ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σύστημα Υπέρ Ταχείας Δειγματοληψίας και Επεξεργασίας Σήματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Π. Καρυώτης

Επιβλέπων : Νικόλαος Ουζουνογλου
Καθηγητής Ε.Μ.Π

Αθήνα,Νοέμβριος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σύστημα Υπέρ Ταχείας Δειγματοληψίας και Επεξεργασίας Σήματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Π. Καρυώτης

Επιβλέπων : Νικόλαος Ουζούνογλου
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Νοεμβρίου 2006.

.....	.....	.....
Ν.Ουζούνογλου	Π.Φράγκος	Δ. Κακλαμάνη
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Αν. Καθηγήτρια

Αθήνα, Νοέμβριος 2006

.....
Γεώργιος Π. Καρυώτης

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

Copyright © Γεώργιος Π. Καρυώτης , 2006.

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ο σκοπός αυτής της διπλωματικής εργασίας ήταν η ανάπτυξη ενός συστήματος υπέρ ταχείας δειγματοληψίας και επεξεργασίας σήματος. Το σύστημα που σχεδιάστηκε κάνει δειγματοληψία 100 MSps με ανάλυση 12 ψηφίων. Στη συνέχεια επεξεργάζεται σε πραγματικό χρόνο τα δεδομένα με συχνότητα 100 MHz.

Η επεξεργασία ουσιαστικά είναι η σάρωση του φάσματος συχνοτήτων του σήματος και η επιλογή των οκτώ συχνοτήτων με τη μεγαλύτερη ενέργεια. Οι οκτώ αυτές συχνότητες, αντιστοιχίζονται σε 8 κανάλια όπως αυτά ορίζονται από τις προδιαγραφές των ταλαντωτών με τους οποίους το σύστημα θα διασυνδεθεί στην έξοδο, και οδηγούνται μέσω της κατάλληλης διεπαφής σε αυτούς. Επίσης το σύστημα παρέχει τη δυνατότητα διασύνδεσης και με υπολογιστή μέσω USB διασύνδεσης.

Το σύστημα χρησιμοποιεί υλικά τεχνολογικής αιχμής όπως ο Α/Ψ μετατροπέας AD12401 της Analog Devices με ανάλυση δειγματοληψίας 12 ψηφίων και ταχύτητα 400 MSps και το FPGA της XILINX, της σειράς Virtex 4 – 35SX με δείκτη ταχύτητας -12, το οποίο παρέχει κορυφαίες επιδόσεις στην επεξεργασία σήματος. Επιπλέον η λειτουργία του έχει εξομοιωθεί σε εργαλεία υψηλής και εγγυημένης απόδοσης όπως είναι το Xilinx ISE 7.1.

Τέλος είναι σημαντικό να σημειωθεί ότι το σύστημα που υλοποιήθηκε είναι επεκτάσιμο γιατί χρησιμοποιεί σύγχρονα υλικά τα οποία δίνουν τη δυνατότητα για περαιτέρω επέκταση της επεξεργασίας., και πιθανές ανασχεδιάσεις στο firmware

Λέξεις Κλειδιά

Ψηφιακή Σχεδίαση , FPGA Virtex 4 , USB , VHDL , Υπέρ Ταχεία Δειγματοληψία , Επεξεργασία Σήματος.

Abstract

The aim of this diplomatic dissertation was the development of a system with high speed sampling rate and digital process of an analog signal. The system has been designed to perform data acquisition at 100 MSps with 12 bits resolution. Then it processes in real time the data with frequency 100 MHz.

The signal processing performed by the system is the monitoring of the frequency spectrum of the sampled signal and the extraction of the eight frequencies with the highest energies. Those eight frequencies are carefully matched to eight corresponding channels in accordance with the specifications of the oscillators that will be connected to the output of the system and will be controlled by it. Then the eight channels are fed to the oscillators via the implemented interface. The system provides the capability of communication with a personal computer via a USB interface.

The system uses state of the art components such as the A/D converter AD12401 of Analog Devices that provides a sampling rate of 400 MSps with 12 bits resolution and the FPGA created by XILINX, of the product line Virtex 4 – 35SX with speed grade -12 that has unique performance at digital signal processing functions. Moreover its operation has been simulated with tools of high precision and guaranteed outcome such as Xilinx ISE 7.1.

Last but not least, it is important to be stated that the designed system is fully upgradeable because it uses modern devices that give us the possibility for further extension of the signal processing architecture, and possible firmware updates.

KEY WORDS

Digital Design, FPGA Virtex 4, USB, VHDL, High Speed Sampling, Digital Signal Processing.

Ολοκληρώνοντας αυτή τη διπλωματική εργασία θα ήθελα να ευχαριστήσω τον Καθηγητή μου κύριο Νικόλαο Ουζούνογλου για την ανάθεση και την επίβλεψη της διπλωματικής καθώς και το Διδάκτωρ Αναστάσιο Σαλή για την ορθή καθοδήγησή και ουσιαστική βοήθεια που μου παρείχε καθόλη τη διάρκεια της εργασίας.

Γεώργιος Καρυώτης

Αθήνα, Νοέμβριος 2006.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ ΠΙΝΑΚΩΝ ΚΑΙ ΣΧΗΜΑΤΩΝ

Πίνακες :

Κεφάλαιο 2

Κεφάλαιο 3

<u>Σχήματα</u>

Κεφάλαιο 2

Κεφάλαιο 3

Κεφάλαιο 4

Κεφάλαιο 1 : Γενική Περιγραφή Συστήματος

Στόχος αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός ενός συστήματος υπέρ ταχείας (high speed) δειγματοληψίας και επεξεργασία σήματος. Το σύστημα κάνει δειγματοληψία με ρυθμό 100 Msamples/s και ανάλυση 12 bit και λειτουργεί στην συχνότητα 100 MHz. Τα βασικά μέρη του συστήματος είναι ένας μετατροπέας αναλογικού σήματος σε ψηφιακό, ένα κύκλωμα υλοποίησης του διακριτού μετασχηματισμού Φουριέ καθώς και κυκλώματα που κάνουν την διασύνδεση με το phase-locked loop (PLL) και με τον υπολογιστή. Στην συνέχεια παραθέτουμε μια σύνοψη των προδιαγραφών.

Το σύστημά αναλύει το αναλογικό σήμα σε ψηφιακό με δειγματοληψία στα 100 Msps με ανάλυση (resolution) 2^{12} διακριτών τιμών, στην συνέχεια επεξεργάζεται και αναλύει τα δεδομένα σε έναν διακριτό μετασχηματισμό Φουριέ. Η ουσιαστική επεξεργασία είναι ότι σαρώνει όλο το φάσμα συχνοτήτων του σήματος και διαλέγει τις οκτώ συχνότητες που μεταφέρουν το μεγαλύτερο μέρος της ενέργειας του σήματος και στη συνέχεια το οδηγούν σε οκτώ PLL Synthesizer. Μια προδιαγραφή που έχει δοθεί είναι η ευκρίνεια συχνότητας(frequency resolution) αποτελεσμάτων του ΔΜΦ (Διακριτού Μετασχηματισμού Φουριέ) που πρέπει να είναι 25 KHz, αυτό για να επιτευχθεί με το παρόν υλοποιημένο σύστημα απαιτεί 4096 σημεία μετασχηματισμού (transfer points) .

Οι οκτώ πρώτοι δείκτες - σημεία μετασχηματισμού με την μεγαλύτερη ενέργεια του σήματος μας ενδιαφέρουν και αυτές θα αποθηκεύονται σε μια μνήμη RAM, στην συνέχεια θα αποστέλλονται σε υπολογιστή με την χρήση USB διασύνδεσης και σε ένα phase-locked loop (PLL). Η διασύνδεση με το PLL πρέπει να πληροί τις εξής προδιαγραφές:

- Στο PLL θα αποστέλλονται μια διεύθυνση 4 bit, ένας διάδρομος δεδομένων 16 bit και ένα σήμα ενός bit για να ένδειξη ότι τα δεδομένα είναι έγκυρα.

- Στο διάδρομο δεδομένων μεταφέρονται οι αριθμοί των καναλιών ,δηλαδή οι αριθμοί –δείκτες των σημείων μετασχηματισμού του ΔΜΦ που έχουν την μεγαλύτερη ενέργεια. Αυτά τα κανάλια είναι ταξινομημένα κατά φθίνουσα σειρά σύμφωνα με την ενέργεια που αντιστοιχεί σε αυτά.
- Εάν κάποιο δεδομένο δεν είναι έγκυρο τότε στέλνουμε μια ειδική λέξη 16 bit για να παραμείνει ο δέκτης (PLL) ανενεργός .
- Τα δεδομένα θα μεταφέρονται σειριακά και η χρονική απόσταση μεταξύ 2 δεδομένων θα είναι 100 ns
- Τέλος στο PLL θα αποστέλλονται κάθε 20 μs νέα δεδομένα για μετάδοση .

Κεφάλαιο 2 : Επιλογή Υλικών

Η επιλογή των υλικών είναι σημαντική για ένα σύστημα που δουλεύει σε συχνότητα 100 MHz και οι προδιαγραφές του είναι απαιτητικές. Συγχρόνως όλα τα υλικά επιλέχτηκαν έτσι ώστε να μπορούμε μελλοντικά να επεκτείνουμε το κύκλωμά μας για να έχει δειγματοληπτικό ρυθμό 400 MSps και να μπορεί να λειτουργεί το σύστημα σε μεγαλύτερες συχνότητες από αυτή των 100 MHz, να μπορεί να φτάσει στα 400 MHz. Επίσης παρουσιάζουμε τα εργαλεία προσομοίωσης και σύνθεσης που χρησιμοποιήσαμε καθώς και την γλώσσα περιγραφής υλικού που επιλέξαμε. Στο τέλος αυτού του κεφαλαίου αναφερόμαστε περιληπτικά στις επιμέρους λειτουργικές οντότητες (HDL blocks) που έχουμε χρησιμοποιήσει.

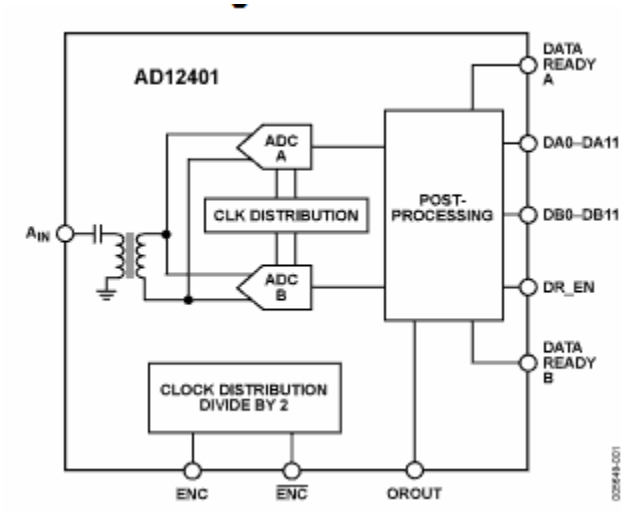
2.1 : Επιλογή Μετατροπέα Αναλογικού σήματος σε ψηφιακό

Η πρώτη επιλογή ήταν ο μετατροπέας αναλογικού σήματος σε ψηφιακό. Ο A/D Converter που ικανοποιούσε τις προδιαγραφές ήταν ο AD12401 της Analog Devices. Τα χαρακτηριστικά του AD12401 είναι:

Ανάλυση (Resolution)	**12 bit**
Μέγιστος Ρυθμός Δειγματοληψίας (Throughput Rate)	**400MSPS**
Μέγιστη Κατανάλωση Ισχύος	**6.8W**
Διεπαφή (Interface)	**LVDS , παράλληλα**

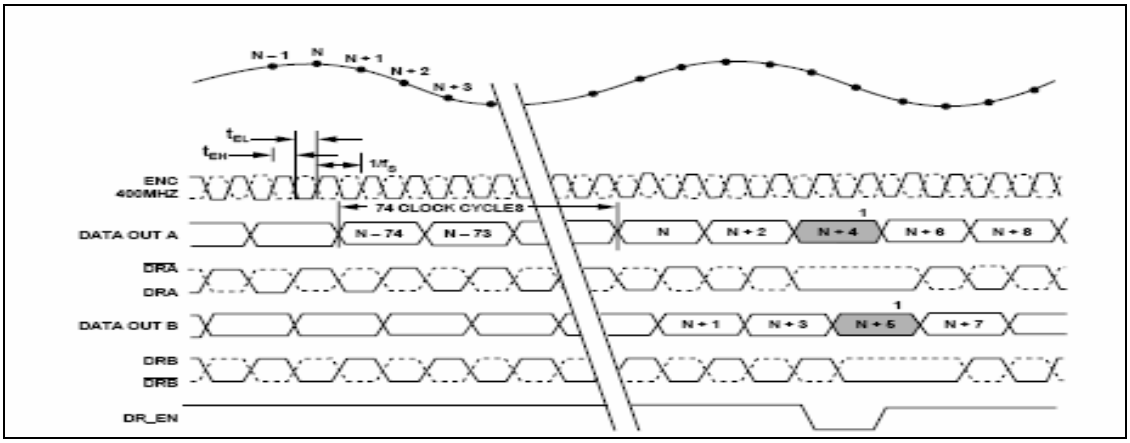
-ΠΙΝΑΚΑΣ 2.1-

Το λειτουργικό διάγραμμα του μετατροπέα είναι το εξής:



-ΣΧΗΜΑ 2.1-

Ο Α/Ψ μετατροπέας AD12401 έχει δύο θύρες εξόδου για δεδομένα τις Α και Β καθώς και δύο σήματα DATA_READYA και DATA_READYB τα οποία όποτε λαμβάνουν την λογική τιμή ‘1’ σημαίνει ότι αποστέλλουν νέα δεδομένα. Επειδή το κύκλωμα λειτουργεί μέχρι και 400 MHz τα σήματα εξόδου μπορούν να εξέρχονται με συχνότητα 200 MHz. Παρατίθεται και το διάγραμμα του χρονισμού του μετατροπέα:



-ΣΧΗΜΑ 2.2-

2.2 : Επιλογή FPGA

Για την υλοποίηση του συστήματος χρησιμοποιήσαμε δύο διαφορετικά FPGA το ένα ,το Virtex 4, είναι αφοσιωμένο με την λειτουργία του κυρίως συστήματος που επεξεργάζεται τα δεδομένα και το δεύτερο ,το Spartan3E , υλοποιεί το κύκλωμα διασύνδεσης του κυρίως κυκλώματος με τον USB microcontroller.

Το FPGA που χρησιμοποιήσαμε για να υλοποιήσουμε το ψηφιακό σύστημα επεξεργασίας και λήψης σήματος είναι το Virtex 4 της Xilinx και συγκεκριμένα το μοντέλο XC 4VSX35 -12 FF668. Το Virtex 4 προσφέρει στους σχεδιαστές μια μεγάλη ποικιλία από πλατφόρμες FPGA που ικανοποιούν διαφορετικές απαιτήσεις. Εμείς προτιμήσαμε την SX πλατφόρμα που επιτρέπει υψηλών επιδόσεων (high performance) λύσεις για εφαρμογές ψηφιακής επεξεργασίας σήματος (DSP). Ένα μεγάλο πλεονέκτημα του FPGAVirtex 4 είναι ότι μπορεί να επεξεργαστεί δεδομένα μέχρι και σε συχνότητα 500 MHz. Επίσης προσφέρει μια μεγάλη ποικιλία στον τρόπο διασύνδεσης του FPGA με εξωτερικές συσκευές όπως LVTTL , LVCMOS ,PCI, LVDS, Hypertransport. Στο σύστημα μας επιθυμούσαμε επεξεργασία δεδομένων σε συχνότητα 100 MHz, αλλά διαλέξαμε την πλατφόρμα με βαθμό ταχύτητας (speed grade) -12, που φανερώνει ότι το FPGA επεξεργάζεται τα δεδομένα με ταχύτητα 500 MHz επειδή θέλαμε όλες οι λειτουργικές μονάδες που απαρτίζουν το κύκλωμα να λειτουργούν στα 100 MHz. Επιπλέον το μοντέλο XC 4VSX35 -12 FF668 μας παρέχει 448 πύλες εισόδου – εξόδου που είναι ένας ικανοποιητικός αριθμός για το σύστημά μας . Στην συνέχεια παρουσιάζουμε τα χαρακτηριστικά του XC 4VSX35 :

Χαρακτηριστικά XC 4VSX35	
Logic Cells	**34,560**
Block RAM/FIFO w/ECC (18 kbits each)	**192**
Total Block RAM (kbits)	**3,456**

Digital Clock Managers (DCM)	**8**
Phase-matched Clock Dividers (PMCD)	**4**
Max Differential I/O Pairs	**224**
XtremeDSP™ Slices	**192**
Configuration Memory Bits	**14,476,608**
Max SelectIO	**448**

--ΠΙΝΑΚΑΣ2.2-

Όλα τα παραπάνω φανερώνουν ότι το Virtex 4 έχει δυνατότητες που το κάνουν κατάλληλο για την υλοποίηση του προς κατασκεύη συστήματος .

2.3 : Επιλογή USB μικροελεγκτή

Στην συνέχεια επιλέξαμε το CY7C68013A ως USB μικροελεγκτή (microcontroller) της εταιρίας CYPRESS PERFOM. Ο CY7C68013A USB μικροελεγκτής ανήκει στην κατηγορία **EZ-USB FX2LP** και υποστηρίζει μεταφορά δεδομένων τόσο στην πλήρη ταχύτητα (Full-Speed 12 Mbits/sec) όσο και στην υψηλή ταχύτητα (High-Speed 400 Mbits/sec). Οι USB 2.0 μικροελεγκτές της κατηγορίας EZ-USB FX2LP είναι χαμηλής κατανάλωσης και υψηλής ολοκλήρωσης. Σε ένα μόνο ολοκληρωμένο τσιπ υπάρχει USB 2.0 μεταγωγέας (**USB 2.0 transceiver**), μια σειριακή μηχανή διεπαφής (**serial interface engine ,SIE**), 8051 μικροεπεξεργαστής (**enhanced 8051 microprocessor**) και περιφερειακές μονάδες διεπαφής που εύκολα προγραμματίζονται. Η σειριακή μηχανή διεπαφής (SIE) χειρίζεται το μεγαλύτερο μέρος των πρωτοκόλλων 1.1 και 2.0 USB στο υλικό και έτσι ο μικροεπεξεργαστής 8051 είναι ελεύθερος για υλοποιήσεις άλλων λειτουργιών. Επίσης η γενικού προγραμματισμού διεπαφή (General Programmable Interface

GPIF) διευκολύνει τη σύνδεση με ποικιλία διεπαφών όπως ATA ,UTOPIA ,EPP ,PCMCIA .

Στην συνέχεια παραθέτουμε μερικά χαρακτηριστικά των **EZ-USB FX2LP** μικροελεγκτών και ειδικότερα του CY7C68013A :

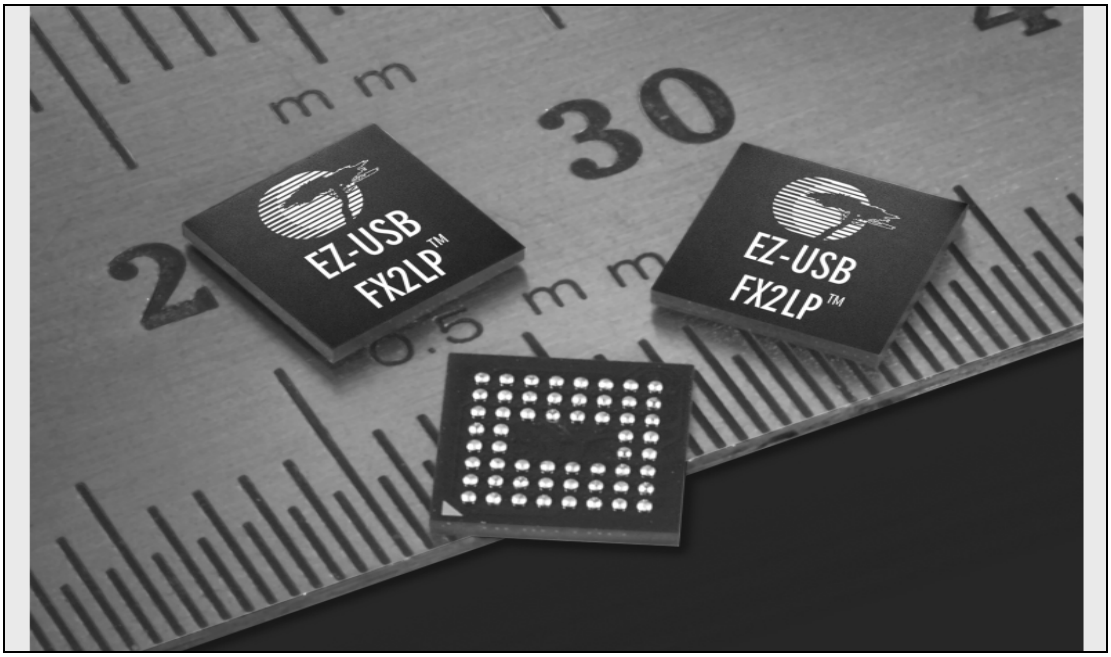
Χαρακτηριστικά:

- **Συμβατό με τις προδιαγραφές του USB 2.0 –USB-IF high speed**
- **Σε ένα ολοκληρωμένο ύπαρξη USB 2.0 μεταδότη –μεταγωγέα (transceiver) , έξυπνο SIE , ενσωματωμένος 8051 μικροεπεξεργαστής (microprocessor)**
- **Εξαιρετικά χαμηλή τροφοδοσία (Ultra Low Power)**
- **16 Kbytes στο ολοκληρωμένο για κώδικα / δεδομένα RAM**
- **Σημεία προσαρμογής για κανάλια επικοινωνίας για BULK /INTERRUPT / ISOCHRONOUS μεταφορές δεδομένων**
- **Ένα επιπλέον σημείο προσαρμογής 64-byte για BULK /INTERRUPT**
- **Ύπαρξη GPIF (General Programmable Interface)**
- **Ολοκληρωμένο , 8051**
  - **Η κεντρική μονάδα λειτουργεί (CPU)σε 48-MHz, 24-MHz , 12-MHz**
    - **4 κύκλοι ρολογιού και κάθε κύκλο εντολής**
      - **Τρείς μετρητές / χρονιστές**
        - **Αυξημένο σύστημα διακοπών**
          - **Δύο δείκτες δεδομένων**

- **Λειτουργία στα 3.3 Volt με ανοχή σε 5 Volt εισόδους**
- **Ξεχωριστοί απομονωτες (buffers) δεδομένων για εγκατάσταση (Set-up) και για δεδομένα μεταφορας τύπου Control**

--ΠΙΝΑΚΑΣ 2.3-

Παραθέτουμε και μια φωτογραφία του ολοκληρωμένου EZ-USB FX2LP:



-ΣΧΗΜΑ 2.3-

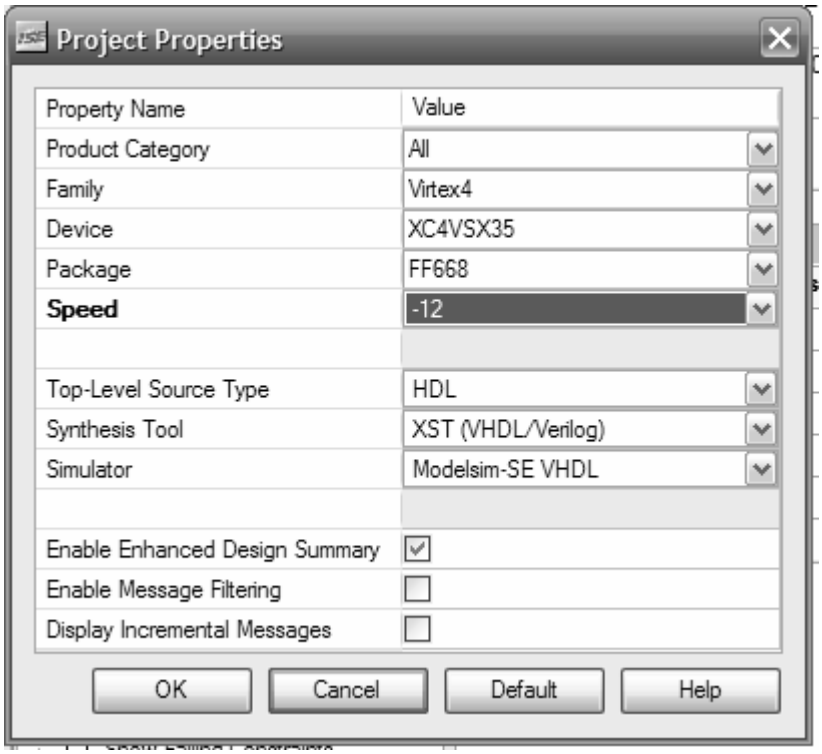
2.4 :Εργαλεία Προσομοίωσης

Η εταιρία Xilinx προσφέρει το κατάλληλο λογισμικό για το σχεδιασμό ,την προσομοίωση και τον προγραμματισμό του FPGA Virtex-4. Το λογισμικό Xilinx ISE (Integrated Software Environment) ενσωματώνει όλα τα εργαλεία που χρειάζονται για την λογική – ψηφιακή σχεδίαση όλων των FPGA που παράγει η Xilinx . Εμείς χρησιμοποιήσαμε την έκδοση 8,1i του Xilinx ISE. Το ISE 8,1i προσφέρει τις εξής δυνατότητες:

- Σύνθεση του ψηφιακού κυκλώματος(Design Synthesis) που γράφεται σε γλώσσα περιγραφής υλικού (HDL)
- Εφαρμογή του σχεδιασμού του ψηφιακού κυκλώματος (Design Implementation)
- Προσομοίωση συμπεριφοράς(Behavioral Simulation)
- Προσομοίωση λειτουργική (Functional Simulation)
- Στατική χρονική ανάλυση (Static Timing Analysis)
- Προσομοίωση χρονισμού (Timing Simulation)

- Προγραμματισμός Xilinx συσκευής , FPGA (Xilinx Device Programming)
- Κυκλωματική επαλήθευση του σχεδιασμού ,(In-circuit Verification)

Όπως φαίνεται και στην παρακάτω εικόνα που παρουσιάζει τις ιδιότητες της εργασίας που έχουμε υλοποιήσει, για εργαλείο σύνθεσης (Synthesis Tool) έχουμε χρησιμοποιήσει το XST, ενώ για προσομοιωτή το Modelsim-SE. Επίσης φαίνεται και ποιο FPGA έχουμε επιλέξει , το XC 4VSX35 -12FF668.



-ΣΧΗΜΑ 2.4-

Το XST (Xilinx Synthesis Technology) είναι το κύριο εργαλείο του ISE για να κάνει σύνθεση και βελτιστοποίηση του ψηφιακού κυκλώματος που έχει σχεδιαστεί στο ISE. Επιπλέον παρέχει την τεχνική σύνθεσης ‘‘εξισορρόπησης – καταχωρητών’’ (Register Balancing), του οποίου ο στόχος είναι να καλυφθούν οι απαιτήσεις για συγχρονισμό του ψηφιακού κυκλώματος με τη μετακίνηση των καταχωρητών είτε προς τα εμπρός είτε προς τα πίσω σε σχέση με τη λογική του

κυκλώματος για να αυξηθεί η συχνότητα των ρολογιών. Μια ακόμα δυνατότητα του XST είναι ότι εμφανίζει το σχηματικό της RTL (Register Transfer Level)σύνθεσης.

Το ModelSim SE είναι ένα προϊόν της εταιρίας ModelSim για εξομοίωση (simulation) και διόρθωση (debug) γλωσσών περιγραφής υλικού, που συνδυάζει υψηλές επιδόσεις και ένα πολύ καλό γραφικό περιβάλλον. Συγκεκριμένα χρησιμοποιήσαμε την έκδοση 6.1b για την εμφάνιση και τη μελέτη των αποτελεσμάτων της προσομοίωσης του κυκλώματος που κατασκευάσαμε στο ISE.

Είναι σημαντικό να σημειώσουμε ότι η γλώσσα περιγραφής υλικού (hardware description language ,HDL) που χρησιμοποιήσαμε για να μοντελοποιήσουμε το σύστημά μας είναι η VHDL. Με τη χρήση της VHDL τα κυκλώματα μπορούν να περιγραφούν σε υψηλό επίπεδο χρησιμοποιώντας αλγόριθμους αλλά και σε χαμηλό επίπεδο, αυτό των λογικών πυλών. Επιπλέον μπορεί να περιγράψει την συμπεριφορά ενός σύγχρονου ή ακόμα και ασύγχρονου ψηφιακού κυκλώματος. Ένα κύκλωμα μπορεί να μοντελοποιηθεί σαν ιεραρχική σύνθεση πολλών υποκυκλωμάτων. Επίσης με χρήση της γλώσσας αυτής μπορούν να παραχθούν κυματομορφές των σημάτων του κυκλώματος που περιγράφεται . Όλα τα παραπάνω συνθέτουν μια κατανοητή περιγραφή των ψηφιακών κυκλωμάτων, μέσω της οποίας μπορούμε να κάνουμε προσομοίωση του κυκλώματος.

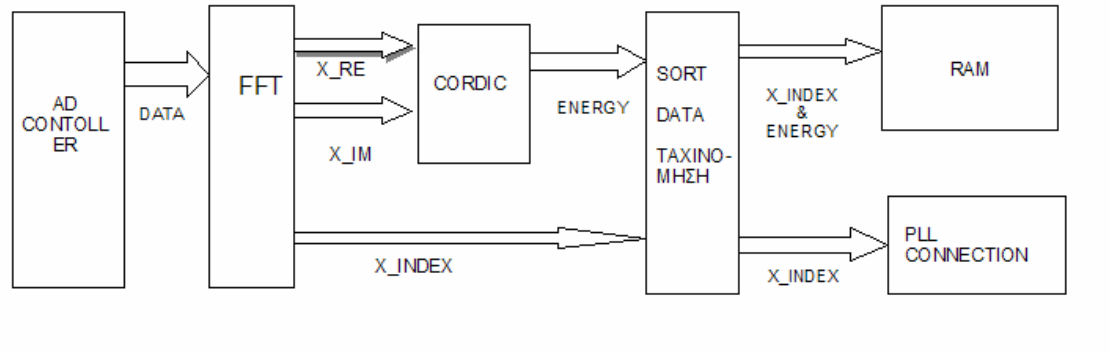
Η VHDL υποστηρίζει τους εξής τρείς βασικούς και διαφορετικούς τρόπους περιγραφής

- Δομική (Structural)
- Ροής Δεδομένων (Dataflow)
- Συμπεριφοράς (Behavioral)

Εμείς προτιμήσαμε τη δομική περιγραφή για να περιγράψουμε το κυρίως κύκλωμά, και μια μίξη των τριών διαφορετικών τρόπων στα επιμέρους υποκυκλώματα.

2.5 : Καταμερισμός Λειτουργιών Σε Λειτουργικές Μονάδες

Για να υλοποιηθεί το σύστημα λήξης και επεξεργασίας σήματος στα 100 Msamples/s με τις προδιαγραφές που αναφέραμε, έπρεπε να διαιρέσουμε το κύκλωμα μας σε επιμέρους υποκυκλώματα-μονάδες (components). Αυτή η υποδιαίρεση διευκολύνει τόσο τον προγραμματισμό όσο την υλοποίηση και σύνθεση του κυκλώματος . Η αρχική ιδέα ήταν να υλοποιήσουμε τις μονάδες που φαίνονται στο παρακάτω σχέδιο:



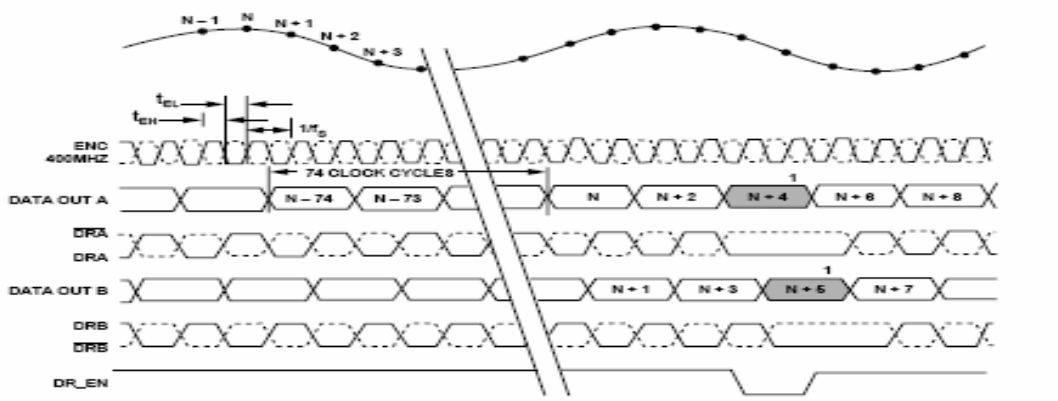
-ΣΧΗΜΑ 2.5-

Το βασικό μέρος του κυκλώματος είναι ο διακριτός μετασχηματισμός Φουριέ (ΔΜΦ , DFT). Σε αυτόν στέλνουμε τα δεδομένα από το μετατροπέα αναλογικού σήματος σε ψηφιακό για την περαιτέρω ανάλυση του σήματος. Ο ΔΜΦ αναλύει το ψηφιακό σήμα στις συχνότητες που το συνθέτουν. Στο κύκλωμά μας κατασκευάσαμε το διακριτό μετασχηματισμό Φουριέ με την βοήθεια του ISE που παρέχει το κύκλωμα για τον γρήγορο μετασχηματισμό Φουριέ (Fast Fourier transform ,FFT), ο οποίος είναι ουσιαστικά ένας γρήγορος και αποδοτικός αλγόριθμος για την υλοποίηση του ΔΜΦ.

Όμως είναι απαραίτητα και άλλα υποκυκλώματα που στόχο έχουν είτε την επιπλέον επεξεργασία του σήματος είτε την εσωτερική επικοινωνία μεταξύ των

υποκυκλωμάτων και την εξωτερική του επικοινωνία του FPGA με τον Α/Ψ μετατροπέα, τον PLL συνθέτη και τον USB μικροελεγκτή. Μερικές εσωτερικές διεπαφές δεν φαίνονται στο παραπάνω σχήμα και αυτό γιατί δεν ήταν στο αρχικό σχεδιασμό αλλά ήταν αναγκαία για τη σωστή λειτουργία του ψηφιακού συστήματος.

Η πρώτη μονάδα (block , component) είναι ο adcontroller. Σκοπός αυτής της μονάδας είναι η μεταφορά δεδομένων από τον Α/Ψ μετατροπέα στο κύκλωμα του FFT στην σωστή σειρά. Ο Α/Ψ μετατροπέας έχει την δυνατότητα να στέλνει τα δεδομένα από δύο διαφορετικά κανάλια ,τα οποία δεν είναι συγχρονισμένα, με συχνότητα 200 MHz. Στο κύκλωμα θα ορίσουμε να στέλνονται τα δεδομένα με συχνότητα 50 MHz ώστε να είναι δυνατή η επεξεργασία αυτών με συχνότητα 100MHz.



-ΣΧΗΜΑ 2.6-

Έτσι ο FFT λαμβάνει τα δεδομένα από ένα κανάλι με συχνότητα 100 MHz ,δηλαδή με περίοδο 10 nsec.

Μετά από τον Α/Ψ μετατροπέα και τον FFT είναι το κύκλωμα που μετατρέπει το πραγματικό μέρος και το φανταστικό μέρος των αποτελεσμάτων από αριθμούς της μορφής συμπλήρωμα ως προς 2 σε μορφή 1QN. Αυτό το κύκλωμα είναι αναγκαίο γιατί το cordic δέχεται τα δεδομένα σε 1QN μορφή και όχι σε μορφή συμπλήρωμα ως προς 2. Στην συνέχεια το cordic λαμβάνει τα δεδομένα και μας δίνει την ενέργεια που αντιστοιχεί σε κάθε εύρος συχνοτήτων, δηλαδή σε κάθε σημείο μετασχηματισμού του ΔΜΦ.

Επειδή επιθυμούμε τις οχτώ συχνότητες με το μεγαλύτερο πλάτος-ενέργεια που εμφανίζονται στο φάσμα του αναλογικού συστήματος, οι οποίες πρέπει να είναι απαλλαγμένες από το θόρυβο, κατασκευάσαμε το υποκύκλωμα ‘‘ταξινόμηση’’ (component taxinomisi_reg). Στόχος του υποκυκλώματος ‘‘ταξινόμηση ’’ είναι να βρει τα σημεία του μετασχηματισμού με τις οχτώ μεγαλύτερες ενέργειες οι οποίες να μην αντιστοιχούν στα επίπεδα του θορύβου που εμείς έχουμε ορίσει και να τις στέλνει στις επόμενες μονάδες του κυκλώματος, την ‘‘PLL_SYNDESI’’ και την “regist”. Η μονάδα ‘‘PLL_SYNDESI’’ εξυπηρετεί ουσιαστικά τη διασύνδεση του FPGA με το PLL συνθέτη (PLL synthesizer) με τις κατάλληλες προδιαγραφές. Ενώ το υποκύκλωμα ‘‘regist’’ λαμβάνει τους οχτώ παράλληλους καταχωρητές από την προηγούμενη μονάδα και τους αποστέλλει σειριακά στην επόμενη μονάδα την ‘‘no_negative_regs’’ για να απομονωθούν τα έγκυρα δεδομένα. Αυτά τελικά θα σταλθούν για αποθήκευση σε μια μνήμη RAM που μας παρέχεται από το ISE η οποία ελέγχεται από την μονάδα ‘‘Control_Of_Ram’’.

Για την επικοινωνία του συστήματός μας με τον USB μικροελεγκτή κατασκευάσαμε ένα επιπλέον ανεξάρτητο με το προηγούμενο κύκλωμα σε FPGA Spartan 3E. Το κύκλωμα αυτό ονομάζεται ‘‘Master_Mode’’ και ουσιαστικά λαμβάνει τα δεδομένα από το σύστημά μα ς και με κατάλληλα σήματα έλεγχου και κατάλληλη μορφοποίηση των δεδομένων τα αποστέλλει στο USB μικροελεγκτή CY7C68013A.

Κεφάλαιο 3 : Περιγραφή όλων των Λειτουργικών Μονάδων

Στο κεφάλαιο ¨Καταμερισμός Λειτουργιών Σε Υποκυκλώματα ¨ παρουσιάσαμε περιληπτικά τις επιμέρους λειτουργικές οντότητες του συστήματος λήξης και επεξεργασίας σήματος στα 100 Msamples/s των 12 bit. Στη συνέχεια θα παρουσιάσουμε αυτές τις μονάδες εκτενώς , περιγράφοντας όλα τα χαρακτηριστικά τους , εισόδους , εξόδους , τρόπος λειτουργίας , προβλήματα που βρήκαμε κατά την διάρκεια της σχεδίασης καθώς και τα διαγράμματα χρονισμού. Επίσης παραθέτουμε και τη μέγιστη συχνότητα λειτουργίας κάθε υποκυκλώματος σύμφωνα με το ‘‘Synthesi Report ’’ του Xilinx ISE .

3.1: Project_100Msps

Το project_100MSps είναι το τελικό ψηφιακό κύκλωμα που κατασκευάσαμε . Η περιγραφή που επιλέξαμε είναι η δομική (Structural). Το τελικό κύκλωμα περιλαμβάνει όλες τις λειτουργικές μονάδες (HDL blocks) που σχεδιάσαμε , οι οποίες διασυνδέονται με τη σωστή σειρά .

Χαρακτηριστικά

Μέγιστη Συχνότητα Λειτουργίας 112.633MHz

Είσοδοι

Όνομα εισόδου	**Μέγεθος**	**Περιγραφή**
clk_p	1	Ρολόι , το πραγματικό ψηφίο
clk_n	1	Ρολόι , το συμπληρωματικό ψηφίο.
dra_p	1	Σήμα για υπόδειξη ότι το κανάλι Α είναι έτοιμο ,πραγματική τιμή

dra_n	1	Κανάλι Α έτοιμο , συμπληρωματικό ψηφίο
drb_p	1	Κανάλι Β έτοιμο , πραγματικό ψηφίο
drb_n	1	Κανάλι Β έτοιμο , συμπληρωματικό ψηφίο
dat_a_p	12	Διάδρομος δεδομένων του Α καναλιού , πραγματικά ψηφία
dat_a_n	12	Διάδρομος δεδομένων του Α καναλιού , συμπληρωματικά ψηφία
dat_b_p	12	Διάδρομος δεδομένων του Β καναλιού , πραγματικά ψηφία
dat_b_n	12	Διάδρομος δεδομένων του Β καναλιού , συμπληρωματικά ψηφία
start_in	1	Σήμα έναρξης του FFT,πραγματικό ψηφίο

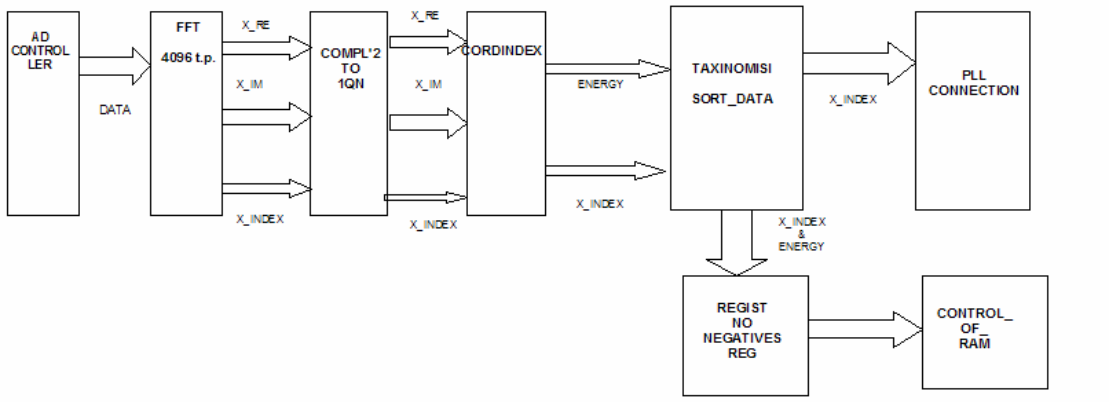
Έξοδοι

Όνομα εξόδου	**Μέγεθος**	**Περιγραφή**
address	4	Η διεύθυνση που δείχνει πιο κανάλι είναι στην έξοδο
dat_out	16	Το κανάλι εισόδου
dat_val	1	Σήμα πληροφόρησης ότι τα δεδομένα είναι έτοιμα και έγκυρα

d_out	27	Η σειριακή έξοδος των δεδομένων από την B πόρτα της Ram
enb	1	Σήματα πληροφόρησης ότι τα δεδομένα είναι έτοιμα και έγκυρα
p_en	1	

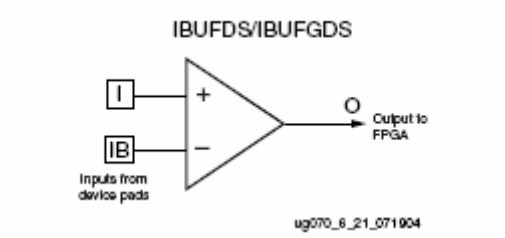
-ΠΙΝΑΚΑΣ 3.1-

Το κύκλωμα 'project_100MSps' υλοποιεί όλη την επιθυμητή λειτουργία του συστήματός μας. Λαμβάνει τα δεδομένα από τον Α/Ψ μετατροπέα και τα επεξεργάζεται σύμφωνα με τις προδιαγραφές του συστήματος. Ουσιαστικά το project_100MSps υλοποιεί το εξής κύκλωμα :

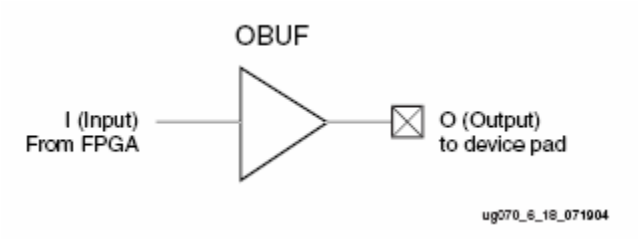


-ΣΧΗΜΑ 3.1-

Για την εύρυθμη λειτουργία του συστήματος προσθέσαμε τόσο στην είσοδο όσο και στην έξοδο ειδικούς απομονωτές (buffers) . Οι απομονωτές εισόδου είναι οι IBUFGDS και IBUFDS , ως είσοδο έχουν ένα διαφορικό ζευγάρι ψηφίων και έξοδο το σήμα που θα χρησιμοποιηθεί στο κύκλωμα. Το IBUFGDS είναι ο κατάλληλος απομονωτής για τη διασύνδεση του ρολογιού. Στην έξοδο χρησιμοποιήσαμε τους απομονωτές OBUF ,οι οποίοι έχουν μία είσοδο και μία έξοδο .Παρουσιάζουμε τα σχηματικά τους :



-ΣΧΗΜΑ 3.2α-



--ΣΧΗΜΑ 3.2β-

Πρέπει να προσθέσουμε ότι οι απομονωτές εισόδου διασυνδέονται με τον Α/Ψ μετατροπέα μέσω της τεχνολογίας LVDS (**Low voltage differential signaling**). Η Xilinx παρέχει εσωτερικό τερματισμό στην περίπτωση που η τροφοδοσία εξόδου του FPGA Virtex-4 η VCCO ισούται με 2.5 Volt , στην περίπτωση μας η VCCO ισούται με 3,3 Volt άρα πρέπει να χρησιμοποιήσουμε εξωτερικό τερματισμό .Χρησιμοποιήσαμε LVDS-διαφορική είσοδο επειδή τα δεδομένα που εισέρχονται από τον Α/Ψ μετατροπέα χρησιμοποιούν αυτήν την διασύνδεση. Αντίθετα οι απομονωτές εξόδου χρησιμοποιούν τεχνολογία LVCMOS η οποία έχει μονή έξοδο (single ended output). Δεν χρειάζεται διαφορική έξοδο γιατί τα δεδομένα εξέρχονται με αργό ρυθμό. Τα δεδομένα προς τον PLL Synthesizer εξέρχονται με ρυθμό 10 MHz ενώ προς το κύκλωμα του USB Controller με ρυθμό 40 MHz .

3.2 : Κύκλωμα που ελέγχει τα δεδομένα του Α/Ψ Μετατροπέα

Η πρώτη λειτουργική οντότητα είναι το υποκύκλωμα ¨ ad_contoller ¨ .Όπως έχουμε παραθέσει σε προηγούμενη παράγραφο της εργασίας, σκοπός αυτού του κυκλώματος είναι η μεταφορά των δεδομένων στην κατάλληλη σειρά. Τα χαρακτηριστικά του παραθέτονται παρακάτω :

Χαρακτηριστικά

Μέγιστη Συχνότητα Λειτουργίας 400 MHz

Είσοδοι :

clk :κεντρικό ρολόι

dready_A ,dready_B: δεδομένα έτοιμα για λήψη

dat_a, dat_b: τα δεδομένα εισόδου

Έξοδοι :

datout : τα δεδομένα εξόδου

-ΠΙΝΑΚΑΣ 3.2α-

Προδιαγραφές Κυκλώματος

Οι τέσσερις είσοδοι λαμβάνουν δεδομένα από τον Α/Ψ μετατροπέα με ρυθμό 50 MHz Διασύνδεση LVDS

Η έξοδος datout αποστέλλει τα παράλληλα δεδομένα εισόδου dat_a, dat_b σειριακά με ρυθμό 100 MHz . Επίσης το datout τροποποιείται κατάλληλα για να είναι 14 ψηφία .

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

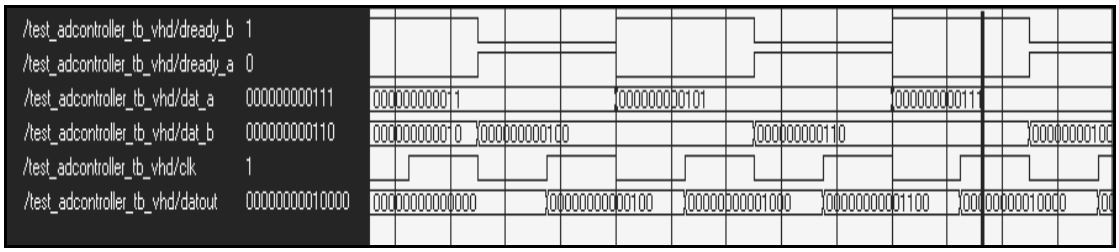
-ΠΙΝΑΚΑΣ 3.2β-

Σε κάθε αρνητικό παλμό του dready_A και του dready_B εισέρχονται νέα δεδομένα από το Α/Ψ μετατροπέα AD12401 , τα dat_a και dat_b αντίστοιχα. Τα dready_A, dready_B είναι τα σήματα που ενημερώνονται για την είσοδο νέων δεδομένων. Στο σύστημά μας εφαρμόζουμε ρολόι 100 MHz στο Α/Ψ μετατροπέα AD12401 οπότε θα εξέρχονται νέα δεδομένα από τις δύο θύρες δεδομένων dat_a και dat_b κάθε 50 MHz .

Η έξοδος datout είναι 14 bit και τα νέα δεδομένα εξέρχονται με κάθε θετικό παλμό του κεντρικού ρολογιού .

Το ‘‘ad_contoller’’ είναι σχεδιασμένο με τον τρόπο περιγραφής συμπεριφοράς. Αποτελείται από μια διεργασία η οποία σε κάθε θετικό παλμό του ρολογιού clk ελέγχει τα σήματα dready_A και dready_B. Εάν το dready_A είναι στο λογικό επίπεδο ‘1’ τότε στην έξοδο αποστέλλονται τα δεδομένα από την θύρα dat_a αλλιώς αποστέλλονται τα δεδομένα από την θύρα dat_b. Θα πρέπει να σημειώσουμε εδώ ότι κάνουμε μια μετατροπή των δεδομένων από 12 bit σε 14 bit προσθέτοντας δύο επιπλέον μηδενικά στην αρχή των δυαδικών αριθμών , δηλαδή στα δύο λιγότερα σημαντικά ψηφία. Προτιμήσαμε αυτήν την επιλογή έναντι της επιλογής να τοποθετήσουμε δύο μηδενικά στα πιο σημαντικά ψηφία του νέου αριθμού , γιατί έτσι αλλοιώνεται λιγότερο ο νέος αριθμός –δεδομένο .

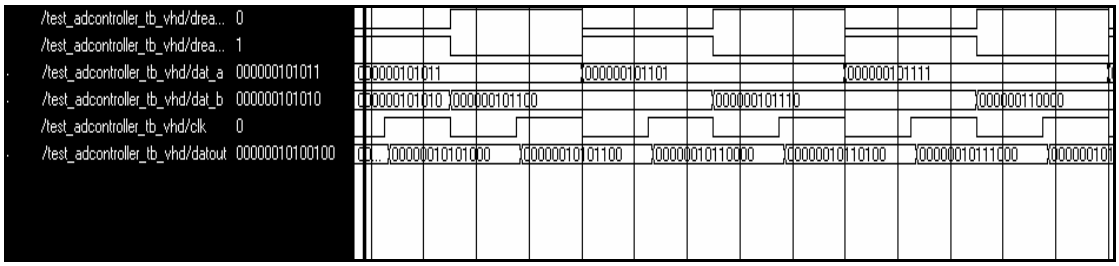
Στην επόμενη εικόνα παραθέτουμε τα αποτελέσματα από την προσομοίωση της συμπεριφοράς του κυκλώματος (behavioral simulation):



-ΣΧΗΜΑ 3.3-

Όπως παρατηρούμε στην είσοδο dat_a εισέρχονται περιττοί αριθμοί ενώ στην είσοδο dat_b εισέρχονται άρτιοι αριθμοί και στην έξοδο datout έχουμε σε κάθε θετικό παλμό του clk νέα δεδομένα. Για κάθε ζεύγος dat_a, dat_b πρώτα εξέρχεται το dat_a, και μετά το dat_b , όπως θα έπρεπε . Όμως από την προσομοίωση της συμπεριφοράς του κυκλώματος δεν προκύπτει η πραγματική συμπεριφορά του , αλλά προκύπτει από την προσομοίωση του κυκλώματος μετά τη σύνθεση και την εφαρμογή του σχεδιασμού στο FPGA. Στο επόμενο διάγραμμα χρονισμού

παρατηρούμε ότι τα δεδομένα εξέρχονται τη σωστή χρονική στιγμή , σε κάθε θετικό παλμό του κεντρικού ρολογιού (clk)και με τη σωστή σειρά .



-ΣΧΗΜΑ 3.4-

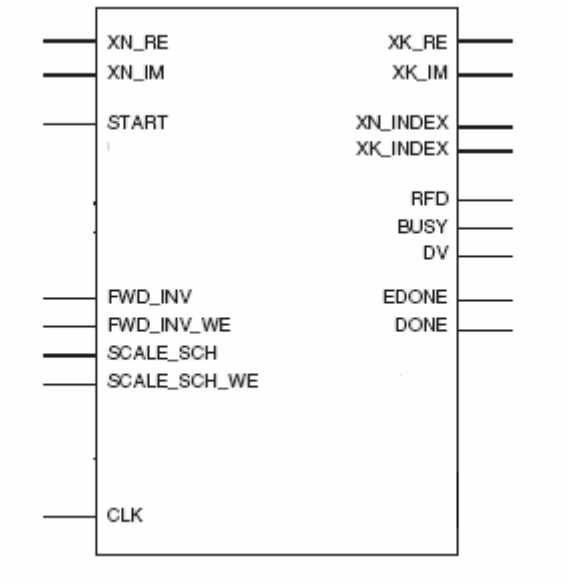
3.3 : Κύκλωμα Διακριτού Μετασχηματισμού Φουριε :

Όπως έχουμε παραθέσει προηγουμένως το Xilinx ISE παρέχει την υλοποίηση του ΔΜΦ. Η λειτουργική μονάδα του Γρήγορου Μετασχηματισμού Φουριε , FFT , υλοποιεί τον ΔΜΦ και στηρίζεται στον αλγόριθμο Cooley –Tukey. Ο Γρήγορος Μετασχηματισμός Φουριέ του ISE παρέχει ένα πλήθος από επιλογές σχετικά με τις κυκλωματικές ιδιότητές του, τις οποίες δείχνουμε παρακάτω:

Ιδιότητες Μονάδας Γρήγορου Μετασχηματισμού Φουριέ
Υλοποιεί και τον κανονικό Γρήγορου Μετασχηματισμού Φουριέ και τον Αντίστροφο Γρήγορου Μετασχηματισμού Φουριέ
Το μήκος του μετασχηματισμού (σημεία μετασχηματισμού) $N=2^m$,m =3-16
Ακρίβεια δεδομένων b_x=8-24
Ακρίβεια της φάσης b_w=8-24
Διαθέτει συγκεντρωμένη RAM και κατανεμημένη RAM για αποθήκευση δεδομένα και φάση .
Παρέχει τρείς διαφορετικές αρχιτεκτονικές για να επιλέξουμε μεταξύ του μεγέθους της οντότητας και της ταχύτητας υλοποίησης του μετασχηματισμού o Pipelined , Η πιο γρήγορη υλοποίηση o Radix-4 , Προσφέρει μια ενδιάμεση λύση τόσο σε μέγεθος του κυκλώματος όσο και στην ταχύτητα υλοποίησης Radix-2,Έχει το μικρότερο κύκλωμα

-ΠΙΝΑΚΑΣ 3.3-

Στη συνέχεια παραθέτουμε το απλό σχέδιο και τα χαρακτηριστικά του FFT που υλοποιήσαμε στο σύστημά μας.



-ΣΧΗΜΑ 3.5-

Χαρακτηριστικά FFT

Μέγιστη Συχνότητα Λειτουργίας 285 MHz

Αρχιτεκτονική Κυκλώματος Pipelined , Η πιο γρήγορη υλοποίηση

Είσοδοι

Όνομα εισόδου	**Μέγεθος**	**Περιγραφή**
xn_re	14	Διάδρομος δεδομένων εισόδου το πραγματικό μέρος
xn_im	14	Διάδρομος δεδομένων εισόδου το φανταστικό μέρος

start	1	Το σήμα για την έναρξη του FFT
fwd_inv	1	Εάν fwd_inv = 1 τότε υπολογίζεται ο ευθύς ΔΜΦ αλλιώς ο αντίστροφος
fwd_inv_we	1	Σήμα για το fwd_inv
scale_sch	12	Scaling Schedule
scale_sch_we	1	Σήμα για το scale_sch
clk	1	ρολόι
	Έξοδοι	
Όνομα εξόδου	**Μέγεθος**	**Περιγραφή**
xk_re	14	Διάδρομος δεδομένων εξόδου το πραγματικό μέρος
xk_im	14	Διάδρομος δεδομένων εισόδου το φανταστικό μέρος
xn_index	12	Ο δείκτης του δεδομένου εισόδου
xk_index	12	Ο δείκτης του δεδομένου εξόδου
rfd	1	Έτοιμα για να δεχτεί δεδομένα
busy	1	Υπολογίζεται ο μετασχηματισμός
dv	1	Τα δεδομένα έγκυρα
edone	1	Το edone
done	1	και το done φανερώνουν ότι έχει υπολογιστεί ο

μετασχηματισμός

-ΠΙΝΑΚΑΣ 3.4α-

Προδιαγραφές Κυκλώματος

Η είσοδος xn_re λαμβάνει τα δεδομένα από την έξοδο του ad_controller ,dat_out , και βάση αυτών υπολογίζεται ο μετασχηματισμός Φουριέ.
Η είσοδος xn_im είναι σταθερή και έχει την τιμή '00000000000000'
Υπολογίζει το Διακριτό Μετασχηματισμό Φουριέ 4096 σημείων και η ευκρίνεια της συχνότητας που προσφέρει είναι 25KHz

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.4β-

Για την κατάλληλη λειτουργία του κυκλώματος έχουμε θέσει τις εισόδους :

- fwd_inv ίσο με το λογικό ένα (1) για να υλοποιεί το σύστημα τον Ευθύ Γρήγορο Μετασχηματισμό Φουριέ και fwd_inv_we =1 .
- scale_sch ίσο με [222222] και το scale_sch_we ίσο με 1 .
- Ο διάδρομος δεδομένων εισόδου του φανταστικού μέρους ισούται συνεχώς με μηδέν "000000000000000".

Σε αυτό το σημείο πρέπει να εξηγήσουμε γιατί χρειάζεται μετασχηματισμός 4096 σημείων για να έχουμε ευκρίνεια συχνότητας 25KHz. Η ευκρίνεια (P) προκύπτει συναρτήσει του αριθμού των σημείων του μετασχηματισμού και του ρυθμού δειγματοληψίας. Η ευκρίνεια συχνότητας είναι το πηλίκο της συχνότητας δειγματοληψίας με τον αριθμό των δειγμάτων :

$$P=\frac{f}{N}$$

Από την παραπάνω σχέση προκύπτει ότι η ευκρίνεια του συστήματος για να είναι 25 KHz , ο αριθμός των δειγμάτων πρέπει να είναι 4096 .Άρα πρέπει να έχουμε ένα μετασχηματισμό Φουριέ των 4096 σημείων .

Δεν θα παρουσιάσουμε για τον Γρήγορο Μετασχηματισμό Φουριέ διαγράμματα χρονισμού αφού το κύκλωμα είναι κατασκευασμένο από την ίδια εταιρία γεγονός που εγγυάται σίγουρα και σωστά αποτελέσματα .

3.4 : Κύκλωμα μετατροπής δεδομένων από συμπλήρωμα ως προς 2 σε 1QN μορφή

Στη συνέχεια είναι αναγκαίο το κύκλωμα μετατροπής της παράστασης του αριθμού από συμπλήρωμα ως προς 2 σε 1QN μορφή, για να είναι τα δεδομένα στην κατάλληλη παράσταση για την είσοδο των δεδομένων στο cordic. Αυτόν το σκοπό εξυπηρετεί η λειτουργική μονάδα ‘‘compl2_to_1qnformat ’’ . Παρουσιάζουμε τα χαρακτηριστικά του υποκυκλώματος :

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 500 MHz		
	Είσοδοι	
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
x_ re	14	Οι είσοδοι σε μορφή συμπλήρωμα ως προς 2
x_im	14	
x_index	12	Ο δείκτης των δεδομένων εισόδου
	Έξοδοι	
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
x_re_1qf	15	Οι έξοδοι σε μορφή 1QN
x_im_1qf	15	
x_index_1qf	12	Ο δείκτης των δεδομένων εξόδου .

-ΠΙΝΑΚΑΣ 3.5α-

Προδιαγραφές Κυκλώματος

Το κύκλωμα λαμβάνει τα δεδομένα από τον FFT που είναι σε παράσταση αριθμού συμπλήρωμα ως προς 2 και τα μετατρέπει σε παράσταση αριθμού 1QN .

Οι έξοδοι x_re_1qf , x_im_1qf αναπαριστούν τις εισόδους x_ re και x_im σε μορφή 1QN μορφή .

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.5β-

Για αρχή θα θέλαμε να παρουσιάσουμε τις δύο διαφορετικές μορφές αναπαράστασης αριθμών, το συμπλήρωμα ως προς δύο και την 1QN μορφή .Στην παράσταση συμπληρώματος ως προς 2 το MSB του αριθμού χρησιμοποιείται για την παράσταση του προσήμου. Όταν ο αριθμός είναι θετικός , τα υπόλοιπα (MSB =0) ψηφία του αριθμού παριστάνουν το μέτρο του αριθμού σε δυαδική μορφή . Όταν ο αριθμός είναι αρνητικός (MSB=1), το μέτρο του αριθμού δίνεται από το συμπλήρωμα ως προς 2 του συνόλου των ψηφίων του αριθμού .

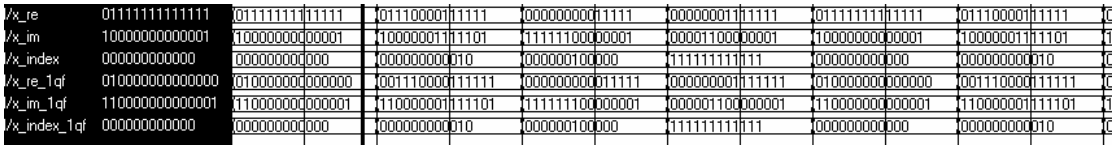
Το συμπλήρωμα ως προς 2 ορίζεται ως το συμπλήρωμα ως προς ένα προσαυξημένο κατά μία μονάδα. Έτσι το συμπλήρωμα ως προς 2 του 10010 είναι το 01101+1=01110. Δυστυχώς η αναπαράσταση των αριθμών με τη μορφή του συμπληρώματος ως προς 2 είναι αποδοτική στην πρόσθεση και την αφαίρεση των αριθμών, ενώ στην ψηφιακή ανάλυση του σήματος είναι χρησιμότερη η αναπαράσταση του αριθμού σε 1QN . Το cordic χρησιμοποιεί αυτήν τη μορφή .

Η 1QN μορφή είναι ουσιαστικά ένας αριθμός σε συμπλήρωμα ως προς 2 με υποδιαστολή. Το MSB και σε αυτήν την παράσταση αριθμών ορίζει το πρόσημο ,όταν MSB=0 τότε ο αριθμός είναι θετικός ενώ όταν MSB =1 ο αριθμός είναι αρνητικός. Έτσι ο αριθμός 01000000000000=0.1000000000000 $=2^0$ = +1 , ενώ ο αριθμός 110000000000000 =1.10000000000000 = -2^0 = -1, 11100000000000 =1.1100000000000 = -2^0 $+2^{-1}$ = -0.5 .

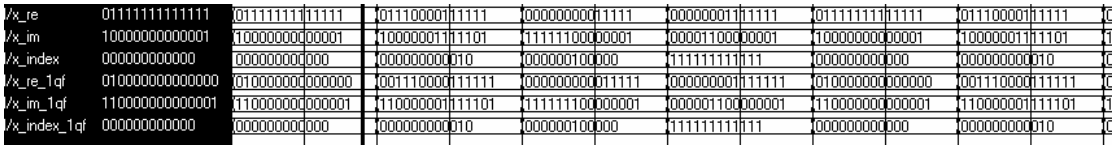
Ο κώδικας είναι γραμμένος με τον τρόπο περιγραφής της συμπεριφοράς (Behavioral). Υπάρχουν δύο διεργασίες (process), η πρώτη για τη μετατροπή του x_ re σε 1QN παράσταση και η δεύτερη για το x_im σε 1QN παράσταση. Η μετατροπή είναι απλή εάν ο αριθμός είναι θετικός. Τότε προσθέτουμε ένα επιπλέον ψηφίο 0 στην αρχή. Στην περίπτωση που ο αριθμός είναι ο 01111111111111 =+1 τότε τον μετατρέπουμε στον 0100000000000000. Εάν ο αριθμός είναι αρνητικός προσθέτουμε ένα ψηφίο 1 στην αρχή , έτσι ο 1000000000000 στο συμπλήρωμα ως προς 2 γίνεται 1.1000000000000 στην 1QN παράσταση αριθμών. Αυτή η μετατροπή αύξησε τα δεδομένα κατά ένα ψηφίο ,οπότε από 14 ψηφία έχουμε 15 ψηφία πληροφορίας .

Σημαντικά προβλήματα στη συγκεκριμένη μονάδα δεν αντιμετωπίσαμε , μας δυσκόλεψε όμως η εύρεση στοιχείων που να περιγράφουν ακριβώς την 1QN και γενικότερα την QN μορφή αριθμών. Για εύρεση αυτών των στοιχείων απευθυνθήκαμε στο διαδίκτυο .

Παραθέτουμε τα διαγράμματα χρονισμού τόσο τα επιθυμητά όσο και τα πραγματικά :



--Σχήμα 3.6α-



--Σχήμα 3.6β-

Στο πρώτο σχήμα παρουσιάζεται η επιθυμητή συμπεριφορά ενώ στην δεύτερη η πραγματική . Από τη σύκριση των δύο διαγραμμάτων παρατηρείαι η ομοιότητα των λειτουργιών.

3.5:Κύκλωμα εύρεσης της Ενέργειας του σήματος

Η λειτουργική μονάδα ‘‘corxnindex’’ λαμβάνει το πραγματικό και φανταστικό μέρος του αριθμού και εξάγει το πλάτος αυτού , δηλαδή στη συγκεκριμένη περίπτωση την ενέργεια του σήματος .

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 245 MHz		
	Είσοδοι	
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
x_ re	15	Οι είσοδοι σε μορφή
x_im	15	συμπλήρωμα ως προς 2
x_index_in	12	Ο δείκτης των δεδομένων εισόδου
clk	1	Ρολόι
	Έξοδοι	
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
platos	15	Το πλάτος των 2 εισόδων σε μορφή 1QN
rdy	1	Τα δεδομένα έγκυρα
x_index_out	12	Ο δείκτης των δεδομένων εξόδου .

-ΠΙΝΑΚΑΣ 3.6α-

Προδιαγραφές Κυκλώματος
Το κύκλωμα πρέπει να λαμβάνει τα δεδομένα από σε μορφή αριθμού 1QN καθώς και το δείκτη –σημείο μετασχηματισμού που αντιστοιχεί στα δεδομένα .
Είναι απαραίτητο να μην εισάγονται τιμές μαγαλύτερες από 1 και μικρότερες από -1 .
Η έξοδος platos περιέχει το πλάτος (ενέργεια) των εισόδων x_im και x_ re .
Ταυτόχρονα εξέρχεται και ο δείκτης –σημείο μετασχηματισμού που αντιστοιχεί στο platos. Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.6β-

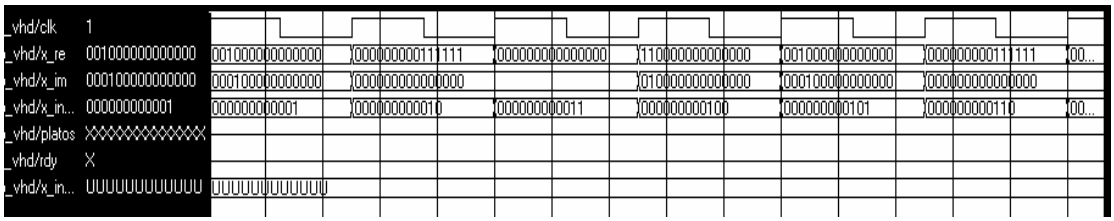
Στο corxnindex χρησιμοποιήσαμε άλλο ένα έτοιμο κύκλωμα από το ISE το cordic. Το cordic υλοποιεί όλες τις τριγωνομετρικές συναρτήσεις καθώς και τους μετασχηματισμούς από πολικές συντεταγμένες σε καρτεσιανές και τον αντίστροφο. Στο κύκλωμα υλοποιήσαμε το μετασχηματισμό από καρτεσιανές συντεταγμένες σε πολικές και μας ενδιέφερε μόνο το πλάτος του αριθμού στις πολικές συντεταγμένες και όχι η φάση .

Κατά τη διάρκεια του υπολογισμού του πλάτους το cordic εισάγει καθυστέρηση ίση με 19 κύκλους ρολογιού. Όταν δημιουργήσαμε το cordic (Customize –Generate) το ISE μας ενημέρωσε για αυτήν την καθυστέρηση. Επιπλέον ελέγξαμε μέσω της προσομοίωσης αν αυτή η καθυστέρηση είναι σταθερή όταν εισέρχονται μεγάλες τιμές, οπότε υποθετικά η δυσκολία υλοποίησης του κυκλώματος είναι μεγαλύτερη. Παρατηρήσαμε ότι όντως η καθυστέρηση του κυκλώματος είναι σταθερή στους 19 κύκλους ρολογιού .

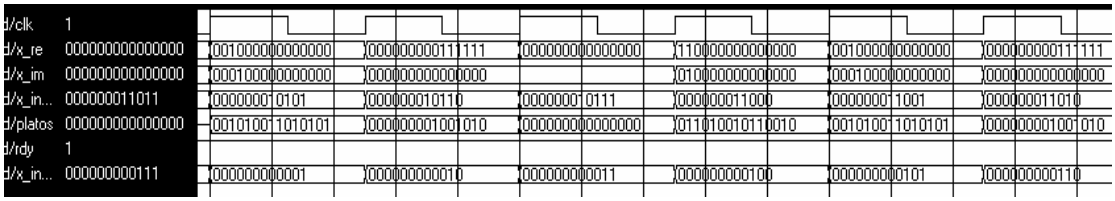
Αυτή η καθυστέρηση όμως προκαλεί πρόβλημα στο συγχρονισμό του δείκτη των δεδομένων εξόδου (x_index_out) με τα δεδομένα που αντιστοιχούν σε αυτόν .Σε κάθε θετικό παλμό του ρολογιού εισέρχονται νέα δεδομένα μαζί με τον δείκτη δεδομένων τους από το κύκλωμα του Γρήγορου Μετασχηματισμού Φουριέ. Το αντίστοιχο ακριβώς πρέπει να συμβαίνει στην έξοδο όπου πρέπει να εξέρχονται τα δεδομένα μαζί με το δείκτη που αντιστοιχεί σε αυτά. Για να επιτύχουμε αυτόν το συγχρονισμό έπρεπε να εισάγουμε καθυστέρηση 19 κύκλων ρολογιού στην διάδοση του δείκτη δεδομένων από την εισόδου προς την έξοδο. Κατασκευάσαμε δεκαεννέα θετικά ακμοπυροδότητους καταχωρητές οι οποίοι εισάγουν την κατάλληλη καθυστέρηση .

Σε αυτό το σημείο πρέπει να γράψουμε για το πρόβλημα χρονισμού που παρουσιάστηκε στην πραγματική προσομοίωση. Αντί για 19 κύκλους ρολογιού καθυστέρηση το cordic είχε 18 κύκλους ρολογιού καθυστέρηση οπότε έπρεπε να αφαιρέσουμε έναν καταχωρητή για να υπάρχει συμφωνία του δείκτη δεδομένων εξόδου με τα δεδομένα .

Στη συνέχεια παρουσιάζουμε τα διαγράμματα χρονισμού ξεκινώντας από τα επιθυμητά :



-ΣΧΗΜΑ 3.7α-



-ΣΧΗΜΑ 3.7β-

Στο πρώτο διάγραμμα φαίνονται τα αρχικά τέσσερα δεδομένα των x_re , x_im , x_index_in .Ενώ στο δεύτερο φαίνεται η έξοδος των πρώτων τεσσάρων δεδομένων από το κύκλωμα . Παραθέτουμε τον παρακάτω πίνακα για να είναι τα δεδομένα πιο ευκρινή.

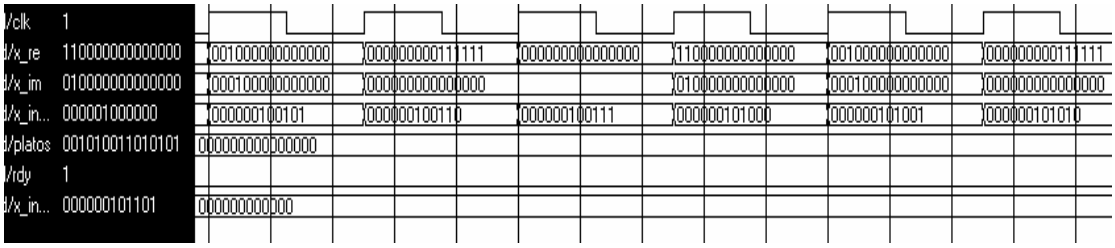
Είσοδοι		
xn_re	xn_im	x_index_in
1100000000000000	0100000000000000	0000000000001
0010000000000000	0001000000000000	0000000000010
0000000000111111	0000000000000000	0000000000011
0000000000000000	0000000000000000	0000000000100

-ΠΙΝΑΚΑΣ 3.7α-

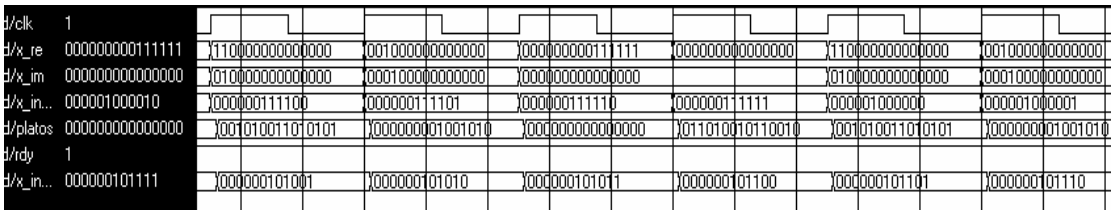
Έξοδοι	
platos	x_index_out
011010010110010	0000000000001
001010011010101	0000000000010
0000000001001010	0000000000011
0000000000000000	0000000001000

-ΠΙΝΑΚΑΣ 3.7β-

Στη συνέχεια παραθέτουμε τα διαγράμματα τα πραγματικά :



-ΣΧΗΜΑ 3.8α-



-ΣΧΗΜΑ 3.8β-

Είσοδοι		
xn_re	xn_im	x_index_in
1100000000000000	0100000000000000	0000000101001
0010000000000000	0001000000000000	0000000101010
0000000000111111	0000000000000000	0000000101011
0000000000000000	0000000000000000	0000000101100

-ΠΙΝΑΚΑΣ 3.8α-

Έξοδοι	
platos	x_index_out
0110100101100 10	0000000101001
0010100111010101	0000000101010
0000000010010 10	0000000101011
0000000000000000	0000000101100

-ΠΙΝΑΚΑΣ 3.8β-

Συγκρίνοντας τα πραγματικά διαγράμματα με τα επιθυμητά διακρίνουμε την ορθότητα των αποτελεσμάτων τόσο στην αλληλουχία αυτών όσο και στις τιμές τους.

Αλλά επίσης παρατηρούμε ότι στην πραγματική συμπεριφορά του κυκλώματος υπάρχει ένα διάστημα αρχικοποίησης της λειτουργίας του που διαρκεί 100 nsec σε αντίθεση με το επιθυμητό που αυτή η διάρκεια είναι μόλις 4 nsec.

3.6 : Κύκλωμα Εύρεσης και Ταξινόμησης των Οκτώ Δεικτών με την Μεγαλύτερη Ενέργεια

Σκοπός του κυκλώματος ‘‘taxinomisi_reg’’ είναι να βρεθούν οι οκτώ δείκτες του ψηφιακού σήματος που αντιστοιχούν σε σημεία του μετασχηματισμού με τη μεγαλύτερη ενέργεια, να ελεγχθούν ότι είναι πάνω από τα επίπεδα θορύβου που έχουμε ορίσει καθώς και να ταξινομηθούν κατά φθίνουσα σειρά.

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 112 MHz		
	Είσοδοι	
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
clk	1	Ρολόι
magnitude_in	15	Η ενέργεια που αντιστοιχεί στον δείκτη εισόδου
x_index_in	12	Ο δείκτης των δεδομένων εισόδου
	Έξοδοι	
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
out_en	1	Σήμα πληροφόρησης ότι τα δεδομένα είναι έτοιμα
reg1out	27	Οι οκτώ καταχωρητές που περιέχουν τους οκτώ δείκτες με τις μεγαλύτερες ενέργειες
reg2out	27	
reg3out	27	
reg4out	27	
reg5out	27	
reg6out	27	

reg7out	27
reg8out	27

-ΠΙΝΑΚΑΣ 3.9α-

Προδιαγραφές Κυκλώματος

Το κύκλωμα λαμβάνει τα δεδομένα από το κύκλωμα ‘‘corxnindex’’ και τα ταξινομεί κατάλληλα ώστε για κάθε μετασχηματισμό Φουριέ να εξάγει τους οκτώ καταχωρητές που θα περιέχουν τους δείκτες με την μεγαλύτερη ενέργεια .
Σε κάθε καταχωρητή regxout περιέχεται ο δείκτης (τα δώδεκα σημαντικότερα σημεία) καθώς και η ενέργεια (τα δεκαπέντε λιγότερο σημαντικά ψηφία) που αντιστοιχεί στον δείκτη .
Οι έξοδοι είναι ταξινομημένοι κατά φθίνουσα σειρά , στον reg1out είναι ο δείκτης με την μεγαλύτερη ενέργεια , ενώ στον reg8out ο δείκτης με την μικρότερη .

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.9β-

Τα δεδομένα εξόδου reg1out έως reg8out αποθηκεύουν τόσο το δείκτη δεδομένων (reg_out(26 downto 15)) όσο και την ενέργεια που αντιστοιχεί στο δείκτη δεδομένων (reg_out(14 downto 0)) .Όταν το out_en αποκτήσει την τιμή 1 σημαίνει ότι τα δεδομένα εξόδου είναι έγκυρα , δηλαδή έχουν βρεθεί τα σημεία του μετασχηματισμού με τη μεγαλύτερη ενέργεια και έχουν ταξινομηθεί. Η αρχιτεκτονική που χρησιμοποιήσαμε είναι η περιγραφή συμπεριφοράς (Behavioral).

Πριν συνεχίσουμε με την περιγραφή του κυκλώματος είναι χρήσιμο να παρουσιάσουμε ένα τέχνασμα που κάναμε. Από το κύκλωμα του cordic , corxnindex , λαμβάνουμε την ενέργεια το οποίο σημαίνει ότι το δεδομένο magnitude_in αναπαριστά ένα θετικό αριθμό , άρα το MSB , το μεγαλύτερης σημασίας ψηφίο δηλαδή το δέκατο πέμπτο ψηφίο , είναι πάντα 0. Εκμεταλλευτήκαμε αυτό το δεδομένο για να ελέγχουμε στη συνέχεια του κυκλώματος αν τα δεδομένα μας είναι έγκυρα. Έτσι κάθε φορά που αρχικοποιούμε τους εσωτερικούς καταχωρητές το δέκατο πέμπτο γίνεται ‘1’και αλλάζει μόνο όταν εισέλθουν έγκυρα δεδομένα. Το

δέκατο πέμπτο ψηφίο τόσο των εσωτερικών καταχωρητών όσο και των δεδομένων εξόδου reg1out …. reg8out είναι ουσιαστικά ένα σήμα σηματοδοσίας για το υπόλοιπο κύκλωμα. Παραθέτουμε ένα παράδειγμα για να δείξουμε την χρησιμότητα αυτού του τεχνάσματος: έστω ότι το προς επεξεργασία σήμα είναι ένα ημίτονο με συχνότητα 25 MHz τότε θα έχει μόνο μία αρμονική συχνότητα την πρώτη άρα μετά την ταξινόμηση μόνο το reg1out θα έχει έγκυρα δεδομένα αλλά χωρίς το σήμα σηματοδοσίας δεν θα μπορούσαμε να το διακρίνουμε. Αντιθέτως με το σήμα σηματοδοσίας γνωρίζουμε ποιά δεδομένα είναι έγκυρα και ποιά είναι άκυρα ανά πάσα στιγμή , πράγμα το οποίο θα είναι χρήσιμο στη συνέχεια του κυκλώματος .

Το κύκλωμα αποτελείται από 2 διεργασίες . Η πρώτη διεργασία ‘‘ reg ’’ είναι υπεύθυνη για την αρχικοποίηση των εσωτερικών καταχωρητών , την προσωρινή αποθήκευση των έγκυρων δεδομένων καθώς και την έξοδο των δεδομένων την κατάλληλη στιγμή .Όταν το x_index_in γίνει ‘‘ 111111111111 ’’ τότε το out_en γίνεται 1 και τα έγκυρα δεδομένα εξέρχονται . Πρέπει να σημειώσουμε ότι σε κάθε θετικό μέτωπο του ρολογιού η διεργασία ‘‘reg’’ ανανεώνει τους καταχωρητές σύμφωνα με τα δεδομένα που στέλνει η δεύτερη διεργασία ‘‘sort_the_data ’’.

Η δεύτερη διεργασία ‘‘sort_the_data ’’ λαμβάνει τα νέα δεδομένα και τα συγκρίνει με τα υπάρχοντα δεδομένα που είναι στους εσωτερικούς καταχωρητές. Εάν τα νέα δεδομένα είναι μεγαλύτερα από τα δεδομένα σε ένα εσωτερικό καταχωρητή τότε συμβαίνει μια ολίσθηση των καταχωρητών μεταξύ τους ώστε να αποθηκευτούν τα δεδομένα στον κατάλληλο καταχωρητή. Δηλαδή έχουμε 8 εσωτερικού καταχωρητές reg1_in, reg2_in ,reg3_in,….. reg8_in, στον reg1_in είναι αποθηκευμένος ο δείκτης δεδομένων με τη μεγαλύτερη ενέργεια. Εάν τα νέα δεδομένα έχουν ενέργεια μεγαλύτερη από την ενέργεια στον reg1_in τότε λαμβάνει χώρα μια ολίσθηση των περιεχομένων των καταχωρητών ώστε τα δεδομένα να έχουν τη σωστή θέση , οπότε συμβαίνει το εξής :

```
reg8_in<=reg7_in ; reg7_in<=reg6_in ; reg6_in<=reg5 _in;
reg5_in<=reg4_in ; reg4_in<=reg3 _in ;reg3_in<=reg2_in;
reg2_in<=reg1_in;
reg1_in<=x_index & neo ;
```

Επειδή οι εντολές εκτελούνται ακολουθιακά δεν υπάρχει περίπτωση να χαθούν τα υπάρχοντα δεδομένα. Στην περίπτωση που τα νέα δεδομένα ισούνται με ένα ήδη υπάρχον τότε συμβαίνει ξανά μια ολίσθηση μεταξύ των περιεχομένων των καταχωρητών ώστε να ταξινομηθούν σωστά. Στην περίπτωση που τα δεδομένα εισόδου δεν είναι μεγαλύτερα από κανένα υπάρχον τότε δε συμβαίνει τίποτα. Πρέπει να σημειώσουμε σε αυτό στο σημείο ότι για να διατηρηθεί σωστή η σειρά των καταχωρητών σύμφωνα με το μέγεθος της ενέργειας πρέπει πρώτα να γίνει σύγκριση του νέου δεδομένου με τον πρώτο καταχωρητή, με τον reg1_in , και μετά με το δεύτερο καταχωρητή τον reg2_in και στο τέλος με τον όγδοο καταχωρητή τον reg8_in .Τέλος είναι σημαντικό να σημειώσουμε ότι το μέγεθος του θορύβου ορίζεται εσωτερικά στην σταθερά (constant) ‘‘ noise_reg ’’.

Τα προβλήματα που συναντήσαμε για να υλοποιήσουμε το κύκλωμα εύρεσης και ταξινόμησης των οκτώ δεικτών με τη μεγαλύτερη ενέργεια ήταν τα περισσότερα και πιο χρονοβόρα στην επίλυση τους σε σύγκριση με τα προβλήματα στα υπόλοιπα κυκλώματα .Το πρώτο πρόβλημα ήταν θέμα λογικής και προγραμματισμού. Στην προσπάθειά μας να βρούμε ένα σωστό και αποδοτικό αλγόριθμο για ταξινόμηση δεδομένων προσπαθήσαμε να υλοποιήσουμε αλγόριθμους αποδοτικούς σε γλώσσες υψηλού επιπέδου όπως C ,Pascal ,Java ,πράγμα άστοχο γιατί γλώσσες περιγραφής υλικού (HDL) δεν είναι αποδοτικές για τέτοιους αλγόριθμους και επιπλέον η σύνθεσή τους και εφαρμογή τους σε FPGA είναι σχεδόν αδύνατη. Το δεύτερο λάθος στην σχεδίαση ήταν ότι δεν είχαμε υλοποιήσει εξ’αρχής την πρώτη διεργασία ‘‘reg’’ με αποτέλεσμα να μην αποθηκεύονται προσωρινά οι εσωτερικοί καταχωρητές οπότε η ταξινόμηση να μην υλοποιείται σωστά. Τρίτο και σημαντικότερο πρόβλημα ήταν ότι για να δουλέψει σωστά το κύκλωμα έπρεπε τόσο το κύκλωμα όσο και οι εσωτερικές διεργασίες να είναι συγχρονισμένες με το κεντρικό ρολόι. Για ένα μεγάλο χρονικό διάστημα προσπαθούσαμε να υλοποιήσουμε το κύκλωμα με ασύγχρονη λειτουργία. Αυτό είχε ως αποτέλεσμα να δουλεύει σωστά το σύστημα στην προσομοίωση της συμπεριφοράς αλλά δε δούλευε σωστά στην πραγματική προσομοίωση. Είτε παρουσιαζόταν ‘‘Iteration limit’’ και διακοπτόταν η προσομοίωση είτε δεν υλοποιούνταν η ταξινόμηση σωστά . Το να μην υλοποιηθεί καθόλου η ταξινόμηση, οφειλόταν στο γεγονός ότι είχαμε πολλά σήματα στη λίστα της διεργασίας (sensitivity list) με αποτέλεσμα η διεργασία να πραγματοποιείται πολλές φορές και να μην υπάρχει σωστό αποτέλεσμα.

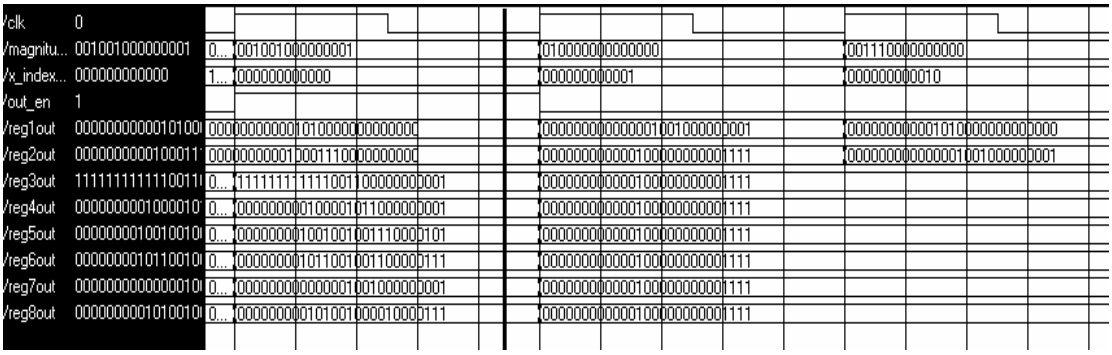
Όταν κατά τη διάρκεια της προσομοίωσης εμφανιζόταν το μήνυμα ‘‘Iteration limit’’ την χρονική στιγμή 140 ns το πρόβλημα ήταν ότι το κύκλωμα επαναλάμβανε έναν βρόγχο με άπειρη καθυστέρηση δέλτα . Η καθυστέρηση δέλτα ή ο χρόνος δέλτα είναι ένα μικρό χρονικό διάστημα της τάξης των μερικών νανοδευτερολέπτων που χρειάζεται για να γίνει μια ανάθεση τιμής σε ένα σήμα .

Στη συνέχεια παραθέτουμε τα διαγράμματα χρονισμού της λειτουργικής μονάδας ‘‘taxinomisi_reg’’ πρώτα το επιθυμητό και μετά το πραγματικό. Για να ελέγξουμε τη συμπεριφορά του κυκλώματος εισαγάγαμε τις εξής 16 τιμές :

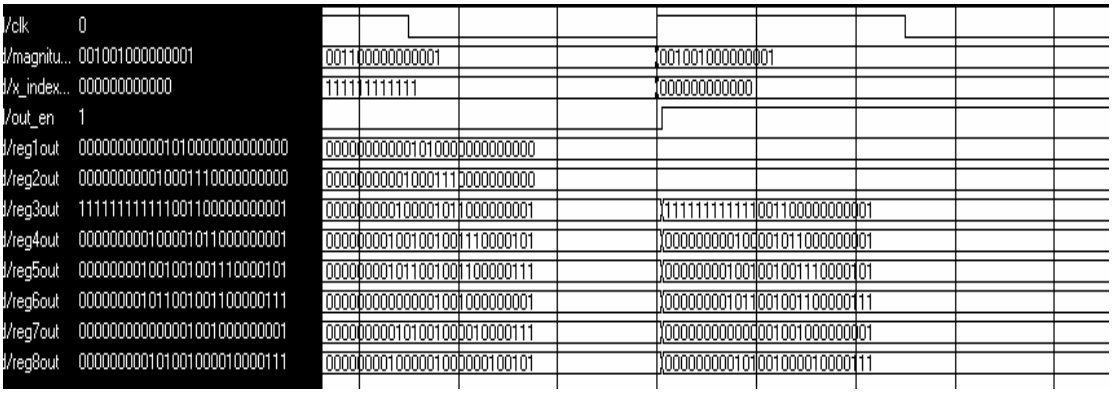
Είσοδοι	
x_index_in(δείκτες –σημεία μετασχηματισμού)	magnitude_in(πλάτος που αντιστοιχεί του δείκτη)
000000000000	001001000000001
000000000001	010000000000000
000000000010	001110000000000
000000000011	000000011001111
000000000100	001011000000001
000000000101	001000010000111
000000000110	000000000011101
000000000111	000000000100101
000000001000	001000000100101
000000001001	001001110000101
000000001000	000001010111010
000000001011	001001100000111
000000001100	000000010000111
000000001101	000000000000111
000000001110	000000000000111
111111111111	001100000000001
Έξοδοι	
00000000000101000000000000	
00000000001000111000000000	

11111111111001100000000001
00000000010000101100000001
00000000100100100111000101
00000000101100100110000111
00000000000000010010000001
00000000010100100010000111

-ΠΙΝΑΚΑΣ 3.10-



-ΣΧΗΜΑ 3.9α-



-ΣΧΗΜΑ 3.9β-

Από τα παραπάνω διαγράμματα παρατηρούμε την ομοιότητα των αποτελεσμάτων. Αυτό υποδεικνύει την σωστή λειτουργία του πραγματικού συστήματος .

3.7 : Κύκλωμα για τη Διασύνδεση του FPGA με το PLL Synthesizer

Η μονάδα ‘‘PLL_SYNDESI’’ εξυπηρετεί τον σκοπό της διασύνδεσης των δεδομένων από την μονάδα ‘‘taxinomisi_reg’’ προς την εξωτερική μονάδα που συνθέτει το **phase-locked loop** (PLL).

Χαρακτηριστικά

Μέγιστη Συχνότητα Λειτουργίας 296MHz

Είσοδοι

Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
clk	1	Ρολόι
din_val	1	Τα δεδομένα είναι έτοιμα για λήψη
dat1	13	Στα δεδομένα αυτά το lsb φανερώνει την εγκυρότητα τους , ενώ τα υπόλοιπα ψηφία μεταφέρουν την πληροφορία για το ποιό κανάλι θα κλειδώσει το PLL
dat2	13	
dat3	13	
dat4	13	
dat5	13	
dat6	13	
dat7	13	
dat8	13	

Έξοδοι

Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
address	4	Η διεύθυνση που δείχνει ποιό κανάλι είναι στην έξοδο
dat_out	16	Το κανάλι εισόδου
dat_val	1	Σήμα πληροφόρησης ότι τα δεδομένα είναι έτοιμα και έγκυρα

-ΠΙΝΑΚΑΣ 3.11α-

Προδιαγραφές Κυκλώματος

Το κύκλωμα ‘‘PLL_SYNDESI’’ λαμβάνει τα οκτώ δεδομένα datx (1<x<8) όταν το din_val είναι στο λογικό ‘1’ . Η address φανερώνει σε ποίον PLL Synthesizer θα σταλει το δεδομένο dat_out . Το PLL Synthesizer ενημερώνεται με νέα δεδομένα κάθε 20 μsec .Το σήμα dat_val πρέπει να είναι κεντραρισμένο σε κάθε ζεύγος address και dat_out και η χρονική διαφορά δύο διαδοχικών σημάτων dat_val είναι 100 nsec.

Επίσης όταν δεν υπάρχουν έγκυρα δεδομένα εξάγεται μια προκαθορισμένη λέξη , η ‘111111111111’.

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.11β-

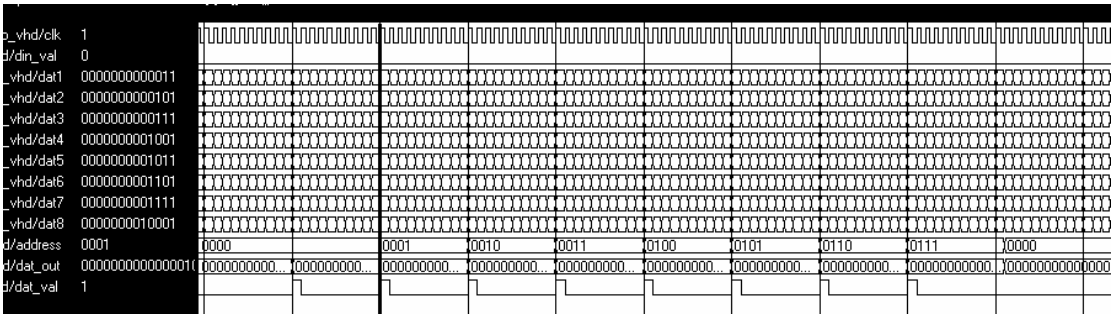
Η μονάδα θα πρέπει να πληροί όλες τις προδιαγραφές που απαιτεί η διασύνδεση. Αυτές τις έχουμε ήδη παραθέσει στο κεφάλαιο ‘Γενική Περιγραφή Συστήματος ’. Το κύκλωμα είναι συγχρονισμένο με τη θετική ακμή του ρολογιού. Η είσοδος δέχεται τους αριθμούς των καναλιών ταξινομημένους σύμφωνα με την ενέργεια που αντιστοιχεί στο καθένα. Όταν din_val =1 τότε τα δεδομένα που εισέρχονται στο ‘‘PLL_SYNDESI’’ είναι έγκυρα. Στη συνέχεια εξάγονται οι αριθμοί των καναλιών με τη σωστή σειρά σειριακά και στη σωστή χρονική στιγμή.

Έχουμε περιγράψει το κύκλωμα συνδυάζοντας και τους τρεις διαφορετικούς τρόπους περιγραφής , δομική , ροής δεδομένων και συμπεριφοράς. Το δομικό μέρος είναι το υποκύκλωμα ‘‘reg_egkiro’’. Αυτό λαμβάνει τα δεδομένα και τα ελέγχει εάν είναι έγκυρα , δηλαδή εάν το μικρότερης σημασία ψηφίο είναι ‘0’ αλλιώς στην έξοδό του μεταφέρει την λέξη-κλειδί για τη δήλωση ότι το δεδομένο είναι άκυρο. Η λέξη κλειδί που έχουμε ορίσει είναι ‘‘not_valid_reg’’= ‘‘111111111111111’’. Στο κύκλωμα ‘‘PLL_SYNDESI’’ υλοποιούνται παράλληλα οκτώ υποκυκλώματα ‘‘reg_egkiro’’ για την παράλληλη επεξεργασία των οκτώ εισόδων .Στη συνέχεια μέσω δύο διεργασιών ελέγχουμε τη σωστή έξοδο των καναλιών προς το PLL _Synthesizer . Η πρώτη διεργασία ουσιαστικά είναι ένα χρονόμετρο το οποίο

στέλνει κάθε 20 μsec εσωτερικό σήμα για να αρχίσει η μεταφορά των καναλιών προς το PLL _Synthesizer. Η δεύτερη διεργασία όταν ενεργοποιηθεί με το εσωτερικό σήμα ‘ren’ αποστέλλει στην έξοδο τα σωστά δεδομένα , address , dat_out , dat_val. Πρώτα στέλνει το πρώτο κανάλι και στη συνέχεια μετά από 100 nsec δεύτερο κανάλι και ούτω κάθε εξής. Για τη σωστή χρονική διάρκεια τόσο των 100 nsec όσο και των 20 μsec χρησιμοποιήσαμε εσωτερικούς μετρητές τους cnt100n και cnt20u. Επίσης γράφουμε ότι το κύκλωμα χρησιμοποιεί την περιγραφή ροής δεδομένων γιατί το εσωτερικό σήμα ‘‘ren’’ καθορίζεται εκτός των καταχωρητών ‘‘reg_egkiro’’ και των δύο διεργασιών .

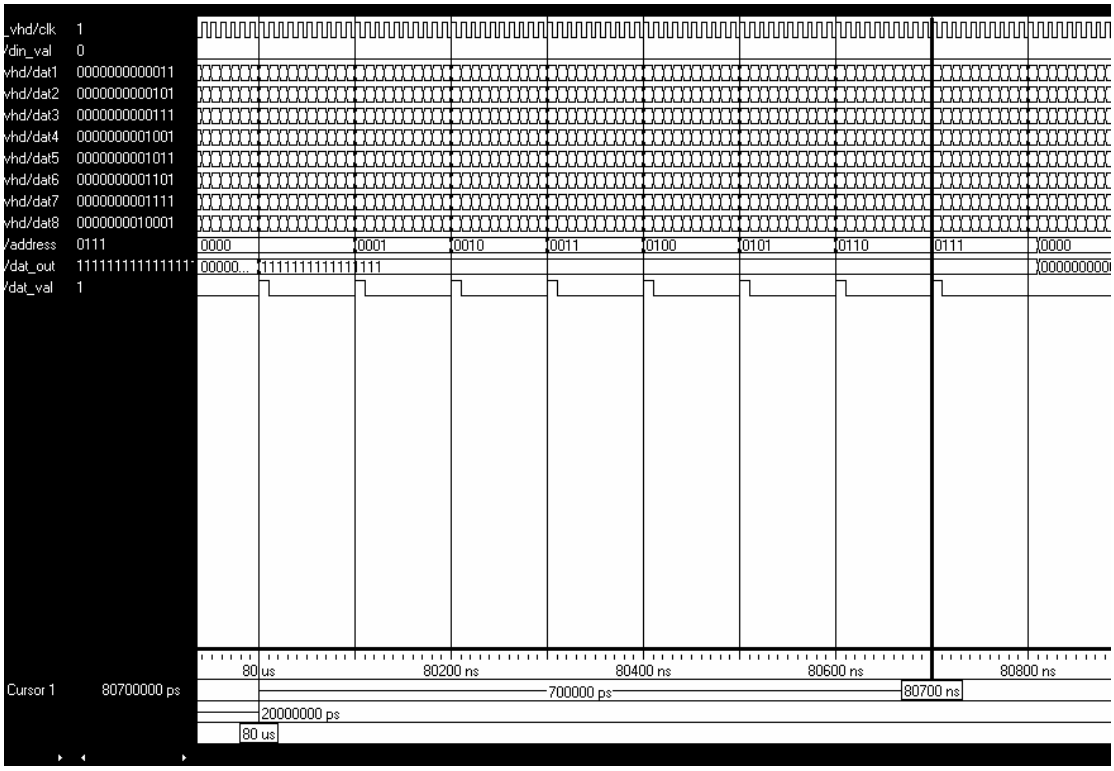
Σημαντικά προβλήματα στην υλοποίηση του κυκλώματος ‘‘PLL_SYNDESI’’ δε συναντήσαμε παρά μόνο στο θέμα του σωστού χρονισμού των δύο διεργασιών ώστε να λειτουργούν περιοδικά , δηλαδή κάθε 20 μsec η πρώτη και κάθε 100 nsec η δεύτερη.

Στη συνέχεια παρουσιάζουμε τα διαγράμματα χρονισμού . Αρχικά τα επιθυμητά δεδομένα είναι :



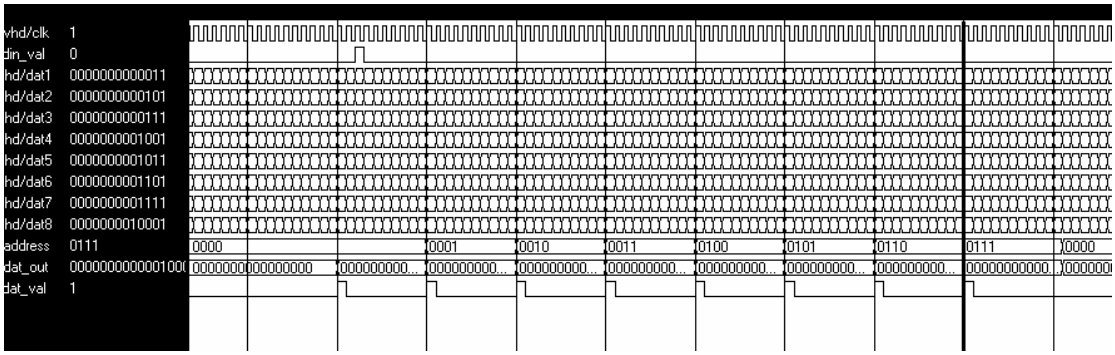
-ΣΧΗΜΑ 3.10α-

Στο πρώτο διάγραμμα (σχήμα 3,10α)φαίνεται η έξοδος του ‘‘PLL_SYNDESI’’ όταν τα δεδομένα είναι έγκυρα. Αντίθετα στο δεύτερο διάγραμμα (σχήμα3,10β) φαίνεται η έξοδος όταν τα δεδομένα είναι άκυρα. Επίσης στο δεύτερο φαίνονται και οι απαιτήσεις του κυκλώματος σχετικά με τους χρονισμούς , δηλαδή τα δεδομένα να εξάγονται κάθε 20μsec και να διαρκούν 100 nsec το καθένα .

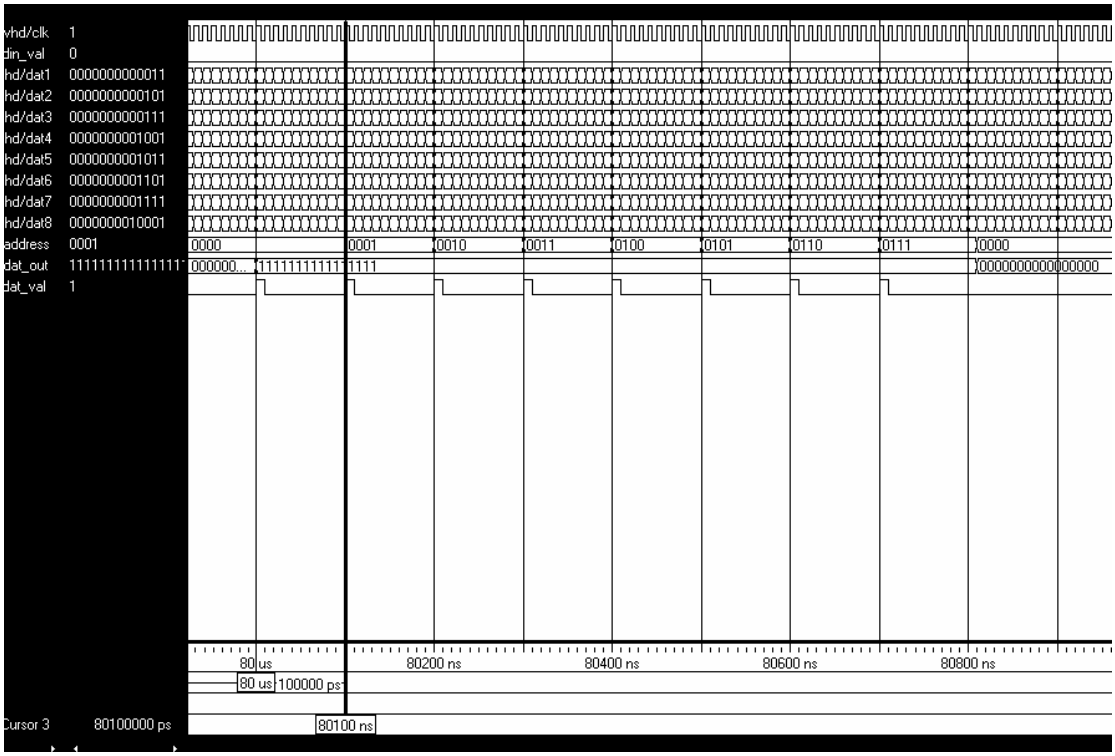


-ΣΧΗΜΑ 3.10β-

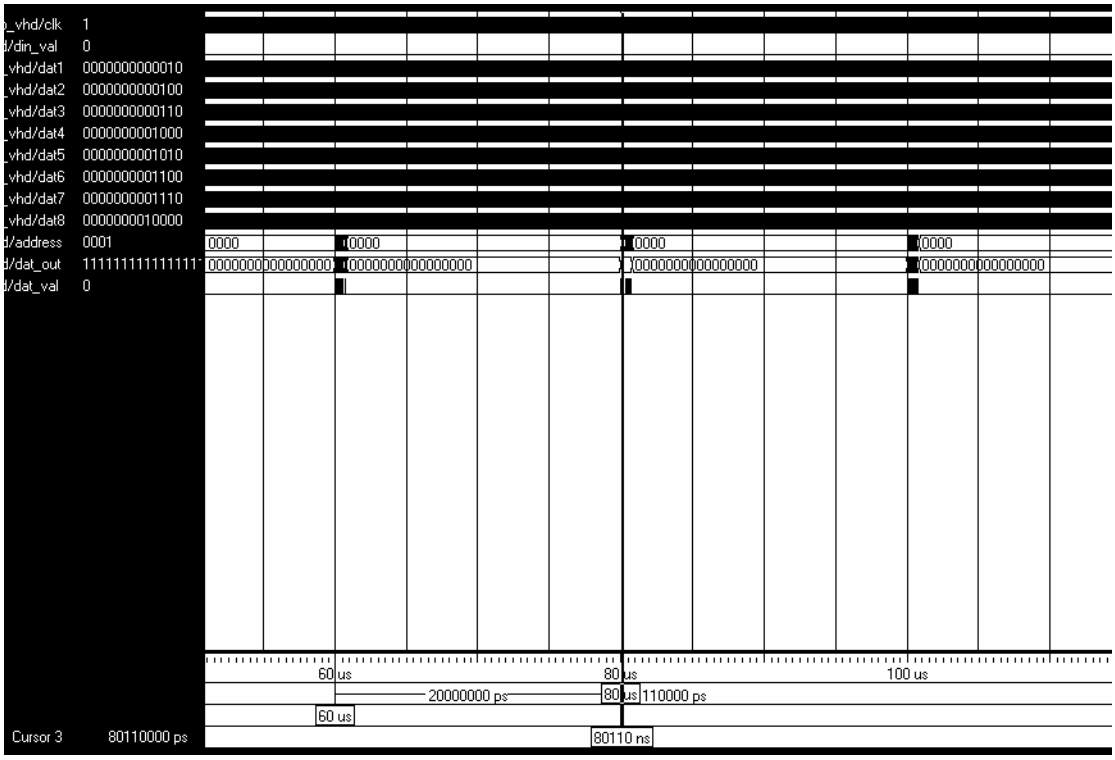
Τα πραγματικά διαγράμματα φαίνονται παρακάτω, όπου και παρατηρούμε την συμφωνία των πραγματικών αποτελεσμάτων με τα επιθυμητά. Στο πρώτο παρατηρείται η συμπεριφορά για τα έγκυρα δεδομένα , στο δέυτερο η συμπεριφορά για τα άκυρα δεδομένα, στο τρίτο φαίνεται η χρονική διαφορά μεταξύ δύο διαδοχικών εξόδων :



-ΣΧΗΜΑ 3.11α-



-ΣΧΗΜΑ 3.11β-



-ΣΧΗΜΑ 3.11γ-

Θα πρέπει να σημειώσουμε μια διαφορά στο όγδοο δεδομένο που εξέρχεται τόσο στα επιθυμητά διαγράμματα όσο και στα πραγματικά. Αντί να έχει χρονική διάρκεια 100nsec έχει 110 nsec. Αυτό όμως δεν είναι ουσιαστικό πρόβλημα γιατί το όγδοο δεδομένο είναι το τελευταίο και έτσι το PLL Synthesizer θα έχει λάβει όλα τα δεδομένα σωστά .

3 .8 :Κύκλωμα 8 καταχωρητών σε 1 καταχωρητή

Σκοπός αυτού του κυκλώματος ‘‘regist’’ είναι να δέχεται τα οκτώ δεδομένα από το κύκλωμα της ταξινόμησης και να τα εξάγει σειριακά, δηλαδή σε κάθε θετικό μέτωπο του ρολογιού να εξέρχεται ένα δεδομένο από τα οκτώ .Το κύκλωμα ‘‘regist’’ το υλοποιήσαμε για να γίνει το κύκλωμα Control_Of_Ram , που παρουσιάζεται στην συνέχεια , πιο αποδοτικό .

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 425 MHz		
Είσοδοι		
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
reg1_in	27	
reg2_in	27	Δεδομένα εισόδου που θα
reg3_in	27	εξέλθουν στην συνέχεια
reg4_in	27	σειριακά
reg5_in	27	
reg6_in	27	
reg7_in	27	
reg8_in	27	
da_val	1	Τα δεδομένα είναι έτοιμα για λήψη
clk	1	Ρολόι

Έξοδοι		
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
reg_serial_out	27	Η σειριακή έξοδος
rvd	1	Σήμα πληροφόρησης ότι τα δεδομένα είναι έτοιμα και έγκυρα

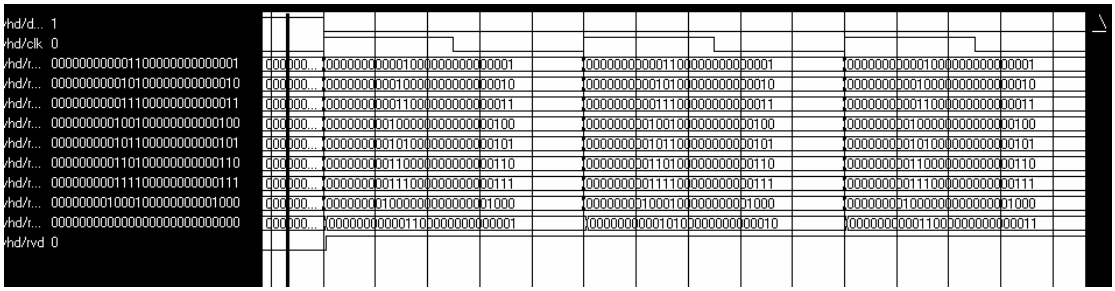
-ΠΙΝΑΚΑΣ 3.12α-

Προδιαγραφές Κυκλώματος
Το κύκλωμα λαμβάνει παράλληλα τα επιθυμητά δεδομένα –καταχωρητές (regx_in) όταν da_val είναι στο λογικό ‘1’.
Στη συνέχεια οι καταχωρητές εξέρχονται σειριακά μέσω της εξόδου reg_serial_out , με πρώτον τον reg1_in και τελευταίο τον reg8_in και θέτει το σήμα rvd στο λογικό ‘1’
Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.12β-

Το κύκλωμα ‘‘regist’’ έχει περιγραφεί σύμφωνα με τη συμπεριφορά που θέλαμε να έχει ,(Behavioral architecture). Έχουμε δημιουργήσει δύο διεργασίες .Η πρώτη διεργασία λαμβάνει τα νέα δεδομένα, τα αποθηκεύει προσωρινά και την κατάλληλη στιγμή ενεργοποιεί τη δεύτερη διεργασία. Η δεύτερη διεργασία με τη σειρά της, διοχετεύει προς την έξοδο το σωστό εσωτερικό καταχωρητή .

Στη συνέχεια παρατίθενται τα διαγράμματα χρονισμού της επιθυμητής λειτουργίας του κυκλώματος :



-ΣΧΗΜΑ 3.12-

Στο προηγούμενο διάγραμμα παρατηρείται ότι εξέρχεται πρώτα το περιεχόμενο της εισόδου reg1_in μετά το περιεχόμενο του reg2_in και ούτω καθεξής .

3.9 : Κύκλωμα Διαχωρισμού Έγκυρων Δεδομένων από τα Μη-έγκυρα

Όπως και το προηγούμενο κύκλωμα ‘‘regist’’ έτσι και ‘‘no_negative_regs ’’ σκοπό έχει να εισέρχονται με τη σωστή σειρά τα έγκυρα δεδομένα στο κύκλωμα ‘‘Control_Of_Ram’’ .Το ‘‘no_negative_regs ’’ απομονώνει τα θετικά –έγκυρα δεδομένα από τα αρνητικά –άκυρα και στη συνέχεια εξάγει τα έγκυρα μόνο δεδομένα .

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 400 MHz		
	Είσοδοι	
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
clk	1	Ρολόι
d_value	1	Τα δεδομένα είναι έτοιμα για λήψη
reg_in	27	Δεδομένα εισόδου
	Έξοδοι	
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
reg_serial_out	27	Η σειριακή έξοδος των θετικών δεδομένων

rvd	1	Σήμα πληροφόρησης ότι τα δεδομένα είναι έτοιμα και έγκυρα

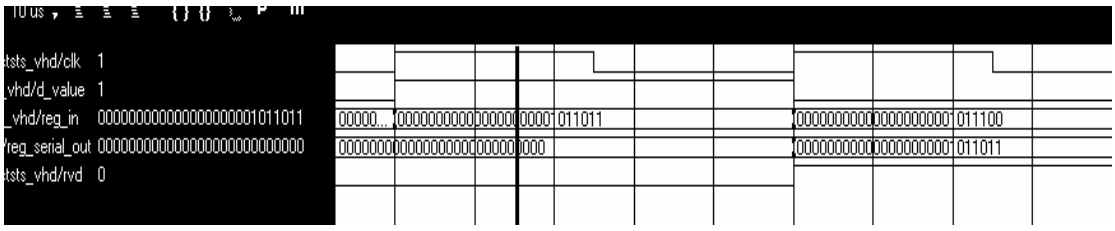
-ΠΙΝΑΚΑΣ 3.13α-

Προδιαγραφές Κυκλώματος
Το κύκλωμα λαμβάνει το δεδομένο reg_in όταν το d_value στην λογική τιμή ‘1’. Ελέγχει αν το δεδομένο είναι έγκυρο ,να ισχύει η εσωτερική σχέση ,αν είναι έγκυρο τότε τα δεδομένα εξέρχονται και τίθεται το rvd στη λογική τιμή ‘1’. Αλλιώς η έξοδος παραμένει μηδενική και το rvd λαμβάνει τη τιμή ‘0’.
Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

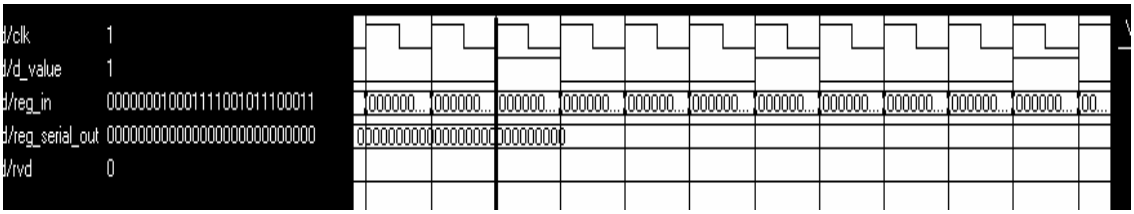
-ΠΙΝΑΚΑΣ 3.13β-

Η ανεύρεση των έγκυρων δεδομένων είναι εύκολη διαδικασία καθώς το σήμα σηματοδοσίας , το δέκατο πέμπτο ψηφίο του reg_in , μας υποδεικνύει ποιός αριθμός είναι έγκυρος και ποιός άκυρος. Το δέκατο πέμπτο ψηφίο πρέπει να είναι ‘0’ για να θεωρηθεί έγκυρο., αλλιώς θεωρείται άκυρο.

Το κύκλωμα λοιπόν υλοποιεί ένα απλό έλεγχο του ψηφίου σηματοδοσίας και αποστέλλει τα έγκυρα δεδομένα. Τα δυο επόμενα διαγράμματα παρουσιάζουν την επιθυμητή λειτουργία του συστήματος:

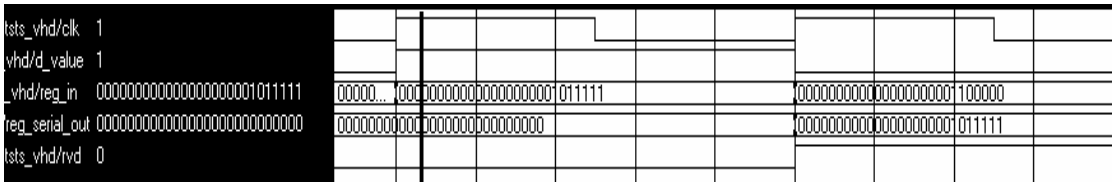


-ΣΧΗΜΑ3.13 α-

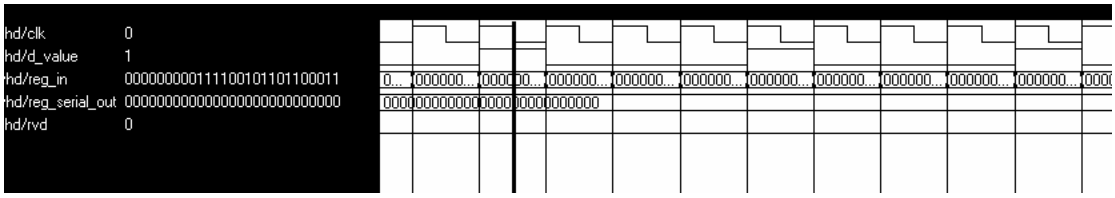


-ΣΧΗΜΑ3.13 β-

Στο διάγραμμα 3.13α φαίνεται η λειτουργία του κυκλώματος όταν το δεδομένο είναι έγκυρο ενώ στο 3.13β όταν το δεδομένο έχει το δέκατο πέμπτο ψηφίο ίσο με το λογικό ‘1’. Στα επόμενα διαγράμματα 3.14α και 3.14β παρατίθεται η λειτουργία του πραγματικού συστήματος που είναι ίδια με την επιθυμητή.



-ΣΧΗΜΑ3.14 α-



-ΣΧΗΜΑ3.14 β-

3.10 : Κύκλωμα ελέγχου της Μνήμης RAM

Το κύκλωμα ‘‘Control_Of_Ram’’ αποθηκεύει τα σταλμένα δεδομένα και επίσης τα αποστέλλει στην έξοδο την κατάλληλη στιγμή.

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 385 MHz		
Είσοδοι		
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
Clk	1	Ρολόι
reg_in	27	Δεδομένα εισόδου
ena	1	Επιτρέπει την χρησιμοποίηση της Α εισόδου της Ram
wea	1	Τα δεδομένα είναι έτοιμα για λήψη και εγγραφή στην Α είσοδο της Ram
Έξοδοι		
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
d_out	29/27	Η σειριακή έξοδος των δεδομένων από την Β πόρτα της Ram
enb_en	1	Σήματα πληροφόρησης
p_out	1	ότι τα δεδομένα είναι έτοιμα και έγκυρα

-ΠΙΝΑΚΑΣ 3.14α-

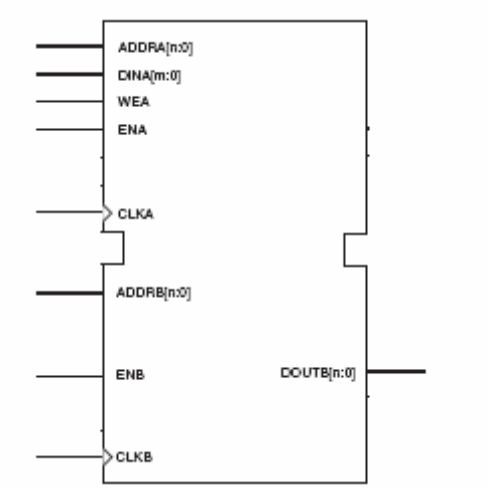
Προδιαγραφές Κυκλώματος
Αποθηκεύει τα δεδομένα σε μια RAM 1024 θέσεων μνήμης .Κάθε φορά που γεμίζει η μνήμη , αποθηκεύει τα νέα δεδομένα από την αρχή .

Όταν έχουν αποθηκευτεί 256 νέα δεδομένα τότε αυτά εξέρχονται σειριακά και κάθε 50nsec .

Η συχνότητα λειτουργίας του κυκλώματος είναι 100 MHz

-ΠΙΝΑΚΑΣ 3.14β-

Το Xilinx ISE και κατ' επέκταση το Virtex 4 μας παρέχει ένα μεγάλο αριθμό από μνήμες για να υλοποιήσουμε την Ram. Επιλέξαμε την Dual Port Block Memory Core 6.3, της οποίας οι διασυνδέσεις φαίνονται παρακάτω:



-ΣΧΗΜΑ 3.15-

Η μνήμη έχει δύο εισόδους για τη διεύθυνση, μία για κάθε πόρτα A|B (ADDR [A|B]). Η ADDRA δείχνει σε ποιά διεύθυνση θα αποθηκευτούν τα εισερχόμενα δεδομένα του διαδρόμου δεδομένων DINA [m:0] όταν ισχύει ENA ='1' και WEA='1'. Αντίθετα η διεύθυνση ADDRB δείχνει ποιάς διεύθυνσης το περιεχόμενο θα σταλεί στην πύλη DOUTB [m:0] και συγχρόνως στην έξοδο του κυκλώματός μας. Τόσο η πύλη A όσο και η πύλη B είναι συγχρονισμένες με την θετική ακμή των ρολογιών CLKA και CLKB αντίστοιχα. Είναι σημαντικό να παρουσιάσουμε μερικά χαρακτηριστικά της Dual Port Block Memory Core 6.3:

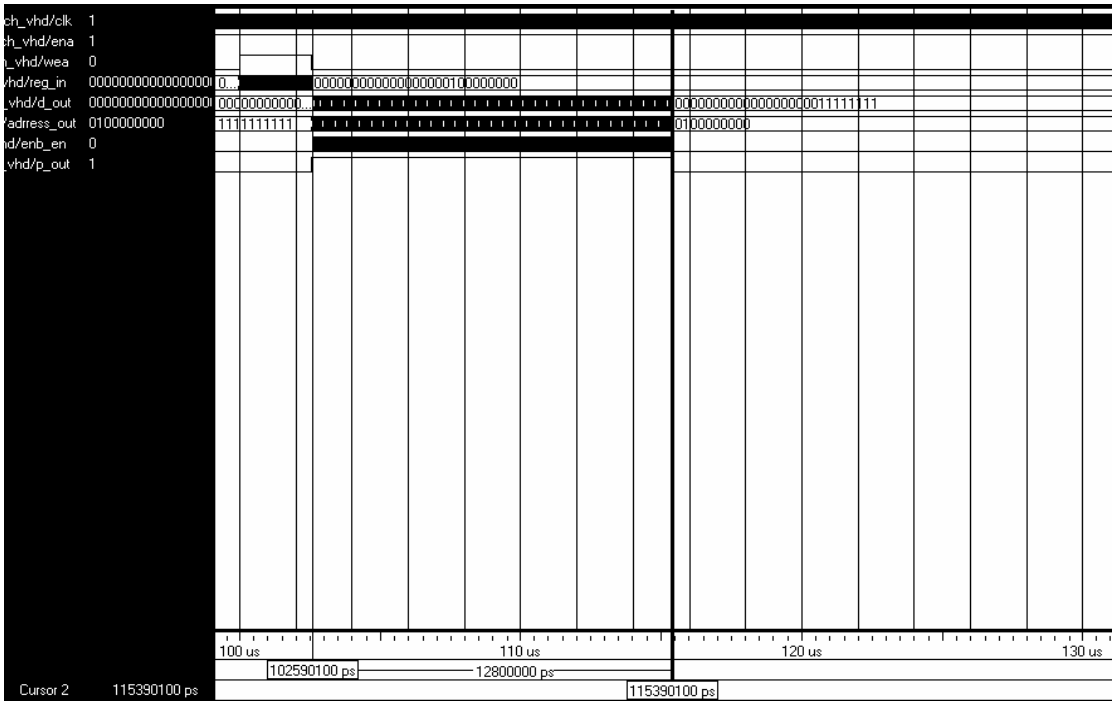
Χαρακτηριστικά DP Block Memory v6.3
Παρέχει εναλλακτικές αρχιτεκτονικές για την εγγραφή στη μνήμη
• Πρώτα εγγραφή και μετά διάβασμα της μνήμης (Read after Write)
• Πρώτα διάβασμα και μετά εγγραφή στη μνήμη (Read before Write)
Υποστηρίζει δεδομένα μήκους λέξης από 1 έως 256 ψηφία
Υποστηρίζει μέγεθος μνήμης από 2 έως 1M διευθύνσεις
Οι θύρες A και B είναι ανεξάρτητες μεταξύ τους

-ΠΙΝΑΚΑΣ 3.15-

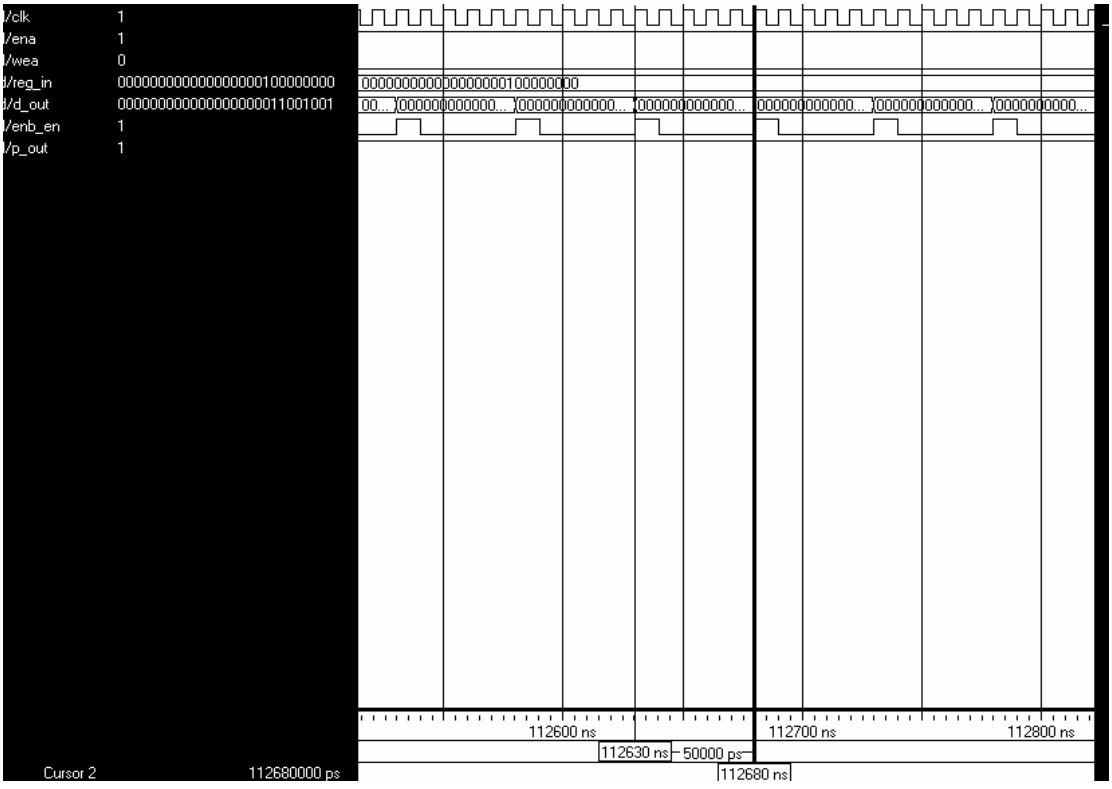
Η RAM που υλοποιείται στο κύκλωμα έχει μήκος λέξης 27 ψηφία, αποθηκευτικό χώρο 1024 δεδομένων και υποστηρίζει την αρχιτεκτονική ‘‘Read after Write’’

Το υποκύκλωμα ‘‘Control_Of_Ram’’ ελέγχει ουσιαστικά την Ram που έχουμε κατασκευάσει. Όποτε εισέρχεται ένα νέο δεδομένο, το ‘‘Control_Of_Ram’’, το αποθηκεύει στην αμέσως επόμενη διεύθυνση που δεν είναι κατειλημμένη. Όταν φτάσει στην τελευταία διεύθυνση τότε συνεχίζει την εγγραφή στην αρχή της μνήμης ,αρχική διεύθυνση. Η σημαντικότερη εργασία που πραγματοποιεί η λειτουργική μονάδα ‘‘Control_Of_Ram’’ είναι ο έλεγχος της εξόδου προς το Usb_microcontroller. Κάθε φορά που η ADDRA λαμβάνει την τιμή ‘255’ ή την ‘511’ ή‘767’ και τέλος ‘1023’ , η ‘‘Control_Of_Ram’’ στέλνει 256 δεδομένα προς τον Usb_microcontroller. Αυτά τα δεδομένα δε στέλνονται σε κάθε θετικό παλμό του ρολογιού αλλά κάθε 50 nsec. Αυτό συμβαίνει επειδή το Usb_microcontroller δε λειτουργεί με συχνότητα 100MHz αλλά το πολύ με συχνότητα 48MHz, έτσι εμείς διαλέξαμε να λειτουργεί το Usb_microcontroller με συχνότητα 40 MHz. Γι αυτό τα δεδομένα αποστέλλονται κάθε 50 ns. Σε επόμενη παράγραφο θα περιγράψουμε τη διασύνδεση του FPGA με το Usb_microcontroller .

Παραθέτουμε τα διαγράμματα χρονισμού του κυκλώματος, τόσο τα επιθυμητά όσο και τα πραγματικά. Πρέπει να σημειώσουμε ότι για να ελέγξουμε το κύκλωμα έχουμε θέσει και μια επιπλέον έξοδο address_out ,που μεταφέρει την τιμή της διεύθυνσης της θύρας B την ADDR_B:



-ΣΧΗΜΑ 3.16α-

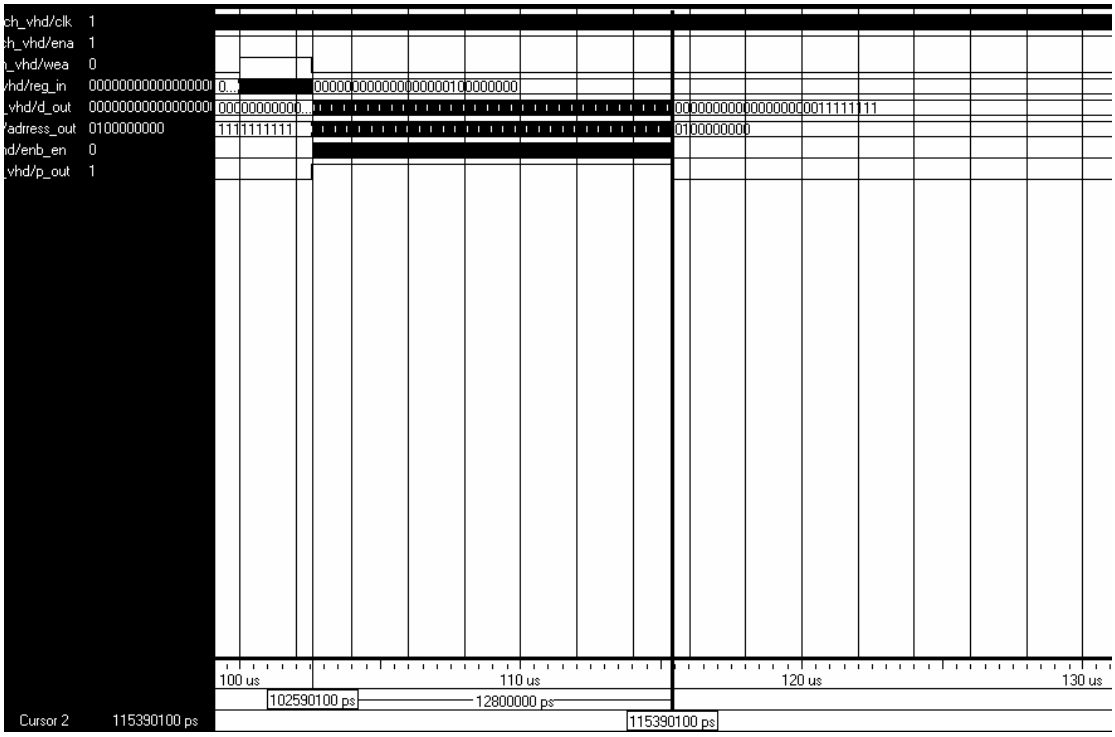


-ΣΧΗΜΑ 3.16β-

Στο διάγραμμα 3.16α παρατηρείται ότι η χρονική απόσταση μεταξύ του πρώτου δεδομένου που εξέρχεται από την RAM με αυτή του τελευταίου είναι 12800 nsec .Αν διαιρέσουμε την χρονική απόσταση με την διάρκεια του κάθε εξερχόμενου

δεδομένου που είναι 50 nsec θα υπολογίσουμε πόσα δεδομένα έχουν μεταφερθεί . Άρα έχουν εξέλθει 256 δεδομένα . Επίσης παρατηρούμε ότι η πρώτη διεύθυνση είναι η '000000000' ενώ η τελευταία είναι η '0001111111' , που επιβεβαιώνει το προηγούμενο αποτέλεσμα .

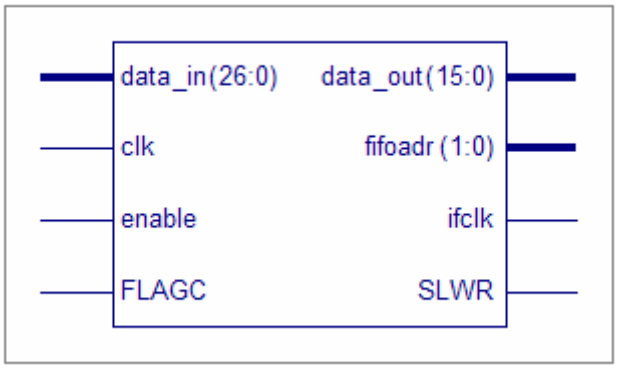
Στο δεύτερο διάγραμμα 3.16β παρουσιάζεται η απόσταση μεταξύ δύο νέων δεδομένων που είναι 50 nsec, τόση όση είναι η προδιαγραφή. Το επόμενο διάγραμμα παρουσιάζει την πραγματική συμπεριφορά του κυκλώματος και φανερώνει τη σωστή λειτουργία του συστήματος, ότι δηλαδή εξέρχονται 256 δεδομένα κάθε φορά που ενεργοποιείται η μεταφορά δεδομένων .



-ΣΧΗΜΑ 3.17-

3.11 : Διασύνδεση με το Usb microcontroller

Στη συνέχεια παρουσιάζουμε το κύκλωμα που πραγματοποιεί τη διασύνδεση του κυρίως κυκλώματος με το USB μικροελεγκτή . Το κύκλωμα έχει υλοποιηθεί σε SPARTAN 3E και ονομάζεται ''Master Mode ''



-ΣΧΗΜΑ 3.18-

Χαρακτηριστικά		
Μέγιστη Συχνότητα Λειτουργίας 230 MHz		
Είσοδοι		
Όνομα Εισόδου	**Μέγεθος**	**Περιγραφή**
clk	1	Ρολόι
data_in	27	Δεδομένα εισόδου
enable	1	Σήμα για την εγκυρότητα των δεδομένων εισόδου
FLAGC	1	Σήμα Ενημέρωσης της κατάστασης της FIFO του USB
Έξοδοι		
Όνομα Εξόδου	**Μέγεθος**	**Περιγραφή**
data_out	16	Η σειριακή έξοδος των δεδομένων προς την FIFO

		του USB
SLWR	1	Σήμα για εγγραφή στο USB
Ifclk	1	Σήμα εξωτερικού χρονισμού της FIFO του USB
fifoadr	1	Διεύθυνση που καθορίζει ποία FIFO θα χρησιμοποιηθεί

-ΠΙΝΑΚΑΣ 3.16α-

Προδιαγραφές Κυκλώματος
Όταν το enable είναι στη λογική τιμή ‘1’ και το FLAGC στη τιμή ‘0’ , το κύκλωμα λαμβάνει τα δεδομένα data_in . Στη συνέχεια τα στέλνει ανά 2 bytes με την κατάλληλη σηματοδοσία στο USB μικροελεγκτή .
Η συχνότητα λειτουργίας του κυκλώματος είναι 40 MHz

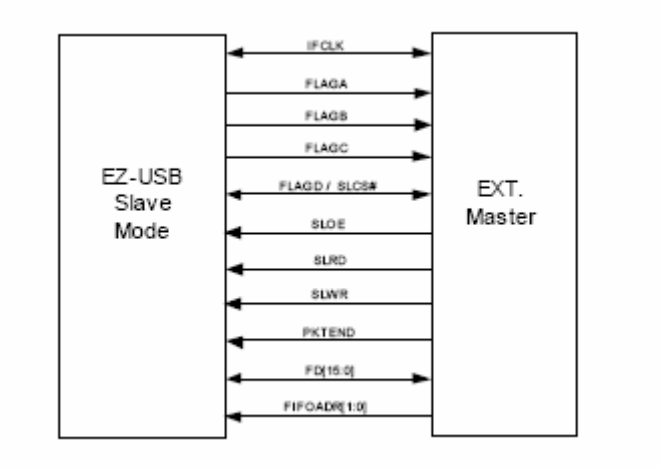
-ΠΙΝΑΚΑΣ 3.16β-

Το ‘‘Master Mode ’’ απαρτίζεται από δύο υποκυκλώματα τα ‘‘Receive_Data’’ και ‘‘Interconnection’’ και λειτουργεί σε συχνότητα 40 MHz . Το πρώτο λαμβάνει τα δεδομένα, τα διαχωρίζει σε δύο καταχωρητές των 16 ψηφίων και τα αποστέλλει στην επόμενη λειτουργική μονάδα σε δύο κύκλους του ρολογιού . Πρώτα αποστέλλει τα πιο σημαντικά ψηφία και στο δεύτερο κύκλο τα λιγότερο σημαντικά ψηφία . Για να γίνεται αυτός ο διαχωρισμός κατανοητός από τον USB μικροελεγκτή το δέκατο έκτο ψηφίο χρησιμοποιείται ως σημαία . Έτσι όταν το δέκατο έκτο ψηφίο είναι στο λογικό επίπεδο ‘1’ τα υπόλοιπα ψηφία αποτελούν τα σημαντικότερα ψηφία του δεδομένου εισόδου ενώ όταν ‘0’ τότε μεταφέρονται στην έξοδο τα λιγότερα σημαντικά ψηφία .Στη συνέχεια θα περιγράψουμε τον μικροελεγκτή και πώς πρέπει να δουλεύει η σύνδεση .

Η οικογένεια Usb μικροελεγκτών **EZ-USB FX2LP** προσφέρει αρκετούς τρόπους για να λαμβάνει δεδομένα :

- Τέσσερα προγραμματιζόμενα σημεία διεπαφής
- I^2C διεπαφή για μεταφορά δεδομένων
- FIFO (που λειτουργεί τόσο σε 'master' διασύνδεση όσο 'slave' διασύνδεση).

Για την διασύνδεση μεταξύ του Usb_microcontroller και του FPGA χρησιμοποιήσαμε την επιλογή 'Slave FIFOs'. Για να υλοποιηθεί αυτή η διασύνδεση έπρεπε να κατασκευάσουμε ένα καινούργιο κύκλωμα το οποίο θα λειτουργεί ως 'master' που θα λαμβάνει τα δεδομένα από το κύκλωμα 'project_100MSps' και θα τα στέλνει σε μια από τις τέσσερις διαθέσιμες FIFO που θα είναι σε 'Slave Mode'. Στο παρακάτω σχήμα παρουσιάζονται τα σήματα που είναι απαραίτητα για τη διασύνδεση:



-ΣΧΗΜΑ 3.19-

Είναι απαραίτητη και η εξήγηση αυτών των σημάτων:

- IFCLK(Interface Clock): Το ρολόι της διασύνδεσης μπορεί να είναι είτε εσωτερικό είτε να ελέγχεται από εξωτερική πηγή . ΤΟ IFCLK μπορεί να οδηγήσει ρολόγια από συχνότητα 5 MHz έως 48 MHz . Εμείς επιλέξαμε συχνότητα 40 MHz και να ελέγχεται από εξωτερική πηγή δημιουργίας ρολογιού.
- FLAGA : είναι μία από τις τέσσερις συνολικά σημαίες που είναι διαθέσιμες για την υπόδειξη αν η FIFO είναι πλήρης , άδεια ή 'programmable-level '.

Στην προκαθορισμένη λειτουργία το FLAGA υποδεικνύει το σήμα ‘programmable-level ’.

- FLAGB : είναι μία από τις τέσσερις συνολικά σημαίες που είναι διαθέσιμες για την υπόδειξη αν η FIFO είναι πλήρης , άδεια ή ‘programmable-level ’.Στην προκαθορισμένη λειτουργία σηματοδοτεί την κατάσταση πλήρης FIFO.
- FLAGC : είναι μία από τις τέσσερις συνολικά σημαίες που είναι διαθέσιμες για την υπόδειξη αν η FIFO είναι πλήρης , άδεια ή ‘programmable-level ’.Στην προκαθορισμένη λειτουργία σηματοδοτεί την κατάσταση άδεια FIFO
- FLAGD/SLCS : έχει διπλή σημασία είτε μπορεί να χρησιμοποιηθεί ως σημαία για την υπόδειξη της κατάστασης της FIFO είτε χρησιμοποιείται για την αποκοπή του Usb_microcontroller από το διάδρομο δεδομένων της FIFO ώστε για παράδειγμα να συνδεθούν άλλες συσκευές
- SLOE : όταν είναι ενεργό (SLOE= ‘0’) τότε τα δεδομένα στέλνονται από το δίαδρομο δεδομένων FD από μια FIFO προς την συσκευή που είναι σε ‘Master Mode’
- SLRD :επιτρέπει την συσκευή ‘Master Mode’ να διαβάσει τα δεδομένα από την FIFO
- SLWR : επιτρέπει την εγγραφή στην FIFO από την συσκευή που είναι σε ‘Master Mode’
- PKTEND : δίνει εντολή να προωθηθεί ένα πακέτο που έχει λιγότερα ψηφία από τα προκαθορισμένα, όπως έχει δηλωθεί στον καταχωρητή του μικροελεγκτή EPXAUTOINLENH: L
- FD[15:0] : είναι ο διάδρομος δεδομένων και είναι αμφίδρομος
- FIFOADR[1:0] : καθορίζει ποία από τις τέσσερις FIFO θα χρησιμοποιηθεί

Τα σήματα FLAGA , FLAGB , FLAGC , FLAGD/SLCS , SLOE , SLRD , SLWR είναι ενεργά όταν είναι στην κατάσταση ‘0’. Η διασύνδεση είναι μονόδρομη ,δηλαδή από την συσκευή ‘Master Mode’ προς το Usb_microcontroller, καθώς και αποκλειστική. Μεταξύ αυτών των δύο συσκευών μερικά από αυτά τα σήματα μπορούν να είναι σταθερά . Παραθέτουμε τις τιμές τους :

FLAGD/SLCS	‘0’
SLOE	‘1’
SLRD	‘1’
PKTEND	‘1’

-ΠΙΝΑΚΑΣ 3.17-

Οι Usb_microcontroller της κατηγορίας EZ-USB FX2LP παρέχουν τέσσερις FIFO, τις EP2 ,EP4 ,EP6 ,EP8. Οι EP2 ,EP6 μπορούν να λάβουν μέχρι και 1024 Bytes ενώ οι EP4 ,EP8 μέχρι 512 Bytes. Επιλέξαμε να χρησιμοποιήσουμε την EP2 γιατί θέλουμε κάθε φορά να στέλνουμε 1024 Bytes από δεδομένα ,έτσι θέσαμε στη διεύθυνση FIFOADR[1:0] την τιμή ‘00’.

Για να έχουν τα σήματα την παραπάνω σημασία καθώς και για να έχει η FIFO EP2 χώρο μνήμης 1024 Bytes πρέπει μερικοί εσωτερικοί καταχωρητές των μικροελεγκτών της κατηγορίας EZ-USB FX2LP να αρχικοποιηθούν με την κατάλληλη τιμή. Παραθέτουμε τους καταχωρητές , την λειτουργία τους και την τιμή που πρέπει να έχουν .

Ο πρώτος καταχωρητής είναι ο IFCONFIG που καθορίζει την διεπαφή του USB_microcontroller. Για να λειτουργήσει σωστά το κύκλωμα, όπως το έχουμε καθορίσει, θα πρέπει η τιμή του να είναι ‘00100011’ . Τα σημαντικότερα ψηφία του καταχωρητή αναλύονται παρακάτω. Τα δύο LSB καθορίζουν ότι το η διεπαφή που υλοποιείται είναι η SLAVE FIFO. Ενώ το MSB καθορίζει ότι το IFCLK θα συνδέεται εξωτερικά με ρολόι .

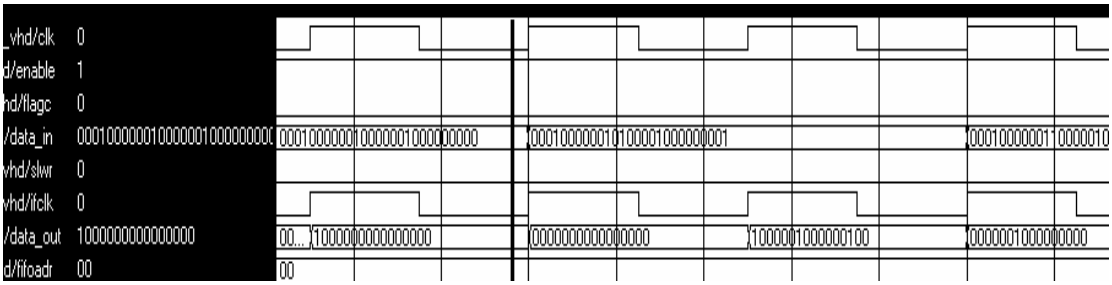
Οι καταχωρητές PINFLAGSAB και PINFLAGCD καθορίζουν την λειτουργία των σημάτων FLAGA , FLAGB , FLAGC , FLAGD . Οι τιμές που πρέπει να έχουν είναι οι προκαθορισμένες (Default) , δηλαδή ‘00000000’ .Για να είναι τα σήματα που μας ενδιαφέρουν ενεργά στη χαμηλή τιμή (active low) θα πρέπει ο καταχωρητής FIFOINPOLAR να έχει την τιμή ‘00000000’. Επειδή μας ενδιαφέρει η FIFO EP2 μέσω του EP2CFG καθορίζουμε την λειτουργία της και η τιμή που πρέπει να έχει είναι ‘11011010’. Με αυτή την τιμή έχουμε θέσει το EP2 να είναι ενεργό , να δέχεται δεδομένα , τα δεδομένα που λαμβάνει να είναι τύπου ‘ISOCHRONOUS’ και τέλος το μέγεθος των δεδομένων να είναι 1024 Bytes .

Ο επόμενος καταχωρητής που θέλει αρχικοποίηση είναι EP2FIFOCFG και η τιμή του θα πρέπει να είναι ‘00000101’ . Έχοντας αυτήν την τιμή ο καταχωρητής EP2FIFOCFG μπορούμε να στείλουμε τα δεδομένα συνεχώς . Σημαντικό είναι ότι ορίζοντας το LSB να έχει την τιμή ‘1’ καθορίζουμε ότι η είσοδος είναι 16 ψηφία (FD[15:0]) και όχι οκτώ . Οι δύο τελευταίοι καταχωρητές που έχουν σημασία είναι ο EP2AUTOINLENH και EP2AUTOINLENL , η τιμή του πρώτου πρέπει να είναι ‘00000100’ ενώ του δεύτερου πρέπει να είναι ‘00000000’ , οπότε η EP2 FIFO θα μπορεί να λαμβάνει δεδομένα των 1024 Bytes .Συνοψίζοντας, παρουσιάζουμε τους σημαντικότερους καταχωρητές του EZ-USB FX2LP που επηρεάζουν τη λειτουργία τις διεπαφής που υλοποιούμε και τις κατάλληλες τιμές που πρέπει να έχουν:

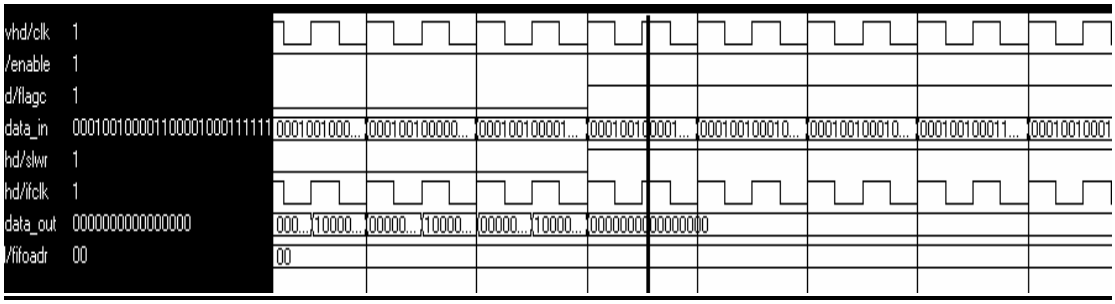
Καταχωρητής (Register)	**Τιμή**
IFCONFIG	‘00100011’
PINFLAGSAB / PINFLAGCD	‘00000000’
EP2CFG	‘11011010’
EP2FIFOCFG	‘00000101’
EP2AUTOINLENH	‘00000100’
EP2AUTOINLENL	‘00000000’

-ΠΙΝΑΚΑΣ 3.18-

Για να συμπληρώσουμε τη μελέτη του κυκλώματος παραθέτουμε τα πραγματικά διαγράμματα χρονισμού :



-ΣΧΗΜΑ 3.20α-



-ΣΧΗΜΑ 3.20β-

Στο πρώτο σχήμα παρατηρούμε την είσοδο και την έξοδο όταν τα σήματα ελέγχου έχουν την κατάλληλη τιμή για να εξέλθουν τα δεδομένα . Η μόνη παρατήρηση είναι ότι τα σωστά δεδομένα εξέρχονται με καθυστέρηση ενός κύκλου ρολογιού . Στο δεύτερο σχήμα παρατηρούμε τη συμπεριφορά του συστήματος όταν FLAGC = '1' , τότε η έξοδος αυτομάτως γίνεται μηδέν με αποτέλεσμα να χάνονται δεδομένα . Αυτό όμως δεν είναι μειονέκτημα γιατί ο USB μικροελεγκτής είναι σχεδόν απίθανο να βρεθεί σε αυτή την κατάσταση ,λόγω κατάλληλου σχεδιασμού.

3.12 : Παρατηρήσεις

Στο σημείο αυτό είναι χρήσιμο να σημειωθούν μερικές παρατηρήσεις τόσο για τον κώδικα που έχει γραφτεί όσο και για το ίδιο το κύκλωμα .

Ο κώδικας που περιγράφει το παραπάνω σύστημα εκμεταλλεύεται πλήρως τις δυνατότητες που παρέχει η VHDL . Οι λειτουργικές μονάδες έχουν περιγραφεί με συνδυασμό και των τριών διαθέσιμων τρόπων περιγραφής ενός ψηφιακού κυκλώματος που παρέχει η VHDL . Ένα σημαντικό χαρακτηριστικό του κώδικα και ουσιαστικά του κυκλώματος , είναι η δυνατότητα εύκολης διαμόρφωσής του , ώστε να λαμβάνει δεδομένα μεταβλητού μεγέθους . Επίσης διαμορφώσιμο είναι και το μέγεθος του μετασχηματισμού Φουριέ . Αρκεί η αλλαγή των σταθερών N και K που βρίσκονται στο 'generic' του κάθε κυκλώματος , και η κατάλληλη προσαρμογή των κυκλωμάτων που παρέχει η Xilinx , το FFT, το cordic και η RAM .

Μια σημαντική παρατήρηση είναι , ότι η λειτουργική μονάδα που μειώνει τη συχνότητα λειτουργίας του κυκλώματος είναι το ‘‘taxinomisi_reg’’ . Σε περίπτωση μελλοντικής επέκτασης θα πρέπει να ληφθεί σοβαρά υπόψη ότι το κύκλωμα δε μπορεί να υπερβεί τη συχνότητα λειτουργίας 112 MHz.

Κεφάλαιο 4 : Ολοκληρωμένη Λειτουργία Τελικού Κυκλώματος

4.1 :Προδιαγραφές Τελικού Κυκλώματος

Στόχος αυτής της διπλωματικής εργασίας ήταν η κατασκευή ενός συστήματος λήξης και επεξεργασίας σε πραγματικό χρόνο σήματος σε υπερύψηλες συχνότητες και με υψηλό ρυθμό δειγματοληψίας. Τα βασικά χαρακτηριστικά του συστήματός μας είναι:

- Ρυθμός δειγματοληψίας 100MSps με ανάλυση-ευκρίνεια 12 ψηφίων .
- Επεξεργασία (Πραγματικού Χρόνου) 100 MHz
- Ευκρίνεια συχνότητας 25 KHz

Για να επιτύχουμε τις παραπάνω προδιαγραφές χρησιμοποιήσαμε υλικά τεχνολογικής αιχμής όπως ο Α/Ψ μετατροπέας AD12401 και το FPGA Virtex 4. Η ευκρίνεια 25 KHz που απαιτείται για τη συχνότητα , επιτεύχθηκε με μετασχηματισμό Φουριέ 4096 σημείων.

Επιπλέον χαρακτηριστικά στο σύστημά μας προσθέτουν οι διεπαφές του κυκλώματος με έναν PLL Synthesizer και με έναν USB μικροελεγκτή .Οι αρχικές προδιαγραφές που είχαν τεθεί για τη σύνδεση με τον PLL Synthesizer έχουν υλοποιηθεί με αυστηρότητα, η μόνη διαφορά είναι ότι η διάρκεια του όγδοου καναλιού είναι 110 nsec αντί 100 nsec. Αυτό όμως δεν είναι σημαντικό πρόβλημα , γιατί το όγδοο κανάλι είναι το τελευταίο και γι'αυτό το λόγο δε θα δημιουργηθεί πρόβλημα στη λειτουργία του PLL Synthesizer.

Επιπλέον για να υλοποιήσουμε τη διασύνδεση με τον USB μικροελεγκτή θέσαμε μερικές προδιαγραφές για τη χρονική στιγμή της διέλευσης των δεδομένων προς τον μικροελεγκτή και για τη χρονική διάρκεια του κάθε δεδομένου. Έτσι κάθε φορά που έχουν αποθηκευτεί 256 νέα δεδομένα το σύστημά μας τα εξάγει .Η τελευταία προδιαγραφή που έχει υλοποιηθεί είναι ότι κάθε εξερχόμενο δεδομένο πρέπει να έχει διάρκεια 50 nsec .

4.2 : Προσομοίωση του Τελικού Κυκλώματος

Το επόμενο στάδιο μετά την επιμέρους σύνδεση των λειτουργικών μονάδων στο κύκλωμα 'project_100MSps' που υλοποιεί το σύστημα είναι η σύνθεσή του και η προσομοίωσή του με τα κατάλληλα εργαλεία ,δηλαδή το Xilinx ISE και το Modelsim . Για να έχουμε πλήρη δεδομένα για το σύστημα προσομοιώσαμε το σύστημά μας με τρείς διαφορετικές εισόδους. Η πρώτη είσοδος αντιστοιχούσε σε ένα ημίτονο συχνότητας 25 MHz , η δεύτερη σε ένα τριγωνικό παλμό 12,5 MHz και η τρίτη είσοδος σε ένα τετραγωνικό παλμό 6,25 MHz. Για να παρατηρήσουμε το κύκλωμα καλύτερα στην προσομοίωση προσθέσαμε μερικά σήματα που θα απεικονίζουν την έξοδο των λειτουργικών μονάδων ''corxnindex'' και ''regist'' . Αυτά τα σήματα είναι x_index_out , platos_out ,reg_serial_out & rvd . Τέλος πρέπει να σημειωθεί ότι τα σημεία του μετασχηματισμού (n) αντιστοιχίζονται σε συχνότητες (f(n)) σύμφωνα με το τύπο :

$$f(n)=\frac{n}{(N\cdot T)}$$

Όπου N : συνολικός αριθμός σημείων ,
T : περίοδος δειγματοληψίας .

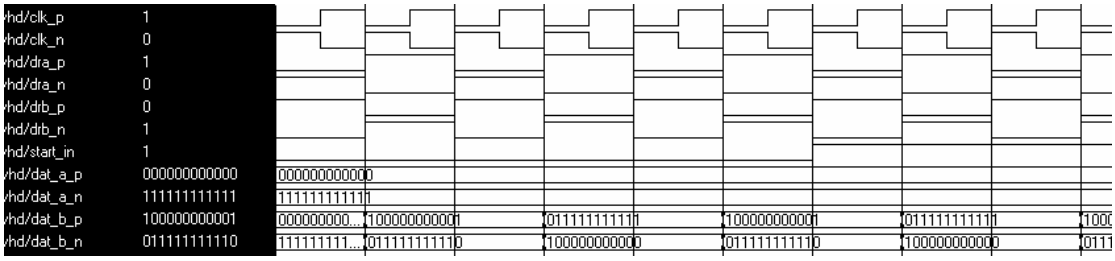
4.2.1 : Προσομοίωση ημιτονοειδούς εισόδου 25 MHz :

Όταν ο Α/Ψ μετατροπέας με δειγματοληπτική ικανότητα 100 MSps κάνει δειγματοληψία σε ένα ημίτονο συχνότητας 25 MHz , λαμβάνει σε κάθε περίοδο του ημιτόνου τέσσερις τιμές. Αν θεωρήσουμε ότι η πρώτη είναι η '000000000000' δηλαδή μηδέν τότε οι άλλες τρεις θα είναι '011111111111' , '000000000000' και τέλος '100000000001'.

Είσοδοι του Κυκλώματος(X_i 0=<i =<4095)	
X_0 ,X_4 ,X_8 ,X_{12} …….	'000000000000'
X_1 ,X_5 ,X_9 ,X_{13} …….	'011111111111'
X_2 ,X_6 ,X_{10} ,X_{14} …….	'000000000000'
X_3 ,X_7 ,X_{11} ,X_{15} …….	'100000000001'

--ΠΙΝΑΚΑΣ4.1 –

Από τη θύρα Α θα εισέρχονται τα δεδομένα X_0 ,X_2 ,X_4 , X_6 ,….. ενώ από τη θύρα Β X_1 ,X_3 ,X_5 , X_7 ,….. .Στο παρακάτω διάγραμμα χρονισμού φαίνονται οι είσοδοι του συστήματος :



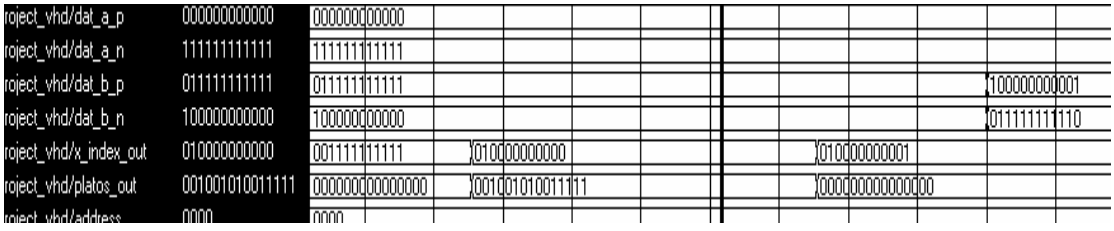
-ΣΧΗΜΑ 4.1-

Στη συνέχεια παρουσιάζουμε τα αποτελέσματα των δεδομένων εξόδου , αντιστοιχίζουμε τους δείκτες με τις κεντρικές συχνότητες και εξετάζουμε αν πληρούν τις απατήσεις που μας έχουν θέσει και αυτές που εμείς έχουμε θέσει για τη σωστή λειτουργία .

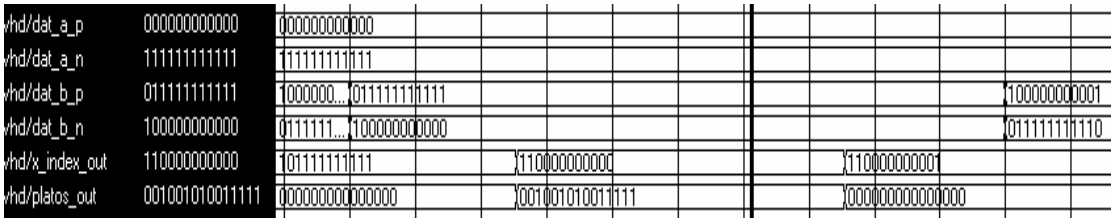
Μη Μηδενικά σημεία μετασχηματισμού

Σημεία Μετασχηματισμού	**Συχνότητα (MHz)**	**Ενέργεια**
‘010000000000’	25	001001010011111
‘110000000000’	75	001001010011111

--ΠΙΝΑΚΑΣ4.2 -



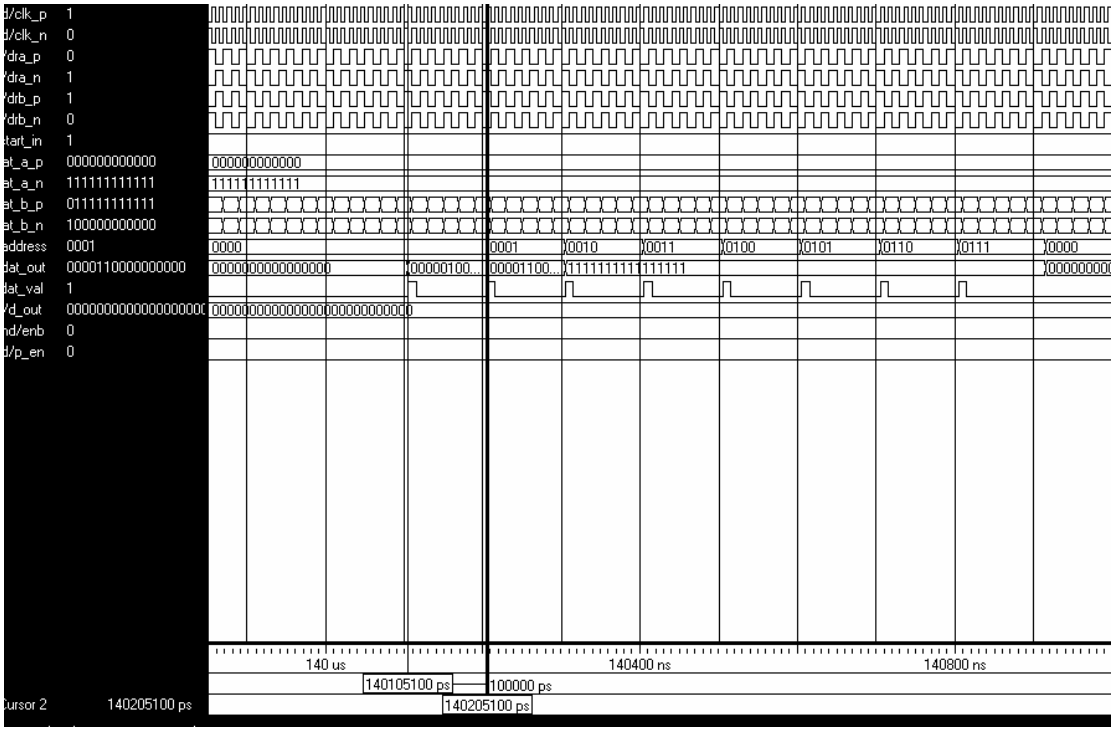
-ΣΧΗΜΑ 4.2-



-ΣΧΗΜΑ 4,3-

Στα διαγράμματα χρονισμού 4.2 και 4.3 παρατηρούμε την επαλήθευση του πίνακα 4.2 .

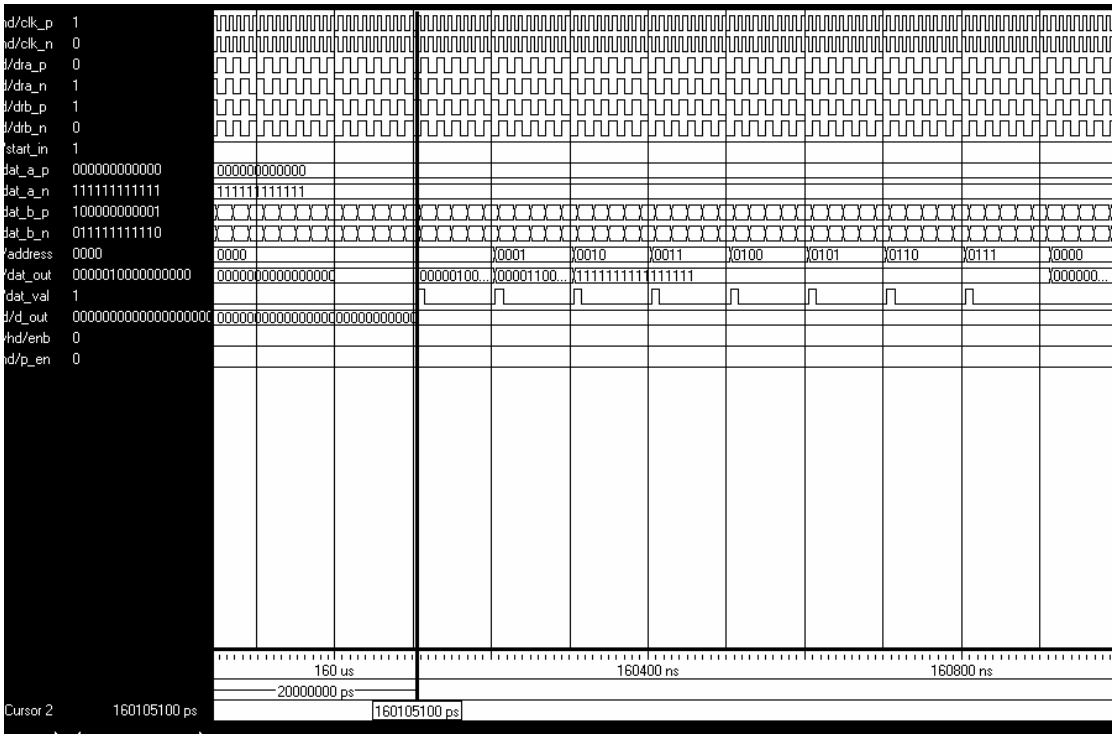
Στα επόμενα διάγραμμα 4.4 4.5 παρουσιάζουμε τα διάγραμμα χρονισμού της εξόδου που συνδέει το κύκλωμα με το PLL synthesizer και παρατηρούμε αν οι προδιαγραφές πληρούνται :



-ΣΧΗΜΑ 4.4-

Στο διάγραμμα 4.4 τα δεδομένα είναι στη σωστή σειρά με τον κατάλληλο χρονισμό , δηλαδή η απόσταση των δεδομένων είναι 100 nsec που βρίσκεται σε συμφωνία με τις προδιαγραφές του συστήματος . Επίσης παρατηρούμε ότι όταν δεν υπάρχουν δεδομένα στέλνεται η λέξη '1111111111111111'.

Στη συνέχεια στο 4.5 σχήμα παρουσιάζεται η χρονική απόσταση που στέλνονται δεδομένα προς τον PLL synthesizer και διαπιστώνεται ότι η απόσταση είναι 20 μsec, όπως απαιτείται .



-ΣΧΗΜΑ 4.5-

4.2.2 :Προσομοίωση του συστήματος με είσοδο τριγωνικό παλμό συχνότητας 12.5 MHz :

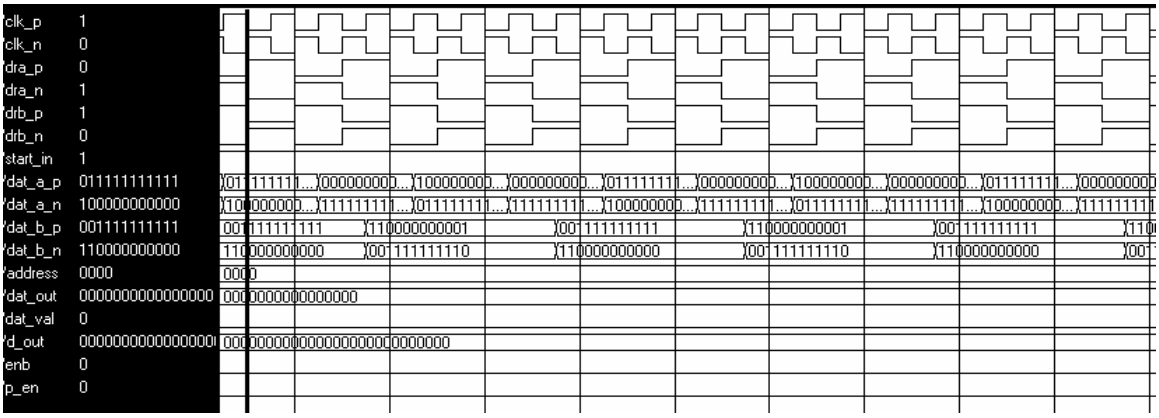
Η δεύτερη προσομοίωση έχει ως είσοδο έναν τριγωνικό παλμό με περίοδο 80 nsec, δηλαδή είναι ακέραιο πολλαπλάσιο της συχνότητας λήψης και επεξεργασίας του κυκλώματος . Ο Α/Ψ μετατροπέας AD12401 θα αποστέλλει για κάθε περίοδο του τριγωνικού παλμού οκτώ δεδομένα :

Είσοδοι του Κυκλώματος(X_i 0=<i =<4095)	
X_0 ,X_8, X_{16} ,……	‘000000000000’
X_1 ,X_9, X_{17} ,……	‘001111111111’
X_2 ,X_{10}, X_{18} ,……	‘011111111111’

X_3 ,$X_{11,}$ X_{19} ,……	‘001111111111’
X_4 ,X_{12} , X_{20} ,……	‘000000000000’
X_5 ,$X_{13,}$ X_{21} ,……	‘110000000001’
X_6 ,$X_{14,}$ X_{22} ,……	‘100000000001’
X_7 ,$X_{15,}$ X_{23} ,……	‘110000000001’

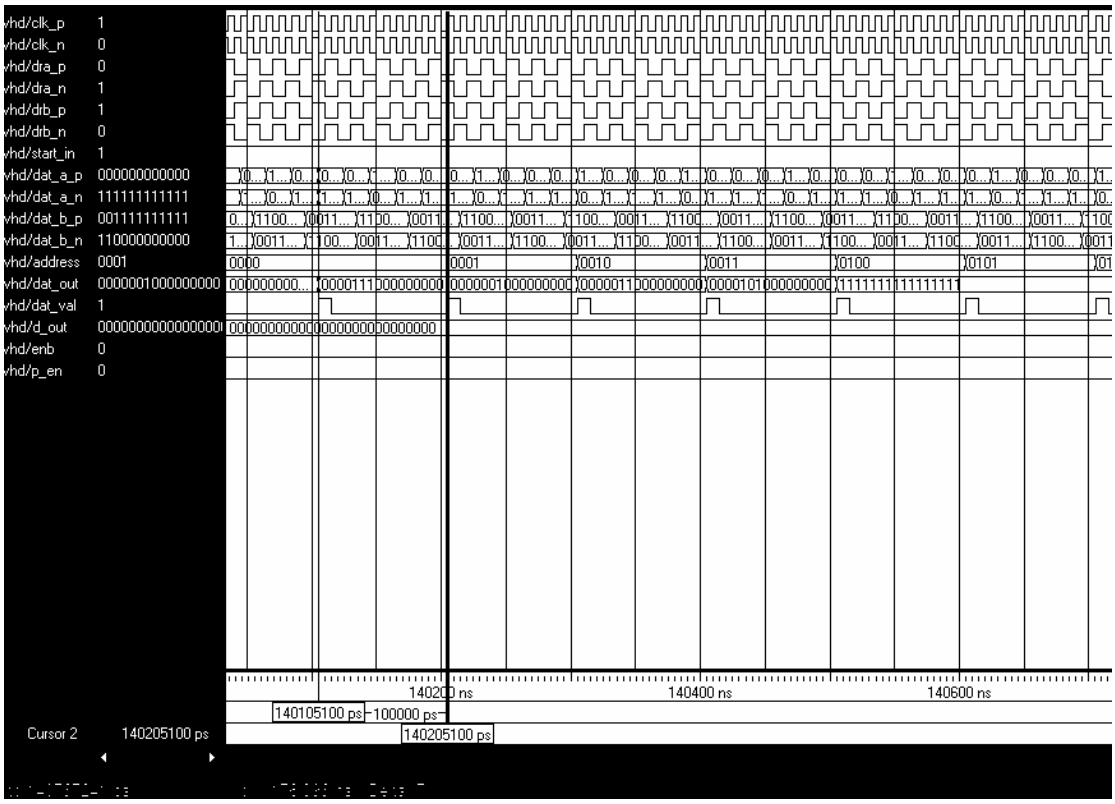
--ΠΙΝΑΚΑΣ4.3 -

Από τη θύρα A θα εισέρχονται τα δεδομένα X_0 ,X_2 ,X_4 , X_6 ,….. ενώ από τη θύρα B X_1 ,X_3 ,X_5 , X_7 ,….. .Στο παρακάτω διάγραμμα χρονισμού φαίνονται οι είσοδοι του συστήματος :



-ΣΧΗΜΑ 4.6-

Στη συνέχεια παρουσιάζουμε τα δεδομένα που εξέρχονται από τη διασύνδεση με το PLL για να εμφανίσουμε ποιά σημεία του μετασχηματισμού έχουν μη μηδενική ενέργεια :



-ΣΧΗΜΑ 4.7-

Από το σχήμα 4.7 παρατηρούμε ότι στα κανάλια του PLL synthesizer στέλνονται με σειρά προτεραιότητας οι δείκτες , '111000000000', '001000000000' , '011000000000' , '101000000000'. Στον πίνακα παρουσιάζουμε σε ποιές συχνότητες αντιστοιχούν τα παραπάνω σημεία της επεξεργασίας του σήματος :

Σημεία Μετασχηματισμού	**Συχνότητα (MHz)**	**Ενέργεια**
'001000000000'	12.5	000111111100011
'011000000000'	37,5	000001010111100
'101000000000'	62,5	000001010111011
'111000000000'	87.5	000111111100101

Δείκτες που εμφανίζονται στο PLL synthesizer κατά σειρά εξόδου

111000000000

001000000000

011000000000

101000000000

--ΠΙΝΑΚΑΣ4.4 -

Τα παραπάνω αποτελέσματα συμπίπτουν με τα επιθυμητά (Behavioral simulation) καθώς και με την μαθηματική θεωρία Φουριέ αφού ένας τριγωνικός παλμός με συχνότητα 12.5 MHz θα έχει στο φάσμα του αυτές τις συχνότητες

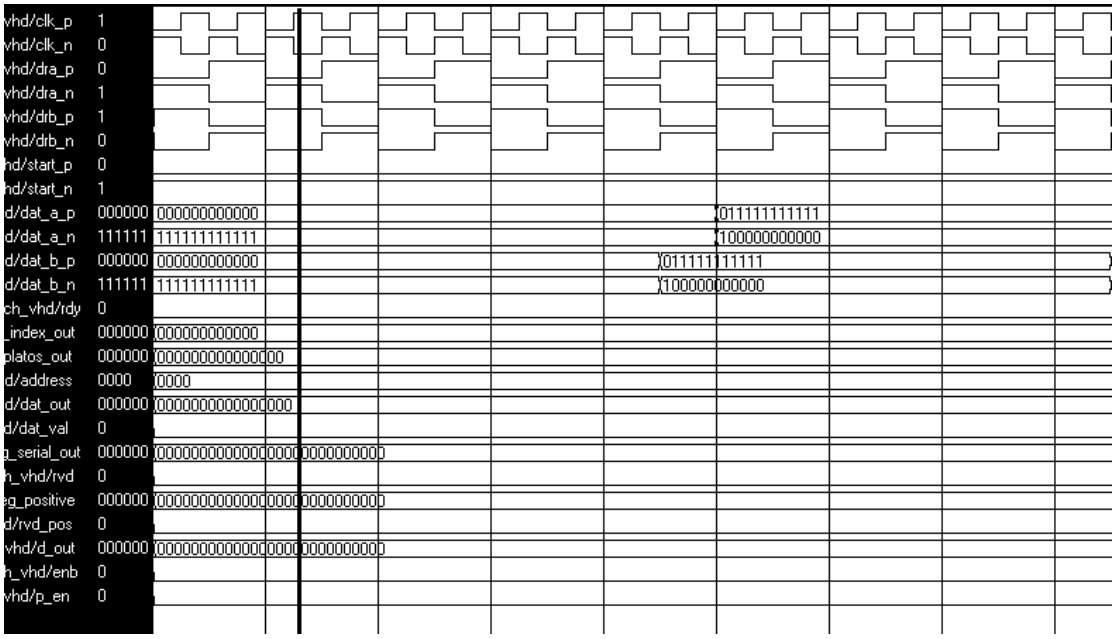
4.2.3 : Προσομοίωση του συστήματος με είσοδο τετραγωνικό παλμό συχνότητα 6,25 MHz :

Στην τρίτη και τελευταία προσομοίωση είχαμε είσοδο , σήμα με αρκετές αρμόνικες συχνότητες . Αφού το σήμα είναι ένας τετραγωνικός παλμός συχνότητας 6,25 MHz οι είσοδοι του FPGA θα είναι περιοδικά οι εξής :

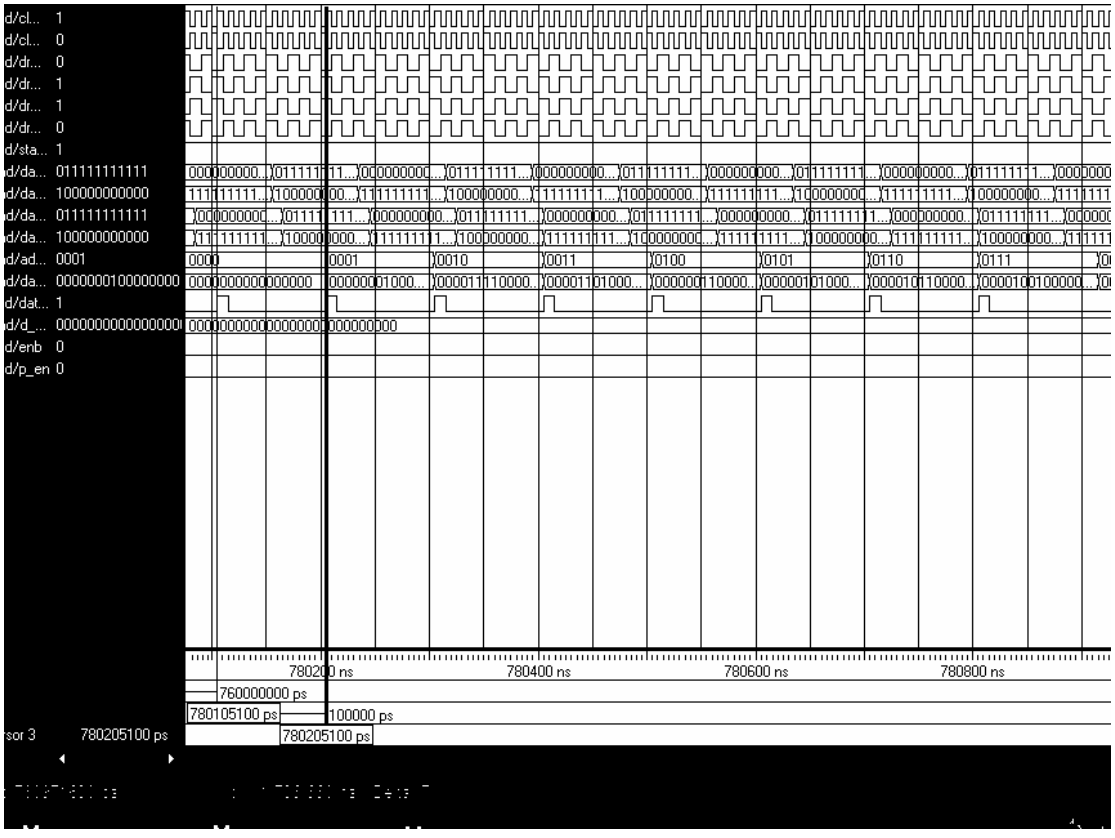
Είσοδοι	
dat_a_p	**dat_b_p**
'111111111111'	'0111111111111'
'0111111111111'	'0111111111111'
'0111111111111'	'0111111111111'
'0111111111111'	'0111111111111'
'0000000000000'	'0000000000000'
'0000000000000'	'0000000000000'
'0000000000000'	'0000000000000'
'0000000000000'	'0000000000000'

--ΠΙΝΑΚΑΣ4.5 –

Στη συνέχεια παρουσιάζουμε τα διαγράμματα της προσομοίωσης .Αρχικά παρουσιάζουμε στο πρώτο διάγραμμα τα δεδομένα εισόδου που είναι όπως τα παραπάνω :



-ΣΧΗΜΑ 4.8-



-ΣΧΗΜΑ 4.9-

Στο δεύτερο διάγραμμα παρουσιάζουμε τα αποτελέσματα της προσομοίωσης για τη διασύνδεση του FPGA με το PLL synthesizer. Όπως φαίνεται εξάγονται οκτώ δεδομένα, τα οκτώ κανάλια με τις μεγαλύτερες ενέργειες .Θα πρέπει να σημειώσουμε ότι ο τετραγωνικός παλμός έχει εννέα δείκτες σημείων

μετασχηματισμού με μη μηδενική ενέργεια αλλά όπως είναι λογικό από την κατασκευή του κυκλώματος μόνο οι οκτώ δείκτες με τη μεγαλύτερη ενέργεια θα εμφανιστούν . Στον παρακάτω πίνακα παρουσιάζουμε όλα τα σημεία με μη μηδενική ενέργεια καθώς και ποιά έχουν εμφανιστεί στην έξοδο για το PLL synthesizer :

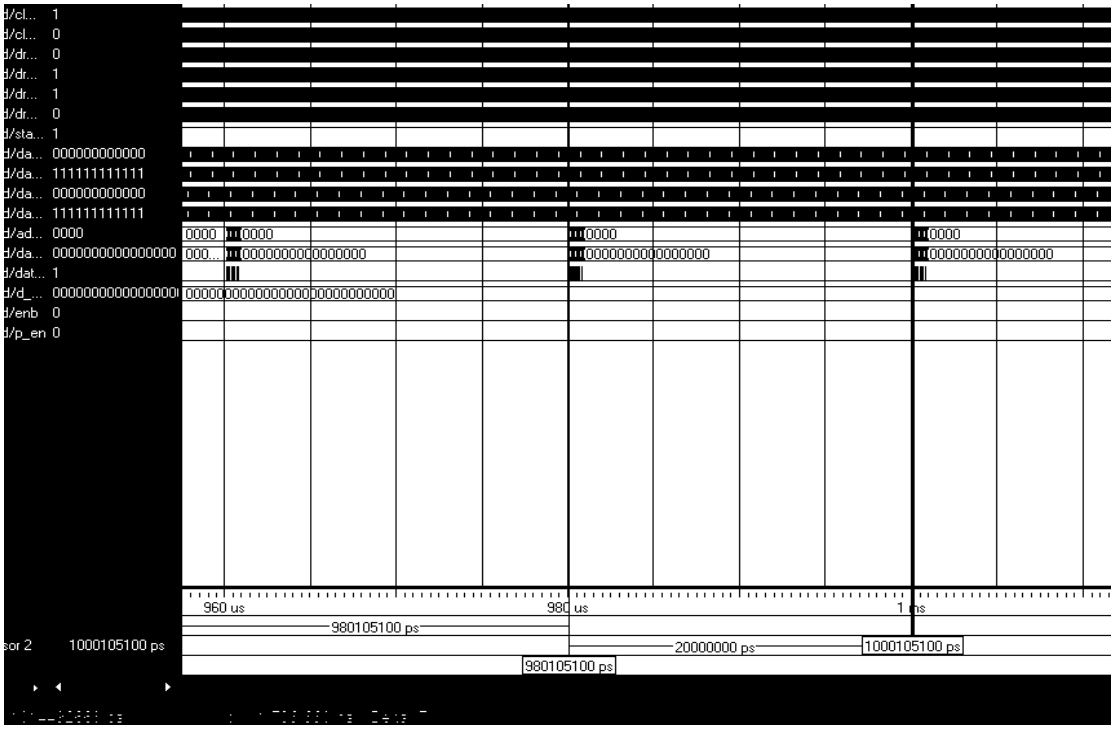
Δείκτης Σημείου Μετασχηματισμού	**Συχνότητα (MHz)**	**Ενέργεια**
000000000000	0	001001010011111
000100000000	6.25	000101111110000
001100000000	18.75	000010000110000
010100000000	31.25	000001011001101
011100000000	43.75	000001001100000
100100000000	56.25	000001001100000
101100000000	68.75	000001011001101
110100000000	81.25	000010000110000
111100000000	93.75	000101111110000

Δείκτες που εμφανίζονται στο PLL synthesizer κατά σειρά εξόδου

000000000000
000100000000
111100000000
001100000000
110100000000
010100000000
101100000000
011100000000

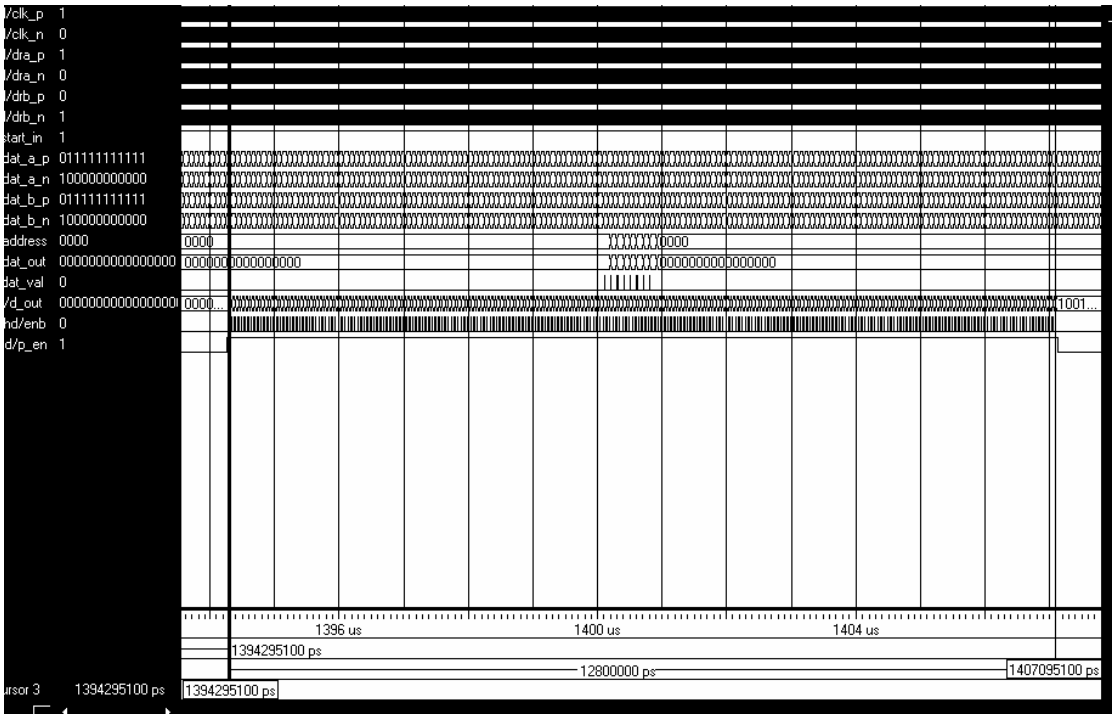
--ΠΙΝΑΚΑΣ4.6 -

Στο σχήμα 4.10 φαίνεται η χρονική διαφορά μεταξύ δύο ενεργοποιήσεων της εξόδου που στέλνει δεδομένα προς το PLL synthesizer . Η χρονική διαφορά είναι 20 μsec , η οποία είναι η αναμενόμενη .

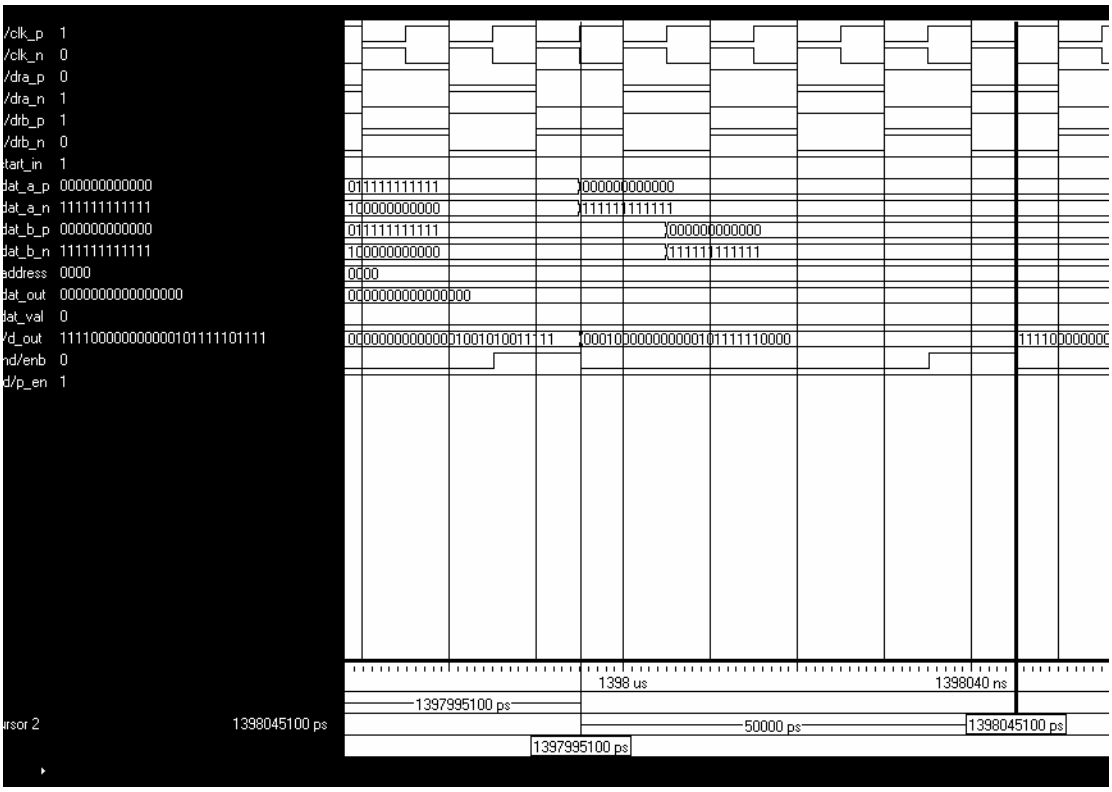


-ΣΧΗΜΑ 4.10-

Στην συνέχεια παρατίθενται δύο διαγράμματα που παρουσιάζουν τη σωστή λειτουργία του κυκλώματος ‘‘Control_of_Ram’’ . Από τα διαγράμματα χρονισμού παρατηρείται η ορθότητα της λειτουργίας της ‘‘Control_of_Ram’’ , που είναι σε συμφωνία με τις προδιαγραφές που έχουν τεθεί .Στο σχήμα 4.11 φαίνεται η χρονική διαφορά μεταξύ του πρώτου δεδομένου με το τελευταίο που είναι 1280 nsec .Άρα τα δεδομένα που μεταφέρονται είναι 256 . Στο τελευταίο σχήμα 4.12 παρατηρούμε την χρονική διάρκεια του κάθε δεδομένου στην έξοδο ‘d_out’ του κυκλώματος , η οποία είναι 50 nsec.



-ΣΧΗΜΑ 4.11-



-ΣΧΗΜΑ 4.12-

4.3 :Χαρακτηριστικά του Κυκλώματος

Το εργαλείο εξομοίωσης της λειτουργίας του FPGA , ISE Xilinx ,παρέχει όλες τις απαραίτητες πληροφορίες για το κύκλωμα . Στην συνέχεια παρατίθενται πληροφορίες από το ISE που πληροφορούν για την κατάληψη του FPGA και για την μέγιστη συχνότητα λειτουργίας.

.

PROJECT_100 Project Status			
Project File:	PROJECT_100.ise	**Current State:**	Translated
Module Name:	project_100MSps	• **Errors:**	No Errors
Target Device:	xc4vsx35-12ff668	• **Warnings:**	411 Warnings
Product Version:	ISE, 8.1.03i	• **Updated:**	Τετ 15. Νοε 19:24:58 2006

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	5241	15360	34%
Number of Slice Flip Flops	8190	30720	26%
Number of 4 input LUTs	7863	30720	25%
Number of bonded IOBs	105	448	23%
Number of FIFO16/RAMB16s	26	192	13%
Number of GCLKs	1	32	3%
Number of DSP48s	15	192	7%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Τετ 15. Νοε 19:22:22 2006	0	18 Warnings	56 Infos
Translation Report	Current	Τετ 15. Νοε 19:24:55 2006	0	393 Warnings	0

-ΣΧΗΜΑ 4.13-

Το σχήμα 4.13 μας πληροφορεί για την κατάληψη του FPGA όταν υλοποιείται το κύκλωμα ‘‘project_100MSps’’ .

Τέλος είναι σημαντική η παράθεση της μέγιστης συχνότητας στην οποία μπορεί να λειτουργήσει το κύκλωμα. Παρατίθεται η πληροφορία που παρέχεται από το ISE :

Timing Summary:

Speed Grade: -12

Minimum period: 8.900ns (Maximum Frequency: 112.354MHz)

Minimum input arrival time before clock: 1.905ns

Maximum output required time after clock: 5.296ns

Maximum combinational path delay: No path found

Άρα η μέγιστη συχνότητα λειτουργίας είναι 112.354MHz . Πρέπει να σημειωθεί ότι η συχνότητα λειτουργίας του ‘‘project_100MSps’’ συμπίπτει με την μέγιστη συχνότητα λειτουργίας του υποκυκλώματος ‘‘taxinomisi_reg’’ .Αυτό σημαίνει ότι η λειτουργική μονάδα ‘‘taxinomisi_reg’’ μειώνει την συχνότητα λειτουργίας του ολόκληρου του κυκλώματος .Όλες οι άλλες λειτουργικές μονάδες λειτουργούν σε υψηλότερη συχνότητα , τουλάχιστον μεγαλύτερες από 225 MHz .

Κεφάλαιο 5 : Μελλοντική Εργασία - Συμπεράσματα

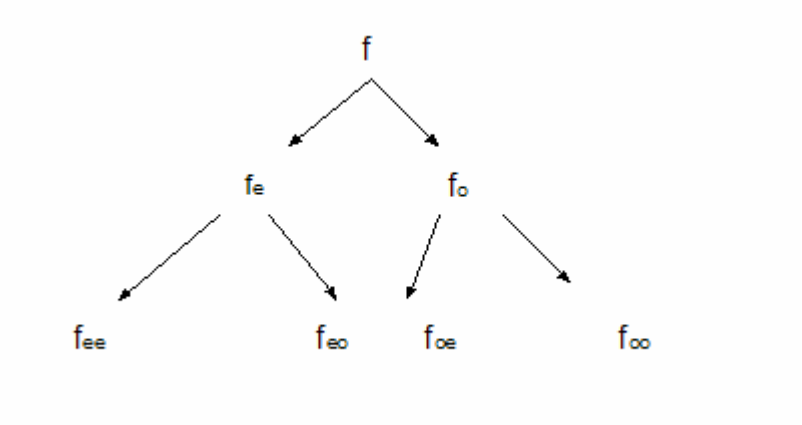
5.1 :Μελλοντική Εργασία

Είναι σημαντικό να παρατεθεί ένας μελλοντικός στόχος . Αυτός είναι η πλήρης εκμετάλλευση της δειγματοληπτικής ικανότητας του Α/Ψ μετατροπέα AD12401 που είναι 400MSps .Για να επιτευχθεί αυτός ο στόχος θα πρέπει το σύστημά να επεξεργάζεται δεδομένα με συχνότητα 400 MHz . Αυτό όμως δεν είναι τόσο πραγματοποιήσιμο , γιατί αρκετές βασικές μονάδες δεν λειτουργούν σε συχνότητα ρολογιού 400 MHz , όπως είναι το κύκλωμα ‘‘ taxinomisi_reg’’ που δουλεύει με συχνότητα το πολύ 112 MHz.

Στη συνέχεια θα παρατεθούν δύο ιδέες που θα βελτιώσουν μελλοντικά το κύκλωμα . Η πρώτη ιδέα αφορά τη βελτίωση του κυκλώματος ‘‘taxinomisi_reg ’’ για να λειτουργεί σε συχνότητα τουλάχιστον 200 MHz. Για να γίνει αυτό θα πρέπει να σχεδιαστεί με pipelined αρχιτεκτονική , δηλαδή να διασπαστεί σε επιμέρους τμήματα – λειτουργικές μονάδες, που θα υλοποιούνται σειριακά. Οι νέες λειτουργικές μονάδες θα είναι μικρότερες και απλούστερες του κυκλώματος taxinomisi_reg που έχουμε παρουσιάσει σε αυτήν την εργασία. Έτσι θα υπάρχουν ειδικές μονάδες για την ανίχνευση του θορύβου, την αποθήκευση των δεδομένων και την ταξινόμηση. Η ταξινόμηση θα μπορούσε να γινόταν με οχτώ διαφορετικά κυκλώματα. Στο πρώτο κύκλωμα θα είναι προσωρινά αποθηκευμένος ο καταχωρητής με τη μικρότερη ενέργεια, όταν ένα νέο δεδομένο απαιτεί ταξινόμηση θα εισέρχεται στο πρώτο κύκλωμα. Εάν είναι μεγαλύτερος από το περιεχόμενο του καταχωρητή τότε θα αποθηκεύεται στον πρώτο καταχωρητή και θα αποστέλλεται στην επόμενο κύκλωμα για την δεύτερη σύγκριση αλλιώς η διαδικασία ταξινόμηση θα σταματάει σε αυτό το σημείο . Αυτή η διαδικασία θα είναι συνεχής και θα υλοποιείται σειριακά.

Η δεύτερη ιδέα αφορά την εκμετάλλευση των ιδιοτήτων του FFT. Όταν αυξάνει το κύκλωμα τότε Virtex 4 λειτουργεί σε μικρότερες συχνότητες από αυτές που λειτουργούν τα επιμέρους κυκλώματα. Με αυτήν τη λογική για να έχουμε σίγουρη επεξεργασία και πλήρη εκμετάλλευση της δειγματοληπτικής ικανότητας του Α/Ψ μετατροπέα θα πρέπει η συχνότητα λειτουργίας να είναι μικρότερη από την

μέγιστη δυνατή. Θα χρησιμοποιήσουμε την αρχή υλοποίησης του Γρήγορου Μετασχηματισμού Φουριέ . Ο αλγόριθμος του FFT βασίζεται στον συνεχή διαχωρισμό του διακριτού σήματος σε περιττά και ζυγά μέρη :



-Σχήμα 5.1-

Αυτό συμβαίνει συνεχώς έως ότου όλοι οι αριθμοί να είναι ομαδοποιημένοι σε ζεύγη. Στην συνέχεια υπολογίζεται ο ΔΜΦ με βάση την σχέση :

$$Df(n) = Df_e(n) + w^n{}_N\, Df_o(n) \quad , 0 =< n <= N/2 - 1 \quad , \text{όπου } w_N = e^{-2\pi / N}$$

$$Df(N/2 + n) = Df_e(n) - w^n{}_N\, Df_o(n) \quad , 0 =< n <= N/2 - 1$$

Με αυτήν την παρατήρηση μπορούμε να εκμεταλλευτούμε την διπλή έξοδο του Α/Ψ μετατροπέα , έτσι ώστε τα δεδομένα του Α/Ψ μετατροπέα να εισέρχονται σε δύο ξεχωριστά κυκλώματα FFT . Στη συνέχεια θα υλοποιείται ένα κύκλωμα που θα υπολογίζει το τελευταίο στάδιο του FFT , δηλαδή τις δύο παραπάνω σχέσεις . Με αυτόν τον τρόπο θα εισέρχονται νέα δεδομένα με ρυθμό 400 MHz αλλά το κύκλωμα ουσιαστικά θα λειτουργεί με ρολόι 200 MHz . Αυτή η υποβάθμιση της συχνότητας θα μπορούσε να επεκταθεί και αντί να λειτουργεί το σύστημα στα 200 MHz , να λειτουργεί στα 100 MHz. Αυτό θα γινόταν πραγματικότητα με επιπλέον διαχωρισμό των δεδομένων σε ζεύγη και την υλοποίηση τεσσάρων FFT που θα λειτουργούν με συχνότητα 100 MHz.

5.2 : Συμπεράσματα

Το ψηφιακό κύκλωμα που σχεδιάστηκε και παρουσιάστηκε σε αυτήν την εργασία είναι ένα πλήρες σύστημα υπέρ ταχείας λήψης και επεξεργασίας ενός αναλογικού σήματος .Επίσης η επεξεργασία είναι πραγματικού χρόνου .Η υψηλή δειγματοληπτική ικανότητα(100 MSps) και η γρήγορη επεξεργασία του σήματος (100 MHz) , σε συνδυασμό με την επικοινωνία με PLL Synthesizer και με USB Microcotroller το καθιστά ένα ολοκληρωμένο σύγχρονο σύστημα. Πλεονέκτημα του συστήματος είναι ότι υλοποιείται από κώδικα εύκολο στη διαμόρφωση του και στην επέκταση .

Η βασική επεξεργασία είναι η σάρωση όλου του φάσματος συχνοτήτων και η εύρεση των οκτώ συχνοτήτων του σήματος με την μεγαλύτερη ενέργεια . Με αυτήν την πληροφορία θα μπορούσαν οκτώ PLL να κλειδώνουν σε αυτές τις συχνότητες και να ελέγχουν την πληροφορία που μεταφέρει το σήμα. Αυτή η επεξεργασία είναι χρήσιμη σε ένα εύρος από εφαρμογές. Επιπλέον η αποθήκευση των δεδομένων σε μνήμη και η διασύνδεση με USB μικροελεγκτή προσφέρει τη δυνατότητα διασύνδεσης με υπολογιστή για περαιτέρω επεξεργασία και ανάλυση του σήματος .

Οι συσκευές που έχουν χρησιμοποιηθεί για την πραγματοποίηση εγγυούνται την σωστή λειτουργία αυτού του κυκλώματος . Οι δυνατότητες του Α/Ψ μετατροπέα AD12401 , του FPGA Virtex 4 και του USB Microcontroller CY7C68013A είναι τέτοιες που μπορεί μελλοντικά να προστεθούν επιπλέον λειτουργίες και να αυξηθεί τόσο η δειγματοληπτική ικανότητα όσο και η συχνότητα λειτουργίας του συστήματος .

ΠΑΡΑΡΤΗΜΑ Α
Κώδικες:

project_100MSps:

```
library IEEE;
Library UNISIM;
use UNISIM.vcomponents.all;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity project_100MSps is
GENERIC(N : integer :=14;--megethos lexis se bit
      K :integer :=11 --megethos se bit twn x_index meiwmeno kata 1

);
port (


          clk_p: IN std_logic;
  clk_n: IN std_logic;

          dra_p : IN std_logic;
          dra_n : IN std_logic;
          drb_p: IN std_logic;
          drb_n : in std_logic ;
          -- ta dedomena eisodou
           dat_a_p :in std_logic_vector(11 downto 0);
           dat_a_n :in std_logic_vector(11 downto 0);
           dat_b_p :in std_logic_vector(11 downto 0);
           dat_b_n :in std_logic_vector(11 downto 0);
           start_in: IN std_logic;


   address : out std_logic_vector(3 downto 0);
   dat_out: out std_logic_vector(15 downto 0);
   dat_val : out std_logic ;
```

```
            d_out : out std_logic_vector(N+K+1 downto 0);
            enb : out  STD_LOGIC;
            p_en:out std_logic
           );
end project_100MSps;

architecture Behavioral of project_100MSps is

component ad_contoller
port (
 clk:in std_logic; --roloi gia 400 MHz
 dready_A ,dready_B :in std_logic;--ta rologia eisodou
                                  --twn dedomenwn
 dat_a , dat_b :in std_logic_vector(11 downto 0);
 -- ta dedomena eisodou
 datout : out std_logic_vector(13 downto 0)
 -- ta dedomena exodou

);
end component ad_contoller ;


---------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component fft
            port (
            xn_re: IN std_logic_VECTOR(13 downto 0);
            xn_im: IN std_logic_VECTOR(13 downto 0);
            start: IN std_logic;
            fwd_inv: IN std_logic;
            fwd_inv_we: IN std_logic;
            scale_sch: IN std_logic_VECTOR(11 downto 0);
            scale_sch_we: IN std_logic;
            clk: IN std_logic;
            xk_re: OUT std_logic_VECTOR(13 downto 0);
            xk_im: OUT std_logic_VECTOR(13 downto 0);
            xn_index: OUT std_logic_VECTOR(11 downto 0);
            xk_index: OUT std_logic_VECTOR(11 downto 0);
```

```
            rfd: OUT std_logic;
            busy: OUT std_logic;
            dv: OUT std_logic;
            edone: OUT std_logic;
            done: OUT std_logic);
end component;

-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of fft: component is "true";

-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of fft: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------
component compl2_to_1qnformat
generic (K: integer :=11;
  --orismos tou megethous tou x_index se sxesi me ton fft pou exoume
          N: INTEGER :=14 );
           --orismos tou megetous twn eisodwn kai exodwn twn dedomenwn
           --basi pali tou fft pou exoume
     port (
       x_re  : in std_logic_vector (N-1 downto 0 );
       x_im  : in std_logic_vector (N-1 downto 0 );
                              x_index : in std_logic_vector (K downto 0);

       x_re_1qf : out std_logic_vector(N downto 0);
                              x_im_1qf : out std_logic_vector(N downto 0);
       x_index_1qf : out std_logic_vector (K downto 0)
     );



end component compl2_to_1qnformat;

component  corxnindex is
generic(K : INTEGER :=11;
       N :INTEGER :=14);
 port (  x_re: in std_logic_VECTOR(N downto 0);
                x_im: IN std_logic_VECTOR(N downto 0);
```

```
            x_index_in : in std_logic_vector(K downto 0);
            clk: IN std_logic ;
            platos: OUT std_logic_VECTOR(N downto 0);
            rdy: OUT std_logic;
            x_index_out : out std_logic_vector(K downto 0)
        );
end component corxnindex;

component taxinomisi_reg is
generic (N:integer :=14;--to megethos twn registers
       K:integer :=11 );-- to megethos tou x_index
 port(clk : in std_logic;
   magnitude_in: in std_logic_vector(N downto 0);
   x_index_in : in std_logic_vector(K downto 0);

   --ta dedomena einai egkua
          out_en : out std_logic;
          reg1out : out std_logic_vector(N+K+1 downto 0);
   reg2out : out std_logic_vector(N+K+1 downto 0);
   reg3out : out std_logic_vector(N+K+1 downto 0);
   reg4out : out std_logic_vector(N+K+1 downto 0);
   reg5out : out std_logic_vector(N+K+1 downto 0);
   reg6out : out std_logic_vector(N+K+1 downto 0);
   reg7out : out std_logic_vector(N+K+1 downto 0);
   reg8out : out std_logic_vector(N+K+1 downto 0)
 );

end component taxinomisi_reg;


component PLL_SYNDESI is
generic (N:integer :=14;--to megethos twn registers pou exoun to platos
       K:integer :=11 );--to megethos twn xn_index , twn pin!!
port (
     clk :in std_logic ;
                  din_val : in std_logic ;

                  dat1 : in std_logic_vector(K+1 DOWNTO 0) ;
                  --to xn_index kai to 1 bit (to lsb )
            --pou deixnei an einai valid i oxi
```

```
                dat2 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat3 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat4 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat5 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat6 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat7 : in std_logic_vector(K+1 DOWNTO 0) ;
                dat8 : in std_logic_vector(K+1 DOWNTO 0) ;

                address : out std_logic_vector(3 DOWNTO 0) ;
                dat_out : out std_logic_vector(15 DOWNTO 0);
                dat_val : out std_logic

   );
end component PLL_SYNDESI;




component  regist is
generic (N:integer :=14;--to megethos twn registers
        K:integer :=11 );--to megethos tou t.p

   Port ( reg1_in : in std_logic_vector(N+K+1 downto 0);
        reg2_in : in std_logic_vector(N+K+1 downto 0);
        reg3_in : in std_logic_vector(N+K+1 downto 0);
        reg4_in : in std_logic_vector(N+K+1 downto 0);
        reg5_in : in std_logic_vector(N+K+1 downto 0);
        reg6_in : in std_logic_vector(N+K+1 downto 0);
        reg7_in : in std_logic_vector(N+K+1 downto 0);
        reg8_in : in std_logic_vector(N+K+1 downto 0);
        da_val : in  STD_LOGIC;
                        clk : in STD_LOGIC;



                        reg_serial_out : out std_logic_vector(N+K+1 downto 0);
        rvd :out std_logic -- ta dedomena exw einai valid
                        );
end component regist;
```

```
component no_negative_regs is
generic (N:integer :=14;--to megethos twn registers
        K:integer :=11 );--to megethos tou t.p
Port (
                clk: IN std_logic;
                        d_value : in std_logic ;--otan d_val='1' tote einai valid oi registers
                        reg_in : in std_logic_vector(N+K+1 downto 0);



       reg_serial_out : out std_logic_vector(N+K+1 downto 0);
       rvd :out std_logic -- ta dedomena exw einai valid
                                    );

end  component no_negative_regs;

component Control_Of_Ram is
generic(N:integer :=14;
       K:integer :=11);
Port (
                clk : in std_logic;
                reg_in : in  STD_LOGIC_VECTOR (N+K+1 downto 0);
         ena : in  STD_LOGIC;
         wea : in  STD_LOGIC;

                                    d_out : out  STD_LOGIC_VECTOR (N+K+1 downto 0);


                                    enb_en : out  STD_LOGIC;
         p_out :out STD_LOGIC
                                     );
end component Control_Of_Ram;




--constants for fft input
constant fwd_inv:  std_logic :='1';
constant fwd_inv_we:  std_logic:='1';
constant scale_sch:  std_logic_VECTOR(11 downto 0):= "101010101010";
```

```
constant scale_sch_we:  std_logic:='1';


--signal for adcontroller
signal clk : std_logic ;
signal  dready_A :std_logic ;
signal  dready_B :std_logic ;
signal dat_a :std_logic_vector(11 downto 0 );
signal dat_b :std_logic_vector(11 downto 0 );


--signals for fft input
signal xn_re_next : std_logic_vector(13 downto 0):=(others=>'0');
signal xn_im_reg : std_logic_vector(13 downto 0):=(others=>'0');
signal start : std_logic ;
--output signals for fft

signal   xk_re_fft:  std_logic_VECTOR(N-1 downto 0);
signal   xk_im_fft:  std_logic_VECTOR(N-1 downto 0);
signal    xn_index_fft:  std_logic_VECTOR(K downto 0);
signal    xk_index_fft:  std_logic_VECTOR(K downto 0);
signal    rfd_fft:  std_logic;
signal    busy_fft :  std_logic;
signal    dv_fft :  std_logic;
signal    edone_fft:  std_logic;
signal    done_fft:  std_logic ;

--output of translate1qntotwo
signal   xk_re_1qf :  std_logic_vector(N downto 0);
signal    xk_im_1qf :  std_logic_vector(N downto 0);
signal   xk_index_1qf :  std_logic_vector (K downto 0);

--output  cordix
signal rdy_cor  : std_logic ;
signal platos_cor : std_logic_vector(N downto 0);
signal x_index_cor :std_logic_vector(K downto 0);

--output taxinomisi
signal  out_en_tax : std_logic;
signal  reg1out_tax : std_logic_vector(N+K+1 downto 0);
```

```
signal  reg2out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg3out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg4out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg5out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg6out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg7out_tax : std_logic_vector(N+K+1 downto 0);
signal  reg8out_tax : std_logic_vector(N+K+1 downto 0);

--output seiriakoi registers :regist
signal  reg_serial_regist : std_logic_vector(N+K+1 downto 0);
signal  rvd_regist :std_logic;

--output no_negative_regs
signal reg_noneg : std_logic_vector(N+K+1 downto 0);
signal rvd_noneg : std_logic;

--output Control_Ram

signal d_out_cr : std_logic_vector(N+K+1 downto 0);
signal enb_en : std_logic ;
signal p : std_logic ;

--output PLL_Syndesi
signal address_reg : std_logic_vector(3 downto 0);
signal dat_out_reg: std_logic_vector(15 downto 0);
signal dat_val_reg : std_logic ;



begin

 CLK_IBUF  : IBUFGDS
   generic map (
      DIFF_TERM => FALSE, -- Differential Termination (Virtex-4 only)

      IOSTANDARD => "DEFAULT")
   port map (
      O => clk ,
      I => clk_p,
      IB => clk_n
```

```
  STRATRR : IBUF
  generic map (
    IBUF_DELAY_VALUE => "0",
    IFD_DELAY_VALUE => "AUTO",
    IOSTANDARD => "DEFAULT")
  port map (
    O => START,
    I => START_IN
  );



dready_A_IBUF :              IBUFDS
            generic map (
    CAPACITANCE => "DONT_CARE",
    DIFF_TERM => FALSE,

    IOSTANDARD => "DEFAULT")

            Port map (
            I=> dra_p ,
            IB=> dra_N ,
            O=> dready_A

            );

            dready_B_IBUF :              IBUFDS
                        generic map (
    CAPACITANCE => "DONT_CARE",
    DIFF_TERM => FALSE,

    IOSTANDARD => "DEFAULT")

            Port map (
            I=>drb_p,
            IB=>drb_n,
            O=> dready_B

            );
```

```
        data_a_all : for i in 11 downto 0 generate
            dat_a_ibuf : IBUFDS
                generic map (
    CAPACITANCE => "DONT_CARE",
    DIFF_TERM => FALSE,

    IOSTANDARD => "DEFAULT")

            Port map (
              I=>dat_a_p(i),
              IB=>dat_a_n(i),
              O=> dat_a(i) );

                  end generate  data_a_all ;


data_b_all : for i in 11 downto 0 generate
                dat_b_ibuf : IBUFDS
                    generic map (
     CAPACITANCE => "DONT_CARE",
    DIFF_TERM => FALSE,

    IOSTANDARD => "DEFAULT")

            Port map (
              I=>dat_b_p(i),
              IB=>dat_b_n(i),
              O=> dat_b(i) );

                  end generate  data_b_all ;

adc : ad_contoller
port map (
clk =>clk ,
 dready_A => dready_A ,
  dready_B => dready_B ,
 dat_a  =>dat_a,
 dat_b =>dat_b ,
```

```
 datout => xn_re_next

);




---------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
fft_4096_tp : fft
                        port map (
                                xn_re => xn_re_next,
                                xn_im => xn_im_reg,
                                start => start,
                                fwd_inv => fwd_inv,
                                fwd_inv_we => fwd_inv_we,
                                scale_sch => scale_sch,
                                scale_sch_we => scale_sch_we,
                                clk => clk,
                          xk_re => xk_re_fft,
                                xk_im => xk_im_fft,
                                xn_index => xn_index_fft,
                                xk_index => xk_index_fft,
                                rfd => rfd_fft,
                                busy => busy_fft,
                                dv => dv_fft,
                                edone => edone_fft,
                                done => done_fft);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

translate : compl2_to_1qnformat
port map (
  x_re =>xk_re_fft,
  x_im => xk_im_fft,
  x_index => xk_index_fft,

  x_re_1qf =>xk_re_1qf,
  x_im_1qf => xk_im_1qf,
  x_index_1qf =>xk_index_1qf
);
```

```
cordixx : corxnindex
port map (x_re=>xk_re_1qf,
        x_im => xk_im_1qf,
        x_index_in =>xk_index_1qf,
                clk =>clk ,
                platos =>platos_cor ,
                rdy => rdy_cor ,
               x_index_out =>x_index_cor
);

taxinomisi : taxinomisi_reg
port map (
clk=>clk,
magnitude_in=>platos_cor,
x_index_in =>x_index_cor,

out_en =>out_en_tax,
reg1out =>reg1out_tax,
reg2out =>reg2out_tax,
reg3out =>reg3out_tax,
reg4out =>reg4out_tax,
reg5out =>reg5out_tax,
reg6out =>reg6out_tax,
reg7out =>reg7out_tax,
reg8out =>reg8out_tax

);

pll_connection : PLL_SYNDESI
port map(
clk =>clk ,
din_val =>out_en_tax,
dat1 =>reg1out_tax(N+K+1 DOWNTO N),
dat2 =>reg2out_tax(N+K+1 DOWNTO N),
dat3 =>reg3out_tax(N+K+1 DOWNTO N),
dat4 =>reg4out_tax(N+K+1 DOWNTO N),
dat5 =>reg5out_tax(N+K+1 DOWNTO N),
dat6 =>reg6out_tax(N+K+1 DOWNTO N),
dat7 =>reg7out_tax(N+K+1 DOWNTO N),
```

```
dat8 =>reg8out_tax(N+K+1 DOWNTO N),

address =>address_reg,
dat_out =>dat_out_reg,
dat_val =>dat_val_reg

);

seiriakos_registers : regist
port map (
reg1_in=>reg1out_tax ,
reg2_in=>reg2out_tax ,
reg3_in=>reg3out_tax ,
reg4_in=>reg4out_tax ,
reg5_in=>reg5out_tax ,
reg6_in=>reg6out_tax ,
reg7_in=>reg7out_tax ,
reg8_in=>reg8out_tax ,
da_val =>out_en_tax  ,
clk=>clk ,

reg_serial_out=>reg_serial_regist,
rvd =>rvd_regist
);

no_neg_regs : no_negative_regs
port map(clk =>clk,
        d_value=>rvd_regist,
                reg_in=>reg_serial_regist,
                                reg_serial_out=>reg_noneg,
                                rvd=>rvd_noneg


);

RAM_63_CONTROL : Control_Of_Ram
PORT MAP(clk =>clk,
reg_in=>reg_noneg,
ena=>'1',
wea=>rvd_noneg,
```

```
d_out=>d_out_cr,
enb_en=>enb_en,
p_out=>p

);


address_all : for i in 3 downto 0 generate
address_output: OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => address(i),
    I => address_reg(i)
  );
end generate;

dat_out_all : for i in 15 downto 0 generate
dat_out_output: OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => dat_out(i),
    I => dat_out_reg(i)
  );
end generate;

dat_val_output :OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => dat_val,
    I => dat_val_reg
  );
```

```
p_en_output : OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => p_en,
    I => p
  );

enb_output : OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => enb,
    I => enb_en
  );

d_out_all : for i in 26 downto 0 generate
        dout_output : OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => d_out(i),
    I => d_out_cr(i)
  );
end generate;
end Behavioral;
```

ad_contoller:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity ad_contoller is
port (
 clk:in std_logic; --roloi g
 dready_A ,dready_B :in std_logic;--ta rologia eisodou
                                        --twn dedomenwn
 dat_a , dat_b :in std_logic_vector(11 downto 0);-- ta dedomena eisodou

 datout : out std_logic_vector(13 downto 0)-- ta dedomena exodou

);
end ad_contoller;
architecture Behavioral of ad_contoller is
begin

process
begin
wait until (clk'event and clk='1');
 if( dready_A ='1' and dready_B ='0')then
  datout<=dat_a&"00";
  elsif( dready_B ='1' and dready_A ='0')then
  datout<=dat_b&"00";
  else
    datout<=(datout'range=>'0');
  end if;
```

```
  end process;


end Behavioral;
```

compl2_to_1qnformat :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity compl2_to_1qnformat is
generic (K: integer :=11;   --orismos tou megethous tou x_index se sxesi me ton fft
                            -- pou exoume
          N: INTEGER :=14 ); --orismos tou megetous twn eisodwn kai exodwn twn
                            -- dedomenwn basi pali tou fft pou exoume

  port (
        x_re  : in std_logic_vector (N-1 downto 0 );-- h pragmatiki kai fantastiki eisodos
                                                    -- tou metatropea einai se symplyrwma 2
        x_im  : in std_logic_vector (N-1 downto 0 );
        x_index : in std_logic_vector (K downto 0);
        x_re_1qf : out std_logic_vector(N downto 0);
         x_im_1qf : out std_logic_vector(N downto 0);
        x_index_1qf : out std_logic_vector (K downto 0)
```

```
      );
end compl2_to_1qnformat;

architecture Behavioral of compl2_to_1qnformat is
  begin

    process (x_re)
       begin
          if (x_re(N-1)='0') then
            if (x_re(N-2 downto 0)="1111111111111") then
              x_re_1qf<="010000000000000";--to ena sto 1qn format .
             else x_re_1qf<= '0'&x_re(N-1 downto 0);
            end if ;
          else  x_re_1qf<= '1'&x_re(N-1 downto 0);-- to -1 sto 1qn
       end if ;
    end process;
    process (x_im)
       begin
          if (x_im(N-1)='0') then
            if (x_im(N-2 downto 0)="1111111111111") then
              x_im_1qf<="010000000000000";--to ena sto 1qn format .
             else x_im_1qf<= '0'&x_im(N-1 downto 0);
            end if ;
          else  x_im_1qf<= '1'&x_im(N-1 downto 0);-- to -1 sto 1qn
       end if ;
    end process;
 x_index_1qf<=x_index;

end Behavioral;
```

corxnindex :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity corxnindex is
generic(K : INTEGER :=11;
       N :INTEGER :=14);
 port (  x_re: in std_logic_VECTOR(N downto 0);
                 x_im: IN std_logic_VECTOR(N downto 0);
                x_index_in : in std_logic_vector(K downto 0);
               clk: IN std_logic ;
              platos: OUT std_logic_VECTOR(N downto 0);
              rdy: OUT std_logic;
              x_index_out : out std_logic_vector(K downto 0)
            );
end corxnindex;

architecture Behavioral of corxnindex is


-- The following code must appear in the VHDL architecture header:

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component cordic
            port (
            x_in: IN std_logic_VECTOR(14 downto 0);
            y_in: IN std_logic_VECTOR(14 downto 0);
            x_out: OUT std_logic_VECTOR(14 downto 0);
            rdy: OUT std_logic;
            clk: IN std_logic);
end component;

-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of cordic: component is "true";

-- Synplicity black box declaration
attribute syn_black_box : boolean;
```

```
attribute syn_black_box of cordic: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------



component regs
generic(K : INTEGER :=11);
port(
d : in std_logic_vector(K downto 0 );
clk: in std_logic ;
q : out std_logic_vector(K downto 0)

);
end component ;

--thelw kathusterisi 21 periodous .
subtype word is std_logic_vector(K downto 0);
type regis is array (1 to 19 ) of word ;--gia allagi kathisteriseisallazeis
                                --to 19kuklwn rologiou .

signal s_reg :regis ;
signal anti_clk:std_logic;--gia na bgainoun ta dedomena apo to cordic
                        --otan einai thetiki i akmh tou rologiou!!!
signal platos_reg :std_logic_vector(N downto 0):=(others=>'0');
begin

anti_clk<=not clk;
s_reg(1)<=x_index_in;
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
the_cordic : cordic
                        port map (
                                x_in => x_re,
                                y_in => x_im,
                                x_out => platos_reg,
                                rdy => rdy,
                                clk => anti_clk);
-- INST_TAG_END ------ End INSTANTIATION Template ------------
```

```
 registerss : for i in  1 to 18 generate--otan allazei i kathisterisei allazei kai to 20
     reggs : regs port map (d=>s_reg(i),clk=>clk , q=>s_reg(i+1));
 end generate registerss;

process
  begin
  wait until (clk'event and clk='1');

             platos <=platos_reg;
    x_index_out<=s_reg(19);
  end process;

end Behavioral;
```

taxinomisi_reg:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity taxinomisi_reg is
generic (N:integer :=14;--to megethos twn registers
        K:integer :=11 );-- to megethos tou x_index
 port(      clk : in std_logic;
             magnitude_in: in std_logic_vector(N downto 0);
             x_index_in : in std_logic_vector(K downto 0);
            out_en : out std_logic; --ta dedomena einai egkura
             reg1out : out std_logic_vector(N+K+1 downto 0);
             reg2out : out std_logic_vector(N+K+1 downto 0);
            reg3out : out std_logic_vector(N+K+1 downto 0);
            reg4out : out std_logic_vector(N+K+1 downto 0);
            reg5out : out std_logic_vector(N+K+1 downto 0);
            reg6out : out std_logic_vector(N+K+1 downto 0);
            reg7out : out std_logic_vector(N+K+1 downto 0);
            reg8out : out std_logic_vector(N+K+1 downto 0)
 );
end taxinomisi_reg;

architecture Behavioral of taxinomisi_reg is
constant epana_reg : std_logic_vector(K downto 0):=(others=>'1');

constant x_ind_reg : std_logic_vector(K downto 0):=(others=>'0');
constant noise_reg : std_logic_vector(N-1 downto 0):="00000000001111";
--orizeis ton thoruvo

--eswterikoi kataxwrites!!!

 signal reg1 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg2 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg3 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg4 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg5 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
```

```
 signal reg6 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg7 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg8 : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg1_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg2_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg3_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg4_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg5_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg6_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg7_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal reg8_in : std_logic_vector(N+K+1 downto 0):=x_ind_reg & '1' & noise_reg;
 signal neo : std_logic_vector(N downto 0):= '1' & noise_reg;
 signal x_index : std_logic_vector(K downto 0):=(others=>'0');



begin

    neo<=magnitude_in;
    x_index<=x_index_in;

reg:process --eswtwriki apothikeusi twn
              -- apotelesmata tis sugkrisis kai
              -- exodo twn apotelesmatwn !!
  begin
    wait until (clk'event and clk='1');
     --sunexis exodos dedomenwn
            reg8out<=reg8_in;reg7out<=reg7_in;
          reg6out<=reg6_in;reg5out<=reg5_in;
          reg4out<=reg4_in;reg3out<=reg3_in;
            reg2out<=reg2_in;reg1out<=reg1_in;

            reg8<=reg8_in;reg7<=reg7_in;
            reg6<=reg6_in;reg5<=reg5_in;
            reg4<=reg4_in;reg3<=reg3_in;
         reg2<=reg2_in ;reg1<=reg1_in ;


          if(x_index = epana_reg) then
```

```
                out_en<='1';

              reg8<=x_ind_reg & '1' & noise_reg;   reg7<=x_ind_reg & '1' & noise_reg;
                reg6<=x_ind_reg & '1' & noise_reg;   reg5<=x_ind_reg & '1' & noise_reg;
                reg4<=x_ind_reg & '1' & noise_reg;   reg3<=x_ind_reg & '1' & noise_reg;
                 reg2<=x_ind_reg & '1' & noise_reg;   reg1<=x_ind_reg & '1' & noise_reg;
              else
                      out_en<='0';


            end if;
          end process;
sort_the_data : process
           begin
             wait until (clk'event and clk='0');
                reg8_in<=reg8 ;
                   reg7_in<=reg7;
                reg6_in<=reg6;
                  reg5_in<=reg5;
                 reg4_in<=reg4;
                 reg3_in<=reg3;
                reg2_in<=reg2;
               reg1_in<=reg1;

            if((neo(N-1 downto 0)>reg1(N-1 downto 0))) then
            --enhmerwsi eswterikwn kataxwritwn
                 reg8_in<=reg7;reg7_in<=reg6;reg6_in<=reg5;
               reg5_in<=reg4;reg4_in<=reg3;reg3_in<=reg2;
                 reg2_in<=reg1;
                 reg1_in<=x_index & neo ;
    elsif((neo(N-1 downto 0)>reg2(N-1  downto 0))and((neo(N-1  downto
                         0)<=reg1(N-1  downto 0)))) then
              --enhmerwsi eswterikwn kataxwritwn

            reg8_in<=reg7;reg7_in<=reg6;
          reg6_in<=reg5;reg5_in<=reg4;
          reg4_in<=reg3;reg3_in<=reg2;
          reg2_in<=x_index & neo ;
           reg1_in<=reg1;
```

```
 elsif((neo(N-1  downto 0)>reg3(N-1  downto 0))and((neo(N-1  downto
0)<=reg2(N-1  downto 0))))then
            --enhmerwsi eswterikwn kataxwritwn
            reg8_in<=reg7;reg7_in<=reg6;
            reg6_in<=reg5;reg5_in<=reg4;
              reg4_in<=reg3;
             reg3_in<=x_index & neo ;
           reg2_in<=reg2;reg1_in<=reg1;


elsif((neo(N-1  downto 0)>reg4(N-1  downto 0))and((neo(N-1  downto 0)<=reg3(N-1
                                                          downto 0))))then
--enhmerwsi eswterikwn kataxwritwn
                  reg8_in<=reg7;reg7_in<=reg6;
                reg6_in<=reg5;reg5_in<=reg4;
                reg4_in<=x_index & neo ;
                reg3_in<=reg3;
                  reg2_in<=reg2;reg1_in<=reg1;
elsif((neo(N-1  downto 0)>reg5(N-1  downto 0))and((neo(N-1  downto 0)<=reg4(N-1
                                                            downto 0)))) then
 --enhmerwsi eswterikwn kataxwritwn

                   reg8_in<=reg7;
                reg7_in<=reg6;reg6_in<=reg5;
                reg5_in<=x_index & neo ;
                reg4_in<=reg4;
                reg3_in<=reg3;
                   reg2_in<=reg2;reg1_in<=reg1;
 elsif((neo(N-1  downto 0)>reg6(N-1  downto 0))and((neo(N-1  downto 0)<=reg5(N-1  downto
0))))then                --allaxe to reg6_in
   --enhmerwsi eswterikwn kataxwritwn
                   reg8_in<=reg7;
                reg7_in<=reg6;
                  reg6_in<=x_index & neo ;
                     reg5_in<=reg5;reg4_in<=reg4;
                     reg3_in<=reg3;
                  reg2_in<=reg2;reg1_in<=reg1;
elsif((neo(N-1  downto 0)>reg7(N-1  downto 0))and((neo(N-1  downto 0)<=reg6(N-1
downto 0)))) then
            --enhmerwsi eswterikwn kataxwritwn
```

```
                    reg8_in<=reg7;
                       reg7_in<=x_index & neo ;
                     reg6_in<=reg6;reg5_in<=reg5;reg4_in<=reg4;
                   reg3_in<=reg3;
                     reg2_in<=reg2;reg1_in<=reg1;
 elsif((neo(N-1  downto 0)>reg8(N-1  downto 0))and((neo(N-1  downto 0)<=reg7(N-1  downto 0))))
then
              --enhmerwsi eswterikwn kataxwritwn
                   reg8_in<=x_index & neo ;
                 reg7_in<=reg7;
                  reg6_in<=reg6;reg5_in<=reg5;reg4_in<=reg4;
                  reg3_in<=reg3;
                     reg2_in<=reg2;reg1_in<=reg1;
   end if;

 end process  sort_the_data;
end Behavioral;
```

PLL_SYNDESI :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PLL_SYNDESI is
generic (N:integer :=14; --to megethos twn registers pou exoun to platos
        K:integer :=11 ); --to megethos twn xn_index , twn pin!!
port (
         clk :in std_logic ;
            din_val : in std_logic ;
            dat1 : in std_logic_vector(K+1 DOWNTO 0) ;
            --to xn_index kai to 1 bit (to lsb )
          --pou deixnei an einai valid i oxi
            dat2 : in std_logic_vector(K+1 DOWNTO 0) ;
            dat3 : in std_logic_vector(K+1 DOWNTO 0) ;
            dat4 : in std_logic_vector(K+1 DOWNTO 0) ;
            dat5 : in std_logic_vector(K+1 DOWNTO 0) ;
            dat6 : in std_logic_vector(K+1 DOWNTO 0) ;
            dat7 : in std_logic_vector(K+1 DOWNTO 0) ;
```

```
            dat8 : in std_logic_vector(K+1 DOWNTO 0) ;

            address : out std_logic_vector(3 DOWNTO 0) ;
            dat_out : out std_logic_vector(15 DOWNTO 0) ;

            dat_val : out std_logic
    );
end PLL_SYNDESI;
architecture Behavioral of PLL_SYNDESI is

component reg_egkiro
 generic(N:integer :=14;--to megethos twn registers pou exoun to platos
        K:integer :=11 );--to megethos twn xn_index , twn pin!!
    Port ( clk : in  STD_LOGIC ;
             en :  in  STD_LOGIC ;
           reg_in : in std_logic_vector( K+1 DOWNTO 0);
             reg_out : out std_logic_vector( 15 DOWNTO 0)
             );
end component reg_egkiro;

subtype word1 is std_logic_vector(K+1 DOWNTO 0);
type regtype1 is array (7 downto 0) of word1 ;-- theloume 8 registers an theloume 16
                                -- registers  tote 15 anti 7.

subtype word2 is std_logic_vector(15 DOWNTO 0);
type regtype2 is array (7 downto 0) of word2 ;-- theloume 8 registers an theloume 16
                                -- registers  tote 15 anti 7.
signal m : std_logic:='0';
signal ren : std_logic :='0';

signal cnt20u : std_logic_vector(10 downto 0):=(others=>'0');
signal cnt100n :std_logic_vector(3 downto 0) :=(others=>'0');

signal count : std_logic_vector(3 downto 0):=(others=>'0');
signal address_reg : std_logic_vector(3 downto 0):=(others=>'0');
signal register_out : std_logic_vector( 15 DOWNTO 0):=(others=>'0');
signal dout :std_logic :='0';

signal reg_in : regtype1 ;
```

```
signal reg1_out : regtype2 ;

signal reg2_out : regtype2 ;

begin
            reg_in(0)<=dat1;
            reg_in(1)<=dat2;
            reg_in(2)<=dat3;
            reg_in(3)<=dat4;
            reg_in(4)<=dat5;
            reg_in(5)<=dat6;
            reg_in(6)<=dat7;
            reg_in(7)<=dat8;

        reg_start : for i in 0 to 7 generate
       regs : reg_egkiro port map (clk ,din_val ,reg_in(i),reg1_out(i) );
                    end generate reg_start;


          process
           begin
          --          wait for DELAY1;
              wait until (clk'event and clk='1' );

                    if(cnt20u="11111001111") then--tote einai ta 20us.

                     reg2_out(0)<= reg1_out(0);
                     reg2_out(1)<= reg1_out(1);
                     reg2_out(2)<= reg1_out(2);
                     reg2_out(3)<= reg1_out(3);
                     reg2_out(4)<= reg1_out(4);
                     reg2_out(5)<= reg1_out(5);
                     reg2_out(6)<= reg1_out(6);
                     reg2_out(7)<= reg1_out(7);

                    cnt20u<=(cnt20u'range =>'0');
                     m<='1';

                    else
                      cnt20u<=cnt20u+'1';
```

```
                    m<='0';
            end if;

end process;

process
 begin
  wait until(clk'event and clk='1');
       dout<='0';
        if(ren='1')then
            if(cnt100n="0000")then

                cnt100n<="1001";--deka kuklous rologiou i anamoni
           --gia ta epomena dedomena
 case count is
when"0000" =>register_out<=reg2_out(0);
            address_reg<="0000";
             dout<='1';
            count<=count+'1';
when"0001" => register_out<=reg2_out(1);
            address_reg<="0001";
              dout<='1';
              count<=count+'1';

when"0010" => register_out<=reg2_out(2);
            address_reg<="0010";
             dout<='1';
            count<=count+'1';
when"0011" =>  register_out<=reg2_out(3);
            address_reg<="0011";
           dout<='1';
            count<=count+'1';
  when"0100" => register_out<=reg2_out(4);
            address_reg<="0100";
             dout<='1';
            count<=count+'1';
when"0101" =>register_out<=reg2_out(5);
            address_reg<="0101";
            dout<='1';
            count<=count+'1';
```

```
            when"0110" =>register_out<=reg2_out(6);
                        address_reg<="0110";
                        count<=count+'1';
             when"0111" =>register_out<=reg2_out(7);
                        address_reg<="0111";
                         dout<='1';
                        count<=count+'1';
            when"1000" =>
                                cnt100n<="0000";
                            count<="0000";

             when others => register_out<=(register_out'range=>'0');
           end case ;

  else
      cnt100n<=cnt100n-'1';

 end if;

 else
 register_out<=(register_out'range=>'0');
 address_reg<="0000";
  end if ;

end process;
 ren<=m or count(3)  or count(2) or count(1) or count(0) ;
 address <=address_reg;
 dat_out <=register_out;
 dat_val <=dout;
end Behavioral;
```

Regist :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity regist is
generic (N:integer :=14;--to megethos twn registers
          K:integer :=11 );--to megethos tou t.p

   Port ( reg1_in : in std_logic_vector(N+K+1 downto 0);
          reg2_in : in std_logic_vector(N+K+1 downto 0);
          reg3_in : in std_logic_vector(N+K+1 downto 0);
          reg4_in : in std_logic_vector(N+K+1 downto 0);
          reg5_in : in std_logic_vector(N+K+1 downto 0);
          reg6_in : in std_logic_vector(N+K+1 downto 0);
          reg7_in : in std_logic_vector(N+K+1 downto 0);
          reg8_in : in std_logic_vector(N+K+1 downto 0);
          da_val : in  STD_LOGIC;
           clk : in STD_LOGIC;

          reg_serial_out : out std_logic_vector(N+K+1 downto 0);
          rvd :out std_logic -- ta dedomena exw einai valid
                                  );
end regist;

architecture Behavioral of regist is

 signal  reg1 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg2 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg3 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg4 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg5 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg6 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg7 :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg8 :std_logic_vector(N+K+1 downto 0):=(others=>'0');

 signal  reg1_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
 signal  reg2_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
```

```
signal  reg3_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
signal  reg4_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
signal  reg5_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
signal  reg6_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
signal  reg7_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');
signal  reg8_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');

signal  reg_temp :std_logic_vector(N+K+1 downto 0):=(others=>'0');

signal cnt : std_logic_vector(2 downto 0) := "000";
signal sen : std_logic := '0';
begin

 process
  begin
          wait until(clk'event and clk='1');

        if (da_val = '1') then
                              reg1_temp <= reg1_in; reg2_temp <= reg2_in;
                              reg3_temp <= reg3_in; reg4_temp <= reg4_in;
                              reg5_temp <= reg5_in; reg6_temp <= reg6_in;
                              reg7_temp <= reg7_in; reg8_temp <= reg8_in;

                              reg1 <= reg1_in; reg2 <= reg2_in;
                              reg3 <= reg3_in; reg4 <= reg4_in;
                              reg5 <= reg5_in; reg6 <= reg6_in;
                              reg7 <= reg7_in; reg8 <= reg8_in;
       else
                               reg1 <= reg1_temp; reg2 <= reg2_temp;
                             reg3 <= reg3_temp; reg4 <= reg4_temp;
                             reg5 <= reg5_temp; reg6 <= reg6_temp;
                             reg7 <= reg7_temp; reg8 <= reg8_temp;

       end if ;

end process;

process
begin
wait until(clk'event and clk='1');
```

```
    reg_temp<=(reg_temp'range=>'0');
            rvd<='0';

                        if(sen ='1') then

                          if(cnt ="000") then
                                    reg_temp<=reg1 ;
                                     rvd<='1';
                        elsif(cnt ="001") then
                                     reg_temp<=reg2 ;
                                      rvd<='1';
                         elsif(cnt ="010") then
                                      reg_temp<=reg3 ;
                                       rvd<='1';
                        elsif(cnt ="011") then
                                      reg_temp<=reg4 ;
                                        rvd<='1';
                        elsif(cnt ="100") then
                                      reg_temp<=reg5 ;
                                       rvd<='1';
                        elsif(cnt ="101") then
                                      reg_temp<=reg6 ;
                                        rvd<='1';
                         elsif(cnt ="110") then
                                      reg_temp<=reg7 ;
                                        rvd<='1';
                       elsif(cnt ="111") then
                                      reg_temp<=reg8 ;
                                        rvd<='1';
                        end if;

                        cnt<=cnt+"001";
            end if;
  end process;
  sen <=p or cnt(2) or cnt(1) or cnt(0);
  reg_serial_out <=reg_temp ;
end Behavioral;
```

no_negative_regs:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity no_negative_regs is
generic (N:integer :=14;--to megethos twn registers
        K:integer :=11 );--to megethos tou t.p
Port (
                clk: IN std_logic;
                 d_value : in std_logic ;--otan d_val='1' tote einai valid oi registers
                  reg_in : in std_logic_vector(N+K+1 downto 0);

                  reg_serial_out : out std_logic_vector(N+K+1 downto 0);
                  rvd :out std_logic -- ta dedomena exw einai valid
       );
end no_negative_regs;

architecture Behavioral of no_negative_regs is

   begin

      process
        begin
         wait until(clk'event and clk='1');

            reg_serial_out<=( reg_serial_out'range=>'0');
            rvd<='0';
             if (d_value = '1') then

               if(reg_in(14)='0') then

                  reg_serial_out<=reg_in;  --reg_in(N+K+1 downto 16 ) & reg_in(14 downto 0 ) ;
                  -- ta dedomena einai se morfi sumplirwma ws pros 2
```

```
                rvd<='1';
              elsif(reg_in(14)='1') then
                rvd<='0';

            end if;
        end if;
 end process ;
end Behavioral ;
```

regs:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity regs is
generic(K : INTEGER :=11);
port(
d : in std_logic_vector(K downto 0 );
clk: in std_logic ;

q : out std_logic_vector(K downto 0)

);

end regs;

architecture Behavioral of regs is

begin

process
begin
wait until (clk'event and clk='1');
q<=d;

end process;
end Behavioral;
```

reg_egkiro :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reg_egkiro is
 generic(N:integer :=14;--to megethos twn registers pou exoun to platos
        K:integer :=11 );--to megethos twn xn_index , twn pin!!
    Port ( clk : in  STD_LOGIC ;
                  en :  in  STD_LOGIC ;
          reg_in : in std_logic_vector( K+1 DOWNTO 0);
                                    reg_out : out std_logic_vector( 15 DOWNTO 0)


                );
end reg_egkiro;

architecture Behavioral of reg_egkiro is

constant not_valid_reg : std_logic_vector( 15 DOWNTO 0):=(others=>'1');
signal reg_reg : std_logic_vector( 15 DOWNTO 0):=(others=>'0');
begin
process
  begin
    wait until (clk'event and clk='1');

                        if(en='1') then
                         if(reg_in(0)='0') then
                           reg_reg<="0000"&reg_in(K+1 DOWNTO 1);
                                    else
                                    reg_reg<= not_valid_reg;
                          end if;
                        end if;

end process ;
reg_out <=reg_reg;

end Behavioral;
```

Control_Of_Ram :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Control_Of_Ram is
generic(N:integer :=14;
      K:integer :=11);
   Port (
            clk : in std_logic;
            reg_in : in  STD_LOGIC_VECTOR (N+K+1 downto 0);
        ena : in  STD_LOGIC;
        wea : in  STD_LOGIC;
        --enb : in  STD_LOGIC;
                              d_out : out  STD_LOGIC_VECTOR (N+K+1 downto 0);
                              --adrress_out : out STD_LOGIC_VECTOR (9 downto 0);
                              enb_en : out  STD_LOGIC;
        p_out :out STD_LOGIC
                              );
end Control_Of_Ram;

architecture Behavioral of Control_Of_Ram is


---------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component ram_63
          port (
          addra: IN std_logic_VECTOR(9 downto 0);
          addrb: IN std_logic_VECTOR(9 downto 0);
          clka: IN std_logic;
          clkb: IN std_logic;
          dina: IN std_logic_VECTOR(26 downto 0);
          doutb: OUT std_logic_VECTOR(26 downto 0);
          ena: IN std_logic;
          enb: IN std_logic;
          wea: IN std_logic);
end component;
```

```
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of ram_63: component is "true";

-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of ram_63: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------


constant count25_6_us :std_logic_vector(10 downto 0) := "10100000101";--12800nsec
constant count25c_ns :std_logic_vector(2 downto 0) := "100";--4 kykloi rologiou

signal addra_reg : STD_LOGIC_VECTOR (9 downto 0):=(others=>'0') ;

signal addrb_reg : STD_LOGIC_VECTOR (9 downto 0):=(others=>'1') ;
signal data : STD_LOGIC_VECTOR (N+K+1 downto 0):=(others=>'0') ;
signal addrb_2 : STD_LOGIC_VECTOR (9 downto 0):=(others=>'0') ;


signal count_all : std_logic_vector(10 downto 0):=(others=>'0') ;
signal count_50ns:std_logic_vector(2 downto 0) :=(others=>'0');
signal clka : STD_LOGIC:='0';
signal clkb : STD_LOGIC:='0';
signal ena_out  :   STD_LOGIC:='0';
signal wea_out :   STD_LOGIC:='0';
signal enb_out :   STD_LOGIC:='0';
signal enb_out2 :   STD_LOGIC:='0';

signal p:std_logic:='0';

begin
process
 begin

 wait until (clk'event and clk ='0');

   ena_out<='0';
            wea_out<='0';
```

```
 if (ena ='1' )then
  ena_out<='1';
  if(wea ='1') then
    wea_out <='1';
                    addra_reg<=addra_reg+'1';
                    data<=reg_in;
           end if;
          end if;


end process ;

process

 begin
  wait until (clk'event and clk ='0');

  if(count_all ="0000000000000")then
      p<='0';
                     else
           p<='1';
           count_all <=count_all -'1';
 end if ;

  if( addra_reg="0011111111"or addra_reg="0111111111"  or
           addra_reg="1011111111" or addra_reg="1111111111")then

            count_all <=count25_6_us ;


  end if ;

 end process;
process
 begin
 wait until (clk'event and clk ='0');
    enb_out<='0';
  if(p='1')then
```

```
            case count_50ns is
             when  "000" => addrb_reg<=addrb_reg+'1';
                        enb_out<='1';
                        count_50ns<=count25c_ns;
                      when others => count_50ns<=count_50ns - '1';
            end case ;
  end if;
end process ;

process
begin

 wait until (clk'event and clk ='1');
 enb_out2<=enb_out;
 enb_en <=enb_out;
 addrb_2<=addrb_reg;
end process;

p_out<=p;
clkb<=clk;
clka<= not clk ;

------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
ram63: ram_63
                      port map (
                                  addra => addra_reg,
                                  addrb => addrb_2,
                                  clka => clka,
                                  clkb => clkb,
                                  dina => data,
                                  doutb => d_out,
                                  ena => ena_out,
                                  enb => enb_out2,
                                  wea => wea_out);
-- INST_TAG_END ------ End INSTANTIATION Template ------------


end Behavioral;
```

MASTER_USB_CONN:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MASTER_USB_CONN is
port(
            clk : in std_logic ;--leitourgei sta 40 MHz
            enable : in std_logic ;
            data_in : in std_logic_vector(26 downto 0);
                  FLAGC : in std_logic ; --  EF sima gnwstopoiisis , to EP2 einai eleuthero
              --gia edomena

            SLWR : out std_logic ;-- sima gia na epitrepsi sto usb controller
               -- na grapsei ta dedomena
           ifclk:out std_logic;
          data_out :out       std_logic_vector(15 downto 0) ;
           fifoadr : OUT std_logic_vector(1 downto 0)

         );


end MASTER_USB_CONN;

architecture Behavioral of MASTER_USB_CONN is
component  Receive_Data
 port(
            clk : in std_logic ;--leitourgei sta 40 MHz
            enable : in std_logic ;
            data_in : in std_logic_vector(26 downto 0);

            enable_out : out std_logic ;
             data_out : out std_logic_vector(15 downto 0)
          );
end component  Receive_Data;

component Interconnection
port(
```

```
clk : in std_logic ;--leitourgei sta 40 MHz
enable : in std_logic ;
data_in : in std_logic_vector(15 downto 0) ;
FLAGC : in std_logic ; --  EF sima gnwstopoiisis , to EP2 einai eleuthero
                --gia edomena
SLWR : out std_logic ;-- sima gia na epitrepsi sto usb controller
               -- na grapsei ta dedomena
ifclk:out std_logic;
data_out :out       std_logic_vector(15 downto 0) ;
fifoadr : OUT std_logic_vector(1 downto 0)
);
end component  Interconnection;
signal enable_out_rec : std_logic ;
signal data_out_rec : std_logic_vector(15 downto 0) ;
begin

receive_virt4 :Receive_Data
port map(
clk=>clk,
enable=>enable,
data_in=>data_in,
enable_out=>enable_out_rec,
data_out=>data_out_rec
);

send_to_usb :Interconnection
port map (
clk=>clk,
enable=>enable_out_rec,
data_in=>data_out_rec,
FLAGC=>FLAGC,

SLWR =>SLWR ,
ifclk=>ifclk,
data_out =>data_out,
fifoadr=>fifoadr
);

end Behavioral;
```

Receive_Data :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

            entity Receive_Data is
             port(
               clk : in std_logic ;--leitourgei sta 40 MHz
               enable : in std_logic ;
               data_in : in std_logic_vector(26 downto 0);

               enable_out : out std_logic ;
             data_out : out std_logic_vector(15 downto 0)
            );
            end entity Receive_Data;

            architecture Behavioral of Receive_Data is
             signal  reg_temp_1:std_logic_vector(26 downto 0):=(others=>'0');

             signal other : std_logic :='1';
             signal in_en : std_logic:='0';
              begin

                process

                begin
                 wait until(clk'event and clk='1');
           reg_temp_1<=data_in ;

           enable_out<='0';
           data_out<=(data_out'range=>'0');

              if(enable='1')then

                           enable_out<='1';
                            other<=not other;
                                 if(other='0')then
                                    data_out<="100"&reg_temp_1(26 downto 14);
```

```
                    else
                        data_out<="00"&reg_temp_1(13 downto 0);
            end if;
        end if;
end process;
 end ;
```

Interconnection:

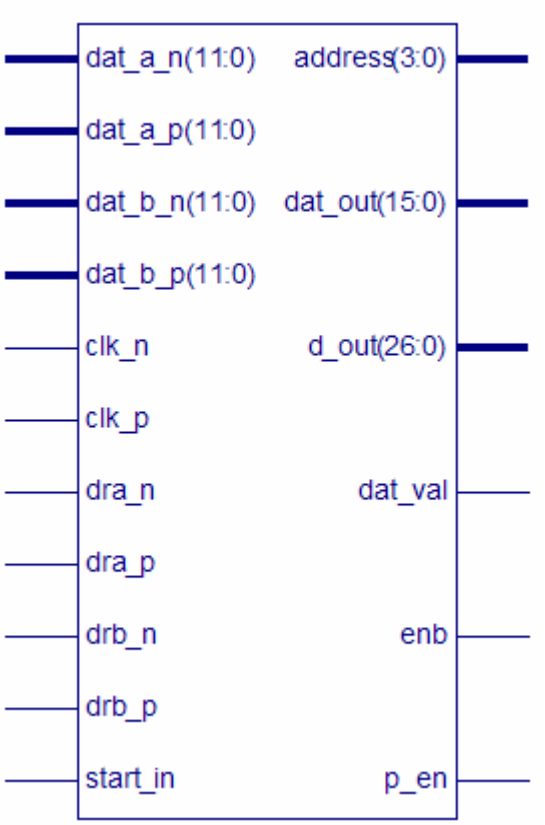
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Interconnection is
port(
clk : in std_logic ;--leitourgei sta 40 MHz
enable : in std_logic ;
data_in : in std_logic_vector(15 downto 0) ;

FLAGC : in std_logic ; --  EF sima gnwstopoiisis , to EP2 einai eleuthero
                --gia edomena

SLWR : out std_logic ;-- sima gia na epitrepsi sto usb controller
               -- na grapsei ta dedomena
ifclk:out std_logic;
data_out :out       std_logic_vector(15 downto 0) ;
fifoadr : OUT std_logic_vector(1 downto 0)
);
end Interconnection;

architecture Behavioral of Interconnection is
signal reg_1 : std_logic_vector(15 downto 0):=(others=>'0');
signal en_insert : std_logic:='0';

begin

insert_data : process
            begin
             wait until(clk'event and clk='1');
```

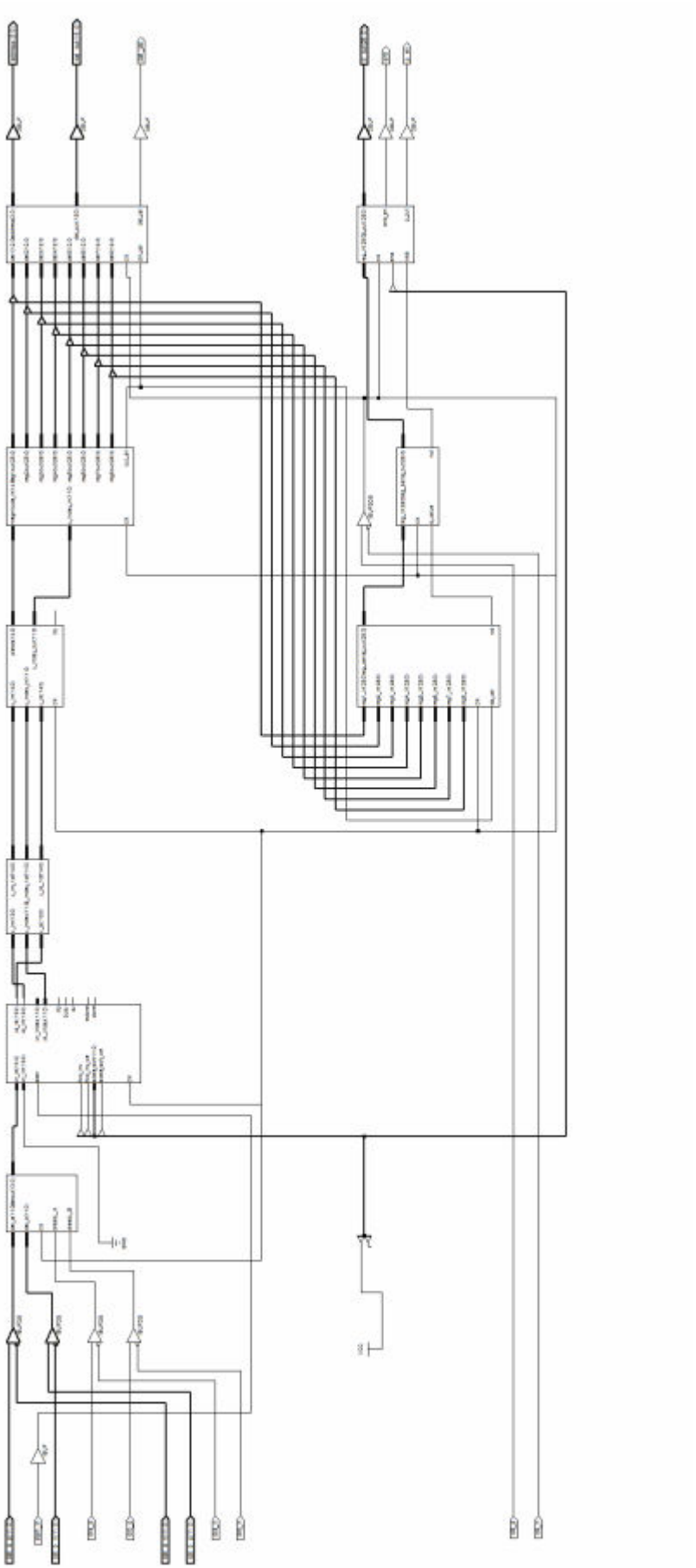
```
            if(enable='1')then

                    reg_1<=data_in;

                    en_insert <='1';
          else

                    en_insert <='0';
           end if;

                    end process;

output_config : process
           begin
           wait until(clk'event and clk='1');
             fifoadr<="00";--kathorizoume oti       xrhsimopoioyme to
                                                    --EP2 PORT gia eisodo !!
              SLWR<='1';-- to usb controller den epitrpei ta dedomena
                                                --na eiserxontai
             data_out<=(data_out'range=>'0');
                    if (en_insert='1')then
                     if (FLAGC ='0' )then--ean to EP2 einai eleuthero
                                    -- tote steile ta dedomena
                            SLWR<='0';--epitrepei thn eggrafi sto usb controller
                              data_out<=reg_1;--stelnei ta dedomena.
                      end if;
                     end if;

          end process;
ifclk<=clk; --to roloi eisodou twn dedomenwn toy USB einai sundedemeno to to clk
--gia sugxronismo


end Behavioral;
```

ΠΑΡΑΡΤΗΜΑ Β

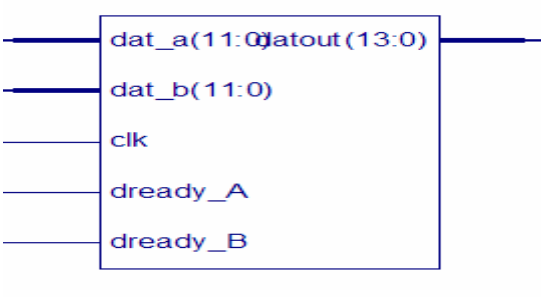
Διαγράμματα των Κυκλωμάτων

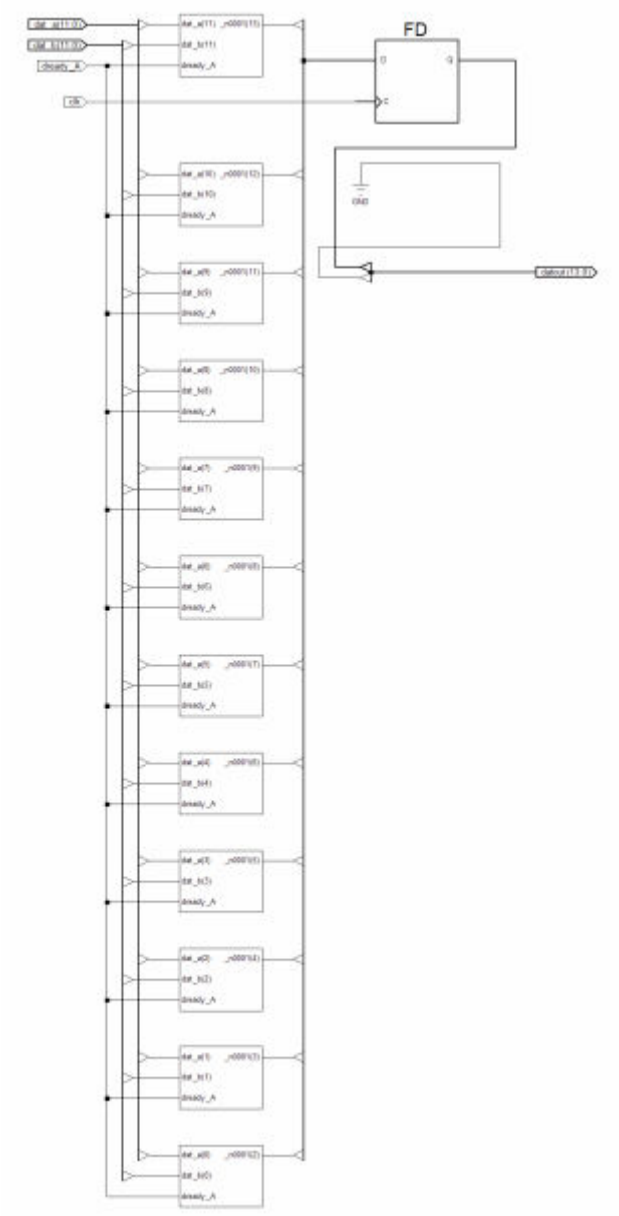
project_100MSps:



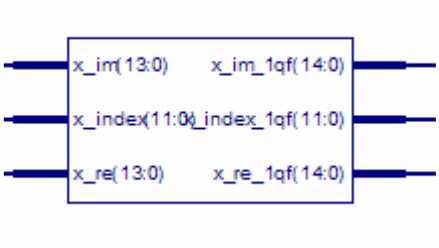


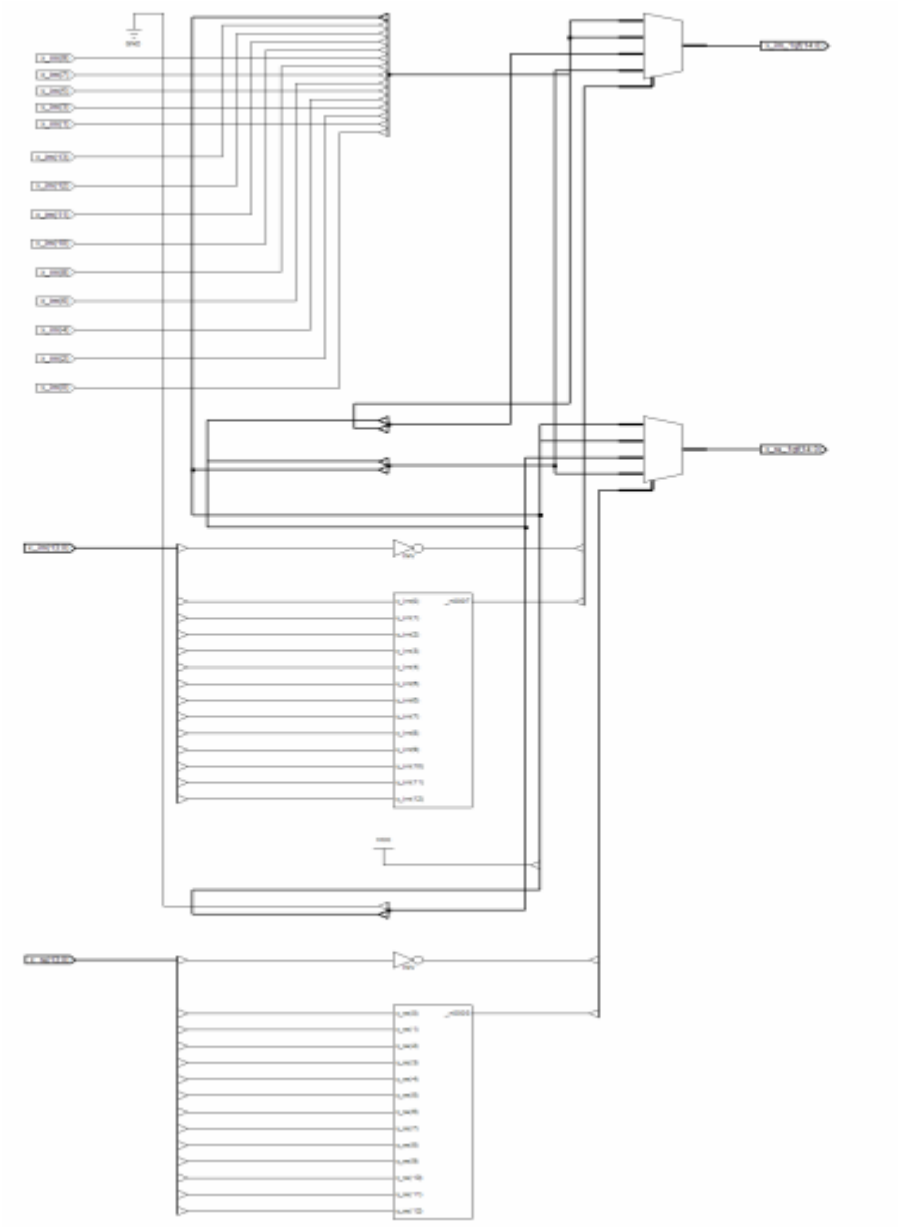
ad contoller:



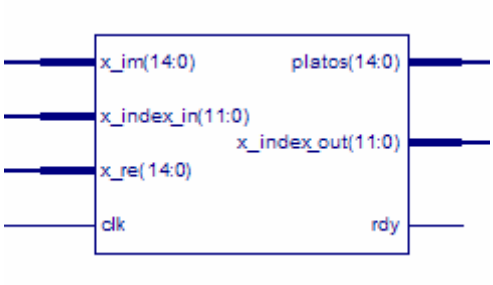


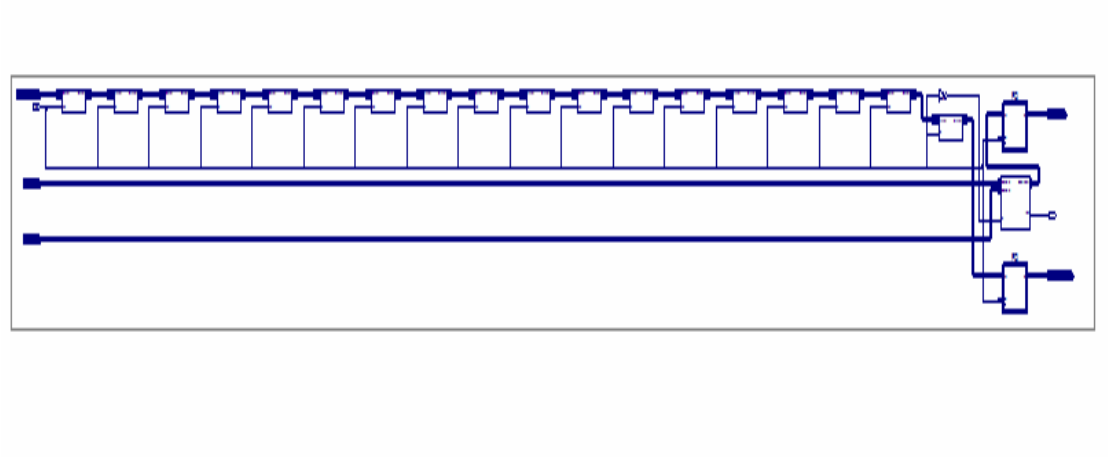
compl2_to_1qnformat :



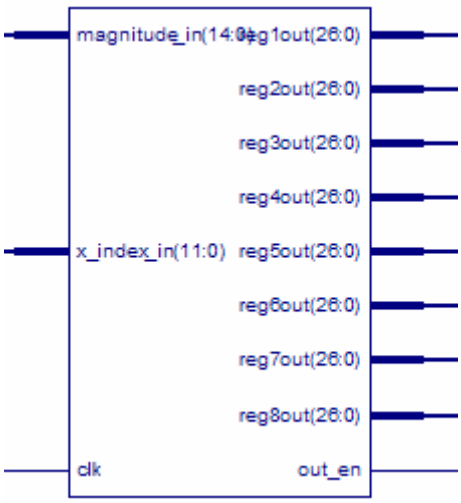


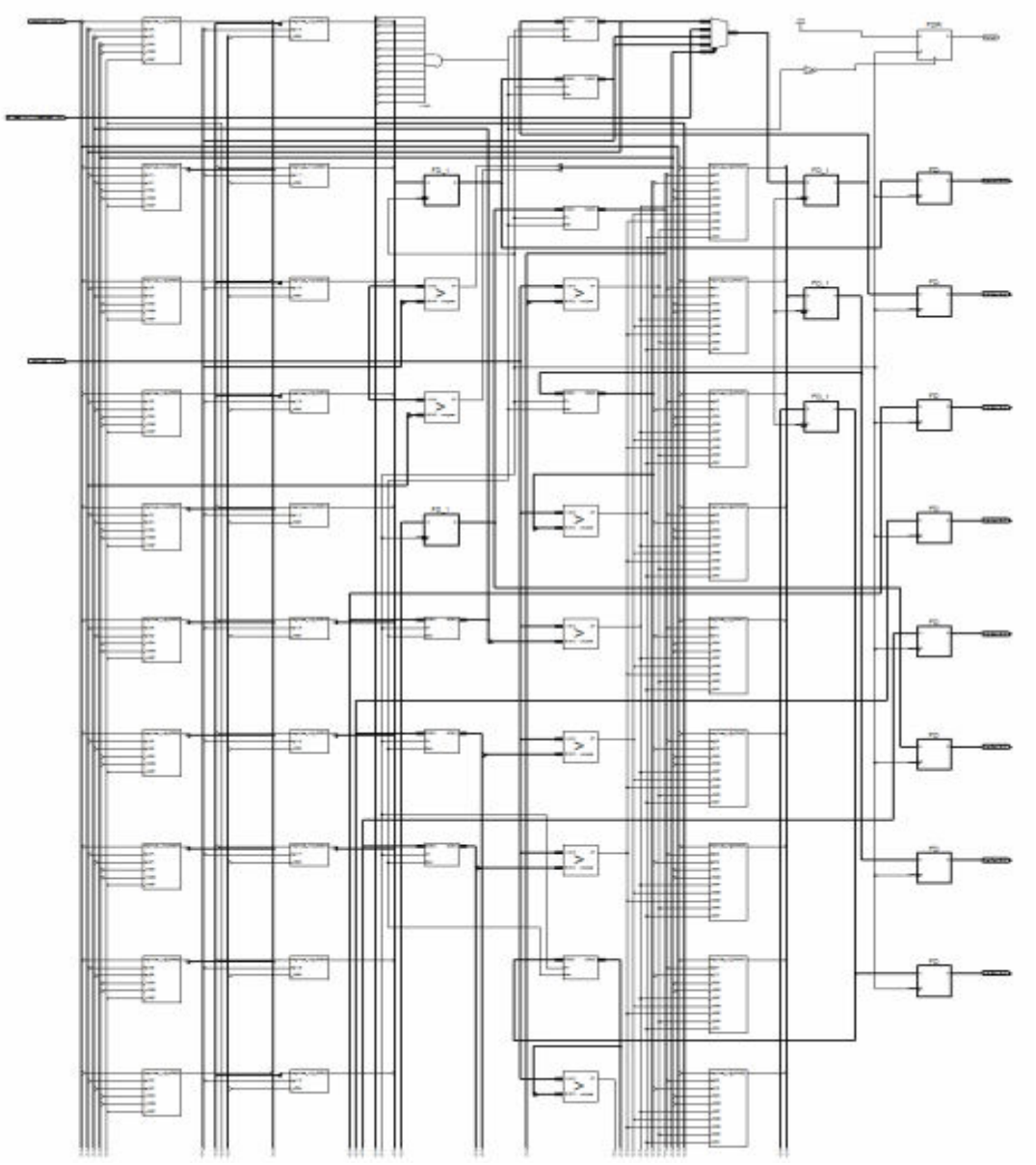
corxnindex :



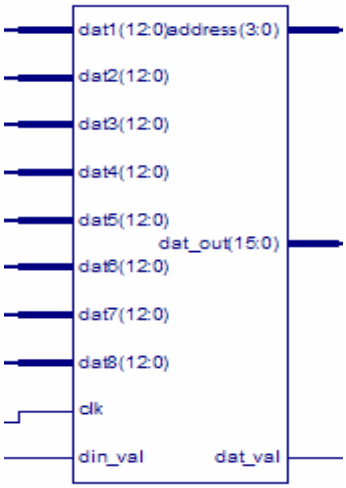


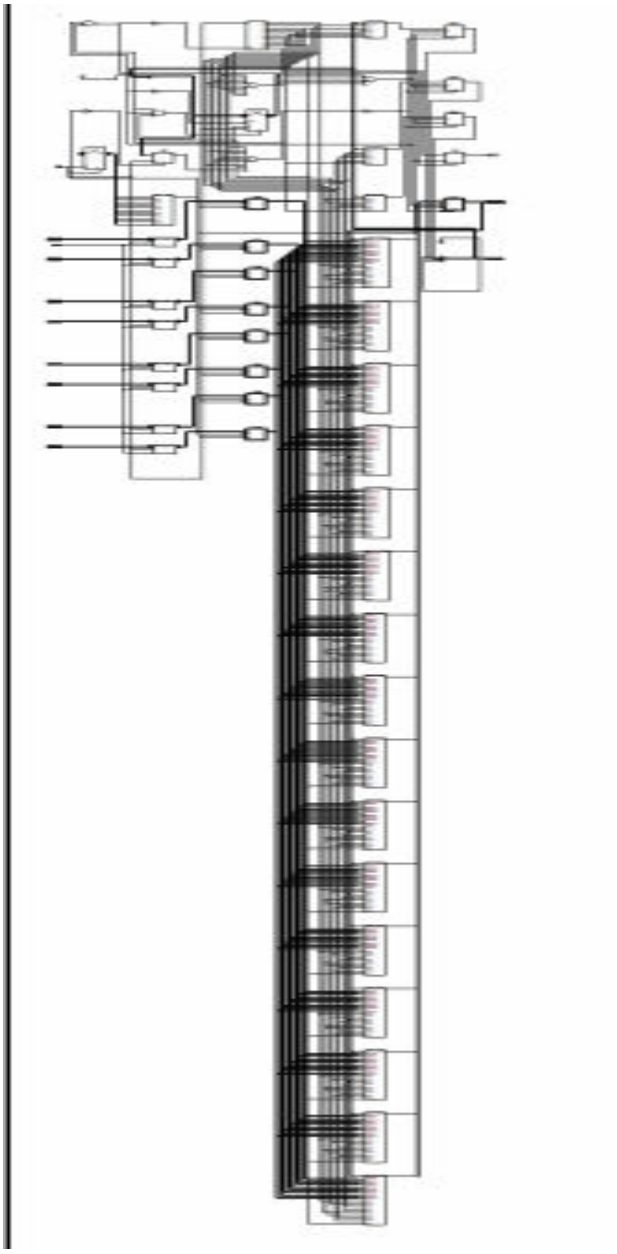
taxinomisi_reg:



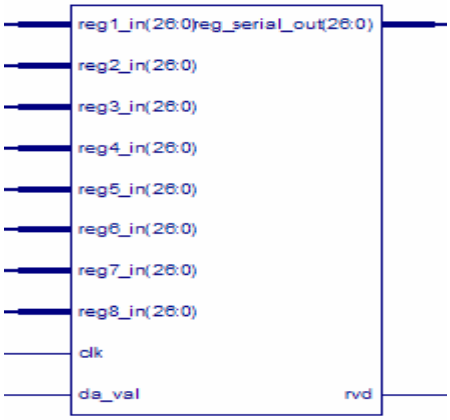


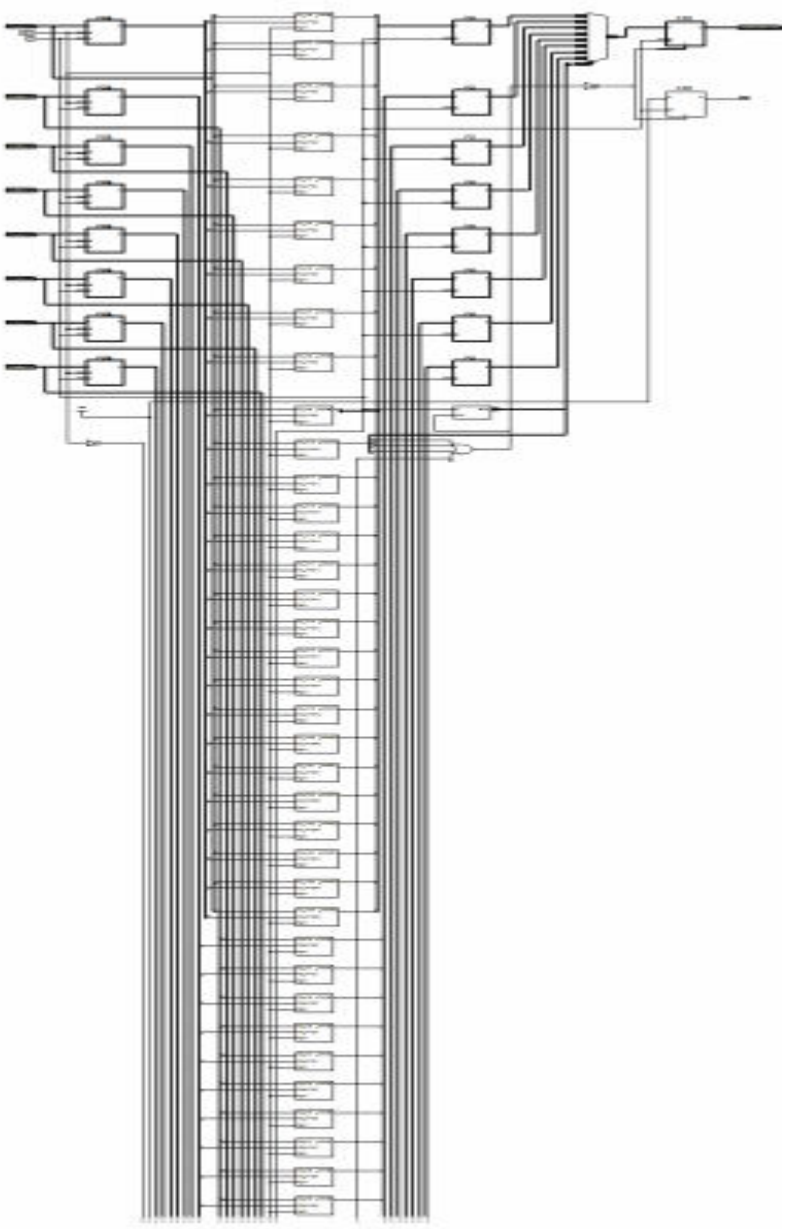
Pll_syndesi


dat1(12:0)
address(3:0)
dat2(12:0)
dat3(12:0)
dat4(12:0)
dat5(12:0)
dat_out(15:0)
dat6(12:0)
dat7(12:0)
dat8(12:0)
clk
din_val
dat_val

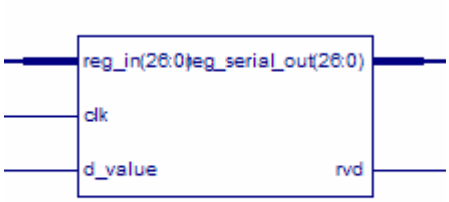


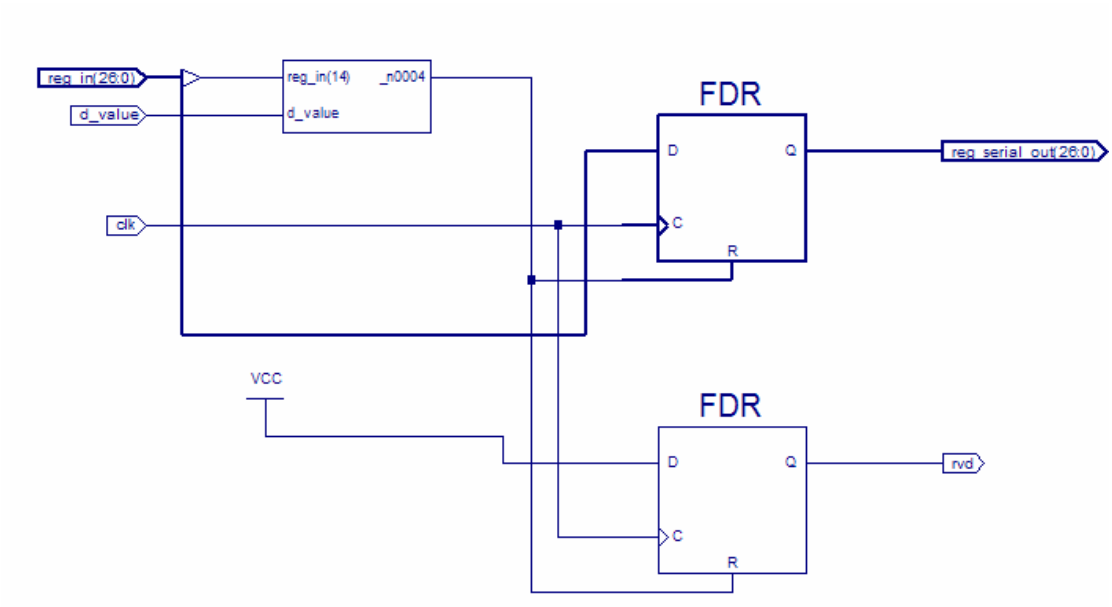
Regist :


reg1_in(26:0)
reg2_in(26:0)
reg3_in(26:0)
reg4_in(26:0)
reg5_in(26:0)
reg6_in(26:0)
reg7_in(26:0)
reg8_in(26:0)
clk
da_val
reg_serial_out(26:0)
rvd

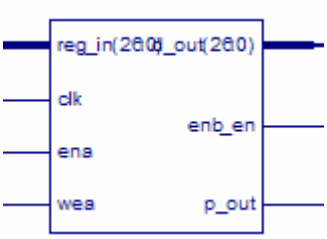


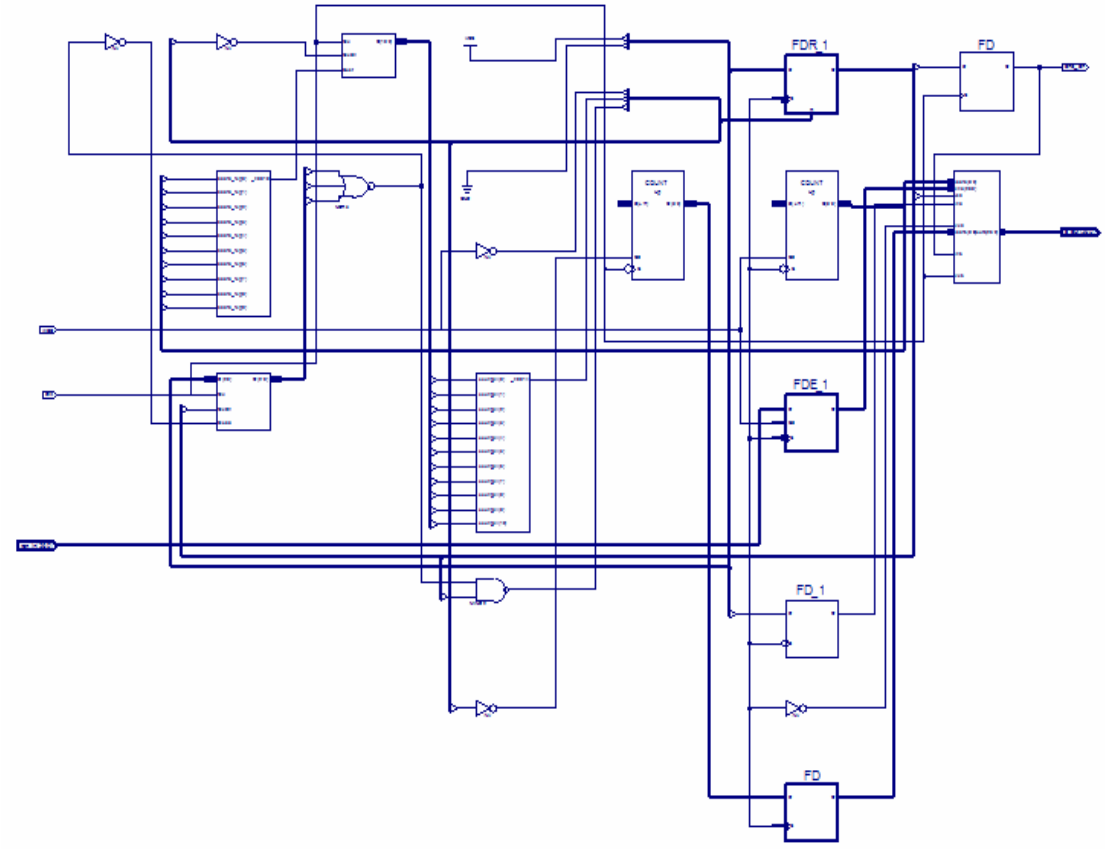
no_negative_regs:



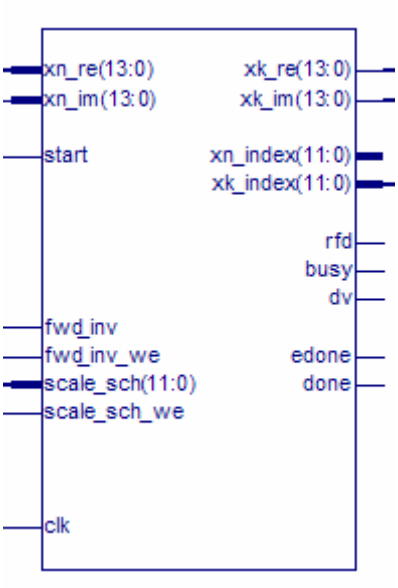


Control Of Ram :

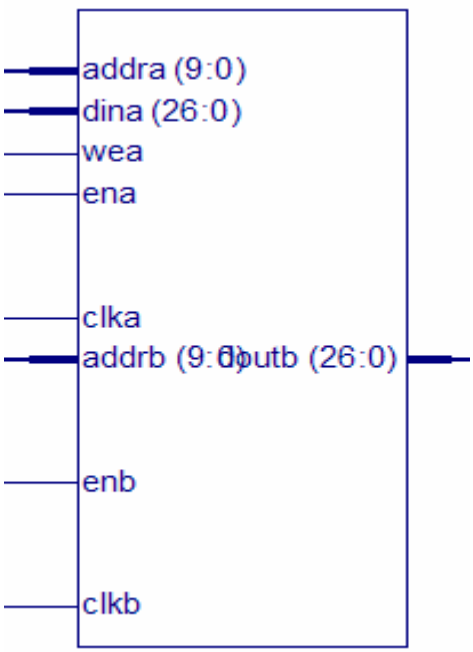




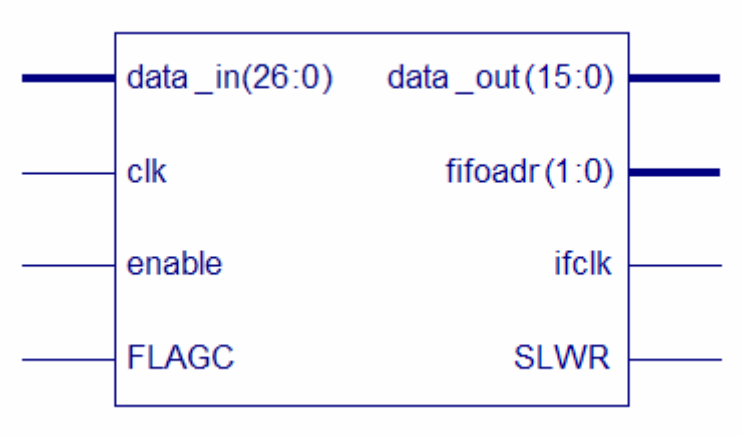
FFT :

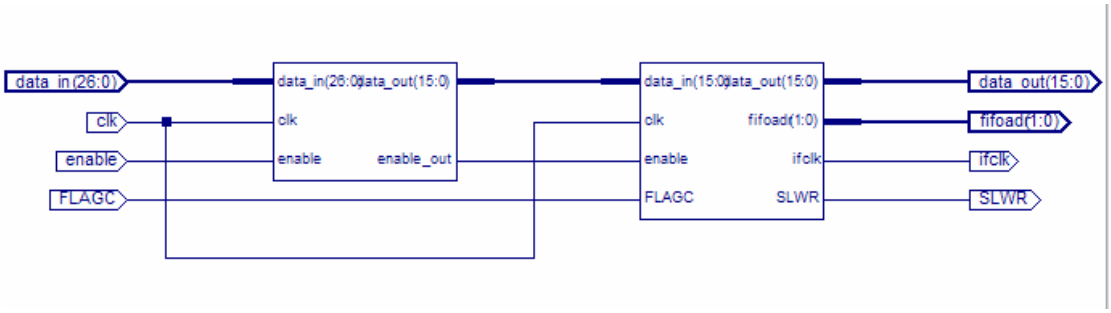

xn_re(13:0)
xn_im(13:0)
start
fwd_inv
fwd_inv_we
scale_sch(11:0)
scale_sch_we
clk
xk_re(13:0)
xk_im(13:0)
xn_index(11:0)
xk_index(11:0)
rfd
busy
dv
edone
done

DP RAM 6.3 :

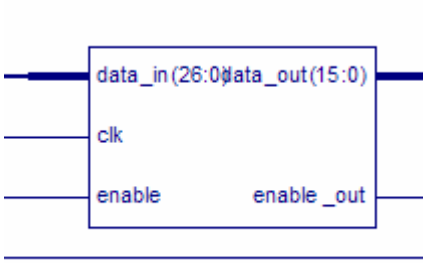

addra (9:0)
dina (26:0)
wea
ena
clka
addrb (9:0)
doutb (26:0)
enb
clkb

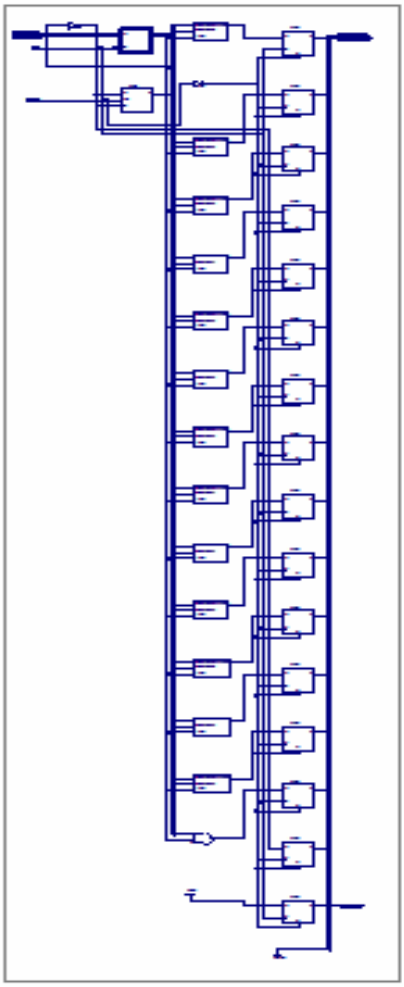
MASTER_USB_CONN:



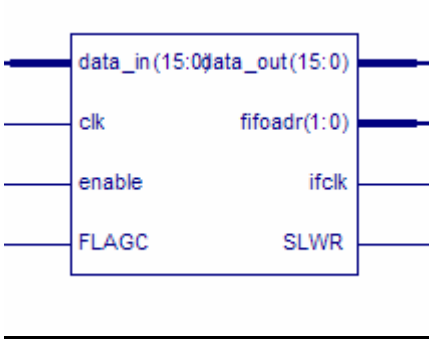


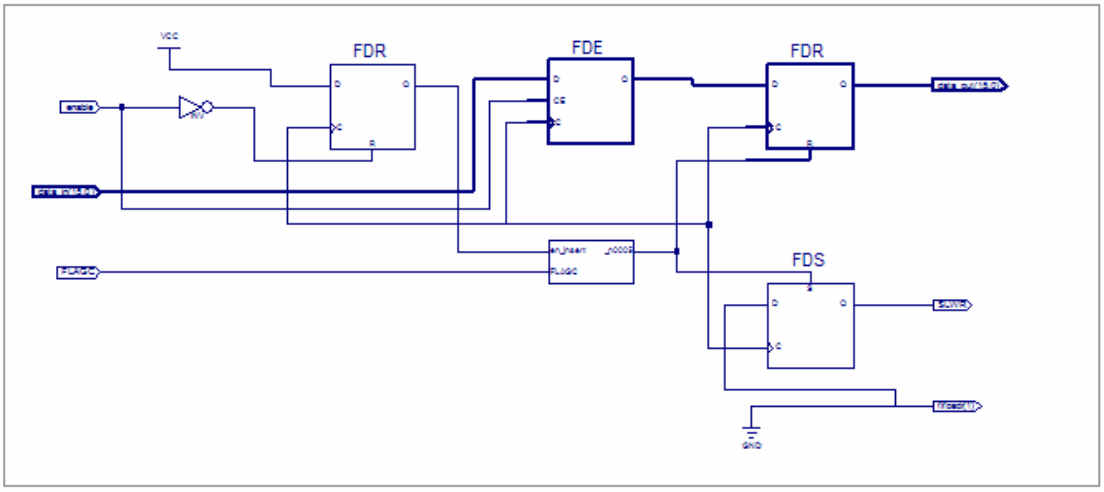
Receive data :





Intercoonection:





Βιβλιογραφία :

VHDL:

1.Sudhakar Yalamanchili ‘VHDL Starter’s Guide’.
2.Κ.Ζ.Πεκμεστζή Ψηφιακά Συστήματα VLSI

Fourier Transform

1 G. Bachman L. Nasrici E. Beckenstein Fourier and Wavelets Analysis
2 Springer Texts in Applied Mathematics

Α/Ψ Μετατροπέας

1 Analog Devices Datasheet AD12401

Xilinix Virtex 4 ISE

1 Virtex User Guide
2 XST User Guide

USB Microcontroller

1 EZ-USB Technical Reference Manual EZ-USB_TRM
2 Datasheet cy7c68013a_8

Web Sites

www.wikipedia.org
www.xilinx.com
www.usb.org
www.cypress.com