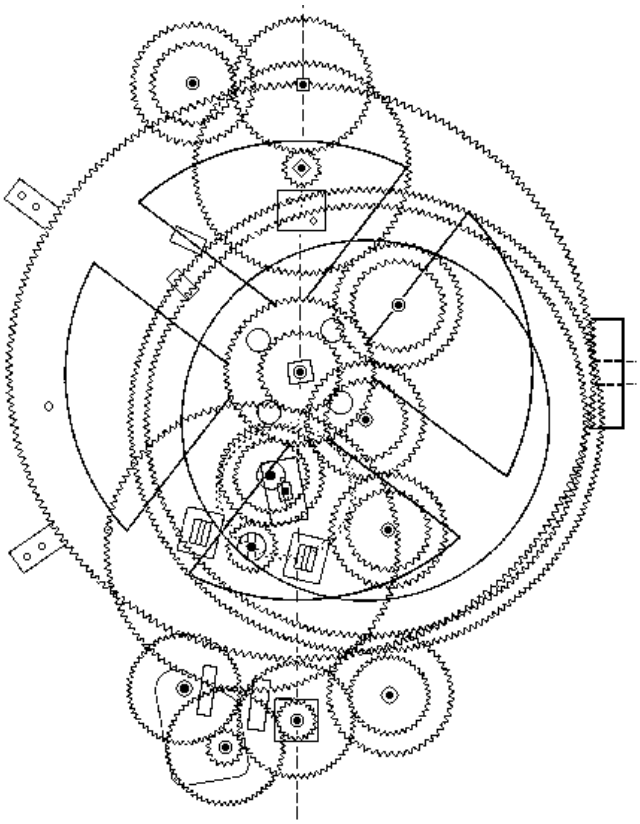


Διπλωματική Εργασία με θέμα:

"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"

Επιβλέπων καθηγητής:

Παπαευσταθίου Ιωάννης

Επιβλέπων διδακτορικός φοιτητής:

Μεϊντάνης Δημήτριος

Του προπτυχιακού φοιτητή:

Καρτσωνάκη Ιωάννη

I tamed the Intrepid Ibex

Περιεχόμενα:

Περίληψη

Στην αναζήτηση τρόπου γρήγορης επικοινωνίας και μεταφοράς δεδομένων μεταξύ ενός Ηλεκτρονικού Υπολογιστή και μιας αναδιατασσομένης συσκευής Virtex-5 FPGA, καταλήξαμε στην χρήση του διαύλου PCI-Express. Για να υποστηρίξουμε την επικοινωνία μέσω του διαύλου αυτού, κληθήκαμε να συντάξουμε έναν οδηγό λειτουργικού συστήματος (device driver), για το ελεύθερο λογισμικό Linux, σε γλώσσα προγραμματισμού C. Επίσης δημιουργήσαμε την σχεδίαση που υλοποιεί τον πυρήνα της αναδιατασσόμενης συσκευής, σε γλώσσα προγραμματισμού VHDL, που υποστηρίζει την επικοινωνία μέσω του διαύλου PCI-Express με μια πλακέτα XUP-V5 της εταιρίας Xilinx. Η μεταφορά δεδομένων γίνεται με την βοήθεια ενός εκτελέσιμου προγράμματος σε επίπεδο χρήστη (user level), το οποίο συντάχθηκε με την βοήθεια της γλώσσας προγραμματισμού C και υποστηρίζει τις βασικές εντολές read/write δεδομένων από και προς την FPGA συσκευή. Στην διπλωματική αυτή πετύχαμε την αξιόπιστη μεταφορά δεδομένων μεταξύ του ηλεκτρονικού υπολογιστή και της περιφερειακής συσκευής XUP-V5, μέσω του διαύλου PCI-Express.

2. Εισαγωγή

Εισαγωγή

Ενώ οι τεχνολογία αναδιατασσόμενων συσκευών εξελίσσεται και προσφέρει στους μηχανικούς ταχύτητα και αξιοπιστία επεξεργασίας δεδομένων, καθώς και πληθώρα νέων ιδιοτήτων, δεν υπάρχει ένας γρήγορος και αξιόπιστος τρόπος να μεταφέρουμε δεδομένα στην FPGA από το επίπεδο χρήστη (user level). Οι σύγχρονες συσκευές που επικρατούν στην αγορά υποστηρίζουν ένα μεγάλο αριθμό από ταχύτατους τρόπους επικοινωνίας όπως USB 2.0, Ethernet, PCI-Express κ.α. αλλά δεν υπάρχει το ελεύθερο λογισμικό για την υποστήριξη της επικοινωνίας αυτής με το λογισμικό σύστημα. Φυσικά υπάρχουν εταιρίες οι οποίες αναλαμβάνουν να λύσουν αυτό το πρόβλημα, προσφέροντας στο κοινό Drivers που υποστηρίζουν την επικοινωνία (και όχι μόνο) μέσω των διαύλων αυτών, ανάλογα με το λειτουργικό σύστημα (Windows, Linux, Macintosh) και ανάλογα με την αναδιατασσόμενη συσκευή (πχ Virtex-II, Virtex-5, Virtex-6 για την εταιρία Xilinx) που χρησιμοποιείται.

Εμείς σαν ίδρυμα με λαμπρό ιστορικό και διακρίσεις στην ευρεία επιστημονική κοινότητα, και ιδιαίτερα στον τομέα των Μικροεπεξεργαστών και Υλικού (MHL) σκεφτήκαμε να συντάξουμε έναν οδηγό λογισμικού που να υποστηρίζει την επικοινωνία μεταξύ της γενιάς Virtex-5 αναδιατασσόμενων συσκευών (η Virtex-6 παρουσιάστηκε αφού είχαμε ολοκληρώσει τον Οδηγό μας) με τον Ηλεκτρονικό Υπολογιστή μέσω του διαύλου PCI-Express. Με τον τρόπο αυτό, συνάδελφοι εντός και εκτός του ιδρύματος μας θα έχουν την δυνατότητα να επωφεληθούν τον Driver μας καθώς η αρχική έκδοση του θα οριστεί ως shareware και θα διανεμηθεί στην επιστημονική κοινότητα και στο internet με πολλαπλή σκοπιμότητα όπως να χρησιμοποιηθεί από συναδέλφους μηχανικούς ή να βελτιωθεί και εμπλουτιστεί με νέες λειτουργίες.

Η έλλειψη τεχνογνωσίας και η μοναδικότητα του εγχειρήματος αυτού το κατέστησε σίγουρα ως μια μεγάλη πρόκληση για τον μηχανικό, αλλά και μια μεγάλη «δυσκολία» καθώς τα προβλήματα που παρουσιάστηκαν σαφώς και ήταν πολλά, άσχετα αν η λύση ήταν απλή ή σύνθετη, η μη ύπαρξη feedback προκαλούσε κωλυσιεργίες στην υλοποίηση για απλά προβλήματα που παρουσιάζονταν. Σαφώς με υπομονή και καθαρό μυαλό αυτά ξεπεράστηκαν, και έτσι ο πρώτος Shareware Linux Driver που υποστηρίζει την επικοινωνία της αναδιατασσόμενης συσκευής Virtex-5 embedded σε board xupv5-lx110t της εταιρίας Xilinx μέσω του διαύλου PCI-Express 1x Lane είναι γεγονός.

3. Εφάμιλλη Εργασία

Εφάμιλλη Εργασία

Shareware Linux Driver

Έπειτα από αναζήτηση στο διαδίκτυο δεν βρήκαμε κάποιο shareware Οδηγό ο οποίος να είναι εκτελέσιμος ή να προορίζεται για την γενιά αναδιατασσόμενων συσκευών Virtex-5.

Η μόνη shareware υλοποίηση η οποία βρέθηκε στο διαδίκτυο, και μας έδωσε ένα πλάνο πάνω στο οποίο να στηριχθούμε και να δούμε την μορφή που οφείλει να έχει ένας οδηγός λογισμικού που προορίζεται για την υποστήριξη επικοινωνίας μέσω PCI-Express ήταν ένας οδηγός που υποστηρίζει την επικοινωνία μιας συσκευής Spartan-3 της Xilinx με τον Ηλεκτρονικό Υπολογιστή μέσω του διαύλου PCI-Express χωρίς όμως να παρέχεται ο κώδικας VHDL για την Spartan-3. Επίσης το λογισμικό για το οποίο προορίζονταν ο Οδηγός αυτός ήταν για μια πεπαλαιωμένη έκδοση Linux Kernel.

Ουσιαστικά ο συγκεκριμένος οδηγός δεν ήταν εκτελέσιμος από εμάς, αλλά μας βοήθησε στο να κατανοήσουμε την μορφή που πρέπει να έχει ο Linux Driver και να δούμε στην πράξη πως συντάσσεται ένας οδηγός ελεύθερου λογισμικού, καθώς οι μέχρι τώρα πληροφορίες που είχαμε συλλέξει ήταν σε θεωρητικό επίπεδο και δεν ήταν εύκολα κατανοητή η σωστή δομή που πρέπει να ακολουθηθεί.

Ο συνάδελφος μηχανικός που σύνταξε τον παραπάνω Οδηγό λογισμικού ονομάζεται *Wojciech A. Koszek*. Από έρευνα που πραγματοποιήθηκε στο διαδίκτυο δεν βρέθηκε κάπου το αποτέλεσμα της δουλειάς του, ή μετρήσεις ταχύτητας μεταφοράς δεδομένων που υποστηρίζει με τον Driver της υλοποίησης του. Τον Driver αυτόν τον αποθηκεύσαμε από το διαδίκτυο από τον ιστοχώρο *Xilinx User Community Forums* της εταιρίας Xilinx.

http://forums.xilinx.com/xlnx/

Pay n Play

Πολλές εταιρίες προμηθεύουν την αγορά με Drivers που υποστηρίζουν την επικοινωνία μέσω του διαύλου PCI-Express.

Μια από αυτές είναι η Jungo η οποία προσφέρει οδηγούς που υποστηρίζουν την επικοινωνία μέσω PCI/PCI-e για λειτουργικά συστήματα Windows/Linux για οποιαδήποτε έκδοση Silicon Device κυκλοφορεί στο εμπόριο.

Ενδεικτικά, οι τιμές των προϊόντων WinDriver της Jungo για υποστήριξη επικοινωνίας μέσω PCIe για το λειτουργικό σύστημα Linux κυμαίνονται από $3.999 για Licence ενός Ηλεκτρονικού Υπολογιστή και φτάνουν έως τα $7.999 για πακέτο με Licence για τρεις Ηλεκτρονικούς Υπολογιστές. Η έκδοση του όμοιου Driver για το λειτουργικό σύστημα των Windows κυμαίνεται στα ίδια επίπεδα τιμών.

Η εταιρία υπόσχεται εύκολη εγκατάσταση και χρηστικότητα ακόμα και από άτομα τα οποία δεν είναι εξειδικευμένα σε αυτόν τον τομέα της επιστήμης και άμεση υποστήριξη στην περίπτωση που αυτή είναι απαραίτητη.

Παρακάτω επισημαίνεται η διεύθυνση της εταιρίας αυτής για τα προϊόντα της όσων αφορά τον δίαυλο PCI-Express

http://www.jungo.com/st/windriver_driver_development_pci_express.html

FPGA Bridges

PCI Express-to-High Speed Serial (HSS) bridge

http://www.design-reuse.com/news/20205/pci-express-bridge-voip-platform.html

PCIe-to-Local Bus Bridge

http://www.pldesignline.com/showArticle.jhtml;jsessionid=BXLBEKE1MRZW1QE1GHPSKHWATMY32JVN?articleID=210300269

AHB to PCIe Bridge

http://www.fpgacentral.com/fpga-ip/core/amba-ahb-pci-bridge

4. Θεωρητικά

Θεωρητικά

Λειτουργικό Σύστημα Linux

Το λειτουργικό σύστημα ελευθέρου λογισμικού το οποίο επιλέχθηκε για να υποδεχθεί τον οδηγό που δημιουργήσαμε δεν ήταν άλλο από ένα ευρείας διάδοσης Linux λειτουργικό σύστημα, και συγκεκριμένα, η τελευταία τότε έκδοση **Ubuntu 08.10 Intrepid Ibex** (Ατρόμητο Κρι-Κρι). Ο αριθμός έκδοσης 08.10 σημαίνει ότι η ολοκληρωμένη έκδοση του συγκεκριμένου λογισμικού δόθηκε στην κυκλοφορία τον $10^{ο}$ μήνα του 2008.

Η λήψη της έκδοσης του λειτουργικού συστήματος ελευθέρου λογισμικού έγινε από την επίσημη ιστοσελίδα της ομάδας η οποία είναι η εξής:

http://www.ubuntu.com/getubuntu/download

Το λογισμικό αυτό επιλέχθηκε επειδή είναι ελεύθερο λογισμικό, και μας δίνει την δυνατότητα να κάνουμε παρεμβάσεις στον πυρήνα του σχετικά εύκολα χωρίς να χρειάζεται να αποκτήσουμε κάποια έγκριση από την ομάδα παραγωγής του.

Επίσης είναι ευρέως διαδεδομένο, το οποίο σημαίνει ότι πρώτων έχουν γίνει δοκιμές και έχει ελεγχθεί η σταθερότητα του σαν λειτουργικό σύστημα, και δεύτερον μπορεί εύκολα κάποιος που έχει υπολογιστή να εγκαταστήσει το λειτουργικό σύστημα και να εκτελέσει τον οδηγό του διαύλου PCI-Express στον υπολογιστή του.

Τέλος, ένα ακόμα πλεονέκτημα είναι ότι οι μεταγενέστερες εκδόσεις Linux, και ένα παραπάνω τα Ubuntu, ακόμα και αν αλλάξει η έκδοση που χρησιμοποιείται ο Οδηγός θα παραμένει εκτελέσιμος και σε αυτές, αφού υπάρχει συμβατότητα από το παρών προς το μέλλων. Στην ανάποδη περίπτωση, ένα λειτουργικό σύστημα προγενέστερο της έκδοσης 08.10 ίσως παρουσιάζει μικρό-αλλαγές και προβλήματα στην εκτέλεση του οδηγού, αλλά και σε αυτήν την περίπτωση, οι αλλαγές περιορίζονται κυρίως σε αλλαγές αρχείων επικεφαλίδων (header files *.h) και οι οποίες είναι εύκολα αντιστρέψιμες.

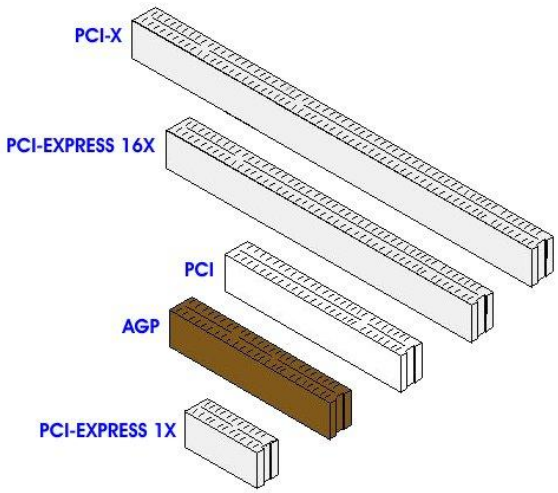
Επειδή οι ομάδες παραγωγής των διαφόρων εκδόσεων λειτουργικών συστημάτων ελευθέρου λογισμικού δεν περιέχουν όλα τα αρχεία του πυρήνα στην έκδοση που αποθηκεύουμε από την ιστοσελίδα τους άλλα τις αναγκαίες βιβλιοθήκες οι οποίες είναι απαραίτητες για την λειτουργία του πυρήνα. Για να κάνουμε εκτέλεση και εγκατάσταση των αρχείων του πυρήνα, κατεβάσαμε από την ιστοσελίδα

http://www.kernel.org/

την τελευταία σταθερή πλήρη έκδοση αρχείων πυρήνα η οποία ήταν η υπ'αριθμό **2.6.28.7**

Δίαυλος PCI-Express

Το όνομα του διαύλου PCI-Express προέρχεται από τα αρχικά των λέξεων *Peripheral Component Interconnect Express*. Παρουσιάστηκε το 2004 από την εταιρία Intel σε αντικατάσταση των προηγούμενων διαύλων PCI και AGP στον εξοπλισμό των ηλεκτρονικών υπολογιστών. Χρησιμεύει για την ένωση πρόσθετων καρτών μονάδων με την ομάδα κεντρικών ολοκληρωμένων κυκλωμάτων στην μητρική πλακέτα του υπολογιστή.



Περιγραφή του διαύλου

Σε σύγκριση με τον παλαιότερο δίαυλο PCI, ο δίαυλος PCIe δεν είναι κοινό σύστημα διαύλων, αλλά σύνολο χωριστών σειριακών απευθείας συνδέσεων. Η μεταφορά των δεδομένων γίνεται μέσω παρόδων (lanes), ενώ κάθε πάροδος αποτελείται από ένα ζεύγος αγωγών για την εκπομπή και ένα δεύτερο ζεύγος αγωγών για την λήψη των δεδομένων.

Η σύνδεση των παρόδων μεταξύ της ομάδας των κεντρικών ολοκληρωμένων κυκλωμάτων και των πρόσθετων υλικών (τις κάρτες δηλαδή) γίνεται με την βοήθεια ηλεκτρονικού μεταγωγέα. Για την κωδικοποίηση των δεδομένων εφαρμόζεται ο κώδικας 8B/10B.

Παρόλο που ο τρόπος λειτουργίας του είναι εντελώς αντίθετη με αυτή του παλαιότερου υλικού, το PCIe είναι πλήρως συμβατικό και οι κάρτες που χρησιμοποιούνται δεν επιβάλουν καμία ανάγκη μετατροπής, ούτε στο λειτουργικό σύστημα, αλλά ούτε και σε οδηγούς και λογισμικά.

Ο δίαυλος PCIe είναι πλήρως διπλής κατεύθυνσης (fully duplex) επικοινωνίας με ταχύτητα 1,25 GHz, η οποία θεωρητικά επιτρέπει ως μέγιστη ταχύτητα μεταφοράς δεδομένων τα 250 MByte/sec για κάθε πάροδο προς κάθε κατεύθυνση. Στην πραγματικότητα γίνονται εφικτές ταχύτητες των 240 MByte/s όταν μεταφέρονται δεδομένα μεγάλης διάρκειας.

Όταν χρησιμοποιείται μόνο μία πάροδος, τότε ο δίαυλος ονομάζεται PCIe x1. Οι πάροδοι μπορούν να χρησιμοποιούνται και σε συνδυασμό, έτσι ώστε να αυξάνεται η ταχύτητα.

PCIe x1 - χρησιμοποιεί μία πάροδο
PCIe x4 - χρησιμοποιεί τέσσερις παρόδους
PCIe x16 - χρησιμοποιεί δέκα έξι παρόδους
PCIe x32 - χρησιμοποιεί τριάντα δύο παρόδους

Συνήθως χρησιμοποιείται ο τύπος PCIe x1 ως αντικαταστάτης του παλαιότερου PCI, ενώ για τις κάρτες γραφικών χρησιμοποιείται ο τύπος PCIe x16ως αντικαταστάτης του παλαιότερου AGP.

Στους επαγγελματικούς επεξεργαστές χρησιμοποιούνται επιπλέον οι τύποι x4, x8 και x32.
Τα βύσματα των διαφόρων τύπων είναι συμβατά προς τα πάνω, δηλαδή βύσματα με μικρότερο αριθμό στον τύπο χωράνε σε πρίζες με τον ίδιο ή μεγαλύτερο αριθμό στον τύπο. Σε αυτή την περίπτωση, οι πάροδοι που δεν τροφοδοτούνται μένουν ανενεργοί.

Χαρακτηριστικά του Endpoint Block Plus v1.12 για το PCI Express®

Τα χαρακτηριστικά του Endpoint Block της Xilinx για το PCI Express:

- Υψηλή απόδοση, ευελιξία, επεκτασιμότητα και αξιοπιστία σε συναλλαγές I/O με τον πυρήνα.
  - Σε συμμόρφωση με τα χαρακτηριστικά που αναφέρονται στην PCI Express Base Specification v1.1.
  - Συμβατό με το συμβατικό μοντέλο λογισμικού PCI.
- Ενσωματώνει την τεχνολογία της Xilinx Smart-IP™ που εγγυάται απόλυτο συγχρονισμό.
- Χρησιμοποιεί τους RocketIO GTP Transceivers για τις συσκευές Virtex-5 LXT, SXT και TXT ενώ χρησιμοποιεί τους RocketIO GTX Transceivers για τις συσκευές Virtex-5 FXT.
  - 2.5 Gbps ταχύτητα γραμμής (Line speed)
  - Υποστηρίζει λειτουργία σε 1-lane, 2-lane, 4-lane, 8-lane.
  - Ελαστικές μνήμες (elastic buffers) και παραγωγή ρολογιών
  - Αυτόματη επαναφορά ρολογιού
- Υποστηρίζει 8B/10B κωδικοποίηση και αποκωδικοποίηση δεδομένων
- Υποστηρίζει αναστροφή και αλλαγή πολικότητας των γραμμών όταν το απαιτεί η σχεδίαση.
- Τυποποιημένη διεπαφή για τον χρήστη (user interface)
  - Εύκολο στη χρήση πρωτόκολλο βασισμένο στην μεταφορά πακέτων
  - Πλήρως αμφίδρομη (full duplex) επικοινωνία
  - Αλλεπάλληλες συναλλαγές κάνουν δυνατή την εκμετάλλευση μεγαλύτερου εύρους μεταφοράς δεδομένων
  - Υποστηρίζει έλεγχο ροής (flow control) και διακοπή συναλλαγής που βρίσκεται σε λειτουργία για δεδομένα που μεταδίδονται.
  - Υποστηρίζει έλεγχος ροής δεδομένων που λαμβάνονται
- Υποστηρίζει απομάκρυνση των εσφαλμένων πακέτων για αναγνώριση σφαλμάτων και επαναφορά.

- Σε συμμόρφωση με τις λειτουργίες διαχείρισης ενέργειας του PCI/PCI Express
- Υποστηρίζει μέγιστο ωφέλιμο φορτίο ανά συναλλαγή τα 512bytes
- Πλήρως συμβατό με τους κανόνες διευθέτησης συναλλαγών του PCI-Express

Aurora

Το πυρήνας 8B/10B Aurora αποτελεί ένα πρωτόκολλο υψηλής ταχύτητας σειριακής επικοινωνίας. Το πρωτόκολλο είναι ανοιχτό από άποψη registration, και άδεια χρήσης του μπορεί να εξασφαλίσει οποιοσδήποτε ενδιαφέρεται να το χρησιμοποιήσει.

Παραγωγή με τα εργαλεία της Xilinx

Η παραγωγή του πυρήνα του Aurora γίνεται με τα εργαλεία της Xilinx. Υπάρχουν διαθέσιμες εκδόσεις για τις περισσότερες συσκευές που υποστηρίζουν την επικοινωνία μέσω του πρωτοκόλλου aurora. Το πρωτόκολλο χρησιμοποιείται συνήθως σε εφαρμογές που απαιτούν απλές υλοποιήσεις, με γρήγορη μεταφορά δεδομένων και χαμηλό κόστος.

Το λογισμικό Core Generator της Xilinx παρέχει τον πηγαίο κώδικα υλοποίησης του πρωτοκόλλου Aurora με κωδικοποίηση 8B/10B (είναι δυνατή και η κωδικοποίηση 64B/66B) και με επιλεγόμενο μέγεθος των δεδομένων που μεταφέρονται (data width).

Οι πυρήνες που δημιουργούνται υποστηρίζουν επικοινωνία που μπορεί να είναι μονόδρομη (simplex) ή πλήρως αμφίδρομη (full-duplex). Επίσης διαθέτουν διεπαφή σε επίπεδο χρήστη (user interface) και προαιρετικά έναν ελεγκτή ροής δεδομένων (flow control).

Τα προτερήματα του πρωτοκόλλου Aurora είναι:
- Γενικής σκοπιμότητας κανάλια δεδομένων με throughput που κυμαίνεται από τα 400Mbps έως τα 83.2Gbps
- Υποστηρίζει έως 16 από τους 48 RocketIO GTP/GTX Transceivers της αναδιατασσόμενης συσκευής Virtex-5 FPGA ή 16 GTX transceivers της συσκευής Virtex-6 FPGA.

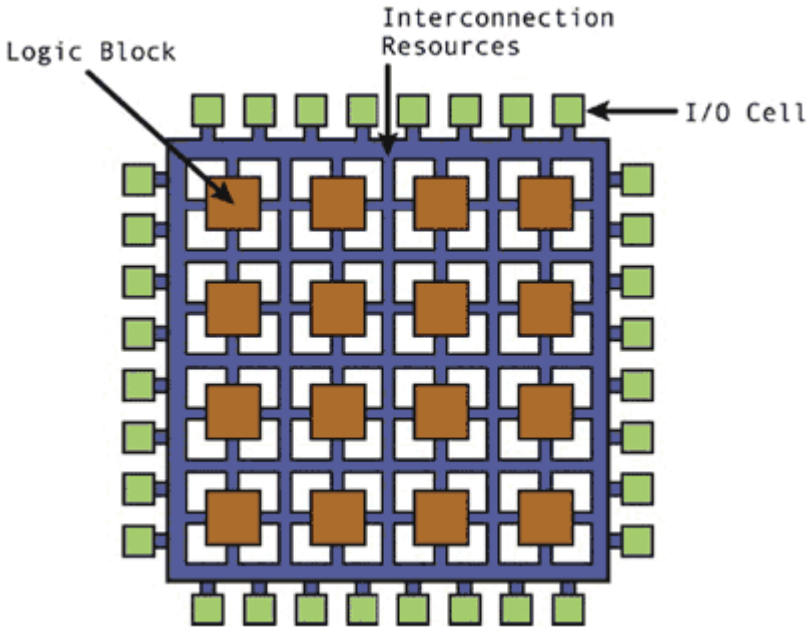
Ο πυρήνας Aurora με κωδικοποίηση 8B/10B υποστηρίζει τις παρακάτω ταχύτητες μεταφοράς δεδομένων ανά lane.
- Virtex-5 FPGA RocketIO GTP: 500 Mbps έως 3.75 Gbps
- Virtex-5 FPGA RocketIO GTX: 750 Mbps έως 6.5 Gbps
- Virtex-6 FPGA GTX: 750 Mbps έως 6.5 Gbps

Διατάξεις Πυλών Προγραμματιζόμενου Πεδίου (FPGA)

Οι λογικές Διατάξεις Πυλών Προγραμματιζόμενου Πεδίου (Field Programmable Gate Array) είναι διατάξεις προγραμματιζόμενης λογικής που υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων. Οι διατάξεις FPGA περιέχουν λογικές βαθμίδες για την υλοποίηση των ζητούμενων συναρτήσεων.

Γενική Δομή

Η γενική δομή μιας διάταξης FPGA περιέχει τρία είδη πόρων: λογικές βαθμίδες, βαθμίδες εισόδου/εξόδου για την σύνδεση με τους ακροδέκτες της συσκευασίας και διακόπτες, και γραμμές εσωτερικής διασύνδεσης.



Οι λογικές βαθμίδες οργανώνονται με την μορφή διδιάστατης σειράς και οι γραμμές σύνδεσης οργανώνονται ως οριζόντια και κατακόρυφα κανάλια δρομολόγησης (routing channels) ανάμεσα στις γραμμές και στήλες των λογικών βαθμίδων. Τα κανάλια αυτά εμπεριέχουν καλώδια και προγραμματιζόμενους διακόπτες που επιτρέπουν τις λογικές βαθμίδες να διασυνδέονται πολλούς τρόπους.

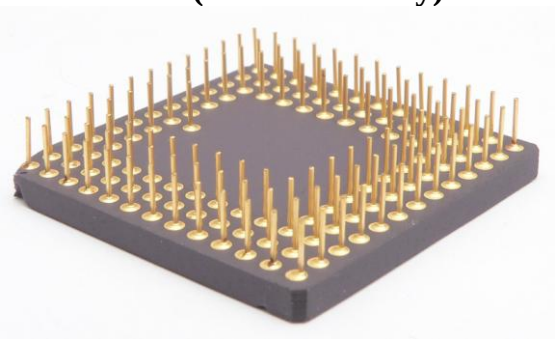
Οι διατάξεις FPGA μπορούν να χρησιμοποιηθούν για την υλοποίηση λογικών κυκλωμάτων μεγέθους μεγαλύτερου από μερικές εκατοντάδες χιλιάδες ισοδύναμες λογικές πύλες.

Οι διατάξεις FPGA διατίθενται σε διάφορες συσκευασίες, περιλαμβάνοντας τις παρακάτω:

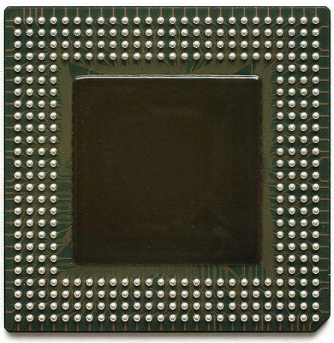
PLCC (Plastic leaded chip carrier)

QFP (Quad Flat Package)

PGA (Pin Grid Array)



BGA (Ball Grid Array)



Κάθε λογική βαθμίδα ενός FPGA έχει τυπικά ένα μικρό αριθμό εισόδων και μια έξοδο. Σήμερα υπάρχουν στο εμπόριο ένας αριθμός προϊόντων FPGA τα οποία διαθέτουν διάφορα είδη λογικών βαθμίδων. Η πιο ευρέως χρησιμοποιούμενη λογική βαθμίδα είναι ο πίνακας αναζήτησης ή αναφοράς (lookup table, LUT), ο οποίος περιέχει κυψέλες αποθήκευσης (storage cells) που χρησιμοποιούνται για την υλοποίηση μιας μικρής λογικής συνάρτησης. Κάθε κυψέλη μπορεί να κρατήσει μία λογική τιμή 0 ή 1. Η αποθηκευμένη τιμή μεταφέρεται στην έξοδο της κυψέλης αποθήκευσης. Μπορούν να αναπτυχθούν πίνακες LUT σε διάφορα μεγέθη. Όπου το μέγεθος ορίζεται από τον αριθμό των εισόδων.

5. Οδηγός λογισμικού

The PCI Express Port Bus Driver

Χαρακτηριστικά και Βασικές Συναρτήσεις

Δομικά μέρη και πληροφορίες που απαρτίζουν την υλοποίηση ενός οδηγού λειτουργικού συστήματος Linux για την υποστήριξη βασικών λειτουργιών όπως αναγνώριση και εγκατάσταση συσκευής, εγκαθίδρυσης σύνδεσης με το λειτουργικό σύστημα, απόδοση εικονικής διεύθυνσης. Επίσης υποστήριξη εντολών απεγκατάστασης συσκευών όπως η εντολή απομάκρυνσης (remove)

Βασικός σχεδιασμός ένα οδηγού ελεύθερου λογισμικού (Linux Driver) για την διασύνδεση ηλεκτρονικού υπολογιστή (PC) με συσκευή υλικού μέσω του διαύλου PCI-Express.

Η PCI-Express Port αποτελείται από μια λογική δομή PCI-PCI Bridge. Υπάρχουν δύο είδη PCI-Express Port. Η πύλη πηγής, (Root Port) και η πύλη διακόπτη (Switch Port). Η Root Port προέρχεται από μια σύνδεση PCI-Express με ένα δίκτυο PCI-Express Port, και η Switch Port συνδέει δεσμούς PCI-Express με εσωτερικής λογικής διαύλους PCI. Η Switch Port, της οποίας ο δευτερεύον δίαυλος εκπροσωπεί την εσωτερική λογική δρομολόγησης του διακόπτη, ονομάζεται πύλη ροής προ του συστήματος του διακόπτη ή διαφορετικά Switchs' Upstream Port. Η Switch's Downstream Port ή αλλιώς πύλη ροής μετά συστήματος του διακόπτη γεφυρώνει τον δίαυλο εσωτερικής δρομολόγησης του διακόπτη με έναν δίαυλο που αντιπροσωπεύει την downstream PCI-express σύνδεση από τον διακόπτη PCI-Express.

Μια PCI-Express Port μπορεί να παρέχει έως τέσσερις διαφορετικές λειτουργίες, οι οποίες αναφέρονται ως υπηρεσίες, ανάλογα με το είδος της πύλης. Οι υπηρεσίες πύλης του PCI-Express συνοψίζονται σε: εγγενή υποστήριξη σύνδεσης (native hotplug support (HP)), υποστήριξη διαχείρισης συμβάντων ενέργειας (power management event support (PME)), προηγμένη αναφορά σφαλμάτων (advanced error reporting support (AER)) και υποστήριξη εικονικών καναλιών (virtual channel support (VC)).Αυτές οι υπηρεσίες μπορούν να υποστηριχθούν είτε από έναν σύνθετο οδηγό, είτε να διανεμηθούν και διεκπεραιωθούν ανεξάρτητα κάθε ένας από τον αντίστοιχο οδηγό υπηρεσίας.

Γιατί να χρησιμοποιήσουμε έναν οδηγό διαύλου πύλης PCI-Express?

Στους υπάρχοντες πυρήνες λειτουργικών συστημάτων ελευθέρου λογισμικού (Linux Kernels) το μοντέλο που χρησιμοποιείται για οδηγούς συσκευών ελευθέρου λογισμικού (Linux Drivers) επιτρέπει σε κάθε φυσική συσκευή να οδηγείται από έναν μόνο οδηγό. Η PCI-Express Port είναι μια συσκευή PCI-PCI Bridge με πολλαπλές ανεξάρτητες υπηρεσίες (Services). Για να διατηρηθεί η απλοϊκότητα και η εύκολη πρόσβαση, κάθε υπηρεσία δύναται να έχει ξεχωριστό ανεξάρτητο οδηγό λογισμικού. Σε αυτή την περίπτωση, πολλοί οδηγοί υπηρεσιών θα ανταγωνίζονται μεταξύ τους για μια συσκευή PCI-PCI Bridge.

Για παράδειγμα, αν ο οδηγός υποστήριξης της υπηρεσίας προηγμένης αναφοράς σφαλμάτων (Advanced Error Reporting) έχει φορτωθεί πρώτος, διεκδικεί την Root Port της PCI-PCI Bridge. Ο πυρήνας τότε δεν μπορεί να φορτώσει άλλο οδηγό που να υποστηρίζει κάποια άλλη υπηρεσία για την συγκεκριμένη Root Port. Με άλλα λόγια είναι αδύνατο να έχουμε πολλαπλούς οδηγούς υπηρεσιών φορτωμένους και εκτελέσιμους για μια συσκευή PCI-PCI Bridge ταυτόχρονα χρησιμοποιώντας το υπάρχον μοντέλο οδηγών.
Για να έχουμε την δυνατότητα να εκτελούμε πολλαπλούς οδηγούς υπηρεσιών ταυτόχρονα οφείλουμε να έχουμε έναν PCI-Express Port Bus Driver ο οποίος να διαχειρίζεται όλες τις πύλες PCI-Express οι οποίες έχουν καταληφθεί και διανέμει όλες τις αιτήσεις των υποστηριζόμενων υπηρεσιών στους αντίστοιχους οδηγούς υπηρεσιών, όπως απαιτείται. Τα πλεονεκτήματα χρήσης του PCI-Express Port Bus Driver αναφέρονται παρακάτω:

- Επιτρέπει σε πολλαπλούς οδηγούς υπηρεσιών να εκτελούνται ταυτόχρονα σε μια συσκευή πύλης PCI-PCI Bridge.
- Επιτρέπει οδηγούς υπηρεσιών οι οποίοι έχουν υλοποιηθεί με ανεξάρτητη προσέγγιση μεταξύ τους.
- Επιτρέπει σε έναν οδηγό υπηρεσίας να εκτελείται σε πολλαπλές συσκευές πύλης PCI-PCI Bridge.
- Διαχείριση και κατανομή των πόρων μιας συσκευής πύλης PCI-PCI Bridge στους αιτούντες οδηγούς υπηρεσιών.

Configuring the PCI Express Port Bus Driver vs. Service Drivers

Ένταξη της υποστήριξης του οδηγού PCI-Express Port Bus στον πυρήνα λειτουργικού συστήματος Linux.

Η ένταξη του οδηγού PCI-Express Port Bus εξαρτάται από το κατά πόσο η υποστήριξη του PCI-Express συμπεριλαμβάνεται στην ρύθμιση του πυρήνα του λειτουργικού συστήματος. Ο πυρήνας αυτόματα θα συμπεριλάβει τον οδηγό PCI-Express Port Bus σαν έναν οδηγό πυρήνα (Kernel Driver) όταν η υποστήριξη του PCI-Express έχει επιλεχθεί στον πυρήνα.

Ενεργοποίηση της υποστήριξης οδηγών υπηρεσιών (Enabling Service Driver Support)

Οι οδηγοί συσκευών PCI έχουν υλοποιηθεί βασιζόμενοι στο μοντέλο οδηγών συσκευών ελεύθερου λειτουργικού συστήματος Linux. Όλοι οι οδηγοί υπηρεσιών είναι οδηγοί συσκευών PCI. Όπως αναφέρθηκε παραπάνω, είναι αδύνατον να φορτωθεί κάποιος οδηγός υπηρεσίας όταν ο πυρήνας έχει φορτώσει τον οδηγό PCI-Express Port Bus. Για την εκπλήρωση των κριτηρίων του μοντέλου οδηγού PCI-Express Port Bus χρειάζονται μικρό-αλλαγές στους υφιστάμενους οδηγούς υπηρεσιών, οι οποίες όμως αλλαγές δεν έχουν καμία επίπτωση στην λειτουργικότητα των συγκεκριμένων οδηγών υπηρεσιών.

Ένας οδηγός υπηρεσίας υποχρεούνται να χρησιμοποιεί τις δύο συναρτήσεις που παρουσιάζονται παρακάτω έτσι ώστε να καταχωρείται η υπηρεσία με τον οδηγό PCI-Express Port Bus. Είναι σημαντικό ένας οδηγός υπηρεσίας να αρχικοποιεί την δομή δεδομένων *pcie_port_service_driver* η οποία συμπεριλαμβάνεται στο αρχείο επικεφαλίδων */include/linux/pcieport.if.h* πριν την κλήση των δύο συναρτήσεων αυτών. Αν δεν πραγματοποιηθεί αυτή η αρχικοποίηση, τότε θα προκληθεί αν-αντιστοιχία ταυτότητας, η οποία εμποδίζει τον οδηγό PCI-Express Port Bus να φορτώσει έναν οδηγό υπηρεσίας.

- pcie_port_service_register

*int pcie_port_driver_register (struct pcie_port_service_driver *new)*

Αυτή η συνάρτηση αντικαθιστά την συνάρτηση pci_module_init του μοντέλου οδηγών του λειτουργικού συστήματος ελευθέρου λογισμικού Linux. Ένας οδηγός υπηρεσίας πρέπει πάντα να καλεί την συνάρτηση pcie_port_service_register κατά την αρχικοποίηση μιας αυτοτελούς μονάδας (module). Σημειώνουμε ότι αφού φορτωθεί ο οδηγός υπηρεσίας τότε κλήσεις συναρτήσεων όπως pci_enable_device(dev) και pci_set_master(dev) δεν είναι πλέον απαραίτητη δεδομένου ότι αυτές οι κλήσεις εκτελούνται από τον οδηγό PCI Port Bus.

- pcie_port_service_unregister

*void pcie_port_service_unregister(struct pcie_port_service_driver *new)*

Η συνάρτηση pcie_port_service_unregister αντικαθιστά την συνάρτηση pci_unregister_driver του μοντέλου οδηγών του λειτουργικού συστήματος ελευθέρου λογισμικού (Linux). Καλείται πάντα από τον οδηγό υπηρεσίας όταν μια αυτοτελής μονάδα (module) εξέρχεται.

Δείγμα κώδικα οδηγού υπηρεσίας ο οποίος αρχικοποιεί την δομή δεδομένων της πύλης του οδηγού υπηρεσίας.

(Ο παρακάτω κώδικας υπάρχει στα έγγραφα κάθε πυρήνα λειτουργικού συστήματος ελευθέρου λογισμικού στη διεύθυνση /usr/src/linux/Documentation/PCI.)

```
static struct pcie_port_service_id service_id[] = { {
        .vendor = PCI_ANY_ID,
        .device = PCI_ANY_ID,
        .port_type = PCIE_RC_PORT,
        .service_type = PCIE_PORT_SERVICE_AER,
        }, { /* end: all zeroes */ }
};

static struct pcie_port_service_driver root_aerdrv = {
        .name           = (char *)device_name,
        .id_table       = &service_id[0],

        .probe          = aerdrv_load,
        .remove         = aerdrv_unload,

        .suspend        = aerdrv_suspend,
        .resume         = aerdrv_resume,
};
```

Δείγμα κώδικα οδηγού υπηρεσίας ο οποίος καταχωρεί/απελευθερώνει έναν οδηγό υπηρεσίας

```
static int __init aerdrv_service_init(void)
{
        int retval = 0;

        retval = pcie_port_service_register(&root_aerdrv);
if (!retval) {
        /*
         * FIX ME
         */
}
return retval;
}

static void __exit aerdrv_service_exit(void)
{
        pcie_port_service_unregister(&root_aerdrv);
}

module_init(aerdrv_service_init);
module_exit(aerdrv_service_exit);
```

Possible Resource Conflicts

Καθώς όλοι οι οδηγοί υπηρεσιών για μια συσκευή PCI-PCI Port Bridge επιτρέπεται να εκτελούνται ταυτόχρονα, παρακάτω παρατίθενται μερικές από τις πιθανές συγκρούσεις πόρων που ενδέχεται να προκληθούν καθώς και προτεινόμενες λύσεις που επιλύουν τα προβλήματα αυτά.

- **MSI Vector Resource**

Η δομή MSI Capability Structure επιτρέπει στον οδηγό λογισμικού μιας συσκευής να καλέσει την συνάρτηση pci_enable_msi και να ζητήσει Interrupts βασιζόμενους στο MSI. Μόλις τα Interrupts MSI ενεργοποιηθούν στη συσκευή, η συσκευή παραμένει σε αυτή την κατάσταση έως την στιγμή όπου ένας οδηγός καλέσει την συνάρτηση pci_disable_msi για να απενεργοποιήσει τα Interrupts MSI και να επιστρέψει στην INTx κατάσταση εξομοίωσης. Δεδομένου ότι οδηγοί υπηρεσιών της ίδιας πύλης PCI-PCI Bridge μοιράζονται την ίδια φυσική συσκευή, τότε εάν ένας ξεχωριστός οδηγός υπηρεσίας καλέσει τις συναρτήσεις pci_enable_msi/pci_disable_msi ενδέχεται να προκαλέσει απροσδιόριστη συμπεριφορά. Για παράδειγμα, δυο οδηγοί υπηρεσιών εκτελούνται στην ίδια φυσική Root Port. Και οι δύο οδηγοί υπηρεσιών καλούν την συνάρτηση pci_enable_msi ζητώντας Interrupts βασισμένους στο MSI. Ένας οδηγός υπηρεσίας δύναται να μην γνωρίζει αν υπάρχει άλλος οδηγός να εκτελείται στην ίδια Root Port. Εάν κάποιος από αυτούς καλέσει την συνάρτηση pci_disable_msi, τότε τοποθετεί τον άλλο οδηγό υπηρεσίας σε λάθος λειτουργία interrupt.

Για την αποφυγή αυτής της κατάστασης, σε όλους τους οδηγούς υπηρεσιών, απαγορεύεται η επιλογή Interrupt mode για την συσκευή τους. Ο οδηγός PCI-Express Port Bus είναι υπεύθυνος να καθορίζει την Interrupt Mode, και αυτό να είναι ξεκάθαρο για όλους τους οδηγούς υπηρεσιών. Οι οδηγοί υπηρεσιών χρειάζεται να γνωρίζουν μόνο την τιμή IRQ η οποία αντιστοιχίζεται με το πεδίο IRQ της δομής pcie_device, η οποία εναποτίθεται όταν ο οδηγός PCI-Express Port Bus ερευνά κάθε οδηγό υπηρεσίας. Οι οδηγοί υπηρεσιών οφείλουν να χρησιμοποιούν την δομή:

(struct pcie_device)dev->irq*

για να εκτελέσουν *request_irq/free_irq.* Επιπλέον η Interrupt Mode είναι αποθηκευμένη στον τομέα *interrupt_mode* της δομής *pcie_device.*

- **MSI-X Vector Resource**

Παρόμοια με το MSI ένας οδηγός για μια συσκευή που υποστηρίζει MSI-X δύναται να καλέσει την συνάρτηση pcie_enable_msix ζητώντας διακόπτες MSI-X. Σε όλους τους οδηγούς υπηρεσιών δεν επιτρέπεται η επιλογή Interrupt Mode για την συσκευή τους. Ο οδηγός του PCI-Express Port Bus είναι υπεύθυνος να καθορίζει την κατάσταση Interrupt, και αυτό να είναι ξεκάθαρο για όλους τους οδηγούς υπηρεσιών. Κάθε απόπειρα από κάποιον οδηγό υπηρεσίας να καλέσει την συνάρτηση pci_enable_msix/pci_disable_msix μπορεί να καταλήξει σε απροσδιόριστης συμπεριφοράς αποτελέσματα. Οι οδηγοί υπηρεσιών οφείλουν να χρησιμοποιούν την δομή:

(struct pcie_device)dev->irq*

για να εκτελέσουν εντολές *request_irq/free_irq.*

- **PCI Memory/IO Mapped Regions**

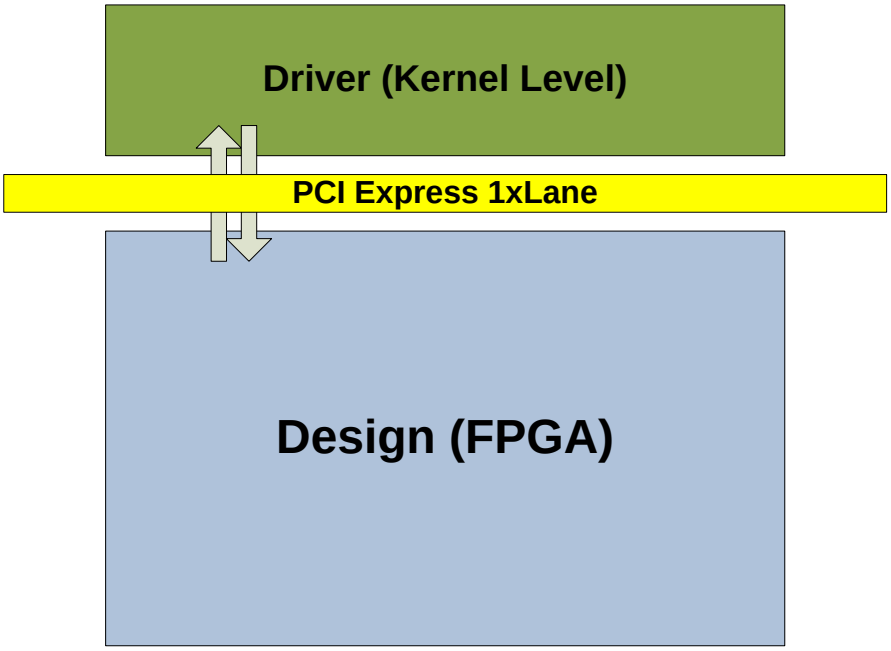
Οδηγοί υπηρεσιών για υποστήριξη διαχείρισης συμβάντων ενέργειας (power management event support (PME)), προηγμένη αναφορά σφαλμάτων (advanced error reporting support (AER)), εγγενή υποστήριξη σύνδεσης (native hotplug support (HP)), και υποστήριξη εικονικών καναλιών (virtual channel support (VC)) προσπελάσουν τον χώρο διαμόρφωσης της πύλης PCI-Express. Σε όλες τις περιπτώσεις, οι καταχωρητές (Registers) που προσπελαύνονται είναι ανεξάρτητοι μεταξύ τους. Αυτή η ενημέρωση προϋποθέτει ότι όλοι οι οδηγοί υπηρεσιών συμπεριφέρνονται ορθά και δεν προκαλούν αλλοίωση στις ρυθμίσεις των άλλων οδηγών υπηρεσιών.

- **PCI Config Registers**

Κάθε οδηγός υπηρεσίας εκτελεί τις λειτουργίες διαμόρφωσης του PCI σύμφωνα με τις δικές του δομικές ικανότητες, εκτός από την δομική ικανότητα του PCI-Express στην οποία ο καταχωρητής ελέγχου πηγής (Root Control register) και ο καταχωρητής ελέγχου συσκευής (Device Control register) είναι κοινός για τις υπηρεσίες PME και AER. Αυτή η ενημέρωση προϋποθέτει ότι όλοι οι οδηγοί υπηρεσιών συμπεριφέρνονται ορθά και δεν προκαλούν αλλοίωση στις ρυθμίσεις των άλλων οδηγών υπηρεσιών.

Υλοποίηση της σύνδεσης Virtex-5 με Η/Υ (Linux Driver)

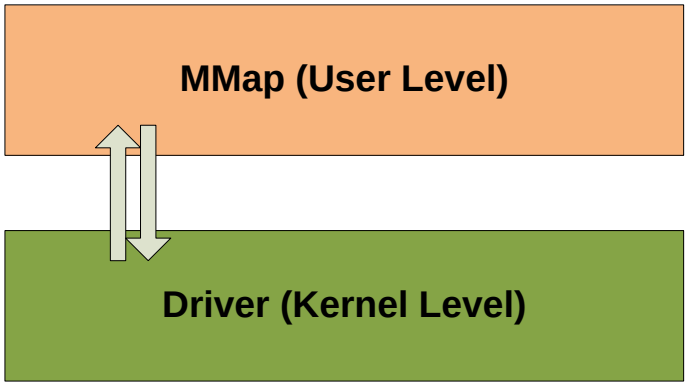
Για την σύνδεση του Ηλεκτρονικού Υπολογιστή με τον αναδιατασσόμενη συσκευή Virtex-5 της εταιρίας Xilinx συντάξαμε ένα οδηγό λειτουργικού συστήματος ελευθέρου λογισμικού Linux. Ο οδηγός αυτός ικανοποιεί τα πρότυπα οδηγών που αναφέρονται παραπάνω για την υποστήριξη της επικοινωνίας συσκευής με τον Ηλεκτρονικό Υπολογιστή μέσω του διαύλου PCI Express.



Να σημειώσουμε ότι ο Linux Driver δεν εκτελεί ο ίδιος τις λειτουργίες μεταφοράς και λήψης δεδομένων. Οι λειτουργίες που υποστηρίζει είναι η εγκατάσταση της συσκευής στον πυρήνα (register the device) και η απομάκρυνση της συσκευής (unregister the device) από τον πυρήνα του λειτουργικού συστήματος ελευθέρου λογισμικού. Υποστηρίζει το λεγόμενο Probe της συσκευής, δηλαδή κάνει «ορατή» την συσκευή στο λειτουργικό σύστημα. Οι βασικές συναρτήσεις και δομές που αναλαμβάνουν την σύνδεση παρουσιάστηκαν παραπάνω. Επίσης υλοποιεί την μεθοδολογία για μεταφορά δεδομένων, δηλαδή υποστηρίζει το memory map μεταξύ της συσκευής και του λειτουργικού, αλλά ωστόσο χρειάζεται ξεχωριστή εφαρμογή η οποία θα πραγματοποιήσει την σύνδεση από το επίπεδο χρήστη στο επίπεδο πυρήνα.

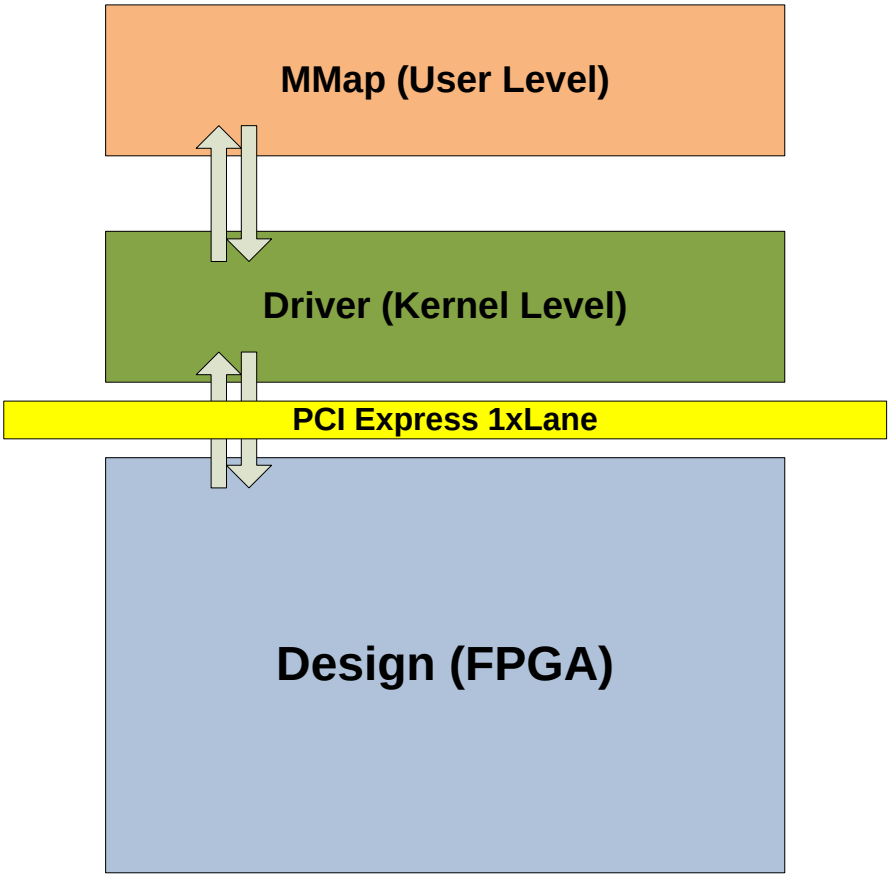
Υλοποίηση της υποστήριξης λειτουργίας I/O (MMap)

Ο οδηγός λειτουργικού συστήματος ελευθέρου λογισμικού (Linux Driver) που συντάξαμε αναλαμβάνει να συνδέσει την αναδιατασσόμενη συσκευή που χρησιμοποιούμε (Virtex-5) με τον Ηλεκτρονικό Υπολογιστή σε επίπεδο πυρήνα (Kernel Level) του λειτουργικού συστήματος Linux.



Αφού λοιπόν, πραγματοποιηθεί η σύνδεση της συσκευής με το λειτουργικό σύστημα του υπολογιστή, χρειαζόμαστε μια εφαρμογή, η οποία θα επικοινωνεί με τον παραπάνω Linux Driver και θα αποκτά πρόσβαση σε λειτουργίες μεταφοράς και λήψεις δεδομένων από την FPGA.

Ουσιαστικά η συνολική σχεδίαση που υποστηρίζει την επικοινωνία της FPGA με τον χρήστη Ηλεκτρονικού Υπολογιστή μέσω Linux Driver έχει την παρακάτω μορφή:



Η υλοποίηση της εφαρμογής που αναλαμβάνει την υποστήριξη μεταφοράς δεδομένων είναι απλή σε λειτουργία. Ουσιαστικά το κλειδί της λειτουργίας της εφαρμογής αυτής είναι η συνάρτηση mmap της γλώσσας προγραμματισμού C.

void * **mmap** (*void *address, size_t length,int protect, int flags, int filedes, off_t offset*)

Η συνάρτηση mmap() εγκαθιδρύει μια σύνδεση μεταξύ του χώρου διευθύνσεων μιας διαδικασίας και την οδηγεί σε μια εικονική διεύθυνση. Ουσιαστικά κάνει την σύνδεση του Kernel Driver με την εφαρμογή και επιστρέφει έναν δείκτη σε μια εικονική διεύθυνση στην οποία γίνεται προσπέλαση για ανάγνωση και εγγραφή. Έπειτα, τον δείκτη στην διεύθυνση που επιστρέφει η συνάρτηση αυτή, τον διαχειριζόμαστε όπως αρμόζει στις δικές μας ανάγκες. Ορίσματα της συνάρτησης mmap καθορίζουν το πλήθος των δυνατών διευθύνσεων. Για παράδειγμα, στην δική μας υλοποίηση έχουμε κάνει memory map για μια δυνατή διευθυνσιοδότηση 4096 διευθύνσεων. Από αυτές τις διευθύνσεις που υποστηρίζει ο Linux Driver μας, εμείς θα χρησιμοποιήσουμε μόνο τις τρεις (0x01C,0x024,0x054).

Για να είμαστε σίγουροι ότι τα δεδομένα που μεταφέρονται από τον Ηλεκτρονικό Υπολογιστή στην FPGA (και αντίστροφα) μέσω του διαύλου PCI Express είναι έγκυρα, υλοποιήσαμε την μεθοδολογία Polling.

Η εφαρμογή της μεθοδολογίας Polling δεν πραγματοποιήθηκε μέσα στον Linux Driver που δημιουργήσαμε, αλλά στο λογισμικό διασύνδεσης του χρήστη με τον Driver σε επίπεδο χρήστη (user level). Δηλαδή ενσωματώθηκε στο λογισμικό το οποίο πραγματοποιεί το memory map και υποστηρίζει τις εντολές αποστολής και λήψης.

Η μεθοδολογία Polling

Polling στην Επιστήμη Υπολογιστών

Polling (Δημοσκόπηση) στην επιστήμη υπολογιστών αναφέρεται ως η μέθοδος όπου ελέγχεται δειγματοληπτικά η κατάσταση μιας εξωτερικής συσκευής από ένα πρόγραμμα πελάτη (client-program) ως σύγχρονη δραστηριότητα. Η μέθοδος Polling συνήθως συναντάται από άποψη λειτουργιών Εισόδου/Εξόδου (I/O) καθώς επίσης αναφέρεται και ως Polled I/O ή λογισμικά οδηγημένη I/O (software driven I/O).

Η μέθοδος Polling συχνά συναντάται να χρησιμοποιείται ταυτόσημα με την μέθοδο busy-wait Polling (busy-waiting). Σε αυτή την περίπτωση όταν μια λειτουργία Εισόδου/Εξόδου χρειάζεται να πραγματοποιηθεί ο Υπολογιστής ελέγχει την κατάσταση της συσκευής Εισόδου/Εξόδου έως ότου αυτή είναι έτοιμη. Μόλις η συσκευή είναι έτοιμη, γίνεται προσπέλαση και πραγματοποιείται η επιθυμητή λειτουργία. Με άλλα λόγια ο Υπολογιστής περιμένει την συσκευή να προετοιμαστεί.

Η μέθοδος Polling επίσης αναφέρεται στην περίπτωση όπου μια συσκευή ελέγχεται επαναλαμβανόμενα για την ετοιμότητα της. Στην περίπτωση που δεν είναι σε ετοιμότητα τότε ο υπολογιστής πραγματοποιεί μια άλλη λειτουργία. Αν και όχι τόσο σπάταλη σε κύκλους ρολογιού της CPU όσο η μεθοδολογία του busy-wait που αναφέρθηκε παραπάνω, αυτή η τεχνολογία δεν είναι τόσο αποτελεσματική όσο η εναλλακτική λύση στο Polling, η χρησιμοποίηση Interrupt I/O.

Σε ένα απλό σύστημα το οποίο εκτελεί μια συγκεκριμένη λειτουργία, ακόμα και η μεθοδολογία του busy-wait είναι απολύτως αρμοστή εάν καμία άλλη λειτουργία δεν πραγματοποιείται πριν την εκτέλεση της Εισόδου / Εξόδου δεδομένων. Όμως τις περισσότερες φορές αυτή είναι συνέπεια μιας δομής απλού υλικού (simple Hardware) ή δομής μη πολλαπλών λειτουργικών συστημάτων (non-multitasking Operating systems).

Συχνά η μεθοδολογία Polling είναι στενά συνδεμένη με hardware χαμηλού επιπέδου. Για παράδειγμα, θα χρησιμοποιήσουμε Polling για να ελέγξουμε την παράλληλη θύρα ενός εκτυπωτή για να δούμε εάν είναι έτοιμη για να δεχθεί ένα ακόμα χαρακτήρα τόσο μικρό όσο το Bit ενός Byte. Αυτό το Bit αναπαριστά, κατά την διαδικασία ανάγνωσης, αν ένα μονό καλώδιο στην διεπαφή του Ηλεκτρονικού Υπολογιστή με τον εκτυπωτή είναι σε Υψηλή ή Χαμηλή τάση. Η I/O εντολή η οποία διαβάζει αυτό το Byte, αμέσως μεταφέρει την κατάσταση της τάση από οκτώ καλώδια σε οκτώ κυκλώματα (flip flops) που σχηματίζουν ένα byte του μητρώου της CPU.

Διαδικασία Ανάγνωσης (από την πλευρά του Software)

Για να γίνει ανάγνωση χρησιμοποιούμε την μέθοδο του Δημοψηφίσματος (Polling). Διαβάζουμε πρώτα μια συγκεκριμένη διεύθυνση της οποίας η τιμή μας ενημερώνει για την κατάσταση στην FPGA, δηλαδή αν υπάρχουν έγκυρα δεδομένα στην μνήμη της FPGA ή αν είναι άδεια. Αν υπάρχουν έγκυρα δεδομένα, τότε μπορούμε διαβάζοντας την διεύθυνση ανάγνωσης να λάβουμε μια στοιχειώδη ποσότητα (32Bit) έγκυρων δεδομένων. Έπειτα θα διαβάσουμε εκ νέου την διεύθυνση ενημέρωσης κατάστασης για έγκυρα δεδομένα για να σιγουρευτούμε ότι υπάρχουν ακόμα ωφέλιμα δεδομένα στην μνήμη. Αυτή η εναλλάξ λογική μπορεί να παρουσιάζεται χρονοβόρα καθώς απαιτεί δύο αναγνώσεις μνήμης, για την πιθανή λήψη μιας στοιχειώδης ποσότητας των 32 bit συν την επιπλέον λογική σε επίπεδο software που χρησιμοποιείται, αλλά είναι απαραίτητη για να είμαστε σίγουροι ότι τα δεδομένα που λαμβάνει ο χρήστης είναι έγκυρα.

Παράδειγμα κώδικα σε C:

```
for (i=0;i<Length;i++);
{
        if ( *(p+Valid_Address)=0 )
                Data = *(p+Read_Address)
}
```

Ο παραπάνω κώδικας, παρουσιάζει την βασική αρχή του τρόπου λειτουργίας Polling από την πλευρά του λογισμικού. Αρχικά γίνεται ανάγνωση στην διεύθυνση Valid_Address. Όταν γίνει αίτηση για ανάγνωση σε αυτή την διεύθυνση, αυτό που θα επιστρέψει η FPGA είναι 0 ή 1. Η τιμή που επιστρέφεται ελέγχεται και αν είναι 0, σημαίνει ότι υπάρχουν έγκυρα δεδομένα στην FPGA και δεν είναι άδεια ή τα δεδομένα που υπάρχουν στην μνήμη δεν είναι «σκουπίδια». Αν είναι 1, σημαίνει ότι δεν υπάρχουν δεδομένα, και οπότε δεν πραγματοποιείται λήψη δεδομένων από την FPGA.

Στην περίπτωση που υπάρχουν έγκυρα δεδομένα, τότε γίνεται η ανάγνωση της Read_Address και μεταφέρεται μια ποσότητα 32Bit, από την FPGA στο λογισμικό χρήστη μέσω του διαύλου PCI-Express. Να σημειώσουμε ότι η Valid_Address και η Read_Address έχουν συγκεκριμένες τιμές. Για οποιαδήποτε άλλη διεύθυνση, η FPGA αγνοεί την αίτηση ανάγνωσης σαν να μην πραγματοποιήθηκε ποτέ.

Τιμές Διευθύνσεων:

Valid Address: 0x01C
Read Address: 0x054

Διαδικασία Εγγραφής (από την πλευρά του Software)

Ο χρήστης στην εκτέλεση του λογισμικού για εγγραφή στην FPGA, μπορεί να γράψει μόνο σε μια συγκεκριμένη ουσιαστικά διεύθυνση της μνήμης. Στην πραγματικότητα η μνήμη έχει αντικατασταθεί από έναν καταχωρητή ο οποίος είναι ενεργός για εγγραφή δεδομένων που αποστέλλονται μόνο όταν η διεύθυνση εγγραφής είναι μια συγκεκριμένη (write address).

Παράδειγμα κώδικα σε C:

```
int i=0; Data=0;
for (i=0;i<Length;i++);
{
        *(p+Write_Address)=Data;
        Data++;
}
```

Ο παραπάνω κώδικας μας δείχνει το μπλοκ κώδικα λογισμικού το οποίο θα μπορούσε να χρησιμοποιηθεί για την αποστολή δεδομένων στην FPGA.. Η Write Address είναι μια συγκεκριμένη διεύθυνση. Τα δεδομένα Data είναι μεγέθους 32Bit. Οπότε η FPGA θα λάβει Length αριθμό από 32Bit ποσότητες Data. Να σημειωθεί εδώ, ότι τα δεδομένα Data θα εγγραφούν στην FPGA μόνο όταν η διεύθυνση Write Address είναι η επιλεχθείσα ενώ για διεύθυνση διαφορετική απορρίπτονται. Τα δεδομένα Data μπορεί να προέρχονται είτε από ανάγνωση αρχείου, είτε να είναι ένα απλό count ή οποιοδήποτε άλλο μέσω το οποίο μπορεί να μας τροφοδοτήσει δεδομένα για αποστολή. Στο παραπάνω παράδειγμα είναι ένα Count από το μηδέν έως το Length.

Τιμή Διεύθυνσης:

Write Address: 0x024

6. Υλοποίηση Σχεδίασης

Υλοποίηση Σχεδίασης

Παραγωγή PCI-Express Core μέσω Core Generator

Με τα εργαλεία της Xilinx, και συγκεκριμένα με το εργαλείο Core Generator, δημιουργήσαμε τον πυρήνα του διαύλου PCI-Express, τον οποίο χρησιμοποιήσαμε στην σχεδίαση μας. Επιλέξαμε να δουλέψουμε με την γλώσσα VHDL έναντι της γλώσσας Verilog καθώς έχει αποκτηθεί εμπειρία στην VHDL μέσα από μαθήματα όπως της Οργάνωσης Υπολογιστών και της Αρχιτεκτονικής Υπολογιστών.

Επιστρέφοντας στις παραμέτρους που χρησιμοποιήσαμε για την δημιουργία του πυρήνα PCI-Express μέσω του εργαλείου Core Generator, αρχικά πρέπει να δηλώσουμε την συσκευή για την οποία προορίζεται ο πηγαίος κώδικας που θα παραχθεί. Έτσι, στην αρχική καρτέλα του Project επιλέγουμε:

Family: Virtex-5
Device: xc5vlx110t
Package: ff1136
Speed Grade : -1

Η πλακέτα που χρησιμοποιούμε για να επαληθεύσουμε την λειτουργία του Driver πρόκειται για ένα board XUPV5-LX110T της εταιρίας Xilinx. Η σειρά XUP (Xilinx University Program) της εταιρίας Xilinx προορίζεται συγκεκριμένα για πανεπιστημιακή χρήση. Το XUP-V5 που χρησιμοποιούμε πρόκειται για ένα board πανομοιότυπο με τα board της εταιρίας ML505-506-507 μόνο που στο συγκεκριμένο έχει τοποθετηθεί συσκευή αναδιατασσόμενης λογικής xc5vlx110t.

Έπειτα, αφού επιλέξουμε την δημιουργία **Standard Bus Interfaces** -> **PCI Express** -> **Endpoint Block Plus for PCI Express,** θέτουμε τις ακόλουθες τιμές στις μεταβλητές όπου χρειάζεται.

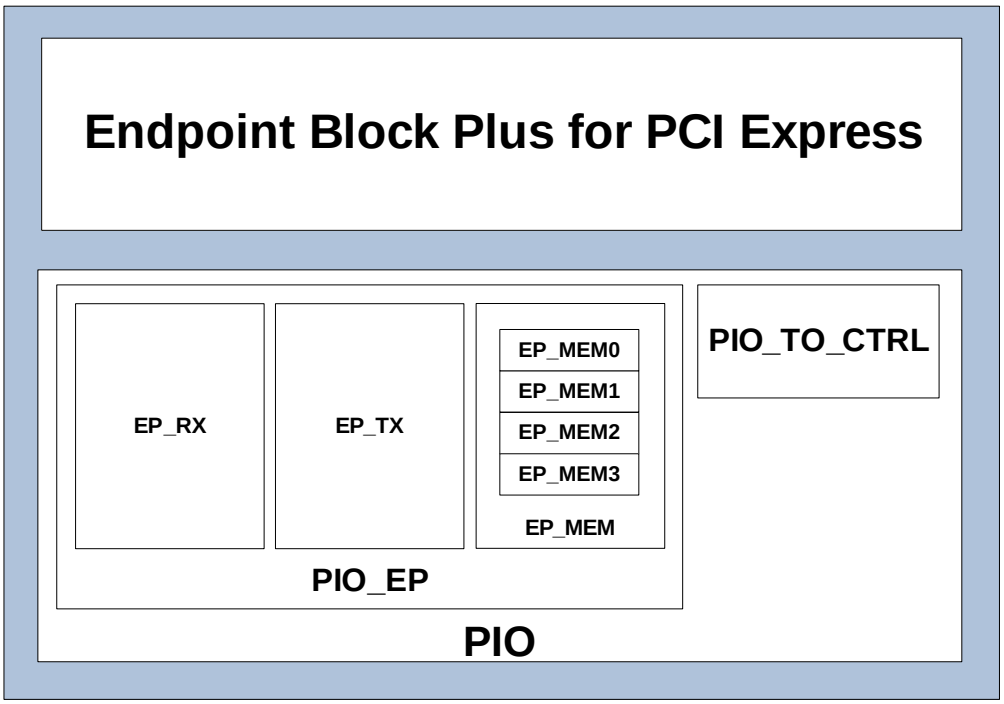
Reference Clock Frequency: 100MHz
Number Of Lanes: x1
Interface Frequency: 62.50MHz
Vector ID: 10EE
Device ID: 5050

Τα παραπάνω σημαίνουν ότι, ο πυρήνας PCI Express έχει σαν έξοδο ένα ρολόι 100MHz το οποίο μπορεί να χρησιμοποιηθεί σε οποιοδήποτε άλλο μέρος της σχεδίασης, ότι η υλοποίηση του διαύλου PCI Express είναι υλοποίηση μιας full duplex γραμμής (1xLane), και ότι η μέγιστη συχνότητα μεταφοράς των πακέτων μέσω του διαύλου είναι στα 62.50MHz. Τέλος οι τιμές Vector ID, και Device ID χρησιμοποιούνται στην αναγνώριση της πλακέτας από το λειτουργικό σύστημα.

PCI Express Core

Ο πυρήνας του PCI-Express που δημιουργήσαμε, υποστηρίζει τις λειτουργίες της ανάγνωσης και εγγραφής από/σε μια ενσωματωμένη μνήμη RAM στον πυρήνα της FPGA. Η εταιρία Xilinx παρέχει την σχεδίαση αυτήν σε αυτή την μορφή έτσι ώστε ο μηχανικός να έχει μια βάση την οποία έχει την δυνατότητα να κατανοήσει, και να εξελίξει στην μορφή που του χρειάζεται για την σχεδίαση του.

Ένα σχηματικό παράδειγμα του πυρήνα που παράγεται με τα εργαλείο της Xilinx είναι το παρακάτω.



Παρατηρούμε ότι στο top level της σχεδίασης βρίσκεται το Endpoint Block Plus για το PCI Express και το module PIO.

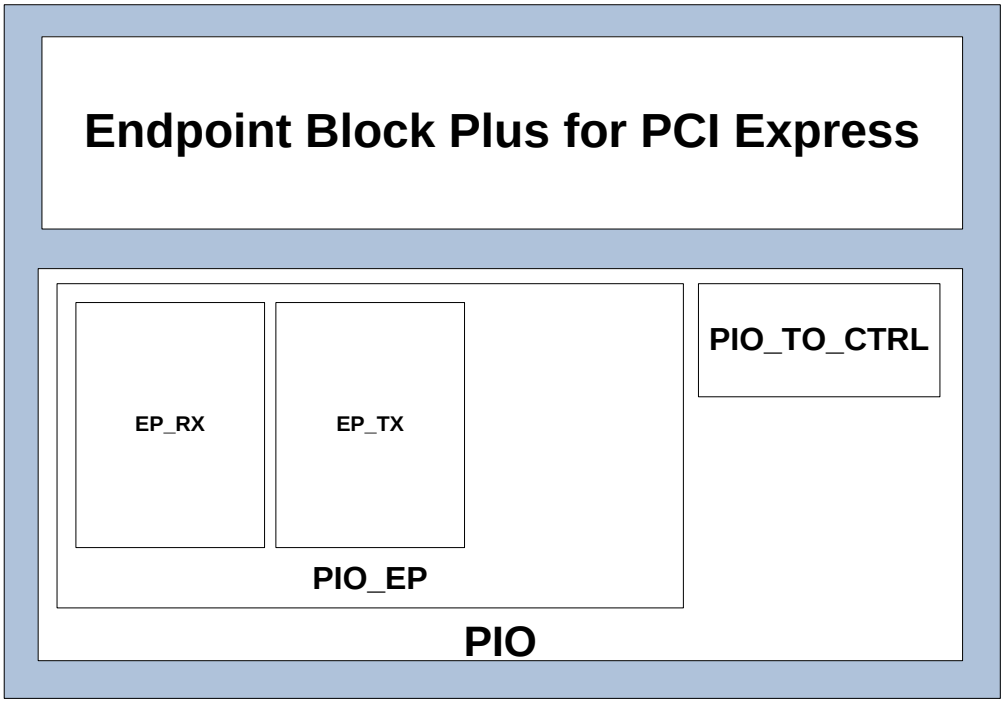
Το Endpoint Block Plus για το PCI-Express αποτελεί το module που πραγματοποιεί την διασύνδεση μέσω του διαύλου PCI-Express. Σε αυτό το module καταλήγουν τα δεδομένα για μεταφορά καθώς και όλα τα σήματα ελέγχου που είναι απαραίτητα για την πραγματοποίηση της μεταφοράς.

Το PIO (Programmed Input Output) αποτελεί ένα σημείο αρχής που προσφέρει η Xilinx για τους μηχανικούς που ενδιαφέρονται να χρησιμοποιήσουν τον δίαυλο PCI-Express. Στο PIO, γίνεται η αποθήκευση των δεδομένων που λαμβάνονται, και πραγματοποιείται η προετοιμασία για την αποστολή πακέτων. Τα εσωτερικά modules EP_TX και EP_RX αναλαμβάνουν την προετοιμασία των πακέτων σε μορφή τέτοια ώστε να τα δεχθεί το Endpoint Block και να πραγματοποιηθεί η μεταφορά. Επίσης παρατηρούμε ότι μέσα στο PIO βρίσκονται και οι μνήμες RAM που χρησιμοποιούνται για την αποθήκευση και ανάγνωση των δεδομένων που μεταφέρονται.

Στην σχεδίαση μας, επεξεργαστήκαμε τα περιεχόμενα του module PIO, αφαιρέσαμε ότι δεν ήταν λειτουργικό για εμάς, και προσθέσαμε extra λογική όπου ήταν απαραίτητη για να πετύχουμε το ζητούμενο αποτέλεσμα.

Αφού εξετάσαμε, εκτελέσαμε και κατανοήσαμε πλήρως την λειτουργία της σχεδίασης που μας προσφέρεται, προβήκαμε στις ακόλουθες ενέργειες:

- Αρχικά καταργήσαμε την χρήση ενσωματωμένης εσωτερικής μνήμης RAM μαζί με την λογική που έλεγχε τις διαδικασίες εγγραφής και ανάγνωσης. Οπότε το κομμάτι του κώδικα που χρησιμοποιήσαμε αυτούσιο στη σχεδίαση μας έχει την παρακάτω μορφή.



Πρόκειται δηλαδή για μια σχεδίαση η οποία υλοποιεί την διεπαφή της FPGA μέσω του διαύλου PCI-Express καθώς και τα στοιχεία εκείνα της σχεδίασης τα οποία θα προετοιμάσουν τα πακέτα προς αποστολή, και θα παραλάβουν τα δεδομένα από τα πακέτα που λαμβάνονται.

- Στο σημείο αυτό ενώσαμε την δική μας υλοποίηση η οποία παίρνει τα δεδομένα ανάγνωσης/εγγραφής μαζί με τα σήματα που ελέγχουν τις λειτουργίες αυτές (Write Enable, Read Enable, Address) και τα προωθεί σε έναν καταχωρητή για κάθε λειτουργία. Δηλαδή υπάρχει καταχωρητής στον οποίο αποθηκεύεται η ποσότητα προς εγγραφή (ενώ στην ανάλογη περίπτωση η αποθήκευση των δεδομένων γινόταν στην εσωτερική μνήμη EP_MEM από τον RX Transceiver), και επίσης υπάρχει καταχωρητής ο οποίος τροφοδοτεί τον TX Transceiver με δεδομένα για να πραγματοποιηθεί η λειτουργία της ανάγνωσης (ο καταχωρητής αυτός υπάρχει στην θέση της ίδιας μνήμης EP_MEM με παραπάνω)

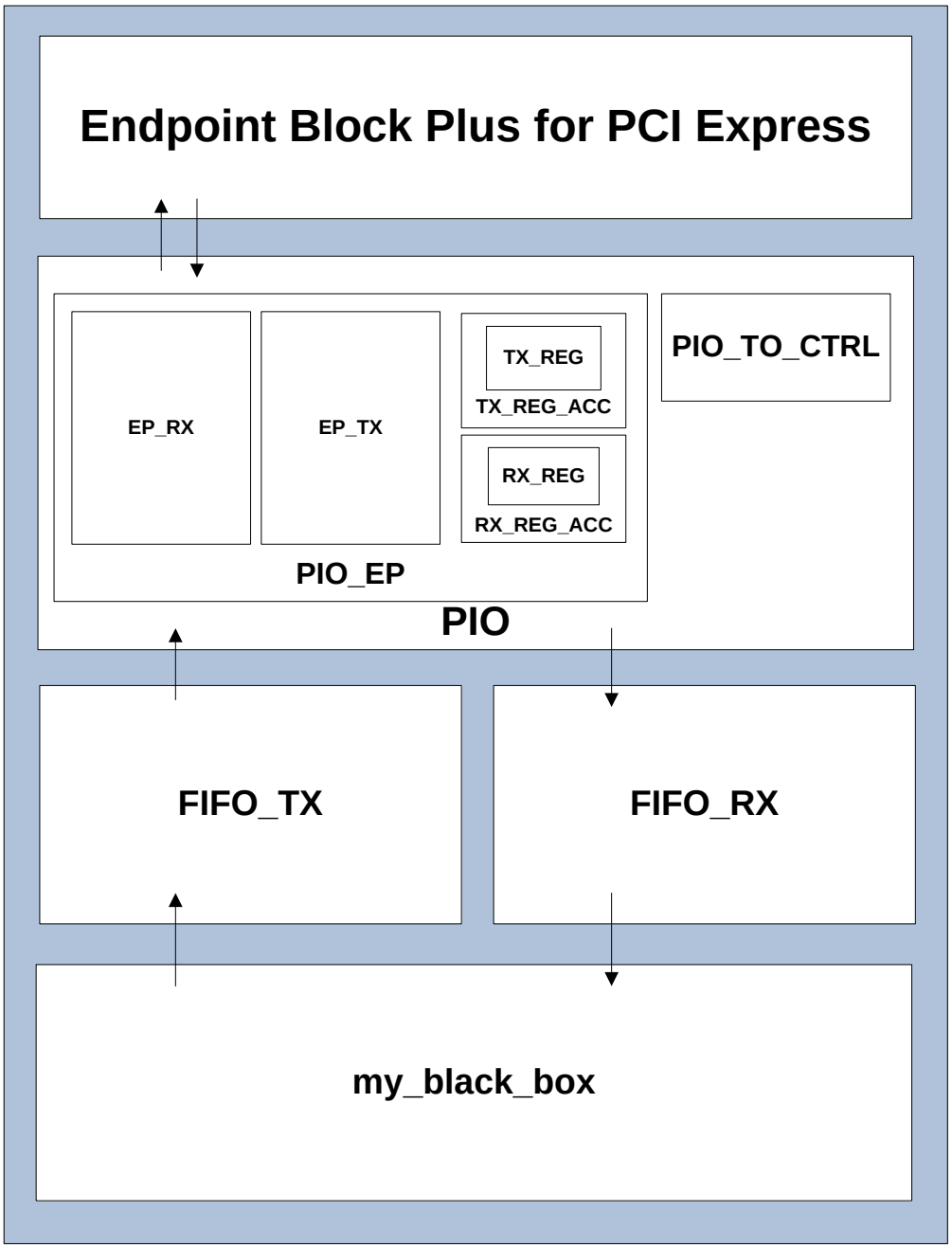
- Έπειτα, προωθήσαμε τα σήματα των δύο καταχωρητών αρκετά επίπεδα πάνω στην σχεδίαση μας έως ότου φτάσαμε στο top level της σχεδίασης και συνδέσαμε στους δυο παραπάνω καταχωρητές, οι οποίοι είναι προ των δυο Wrapper (RX-TX) του PCIe, σε δυο μνήμες λογικής ουράς-FIFO, δηλαδή το πρώτο δεδομένο που αποθηκεύεται (stored) είναι και το πρώτο δεδομένο που εξέρχεται (loaded).

- Οπότε όταν κάνουμε εγγραφή στην FPGA δεδομένα, δηλαδή όταν μεταφέρουμε δεδομένα προς την FPGA μέσω του διαύλου PCI-Express, τότε τα δεδομένα αυτά αποθηκεύονται αρχικά στον καταχωρητή που είναι αμέσως μετά τον Wrapper και στην συνέχεια σε μια FIFO. Ενώ όταν κάνουμε ανάγνωση δεδομένων, τα δεδομένα που μεταφέρονται από την FPGA στον χρήστη, μέσω του διαύλου PCI-Express προέρχονται από τον καταχωρητή που τροφοδοτείται από μια ανεξάρτητη μνήμη FIFO.

- Οι δύο FIFO που χρησιμοποιούνται συνδέονται σε ένα νέο κομμάτι της σχεδίασης, το οποίο ονομάσαμε «μαύρο κουτί» (my_black_box) για τον λόγο ότι ο χρήστης σε αυτό το σημείο θα συνδέσει την σχεδίαση του. Αναλυτικότερα για αυτό το κομμάτι της σχεδίασης θα παρουσιάσουμε παρακάτω.

- Επιπλέον, προσθέσαμε λογική έτσι ώστε ανάλογα την διεύθυνση εγγραφής/ανάγνωσης να πραγματοποιούνται συγκεκριμένες λειτουργίες. Η παραπάνω τροποποίηση έγινε για να γνωρίζει η σχεδίαση μας πότε τα δεδομένα που μεταφέρονται για εγγραφή είναι έγκυρα και πότε η διαδικασία ανάγνωσης είναι κατά βούληση.

Οπότε το σχηματικό παράδειγμα της ολοκληρωμένης σχεδίασης μας συνοψίζεται στο παρακάτω σχήμα.. Παρατηρούμε τις δυο FIFO, το module my_black_box, και τους δυο καταχωρητές που περιβάλλονται απο την λογική που επιτρέπει εγγραφή σε συγκεκριμένες διευθύνσεις



Εγγραφή (από την πλευρά της FPGA**)**

Ο χρήστης στην εκτέλεση του λογισμικού για εγγραφή στην FPGA, πρέπει με κάποιο τρόπο να ενημερώσει την FPGA ότι τα δεδομένα που θα αποσταλούν είναι έγκυρα. Δηλαδή ότι ο χρήστης του λογισμικού εις γνώση του κάνει την μεταφορά των δεδομένων. Για τον λόγο αυτό, ορίσαμε μια διεύθυνση εγγραφής η οποία θα χρησιμοποιείται αποκλειστικά για εγγραφή. Έτσι όταν στην FPGA φτάσει μια εντολή λήψης δεδομένων από τον Ηλεκτρονικό Υπολογιστή που προτίθεται να γράψει στην συγκεκριμένη αυτή διεύθυνση, τότε, και μόνον τότε, τα δεδομένα θα μεταφερθούν με επιτυχία και θα οδηγηθούν σε αποθήκευση. Σε διαφορετική περίπτωση τα δεδομένα θα απορριφθούν.

Η Write Address είναι μια συγκεκριμένη διεύθυνση, (στην σχεδίαση μας επιλέχθηκε αυθαίρετα η διεύθυνση 0x01C). Ο τρόπος που υλοποιήθηκε αυτή η συνθήκη είναι παρεμβάλλοντας μια συνθήκη αναγνώρισης σε επίπεδο FPGA, έτσι ελέγχουμε κάθε φορά την διεύθυνση ανάγνωσης κρατώντας το Write Enable ανενεργό έως ότου η διεύθυνση είναι η επιθυμητή. Όταν η διεύθυνση εγγραφής είναι η επιθυμητή τότε για ένα κύκλο ρολογιού ενεργοποιούμε την εγγραφή και μετά την απενεργοποιούμε μέχρι την επόμενη φορά που θα κληθεί να μεταφερθεί κάποιο πακέτο από τον Η/Υ.

Ανάγνωση (από την πλευρά της FPGA **)**

Για να γίνει ανάγνωση χρησιμοποιούμε την μέθοδο του Δημοψηφίσματος (Polling). Διαβάζουμε πρώτα μια διεύθυνση που μας ενημερώνει αν υπάρχουν έγκυρα δεδομένα στην μνήμη της FPGA ή αν είναι άδεια. Αν υπάρχουν έγκυρα δεδομένα, τότε μπορούμε διαβάζοντας την διεύθυνση ανάγνωσης να λάβουμε μια στοιχειώδη ποσότητα (32Bit) έγκυρων δεδομένων. Έπειτα θα διαβάσουμε εκ νέου την διεύθυνση ενημέρωσης για έγκυρα δεδομένα για να σιγουρευτούμε ότι υπάρχουν ακόμα ωφέλιμα δεδομένα στην μνήμη. Αυτή η εναλλάξ λογική μπορεί να παρουσιάζεται χρονοβόρα καθώς απαιτεί δύο αναγνώσεις μνήμης, για μια στοιχειώδη ποσότητα συν την επιπλέον λογική, αλλά είναι απαραίτητη για να είμαστε σίγουροι ότι τα δεδομένα που λαμβάνει ο χρήστης είναι έγκυρα.

Για να είμαστε σίγουροι ότι τα δεδομένα που θα μεταφέρουμε από την FPGA στην Η/Υ είναι έγκυρα και όχι σκουπίδια, πρώτη κίνηση μας είναι να αναγνώσουμε μια διεύθυνση που ονομάζουμε διεύθυνση Valid. Όταν στην FPGA φθάσει εντολή ανάγνωσης αυτής της διεύθυνσης, τότε ελέγχουμε το σήμα TX_fifo_empty, δηλαδή το βοηθητικό σήμα της FIFO προ του TX Wrapper που δηλώνει αν υπάρχουν δεδομένα ή αν είναι άδεια. Όσο είναι άδεια, δεν υποστηρίζεται η διαδικασία της ανάγνωσης. Αν υπάρχουν δεδομένα, τότε πρέπει να γίνει αίτηση ανάγνωσης σε μια ακόμα συγκεκριμένη διεύθυνση. Την διεύθυνση ανάγνωσης δεδομένων (Read Address). Η διαδικασία ανάγνωσης είναι εφικτή για μια στοιχειώδη ποσότητα δεδομένων των 32BIT. Έπειτα πρέπει να γίνει εκ νέου έλεγχος της διεύθυνσης Valid για έγκυρα δεδομένα, και πράττουμε αναλόγως.

FIFO_RX - FIFO_TX

Οι δυο FIFO που χρησιμοποιήθηκαν παράχθηκαν από τον Core Generator της Xilinx. Δημιουργούμε ένα project με τα στοιχεία της αναδιατασσόμενης συσκευής που χρησιμοποιούμε. Έπειτα επιλέγουμε **Memories & Storage Elements** -> **FIFOs** -> **Fifo Generator.**

Επιλέγουμε Independent Clocks (Wr_Clk, Rd_Clk) Block Ram καθώς το συγκεκριμένο είδος FIFO καλύπτει τις ανάγκες μας, οι οποίες είναι:

- Ξεχωριστό ρολόι ανάγνωσης από το ρολόι εγγραφής

- Διαφορετικό μέγεθος δεδομένων ανάγνωσης και εγγραφής (πχ στο PCIe-to-Aurora Bridge, η δυο FIFO από την πλευρά του PCIe διάβαζαν/έγραφαν δεδομένα μεγέθους 32Bit με ρολόι στα 62.50MHz ενώ από την πλευρά του Aurora έγραφαν/διάβαζαν ποσότητες των 16Bit χρησιμοποιώντας ρολόι στα 156.25MHz)

Το μέγεθος των FIFO επιλέχθηκε στις 4096 διευθύνσεις των 32Bit δεδομένων. Δηλαδή η μέγιστη χωρητικότητα είναι 16KByte. Η χωρητικότητα των FIFO που παράχθηκε είναι μικρή, αλλά αρκετή αν συνειδητοποιήσουμε ότι ο ρόλος τους στην σχεδίαση είναι να αποθηκεύουν προσωρινά τα δεδομένα (temporary buffer) έως ότου προωθηθούν στο επόμενο κομμάτι της σχεδίασης (my_black_box, aurora).

my_black_box

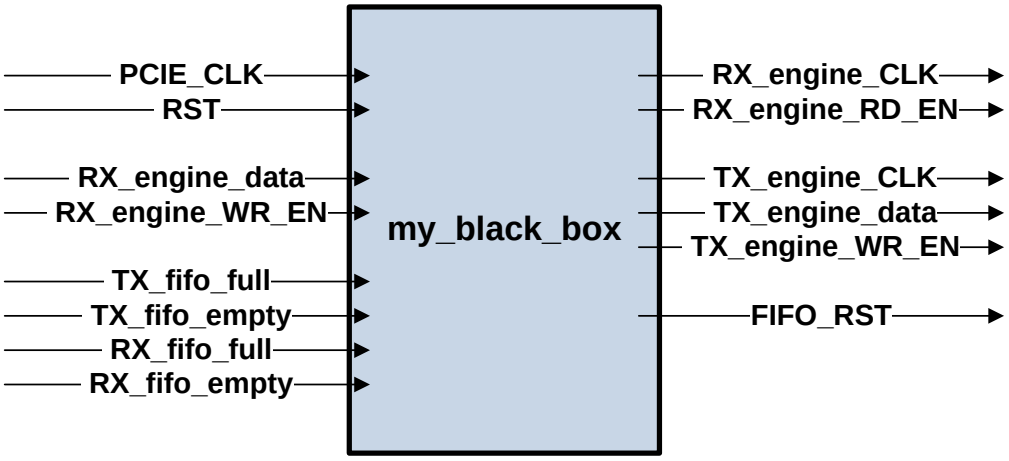
Το κομμάτι αυτό της σχεδίασης το ονομάσαμε έτσι, καθώς σε αυτό το σημείο ο χρήστης, που προτίθεται να χρησιμοποιήσει την σχεδίαση που υποστηρίζει την επικοινωνία μέσω του PCI-Express, συνδέει την δική του σχεδίαση. Μέσα στο my_black_box υπάρχουν:

- για την FIFO ανάγνωσης δεδομένων μέσω του διαύλου PCIe (FIFO_TX) υπάρχει η διεπαφή των σημάτων ενεργοποίησης εγγραφής (Write enable), δεδομένα προς την FIFO (Data in), καθώς και σήματα FIFO άδεια/γεμάτη που χρησιμεύουν στον χρήστη για την λογική που θα ακολουθηθεί.

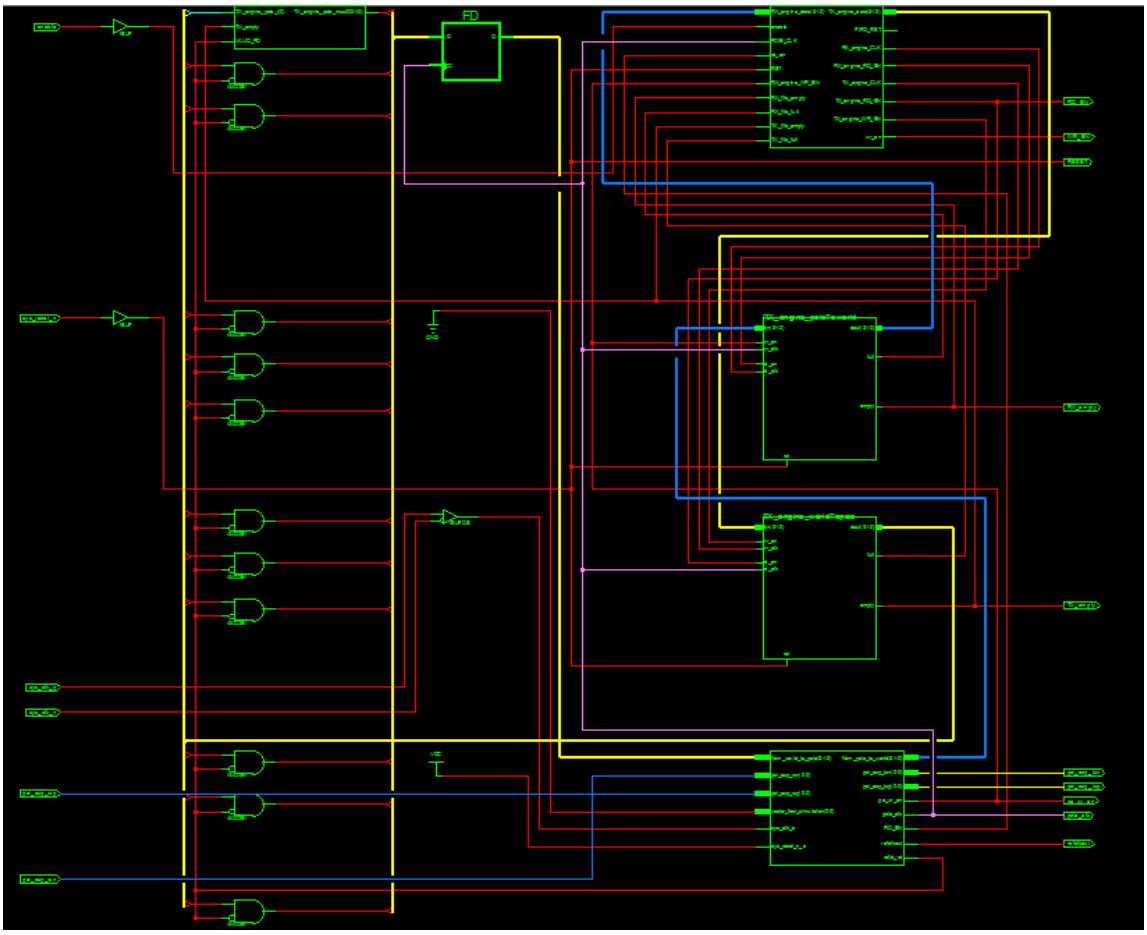
- για την FIFO εγγραφής δεδομένων (FIFO_RX) στην FPGA μέσω του διαύλου PCIe υπάρχει η διεπαφή των σημάτων ενεργοποίησης ανάγνωσης (Read Enable), δεδομένα που αναγνώστηκαν από την FIFO_RX (Data out), καθώς και τα σήματα FIFO άδεια/γεμάτη που θα βοηθήσουν τον χρήστη να κατανοήσει τις συνθήκες που επικρατούν για να γίνει ανάγνωση/εγγραφή.

- επίσης υπάρχει σαν είσοδος το ρολόι του PCIe, και σαν έξοδος τα ρολόγια: ρολόι εγγραφής στην FIFO_TX, ρολόι ανάγνωσης από την FIFO_RX. Ο χρήστης έχει την δυνατότητα
  i. να χρησιμοποιήσει το ρολόι λειτουργίας του διαύλου PCI-Express για να τροφοδοτήσει τα δύο τελευταία ρολόγια
  ii. να χρησιμοποιήσει δικά του ρολόγια, ή ρολόγια που προέρχονται από την δική του σχεδίαση και τον ικανοποιούν.
  iii. Επίσης, έχει την δυνατότητα να επιλέξει διαφορετικό ρολόι για την μια FIFO και διαφορετικό για την άλλη καθώς είναι ανεξάρτητες, με διαφορετικό ρολόι ανάγνωσης και εγγραφής.

- τέλος υπάρχει ένα σήμα αρχικοποίησης (FIFO_RST) το οποίο είναι παράλληλο με το κανονικό reset του διαύλου PCI-Express και χρησιμεύει στο να συνδέσει ο χρήστης το συγκεκριμένο σήμα, με το ολικό Reset της σχεδίασης του.

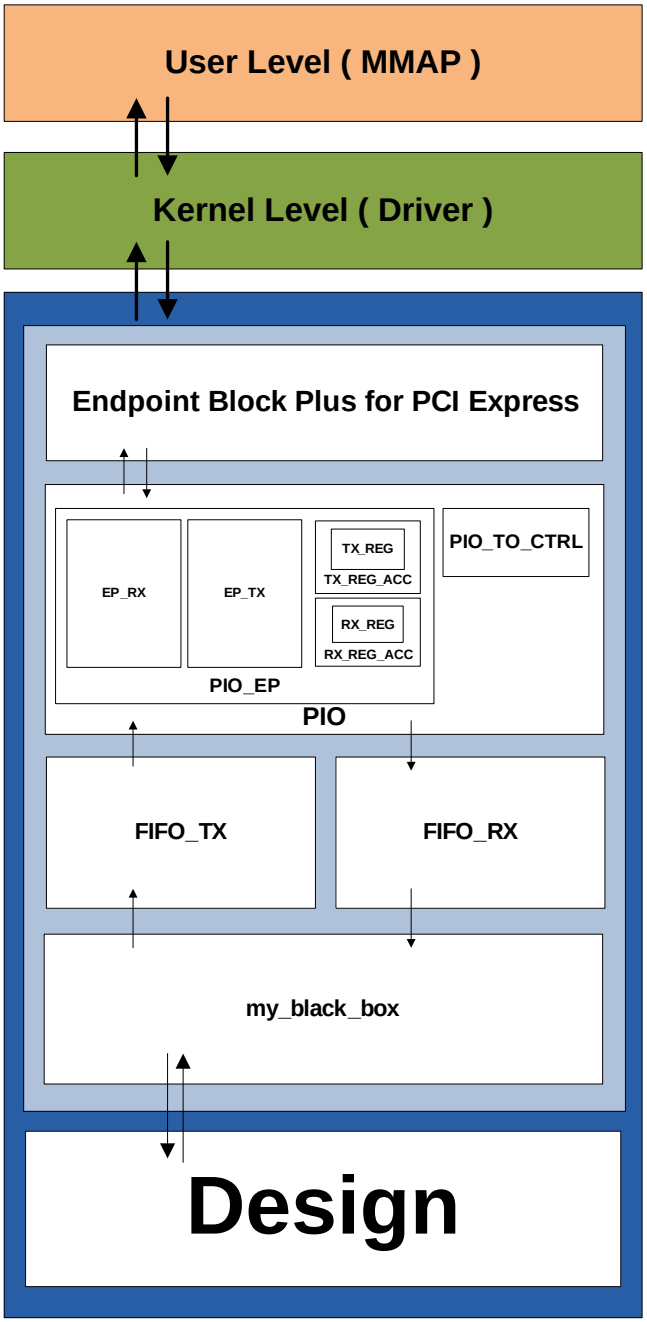


Ουσιαστικά το component my_black_box πρόκειται για το μέρος της σχεδίασης όπου θα συνδεθεί η FPGA που υποστηρίζει την επικοινωνία μέσω του PCI-Express με μια οποιαδήποτε άλλη σχεδίαση από/προς την οποία θέλουμε να μεταφέρουμε δεδομένα μέσω του διαύλου PCI-Express. Ο χρήστης έχει την δυνατότητα να επιλέξει ο ίδιος τις συνθήκες κάτω από τις οποίες θα γίνεται η μεταφορά δεδομένων, τα ρολόγια που θα χρησιμοποιηθούν και τον καλύπτουν, και φυσικά να εισάγει τα δεδομένα προς εγγραφή/ανάγνωση.



Μπλε γραμμή: Receive Engine
Κίτρινη γραμμή: Transmit Engine
Ροζ γραμμή: PCI Express Clock

Τελική μορφή

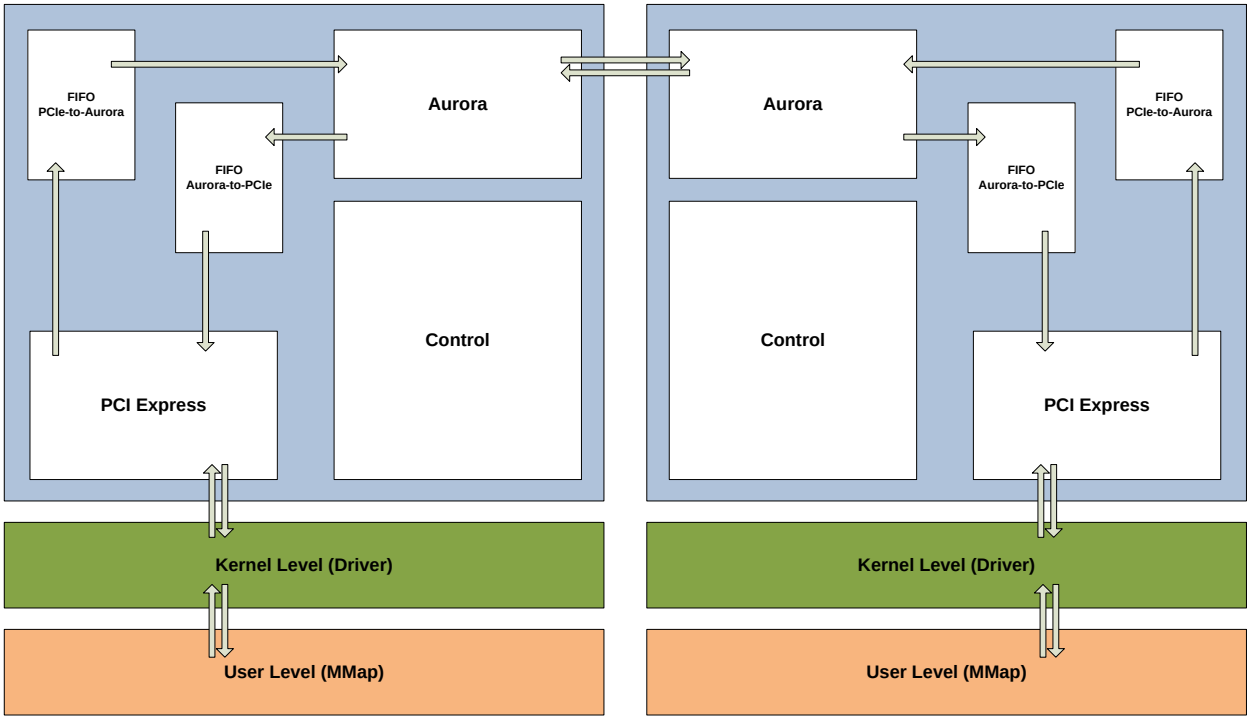


Στην δική μας σχεδίαση, επειδή δεν έχουμε εισάγει κάποια σχεδίαση-τροφοδότη δεδομένων, απλά συνδέσαμε τα δεδομένα ανάγνωσης της RX-FIFO με τα δεδομένα εγγραφής της TX-FIFO (μέσα στο my_black_box), καθώς χρησιμοποιήσαμε και το ρολόι του PCIe για να οδηγήσουμε τα δυο ελεύθερα ρολόγια της κάθε FIFO. Επίσης προσθέσαμε απλή λογική στο my_black_box η οποία δέχεται τα σήματα Enable/Empty/Full και οργανώνει την μεταφορά δεδομένων. Οπότε έχουμε σαν αποτέλεσμα να αποθηκεύουμε δεδομένα στην FPGA και μετά να κάνουμε ανάγνωση των δεδομένων αυτών. Φυσικά αυτή η λογική δεν ωφελεί σε κάτι παραπάνω από την απόδειξη της σωστής λειτουργίας της FPGA καθώς και του λογισμικού οδήγησης του PCI-Express.

PCIe-to-Aurora Bridge

Αφού ολοκληρώσαμε και δοκιμάσαμε την λειτουργία του PCI-Express Linux Driver και της σχεδίασης της FPGA που υποστηρίζει την επικοινωνία αυτή μέσω του διαύλου PCI-Express, σκεφτήκαμε σαν μια πιο ολοκληρωμένη υλοποίηση να συνδέσουμε στην σχεδίαση μας, έναν πυρήνα μεταφοράς δεδομένων που υποστηρίζει το πρωτόκολλο Aurora. Οπότε ο χρήστης από τον Ηλεκτρονικό Υπολογιστή θα έχει την δυνατότητα μέσω Software (MMap) να στείλει δεδομένα στην FPGA, η οποία θα τα προωθήσει μέσω του Aurora σε μια άλλη συσκευή (στον ίδιο ή διαφορετικό Η/Υ) η οποία θα λειτουργεί με ακριβώς την ίδια λογική, αλλά εκτελώντας την ανάποδη λειτουργία. Δηλαδή θα λάβει τα δεδομένα μέσω Aurora, και θα μεταφέρει στον Ηλεκτρονικό Υπολογιστή μέσω του Software χρησιμοποιώντας τον δίαυλο PCI-Express.

Ένα σχηματικό παράδειγμα της σχεδίασης που αναφέρουμε είναι το παρακάτω:



Παρατηρούμε ότι τα δομικά μέρη της σχεδίασης PCI-Express και FIFO παραμένουν ίδια ενώ στη θέση του my_black_box συνδέσαμε τον πηγαίο κώδικα που υλοποιεί την μεταφορά δεδομένων χρησιμοποιώντας το πρωτόκολλο Aurora. Επίσης χρειάζεται μια βαθμίδα ελέγχου η οποία θα ελέγχει τα σήματα ενεργοποίησης και θα κάνει δυνατή την μεταφορά των δεδομένων.

Οι δυο FIFO που χρησιμοποιούνται είναι απαραίτητες καθώς:

- η ταχύτητα αποστολής και λήψης του Aurora (3.125Gbs/Lane) είναι διαφορετική από την ταχύτητα αποστολής και λήψης του PCI-Express (2.5Gbs/Lane)
- το ρολόι λειτουργίας του Aurora (156.25MHz) είναι διαφορετικό από το ρολόι λειτουργίας του PCI-Express (62.50MHz).

- ο δίαυλος PCI Express μεταφέρει ποσότητες των 32Bit ενώ το Aurora που δημιουργήσαμε μεταφέρει ποσότητες των 16BIT.

Ο λόγος που επιλέξαμε το Aurora να μεταφέρει ποσότητες 16Bit αντί για 32Bit είναι ότι αν το PCI Express λαμβάνει δεδομένα με ρυθμό 32Bit*62.50MHz = 32Bit/16nsec και το Aurora λαμβάνει 32Bit*156.25MHz = 32Bit/6.4ns τότε η διαφορά είναι αρκετά μεγάλη και θα υπήρχε μεγάλος συνωστισμός δεδομένων στις FIFO. Οπότε μεταφέροντας ποσότητες 16BIT μέσω του Aurora, πετυχαίνουμε μεταφορά με ρυθμό 32Bit/12.8nsec το οποίο δεν αποτελεί τόσο μεγάλο χάσμα όσο πριν. Εξάλλου το Bottleneck της σχεδίασης μας αυτής είναι το PCI-Express και όχι το Aurora.

Αν δεν υπήρχαν οι FIFO, τότε θα είχαμε απώλεια δεδομένων είτε επειδή δεν θα προλάβαιναν τα εισερχόμενα με μεγαλύτερη ταχύτητα δεδομένα να γίνουν πακέτα και να αποσταλούν, είτε επειδή το γρηγορότερο ρολόι θα διάβαζε δυο φορές τα ίδια δεδομένα μέχρι να φθάσουν καινούργια από το αργό. Χρησιμοποιώντας τις FIFO, και ελέγχοντας την κατάσταση τους με τα σήματα Empty/Full καθώς και τα προγραμματιζόμενα Almost Empty/Full μπορούμε να γνωρίζουμε την ποσότητα των δεδομένων, και χρησιμοποιώντας τα παραπάνω σήματα να προλάβουμε τα δυσάρεστα αποτελέσματα με την χρήση κατάλληλης λογικής που διαχειρίζεται τις λειτουργίες ανάγνωσης/εγγραφής.

Δημιουργώντας το Aurora Protocol

Με τα εργαλεία της Xilinx, και συγκεκριμένα με το Core Generator δημιουργήσαμε τον κώδικα που υλοποιεί το πρωτόκολλο Aurora.

Αρχικά δημιουργήσαμε ένα project για την αναδιατασσόμενη συσκευή που χρησιμοποιούμε θέτοντας τις ακόλουθες παραμέτρους όπως και στην δημιουργία του πυρήνα του διαύλου PCI Express:

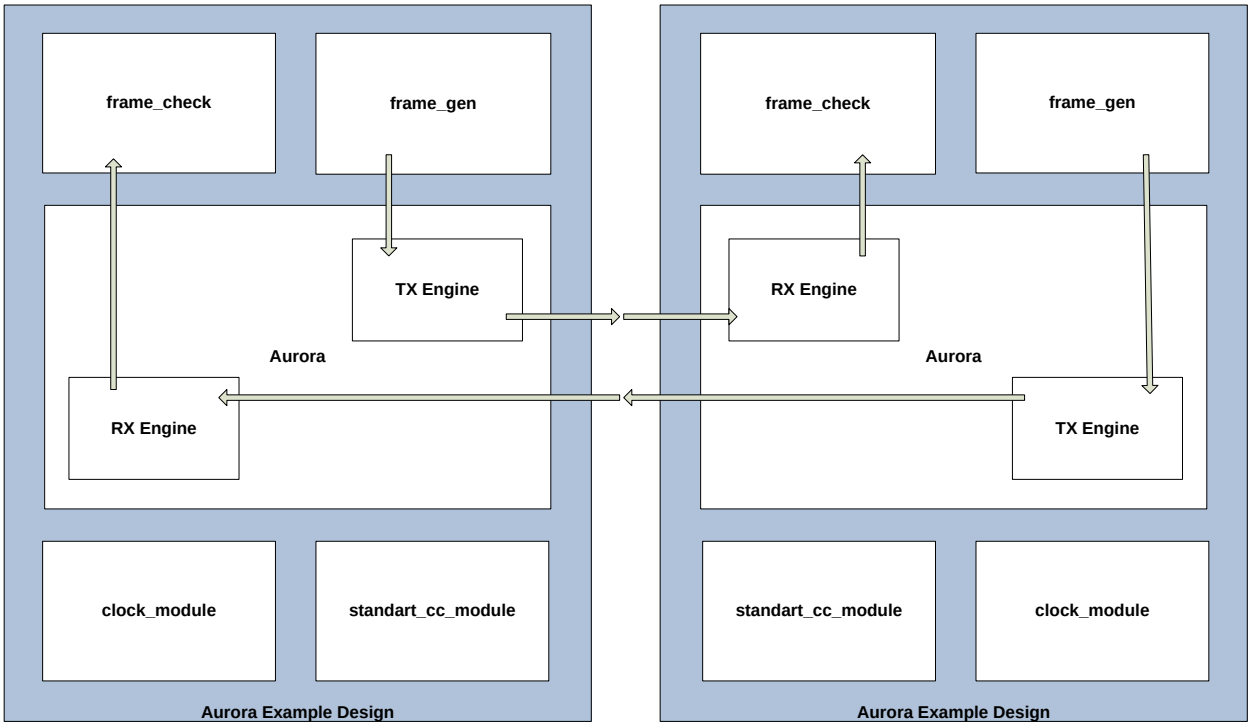
Family: Virtex-5
Device: xc5vlx110t
Package: ff1136
Speed Grade : -1

Έπειτα αφού επιλέξουμε **Communication & Networking** -> **Serial Interfaces** -> **GTP Aurora** εμφανίζεται το menu επιλογών του πρωτοκόλλου Aurora που πρόκειται να υλοποιήσουμε.

Επιλέγουμε:

Silicon version: PRODUCTION
Aurora Lanes: 1
Interface: Framing
Line Rate: 3.125Gbps
Reference Clock: 156.25MHz
GTP Placement: τον ίδιο GTP με του PCI Express
Clock Source: GREF_CLK ή GTPD# CLK

Ο πηγαίος κώδικας παράγεται σε γλώσσα VHDL αποτελεί όπως και στην ανάλογη περίπτωση με τον πυρήνα του PCI Express ένα σημείο αρχής για τους μηχανικούς που ενδιαφέρονται να εξελίξουν στο πρωτόκολλο Aurora στην σχεδίαση τους. Ο κώδικας που παράγεται μέσω του εργαλείου Core Generator έχεις την παρακάτω μορφή:



Ας σταθούμε λίγο στα κομμάτια της υλοποίησης frame_check και frame_gen. Το component frame_gen δημιουργεί τυχαία δεδομένα τα οποία σχηματίζονται σε πακέτα και οδηγούνται στον μηχανισμό μετάδοσης (Transmit Engine) του Aurora και έπειτα προς την έξοδο για να ληφθεί από άλλο πρωτόκολλο Aurora.

Τα δεδομένα που αποστέλλονται από την παραπάνω διαδικασία, λαμβάνονται από τον μηχανισμό λήψης (Receive Engine) του πρωτοκόλλου Aurora μιας ίδιας ή διαφορετικής συσκευής σε έναν ίδιο ή διαφορετικό Ηλεκτρονικό Υπολογιστή. Έπειτα οδηγούνται στο component frame_check όπου ελέγχεται αν το πακέτο που στάλθηκε, έφθασε επιτυχώς ή αν έχει λάθη. Και αν έχει λάθη ή αλλοίωση ποιος ο αριθμός των λαθών.

Η παραπάνω λειτουργία που αναφέρθηκε συνοπτικά, είναι η λειτουργία που προσφέρεται κατά την δημιουργία του πρωτοκόλλου Aurora με τα εργαλεία της Xilinx.

Αφού κατανοήσαμε πλήρως τον τρόπο λειτουργίας του πρωτοκόλλου Aurora:

- χρησιμοποιήσαμε αυτούσιο το κομμάτι της σχεδίασης που αναλαμβάνει την μεταφορά και την λήψη των πακέτων.

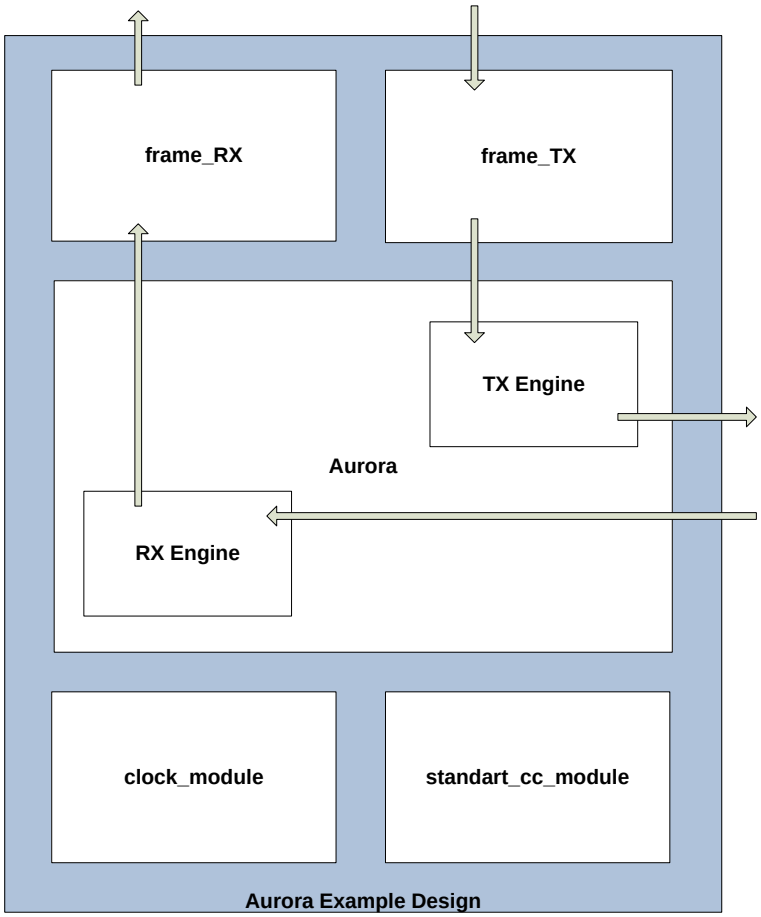
- Αφαιρέσαμε τους μηχανισμούς frame_check και frame_gen αφού χρησιμοποιούνται για την επαλήθευση της σωστής λειτουργίας του Aurora και μόνον.

- Με βάση την λειτουργία των μονάδων frame_check και frame_gen δημιουργήσαμε ανάλογες μονάδες οι οποίες έχουν τις εξής λειτουργίες:

    i. η μια λαμβάνει δεδομένα και δημιουργεί τα πακέτα προς αποστολή. Κάτι ανάλογο με το frame_gen μόνο που τα δεδομένα δεν είναι τυχαία, αλλά προέρχονται από εισαγωγή στην μονάδα. Τα πακέτα οδηγούνται στον μηχανισμό αποστολής του πρωτοκόλλου Aurora.

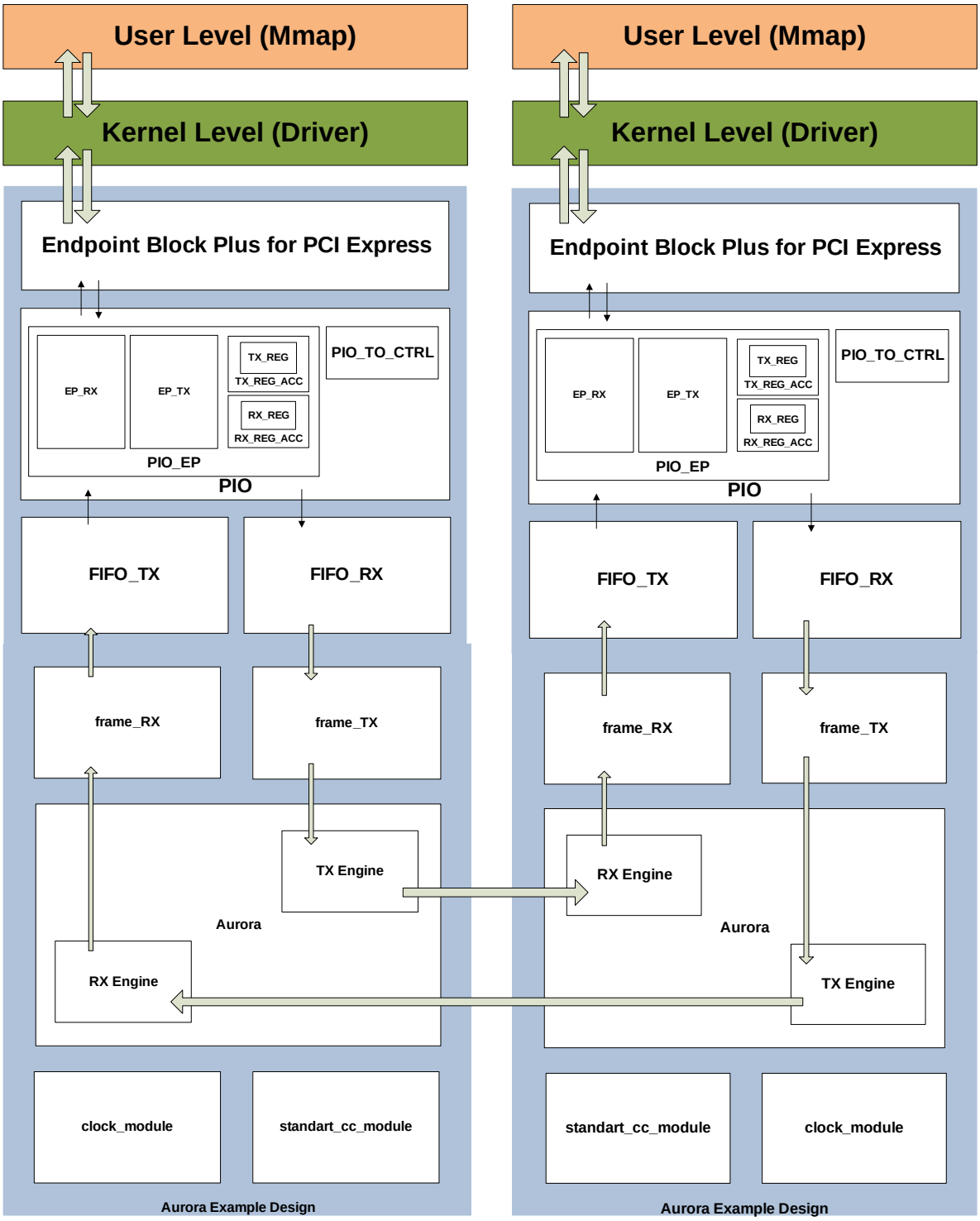
    ii. στην άλλη εισέρχονται τα πακέτα που έχουν ληφθεί από τον μηχανισμό λήψης (Receive Engine) του πρωτοκόλλου Aurora, αναλύονται, και απομονώνονται τα δεδομένα. Τέλος εξέρχονται από την μονάδα και οδηγούνται σε άλλο μηχανισμό της σχεδίασης.

Η τελική μορφή του πρωτοκόλλου Aurora όπως προετοιμάστηκε για να χρησιμοποιηθεί στην σχεδίαση μας είναι η παρακάτω:



Υπάρχουν δυο σήματα I/O στα οποία οδηγούνται τα δεδομένα που θέλουμε να αποστείλουμε χρησιμοποιώντας το πρωτόκολλο Aurora ή τα δεδομένα τα οποία ελήφθησαν μέσω του πρωτόκολλου Aurora.
Καθώς το πρωτόκολλο Aurora προετοιμάζεται για να συνδεθεί με τον δίαυλο PCI-Express, και να δημιουργηθεί η PCIe-to-Aurora bridge αρκετές ακόμα επεμβάσεις πραγματοποιήθηκαν για να μπορεί να είναι λειτουργικό το εγχείρημα αυτό.

Η τελική μορφή του PCIe-to-Aurora Bridge είναι η παρακάτω:



Προβλήματα

Τα προβλήματα τα οποία παρουσιάστηκαν κατά την υλοποίηση τόσο της σχεδίασης της επικοινωνίας του Ηλεκτρονικού Υπολογιστή μέσω του διαύλου PCI Express με οποιαδήποτε άλλη σχεδίαση όσο και της σχεδίασης PCIe-to-Aurora Bridge ήταν πολλά.

Μάλιστα μερικά από αυτά όσον αφορά την ολοκληρωμένη υλοποίηση του Bridge PCIe-to-Aurora **ΔΕΝ** λύθηκαν, κάτι το οποίο μας στέρησε την δυνατότητα να παρουσιάσουμε μετρήσεις και να θεωρήσουμε την συγκεκριμένη προσπάθεια σαν μια προσπάθεια η οποία στέφθηκε με επιτυχία.

Η PCIe-to-Aurora Bridge σαν σχεδίαση είναι ολοκληρωμένη καθώς όλα τα προβλεπόμενα για να λειτουργήσει η σχεδίαση έχουν πραγματοποιηθεί. Η επικοινωνία μέσω του διαύλου PCI-Express είναι λειτουργική, οι μνήμες FIFO για προσωρινή αποθήκευση των δεδομένων καθώς και ο ελεγκτής που παράγει τα σήματα ελέγχου της μεταφοράς δεδομένων έχουν παραχθεί. Όλες οι απαιτούμενες ενέργειες έχουν πραγματοποιηθεί.

Το πρόβλημα το οποίο δεν μπορέσαμε να ξεπεράσουμε, και το οποίο μας στερεί την ικανοποίηση να παρουσιάσουμε την λειτουργική PCIe-to-Aurora Bridge είναι το ρολόι τροφοδοσίας της σχεδίασης του Aurora. Κύριο πρόβλημα οι ασάφειες στην βιβλιογραφία της Xilinx, όσων αφορά το είδος τον ρολογιών που χρειάζονται συγκεκριμένα modules της σχεδίασης, μας οδήγησαν στην λανθασμένη τροφοδοσία του πρωτοκόλλου με ακατάλληλα σήματα, και σαν αποτέλεσμα την αδυναμία λειτουργίας.

Το συγκεκριμένο πρόβλημα, θεωρητικά είναι εύκολα προσπελάσιμο, και πρακτικά θα χρειαστεί ελάχιστο χρόνο για κάποιον μηχανικό με εμπειρία στα πρωτόκολλα aurora που υλοποιούνται σε Virtex-5.

Όσων αφορά το είδος των προβλημάτων που παρουσιάστηκαν:

- **για την επικοινωνία μέσω του Διαύλου PCI Express:**

    - κύριο πρόβλημα ήταν η έλλειψη δυνατότητας εξομοίωσης της σχεδίασης (non simulatable design) από τα εργαλεία της Xilinx. Η έλλειψη της δυνατότητας αυτής, μας ανάγκασε να δημιουργήσουμε τον πλήρη κώδικα της σχεδίασης σε θεωρητικό επίπεδο, και αφού ολοκληρώθηκαν όλα τα ζητούμενα κομμάτια, και ήταν σε θέση να προγραμματιστεί η FPGA, τότε, και μόνον τότε ήμασταν σε θέση να λειτουργήσουμε ουσιαστικά την σχεδίαση μας, και να δούμε στην πράξη τι προβλήματα προκύπτουν και που χρειάζεται βελτίωση ο κώδικας.

    - Δυσκολία παρουσιάστηκε επίσης στο ότι για να πραγματοποιηθεί αποσφαλμάτωση του κώδικα έπρεπε να περιμένουμε να ολοκληρωθεί η σχεδίαση και να προγραμματιστεί η FPGA με μια έως τότε αγνώστου λειτουργικότητας σχεδίαση, η οποία οδηγείται από έναν αγνώστου λειτουργικότητας οδηγό λειτουργικού συστήματος ελευθέρου λογισμικού (Linux Driver) κάνοντας μεταφορά δεδομένων σε επίπεδο χρήστη (User

Level) εκτελώντας ένα λογισμικό (mmap) το οποίο είναι επίσης αγνώστου λειτουργικότητας. Καθώς τα παραπάνω δεν μπορούσαν να ελεγχθούν αυτόνομα όσον αφορά την λειτουργικότητα τους (πέρα από το αν υπάρχουν συντακτικά σφάλματα στον κώδικα τους), έπρεπε να περιμένουμε να ολοκληρωθεί το σύνολο των απαιτούμενων μερών της τελικής σχεδίασης και αφού χρησιμοποιηθούν όλα μαζί να αποφασίσουμε εάν ανταποκρίνονται στις προσδοκίες μας ή αν χρειάζονται βελτιώσεις. Η αποσφάλματωση ήταν δύσκολη καθώς δεν μπορούσαμε να είμαστε σίγουροι σε πιο σημείο και σε πιο μέρος της σχεδίασης υπάρχει το πρόβλημα. Μόνο εικασίες για την πιθανή αιτία ενός προβλήματος μπορούσαν να γίνουν, έως ότου σιγουρευτούμε για την σωστή λειτουργία των βασικών μερών (Linux Driver, MMap).

- Ελλιπής υποστήριξη από την εταιρία που μας προμηθεύει τις συσκευές αναδιατασσόμενης λογικής Virtex-5 πάνω στην υλοποίηση του κώδικα VHDL για την υποστήριξη μεταφοράς δεδομένων μέσω PCI Express. Υπάρχουν διαθέσιμες άφθονες σημειώσεις θεωρητικού χαρακτήρα, αλλά δεν υπάρχει μια παραπλήσια σχεδίαση με το ζητούμενο έτσι ώστε να υπάρχει μια βάση πάνω στην οποία να γίνει αναζήτηση λύσης που να αρμόζει στα δικά μας πρότυπα και προσπάθεια βελτίωσης.

- Μοναδικότητα της υλοποίησης αυτής καθώς στο διαδίκτυο δεν βρέθηκε παραπλήσια σχεδίαση. Συνεπάγεται έλλειψη εμπειρίας και τεχνικής βοήθειας ακόμα και για μικρά προβλήματα που παρουσιάστηκαν το οποίο οδηγεί σε μεγάλο χρόνο αποσφαλμάτωσης.

- **Για την υλοποίηση του Aurora Protocol**

  - Ανεπαρκή υποστήριξη από την πλευρά της εταιρίας πάνω στην υλοποίηση του πρωτοκόλλου Aurora. Ασαφή θεωρητική κάλυψη των αναγκών για την λειτουργία του πρωτοκόλλου.
  - Βοηθητικές υλοποιήσεις για την αναδιατασσόμενη συσκευή Virtex-5, απροσδιορίστου λειτουργικότητας, χωρίς καν να προορίζονται για την αναδιατασσόμενη συσκευή Virtex-5 αλλά για την συσκευή Virtex-II PRO. Χαρακτηριστικό παράδειγμα προβλήματος τέτοιου είδους υπάρχει όσων αφορά το DCM module που οδηγεί τα ρολόγια στο πρωτόκολλου Aurora, καθώς η Xilinx ενώ ξεκαθαρίζει ότι το module DCM προορίζεται για υλοποιήσεις σε αναδιατασσόμενες συσκευές έως Virtex II PRO, και για τις συσκευές Virtex-5 χρησιμοποιεί το module DCM_ADV (Advance), παρόλα αυτά στην υλοποίηση του πρωτοκόλλου για την Virtex-5 από τον Core Generator ή στο παράδειγμα που έχει έτοιμο η Xilinx, το module που ενσωματώνεται είναι ένα DCM, το οποίο DCM δεν υποστηρίζεται από την Virtex-5.
  - Αντιφατικές συλλογές προδιαγραφών. Σε βιβλιογραφία της εταιρίας παρουσιάζονται οι προδιαγραφές λειτουργίας του πρωτοκόλλου όσον αφορά τα βασικά σήματα λειτουργίας ρολογιών καθώς και άλλα πιο εξειδικευμένα, και σε επόμενη αναφορά καταρρίπτονται οι προαναφερθείσες προδιαγραφές απαιτώντας διαφορετική υλοποίηση.

- Έλλειψη εμπειρίας πάνω στην εφαρμογή του πρωτοκόλλου Aurora στην αναδιατασσόμενη συσκευή Virtex-5.

7. Μελλοντική εργασία

Μελλοντική Εργασία

Σαν μελλοντική εργασία, λαμβάνοντας υπόψη ότι αυτή τη στιγμή υπάρχει

i) ένας οδηγός λειτουργικού συστήματος ελευθέρου λογισμικού που υποστηρίζει την επικοινωνία της αναδιατασσόμενης συσκευής Virtex-5 με τον Η/Υ,
ii) μια εφαρμογή η οποία αναλαμβάνει την μεταφορά των δεδομένων και υλοποιεί την μεθοδολογία Polling σε επίπεδο Software.
iii) ο κώδικας προγραμματισμού της FPGA για επικοινωνία μέσω του διαύλου PCI Express 1x Lane,
iv) μια ολοκληρωμένη αλλά μη λειτουργική έκδοση όπου τα δεδομένα μεταφέρονται μέσω του διαύλου PCI Express στην FPGA και έπειτα αποστέλλονται μέσω του πρωτοκόλλου Aurora σε άλλη αναδιατασσόμενη συσκευή ίδιου ή διαφορετικού Η/Υ (PCIe-to-Aurora Bridge).

Με βάση τα παραπάνω οι μελλοντικές βλέψεις για βελτίωση του υλικού που υπάρχει ήδη διαθέσιμο **σε επίπεδο Hardware** είναι:

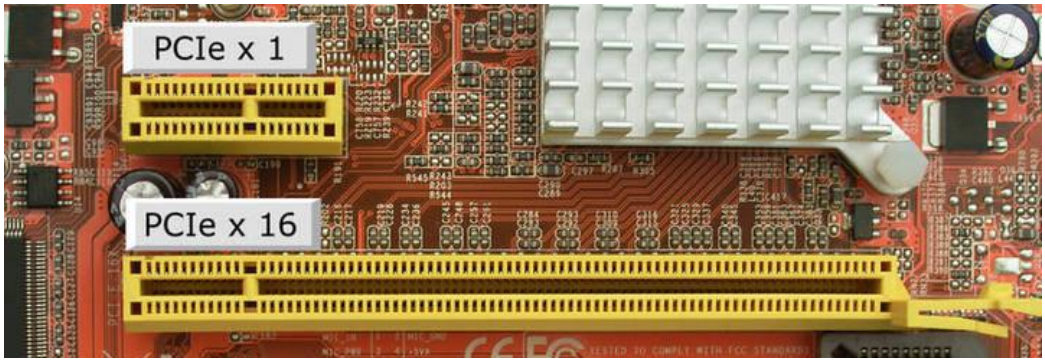
- Προφανώς η ολοκλήρωση του PCIe-to-Aurora Bridge. Ουσιαστικά στο εν λόγω κομμάτι δεν υπάρχει κάποιο σφάλμα στον κώδικα της υλοποίησης. Το σφάλμα είναι σε επίπεδο κατανόησης των χαρακτηριστικών της λειτουργίας του πρωτοκόλλου Aurora, καθώς και εύρεση τρόπου τροφοδότησης με ρολόι το κύκλωμα. Κατά 90% θα χρησιμοποιηθεί το DCM_ADV (Digital Clock Manager Advance) module της Xilinx που παράγεται από τον Core Generator και προορίζεται για αναδιατασσόμενες συσκευές Virtex-4 και Virtex-5.

- Δημιουργία και άλλων λειτουργικών PCIe-to Bridges όπως μια ενδεχόμενη PCIe-to-Ethernet Bridge εκμεταλλευόμενοι την universal μορφή που έχει πάρει η σχεδίαση μας. Η μορφή της σχεδίασης μας καλείται universal επειδή δόθηκε έμφαση στο να είναι εύκολα προσβάσιμη και λειτουργική από χρήστες που θέλουν να συνδέσουν την δική τους σχεδίασης και να εκμεταλλευτούν την ικανότητα μεταφοράς δεδομένων στην FPGA μέσω του Η/Υ. Για αυτόν τον λόγο και το αρχείο της σχεδίασης my_black_box συντάχθηκε με τρόπο τέτοιο που να ευνοεί αυτή την προσπάθεια.

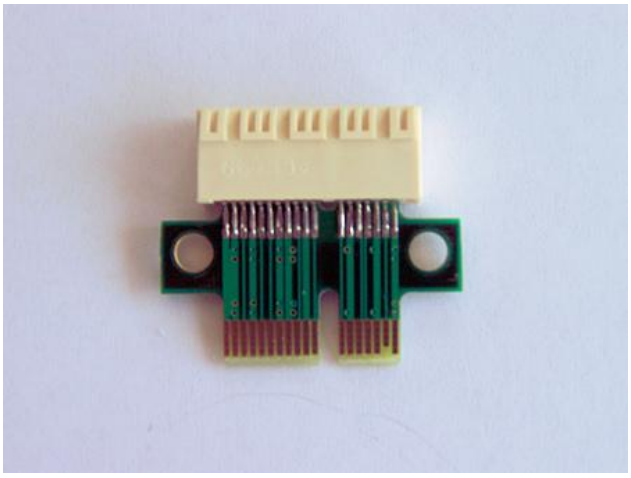
- Χρησιμοποίηση περισσότερων γραμμών μετάδοσης υποστηριζόμενες από το PCI Express για επίτευξη μεγαλύτερων ταχυτήτων μεταφοράς. Στην εργασία μας υποστηρίζουμε μια γραμμή μεταφοράς δεδομένων (1x Lane), ενώ κατά την παραγωγή του πυρήνα PCI Express από τα εργαλεία της Xilinx, οι δυνατές επιλογές για την συγκεκριμένη αναδιατασσόμενη συσκευή είναι 1x 4x 8x Lane. Χρησιμοποίηση περισσότερων γραμμών μετάδοσης ισοδυναμεί με υποστήριξη μεταφοράς μεγαλύτερου όγκου δεδομένων ανά χρονική μονάδα (μεγαλύτερο throughput).

Να σημειώσουμε εδώ ότι το board που φιλοξενεί την αναδιατασσόμενη συσκευή που χρησιμοποιούμε έχει θύρα διαύλου μόνο για 1x Lane, και άρα η συγκεκριμένη αναβάθμιση απαιτεί και αλλαγή του board με ένα που υποστηρίζει διασύνδεση με περισσότερες γραμμές μετάδοσης.

Παρακάτω φαίνεται η διαφορά στο μέγεθος του1x Lane slot από αυτήν της πύλης του 16x Lane slot.



Το board xupV5 που ενσωματώνει την αναδιατασσόμενη συσκευή xc5vlx110t που χρησιμοποιήσαμε υποστηρίζει μόνο 1x lane slot, το οποίο σαν μέγεθος φαίνεται παρακάτω.



Μελλοντική εργασία σε επίπεδο software:

- Πρωταρχικός στόχος σε μια ενδεχόμενη συνέχεια της εργασίας αυτής οφείλει να είναι η υλοποίηση υποστήριξης Interrupt προς αντικατάσταση της μεθοδολογίας Polling για αξιόπιστη μεταφορά των δεδομένων μέσω του διαύλου PCI Express. Η υλοποίηση Interrupt θα εμπεριέχεται στην οδηγό λογισμικού που συντάξαμε. Η χρήση Interrupt είναι σαφώς πιο γρήγορη σαν διαδικασία από την μεθοδολογία Polling, αλλά η πολυπλοκότητα της σαν υλοποίηση καθώς και η απειρία μας σε σύνταξη οδηγών λογισμικού, μας οδήγησε στην εκμετάλλευση της απλούστερης σε λογική αλλά και περισσότερο χρονοβόρας μεθόδου του Polling.

- Αλλαγές με στόχο την επίτευξη μεγαλύτερης ταχύτητας πέρα από την χρήση Interrupt handler. Αυτό μπορεί να πραγματοποιηθεί με αποστολή/λήψη πακέτων μεταβλητού μεγέθους όπου ο χρήστης θα επιλέγει την χωρητικότητα ανά μεταφερόμενο πακέτο. Σε αυτή την έκδοση, τα πακέτα που αποστέλλονται και λαμβάνονται έχουν το μικρότερο δυνατό μέγεθος (32bit). Η αποστολή πακέτων δεδομένων μεγάλου μεγέθους αυξάνει την ταχύτητα μεταφοράς των δεδομένων ανά μονάδα χρόνου. Γενικότερα η πιο μεγάλη σε κόστος μεταφορά δεδομένων είναι αυτή της στοιχειώδους μονάδας, καθώς προετοιμάζεται πλήρως το πακέτο γίνονται όλες οι απαραίτητες διαδικασίες, επισυνάπτονται επικεφαλίδες, SOF, EOF αλλά αποστέλλεται η μικρότερη δυνατή ποσότητα. Αν τα παραπάνω γινόντουσαν μια φορά για ένα πακέτο μεγέθους 4Kbytes αντί για 4byte, τότε η διαφορά θα ήταν αξιοσημείωτη.

- Σύνταξη εφαρμογής σε επίπεδο χρήστη (user level) η οποία θα προσφέρει επιπλέον επιλογές όπως αναζήτηση αρχείου και αποστολή του στην FPGA, ή είσοδος δεδομένων από το πληκτρολόγιο, ή κάποιες δομημένες συναρτήσεις επεξεργασίας-δημιουργίας δεδομένων προς αποστολή, καθώς και λειτουργίες όπως αποκωδικοποίησης δεδομένων που έχουν ληφθεί, αποθήκευση σε αρχείο και οτιδήποτε άλλο μπορεί να είναι χρήσιμο σαν επεξεργασία των δεδομένων που λαμβάνονται/αποστέλλονται.

- Δημιουργία script το οποίο μόλις τρέχει ο χρήστης αυτόματα θα γίνονται όλες οι αναγκαίες αλλαγές στον πυρήνα του λειτουργικού συστήματος Linux για να υποδεχθούν τον οδηγό λογισμικού που συντάξαμε. Επίσης θα μπορούσε να γίνει μια προσπάθεια να υποστηρίζεται η επιλογή του πυρήνα του λειτουργικού συστήματος που χρησιμοποιούμε, και ανάλογα με την έκδοση να γίνονται οι κατάλληλες αλλαγές στον driver έτσι ώστε να είναι συμβατό με προγενέστερες αλλά και μεταγενέστερες εκδόσεις Linux. Δηλαδή στην εκτέλεση του script ο χρήστης να επιλέγει από μια λίστα εκδόσεων πυρήνα την δικιά του, και να γίνεται η εγκατάσταση του οδηγού στον πυρήνα αυτόματα, γνωρίζοντας ότι είναι συμβατός και λειτουργικός με αυτή την έκδοση.

- Επίσης δημιουργία script το οποίο να εκτελείται και να γίνεται αυτόματα η εκτέλεση του οδηγού. Δηλαδή να μην χρειάζεται η χειροκίνητη εκτέλεση του driver (insmod, find major-minor, mkod, rmmod) αλλά να διευκολύνεται ο χρήστης ο οποίος δεν έχει εξοικείωση σε λειτουργικά συστήματα τύπου Linux.

8. Επαλήθευση Λειτουργίας

Επαλήθευση λειτουργίας Software - Hardware

Για να εξετάσουμε την σωστή λειτουργία του συνδυασμού του Linux Driver (Kernel Level) με την εφαρμογή που εκτελεί την αποστολή και λήψη των δεδομένων σε επίπεδο χρήστη (User Level) και την σχεδίαση μας σε επίπεδο Hardware, εκτελέσαμε την παρακάτω διαδικασία:

- Αρχικά, εγκαταστήσαμε στην αναδιατασσόμενη συσκευή μια σχεδίαση που ήμασταν σίγουροι ότι είναι λειτουργική. Η σχεδίαση αυτή δεν ήταν απαραίτητο να μας ενδιαφέρει ή να εκτελεί κάποια συγκεκριμένη λειτουργία, απλά θέλαμε να υπάρχει προγραμματισμένη η FPGA έτσι ώστε να μπορεί να αναγνωριστεί από τον Η/Υ. Συγκεκριμένα εγκαταστήσαμε τον PCI Express Core αυτούσιο όπως τον παράγει το εργαλείο Core Generator της Xilinx.

- Για να ελέγξουμε ότι όντως η FPGA έχει προγραμματιστεί από μια λειτουργική σχεδίαση, και πιο συγκεκριμένα από το παράδειγμα σχεδίασης που υποστηρίζει επικοινωνία μέσω του PCI Express, αποθηκεύσαμε από το διαδίκτυο την εφαρμογή PCI-tree. Το PCI-tree πρόκειται για ένα shareware πρόγραμμα το οποίο ελέγχει όλες τις συσκευές που είναι ενσωματωμένες στην μητρική πλακέτα του Η/Υ. Εκτελείται σε λειτουργικό σύστημα Microsoft Windows και εκτός του να παρουσιάζει τις συσκευές που υπάρχουν στην μητρική πλακέτα, δίνει την δυνατότητα να πραγματοποιηθούν περιορισμένης έκτασης μεταφορές δεδομένων στις συσκευές που υποστηρίζουν τέτοιου είδους λειτουργία.

- Αφού προγραμματίσαμε την FPGA και σιγουρευτήκαμε ότι η σχεδίαση είναι λειτουργική, εγκαταστήσαμε τον οδηγό λογισμικού που συντάξαμε και τον εκτελέσαμε στο λειτουργικό σύστημα Linux. Αυτό που μας ενδιέφερε σε αυτό το στάδιο εκτός του να μπορέσει να εγκατασταθεί χωρίς προβλήματα ήταν να αναγνωρίσει την συσκευή μας, να «βλέπει» δηλαδή όπως λέγεται το λειτουργικό σύστημα την συσκευή που υπάρχει στην μητρική πλακέτα με τα συγκεκριμένα Vector ID και Device ID.

- Η διαβεβαίωση ότι πράγματι ο Driver έκανε ορατή στο λειτουργικό σύστημα την αναδιατασσόμενη συσκευή ήρθε από τον kernel log. Διαβάζοντας τον Kernel Log κατά την εφαρμογή του driver εμφάνισε τα Vector ID, και Device ID, καθώς και κάποιες addresses οι οποίες έχουν επιλεχθεί να εμφανίζονται πληροφοριακά. Οπότε σε αυτό το σημείο, η συσκευή έχει γίνει Probe στο λειτουργικό σύστημα.

- Το λειτουργικό μας σύστημα έχει αναγνωρίσει την συσκευή μας, σε αυτό το σημείο πρέπει να δοκιμάσουμε τις συναρτήσεις που εκτελούν την επικοινωνία μέσω του διαύλου PCI Express. Για να γίνει αυτό, έπρεπε να αφήσουμε στην άκρη για λίγο την σχεδίαση του Software, και να κάνουμε παρεμβάσεις στην σχεδίαση του Hardware. Καταργήσαμε την εσωτερική μνήμη της FPGA και τα σήματα ελέγχου εγγραφής τα οδηγήσαμε σε καταχωρητές, την έξοδο των καταχωρητών σε LED μέσω του αρχείου .ucf. Με αυτόν τον τρόπο όποτε θα πραγματοποιούνταν κάποια προσπέλαση στην μνήμη της FPGA θα άναβαν τα LED.

- Αφού εκτελέσαμε την εφαρμογή και πραγματοποιήσαμε μια υποτιθέμενη εγγραφή στην μνήμη της FPGA, αυτό που περιμέναμε για να βεβαιωθούμε ότι στην FPGA πράγματι έφτασε μια εντολή μεταφοράς δεδομένων από τον χρήστη, ήταν να ανάψουν τα LED στη συσκευή. Τα LED άναψαν (υπάρχει και σχετικό video στο internet), οπότε η λειτουργία του Driver μαζί με την εφαρμογή σε επίπεδο χρήστη εικάζεται ότι λειτουργούν σωστά.

- Πλέον από την άποψη του software διαφαίνεται κάποιου είδους λειτουργία η οποία συμφωνεί με τα ζητούμενα. Σωστή λειτουργία δεν θεωρείται να έχει επιτευχθεί έως ότου στείλουμε δεδομένα στην FPGA και τα λάβουμε πίσω σωστά. Άρα στο επόμενο στάδιο, υλοποιούμε την σχεδίαση της FPGA. Όπως έχει αναλυθεί σε προηγούμενο κεφάλαιο, προβαίνουμε στις απαραίτητες αλλαγές και καταλήγουμε στην τελική σχεδόν σχεδίαση. Τα σήματα στο component my_black_box ενώνονται μεταξύ τους, δηλαδή η είσοδος δεδομένων συνδέεται με την έξοδο δεδομένων, και τα ρολόγια που τροφοδοτούν τις FIFO είναι ενωμένα με το ρολόι της κεντρικής σχεδίασης.

- Στο σημείο αυτό έχουμε ένα σύνολο από μέρη τα οποία, θεωρητικά ακόμα, έχουν την ικανότητα να μεταφέρουν δεδομένα από το επίπεδο χρήστη, με την βοήθεια του Linux Driver μέσω του διαύλου PCI Express στην FPGA. Αφού τα σήματα των FIFO είναι ενωμένα μεταξύ τους, αυτό σημαίνει ότι αν πραγματοποιηθεί αρχικά μια λειτουργία αποστολής δεδομένων στην FPGA τότε αν όλα δουλεύουν σωστά θα υπάρχουν δεδομένα να μεταφερθούν και από την FPGA στον χρήστη.

- Για να διαβεβαιώσουμε την αξιόπιστη μεταφορά δεδομένων προστέθηκε στην εφαρμογή σε επίπεδο χρήστη η μεθοδολογία του Polling καθώς πραγματοποιήθηκαν και οι απαιτούμενες αλλαγές σε επίπεδο Hardware.

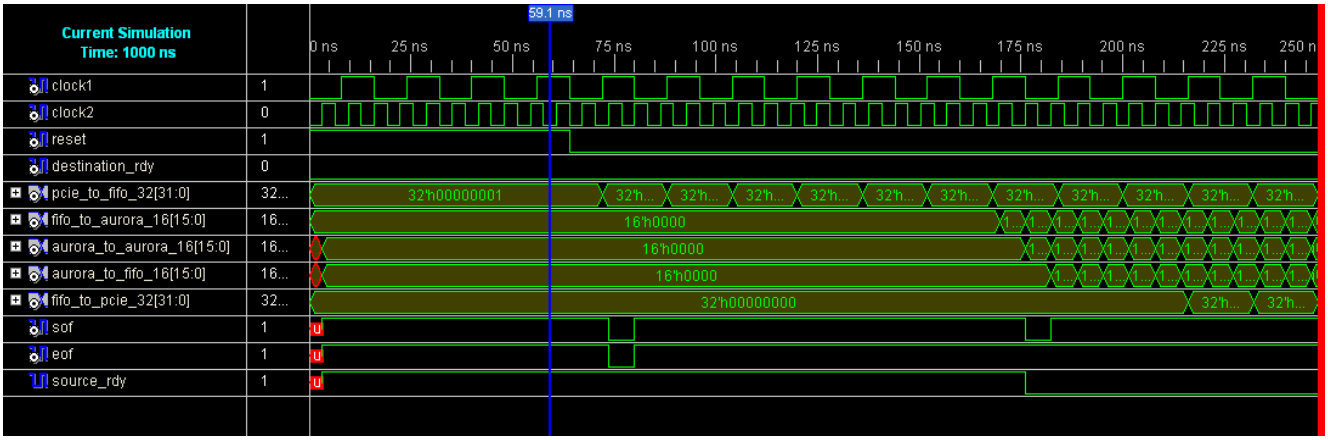
- Η εφαρμογή τέθηκε σε λειτουργία, και μετά από λίγο κόπο για αντιμετώπιση προβλημάτων τα οποία δεν είχαν υπολογιστεί, πετύχαμε την αξιόπιστη μεταφορά δεδομένων.

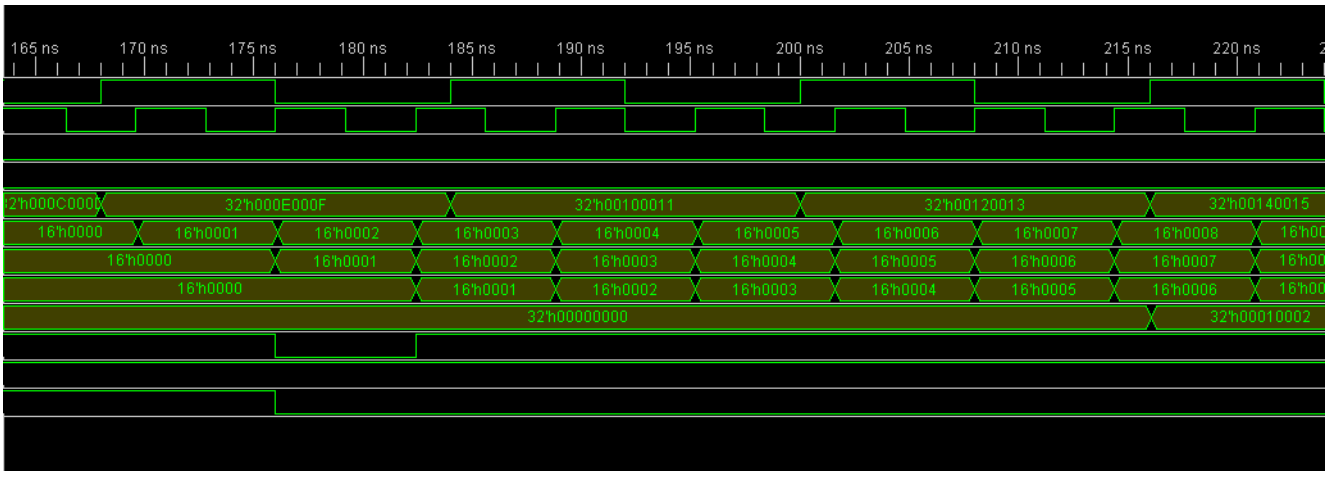
Να προσθέσουμε σε αυτό το σημείο ότι σε όλη την διάρκεια των δοκιμών ένας παλμογράφος μας ενημέρωνε για την σωστή λειτουργία των παραπάνω, καθώς είχαμε την δυνατότητα μέσω αυτού να παρατηρήσουμε τα σήματα εντός της FPGA όπως τα ρολόγια και τα enables εγγραφής και ανάγνωσης, καθώς και την χρονική αντιστοιχία των λειτουργιών. Τέλος σήματα flag χρησιμοποιήθηκαν σε διάφορα στάδια της υλοποίησης για να παρατηρηθούν διάφορες λειτουργίες που υποστηρίζει η σχεδίαση μας. Τα σήματα αυτά μέσω κατάλληλης μορφοποίησης του αρχείου .ucf οδηγούνται σε ελεύθερα pin της πλακέτας μας και έτσι υπήρχε η δυνατότητα να ελεγχθούν από παλμογράφο.

Σε αυτό το σημείο να σημειώσουμε ότι στην εξέλιξη της σχεδίασης για την επίτευξη του PCIe-to-Aurora Bridge θα μπορούσαμε να αναφέρουμε τις προσπάθειες που έγιναν, αλλά δεν θεωρούμε ότι υπάρχει νόημα να γίνει κάτι τέτοιο χωρίς να υπάρχει ένα θετικό αποτέλεσμα να παρουσιαστεί.

Simulation

Κάποια κομμάτια της σχεδίασης ελέγχθηκαν αρχικά σε επίπεδο simulation. Ένα από αυτά ήταν μια μηχανή πεπερασμένων καταστάσεων που προορίζονται να λάβει το ρόλο του ελεγκτή ο οποίος θα χρησιμοποιούνταν στο top level της σχεδίασης του PCIe-to-Aurora Bridge. Να σημειώσουμε, ότι ο ελεγκτής ακόμα υπάρχει στην υλοποίηση του PCIe-to-Aurora Bridge αλλά δεν έχουμε μπορέσει ακόμα να ελέγξουμε πρακτικά την λειτουργία του. Σε θεωρητικό επίπεδο όμως, αν εξαιρέσουμε τους Wrapper της σχεδίασης, επιτυγχάνουμε αξιόπιστη μεταφορά και μέσω του PCIe-to-Aurora Bridge.





Παραπάνω βλέπουμε το simulation του ελεγκτή που αναφέρθηκε παραπάνω. Στη συγκεκριμένη σχεδίαση:

- τυχαία δεδομένα (count) εισάγονται στην μνήμη FIFO που υπάρχει από την πλευρά του PCI Express (αυτό το έχουμε επιτύχει και στην υλοποίηση, καθώς η μεταφορά δεδομένων μέσω του PCI Express είναι εφικτή). Τα δεδομένα που προέρχονται από το PCI Express είναι 32 BIT, αλλά το Aurora δέχεται δεδομένα των 16 Bit. Οπότε η FIFO που χρησιμοποιούνται εκτός του ότι δουλεύουν με διαφορετικά ρολόγια ανάγνωσης και εγγραφής, έχουν και διαφορετικό μέγεθος δεδομένων.

- Έπειτα τα δεδομένα εξέρχονται από την FIFO και εισέρχονται στο component frame_TX το οποίο όπως έχουμε εξηγήσει σε προηγούμενο κεφάλαιο, παράγει τα πακέτα προς αποστολή και δίνει τιμές στα σήματα που χρησιμοποιούνται για την «χειραψία» (handshake) του πρωτοκόλλου Aurora.

- Στην συνέχεια το πακέτο λαμβάνεται από το πρωτόκολλο Aurora του παραλήπτη και οδηγείται στο component frame_RX, το οποίο όπως έχουμε εξηγήσει λαμβάνει τα πακέτα, και βγάζει σαν έξοδο τα ωφέλιμα δεδομένα.

- Τα δεδομένα αυτά μεταφέρονται στην FIFO και από 16Bit ποσότητες γίνονται 32Bit, οπότε τις λαμβάνει ο πυρήνας του PCI Express και τους μεταφέρει μέσω του Driver στον χρήστη.

Σε θεωρητικό επίπεδο όπως παρατηρούμε από τα παραπάνω η επικοινωνία μέσω του PCIe-to-Aurora Bridge είναι λειτουργική.

9. Αποτελέσματα – Μετρήσεις

Αποτελέσματα

Η αξιόπιστη μεταφορά δεδομένων από τον Η/Υ στην αναδιατασσόμενη συσκευή Virtex-5 μέσω του διαύλου PCI Express ήταν ο στόχος που θέλαμε να επιτεύξουμε. Κάτι τέτοιο έγινε δυνατό χρησιμοποιώντας τα προσφερόμενα σε εμάς μέσα.

Ένας στόχος ο οποίος ναι μεν υλοποιήθηκε αλλά δεν καταφέραμε να εξετάσουμε την λειτουργία του εκτός από εξομοίωση σε θεωρητικό και μόνον επίπεδο είναι η PCIe-to-Aurora Bridge. Όμως η φύση του προβλήματος στη συγκεκριμένη σχεδίαση, η τεχνογνωσία που έχει αποκτηθεί όσον αφορά τα πρωτόκολλα Aurora, η ολοκληρωμένη σχεδίαση της γέφυρας καθώς και οι μήνες δουλειάς πάνω στην υλοποίηση αυτή, κάνουν επιτακτική την ανάγκη για ολοκλήρωση και χρήση της, είτε λόγω συναισθηματικών παραγόντων (μήνες δουλειάς) είτε από άποψη μετέπειτα χρήσης από κάποιον συνάδελφο.

Ταχύτητα

Η ταχύτητα μεταφοράς δεδομένων ελέγχθηκε με την βοήθεια εφαρμογής η οποία βρίσκεται σε επίπεδο χρήστη. Δεδομένα μεγάλου όγκου (1Giga Byte) στάλθηκαν στην FPGA και έπειτα ελήφθησαν από την FPGA σε user level.

Η μεταφορά που πραγματοποιήθηκε να σημειώσουμε ότι δεν είναι ενδεικτική των δυνατοτήτων του διαύλου PCI Express, καθώς υπάρχουν πάρα πολλά σημεία βελτίωσης.

Η ταχύτητα που επιτεύχθηκε στο Transmit έφτασε τα 268,43MBps ενώ στο Receive περιορίστηκε στα 9Mbps.

Προφανώς υπάρχει πληθώρα βελτιώσεων που μπορούν να πραγματοποιηθούν σαν Μελλοντική εργασία όπως εφαρμογή Interupt Handler και κωδικοποίηση δημιουργία πακέτων μεγαλύτερα αυτών της στοιχειώδους ποσότητας (32Bit).