Ηράκλειο - 2015

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΤΕΙ Κρητης - Τμημα Μηχανικων Πληροφορικης

Ανάπτυξη συστηματος κατανεμημενης μνημης σε πολυπυρηνα ενσωματωμενα συστηματα

Δημήτριος Βουρβουλάκης – Α.Μ: 2418, Παναγιώτης Χριστοδούλου – Α.Μ: 2639 | Επιβλέπων καθηγητής: Γεώργιος Κορνάρος

Ευχαριστίες

Η παρούσα πτυχιακή εργασία, εκπονήθηκε από τους φοιτητές Παναγιώτη Χριστοδούλου και Δημήτρη Βουρβουλάκη του τμήματος Μηχανικών Πληροφορικής του ΤΕΙ Κρήτης, κατά το ακαδημαϊκό έτος 2013 – 2014 υπό την επίβλεψη του καθηγητή κ. Γεώργιου Κορνάρου και παραδόθηκε τον Ιανουάριο του έτους 2015. Στον καθηγητή μας, κύριο Κορνάρο οφείλουμε τις θερμές μας ευχαριστίες για την υποστήριξη και την καθοδήγηση του, κατά την διάρκεια της πτυχιακής μας εργασίας.

Abstract

The purpose of the thesis, is the development of distributed memory in embedded multinucleated systems. In this case, we will use reprogrammable integrated circuits FPGA's, which will develop distributed memories. In terms of architecture, the FPGA's contain general purpose circuits CLB (Combination logic block) that have two basic functions:

- to perform logical functions and,
- to have the potential to function as of RAM.

Our goal, is to simultaneously connect a number of FPGA's, which will use both functions mentioned above, to achieve fast processing applications multinucleated systems. The thesis will be developed in the Xilinx's board, Spartan 3. VHDL language used to develop the necessary source code.

Σύνοψη

Σκοπός της πτυχιακής εργασίας, είναι η ανάπτυξη κατανεμημένων μνημών σε πολυπύρηνα ενσωματωμένα συστήματα. Στην προκειμένη περίπτωση, θα χρησιμοποιήσουμε επαναπρογραμματιζόμενα ολοκληρωμένα κυκλώματα FPGA's, στα οποία θα αναπτύξουμε κατανεμημένες μνήμες. Από άποψη αρχιτεκτονικής, τα FPGA's περιέχουν γενικού σκοπού κυκλώματα CLB (Combination logic block) τα οποία έχουν δυο βασικές λειτουργίες:

- εκτελούν λογικές συναρτήσεις και
- έχουν την δυνατότητα να λειτουργούν σαν μνήμες RAM.

Στόχος μας, είναι να συνδέσουμε ταυτόχρονα κάποιο αριθμό από FPGA's, τα οποία θα χρησιμοποιούν και τις δύο λειτουργίες που προαναφέρθηκαν, για να επιτύχουμε γρήγορη επεξεργασία εφαρμογών σε πολυπύρηνα συστήματα. Η πτυχιακή εργασία, θα αναπτυχθεί στο board της Xilinx, Spartan 3. Θα χρησιμοποιηθεί η γλώσσα VHDL, για την ανάπτυξη του απαραίτητου πηγαίου κώδικα.

Πίνακας περιεχομένων

Πίνακας εικόνων

Λίστα Πινάκων

1. Εισαγωγή

Τα FPGA είναι ολοκληρωμένα κυκλώματα γενικού σκοπού. Απο άποψη αρχιτεκτονικής, περιέχουν κυκλώματα γενικού σκοπού, τα CLB. Τα CLB έχουν δύο λειτουργίες. Μπορούν είτε να εκτελούν λογικές συναρτήσεις, είτε να λειτουργούν ως μνήμη RAM. Αυτό είναι το στοιχείο που διαφοροποιεί τα FPGA από τα υπόλοιπα ολοκληρωμένα κυκλώματα.

1.1. Περίληψη

i. Πληροφορίες για τα FPGA

Το FPGA ή Field Programmable Gate Array είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL και άλλα. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία.

Για να προγραμματίσουμε ένα FPGA, γράφουμε τον κώδικά μας σε γλώσσα περιγραφής υλικού HDL (Hardware Description Language). Οι πιο γνωστές περιγραφής υλικού είναι η VHDL, η AHDL και η Verilog. Στην πτυχιακή μας, χρησιμοποιούμε την γλώσσα VHDL.

Το FPGA έχει παρόμοιο πεδίο εφαρμογών με άλλα προγραμματιζόμενα ολοκληρωμένα ψηφιακά κυκλώματα όπως τα PLD και τα ASIC. Όμως τα ιδιαίτερα χαρακτηριστικά του FPGA είναι τα εξής:

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

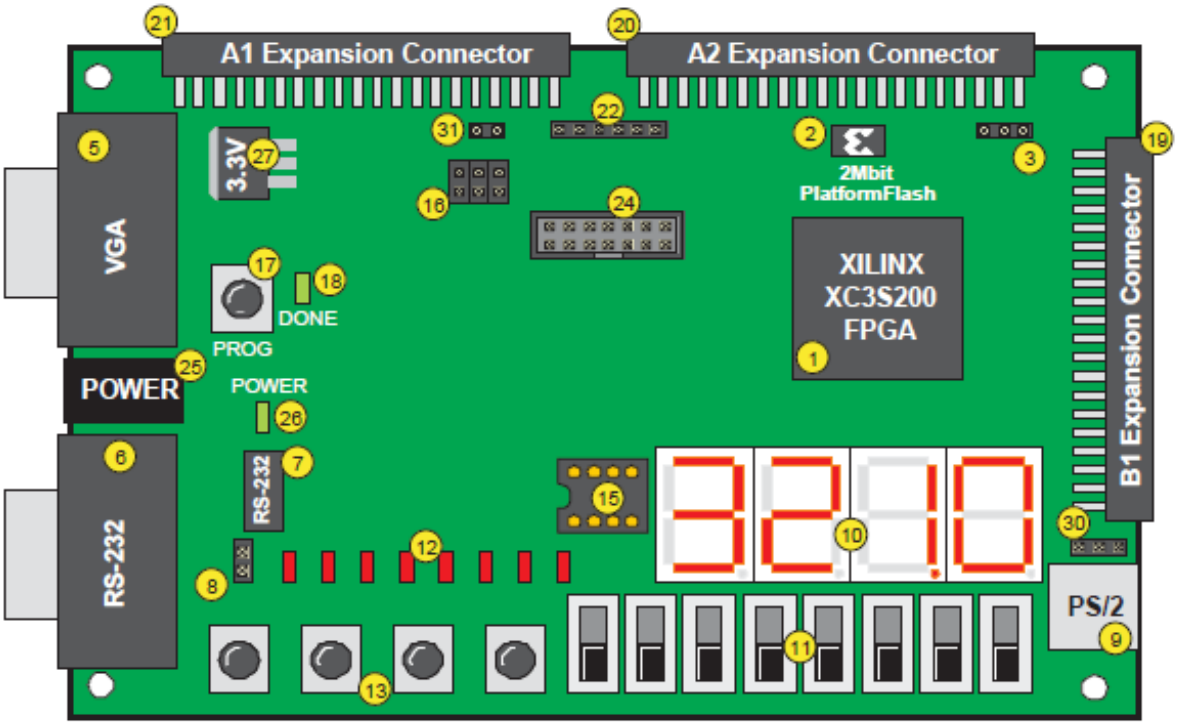
Έτσι το FPGA είναι ιδιαίτερα κατάλληλο εκεί που οι παράμετροι λειτουργίας πρέπει να αλλάζουν συχνά ή σε μικρές ποσότητες παραγωγής, ενώ το ASIC, λόγω μαζικής παραγωγής, είναι φτηνότερο εκεί που απαιτούνται μεγάλες ποσότητες και η επιθυμητή λειτουργία είναι αυστηρά προκαθορισμένη (το ASIC δεν επαναπρογραμματίζεται).

Βασική δομική μονάδα του FPGA είναι το λογικό μπλοκ, με τη χρήση του οποίου υλοποιούνται οι λογικές συναρτήσεις που εκφράζουν τη λειτουργία ενός ψηφιακού κυκλώματος. Ανάλογα με το μέγεθος του κυκλώματος πολλά λογικά μπλοκ συνδέονται για να υλοποιήσουν το πλήθος των απαραίτητων λογικών συναρτήσεων.

ii. Το board Spartan 3 της Xilinx

Το FPGA που χρησιμοποιούμε, είναι ενσωματωμένο στο board **Spartan 3**, της αμερικάνικης εταιρίας Xilinx. Το board αυτό, έχει εξοπλιστεί με περιφερειακές θύρες που επιτρέπουν εν τέλει στο FPGA, να χειρίζεται περιφερειακά συστήματα όπως:

- Οθόνη
- Πληκτρολόγιο
- Ποντίκι
- Σειριακές θύρες
- Άλλες ηλεκτρονικές συσκευές μέσω των expansion connectors



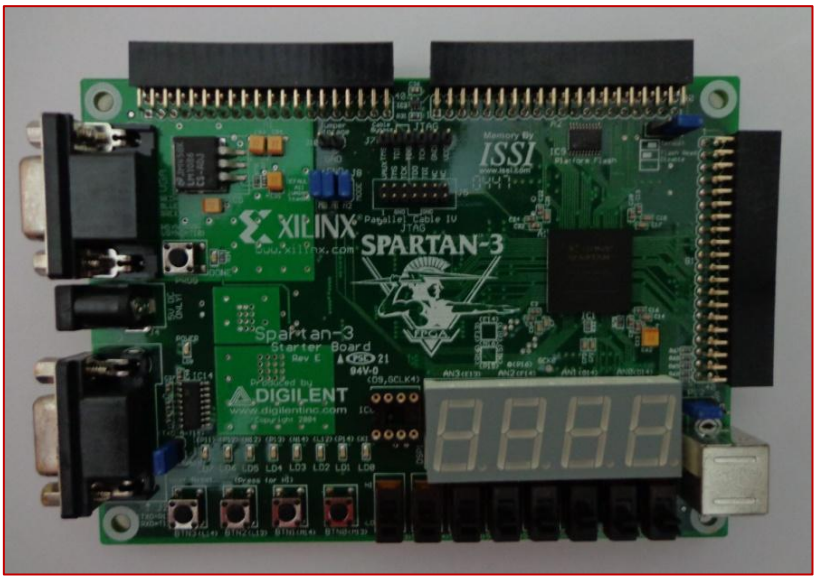
Εικόνα 1 - Επάνω όψη του Spartan 3

Πιο αναλυτικά, στην παραπάνω εικόνα βλέπουμε πως υπάρχουν οι εξής θύρες:

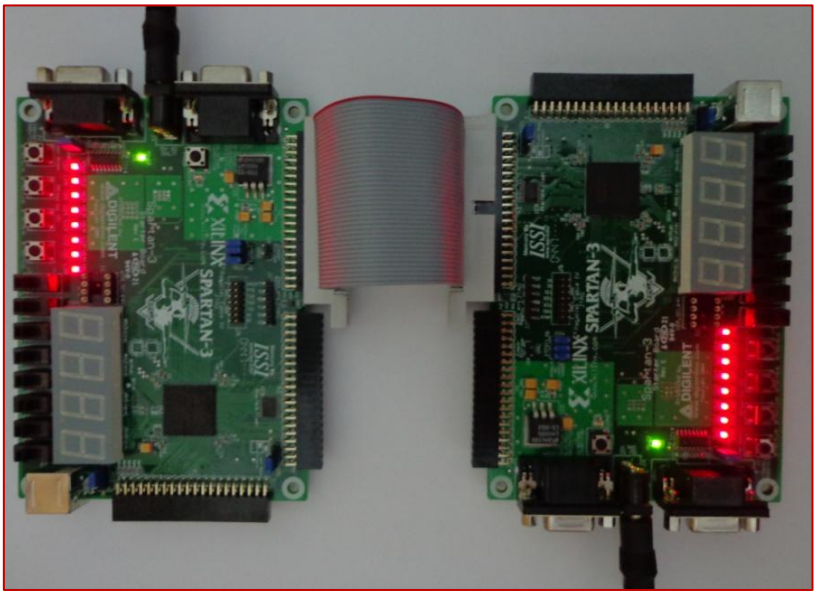
- Ο αριθμός 5, αντιστοιχεί στην θύρα VGA, που επιτρέπει σύνδεση με οθόνη.
- Ο αριθμός 6, αντιστοιχεί σε σειριακή θύρα, που επιτρέπει σύνδεση με υπολογιστή.
- Ο αριθμός 9, αντιστοιχεί σε θύρα πληκτρολογίου/ποντικιού.
- Οι αριθμοί 19, 20, 21 είναι οι τρείς expansion connectors, που μας δίνουν την δυνατότητα να συνδέσουμε τα FPGA μεταξύ τους. Κάθε expansion connector, έχει 40 εισόδους - εξόδους και κάθε μία από αυτές, είναι είσοδος - έξοδος του ενός bit.
- Ο αριθμοί 22 και 24, είναι οι θύρες μεταφοράς δεδομένων JTAG, που επιτρέπουν την σύνδεση με προσωπικό υπολογιστή, τον προγραμματισμό του board και την μεταφορά δεδομένων μεταξύ των δύο συσκευών.

Ακόμα, στην εικόνα 1 υπάρχουν επισημάνσεις οι οποίες δεν είναι θύρες, αλλά είναι σημαντικά στοιχεία του FPGA τα οποία θα μας απασχολήσουν στην παρούσα πτυχιακή. Κάποια από αυτά είναι τα εξής:

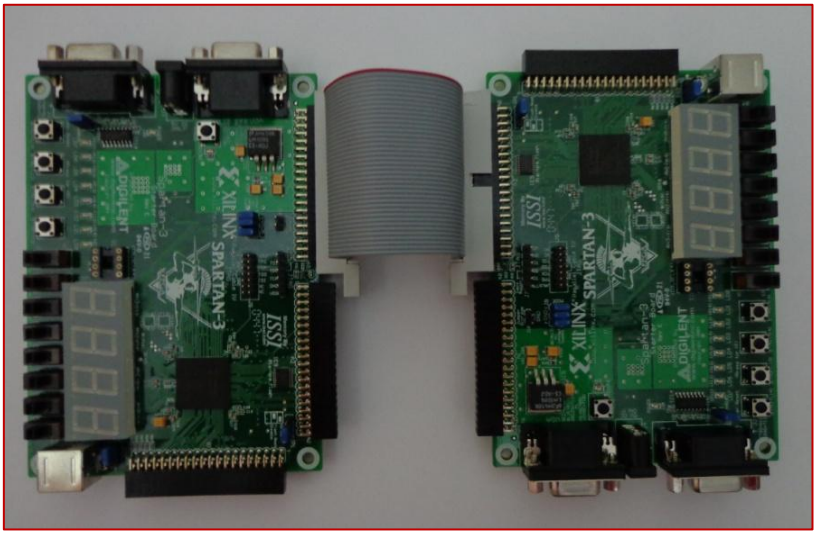
- Ο αριθμός 1, είναι το chip FPGA.
- Ο αριθμός 10, αντιστοιχεί στο Seven Segment Display, το οποίο είναι ουσιαστικά μια οθόνη 4 κομματιών, με 7 στοιχεία ανά κομμάτι.
- Ο αριθμός 11, αντιστοιχεί σε 8 διακόπτες (switches) που όταν ενεργοποιηθούν, ο καθένας δίνει λογικό άσσο.
- Ο αριθμός 12, αντιστοιχεί στα 7 leds του board.
- Ο αριθμός 13, αντιστοιχεί σε 4 κουμπιά (push buttons), που όταν πατηθούν δίνουν λογικό άσσο, για όσο διάστημα είναι πατημένα.
- Ο αριθμός 17, κάνει reset το FPGA (το επαναπρογραμματίζει στην αρχική του κατάσταση).
- Ο αριθμός 18 ενεργοποιείται κατά την διάρκεια του reset.
- Ο αριθμός 25, αντιστοιχεί στην υποδοχή της τροφοδοσίας.
- Ο αριθμός 26, είναι ένα λαμπάκι που μας ενημερώνει για την ύπαρξη τάσης στο board.



Εικόνα 2 - Το board της Xilinx (Spartan - 3)



Εικόνα 3 - Δύο boards συνδεδεμένα με καλωδιοταινία (και παροχή τάσης)



Εικόνα 4 - Δύο boards συνδεδεμένα με καλωδιοταινία

1.2. Κίνητρο για την διεξαγωγή της εργασίας

Οι δυνατότητες που σου παρέχει ένα κύκλωμα FPGA, είναι πολλές. Ουσιαστικά, εφόσον είναι ένα αυτοδύναμο σύστημα, έχεις την δυνατότητα να φτιάξεις ένα δικό σου υπολογιστικό σύστημα. Στο board της Xilinx, έχουμε την δυνατότητα, όπως προαναφέραμε, να προσθέσουμε συσκευές εισόδου (ποντίκι, πληκτρολόγιο) και συσκευές εξόδου (οθόνη). Βέβαια, το μειονέκτημα αυτού του υπολογιστικού συστήματος, είναι ότι απαιτεί εξωτερικό μικροεπεξεργαστή ή μνήμη με μόνιμη συγκράτηση δεδομένων από το οποίο θα προγραμματίζεται, εφόσον δεν υπάρχει αυτοδύναμη μνήμη συγκράτησης δεδομένων.

Η σκέψη μας, πάνω στην οποία εν τέλει βασίσαμε την εργασία, είναι πως θα μπορέσουμε να εκμεταλλευτούμε ένα τέτοιο σύστημα (FPGA και PC) ώστε μελλοντικά να βελτιώσουμε και να παραμετροποιήσουμε τις αποδόσεις του υπολογιστικού συστήματος, βάσει των δικών μας αναγκών και των εκάστοτε αναγκών που θα προκύψουν. Οι εκάστοτε ανάγκες που θα προκύψουν, μπορούν να υλοποιηθούν από εμάς τους ίδιους, αν εκμεταλλευτούμε το γεγονός ότι, εν αντιθέσει με τα ASIC, τα FPGA είναι επαναπρογραμματιζόμενα.

1.3. Σκοπός και στόχοι της εργασίας

Η εργασία μας, έχει σαν σκοπό να αναπτύξουμε μεθόδους, οι οποίες θα προσφέρουν βελτίωση στην απόδοση της επεξεργασίας δεδομένων μέσω των κυκλωμάτων FPGA. Στόχος μας, είναι να συνδέσουμε ταυτόχρονα κάποιο αριθμό από FPGA, έτσι ώστε να επιτύχουμε γρήγορη επεξεργασία εφαρμογών σε πολυπύρηνα συστήματα. Ένα παράδειγμα είναι να μπορεί να γίνει η επεξεργασία πολυμεσικών εφαρμογών στα FPGA, έτσι ώστε να μην καταναλώνεται επεξεργαστική ισχύ από τον υπολογιστή μας.

1.4. Δομή εργασίας

Η εργασία, αποτελείται από δύο μέρη. Το πρώτο μέρος, θα αποτελέσει μια εισαγωγή στον κόσμο των FPGA και συγκεκριμένα, θα δείξουμε

- την δομή του board
- τα περιφερειακά του μέρη (switches, expansion connectors, push buttons)
- την διασύνδεση και αλληλεπίδραση συνδεδεμένων boards, εκμεταλλευόμενοι κυρίως τις εξωτερικές τους συνδέσεις

έτσι ώστε στο δεύτερο μέρος να επιτύχουμε την ταυτόχρονη επεξεργασία δεδομένων σε συνδεδεμένα FPGA’s.

2. Μεθοδολογία υλοποίησης

2.1. Μέθοδος ανάλυσης και ανάπτυξης πτυχιακής

Το πρόγραμμα που θα χρησιμοποιήσουμε καθ' όλη την διάρκεια της πτυχιακής εργασίας, είναι το **Xilinx ISE Design Suite 14.2.** Τα βασικά βήματα της λειτουργίας του προγράμματος αυτού, θα αναλυθούν παρακάτω έτσι ώστε ο αναγνώστης να είναι σε θέση, ακόμα και αν δεν έχει ιδιαίτερες γνώσεις, να μπορεί να περιηγηθεί και είναι σε θέση να το χρησιμοποιήσει.

Στο πρώτο μέρος της πτυχιακής, δημιουργούμε κώδικα έτσι ώστε να καταλάβουμε την λειτουργία του board. Έπειτα, τις γνώσεις που αποκομίσαμε, θα τις χρησιμοποιήσουμε στο δεύτερο μέρος της εργασίας, όπου έχουμε σκοπό να πετύχουμε διαμερισμό και ταχύτερη επεξεργασία των δεδομένων. Οι τρείς κατηγορίες του πρώτου μέρους, πάνω στις οποίες εργαστήκαμε είναι οι εξής:

- Μεταφορά δεδομένων από τις εισόδους (push buttons / switches) του πρώτου board και εμφάνιση αποτελεσμάτων στις εξόδους leds του δεύτερου board και αντίστροφα, δηλαδή από το δεύτερο board στο πρώτο.
- Μεταφορά δεδομένων από τις εισόδους (push buttons / switches) του πρώτου board και εμφάνιση αποτελεσμάτων στις εξόδους του Seven Segment Display (SSD) του δεύτερου board και αντίστροφα.
- Δημιουργία μνημών fifo στο FPGA του πρώτου board και αποστολή των δεδομένων της fifo, αρχικά στα leds και έπειτα στις εξόδους expansion connectors, έτσι ώστε να εμφανιστούν τα αποτελέσματα και στο δεύτερο board και αντίστροφα.

Οι παραπάνω 3 κατηγορίες, θα αναλυθούν εκτενέστερα στο κεφάλαιο 4 (Κύριο μέρος της εργασίας, «Μέρος Α»).

Στο δεύτερο μέρος της πτυχιακής, θα αξιοποιήσουμε τις πληροφορίες και τις γνώσεις που έχουμε αποκομίσει από το πρώτο μέρος των πειραματικών δοκιμών της εργασίας μας, έτσι ώστε να χρησιμοποιούμε κάθε μέρος του FPGA και του board για να μπορέσουμε να χρησιμοποιούμε τις Sram των συνδεδεμένων boards.

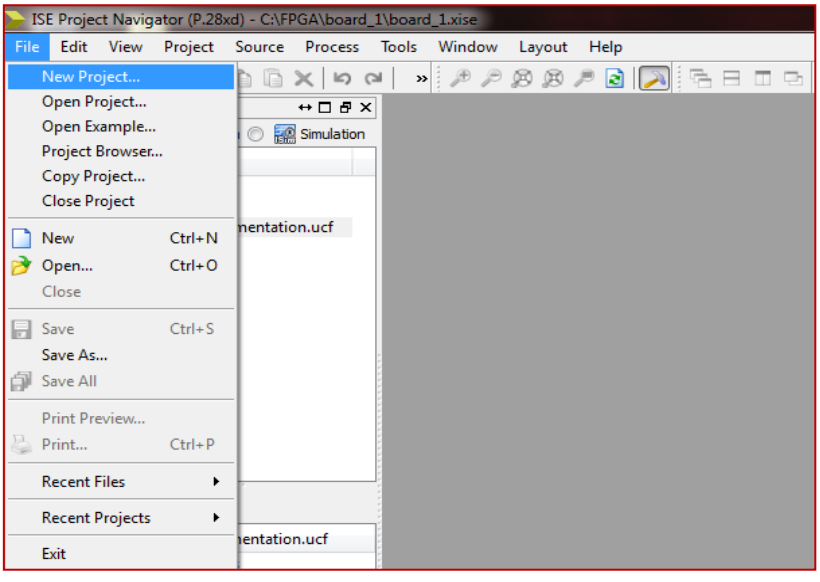
- Θα δημιουργήσουμε δικά μας κομμάτια κώδικα, τα οποία θα είναι προσαρμοσμένα, έτσι ώστε μέρος των δεδομένων να αποθηκεύονται τόσο στην Sram του FPGA, όσο και στην Sram ενός δεύτερου FPGA που έχουμε συνδέσει, το οποίο δεν εκτελεί κάποια διεργασία, άρα θα μπορεί να χρησιμοποιηθεί ώστε να επεξεργαστεί δεδομένα για λογαριασμό του πρώτου FPGA.

- Κάθε FPGA, θα έχει μια fifo η οποία θα στέλνει δεδομένα στην Sram αλλά και σε άλλα διαθέσιμα board, με τα οποία είναι ήδη συνδεδεμένα έτσι ώστε αυτά τα board να επεξεργαστούν τα εισερχόμενα δεδομένα στα FPGA τους και έπειτα να στείλουν τα αποτελέσματα πίσω στο αρχικό FPGA.

2.2. Το πρόγραμμα Xilinx ISE Design Suite 14.2

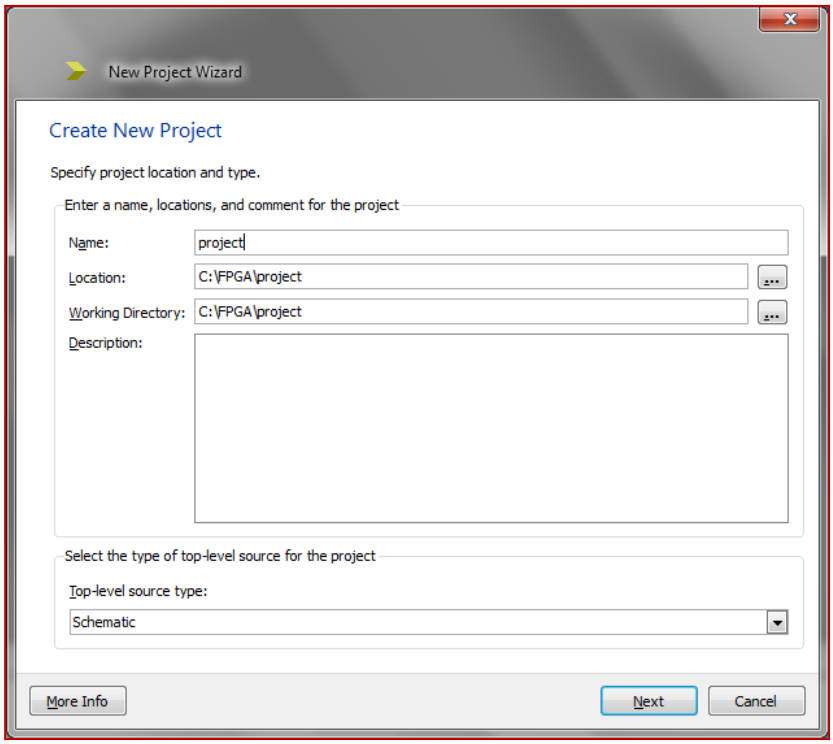
Το κύριο μέρος της εργασίας, είναι υλοποιημένο σε κώδικα VHDL. Ο κώδικας αυτός έχει αναπτυχθεί, μέσω του προγράμματος **Xilinx ISE Design Suite 14.2.** Το συγκεκριμένο πρόγραμμα μας δίνει αρκετές επιπλέον δυνατότητες έτσι ώστε να μπορούμε να συνδυάζουμε αρκετά εργαλεία μαζί. Το πρόγραμμα αυτό, είναι ένα εργαλείο λογισμικού της εταιρίας **Xilinx**, που επιτρέπει στον χρήστη - προγραμματιστή να συνθέσει τα δικά του σχέδια σε γλώσσες περιγραφής υλικού. Στη μηχανική υπολογιστών, μια γλώσσα περιγραφής υλικού (hardware description language ή **HDL**) είναι μια γλώσσα που ανήκει σε μια κλάση γλωσσών προγραμματισμού, γλωσσών προδιαγραφών ή γλωσσών μοντελοποίησης για την τυπική περιγραφή και σχεδίαση ηλεκτρονικών κυκλωμάτων και συνηθέστερα, ψηφιακής λογικής. Μπορεί να περιγράψει τη λειτουργία, τη σχεδίαση και την οργάνωση του κυκλώματος, μαζί με δοκιμές που επιβεβαιώνουν τη λειτουργία του μέσω προσομοίωσης. Οι γλώσσες περιγραφής υλικού χρησιμοποιούνται για τη συγγραφή εκτελέσιμων προδιαγραφών για κάποιο συγκεκριμένο υλικό (στην περίπτωσή μας, κυκλώματα γενικού σκοπού FPGA). Ξεκινώντας, για να δημιουργήσουμε τον δικό μας κώδικα σε γλώσσα VHDL, κάνουμε τα εξής βήματα:

α) Εφόσον ανοίξουμε το πρόγραμμα **ISE Design Suite 14.2**, στο κεντρικό παράθυρο, επιλέγουμε από το μενού επιλογών, File > New Project...



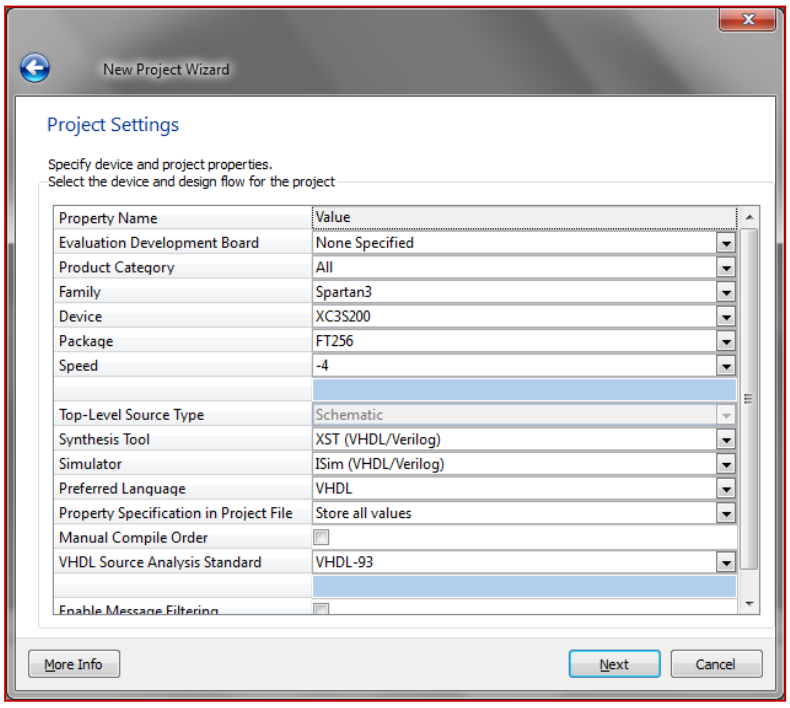
Εικόνα 5 – Η αρχική οθόνη του προγράμματος

β) Στο νέο παράθυρο που ανοίγεται, δίνουμε το επιθυμητό όνομα, ορίζουμε την τοποθεσία που θέλουμε να αποθηκευτεί, δίνουμε τυχόν πληροφορίες στο πεδίο Description και πατάμε το next (εικόνα 6).



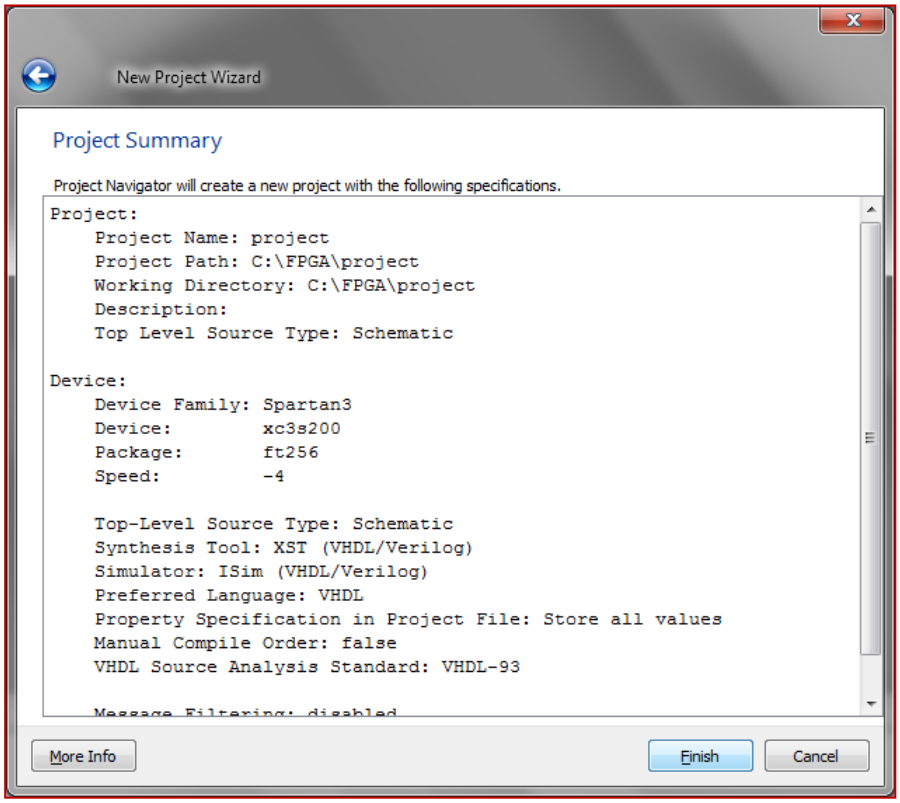
Εικόνα 6 - Ξεκινώντας ένα νέο project

Στο νέο παράθυρο (εικόνα 7) πατάμε next, χωρίς να αλλάξουμε τις ρυθμίσεις.



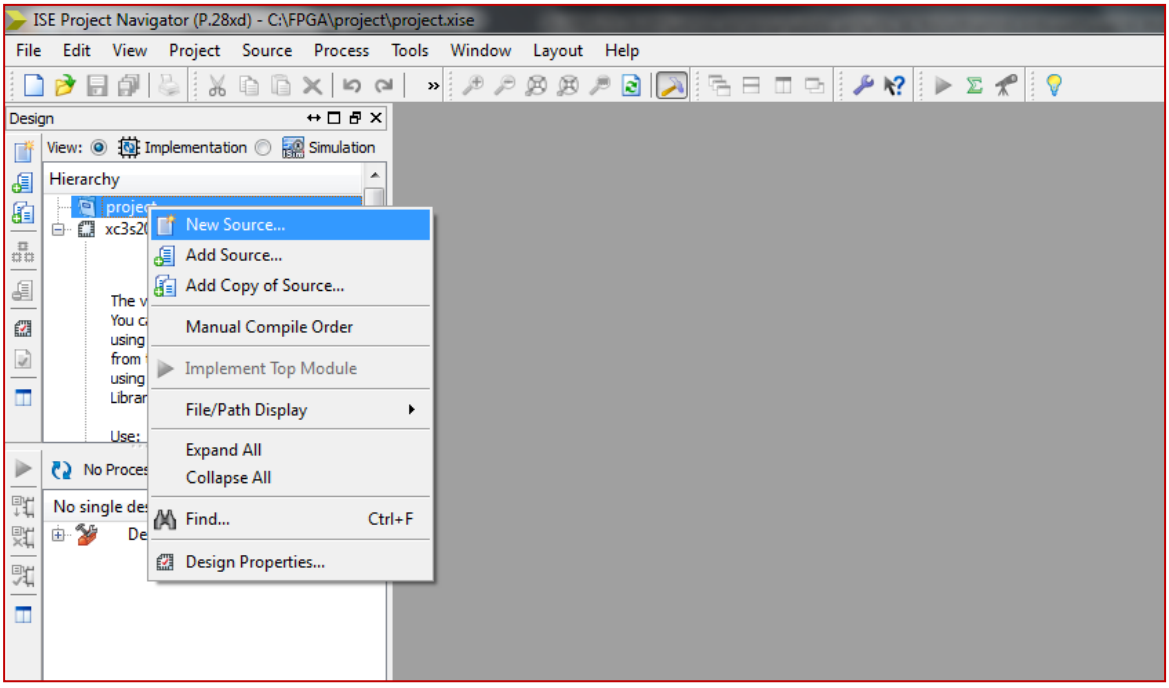
Εικόνα 7 - Αρχικές ρυθμίσεις ενός νέου project

Τέλος στο τρίτο παράθυρο (εικόνα 8) πατάμε finish.



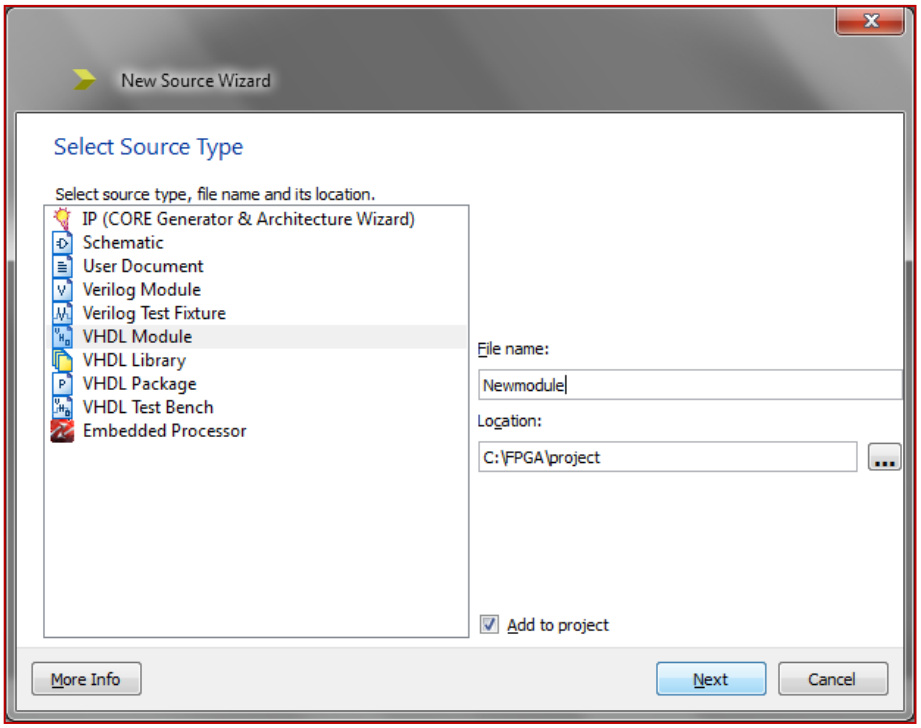
Εικόνα 8 - Το project ολοκληρώθηκε

γ) Το πρότζεκτ μας έχει δημιουργηθεί. Τώρα πατάμε δεξί κλικ στο πρότζεκτ και επιλέγουμε το New Source...



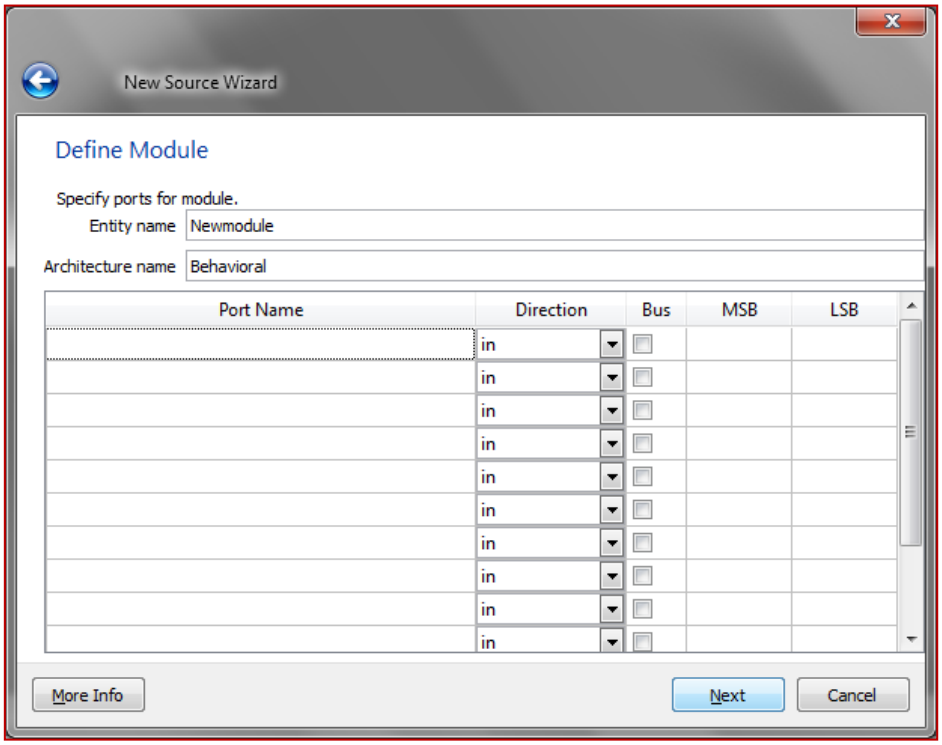
Εικόνα 9 - Προσθέτουμε ένα νέο module

Στο νέο παράθυρο που θα μας ανοίξει (εικόνα 10), δίνουμε το όνομα που θέλουμε στο νέο αρχείο, επιλέγουμε από την λίστα πηγών το VHDL Module και πατάμε next.



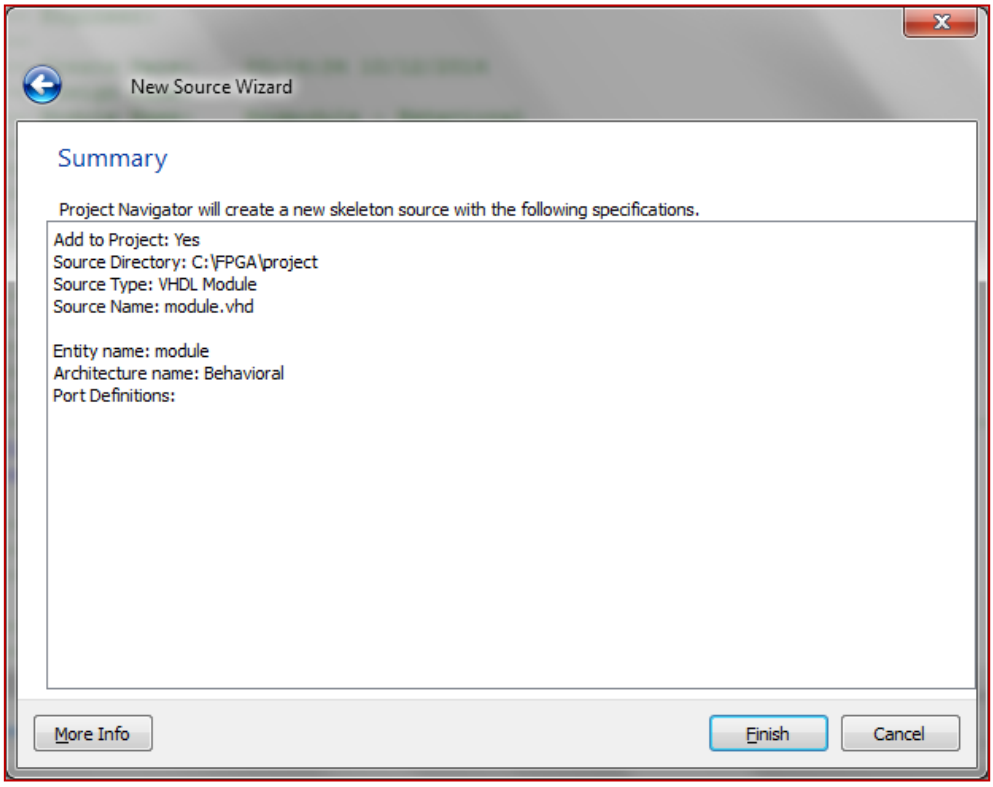
Εικόνα 10 - Το VHDL module, είναι το αρχείο που χρειαστούμε

Στα επόμενα δύο παράθυρα, πατάμε next...



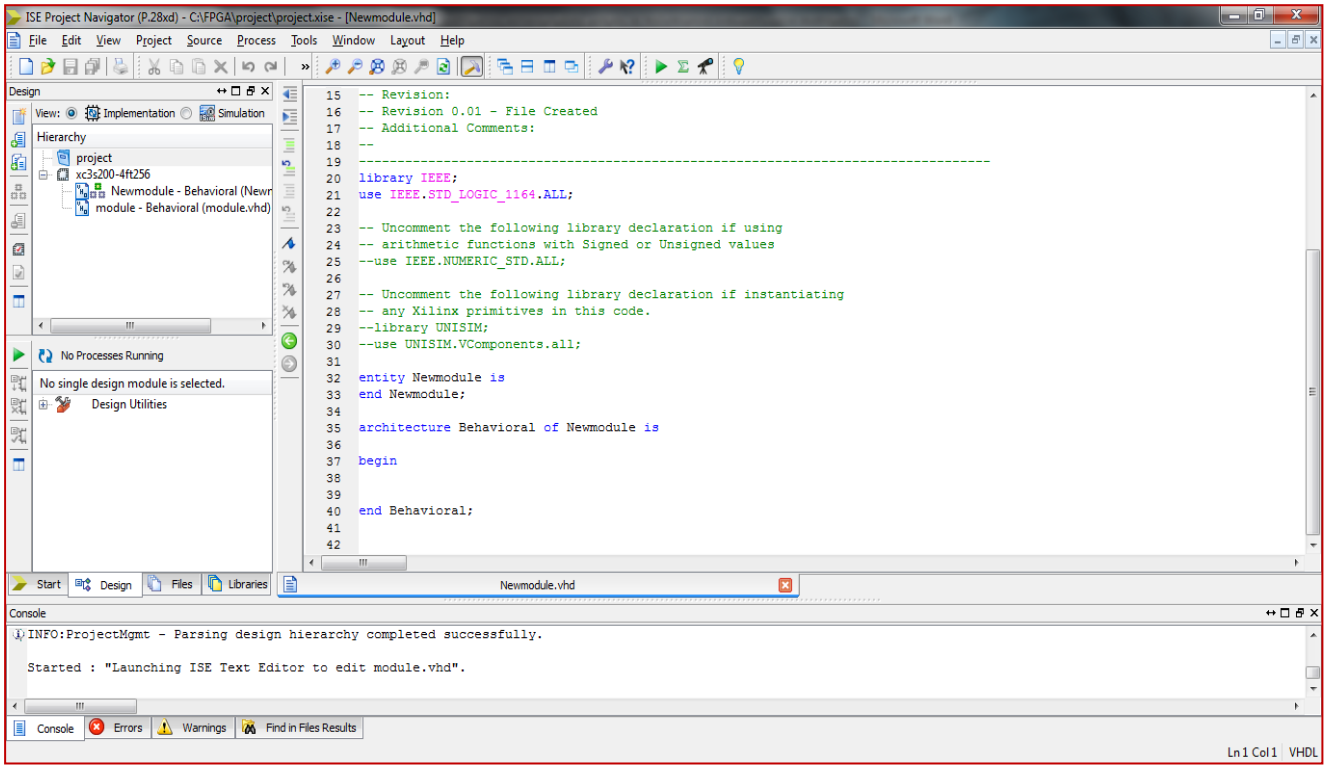
Εικόνα 11 - Επιπλέον παραμετροποιήσεις που δεν χρειάζονται αλλαγή

και finish.



Εικόνα 12 - Το VHDL module, δημιουργήθηκε

Είμαστε πλέον έτοιμοι να ξεκινήσουμε να γράφουμε κώδικα.



Εικόνα 13 - Το project δημιουργήθηκε και πλέον γράφουμε κώδικα

Το αρχείο VHDL module, με όνομα Newmodule, είναι το κύριο αρχείο στο οποίο γράφουμε τον κώδικά μας. Μπορούμε να προσθέσουμε και άλλα αρχεία στο πρότζεκτ, είτε VHDL module, είτε ένα εκ των υπόλοιπων επιλογών της εικόνας 10. Επίσης, σημαντικό αρχείο είναι το ucf, το οποίο το δημιουργούμε κάνοντας δεξί κλικ στο module και επιλέγοντας ucf. Σε αυτό το αρχείο, συνδέουμε τις μεταβλητές του κώδικά μας, με το hardware του board. Για παράδειγμα, στο ucf η εντολή **NET "sw(0)" LOC = "F12";** , σημαίνει ότι στον VHDL κώδικά μας έχουμε την μεταβλητή sw(0). Η μεταβλητή αυτή, συνδέεται με το FPGA pin F12, το οποίο είναι το πρώτο switch από δεξιά (εικόνα 1). Παρακάτω, θα δούμε και το αρχείο IP (Core generator & architecture wizard).

Στο VHDL κώδικά μας, θα δούμε δύο ενότητες. Την ενότητα entity και την ενότητα architecture.

Η ενότητα **entity** (οντότητα), είναι μια λίστα με τις προδιαγραφές όλων των ακροδεκτών (θυρών, Ports) εισόδου και εξόδου του κυκλώματος. Η σύνταξη της φαίνεται παρακάτω.

```
ENTITY όνομα_οντότητας IS
        PORT (
            Όνομα_θύρας: κατάσταση_σήματος     τύπος σήματος;
            );
END όνομα_οντότητας;
```

Η κατάσταση ενός σήματος μπορεί να είναι IN (είσοδος), OUT (έξοδος), INOUT (είσοδος - έξοδος), ή BUFFER (προσωρινή αποθήκευση).
Ο τύπος ενός σήματος μπορεί να είναι BIT, STD_LOGIC, INTEGER, κ.λπ.

Η ενότητα **architecture** (αρχιτεκτονική) είναι ένα μέρος του VHDL module, το οποίο περιέχει την περιγραφή του τρόπου με τον οποίο πρέπει να συμπεριφέρεται (δηλαδή να λειτουργεί) το κύκλωμα. Η σύνταξη της είναι η ακόλουθη:

```
ARCITECTURE όνομα_αρχιτεκτονικής OF όνομα_οντότητας IS
    [δηλώσεις]
BEGIN
    (κώδικας)
END όνομα_αρχιτεκτονικής;
```

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity test is
5  Port( a, b: IN BIT;
6        x: OUT BIT );
7  end test;
8
9  architecture Behavioral of test is
10 begin
11 x<=a NAND b;
12 end Behavioral;
```

Εικόνα 14 - Παράδειγμα entity και architecture

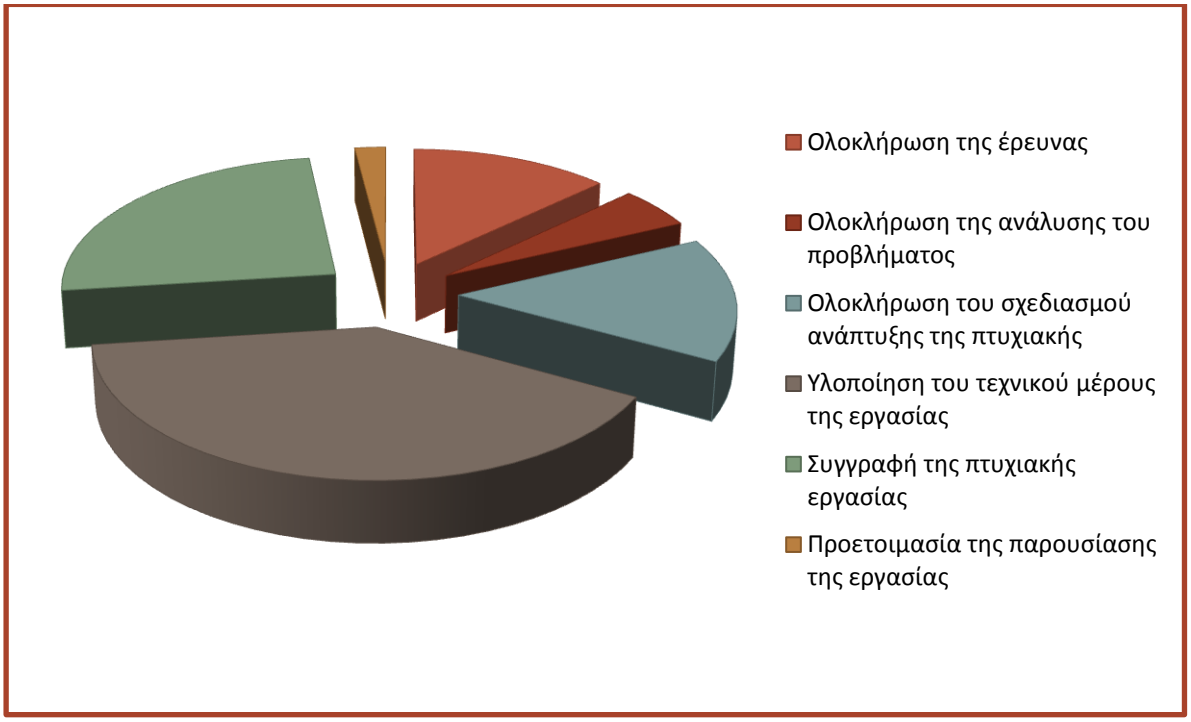
3. Σχέδιο δράσης για την εκπόνηση της πτυχιακής

3.1. State of art

3.2. Σημαντική στόχοι για την ολοκλήρωση της πτυχιακής

- Ολοκλήρωση της έρευνας 13 %
- Ολοκλήρωση της ανάλυσης του προβλήματος 5 %
- Ολοκλήρωση του σχεδιασμού ανάπτυξης της πτυχιακής 15 %
- Υλοποίηση του τεχνικού μέρους της εργασίας 40 %
- Συγγραφή της πτυχιακής εργασίας 25 %
- Προετοιμασία της παρουσίασης της εργασίας 2 %

3.2.1. Χρονοδιάγραμμα αποπεράτωσης της εργασίας



4. Κύριο μέρος της εργασίας

Η εργασίας μας, χωρίζεται σε 2 ενότητες - μέρη. Ο διαχωρισμός αυτός γίνεται επειδή και εμείς οι ίδιοι, αναλύσαμε τους στόχους της πτυχιακής μας εργασίας και αποφασίσαμε ότι αρχικά χρειάζεται να καταλάβουμε το πώς μεταφέρονται τα δεδομένα από το ένα board στο άλλο, πως συμπεριφέρονται οι μνήμες fifo κατά την μεταφορά δεδομένων από το ένα board στο άλλο (Μέρος Α) και εν τέλει, να υλοποιήσουμε την παράλληλη επεξεργασία των δεδομένων μεταξύ δύο ή και περισσότερων board της Xilinx (Μέρος Β).

4.1 Μέρος Α

4.1.1 Σύνδεση των δύο Spartan-3

Ξεκινώντας το πρώτο μέρος της πτυχιακής μας, έχουμε σαν στόχο να κατανοήσουμε την εσωτερική δομή του board και πως αυτό, μπορεί να στείλει τα δεδομένα στις εξόδους του (οι έξοδοι που μας ενδιαφέρουν, είναι οι τρείς expansion connectors).

i. Ανάλυση του προβλήματος

Ποιό είναι το ζητούμενό μας; Αρχικά θα συνδέσουμε τα δύο boards έτσι ώστε με τα switches του ενός board, να ενεργοποιούμε τα leds του άλλου board. Για να το πετύχουμε εργαζόμαστε ως εξής. Χρησιμοποιούμε τα A2 και B1 Expansion Connectors (34 I/O user pins) για την σύνδεση μεταξύ των δύο boards. Τα ελεύθερα pins φαίνονται στους πίνακες των σελίδων 25 και 26.

Αρχικά, είναι σημαντικό να αναφέρουμε τι θα χρησιμοποιήσουμε: δύο (2) board Spartan - 3 της Xilinx, οκτώ (8) switches ανά board, οκτώ (8) leds ανά board και δύο (2) expansion connectors από το κάθε board. Η σχηματική σύνδεση του παραπάνω κυκλώματος, φαίνεται στην σελίδα 32. Παρακάτω αναφέρουμε αναλυτικότερα τα βήματά μας.

ii. Σχεδιασμός υλοποίησης

Ξεκινώντας την διαδικασία, αρχίζουμε δημιουργώντας ένα νέο project στο πρόγραμμα **ISE Design Suite 14.2** (βλέπε παράδειγμα ενός project στις σελίδες 15-19). Όταν ολοκληρωθεί η δημιουργία του project, μας εμφανίζεται η εικόνα 13. Σε αυτό το σημείο θα γράψουμε τον VHDL κώδικά μας. Ο κώδικας σε αυτό το σημείο, θα περιέχει μεταβλητές εισόδων και εξόδων στην ενότητα entity, και μια ενότητα τύπου architecture, η οποία θα κάνει το κύριο μέρος της δουλειάς μας, δηλαδή θα αντιστοιχεί τις εισόδους και τις εξόδους. Βάσει της παραπάνω αρχιτεκτονικής, θα φτιάξουμε την δική μας ενότητα architecture. Όπως προαναφέραμε, θα χρησιμοποιήσουμε leds, switches, expansion connectors και εσωτερικές μεταβλητές. Πως θα συνδεθούν αυτά μεταξύ τους έτσι ώστε να πετύχουμε το αποτέλεσμα μας;

Παρακάτω εξηγούμε πως πετυχαίνουμε αυτή την σύνδεση και επισυνάπτουμε τον κώδικα που χρησιμοποιήσαμε (VHDL και ucf κώδικες).

- Στο VHDL module και συγκεκριμένα στο architecture, θα χρησιμοποιήσουμε τύπους δεδομένων **std logic** έτσι ώστε να μπορούμε να εργαζόμαστε σε επίπεδο κώδικα.
- Στο UCF file, θα συνδέσουμε στις μεταβλητές του VHDL module, θα τα connectors των expansion connectors.

Ο κώδικας VHDL που δημιουργήσαμε:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity board_1 is
5  port(
6  --dilwseis gia exodo sto board_2 me swiches apo to board_1
7  sw : in std_logic_vector(7 downto 0);
8  output : out std_logic_vector (7 downto 0);
9  --dilwseis gia tin eisodo apo to board_2 kai provoli sto board_1
10 input : in std_logic_vector(7 downto 0);
11 led : out std_logic_vector(7 downto 0)
12 );
13 end board_1;
14
15 architecture Behavioral of board_1 is
16 begin
17 p1: process(sw)
18 begin
19 --kwdikas gia apostoli dedomenwn sto board_2
20 for i in 0 to 7 loop
21    if sw(i)='1' then
22       output(i)<='0';
23    else
24       output(i)<='1';
25    end if;
26 end loop;
27
28 --kwdikas gia apodoxi dedomenwn apo to board_2
29 for j in 0 to 7 loop
30    led(j)<=not input(j);
31 end loop;
32 end process;
33
34 end Behavioral;
```

Εικόνα 15 - Ο VHDL κώδικας για αποστολή και αποδοχή δεδομένων

Γιατί χρησιμοποιήσαμε τέσσερις μεταβλητές (sw, output, input, led), ενώ οι μεταβλητές που μας ενδιαφέρουν είναι δύο (sw, led); Αυτό γίνεται επειδή δεν μπορούμε να αντιστοιχήσουμε μια μεταβλητή σε δύο pins του FPGA, δηλαδή δεν μπορούμε πούμε ότι το sw (0) αντιστοιχεί στο pin F12 και στο pin του connector 22 (C16). Για τεχνικούς λόγους δεν είναι αποδεκτό, οπότε αυτό που κάνουμε είναι το εξής: συνδέουμε τις μεταβλητές sw με τα pins του FPGA που θέλουμε και έπειτα εκχωρούμε στις μεταβλητές output το περιεχόμενο των μεταβλητών sw. Τέλος, συνδέουμε τις μεταβλητές output με τα pins των expansion connectors.

Το περιεχόμενο του αρχείου UCF file:

```
// sindesi twn metavlitwn me ta swiches
NET "sw(0)" LOC = "F12";
NET "sw(1)" LOC = "G12";
NET "sw(2)" LOC = "H14";
NET "sw(3)" LOC = "H13";
NET "sw(4)" LOC = "J14";
NET "sw(5)" LOC = "J13";
NET "sw(6)" LOC = "K14";
NET "sw(7)" LOC = "K13";
// apostoli dedomenwn stin exodo B1 Expansion
// Connector tou board_1 (gia metafora sto board_2)
NET "output(0)" LOC = "C16"; //22
NET "output(1)" LOC = "D15"; //23
NET "output(2)" LOC = "D16"; //24
NET "output(3)" LOC = "E15"; //25
NET "output(4)" LOC = "E16"; //26
NET "output(5)" LOC = "F15"; //27
NET "output(6)" LOC = "G15"; //28
NET "output(7)" LOC = "G16"; //29
// apodoxi dedomenwn stin eisodo A2 E.C. tou board_1
// (dedomena pou esteile to board_2)
NET "input(0)" LOC = "D5"; //5
NET "input(1)" LOC = "C5"; //6
NET "input(2)" LOC = "D6"; //7
NET "input(3)" LOC = "C6"; //8
NET "input(4)" LOC = "E7"; //9
NET "input(5)" LOC = "C7"; //10
NET "input(6)" LOC = "D7"; //11
NET "input(7)" LOC = "C8"; //12
// sindesi twn leds me ta dedomena apodoxis
NET "led(0)" LOC = "K12";
NET "led(1)" LOC = "P14";
NET "led(2)" LOC = "L12";
NET "led(3)" LOC = "N14";
NET "led(4)" LOC = "P13";
NET "led(5)" LOC = "N12";
NET "led(6)" LOC = "P12";
NET "led(7)" LOC = "P11";
```

Εικόνα 16 - Το ucf file

Στην εικόνα 16, βλέπουμε αυτό που αναφέραμε στην προηγούμενη σελίδα. Παρόλο που στην ουσία το ζευγάρι των μεταβλητών sw και output, όπως και το ζευγάρι των μεταβλητές input και led, έχουν το ίδιο περιεχόμενο, τις στέλνουμε σε διαφορετικά pins αφού δεν μπορούμε να στείλουμε μια μεταβλητή σε δύο pins. Αν μια μεταβλητή σταλθεί μέσω του ucf file σε δύο pins, ο κώδικας δεν περνάει από το compile και βγάζει error. Παρακάτω (σελίδα 27), αναφέρουμε πιο αναλυτικά τον τρόπο εργασίας μας.

Πίνακες για τα expansion connectors

Εδώ βλέπουμε, πόσα pins είναι ελεύθερα σε κάθε connector.

Connector	User I/O	SRAM	JTAG	Serial Configuration	Parallel Configuration
A1	32	Address OE#, WE# Data[7:0] to IC10 only	√		
A2	34			√	
B1	34			√	√

Εικόνα 17 - Πληροφορίες για τους τρείς expansion connectors

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(N8)	ADR0
DB0	(N7) SRAM IC10 IO0	5	6	(L5) SRAM A0	ADR1
DB1	(T8) SRAM IC10 IO1	7	8	(N3) SRAM A1	ADR2
DB2	(R6) SRAM IC10 IO2	9	10	(M4) SRAM A2	ADR3
DB3	(T5) SRAM IC10 IO3	11	12	(M3) SRAM A3	ADR4
DB4	(R5) SRAM IC10 IO4	13	14	(L4) SRAM A4	ADR5
DB5	(C2) SRAM IC10 IO5	15	16	(G3) SRAM WE#	WE
DB6	(C1) SRAM IC10 IO6	17	18	(K4) SRAM OE#	OE
DB7	(B1) SRAM IC10 IO7	19	20	(P9) FPGA DOUT/BUSY	CSA
LSBCLK	(M7)	21	22	(M10)	MA1-DB0
MA1-DB1	(F3) SRAM A6	23	24	(G4) SRAM A5	MA1-DB2
MA1-DB3	(E3) SRAM A8	25	26	(F4) SRAM A7	MA1-DB4
MA1-DB5	(G5) SRAM A10	27	28	(E4) SRAM A9	MA1-DB6
MA1-DB7	(H4) SRAM A12	29	30	(H3) SRAM A11	MA1-ASTB
MA1-DSTB	(J3) SRAM A14	31	32	(J4) SRAM A13	MA1-WRITE
MA1-WAIT	(K5) SRAM A16	33	34	(K3) SRAM A15	MA1-RESET
MA1-INT	(L3) SRAM A17	35	36	JTAG Isolation	JTAG Isolation
TMS	(C13) FPGA JTAG TMS	37	38	(C14) FPGA JTAG TCK	TCK
TDO-ROM	Platform Flash JTAG TDO	39	40	Header J7, pin 3	TDO-A

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3
PA-IO4	(D6)	7	8	(C6)	PA-IO5
PA-IO6	(E7)	9	10	(C7)	PA-IO7
PA-IO8	(D7)	11	12	(C8)	PA-IO9
PA-IO10	(D8)	13	14	(C9)	PA-IO11
PA-IO12	(D10)	15	16	(A3)	PA-IO13
PA-IO14	(B4)	17	18	(A4)	PA-IO15
PA-IO16	(B5)	19	20	(A5)	PA-IO17
PA-IO18	(B6)	21	22	(B7)	MA2-DB0
MA2-DB1	(A7)	23	24	(B8)	MA2-DB2
MA2-DB3	(A8)	25	26	(A9)	MA2-DB4
MA2-DB5	(B10)	27	28	(A10)	MA2-DB6
MA2-DB7	(B11)	29	30	(B12)	MA2-ASTB
MA2-DSTB	(A12)	31	32	(B13)	MA2-WRITE
MA2-WAIT	(A13)	33	34	(B14)	MA2-RESET
MA2-INT/GCK4	(D9) Oscillator socket	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

Εικόνα 18 - A1 (αριστερά) και A2 (δεξιά) Expansion connectors

Ο κύριος σκοπός των expansion connectors, είναι η σύνδεση με άλλες συσκευές και η μεταφορά των δικών μας δεδομένων προς αυτές (ή και η αποδοχή δεδομένων από τις συσκευές αυτές). Στην περίπτωση μας, η άλλη συσκευή που θα συνδεθεί θα είναι και αυτή ένα board με FPGA. Μεταφορά δεδομένων μπορούμε να πετύχουμε μέσω κατάλληλου κώδικα, και από άλλες θύρες, όπως η θύρα PS/2 (πληκτρολόγιο ή ποντίκι, εικόνα 1, αριθμός 9), η θύρα VGA (έξοδος σε οθόνη, εικόνα 1, αριθμός 5) και η θύρα RS-232 (μεταφορά δεδομένων από και προς τον υπολογιστή, εικόνα 1, αριθμός 6).

FPGA Pin	Connector		FPGA Pin
	1	2	
V_{CCO} (all banks)	3	4	(N8)
(N7) SRAM IC10 IO0	5	6	(L5) SRAM A0
(T8) SRAM IC10 IO1	7	8	(N3) SRAM A1
(R6) SRAM IC10 IO2	9	10	(M4) SRAM A2
(T5) SRAM IC10 IO3	11	12	(M3) SRAM A3
(R5) SRAM IC10 IO4	13	14	(L4) SRAM A4
(C2) SRAM IC10 IO5	15	16	(G3) SRAM WE#
(C1) SRAM IC10 IO6	17	18	(K4) SRAM OE#
(B1) SRAM IC10 IO7	19	20	(P9) FPGA DOUT/BUSY
(M7)	21	22	(M10)
(F3) SRAM A6	23	24	(G4) SRAM A5
(E3) SRAM A8	25	26	(F4) SRAM A7
(G5) SRAM A10	27	28	(E4) SRAM A9
(H4) SRAM A12	29	30	(H3) SRAM A11
(J3) SRAM A14	31	32	(J4) SRAM A13
(K5) SRAM A16	33	34	(K3) SRAM A15
(L3) SRAM A17	35	36	JTAG Isolation
(C13) FPGA JTAG TMS	37	38	(C14) FPGA JTAG TCK
Platform Flash JTAG TDO	39	40	Header J7, pin 3

Εικόνα 19 - B1 Expansion connector

Σημειώσεις για τα expansion connectors: Στο **A1 expansion connector** δεν χρησιμοποιούνται από τον προγραμματιστή, οι connectors **1** (GRD), **2** (V), **20** (το οποίο είναι σήμα διαμόρφωσης και εναλλάσσεται κατά την διάρκεια ενός process), **36**, **37**, **38**, **39**, **40** (οι connectors 36-40 χρησιμοποιούνται για το JTAG).
Στο **A2 expansion connector** δεν χρησιμοποιούνται οι connectors **1, 2, 36, 37, 38, 39** (τα pins 36-39 περιλαμβάνουν σήματα που ρυθμίζουν το FPGA σε master ή slave σειριακή κατάσταση).
Στο **B1 expansion connector** δεν χρησιμοποιούνται οι connectors **1, 2, 36, 37, 38, 39** (τα pins 36-39 περιλαμβάνουν σήματα που ρυθμίζουν το FPGA σε master ή slave σειριακή κατάσταση. Για το **B1** αυτοί οι connectors και επιπρόσθετα **οι 5, 7, 9, 11, 13, 15, 17, 19, 20** παρέχουν σήματα τα οποία ρυθμίζουν το FPGA και σε master ή slave παράλληλη κατάσταση).
Όσοι connectors έχουν αναφερθεί παραπάνω, είναι προτιμότερο να μην χρησιμοποιούνται γενικά, αλλά υπάρχει η δυνατότητα να χρησιμοποιηθούν από τον χρήστη, σε περιπτώσεις που τα pins, δεν δεσμεύονται από το ίδιο το FPGA (όπως για παράδειγμα κατά την χρήση του JTAG, της SRAM ή σε καταστάσεις master - slave). Οι δύο connectors που δεν πρέπει ποτέ να συνδέονται είναι οι **1** και **2** σε κάθε expansion connector.

iii. Πειραματικό μέρος - Δουλεύοντας στο πρώτο board

Στο πρώτο board θα χρησιμοποιήσουμε αρχικά τα switches 0 έως 7. Στις παρακάτω εικόνες, θα δούμε πως μπορούμε να στείλουμε τα switches στην έξοδο του **B1 Expansion Connector**. Στην πραγματικότητα, δεν συνδέουμε κατευθείαν τα switches με τα expansion connectors, μιας και κάτι τέτοιο δεν είναι τεχνικά επιτρεπτό. Για να το πετύχουμε αυτό, σε αυτό το σημείο θα χρησιμοποιήσουμε το **ISE Design Suite 14.2.** Δημιουργούμε ένα νέο πρότζεκτ σύμφωνα με όσα αναφέραμε στο κεφάλαιο 2.2 και μέσω του κατάλληλου κώδικα (βλέπε παρακάτω), θα πετύχουμε την μεταφορά των bits στα expansion connectors. Θα χρειαστούμε δύο ειδών αρχεία στο project μας, ένα αρχείο VHDL module και έπειτα σε αυτό το αρχείο θα προσθέσουμε ένα αρχείο ucf file. Πρώτα συνδέουμε τα switches - διακόπτες (εικόνα 1, αριθμός 11) με μεταβλητές **in std_logic** (στο ucf) έτσι ώστε να αποθηκεύονται οι τιμές που έχουμε από το κάθε switch. Έπειτα αυτές τις τιμές, τις στέλνουμε σε νέες μεταβλητές **out std_logic** (στο VHDL module) και αυτές με την σειρά τους, συνδέονται (μέσω ucf) στους expansion connectors. Η διαδικασία αυτή γίνεται επειδή δεν μπορούμε να αντιστοιχήσουμε τα FPGA pins με τα expansion connector pins, καθώς κάτι τέτοιο είναι τεχνικά μη αποδεκτό. Για παράδειγμα, αν έχουμε το "F12" pin, που είναι το πρώτο switch, η εντολή **NET "F12" LOC = "C16";** δεν είναι αποδεκτή, δημιουργεί error και ο κώδικας δεν κάνει compile. Το σωστό είναι το εξής:

```
NET "sw(0)" LOC = "F12";          // στο ucf file
output(0) <=sw(0);                // στο vhdl file που έχουμε ως top module
NET "output(0)" LOC = "C16";      // στο ucf file
```

Οι τρείς παραπάνω γραμμές είναι χαρακτηριστικό παράδειγμα για το πως στέλνουμε τα δεδομένα μας στις εξόδους των expansion connectors.

```
15  architecture Behavioral of board_1 is
16  begin
17  p1: process(sw)
18  begin
19  --kwdikas gia apostoli dedomenwn sto board_2
20  for i in 0 to 7 loop
21     if sw(i)='1' then
22        output(i)<='0';
23     else
24        output(i)<='1';
25     end if;
26  end loop;
```

Εικόνα 20 - Το κομμάτι του VHDL κώδικα που χρειαζόμαστε

```
1  // sindesi twn metavlitwn me ta swiches
2  NET "sw(0)" LOC = "F12";
3  NET "sw(1)" LOC = "G12";
4  NET "sw(2)" LOC = "H14";
5  NET "sw(3)" LOC = "H13";
6  NET "sw(4)" LOC = "J14";
7  NET "sw(5)" LOC = "J13";
8  NET "sw(6)" LOC = "K14";
9  NET "sw(7)" LOC = "K13";
10 // apostoli dedomenwn stin exodo B1 Expansion
11 // Connector tou board_1 (gia metafora sto board_2)
12 NET "output(0)" LOC = "C16"; // connector 22
13 NET "output(1)" LOC = "D15"; // connector 23
14 NET "output(2)" LOC = "D16"; // connector 24
15 NET "output(3)" LOC = "E15"; // connector 25
16 NET "output(4)" LOC = "E16"; // connector 26
17 NET "output(5)" LOC = "F15"; // connector 27
18 NET "output(6)" LOC = "G15"; // connector 28
19 NET "output(7)" LOC = "G16"; // connector 29
```

Εικόνα 21 - Σύνδεση των switches με το expansion connector

Έπειτα θα συνδέσουμε την μεταβλητή led (0 έως 7) με τους connectors του **A2 Expansion Connector**, για να δέχονται σήματα από το δεύτερο board. Για την αποδοχή δεδομένων εργαζόμαστε ως εξής:

```
NET "ld(0)" LOC = "K12";        // στο ucf file
input(0) <=ld(0);               // στο vhdl file που έχουμε ως top module
NET "input(0)" LOC = "D5";      // στο ucf file
```

Οι παρακάτω δύο εικόνες δείχνουν στην πράξη αυτό που μόλις αναφέραμε:

```
28 --kwdikas gia apodoxi dedomenwn apo to board_2
29 for j in 0 to 7 loop
30    led(j)<=not input(j);
31 end loop;
32 end process;
33
34 end Behavioral;
```

Εικόνα 22 - Αποδοχή δεδομένων

Σε αντίθεση με την εικόνα 20, δεν χρειάζεται κώδικας επιλογής (if - else), καθώς τα δεδομένα έρχονται έχοντας συγκεκριμένη τιμή. Στη εικόνα 20, χρησιμοποιούμε το if - else, επειδή τα δεδομένα δεν έχουν συγκεκριμένη τιμή αλλά μπορούν να αλλάξουν ανά πάσα στιγμή. Τέλος το **not** στην γραμμή 30 στην εικόνα 22, χρησιμεύει για να λύσουμε το θέμα της αντιστροφής λογικής που παρατηρήσαμε και εξηγούμε αναλυτικότερα στην επόμενη σελίδα.

```
20  // apodoxi dedomenwn stin eisodo A2 E.C. tou board_1
21  // (dedomena pou esteile to board_2)
22  NET "input(0)" LOC = "D5"; // connector 5
23  NET "input(1)" LOC = "C5"; // connector 6
24  NET "input(2)" LOC = "D6"; // connector 7
25  NET "input(3)" LOC = "C6"; // connector 8
26  NET "input(4)" LOC = "E7"; // connector 9
27  NET "input(5)" LOC = "C7"; // connector 10
28  NET "input(6)" LOC = "D7"; // connector 11
29  NET "input(7)" LOC = "C8"; // connector 12
30  // sindesi twn leds me ta dedomena apodoxis
31  NET "led(0)" LOC = "K12";
32  NET "led(1)" LOC = "P14";
33  NET "led(2)" LOC = "L12";
34  NET "led(3)" LOC = "N14";
35  NET "led(4)" LOC = "P13";
36  NET "led(5)" LOC = "N12";
37  NET "led(6)" LOC = "P12";
38  NET "led(7)" LOC = "P11";
```

Εικόνα 23 - Σύνδεση στο ucf file

Σημείωση: Όταν συνδέσαμε τα δύο boards μεταξύ τους, παρατηρήσαμε ότι στα expansion connectors, επικρατεί η αντίστροφη λογική. Δηλαδή όταν από το board 1 στέλναμε λογικό άσσο, στο board 2 θα ερχόταν λογικό μηδέν. Οπότε πριν στείλουμε κάθε δεδομένο – bit έπρεπε να του αντιστρέψουμε την κατάσταση. Αυτό γινόταν με κώδικα της μορφής:

```
if sw(0) = '1' then
  output(0) <='0';
else
  output(0) <='1';
end if;
```

(Βλέπε εικόνα 20, γραμμές 21 έως 26).

Ο κώδικας if - else, χρησιμοποιείται καθώς οι είσοδοι sw 0 έως 7, δεν έχουν σταθερή τιμή, αλλά αλλάζουν από τον χρήστη. Στο κομμάτι του κώδικα που ασχολούμαστε με την αποδοχή των δεδομένων, τα δεδομένα έχουν συγκεκριμένη τιμή και δεν αλλάζουν στο board αποδοχής, αλλά στο board αποστολής. Οπότε δεν χρειαζόμαστε if - else, στο συγκεκριμένο σημείο, αλλά χρειαζόμαστε το **not** (εικόνα 22) ώστε να αλλάξουμε αντιστρέψουμε τα εισερχόμενα δεδομένα που έχουν αντιστραφεί κατά την είσοδο. Ο κώδικας που χρειαζόμαστε φαίνεται και παρακάτω:

```
for i in 0 to 7 loop
  led(j) <= not input(j);
end loop;
```

iv. Σύνδεση μεταξύ των boards

Στους παρακάτω δύο πίνακες, φαίνεται ο τρόπος με τον οποίο επικοινωνούν τα δύο boards σύμφωνα με την παρακάτω συνδεσμολογία:

Board (1) / B1 E.C.	->	Board (2) / A2 E.C.
Connector 22	->	Connector 13
Connector 23	->	Connector 14
Connector 24	->	Connector 15
Connector 25	->	Connector 16
Connector 26	->	Connector 17
Connector 27	->	Connector 18
Connector 28	->	Connector 19
Connector 29	->	Connector 20

Πίνακας 1 - Σύνδεση των E.C. μεταξύ τους

Board (1) / A2 E.C.	->	Board (2) / B1 E.C.
Connector 5	->	Connector 6
Connector 6	->	Connector 8
Connector 7	->	Connector 10
Connector 8	->	Connector 12
Connector 9	->	Connector 14
Connector 10	->	Connector 16
Connector 11	->	Connector 18
Connector 12	->	Connector 21

Πίνακας 2 - Σύνδεση των E.C. μεταξύ τους

Η σύνδεση μεταξύ των expansion connectors, μπορεί να γίνει είτε με απλά καλώδια, είτε με καλωδιοταινία. Η χρήση της καλωδιοταινίας, είναι προτιμότερη καθώς είναι δυσκολότερο και περισσότερο χρονοβόρο να δουλεύεις με μεμονωμένα καλώδια. Τα μεμονωμένα καλώδια είναι καλό να προτιμηθούν σε περιπτώσεις που χρειαζόμαστε λίγους connectors από τους expansion connectors.

* Το E.C. που βλέπουμε στους δύο παραπάνω πίνακες είναι συντομογραφία για τους Expansion Connectors.

Σημείωση: Προηγουμένως χρησιμοποιήσαμε not για να αποφύγουμε την αντίστροφη λογική. Εναλλακτικά, αντί να χρησιμοποιήσουμε το **not**, μπορούμε να το παραλείψουμε και να χρησιμοποιήσουμε μεταβλητές τύπους **inout**, αντί για **in** και **out**.

v. Πειραματικό μέρος - Δουλεύοντας στο δεύτερο board

Στο δεύτερο board, θα πάρουμε από τους ακροδέκτες του **A2 Expansion Connector**, τις εισόδους και θα τις στείλουμε στα leds (φαίνονται τα pins στην εικόνα 24). Έπειτα, θα συνδέσουμε τους ακροδέκτες του **B1 Expansion Connector** στα switches.

```
20  // apodoxi dedomenwn stin eisodo A2 E.C.
21  // tou board_2 (dedomena pou esteile to board_1)
22  NET "input(0)" LOC = "D8"; // connector 13
23  NET "input(1)" LOC = "C9"; // connector 14
24  NET "input(2)" LOC = "D10";// connector 15
25  NET "input(3)" LOC = "A3"; // connector 16
26  NET "input(4)" LOC = "B4"; // connector 17
27  NET "input(5)" LOC = "A4"; // connector 18
28  NET "input(6)" LOC = "B5"; // connector 19
29  NET "input(7)" LOC = "A5"; // connector 20
30  // sindesi twn leds me ta dedomena apodoxis
31  NET "led(0)" LOC = "K12";
32  NET "led(1)" LOC = "P14";
33  NET "led(2)" LOC = "L12";
34  NET "led(3)" LOC = "N14";
35  NET "led(4)" LOC = "P13";
36  NET "led(5)" LOC = "N12";
37  NET "led(6)" LOC = "P12";
38  NET "led(7)" LOC = "P11";
```

Εικόνα 24 - Σύνδεση των εισερχόμενων δεδομένων με τα leds

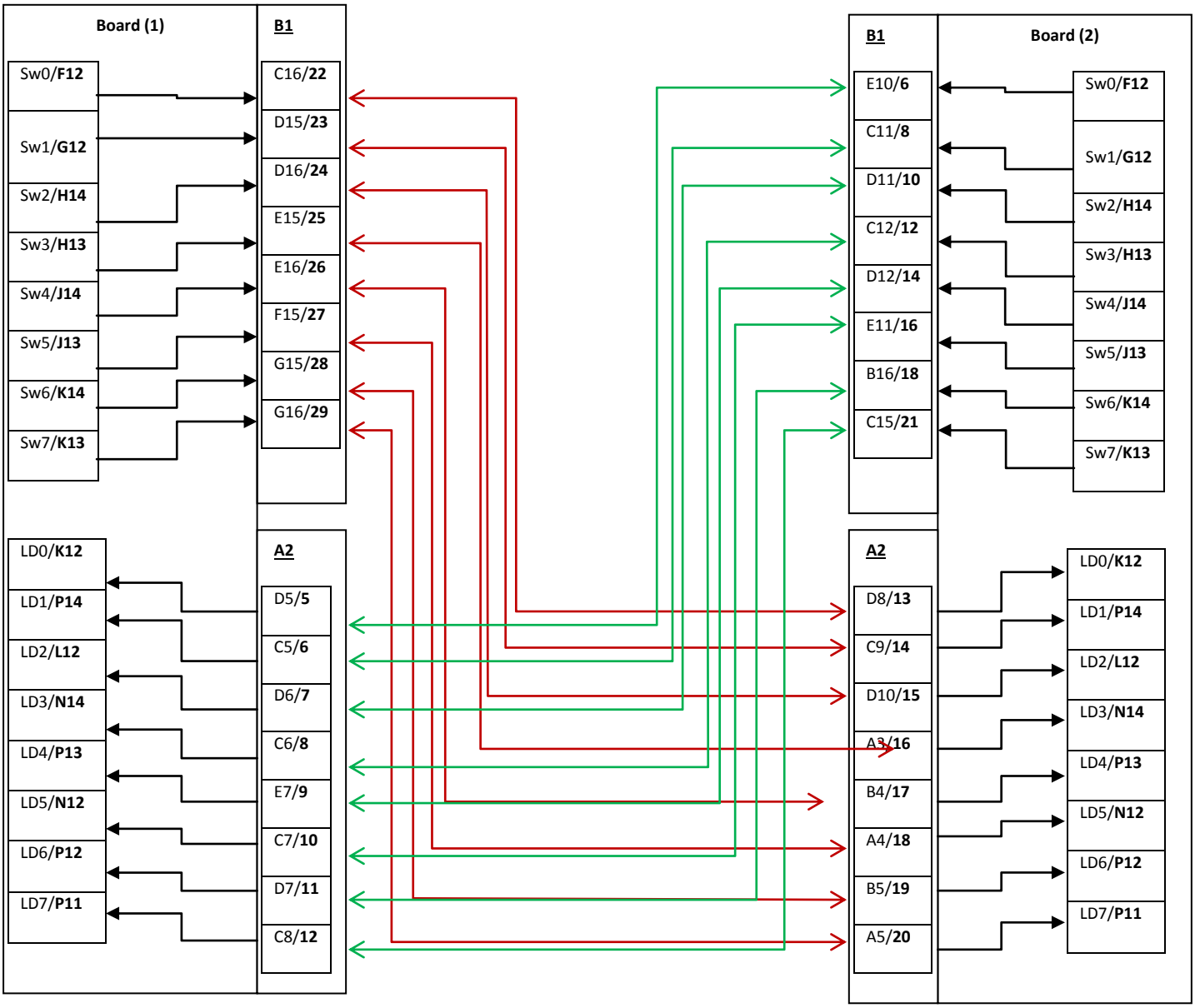
Τέλος συνδέουμε και τα switches με τους ακροδέκτες.

```
1   // sindesi twn metavlitwn me ta swiches
2   NET "sw(0)" LOC = "F12";
3   NET "sw(1)" LOC = "G12";
4   NET "sw(2)" LOC = "H14";
5   NET "sw(3)" LOC = "H13";
6   NET "sw(4)" LOC = "J14";
7   NET "sw(5)" LOC = "J13";
8   NET "sw(6)" LOC = "K14";
9   NET "sw(7)" LOC = "K13";
10  // apostoli dedomenwn stin exodo B1 E.C.
11  // tou board_2 (gia metafora sto board_1)
12  NET "output(0)" LOC = "E10"; // connector 6
13  NET "output(1)" LOC = "C11"; // connector 8
14  NET "output(2)" LOC = "D11"; // connector 10
15  NET "output(3)" LOC = "C12"; // connector 12
16  NET "output(4)" LOC = "D12"; // connector 14
17  NET "output(5)" LOC = "E11"; // connector 16
18  NET "output(6)" LOC = "B16"; // connector 18
19  NET "output(7)" LOC = "C15"; // connector 21
```

Εικόνα 25 - Σύνδεση των switches με τα E.C.

Σημείωση: Ο VHDL κώδικας για το δεύτερο board είναι ακριβώς ο ίδιος με το πρώτο board (βλέπε εικόνα 15) αφού κάνουν τα ίδια πράγματα. Το μόνο που αλλάξαμε είναι το ucf file, στο οποίο απλά χρησιμοποιήσαμε διαφορετικούς connectors.

vi. Σχηματική σύνδεση μεταξύ των δύο boards



Σημαντικό: Στο παραπάνω σχήμα, βλέπουμε μια απεικόνιση η οποία προσεγγίζει όσο αυτό είναι δυνατόν την πραγματική υλοποίηση. Τα πράσινα και κόκκινα βελάκια, αντιστοιχούν σε απλά καλώδια ή καλωδιοταινίες (ανάλογα τι επιλέγουμε να χρησιμοποιήσουμε). Τα μαύρα βελάκια (εντός των board), αντιστοιχούν στον κώδικα που υλοποιήθηκε σε γλώσσα VHDL (εικόνα 15).

4.1.2 Σύνδεση των δύο Spartan-3 για το Seven Segment Display

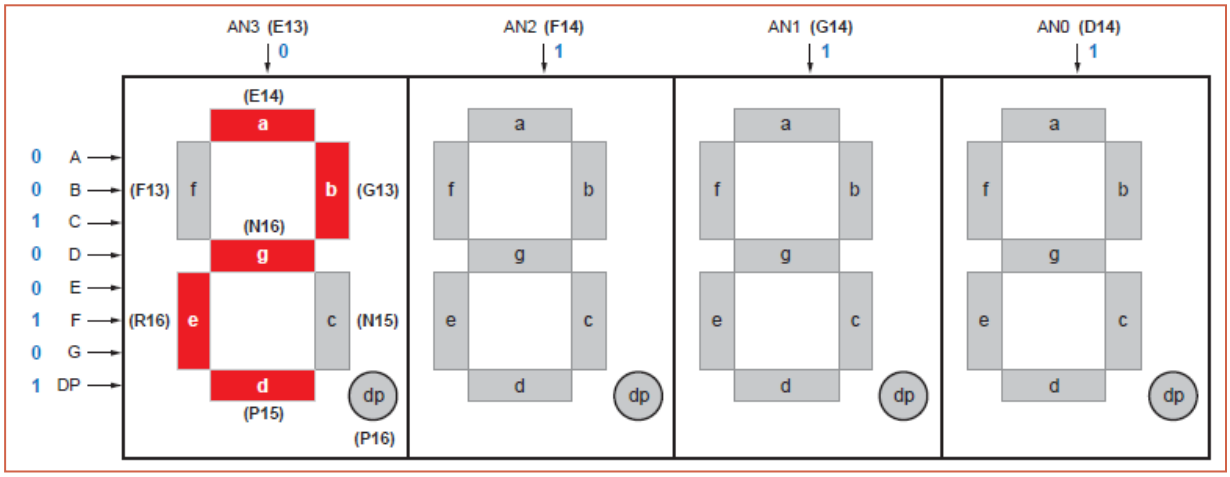
Στην ενότητα 4.1.1, είδαμε πως μπορούμε με τους διακόπτες - switches του ενός board να ενεργοποιήσουμε τα λαμπάκια - leds του άλλου board. Στην ενότητα 4.1.2, θα δούμε κάτι παρόμοιο, αλλά αυτή την φορά δεν θα ενεργοποιούμε τα leds, αλλά ένα άλλο μέρος του board, το Seven Segment Display (εικόνα 1, αριθμός 10). Επίσης δεν θα χρησιμοποιήσουμε switches, αλλά με τον προγραμματισμό καθενός board, το άλλο board θα μετράει από το 1 έως το F (στο δεκαεξαδικό σύστημα).

i. Ανάλυση του προβλήματος

Ποιό είναι το ζητούμενο μας; Εφόσον συνδέσουμε τα δύο boards, θα προγραμματίζουμε το πρώτο board, θα τρέχει στο FPGA του board ο κώδικάς μας, αλλά τα αποτελέσματα του compile θα φαίνονται στο seven segment display (εικόνα 1, αριθμός 10) του δεύτερου board όπου θα εμφανίζεται ανά ένα δευτερόλεπτο οι αριθμοί από το 0 έως το F. Η μέτρηση θα σταματάει στο F (15 στο δεκαεξαδικό σύστημα) και έπειτα θα ξανά ξεκινάει από το μηδέν.

ii. Σχεδιασμός υλοποίησης

Χρησιμοποιούμε τα A1 και A2 Expansion Connector (32, 34 I/O user pins, βλέπε σελίδα 15-16), για την μεταφορά των δεδομένων. Τα ελεύθερα pins φαίνονται στους πίνακες της επόμενης σελίδας. Δεν θα χρησιμοποιήσουμε εξωτερικά buttons ή switches πλην ενός push button το οποίο θα κάνει reset το board στο οποίο πατήθηκε. Με το πάτημα του reset, στο άλλο board η μέτρηση θα ξεκινάει ξανά από το μηδέν.



Εικόνα 26 - Το Seven Segment Display

Ας δώσουμε ένα παράδειγμα για να καταλάβουμε καλύτερα την παραπάνω εικόνα. Αν θελήσουμε να εργαστούμε στο πρώτο από τα 4 στοιχεία, θα πρέπει να ενεργοποιήσουμε το FPGA pin E13. Για να εμφανιστεί ο αριθμός 2, θα πρέπει να ενεργοποιήσουμε τα leds E14 (a), G13 (b), N16 (g), R16 (e), P15 (d).

Ακολουθήσαμε την ίδια λογική με την ενότητα 4.1.1, δηλαδή γίνεται η επεξεργασία στο ένα board και στέλνονται τα αποτελέσματα μέσω των εξόδων A1, A2 στο άλλο board. Έπειτα το άλλο board δέχεται τις εισόδους, τις τοποθετεί σε μεταβλητές τύπου in **std_logic** και μέσα στην **process,** τις στέλνει σε εξωτερικές μεταβλητές **out std_logic** οι οποίες με την σειρά τους συνδέονται στο **ucf file** με το SSD (Seven Segment Display).

Σημείωση: Όταν συνδέσαμε τα δύο boards μεταξύ τους, παρατηρήσαμε ότι στα expansion connectors, επικρατεί η αντίστροφη λογική. Δηλαδή όταν από το board 1 στέλναμε άσσο, στο board 2 θα ερχόταν μηδενικό. Σαν αποτέλεσμα είχαμε να ενεργοποιούνται στο SSD ακριβώς τα αντίθετα leds από αυτά που θέλαμε. Οπότε πριν στείλουμε κάθε δεδομένο – bit έπρεπε να του αντιστρέψουμε την κατάσταση. Για την πετύχουμε αυτή την αλλαγή, ενώ στο ίδιο board ο κώδικας του αρχείου bcd2seg θα ήταν αυτός της εικόνας 27, εμείς μετατρέψαμε τους λογικούς άσσους, σε λογικά μηδέν και το αντίστροφο (εικόνες 28). Στο αρχείο bcd2seg (εξηγούμε παρακάτω τι κάνει το συγκεκριμένο αρχείο), αντιστρέψαμε κάθε bit που αντιστοιχούσε σε κάθε μία έξοδο.

```
22
23  type slv_array is array(0 to 15) of std_logic_vector(6 downto 0);
24  constant conversionTable : slv_array :=
25  ( -- ABCDEFG
26
27  "1111110", -- 0
28  "0110000", -- 1
29  "1101101", -- 2
30  "1111001", -- 3
31  "0110011", -- 4
32  "1011011", -- 5
33  "1011111", -- 6
34  "1110000", -- 7
35  "1111111", -- 8
36  "1111011", -- 9
37  "1110111", -- A
38  "0011111", -- B
39  "1001110", -- C
40  "0111101", -- D
41  "1001111", -- E
42  "1000111" -- F
43  );
44
```

Εικόνα 27- Ο VHDL κώδικας που θα χρησιμοποιούσαμε αν δουλεύαμε με 1 board

Σημείωση: βλέπουμε ότι ο αριθμός 0 (μηδέν) γράφεται ως **1111110**. Η αρίθμηση είναι αντίστοιχη της σειράς του αγγλικού αλφάβητου, δηλαδή a=πρώτο bit, b=δεύτερο bit, c=τρίτο bit κ.ο.κ

Λόγω της ανάστροφης λογικής, ο παραπάνω κώδικας, αλλάζει ως εξής:

```
22
23  type slv_array is array(0 to 15) of std_logic_vector(6 downto 0);
24  constant conversionTable : slv_array :=
25  ( -- ABCDEFG
26  "0000001", -- 0
27  "1001111", --"0110000", -- 1
28  "0010010",--"1101101", -- 2
29  "0000110",--"1111001", -- 3
30  "1001100",--"0110011", -- 4
31  "0100100",--"1011011", -- 5
32  "0100000",--"1011111", -- 6
33  "0001111",--"1110000", -- 7
34  "0000000",--"1111111", -- 8
35  "0000100",--"1111011", -- 9
36  "0001000",--"1110111", -- A
37  "1100000",--"0011111", -- B
38  "0110001",--"1001110", -- C
39  "1000010",--"0111101", -- D
40  "0110000",--"1001111", -- E
41  "0111000"--"1000111" -- F
42  );
43
```

Εικόνα 28 - Ο κώδικας όπως διαμορφώθηκε λόγω της ανάστροφης λογικής

iii. Πειραματικό μέρος – Δουλεύοντας στα boards

Παρακάτω ακολουθεί το αρχείο toplevel το οποίο έχει οριστεί ως Top Module, δηλαδή κύριο αρχείο του project.

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity toplevel is
 7  port (
 8    clk : in std_logic;
 9    reset : in std_logic;
10    -- LED display
11    disp : out std_logic_vector(6 downto 0);
12    dp : out std_logic;
13    an : out std_logic_vector(3 downto 0);
14
15    disp_in : in std_logic_vector(6 downto 0);
16    dp_in : in std_logic;
17    an_in : in std_logic_vector(3 downto 0);
18
19    disp_out : out std_logic_vector(6 downto 0);
20    dp_out : out std_logic;
21    an_out : out std_logic_vector(3 downto 0)
22
23  );
24  end toplevel;
25
```

Εικόνα 29 - Ο VHDL κώδικας (μέρος α)

```
26  architecture structural of toplevel is
27
28    component bcd2seg
29    port (
30    i_bcd : in integer range 0 to 15;
31    o_7seg : out std_logic_vector(6 downto 0)
32    );
33    end component;
34
35    component counter
36    generic (
37    G_MAX : natural := 15; -- maximum range
38    G_INIT : natural := 15; -- initial value
39    G_UP : boolean := true -- count up?
40    );
41    port (
42    i_clk : in std_logic;
43    i_reset : in std_logic;
44    i_enable : in std_logic;
45    o_value : out natural range 0 to G_MAX
46    );
47  end component;
48
49    constant DELAY_INIT : integer := 50000000;
50    signal counterEnable : std_logic;
51    signal counterValue : integer range 0 to 15 := 0;
52    signal delay : integer range 0 to DELAY_INIT-1 := DELAY_INIT-1;
53    signal displayData : std_logic_vector(6 downto 0);
54
55  begin
56
57    delayCounter : counter
58    generic map (
59    G_MAX => DELAY_INIT-1,
60    G_INIT => DELAY_INIT-1,
61    G_UP => false
62    )
63    port map (
64    i_clk => clk,
65    i_reset => reset,
66    i_enable => '1',
67    o_value => delay
68    );
```

Εικόνα 30 - Ο VHDL κώδικας (μέρος β)

Οι εικόνες 29, 30, 31 αντιστοιχούν στον κώδικα του ενός εκ των τεσσάρων αρχείων που χρησιμοποιούμε συνολικά στο συγκεκριμένο project. Η ενότητα entity χρησιμεύει στην δήλωση των μεταβλητών. Η ενότητα architecture περιέχει τα components, τα port maps και την process που εκτελεί στην αποστολή των δεδομένων. Τα components χρησιμεύουν έτσι ώστε να χρησιμοποιούμε μεταβλητές από άλλα αρχεία του ίδιου βέβαια project. Οι τιμές των άλλων αρχείων, αλλάζουν σε πραγματικό χρόνο κατά την διάρκεια εκτέλεσης της προσομοίωσης.

```
70    displayCounter : counter
71    generic map (
72    G_MAX => 15,
73    G_INIT => 0,
74    G_UP => true
75    )
76    port map (
77    i_clk => clk,
78    i_reset => reset,
79    i_enable => counterEnable,
80    o_value => counterValue
81    );
82
83    encoder : bcd2seg
84    port map (
85    i_bcd => counterValue,
86    o_7seg => displayData
87    );
88
89    counterEnable <= '1' when delay = 0 else '0';
90    disp <= not displayData; -- the LED display is active-low driven
91    dp <= '1';
92    an <= "1111";
93
94  p1:process(dp_in)
95  begin
96  dp_out <= dp_in;
97  disp_out(6 downto 0) <=  disp_in(6 downto 0);
98  an_out(3 downto 0) <=  an_in(3 downto 0);
99  end process;
100
101 end structural;
```

Εικόνα 31 - Ο VHDL κώδικας (μέρος γ)

Εδώ (σειρές 94 έως 99) βλέπουμε μια απλή process η οποία το μόνο που κάνει είναι να αντιστοιχεί τις εισερχόμενες μεταβλητές από τα expansion connectors και να τα αντιστοιχεί σε εξερχόμενες μεταβλητές. Οι μεταβλητές αυτές (εισερχόμενες και εξερχόμενες), συνδέονται με τα δεδομένα μέσω του ucf file.

Ας δώσουμε ένα παράδειγμα για το πως θα κινηθεί μια μεταβλητή. Απο το board A, θα πάει στους expansion connector του board. Απο εκεί θα μεταφερθεί στους expansion connectors του board B. Τότε θα γίνει η σύνδεση μέσω του ucf και αυτά τα δεδομένα θα εμφανιστούν στο seven segment display του board B.

Οι υπόλοιπες μεταβλητές δεν αλλάζουν σε αυτό το αρχείο, αλλά στα αρχεία counter και bcd2seg. Παρακάτω αναφέρουμε αναλυτικότερα τι ακριβώς κάνουν αυτά τα δύο αρχεία. Οι μεταβλητές των αρχείων αυτών, περνάνε στο αρχείο toplevel μέσω των ενοτήτων των components.

```
1  //A2 - apostoli dedomenwn apo to board_1 stis exodous
2  NET "clk" LOC = "T9";
3  NET "reset" LOC ="F12";
4
5  NET "dp" LOC = "D5"; //5
6  NET "an(0)" LOC = "D6"; //7
7  NET "an(1)" LOC = "E7"; //9
8  NET "an(2)" LOC = "D7"; //11
9  NET "an(3)" LOC = "D8"; //13
10 NET "disp(6)" LOC = "D10"; //15
11 NET "disp(5)" LOC = "B4"; //17
12 NET "disp(4)" LOC = "B5"; //19
13 NET "disp(3)" LOC = "B6"; //21
14 NET "disp(2)" LOC = "A7"; //23
15 NET "disp(1)" LOC = "A8"; //25
16 NET "disp(0)" LOC = "B10"; //27
17 //A1 - apodoxi dedomenwn apo to board_2 stis eisodous
18 NET "dp_in" LOC = "N7"; //5
19 NET "an_in(0)" LOC = "T8";  //7
20 NET "an_in(1)" LOC = "R6"; //9
21 NET "an_in(2)" LOC = "T5"; //11
22 NET "an_in(3)" LOC = "R5"; //13
23 NET "disp_in(6)" LOC = "C2"; //15
24 NET "disp_in(5)" LOC = "C1"; //17
25 NET "disp_in(4)" LOC = "B1"; //19
26 NET "disp_in(3)" LOC = "M7"; //21
27 NET "disp_in(2)" LOC = "F3"; //23
28 NET "disp_in(1)" LOC = "E3"; //25
29 NET "disp_in(0)" LOC = "G5"; //27
30 //board - antistoixisi dedomenv eisodou sto SSD
31 NET "dp_out" LOC = "P16";
32 NET "an_out(0)" LOC = "D14";
33 NET "an_out(1)" LOC = "G14";
34 NET "an_out(2)" LOC = "F14";
35 NET "an_out(3)" LOC = "E13";
36 NET "disp_out(6)" LOC = "E14";
37 NET "disp_out(5)" LOC = "G13";
38 NET "disp_out(4)" LOC = "N15";
39 NET "disp_out(3)" LOC = "P15";
40 NET "disp_out(2)" LOC = "R16";
41 NET "disp_out(1)" LOC = "F13";
42 NET "disp_out(0)" LOC = "N16";
```

Εικόνα 32 - Ο ucf κώδικας

Οι γραμμές 5 έως 16 χρησιμεύουν στην αποστολή των δεδομένων από το FPGA, στις εξόδους των Expansion connectors. Οι γραμμές 18 έως 29 χρησιμεύουν στην αποδοχή των δεδομένων από τους expansion connectors. Η εμφάνιση των δεδομένων που έρχονται από τις γραμμές αυτές συνδέονται στον VDHL κώδικα και αντιστοιχίζονται στις γραμμές 31 έως 42.

Παρακάτω βλέπουμε το αρχείο counter

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity counter is
7  generic (
8  G_MAX : natural := 15; -- maximum range
9  G_INIT : natural := 15; -- initial value
10 G_UP : boolean := true -- count up?
11 );
12 port (
13 i_clk : in std_logic;
14 i_reset : in std_logic;
15 i_enable : in std_logic;
16 o_value : out natural range 0 to G_MAX
17 );
18 end counter;
19
20 architecture Behavioral of counter is
21   signal value : natural range 0 to G_MAX;
22
23 begin
24 -- check generics
25 assert G_MAX >= G_INIT report "G_INIT greater G_MAX"
26 severity FAILURE;
27 count : process(i_clk, i_reset)
28 begin
29 if i_reset = '1' then
30 value <= G_INIT;
31 elsif rising_edge(i_clk) and i_enable = '1' then
32 if G_UP then
33 value <= value + 1;
34 else
35 value <= value - 1;
36 end if;
37 end if;
38 end process;
39 o_value <= value;
40
41 end Behavioral;
```

Εικόνα 33 - Το αρχείο counter

Το αρχείο counter, αρχικά ελέγχει αν γίνεται reset από το push button . Αν γίνετε reset, παίρνει την αρχική τιμή του G_UNIT και ξεκινάει από το μηδέν. Αν δεν γίνεται reset, τότε στην πρώτη ακμή του ρολογιού, θα προσθέσει στην μεταβλητή value μια μονάδα. Η ακμή του ρολογιού δίνεται από τοπικό ρολόι που έχει πάνω το board. Τέλος η μεταβλητή o_value κρατά την τιμή ώστε να χρησιμεύει και στα υπόλοιπα συνδεδεμένα αρχεία του project.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5  entity bcd2seg is
6  port (
7    i_bcd : in integer range 0 to 15;
8    o_7seg : out std_logic_vector(6 downto 0)
9  );
10 end bcd2seg;
11 architecture behavioral of bcd2seg is
12 --    A
13 --  -----
14 -- F|   |B
15 --  | G |
16 --  -----
17 -- E|   |C
18 --  | D |
19 --  -----
20 type slv_array is array(0 to 15) of std_logic_vector(6 downto 0);
21 constant conversionTable : slv_array :=
22 ( -- ABCDEFG
23 "0000001", -- 0
24 "1001111", --"0110000", -- 1
25 "0010010",--"1101101", -- 2
26 "0000110",--"1111001", -- 3
27 "1001100",--"0110011", -- 4
28 "0100100",--"1011011", -- 5
29 "0100000",--"1011111", -- 6
30 "0001111",--"1110000", -- 7
31 "0000000",--"1111111", -- 8
32 "0000100",--"1111011", -- 9
33 "0001000",--"1110111", -- A
34 "1100000",--"0011111", -- B
35 "0110001",--"1001110", -- C
36 "1000010",--"0111101", -- D
37 "0110000",--"1001111", -- E
38 "0111000"--"1000111" -- F
39 );
40 begin
41 o_7seg <= conversionTable(i_bcd);
42 end Behavioral;
```

Εικόνα 34 - Το αρχείο bcd2seg

Το αρχείο bcd2seg, περιέχει έναν πίνακα 16 στοιχείων. Ο σκοπός είναι να εμφανίζει τις τιμές από 0 έως το F (δεκαεξαδικό σύστημα). Στις γραμμές 12 έως 19 βλέπουμε μια προσομοίωση ενός στοιχείου του seven segment display. Το A αντιστοιχεί στο πάνω led και το B αντιστοιχεί στο πάνω δεξιά led. Με την ίδια λογική αντιστοιχίζονται και τα υπόλοιπα leds του SSD. Αν για παράδειγμα θέλουμε να εμφανιστεί ο αριθμός μηδέν (0), τότε θα πρέπει να ενεργοποιηθούν όλα τα leds του SSD πλην του μεσαίου led (G). Οπότε θα δώσουμε την τιμή «1111110». Βέβαια αυτή η τιμή θα δινόταν αν θέλαμε να το εμφανίσουμε στο ίδιο board. Εμείς όμως θέλουμε να εμφανίσουμε το μηδέν στο δεύτερο board και όπως αναφέραμε προηγουμένως, κατά την διασύνδεση επικρατεί η ανάστροφη λογική. Οπότε θα δώσουμε ακριβώς τις αντίθετες τιμές, δηλαδή «0000001» και έτσι λύνουμε το πρόβλημά μας. Βάσει αυτής της λογικής δημιουργούμε έναν πίνακα 16 θέσεων (slv_array) και για κάθε αριθμό του πίνακα, η ανάλογη τιμή θα εμφανίζεται στο SSD.

iv. Σύνδεση μεταξύ των boards

Η δουλειά μας σε αυτή την περίπτωση είναι ευκολότερη σε σχέση με την ενότητα 4.1.1 μιας και δεν έχουμε τα switches, αλλά έχουμε μόνο ένα εξωτερικό παράγοντα, το push button που κάνει reset και ξεκινάει το μέτρημα από την αρχή. Συνδέουμε τα 2 boards σύμφωνα με την παρακάτω συνδεσμολογία:

Board 1 / E.C A2	->	Board 2 / E.C. A1
Connector 5	->	Connector 5
Connector 7	->	Connector 7
Connector 9	->	Connector 9
Connector 11	->	Connector 11
Connector 13	->	Connector 13
Connector 15	->	Connector 15
Connector 17	->	Connector 17
Connector 19	->	Connector 19
Connector 21	->	Connector 21
Connector 23	->	Connector 23
Connector 25	->	Connector 25
Connector 27	->	Connector 27

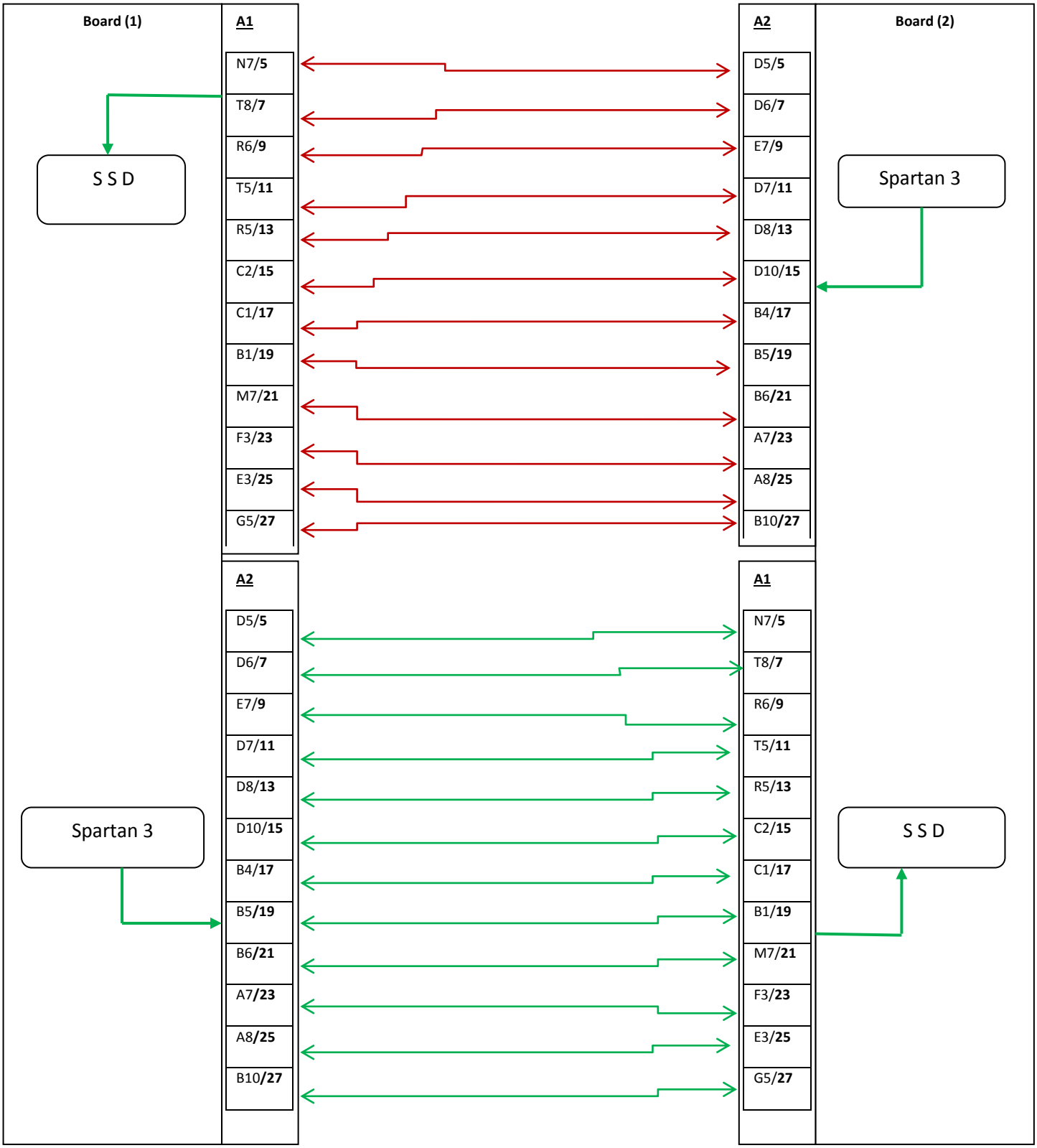
Πίνακας 3 - Σύνδεση των E.C. μεταξύ τους (SSD)

Board 2 / E.C A2	->	Board 1 / E.C. A1
Connector 5	->	Connector 5
Connector 7	->	Connector 7
Connector 9	->	Connector 9
Connector 11	->	Connector 11
Connector 13	->	Connector 13
Connector 15	->	Connector 15
Connector 17	->	Connector 17
Connector 19	->	Connector 19
Connector 21	->	Connector 21
Connector 23	->	Connector 23
Connector 25	->	Connector 25
Connector 27	->	Connector 27

Πίνακας 4 - Σύνδεση των E.C. μεταξύ τους (SSD)

Σημείωση: Ο κώδικας είναι ακριβώς ίδιος και για τα δύο boards, οπότε προγραμματίζουμε κάθε FPGA, με το ίδιο αρχείο.

v. Σχηματικό σύνδεση μεταξύ των boards



Σημείωση: Τα πράσινα και κόκκινα βελάκια αντιστοιχούν σε καλώδια ή καλωδιοταινίες. Σε κάθε board, τα δεδομένα από το FPGA πηγαίνουν για αποστολή στους E.C. A2, ενώ οι E.C. A1 κάνουν αποδοχή τα δεδομένα και τα στέλνουν στο SSD.

4.1.3 Σύνδεση των δύο Spartan - 3 για την μεταφορά των εξόδων - δεδομένων από την fifo του ενός board στο άλλο

Στην παρούσα ενότητα, θα κάνουμε κάτι διαφορετικό από τις προηγούμενες δύο. Θα χρησιμοποιήσουμε περισσότερο το FPGA και ειδικότερα τις μνήμες fifo που παράγει το πρόγραμμα **ISE Design Suite 14.2.**

i. Ανάλυση του προβλήματος

Θέλουμε να δίνουμε τιμές εισόδων από τα switches του ενός board και οι τιμές αυτές να αποθηκεύονται αρχικά σε μία μνήμη fifo η οποία δημιουργείται στο πρώτο board αλλά παράλληλα να υπάρχει η δυνατότητα να μεταφερθούν μέσω των expansion connectors και στο δεύτερο board. Ο στόχος μας είναι μάθουμε να χειριζόμαστε τις μνήμες τύπου fifo ώστε να εκμεταλλευτούμε τις σημαντικές λειτουργίες που μας παρέχει.

ii. Σχεδιασμός υλοποίησης

Αρχικά, σε κάθε board φτιάχνουμε μέσω του IP (Core generator & architecture wizard), μνήμες fifo. Έπειτα, συνδέοντας τα δύο board της Xilinx, θέλουμε να μεταφέρουμε τα δεδομένα-εξόδους της fifo από το πρώτο board, στο δεύτερο board.

iii. Δημιουργώντας μνήμες fifo

Και στα δύο boards θα ακολουθηθεί η ίδια διαδικασία.

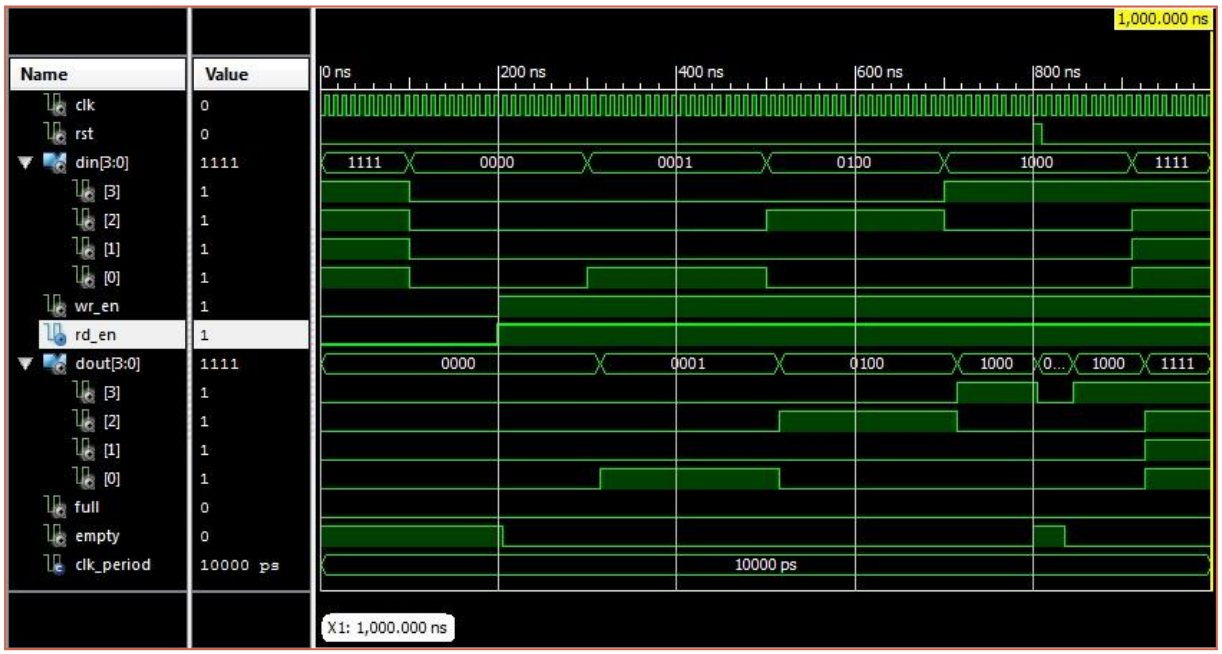
Διαδικασία: Σε κάθε board, θα χρησιμοποιηθούν 6 switches, 4 εκ των οποίων θα είναι οι είσοδοι στην 4-bit fifo. Τα άλλα 2 θα είναι χρησιμοποιηθούν για να δώσουμε τιμές στις τιμές write, read. Ακόμη, θα χρησιμοποιηθεί και ένα push button που θα κάνει reset την fifo. Τέλος, θα χρησιμοποιούμε και τα leds, για να εμφανίζουμε τις τιμές των εξόδων των fifo.

a) Μέσω του IP (Core generator & architecture wizard), έχουμε δημιουργήσει τις μνήμες fifo και έχει δημιουργηθεί ο ανάλογος κώδικας σε γλώσσα VHDL.
b) Μέσω του ucf, συνδέσουμε τις μεταβλητές din (0) - din (3) που είναι οι είσοδοι της fifo με τα sw (0) - sw (3). Ακόμα συνδέουμε και τις μεταβλητές write, read, reset με τα sw (6), sw (7) και button (0) αντίστοιχα.
c) Οι είσοδοι της fifo πηγαίνουν στον Spartan - 3, το οποίο επεξεργάζεται τα δεδομένα και παράγει τις ανάλογες εξόδους. Τις εξόδους που παράγει ο Spartan - 3, τις στέλνουμε αρχικά μέσω των κατάλληλων συνδέσεων (στο ucf), στα leds του board.
d) Έπειτα, από τα leds του board, στέλνουμε τα δεδομένα στον expansion connector A2 (ξανά μέσω ucf γίνεται η σύνδεση).

e) Από κει και πέρα συνδέσουμε τα expansion connector όπως φαίνεται στους πίνακες της σελίδας 50, μεταξύ τους και εμφανίζουμε τα δεδομένα στα leds του δεύτερου board.
f) Σαν επόμενο βήμα, θα μπορούσαμε τις εισόδους από το πρώτο board να τις στείλουμε στο Spartan - 3 του δεύτερου board για επεξεργασία (π.χ. σαν εισόδους στην νέα fifo).

Η μνήμη fifo (first in, first out), είναι μια μέθοδος οργάνωσης και αποθήκευσης δεδομένων κατά την οποία η παλαιότερη εγγραφή, επεξεργάζεται πρώτα και παράγεται η ανάλογη έξοδος. Η μνήμη fifo θα μας φανεί ιδιαίτερα χρήσιμη στο δεύτερο μέρος της εργασίας μας. Προς το παρόν, ξεκινώντας θα κάνουμε κάποιες δοκιμές έτσι ώστε να δούμε πως ακριβώς λειτουργούν οι μνήμες fifo που παράγει το πρόγραμμα **ISE Design Suite 14.2**, και σε τι χρησιμεύουν οι νέες μεταβλητές που υπάρχουν (reset, full, empty, read enable, write enable).

Στην παρακάτω εικόνα βλέπουμε το αποτέλεσμα ενός δοκιμαστικού μέσω του simulation (isim) του **ISE Design Suite 14.2**. Το simulation αυτό, ήταν η αρχική δοκιμή που μας έκανε να καταλάβουμε πως ακριβώς δουλεύει μια μνήμη fifo.

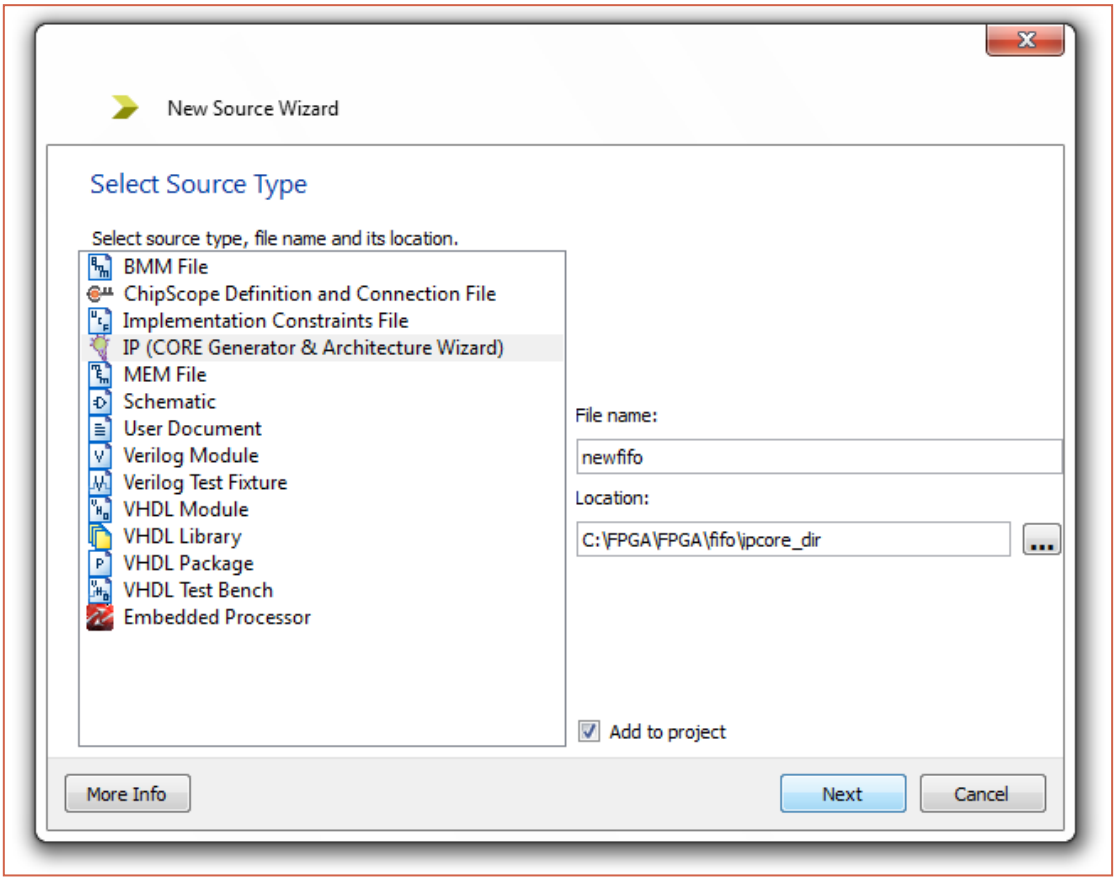


Εικόνα 35 - Προσομοίωση της μνήμης fifo

Η παραπάνω μνήμη fifo είναι μνήμη 4-bit δηλαδή τεσσάρων εισόδων, και 16 θέσεων (ο αριθμός των θέσεων ρυθμίζεται από τον προγραμματιστή). Εμείς στο συγκεκριμένο παράδειγμα δίνουμε τέσσερις διαφορετικές τιμές σε τυχαία χρονικά διαστήματα. Όσο η μεταβλητή wr_en που είναι συνδεδεμένη με εξωτερικό switch του board, είναι μηδέν, τότε δεν μπορούμε να αποθηκεύσουμε τίποτα στην μνήμη, ότι τιμές και να δίνουμε από τα άλλα switches. Μόλις το wr_en ενεργοποιηθεί και για όσο χρονικό διάστημα είναι ενεργοποιημένο, τότε θα μπορούμε να δίνουμε

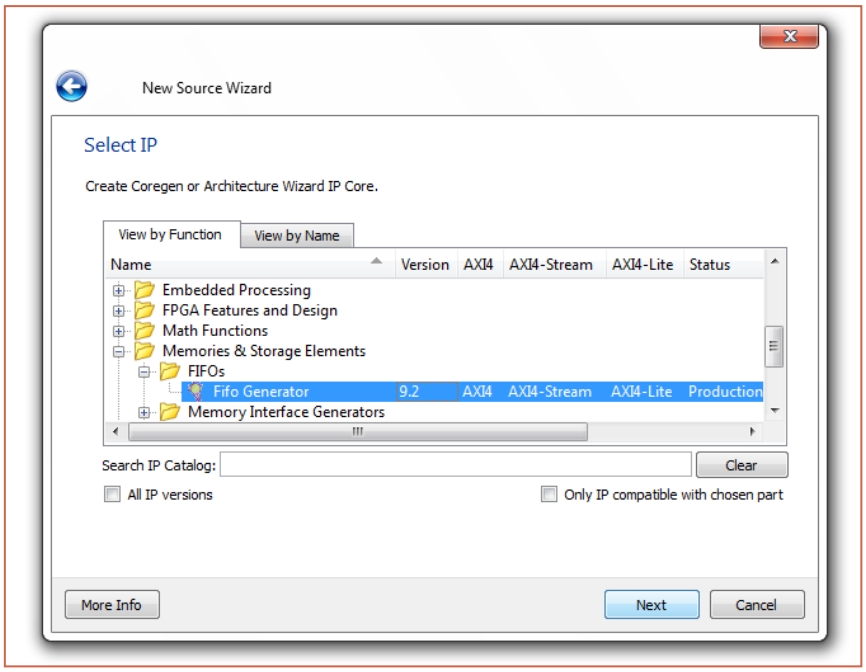
τιμές και αυτές να αποθηκεύονται στην fifo. Επόμενη μεταβλητή που βλέπουμε είναι το rd_en. Και αυτή η μεταβλητή είναι συνδεδεμένη με εξωτερικό switch του board. Όσο το rd_en είναι μηδέν, οι έξοδοι που παράγει η fifo, δεν αλλάζουν. Όταν ενεργοποιηθεί, τότε το dout (οι έξοδοι της fifo) μπορούν να πάρουν τις νέες τιμές που είχαμε δώσει όσο το rd_en ήταν απενεργοποιημένο. Τέλος υπάρχουν και οι μεταβλητές full, empty, clk_period. Οι μεταβλητές full και empty μας ενημερώνουν πότε η fifo είναι τελείως άδεια, δηλαδή δεν έχει κάποια τιμή και πότε έχει γεμίσει, δηλαδή πότε συμπληρώθηκαν όλες οι θέσεις μνήμης. Το clk_period είναι μια μεταβλητή που χρησιμοποιούμε για να ρυθμίσουμε εμείς το χρονικό διάστημα στο οποίο η fifo μπορεί να δεχτεί ή να εξάγει μια τιμή. Την χρησιμοποιούμε για λόγους ευκολίας μιας και το ρολόι του FPGA κάνει έναν πλήρη κύκλο ρολογιού σε μόλις 10 ns. Παρακάτω αναφέρουμε πως μπορούμε να δημιουργήσουμε μια μνήμη fifo στο ISE **Design Suite 14.2**.

Πως δημιουργούμε όμως μια μνήμη fifo; Στο αρχείο που θέλουμε, πατάμε δεξί κλικ και έπειτα επιλέγουμε το IP(CORE generator & Architecture wizard). Έπειτα πατάμε next.



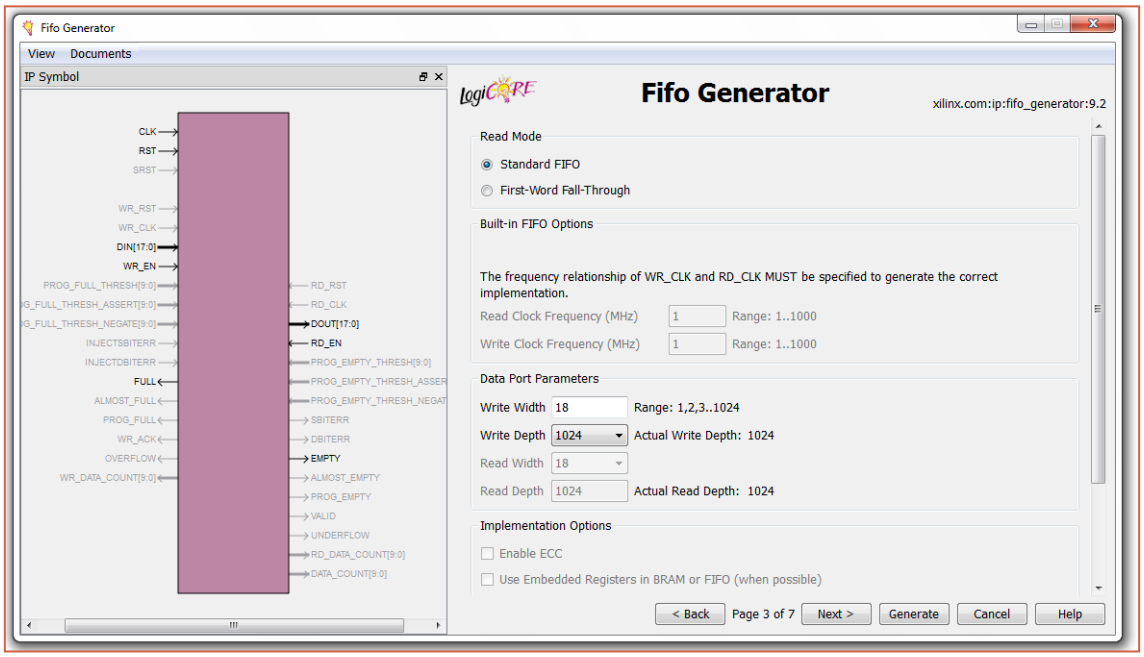
Εικόνα 36 - Πως ξεκινάει η δημιουργία μιας μνήμης fifo

Στην νέα οθόνη που εμφανίζεται επιλέγουμε Memories & Storage Elements > FIFOs > Fifo Genenerator.



Εικόνα 37 - Επιλογή του είδους μνήμης

Στην επόμενη οθόνη, θα μας εμφανιστούν 7 καρτέλες. Το μοναδικό που μας ενδιαφέρει στην παρούσα φάση είναι η τρίτη καρτέλα.



Εικόνα 38 - Η καρτέλα που ουσιαστικά διαμορφώνει το μέγεθος της μνήμης

Στο πεδίο Write Width επιλέγουμε το εύρος των μεταβλητών εισόδου που θα δώσουμε. Εμείς επιλέγουμε 4 (μέγεθος λέξης), και στο πεδίο Write Depth επιλέγουμε 16, το οποίο αντιστοιχεί στις 16 θέσεις μνήμης που θέλουμε να έχουμε στο συγκεκριμένο παράδειγμα με 4 μεταβλητές εισόδου (0000 - 1111). Τέλος πατάμε το Generate και η fifo δημιουργήθηκε και προστέθηκε στο project μας.

iv. Πειραματικό μέρος - δουλεύοντας με τα boards

Την προσομοίωση της εικόνας 35, θα την κάνουμε πράξη στο board Spartan - 3. Απο το πρώτο board θα δίνουμε τις 16 διαφορετικές δυνατές τιμές από τέσσερα switches (0000 - 1111) και στο δεύτερο board θα βλέπουμε πότε η μνήμη fifo του πρώτου board είναι γεμάτη και πότε έχει χώρο και τί τιμές έχουμε κάθε χρονική στιγμή (ουσιαστικά θα μεταφέρεται μέσω των expansion connectors η έξοδος της fifo). Ακολουθεί ο κώδικας της fifo και αναλυτικότερη επεξήγηση της διαδικασίας.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY test1 IS
5  signal clk : std_logic;
6     signal rst : std_logic := '0';
7     signal din : std_logic_vector(3 downto 0);
8     signal wr_en : std_logic ;
9     signal rd_en : std_logic ;
10    signal bt2: std_logic;
11    signal bt1 : std_logic;
12    signal dout : std_logic_vector(3 downto 0);
13    signal mout: std_logic_vector(3 downto 0);
14    signal leds: std_logic_vector(7 downto 0);
15 END test1;
16
17 ARCHITECTURE behavior OF test1 IS
18
19     COMPONENT fifo
20     PORT(
21          clk : IN  std_logic;
22          rst : IN  std_logic;
23          din : IN  std_logic_vector(3 downto 0);
24          wr_en : IN  std_logic;
25          rd_en : IN  std_logic;
26          dout : OUT  std_logic_vector(3 downto 0);
27          full : OUT  std_logic;
28          empty : OUT  std_logic
29
30         );
31     END COMPONENT;
32
33    signal full : std_logic;
34    signal empty : std_logic;
35
36 BEGIN
37
38    uut: fifo PORT MAP (
39           clk => clk,
40           rst => rst,
41           din => din,
42           wr_en => wr_en,
```

Εικόνα 39 - Κώδικας (μέρος Α)

Στις σειρές 4 έως 15 βλέπουμε την ενότητα entity η οποία περιλαμβάνει την δήλωση των σημάτων που θα χρησιμοποιήσουμε. Στις σειρές 19 έως 31 περιέχονται τα σήματα που έχει δημιουργήσαμε μέσω το **coregent** του **ISE Design suite 14.2.** όπως αναφέραμε και στην προηγούμενη ενότητα, μπορούμε να χρησιμοποιήσουμε τα σήματα αυτά μέσω της ενότητας component, σε οποιοδήποτε αρχείο ενός project. Στις σειρές 49 έως 91 βλέπουμε τον κώδικα if - else που χρησιμοποιήσαμε

έτσι ώστε να εμφανίζουμε τα αποτελέσματα στα leds. Η διαδικασία γεμίσματος της fifo είναι εσωτερική και δεν υπάρχει κώδικας. Βέβαια μέσω του simulation βλέπουμε όλες τις δυνατές τιμές που μπορεί να πάρει η fifo που δημιουργήσαμε.

```
43              rd_en => rd_en,
44              dout => dout,
45              full => full,
46              empty => empty
47            );
48
49      stim_proc: process
50      begin
51
52      if bt1 ='1' then
53      wr_en<='1';
54      else
55      wr_en<='0';
56      end if;
57
58      if bt2 ='1' then
59      rd_en<='1';
60      else
61      rd_en<='0';
62      end if;
63
64  mout<= not dout;
65
66      if leds(0) ='1' then
67      leds(4)  <='0';
68      else
69      leds(4)<='1';
70      end if;
71
72      if leds(1) ='1' then
73      leds(5)  <='0';
74      else
75      leds(5)<='1';
76      end if;
77
78      if leds(2) ='1' then
79      leds(6)  <='0';
80      else
81      leds(6)<='1';
82      end if;
83
```

Εικόνα 40 - Κώδικας (Μέρος Β)

```
84      if leds(3) ='1' then
85      leds(7)  <='0';
86      else
87      leds(7)<='1';
88      end if;
89
90      end process;
91  END;
```

Εικόνα 41 - Κώδικας (Μέρος Γ)

Ο κώδικας του ucf:

```
1  //sindesi gia metafora dedomenwn mesw fifo's
2  NET "clk" LOC = T9;
3  NET "bt1" LOC = K14;
4  NET "bt2" LOC = K13;
5  NET "rst" LOC = M13; //Button 0
6  NET "din(0)" LOC = F12;
7  NET "din(1)" LOC = G12;
8  NET "din(2)" LOC = H14;
9  NET "din(3)" LOC = H13;
10
```

Εικόνα 42 - Οι εξωτερικές μεταβλητές και το ρολόι

Αρχικά στην εικόνα 42, βλέπουμε την σύνδεση των switches που θα χρησιμοποιήσουμε για να δώσουμε τιμές (din (0) - din (3)), τις μεταβλητές write και read (bt1, bt2), το reset και το ρολόι.

```
11 //anama twn leds gia sto board 1 gia epivevaiwsi tvn dedomenwn
12 NET "dout(0)" LOC = K12;
13 NET "dout(1)" LOC = P14;
14 NET "dout(2)" LOC = L12;
15 NET "dout(3)" LOC = N14;
16
17 //metafora dedomenwn apo tis exodous tou board 1 sto board2 (output)
18 NET "mout(0)" LOC = D5; //exp2 pin 5
19 NET "mout(1)" LOC = D6; //exp2 pin 7
20 NET "mout(2)" LOC = E7; //exp2 pin 9
21 NET "mout(3)" LOC = D7; //exp2 pin 11
22
```

Εικόνα 43 - Οι έξοδοι dout της fifo και οι μεταβλητές mout που πάνε στους E.C.

Στην εικόνα 43, συνδέουμε τις εξόδους της fifo στα leds, έτσι ώστε να μπορούμε να βλέπουμε σε πραγματικό χρόνο τι τιμές παίρνουν. Αν για παράδειγμα δώσουμε την τιμή 1110, τότε θα ενεργοποιηθούν τρία leds και ένα θα παραμείνει απενεργοποιημένο. Τα αποτελέσματα της fifo θα τα στείλουμε και στους expansion connectors έτσι ώστε να μπορούν να επεξεργαστούν και από το δεύτερο board για οποιοδήποτε λόγο. Τέλος στην εικόνα 44 βλέπουμε 2 τετράδες με leds. Στην μια τετράδα βλέπουμε τα που χρησιμοποιούμε στα board και στην άλλη τα δεδομένα που μας στέλνει το άλλο board.

```
23 //eiserxomena dedomena apo to board 2 sto board 1 (input)
24 NET "leds(0)" LOC = C5; //exp2 pin 6
25 NET "leds(1)" LOC = C6; //exp2 pin 8
26 NET "leds(2)" LOC = C7; //exp2 pin 10
27 NET "leds(3)" LOC = C8; //exp2 pin 12
28
29 //emfanish twn eiderxomenwn dedomenwn sta leds
30 NET "leds(4)" LOC = P13;
31 NET "leds(5)" LOC = N12;
32 NET "leds(6)" LOC = P12;
33 NET "leds(7)" LOC = P11;
```

Εικόνα 44 - Σύνδεση των leds

v. Συνδεσμολογία μεταξύ των 2 board

Συνδέουμε τα 2 boards σύμφωνα με την παρακάτω συνδεσμολογία όπως φαίνεται στους παρακάτω δύο πίνακες:

Board 1 / E.C A2	->	Board 2 / E.C. A2
Connector 5	->	Connector 6
Connector 7	->	Connector 8
Connector 9	->	Connector 10
Connector 11	->	Connector 12

Πίνακας 5 - Σύνδεση των E.C. μεταξύ τους (fifo)

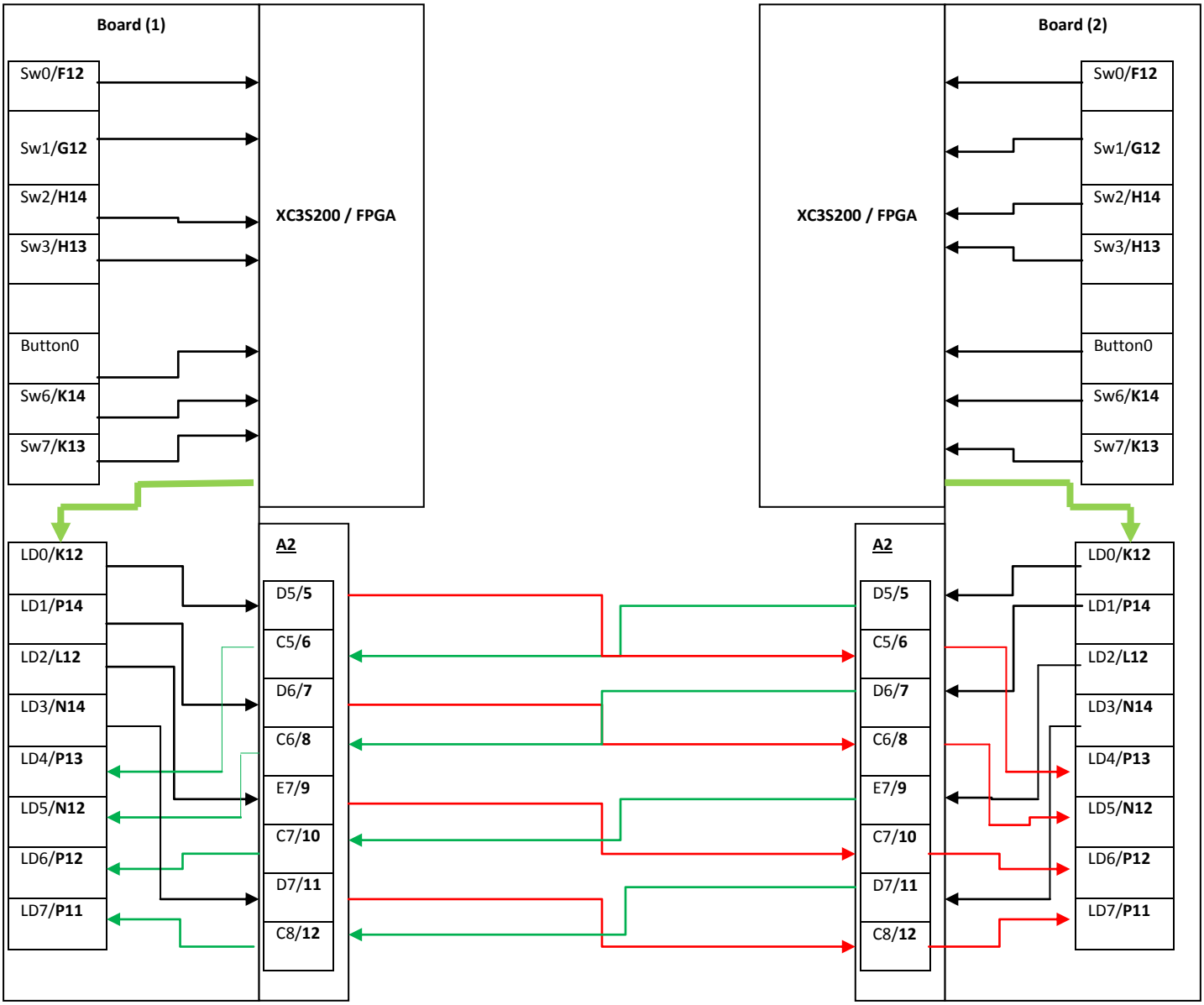
Board 2 / E.C A2	->	Board 1 / E.C. A2
Connector 5	->	Connector 6
Connector 7	->	Connector 8
Connector 9	->	Connector 10
Connector 11	->	Connector 12

Πίνακας 6 - Σύνδεση των E.C. μεταξύ τους (fifo)

Η σύνδεση μεταξύ των expansion connectors, μπορεί να γίνει είτε με απλά καλώδια, είτε με καλωδιοταινία. Η χρήση της καλωδιοταινίας, είναι προτιμότερη καθώς είναι δυσκολότερο και περισσότερο χρονοβόρο να δουλεύεις με μεμονωμένα καλώδια. Τα μεμονωμένα καλώδια είναι καλό να προτιμηθούν σε περιπτώσεις που χρειαζόμαστε λίγα connectors από τους expansion connectors.

* Το E.C. που βλέπουμε στους δύο παραπάνω πίνακες είναι συντομογραφία για τους Expansion Connectors.

vi. Σχηματική σύνδεση μεταξύ των boards



Σημαντικό: Στο παραπάνω σχήμα, βλέπουμε μια απεικόνιση η οποία προσεγγίζει όσο αυτό είναι δυνατόν την πραγματική υλοποίηση. Τα πράσινα και κόκκινα βελάκια εκτός των board, αντιστοιχούν σε απλά καλώδια ή καλωδιοταινίες (ανάλογα τι επιλέγουμε να χρησιμοποιήσουμε). Τα μαύρα, κόκκινα και πράσινα βελάκια (εντός των board), αντιστοιχούν στον κώδικα του project μας.

4.2 Μέρος Β

4.2.1 Γενικές πληροφορίες

Στο Μέρος Β της εργασίας μας, θα προσπαθήσουμε να ενώσουμε 2 board της Xilinx, Spartan - 3 και να χρησιμοποιήσουμε τις μνήμης Sram που έχει εσωτερικά το FPGA. Ας αναφέρουμε κάποια πράγματα για την μνήμη Sram, ώστε να καταλάβουμε καλύτερα τι είναι και πως θα το χρησιμοποιήσουμε.

Κάθε board Spartan – 3, έχει στο FPGA του μια μνήμη ram. Η μνήμη αυτή μπορεί να χρησιμοποιηθεί με δύο τρόπους:

- Ως μια ενιαία μνήμη 32 θέσεων των 256K, (256K x 32)
- Είτε ως δυο ανεξάρτητες μνήμες 16 θέσεων, (256K x 16)

Οι δύο Sram μοιράζονται κοινά σήματα εγγραφής (WE#), εξόδου (OE#) και διεύθυνσης (A[17:0]). Παρ' όλα αυτά κάθε ανεξάρτητη μνήμη, έχει ξεχωριστό σήμα ενεργοποίησης (CE#), το οποίο επιτρέπει την έλεγχο της και ξεχωριστό σήμα επιλογής υψηλής ή χαμηλής 16-bit λέξης δεδομένων, αντίστοιχα.

Το σήμα διεύθυνσης είναι κοινό και αποτελείται από ένα σήμα των 18-bit. Αυτή η διεύθυνση όταν χρησιμοποιούμε την Sram, χρησιμοποιεί τα παρακάτω pins του FPGA στον Expansion Connector A1:

Address Bit	FPGA Pin	A1 Expansion Connector Pin
A17	L3	35
A16	K5	33
A15	K3	34
A14	J3	31
A13	J4	32
A12	H4	29
A11	H3	30
A10	G5	27
A9	E4	28
A8	E3	25
A7	F4	26
A6	F3	23
A5	G4	24
A4	L4	14
A3	M3	12
A2	M4	10
A1	N3	8
A0	L5	6

Εικόνα 45 - Τα pins της διεύθυνσης

Τα σήματα εγγραφής και εξόδου (WE#, OE#), χρησιμοποιούν τα εξής pins:

Signal	FPGA Pin	A1 Expansion Connector Pin
OE#	K4	16
WE#	G3	18

Εικόνα 46 - Σήματα εγγραφής και εξόδου

Τα δεδομένα εισόδου (IO[15:0]), επιλογής μνήμης (CE1#), υψηλής και χαμηλής λέξης δεδομένων (UB1#, UB2#), φαίνονται στις παρακάτω δύο εικόνες:

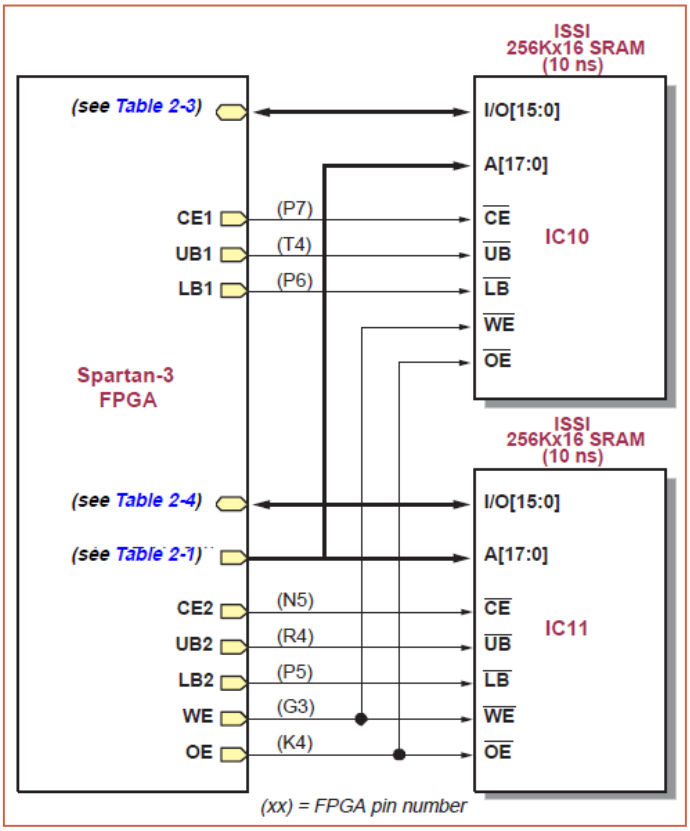
Signal	FPGA Pin	A1 Expansion Connector Pin
IO15	R1	
IO14	P1	
IO13	L2	
IO12	J2	
IO11	H1	
IO10	F2	
IO9	P8	
IO8	D3	
IO7	B1	19
IO6	C1	17
IO5	C2	15
IO4	R5	13
IO3	T5	11
IO2	R6	9
IO1	T8	7
IO0	N7	5
CE1 (chip enable IC10)	P7	
UB1 (upper byte enable IC10)	T4	
LB1 (lower byte enable IC10)	P6	

Εικόνα 47 - Πρώτη μνήμη

Signal	FPGA Pin
IO15	N1
IO14	M1
IO13	K2
IO12	C3
IO11	F5
IO10	G1
IO9	E2
IO8	D2
IO7	D1
IO6	E1
IO5	G2
IO4	J1
IO3	K1
IO2	M2
IO1	N2
IO0	P2
CE2 (chip enable IC11)	N5
UB2 (upper byte enable IC11)	R4
LB2 (lower byte enable IC11)	P5

Εικόνα 48 - Δεύτερη μνήμη

Παρακάτω, παραθέτουμε ένα σχεδιάγραμμα το οποίο παρουσιάζει την συνδεσμολογία όλων των παραπάνω στοιχείων με το FPGA.



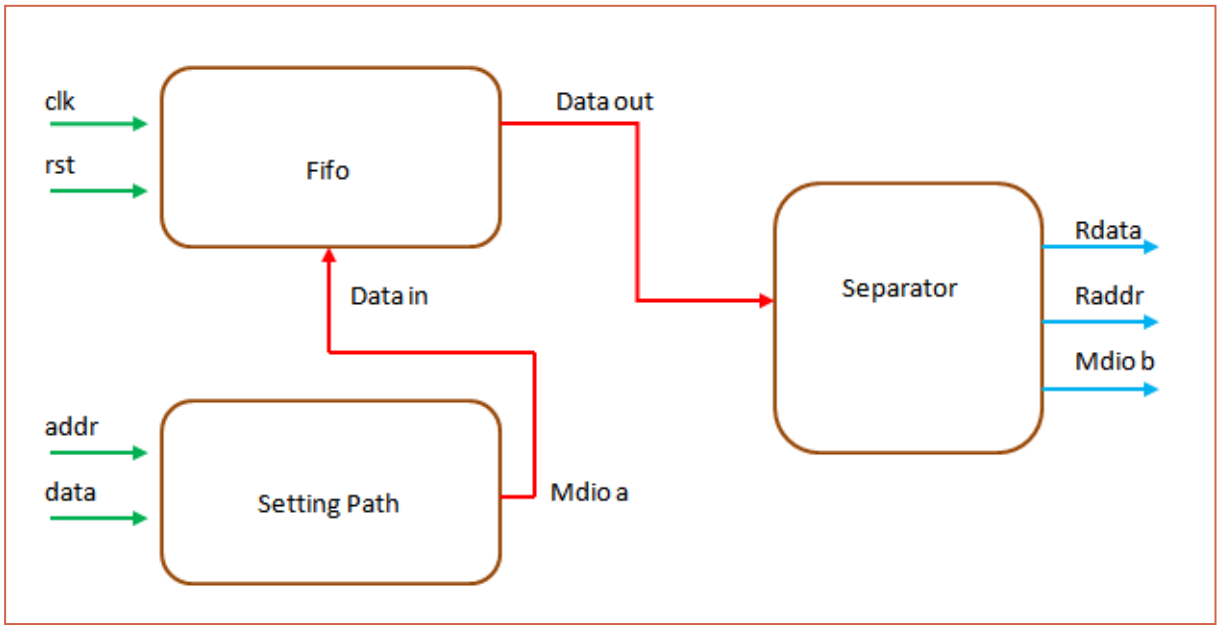
Εικόνα 49 - Η επικοινωνία του FPGA με τις 2 sram

i. Ανάλυση προβλήματος

Στην εργασία μας, θα χρησιμοποιήσουμε την μνήμη Sram, έτσι ώστε να γίνεται διαμερισμός των δεδομένων της κεντρικής Sram, στα υπόλοιπα συνδεδεμένα boards με σκοπό να πετύχουμε την γρηγορότερη επεξεργασία των δεδομένων αυτών.

ii. Σχεδιασμός λύσης

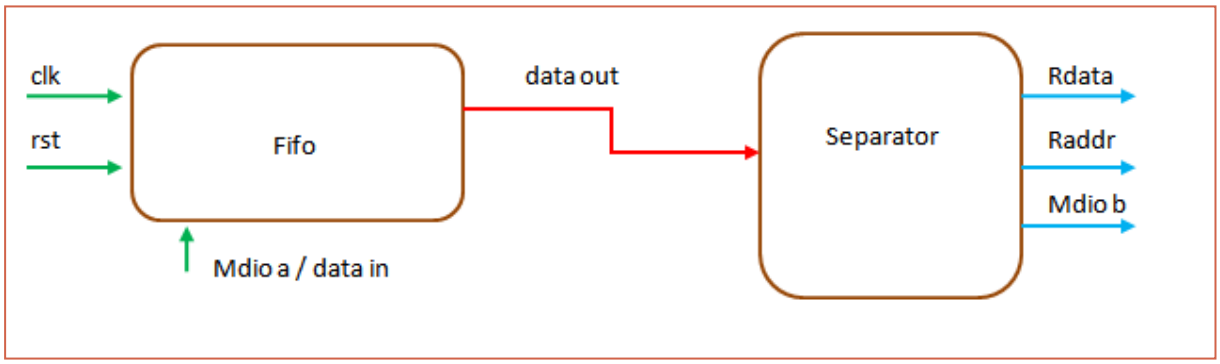
Παρακάτω, ακολουθούν δύο εικόνες στις οποίες θα προσπαθήσουμε να αποτυπώσουμε, υπό μορφή σχήματος, την διαδρομή που θα ακολουθήσουν τα δεδομένα μας. Η εικόνα 50, αντιστοιχεί μέρος του κώδικα που θα τρέχει στο πρώτο board, ενώ η εικόνα 51, στο μέρος του κώδικα των υπόλοιπων board που θα συνδέσουμε.



Εικόνα 50 - Σχηματικό του πρώτου board

Στην εικόνα 50, έχουμε αρχικά τέσσερις εισόδους (clk, rst, addr, data). Η είσοδος clk αντιστοιχεί στο ρολόι συγχρονισμού, το rst στο σήμα που επανεκκινεί το κύκλωμα, το addr αντιστοιχεί στην διεύθυνση μνήμης και το data που αντιστοιχεί στα δεδομένα που εισέρχονται στην μνήμη. Απο το Setting path, έχουμε μια έξοδο το Mdio a, το αντιστοιχίζεται στη μεταβλητή Data in, η οποία με την σειρά της είναι μεταβλητή εισόδου στην μνήμη fifo που έχουμε δημιουργήσει. Η fifo, έχει έξοδο ίση με τα bit εισόδου (μεταβλητή Data in). Η έξοδος Data out, χρησιμοποιείται σαν είσοδος στο κομμάτι κώδικα (δηλαδή μια process) με το όνομα Separator. Ο Separator, έχει σκοπό να διαχωρίσει τα σήματα εισόδου από την διεύθυνση μνήμης.

Η λειτουργία του Separator, θα αναλυθεί εκτενέστερα, αργότερα. Εν τέλει, ο Separator βγάζει τρείς εξόδους (Rdata, Raddr, Mdio b). Η έξοδος Rdata αντιστοιχεί στα δεδομένα που θα αποθηκευτούν στην Sram του πρώτου board (η αντιστοίχηση θα γίνει μέσω του ucf) και αντίστοιχα η έξοδος Raddr αντιστοιχεί στην διεύθυνση μνήμης των δεδομένων Rata, ενώ τέλος η έξοδος Mdio b, είναι τα δεδομένα τα οποία αποστέλλονται αρχικά στους expansion connectors και έπειτα αποθηκεύονται στην fifo του δεύτερου board. Ο κώδικας αυτός, κατά κάποιο τρόπο λειτουργεί σαν master.



Εικόνα 51 - Σχηματικό των υπόλοιπων boards

Στην εικόνα 51, βλέπουμε το κομμάτι του κώδικα που τρέχει στα board που λειτουργούν σαν slaves. Παρατηρούμε ότι αρχικά έχουμε μόνο δύο εισόδους (clk, rst). Η είσοδος clk αντιστοιχεί στο ρολόι συγχρονισμού, το rst στο σήμα που επανεκκινεί το κύκλωμα. Η επιπλέον είσοδος είναι το Mdio a, η οποία είναι είσοδος δεδομένων που έρχονται από το πρώτο board (master), στο δεύτερο board (slave). Απο την fifo του κυκλώματος, παράγεται μία έξοδος, η data out, η οποία πηγαίνει στον Separator και διαχωρίζει τα σήματα. Ο Separator λειτουργεί όπως και στο master board, δηλαδή διαχωρίζει το σήμα εισόδου και παράγει τα σήματα εξόδου Mdio b, τα δεδομένα Mdata που αποθηκεύονται τοπικά στην Sram και το σήμα διευθύνσεως Raddr που αντιστοιχεί σε αυτά τα δεδομένα.

iii. Πειραματικό μέρος

Ο κώδικας μας, αποτελείται από 3 αρχεία VHDL, το αρχείο test_mux, το αρχείο mux (που περιέχει τον κώδικα του Separator) και το αρχείο STD_FIFO που είναι η fifo μας.

a) Το αρχείο test_mux

Το αρχείο test_mux, είναι το βασικό αρχείο του project. Ακολουθούν εικόνες οι οποίες περιέχουν τον κώδικα του αρχείου. Κάτω από κάθε εικόνα, υπάρχουν σχόλια και σημειώσεις οι οποίες αναφέρουν την λειτουργία του κώδικα.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4
5  ENTITY test_mux IS
6  END test_mux;
7
8  ARCHITECTURE behavior OF test_mux IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11     COMPONENT mux
12     PORT(
13          clk : IN  std_logic;
14          rst : IN  std_logic;
15          Mdio_a : IN  std_logic_vector(17 downto 0);
16          SRaddr : OUT  std_logic_vector(17 downto 0);
17          SRdata : OUT  std_logic_vector(15 downto 0);
18          Mdio_b : OUT  std_logic_vector(17 downto 0)
19         );
20     END COMPONENT;
21
22    -- Inputs
23    signal clk : std_logic := '0';
24    signal rst : std_logic := '1';
25    signal en : std_logic := '0';
26    signal Mdio_a : std_logic_vector(17 downto 0):= (others =>'0');
27
28    -- Outputs
29    signal SRaddr : std_logic_vector(17 downto 0);
30    signal SRdata : std_logic_vector(15 downto 0);
31    signal Mdio_b : std_logic_vector(17 downto 0);
32
33    -- Clock period definitions
34    constant clk_period : time := 10 ns;
35
36    -- my signals
37    signal wr : std_logic :='1';
38    signal wr2 : std_logic :='0';
39    signal switch : std_logic := '0';
40    signal Saddr : std_logic_vector(17 downto 0):= (others =>'0');
41    signal Sdata : std_logic_vector(15 downto 0):= x"0000";
42
```

Εικόνα 52 - Μέρος Α

Στις γραμμές 11 έως 20, υπάρχει ένα component το οποίο περιέχει τα σήματα του δεύτερου VHDL αρχείου, του mux. Στις σειρές 23 έως 41 αρχικοποιούνται τα σήματα που θα χρησιμοποιήσουμε. Σήματα εισόδου (Inputs), σήματα εξόδου (outputs) και εσωτερικά σήματα (my signals).

```
43  BEGIN
44
45     -- Instantiate the Unit Under Test (UUT)
46     uut: mux PORT MAP (
47            clk => clk,
48            rst => rst,
49            Mdio_a => Mdio_a,
50            SRaddr => SRaddr,
51            SRdata => SRdata,
52            Mdio_b => Mdio_b
53          );
54
55     -- Clock process definitions
56     clk_process :process
57     begin
58        clk <= '0';
59        wait for clk_period/2;
60        clk <= '1';
61        wait for clk_period/2;
62     end process;
63
```

Εικόνα 53 - Μέρος Β

Στις γραμμές 46 έως 53 συνδέουμε τα αρχεία test_mux και mux και αντιστοιχίζουμε τα μεταξύ τους σήματα. Παρακάτω στις γραμμές 56 έως 62 υπάρχει μια process (clk_process) η οποία ρυθμίζει την περίοδο του κυκλώματος μας. Στην επόμενη εικόνα, ακολουθεί η process Setting Path.

```
64  -- setting path (master board)
65  Setting_path:  process(Sdata,en,switch,Saddr)
66  variable path, cnt : integer range 0 to 4;
67  begin
68
69  if ( wr='1' and en='1')then
70     Mdio_a <=conv_std_logic_vector(path,2)&Sdata;
71     wr2<='1';
72  end if;
73
74  if(en='0' and wr2='1') then
75  Mdio_a<=Saddr;
76     if (path < 1 ) then
77        path:=path+1;
78        wr<='0';
79     else
80        path:=0;
81        wr<='0';
82     end if;
83  end if;
84
85  if( en='1' and wr='0' and wr2='1') then
86  wr<='1';
87  wr2<='0';
88     if switch='1' then
89        switch<='0';
90     else
91        switch<='1';
92     end if;
93  end if;
94  end process;
95  -- end of setting path
```

Εικόνα 54 - Μέρος Γ (Setting Path)

Το Setting Path, είναι ένα σημαντικό μέρος του κώδικά μας στο αρχείο test_mux. Αναφερθήκαμε σε αυτό, στην εικόνα 50 ως ένα εκ των κύριων μερών του κυκλώματος μας. Ουσιαστικά είναι μια process η οποία κάνει τρείς επιμέρους ελέγχους με τον κώδικα ελέγχου if - else. Η process λειτουργεί σε 3 καταστάσεις που αντιστοιχούν στους 3 ελέγχους που κάνει. Αρχικά ελέγχει τα δύο σήματα εγγραφής (wr, wr2) και το σήμα ελέγχου en (enable). Στον πρώτο έλεγχο - κατάσταση (σειρές 69 έως 72), αν τόσο το en όσο και το wr έχουν λογικό άσσο, τότε βάζει στην μεταβλητή Mdio a, το Sdata στο οποίο προσθέτουμε 2 - bit που ορίζουν το board που θα πάνε. Το path έχει οριστεί σαν μια ακέραια μεταβλητή, οπότε χρησιμοποιούμε την εντολή convert_std_logic για να την μετατρέψουμε σε λογική μεταβλητή. Έπειτα, η μεταβλητή wr2, γίνεται ίση με άσσο. Όταν το wr2 γίνει άσσος, τότε γίνεται ο έλεγχος των επόμενων 2 καταστάσεων (για en ίσο με μηδέν ή ένα). Στον δεύτερο έλεγχο - κατάσταση, εάν η μεταβλητή en είναι ίση με 0, τότε τα δεδομένα της μεταβλητής Saddr αποθηκεύονται στο Mdio a. Έπειτα η εσωτερική μεταβλητή path, αποφασίζει αν θα κρατήσει τα δεδομένα σε αυτό το board ή θα τα στείλει στους expansion connectors. Στον τρίτο και τελευταίο έλεγχο - κατάσταση, ελέγχουμε τόσο το en, όσο και τα σήματα wr, wr2.

Τέλος, ακολουθεί μια process 145 σειρών η οποία παίρνει τυχαίες τιμές στις μεταβλητές Sdata και Saddr, με σκοπό να δημιουργηθεί μια ρεαλιστική προσομοίωση του κυκλώματος, μέσω του isim. Για λόγους εξοικονόμησης χώρους, στην εικόνα 55 δείχνουμε μερικές μόνο σειρές από τον κώδικα, μιας και οι σειρές που δεν φαίνονται, έχουν τα ίδια δεδομένα σε επανάληψη.

```
 97  stim_proc: process
 98     variable data ,addr :   integer range 0 to 32;
 99     begin
100     wait for 20 ns;
101        rst<='0';
102        addr:=addr+2;
103        data:=data+3;
104        en<='1';
105  Sdata<=conv_std_logic_vector(data,16);
106  Saddr<=conv_std_logic_vector(addr,18);
107  wait for 10 ns;
108     en<='0';
109  wait for 10 ns;
110        addr:=addr+2;
111        data:=data+3;
112        en<='1';
113  Sdata<=conv_std_logic_vector(data,16);
114  Saddr<=conv_std_logic_vector(addr,18);
```

Εικόνα 55 - Μέρος Δ (κώδικας προσομοίωσης)

b) Το αρχείο mux

Το επόμενο αρχείο του συγκεκριμένου project, είναι το αρχείο mux. Σε αυτό το αρχείο, περιλαμβάνετε το κομμάτι κώδικα στο οποίο αναφερθήκαμε με το όνομα Separator, στις σελίδες 55 έως 56 και στις εικόνες 50 και 51. Η process Separator, είναι μια διεργασία κατά την οποία διαχωρίζονται τα σήματα εισόδου και τα κατατάσσονται σε δύο κατηγορίες. Η πρώτη κατηγορία, είναι τα σήματα που μένουν στο board (Rdata) και στέλνονται στην Sram και η δεύτερη κατηγορία είναι τα δεδομένα που αποστέλλονται σε άλλο board (Mdio b).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mux is
5        port(
6           clk      : IN  std_logic;
7           rst      : IN  std_logic;
8           Mdio_a      : IN  std_logic_vector(17 downto 0); -- to fifo
9           SRaddr   : out  std_logic_vector(17 downto 0);   -- Sram_address
10          SRdata      : OUT  std_logic_vector(15 downto 0);-- Sram_data
11          Mdio_b   : OUT  std_logic_vector(17 downto 0)    -- forward
12          );
13 end mux;
14
15 architecture Behavioral of mux is
16    COMPONENT STD_FIFO
17    PORT(
18          clk : IN  std_logic;
19          rst : IN  std_logic;
20          DataIn : IN  std_logic_vector(17 downto 0);
21          WriteEn : IN  std_logic;
22          ReadEn : IN  std_logic;
23          DataOut : OUT  std_logic_vector(17 downto 0);
24          full : OUT  std_logic;
25          empty : OUT  std_logic
26         );
27    END COMPONENT;
```

Εικόνα 56 - Δηλώσεις των σημάτων

Στην εικόνα 56, γίνεται η δήλωση των σημάτων που χρησιμοποιεί το αρχείο. Επίσης αρχικοποιείται το αρχείο STD_FIFO, το οποίο θα είναι το τρίτο μας αρχείο που περιέχει μια μνήμη fifo.

Στην εικόνα 57, γίνεται η δήλωση των εσωτερικών μεταβλητών που θα χρησιμοποιηθούν και η αντιστοίχιση των μεταβλητών (μέσω του port map) του αρχείου mux, με τις μεταβλητές του αρχείου STD_FIFO.

```
28  -- mysignal
29  signal rd      : std_logic :='1';
30  signal wr      : std_logic :='1';
31  signal empty   : std_logic;
32  signal full    : std_logic;
33  signal check   : std_logic_vector(17 downto 0);
34  signal wrd     : std_logic:='1' ;
35  signal wrin    : std_logic :='0';
36  signal wrad    : std_logic:='1' ;
37  signal en      : STD_LOGIC_VECTOR(1 DOWNTO 0):="00";
38  signal flag    : std_logic :='0';
39  signal switch  :  STD_LOGIC:='0';
40
41  begin
42     fifo_link: STD_FIFO
43     PORT MAP (
44            clk =>clk,
45            rst => rst,
46            DataIn => Mdio_a,
47            WriteEn => wr,
48            ReadEn => rd,
49            DataOut => check,
50            full => full,
51            empty => empty
52            );
```

Εικόνα 57 - Δήλωση σημάτων

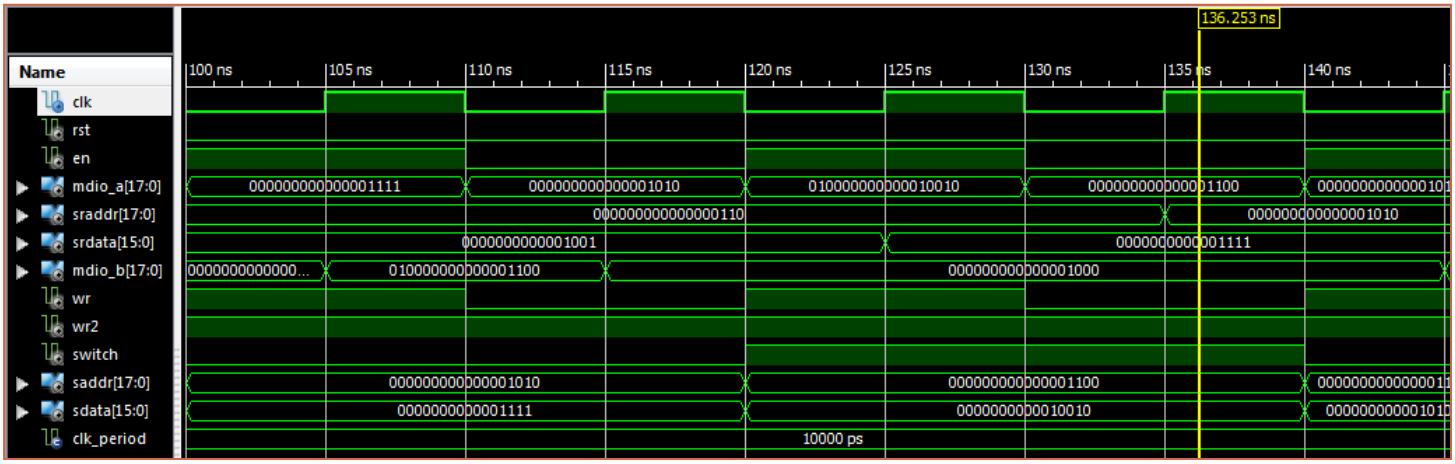
Παρακάτω στην εικόνα 58, βλέπουμε την process Separator.

```
53  -- Separator
54  Separator: process(clk,check)
55  variable Separator,cnt,cnt2: integer range 0 to 10;
56  begin
57     rd<='1';
58     if (check(1 downto 0) = not "00" )then
59        wrin<='1';
60     end if;
61
62     if ( check(17 downto 16) ="00" and wrin ='1') then
63
64        if (clk'event and clk ='1' and cnt=0) then
65           SRdata<=check(15 downto 0);
66           cnt:=1;
67        elsif (clk'event and clk ='1' and cnt=1)then
68           SRaddr<=check(17 downto 0);
69           cnt:=0;
70           en<="11";
71           wrin<='0';
72        end if;
73     elsif( en ="11") then
74        if (clk'event and clk ='1' and cnt2=0) then
75           Mdio_b<=check;
76           cnt2:=1;
77        elsif (clk'event and clk ='1' and cnt2 =1) then
78           Mdio_b<=check;
79           cnt2:=0;
80           wrin<='1';
81           en<="00";
82        end if;
83     end if;
84  end process; -- end of Separator
85  end Behavioral;
```

Εικόνα 58 - Η process Separator

Η process Separator, λειτουργεί ως εξής: ενεργοποιεί την μεταβλητή διαβάσματος και αν δει, μέσω του ελέγχου της if - end if (μεταβλητή check, διάφορη του 00), ότι τα εισερχόμενα δεδομένα είναι για το παρόν board, ενεργοποιεί την μεταβλητή εγγραφής (σειρές 57 έως 60). Απο κει και πέρα, πάλι μέσω μιας if - else, κοιτάει τα δύο πρώτα bit των 18 - bit δεδομένων.

- Αν αυτά τα δύο ψηφία (δηλαδή η επικεφαλίδα των δεδομένων) είναι 00, τότε κρατάει τα υπόλοιπα 16 bit και τα αποθηκεύει στην μεταβλητή Srdata, ενώ στη συνεχεία κάνει το σήμα cnt ίσο με λογικό άσσο, δηλαδή μετράει ότι πέρασε τα δεδομένα και ενεργοποιείται το δεύτερο μέρος του κώδικα για να περάσει και τη μεταβλητή Sraddr η οποία θα στείλει τα δεδομένα στη μνήμη Sram του FPGA.
- Αν τα δύο πρώτα ψηφία (η επικεφαλίδα) είναι 11, τότε στέλνει το σύνολο των 18 bit στην μεταβλητή Mdio B. Με την σειρά της, η μεταβλητή Mdio B, θα στείλει μέσω του ucf, τα δεδομένα στους expansion connectors.



Εικόνα 59 - Προσομοίωση του κώδικα

Η εικόνα 59, είναι η προσομοίωση του κώδικά μας, με τυχαίες τιμές που δώσαμε στο αρχείο test_mux, στην process stim_proc. Το σήμα mdio_a, είναι τα εναλλάξ δεδομένα sdata και saddr. Απο το σήμα srdata, φαίνεται ότι στην τοπική Sram, μένουν μόνο δεδομένα όπου τα 2 πρώτα bit είναι 00. Απο την άλλη πλευρά, στην μεταβλητή mdio_b (δεδομένα προς αποστολή στο επόμενο board), μπαίνουν και σήματα με επικεφαλίδα 00 και 01.

Στα σήματα saddr, sdata, είναι οι τυχαίες τιμές που δίνει ο χρήστης μέσω της process stim_proc.

c) Το αρχείο STD_FIFO

Τέλος, το τελευταίο αρχείο του project είναι το αρχείο STD_FIFO. Το συγκεκριμένο αρχείο είναι μια fifo ανοιχτού κώδικα, σε γλώσσα VHDL.

```
1  library IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  entity STD_FIFO is
6     Generic (
7        constant DATA_WIDTH : positive := 18;
8        constant FIFO_DEPTH : positive := 1024
9     );
10    Port (
11       CLK     : in  STD_LOGIC;                                    -- Clock input
12       RST     : in  STD_LOGIC;                                    -- Active high reset
13       WriteEn : in  STD_LOGIC;                                    -- Write enable signal
14       DataIn  : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);   -- Data input bus
15       ReadEn  : in  STD_LOGIC;                                    -- Read enable signal
16       DataOut : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0):= (others=>'0');   -- Data output bus
17       Empty   : out STD_LOGIC:='1';                                  -- FIFO empty flag
18       Full    : out STD_LOGIC:='1'                                  -- FIFO full flag
19    );
20 end STD_FIFO;
21
22 architecture Behavioral of STD_FIFO is
23 begin
24    -- Memory Pointer Process
25    fifo_proc : process (CLK)
26       type FIFO_Memory is array (0 to FIFO_DEPTH - 1) of STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
27       variable Memory : FIFO_Memory;
28       variable Head : natural range 0 to FIFO_DEPTH - 1;
29       variable Tail : natural range 0 to FIFO_DEPTH - 1;
30       variable Looped : boolean;
31    begin
32       if rising_edge(CLK) then
33          if RST = '1' then
34             Head := 0;
35             Tail := 0;
36             Looped := false;
37             Full  <= '0';
38             Empty <= '1';
39          else
```

Εικόνα 60 - Δηλώσεις της fifo και έναρξη της process

```
40             if (ReadEn = '1') then
41                if ((Looped = true) or (Head /= Tail)) then
42                   -- Update data output
43                   DataOut <= Memory(Tail);
44                   -- Update Tail pointer as needed
45                   if (Tail = FIFO_DEPTH - 1) then
46                      Tail := 0;
47                      Looped := false;
48                   else
49                      Tail := Tail + 1;
50                   end if;
51                end if;
52             end if;
53
54             if (WriteEn = '1') then
55                if ((Looped = false) or (Head /= Tail)) then
56                   -- Write Data to Memory
57                   Memory(Head) := DataIn;
58                   -- Increment Head pointer as needed
59                   if (Head = FIFO_DEPTH - 1) then
60                      Head := 0;
61                      Looped := true;
62                   else
63                      Head := Head + 1;
64                   end if;
65                end if;
66             end if;
67             -- Update Empty and Full flags
68             if (Head = Tail) then
69                if Looped then
70                   Full <= '1';
71                else
72                   Empty <= '1';
73                end if;
74             else
75                Empty <= '0';
76                Full  <= '0';
77             end if;
78          end if;
79       end if;
80    end process;
81 end Behavioral;
```

Εικόνα 61 - Συνθήκες ελέγχου της process

5. Αποτελέσματα

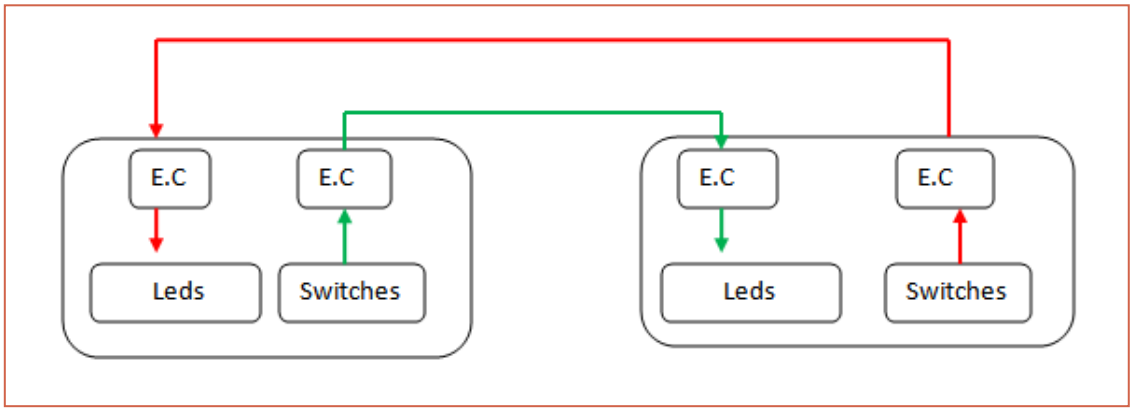
Το FPGA είναι ένα κύκλωμα που μας δίνει την δυνατότητα να παράγουμε εμείς τα δικά μας κυκλώματα, είτε με την μορφή σχεδίασης (τοποθετώντας και ενώνοντας λογικές πύλες της ψηφιακής σχεδίασης), είτε μέσω γλωσσών περιγραφής υλικού (HDL, VHDL, Verilog HDL, AHDL). Στην παρούσα πτυχιακή, προσπαθήσαμε να δημιουργήσουμε τα δικά μας κυκλώματα που θα επιτρέπουν την αλληλεπίδραση μεταξύ των boards και εν τέλει την χρησιμοποίηση των Sram για να επιτύχουμε γρήγορη επεξεργασία εφαρμογών.

Με την ολοκλήρωση της πτυχιακής μας, μπορέσαμε να βγάλουμε ενδιαφέροντα συμπεράσματα για τα FPGA και για την μεταξύ τους διασύνδεση. Τα αποτελέσματα των πειραματικών διαδικασιών μας, έδειξαν ότι δύο ή περισσότερα FPGA, μπορούν να συνδεθούν μεταξύ τους και να αλληλεπιδράσουν. Σημαντικό ρόλο σε αυτό, παίζει το γεγονός ότι η κατασκευάστρια εταιρεία, έχει δώσει αυτή την δυνατότητα στο board μέσω των expansion connectors.

5.1 Συμπεράσματα

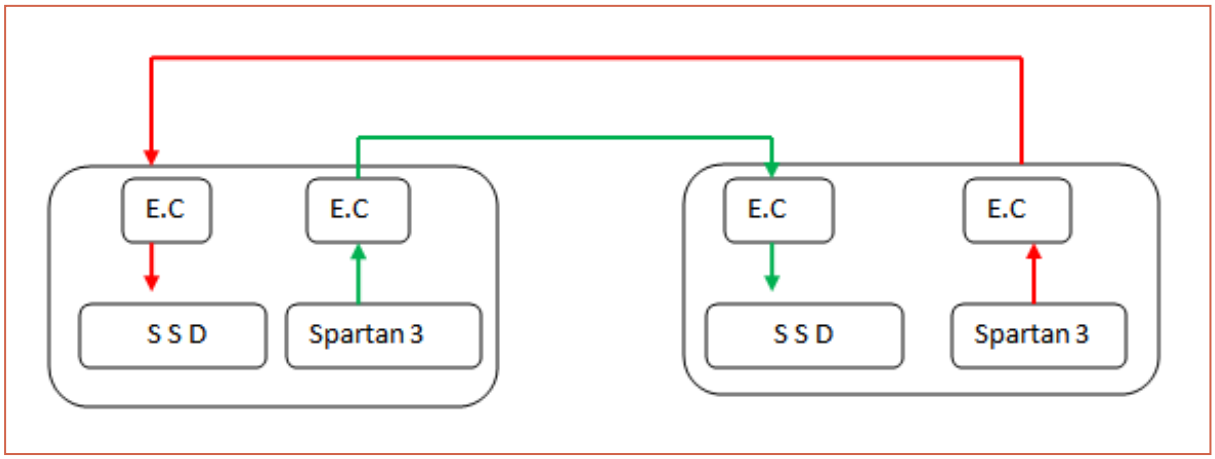
Ας δούμε τα συμπεράσματα που αντλήσαμε μέσω των πειραματικών δοκιμών μας. Χωρίσαμε την πτυχιακή μας σε δύο μέρη, στο πρώτο που κάνουμε απλούστερες δοκιμές με μεταφορά απλών δεδομένων και χειρισμό των στοιχείων του board, ώστε να ελέγχουμε με τις εισόδους του ενός board (push buttons, switches), τις εξόδους του άλλου board (leds, seven segment display). Παράλληλα δημιουργήσαμε και μνήμες τύπου fifo και μεταφέραμε τις εξόδους της fifo από το ένα board στο άλλο.

Στο πρώτο μέρος, στην ενότητα 4.1.1, με τα switches του πρώτου board και με κατάλληλο κώδικα, μέσω των expansion connectors, ενεργοποιούσαμε και απενεργοποιούσαμε τα leds του δεύτερου board και αντίστροφα.



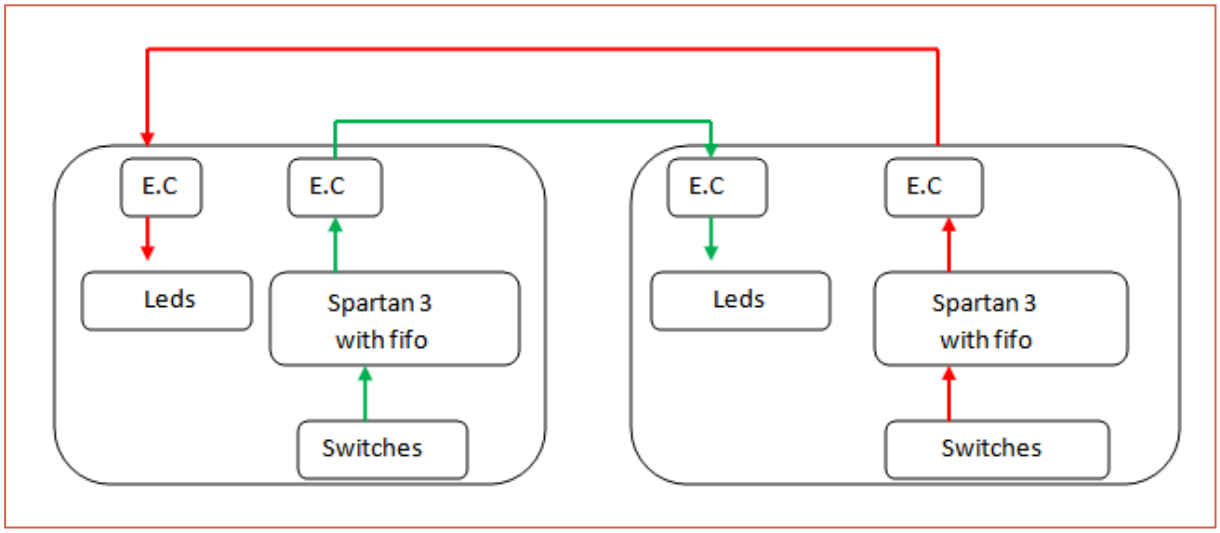
Εικόνα 62 - Η ενότητα 4.1.1

Στην ενότητα 4.1.2, το FPGA έστελνε τα αποτελέσματα του κώδικα που έτρεχε στο πρώτο board, στους expansion connectors και ενεργοποιούνταν τα leds του Seven segment display του δεύτερου board και αντίστροφα.



Εικόνα 63 - Η ενότητα 4.1.2

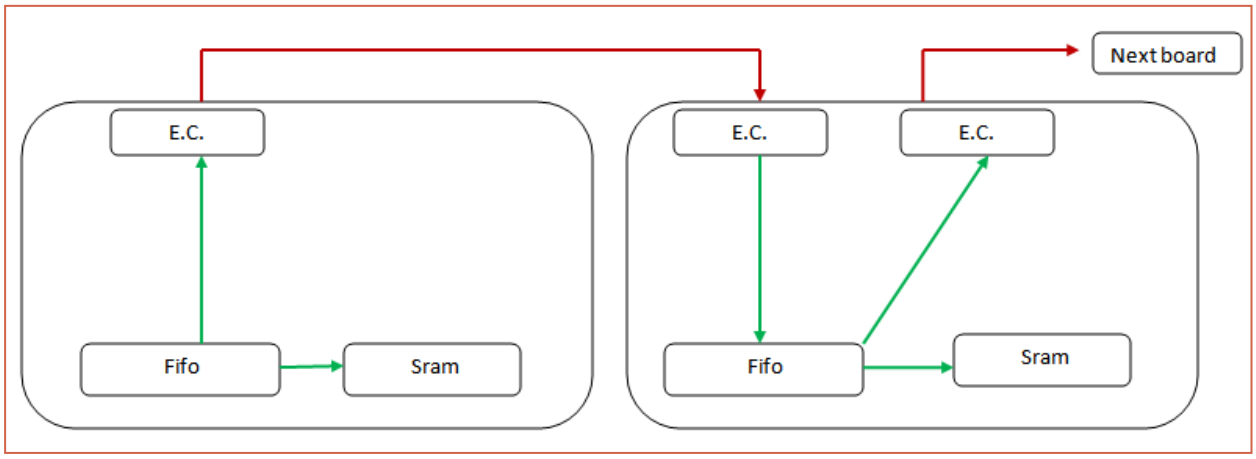
Στο τέλος του πρώτου μέρους, στην ενότητα 4.1.3, δίναμε τιμές σε μια μνήμη fifo 16 θέσεων και ο 4 - bit αριθμός που είχε κάθε στιγμή μια θέση της μνήμης, εμφανιζόταν υπό την μορφή ενεργοποιημένων leds, στο δεύτερο board. Η μνήμη fifo, δημιουργούνταν στο FPGA καθενός board.



Εικόνα 64 - Η ενότητα 4.1.3

Στο δεύτερο μέρος της εργασίας μας, δημιουργήσαμε τοπικά σε κάθε board, μνήμες fifo. Τα αποθηκευμένα δεδομένα της 18-bit fifo, αποθηκεύονταν στην μνήμη

Sram του FPGA. Παράλληλα, μέρος των δεδομένων δεν θα αποθηκεύονται στην τοπική Sram, αλλά θα στέλνονται στο δεύτερο board και συγκεκριμένα στην fifo του νέου board και έπειτα στην Sram του board.



Εικόνα 65 - Μέρος Β

Το πρώτο board στέλνει μέρος των δεδομένων στην Sram και το υπόλοιπο μέρος στους Expansion connectors. Οι Expansion connectors, μεταφέρουν τα δεδομένα στο επόμενο board. Εκεί, πάλι τα δεδομένα μπαίνουν σε μια fifo. Η process Separator στέλνει τα μισά δεδομένα στην τοπική Sram και τα υπόλοιπα εκ νέου σε έναν άλλο Expansion connector, ο οποίος με την σειρά του θα τα στείλει στο επόμενο συνδεδεμένο board.

Με αυτόν τον τρόπο, πετυχαίνουμε τον διαμοιρασμό των δεδομένων έτσι ώστε να επεξεργαστούν πιο γρήγορα, μιας και δεν θα χρειάζεται να περιμένουν να αδειάζει η Sram, αλλά θα πηγαίνουν σε μια άδεια Sram ενός άλλου συνδεδεμένου board. Απο κει και πέρα, από κάθε expansion connector, μπορούμε να πάρουμε τα δεδομένα που έχει κάθε Sram και να τα στείλουμε προς τον επεξεργαστή.

Σαν συμπέρασμα της πτυχιακής μας εργασίας, είδαμε ότι τα FPGA είναι ολοκληρωμένα κυκλώματα που λόγω της κατασκευής τους μας δίνουν αρκετές δυνατότητες. Η δυνατότητα τόσο να εκτελεί συναρτήσεις και κυκλώματα (δηλαδή και τους κώδικές μας), όσο και η δυνατότητα να λειτουργεί σαν μνήμη Ram, μας επέτρεψαν στο ίδιο ολοκληρωμένο σύστημα να τρέχουμε το λογισμικό μας και παράλληλα να επεξεργαζόμαστε τα δεδομένα τοπικά στην μνήμη. Μάλιστα εάν η μνήμη γεμίσει, για λόγους χρόνου, μπορούμε να στείλουμε τα δεδομένα σε άλλα συνδεδεμένα boards, τα οποία θα εκτελούν αντίστοιχη διαδικασία με το πρώτο board. Έτσι με αυτό τον τρόπο μπορούμε να επιτύχουμε γρήγορη επεξεργασία εφαρμογών σε πολυπύρηνα συστήματα.

6. Βιβλιογραφία

[1] https://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD

[2] http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf

[3] https://en.wikipedia.org/wiki/Field-programmable_gate_array

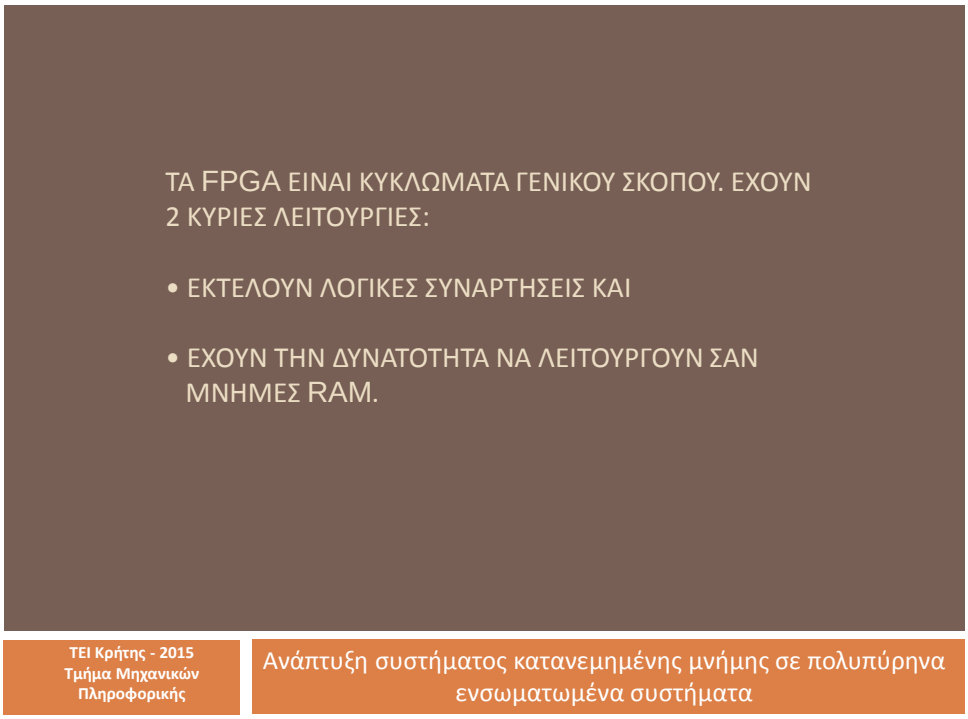
[4] http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[5] http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[6] Σχεδιασμός κυκλωμάτων με VHDL (Volnei A. Pedroni, εκδόσεις κλειδάριθμος).

7. Παράρτημα

Παρουσίαση της πτυχιακής



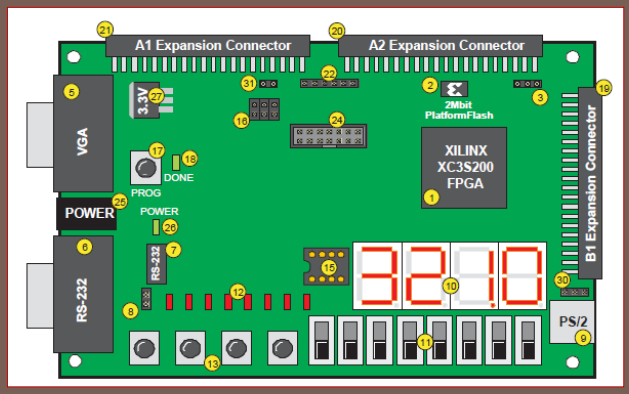
ΣΤΗΝ ΠΤΥΧΙΑΚΗ ΜΑΣ:

- Ο ΚΩΔΙΚΑΣ ΕΙΝΑΙ ΓΡΑΜΜΕΝΟΣ ΣΕ ΓΛΩΣΣΑ VHDL (ΓΛΩΣΣΑ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ).
- ΧΡΗΣΙΜΟΠΟΙΟΥΜΕ 2 ΣΥΝΔΕΔΕΜΕΝΑ BOARDS ΠΟΥ ΕΧΟΥΝ FPGA.

Ο ΣΤΟΧΟΣ ΜΑΣ

- ΣΚΟΠΟΣ ΜΑΣ ΕΙΝΑΙ ΝΑ ΣΥΝΔΕΣΟΥΜΕ ΔΥΟ BOARD ΜΕΣΩ ΤΩΝ EXPANSION CONNECTORS.
- ΕΤΣΙ ΘΑ ΜΕΤΑΦΕΡΟΥΜΕ ΤΑ ΔΕΔΟΜΕΝΑ ΤΗΣ ΜΝΗΜΗΣ SRAM ΑΠΟ ΤΟ ΕΝΑ BOARD ΣΤΟ ΑΛΛΟ.
- ΜΕ ΑΥΤΟ ΤΟΝ ΤΡΟΠΟ, ΕΠΙΤΥΓΧΑΝΟΥΜΕ ΓΡΗΓΟΡΟΤΕΡΟ ΔΙΑΜΕΡΙΣΜΟ ΤΩΝ ΔΕΔΟΜΕΜΩΝ.

ΤΟ BOARD ΤΗΣ XILINX (SPARTAN 3)



ΠΡΩΤΗ ΦΑΣΗ ΤΗΣ ΠΤΥΧΙΑΚΗΣ

- ΕΛΕΓΧΟΣ ΤΩΝ ΠΕΡΙΦΕΡΕΙΑΚΩΝ ΜΕΡΩΝ ΤΟΥ BOARD

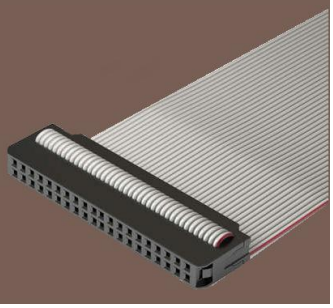
I) ΜΕΤΑΦΟΡΑ ΔΕΔΟΜΕΝΩΝ ΜΕ ΧΡΗΣΗ PUSH BUTTONS ΚΑΙ SWITCHES ΚΑΙ ΕΜΦΑΝΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΤΑ LEDS
II) ΑΥΤΟΜΑΤΗ ΜΕΤΑΦΟΡΑ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΕΜΦΑΝΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΤΗΝ ΟΘΟΝΗ SEVEN SEGMENT DISPLAY
III) ΜΕΤΑΦΟΡΑ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΜΙΑΣ ΜΝΗΜΗΣ FIFO, ΣΤΟ ΔΕΥΤΕΡΟ BOARD

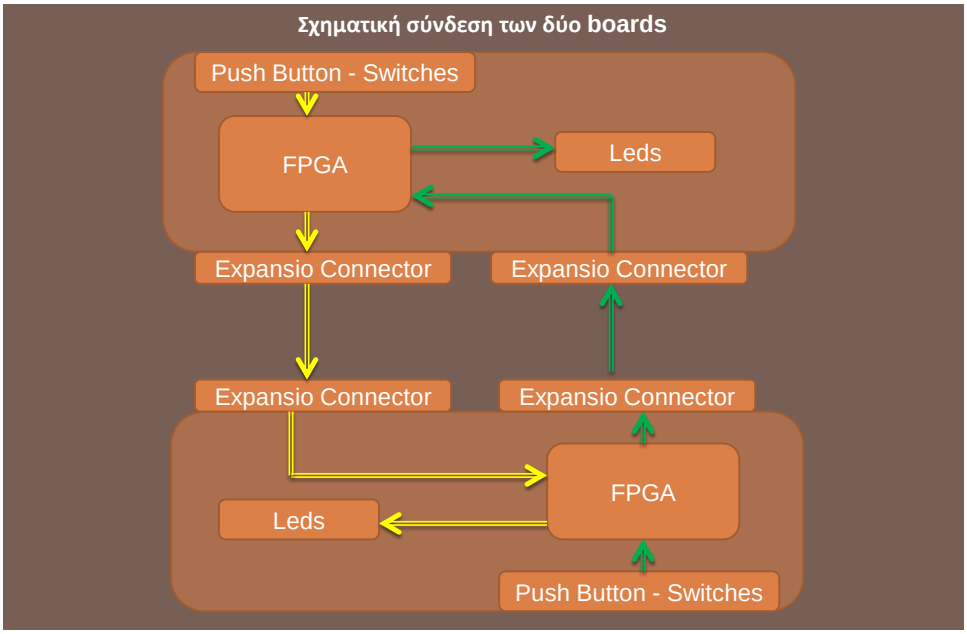
*ΣΗΜΕΙΩΣΗ: Η ΜΕΤΑΦΟΡΑ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΟ ΕΝΑ BOARD ΣΤΟ ΑΛΛΟ, ΕΠΙΤΥΓΧΑΝΕΤΑΙ ΜΕΣΩ ΤΩΝ EXPANSION CONNECTORS.

Ι) ΣΤΟ ΠΡΩΤΟ ΜΕΡΟΣ ΤΗΝ ΠΡΩΤΗΣ ΦΑΣΗΣ, ΧΡΗΣΙΜΟΠΟΙΟΥΜΕ SWITCHES, PUSH BUTTONS ΚΑΙ LEDS ΕΤΣΙ ΩΣΤΕ ΝΑ ΕΙΣΑΓΟΥΜΕ ΚΑΙ ΝΑ ΕΞΑΓΟΥΜΕ ΔΕΔΟΜΕΝΑ.

- ΑΠΟ ΤΑ SWITCHES ΚΑΙ ΤΑ PUSH BUTTONS, ΔΙΝΟΥΜΕ ΣΗΜΑ ΣΕ ΤΟΠΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ ΤΟΥ ΚΩΔΙΚΑ ΜΑΣ.
- ΑΝΑΛΟΓΑ ΤΙΣ ΕΙΣΟΔΟΥΣ, ΛΑΜΒΑΝΟΥΝ ΣΗΜΑ ΟΙ ΜΕΤΑΒΛΗΤΕΣ ΕΞΟΔΟΥ.
- ΟΙ ΜΕΤΑΒΛΗΤΕΣ ΕΞΟΔΟΥ ΔΙΝΟΥΝ ΣΗΜΑ ΣΤΟΥΣ EXPANSION CONNECTORS ΤΟΥ ΠΡΩΤΟΥ BOARD

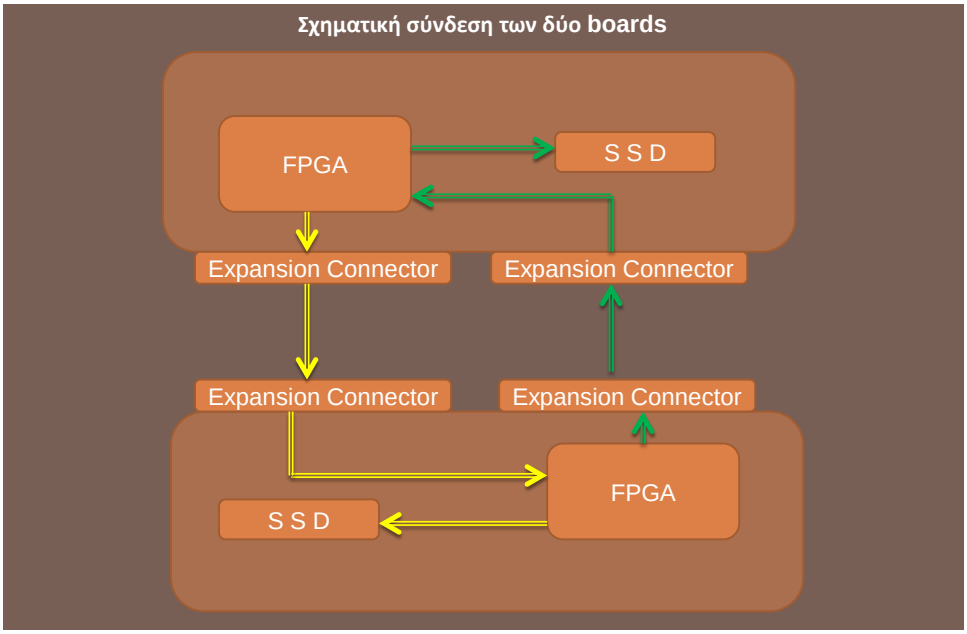
- ΟΙ EXPANSION CONNECTORS ΤΟΥ ΔΕΥΤΕΡΟΥ BOARD, ΣΤΕΛΝΟΥΝ ΤΑ ΚΑΤΑΛΛΗΛΑ ΣΗΜΑΤΑ ΣΤΟ FPGA.
- ΑΝΑΛΟΓΑ ΜΕ ΤΟΝ ΚΩΔΙΚΑ ΜΑΣ, ΠΑΡΑΓΟΝΤΑΙ ΚΑΙ ΟΙ ΕΞΟΔΟΙ (ΕΝΕΡΓΟΠΟΙΗΣΗ Ή ΑΠΕΝΕΡΓΟΠΟΙΗΣΗ ΤΩΝ LEDS ΣΤΗΝ ΠΡΟΚΕΙΜΕΝΗ ΠΕΡΙΠΤΩΣΗ)
- Η ΣΥΝΔΕΣΗ ΜΕΤΑΞΥ ΤΩΝ EXPANSION CONNECTORS ΓΙΝΕΤΑΙ ΜΕ ΚΑΛΩΔΙΟΤΑΙΝΙΑ (CABLE BUS), ΤΩΝ 40 PINS.





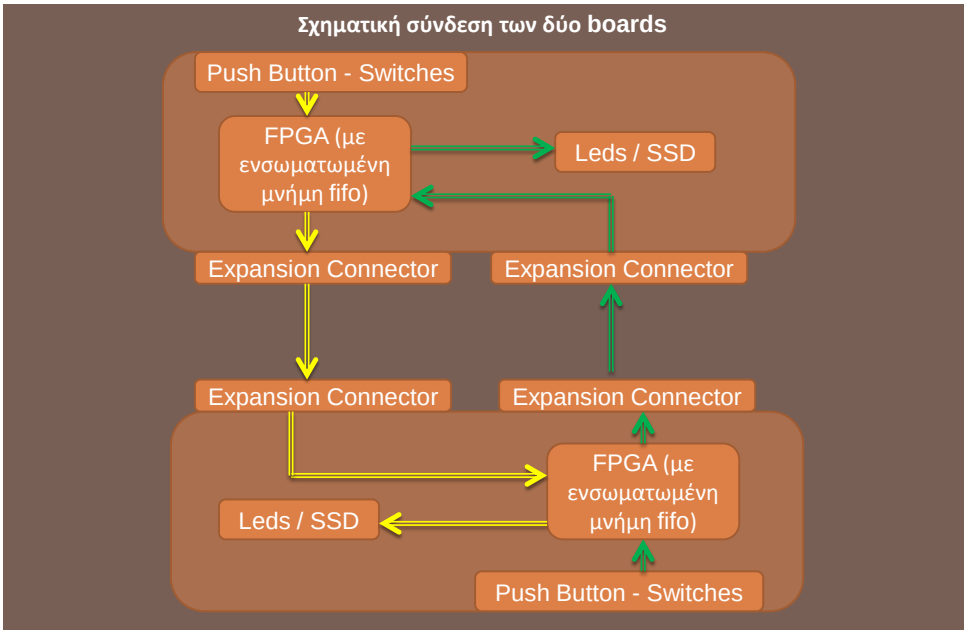
ΙΙ) ΣΤΟ ΔΕΥΤΕΡΟ ΜΕΡΟΣ ΤΗΣ ΠΡΩΤΗΣ ΦΑΣΗΣ, ΣΤΟ FPGA ΤΟΥ ΠΡΩΤΟΥ BOARD, ΤΡΕΧΕΙ ΚΩΔΙΚΑΣ Ο ΟΠΟΙΟΣ ΜΕΤΡΑΕΙ ΑΠΟ ΤΟ 1 ΕΩΣ ΤΟ F (ΣΤΟ ΔΕΚΑΕΞΑΔΙΚΟ ΑΡΙΘΜΗΤΙΚΟ ΣΥΣΤΗΜΑ).

- ΤΑ ΔΕΔΟΜΕΝΑ ΑΥΤΑ, ΜΕΤΑΦΕΡΟΝΤΑΙ ΣΤΟ ΔΕΥΤΕΡΟ BOARD ΚΑΙ Η ΜΕΤΡΗΣΗ ΕΜΦΑΝΙΖΟΝΤΑΙ ΣΤΟ SEVEN SEGMENT DISPLAY.
- ΑΥΤΗ Η ΔΥΝΑΤΟΤΗΤΑ ΜΠΟΡΕΙ ΝΑ ΒΟΗΘΗΣΕΙ ΤΟΝ ΧΡΗΣΤΗ ΤΟΥ ΔΕΥΤΕΡΟΥ BOARD ΚΑΘΩΣ ΒΛΕΠΕΙ ΣΤΗΝ ΟΘΟΝΗ ΤΟΥ ΤΙ ΓΙΝΕΤΕ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ, ΣΤΟ ΠΡΩΤΟ BOARD.
- ΕΤΣΙ ΜΠΟΡΕΙ ΝΑ ΕΞΑΓΕΙ ΠΛΗΡΟΦΟΡΙΕΣ ΚΑΙ ΕΝΔΕΧΟΜΕΝΩΣ ΝΑ ΠΡΟΒΑΙΝΕΙ ΣΕ ΠΡΑΞΕΙΣ ΔΙΑΔΡΑΣΗΣ (Π.Χ. ΑΝΑΛΟΓΑ ΤΙΣ ΤΙΜΕΣ ΤΟΥ SSD, ΝΑ ΜΠΟΡΕΙ ΝΑ ΣΤΕΛΝΕΙ ΚΑΙ ΑΥΤΟΣ ΔΕΔΟΜΕΝΑ ΣΤΟ ΑΛΛΟ BOARD).



III) ΣΤΟ ΤΡΙΤΟ ΜΕΡΟΣ ΤΗΣ ΠΡΩΤΗΣ ΦΑΣΗΣ, ΔΗΜΙΟΥΡΓΟΥΜΕ ΤΟΠΙΚΑ ΣΕ ΚΑΘΕ FPGA, ΜΝΗΜΕΣ FIFO.

- ΑΝ Η FIFO ΓΕΜΙΣΕΙ, ΜΠΟΡΕΙ ΝΑ ΣΤΕΙΛΕΙ ΤΑ ΔΕΔΟΜΕΝΑ ΣΤΟ ΔΕΥΤΕΡΟ BOARD, ΕΠΕΙΤΑ ΝΑ ΑΔΕΙΑΣΕΙ ΚΑΙ ΝΑ ΕΙΝΑΙ ΕΤΟΙΜΗ ΓΙΑ ΝΕΑ ΧΡΗΣΗ.
- ΣΤΟ ΔΕΥΤΕΡΟ BOARD, ΣΤΗΝ ΤΟΠΙΚΗ FIFO, ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΤΑ ΔΕΔΟΜΕΝΑ ΤΗΣ ΑΛΛΗΣ FIFO.
- ΜΕ ΤΑ LEDS ΚΑΙ ΤΟ SSD ΜΑΘΑΙΝΟΥΜΕ ΠΛΗΡΟΦΟΡΙΕΣ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΚΑΤΑΣΤΑΣΗ ΤΗΣ ΜΝΗΜΗΣ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ.



ΤΕΙ Κρήτης - 2015
Τμήμα Μηχανικών Πληροφορικής

Ανάπτυξη συστήματος κατανεμημένης μνήμης σε πολυπύρηνα ενσωματωμένα συστήματα

ΔΕΥΤΕΡΗ ΦΑΣΗ ΠΤΥΧΙΑΚΗΣ

- ΣΤΗΝ ΔΕΥΤΕΡΗ ΦΑΣΗ ΤΗΣ ΠΤΥΧΙΑΚΗΣ ΜΑΣ ΘΑ ΠΑΜΕ ΕΝΑ ΒΗΜΑ ΠΑΡΑΚΑΤΩ, ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΤΙΣ ΓΝΩΣΕΙΣ ΠΟΥ ΑΠΟΚΟΜΙΣΑΜΕ ΑΠΟ ΤΗΝ ΠΡΩΤΗ ΦΑΣΗ.
- ΘΑ ΣΥΝΔΕΣΟΥΜΕ ΔΥΟ BOARD, ΚΑΙ ΕΚ ΝΕΟΥ ΘΑ ΔΗΜΙΟΥΡΓΗΣΟΥΜΕ ΤΟΠΙΚΑ ΜΝΗΜΕΣ FIFO ΣΕ ΚΑΘΕ FPGA.

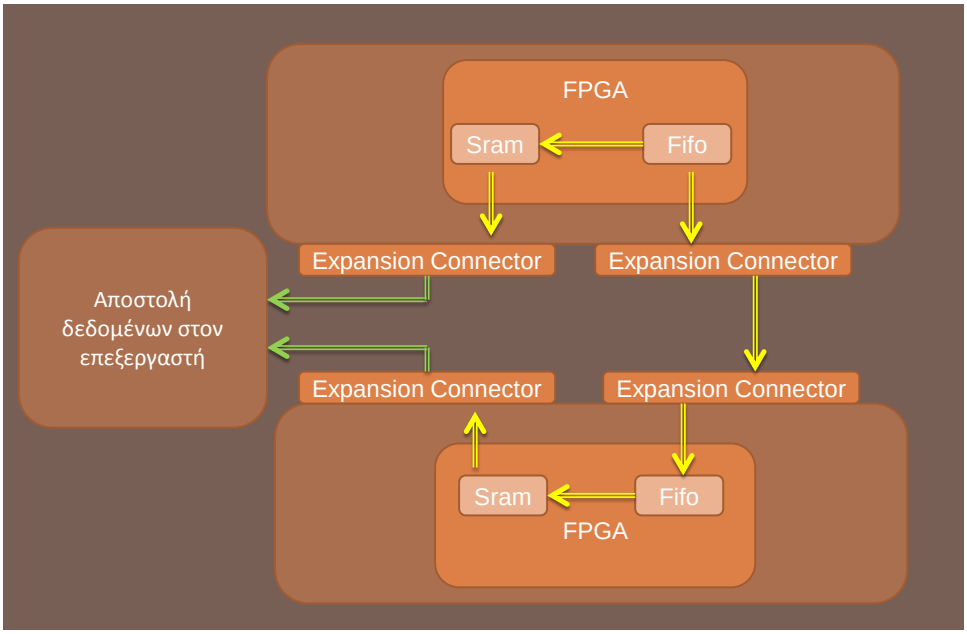
ΤΕΙ Κρήτης - 2015
Τμήμα Μηχανικών Πληροφορικής

Ανάπτυξη συστήματος κατανεμημένης μνήμης σε πολυπύρηνα ενσωματωμένα συστήματα

- ΣΤΟ ΕΝΑ BOARD, ΘΑ ΕΙΣΕΡΧΟΝΤΑΙ ΔΕΔΟΜΕΝΑ ΤΑ ΟΠΟΙΑ ΑΡΧΙΚΑ ΘΑ ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΣΤΗΝ ΤΟΠΙΚΗ FIFO.
- ΕΝΑ ΜΕΡΟΣ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΘΑ ΠΑΝΕ ΠΡΟΣ ΑΠΟΘΗΚΕΥΣΗ ΣΤΗΝ SRAM ΤΟΥ FPGA ΚΑΙ ΤΑ ΥΠΟΛΟΙΠΑ ΘΑ ΠΗΓΑΙΝΟΥΝ ΜΕΣΩ ΤΩΝ EXPANSION CONNECTORS ΣΤΟ ΔΕΥΤΕΡΟ BOARD.
- ΣΤΟ ΔΕΥΤΕΡΟ BOARD ΘΑ ΑΚΟΛΟΥΘΕΙΤΕ Η ΙΔΙΑ ΔΙΑΔΙΚΑΣΙΑ. ΤΑ ΕΙΣΕΡΧΟΜΕΝΑ ΔΕΔΟΜΕΝΑ ΘΑ ΜΠΑΙΝΟΥΝ ΣΕ ΜΙΑ FIFO.
- Η FIFO ΜΕ ΤΗΝ ΣΕΙΡΑ ΤΗΣ ΘΑ ΣΤΕΛΝΕΙ ΕΝΑ ΜΕΡΟΣ ΤΩΝ ΝΕΩΝ ΔΕΔΟΜΕΝΩΝ ΣΤΗΝ ΤΟΠΙΚΗ SRAM ΚΑΙ ΤΑ ΥΠΟΛΟΙΠΑ ΔΕΔΟΜΕΝΑ ΣΕ ΑΛΛΟ ΣΥΝΔΕΔΕΜΕΝΟ BOARD (ΑΝ ΦΥΣΙΚΑ ΥΠΑΡΧΕΙ).

- ΕΙΝΑΙ ΣΗΜΑΝΤΙΚΟ ΝΑ ΑΝΑΦΕΡΟΥΜΕ ΟΤΙ ΤΑ ΔΕΔΟΜΕΝΑ ΠΟΥ ΣΤΕΛΝΟΝΤΑΙ, ΕΧΟΥΝ ΕΠΙΚΕΦΑΛΙΔΑ Η ΟΠΟΙΑ ΤΟΠΟΘΕΤΕΙΤΑΙ ΑΝΑΛΟΓΑ ΜΕ ΤΟ BOARD ΣΤΟ ΟΠΟΙΟ ΒΡΙΣΚΟΝΤΑΙ.
- ΓΙΑ ΠΑΡΑΔΕΙΓΜΑ, ΣΕ ΠΑΚΕΤΟ 18 BITS, ΤΑ 2 ΠΡΩΤΑ BITS ΕΙΝΑΙ Η ΕΠΙΚΕΦΑΛΙΔΑ ΚΑΙ ΤΑ ΥΠΟΛΟΙΠΑ 16 ΕΙΝΑΙ ΤΑ ΔΕΔΟΜΕΝΑ ΠΟΥ ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΣΤΗΝ SRAM.
- ΤΑ ΔΕΔΟΜΕΝΑ ΤΟΥ ΠΡΩΤΟΥ BOARD ΘΑ ΕΧΟΥΝ ΕΠΙΚΕΦΑΛΙΔΑ 00, ΤΟΥ ΔΕΥΤΕΡΟΥ 01 Κ.Ο.Κ

Επικ	Επικ	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data



ΣΥΜΠΕΡΑΣΜΑΤΑ

- Η ΠΑΡΟΥΣΑ ΠΤΥΧΙΑΚΗ, ΗΤΑΝ ΜΙΑ ΕΡΕΥΝΗΤΙΚΗ ΕΡΓΑΣΙΑ ΜΕ ΑΝΤΙΚΕΙΜΕΝΟ ΤΑ ΣΥΣΤΗΜΑΤΑ FPGA.
- Ο ΣΤΟΧΟΣ ΜΑΣ ΗΤΑΝ ΝΑ ΦΤΙΑΞΟΥΜΕ ΕΝΑ ΣΥΣΤΗΜΑ, ΤΟ ΟΠΟΙΟ ΔΕΧΕΤΑΙ ΔΕΔΟΜΕΝΑ ΣΕ ΜΟΡΦΗ BITS ΚΑΙ ΧΡΗΣΙΜΟΠΟΙΕΙ ΤΗΝ SRAM ΤΟΥ FPGA ΓΙΑ ΝΑ ΑΠΟΘΗΚΕΥΣΕΙ ΤΑ ΔΕΔΟΜΕΝΑ ΑΥΤΑ, ΠΟΥ ΘΑ ΧΡΗΣΙΜΟΠΟΙΗΣΕΙ Ο ΕΠΕΞΕΡΓΑΣΤΗΣ ΤΟΥ ΥΠΟΛΟΓΙΣΤΗ.
- ΜΕ ΤΗΝ ΧΡΗΣΗ ΠΟΛΛΑΠΛΩΝ ΣΥΝΔΕΔΕΜΕΝΩΝ BOARDS, ΕΧΟΥΜΕ ΤΗΝ ΔΥΝΑΤΟΤΗΤΑ ΑΝΤΙ ΝΑ ΧΡΗΣΙΜΟΠΟΗΣΟΥΜΕ ΜΟΝΟ ΜΙΑ SRAM 256K X 32, ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΣΟΥΜΕ ΔΥΟ Ή ΚΑΙ ΠΕΡΙΣΣΟΤΕΡΕΣ ΩΣΤΕ ΤΑ ΔΕΔΟΜΕΝΑ ΝΑ ΚΙΝΟΥΝΤΑΙ ΠΙΟ ΓΡΗΓΟΡΑ, ΕΛΑΧΙΣΤΟΠΟΙΩΝΤΑΣ ΤΟΝ ΧΡΟΝΟ ΕΠΕΞΕΡΓΑΣΙΑΣ.

ΕΡΩΤΗΣΕΙΣ;

ΣΑΣ ΕΥΧΑΡΙΣΤΟΥΜΕ ΓΙΑ ΤΟΝ ΧΡΟΝΟ ΣΑΣ!