Πολυτεχνειο Κρητης

Τμημα Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

Πλατφόρμα πειραματικής ανάπτυξης για αναδιατασσόμενη λογική βασισμένη σε μοντέλο εξυπηρετητή - πελάτη

Νικόδημος Γεωργιάδης

Εξεταστική Επιτροπή
Καθηγητής Απόστολος Δόλλας (Επιβλέπων)
Καθηγητής Διονύσιος Πνευματικάτος
Αναπληρωτής Καθηγητής Γιάννης Παπαευσταθίου

Χανιά, Ιούνιος 2010

Περίληψη

Στις μέρες μας, ένας σημαντικός παράγοντας που περιορίζει την πλήρη αξιοποίηση των δυνατοτήτων που μας προσφέρει η σύγχρονη τεχνολογία συσκευών αναδιατασσόμενης λογικής, είναι η έλλειψη ενός γρήγορου, αξιόπιστου και εύχρηστου τρόπου μεταφοράς δεδομένων στο επίπεδο του χρήστη μεταξύ ηλεκτρονικού υπολογιστή (Η/Υ) και FPGA (field-programmable gate array). Αυτός είναι και ο αποτρεπτικός λόγος δημιουργίας πλήρων συστημάτων τα οποία θα έχουν ταυτόχρονα software και hardware μέρος.

Στα πλαίσια της διπλωματικής εργασίας μελετήθηκε και υλοποιήθηκε ένα εύχρηστο εργαλείο το οποίο βασίζεται στο μοντέλο εξυπηρετητή - πελάτη και δίνει στον χρήστη την δυνατότητα της εύκολης χρήσης του πρωτοκόλλου Gigabit Ethernet και του οδηγού PCI-Express τα οποία υλοποιήθηκαν από προπτυχιακούς φοιτητές του εργαστηρίου Μικροεπεξεργαστών και Υλικού στις διπλωματικές εργασίες τους. Ο πελάτης έχει την δυνατότητα απομακρυσμένης πρόσβασης σε αναπτυξιακό σύστημα το οποίο βρίσκεται στον εξυπηρετητή.

Η χρήση του εργαλείου Re. DO. FPGA (Remote Download FPGA) επιτρέπει στον χρήστη να χρησιμοποιήσει εύκολα το πρωτόκολλο MTP (MHL Trasfer Protocol) και του οδηγού χρήσης του PCI-Express με σκοπό την εύκολη υλοποίηση πλήρων συστημάτων μιας και με χρήση σταθερών διεπαφών αυτοματοποιείται η διαδικασία χρήσης τους. Το Re. Do. FPGA χρησιμοποιεί το μοντέλο εξυπηρετητή - πελάτη με σκοπό να μειώσει τον απαιτούμενο αριθμό αναπτυξιακών από τους φοιτητές μιας και μπορεί να γίνει χρήση τους μέσω δικτύου.

Η σχεδίαση υλοποιήθηκε για μία αναπτυξιακή πλατφόρμα της οικογένειας Virtex 5 σε Linux λειτουργικό.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Κεφάλαιο 1

Εισαγωγή

Στο πρώτο κεφάλαιο γίνεται αναφορά στο πρόβλημα το οποίο αντιμετωπίζουμε, ο σκοπός και η συνεισφορά της παρούσας διπλωματικής εργασίας και περιγράφεται η δομή της.

1.1 Πρόβλημα

Παρατηρείται ότι ένας μεγάλος αριθμός διπλωματικών εργασιών που διεκπερεώνονται στο Πολυτεχνείο Κρήτης δεν αξιοποιούνται κατάλληλα. Κάτι τέτοιο ισχύει και στο Εργαστήριο Μικροεπεξεργαστών και Υλικού (MHL). Πλέον διεκπεραιώνονται διπλωματικές εργασίες οι οποίες αν αξιοποιηθούν κατάλληλα θα δημιουργηθούν χρήσιμα εργαλεία.

Επίσης, πλέον οι διπλωματικές εργασίες στο Εργαστήριο Μικροεπεξεργαστών και Υλικού απαιτείται η χρησιμοποίηση αναδιατασσόμενων συσκευών με σκοπό τη δημιουργία διαφόρων περιφερειακών και η υλοποίηση πλήρων συστημάτων. Αυτό όμως απαιτεί την ύπαρξη μεγάλου αριθμού αναδιατασσόμενων συσκευών, κυρίως της οικογένειας Virtex-5, πράγμα το οποίο δεν είναι εφικτό λόγο κόστους αγοράς.

Το κύριο πρόβλημα που αντιμετωπίζουν όμως οι σχεδιάσεις είναι το πλήθος των δεδομένων που χρειάζονται για επεξεργασία, μιας και οι διαθέσιμοι πόροι της FPGA δεν επαρκούν για αποθήκευση αυτών σε μνήμες. Αυτό έχει ως αποτέλεσμα τον περιορισμό των δεδομένων που μπορούν να χρησιμοποιηθούν και την μη δυνατότητα δημιουργίας ενός πλήρους συστήματος. Επίσης για την αλλαγή των δεδομένων χρειάζεται να γίνει δημιουργία ενός καινούργιου Bitstream, διαδικασία η οποία είναι χρονοβόρα.

Η υλοποίηση πλήρων συστημάτων απαιτεί ένα αξιόπιστο μεσο μεταφοράς δεδομένων στο επίπεδο του χρήστη μεταξύ ηλεκτρονικού υπολογιστή (Η/Υ) και FPGA (field-programmable gate array). Τέλος το μέσο το οποίο επιθυμεί να χρησιμοποιήσει ο χρήστης, εκτός από αξιόπιστο, πρέπει να είναι εύχρηστο και να προσαρμόζεται εύκολα στη σχεδίαση του.

1.2 Σκοπός

Στόχος της διπλωματικής εργασίας είναι η ανάπτυξη ενός εύχρηστου εργαλείου το οποίο να χρησιμοποιεί το μοντέλο εξυπηρετητή - πελάτη. Ο πελάτης θα έχει την δυνατότητα απομακρυσμένης πρόσβασης σε αναπτυξιακό σύστημα για προγραμματισμό του. Επίσης θα μπορεί να δημιουργήσει νέο project, να χρησιμοποιήσει project που έχει ήδη δημιουργήσει ή να στήλει έτοιμο Bitstream στον εξυπηρετητή και αφού προγραμματιστεί το αναπτυξιακό σύστημα, που βρίσκεται στον εξυπηρετητή, να λάβει μέσω δικτύου την έξοδο της σχεδίασης του. Τέλος, θα έχει την δυνατότητα να αξιοποιεί την τεχνολογία Gigabit Ethernet και PCI-Express 1x Lane που υποστηρίζονται από τα αναπτυξιακά συστήματα της οικογένειας Virtex-5 .

Ο εξυπηρετητής, στον οποίο θα βρίσκεται το αναπτυξιακό, θα είναι δέκτης εντολών από τον πελάτη και θα αυτοματοποιεί την διαδικασία εισαγωγής των σχεδιάσεων για χρήση των τεχνολογιών Gigabit Ethernet και PCI-Express 1x Lane. Επίσης θα δίνει την δυνατότητα στον πελάτη για παράλληλη χρήση των δύο τεχνολογιών.

Για το σκοπό αυτό χρησιμοποιείται ο driver για χρήση του διαύλου PCI-Express 1x Lane, που έχει υλοποιηθεί στα πλαίσια της διπλωματικής εργασίας του Ιωάννη Καρτσωνάκη[1] και το πρωτόκολλο για χρήση του Gigabit Ethernet, που υλοποιήθηκε από τον Δημήτριο Βασιλόπουλο[2].

Η προσαρμογή των προανεφερθέντων μέσων μεταφεράς δεδομένων και ειδικά του PCI-Express στις σχεδιάσεις άλλων χρηστών δεν είναι απλή διαδικασία μιας και επιθυμείται η χρήση του εργαλείου Xilinx EDK 10.1 και όχι του Xilinx ISE Design Suite 10.1. Για το λόγο αυτό το σύστημα που υλοποιήθηκε διευκολύνει το χρήστη στην προσαρμογή και χρήση των μέσων μεταφοράς δεδομένων. Επίσης, η χρήση του εργαλείου έχει ως συνέπεια την μείωση του αριθμού των απαιτούμενων αναδιατασσόμενων συσκευών από τους φοιτητές για την υλοποίηση των διπλωματικών εργασιών τους μιας και δίνεται η δυνατότητα για απομακρυσμένη πρόσβαση σε αυτές.

Το γεγονός ότι στο εργαστήριο δεν υπήρχε πρότερη εμπειρία σε αυτόν τον τομέα, η έλλειψη επαρκούς υποστήριξης για τα αναπτυξιακά συστήματα που είχαμε από την εταιρία κατασκευής και του λειτουργικού Linux για το PCI-Express, δημιούργησαν σημαντικά εμπόδια καθώς ακόμα και για προβλήματα των οποίων η λύση ήταν σχετικά απλή καταβλήθηκε έντονη προσπάθεια και καταναλώθηκε πολύς χρόνος.

1.3 Οργάνωση Διπλωματικής Εργασίας

Στη συνέχεια η εργασία περιλαμβάνει ακόμη εφτά κεφάλαια.

Αρχικά, στο κεφάλαιο 2 περιγράφονται οι λειτουργικότητες του Client και Server και

παρουσιάζεται μία επισκόπηση των πρωτοκόλλων UDP, TCP, του πρωτοκόλλου δικτύου SSH και τέλος της γλώσσας προγραμματισμού Python και της βιβλιοθήκης PyQt4.

Στο 3ο κεφάλαιο γίνεται αναφορά σε προηγούμενη σχετική έρευνα που έχει γίνει στο Πολυτεχνείο Κρήτης, και πιο συγκεκριμένα στο εργαστήριο Μικροεπεξεργαστών και Υλικού, και από καθηγητές του τομέα σε Πανεπιστημιακά ιδρύματα του εξωτερικού.

Συνεχίζοντας στο κεφάλαιο 4 γίνεται περιγραφή της αρχιτεκτονικής που δημιουργήσαμε και το τι πρέπει να γνωρίζει ο χρήστης του συστήματος. Επίσης περιγράφεται μέσω διαγραμμάτων ροής η διαδικασία χρήσης της αναδιατασσόμενης συσκευής ατομικά και η χρήση της μέσω του εργαλείου που υλοποιήθηκε.

Στο επόμενο κεφάλαιο, το κεφάλαιο 5, περιγράφονται οι αρχιτεκτονικές που υλοποιήθηκαν στο εργαλείο Xilinx EDK και στην συνέχεια γίνεται μια λεπτομερής περιγραφή των λειτουργιών του Client και των βημάτων που ακολουθεί ο Server σε κάθε μία από αυτή.

Στο 6ο κεφάλαιο γίνεται πιστοποίηση της λειτουργικότητας του εργαλείου μέσω παραδειγμάτων που υλοποιήθηκαν. Τα αποτελέσματα και οι μετρήσεις ως προς την απόδοση του συστήματος και η ευχρηστία του Re.Do. FPFA παρουσιάζονται στο 7ο κεφάλαιο. Στο κεφάλαιο 8 διατυπώνονται τα συμπεράσματα καθώς και προτάσεις για μελλοντική βελτίωση και επέκταση της σχεδίασης.

Τέλος, στο παράρτημα A, B και ^ περιγράγεται η διαδιακασία χρήσης του εργαλείου Xilinx EDK, εισαγωγής της αρχιτεκτονικής χρήσης του PCI-Express και του πρωτοκόλλου MTP σε EDK Project αντίστοιχα.

Κεφάλαιο 2

Σχετικές Τεχνολογίες

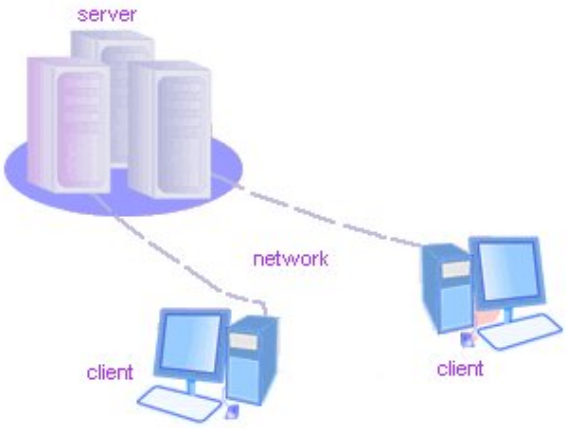
Στο κεφάλαιο αυτό γίνεται αναφορά στα εργαλεία και στις τεχνικές οι οποίες χρησιμοποιήθηκαν στα πλαίσια της διπλωματικής εργασίας. Αρχικά περιγράφεται το μοντέλο Client - Server. Στην συνέχεια γίνεται αναφορά στα πρωτόκολλα επικοινωνίας, στο πρωτόκολλο δικτύου SSH και στην γλώσσα προγραμματισμού Python και πιο συγκεκριμένα στην βιβλιοθήκη PyQt.

2.1 Μοντέλο Client - Server

Η χρήση του μοντέλου Client - Server είναι μια εφαρμογή διανεμημένης αρχιτεκτονικής η οποία χωρίζει τους στόχους ή τα φορτία εργασίας μεταξύ των φορέων παραχής υπηρεσιών (Servers) και των αιτούντων υπηρεσιών (Client)[3].Συχνά οι Clients και οι Servers λειτουργούν πέρα από ένα δίκτυο υπολογιστών, σε χωριστό υλικό (σχήμα 2.1). Ένας Server είναι ένα μηχάνημα υψηλής απόδοσης το οποίο τρέχει ένα ή περισσότερα προγράμματα και οι πόροι του διαμοιράζονται στους Client. Ένας Client δεν μοιράζεται τους πόρους του, αλλά ζητά από τον Server τη λειτουργία υπηρεσιών. Οι Client επομένως αρχίζουν την επικοινωνία με τον Server ο οποίος αναμένει τα εισερχόμενα αιτήματα.

Γενικά, το Client-Server computing αναφέρεται σε μια βασική αλλαγή στο στυλ των υπολογιστών, την αλλαγή από τα συστήματα που βασίζονται στα μηχανήματα στα συστήματα που βασίζονται στον χρήστη. Ειδικότερα, ένα σύστημα Client-Server είναι ένα σύστημα στο οποίο το δίκτυο ενώνει διάφορους υπολογιστικούς πόρους, ώστε οι Clients να μπορούν να ζητούν υπηρεσίες από έναν Server, ο οποίος προσφέρει πληροφορίες ή επιπρόσθετη υπολογιστική ισχύ.

Με άλλα λόγια, στο Client-Server μοντέλο, ο Client θέτει μια αίτηση και ο Server επιστρέφει μια ανταπόκριση ή κάνει μια σειρά από ενέργειες. Ο Server μπορεί να ενερ-



Σχήμα 2.1: Μοντέλο Client - Server.

γοποιείται άμεσα για την αίτηση αυτή ή να προσθέτει την αίτηση σε μια ουρά. Η άμεση ενεργοποίηση για την αίτηση μπορεί, για παράδειγμα, να σημαίνει ότι ο Server υπολογίζει έναν αριθμό και τον επιστρέφει αμέσως στον Client. Η τοποθέτηση της αίτησης σε μια ουρά μπορεί να σημαίνει ότι η αίτηση πρέπει να τεθεί σε αναμονή για να εξυπηρετηθεί.

Το Client-Server computing είναι πολύ σημαντικό, διότι επιτυγχάνει τα εξής:

- Αποτελεσματική χρήση της υπολογιστικής ισχύος.
- Μείωση του κόστους συντήρησης, δημιουργώντας συστήματα Client-Server που απαιτούν λιγότερη συντήρηση και κοστίζουν λιγότερο στην αναβάθμιση.
- Αύξηση της παραγωγικότητας, προσφέροντας στους χρήστες ξεκάθαρη πρόσβαση στις αναγκαίες πληροφορίες μέσω σταθερών και εύκολων στην χρήση διασυνδέσεων.
- Αύξηση της ευελιξίας και της δυνατότητας δημιουργίας συστημάτων που υποστηρίζουν πολλά περιβάλλοντα.

Στο βασικό Client-Server μοντέλο η πλευρά του Client πρώτα στέλνει ένα μήνυμα για να καλέσει σε ετοιμότητα τον Server. Από τη στιγμή που ο Client και ο Server έχουν επικοινωνία μεταξύ τους, ο Client μπορεί να υποβάλλει την αίτησή του [4].

2.1.1 Client

Ο Client [4] είναι ο αιτών των υπηρεσιών και δεν μπορεί παρά να είναι ένας υπολογιστής. Οι υπηρεσίες που ζητούνται από τον Client μπορεί να υπάρχουν στους ίδιους σταθμούς

εργασίας ή σε απομακρυσμένους σταθμούς εργασίας που συνδέονται μεταξύ τους μέσω ενός δικτύου. Ο Client ξεκινάει πάντα την επικοινωνία.

Ο Client παίζει τέσσερις βασικούς ρόλους, είναι στην πραγματικότητα το κέντρο της Client-Server εφαρμογής. Ο χρήστης αλληλεπιδρά με τον Client, ο Client ξεκινάει το μεγαλύτερο μέρος της ανάπτυξης της εφαρμογής, και ο Server υπάρχει για να απαντάει στις ανάγκες του Client.

2.1.2 Server

Ο Server [4] απαντάει στις αιτήσεις που γίνονται από τους Clients . Ένας Client μπορεί να ενεργεί ως Server εάν λαμβάνει και επεξεργάζεται αιτήσεις. Οι Servers δεν ξεκινάνε τις επικοινωνίες, περιμένουν τις αιτήσεις των Clients.

Τα συστατικά του Server είναι πολύ απλά. Μια Server μηχανή πρέπει να μπορεί να κάνει τα ακόλουθα :

- Να αποθηκεύει, να ανακτά και να προστατεύει πληροφορίες.
- Να επιθεωρεί τις αιτήσεις των Clients.
- Να δημιουργεί εφαρμογές διαχείρισης πληροφοριών, όπως δημιουργία αντιγράφων, ασφάλεια κτλ.
- Να διαχειρίζεται πληροφορίες.

Ο Server είναι η καρδιά του Client-Server συστήματος. Οι Servers είναι τα σημεία όπου αποθηκεύονται οι πληροφορίες και εκτελούνται οι εργασίες. Σήμερα, ο Server μπορεί να είναι οποιαδήποτε μορφή υπολογιστή.

2.2 Πρωτόκολλα Επικοινωνίας

2.2.1 UDP

Το πρωτόκολλο User Datagram Protocol (UDP) [5] είναι ένα από τα βασικά πρωτόκολλα που χρησιμοποιούνται στο Διαδίκτυο. Μία εναλλακτική ονομασία του πρωτοκόλλου είναι Universal Datagram Protocol. Διάφορα προγράμματα χρησιμοποιούν το πρωτόκολλο UDP για την αποστολή σύντομων μηνυμάτων από τον έναν υπολογιστή στον άλλον μέσα σε ένα δίκτυο υπολογιστών.

Ένα από τα κύρια χαρακτηριστικά του UDP είναι ότι δεν εγγυάται αξιόπιστη επικοινωνία. Τα πακέτα UDP που αποστέλλονται από ένα υπολογιστή μπορεί να φτάσουν στον παραλήπτη με λάθος σειρά, διπλά ή να μην φτάσουν καθόλου εάν το δίκτυο έχει μεγάλο

φόρτο. Αντιθέτως, το πρωτόκολλο TCP διαθέτει όλους τους απαραίτητους μηχανισμούς ελέγχου και επιβολής της αξιοπιστίας και συνεπώς μπορεί να εγγυηθεί την αξιόπιστη επικοινωνία μεταξύ των υπολογιστών. Η έλλειψη των μηχανισμών αυτών από το πρωτόκολλο UDP το καθιστά αρκετά πιο γρήγορο και αποτελεσματικό, τουλάχιστον για τις εφαρμογές εκείνες που δεν απαιτούν αξιόπιστη επικοινωνία.

Οι εφαρμογές audio και video streaming χρησιμοποιούν κατά κόρο πακέτα UDP. Για τις εφαρμογές αυτές είναι πολύ σημαντικό τα πακέτα να παραδοθούν στον παραλήπτη σε σύντομο χρονικό διάστημα ούτως ώστε να μην υπάρχει διακοπή στην ροή του ήχου ή της εικόνας. Κατά συνέπεια προτιμάται το πρωτόκολλο UDP διότι είναι αρκετά γρήγορο, παρόλο που υπάρχει η πιθανότητα μερικά πακέτα UDP να χαθούν. Στην περίπτωση που χαθεί κάποιο πακέτο, οι εφαρμογές αυτές διαθέτουν ειδικούς μηχανισμούς διόρθωσης και παρεμβολής ούτως ώστε ο τελικός χρήστης να μην παρατηρεί καμία αλλοίωση ή διακοπή στην ροή του ήχου και της εικόνας λόγω του χαμένου πακέτου. Σε αντίθεση με το πρωτόκολλο TCP, το UDP υποστηρίζει broadcasting, δηλαδή την αποστολή ενός πακέτου σε όλους τους υπολογιστές ενός δικτύου, και multicasting, δηλαδή την αποστολή ενός πακέτου σε κάποιους συγκεκριμένους υπολογιστές ενός δικτύου. Η τελευταία δυνατότητα χρησιμοποιείται πολύ συχνά στις εφαρμογές audio και video streaming ούτως ώστε μία ροή ήχου ή εικόνας να μεταδίδεται ταυτόχρονα σε πολλούς συνδρομητές.

2.2.2 TCP

Το TCP [6] (Transmission Control Protocol - Πρωτόκολλο Ελέγχου Μεταφοράς) είναι ένα από τα κυριότερα πρωτόκολλα των Πρωτοκόλλων Διαδικτύου. Βρίσκεται πάνω από το IP protocol (πρωτόκολλο IP). Οι κύριοι στόχοι του πρωτοκόλλου TCP είναι να επιβεβαιώνεται η αξιόπιστη αποστολή και λήψη δεδομένων, επίσης να μεταφέρονται τα δεδομένα χωρίς λάθη μεταξύ του στρώματος δικτύου (network layer) και του στρώματος εφαρμογής (application layer) και, φτάνοντας στο πρόγραμμα του στρώματος εφαρμογής, να έχουν σωστή σειρά. Οι περισσότερες σύγχρονες υπηρεσίες στο Διαδίκτυο βασίζονται στο TCP. Για παράδειγμα το SMTP (port 25), το παλαιότερο (και μη-ασφαλές) Telnet (port 23), το FTP και πιο σημαντικό το HTTP (port 80), γνωστό ως υπηρεσίες World Wide Web (WWW - Παγκόσμιος Ιστός). Το TCP χρησιμοποιείται σχεδόν παντού, για αμφίδρομη επικοινωνία μέσω δικτύου.

Αρχικά το Transmission ήταν Transfer, ένας όρος που προσδιόριζε την μεταβίβαση του ελέγχου στα άκρα του δικτύου TCPIP πριν αποσπαστεί το IP.

2.2.3 Διαφορές μεταξύ TCP και UDP

Το πρωτόκολλο TCP λειτουργεί εγκαθιδρύοντας συνδέσεις μεταξύ του αποστολέα και του παραλήπτη των πακέτων. Από την στιγμή που μία σύνδεση εγκαθιδρυθεί με επιτυχία, όλα τα δεδομένα αποστέλλονται από τον έναν υπολογιστή στον άλλο με την μορφή πακέτων χρησιμοποιώντας την σύνδεση αυτή. Τα κύρια χαρακτηριστικά του TCP είναι τα εξής[7], [6]:

- Αξιοπιστία:
  Το TCP χρησιμοποιεί διάφορους μηχανισμούς ούτως ώστε να διασφαλιστεί ότι τα πακέτα που μεταδίδονται από τον αποστολέα θα φτάσουν σίγουρα στον παραλήπτη και στην σωστή σειρά. Οι μηχανισμοί αυτοί περιλαμβάνουν την επιβεβαίωση λήψης πακέτου από τον παραλήπτη, την επαναποστολή πακέτων που χάθηκαν και τον καθορισμό ενός ελάχιστου χρονικού διαστήματος μέσα στο οποίο κάθε πακέτο που αποστέλλεται θα πρέπει να έχει παραληφθεί (timeout). Στην περίπτωση που χαθεί κάποιο πακέτο, ο αποστολέας προσπαθεί και πάλι να το ξαναστείλει. Επίσης, εάν ο παραλήπτης διαπιστώσει ότι ένα πακέτο δεν παραλήφθηκε, τότε θα ζητήσει από τον αποστολέα να του το ξαναστείλει.

- Σειρά πακέτων:
  Εάν δύο πακέτα αποσταλούν σε μία σύνδεση το ένα μετά το άλλο, τότε το πρωτόκολλο TCP εγγυάται ότι θα φτάσουν στον παραλήπτη με την ίδια σειρά με την οποία στάλθηκαν. Στην περίπτωση που λείπει ένα πακέτο και έρθουν μελλοντικά πακέτα, τότε αυτά κατακρατούνται στην προσωρινή μνήμη (buffer) μέχρι να φτάσει το πακέτο που λείπει. Τότε αναδιατάσσονται και εμφανίζονται με την σωστή σειρά στον παραλήπτη.

- Βαρύτητα:
  Το πρωτόκολλο TCP θεωρείται ιδιαίτερα βαρύ, δεδομένου ότι χρειάζονται τουλάχιστον 3 πακέτα για την εγκαθίδρυση της σύνδεσης, πρίν ακόμη μεταδοθεί οποιοδήποτε πακέτο δεδομένων. Επίσης, οι μηχανισμοί αξιοπιστίας που υλοποιεί το κάνουν ακόμη πιο βαρύ, πράγμα που έχει φυσικά σημαντικό αντίκτυπο στην ταχύτητα μετάδοσης δεδομένων.

Το UDP είναι ένα πιο απλό και ελαφρύ πρωτόκολλο, στο οποίο δεν υπάρχει η έννοια της σύνδεσης. Κάθε πακέτο UDP διανύει το δίκτυο σαν μία ξεχωριστή αυτόνομη μονάδα και όχι σαν μία σειρά πακέτων σε μία σύνδεση, όπως στο TCP. Τα κύρια χαρακτηριστικά του UDP είναι τα εξής[5],[7]:

- Αναξιόπιστο:
  Κατά την αποστολή ενός πακέτου, ο αποστολέας δεν είναι σε θέση να γνωρίζει εάν το πακέτο θα φτάσει σωστά στον προορισμό του ή εάν θα χαθεί μέσα στο δίκτυο. Δεν έχει προβλεφθεί η δυνατότητα επιβεβαίωσης λήψης πακέτου από τον παραλήπτη, ούτε η επαναμετάδοση ενός χαμένου πακέτου.

- Δεν υπάρχει σειρά:
  Τα πακέτα UDP, σε αντίθεση με το TCP, δεν αριθμούνται και κατά συνέπεια δεν υπάρχει κάποια συγκεκριμένη σειρά με την οποία θα πρέπει να φτάσουν στον παραλήπτη.

- Ελαφρύ:
  Το πρωτόκολλο αυτό καθ' αυτό είναι πολύ ελαφρύ σε σύγκριση με το TCP διότι δεν εφαρμόζει όλους τους μηχανισμούς αξιόπιστης επικοινωνίας που υπάρχουν στο δεύτερο. Αυτό έχει ως συνέπεια να είναι αρκετά πιο γρήγορο.

- Datagrams:
  Κάθε πακέτο UDP ονομάζεται επίσης και datagram, θεωρείται δε ως μεμονωμένη οντότητα που θα πρέπει να μεταδοθεί ολόκληρη. Κατά συνέπεια δεν υφίσταται η έννοια της διοχέτευσης πακέτων μέσα σε ένα κανάλι/σύνδεση.

2.3 SSH Server

Secure Shell ή SSH[8]είναι ένα πρωτόκολλο δικτύου που επιτρέπει την ανταλλαγή δεδομένων χρησιμοποιώντας ένα ασφαλές κανάλι μεταξύ δύο δικτυωμένων συσκευών. Αρχικά χρησιμοποιήθηκε σε GNU/Linux και σε Unix συστήματα για πρόσβαση σε λογαριασμούς Shell. Ο SSH σχεδιάστηκε ως η αντικατάσταση του Telnet και άλλα μη ασφαλή απομακρυσμένα shells, τα οποία στέλνουν τις πληροφορίες, ειδικότερα τους κωδικούς πρόσβασης σε κείμενο, πράγμα το οποίο τα καθιστά ευαίσθητα στην ανάλυση πακέτων. Η κρυπτογράφηση που χρησιμοποιείται από το SSH παρέχει την εμπιστευτικότητα και την ακεραιότητα των στοιχείων πάνω από ένα μη ασφαλές δίκτυο.

2.4 Python

Η Python [9] είναι μια γενικής χρήσης υψηλού επιπέδου γλώσσα προγραμματισμού, της οποίας η φιλοσοφία σχεδιασμού της δίνει έμφαση στην αναγνωσιμότητα του κώδικα. Η Python έχει ως στόχο να συνδυάζει αξιοσημείωτη ισχύ με πολύ σαφή σύνταξη και να διαθέτει μεγάλο και ολοκληρωμένο αριθμό πρότυπων βιβλιοθήκων.

Η Python υποστηρίζει πολλαπλά παραδείγματα προγραμματισμού, κατά κύριο λόγο, αλλά δεν περιορίζεται σε αντικειμενοστρεφή προγραμματισμό και σε μικρότερο βαθμό σε λειτουργικό προγραμματισμό. Διαθέτει ένα πλήρως δυναμικό σύστημα τύπου και αυτόματη διαχείριση μνήμης, παρόμοια με εκείνη του συστήματος, Ruby, Perl, και Tcl . Όπως και άλλες δυναμικές γλώσσες, η Python συχνά χρησιμοποιείται ως scripting language , αλλά χρησιμοποιείται επίσης σε ένα ευρύ φάσμα non-scripting contexts.

2.4.1 Python Qt4

Η PyQt [10] είναι ένα σύνολο εργαλείων για τη δημιουργία GUI εφαρμογών. Πρόκειται για μια ανάμιξη Python γλώσσα προγραμματισμού και της βιβλιοθήκης Qt . Η επίσημη ιστοσελίδα για την PyQt είναι η `http:\\www.riverbankcomputing.co.uk`. Η βιβλιοθήκη αναπτύχθηκε από τον Phil Thompson.

Η PyQt υλοποιείται ως ένα σύνολο ενοτήτων Python. Έχει πάνω από 300 κατηγορίες και σχεδόν 6000 λειτουργίες και μεθόδους. Είναι ένα multiplatform toolkit. Τρέχει σε όλα τα σημαντικά λειτουργικά συστήματα, συμπεριλαμβανομένων των Unix, Windows και Mac. Η PyQt έχει διπλή άδεια χρήσης. Οι προγραμματιστές μπορούν να επιλέξουν μεταξύ GPL και εμπορική άδεια. Προηγουμένως, η GPL έκδοση ήταν διαθέσιμη μόνο σε Unix. Ξεκινώντας από την έκδοση PyQt 4, η άδεια GPL είναι διαθέσιμη σε όλες τις υποστηριζόμενες πλατφόρμες.

Επειδή υπάρχουν πολλές διαθέσιμες κλάσεις, έχουν χωριστεί σε διάφορες ενότητες όπως φαίνεται και στο σχήμα 2.2.

Σχήμα 2.2: Python Qt modules.

Η ενότητα QtCore περιέχει τον πυρήνα ο οποίος δεν έχει GUI λειτουργικότητα. Η

ενότητα αυτή χρησιμοποιείται για την εργασία με το χρόνο, αρχεία και καταλόγους, διαφόρων τύπων δεδομένων, streams, URLs, τύπους MIME, thread ή process. Η ενότητα QtGui περιέχει τα γραφικά στοιχεία και τις σχετικές κλάσεις. Σε αυτά περιλαμβάνονται για παράδειγμα τα κουμπιά, τα παράθυρα, οι διάφορες μπάρες, γραμμές εργαλείων, χρώματα, γραμματοσειρές κλπ. Η ενότητα QtNetwork περιέχει τις κλάσεις για τον προγραμματισμό του δικτύου. Οι κλάσεις αυτές επιτρέπουν τη χρήση των πρωτόκολλο TCP/IP και UDP για Server και Client. Η QtXml περιλαμβάνει κλάσεις για την εργασία με αρχεία xml. Η ενότητα αυτή παρέχει υλοποίηση και για τις εφαρμογές SAX και DOM API. Η ενότητα QtSvg διαθέτει κλάσεις για την εμφάνιση του περιεχομένου SVG αρχείων. Scalable Vector Graphics (SVG) είναι μια γλώσσα περιγραφής δισδιάστατων γραφικών και γραφικών εφαρμογών σε XML. Η ενότητα QtOpenGL χρησιμοποιείται για την απόδοση 2D και 3D γραφικών χρησιμοποιώντας την OpenGL βιβλιοθήκη. Η ενότητα QtSql διαθέτει κλάσεις για την εργασία με βάσεις δεδομένων.

Κεφάλαιο 3

Σχετική Έρευνα

Στο κεφάλαιο αυτό γίνεται αναφορά σε συναφείς εργασίες οι οποίες υλοποιήθηκαν από ακαδημαϊκές ομάδες του Πολυτεχνείο Κρήτης, και πιο συγκεκριμένα στο εργαστήριο Μικροεπεξεργαστών και Υλικού, και ιδρυμάτων του εξωτερικού.

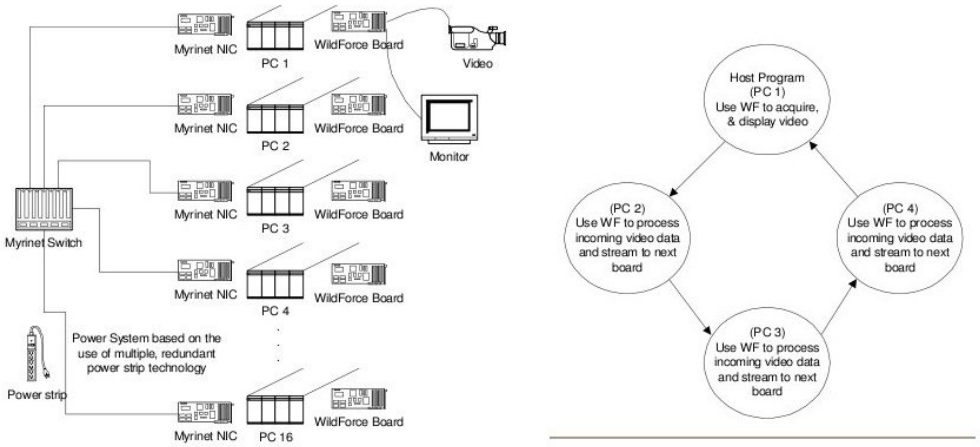
3.1 Συναφείς Εργασίες

3.1.1 Implementing an API for Distributed Adaptive Computing Systems

Η ομάδα του καθ. Peter Athanasστο Virginia Tech υλοποίησε ένα σύστημα το οποίο εκμεταλεύεται την μέθοδο cluster computing η οποία είναι μια οικονομική και αποδοτική μέθοδος κατασκευής παράλληλων υπολογιστών οι οποίοι χρησιμοποιούν τεχνολογία off-the-shelf για να εκμεταλλευτεί τον παραλληλισμό. Η εργασία αυτή δημοσιεύθηκε το 1999[11].

Το σύστημα που υλοποιήθηκε ονομάζεται Tower of Power [11] και το οποίο είναι ένα σύμπλεγμα υπολογιστών. Όπως φαίνεται στο σχήμα 3.1 [11] ο Tower of Power αποτελείται από δεκαέξι υπολογιστές Pentium II εξοπλισμένος ο καθένας από μια WildForce board η οποία είναι συνδεδεμένη με μία κάρτα Myricom LAN/SAN. Οι υπολογιστές είναι συδεδεμένοι μέσω ενός διακόπτη σε δεκαέξι switch. Ένα σύνολο από ογδόντα FPGAs της γενιάς XC4062XL των οποίων η μνήμη διανέμεται σε ολόκληρη την πλατφόρμα με τέτοιο τρόπο ώστε να είναι διαθέσιμοι ως υπολογιστικοί πόροι.

Το περιβάλλον ανάπτυξης εφαρμογών για τα μεγάλα υπολογιστικά συστήματα, δεν είναι ενιαία. Συνήθως ένας προγραμματιστής χρησιμοποιεί VHDL προσομοίωση και διάφορα εργαλεία σύνθεσης για να προγραμματίσει τις FPGA σε ένα adaptive computing boards. Για τον εξωτερικό έλεγχο του αναπτυξιακού, όπως η λήψη νέων αρχιτεκτονικών ή η ρύθμιση



Σχήμα 3.1: Virginia Tech ToP architecture and a "ring" based image-processing pipeline.

της συχνότητας ρολογιού παρέχεται μέσω του API.

Το API είναι προσβάσιμο από ένα πρόγραμμα C (σχήματα 3.2, 3.3 και 3.4 [11]) που ονομάζεται host program.

```
/* user structure to describe nodes  */
ACS_NODE nodes[4];
/* user structure for channels */
ACS_CHANNEL channels[4];
/* status of ACS API commands */
ACS_STATUS status;
ACS_SYSTEM ring;
/* build a ring of 4 WildForce boards */
for (int i=0;i<4;i++) {
        nodes[i].model = "WF4";
        channels[i].src_node = i;
        channels[i].src_port = 0;
        channels[i].dest_node = (i+1) % 4;
        channels[i].dest_port = 1;
}
/* must be first API call */
ACS_Initialize(argc, argv, &status);
ACS_System_Create(&ring, nodes, 4, channels, 4);
/* user program that accesses "ring" object */
ACS_System_Destroy(ring);
ACS_Finalize(); /*must be last API call */
```

Figure 3.2: Code fragment for creating a "ring" system object.

Το host program φροντίζει για τον έλεγχο ολόκληρου του συστήματος. Ο προγραμματιστής χρειάζεται μόνο να γράψει ένα host program χωρίς να τον ενδιαφέρει πόσα αναπτυξιακά έχει το σύστημα. Το host program μπορεί να δίνει πρόσβαση σε περισσότερες κατηγορίες κλήσεων στο API επιτρέποντας λειτουργίες όπως η διαχείρηση του συστήμα-

```
for (int i=0;i<4;i++) {
        /* send bitstream for each ACS board */
        ACS_Configure(config[i],i,ring,&status);
        /* set clock speed    */
        ACS_Clock_Set(clock,i,ring,&status);
        /* start clock          */
        ACS_Run(i,ring,&status);
        /* send reset signal */
        ACS_Reset(i,ring,&status);
}
for (int i=0;i<4;i++) {
        /* write initial data to each board's
           memory */
        ACS_Write(databuf[i],datalen[i],i,
                    brd_addr[i],ring,&status);
        /* then send an interrupt (or inta) signal
           #1 to the board */
        ACS_Interrupt(i,1,ring,&status);
}
```

Figure 3.3: Code fragment for configuring and writing to ACS boards.

```
/* use the ring to process the required number
of images */
for (int i=0;i<NUM_IMAGES;i++) {
        /* send image onto channel associated
           with port 0*/
        ACS_Enqueue(image[i],IMAGESIZE,0,
                    ring,&status);
        /* get resulting image from channel
           associated with port 1 */
        ACS_Dequeue(result_image[i],
                RESULT_SIZE,1,ring,&status);
}
```

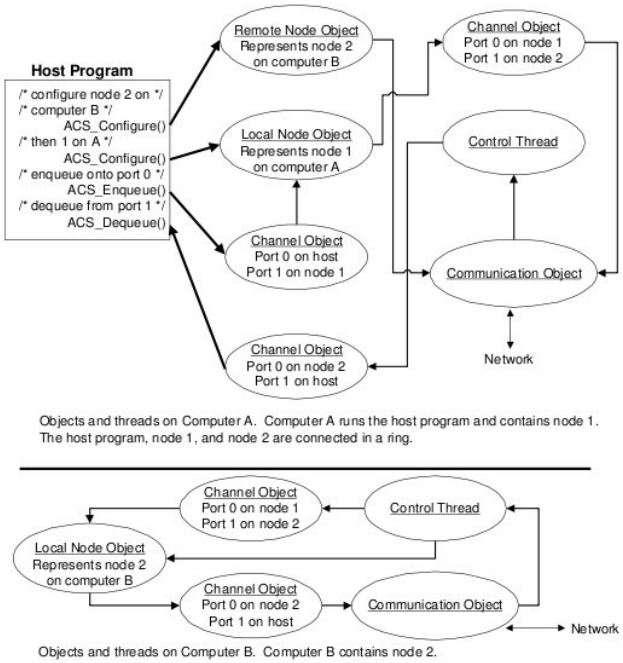
Figure 3.4: Code fragment demonstrating channel-based communication.

τος, τις προσβάσεις μνήμης, στα δεδομένα ροής και στην διαχείριση του αναπτυξιακού.

Πρόσθετες λειτουργίες οι οποίες επιτρέπουν να γίνονται ταυτόχρονα πράξεις σε πολλαπλά αναπτυξιακά είναι επίσης μέρος του API. Ένας από τους σχεδιαστικούς στόχους του ACS API είναι η δημιουργία ενός απλού API για τον έλεγχο σύνθετων συστημάτων. Ο χρήστης πρέπει με την χρήση ενός μικρού συνόλου εντολών να δημιουργήσει ένα λειτουργικό σύστημα.

Η συνολική λειτουργία του API φαίνεται στο σχήμα 3.5 [11].

Με χρήση του API έγιναν υλοποιήσεις που έχουν σχέσεις με γεννητικούς αλγόριθμους [;] και διεπαφές χρήσης των Distributed Adaptive Computing Systems [12].



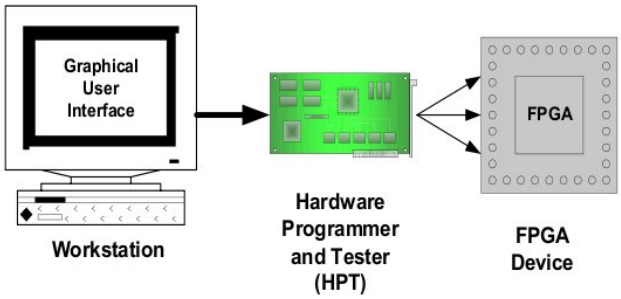
Σχήμα 3.5: Description of objects and threads on two computers.

3.1.2 A Universal Low Cost Run-Time and Programming Environment for Reconfigurable Computing

Στα πλαίσια της διπλωματικής εργασίας του Θωμά Κυριακίδη και Διονύσιο Ευσταθίου, στο Εργαστήριο Μικροεπεξεργαστών και Υλικού, το 2002 αναπτύχθηκε μία γλώσσα προγραμματισμού και ένα Run-time περιβάλλον για τον προγραμματισμό των αναπτυξιακών με την ονομασία ReRun [13]. Αυτό έγινε με σκοπό να αποφευχθεί η χρησιμοποίηση έτοιμων εφαρμογών μεγάλου κόστους. Για την δημιουργία της εφαρμογής χρειάζεται ένας μικροεπεξεργαστής τύπου Atmer AVR ο οποίος έχει χαμηλό κόστος.

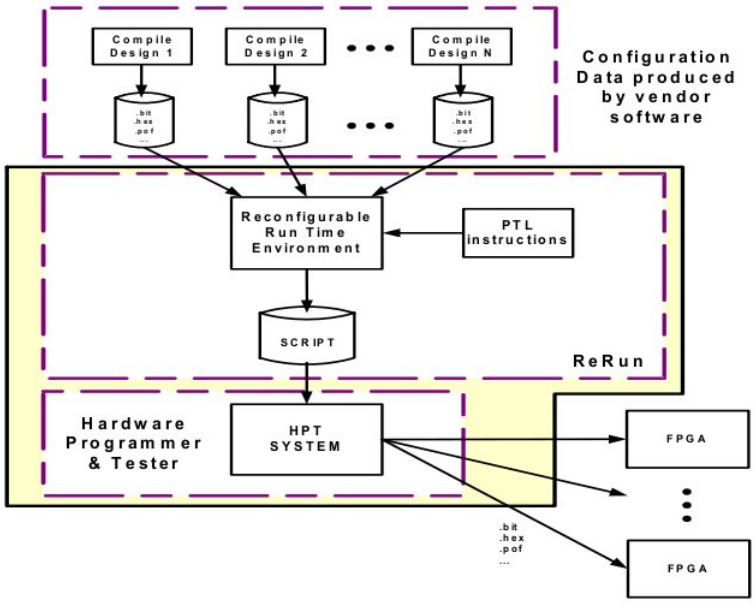
Στο σχήμα 3.6 [13] φαίνεται το σχηματικό διάγραμμα του συστήματος. Το ReRun τρέχει σε ένα ηλεκτρονικό υπολογιστή ο οποίος υποστηρίζει το γραφικό περιβάλλον. Ο υπολογιστής επικοινωνεί με χρήση με χρήση της σειριακής θύρας RS-232 με τον προγραμματιστή υλικού και τον ελεκτή (HPT) και ο οποίος είναι ο μηχανισμός προγραμματισμού. Ο HPT μπορεί να προγραμματίσει το αναπτυξιακό άμεσα μέσω διάφορων μηχανισμών.

Η αρχιτεκτονική του συστήματος που υλοποιήθηκε φαίνεται στο σχήμα 3.7 [13]. Αρχικά δημιουργείται από τον χρήστη το Bitstream από το εργαλείο που ο ίδιος επιθυμεί. Το ReRun αφού προσδιορίσει τα χαρακτηριστικά του αναπτυξιακού και διαβάσει τις εντολές



Σχήμα 3.6: Σχηματικό διάγραμμα ReRun.

για το PTL και το Bitsream δημιουργεί ένα script προγραμματισμού μαζί με τις οδηγίες προγραμματισμού, με το Bitsream του χρήστη και τις εντολές ελέγχου. Το σύστημα μπορεί να υποστηρίξει σήματα και ομάδες σημάτων, χρονικές εξαρτήσεις και τις σχέσεις μεταξύ των σημάτων, τον τρόπο με τον οποίο τα αρχεία θα πρέπει να φορτώνονται στο αναπτυξιακό καθώς και οι ενέργειες για τον έλεγχο του συστήματος.



Σχήμα 3.7: Αρχιτεκτονική ReRun.

Όπως αναφέρθηκε, το ReRun είναι ένα καθολικό Run-Time περιβάλλον για τον προγραμματισμό αναπτυξιακών. Κατά την υλοποίηση του ReRun υλοποιήθηκε μία γλώσσα προγραμματισμού η οποία λέγεται PTL (Programming and Testing Language) μαζί με

γραφικό περιβάλλον εργασίας. Η γλώσσα PTL μπορεί να χρησιμοποιηθεί για την δημιουργία script τα οποία περιγράφουν τις ρυθμίσεις ή τον έλεγχο της διαδικασίας και το οποίο με χρήση του GUI γίνεται συντακτικός και λεκτικός έλεγχος.

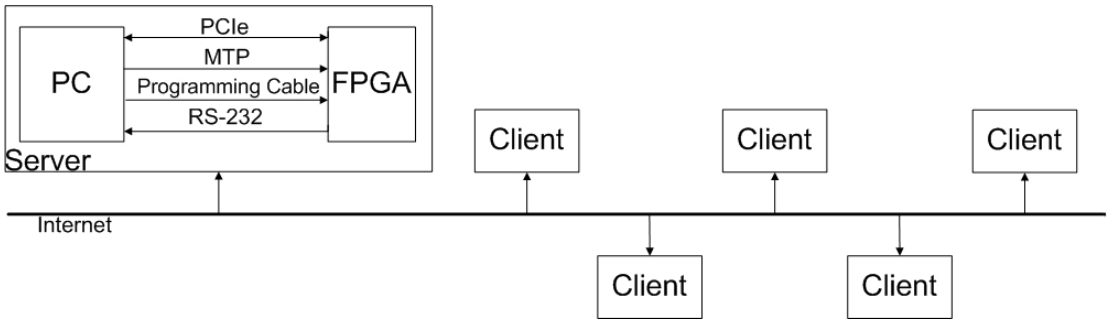
Ο πιο πάνω έλεγχος παράγει δύο αρχεία, το ένα για να κατέβει στο αναπτυξιακό το και το δεύτερο χρησιμοποιείται από το GUI. Στην περίπτωση που το script προορίζεται για τις ρυθμίσεις του αναπτυξιακού, τότε πρέπει να παρέχεται από τον χρήστη το Bitstream. Στην συνέχεια, μέσω του GUI μπορεί να προγραμματιστεί και να πραγματοποιήσει δοκιμές σε όποιο αναπτυξιακό επιθυμεί. Η επικοινωνία μεταξύ του software και του hardware γίνεται με χρήση της σειριακής θύρας RS-232.

Κεφάλαιο 4

Αρχιτεκτονική Συστήματος

Το κεφάλαιο 4 αναφέρεται στις λειτουργικές του Client και του Server και στο τι πρέπει να γνωρίζει ο χρήστης για να μπορεί να χρησιμοποιήσει το εργαλείο και τις μεθόδους σειριακής επικοινωνίας. Τέλος γίνεται περιγραφή, μέσω διαγραμμάτων ροής η αποκλειστική χρήση της αναδιατασσόμενης συσκευής από τον χρήστη και η χρήση της μέσω του εργαλείου που υλοποιήθηκε και ονομάστηκε Re. Do. FPGA (Remote Download FPGA).

Όπως φαίνεται στο σχήμα 4.1, το Re. Do. FPGA αποτελείται από τις λειτουργίες που αφορούν τον Client και την υλοποίηση του κώδικα και των κατάλληλων ρυθμίσεων του Server. Τέλος, με την χρήση του Re. Do. FPGA μπορεί να γίνει απομακρυσμένη χρήση του αναπτυξιακού συστήματος. Γι' αυτό το λόγο έγινε η υλοποίηση κατάλληλων σχεδιάσεων για χρήση του πρωτοκόλλου MTP[2], του οδηγού χρήσης του PCI-Express[1] και των δύο μέσων παράλληλα.



Σχήμα 4.1: Σύστημα Re. Do. FPGA

Με το σύστημα αυτό, ο χρήστης (σε σταθμό Client) μπορεί να:

- Να στέλνει στον Server τα αρχεία της σχεδίασης του

- Να στέλνει στον Server έτοιμο Bitstream
- Να στέλνει στον Server τα δεδομένα του
- Να επεξεργάζεται και να χρησιμοποιεί σχεδιάσης που βρίσκονται ήδη στον Server
- Να λαμβάνει από τον Server την έξοδο της σχεδίασης του

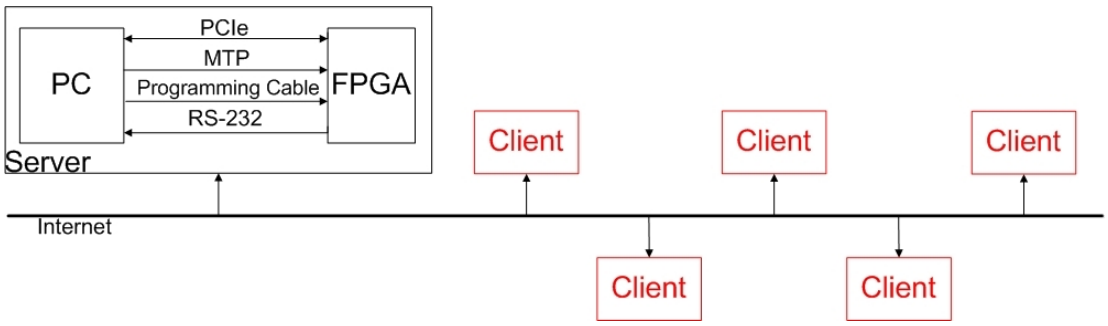
4.1 Αρχιτεκτονική Client - Server

Σκοπός του εργαλείου, όπως αναφέρθηκε στην εισαγωγή, είναι η δημιουργία ενός εύχρηστου εργαλείου με χρήση του μοντέλου εξυπηρετητή - πελάτη για την χρήση της θύρας Ethernet και του διαύλου PCI-Express του αναπτυξιακού μας.

4.1.1 Client

Για την δημιουργία του κώδικα ο οποίος χρησιμοποιείται από τον Client δημιουργήθηκε γραφική διεπαφή σε Python. Μέσω της γραφικής διεπαφής (GUI) και ακολουθώντας τα βήματα της διαδικασίας μπορεί να δημιουργήσει νέα σχεδίαση στον Server, να χρησιμοποιήσει σχεδίαση την οποία δημιούργησε παλαιότερα και βρίσκεται στον χώρο εργασίας του στον Server ή να στείλει στον Server έτοιμο Bitstream. Αφού γίνει προγραμματισμός του αναπτυξιακού, με την σχεδίαση που επέλεξε ο χρήστης, λαμβάνει την επιστρεφόμενη τιμή της σχεδίασης. Όλες οι ενέργειες που αναφέρθηκαν γίνονται με χρήση του δικτύου (internet) και του πρωτοκόλλου TCP.

Όπως φαίνεται στο σχήμα 4.2, ο χρήστης του Re. Do. FPGA δεν έχει στην διάθεση του κάποιο αναπτυξιακό σύστημα, αλλά γίνεται πρόσβαση σε αυτό με την χρήση του Server.



Σχήμα 4.2: Σύστημα Re. Do. FPGA - Client

Κατά την δημιουργία νέας σχεδίασης, μπορεί να χρησιμοποιηθεί το πρωτόκολλο MTP[2], ο οδηγός χρήση του PCI-Express[1] ή και τα δύο μέσα παράλληλα. Για να είναι εφικτή η χρησιμοποίηση τους, ο χρήστης πρέπει να γνωρίζει την διεπαφή και την λειτουγία των σημάτων. Οι τρεις σχεδιάσης, χρήσης του πρωτοκόλλου MTP, PCI-Express και παράλληλης χρήσης τους και οι διεπαφές χρήσης τους περιγράφονται αναλυτικά στις ενότητες 5.1.1, 5.1.2 και 5.1.3 αντίστοιχα.

Τα βήματα και η λειτουργικότητα του Client περιγράφονται στην ενότητα 5.2.1 μέσω μίας περιγραφής σχεδίασης με το σύστημα.

Ρυθμίσεις Client

Το εργαλείο είναι ανεξάρτητο πλατφόρμας μιας και χρησιμοποιήθηκε για την υλοποίηση του Python [14]. Αν ο χρήστης χρησιμοποιεί Linux λειτουργικό θα πρέπει να έχει εγκατεστημένες τις ακόλουθες βιβλιοθήκες:

- python2.6
- python-qt4
- python-qt4-dbg
- python-qt4-gl
- python-qt4-gl-dbg
- python-tk

Οι πιο πάνω βιβλιοθήκες είναι χρήσιμες για να τρέξει το γραφικό μέρος του εργαλείου. Για να είναι εφικτή η επικοινωνία με τον Linux Server, ο οποίος δουλεύει ως SSH Server πρέπει να εγκατασταθεί η βιβλιοθήκη:

- openssh-client

Αν ο χρήστης επιθυμεί να δουλεύει σε λειτουργικό Windows θα πρέπει να γίνεται χρήση του Python κονσόλας για το γραφικό μέρος της εφαρμογής και το εργαλείο PuTTY για την επικοινωνία με τον Server.

Φυσικά για την επικοινωνίας με τον Server απαιτείται η γνώση του IP στο οποίο είναι συνδεδεμένος ο Server και ο χρήστης να έχει λογαριασμό σε αυτόν. Η δημιουργία ενός λογαριασμού είναι στην αρμοδιότητα του διαχειριστή συστήματος (Administrator).

Για να χρησιμοποιήσει κάποιος το εργαλείο πρέπει να ακολουθήσει τα βήματα:

- Με χρήση Linux κονσόλας και με την εντολή "ssh -X username@hostname" όπου hostname το IP του Server γίνεται σύνδεση με τον Server.
- Αφού γίνει με επιτυχία η σύνδεση, στην ίδια κονσόλα εκτελείται ο κώδικας του Server με την εντολή: "sudo server_re_do_fpga.sh".
- Σε μία δεύτερη Linux κονσόλα εκτελείται ο κώδικας του Client με την εντολή: "python client_re_do_fpga.py"

και πλέον μπορεί να χρησιμοποιηθεί το εργαλείο Re.Do.FPGA (Remote Download FPGA).

Χρήση Πρωτοκόλλου GB Ethernet

Όπως αναφέρεται στην διπλωματική εργασία του Δημήτρη Βασιλόπουλου[2] για την μεταφορά δεδομένων από τον υπολογιστή στο αναπτυξιακό μας αναπτύχθηκε το πρωτόκολλο επιπέδου εφαρμογής MTP (MHL Transfer Protocol) [2]. Τα πακέτα του πρωτοκόλλου MTP διακρίνονται σε πακέτα ελέγχου (control packets) και πακέτα δεδομένων (data packets). Το Σχήμα 4.3 [2] δείχνει τη δομή του πακέτου. Τα προς μεταφορά δεδομένα καταλαμβάνουν το πεδίο Data το οποίο μπορεί να έχει μέγεθος ως 1467 bytes ώστε να μπορεί το δεδομενόγραμμα να ενθυλακώνεται σε ένα μόνο πλαίσιο Ethernet. Το πεδίο Data υπάρχει μόνο στα πακέτα δεδομένων. Το πεδίο Type καθορίζει τον τύπο και το πεδίο Number καθορίζει τον αριθμό του πακέτου (unsigned integer - 232 πακέτα).

Σχήμα 4.3: Δομή πακέτου MTP.

Το πρωτοκόλλου MTP δεν εξασφαλίζει αξιόπιστη μεταφορά δεδομένων. Το πρόγραμμα που υλοποιεί το πρωτόκολλο χρησιμοποιεί ένα UDP socket για να μεταδώσει αριθμημένα πακέτα και η σχεδίαση μας, στην πλευρά του αναπτυξιακού συστήματος, τα λαμβάνει και ελέγχει αν έχουμε απώλεια δεδομένων. Στην τελευταία περίπτωση η μεταφορά τερματίζεται.

Τα πακέτα του MTP:

- Πακέτο έναρξης συνόδου (Start of Session). Το πεδίο Type παίρνει την τιμή S (0x53) και το πεδίο Number παίρνει τιμή 0.
- Πακέτο δεδομένων (Data Packet). Το πεδίο Type παίρνει την τιμή D (0x44) και στο πεδίο Number ανατίθεται ο αριθμός του πακέτου (0 ως 232). Το πεδίο Data

έχει μπορεί να έχει μέγεθος 1467 bytes έτσι ώστε το δεδομενόγραμμα να μπορεί να ενθυλακωθεί σε ένα πλαίσιο Ethernet.

- Πακέτο λήξης συνόδου (End Session). Το πεδίο Type παίρνει τιμή E (0x45) και το πεδίο Number παίρνει τιμή του τελευταίου πακέτου δεδομένων αυξημένη κατά ένα.

Για περισσότερες λεπτομέρειες για την υλοποίηση του πρωτοκόλλου MTP και της υλοποίηση του GB Ethernet μπορούν να βρεθούν στο κείμενο της διπλωματικής εργασίας του Δημήτρη Βασιλόπουλου [2]. Παραδείγμα κώδικα δημιουργίας πακέτων και αποστολή αυτών με χρήση UDP socket βρίσκεται στο παράρτημα Ν.

Χρήση PCI-Express

Με χρήση της διπλωματικής εργασίας του Ιωάννη Καρτσωνάκη [1], παρατηρείται ότι η χρήση του PCI-Express χωρίζεται σε δύο διαδικασίες, διαδικασία εγγραφής και ανάγνωσης.

- Διαδικασία Εγγραφής:
  Κατά την εκτέλεση του Software για εγγραφή στο αναπτυξιακό μπορεί να γίνει χρήση μόνο μιας συγκεκριμένης διεύθυνσης μνήμης. Η τιμή της διεύθυνσης αυτής είναι:

  – Write Address: 0x024

  Τα δεδομένα που στέλλονται είναι μεγέθους 32 bits. Τα δεδομένα εγγράφονται στο αναπτυξιακό μόνο όταν η διεύθυνση εγγραφής είναι η πιο πάνω αλλιώς απορρίπτονται.

- Διαδιακασία Ανάγνωσης:
  Για να γίνει ανάγνωση χρησιμοποιείται η μέθοδος της δειγματοληψίας. Αρχικά γίνεται έλεγχος αν υπάρχουν έγκυρα δεδομένα στην μνήμη (Valid Address) του αναπτυξιακού. Αν υπάρχουν έγκυρα δεδομένα τότε γίνεται ανάγνωση από την μνήμη (Read Address) μια ποσότητας 32 bits. Η τιμή των διευθύνσεων που αναφέρθηκαν είναι:

  – Valid Address: 0x01c

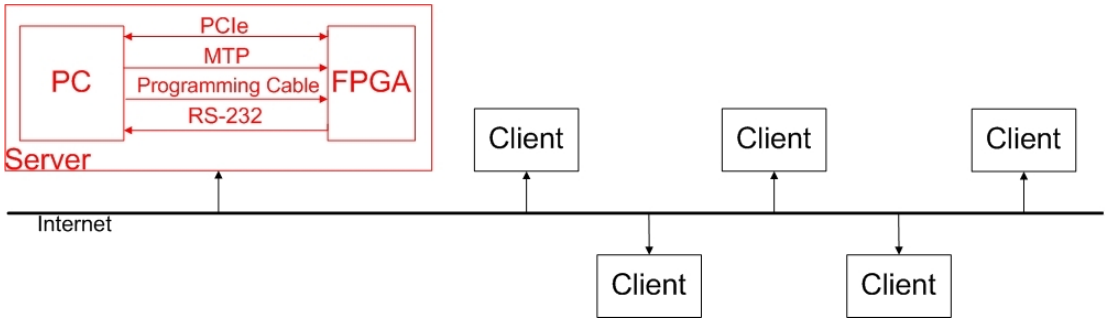
  – Read Address: 0x054

Για περισσότερες λεπτομέρειες για την υλοποίηση του οδηγού για το PCI-Express μπορούν να βρεθούν στο κείμενο της διπλωματικής εργασίας του Ιωάννη Καρτσωνάκη [1]. Παραδείγμα κώδικα αποστολής και λήψης δεδομένων στο αναπτυξιακό με χρήση του PCI-Express βρίσκεται στο παράρτημα Β.

4.1.2 Server

Για την υλοποίηση του κώδικα, ο οποίος απαντάει στις αιτήσεις του Client (λαμβάνονται μέσω δικτύου), χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python. Σε κάθε αίτηση του Client εκτελείται και η ανάλογη ενέργεια.

Ο Server , οπως φαίνεται στο σχήμα 4.4, είναι συνδεδεμένος με το αναπτυξιακό σύστημα και με το δίκτυο (Internet). Για την χρήση του πρωτοκόλλου MTP χρησιμοποιείται μια δεύτερη κάρτα δικτύου η οποία είναι συνδεδεμένη με την V5 Embedded Tri-Mode Ethernet Media Access Controller (MAC) του αναπτυξιακού συστήματος. Επίσης, για να χρησιμοποιηθεί ο οδηγός χρήσης του PCI-Express το αναπτυξιακό σύστημα πρέπει να τοποθετηθεί στο PCI-Express 1x Lane του ηλεκτρονικού υπολογιστή. Για τον προγραμματισμό του αναπτυξιακού (Διαδικασία Download Bitstream) γίνεται χρήση του καλωδίου προγραμματισμού (Cable Programmer) του αναπτυξιακού.



Σχήμα 4.4: Σύστημα Re. Do. FPGA - Server

Όταν ο χρήστης ζητήσει να δημιουργήσει νέο Project, και αφού ληφθεί το μέσω σειριακής επικοινωνίας που θα χρησιμοποιηθεί και τα αρχεία της σχεδίασης, ολοκληρώνεται αυτόματα η αρχιτεκτονική με την εισαγωγή της σχεδίασης χρήσης του μέσου αποστολής των δεδομένων που επιλέχθηκε και δημιουργεί το Bitstream. Επίσης, μπορεί να ληφθεί και να αποθηκευτεί σε αυτόν έτοιμο Bitstream και να χρησιμοποιήσει ο χρήστης (Client) αρχιτεκτονική η οποία βρίσκεται ήδη σε αυτόν.

Τέλος, μία βασική ενέργεια η οποία προβαίνει ο Server είναι ο προγραμματισμός του αναπτυξιακού συστήματος με χρήση του προγραμματιστή. Στην συνέχεια εισάγεται η είσοδος στην αρχιτεκτονική με χρήση του μέσου σειριακής επικοινωνίας που επιλέχθει και λαμβάνεται η έξοδος από αυτήν και η οποία επιστρέφεται μέσω δικτύου στον Client.

Τα βήματα που ακολουθούνται και η λειτουργικότητα του Server περιγράφονται αναλυτικά στην ενότητα 5.2.2.

4.1.3 Ρυθμίσεις Server

Ο Server τρέχει λειτουργικό σύστημα LINUX και πιο συγκεκριμένα έκδοση UBUNTU. Επιλέχθηκε η χρήση Linux λειτουργικού μιας και είναι ελεύθερο λογισμικό και μας δίνει την δυνατότητα για παρεμβάσεις στον πυρήνα του. Επίσης ο οδηγός του διαύλου PCI-Express υλοποιήθηκε για το συγκεκριμένο λειτουργικό. Για να εγκατασταθούν τα απαραίτητα αρχεία στον πυρήνα χρειάζεται η τελευταία σταθερή έκδοση πυρήνα η οποία βρίσκεται στην ιστοσελίδα `http://www.kernel.org` και γίνεται στην συνέχεια το κτίσιμο του σύμφωνα με τις οδηγίες που υπάρχουν. Επίσης χρειάζεται φυσικά ο οδηγός του διαύλου PCI-Express. Τέλος, μέσα από την ιστοσελίδα της Xilinx `http://www.xilinx.com/support/download/index.htm` πρέπει να γίνει λήψη της άδειας (licence) για χρήση του endpoint_blk_plus_v1_9 και το οποίο τοποθετείται στο μονοπάτι /home/username/.Xilinx/Coregen/CoreLicenses.

Τέλος η αναδιατασσόμενη συσκευή τοποθετείται στο PCI-Express 1x Lane του ηλεκτρονικού υπολογιστή [15] και γίνονται οι ρυθμίσεις [15] που περιγράφονται στο παράρτημα Β.

Για την ταυτόχρονη λειτουργία του υπολογιστή ως Server και χρήση του V5 Embedded Tri-Mode Ethernet Media Access Controller (MAC) του αναπτυξιακού είναι επιτακτική η χρήση δεύτερης κάρτας δικτύου, όπως αναφέρθηκε και πιο πάνω, τύπου 1000/100/10 Mbps. Επίσης πρέπει να απενεργοποιηθεί η λειτουργία του πρωτοκόλλου ipv6. Αυτό μπορεί να γίνει με δύο τρόπους, ανάλογα με την Linux έκδοση που χρησιμοποιείται.

Και σε αυτή την περίπτωση πρέπει να γίνουν αλλαγές στις ρυθμίσεις της αναδιατασσόμενης λογικής για να γίνεται χρήση της ταχύτητας 1000 Mbps. Οι ρυθμίσεις αυτές περιγράφονται στο παράρτημα ".

Στην συνέχεια πρέπει να γίνει εγκατάσταση των εργαλειών της Xilinx όπως αυτή περιγράφεται στον παράρτημα Α.

Για να είναι δυνατός ο προγραμματισμός του αναπτυξιακού πρέπει να γίνει χρήση των Cable Driver οι οποίοι βρίσκονται στην ιστοσελίδα `http://rmdir.de/~michael/xilinx/` όπου και υπάρχουν οδηγίες εγκατάστασης τους.

Τέλος για την λειτουργία του Server χρειάζεται να εγκατασταθεί το πακέτο:

- openssh-server

μιας και δουλεύει ως SSH Server .

4.2 Αποκλειστική - Μέσω Re.Do. FPGA χρήση της Αναδιατασσόμενης Συσκευής

Όπως αναφέρθηκε είδη, με την χρήση του Re.Do. FPGA θα μπορεί να γίνει χρήση της αναδιατασσόμενης συσκευής απομακρυσμένα. Στην συνέχεια γίνεται μια σύγκριση όταν η αναδιατασσόμενη συσκευή χρησιμοποιείται αποκλειστικά από τον χρήστη και όταν χρησιμοποιείται μέσω του Re.Do. FPFA. Η σύγκριση θα γίνει για την δημιουργία ενός Project του εργαλείου Xilinx 10.1.

4.2.1 Αποκλειστική Χρήση Αναδιατασσόμενης Συσκευής

Όταν ο κάθε χρήστης έχει στην διάθεση του ένα αναπτυξιακό τότε αυξάνεται ο απαιτούμενος αριθμός αναπτυξιακών που πρέπει να έχει στην διάθεση του το εργαστήριο. Αν πρέπει να γίνει χρήση ενός εκ των δύο μέσων σειριακής μεταφοράς δεδομένων, PCI-Express ή Gigabit Ethernet τότε αναγκαστικά οι υπολογιστές που υλοποιούνται οι αρχιτεκτονικές πρέπει να διαθέτουν κάρτα δικτύου τύπου 1000/100/10 Mbps και PCI-Express slot x1. Επίσης, για να χρησιμοποιηθούν οι πιο πάνω σχεδιάσεις χρειάζονται να γίνουν οι ρυθμίσεις που περιγράφηκαν πιο πάνω (Ρυθμίσεις Server) οι οποίες είναι και χρονοβόρες.

Όπως φαίνεται και στο διάγραμμα ροής του σχήματος 4.5 αφού δημιουργηθεί η αρχιτεκτονική στο εργαλείο Xilinx ISE Design Suite στην συνέχεια πρέπει να δημιουργηθεί και η κατάλληλη αρχιτεκτονική στο εργαλείο Xilinx EDK. Στο βήμα αυτό, αν δεν έχει ο χρήστης μεγάλη εμπειρία, τότε θα αντιμετωπίσει σοβαρά προβλήματα, ειδικά στην εισαγωγή της αρχιτεκτονικής που υλοποίησε στο Xilinx ISE Design Suite όταν αυτή πρέπει να προσαρμοστεί ώστε να χρησιμοποιήται ο οδηγός για PCI-Express ή το πρωτόκολλο Gigabit Ethernet μιας και μπορεί να προκύψουν προβήματα χρονισμού. Επίσης και η διαδικασία εισαγωγής της αρχιτεκτονικής MTP και ειδικά της αρχιτεκτονικής για χρήση του οδηγού PCI-Express δεν είναι καθόλου απλή.

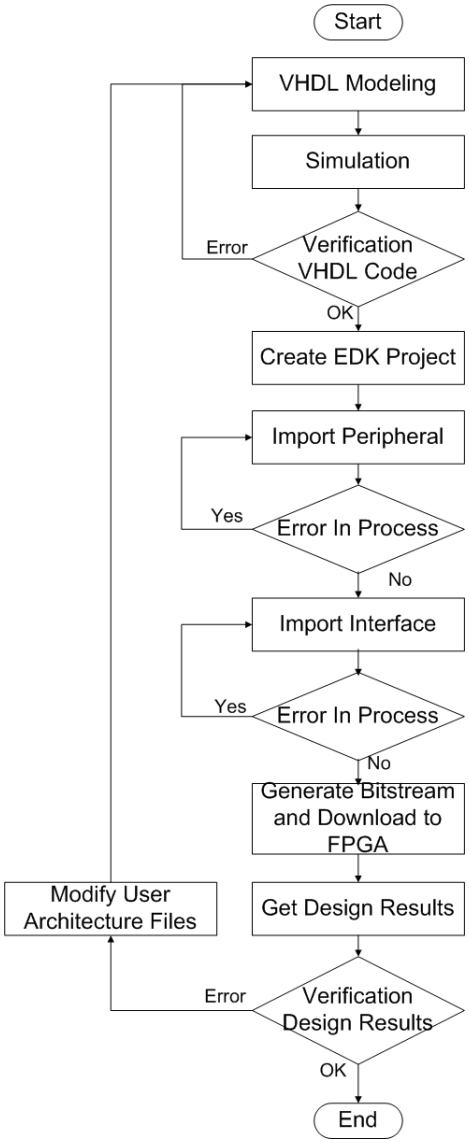
4.2.2 Χρήση αναδιατασσόμενης συσκευής μέσω του Re.Do. FPGA

Όταν γίνεται η χρήση του εργαλείου τότε μειώνεται αμέσως ο απαιτούμενος αριθμός αναδιατασσόμενων συσκευών μιας και χρειάζεται μόνο μια η οποία βρίσκεται στον Server. Πλέον, όπως φαίνεται και από το διάγραμμα ροής του σχήματος 4.6 ο χρήστης πλέον το μόνο που χρειάζεται να υλοποιεί είναι η αρχιτεκτονική του στο εργαλείο Xilinx ISE Design Suite. Στην συνέχεια απλος ακολουθεί τα βήματα του εργαλείου και λαμβάνει τα αποτελέσματα της αρχιτεκτονικής του. Το μόνο που πρέπει να γνωρίζει είναι την διεπαφή του

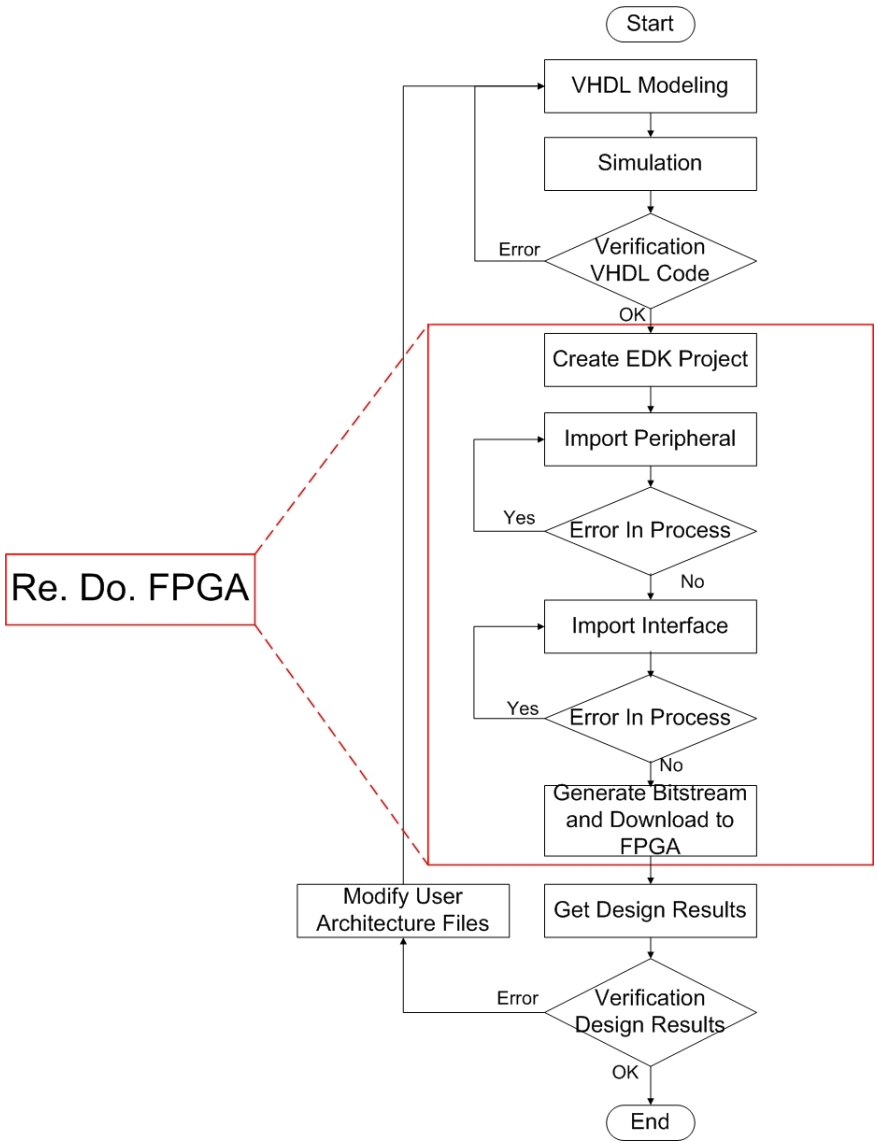
σειριακού μέσου μεταφοράς που επιθυμεί να χρησιμοποιήσει και οι οποίες περιγράφονται αναλυτικά στο κεφάλαιο 5.1.

Αν δεν επιθυμεί κάποιος να χρησιμοποιήσει το PCI-Express ή το Gigabit Ethernet τότε μπορεί να χρησιμοποιήσει την επιλογή αποστολής Bitsream και να του επιστραφούν τα αποτελέσματα της αρχιτεκτονικής του. Ο μόνος περιορισμός που υπάρχει είναι ως προς την οικογένεια αναδιατασσόμενης συσκευής που μπορεί να χρησιμοποιήσει.

Με την χρήση του Re. Do. FPGA μειώνεται ο απαιτούμενος αριθμός αναπτυξιακών από τους φοιτητές μιας και μπορεί να χρησιμοποιηθεί το αναπτυξιακό που βρίσκεται στον Server, αλλά όχι ταυτόχρονα. Τέλος, οι απαιτούμενες ρυθμίσεις για χρήση των σειριακών μέσων μεταφοράς εφαρμόζονται μόνο σε ένα ηλεκτρονικό υπολογιστή (Server).



Σχήμα 4.5: Διάγραμμα ροής αποκλειστικής χρήσης αναδιατασσόμενης συσκευής.



Σχήμα 4.6: Διάγραμμα ροής χρήσης αναδιατασσόμενης συσκευής μέσο του Re.Do. FPGA.

Κεφάλαιο 5

Σχεδίαση Συστήματος

Στο κεφάλαιο αυτό αρχικά περιγράφονται οι αρχιτεκτονικές που υλοποιήθηκαν στο εργαλείο Xilinx EDK και στην συνέχεια γίνεται μια λεπτομερής περιγραφή των λειτουργιών του Client και των βημάτων που ακολουθεί ο Server σε κάθε μία από αυτή.

5.1 Υλοποίηση Hardware Project

Για την υλοποίηση του Hardware μέρους της διπλωματικής εργασίας, χρησιμοποιήθηκε το εργαλείο Xilinx EDK 10.1 και η αναπτυξιακή πλατφόρμα Virtex 5 XC5VLX110T. Επιλέχθηκε η πιο πάνω πλατφόρμα μιας, όπως αναφέρθηκε και στην εισαγωγή, έγινε μια προσπάθεια για χρήση παλαιότερων διπλωματικών εργασιών, πράγμα το οποίο στέφθηκε με επιτυχία, και οι οποίες υλοποιήθηκαν για την συγκεκριμένη πλατφόρμα. Στα πλαίσια της διπλωματικής εργασίας χρησιμοποιήθηκαν οι διπλωματικές εργασίες του Δημήτριου Βασιλόπουλου και Ιωάννη Καρτσωνάκη.

Η διπλωματική εργασία του Δημήτριου Βασιλόπουλου [2] είχε ως θέμα "Μελέτη και πειραματική διάταξη για υψηλής ταχύτητας σειριακή επικοινωνία με FPGA", η οποία υλοποιήθηκε στο εργαλείο Xilinx EDK 10.1, και του Ιωάννη Καρτσωνάκη [1] "Ανάπτυξη οδηγού λειτουργικού συστήματος ανοιχτού λογισμικού (Linux) και VHDL κώδικα για σειριακή επικοινωνία υψηλής ταχύτητας (PCIe) ηλεκτρονικού υπολογιστή με την αναδιατασσόμενη συσκευή Virtex-5 Xilinx" και η οποία υλοποιήθηκε στο εργαλείο Xilinx ISE Design Suite 10.1 .

Για το σκοπό αυτό υλοποιήθηκαν τρία Project. Στο πρώτο γίνεται χρήση του πρωτοκόλλου Gigabit Ethernet, στο δεύτερο χρησιμοποιείται το PCI Express και στο τελευταίο και τα δύο μέσα σειριακής επικοινωνίας παράλληλα. Επίσης γίνεται χρήση του SoftCore επεξεργαστή Microblaze για την δυνατότητα χρήσης της σειριακής θύρας RS232 για ε-

πιστροφή αποτελέσματος όταν γίνεται χρήση του πρωτοκόλλου MTP. Αρχικά και τα τρία περιφερειακά προσαρμόστηκαν σε κοινό Project αλλά λόγο των πόρων που καταναλώνονται απορρίφθηκε. Στον πίνακα 5.1 παρουσιάζονται οι πόροι της FPGA που καταναλώνει η αρχική σχεδίαση. Το ποσοστό των πόρων που καταλαμβάνεται δεν είναι μεγάλο αλλά προτιμήθηκε να μην χρησιμοποιηθεί μιας και έτσι θα μειωνόταν το ποσοστό που θα είχε στην διάθεση του ο χρήστης του εργαλείου.

Device utilization summary	**Used**	**Available**	**Utilization**
Number of Slice Registers	4689	69120	6%
Number of Slice LUTs	4445	69120	6%
Number of Occupied Slices	2206	17280	12%
Number of bonded IOBs	7	640	1%
Number of Block RAM/FIFO	34	148	22%
Number of BUFG/BUFGCTRLs	7	32	21%

Πίνακας 5.1: Χρησιμοποίηση πόρων της FPGA από την αρχική σχεδίαση.

5.1.1 Σχεδίαση Χρήσης Πρωτοκόλλου MTP

Στην σχεδίαση αυτή χρησιμοποιήθηκε η υλοποίηση του Δημήτρη Βασιλόπουλου. Το μόνο που χρειάστηκε να υλοποιηθεί είναι το περιφερειακό στο οποίο ο χρήστης μπορεί να προσθέσει την δική του σχεδίαση. Για να είναι εφικτό, να προστεθεί η σχεδίαση, πρέπει να χρησιμοποιηθεί η πιο κάτω διεπαφή:

```
entity my_black_box is
port(
  mtp_rd_en_out      :  out std_logic;
  mtp_data_in        :  in std_logic_vector(7 downto 0);
  mtp_empty_in       :  in std_logic;
  mtp_full_in        :  in std_logic;
  mtp_overflow_in    :  in std_logic;
  mtp_prog_full_in   :  in std_logic;
  mtp_valid_in       :  in std_logic;
  mtp_underflow_in   :  in std_logic;
  clk                :  in std_logic;
  reset              :  in std_logic;
  FSL_M_enable       :  out std_logic;
  data_out           :  out std_logic_vector(31 downto 0)
);
end my_black_box;
```

Η λειτουργία των σημάτων περιγράφεται στον πίνακα 5.2

Το πιο πάνω είναι αναγκαίο μιας και πρέπει ο χρήστης να δίνει τιμές στα σήματα ελέγχου σύμφωνα με τις απαιτήσεις της σχεδίασης του. Το αποτέλεσμα της σχεδίασης αναγκαστικά επιστρέφεται με σειριακή θύρα RS232 μιας και η υλοποίηση του Gigabit Ethernet δεν παρέχει την λειτουργικότητα για επιστροφή τιμών. Για αυτό το λόγο χρησιμοποιείται ο SoftCore επεξεργαστής Microblaze . Το σχηματικό διάγραμμα της αρχιτεκτονικής φαίνεται στο σχήμα 5.1.

Οι πόροι της FPGA που καταναλώνονται παρουσιάζονται στον πίνακα 5.3.

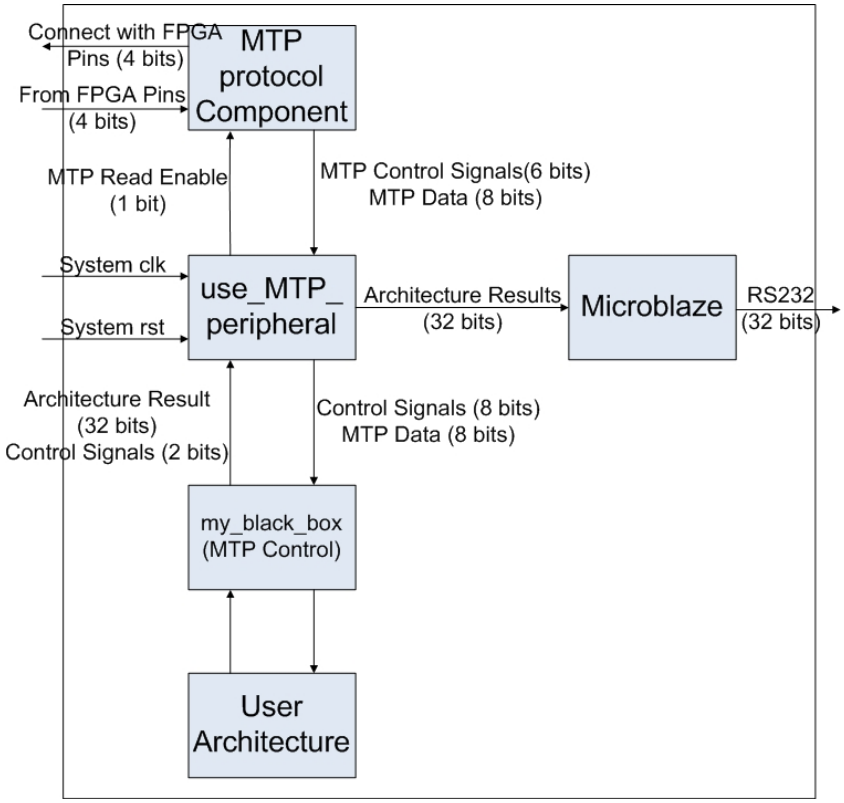
Η διαδικασία εισαγωγής των προαναφερθέντων περιφερειακών περιγράφεται αναλυτικα στο παράρτημα ".

Σήμα	Τύπος	Εύρος (bits)	Περιγραφή
mtp_rd_en_out	Έξοδος	1	Αν η FIFO εισόδου περιέχει δεδομένα η ενεργοποίηση αυτού του σήματος προκαλεί την ανάγνωση δεδομένων
mtp_data_in	Είσοδος	8	Δεδομένα εισόδου
mtp_empty_in	Είσοδος	1	Δηλώνει αν η FIFO εισόδου είναι άδεια
mtp_full_in	Είσοδος	1	Δηλώνει αν η FIFO εισόδο είναι πλήρης
mtp_overflow_in	Είσοδος	1	Δηλώνει ότι υπήρξε μια αποτυχημένη εγγραφή
mtp_prog_full_in	Είσοδος	1	Δηλώνει ότι τα περιεχόμενα της FIFO εισόδου ξεπέρασαν ένα κατώφλι
mtp_valid_in	Είσοδος	1	Δηλώνει αν διαβάζονται έγκυρα δεδομένα
mtp_underflow_in	Είσοδος	1	Δηλώνει ότι η FIFO εισόδο είναι άδεια και το σήμα mtp_rd_en_out είναι ίσο με ένα
clk	Είσοδος	1	Ρολόι συστήματος, 120MHz
reset	Είσοδος	1	Reset συστήματος
FSL_M_enable	Έξοδος	1	Δηλώνει ότι επιθυμήται η εγγραφή τιμής στο FSL Bus
data_out	Έξοδος	32	Η τιμή που επιθυμήται να εγγραφούν στο FSL Bus

Πίνακας 5.2: Διεπαφή χρήσης πρωτοκόλου MTP.

Device utilization summary	Used	Available	Utilization
Number of Slice Registers	1240	69120	1%
Number of Slice LUTs	1372	69120	1%
Number of Occupied Slices	786	17280	4%
Number of bonded IOBs	7	640	1%
Number of Block RAM/FIFO	17	148	11%
Number of BUFG/BUFGCTRLs	3	32	9%

Πίνακας 5.3: Χρησιμοποίηση πόρων της FPGA από την σχεδίαση χρήσης πρωτοκόλλου MTP.



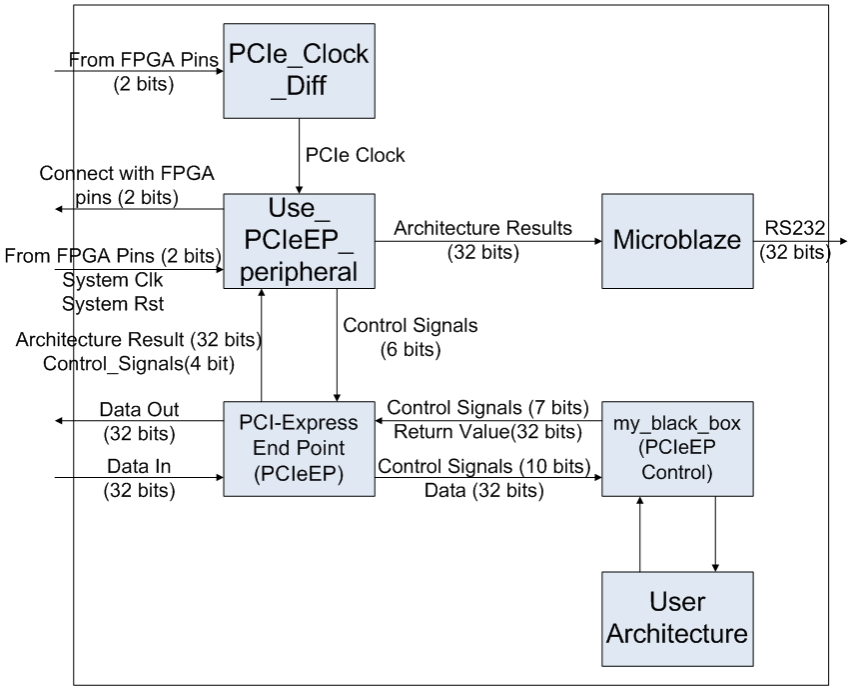
Σχήμα 5.1: Διάγραμμα αρχιτεκτονικής χρήσης πρωτοκόλλου MTP.

5.1.2 Σχεδίαση Χρήσης Οδηγού PCI-Express

Αρχικά προσαρμόσαμε την υλοποίηση του Ιωάννη Καρτσωνάκη η οποία υλοποιήθηκε στο εργαλείο Xilinx ISE Design Suite 10.1 στην σχεδίαση στο Xilinx EDK 10.1. Η διαδικασία αυτή ήταν και η πιο χρονοβόρα στα πλαίσια της διπλωματικής εργασίας μιας και χρειαζόταν να βρεθούν οι σωστές ρυθμίσεις για το ρολόι που χρησιμοποιείται από το PCI-Express. Στο παράρτημα Β περιγάφεται η διαδικασία εισαγωγής του περιφερειακού και στο σχήμα 5.2 φαίνεται το σχηματικό διάγραμμα της αρχιτεκτονικής.

Τέλος με την χρήση UNIX εντολών αποφεύχθηκε η χρήση της Compact Flash για τον προγραμματισμό της FPGA. Πλέον χρησιμοποιείται ο προγραμματιστής της FPGA (Cable USB) χωρίς να χρειάζεται επανεκκίνηση του υπολογιστή. Όπως και στην προηγούμενη σχεδίαση (χρήση Gigabit Ethernet), ο χρήστης το μόνο που χρειάζεται να γνωρίζει είναι τη διεπαφή:

```
entity my_black_box is
port(
  PCIE_CLK            :   in std_logic;
  RST                 :   in std_logic;
  FSL_Rst             :   in std_logic;
  FSL_Clk             :   in std_logic;
  FSL_M_enable        :   out std_logic;

  RX_engine_CLK       :   out std_logic;
  RX_engine_data      :   in std_logic_vector(31 downto 0);
  RX_engine_RD_EN     :   out std_logic;
  RX_engine_WR_EN     :   in std_logic;

  TX_engine_CLK       :   out std_logic;
  TX_engine_data      :   out std_logic_vector(31 downto 0);
  TX_engine_WR_EN     :   out std_logic;
  TX_engine_RD_PCIE   :   in std_logic;
  TX_engine_RD_EN     :   out std_logic;

  FIFO_RST            :   out std_logic;

  TX_fifo_full        :   in std_logic;
  TX_fifo_empty       :   in std_logic;
  RX_fifo_full        :   in std_logic;
  RX_fifo_empty       :   in std_logic
);
end my_black_box;
```



Σχήμα 5.2: Διάγραμμα αρχιτεκτονικής χρήσης οδηγού PCI-Express.

Η λειτουργία των σημάτων περιγράφεται στον πίνακα 5.4

Σήμα	Τύπος	Εύρος (bits)	Περιγραφή
PCIE_CLK	Είσοδος	1	Ρολόι που χρησιμοποιείται από το PCI-Express, 62.5MHz
RST	Είσοδος	1	Reset το οποίο χρησιμοποιείται από το δίαυλο PCI-Express
FSL_RST	Είσοδος	1	Reset συστήματος
FSL_Clk	Είσοδος	1	Ρολόι συστήματος, 120MHz
FSL_M_enable	Έξοδος	1	Δηλώνει ότι επιθυμήται η εγγραφή τιμής στο FSL Bus
RX_engine_CLK	Έξοδος	1	Ρολόι με το οποίο επιθυμήται να διαβάζονται δεδομένα από την FIFO εισόδου
RX_engine_data	Είσοδος	32	Δεδομένα εισόδου
RX_engine_RD_EN	Είσοδος	1	Αν η FIFO εισόδου περιέχει δεδομένα η ενεργοποίηση αυτού του σήματος προκαλεί την ανάγνωση δεδομένων
RX_engine_WR_EN	Είσοδος	1	Όταν έχει την τιμή ένα τότε εγγράφονται δεδομένα στην FIFO εισόδου
TX_engine_CLK	Έξοδος	1	Ρολόι με το οποίο επιθυμήται να εγγράφονται δεδομένα στην FIFO εξόδου
TX_engine_data	Έξοδος	32	Δεδομένα εξόδου
TX_engine_WR_EN	Έξοδος	1	Αν η FIFO εξόδου δεν είναι πλήρης η ενεργοποίηση του προκαλεί εγγραφή δεδομένων σε αυτή
TX_engine_RD_PCIE	Είσοδος	1	Δηλώνει ότι το PCI-Express έχει την δυνατότητα να διαβάσει δεδομένα από την FIFO εξόδου
TX_engine_RD_EN	Έξοδος	1	Αν η FIFO εξόδου περιέχει δεδομένα η ενεργοποίηση αυτού του σήματος προκαλεί την ανάγνωση δεδομένων
FIFO_RST	Έξοδος	1	Με την ενεργοποίηση του γίνονται Reset οι FIFO εισόδου και εξόδου
TX_fifo_full	Είσοδος	1	Δηλώνει αν η FIFO εξόδου είναι πλήρης
TX_fifo_empty	Είσοδος	1	Δηλώνει αν η FIFO εξόδου είναι άδεια
RX_fifo_full	Είσοδος	1	Δηλώνει αν η FIFO εισόδου είναι πλήρης
RX_fifo_empty	Είσοδος	1	Δηλώνει αν η FIFO εισόδου είναι άδεια

Πίνακας 5.4: Διεπαφή χρήσης οδηγούPCI-Express.

Τέλος, στον πίνακα 5.5 παρουσιάζονται οι πόροι της FPGA που καταναλώνονται από την σχεδίαση.

Device utilization summary	Used	Available	Utilization
Number of Slice Registers	4157	69120	6%
Number of Slice LUTs	3790	69120	5%
Number of Occupied Slices	1856	17280	10%
Number of bonded IOBs	4	640	1%
Number of Block RAM/FIFO	16	148	10%
Number of BUFG/BUFGCTRLs	5	32	15%

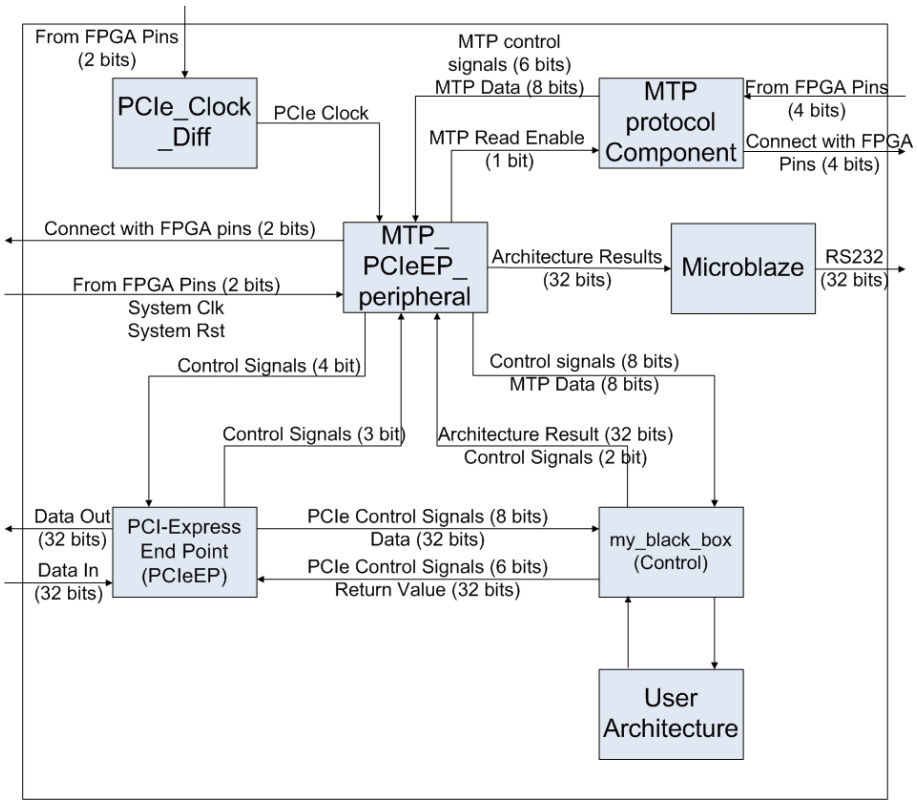
Πίνακας 5.5: Χρησιμοποίηση πόρων της FPGA από την σχεδίαση χρήσης οδηγού PCI-Express.

5.1.3 Σχεδίαση Παράληλλης Χρήσης Πρωτοκόλου MTP και οδηγού PCI-Express

Η τελευταία σχεδίαση που υλοποιήθηκε δίνει στον χρήστη την δυνατότητα παράλληλης χρήσης του Gigabit Ethernet και PCI-Express. Με αυτό τον τρόπο επιτυγχάνεται μεγαλύτερη ταχύτητα είσοδου δεδομένων στην αρχιτεκτονική. Επίσης μπορεί να χρησιμοποιηθεί το PCI-Express για την επιστροφή αποτελεσμάτων, δηλαδή παρέχεται έμμεσα η λειτουργικότητα για επιστροφή τιμών όταν χρησιμοποιείται το MTP. Η διαδιακασία δημιουργίας της σχεδίασης είναι ένας συνδιασμός των δύο προηγούμενων αρχιτεκτονικών και το σχηματικό διάγραμμα της αρχιτεκτονικής φαίνεται στο σχήμα 5.3. Η διαφορά που έχουμε είναι στην διεπαφή που πρέπει να υλοποιήσει ο χρήστης μιας και πλέον χρειάζεται να δίνει τιμές στα σήματα ελέγχου και του PCI-Express και του Gigabit Ethernet. Η διεπαφή είναι:

```
entity my_black_box is
port(
  mtp_rd_en_in          :   out std_logic;
  mtp_data_out          :   in std_logic_vector(7 downto 0);

  mtp_empty_out         :   in std_logic;
  mtp_full_out          :   in std_logic;
  mtp_overflow_out      :   in std_logic;
  mtp_prog_full_out     :   in std_logic;
  mtp_valid_out         :   in std_logic;
  mtp_underflow_out     :   in std_logic;

  RST                   :   in std_logic;
  FSL_Rst               :   in std_logic;
  FSL_Clk               :   in std_logic;
  FSL_M_enable          :   out std_logic;

  PCIE_CLK              :   in std_logic;

  RX_engine_CLK         :   out std_logic;
  RX_engine_data        :   in std_logic_vector(31 downto 0);
  RX_engine_RD_EN       :   out std_logic;
  RX_engine_WR_EN       :   in std_logic;

  TX_engine_CLK         :   out std_logic;
  TX_engine_data        :   out std_logic_vector(31 downto 0);
  TX_engine_WR_EN       :   out std_logic;
  TX_engine_RD_PCIE     :   in std_logic;
  TX_engine_RD_EN       :   out std_logic;

  FIFO_RST              :   out std_logic;

  TX_fifo_full          :   in std_logic;
  TX_fifo_empty         :   in std_logic;
  RX_fifo_full          :   in std_logic;
  RX_fifo_empty         :   in std_logic
);
end my_black_box;
```

Η λειτουργία των σημάτων είναι η ίδια με αυτή που περιγράφηκε πιο πάνω, χρησιμοποίηση ενός εκ των δύο μέσον σειριακής επικοινωνίας.



Σχήμα 5.3: Διάγραμμα αρχιτεκτονικής παράλληλης χρήσης οδηγού PCI-Express και πρωτοκόλλου MTP.

Στον πιο κάτω πίνακα, 5.6, παρουσιάζονται οι πόροι της FPGA που καταναλώνονται από την σχεδίαση.

5.2 Σχεδίαση Client-Server

Το εργαλείο που δημιουργήθηκε, όπως αναφέρεται και στον τίτλο, χρησιμοποιεί το μοντέλο εξυπηρετητή - πελάτη. Ακολουθήται αυτό το μοντέλο μιας και προτιμάται να μην στέλλεται στον Server ένα έτοιμο project αλλά να δημιουργείται αυτόματα σε αυτόν. Αυτό μας προστατεύει από σφάλματα κατά την δημιουργία ενός νέου Project, εξοικονόμηση χρόνου, μιας και στην περίπτωση σφάλματος ο χρήστης μπορεί εύκολα να κάνει διορθώσεις στα αρχεία του.

Device utilization summary	Used	Available	Utilization
Number of Slice Registers	4330	69120	6%
Number of Slice LUTs	3950	69120	5%
Number of Occupied Slices	1939	17280	11%
Number of bonded IOBs	6	640	1%
Number of Block RAM/FIFO	31	148	20%
Number of BUFG/BUFGCTRLs	6	32	18%

Πίνακας 5.6: Χρησιμοποίηση πόρων της FPGA από την σχεδίαση για παράλληλη χρήση οδηγού PCI-Express - MTP.

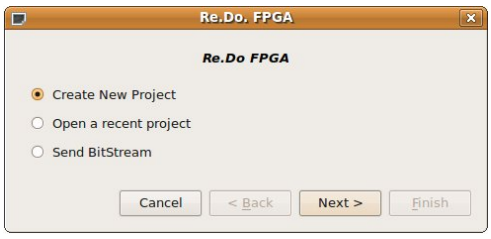
Για την σωστή επικοινωνία μεταξύ Client - Server χρησιμοποιήθηκε TCP [16] πρωτόκολλο μιας και προσδίδει ασφάλεια ότι θα γίνεται σωστή ανταλλαγή μηνυμάτων και πακέτων. Κάτι τέτοιο το δίνει και το UDP πρωτόκολλα αλλά δεν εγγυάται ότι τα μηνύματα και τα πακέτα λαμβάνονται με την σωστή σειρά. Σε κάθε βήμα της διαδικασίας που ακολουθεί ο χρήστης, αποστέλλονται τα ανάλογα μηνύματα στον Server για να προβεί στην ανάλογη ενέργεια.

Μία αναγκαία λειτουργία που χρησιμοποιήθηκε είναι η χρήση επιβεβαιώσεων κατά την ανταλλαγή μηνυμάτων και πακέτων. Αυτό ήταν αναγκαίο μιας και τα δύο πρωτόκολλα δημιουργούν αυτόματα τα πακέτα τους, και τα οποία έχουν σταθερό εύρος, και με την μέθοδο αυτή αποφύγαμε την ταυτόχρονη αποστολή δύο μηνυμάτων στο ίδιο πακέτο κάτι που θα οδηγήσει σε σφάλμα στην διαδικασία του Server.

Για την δημιουργία του Client χρησιμοποιείται η βιβλιοθήκη PyQt4 [17] μέσω της οποίας υλοποιήθηκε το γραφικό περιβάλλον.

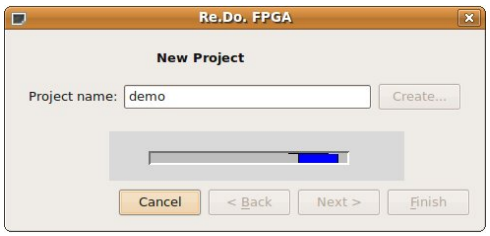
5.2.1 Λειτουργικότητα Client

Το εργαλείο αρχικά δίνει στον χρήστη τρεις επιλογές, όπως φαίνεται και στο σχήμα 5.4

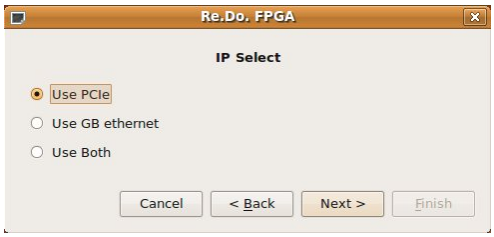


Σχήμα 5.4: Δυνατότητες εργαλείου Re.Do. FPGA.

- Δημιουργία καινούργιου Project
  Κατά την δημιουργία ενός νέου Project ο χρήστης αρχικά καλείται να δώσει το όνομα που επιθυμεί σε αυτό (σχημα 5.5). Στη συνέχεια έχει τις επιλογές της χρησιμοποίησης του διαύλου PCI-Express, Gigabit Ethernet ή και των δύο παράλληλα (σχήμα 5.6) για την αποστολή των δεδομένων του στην αναδιατασσόμενη συσκευή.



Σχήμα 5.5: Επιλογή ονόματος αρχιτεκτονικής.



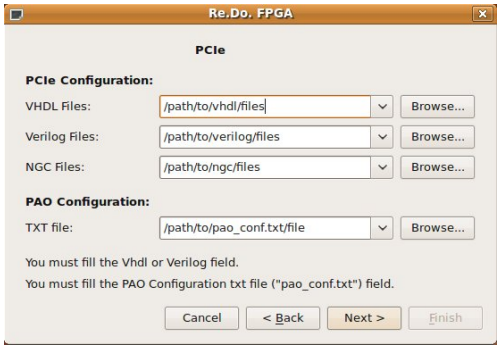
Σχήμα 5.6: Επιλογή μέσου σειριακής επικοινωνίας αρχιτεκτονικής - αναπτυξιακού.

Το μόνο που χρειάζεται να γνωρίζει είναι την διεπαφή που χρησιμοποιείται για κάθε μία από τις περιπτώσεις. Οι διεπαφές περιγράφηκαν και αναλύθηκαν στην ενότητα 5.1

Όπως φαίνεται και στο σχήμα 5.7 ο χρήστης χρειάζεται να υποδείξει το μονοπάτι που βρίσκονται τα αρχεία VHDL, Verilog, Netlist που χρησιμοποιεί. Αναγκαστικά χρειάζεται να υποδειχθούν αρχεία VHDL ή Verilog για να δημιουργηθεί το project με την δική του σχεδίαση. Τέλος χρειάζεται να δοθεί ένα αρχείο με όνομα pao_conf.txt . Στο αρχείο αυτό ο χρήστης πρέπει να αναγράφει τα αρχεία .vhd ή .v που χρησιμοποιεί ξεκινώντας από το κατώτερο στην ιεραρχία προς το ανώτερο.

Η χρησιμότητα του αρχείου pao_conf.txt είναι για την σωστή δημιουργία του αρχείου .pao που είναι υπεύθυνο για την υπόδειξη των αρχείων και της σωστής ιεραρχίας τους που χρησιμοποιούνται από την αρχιτεκτονική.

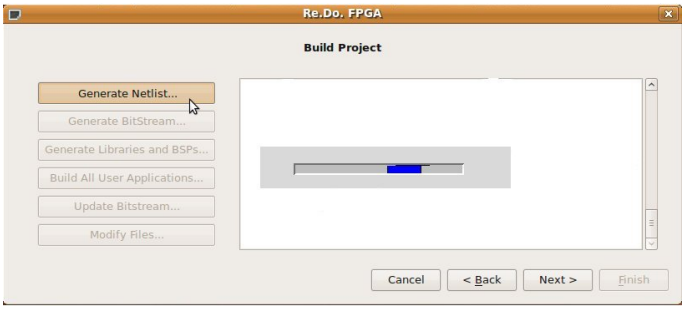
Το πιο πάνω βήμα είναι ίδιο και για τα τρία περιφερειακά. Το μόνο που διαφέρει είναι



Σχήμα 5.7: Επιλογή των αρχείων για αποστολή τους στον Server για δημιουργία του Project.

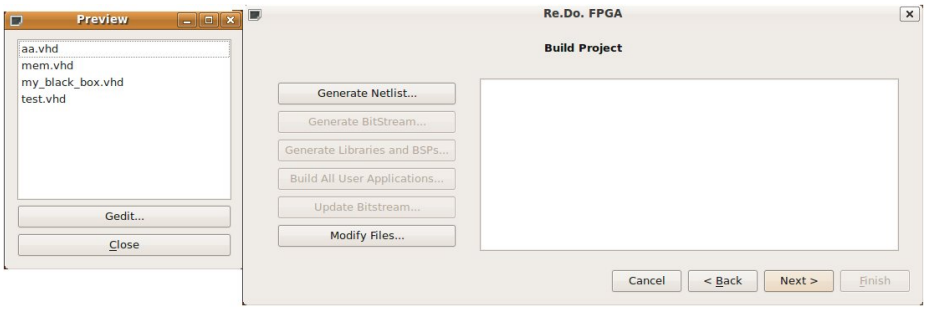
τα μηνύματα που στέλλονται στον Server.

Στην συνέχεια ο χρήστης με μια σειρά από κουμπιά καλείται να δημιουργήσει το download.bit αρχείο (σχήμα 5.8). Στα δεξιά του παραθύρου εμφανίζονται τα αποτελέσματα της διαδικασίας. Η όλη διαδικασία είναι και η πιο χρονοβόρα του εργαλείου. Στην περίπτωση που προκύψει κάποιο σφάλμα η διαδικασία διακόπτεται και μπορεί ο χρήστης να διορθώσει το σφάλμα στα αρχεία του με χρήση της επιλογής Modify Files και να συνεχίσει την διαδικασία κτισίματος του project.



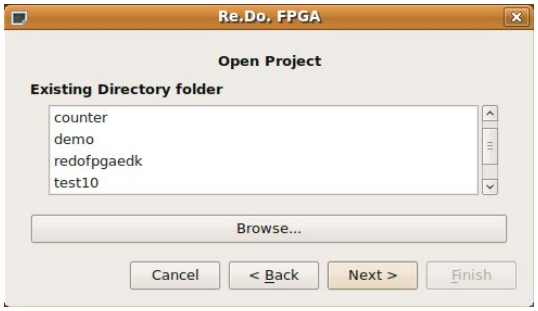
Σχήμα 5.8: Διαδικασία δημιουργίας αρχείου download.bit.

Με την επιλογή Modify Files μπορεί εύκολα μέσα από την λίστα των αρχείων κωδίκων που εμφανίζεται να επιλέξει το αρχείο στο οποίο θέλει να επέμβει και με χρήση του του κουμπιού Text Editor να ανοίξει σε γραφικό περιβάλλον (χρήση του Linux text editor) να κάνει τις αλλαγές (σχήμα 5.9). Η λειτουργία αυτή μπορούσε να παραληφθεί μιας και εύκολα μπορεί ο χρήστης, με χρήση UNIX εντολών, να επιτύχει το ίδιο αποτέλεσμα.



Σχήμα 5.9: Λίστα αρχείων κωδίκων που χρησιμοπούνται από την αρχιτεκτονική με σκοπό την τροποποίηση τους.

- Χρήση Project που βρίσκεται στον χώρο εργασίας του χρήστη στον Server
  Με αυτή την επιλογή ο χρήστης μέσα από μία λίστα που εμφανίζεται στο παράθυρο καλείται να υποδείξει το project που επιθυμεί να χρησιμοποιήσει, όπως φαίνεται και στο σχήμα 5.10.

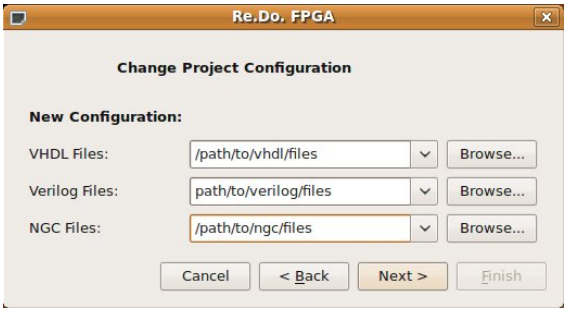


Σχήμα 5.10: Χρήση έτοιμου Project.

Επιλέγοντας το Project που επιθυμεί να δουλέψει του δίνεται η δυνάτοτητα να κάνει αλλαγές στα αρχεία που έστειλε κατά την διαδικασία δημιουργίας του (σχήμα 5.11)

Με την επιλογή αλλαγής αρχείων, ο χρήστης μπορεί να στείλει στον Server καινούργια αρχεία, VHDL, Verilog ή Ngc, όπως φαίνεται και από το σχήμα 5.12. Η μόνη προυπόθεση για να είναι επιτυχής η ενέργεια αυτή πρέπει τα νέα αρχεία που στέλλονται να αντιγράφουν τα υφιστάμενα αρχεία της σχεδίασης του. Η δεύτερη επιλογή μας οδηγεί στο παράθυρο κτισίματος ενός Project, διαδικασία η οποία περιγράφατε πιο πάνω. Τέλος, επιλέγοντας την τρίτη επιλογή (Nothing) οδηγούμαστε στο τελικό παράθυρο το οποίο περιγράφεται στην συνέχεια.

Σχήμα 5.11: Επιλογές κατά το άνοιγμα ενος Project.

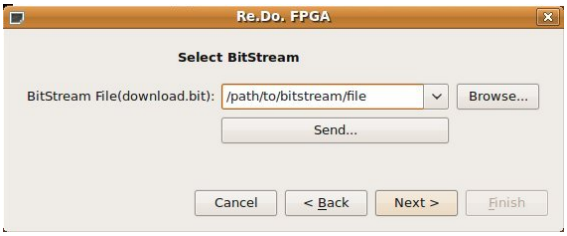


Σχήμα 5.12: Αλλαγή αρχείων της αρχιτεκτονικής.

- Αποστολή BitStream
  Εφόσον ο χρήστης επιλέξει να στείλει BitStream στον server πρέπει να δώσει το μονοπάτι που βρίσκεται το .bit αρχείο (σχήμα 5.13). Η χρήση αυτής της επιλογής προϋποθέτει ότι έχει δημιουργηθεί από το εργαλείο Xilinx EDK 10.1 ή Xilinx ISE Design Suite και για την αναδιατασσόμενη συσκευή Virtex-5 lx110t.

Και στις τρεις πιο πάνω περιπτώσεις, το επόμενο βήμα είναι το ίδιο. Ο χρήστης έχει τρεις επιλογές για επιλογή κώδικα δημιουργίας δεδομένων, σχήμα 5.14.

- Μη χρήση κώδικα:
  Με την επιλογή αυτή οδηγούμαστε στο τελικό παράθυρο (σχημα 5.15) όπου ο χρήστης πατώντας το κουμπί Download BitStream θα εμφανιστούν στον πίνακα τα αποτελέσματα που επιστρέφει σχεδίαση του. Πλέον αν ο χρήστης επιθυμεί να κάνει αλλαγές στην αρχιτεκτονική του, με χρήση του κουμπιού Modify Architecture Files επιστρέφει στο παράθυρο κτισίματος ενός Project και η διαδικασία επαναλαμβάνεται μέχρι να κατέβει η νέα αρχιτεκτονική.



Σχήμα 5.13: Αποστολή έτοιμου αρχείου bitstream (.bit).

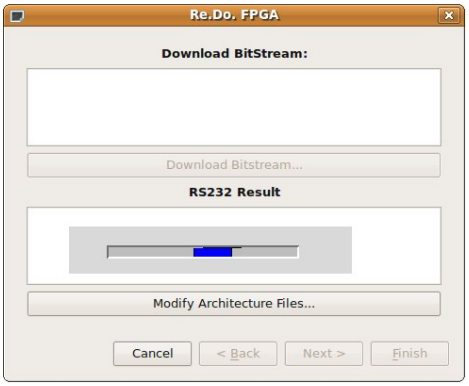
Σχήμα 5.14: Επιλογή τρόπου αποστολής δεδομένων.

- Χρήση κώδικα που ήδη βρίσκεται στον Server:
  Το παράθυρο που εμφανίζεται στον χρήστη είναι αυτό του σχήματος 5.16. Αφού ο χρήστης επιλέξει τον κώδικα που επιθυμεί (πατώντας το κουμπί Browse) και το φάκελο με τα αρχεία δεδομένων του που επιθυμεί να στείλει στον Server, αν υπάρχουν φυσικά, μπορεί να συνεχίσει στο τελευταίο βήμα της διαδικασίας.

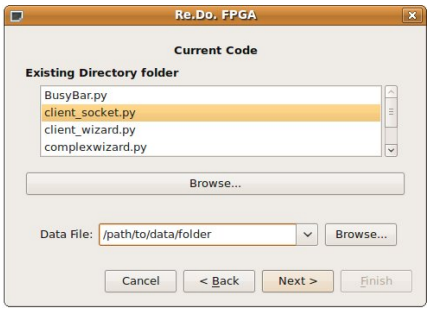
- Χρήση κώδικα χρήστη:
  Με χρήση αυτής της επιλογής, ο χρήστης μπορεί να χρησιμοποιήσει δικό του κώδικα για την δημιουργία των δεδομένων του. Επίσης μπορεί να στείλει και το φάκελο με τα αρχεία δεδομένων το οποίο επεξεργάζεται ο κώδικας του (σχήμα 5.17). Στην συνέχεια οδηγούμαστε στο τελευταίο βήμα της διαδικασίας.

Στο τελευταίο βήμα ο χρήστης αφού πατήσει το κουμπι Download BitStream θα εμφανιστούν στον πίνακα που βρίσκεται στο παράθυρο (σχήμα 5.18) τα αποτελέσματα που επιστρέφει ο κώδικας που έχει επιλέξει. Επίσης μπορεί να δει και κάποιο αποτέλεσμα από την σειριακή θύρα (RS232) αν το υποστηρίζει η σχεδίαση του.

Τέλος, αν ο χρήστης επιθυμεί να κάνει αλλαγές στην αρχιτεκτονική του, με χρήση του κουμπιού Modify Architecture Files επιστρέφει στο παράθυρο κτισίματος ενός Project και η διαδικασία πλέον επαναλαμβάνεται μέχρι το πιο πάνω βήμα, προγραμματισμός του



Σχήμα 5.15: Download BitStream και επιστροφή της εξόδου της σχεδίασης με χρήση RS232.

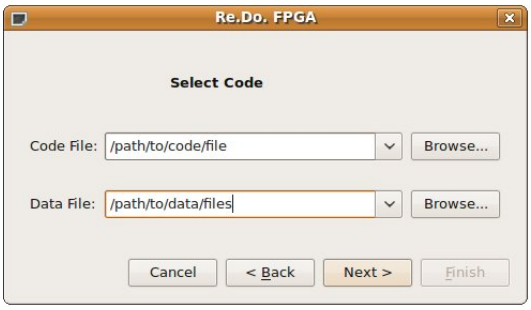


Σχήμα 5.16: Επιλογή κώδικα που βρίσκεται στον Server.

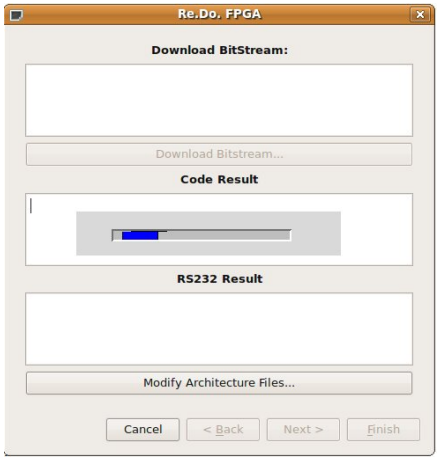
αναπτυξιακού. Όταν γίνεται αποστολή Bitstream τότε η επιλογή "Modify Architecture Files" δεν είναι διαθέσιμη.

Για την υλοποίηση των πιο πάνω χρησιμοποιούνται επίσης και Python Threads [18]. Η χρήση των Threads είναι επιτακτική ειδικά στην διαδικασία κτισίματος του Project για να έχει την δυνατότητα ο χρήστης, αν επιθυμεί, να τερματίσει την διαδικασία. Επίσης χρησιμοποιούνται για την δημιουργία progress bar στις χρονοβόρες διαδικασίες του εργαλείου (δημιουργία Bitstream, Download Bitstream, Αποστολής Bitstream).

Τα βήματα που περιγράφηκαν πιο πάνω αναπαριστώνται από το διάγραμμα ροής του σχήματος 5.19



Σχήμα 5.17: Επιλογή και αποστολή κώδικα από το χρήστη.



Σχήμα 5.18: Download BitStream και επιστροφή των αποτελεσμάτων του κώδικα και της σειριακής θύρας - RS232.

5.2.2 Λειτουργικότητα Server

Όπως αναφέρθηκε στην αρχή του κεφαλαίου, ο Server είναι δέκτης μηνυμάτων και πακέτων από τον Client ανάλογα με το εκάστοτε βήμα που βρίσκεται ο χρήστης του εργαλείου. Αρχικά εκτελείται ένας αριθμός εντολών συστήματος για την δημιουργία των κατάλληλων μεταβλητών συστήματος που χρειάζονται από τα εργαλεία της Xilinx για την σωστή λειτουργία τους καθώς επίσης και για τους Cable Driver. Τέλος γίνεται έλεγχος αν υπάρχει ο φάκελος Workspace και Code στο Home Directory του χρήστη, εάν όχι δημιουργούνται μέσω του κώδικα.

Στην συνέχεια περιγράφονται αναλυτικά τα βήματα που ακολουθεί ο Server ανάλογα με τις επιλογές του χρήστη,

- Δημιουργία καινούργιου Project:
  Αφού λάβει το μήνυμα ότι ο χρήστης επιθυμεί να δημιουργήσει καινούργιο Project ο Server πλέον περιμένει το όνομα που θα δώσει σε αυτό. Αφού το λάβει, δημιουργείται στο χώρο εργασίας (WorkSpace) καινούργιος φάκελος με το όνομα που έδωσε ο χρήστης προσθέτοντας την κατάληξη _project. Πριν όμως από την δημιουργία του φακέλου γίνεται έλεγχος αν δεν υπάρχει ήδη φάκελος με αυτό το όνομα. Αν υπάρχει τότε στέλλεται το ανάλογο μήνυμα στον Client για να δώσει ο χρήστης καινούργιο όνομα.

  Το επόμενο βήμα είναι η επιλογή από τον χρήστη του μέσου μεταφοράς των δεδομένων του (PCI-Express, Gigabit Ethernet ή και των δύο παράλληλα). Αφού ληφθεί το μήνυμα με την επιλογή, τότε αντιγράφονται στο φάκελο που δημιουργήθηκε στο προηγούμενο βήμα τα αρχεία της αντίστοιχης σχεδίασης και τα οποία βρίσκονται ήδη στον Server στο μονοπάτι /usr/src/ για να είναι διαθέσιμοι για όλους τους χρήστες.

  Στην συνέχεια γίνεται λήψη των αρχείων VHDL ή Verilog και τυχόν Ngc που χρησιμοποιούνται. Τα αρχεία αυτά τοποθετούνται στους φακέλους /pcores/peripheral_user_select/vhd/vhdl/, /pcores/peripheral_user_select/vhd/verilog/ και /pcores/peripheral_user_select/netlist/ αντίστοιχα. Επίσης γίνονται αλλαγές στα αρχεία με κατάληξη .pao (εγγραφή των αρχείων VHDL και Verilog) και .bbd (εγγραφή των αρχείων Ngc) τα οποία βρίσκονται στο μονοπάτι /pcores/peripheral_user_select/data/.

  Το τελευταίο βήμα δημιουργίας ενός καινούργιου Project είναι η διαδικασίας κτισίματος του. Δημιουργία Netlist, Bitstream, Generate Libraries and BSPs, Build All User Apllication και τέλος Update Bitstream για να συνδεθεί το hardware κομμάτι με το software μέρος. Μετά το πέρας της κάθε διαδικασίας στέλλεται στον χρήστη ένα αρχείο με το εκάστοτε αποτέλεσμα.

  Αν ληφθεί μήνυμα ότι ο χρήστης επιθυμεί να κάνει αλλαγές στους κώδικες του, τότε περιμένενει να λάβει το αρχείο κώδικα το οποίο επιθυμεί να τροποποιήσει. Μιας και η συνδεση στον Server γίνεται με την επιλογή -X , γίνεται χρήση της εντολής:

  $$\$gedit file\_name$$

  για να ευκολυνθεί ο χρήστης με τις αλλαγές που επιθυμεί να κάνει.

  Σε όλη την πιο πάνω διαδικασία χρησιμοποιείται ένα αρχείο στο οποίο γράφουμε το IP που χρησιμοποιείται και όλα τα αρχεία τα οποία έχουν ληφθεί για την δημιουργία του Project. Το αρχείο δημιουργείται για να γνωρίζουμε το μονοπάτι στο οποίο βρίσκονται τα αρχεία της αρχιτεκτονικής του χρήστη.

- Άνοιγμα έτοιμου Project:
  Με αυτή την επιλογή από τον χρήστη, ο Server είναι υπεύθυνος να στείλει στον Client τα ονόματα όλων των Project , χωρίς την κατάληξη _project που βρίσκονται στο χώρο εργάσιας του. Η κατάληξη _project μας βοηθά να γνωρίζουμε ποιοι φάκελοι περιέχουν κάποιο Project. Πλέον ο Server αναμένει από τον Client το όνομα του φακέλου που επιθυμεί ο χρήστης να δουλέψει και γίνεται αλλαγή στο μονοπάτι εργασίας. Με χρήση του αρχείου που δημιουργείται κατά την διαδικασία δημιουργίας νέου Project, αποστέλλεται στον χρήστη το μέσο μεταφοράς δεδομένων που χρησιμοποιείται.

  Αν ο χρήστης επιθυμεί να τροποποιήσει τα αρχεία του τότε ακολουθείται η διαδικασία κτισίματος ενός Project . Αν θέλει να αλλάξει τα αρχεία του, τότε πρώτα ελέγχεται αν το καινούργιο αρχείο που λαμβάνεται ανήκει ήδη στο Project. Αν ανήκει τότε διαγράφεται το παλιό αρχείο από το μονοπάτι του περιφερειακού που χρησιμοποιείται στο Project και γίνεται η δημιουργία του καινούργιου αρχείου κώδικα. Η πιο πάνω πληροφορία βρίσκεται στο αρχείο που αναφέραμε πιο πάνω.

- Αποστολή BitStream:
  Αφού ληφθεί το κατάλληλο μήνυμα, πλέον ο Server περιμένει το αρχείο. Το αρχείο αποθηκεύεται με την ονομασία download.bit στο χώρο εργασίας του χρήστη.

Όπως αναφέρθηκε κατά την περιγραφή του Client το επόμενο βήμα είναι και για τις τρεις περιπτώσεις ίδιο. Εδώ το μήνυμα που λαμβάνεται είναι κατά το πόσο θα χρησιμοποιηθεί κώδικας που βρίσκεται ήδη στον Server, αν θα σταλεί κώδικας του χρήστη ή αν δεν θα χρησιμοποιηθεί.

- Δεν θα χρησιμοποιηθεί κώδικας:
  Με αυτή την επιλογή η διαδικασία οδηγείται στο τελευταίο βήμα της, αφού πλέον το επόμενο μήνυμα που θα ληφθεί θα είναι για να κατέβει η σχεδίαση στην αναδιατασσόμενη συσκευή. Αφού κατέβει η αρχιτεκτονική στέλλεται στον χρήστη το αποτέλεσμα της διαδιακασίας και στην συνέχεια τα αποτελέσματα τα οποία λαμβάνονται από την σειριακή θύρα (RS232) και είναι η έξοδος της αρχιτεκτονικής. Πλέον το μόνο που επομένει είναι να κλείσει η σύνδεση ή να ζητηθεί τροποποίηση των αρχείων της αρχιτεκτονικής, εκτός και αν έγινε αποστολή Bitsream.

- Χρήση κώδικα που βρίσκεται στον Server:
  Αρχικά στέλλονται στον Client τα ονόματα των αρχείων που βρίσκονται στο χώρο εργασίας του χρήστη και πιο συγκεκριμένα στον φάκελο Code. Στην συνέχεια λαμβάνεται το όνομα του κώδικα που θα χρησιμοποιηθεί και ο φάκελος δεδομένων

που θα χρησιμοποιήσει ο χρήστης. Αν ο κώδικας που θα χρησιμοποιηθεί είναι ένας από τους κώδικες που υλοποιήθηκε στα πλαίσια της διπλωματικής εργασίας τότε τα δεδομένα αποθηκεύονται με την ονομασία data.txt στον φάκελο UseCurrentCode.

- Χρήση κώδικα χρήστη:
  Αφού υποδείξει το αρχείο κώδικα, στέλλεται στον Server και αποθηκεύεται στο χώρο εργασίας του, στον φάκελο Code. Για την αποστολή των δεδομένων ακολουθούμε την ίδια διαδικασία με πιο πάνω.

Για τις περιπτώσεις που ο χρήστης επιθυμεί να χρησιμοποιήσει κώδικα, ο Server περιμένει να πάρει μήνυμα για να κατέβει η σχεδίαση στην αναδιατασσόμενη συσκευή. Αφού επιστραφεί το αποτέλεσμα της διαδιακασίας στον Client εκτελείται ο κώδικας που έχει επιλέξει ο χρήστης και επιστρέφεται η έξοδος της αρχιτεκτονικής μέσω της σειριακής θύρας (RS232) στον Client. Επίσης αν γίνεται χρήση του PCI-Express μπορεί να γίνει χρήση και αυτού του μέσου για επιστροφή των αποτελεσμάτων. Το εργαλείο υποστηρίζει την εκτέλεση κωδίκων C, Python και Perl.

Τέλος, το επόμενο μήνυμα που θα λάβει ο Server είναι του τερματισμού της σύνδεσης ή ότι ο χρήστης επιθυμεί να κάνει κάποιες αλλαγές στην αρχιτεκτονική του (εκτός και αν έγινε αποστολή Bitsream). Αν ληφθεί το μήνυμα για αλλαγή της αρχιτεκτονικής τότε οδηγούμαστε στο βήμα κτισίματος ενός Project και η διαδιακασία επαναλαμβάνεται μέχρι το πιο πάνω βήμα.

Οι λειτουργίες που περιγράφηκαν πιο πάνω αναπαριστώνται από το διάγραμμα ροής του σχήματος 5.20

Όλες οι εντολές που αναφέρθηκαν πιο πάνω που έχουν σχέση με το εργαλείο της Xilinx EDK, Generate Netlist, Generate Bitstream, Generate Libraries and BSPs, Build All User Apllication, Update Bitstream και Download Bitstream εκτελούνται από τις πιο κάτω εντολές:[19][20]

- Generate Makefiles:

$$\$xps\ -nw\ system.xmp$$

$$\%make$$

$$\%save\ make$$

- Generate Netlist:

$$\$make\ -f\ system.make\ netlist$$

- Generate Bitstream:

$$\$make\ -f\ system.make\ bit$$

- Generate Library Files:

$$\$make\ -f\ system.make\ libs$$

- Build User Application:

$$\$make\ -f\ system.make\ program$$

- Initialise BRAM:

$$\$make\ -f\ system.make\ init\_bram$$

- Download Bitstream:

$$\$make\ -f\ system.make\ download$$

Στην περίπτωση που ο χρήστης επίλεξε την διαδικασία αποστολής Bitstream τότε για να κατέβει στην αναδιατασσόμενη λογική χρησιμοποιούμε την εντολή:

- Download Bitstream Using Impact:

$$\$impact\ -batch\ download.cmd$$

όπου το περιεχόμενο του αρχείου download.cmd είναι:

```
setMode -bscan
setCable -p auto
identify
assignfile -p 5 -file download.bit
program -p 5
quit
```

Οι εντολές που χρησιμοποιούνται για την δημιουργία των μεταβλητών συστήματος είναι:

os.environ["XILINX"]="/opt/Xilinx/10.1/ISE"
os.environ["XILINX_EDK"]="/opt/Xilinx/10.1/EDK"
os.environ["LMC_HOME"]="/opt/Xilinx/10.1/ISE/smartmodel/lin/installed_lin"
os.environ["PATH"]="/opt/Xilinx/10.1/ISE/bin/lin:"+os.environ["PATH"]
os.environ["PATH"]="/opt/Xilinx/10.1/EDK/bin/lin:/opt/Xilinx/10.1/EDK/lib/lin:"
+os.environ["PATH"]
os.environ["LD_LIBRARY_PATH"]="/opt/Xilinx/10.1/ISE/lib/lin:/usr/X11R6/lib:"

```
                                    +os.getenv("LMC_HOME")+"/lib/linux.lib:"
                                    +os.getenv("LMC_HOME")+"/lib/amd64.lib"
os.environ["LD_LIBRARY_PATH"]="/opt/Xilinx/10.1/EDK/lib/lin:"+os.environ
                                    ["LD_LIBRARY_PATH"]
os.environ["LD_PRELOAD"]="/usr/src/Cable_Driver/libusb-driver.so"
```

Αν ο χρήστης επιλέξει χρήση του PCI-Express, αρχικά γίνεται έλεγχος αν είναι εγκατεστημένος ο driver που υλοποιήθηκε στα πλαίσια της διπλωματικής εργασίας του Ιωάννη Καρτσωνάκη. Σε διαφορετική περίπτωση εκτελείται ένα .sh script:

```
#/bin/bash

cp -R /usr/src/beta_driver /usr/src/linux-2.6.28.7/drivers/pci/pcie/
cp -R /usr/src/beta_mmap /usr/src/linux-2.6.28.7/drivers/pci/pcie/beta_driver/
cd /usr/src/linux-2.6.28.7/
make
make modules
```

Τέλος πρέπει να δημιουργηθεί η διεργασία στο σύστημα η οποία θα μας επιτρέψει την χρήση του PCI-Express. Γι΄ αυτό τον σκοπό δημιουργήθηκε ένα .sh script το οποίο εκτελείται κατά την εκκίνηση του υπολογιστή με τις ακόλουθες εντολές:

```
#/bin/bash

insmod /usr/src/linux-2.6.28.7/drivers/pci/pcie/beta_driver/
        Xilinx_pcix_driver.ko
mknod /dev/Xilinx_pcix c ‘more /proc/devices | grep Xilinx |
        awk ’printf "%d", strtonum($0)‘ 0
```

Στην συνέχεια, κατά την εκτέλεση του κώδικα εκτελούνται οι εντολές:[21]

```
rmmod pciehp
modprobe pciehp pciehp_force=1
```

Η χρήση των πιο πάνω εντολών επιβάλλεται μιας και το λειτουργικό σύστημα ελέγχει τις PCI-Express θύρες μόνο κατά την εκκίνηση του υπολογιστή. Για να αποφευχθεί η διαδικασία (επανεκκίνηση υπολογιστή) γίνεται επανεκκίνηση του hotplug. Το hotplug είναι υπεύθυνο για όλες τις PCI-Express συσκευές που χρησιμοποιεί ένας υπολογιστής.

Αυτό επιτυγχάνεται, όπως φαίνεται και από τον κώδικα, με διαγραφή της διεργασίας και η υποχρεωτική επανεκκίνηση της.

Όταν γίνεται χρήση του Gigabit Ethernet χρειάζεται να εκτελεστεί η εντολή:

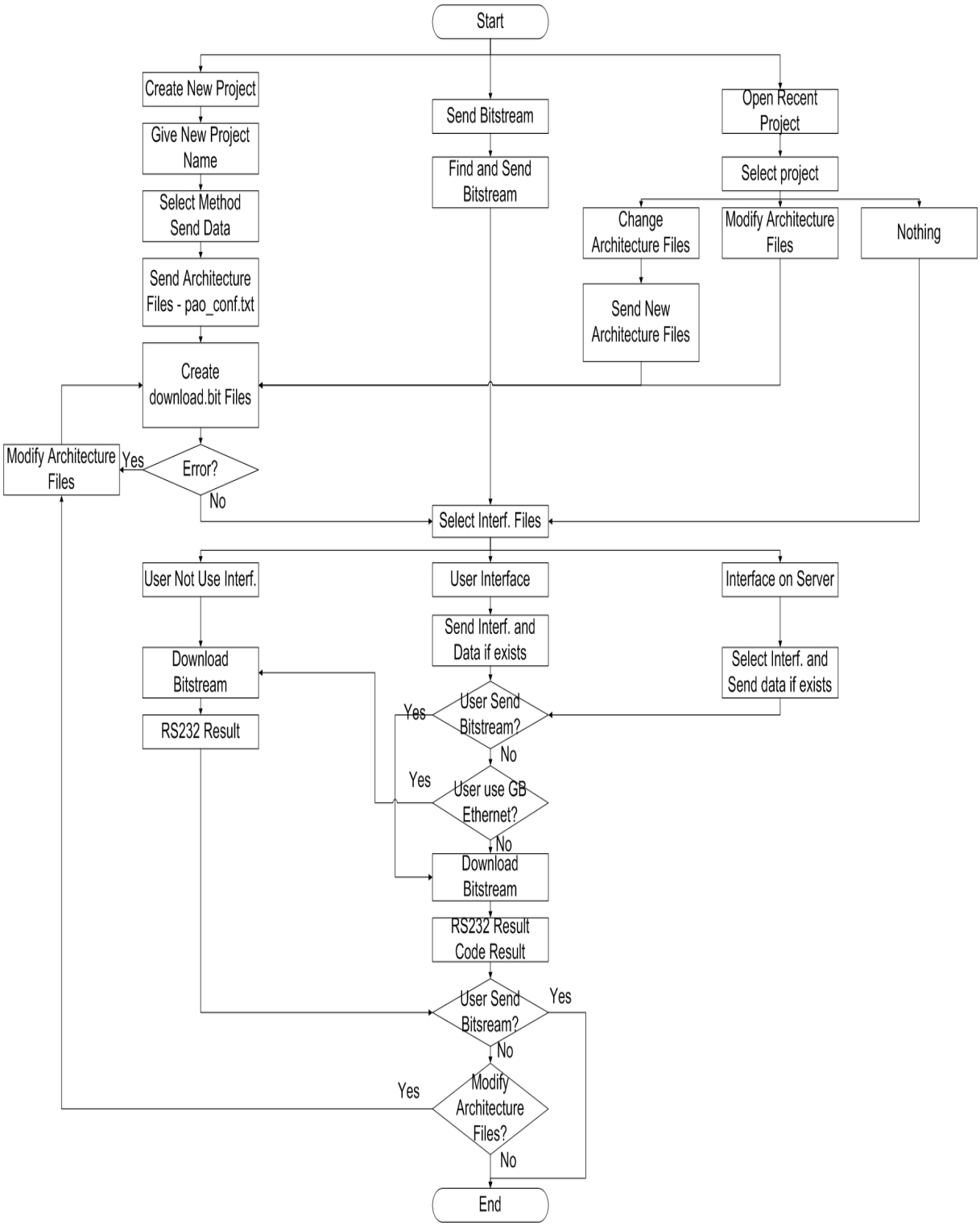
arp -s 1.2.3.5 66:66:66:66:66:66

όπου eth1 είναι οι ρυθμίσεις της σύνδεση στον υπολογιστή,

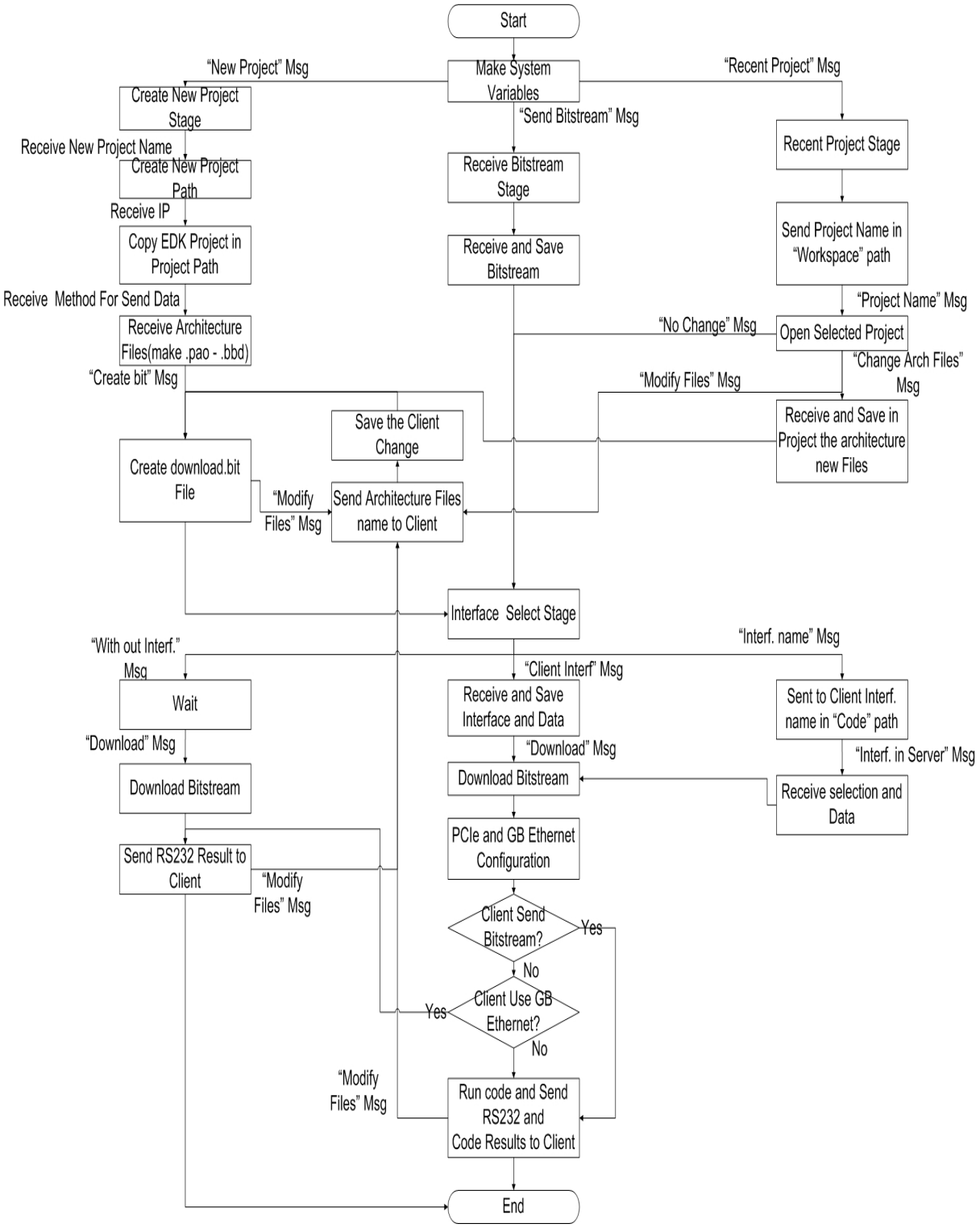
IP Address	:	1.2.3.9
Broadcast Address	:	1.2.3.254
Subnet Mask	:	255.255.255.0

Για το λόγο ότι οι πιο πάνω εντολές που περιγράφηκαν κάνουν επεμβάσεις στον φάκελο συστήματος του λειτουργικού αναγκαστικά η εκτέλεση του κώδικα στον Server πρέπει να εκτελείται αφού πρώτα αποκτηθούν δικαιώματα sudo.

Όλα τα πιο πάνω αρχεία που αναφερθήκαμε βρίσκονται στο μονοπάτι /usr/src/. Έχουν τοποθετηθεί σε αυτό για να είναι εύκολη η πρόσβαση σε όλους τους χρήστες και να εκτελούνται όλοι οι κώδικες πληκτρολογώντας στην κονσόλα απλώς το όνομα τους.



Σχήμα 5.19: Διάγραμμα ροής κώδικα λειτουργίας Client.



Σχήμα 5.20: Διάγραμμα ροής κώδικα λειτουργίας Server.

Κεφάλαιο 6

Επιβεβαίωση ορθής λειτουργίας - Παραδείγματα

Η επιβεβαίωση της σωστής λειτουργίας του εργαλείου Re.Do. FPGA χωρίστηκε σε δύο φάσεις. Στην πρώτη φάση ελέχθηκε η σωστή επικοινωνία μεταξύ Client - Server και των λειτουργιών του εργαλείου και σε δεύτερη φάση η σωστή λειτουργία των τριών σχεδιάσεων που υλοποιήθηκαν.

6.1 Επιβεβαίωση λειτουργείας μοντέλου Client - Server

Για τον σκοπό αυτό έγινε έλεγχος όλων των λειτουργιών του εργαλείου με σκοπό την αποφυγή σφαλμάτων κατά την εκτέλεση των διαφόρων εργασιών. Επίσης έγινε έλεγχος του εργαλείου από συναδέλφους του εργαστηρίου Μικροεπεξεργαστών και Υλικού και από συναδέλφους οι οποίοι έχουν απλώς στοιχειώδης γνώσεις από υλοποίηση αρχιτεκτονικών.

Η κύρια λειτουργία που ελεχθηκε από τους συναδέλφους του εργαστηρίου ήταν η δημιουργία καινούργιου Project ενώ από τους υπόλοιπους συναδέλφους η λειτουργία αποστολής Bitstream. Για τον έλεγχο της αποστολής Bitstream υλοποιήθηκε μία απλή σχεδίαση η οποία υπολογίζει το παραγοντικό μιας τιμής που δίνεται ως όρισμα.

Με αυτό τον τρόπο βελτιώθηκε η επικοινωνία μεταξύ Client - Server μιας και αντιμετωπίστηκαν σφάλματα που οφείλονταν σε λάθος βήματα κατά την διαδικασία δημιουργίας ενός project. Τέλος προστέθηκε επιπρόσθετη λειτουργία (αλλαγή και τροποποίηση αρχείων που βρίσκονται στον Server) η οποία ήταν εισήγηση τους.

6.2 Επιβεβαίωση λειτουργείας Σχεδίασεων

Για την επιβεβαίωση της ορθής λειτουργίας των σχεδιάσεων χρησιμοποιήθηκε το εργαλείο Xilinx Chipscope Pro 10.1, με το οποίο έγινε δειγματοληψία των σημάτων των σχεδιάσεων στην FPGA. Επίσης υλοποιήθηκε σε γλώσσα περιγραφής υλικού (VHDL) ένας αθροιστής ο οποίος προσαρμόστηκε στην κάθε σχεδίαση που υλοποιήθηκε. Για την δημιουργία των σχεδιάσεων χρησιμοποιήθηκε το εργαλείο Re.Do. FPGA .Επίσης, χρησιμοποιήθηκε η λειτουργία για τροποποίηση των αρχείων της σχεδίασης για αλλαγές στην σχεδίαση.

Στην σχεδίαση για χρήση του GB Ethernet τα δεδομένα που λαμβάνονται έχουν μέγεθος οκτώ bits. Για αυτό το λόγο ο αθροιστής που υλοποιήθηκε δέχεται ως είσοδο ένα σήμα οκτώ bits και η έξοδος είναι ένα σήμα τριάντα δύο bits.

6.2.1 Επιβεβαίωση λειτουργείας Σχεδίασης χρήσης πρωτοκόλλου MTP

Αφού υλοποιήθηκε σε VHDL ο αθροιστής και η μονάδα ελέγχου των σημάτων του MTP έγινε χρήση του εργαλείο Re.Do. FPGA για την δημιουργία της σχεδίασης στον Server. Για την μεταφορά δεδομένων υλοποιήθηκε σε γλώσσα Python ένα UDP socket και στάλθηκαν στην FPGA συνολικά ένα GByte ASCII χαρακτήρες σε UDP πακέτα μεγέθους 1 KByte.

Όπως αναφέρθηκε και στο κεφάλαιο 5.1 το MTP δεν παρέχει λειτουργικότητα για επιστροφή τιμών. Για αυτό χρησιμοποήθηκε η σειριακή θύρα RS232 για την επιστροφή του αποτελέσματος. Επίσης, χρησιμοποιήθηκε και το εργαλείο Chipscope pro για δειγματοληψία των σημάτων ελέγχου που χρησιμοποιούνται για διαπίστωση της σωστής λειτουργίας τους.

Για να επιβεβαιωθεί η σωστή λειτουργία και η μη απώλεια πακέτων, εκτός από το εργαλείο Chipscope pro, έγινε σύγκριση του αποτελέσματος του αθροίσματος των δεδομένων των πακέτων με χρήση Software με το αποτέλεσμα που επιστρέφεται από την σχεδίαση στον Client με χρήση της σειριακής θύρας RS232.

Κατά την αποστολή ενός GByte ASCII χαρακτήρες διαπιστώθηκε απώλεια πακέτων λόγω υπερχείλισης των FIFO. Για αυτό το λόγο, για την επιβεβαίωση της σωστής λειτουργίας της σχεδίασης στάλθηκε μικρότερος όγκος δεδομένων, της τάξης των MByte.

6.2.2 Επιβεβαίωση λειτουργείας Σχεδίασης χρήσης οδηγού PCI-Express

Όπως και πιο πάνω υλοποιήθηκε η μονάδα ελέγχου των σημάτων της αρχιτεκτονικής για χρήση του PCI-Express και με χρήση του ίδιου αθροιστή υλοποιήθηκε η σχεδίαση με χρήση του Re.Do. FPGA. Για την μεταφορά των δεδομένων υλοποίθηκε σε γλώσσα C το πρόγραμμα για επικοινωνία με το PCI-Express Slot του ηλεκτρονικού υπολογιστή. Στάλθηκαν στην FPGA συνολικά 1 GByte σε ποσότητες των 32 bits από της οποίες ο αθροιστής εκμεταλλεύεται τα τελευταία οκτώ bits.

Για την επιστροφή του αποτελέσματος χρησιμοποιήθηκε η δυνατότητα επιστροφής τιμών που μας παρέχει ο οδηγός του PCI-Express. Το αποτέλεσμα αυτό συγκρίθηκε με αυτό που επιστρέφεται από το Software. Επίσης και εδώ έγινε χρήση του εργαλείου Chipscope pro για δειγματοληψία των σημάτων ελέγχου που χρησιμοποιούνται για διαπίστωση της σωστής λειτουργίας τους.

Στην συνέχεια υλοποιήθηκε και μια δεύτερη σχεδίαση η οποία αντί να χρησιμοποιεί το ρολόι του διαύλου PCI-Exrpess (62.5 MHz) στις FIFO εισόδου και εξόδου δεδομένων και στην σχεδίαση, χρησιμοποιείται το ρολόι του συστήματος (120 MHz). Και εδώ το αποτέλεσμα που επιστράφηκε από την σχεδίαση είναι το ίδιο με αυτό που επιστράφηκε από το Software.

Η πιο πάνω δυνατότητα είναι εφικτοί μιας και μπορεί να χρησιμοποιηθούν διαφορετικά ρολόγια ανάγνωσης και εγγραφής στις FIFO εισόδου και εξόδου δεδομένων.

Τέλος, η προπτυχιακή φοιτήτρια Ναυσικά Χρυσάνθου χρησιμοποίησε το Re.Do. FPGA για την υλοποίηση της σχεδίασης της. Με χρήση του PCI-Express, για σειριακή επικοινωνία με την FPGA, υλοποίησε ένα πλήρες σύστημα το οποίο εναλλάσεται η λειτουργία Software με Hardware.

6.2.3 Επιβεβαίωση λειτουργείας Σχεδίασης παράλληλης χρήσης GB Ethernet - PCI-Express

Με χρήση της διεπαφής που περιγράφηκε στο κεφάλαιο 5.1 υλοποιήθηκε η μονάδα ελέγχου των σημάτων του MTP και της αρχιτεκτονικής χρήσης του PCI-Experss για παράλληλη χρήση του πρωτοκόλλου GB Ethernet και οδηγού PCI-Express και με την χρήση του εργαλείου Re.Do. FPGA υλοποιήθηκε η σχεδίαση στον Server. Η σχεδίαση αποτελείται από δύο αθροιστές, ο ένας χρησιμοποιεί τα δεδομένα που λαμβάνονται με την χρήση του PCI-Express και ο δεύτερος τα δεδομένα που λαμβάνονται από το GB Ethernet. Στάλθηκαν στην FPGA συνολικά 1 GByte δεδομένων.

Για την αποστολή των δεδομένων έγινε ενας συνδιασμός των προηγούμενων δύο προ-

γραμμάτων, σε C και Python. Στάλθηκαν τα πρώτα 512 MByte σε UDP πακέτα μεγέθους 1 KByte με χρήση του GB Ethernet και τα υπόλοιπα 512 MByte σε ποσότητες των 32 bits με χρήση του PCI-Express.

Όταν στάλθηκε ο πιο πάνω όγκος δεδομένωσν με χρήση του MTP διαπιστώθηκε και σε αυτή την περίπτωση απώλεια πακέτων. Όταν χρησιμοποιήθηκε το ρολόι του διαύλου PCI-Exrpess (62.5 MHz) τότε παρατηρήθηκε μεγαλύτερη απώλεια πακέτων μιας και τα δεδομένα καταναλώνονταν με μικρότερο ρυθμό από αυτόν που κατέφθαναν. Κατά την αποστολή των δεδομένων με χρήση του οδηγού PCI-Express δεν διαπιστώθηκε απώλεια δεδομένων.

Κεφάλαιο 7

Πειραματικά Αποτελέσματα - Ευχρηστία Re.Do. FPGA

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των πειραματικών αποτελεσμάτων της ταχύτητας μεταφοράς των δύο σειριακών μέσων μεταφοράς δεδομένων καθώς και η ευχρηστία του Re.Do. FPGA.

7.1 Πειραματικά Αποτελέσματα

Στα πειράματα μας χρησιμοποιήθηκε ένας Ηλεκτρονικός Υπολογιστής με επεξεργαστή Intel Core 2 2.8 GHz, 1 GByte RAM με Ethernet PHY 1000/100/10 Mbps και λειτουργικό σύστημα Ubuntu 8.10.

Στο αναπτυξιακό στάλθηκαν δεδομένα διαφόρων όγκων και έγινε έλεγχος του αποτελέσματος που επιστράφηκε από την αρχιτεκτονική που υλοποιήθηκε για πιστοποίηση της λειτουργικότητας, ένας αθροιστής, με αυτό που επιστρέφει το software.

Οι μετρήσεις που πάρθηκαν ήταν ο απαιτούμενος χρόνος για να σταλούν δεδομένα στο αναπτυξιακό και να επιστραφεί το άθροισμα τους από τον αθροιστή. Από τους χρόνους αυτούς και με την χρήση της σχέσης:

$$Trasfer\_Rate(Mbps) = \frac{Data\_Transfer(MB) * 2^{20} * 8}{1000^2 * time}$$

μετρήθηκε ο ρυθμός μεταφοράς δεδομένων.

7.1.1 Αποτελέσματα Χρήσης πρωτοκόλλου MTP

Για να χρησιμοποιηθεί το Gigabit Ethernet για την αποστολή των δεδομένων υλοποιήθηκε ένας Python κώδικας μέσω του οποίου χρησημοποιείται ένα UDP Socket. Οι χρόνοι που

πάρθηκαν φαίνονται στον πίνακα 7.1. Για λήψη του αποτελέσματος χρησιμοποιήθηκε η δυνατότητα που μας παρέχει ο οδηγός του PCI-Experss. Αυτό ήταν αναγκαίο μιας και είναι αδύνατο να μετρηθεί ο χρόνος όταν η λήψη γίνεται με χρήση της σειριακής θύρας RS-232. Η διαφορά που θα είχαμε αν γινόταν χρήση της RS-232 θα ήταν μία επιπλέον καθυστέρηση μιας και έχει χαμηλό ρυθμό μετάδοσης δεδομένων. Η σχεδίαση στην οποία μετρήθηκαν οι πιο κάτω χρόνοι καταναλώνει ένα δεδομένο κάθε κύκλο και επιστέφει ένα δεδομένο, το οποίο είναι το άθροισμα των τιμών που στάλθηκαν, με το τέλος του κάθε πακέτου, όταν η τιμή της εισόδου είναι 255 ("11111111"). Ο όγκος δεδομένων που στάλθηκε είναι χωρίς της επιπρόσθετης πληροφορίας που χρειάζεται να προστεθεί στο κάθε πακέτο.

Όγκος Δεδομένων	**Χρόνος (sec)**	**Ταχύτητα Μεταφοράς Δεδομένων (Mbps)**
64 MByte	1.96527	324.94
128 MByte	3.1692	338.45
256 MByte	6.3531	338.02
512 MByte	13.2569	323.98
1024 Mbyte	25.0725	342.6

Πίνακας 7.1: Πειραματικές Μετρήσεις χρήσης Gigabit Ethernet.

Στα πειράματα που εκτελέστηκαν παρατηρήθηκε απώλεια πακέτων όταν μεταφερόταν μεγάλος όγκος δεδομένων και η οποία ήταν διαφορετική σε κάθε εκτέλεση. Αυτό συμβαίνει όταν ο ηλεκτρονικός υπολογιστής δεν διαθέτει επάρκεια σε μνήμα RAM μιας και υπάρχουν σφάλματα σελίδας κατά την διάρκεια της μετάδοσης των δεδομένων καθώς επίσης και λόγω υπερχείλισης των FIFO.

7.1.2 Αποτελέσματα Χρήσης οδηγού PCI-Express

Όπως και κατά την χρησιμοποίηση του Gigabit Ethernet υλοποιήθηκε ένας C κώδικας μέσω του οποίου αποστέλλονται τα δεδομένα στο αναπτυξιακό. Οι χρόνοι που χρειάστηκαν για να ληφθεί το αποτέλεσμα του αθροίσματος φαίνονται στον πίνακα 7.2. Για την λήψη του αποτελέσματος χρησιμοποιήθηκε η δυνατότητα του οδηγού PCI-Express για επιστροφή δεδομένων από το αναπτυξιακό. Όπως και πιο πάνω, η σχεδίαση καταναλώνει ένα δεδομένο ανά κύκλο και επιστρέφεται το αποτέλεσμα της πρόσθεσης των τιμών που στάλθηκαν όταν η τιμή της εισόδου είναι 255 ("11111111").

Για εύρεση της ταχύτητας μεταφοράς δεδομένων από το αναπτυξιακό στον υπολογιστή χρησιμοποιήθηκε μια δεύτερη σχεδίαση η οποία επιστρέφει τις τιμές εισόδου. Σε αυτή την περίπτωση ο μέγιστος όγκος δεδομένων που μπορεί να ληφθεί είναι 16KB. Ο περιο-

Όγκος Δεδομένων	Χρόνος (sec)	Ταχύτητα Μεταφοράς Δεδομένων (Mbps)
64 MByte	2.090	256.88
128 MByte	4.180	256.88
256 MByte	8.360	256.88
512 MByte	16.780	255.96
1024 Mbyte	33.410	257.11

Πίνακας 7.2: Πειραματικές Μετρήσεις χρήσης PCI-Express (Download).

ρισμός αυτός οφείλεται στην υλοποίηση του οδηγού PCI-Express και της αρχιτεκτονικής χρήσης του. Η ταχύτητα μεταφοράς δεδομένων για τον συγκεκριμένο όγκο δεδομένων υπολογίστηκε με την πιο πάνω σχέση στα 9.3Mbps.

Στα πειράματα που εκτελέστηκαν δεν παρατηρήθηκε απώλεια δεδομένων, ούτε κατά την διαδικασία μεταφοράς δεδομένων από τον Η/Υ στο αναπτυξιακό, ούτε κατα την διαδικασία μεταφοράς δεδομένων από το αναπτυξιακό στον Η/Υ (μέχρι 16KB).

7.1.3 Αποτελέσματα Παράλληλης Χρήσης Gigabit Ethernet και PCI-Express

Έγινε ένας συνδυασμός των κωδίκων που χρησιμοποιήθηκαν πιο πάνω και στάλθηκαν ταυτόχρονα δεδομένα (χρήση δύο διεργασιών) με το Gigabit Ethernet και το PCI-Express. Για την λήψη του αποτελέσματα χρησιμοποιήθηκε το PCI-Express. Οι χρόνοι που χρειάστηκαν για την λήψη του αποτελέσματος φαίνονται στον πίνακα 7.3. Πλέον στην σχεδίαση χρησιμοποιούνται δύο αθροιστές. Ο πρώτος εκμεταλλεύεται τα δεδομένα που λαμβάνονται μέσω του πρωτοκόλλου MTP και ο δεύτερος μέσω του οδηγού PCI-Express. Η επιστρεφόμενη τιμή είναι το άθροισμα των επιμέρους αθροισμάτων. Επιστρέφεται τιμή όταν όταν η τιμή που διαβάζεται από τον οδηγό του PCI-Express έχει την τιμή 255.

Στα πειράματα που εκτελέστηκαν παρατηρήθηκε ότι και κατά την χρήση ενός εκ των δύο μέσων. Κατά την χρήση του πρωτοκόλλου MTP παρατηρήθηκε απώλεια πακέτων κατά την μεταφορά μεγάλου όγκου δεδομένω, ενώ κατά την χρήση του οδηγού PCI-Express δεν παρατηρήθηκε απώλεια δεδομένων.

Όγκος δεδομένων (PCI-Express)	Όγκος δεδομένων (Gigabit Ethernet)	Χρόνος (sec)	Ταχύτητα Μεταφοράς Δεδομένων (Mbps)
64 MByte	64 MByte	2.190	490.29
128 MByte	128 MByte	4.100	523.78
256 MByte	256 MByte	8.350	514.37
512 MByte	512 MByte	16.775	512.07
1024 Mbyte	1024 MByte	33.510	512.68

Πίνακας 7.3: Πειραματικές Μετρήσεις παράλληλης χρήσης Gigabit Ethernet - PCI-Express.

7.2 Ευχρηστία Re.Do. FPGA

Για να διαπιστωθεί κατά το πόσο το Re.Do. FPGA είναι εύχρηστο εργαλείο ζητήθηκε η γνώμη φοιτητών από το Εργαστήριο Μικροεπεξεργαστών και Υλικού καθώς και από φοιτητές του τμήματος οι οποίοι όμως έχουν μια στοιχιώδη γνώση υλοποιήσεων απλών αρχιτεκτονικών.

Αρχικά στους φοιτητές δόθηκαν οι σχεδίασης που περιγράφηκαν πιο πάνω, με τις οποίες λήφθηκαν τα πειραματικά αποτελέσματα, και με την χρήση του Re. Do. FPGA υλοποίησαν τις αντίστοιχες σχεδιάσεις και έλαβαν τα αποτελέσματα από αυτές. Στην συνέχεια τους ζητήθηκε να υλοποιήσουν μια απλή σχεδίαση (επιστροφή τιμής εισόδου) για χρήση του οδηγού PCI-Express και η οποία υλοποιήθηκε πάλι με χρήση του Re. Do. FPGA. Τα αποτελέσματα της δοκιμής ήταν απόλυτα ικανοποιητικά καθώς όλοι ολοκλήρωσαν με επιτυχία την δοκιμή.

Τέλος, η χρήση του Re.Do FPGA αυτοματοποιεί την διαδικασία εισαγωγής σειριακών μέσων μεταφοράς σε αρχιτεκτονικές. Με αυτό τον τρόπο μειώνεται ο χρόνος που απαιτείται για την υλοποίηση της επικοινωνίας μεταξύ αναδιατασσόμενης συσκευής και ηλεκτρονικού υπολογιστή.

Με αυτό τον τρόπο, αντιμετωπίστηκε το πρόβλημα εισόδου/εξόδου που αντιμετώπιζαν οι περισσότερες αρχιτεκτονικές. Έτσι πλέον είναι εφικτή η υλοποίηση πλήρων συστημάτων (ταυτόχρονη χρήση software-hardware) μιας και με την χρήση του Re.Do. FPGA αντιμετωπίστηκε και το πρόβλημα της πολυπλοκότητας επικοινωνίας μεταξύ της αρχιτεκτονικής του χρήστη με την αρχιτεκτονική χρήσης ενός εκ των δύο μέσων σειριακής επικοινωνίας.

Τέλος, με την χρήση του Server αντιμετωπίζεται και η ανάγκη των εξειδικευμένων ρυθμίσεων που χρειάζονται στον ηλεκτρονικό υπολογιστή για να είναι δυνατή η χρήση του οδηγού PCI-Express, όπως η αλλαγή του πυρήνα του λειτουργικού συστήματος Linux.

Κεφάλαιο 8

Συμπεράσματα και Μελλοντική Εργασία

Στο τελευταίο αυτό κεφάλαιο αναφέρονται τα συμπεράσματα και μελλοντικές επεκτάσεις του συστήματος μας.

8.1 Συμπεράσματα

Σε αυτή την διπλωματική εργασία μελετήθηκε η δυνατότητα εύκολης χρησιμοποίησης μέσων σειριακής επικοινωνίας υψηλής ταχύτητας, PCI-Express και Gigabit Ethernet για μεταφορά δεδομένων από τον ηλεκτρονικό υπολογιστή σε μια αναδιατασσόμενη συσκευή της οικογένειας Virtex-5. Στα πλαίσια αυτά υλοποιήθηκε ένα εύχρηστο εργαλείο, το Re.Do. FPGA, το οποίο με χρήση του μοντέλου εξυπηρετητή - πελάτη μπορεί ο χρήστης να χρησιμοποιήσει τα πιο πάνω μέσα σειριακής επικοινωνίας. Τέλος, επιβεβαιώθηκε η σωστή λειτουργία του Re.Do. FPGA και συνδέθηκε η σχεδίαση σε ένα μεγαλύτερο σύστημα.

8.2 Μελλοντική Εργασία

Σαν μελλοντική εργασία προτίνονται τα ακόλουθα:

- Το Re.Do. FPGA να υποστηρίζει την ύπαρξη στον Server περισσότερες από μία αναδιατασσόμενη συσκευή της οικογένειας Virtex-5. Αυτό θα είναι εφικτό όταν τροποποιηθεί κατάλληλα ο οδηγός του PCI-Express μιας και υποστηρίζεται μόνο η λειτουργία του ενός PCI-Express slot. Στην συνέχεια πρέπει να τροποποιηθούν οι Cable Driver που είναι υπεύθυνοι για την εισαγωγή σχεδιάσεων στο αναπτυξιακό.

Η αλλαγή αυτή πρέπει να γίνει μιας και πρέπει οι διευθύνσεις των USB να είναι σταθερές για να είναι γνωστό με ποιο USB port έχει επικοινωνία το κάθε αναπτυξιακό. Τέλος, πρέπει να γίνει αλλαγή στο κώδικα Python του Client και του Server μιας και χρησιμοποιείται μόνο μία πόρτα (ports) για την επικοινωνία τους.

- Η δυνατότητα χρήσης διαφορετικού τύπου αναδιατασσόμενων συσκευών από τον χρήστη του Re.Do. FPGA. Και σε αυτή την επέκταση πρέπει να γίνουν τροποποιήσεις στους Cable Driver και στους Python κώδικες του Client και του Server.
- Να γίνει τροποποίηση των κωδίκων για το PCI-Express και του Gigabit Ethernet με σκοπό την χρησιμοποίηση τους σε περισσότερες αναδιατασσόμενες συσκευές Virtex-5 και Virtex-6.
- Να αναπτυχθούν οι αντίστοιχοι κώδικες με χρήση του εργαλείου Xilinx ISE Design Suite 12.1 και Xilinx EDK 12.1 η οποία είναι η τελευταία έκδοση Xilinx που εκδόθηκε.
- Να δίνεται η δυνατότητα στον χρήστη του Re.Do. FPGA να χρησιμοποιεί το εργαλείο Xilinx Chipscope Pro.

Βιβλιογραφία

[1] Ioannis Kartsonakis. **Develepment Driver for PCIe and VHDL code for serial communication between PC - FPGA**. In *MSC Diploma Thesis, ECE Dpt. Technical Univercity Of Crete*, 2009.

[2] Dimitrios Vasilopoulos. **Study and experimental device for high speed serial communication with FPGA**. In *MSC Diploma Thesis, ECE Dpt. Technical Univercity Of Crete*, 2009.

[3] Sun Microsystem. *Distributed Application Architecture*, 2009. `http://java.sun.com/developer/Books/jdbc/ch07.pdf`.

[4] Parallel Distributed Processing Laboratory (PDP Lab), Department of Applied Informatics, University of Macedonia. *CLIENT-SERVER COMPUTING*, 2009. `http://www.it.uom.gr/project/client_server/theoria1.htm#`.

[5] University of Massachusetts James F. Kurose and Polytechnic University Brooklyn Keith W. Ross. *Computer Networking, 5th ed.* Pearson Education, Inc, Boston, USA, 2010.

[6] Douglas E. Comer. *Internetworking with TCP/IP:Principles, Protocols, and Architecture. 1 (5th ed.)*. Prentice Hall, 2006.

[7] Wikipedia, the free encyclopedia. *UDP*, 2010. `http://el.wikipedia.org/wiki/UDP`.

[8] T. Ylonen and Ed. C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4252, January 2006.

[9] Python Software Foundation. *General Python FAQ*, 2010. `http://docs.python.org/faq/general.html#id1`.

[10] ZetCode. *Introduction to PyQt4 toolkit*, 2007. http://zetcode.com/tutorials/pyqt4/introduction/.

[11] M. Jones, L. Scharf, J. Scott, C. Twaddle, M. Yaconis, K. Yao, P. Athanas, and B. Schott. **Implementing an API for distributed adaptive computing systems**. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 222–230, 1999.

[12] P.M. Athanas, S.F. Midkiff, K. Yao, T. Jones, M. Athanas, and F. Midkiff. **Implementing an Application Programming Interface for Distributed Adaptive Computing Systems**. 2000.

[13] A.D.D.E.T. Kyriakides and G. Chania. **A Universal Low Cost Run-Time and Programming Environment for Reconfigurable Computing**. *Shortening the path from specification to prototype*, page 2, 2003.

[14] Python Software Foundation. *Python v2.6.5 documentation*, 2010. http://docs.python.org/.

[15] Xilinx, Inc. All Rights Reserved. *XUPV5-LX110T PCIe x1 Endpoint Plus Design Creation*, 2008. PDF file at http://www.xilinx.com/univ/xupv5-lx110t/design_files/PCIe/XUPV5-LX110T_PCIe_x1_Endpoint_Plus_Design_Creation.pdf.

[16] Gordon McMillan. *Socket Programming HOWTO*, 1998. http://www.amk.ca/python/howto/sockets/.

[17] WIKIBOOKS. *Python Programming/PyQt4*, 2007. http://en.wikibooks.org/wiki/Python_Programming/PyQt4.

[18] N. Matloff and F. Hsu. **Tutorial on threads programming with Python**. *PDF file at http://heather.cs.ucdavis.edu/~{}matloff/Python/PyThreads.pdf*, 2005.

[19] Digilent Inc, MicroBlaze uClinux, project. *Building the Hardware*, 2009. http://www.petalogix.com/resources/documentation/petalinux/userguide/Basics/BuildingHardware.

[20] Xilinx, Inc. All Rights Reserved. *Embedded System Tools Reference Guide*, 2009. PDF file at http://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf.

[21] openSUSE. *Expresscard hotplugging*, 2007. http://en.opensuse.org/Expresscard_hotplugging.

[22] fpgadeveloper.com, edk tutorials for xilinx fpgas. *Tri-mode Ethernet MAC*, 2008. http://www.fpgadeveloper.com/2008/10/tri-mode-ethernet-mac.html.

Appendix A

Χρήση Xilinx σε περιβάλλον Linux

Αρχικά πρέπει να γίνει εγκατάσταση των εργαλείων της Xilinx. Με την εγκατάσταση είναι δυνατή η εκτέλεση των διαφόρων εργασιών μόνο μέσα από το εργαλείο, δεν είναι δυνατή η εκτέλεση τους με χρήση εντολών μέσω UNIX κονσόλας. Για να είνα εφικτό πρέπει να γίνουν οι πιο κάτω ενέργειες:

#Fake libdb-4.1.so:
sudo apt-get install libdb4.X
sudo ln -s libdb-4.X.so /usr/lib/libdb-4.1.so
όπου "X" η τελευταία έκδοση της βιβλιοθήκης

#Fake gmake:
sudo ln -s make /usr/bin/gmake

#Fake microblaze libraries
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-addr2line /bin/mb-addr2line
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-ar /bin/mb-ar
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-as /bin/mb-as
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-c++ /bin/mb-c++
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-c++filt /bin/mb-c++filt
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-cpp /bin/mb-cpp
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-elf2flt /bin/mb-elf2flt
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-flthdr /bin/mb-flthdr
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-g++ /bin/mb-g++

```
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-gcov /bin/mb-gcov
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-gdb /bin/mb-gdb
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-gdbtui /bin/mb-gdbtui
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-gprof /bin/mb-gprof
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-insight /bin/mb-insight
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-ld /bin/mb-ld
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-ld.real /bin/mb-ld.real
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-nm /bin/mb-nm
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-objcopy /bin/mb-objcopy
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-objdump /bin/mb-objdump
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-ranlib /bin/mb-ranlib
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-readelf /bin/mb-readelf
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-size /bin/mb-size
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-strings /bin/mb-strings
sudo ln -s /opt/Xilinx/10.1/EDK/gnu/microblaze/lin/bin/mb-strip /bin/mb-strip
```

Οι εντολές που πρέπει να τρέξουν σε Linux κονσόλα για 32bits λειτουργικό είναι οι ακόλουθες:

$$\$source\ /opt/Xilinx/10.1/ISE/settings32.sh$$

$$\$source\ /opt/Xilinx/10.1/Chipscope/settings32.sh$$

$$\$source\ /opt/Xilinx/10.1/EDK/settings32.sh$$

$$\$export\ LD\_PRELOAD = /path/to/cable - driver/libusb - driver.so$$

Με την πρώτη εντολή είναι δυνατή η χρησιμοποίηση του εργαλείου Xilinx ISE Design Suite 10.1 και η οποία πρέπει να εκτελείται υποχρεωτικά για την χρήση των εργαλειών Xilinx EDK 10.1 και Xilinx Chipscope Pro 10.1. Η τελευταία εντολή χρησιμοποιείται για να γίνει προγραμματισμός του αναπτυξιακού με χρήση των Cable Driver.

Στην συνέχεια για λειτουργία του γραφικού των εργαλείων χρειάζεται να τρέξουμε τις εντολές:

$$\$ise$$

για λειτουργία του Xilinx ISE Design Suite 10.1,

$$\$xps$$

για χρήση του Xilinx EDK 10.1 και

$$\$analyzer.sh$$

για να χρησιμοποιηθεί το Xilinx Chipscope Pro 10.1.

Όλες οι πιο πάνω εντολές πρέπει να χρησιμοποιούνται στην ίδια Linux κονσόλα.

Appendix B

Εισαγωγή Αρχιτεχτονικής Χρήσης PCI-Express

Αρχικά πρέπει να τοποθετηθεί το αναπτυξιακό σύστημα στο PCI-Express 1x Lane του ηλεκτρονικού υπολογιστή (σχήμα B.1 [15]) και γίνονται σε αυτό οι ρυθμίσεις που φαίνονται στο σχήμα B.2 [15].

Figure B.1:

Αφού τοποθετηθεί η απαραίτητη άδεια χρήσης του endpoint_blk_plus_v1_9 στο μονοπάτι /home/username/.Xilinx/Coregen/CoreLicenses δημιουργείται project με χρήση του Xilinx EDK 10.1. Για την υλοποίηση χρησιμοποιείται η πλατφόρμα XUPV5 της οικογένειας ML505. Συγκεκριμένα γίνεται χρήση του πακέτου αναδιατασσόμενης λογικής

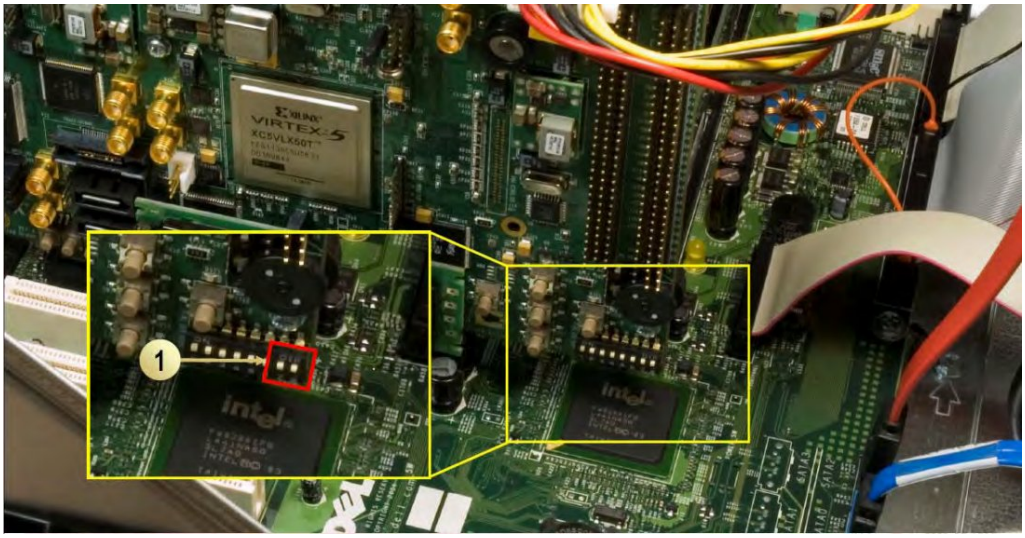
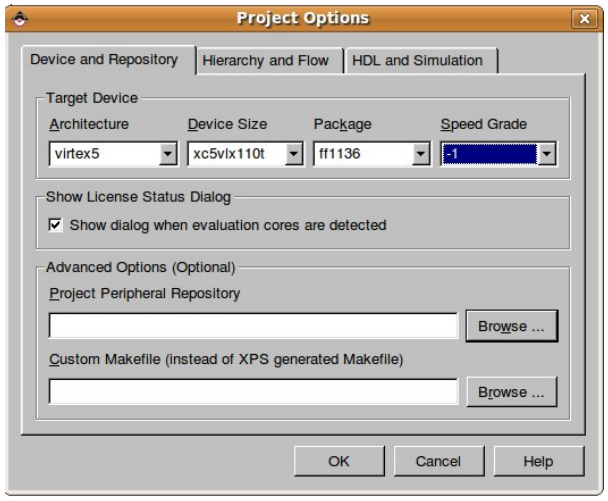


Figure B.2:

xc5vlx110t και Speed Grade -1. Κατά την διαδικασία δημιουργίας του project μπορούν να επιλεχθούν και να εισαχθούν τα περιφερειακά που θα χρησιμοποιούνται από την σχεδίαση που υλοποιείται πλην του PCI-Express.

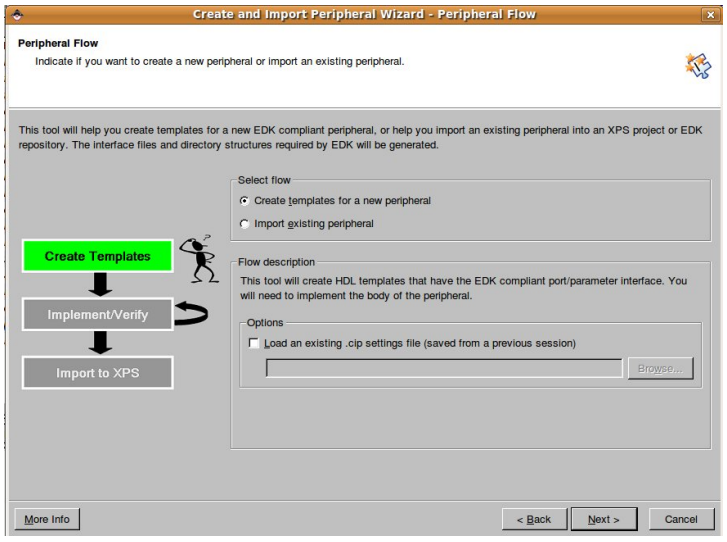


B.1 Δημιουργία Περιφερειακού custom_pcie

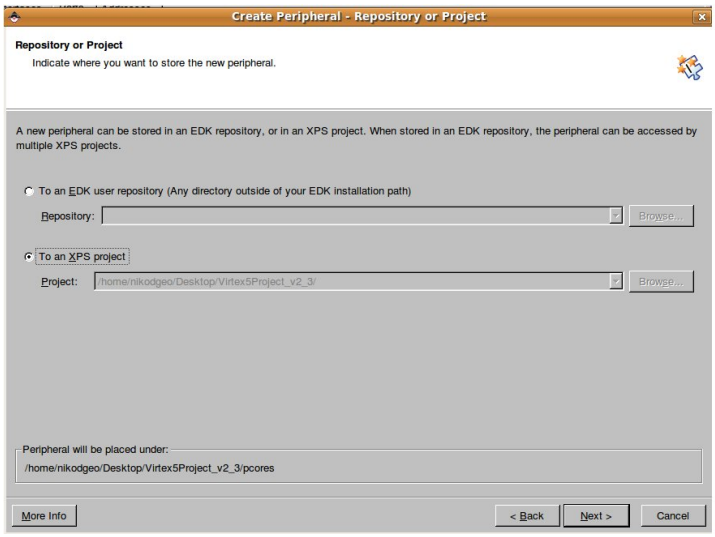
Αφού δημιουργηθεί το project πρέπει να δημιουργηθεί το περιφερειακό για χρήση του PCI-Express με την χρήση της διαδικασίας "Create or Import Peripheral" (Hardware ⇒ Create or Import Peripheral) που προσφέρει το εργαλείο EDK της Xilinx. Ποιο συγκεκριμένα κατά την διαδικασία αυτή πρέπει να επιλεχθεί η επιλογή "Create templates for a new peripheral". Επίσης χρησιμοποιείται το FSL bus για την επικοινωνία του Microblaze με το περιφερειακό που δημιουργείται. Ένα τυπικό όνομα που μπορεί να δοθεί είναι custom_pcie και το οποίο θα χρησιμοποιείται στην συνέχεια.

Τα βήματα που ακολουθούνται για την δημιουργία καινούργιου περιφερειακού είναι τα ακόλουθα:

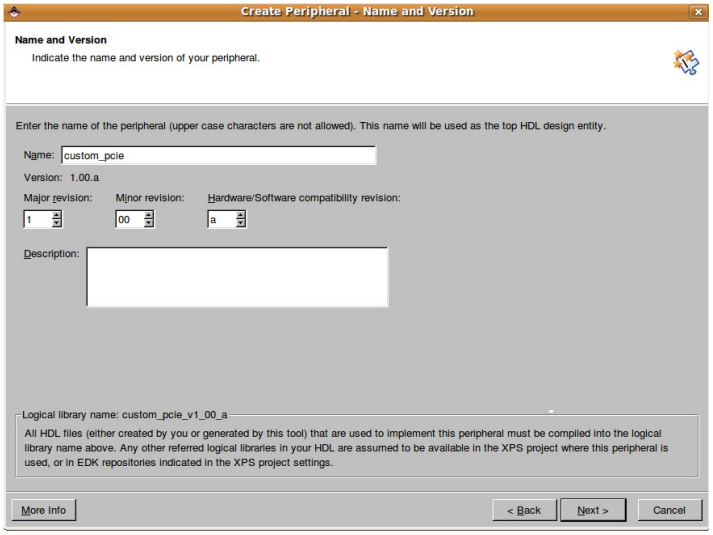
- Peripheral Flow:



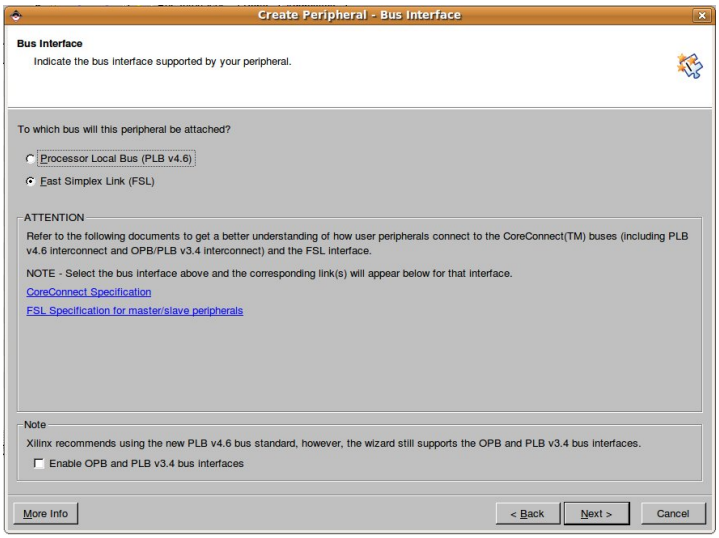
- Repository or Project:



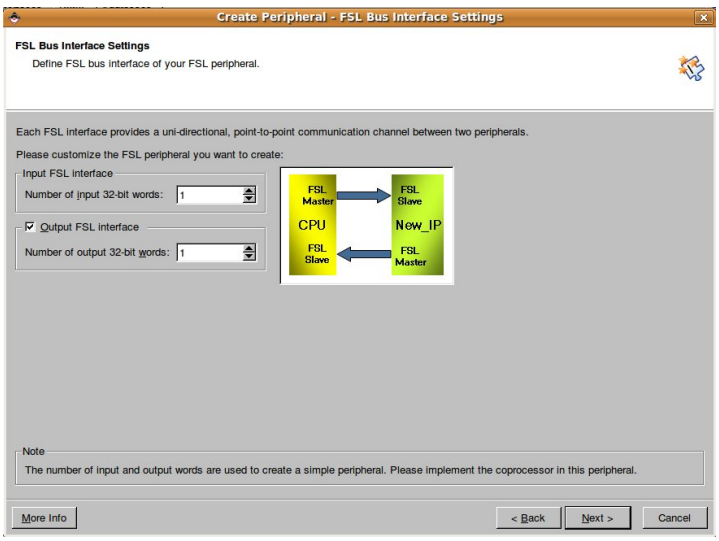
- Name and Version:



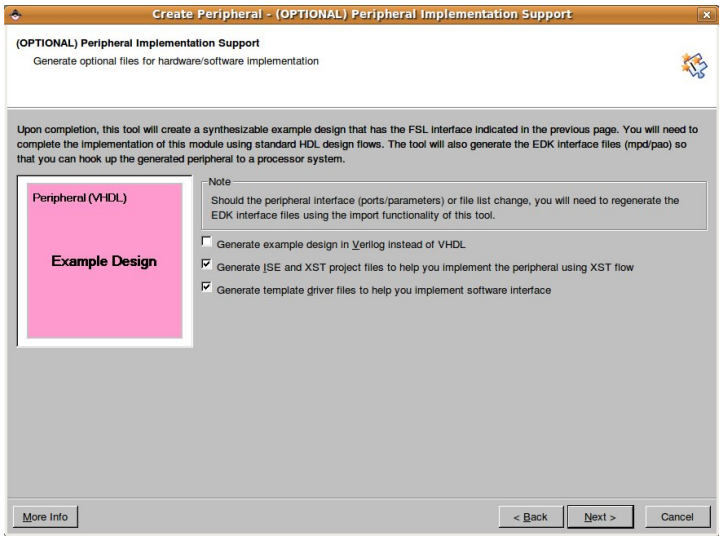
- Bus Interface:



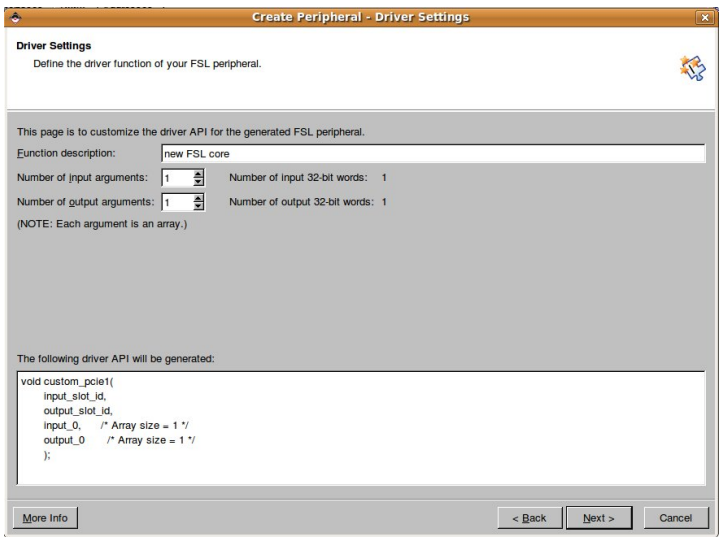
- FSL Bus Interface Settings:



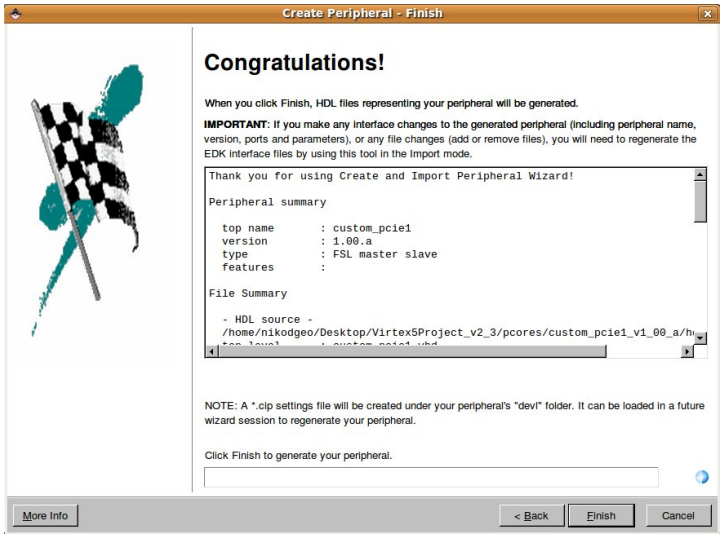
- (Optional) Peripheral Implementation Support:



- Driver Settings:



- Final Step:



Στην συνέχεια αντιγράφονται τα αρχεία με κατάληξη .vhd και τα οποία βρίσκονται στο φάκελο του ISE project στο μονοπάτι /project/path/pcores/custom_pcie_v1_00_a/hdl/vhdl. Για το λόγο ότι στο project για το PCI-Express στο Xilinx ISE Design Suite δημιουργήθηκε και αρχείο με κατάληξη .v, χρειάζεται να δημιουργηθεί ένας φάκελος με την ονομασία verilog και ο οποίος τοποθετείται στο μονοπάτι /project/path/pcores/custom_pcie_v1_00_a/hdl και μέσα στον οποίο θα τοποθετηθεί το αρχείο verilog.

Αφού ολοκληρωθούν τα πιο πάνω χρειάζεται να γίνουν αλλαγές στο top level της σχεδίασης, δηλαδή στο αρχείο custom_pcie.vhd και το οποίο βρίσκεται στο μονοπάτι στο οποίο τοποθετήθηκαν τα vhdl αρχεία κατά το προηγούμενο βήμα, /project/path/pcores/custom_pcie_v1_00_a/hdl. Οι αλλαγές που χρειάζεται να γίνουν είναι στο port map το οποίο θα πάρει την ακόλουθη μορφή:

```
entity custom_pcie is
port(
  pci_exp_rxn          :   in std_logic_vector((1 - 1) downto 0);
  pci_exp_rxp          :   in std_logic_vector((1 - 1) downto 0);
  pci_exp_txn          :   out std_logic_vector((1 - 1) downto 0);
  pci_exp_txp          :   out std_logic_vector((1 - 1) downto 0);
  sys_clk_c            :   in std_logic;
  sys_reset_n          :   in std_logic;

  FSL_Clk              :   in std_logic;
  .......
  FSL_M_Full           :   in std_logic
);
end custom_pcie;
```

Τα τέσσερα πρώτα σήματα είναι χρήσιμα για την είσοδο και έξοδο δεδομένων από το PCI-Express στην σχεδίαση. Το sys_clk_c είναι το ρολόι το οποίο χρησιμοποιείται από το PCI-Express και είναι συχνότητας 62.5MHz και το οποίο θα εξηγηθεί στην συνέχεια πως δημιουργείται. Τέλος το sys_reset_n είναι το reset που χρησιμοποιεί ο δίαυλος PCI-Express και δεν έχει σχέση με το reset που τυχόν χρησιμοποιεί ο χρήστης.

Για να ολοκληρωθούν οι αλλαγές στο top level προστίθεται το component FPGA_to_OS και το οποίο vhdl αρχείο είναι το top level component της σχεδίασης που υλοποιήθηκε στο ISE.

Τέλος, πρέπει να γίνουν οι κατάλληλες αλλαγές στο αρχείο custom_pcie_v2_1_0.pao το οποίο βρίσκεται στο μονοπάτι /project/path/pcores/custom_pcie_v1_00_a/data. Το αρχείο πρέπει να έχει την ακόλουθη μορφή:

lib custom_pcie_v1_00_a PIO_64_TX_ENGINE vhdl
lib custom_pcie_v1_00_a PIO_64_RX_ENGINE vhdl
lib custom_pcie_v1_00_a TX_ENGINE_ACCESS vhdl
lib custom_pcie_v1_00_a RX_ENGINE_ACCESS vhdl
lib custom_pcie_v1_00_a PIO_EP vhdl
lib custom_pcie_v1_00_a PIO_TO_CTRL vhdl
lib custom_pcie_v1_00_a PIO vhdl
lib custom_pcie_v1_00_a pci_exp_1_lane_64b_ep verilog
lib custom_pcie_v1_00_a pci_exp_64b_app vhdl
lib custom_pcie_v1_00_a my_black_box vhdl
lib custom_pcie_v1_00_a TX_engine_worldTopcie vhdl
lib custom_pcie_v1_00_a RX_engine_pcieToworld vhdl
lib custom_pcie_v1_00_a xilinx_pci_exp_ep vhdl
lib custom_pcie_v1_00_a FPGA_to_OS vhdl
lib custom_pcie_v1_00_a custom_pcie vhdl

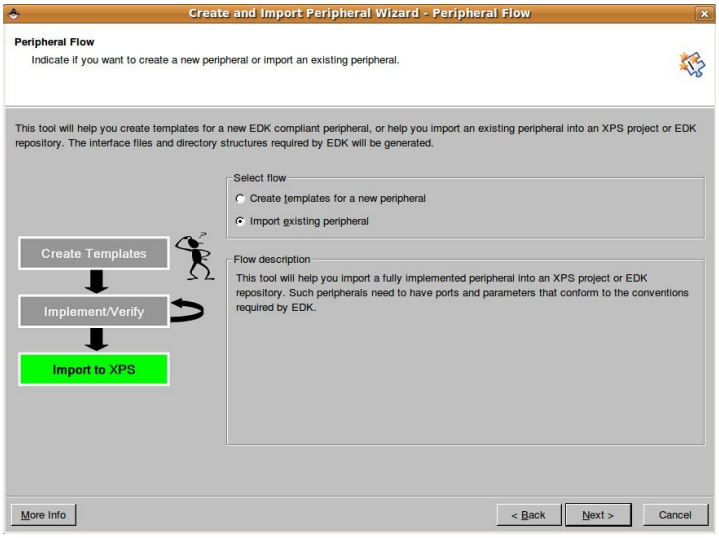
Αν επιθυμείτε τα δεδομένα τα οποία αποστέλλονται από το PCI-Express να χρησιμοποιούνται σε μια αρχιτεκτονική πρέπει τα αρχεία της αρχιτεκτονικής να τοποθετηθούν στο φάκελο που τοποθετήθηκαν τα υπόλοιπα .vhd αρχεία. Στην συνέχεια πρέπει να γίνουν αλλαγές στο αρχείο my_black_box.vhd, στο οποίο πρέπει να προστεθεί το component του top level της σχεδίασης. Τέλος πρέπει να γίνουν και αλλαγές στο αρχείο .pao. Πρέπει να προστεθούν τα αρχεία πάνω από την γραμμή lib custom_pcie_v1_00_a my_black_box vhdl κατά τον ίδιο τρόπο που προστέθηκαν τα υπόλοιπα αρχεία της σχεδίασης.

B.2 Εισαγωγή Περιφερειακού custom_pcie

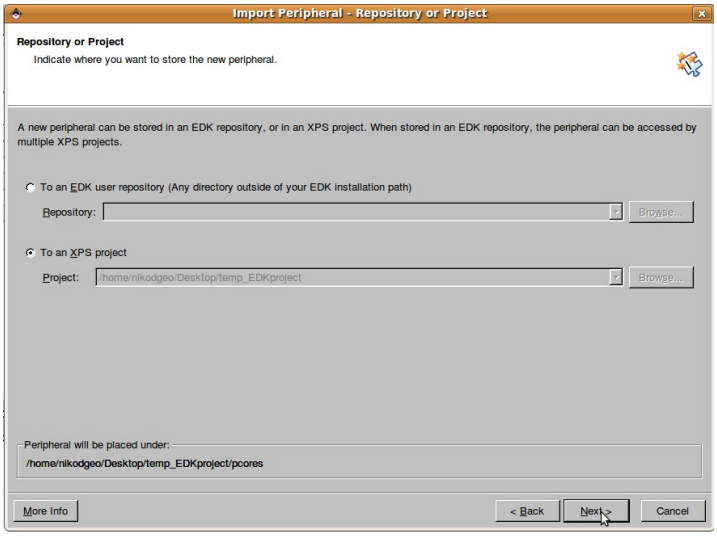
Για να ολοκληρωθεί η διαδικασία δημιουργίας του περιφερειακού πρέπει ξανά μέσω της διαδικασίας Create or Import Peripheral (Hardware ⇒ Create or Import Peripheral), αλλά με χρήση της επιλογής "Import existing peripheral" να εισαχθεί το περιφερειακό στην σχεδίαση.

Ακολουθούν αναλυτικά τα βήματα που πρέπει να ακολουθηθούν:

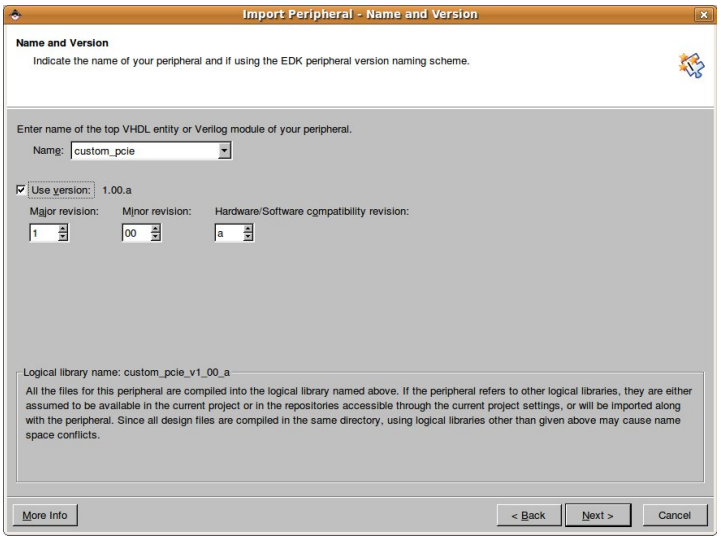
- Peripheral Flow:



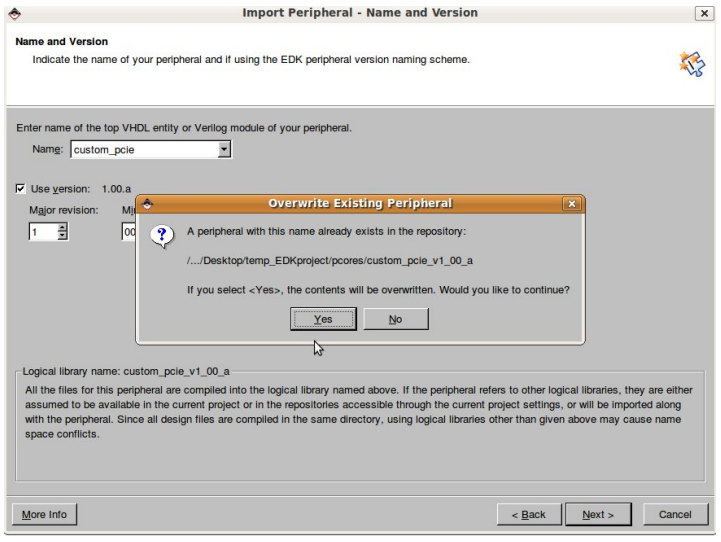
- Repository or Project:



- Name and Version (1):

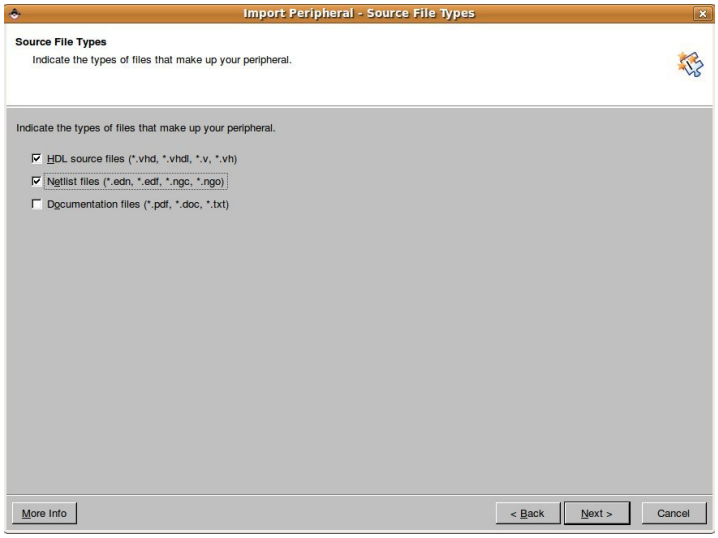


- Name and Version (2):

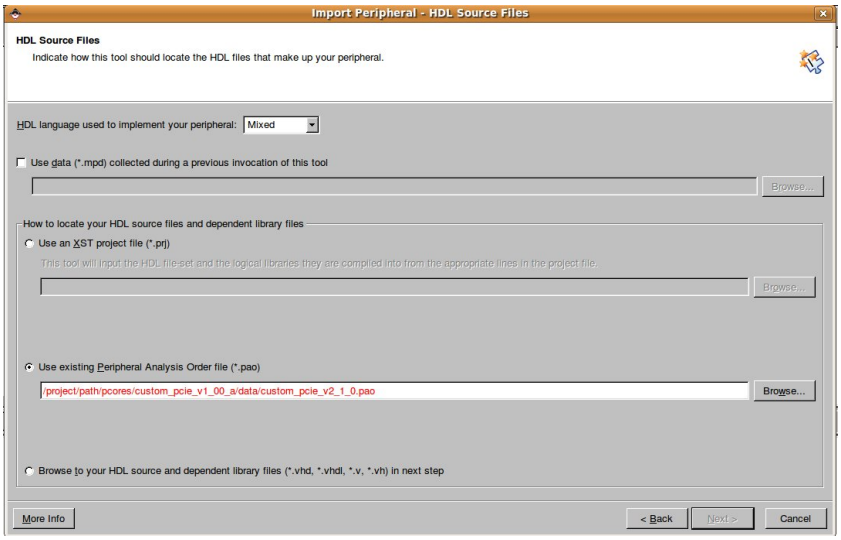


Με χρήση της επιλογής "Yes" δηλώνεται ότι γίνεται αναφορά στο περιφερειακό που δημιουργήθηκε κατά τα βήμα δημιουργίας περιφερειακού.

- Source File Types:

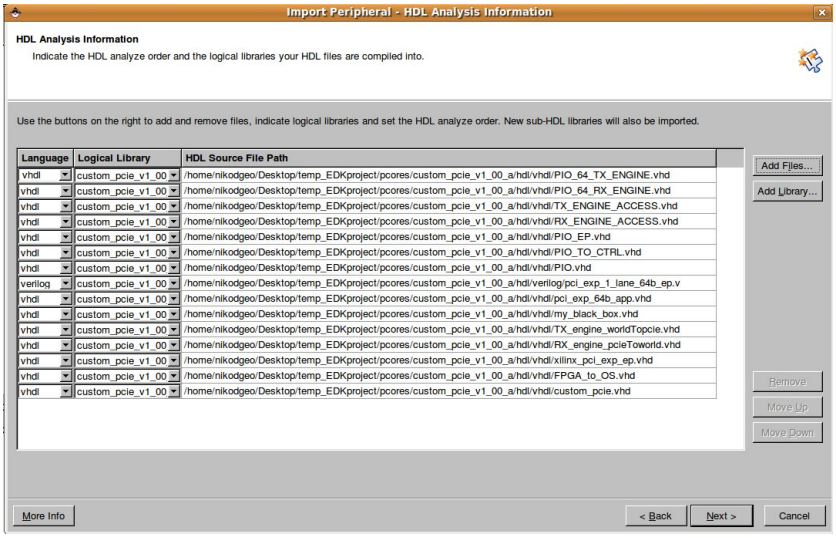


- HDL Source Files:

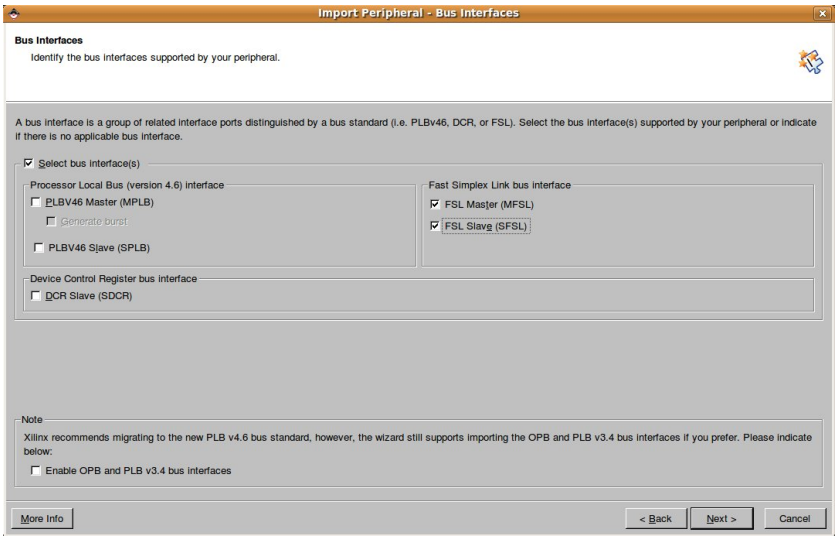


Στην επιλογή για "HDL language used to implement your peripheral" επιλέγεται η επιλογή "Mixed" μιας και χρησιμοποιούνται και vhdl και verilog αρχεία.

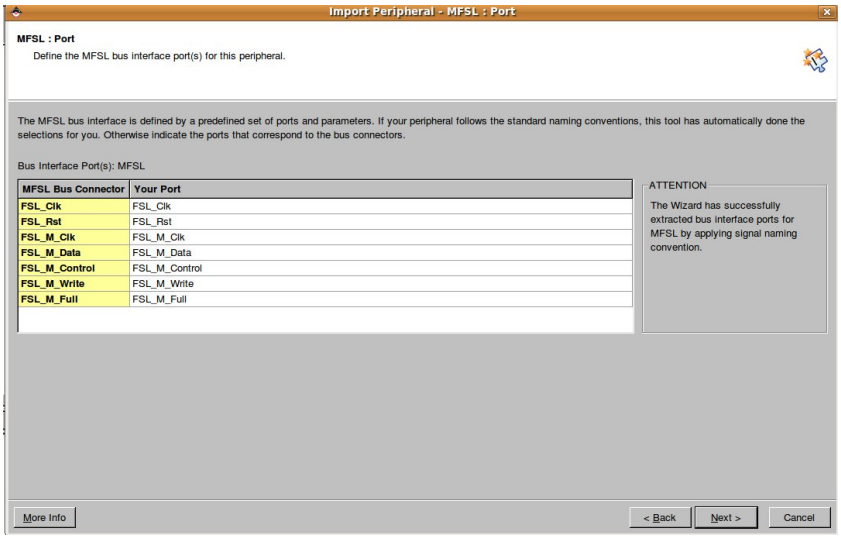
- HDL Analysis Information:



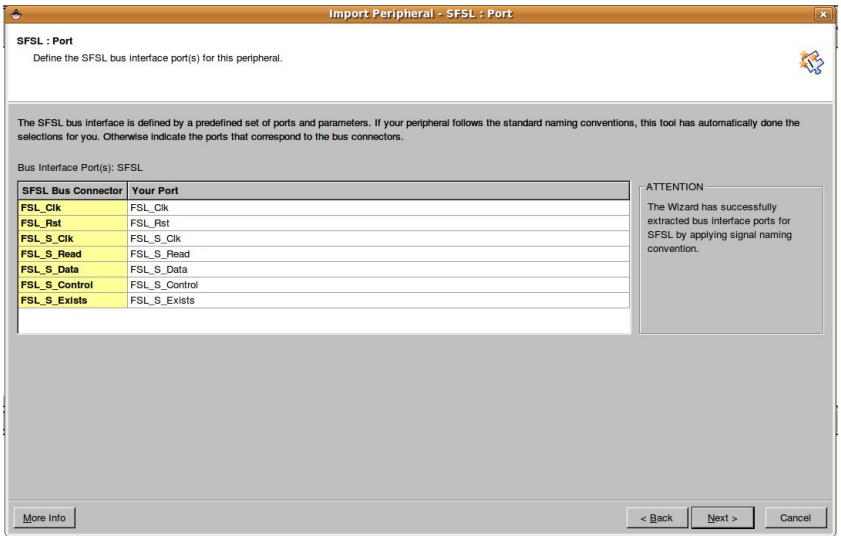
- Bus Interfaces:



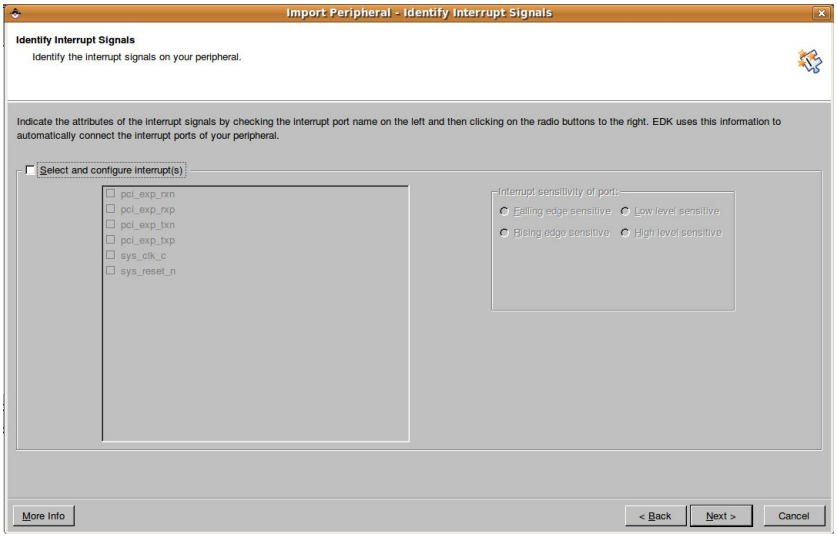
- MFSL : Port:



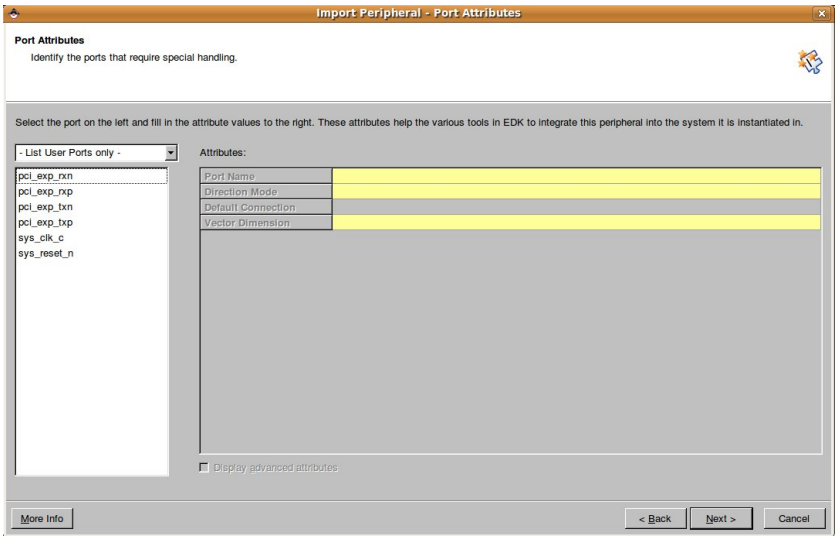
- SFSL : Port:



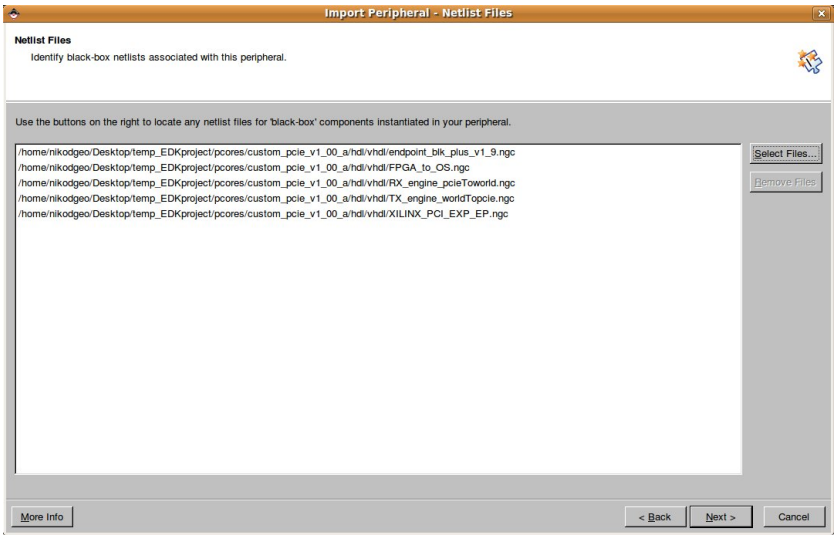
- Identify Interrupt Signals:



- Port Attributes:

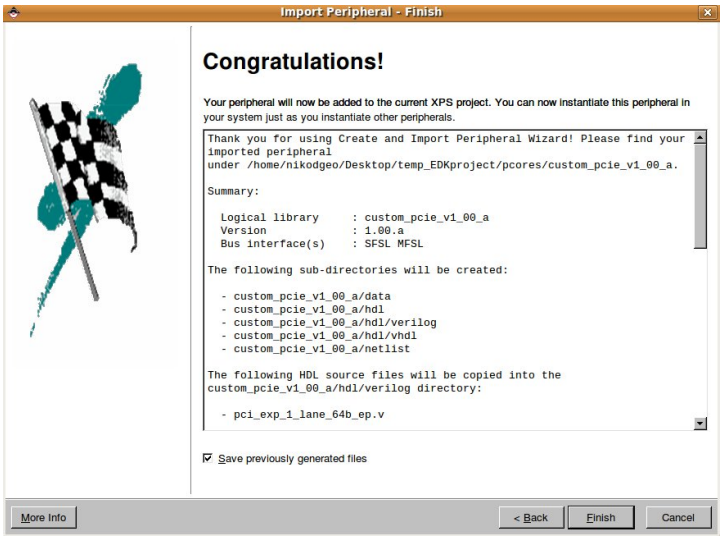


- Netlist Files:



Μέσω της επιλογής "Select Files" επιλέγονται τα αρχεία με κατάληξη .ngc τα οποία χρησιμοποιούνται από την σχεδίαση (τα αρχεία .ngc που χρησιμοποιούνται από το PCI-Express και από την αρχιτεκτονική που δημιουργείται).

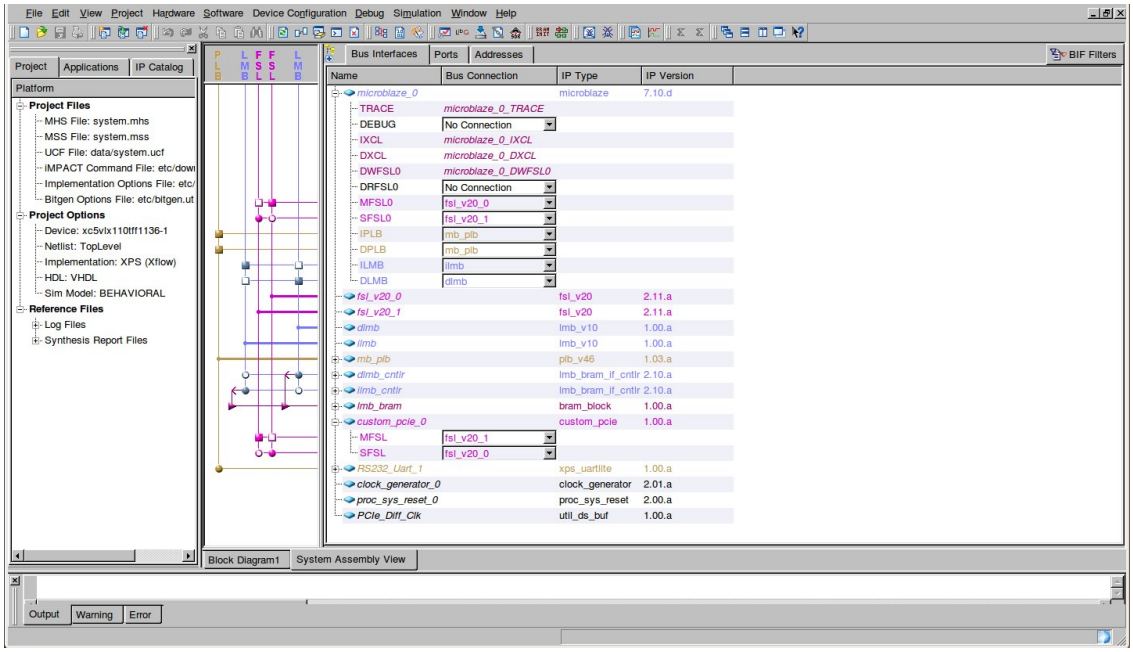
- Final Step:



Β.3 Συνδεσμολογίες και Εισαγωγή Περιφερειακών

Τέλος πρέπει να ενσωματωθεί το περιφερειακό που δημιουργήθηκε πιο πάνω με την υπόλοιπη σχεδίαση. Αυτό μπορεί να γίνει απλά με Drag and Drop ή με την εντολή "Add IP". Το περιφερειακό βρίσκεται στην καρτέλα "IP Catalog ⇒ Project Local pcores ⇒ USER".

Όπως αναφέρθηκε και προηγουμένως για την επικοινωνία μεταξύ του microblaze και του PCI-Express χρειάζονται δύο FSL Bus (Fast Simle Link). Αυτό γίνεται με τον ίδιο τρόπο όπως και κατά την εισαγωγή του περιφερειακού και βρίσκεται στην καρτέλα "IP Catalog ⇒ Bus and Bridge". Στην συνέχεια πρέπει να προστεθεί ένα FSL Link στον microblaze. Αυτό γίνεται με δεξί κλικ στον "microblaze_0", επιλέγεται η επιλογή "Configure IP" και πηγαίνοντας στην καρτέλα Buses επιλέγεται η χρήση ενός FSL Link. Τέλος γίνονται οι συνδέσεις των FSL όπως φαίνεται στην πάρακατω εικόνα.

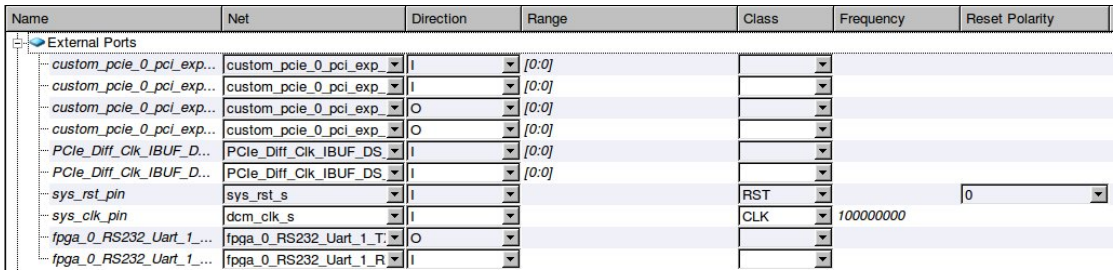


Όπως αναφέρθηκε πιο πάνω, στην περιγραφή δημιουργίας του top level, προστέθηκε μια είσοδος η οποία αφορά το ρολόι που χρησιμοποιεί το PCI-Express. Για να δημιουργηθεί το συγκεκριμένο ρολόι με συχνότητα 62.5 MHz χρειάζεται να χρησιμοποιηθεί από την σχεδίαση το έτοιμο περιφερειακό του EDK, "Differential Signaling IO Buffer" το οποίο μετονομάστηκε σε "PCIe_Diff_Clk". Το περιφερειακό βρίσκεται στον κατάλογο "IP Catalog ⇒ Utility ⇒ Differential Signaling IO Buffer".

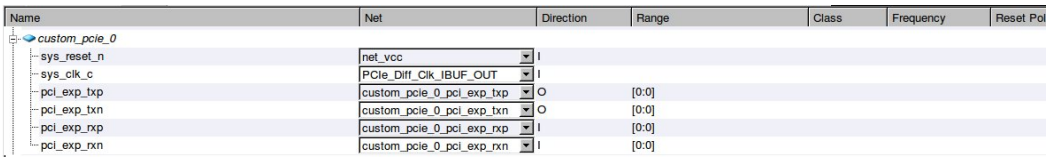
Αφού ολοκληρωθούν τα πιο πάνω, πρέπει να γίνουν οι συνδέσεις των εισόδων και

εξόδων των δύο περιφερειακών που προστέθηκαν, "PCIe_Diff_Clk" και "custom_pcie". Αρχικά χρειάζεται να ενωθούν οι εισόδοι IBUF_DS_N και IBUF_DS_P με χρήση της επιλογής Make External. Με αυτό τον τρόπο γίνεται συνδεσμολογία των δύο εισόδων με τον έξω κόσμο. Πλέον πηγαίνοντας στην καρτέλα External Ports υπάρχουν τα pins PCIe_Diff_Clk_IBUF_DS_P_pin και PCIe_Diff_Clk_IBUF_DS_N_pin και τα οποία θα περιγραφεί στην συνέχεια πως χρησιμοποιούνται. Ως έξοδο χρησιμοποιείται το IBUF_Out και στο οποίο επιλέγεται νέα σύνδεση, New Connection στις επιλογές που δίνονται.

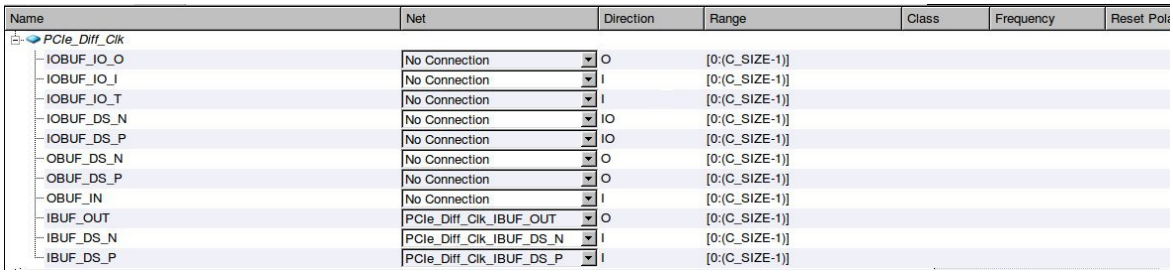
Στην συνέχεια, στο περιφερειακό που δημιουργήθηκε χρειάζεται εκτός απο το reset του χρήστη και ένα επιπλέον reset το οποίο είναι για αποκλειστική χρήση από το PCI-Express. Το ίδιο ισχύει και για το ρολόι, μιας και το PCI-Express δουλεύει στα 62.5MHz και όχι στα 120MHz που δουλεύει η υπόλοιπη σχεδίαση. Το reset (sys_reset_n) συνδέεται το σήμα "net_vcc" και το ρολόι (sys_clk_c) με την έξοδο του περιφερειακού PCIe_Diff_Clk (PCIe_Diff_Clk_IBUF_OUT). Για τις υπόλοιπες εισόδους και εξόδους χρησιμοποιείται η επιλογή Make External μιας και είναι είσοδοι και έξοδοι αποκλειστικά για το PCI-Express. Οι συνδέσεις που περιγράφηκαν πιο πάνω φαίνονται και στα παρακάτω σχήματα:



Name	Net	Direction	Range	Class	Frequency	Reset Polarity
External Ports						
custom_pcie_0_pci_exp...	custom_pcie_0_pci_exp_	I	[0:0]			
custom_pcie_0_pci_exp...	custom_pcie_0_pci_exp_	I	[0:0]			
custom_pcie_0_pci_exp...	custom_pcie_0_pci_exp_	O	[0:0]			
custom_pcie_0_pci_exp...	custom_pcie_0_pci_exp_	O	[0:0]			
PCIe_Diff_Clk_IBUF_D...	PCIe_Diff_Clk_IBUF_DS_	I	[0:0]			
PCIe_Diff_Clk_IBUF_D...	PCIe_Diff_Clk_IBUF_DS_	I	[0:0]			
sys_rst_pin	sys_rst_s	I		RST		0
sys_clk_pin	dcm_clk_s	I		CLK	100000000	
fpga_0_RS232_Uart_1_...	fpga_0_RS232_Uart_1_T.	O				
fpga_0_RS232_Uart_1_...	fpga_0_RS232_Uart_1_R	I				



Name	Net	Direction	Range	Class	Frequency	Reset Pol
custom_pcie_0						
sys_reset_n	net_vcc	I				
sys_clk_c	PCIe_Diff_Clk_IBUF_OUT	I				
pci_exp_txp	custom_pcie_0_pci_exp_txp	O	[0:0]			
pci_exp_txn	custom_pcie_0_pci_exp_txn	O	[0:0]			
pci_exp_rxp	custom_pcie_0_pci_exp_rxp	I	[0:0]			
pci_exp_rxn	custom_pcie_0_pci_exp_rxn	I	[0:0]			



Name	Net	Direction	Range	Class	Frequency	Reset Pola
PCIe_Diff_Clk						
IOBUF_IO_O	No Connection	O	[0:(C_SIZE-1)]			
IOBUF_IO_I	No Connection	I	[0:(C_SIZE-1)]			
IOBUF_IO_T	No Connection	I	[0:(C_SIZE-1)]			
IOBUF_DS_N	No Connection	IO	[0:(C_SIZE-1)]			
IOBUF_DS_P	No Connection	IO	[0:(C_SIZE-1)]			
OBUF_DS_N	No Connection	O	[0:(C_SIZE-1)]			
OBUF_DS_P	No Connection	O	[0:(C_SIZE-1)]			
OBUF_IN	No Connection	I	[0:(C_SIZE-1)]			
IBUF_OUT	PCIe_Diff_Clk_IBUF_OUT	O	[0:(C_SIZE-1)]			
IBUF_DS_N	PCIe_Diff_Clk_IBUF_DS_N	I	[0:(C_SIZE-1)]			
IBUF_DS_P	PCIe_Diff_Clk_IBUF_DS_P	I	[0:(C_SIZE-1)]			

Τέλος για να ολοκληρώσουμε την διαδικασία της συνδεσμολογίας πρέπει να τροποποιήσουμε το UCF αρχείο. Το τελικό UCF πρέπει να έχει την ακόλουθη μορφή:

```
##############################################################################
## This system.ucf file is generated by Base System Builder based on
## the settings in the selected Xilinx Board Definition file. Please
## add other user constraints to this file based on customer design
## specifications.
##############################################################################
Net sys_clk_pin TNM_NET = sys_clk_pin;
Net sys_clk_pin LOC = AH15;
Net sys_clk_pin IOSTANDARD=LVCMOS33;
## System level constraints
```

```
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 10000 ps;;

## IO Devices constraints

#### Module RS232_Uart_1 constraints

Net fpga_0_RS232_Uart_1_RX_pin LOC = AG15;
Net fpga_0_RS232_Uart_1_RX_pin IOSTANDARD=LVCMOS33;
Net fpga_0_RS232_Uart_1_TX_pin LOC = AG20;
Net fpga_0_RS232_Uart_1_TX_pin IOSTANDARD=LVCMOS33;

#### Module Push_Buttons_5Bit constraints

Net sys_rst_pin LOC = AJ6;
Net sys_rst_pin IOSTANDARD=LVCMOS33;
Net sys_rst_pin PULLUP;
Net sys_rst_pin SLEW=SLOW;
Net sys_rst_pin DRIVE=2;

#### Module PCIe_Diff_Clk constraints

Net PCIe_Diff_Clk_IBUF_DS_P_pin<0> LOC=AF4;
Net PCIe_Diff_Clk_IBUF_DS_P_pin<0> IOSTANDARD = LVDS_25;
Net PCIe_Diff_Clk_IBUF_DS_N_pin<0> LOC=AF3;
Net PCIe_Diff_Clk_IBUF_DS_N_pin<0> IOSTANDARD = LVDS_25;

#### Module PCIe_Bridge constraints

Net custom_pcie_0_pci_exp_rxp_pin<0> LOC=AE1;
Net custom_pcie_0_pci_exp_rxp_pin<0> IOSTANDARD = LVDS_25;
Net custom_pcie_0_pci_exp_rxn_pin<0> LOC=AF1;
Net custom_pcie_0_pci_exp_rxn_pin<0> IOSTANDARD = LVDS_25;
Net custom_pcie_0_pci_exp_txp_pin<0> LOC=AD2;
Net custom_pcie_0_pci_exp_txp_pin<0> IOSTANDARD = LVDS_25;
Net custom_pcie_0_pci_exp_txn_pin<0> LOC=AE2;
```

```
Net custom_pcie_0_pci_exp_txn_pin<0> IOSTANDARD = LVDS_25;
```

B.4 Κώδικας χρήσης PCI-Express

Ο πιο κάτω κώδικας υλοποιήθηκε στα πλαίσια της διπλωματικής εργασίας του Ιωάννη Καρτσωνάκη και ο οποίος τροποποιήθηκε για τις ανάγκες της παρούσας διπλωματικής εργασίας. Στον κώδικα αρχικά γίνονται αρχικοποιήσεις για να είναι δυνατή η χρησιμοποίηση του οδηγού PCI-Express. Στην συνέχεια γίνεται εγγραφή τιμών στην διεύθυνση εγγραφής του οδηγού PCI-Express (στέλλονται δεδομένα) και τέλος διαβάζονται τιμές από την διεύθυνση ανάγνωσης του οδηγού PCI-Express (επιστρεφόμενες τιμές).

```
//////////////////////////////////////////////////////////////////
// Beta Memory Map for IO via xupV5 - Linux OS
//
// author: Kartsonakis Ioannis
// email: <kartsonis@gmail.com>
//
//////////////////////////////////////////////////////////////////

#include <sys/mman.h>
#include <sys/types.hh>
#include <errno.h>
#include <fcntl.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define XC_MMAP_LENGTH 4096
#define XC_DEVNAME "/dev/Xilinx_pcix"

/*
* This sample program is supposed to work with Xilinx_pcix driver.
* It maps the memory which the driver can loan to the user space,
* and tries to analyze state of interesting portions of it – these,
* to which physical devices are connected.
*/
```

```
int
main(int argc, char **argv)
{
  FILE * pFile;
  pFile = fopen ("temp.txt","w");

  int wr_addr;
  int valid_addr;
  int rd_addr;
  int size_tran,size_recei;
  int flag_d;
  int fd;
  int e,i;
  unsigned long int count=0;

  int b;
  unsigned long int d;
  unsigned int *p;
  char buf[4096];
  unsigned char *l;

  /*
  * Check -d existance as a first parameter.
  */
  flag_d = e = fd = 0;
  if (argc == 2)
  {
          e = strcmp(argv[1], "-d");
          if (e == 0)
               flag_d = 1;
  }

  fd = open(XC_DEVNAME, O_RDONLY);
  assert(fd != -1 && "open() failed");
```

```
if (flag_d)
        printf("# opened with file descriptor %d", fd);

p = (unsigned int *)mmap(NULL, XC_MMAP_LENGTH, PROT_READ—PROT_WRITE,
MAP_PRIVATE, fd, 0);
if (p == MAP_FAILED)
{
        printf("Error: %s (errno=%d)", strerror(errno), errno);
        exit(EXIT_FAILURE);
}

e = 0;
b=p;

wr_addr=36;
valid_addr=28;
rd_addr=84;

// 268435456 :1 Giga Trasmit/Receive
// 262144 :1 Mega Trasmit/Receive
// 2621440 :10 Mega Trasmit/Receive
// 67108864 :256 Mega Trasmit/Receive

size_tran=67108864;
size_recei=2621440;

// Transmit Data
for (d=0;d<size_tran;d=d+1)
{
        *(p+wr_addr)=count;
        count++;
}

// Receive Data
for (d=0;d<size_recei;d=d+1)
{
```

```
            if ( *(p+valid_addr) == 0 )
                fprintf (pFile,"#%d Read Data %x",d,*(p+rd_addr));
  };

  fclose (pFile);

  /*
  * Deinitialization.
  */
  e = munmap(p, XC_MMAP_LENGTH);
  assert(e != -1 && "munmap() failed");
  e = close(fd);
  assert(e != -1 && "close() failed");
  exit(EXIT_SUCCESS);
}
```

Appendix C

Εισαγωγή Αρχιτεκτονικής MTP

Αρχικα, όπως και κατά την χρησιμοποίηση του PCI-Express πρέπει να γίνουν οι απαραίτητες ρυμίσεις. Για την απενεργοποίηση του πρωτοκόλλου ipv6, ανάλογα με την έκδοση Linux που χρησιμοποιείται, χρησιμοποιούνται οι πιο κάτω εντολές:

- Στο αρχείο "*/etc/modprobe.d/aliases*" γίνεται αλλαγή της γραμμής

  ```
  alias net-pf-10 ipv6
  ```

  σε

  ```
  alias net-pf-10 off
  ```

- Στο αρχείο "*/etc/modprobe.d/blacklist.local*"προστίθεται στο τέλος η γραμμή

  ```
  blacklist ipv6
  ```

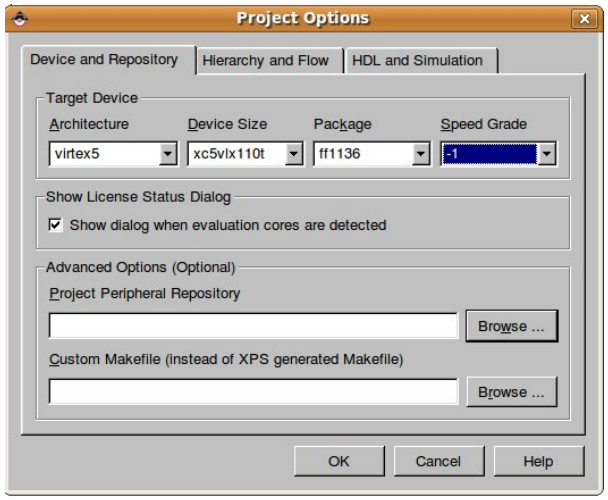
Τέλος πρέπει να ενεργοποιηθεί η λειτουργία του ipv4. Αυτό γίνεται με την χρήση της εντολής

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Οι ρυθμίσεις που χρειάζονται στο αναπτυξιακό σύστημα φαίνονται στο σχήμα C.1 [22].

Figure C.1:

Όπως και κατά την δημιουργία αρχιτεκτονικής η οποία χρησιμοποιεί το PCI-Express, δημιουργείται project με χρήση του Xilinx EDK 10.1. Για την υλοποίηση χρησιμοποιείται η πλατφόρμα XUPV5 της οικογένειας ML505. Συγκεκριμένα γίνεται χρήση του πακέτου αναδιατασσόμενης λογικής xc5vlx110t και Speed Grade -1. Κατά την διαδικασία δημιουργίας του project μπορούν να επιλεχθούν και να εισαχθούν τα περιφερειακά που θα χρησιμοποιούνται από την σχεδίαση που υλοποιείται πλην του PCI-Express.



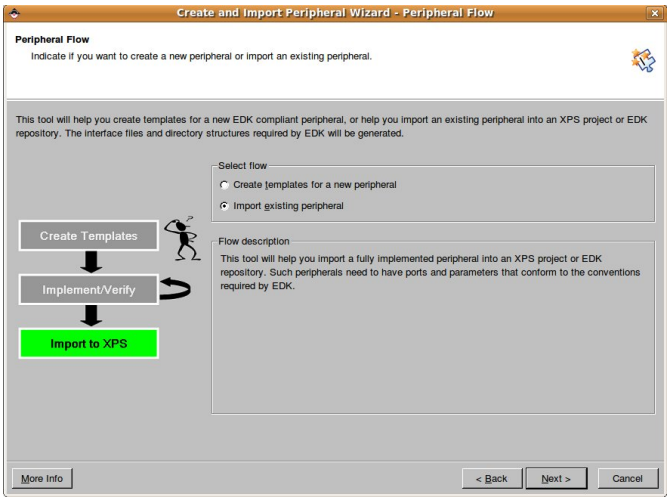
C.1 Δημιουργία Περιφερειακού eth_mac

Το μόνο που χρειάζεται σε αυτό το βήμα είναι να αντιγράψουμε τον φάκελο eth_mac_v1_00_a που δίνεται στο μονοπάτι /project/path/pcores/ .

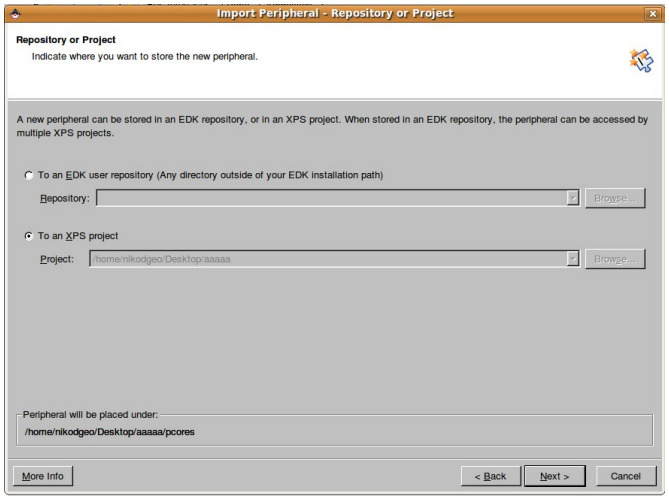
C.2 Εισαγωγή Περιφερειακού eth_mac

Με χρήση της διαδικασίας "Create or Import Peripheral" (Hardware ⇒ Create or Import Peripheral) γίνεται εισαγωγή του περιφερειακού eth_mac στην σχεδίαση. Τα βήματα που ακολουθούνται είναι τα ακόλουθα:

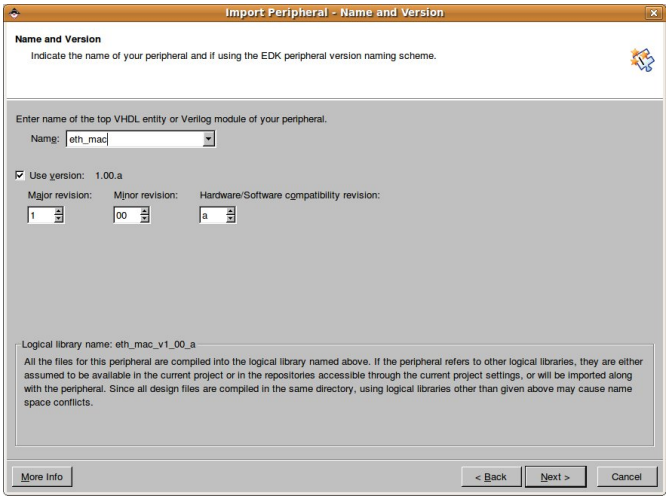
- Peripheral Flow:



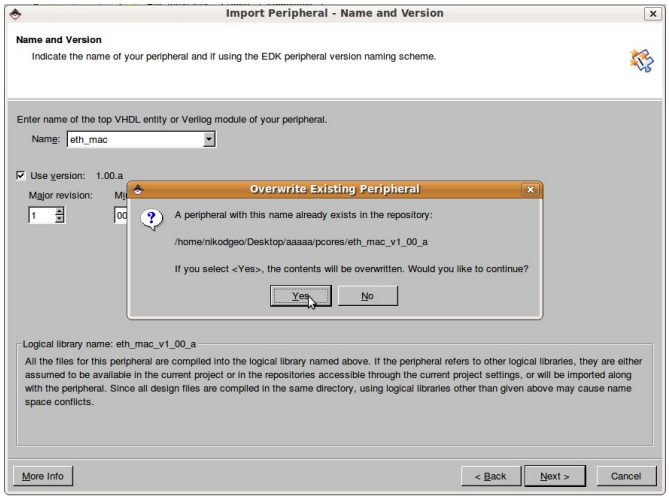
- Repository or Project:



- Name and Version (1):

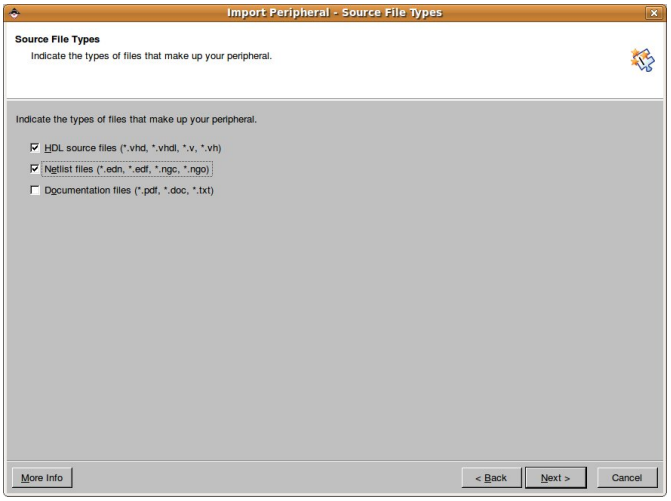


- Name and Version (2): Όπου και πατάμε την επιλογή "Yes" για να δηλώσουμε

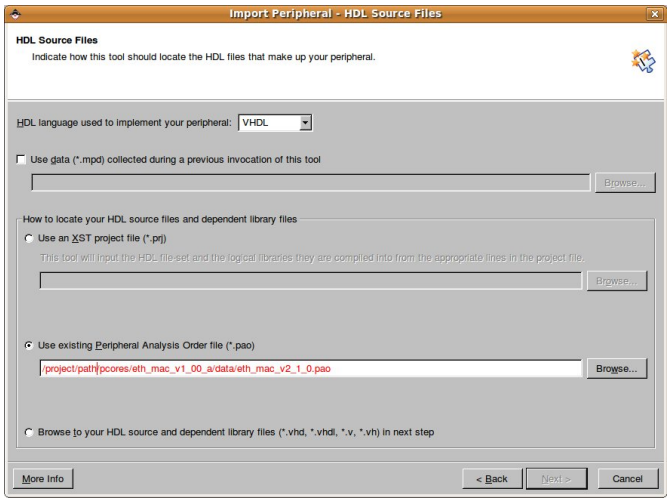


ότι αναφερόμαστε στο περιφερειακό που δημιουργήσαμε κατά τα βήμα δημιουργίας περιφερειακού.

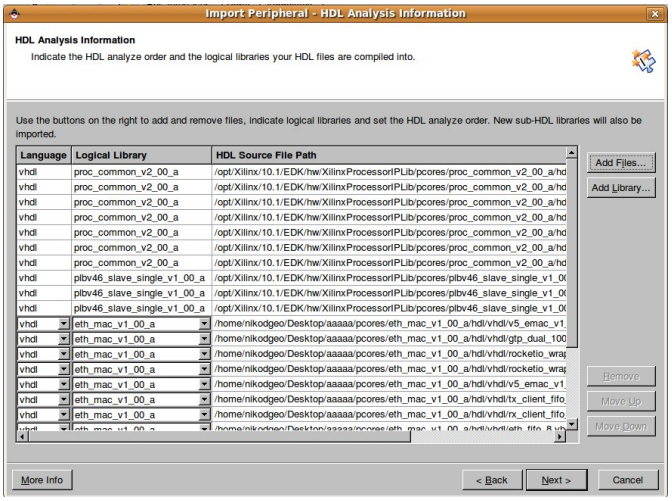
- Source File Types:



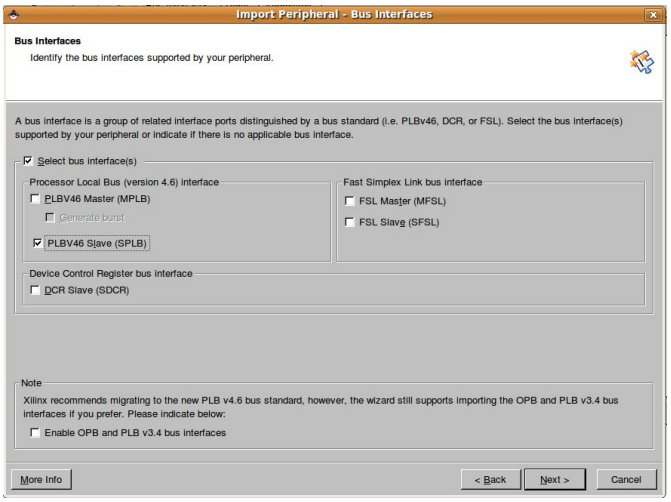
- HDL Source Files:



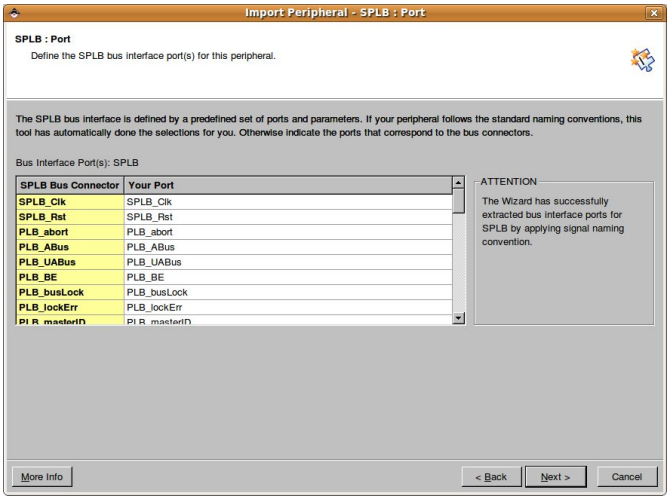
- HDL Analysis Information:



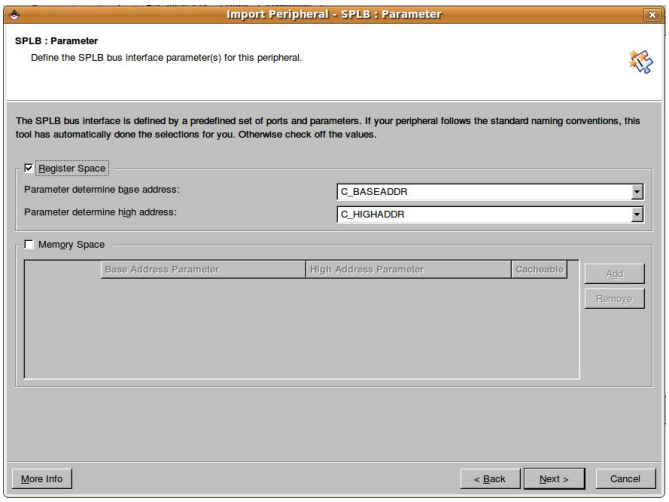
- Bus Interfaces:



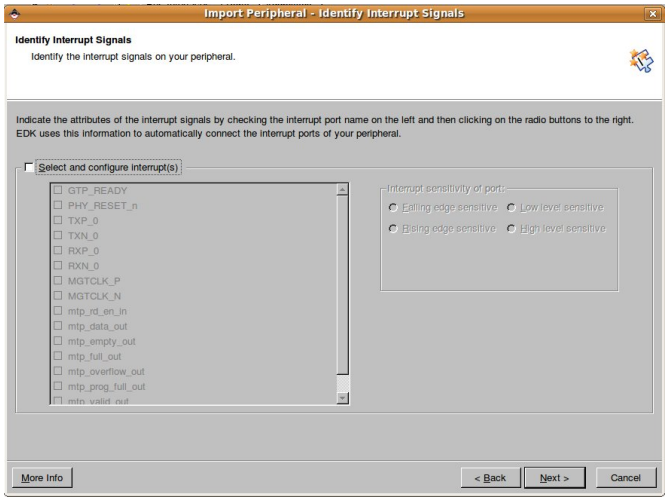
- SPLB : Port:



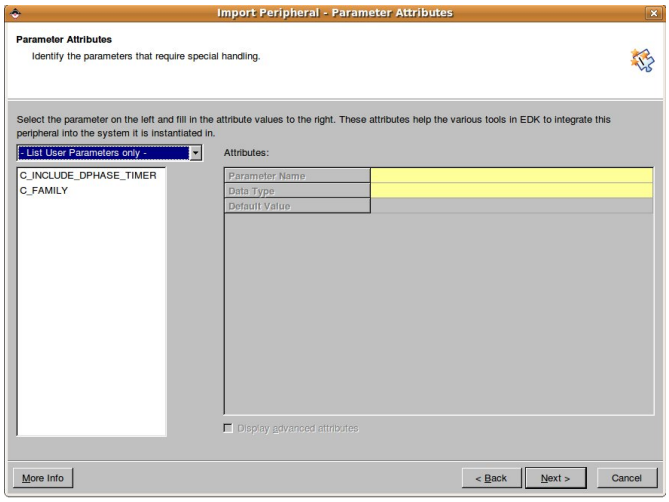
- SPLB Parameter:



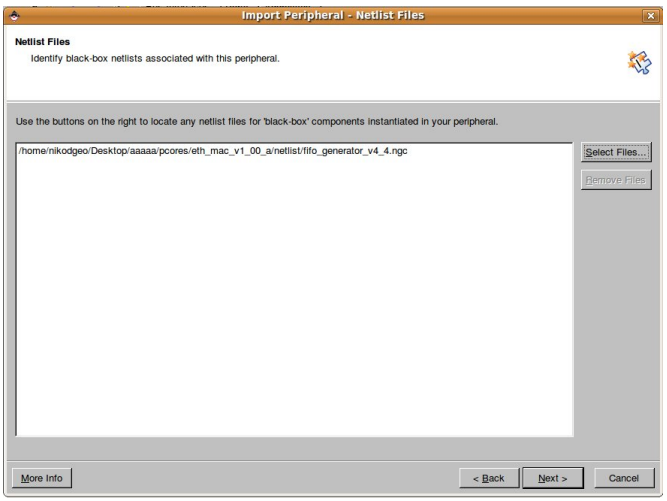
- Identify Interrupt Signals:



- Parameter Attributes:

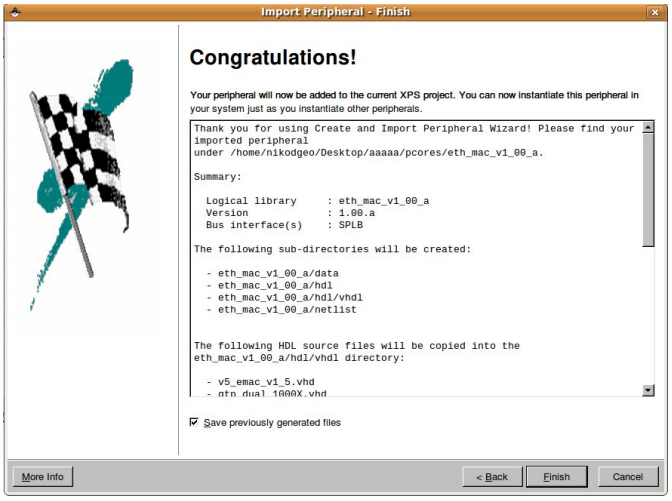


- Netlist Files:



Μέσω της επιλογής "Select Files" επιλέγονται τα αρχεία με κατάληξη .ngc τα οποία χρησιμοποιούνται από την σχεδίαση και βρίσκονται ήδη στον φάκελο netlist.

- Final Step:



C.3 Δημιουργία Περιφερειακού custom_eth_mac για χρήση του περιφερειακού eth_mac

Στην συνέχεια πάλι με χρήση της διαδιακασίας "Create or Import Peripheral" (Hardware ⇒ Create or Import Peripheral) θα δημιουργήσουμε το περιφερειακό μέσο του οποίου θα χρησιμοποιείται το Gigabit Ethernet. Αυτή την φορά για επικοινωνία της αρχιτεκτονικής με τον Microblaze γίνεται χρήση του FSL Bus.

Τα βήματα που ακολουθούνται είναι τα ίδια με αυτά που περιγράφηκαν στο παράρτημα B στην ενότητα "Δημιουργία Περιφερειακού custom_pcie".

Στο αρχείο custom_eth_mac.vhd και το οποίο βρίσκεται στο μονοπάτι /project/path/pcores/custom_eth_mac_v1_00_a/hdl/vhdl πρέπει να γίνουν οι ακόλουθες αλλαγές στο port map:

```
entity custom_eth_mac is
port(
  reset_in              :   in std_logic;
  mtp_rd_en_in          :   out std_logic;
  mtp_data_out          :   in std_logic_vector(7 downto 0);
  mtp_empty_out         :   in std_logic;
  mtp_full_out          :   in std_logic;
  mtp_overflow_out      :   in std_logic;
  mtp_prog_full_out     :   in std_logic;
  mtp_valid_out         :   in std_logic;
  mtp_underflow_out     :   in std_logic;

  FSL_Clk               :   in std_logic;
  .......
  FSL_M_Full            :   in std_logic
);
end custom_eth_mac;
```

Τα vhdl αρχεία της αρχιτεκτονικής τοποθετούνται στο μονοπάτι /project/path/pcores/custom_eth_mac_v1_00_a/hdl/vhdl και τα Verilog στο μονοπάτι /project/path/pcores/custom_eth_mac_v1_00_a/hdl/verilog. Τέλος πρέπει να γίνουν οι κατάλληλες αλλαγές στο αρχείο custom_eth_mac_v2_1_0.pao το οποίο βρίσκεται στο μονοπάτι /project/path/pcores/custom_eth_mac_v1_00_a/data.

C.4 Εισαγωγή Περιφερειακού custom_eth_mac

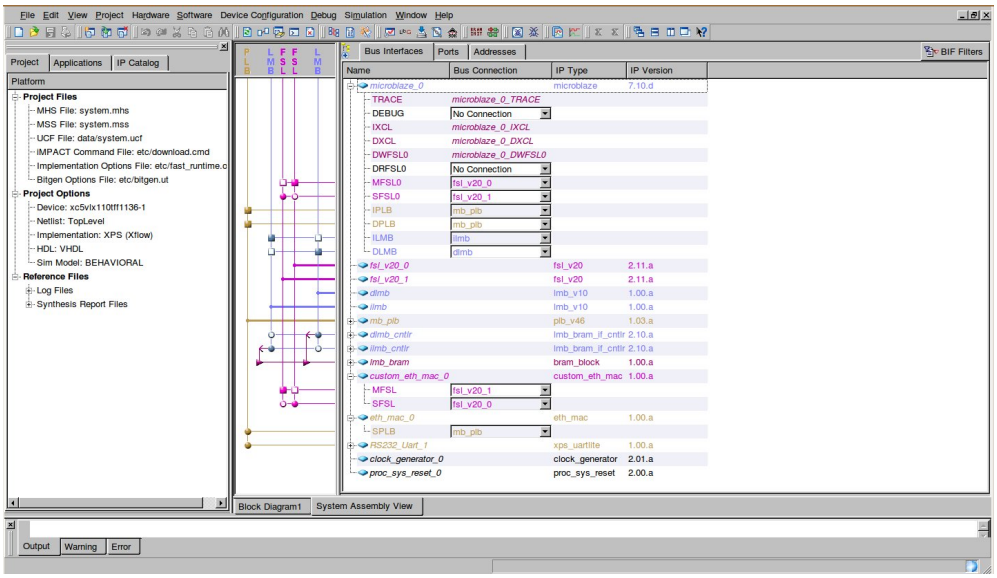
Για την εισαγωγή του περιφερειακού custom_eth_mac ακολουθούνται τα ίδια βήματα με αυτά που περιγράφονται στο παράρτημα Β στην ενότητα "Εισαγωγή Περιφερειακού custom_pcie". Η μόνη διαφορά είναι στο βήμα Source File Types στην οποία επιλογή δεν επιλέγεται η χρησιμοποίηση Netlist Files αν δεν χρησιμοποιούνται από την αρχιτεκτονική.

C.5 Συνδεσμολογίες και Εισαγωγή Περιφερειακών

Τέλος πρέπει να ενσωματωθούν τα δύο περιφερειακά που δημιουργήθηκαν πιο πάνω με την υπόλοιπη σχεδίαση. Αυτό μπορεί να γίνει απλά με Drag and Drop ή με την εντολή "Add IP". Το περιφερειακά βρίσκονται στην καρτέλα "IP Catalog ⇒ Project Local pcores ⇒

USER".

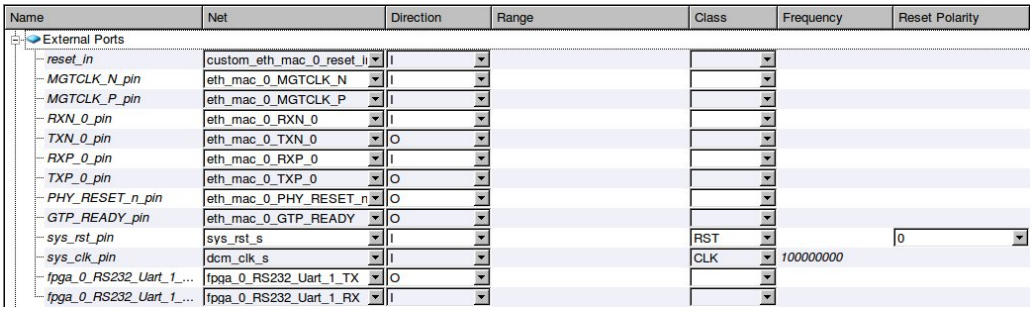
Το περιφερειακό eth_mac συνδέεται σε PLB Bus ενώ το περιφερειακό custom_eth_mac σε FSL Bus. Οι συνδέσεις των FSL και OPB Bus γίνονται όπως φαίνεται στην πάρακατω εικόνα.

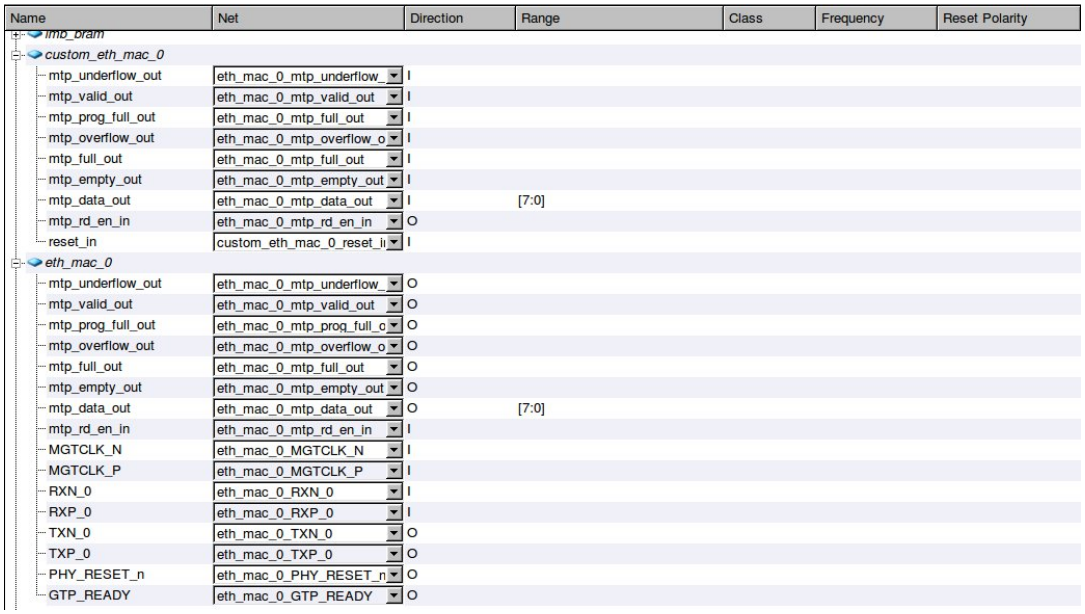


Στην συνέχεια πρέπει να γίνουν οι συνδέσεις των εισόδων και εξόδων των δύο περιφερειακών. Οι εξόδοι του περιφερειακού eth_mac (mtp_data_out, mtp_empty_out, mtp_full_out, mtp_overflow_out, mtp_prog_full_out, mtp_valid_out, mtp_underflow_out) γίνονται είσοδοι στο περιφερειακό custom_eth_mac. Η έξοδος mtp_rd_en_in του περιφερειακού custom_eth_mac γίνεται είσοδος του περιφερειακού eth_mac.

Οι υπόλοιπες εισόδοι/εξόδοι των δύο περιφερειακών ενώνονται με τον έξω κόσμο με την χρήση της επιλογής Make External.

Οι συνδέσεις που περιγράφηκαν πιο πάνω φαίνονται και στα παρακάτω σχήματα:





Name	Net	Direction	Range	Class	Frequency	Reset Polarity
lmb_bram						
custom_eth_mac_0						
mtp_underflow_out	eth_mac_0_mtp_underflow_	I				
mtp_valid_out	eth_mac_0_mtp_valid_out	I				
mtp_prog_full_out	eth_mac_0_mtp_full_out	I				
mtp_overflow_out	eth_mac_0_mtp_overflow_o	I				
mtp_full_out	eth_mac_0_mtp_full_out	I				
mtp_empty_out	eth_mac_0_mtp_empty_out	I				
mtp_data_out	eth_mac_0_mtp_data_out	I	[7:0]			
mtp_rd_en_in	eth_mac_0_mtp_rd_en_in	O				
reset_in	custom_eth_mac_0_reset_i	I				
eth_mac_0						
mtp_underflow_out	eth_mac_0_mtp_underflow_	O				
mtp_valid_out	eth_mac_0_mtp_valid_out	O				
mtp_prog_full_out	eth_mac_0_mtp_prog_full_o	O				
mtp_overflow_out	eth_mac_0_mtp_overflow_o	O				
mtp_full_out	eth_mac_0_mtp_full_out	O				
mtp_empty_out	eth_mac_0_mtp_empty_out	O				
mtp_data_out	eth_mac_0_mtp_data_out	O	[7:0]			
mtp_rd_en_in	eth_mac_0_mtp_rd_en_in	I				
MGTCLK_N	eth_mac_0_MGTCLK_N	I				
MGTCLK_P	eth_mac_0_MGTCLK_P	I				
RXN_0	eth_mac_0_RXN_0	I				
RXP_0	eth_mac_0_RXP_0	I				
TXN_0	eth_mac_0_TXN_0	O				
TXP_0	eth_mac_0_TXP_0	O				
PHY_RESET_n	eth_mac_0_PHY_RESET_n	O				
GTP_READY	eth_mac_0_GTP_READY	O				

Τέλος για να ολοκληρώσουμε την διαδικασία της συνδεσμολογίας πρέπει να τροποποιήσουμε το UCF αρχείο. Το τελικό UCF πρέπει να έχει την ακόλουθη μορφή:

```
##############################################################################
## This system.ucf file is generated by Base System Builder based on the
## settings in the selected Xilinx Board Definition file. Please add other
## user constraints to this file based on customer design specifications.
##############################################################################

Net sys_clk_pin TNM_NET = sys_clk_pin;
Net sys_clk_pin LOC = AH15;
Net sys_clk_pin IOSTANDARD=LVCMOS33;
Net sys_rst_pin LOC = E9;
Net sys_rst_pin IOSTANDARD=LVCMOS33;
Net sys_rst_pin PULLUP;
## System level constraints
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 10000 ps;
Net sys_rst_pin TIG;
```

```
Net reset_in LOC = AJ6;

#### Module RS232_Uart_1 constraints

## IO Devices constraints

Net fpga_0_RS232_Uart_1_RX_pin LOC = AG15;
Net fpga_0_RS232_Uart_1_RX_pin IOSTANDARD=LVCMOS33;
Net fpga_0_RS232_Uart_1_TX_pin LOC = AG20;
Net fpga_0_RS232_Uart_1_TX_pin IOSTANDARD=LVCMOS33;

CONFIG PART = 5vlx110tff1136-1;

##################################
# BLOCK Level constraints
##################################

# EMAC0 Clocking
# 125MHz clock input from BUFG
NET "*CLK125" TNM_NET = "clk_gtp";
TIMEGRP  "v5_emac_v1_5_gtp_clk" = "clk_gtp";
TIMESPEC "TS_v5_emac_v1_5_gtp_clk" = PERIOD
      "v5_emac_v1_5_gtp_clk" 7700 ps HIGH 50 %;

##################################
# LocalLink Level constraints
##################################

# EMAC0 LocalLink client FIFO constraints.

INST "*client_side_FIFO_emac0?tx_fifo_i?rd_tran_frame_tog"
      TNM = "tx_fifo_rd_to_wr_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?rd_retran_frame_tog"
      TNM = "tx_fifo_rd_to_wr_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?rd_col_window_pipe_1"
      TNM = "tx_fifo_rd_to_wr_0";
```

```
INST "*client_side_FIFO_emac0?tx_fifo_i?rd_addr_txfer*"
      TNM = "tx_fifo_rd_to_wr_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?rd_txfer_tog"
      TNM = "tx_fifo_rd_to_wr_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_frame_in_fifo"
      TNM = "tx_fifo_wr_to_rd_0";

TIMESPEC "TS_tx_fifo_rd_to_wr_0" = FROM "tx_fifo_rd_to_wr_0"
      TO "v5_emac_v1_5_gtp_clk" 8000 ps DATAPATHONLY;
TIMESPEC "TS_tx_fifo_wr_to_rd_0" = FROM "tx_fifo_wr_to_rd_0"
      TO "v5_emac_v1_5_gtp_clk" 8000 ps DATAPATHONLY;

# Reduce clock period to allow 3 ns for metastability settling time
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_tran_frame_tog"
      TNM = "tx_metastable_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_rd_addr*"
      TNM = "tx_metastable_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_txfer_tog"
      TNM = "tx_metastable_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?frame_in_fifo"
      TNM = "tx_metastable_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_retran_frame_tog*"
      TNM = "tx_metastable_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_col_window_pipe_0"
      TNM = "tx_metastable_0";

TIMESPEC "ts_tx_meta_protect_0" = FROM "tx_metastable_0" 5 ns DATAPATHONLY;

INST "*client_side_FIFO_emac0?tx_fifo_i?rd_addr_txfer*"
      TNM = "tx_addr_rd_0";
INST "*client_side_FIFO_emac0?tx_fifo_i?wr_rd_addr*"
      TNM = "tx_addr_wr_0";
TIMESPEC "TS_tx_fifo_addr_0" = FROM "tx_addr_rd_0" TO
      "tx_addr_wr_0" 10ns;

## RX Client FIFO
```

```
# Group the clock crossing signals into timing groups
INST "*client_side_FIFO_emac0?rx_fifo_i?wr_store_frame_tog"
      TNM = "rx_fifo_wr_to_rd_0";
INST "*client_side_FIFO_emac0?rx_fifo_i?rd_addr_gray*"
      TNM = "rx_fifo_rd_to_wr_0";

TIMESPEC "TS_rx_fifo_wr_to_rd_0" = FROM "rx_fifo_wr_to_rd_0" TO
      "v5_emac_v1_5_gtp_clk" 8000 ps DATAPATHONLY;
TIMESPEC "TS_rx_fifo_rd_to_wr_0" = FROM "rx_fifo_rd_to_wr_0" TO
      "v5_emac_v1_5_gtp_clk" 8000 ps DATAPATHONLY;


# Reduce clock period to allow for metastability settling time
INST "*client_side_FIFO_emac0?rx_fifo_i?wr_rd_addr_gray_sync*"
      TNM = "rx_metastable_0";
INST "*client_side_FIFO_emac0?rx_fifo_i?rd_store_frame_tog"
      TNM = "rx_metastable_0";

TIMESPEC "ts_rx_meta_protect_0" = FROM "rx_metastable_0" 5 ns;

###################################
# EXAMPLE DESIGN Level constraints
###################################

# Place the transceiver components. Please alter to your chosen transceiver.
INST "*GTP_DUAL_1000X_inst?GTP_1000X?tile0_rocketio_wrapper_i?gtp_dual_i"
      LOC = "GTP_DUAL_X0Y4";
INST MGTCLK_N_pin LOC = P3;
INST MGTCLK_P_pin LOC = P4;

# RXN 0 pin
Net RXN_0_pin LOC = P1;

# RXP 0 pin
Net RXP_0_pin LOC = N1;
```

```
# TXN 0 pin
Net TXN_0_pin LOC = N2;

# TXP 0 pin
Net TXP_0_pin LOC = M2;

# GTP Ready pin
NET GTP_READY_pin LOC = AE24; # LED 7

# PHY Reset pin
NET PHY_RESET_n_pin LOC = J14;
Net PHY_RESET_n_pin IOSTANDARD = LVCMOS25;
Net PHY_RESET_n_pin TIG;

# PHY Autonegotiate ON
INST *?v5_emac EMAC0_PHYINITAUTONEG_ENABLE = TRUE;
```

Για την επιστροφή τιμών στον ηλεκτρονικό υπολογιστή γίνεται χρήση της σειριακής θύρας RS-232. Για να είναι αυτό εφικτό πρέπει να τροποποιηθεί ο κώδικας του Microblaze και ο οποίος βρίσκεται στο μονοπάτι /project/path/TestApp_Peripheral/src. Ένα παράδειγμα κώδικα είναι το ακόλουθο:

```
#include "xparameters.h"

#include "stdio.h"

#include "xutil.h"

#include "custom_eth_mac.h"

int main (void)
{

    int peripheralResult = 0;
    xil_printf("- Entering main() -");
    while(1){
            microblaze_bread_datafsl(peripheralResult,0);
```

```
            xil_printf("Result: %d",peripheralResult);
    }
    print("– Exiting main() –");
}
```

C.6 Κώδικας χρήσης Gigabit Ethernet

Αρχικά πρέπει να δημιουργηθούν τα πακέτα που θα αποσταλούν στο αναπτυξιακό.

```
f = open('package','wb')
loop = 1048576
for k in range(loop):
    for l in range(4):
        count = 0
        for d in range(256):
            f.write(chr(count))
            count = count + 1
f.close()
```

Στην συνέχεια με ένα δεύτερο κώδικα Python στον οποίο γίνεται χρήση ενός UDP socket αποστέλλονται τα πακέτα στο αναπτυξιακό.

```
# Client program

from socket import *
import os

# Set the socket parameters host = '1.2.3.5' # FPGA IP
port = 5000 # MTP port
buf = 512 # data size in Bytes
addr = (host,port)

numberofpackets = 0
# Create socket
UDPSock = socket(AF_INET,SOCK_DGRAM)

mtp_Spacket ="S0000"

UDPSock.sendto(mtp_Spacket,addr)

f = open('package','r')

loop = 1048576
for k in range(loop):
        firstbyte = k // 16777216
        numberofpackets_help = k % 16777216

        secondbyte = numberofpackets_help // 65536
        numberofpackets_help = numberofpackets_help % 65536

        thirdbyte = numberofpackets_help // 256
        numberofpackets_help = numberofpackets_help % 256

        fourthbyte = numberofpackets_help

        a=f.read(1024)
        if k == loop - 1:
                temp = 'D'+chr(firstbyte)+chr(secondbyte)+chr(thirdbyte)+chr(fourthbyte)
```

```
                + a + chr(255)
        else:
                temp = 'D'+chr(firstbyte)+chr(secondbyte)+chr(thirdbyte)+chr(fourthbyte)
                + a
        UDPSock.sendto(temp,addr)

f.close()
endPacketfirstByte = (loop+1) // 16777216
endnumberofpackets_help = (loop+1) % 16777216

endPacketsecondbyte = endnumberofpackets_help // 65536
endnumberofpackets_help = endnumberofpackets_help % 65536
endPacketthirdbyte = endnumberofpackets_help // 256
endnumberofpackets_help = endnumberofpackets_help % 256

endPacketfourthbyte = endnumberofpackets_help
mtp_Epacket = 'E'+chr(endPacketfirstByte)+chr(endPacketsecondbyte)
            +chr(endPacketthirdbyte)+chr(endPacketfourthbyte)

UDPSock.sendto(mtp_Epacket,addr)

UDPSock.close()
```