα δεδομένα θα ανακτηθούν από την μνήμη και τα περιεχόμενα των καταχωρητών reg1 και reg2 θα προστεθούν και θα αποθηκευτούν στον reg3. Η ακολουθία εντολών είναι η εξής:

- $reg1_{out}$, X_{in}
- $reg2_{out}$, choose X, addition, Y_{in}
- Y_{out}, $reg3_{in}$

Τα σήματα ελέγχου που είναι γραμμένα σε μια γραμμή εκτελούνται στον ίδιο κύκλο. Όλα τα υπόλοιπα σήματα παραμένουν σταθερά. Επομένως, στον πρώτο κύκλο τα περιεχόμενα του reg1 γράφονται στον καταχωρητή X μέσω του διαδρόμου. Τότε, τα περιεχόμενα του reg2 τοποθετούνται πάνω στον διάδρομο. Η ALU προσθέτει τα δεδομένα των X και reg2 και αποθηκεύει προσωρινά το αποτέλεσμα στον Y. Στο τελευταίο βήμα, το αποτέλεσμα μετακινείται από τον Y στον reg3 μέσω ενός εσωτερικού διαδρόμου. Μόνο ένας καταχωρητής μπορεί να τοποθετήσει δεδομένα

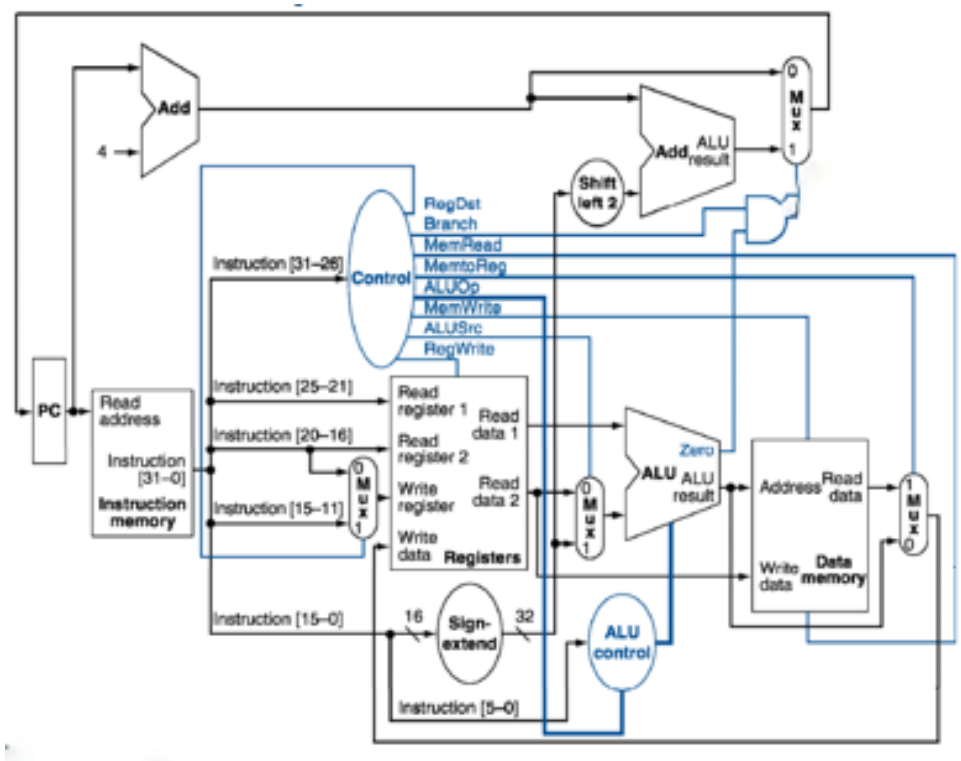
πάνω στον διάδρομο σε κάθε κύκλο, επομένως τα βήματα 2 και 3 δεν μπορούν να συγχωνευτούν. (Processor: Datapath and Control, 2014)

3.3 Μονάδα ελέγχου

Η μονάδα ελέγχου (ΜΕ) χειρίζεται όλα τα σήματα ελέγχου. Διαχειρίζεται και κατευθύνει όλη την ροή εισόδου και εξόδου, προσκομίζει εντολές και κατευθύνει άλλες μονάδες παρέχοντας σήματα ελέγχου και χρονισμού. Η μονάδα ελέγχου θεωρείται ως ο εγκέφαλος επειδή διευθετεί σχεδόν τα πάντα και διασφαλίζει την ομαλή λειτουργία. Ο John von Neumann συμπεριέλαβε την μονάδα ελέγχου στην αρχιτεκτονική του. Στις σύγχρονες σχεδιάσεις, η μονάδα ελέγχου είναι εσωτερικό τμήμα της CPU με την γενικότερη λειτουργία της να παραμένει αμετάβλητη. (Englander, 2009)

Ακριβέστερα, η μονάδα ελέγχου είναι μία ευμεγέθης συλλογή από πολύπλοκα ψηφιακά κυκλώματα που διασυνδέουν και ελέγχουν πολλές εκτελεστικές μονάδες εντός της CPU. Η ΜΕ είναι η πρώτη μονάδα της CPU που δέχεται από ένα εξωτερικά αποθηκευμένο πρόγραμμα μία εντολή. Η ΜΕ τότε αποκωδικοποιεί την εντολή σε μερικά ακολουθιακά βήματα που ελέγχουν και συντονίζουν την διαχείριση των δεδομένων από την CPU. Η σχεδίαση αυτών των ακολουθιακών βημάτων βασίζεται στις ανάγκες της κάθε εντολής και διαφέρουν ως προς τον αριθμό, τη σειρά εκτέλεσης και την ενεργοποίηση των μονάδων. Αυτό οδηγεί σε έναν υπολογιστή που μπορεί να εκτελέσει ένα πρόγραμμα χωρίς να απαιτούνται αλλαγές στο υλικό του. Αυτές οι αναλυτικές οδηγίες της ΜΕ υπαγορεύουν ποια σήματα εντός της ΨΠΘ πρέπει να ενεργοποιηθούν και ποιες μονάδες επιλέγονται με τη σωστή σειρά για να χειριστούν τα δεδομένα. Ανάλογα με τον τύπο της εντολής που εισέρχεται στην ΜΕ, η σειρά και ο αριθμός των ακολουθιακών εντολών μπορούν να διαφοροποιήσουν την επιλογή των τμημάτων που θα χρησιμοποιηθούν για την εκτέλεση της εντολής. Αυτό το χαρακτηριστικό που χρησιμοποιεί αποτελεσματικά μόνο εντολές λογισμικού για τον έλεγχο του υλικού είναι ένας από τους λόγους που οι σύγχρονοι υπολογιστές είναι ευέλικτοι όταν τρέχουν διάφορα προγράμματα. (Mukhopadhyay, 2012)

Ένας συνδυασμός ενός μονοπατιού δεδομένων μαζί με την μονάδα ελέγχου του φαίνεται παρακάτω (οι μπλε γραμμές αντιστοιχούν σε σήματα ελέγχου)

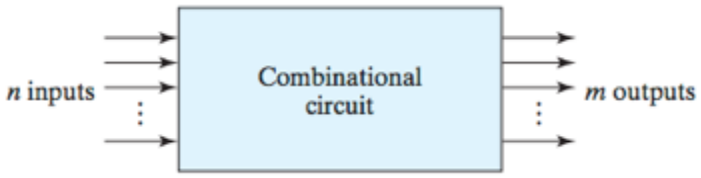


9: MIPS

3.4 Συνδυαστική (ασύγχρονη) σχεδίαση

Ένα συνδυαστικό κύκλωμα αποτελείται από διασυνδέσεις λογικών πυλών. Οι πύλες αντιδρούν στις τιμές των εισόδων τους και παράγουν ένα αποτέλεσμα στην έξοδο τους, μεταμορφώνοντας δυαδική πληροφορία από την είσοδο στην έξοδο τους. Ένα μπλοκ διάγραμμα ενός συνδυαστικού κυκλώματος φαίνεται παρακάτω. Οι n δυαδικές μεταβλητές εισόδου προέρχονται από εξωτερική πηγή. Οι m μεταβλητές εξόδου προέρχονται από την εσωτερική συνδυαστική λογική και συνεχίζουν σε εξωτερικά κυκλώματα. Κάθε μεταβλητή εισόδου και εξόδου υπάρχει ως ένα αναλογικό σήμα του οποίου οι τιμές μεταφράζονται σε ένα δυαδικό σήμα που αντιπροσωπεύει το λογικό 0 και 1. Σε πολλές εφαρμογές, η πηγή και ο προορισμός είναι καταχωρητές. Αν οι καταχωρητές περιλαμβάνονται στις συνδυαστικές πύλες, τότε το κύκλωμα θεωρείται ακολουθιακό. Για n μεταβλητές εισόδου υπάρχουν 2^n δυνατοί συνδυασμοί μεταβλητών εξόδου. Για κάθε πιθανό συνδυασμό εισόδου, υπάρχει μία δυνατή τιμή εξόδου. Επομένως, το συνδυαστικό κύκλωμα μπορεί να αναπαρασταθεί με έναν πίνακα αληθείας που περιέχει την τιμή εξόδου για κάθε συνδυασμό τιμών εισόδου. Ένα συνδυαστικό κύκλωμα μπορεί επίσης να περιγραφεί από m λογικές συναρτήσεις, μία για κάθε μεταβλητή εξόδου. Κάθε συνάρτηση εξόδου περιγράφεται με όρους από τις n μεταβλητές εισόδου.

Οι δυαδικές μεταβλητές εκφράζονται από ηλεκτρικές τάσεις και μερικούς άλλους τύπους σήματος. Αυτά τα σήματα μπορούν να επεξεργαστούν στις ψηφιακές λογικές πύλες και να υλοποιήσουν συναρτήσεις. Υπάρχουν μερικά συνδυαστικά κυκλώματα που χρησιμοποιούνται ευρέως στη σχεδίαση ψηφιακών συστημάτων. Αυτά τα κυκλώματα είναι διαθέσιμα ως ολοκληρωμένα κυκλώματα και υλοποιούν συνήθεις συναρτήσεις της ψηφιακής σχεδίασης. (Mano & Ciletti, 2007)



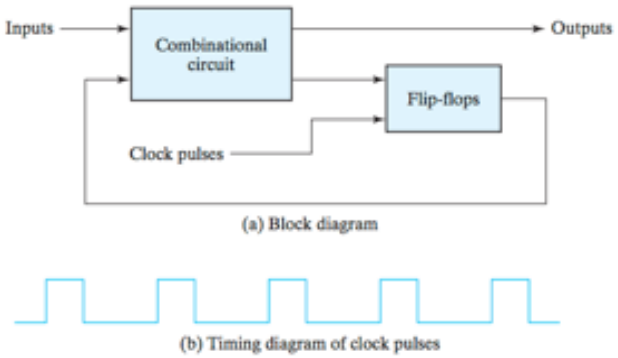
10: Combinational circuit

3.5 Ακολουθιακή (σύγχρονη) σχεδίαση

Το μπλοκ διάγραμμα ενός ακολουθιακού κυκλώματος φαίνεται στο σχήμα 10. Αποτελείται από ένα συνδυαστικό κύκλωμα στο οποίο έχουν συνδεθεί στοιχεία μνήμης για να σχηματίσουν έναν βρόχο ανάδρασης. τα στοιχεία μνήμης αποθηκεύουν δυαδική πληροφορία, η οποία καθορίζει την κατάσταση του κυκλώματος εκείνη τη στιγμή. Το ακολουθιακό κύκλωμα λαμβάνει πληροφορίες από τις εισόδους του και σε συνδυασμό με την τρέχουσα κατάσταση του καθορίζει την επόμενη κατάσταση. Το μπλοκ διάγραμμα φανερώνει ότι οι τιμές των εξόδων είναι συνάρτηση όχι μόνο των εισόδων αλλά και της αποθηκευμένης κατάστασης. Σε αντίθεση, οι έξοδοι ενός συνδυαστικού κυκλώματος εξαρτώνται μόνο από την τιμής της εισόδου.

Υπάρχουν δύο τύποι ακολουθιακών κυκλωμάτων. Ένα σύγχρονο ακολουθιακό κύκλωμα είναι ένα σύστημα του οποίου η συμπεριφορά καθορίζεται από την γνώση των σημάτων σε διακριτές στιγμές του χρόνου. Ένα ακολυθιακό κύκλωμα διαχειρίζεται σήματα που επηρεάζουν τα στοιχεία μνήμης σε διακριτές στιγμές χρόνου. Ο συγχρονισμός επιτυγχάνεται από μία γεννήτρια παλμών που παρέχει ένα σήμα ρολογιού που έχει την μορφή περιοδικών παλμών. Οι παλμοί του ρολογιού διαδίδονται στο σύστημα με τέτοιο τρόπο ώστε οι παλμοί να επηρεάζουν τα στοιχεία μνήμης στην άφιξη τους. Για παράδειγμα, σε ένα κύκλωμα που προσθέτει δύο στοιχεία και αποθηκεύει το αποτέλεσμα, θα έκανε την πρόσθεση και θα αποθήκευε το άθροισμα στην άφιξη ενός παλμού ρολογιού. Ονομάζονται σύγχρονα κυκλώματα επειδή η δραστηριότητα μέσα στο κύκλωμα και η ανανέωση των τιμών στις μνήμες γίνεται με παλμούς ρολογιού. Η σχεδίαση τέτοιων κυκλωμάτων είναι εφικτή επειδή σπάνια αντιμετωπίζουν θέματα αστάθειας και ο χρονισμός τους μπορεί να χωριστεί σε μικρότερα τμήματα.

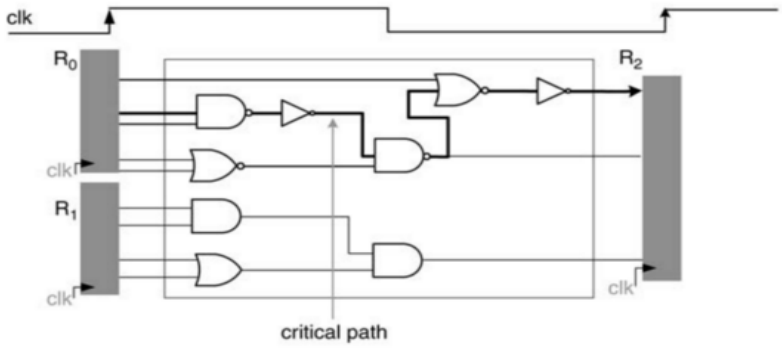
Η μνήμη που χρησιμοποιείται στα ακολουθιακά κυκλώματα ονομάζεται flip flop και μπορεί να αποθηκεύσει ένα μπιτ δυαδικής πληροφορίας. Σε σταθερή κατάσταση, η τιμή του μπορεί να είναι 0 ή 1. Σε ένα ακολουθιακό κύκλωμα, η έξοδος καθορίζεται από την είσοδο και τις τιμές που είναι αποθηκευμένες στα flip flop. Η νέα τιμή αποθηκεύεται όταν συμβεί ένας παλμός ρολογιού. Πριν την έλευση του παλμού, η συνδυαστική λογική που δημιουργεί την είσοδο του flip flop πρέπει να έχει μια σταθερή τιμή. Επομένως, η ταχύτητα λειτουργίας ενός ακολουθιακού κυκλώματος είναι κρίσιμη. Οι καθυστερήσεις διάδοσης παίζουν σημαντικό ρόλο στον καθορισμό της ελάχιστης απόστασης μεταξύ των παλμών του ρολογιού ώστε το κύκλωμα να λειτουργεί σωστά. (Mano & Ciletti, 2007)



11: Block Diagram and Timing Diagram of Clock Pulses

3.6 Μονοπάτια

Στην ψηφιακή σχεδίαση, είναι σύνηθες δεδομένα που παράγονται σε ένα σημείο του κυκλώματος να πρέπει να μεταφερθούν σε ένα άλλο σημείο για να αποθηκευθούν ή να επεξεργαστούν. Η διαδρομή που συνδέει την αφετηρία με τον προορισμό των σημάτων ονομάζεται μονοπάτι. Αν και τα σήματα ταξιδεύουν με μεγάλη ταχύτητα μέσα στο κύκλωμα και οι αποστάσεις δεν είναι μεγάλες, απαιτείται κάποιος χρόνος για την μεταφορά τους. Αυτός ο χρόνος ονομάζεται καθυστέρηση και μπορεί να επηρεάσει σημαντικά την απόδοση ενός κυκλώματος. Καθώς υπάρχει ένας μεγάλος αριθμός μονοπατιών σε κάθε ψηφιακή σχεδίαση, το μεγαλύτερο μονοπάτι, δηλαδή αυτό που απαιτεί τον περισσότεο χρόνο διάδοσης, ονομάζεται κρίσιμο μονοπάτι. Το κρίσιμο μονοπάτι πρέπει να είναι μικρότερο από την επιτρεπόμενη καθυστέρηση που καθορίζεται από τον παλμό ρολογιού.



12: Critical Path

Η καθυστέρηση ενός μονοπατιού είναι το αποτέλεσμα πολλών παραγόντων και περιορισμών κατά την σχεδίαση και τον κύκλο λειτουργίας. Ο επόμενος πίνακας συνοψίζει μερικούς λόγους που καθορίζουν την καθυστέρηση ενός μονοπατιού.

	Silicon foundary engineer	Cell library designer, FPGA chip designer	CAD tools (logic synthesis, place and route)	Designer
Number of levels			synthesis	RTL
Internal cell delay	Physical parameters	cell topology, transistors sizing	cell selection	
Wire delay	Physical parameters		place and route	layout generator
Cell input capacitance	Physical parameters	cell topology, transistors sizing	cell selection	
Cell fanout			synthesis	RTL
Cell drive strength	Physical parameters	transistor sizing	cell selection	

Ένας σχεδιαστής πρέπει να λάβει υπόψη του όλα τα συνδεδεμένα ζεύγη, μονοπάτια από είσοδο σε καταχωρητή και από καταχωρητή σε έξοδο. Τα σχεδιαστικά εργαλεία μπορούν να βοηθήσουν στην αναζήτηση επειδή αναφέρουν τις καθυστερήσεις των μονοπατιών και προσομοιωτές μπορούν να αναλύσουν την χρονική απόδοση των κυκλωμάτων. (Wawrzynek, 2013)

Υπόθεση εργασίας

Το κύκλωμα που μελετήθηκε ήταν μια αριθμητική και λογική μονάδα που υλοποιεί πράξεις πάνω σε δύο 64bit προσημασμένους ακέραιους. Δέχεται επίσης στην είσοδο ένα σήμα 4 bit που ελέγχει την πράξη που θα υλοποιηθεί. Οι πράξεις που υποστηρίζονται είναι:

- Πρόσθεση
- Αφαίρεση
- Αύξηση πρώτου αριθμού κατά ένα
- Αύξηση δεύτερου αριθμού κατά ένα
- Μείωση πρώτου αριθμού κατά ένα
- Μείωση δεύτερου αριθμού κατά ένα
- Πολλαπλασιασμός
- Σύγκριση
- Λογικό AND
- Λογικό OR
- Λογικό XOR
- Λογικό NOT
- Λογικό NAND
- Λογικό NOR
- Ολίσθηση αριστερά
- Ολίσθηση δεξιά

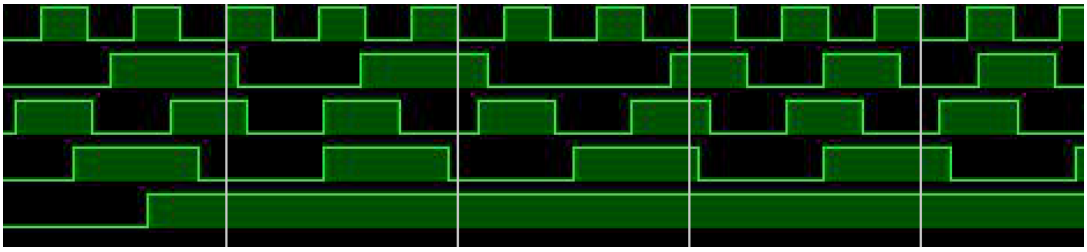
Το κύκλωμα είχε καθυστέρηση 5.7 nanosecond που σημαίνει ότι η μέγιστη συχνότητα λειτουργίας του είναι 175 MHz. ανεξάρτητα από τις τιμές των ορισμάτων ή την πράξη που υλοποιείται, αυτή η συχνότητα ήταν σταθερή σε όλη τη διαδικασία της προσομοίωσης.

Ωστόσο, το κύκλωμα ήταν ικανό να διαχειριστεί ακόμα μεγαλύτερες συχνότητες. Μετά την ανάλυση, βρέθηκε ότι υπάρχει μια περιοχή του κυκλώματος που μπορεί να λειτουργήσει σε μεγαλύτερη συχνότητα (304 MHz) όταν το υπόλοιπο κύκλωμα είναι ανενεργό ή ημιενεργό. Για λόγους ασφαλείας, η μονάδα χρονίστηκε λίγο χαμηλότερα στα 300 MHz όταν η συχνότητα λειτουργίας του κρίσιμου μονοπατιού είναι 175 MHz.

Το κύκλωμα του scheduler χρειάζεται δύο κύκλους ρολογιού για να ανιχνεύσει τις αλλαγές και να επιλέξει το κατάλληλο ρολόι. Για να αυξηθεί η διαπερατότητα του κυκλώματος, δημιουργήθηκε ένα pipeline 3 σταδίων. Δύο κύκλους ρολογιού μετά την αλλαγή σε δεδομένα εισόδου, η νέα συχνότητα είναι έτοιμη και φτάνει πριν τον τρίτο κύκλο μαζί με τα δεδομένα που προκάλεσαν την αλλαγή. Ως αποτέλεσμα, τα δεδομένα επεξεργάζονται με την σωστή συχνότητα και μετά τον τέταρτο κύκλο το αποτέλεσμα είναι σταθερό στην έξοδο της μονάδας.

Τα σήματα που προκαλούν αλλαγές συχνοτήτων είναι τα 4 bit που ελέγχουν την πράξη και μερικά από τα πιο σημαντικά ή ελάχιστα σημαντικά ψηφία των δύο αριθμών εισόδου. Μόνο μερικά από τα ενδιάμεσα ψηφία προκαλούν αλλαγή συχνότητας.

Το test bench περιελέμβανε 150 πράξεις. Οι μισές ήταν πρόσθεση, 20 πολλαπλασιασμοί, 35 λογικές πράξεις και 20 συγκρίσεις. Οι περισσότερες προσθέσεις έγιναν υπό την αυξημένη συχνότητα και ήταν η πράξη με την μεγαλύτερη βελτίωση. 28 από τις 75 πράξεις (37%) έγιναν με την μεγαλύτερη συχνότητα. Ο συνολικός χρόνος εκτέλεσης μειώθηκε κατά 21%. 31% των λογικών πράξεων έγιναν με την αυξημένη συχνότητα και ο χρόνος εκτέλεσης μειώθηκε κατά 18.3%. Οι συγκρίσεις και οι πολλαπλασιασμοί έγιναν κυρίως στην αρχική συχνότητα. Μόνο μία σύγκριση πραγματοποιήθηκε στα 300 MHz βελτιώνοντας τον χρόνο κατά 2.9%. Καμία βελτίωση δεν παρουσιάστηκε στον πολλαπλασιασμό.



13: Slow to Fast Clock Switching

Συμπεράσματα

Η παρούσα διπλωματική εργασία πέτυχε τον στόχο της καθώς η μεθοδολογία που αναπτύχθηκε κατάφερε να λειτουργήσει ένα κύκλωμα πέρα από την προκαθορισμένη του συχνότητα για πρώτη φορά χωρίς λάθη. Το κύκλωμα συνεχώς παρακολουθεί τα σήματα και λειτουργεί σε συχνότητα που δεν επηρεάζει την ορθή λειτουργία του. Επιπλέον, η συγκεκριμένη μεθοδολογία δεν εξαρτάται από την δομή του κυκλώματος και μπορεί να χρησιμοποιηθεί σε οποιοδήποτε hardware IP.

Η παρακολούθηση των σημάτων εισάγει κάποια καθυστέρηση στη σχεδίαση. Ωστόσο, η ολική καθυστέρηση είναι καλύτερη ή ίση με την περίοδο του πιο αργού ρολογιού. Η αλλαγή ρολογιών χρειάζεται στην χειρότερη περίπτωση χρόνο όσο η περίοδος του πιο αργού ρολογιού. Αυτό σημαίνει ότι στην χειρότερη περίπτωση το κύκλωμα λειτουργεί όπως το αρχικό χωρίς τις τροποποιήσεις. Σε κάθε άλλη περίπτωση, το κύκλωμα μπορεί να αυξήσει την συχνότητα του και να λειτουργήσει ταχύτερα.

3. FPGA Timing

3.1 Suggested solution

In the following sections it will be described the whole process of analyzing the circuit and building the feedback circuit as well as "assembling" the units and implementing them on the FPGA chip. In the appendix A, there are some detailed tutorials about Planahead, the custom tool named Planahead Expander as well as step by step guide to implement the new and enhanced circuit on the FPGA and run simulations using Xilinx tools.

3.1.1 Timing information of original circuit

In order to analyze the circuit and study every path that is in the design under examination, Planahead tool by Xilinx will be used. Planahead manages the source code files of the design, synthesizes it and implements it. User is able to insert the desired timing constrains (for instance the greatest possible clock period, or the latest nanosecond that data must be in a stable state) and Planahead tries not to violate any of these constrains. In case that the constrains are too tight and Planahead is not able to meet them successfully, it will be mentioned in the timing report and user will be prompted to insert new and looser timing constrains. Planahead gives also the ability to inspect the inner structure of a circuit, make manual changes and adjusting the optimizations level that will take place in the circuit (for example, how Planahead will handle the input and output pins, the usage of buffers and D flip flops for synchronizarion and many other options).

After synthesizing the design, its output files with the extensions **edf** (netlist of the implemented design) and **twr** (timing report of the implemented design) will be used for further analysis by the custom tool Planahead Expander in order to locate the signals that we are going to study further. In appendix B, the structure of those two files will be explained. Quite briefly, edf file stores all information about how pins are connected to each other and which signals reach each pin of the components used in the design. On the other hand, twr file contains information about the delay found in each path analyzed and reports all violations that may occurred (a negative number representing the slack of a path indicates that some violation happened because the timing constrains were too strict).

Planahead also outputs many other log files and reports in order to help user follow the results of each stage. There are, however, many output files which are encrypted and are not user accessible. Those files are used by Planahead and are not meant to be processed by user. Finally, some of the files contain the same information but in a different file format (to illustrate, twx is exactly the same timing report as in twr file but it uses the xml format instead of the plain text).

a. Finding crossroads

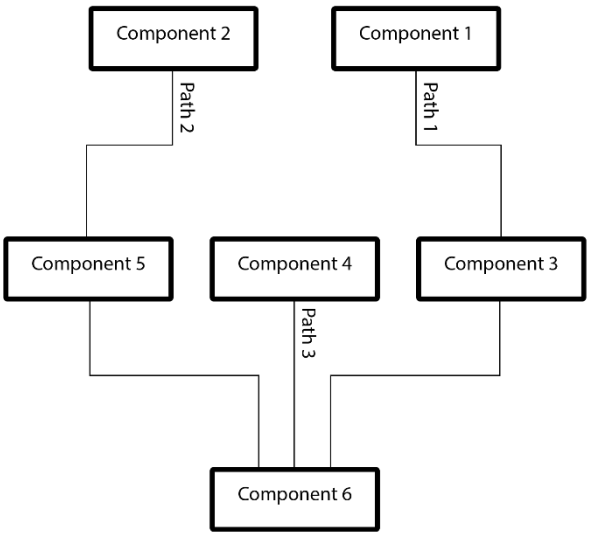
Planahead Expander parses the netlist file (edf file) as well as the timing report (twr file) and stores all information into an internal database. First, the tool parses the edf file and stores the signals of the circuit as well as all the pins of the components they reach. Then, the tool starts parsing the timing report and stores information such as source and destination component, total path delay, components that are crossed by the path and their names in the database.

In timing report, besides component names and delays, signal names which connect the components are also mentioned. When the timing report parser finds the name of a signal in the path under examination, it searches into database for the pins that this particular signal reaches. At this point, the tool makes a pin - path match and it is the first major step into finding the functionality. The matches that Expander makes, are all stored in the database as well. The way that Expander makes the matches justifies the design option to parse the netlist file first. It is worth mentioning that up to this point, Expander is able to perform pin - path matches **correctly** only in circuits that their timing report **does not** include signal or component names in the form of directories (for example, design1/component1/and1/input1). Some of these directories are quite simple and they can be resolved automatically by the application, otherwise the user will be informed via a graphical user interface option that results may not be completely accurate. If the circuit uses **unique** names and identifiers for signals and components, Expander will produce one hundred percent **accurate** results. (Directories are the result of automatic renaming performed by Xilinx tools during synthesis and implemenation. Directories may also occur when the source code of the circuit has an hierarchical structure. Directories are present in the timing report file but they are "disassembled" in the edf file in an unfathomable way. As a result, Expander is not able to successfully match the signal with the corresponding pin).

In addition to the above information, timing report provides detailed information about the delay and its distribution in the path. There are quite a few types of delays mentioned in a timing report (for example, "net" delay explains the delay a signal meets when it propagates through a wire. Net delays are usually higher than other types of "logic" delay). Expander stores the total delay up to the pin currently examined in the database. If the component or the pin is met a second time, Expander compares the current delay to the previous one stored and keeps the greater one because a "worst case scenario" is approached. That way it is certain that afterwards no violation of the the timing constrains of the circuit will happen. A detailed analysis of the delays per path is presented to the user in one of the output files that Expander produces. The whole delay analysis is taking place during the timing report parsing and the results are used later.

After the matching has been done (it takes place in parallel with timing report parsing), the tool is able to locate **meeting points** in the

design. A meeting point is defined as the component that two or more distinct paths are reaching it. For instance, in the following figure it is already known that paths with identification numbers 1 and 2 are reaching pins I1 and I2 of component with name "demonstration". Expander is able to figure out that "demonstration" is a meeting point and uses it for the next steps in the analysis. Meeting points are presented to the user in a pdf file after the application is successfully terminated.



14: Path Example

b. Finding functionality

The functionality of the paths in the meeting points is further analyzed in order to form a "selector" component that will take decisions. In other words, this new component will be able to monitor the functionality of the meeting points and select the best operating frequency to speed up the process with no or insignificant functionality errors.

For each signal which affects the functionality, Expander applies a BFS (breadth first search) algorithm in order to locate the proper signals because the circuit features quite common structure with a graph (components are the nodes of the graph and nets are the edges that connect the graph). The implementation of the algorithm has been modified in order to be adjusted to the data structures used.

Finally, all the signals which were found in this step are presented to the user in an Excel (.xlsx) file in columns along with the delay of each path. That file not only helps user verify and understand the results but it is also needed for the final step of the processing done by Expander.

c. Presenting path signals

The last part of Expander reads the previous Excel file and exports a text file with a worst case approach scenario. That file will be later used by Generator in order to create the "selector" circuit in VHDL that monitors

the functionality and promotes the proper clock frequency back to the original circuit.

Expander reads the signals. For each different signal it finds in the Excel file, it checks the delays of the paths that are controlled by that signal. As mentioned before, it uses a worst case approach, which means that it keeps the largest delay of the paths. This is done to ensure that no timing violations will take place during synthesis, implementation and simulation.

In the text file, it is mentioned the name of each parental signal along with the maximum frequency that it can trigger. That frequency is calculated as the reciprocal of the delay found before. After the creation of this text file, Expander terminates because its job has been done.

Important note about Expander

Expander displays many messages to its console in order to help user track the state of the process. When everything terminates normally, the console will contain a message of successful termination. Otherwise, an error message will be displayed in red which will inform the user about the error.

d. Manual retouch of file

Before generating the VHDL code, some manual changes to the output file of Expander are required in order Generator to run without problems. In particular, the frequencies must be grouped and then sorted in ascending order (Generator uses a binary search algorithm). The user can define **up to six** groups with different frequencies (these numbers are assuming that the user implemented the circuit on a Kintex 7 platform. Numbers **may vary** when using other platforms). These two operations that are taking place manually can be better explained through a simplified example.

In the following table, Expander found ten parental signals each with a different frequency. Please take into consideration that some of these frequencies are quite close. This is a strong indication that those frequencies can target the same group which is going to be characterized by the slowest signal (largest delay). Of course user is able to group the frequencies as desired but groups must always follow the limitations about:

1. Up to how many different groups the digital clock manager can support (up to six in Kintex 7)
2. Respect the upper limits of each signal. Signals cannot be accelerated because this will lead to timing violations of the paths.

Signal name	Signal frequency (MHz)
A	111
B	247
C	250
D	115
E	118
F	222
G	320

H	322
I	360
J	121

2: Example of Expander Output

From the above table it can be claimed that three groups can be created. The first group is going to consist of signals A, D, E and J because their frequencies are close to each other. This group will get a frequency equal to its slowest; in that case 111 mhz. With same thoughts, the second group consists of signals B, C and F with a frequency of 222 mhz. Lastly, the remaining signals will compose the third group with a frequency of 320 mhz.

The file must be manually rearranged by user in order Generator to create the VHDL code for the monitoring and selecting circuit. The above table should be transformed as shown below, in order Generator to function properly.

Signal name	Signal frequency (MHz)
A	111
D	111
E	111
J	111
B	222
C	222
F	222
G	320
H	320
I	320

3: Sorted Generator Input

e. Generating VHDL Code

Generator is a fully automatic tool. It parses the files containing the signal names along with their frequencies as they were **edited manually by the user** which contains a list of signal names and their corresponding frequency. During parsing, Generator creates a list with the different frequencies found in the file. This list is already sorted in ascending order because the input file was created that way. This list helps Generator define the index that Selector should output when a signal it monitors changes.

Generator uses some helping functions (such as convert a string to binary number) as well as a vhdl code generator which creates a VHDL file.

After Generator terminates successfully, an output file entitled "Selector.vhd" is created, which contains synthesizable VHDL code. This code monitors the parental signals specified by Expander and outputs a vector which is the index needed for frequency selection. This file will be later added in to Planahead. (Detailed instructions can be found in the appendix A).

f. Creating digital clock manager (DCM)

The third and last segment of the new and enhanced circuit is the digital clock manager. This component can be easily created via a graphical user interface in a Xilinx tool called Core IP Generator. However, the automatically created code needs some modifications by user in order to be properly implemented into the design.

Digital clock manager is a special structure which deals with multiple clocks in the same circuit. More specificly, a digital clock manager accepts as an input a clock pulse of a user defined frequency (on Kintex 7 the range of accepted frequencies are 100 up to 900 mhz) and produces up to six different clock pulses of user desired frequencies (once again a digital clock manager targeting Kintex 7 supports up to six outputs. Other platforms may support fewer or more output clock pulses). By default, all output clocks are connected to global clock buffers in order to be accessible by the rest of the circuit.

It is worth mentioning that Xilinx does not provide the exact way that the digital clock manager functions. However, it is mentioned that the manager performs suitable multiplications and divisions on the input clock signal in order to generate the desired output pulses. That is the main reason that a digital clock manager **may fail** to produce **exactly** the desired outputs; if frequencies are too close (almost equal), the manager will be unable to perform proper operations and the resulted clocks will not be the desired. DCMs also eliminate clock skew, thereby improving system performance. Similarly, a DCM optionally phase shifts the clock output to delay the incoming clock by a fraction of the clock period.

Another structure needed to build the digital clock manager is called bufgmux and it is provided by Xilinx as well. This is a special multiplexer 2 to 1 (cannot be modified by user) which operates the same way as a normal multiplexer but has some key differences. First of all, bufgmux accepts in its input pins two clock pulses and not signals of std_logic(_vector) as well as a select signal (std_logic only) which selects the clock that will be forwarded to the output. However, the most important difference compared to a simple multiplexer is the way that the clock switching is happening. Because bufgmux drives many other synchronous components with its clock, it must be ensured that the switching will take place fast and no glitches or spikes will appear. The clock signal must always be stable in order not to trigger flip flops accidentally. A normal multiplexer is not able to guarantee such smooth switching so it is unappropriate for such a sensitive task on the fabric.

When the S input changes, the bufgmux does not drive the new input to the output until the previous clock input is Low and the new clock input has a High-to-Low transition (please refer to the next table). By not toggling on the first Low-to-High transition of the input, the output clock pulse is never shorter than the shortest input clock pulse.

Inputs			Outputs
I0	**I1**	**S**	**O**
I0	X	0	I0
X	I1	1	I1
X	X	↑	0
X	X	↓	0

4: Switching between Clocks

If the user needs to connect more than two clock frequencies, user can utilize more bufgmux units into cascode mode (the output of the first multiplexor will become the input of the second and so on). Each bit of the indexing signal will drive a single layer of multiplexor. The output of the last multiplexor will be the desired output of the digital clock manager. It is highly important to mention that each FPGA offers a limited number of units "bufgmux" and user must pay attention to that when grouping the parental signals. The exact number of such units are mentioned in the data sheet of the FPGA used.

Detailed instructions on how a digital clock manager is built and which modifications are required to the output file in order to be properly implemented with the rest of the code can be found in appendix A.

g. Schematic of the enhanced circuit

The new circuit is composed by three parts: the original circuit, the selector and the digital clock manager. The connections of these three segments can be made either in VHDL level or using a tool provided by Xilinx called FPGA editor. The former can be used for input signals and the latter for internal signals which are not known or visible in VHDL level. Detailed instructions on how to use FPGA editor can be found in appendix A.

Data delay can be easily achieved by putting the proper number of D flip flops before the input of the original circuit. This will delay the input until its clock pulse is ready. It is worth mentioning that those flip flops will also be triggered by the clock that is selected by the digital clock manager.

Future Work

The current work could be expanded towards various directions. Some of those are listed below.

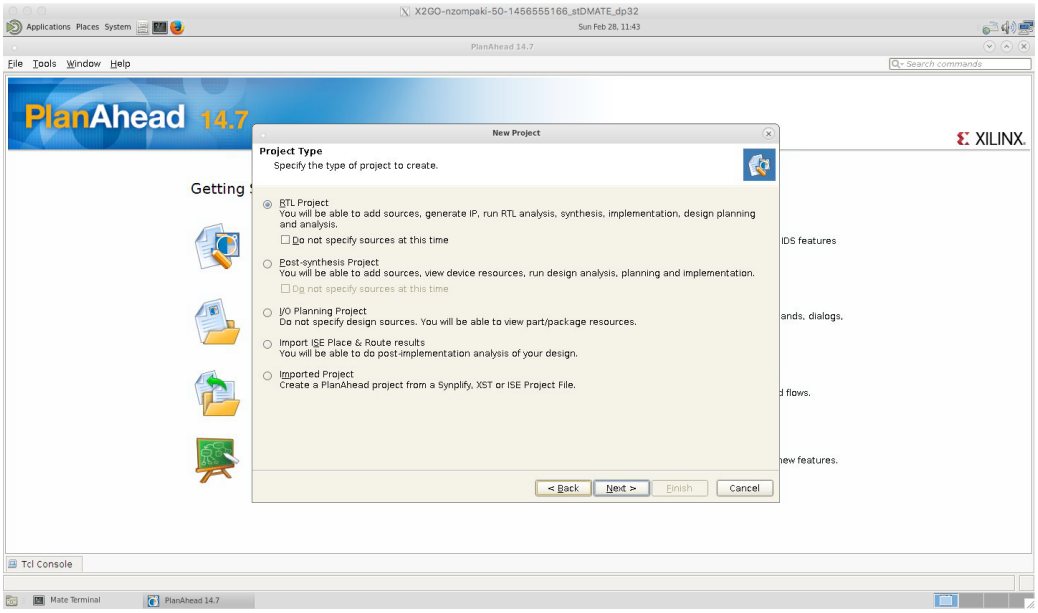
An urging matter that requires further examination is the renaming of components and signals that occurs during the process of the netlists by Planahead, as well as the disintegration of the names that are parsed as directories. Both problems can be surpassed by migrating the current project in the 2014.2 version of Vivado, that deals with those issues effectively. Another advantage of using Vivado is the enhanced and more modern version of FPGA Editor which allows more accurate and user - friendly manual modifications of placement and routing. A proposed solution to the renaming issue is the following. Using the elaborated design, it is possible to find the correct net name whose names need to be preserved and set the correct MARK_DEBUG constraints in the XDC. The correct name for the RndData net is Data because the net exits the module via port Data. After applying MARK_DEBUG constraint on these net names found via the elaborated design using set_property MARK_DEBUG true [get_nets ...], synthesis is able to correctly apply the constraints and preserve the nets. In the netlist, the net name does change, but the MARK_DEBUG properties are preserved. This is an expected behavior as Vivado synthesis does rename ports in the default flatten_hierarchy rebuilt flow. (Vivado Synthesis - Net names are not preserved by mark_debug, 2015)

Furthermore, the project could be expanded by implementing approximate computing, in favor of acceleration of datapath execution and increased exploitation of the slack of the paths. Approximation is not a new idea, as it has been used in areas such as lossy compression and numeric computation; in fact, John von Neumann wrote a paper on it in 1956 (Probabilistic logic and the synthesis of reliable organisms from unreliable components, Automata Studies (Shannon & McCarthy, 1956). According to a Computing Community Consortium blog post on the U.S. Defense Advanced Research Projects Agency (DARPA) 2014 Information Science and Technology (ISAT) Targeted Approximate Computing workshop, a number of researchers are working in this area. (Kugler, 2015)

Appendix A – Detailed Tutorial

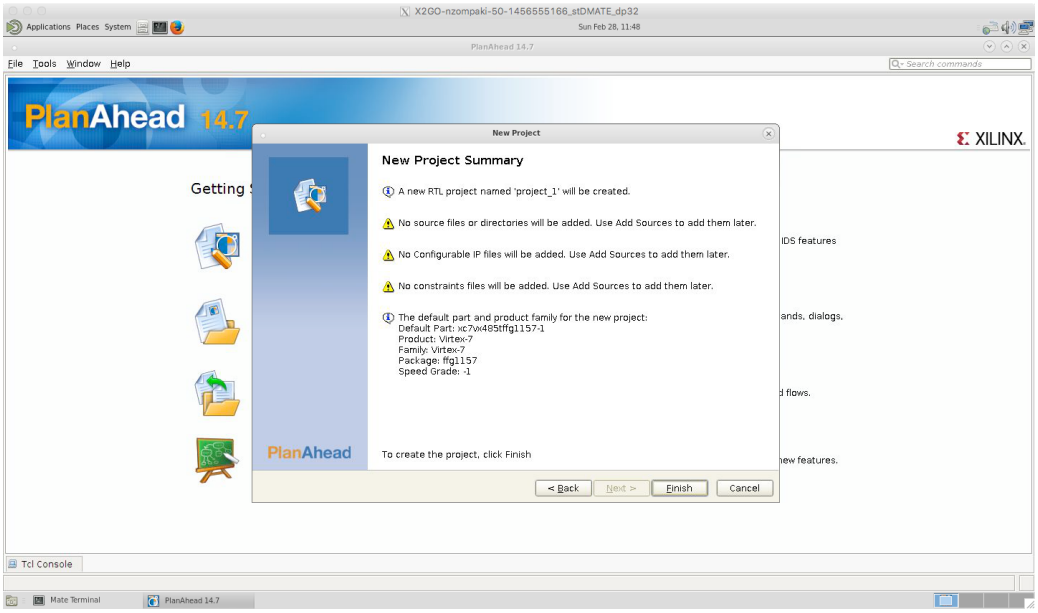
A.1 From VHDL to implementation

1. Launch Planahead and from the opening screen choose “Create new project”
2. Follow the instructions of the pop up window.
3. In screen “Project type” choose RTL project.



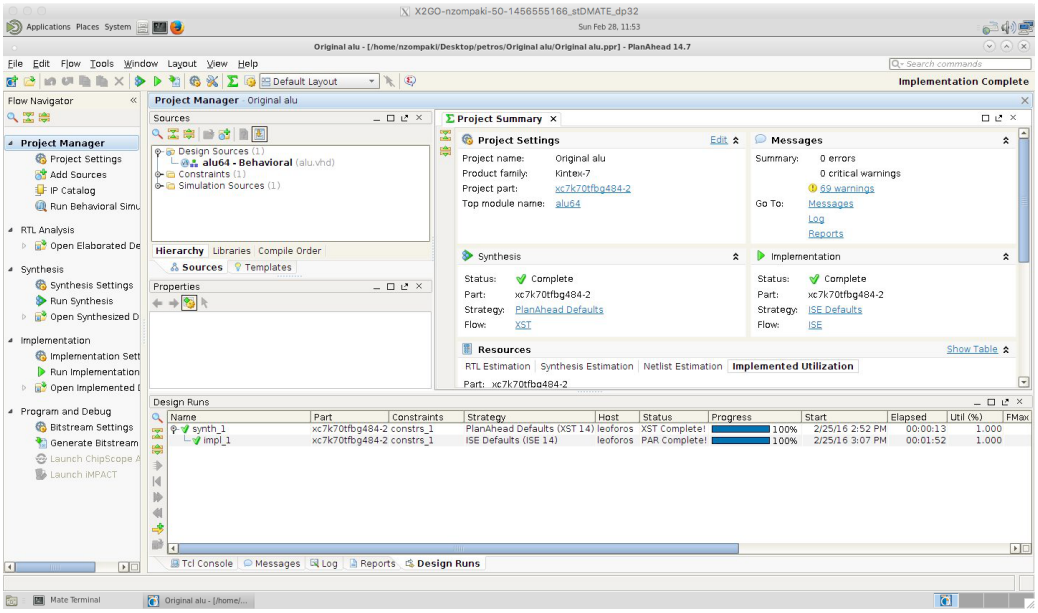
15: Opening Screen of Planahead

4. Specify the source code (in Verilog or VHDL) you want to insert.
5. In screen “Default part” choose the target device you wish
6. Check the settings specified and click “Finish”



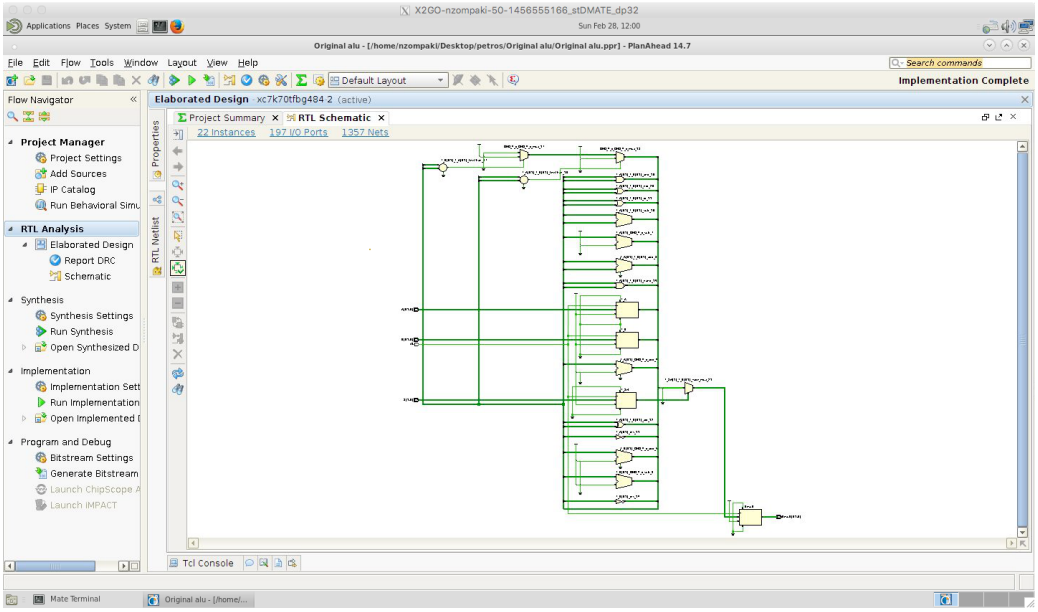
16: New Project Screen

The main screen of Planahead is now open. On the left side there is “Flow Navigator”, which contains all steps needed to implement the design as well as settings panel and many other useful tools.



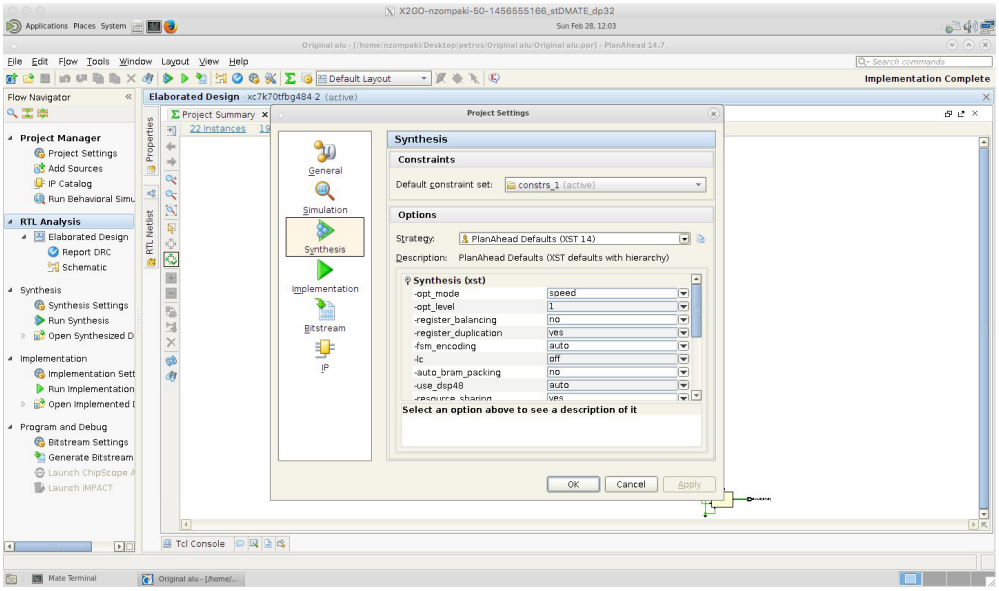
17: Main screen of Planahead

7. (Optional) In case you want to add more source files, click on the “Add sources” option of Flow navigator and follow the instructions of the pop up window.
8. (Optional) Launch Simulator by clicking on “Run behavioral simulation” to verify the circuit operation, make sure that the source files do not contain syntax errors and the code does not have any critical bug.
9. (Optional) Click on “Open Elaborated design” to see a schematic of the circuit and ensure that all connections have been done properly.



18: RTL Schematic

10. Click on “Synthesis settings”. A new window opens which contains all settings which can be configured by user. Default settings are okay, but user can make changes.

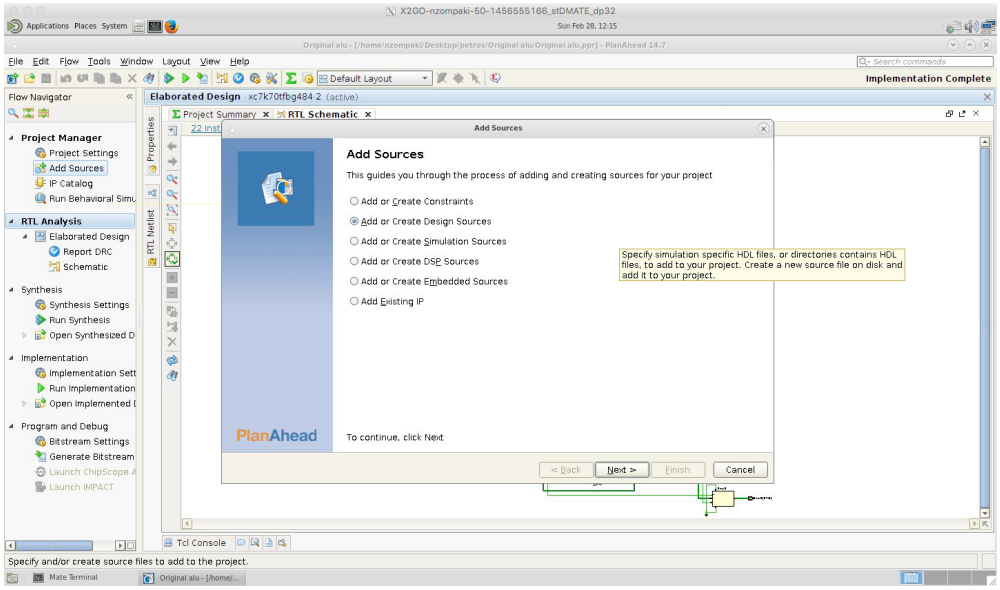


19: Synthesis Settings

11. Click “Apply” and then “OK” to save any changes.

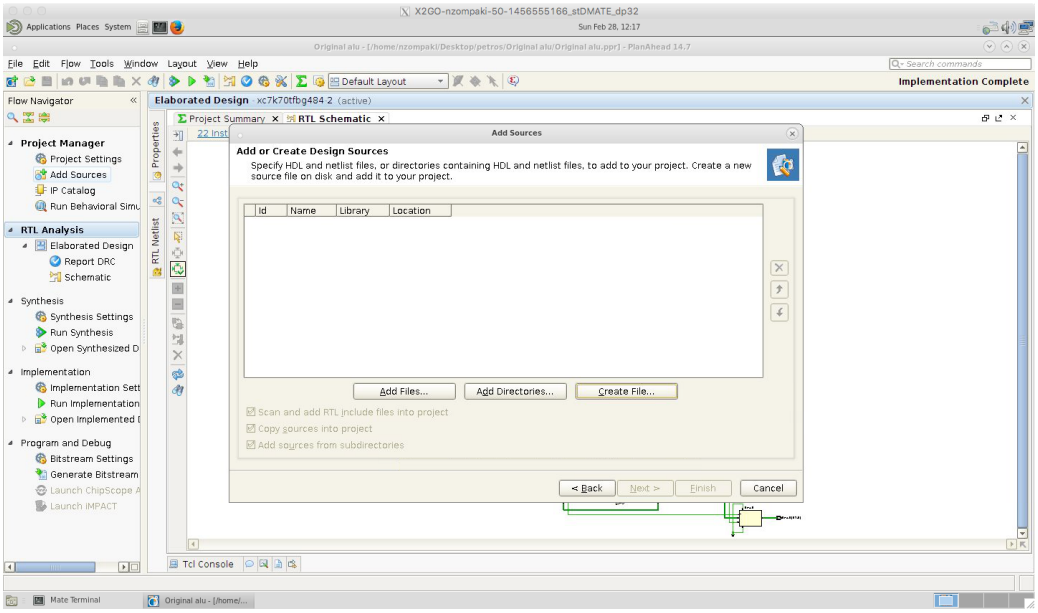
Now a wrapper file must be created in order to create partitions (this will be done later). VHDL top file must contain only one entity in order to be compiled successfully. So, a wrapper file is needed in order to wrap the main circuit inserted before and the two new components that will be inserted later. Wrapper file is like a main function of a common programming language which calls and controls the rest of the source code. Wrapper will be the top module of the entire circuit and will control all the separate source code files.

12. Click on “Add sources” from flow navigator and choose “Add or create design sources”.



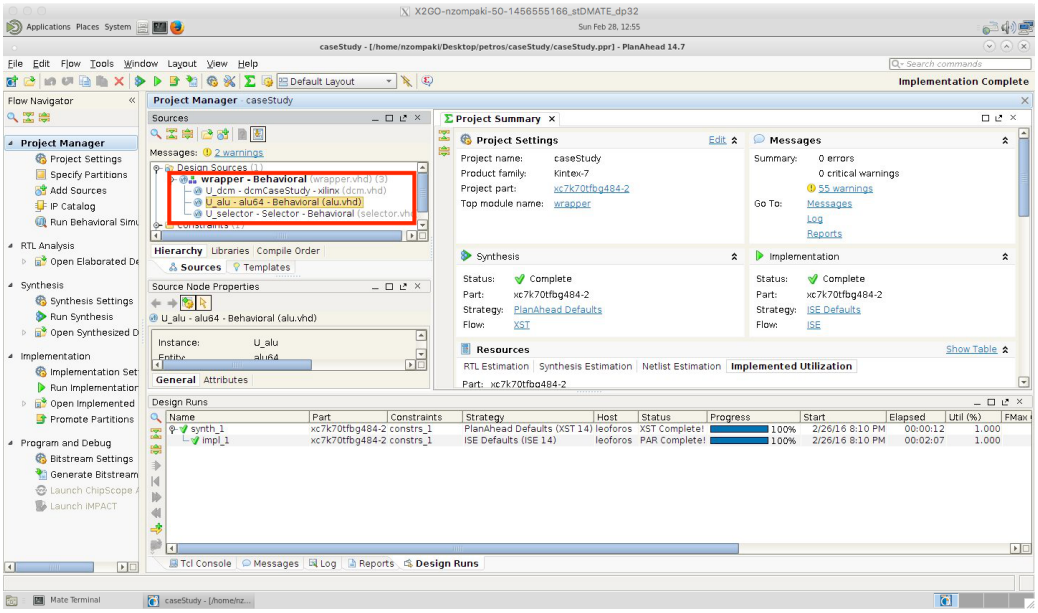
20: Add Sources Screen

13. Click “Create new file”.



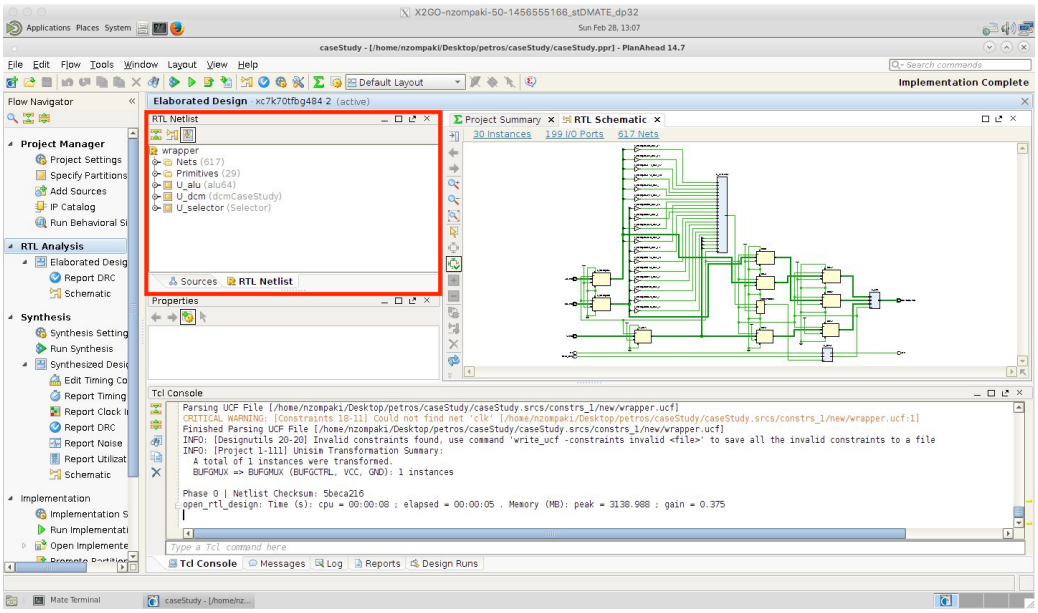
21: New File

14. Follow instructions until the new file is created. The file will be blank and as a result it will produce syntax errors. Write the Verilog or VHDL code for the wrapper (example of source code will be included in Appendix C).
15. After saving the file, make sure that Planahead recognized the hierarchical structure of the project. Wrapper file should be on top and below it should be the circuit.



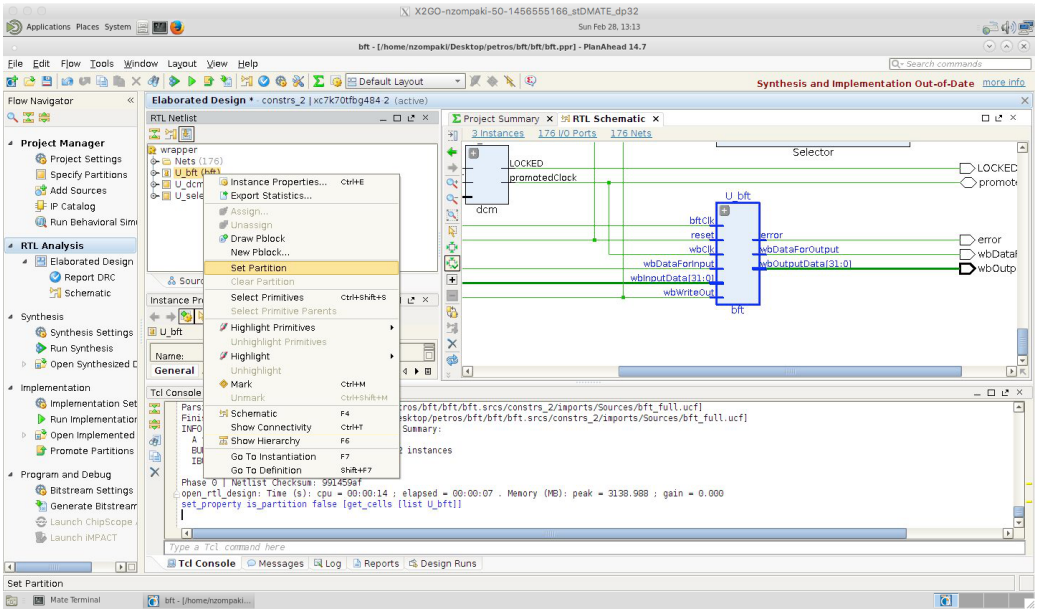
22: Hierarchical Code Structure

16. Click on “Open Elaborated design” from project navigator.
17. From the sources panel make sure that “RTL netlist” tab is selected.



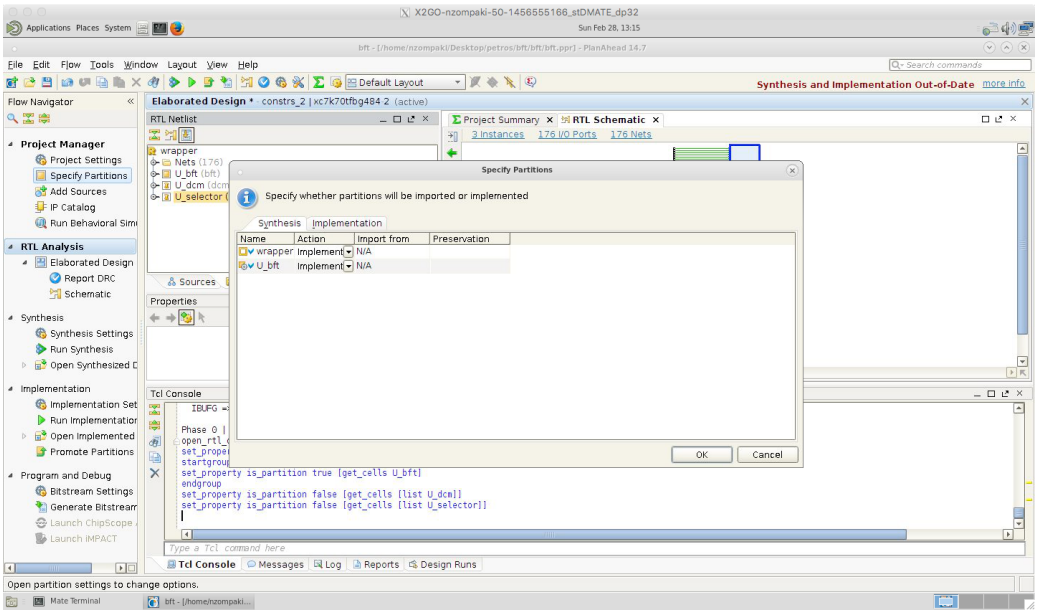
23: Partitions Overview

18. Right click on the name of the original circuit and select “Set Partition”. After that, new options will appear in Flow navigator.



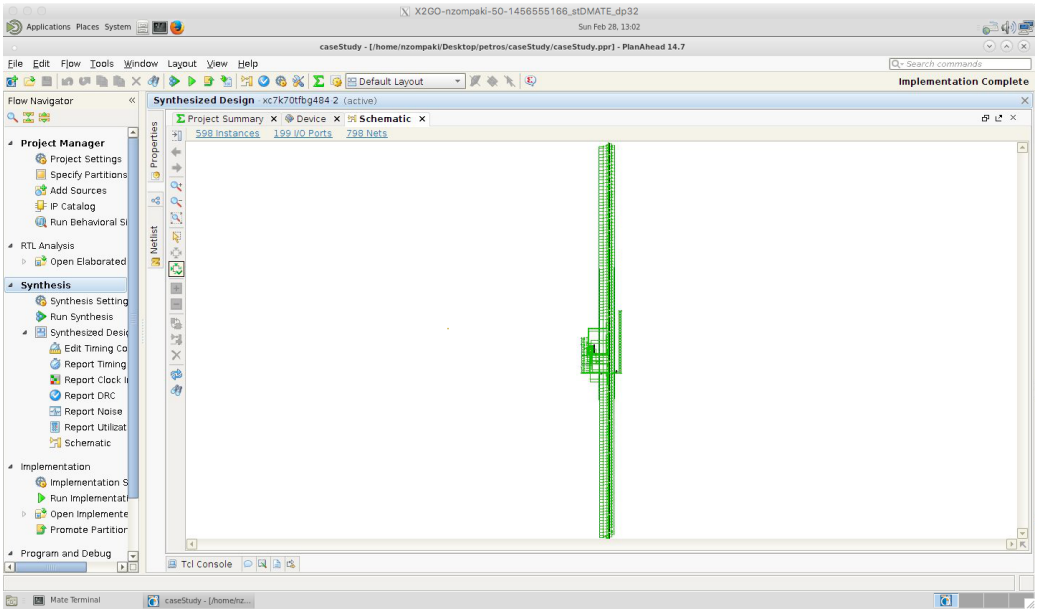
24: Setting a Partition

19. Click on “Specify partition” option from flow navigator and make sure that all actions are set to “implement” for both synthesis and implementation tabs.



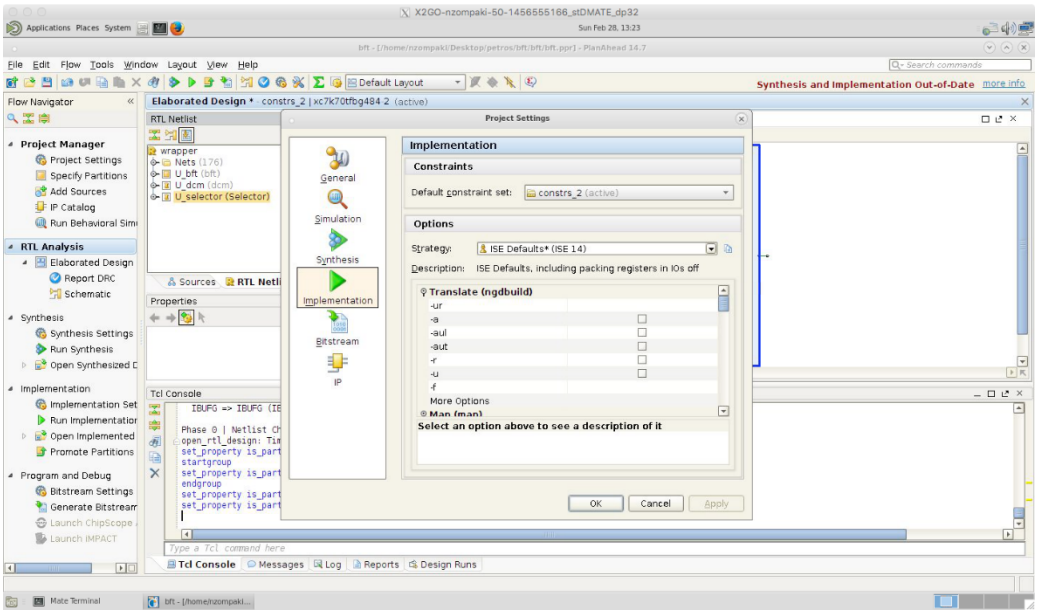
25: Using Partitions

20. Click on “Run Synthesis” from flow navigator. Depending on the circuit, the time for synthesis may be quite long. User can check the progress from the upper right progress bar and from the Tcl console of Planahead.
21. (Optional) After synthesis is complete, click on “Open synthesized design” and then “Schematic” from flow navigator to inspect how the circuit will be implemented on the FPGA.



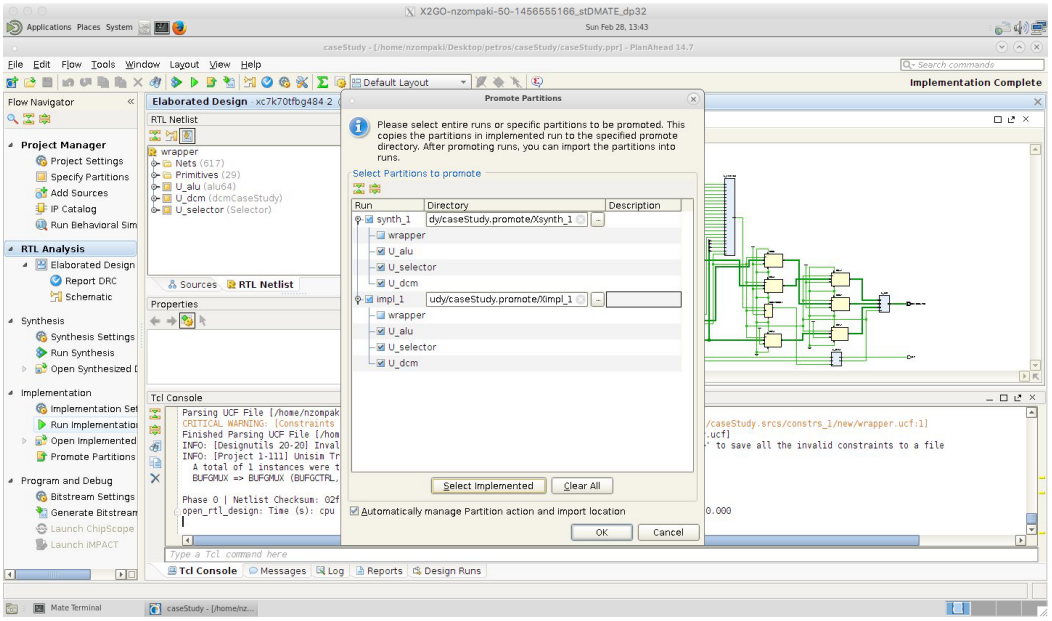
26: Synthesis schematic

22. After synthesis is complete, click on “Implementation settings” from flow navigator to see all available settings. User can make changes although default settings are satisfactory.



27: Implementation Settings

23. Click on “Run implementation” from flow navigator. Depending on the circuit, the time for implementation may be quite long. User can check the progress from the upper right progress bar and from the Tcl console of Planahead.
24. After implementation is complete, click on “promote partitions” from flow navigator. A new window will appear.

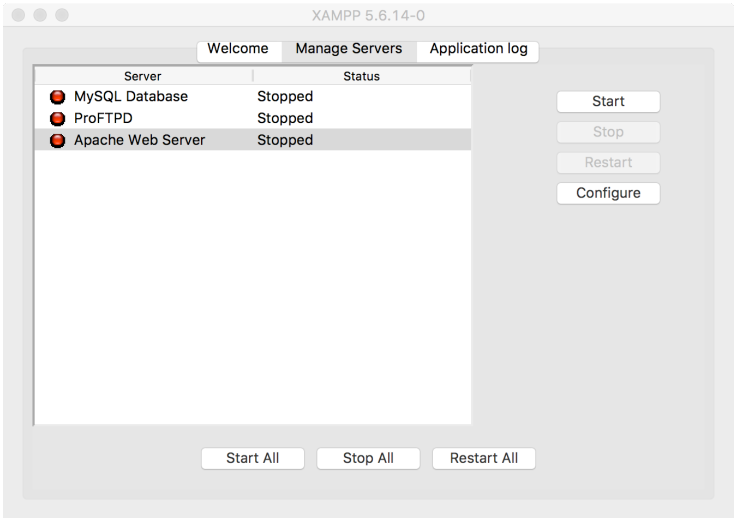


28: Promoting Partitions

25. Make sure to select everything from both synthesis and implementation except wrapper. Click “OK”.

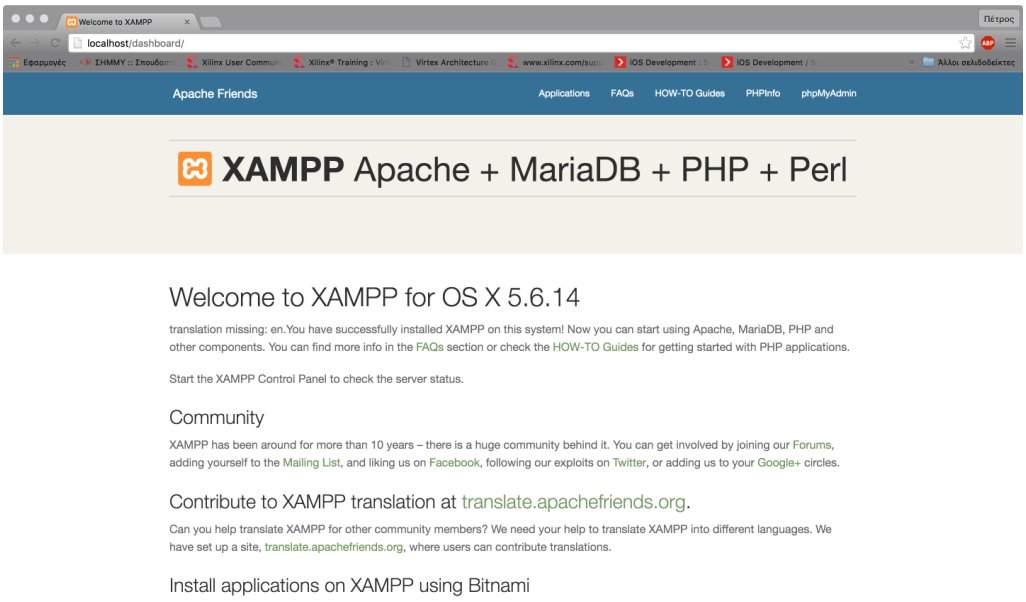
A.2 Analyzing the circuit using Planahead Expander

Expander requires some additional software to function properly. User must have installed both Eclipse and Xampp. Expander is written in Java and runs as an application through Eclipse and uses MySQL server found in Xampp. After successful installation, open Xampp and click “manage servers”. Turn on both MySQL database and Apache web server.



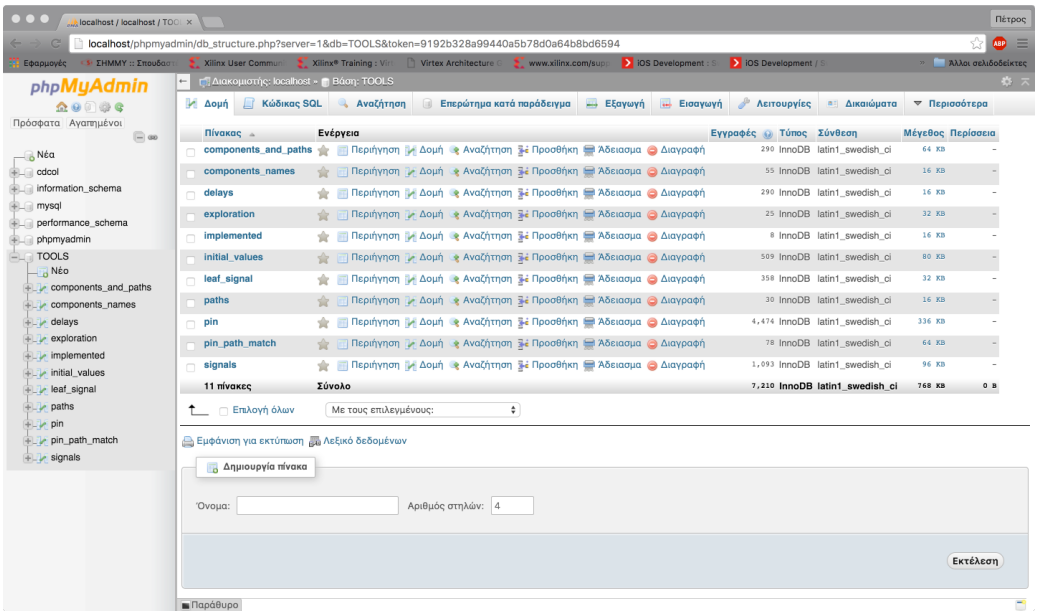
29: Xampp main window

The apache web server is optional and is used to check the information stored into the database that Expander creates. This can be done by visiting “localhost” with a web browser. A page similar to the next one should appear if everything runs fine.



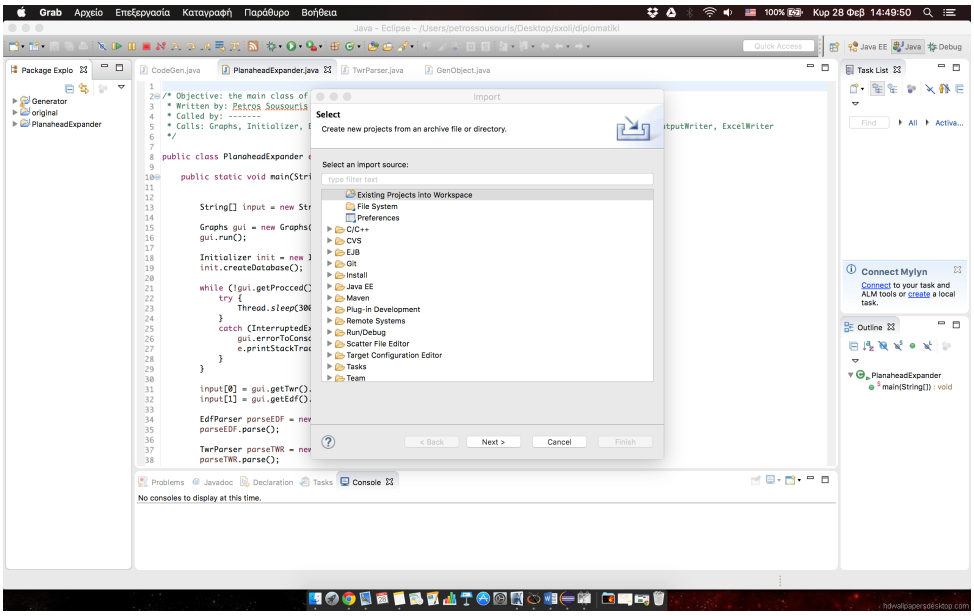
30: localhost main screen

Database is located after clicking “phpmyAdmin” on the upper right side. On the new window that appears, on the left side there will be a database with name “tools”. This is the database that Expander uses.

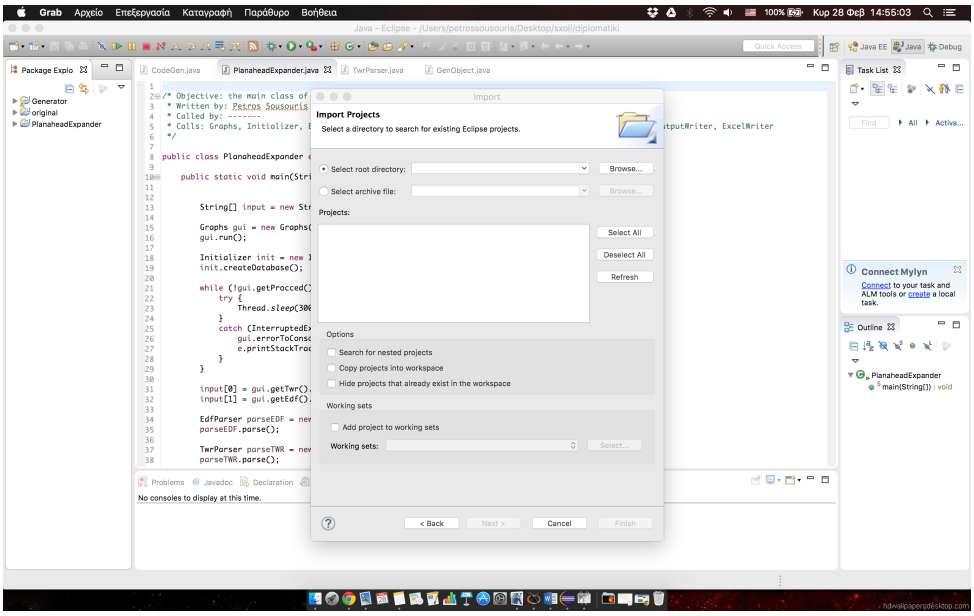


31: Database overview

Now the java source files must be imported to Eclipse so that the can be executed. Launch Eclipse and select as workspace the directory that user wants. Go to file and then choose import. A new window opens. Select "Existing project into workspace" and click next.



32: Importing Expander in Eclipse (1)

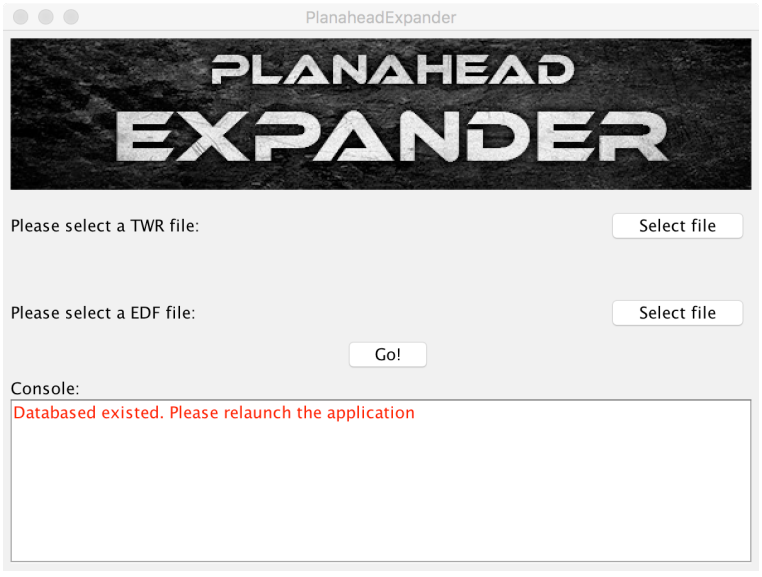


33: Importing Expander in Eclipse (2)

Select the option “Select root directory” and browse to the location that the folder of Expander and Generator are stored. After that, click Next and then finish. Now both Expander and Generator are ready to be launched.

Expander needs two files which were created before in step A1. Files with extensions edf and twr were created after implementation. User should locate both of these files and copy them into the folder that Eclipse uses as workspace.

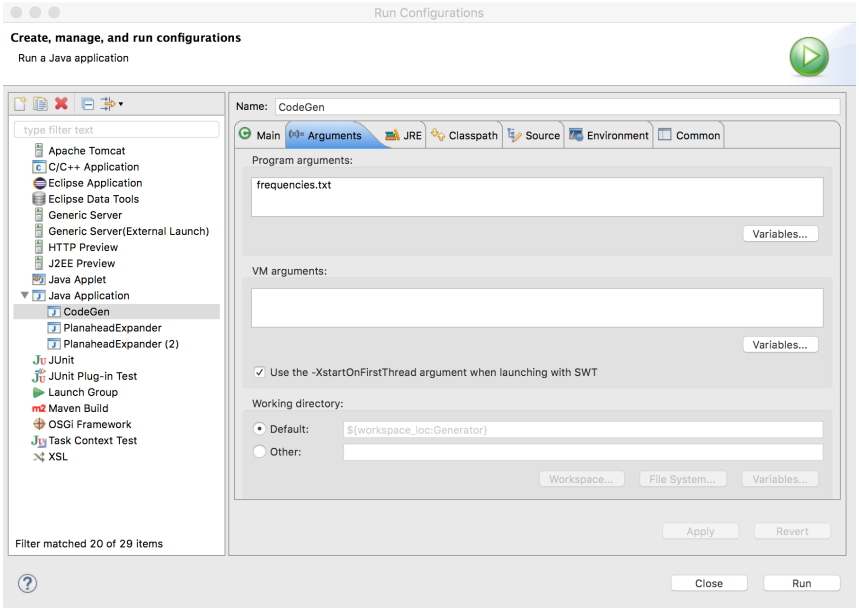
Launch Expander by selecting run, run as, java application. The main window of Expander launches and is ready for usage. If database existed, Expander will delete it and will require a restart. In console Expander will print messages to inform user about the processes and the state of the application. More detailed error messages are displayed on the console of Eclipse.



34: Expander main window

After Expander terminates normally, some output files will have been created. Most of these files are for user information and only one is required for the next step. This file has name “frequrncies.txt” and contains all the names of the control signals with their maximum operating frequency. This file needs some manual editing before proceeding. The frequencies must be grouped according to user’s wishes and then sorted in ascending order.

When manual editing is done, Generator is ready to be launched. First, save the edited text file into the directory that the source code of Generator is saved. Generator also runs inside Eclipse and needs an input argument which is the file name. Select the source file “CodeGen.java” and go to run, run configuration. A new window will open and will look like the next picture.



35: Setting Input Arguments

Select the arguments tab and into the field labeled “program arguments” type in the name of the manually edited file. Click apply and then run. After Generator terminates successfully, into the current workspace a file entitled “Selector.vhd” will have been created. This file contains the source code that will be inserted into the project in Planahead and will compose the second component of the circuit.

For the next step, Planahead is going to be used once again. User can close Eclipse and shut down the MySQL and Apache web servers.

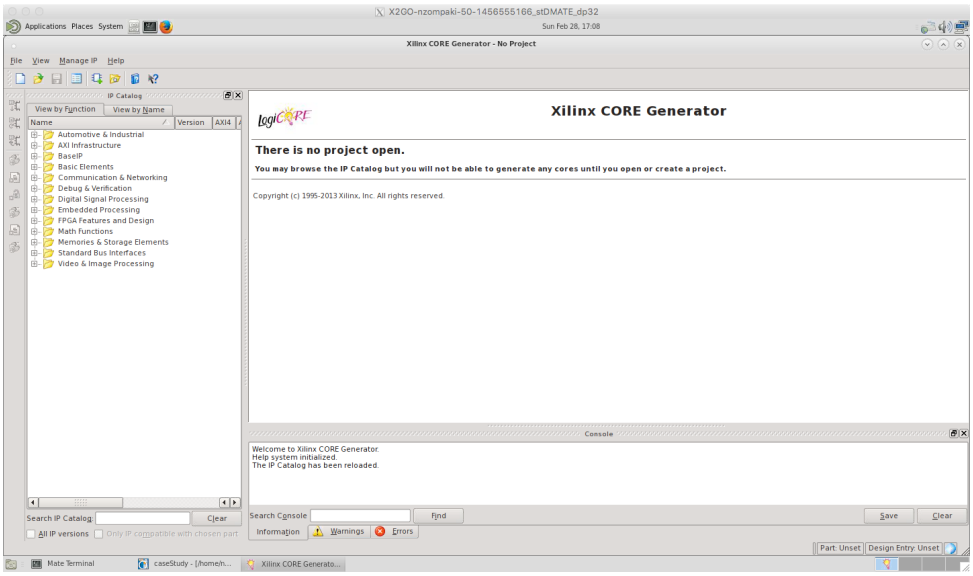
A.3 Inserting Selector into the project

1. In the previous project select “Add sources” from flow navigator and then choose “add or create design files”. Then click “add files” and import the file with title “Selector.vhd”.

2. Once the file is imported, Planahead will update the hierarchy of the design. For now, Selector must be at the same hierarchy level with wrapper. Update the wrapper file so that it uses an instance of Selector and save the changes. After saving is done, the hierarchy is updated and Selector should be below the wrapper.
3. Click on “open elaborated design” in flow navigator and in the sources panel select the “RTL netlist” tab.
4. Right click on the selector and select “set partition”.

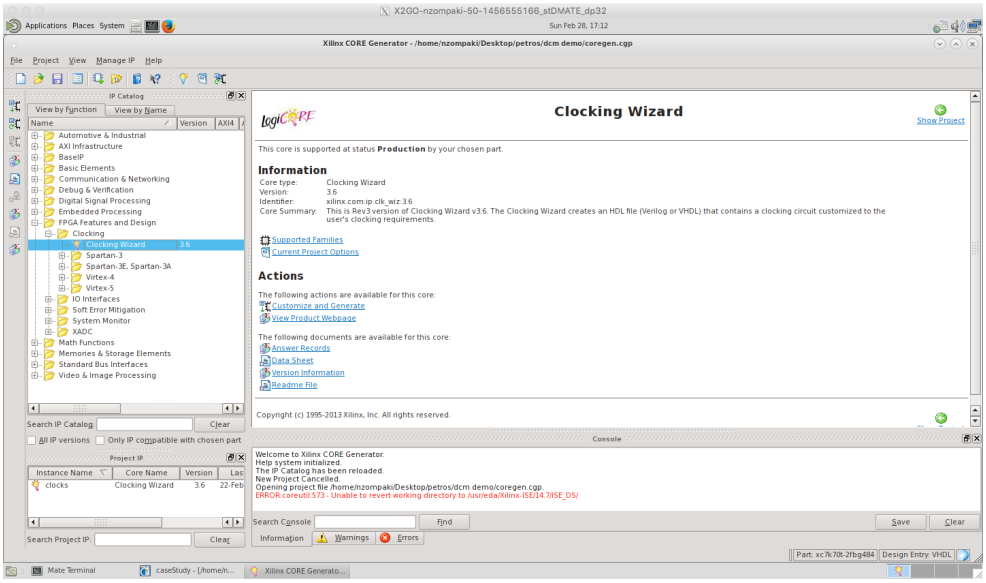
A.4 Building the digital clock manager

1. From flow navigator select the option IP catalog. Alternatively, type the command `coregen -J Xmx1024m` into the tcl console of Planahead. A new window appears.



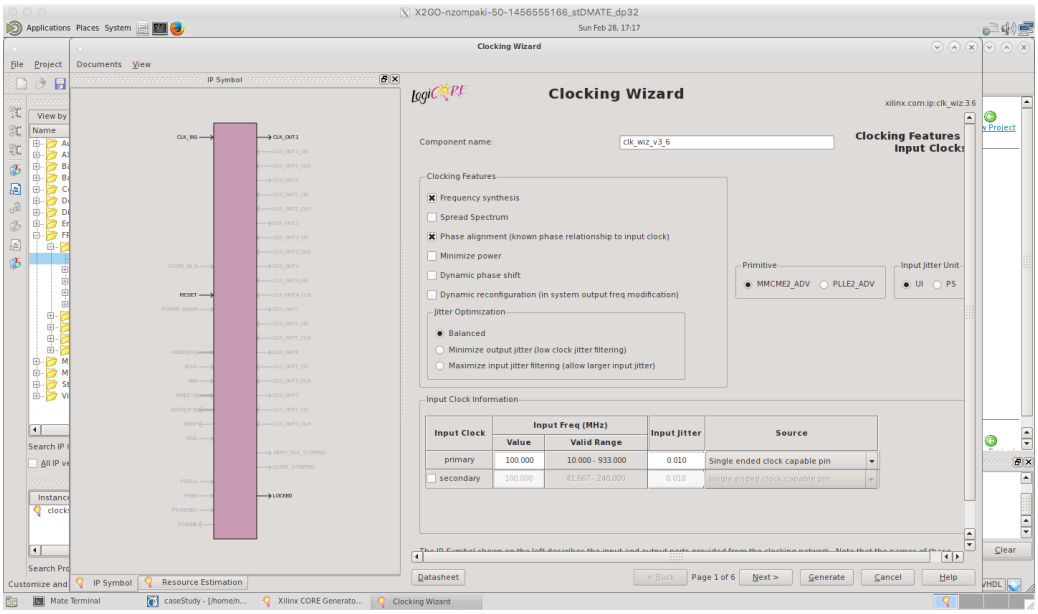
36: Opening Screen of Core Generator

2. Create a new project by clicking File, new project.
3. After the creation of the project, on the left list of options select FPGA features and design, clocking, clocking wizard. Double click that option



37: Locating Clocking Wizard

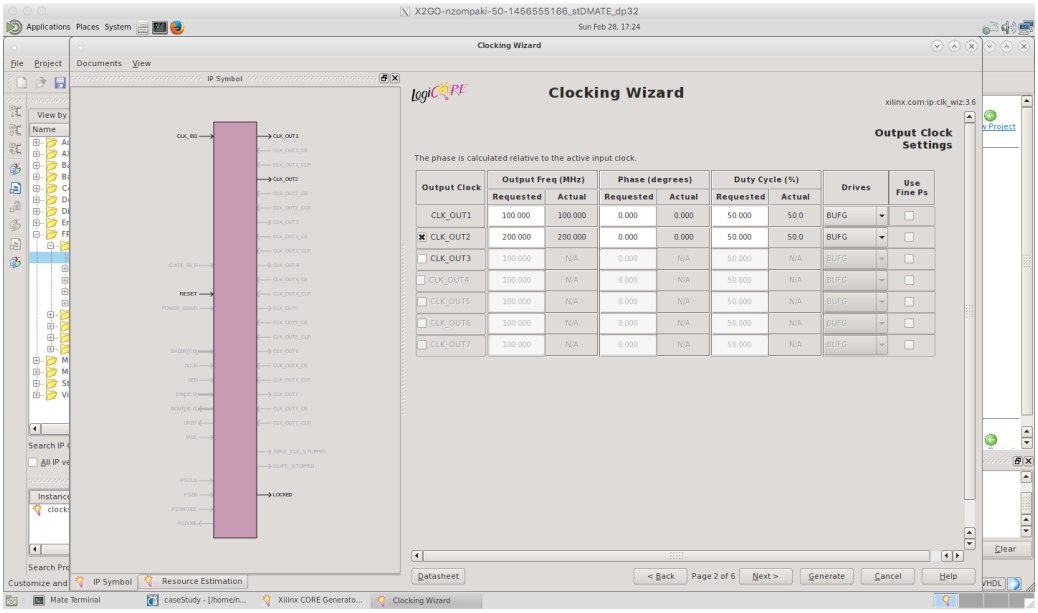
Clocking wizard opens.



38: Defining input frequency

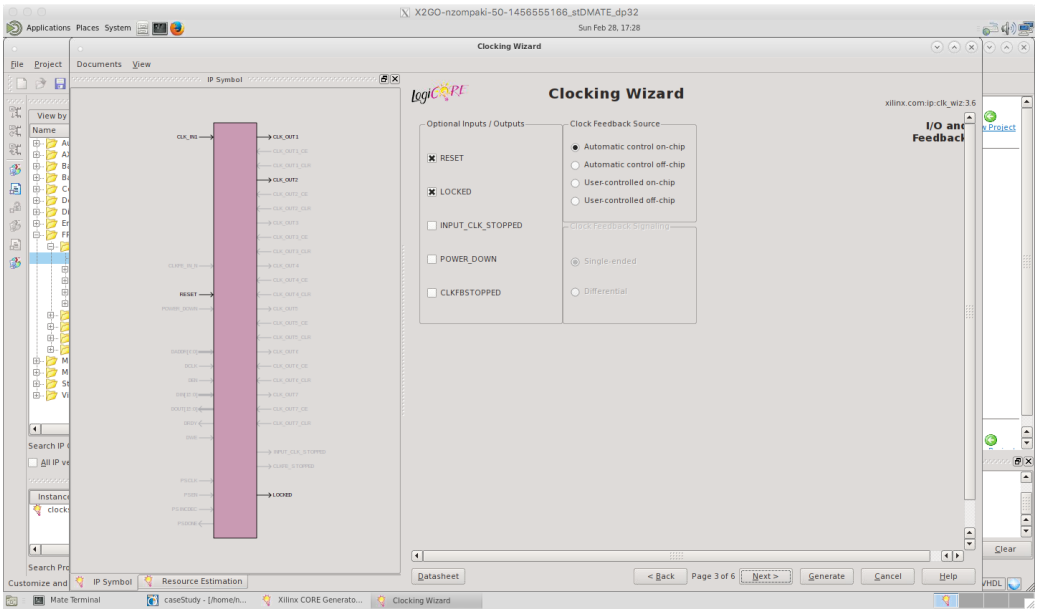
4. Insert a name in the “component name” field and a frequency of input clock in the “primary” field. Click next.

The higher input frequency the more accurately the digital clock manager will produce the frequencies specified by user.



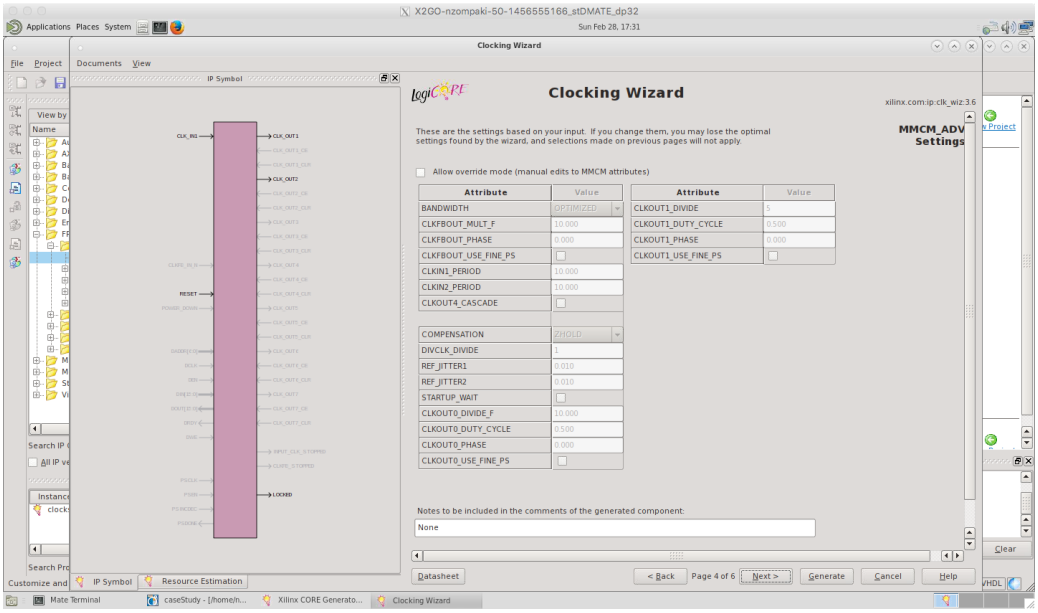
39: Defining output frequencies

5. In the above window, user can define the output frequencies that wishes. Please note that the digital clock manager may not produce the desired frequencies accurately (for example frequencies are too close in value). Click next after you define as many outputs as desired.



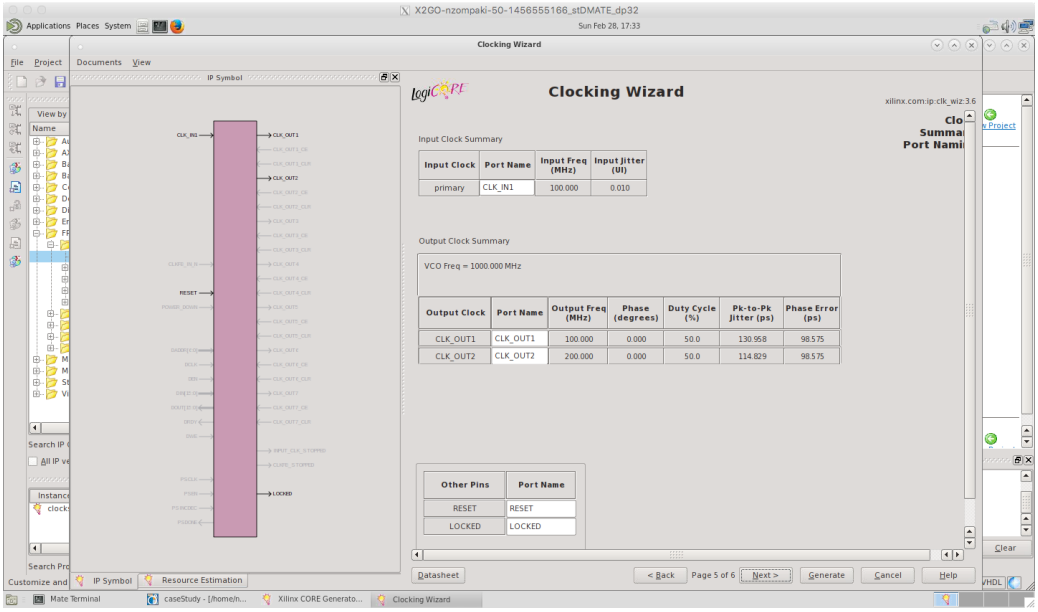
40: Optional pins

6. In this page, select some of the optional input and output signals but leave the “clock feedback source” at its default value.



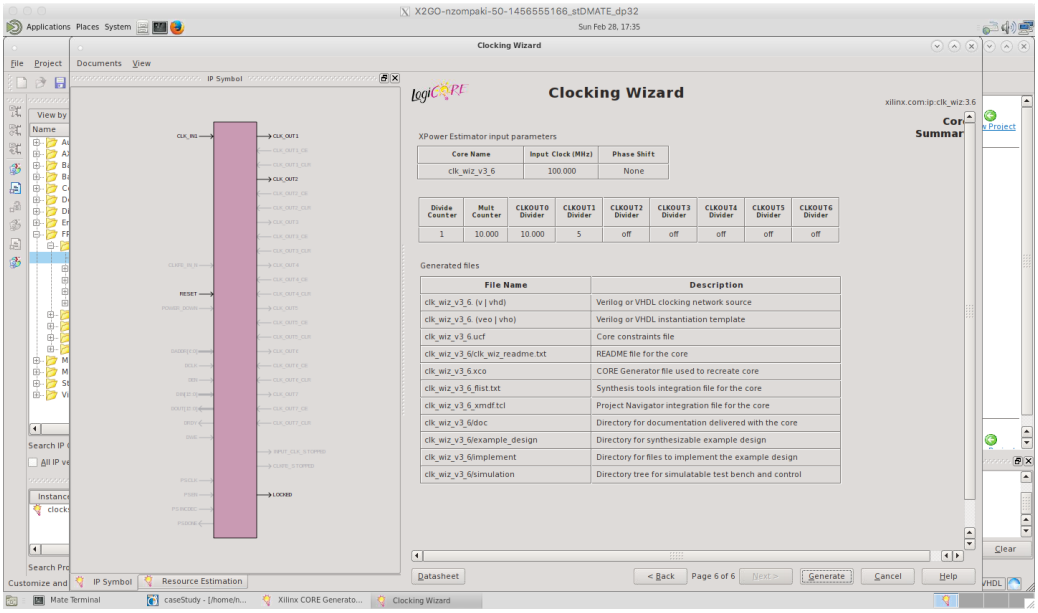
41: Other options

7. Make no changes at the above page.



42: Renaming options

8. (Optional) Rename the input and output pins



43: Settings check

9. Check all the options selected to verify that everything is according to the specifications of the design. If everything is fine, click “generate”.

Clocking wizard creates many files. For the circuit in Planahead, only two of them are needed. Those files are the VHDL file of the digital clock manager (.vhd) and the constrains file (.ucf). Both of them will be inserted in the project of Planahead.

The VHDL file, however, needs some manual modifications because by default the output clocks of the digital clock manager are connected to buffers. It is required that those output clocks will be connected to special multiplexers called “bufgmux”. This connections are done by modifying VHDL code and some examples can be found in the code section of Appendix C. Modify the “entity” declaration of the digital clock manager by inserting a std_logic (or vector) input signal for the selection signal (this must be the same type with the signal output of the selector), delete the output clocks and replace them with one unique output port. At the end of the file, delete the commands that drive output clocks into buffers and replace them with a line of code:

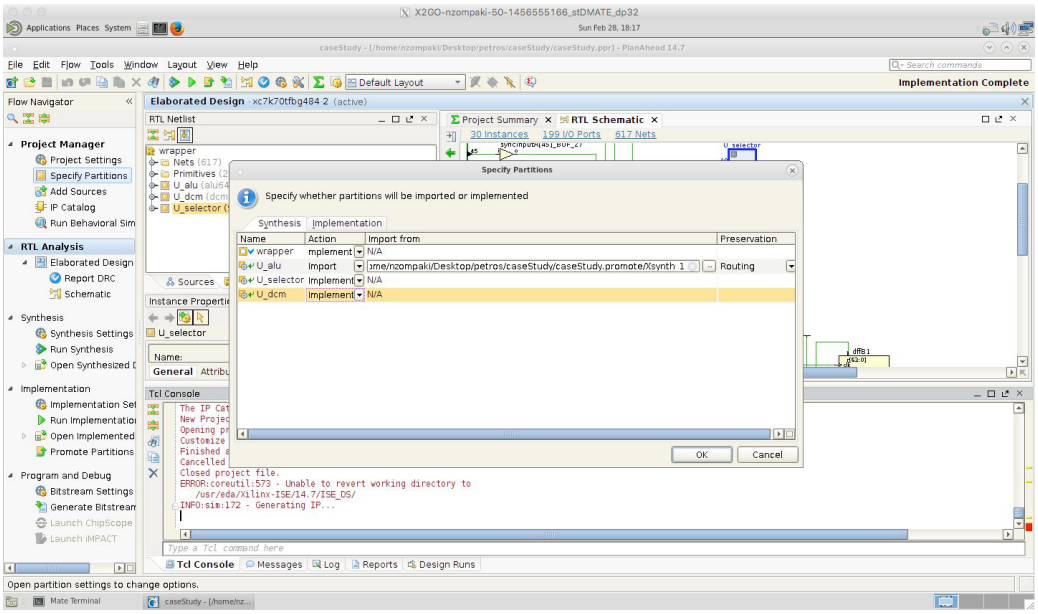
```
Mux : bufgmux port map (select signal, inputs, output);
```

A.5 Inserting the digital clock manager

1. In the previous project select “Add sources” from flow navigator and then choose “add or create design files”. Then click “add files” and import the file with extension .vhd.
2. Once the file is imported, Planahead will update the hierarchy of the design. For now, Selector must be at the same hierarchy level with wrapper. Update the wrapper file so that it uses an instance of digital clock manager by coping the entity declaration into wrapper and changing “entity” keyword to “component”. Create an instance of the

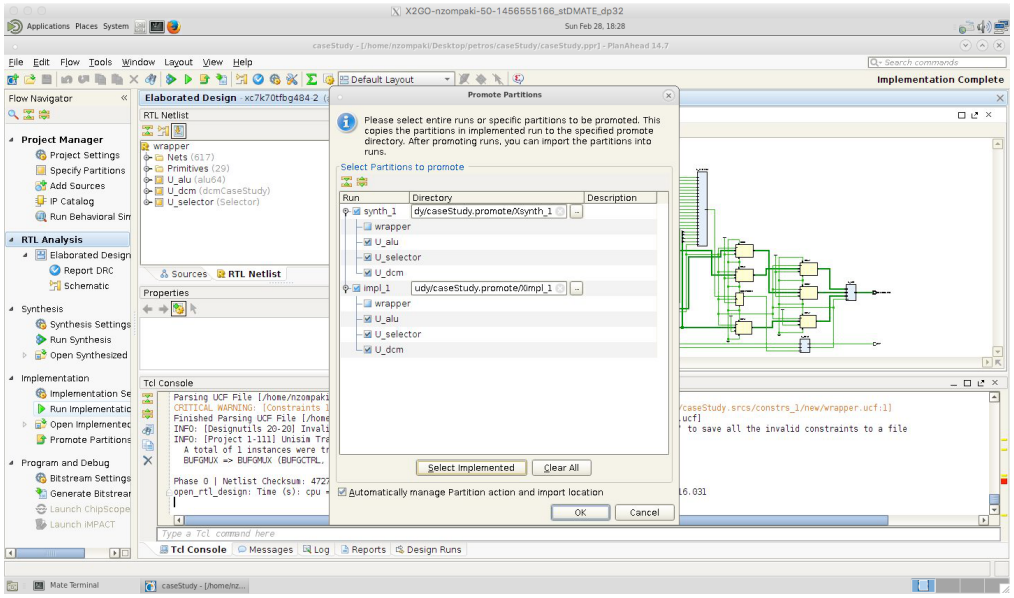
dcm just like the other instances and connect to its pins the proper signals. After saving is done, the hierarchy is updated and digital clock manager should be below the wrapper. The wrapper not only instantiates the different units but also connects the input and output pins of the components via VHDL code. This is achieved, however, only for the external signals of the main circuit that selector monitors. For the internal signals the connections will be made using FPGA Editor.

3. Click on “open elaborated design” in flow navigator and in the sources panel select the “RTL netlist” tab.
4. Right click on the selector and select “set partition”.
5. Click the option “specify partitions” and a new window appears.



44: Importing and implementing partitions

6. All actions should be set to “implement” for both synthesis and implementation tabs except for the original circuit which must remain in “import” action in both tabs.
7. Click on the “run synthesis” from flow navigator.
8. Click on the “run implementation” from flow navigator after synthesis is completed.
9. Click on “promote partitions” and select all options instead of wrapper.

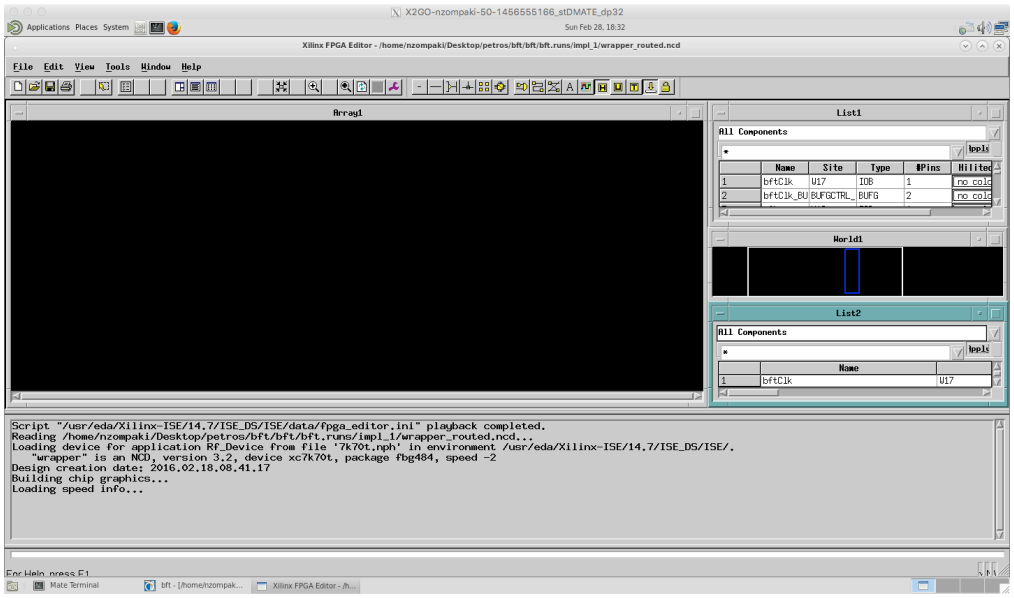


45: Promoting partitions

10. If no internal signals exist, skip section A.6

A.6 Connecting internal signals using FPGA Editor

1. After implementation is completed, click on "open implemented design" and then on "FPGA Editor". A new window appears.



46: FPGA Editor main screen

2. In the command section of FPGA editor type "setattr main edit-mode toggle" to enable editing the design.
3. From list 1 select the name of the component that will be connected. In case that extensive renaming has been done by Planahead, you can return to Planahead and click "open synthesized design" and then "schematic". This will open the design and will help user locate the

signals. The names used in synthesis are the same used by FPGA Editor.

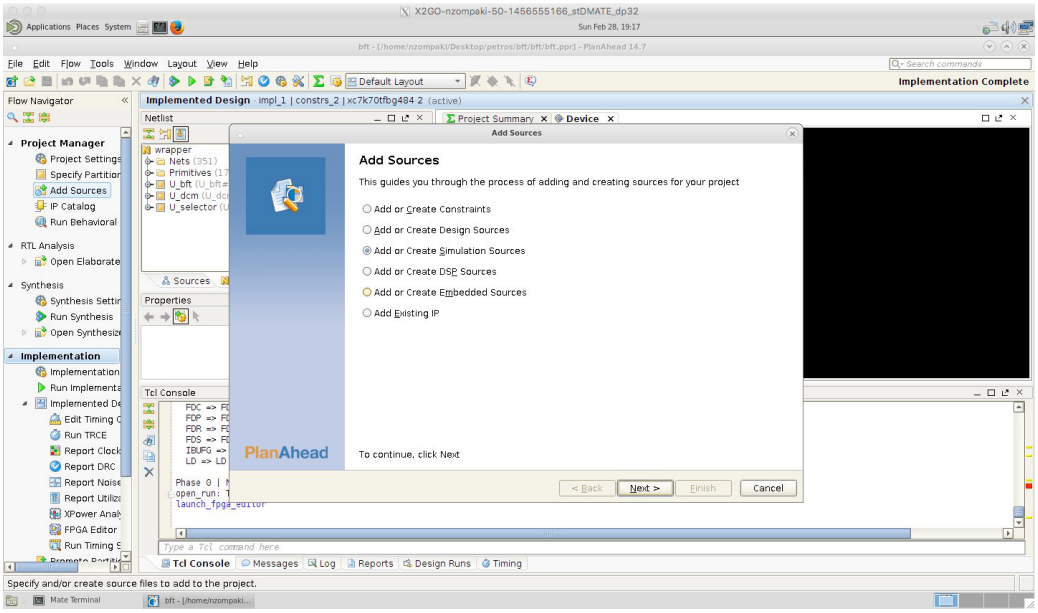
4. From list 2 select the name of the signal that is going to be driven to a specific pin of the component specified. User can locate the names of the signals just like in the previous step.
5. Click on the pin of the component and while holding down the control key click on the name of the signal. Both the signal and the pin of the component must be selected.
6. Type in the command section "route". If an error appears saying that "nothing found to route", then ironically everything is fine.
7. Type in the command section "autoroute". Some messages will appear informing the user that everything went fine.
8. Repeat that procedure as many times as required in order to connect all signal from the main circuit to the inputs of the selector circuit.

It is worth mentioning that changes made with FPGA Editor are not visible in Planahead. For example, the schematic option under Synthesis will not show the changes done. The only way to verify that connections were done correctly is by running the simulator and checking the waveforms.

Please note that there is an online video tutorial for FPGA Editor. Just search for "FPGA Editor" on www.youtube.com.

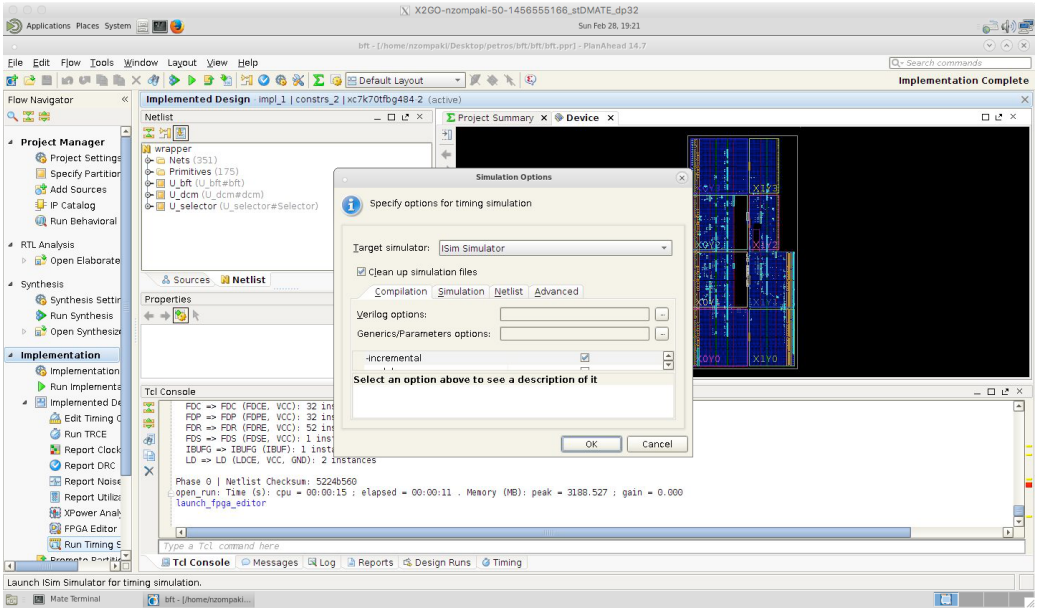
A.7 Simulating the new design

1. Before simulating the circuit, it needs some delay balancing. This will be possible after inserting two rows of D flip flops before the entrance of the original circuit (to balance the two clock cycles delay of selector) and one row of the same components right after the selector output. By defining a signal in VHDL code (which will be the output of the Ds) and by changing their values like "output <= input" in a process which has the clock into its sensitivity list, a D flip flop is created. (Please note that the number of rows of flip flops required depends on the original circuit. However, selector always delays for two clock cycles and the digital clock manager functions asynchronously).
2. Click on the "add sources" from flow navigator an then select "add or create simulation files".



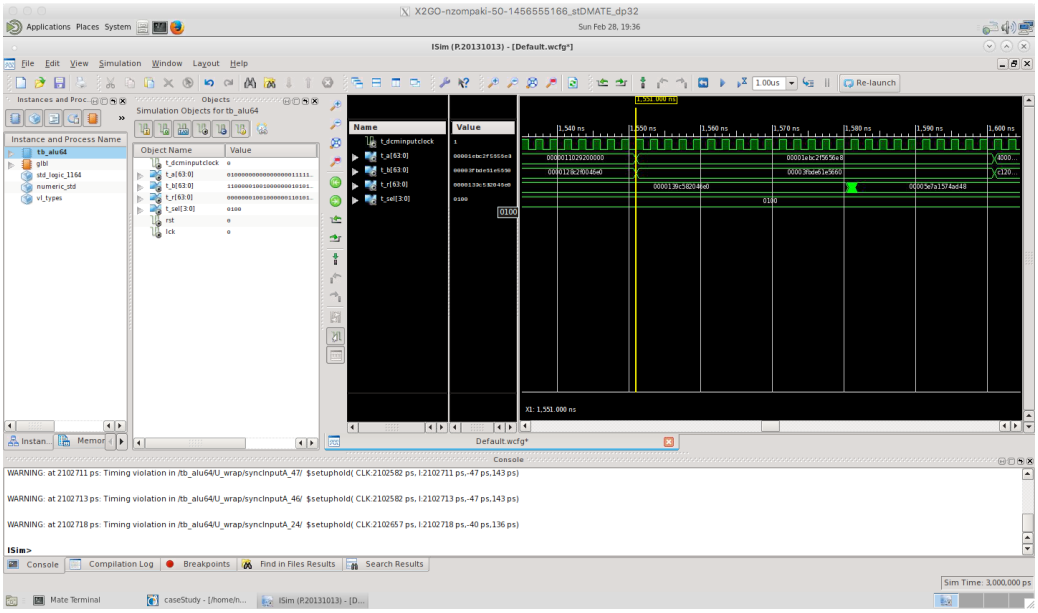
47: Add simulation sources

3. Click “add files” and insert a testbench file written in Verilog or VHDL. Example code of a testbench file written in VHDL can be found in appendix C.
4. After the file is imported, click “run timing simulation”. A new window appears with some settings. User can edit the default settings.



48: Simulator settings

5. Click “OK” and the simulation window will appear. User can add or remove signals in order to verify that the new circuit functions properly.



49: Simulator window

Appendix B

B.1 Edf file explanation

EDIF (Electronic Design Interchange Format, used as edf by Xilinx) is a vendor-neutral format in which to store Electronic netlists and schematics. It was one of the first attempts to establish a neutral data exchange format for the electronic design automation (EDA) industry. The goal was to establish a common format from which the proprietary formats of the EDA systems could be derived. When customers needed to transfer data from one system to another, it was necessary to write translators from one format to other. As the number of formats (N) multiplied, the translator issue became an N-squared problem. The expectation was that with EDIF the number of translators could be reduced to the number of involved systems.

Representatives of the EDA companies Daisy Systems, Mentor Graphics, Motorola, National Semiconductor, Tektronix, Texas Instruments and the University of California, Berkeley established the EDIF Steering Committee in November 1983. Later Hilary Kahn, a computer science professor at the University of Manchester, joined the team and led the development from version EDIF 2 0 0 till the final version 4 0 0.

The general format of EDIF involves using parentheses to delimit data definitions, and in this way it superficially resembles Lisp. The basic tokens of EDIF 2.0.0 were keywords (like library, cell, instance, etc.), strings (delimited with double quotes), integer numbers, symbolic constants (e.g. GENERIC, TIE, RIPPER for cell types) and "Identifiers", which are reference labels formed from a very restricted set of characters. EDIF 3.0.0 and 4.0.0 dropped the symbolic constants entirely, using keywords instead. So, the syntax of EDIF has a fairly simple foundation. A typical EDIF file looks like this:

```
(edif wrapper
  (edifversion 2 0 0)
  (edifLevel 0)
  (keywordmap (keywordlevel 0))
(status
 (written
  (timeStamp 2016 02 21 11 22 29)
  (program "PlanAhead" (version "14.7"))
  (comment "Built on 'Fri Sep 27 19:24:36 MDT 2013'")
  (comment "Built by 'xbuild'")
 )
)
  (Library hdi_primitives
    (edifLevel 0)
    (technology (numberDefinition ))
   (cell IBUF (celltype GENERIC)
     (view netlist (viewtype NETLIST)
       (interface
        (port O (direction OUTPUT))
```

```
        (port I (direction INPUT))
       )
      )
   )
(Library U_alu_alu64_lib
    (edifLevel 0)
    (technology (numberDefinition ))
   (cell (rename U_alu_alu64 "U_alu#alu64") (celltype GENERIC)
     (view view_1 (viewtype NETLIST)
       (interface
        (port clk (direction INPUT))
        (port (array (rename A "A[63:0]") 64) (direction INPUT))
        (port (array (rename B "B[63:0]") 64) (direction INPUT))
        (port (array (rename S "S[3:0]") 4) (direction INPUT))
        (port (array (rename Result "Result[63:0]") 64) (direction
                                                        OUTPUT))
       )
       (contents
         (instance T_R_0 (viewref netlist (cellref FD (libraryref
                                                  hdi_primitives)))
           (property XILINX_REPORT_XFORM (string "FD"))
           (property XSTLIB (boolean (true)))
           (property INIT (string "1'b0"))
         )
         (instance T_R_1 (viewref netlist (cellref FD (libraryref
                                                  hdi_primitives)))
           (property XILINX_REPORT_XFORM (string "FD"))
           (property XSTLIB (boolean (true)))
           (property INIT (string "1'b0"))
         )
(net (rename Mmux_S_3__B_63__wide_mux_20_OUT15_split_63_
"Mmux_S[3]_B[63]_wide_mux_20_OUT15_split[63]") (joined
          (portref O (instanceref
Mmux_S_3__B_63__wide_mux_20_OUT15121))
          (portref D (instanceref T_R_63))
         )
        )
```

The 1.0.0 release of EDIF was made in 1985.

EDIF 2.0.0

The first "real" public release of EDIF was version 2 0 0, which was approved in March 1988 as the standard ANSI/EIA-548-1988. It is published in a single volume. This version has no formal scope statement but what it tries to capture is covered by the defined viewTypes:

- **BEHAVIOR** to describe the behavior of a cell
- **DOCUMENT** to describe the documentation of a cell
- **GRAPHIC** to describe a dumb graphics and text representation of displayable or printable information
- **LOGICMODEL** to describe the logic-simulation model of the cell
- **MASKLAYOUT** to describe an integrated circuit layout
- **NETLIST** to describe a netlist
- **PCBLAYOUT** to describe a printed circuit board
- **SCHEMATIC** to describe the schematic representation and connectivity of a cell

- **STRANGER** to describe an as yet unknown representation of a cell
- **SYMBOLIC** to describe a symbolic layout

The industry tested this release for several years, but finally only the NETLIST view was the one widely used and some EDA tools are still supporting it today for EDIF 2.0.0. (EDIF Overview, 2005)

To overcome problems with the main 2.0.0 standard several further documents got released:

- Electronic Industries Association
- EDIF Monograph Series, Volume 1, Introduction to EDIF, EIA/EDIF-1, Sept. 1988
- EDIF Monograph Series, Volume 2, EDIF Connectivity, EIA/EDIF-2, June 1989
- Using EDIF 2 0 0 for schematic transfer, EIA/EDIF/AG-1, July 1989
- Documentation from Hilary J. Kahn, Department of Computer Science, University of Manchester
- EDIF 2 0 0, An Introductory Tutorial", September 1989
- EDIF Questions and answers, volume one, November 1988
- EDIF Questions and answers, volume two, February 1989
- EDIF Questions and answers, volume three, July 1989
- EDIF Questions and answers, volume four, November 1989
- EDIF Questions and answers, volume five, June 1991

EDIF 3.0.0

Because of some fundamental weaknesses in the 2.0.0 release a new not compatible release 3.0.0 was released in September 1993, given the designation of EIA standard EIA-618. It later achieved ANSI and ISO designations. It is published in 4 volumes. The main focus of this version were the viewTypes NETLIST and SCHEMATIC from 2.0.0. MASKLAYOUT, PCBLAYOUT and some other views were dropped from this release and shifted for later releases because the work for these views was not fully completed.

EDIF 3.0.0 is available from the International Electrotechnical Commission as IEC 61690-1

EDIF 4.0.0

EDIF 4.0.0 was released in late August 1996, mainly to add "Printed Circuit Board" extensions (the original PCBLAYOUT view) to EDIF 3.0.0. This more than doubled the size of EDIF 3.0.0, and is published in HTML format on CD.

EDIF 4.0.0 is available from the International Electrotechnical Commission as IEC 61690-2.

Problems with 2.0.0

To understand the problems users and vendors encountered with EDIF 2.0.0, one first has to picture all the elements and dynamics of the

electronics industry. The people who needed this standard were mainly design engineers, who worked for companies whose size ranged from a house garage to multi-billion dollar facilities with thousands of engineers. These engineers worked mainly from schematics and netlists in the late 1980s, and the big push was to generate the netlists from the schematics automatically. The first suppliers were Electronic Design Automation vendors (e.g., Daisy, Mentor, and Valid formed the earliest predominating set). These companies competed vigorously for their shares of this market.

One of the tactics used by these companies to "capture" their customers was their proprietary databases. Each had special features that the others did not. Once a decision was made to use a particular vendor's software to enter a design, the customer was ever after constrained to use no other software. To move from vendor A's to vendor B's systems usually meant a very expensive re-entry of almost all design data by hand into the new system. This expense of "migration" was the main factor that locked design engineers into using a single vendor.

But the "customers" had a different desire. They saw immediately that while vendor A might have a really nice analog simulation environment, vendor B had a much better PCB or silicon layout auto-router. And they wished that they could pick and choose amongst the different vendors.

EDIF was mainly supported by the electronics design end-users, and their companies. The EDA vendors were involved also, but their motivation was more along the lines of wanting to not alienate their customers. Most of the EDA vendors produced EDIF 2.0.0 translators, but they were definitely more interested in generating high-quality EDIF readers, and they had absolutely no motivation at all to write any software that generated EDIF (an EDIF Writer), beyond threats from customers of mass migration to another vendor's software.

The result was rather interesting. Hardly any software vendor wrote EDIF 2.0.0 output that did not have severe violations of syntax or semantics. The semantics were just loose enough that there might be several ways to describe the same data. This began to be known as "flavors" of EDIF. The vendor companies did not always feel it important to allocate many resources to EDIF products, even if they sold a large number of them. There were several stories of active products with virtually no-one to maintain them for years. User complaints were merely gathered and prioritized. The harder it became to export customer data to EDIF, the more the vendors seemed to like it. Those who did write EDIF translators found they spent a huge amount of time and effort on generating sufficiently powerful, forgiving, artificially intelligent readers, that could handle and piece together the poor-quality code produced by the extant EDIF 2.0.0 writers of the day.

In designing EDIF 3.0.0, the committees were well aware of the faults of the language, the calumny heaped on EDIF 2.0.0 by the vendors and the frustration of the end users. So, to tighten the semantics of the language, and provide a more formal description of the standard, the revolutionary

approach was taken to provide an information model for EDIF, in the information modeling language EXPRESS. This helped to better document the standard, but was done more as an afterthought, as the syntax crafting was done independently of the model, instead of being generated from the model. Also, even though the standard says that if the syntax and model disagree, the model is the standard, this is not the case in practice. The BNF description of the syntax is the foundation of the language inasmuch as the software that does the day-to-day work of producing design descriptions is based on a fixed syntax. The information model also suffered from the fact that it was not (and is not) ideally suited to describing EDIF. It does not describe such concepts as name spaces very well at all, and the differences between a definition and a reference is not clearly describable either. Also, the constructs in EXPRESS for describing constraints might be formal, but constraint description is a fairly complicated matter at times. So, most constraints ended up just being described as comments. Most of the others became elaborate formal descriptions which most readers will never be able to decipher, and therefore may not stand up to automated debugging/compiling, just as a program might look good in review, but a compiler might find some interesting errors, and actually running the program written might find even more interesting errors.

Solutions to edif 2.0.0 problems

The solution to the "flavor" problem of EDIF 2.0.0 was to develop a more specific semantic description in EDIF 3.0.0 (1993). Indeed, reported results of people generating EDIF 3.0.0 translators was that the writers were now much more difficult to get right, due to the great number of semantic restrictions, and the readers are comparatively trivial to develop.

The solution to vendor "conflict of interest" was neutral third-party companies, who could provide EDIF products based on vendor interfaces. This separation of the EDIF products from direct vendor control was critical to providing the end-user community with tools that worked well. It formed naturally and without comment. Engineering DataXpress was perhaps the first such company in this realm, with Electronic Tools Company seeming to have captured the market in the mid to late 1990s. Another dynamic in this industry is EDIF itself. Since they have grown to a rather large size, generating readers and writers has become a very expensive proposition. Usually the third-party companies have congregated the necessary specialists and can use this expertise to more efficiently generate the software. They are also able to leverage code sharing and other techniques an individual vendor could not. By 2000, almost no major vendor produced its own EDIF tools, choosing instead to OEM third-party tools.

Since the release of EDIF 4.0.0, the entire EDIF standards organisation has essentially dissolved. There have been no published meetings of any of the technical subcommittees, the EDIF Experts group, etc. Most of the individuals involved have moved on to other companies or efforts. The newsletter was abandoned, and the Users' Group no longer holds yearly meetings. EDIF 3.0.0 and 4.0.0 are now ANSI, IEC and

European (EN) standards. EDIF Version 3.0.0 is IEC/EN 61690-1, and EDIF Version 4.0.0 is IEC/EN 61690-2. (Guide to EDIF, 2005)

B.2 Twr file explanation

This extension declares the timing report file generated by Xilinx tools. This type of file includes detailed description about the paths analyzed in the circuit and information about their timing behavior, timing violations and constrains verification. Unlike edif files, the timing report format is easily readable and understandable giving information to user about the paths in the design. It is worth mentioning that the content of the timing report depends heavily on the timing constrains that are set by user before synthesis and implementation. The format and structure of the file do not change. A typical example of a timing report (.twr file) looks like the following text document:

```
INFO:Timing:3412 - To improve timing, see the Timing Closure User Guide (UG612).
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
   a 50 Ohm transmission line loading model.  For the details of this model,
   and for more information on accounting for different loading conditions,
   please see the device datasheet.

================================================================================
Timing constraint: OFFSET = IN 1.8 ns BEFORE COMP "clk" "RISING";
For more information, see Offset In Analysis in the Timing Closure User Guide (UG612).

 22249 paths analyzed, 255 endpoints analyzed, 0 failing endpoints
 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum allowable offset is   1.791ns.
--------------------------------------------------------------------------------
Slack:                  0.009ns (requirement - (data path - clock path - clock arrival
+ uncertainty))
  Source:               S[2] (PAD)
  Destination:          T_R_45 (FF)
  Destination Clock:    clk_BUFGP rising
  Requirement:          1.800ns
  Data Path Delay:      3.268ns (Levels of Logic = 15)
  Clock Path Delay:     1.502ns (Levels of Logic = 2)
  Clock Uncertainty:    0.025ns

  Clock Uncertainty:          0.025ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):  0.050ns
    Total Input Jitter (TIJ):   0.000ns
    Discrete Jitter (DJ):       0.000ns
    Phase Error (PE):           0.000ns

  Maximum Data Path at Fast Process Corner: S[2] to T_R_45
    Location             Delay type         Delay(ns)  Physical Resource
                                                       Logical Resource(s)
    -------------------------------------------------  -------------------
    A20.I                Tiopi                 0.396   S[2]
                                                       S[2]
                                                       S_2_IBUF
SLICE_X36Y122.C5     net (fanout=192)   1.026  S_2_IBUF
SLICE_X36Y122.C      Tilo               0.035  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_AS_inv
                                               Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_AS_inv2
SLICE_X23Y101.AX     net (fanout=1)     0.583  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_AS_inv
SLICE_X23Y101.COUT   Taxcy              0.162  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[3]
                                               Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy<3>
SLICE_X23Y102.CIN    net (fanout=1)     0.000  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[3]
SLICE_X23Y102.COUT   Tbyp               0.031  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[7]
                                               Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy<7>
SLICE_X23Y103.CIN    net (fanout=1)     0.000  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[7]
SLICE_X23Y103.COUT   Tbyp               0.031  Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[11]
```

```
                                                       Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy<11>
SLICE_X23Y104.CIN    net (fanout=1)      0.000   Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[11]
SLICE_X23Y104.COUT   Tbyp                0.031   Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy[15]
                                                       Mmux_S[3]_B[63]_wide_mux_20_OUT7_rs_cy<15>
--------------------------------------------------------------------------------------------
Total                                    3.268ns  (1.009ns logic, 2.259ns route)
                                                       (30.9% logic, 69.1% route)
```

Skipping the introductive text, a detailed description and explanation of some of the first properties of the path will be given.

- **Slack**: the first item listed for each path is the slack, which is how much time the path made the constrain by, or in the case of a negative number, how much it is violated by.
- **Source**: the source is the output pin that drives the path.
- **Destination**: the destination is the stopping point of the path.
- **Requirement**: The requirement is the time constrain number.
- **Data path delay**: this line shows the total path delay as well as the number of levels of logic used to implement the timing path.
- **Clock path skew**: The Clock Path Skew is the difference between the time a clock signal arrives at the source flip-flop in a path and the time it arrives at the destination flip-flop. The PAR clock report shows Net Clock Skew. The Net Clock Skew is skew on the clock net. The Clock Path Skew takes the entire clock path into account not just the clock net. This would include the IBUFG delay, net delay to a DCM, delay through a DCM, net delay to global buffer, delay through the global buffer and the clock net delay.
- **Source clock**: The Source Clock is the name of the source clock signal (if any) driving a synchronous source (For Example, FF).
- **Destination clock**: The destination dock is the name of the destination clock signal (if any) driving a synchronous destination (For Example, FF).
- **Clock uncertainty**: The clock uncertainty for an OFFSET constraint might be different than the clock uncertainty on a PERIOD constraint for the same clock. The OFFSET constraint only looks at one clock edge in the equation but the PERIOD constraints takes into account the uncertainty on the clock at the source registers and the uncertainty on the clock at the destination register so that two clock edges are in the equation.

After these items, timing report describes the route that the path has chosen in order to connect the starting and ending point. The file contains information about the slice that it crosses, what type of delay is caused (for example if is delay on a net or due to process in a LUT), how many nanoseconds is the delay and the physical and logical resources of the components that are used inside the slice mentioned before. Finally, the total amount of delay is further analyzed in order to give user more detailed information. In general, the format of the timing report is simple and easy to read. (Timing Analyzer, 2008)

Appendix C – Source Code

In this appendix there will be presented some sample VHDL code so that it made clear how components are connected to each other, how the hierarchical design takes place and how the wrapper file functions and finally how the source code of the digital clock manager, which is automatically generated, can be modified to suit the needs of the project.

C.1 Main circuit of the demo

The main circuit of the demo is a 64 bit arithmetic and logical unit which can perform 16 different operations. This circuit has four inputs; two 64 bit numbers, a 4 bit code which indicates the operation that will take place and the clock. It outputs the final result. The source code of the arithmetic and logic unit is quite simple and self explanatory.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu64 is
port (
      clk:  in std_logic;
      A,B:  in signed(63 downto 0);
      S:          in std_logic_vector(3 downto 0);
      Result: out signed(63 downto 0)
);
end alu64;

architecture Behavioral of alu64 is

      signal T_A, T_B: signed(63 downto 0) := (others => '0');
      signal T_R: signed(63 downto 0) := (others => '0');
      signal T_Sel : std_logic_vector(3 downto 0);

begin

      process(clk)
      begin

            if rising_edge(clk) then

                  T_A <= A;
                  T_B <= B;
                  T_Sel <= S;

                  case T_Sel is
                        when "0000" =>
                                    T_R <= T_A + 1;

                        when "0001" =>
                                    T_R <= T_B + 1;

                        when "0010" =>
                                    T_R <= T_A - 1;
```

```
                        when "0011" =>
                                    T_R <= T_B - 1;

                        when "0100" =>
                                    T_R <= T_A + T_B;

                        when "0101" =>
                                    T_R <= T_A - T_B;

                        when "0110" =>
                                    T_R <= T_A and T_B;

                        when "0111" =>
                                    T_R <= T_A or T_B;

                        when "1000" =>
                                    T_R <= T_A xor T_B;

                        when "1001" =>
                                    T_R <= not T_A;

                        when "1010" =>
                                    T_R <= not T_B;

                        when "1011" =>
                                    if (T_A > T_B) then
                                          T_R <= x"0000000000000001";
                                    elsif (T_A < T_B) then
                                          T_R <= x"0000000000000002";
                                    else
                                          T_R <= x"0000000000000000";
                                    end if;

                        when "1100" =>
                                    T_R <= T_A nand T_B;

                        when "1101" =>
                                    T_R <= T_A nor T_B;

                        when "1110" =>
                                    T_R <= shift_left(T_A,1);

                        when "1111" =>
                                    T_R <= shift_right(T_B,1);

                        when others =>
                                 T_R <= T_R;

                end case;
          end if;
      end process;

      Result <= T_R;


end Behavioral;
```

C.2 Selector of the demo

This is the code generated by the custom tool called Generator. It has inputs that are defined by the main circuit. In this case, it accepts as inputs 21 bits of the inputs of the arithmetic and logical unit that monitor every clock cycle and a clock. It outputs a signal that also depends on the number of different frequencies that the digital clock manages produces. In this demo, only two frequencies exist and as a result Selector outputs a single bit.

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

entity Selector is
port (
    clk : in std_logic;
      B58, B57, A57, A56, S3, S2, S1, S0, B63, A63, B62, A62, B46,
A45, B45, A44, B2, B1, A1, B0, A0 : in std_logic;
      O : out std_logic
);
end Selector;

architecture Behavioral of Selector is

      signal buffB58, buffB57, buffA57, buffA56, buffS3, buffS2,
             buffS1, buffS0 : std_logic;
      signal buffB63, buffA63, buffB62, buffA62, buffB46, buffA45,
             buffB45, buffA44 : std_logic;
      signal buffB2, buffB1, buffA1, buffB0, buffA0 : std_logic;
      signal changeB58, changeB57, changeA57, changeA56, changeS3,
             changeS2, changeS1, changeS0 : std_logic;
      signal changeB63, changeA63, changeB62, changeA62, changeB46,
             changeA45, changeB45, changeA44 : std_logic;
      signal changeB2, changeB1, changeA1, changeB0, changeA0 :
             std_logic;
      signal syncB58, syncB57, syncA57, syncA56, syncS3, syncS2,
             syncS1, syncS0 : std_logic;
      signal syncB63, syncA63, syncB62, syncA62, syncB46, syncA45,
             syncB45, syncA44 : std_logic;
      signal syncB2, syncB1, syncA1, syncB0, syncA0 : std_logic;
      signal temp : std_logic := '0';
      signal low, high : std_logic;

begin

      sync : process(clk)
      begin
            if rising_edge(clk) then
                  syncB58 <= B58;
                  buffB58 <= syncB58;

                  syncB57 <= B57;
                  buffB57 <= syncB57;

                  syncA57 <= A57;
                  buffA57 <= syncA57;

                  syncA56 <= A56;
```

```
            buffA56 <= syncA56;

            syncS3 <= S3;
            buffS3 <= syncS3;

            syncS2 <= S2;
            buffS2 <= syncS2;

            syncS1 <= S1;
            buffS1 <= syncS1;

            syncS0 <= S0;
            buffS0 <= syncS0;

            syncB63 <= B63;
            buffB63 <= syncB63;

            syncA63 <= A63;
            buffA63 <= syncA63;

            syncB62 <= B62;
            buffB62 <= syncB62;

            syncA62 <= A62;
            buffA62 <= syncA62;

            syncB46 <= B46;
            buffB46 <= syncB46;

            syncA45 <= A45;
            buffA45 <= syncA45;

            syncB45 <= B45;
            buffB45 <= syncB45;

            syncA44 <= A44;
            buffA44 <= syncA44;

            syncB2 <= B2;
            buffB2 <= syncB2;

            syncB1 <= B1;
            buffB1 <= syncB1;

            syncA1 <= A1;
            buffA1 <= syncA1;

            syncB0 <= B0;
            buffB0 <= syncB0;

            syncA0 <= A0;
            buffA0 <= syncA0;
      end if;
end process;

changeB58 <= buffB58 xor syncB58;
changeB57 <= buffB57 xor syncB57;
changeA57 <= buffA57 xor syncA57;
changeA56 <= buffA56 xor syncA56;
changeS3 <= buffS3 xor syncS3;
changeS2 <= buffS2 xor syncS2;
```

```
      changeS1 <= buffS1 xor syncS1;
      changeS0 <= buffS0 xor syncS0;
      changeB63 <= buffB63 xor syncB63;
      changeA63 <= buffA63 xor syncA63;
      changeB62 <= buffB62 xor syncB62;
      changeA62 <= buffA62 xor syncA62;
      changeB46 <= buffB46 xor syncB46;
      changeA45 <= buffA45 xor syncA45;
      changeB45 <= buffB45 xor syncB45;
      changeA44 <= buffA44 xor syncA44;
      changeB2 <= buffB2 xor syncB2;
      changeB1 <= buffB1 xor syncB1;
      changeA1 <= buffA1 xor syncA1;
      changeB0 <= buffB0 xor syncB0;
      changeA0 <= buffA0 xor syncA0;

    low <= changeB58 or changeB57 or changeA57 or changeA56 or
           changeS3 or changeS2 or changeS1 or changeS0;
    high <= changeB63 or changeA63 or changeB62 or changeA62 or
            changeB46 or changeA45 or changeB45 or changeA44 or
            changeB2 or changeB1 or changeA1 or changeB0 or changeA0;

    process(clk)
    begin
        if rising_edge(clk) then
        if low = '1' then
            temp <= '0';
        elsif high = '1' then
            temp <= '1';
        else
            temp <= temp;
        end if;
        end if;
    end process;

    O <= temp;

end Behavioral;
```

C.3 Digital clock manager

Digital clock manager is a source code that is generated automatically by Xilinx tools according to user's wishes. However, in order to function properly some modifications are required. Some changes are needed in the declaration section of the component and at the end of the source code instead of driving the pulses to buffers they are connected to a special multiplexer called "bufgmux".

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

entity dcmCaseStudy is
```

```
port
 (-- Clock in ports
  dcm_clk           : in     std_logic;
  -- Clock out ports
  selectSignal : in std_logic;

  promoted : out std_logic;
  -- Status and control signals
  RESET             : in     std_logic;
  LOCKED            : out    std_logic
 );
end dcmCaseStudy;

architecture xilinx of dcmCaseStudy is
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of xilinx : architecture is
"dcmCaseStudy,clk_wiz_v3_6,{component_name=dcmCaseStudy,use_phase_ali
gnment=true,use_min_o_jitter=false,use_max_i_jitter=false,use_dyn_pha
se_shift=false,use_inclk_switchover=false,use_dyn_reconfig=false,feed
back_source=FDBK_AUTO,primtype_sel=MMCM_ADV,num_out_clk=2,clkin1_peri
od=2.000,clkin2_period=10.0,use_power_down=false,use_reset=true,use_l
ocked=true,use_inclk_stopped=false,use_status=false,use_freeze=false,
use_clk_valid=false,feedback_type=SINGLE,clock_mgr_type=MANUAL,manual
_override=false}";

  signal clkfbout         : std_logic;
  signal clkfbout_buf     : std_logic;
  signal clkfboutb_unused : std_logic;
  signal clkout0          : std_logic;
  signal clkout0b_unused  : std_logic;
  signal clkout1          : std_logic;
  signal clkout1b_unused  : std_logic;
  signal clkout2_unused   : std_logic;
  signal clkout2b_unused  : std_logic;
  signal clkout3_unused   : std_logic;
  signal clkout3b_unused  : std_logic;
  signal clkout4_unused   : std_logic;
  signal clkout5_unused   : std_logic;
  signal clkout6_unused   : std_logic;
  -- Dynamic programming unused signals
  signal do_unused        : std_logic_vector(15 downto 0);
  signal drdy_unused      : std_logic;
  -- Dynamic phase shift unused signals
  signal psdone_unused    : std_logic;
  -- Unused status signals
  signal clkfbstopped_unused : std_logic;
  signal clkinstopped_unused : std_logic;
  signal tempClock : std_logic;
begin

  -- Input buffering
  -------------------------------------
--  clkin1_buf : IBUFG
--  port map
--   (O => dcm_clk,
--    I => dcm_clk);

  -- Clocking primitive
  -------------------------------------
  -- Instantiation of the MMCM primitive
  --    * Unused inputs are tied off
```

```
--    * Unused outputs are labeled unused
mmcm_adv_inst : MMCME2_ADV
generic map
 (BANDWIDTH            => "OPTIMIZED",
  CLKOUT4_CASCADE      => FALSE,
  COMPENSATION         => "ZHOLD",
  STARTUP_WAIT         => FALSE,
  DIVCLK_DIVIDE        => 25,
  CLKFBOUT_MULT_F      => 60.125,
  CLKFBOUT_PHASE       => 0.000,
  CLKFBOUT_USE_FINE_PS => FALSE,
  CLKOUT0_DIVIDE_F     => 6.500,
  CLKOUT0_PHASE        => 0.000,
  CLKOUT0_DUTY_CYCLE   => 0.500,
  CLKOUT0_USE_FINE_PS  => FALSE,
  CLKOUT1_DIVIDE       => 4,
  CLKOUT1_PHASE        => 0.000,
  CLKOUT1_DUTY_CYCLE   => 0.500,
  CLKOUT1_USE_FINE_PS  => FALSE,
  CLKIN1_PERIOD        => 2.000,
  REF_JITTER1          => 0.010)
port map
  -- Output clocks
 (CLKFBOUT             => clkfbout,
  CLKFBOUTB            => clkfboutb_unused,
  CLKOUT0              => clkout0,
  CLKOUT0B             => clkout0b_unused,
  CLKOUT1              => clkout1,
  CLKOUT1B             => clkout1b_unused,
  CLKOUT2              => clkout2_unused,
  CLKOUT2B             => clkout2b_unused,
  CLKOUT3              => clkout3_unused,
  CLKOUT3B             => clkout3b_unused,
  CLKOUT4              => clkout4_unused,
  CLKOUT5              => clkout5_unused,
  CLKOUT6              => clkout6_unused,
  -- Input clock control
  CLKFBIN              => clkfbout_buf,
  CLKIN1               => dcm_clk,
  CLKIN2               => '0',
  -- Tied to always select the primary input clock
  CLKINSEL             => '1',
  -- Ports for dynamic reconfiguration
  DADDR                => (others => '0'),
  DCLK                 => '0',
  DEN                  => '0',
  DI                   => (others => '0'),
  DO                   => do_unused,
  DRDY                 => drdy_unused,
  DWE                  => '0',
  -- Ports for dynamic phase shift
  PSCLK                => '0',
  PSEN                 => '0',
  PSINCDEC             => '0',
  PSDONE               => psdone_unused,
  -- Other control and status signals
  LOCKED               => LOCKED,
  CLKINSTOPPED         => clkinstopped_unused,
  CLKFBSTOPPED         => clkfbstopped_unused,
  PWRDWN               => '0',
  RST                  => RESET);
```

```
  -- Output buffering
  -------------------------------------
  clkf_buf : BUFG
  port map
   (O => clkfbout_buf,
    I => clkfbout);

      promote : BUFGMUX port map (tempClock, clkout0, clkout1,
selectSignal);
      promoted <= tempClock;

end xilinx;
```

C.4 Wrapper file

Wrapper file unites all components under a common interface which is seen as a unit by VHDL compiler. This file creates instances of the components, redirects their inputs and their outputs and connects them (when external signals are used). The following code is a simple wrapper file and can be modified to suit custom needs and other components.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity wrapper is
port (
      A,B:  in signed(63 downto 0);
      S:    in std_logic_vector(3 downto 0);
      Result:     out signed(63 downto 0);
      dcm_freq : in std_logic;
      rst : in std_logic;
      lck : out std_logic
);
end wrapper;

architecture Behavioral of wrapper is

    signal index : std_logic := '0';
    signal finalClock : std_logic;
    signal indexBuffer : std_logic;
    signal dffA1, dffB1, dffA2, dffB2, syncInputA, syncInputB  :
           signed(63 downto 0);
    signal dffS1, dffS2, syncS : std_logic_vector(3 downto 0);

    attribute KEEP : string;
    attribute KEEP of syncInputA: signal is "TRUE";
    attribute KEEP of syncInputB: signal is "TRUE";

      component alu64
      port (
             clk:       in std_logic;
             A,B:       in signed(63 downto 0);
             S:   in std_logic_vector(3 downto 0);
             Result:    out signed(63 downto 0)
      );
      end component;
```

```
      component Selector
    port (
      clk:   in std_logic;
      B58:   in std_logic;
      B57:   in std_logic;
      A57:   in std_logic;
      A56:   in std_logic;
      S3:    in std_logic;
      S2:    in std_logic;
      S1:    in std_logic;
      S0:    in std_logic;
      B63:   in std_logic;
      A63:   in std_logic;
      B62:   in std_logic;
      A62:   in std_logic;
      B46:   in std_logic;
      A45:   in std_logic;
      B45:   in std_logic;
      A44:   in std_logic;
      B2:    in std_logic;
      B1:    in std_logic;
      A1:    in std_logic;
      B0:    in std_logic;
      A0:    in std_logic;
      O:     out std_logic
    );
    end component;

    component dcmCaseStudy is
    port
     (-- Clock in ports
      dcm_clk           : in     std_logic;
      -- Clock out ports
      selectSignal : in std_logic;

      promoted : out std_logic;
      -- Status and control signals
      RESET             : in     std_logic;
      LOCKED            : out    std_logic
     );
    end component;

begin

      process(finalClock)
      begin
            if rising_edge(finalClock) then
                  syncInputA <= A;
                  syncInputB <= B;
                  dffA1 <= syncInputA;
                  dffB1 <= syncInputB;
                  dffA2 <= dffA1;
                  dffB2 <= dffB1;
                  syncS <= S;
                  dffS1 <= syncS;
                  dffS2 <= dffS1;
                  indexBuffer <= index;
            end if;
      end process;
```

```
   U_dcm : dcmCaseStudy port map (dcm_freq, indexBuffer, finalClock,
rst, lck);

      U_alu: alu64 port map (finalClock, dffA2, dffB2, dffS2,
Result);

      U_selector : Selector port map (finalClock, syncInputB(58),
syncInputB(57), syncInputA(57), syncInputA(56), syncS(3), syncS(2),
syncS(1), syncS(0), syncInputB(63), syncInputA(63), syncInputB(62),
syncInputA(62), syncInputB(46), syncInputA(45), syncInputB(45),
syncInputA(44), syncInputB(2), syncInputB(1), syncInputA(1),
syncInputB(0), syncInputA(0), index);

end Behavioral;
```

C.5 Test bench of demo circuit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_alu64 is
end entity;

architecture Behavioral of tb_alu64 is

      signal T_dcmInputClock : std_logic := '0';
      signal T_A, T_B, T_R : signed(63 downto 0) := (others => '0');
      signal T_Sel : std_logic_vector(3 downto 0) := "0000";
      signal rst : std_logic := '0';
      signal lck : std_logic := '0';

      component wrapper
      port (
            A,B:  in signed(63 downto 0);
      S:    in std_logic_vector(3 downto 0);
      Result:     out signed(63 downto 0);
      dcm_freq : in std_logic;
      rst : in std_logic;
      lck : out std_logic
            );
      end component;

begin

      U_wrap : wrapper port map (T_A, T_B, T_Sel, T_R,
                                 T_dcmInputClock, rst, lck);

      -- 500 MHz clock for dcm
      process
      begin
            T_dcmInputClock <= '0';
            wait for 1 ns;
            T_dcmInputClock <= '1';
            wait for 1 ns;
      end process;

      process
      begin
            --setup time for dcm
```

```
            wait for 1501 ns;

            --start performing additions
            T_Sel <= "0100";
            T_A <= x"0000011029200000";   --slow clock
            T_B <= x"0000128c2f0046e0";
            wait for 50 ns;

            T_A <= x"00001ebc2f5656e8";   --slow clock
            T_B <= x"00003fbde61e5660";
            wait for 50 ns;

            T_A <= x"40003fbde61e5668";   --fast clock
            T_B <= x"c1202aade61e567e";
            wait for 50 ns;

            T_A <= x"756b6aade75c5c5e";   --fast clock
            T_B <= x"917c08e5471c5cdf";
            wait for 50 ns;

            T_A <= x"0f4ead35776e5bf9";   --slow clock
            T_B <= x"7a3d4993666f1ac1";
            wait for 50 ns;

            T_A <= x"4964ec51b36a5bfb";   --slow clock???
            T_B <= x"7a3d4993666f1ac0";
            wait for 50 ns;

            T_A <= x"4964ed8d97aa199f";   --fast clock
            T_B <= x"7a3d4993dad97c64";
            wait for 50 ns;

            T_A <= x"4964ec945fdab094";   --fast clock
            T_B <= x"7a3d490bb3a68b9a";
            wait for 50 ns;

            T_A <= x"40003fbde61e5668";   --slow clock???
            T_B <= x"c1202aade61e567e";
            wait for 50 ns;

            T_A <= x"756b6aade75c5c5e";   --fast clock
            T_B <= x"917c08e5471c5cdf";
            wait for 50 ns;

            T_A <= x"0f4ead35776e5bf9";   --slow clock
            T_B <= x"7a3d4993666f1ac1";
            wait for 50 ns;

            T_A <= x"4964ec51b36a5bfb";   --fast clock
            T_B <= x"7a3d4993666f1ac0";
            wait for 50 ns;

      T_A <= x"40003fbde61e5668";   --fast clock
            T_B <= x"c1202aade61e567e";
            wait for 50 ns;
      end process;
end Behavioral;
```

Βιβλιογραφία

BDTI Industry Report. (2006). *FPGAs for DSP*. Berkeley Design Technology.

Brown, S., & Rose, J. (n.d.). *Architecture of FPGAs and CPLDs: A Tutorial*. Ανάκτηση από EECG: http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf

Digital System Design Using Data Path and Control Unit. (2013). Ανάκτηση από King Fahd University: http://faculty.kfupm.edu.sa/COE/elrabaa/coe200/DP_CU.pdf

EDIF Overview. (2005). Ανάκτηση από Elgris Technologies: http://www.elgris.com/content/edif_overview.html

Englander, I. (2009). *The Architecture of Computer Hardware, System Software and Networking*. New Jersey: John Wiley & Sons.

FPGA Timing. (2015). Ανάκτηση από Embedded Micro: https://embeddedmicro.com/tutorials/mojo/timing

Guide to EDIF. (2005, July 18). Ανάκτηση από Electronic Industries Alliance: http://web.archive.org/web/20051218041919/http://www.edif.org/introduction.html

Kugler, L. (2015, May 15). *Is 'Good Enough' Computing Good Enough?* Ανάκτηση από Communications of the ACM: http://cacm.acm.org/magazines/2015/5/186012-is-good-enough-computing-good-enough/fulltext#body-2

Kuon, I., Tessier, R., & Rose, J. (2008). *FPGA Architecture: Survey and Challenges*. Ανάκτηση από Imperial College London: http://www.doc.ic.ac.uk/~wl/papers/08/kuon08survey.pdf

Mano, M. M., & Ciletti, M. (2007). *Digital Design*. Pearson Education.

Mukhopadhyay, D. (2012). *Design of Control Path*. Ανάκτηση από Indian Institute of Technology Kharagpur: http://cse.iitkgp.ac.in/~debdeep/teaching/VLSI/slides/ControlPath.pdf

National Instruments. (2012, April 16). *Introduction to FPGA Technology*. Ανάκτηση από NI: http://www.ni.com/white-paper/6984/en/

Processor: Datapath and Control. (2014). Ανάκτηση από Linkoplings Universitet: https://www.ida.liu.se/~TDTS10/info/lectures/Lecture3.pdf

Roosta, R. (2010). *Synchronous Vs Asynchronous Design*. Ανάκτηση από California State University, Northridge: http://www.csun.edu/edaasic/roosta/Syn_Asyn_Design.pdf

Shannon, C., & McCarthy, J. (1956). *Automata Studies*.

Spartan-6 FPGA Clocking Resources. (2015, June 19). Ανάκτηση από Xilinx.com: http://www.xilinx.com/support/documentation/user_guides/ug382.pdf

Synchronous and Asynchronous Circuits. (2006). Ανάκτηση από University of Surrey: http://www.ee.surrey.ac.uk/Projects/CAL/seq-switching/synchronous_and_asynchronous_cir.htm

The Linley Group. (2009). *A Guide to FPGAs for Communications*.

Thompson, M. (2004, July 2). *FPGAs accelerate time to market for industrial designs*. Ανάκτηση από Design & Reuse: http://www.us.design-reuse.com/articles/8190/fpgas-accelerate-time-to-market-for-industrial-designs.html

Timing Analyzer. (2008). Ανάκτηση από Xilinx: http://www.xilinx.com/itp/xilinx10/isehelp/pta_p_ar_timing_constraints.htm

Vivado Synthesis - Net names are not preserved by mark_debug. (2015, May 29). Ανάκτηση από Xilinx: http://www.xilinx.com/support/answers/57727.html

Wawrzynek, J. (2013, March 19). *EECS150 - Digital Design*. Ανάκτηση από Berkeley : http://www-inst.eecs.berkeley.edu/~cs150/sp13/agenda/lec/lec17-timing2.pdf

Wolf, W. (2008). *Computers as Components*. Morgan Kaufmann.